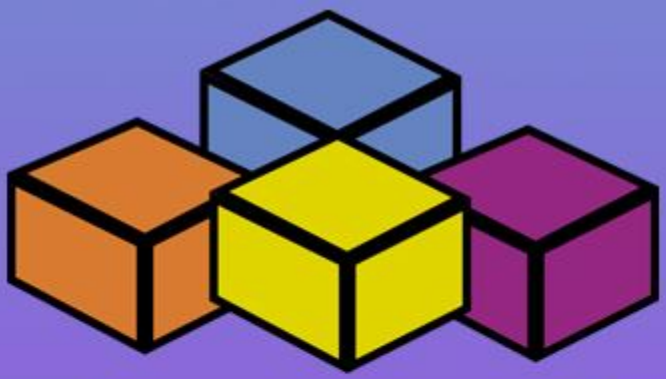
ОФІСНЕ ПРОГРАМУВАННЯ МОВОЮ VBA

Г.П.Коломоєць



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Г.П. Коломоєць

ОФІСНЕ ПРОГРАМУВАННЯ МОВОЮ VBA

Навчальний посібник
для здобувачів ступеня вищої освіти бакалавра
спеціальності 121 "Інженерія програмного забезпечення"
освітньо-професійної програми
"Програмне забезпечення систем"

Затверджено
вченою радою ЗНУ
Протокол № 4 від 28.11.2023 р

Запоріжжя
2023

УДК: 004.91(43)
К612

Коломоєць Г. П. Офісне програмування мовою VBA : навчальний посібник для здобувачів ступеня вищої освіти бакалавра спеціальності 121 "Інженерія програмного забезпечення" освітньо-професійної програми "Програмне забезпечення систем". Запоріжжя : Запорізький національний університет, 2023. 271 с.

Розглянуті засоби програмування, вбудовані у застосунки пакету програм Microsoft Office. Вивчаються синтаксис, типи даних, оператори та основні функції мови програмування Visual Basic for Application (VBA). Викладені основи побудови класів та роботи з об'єктами, а також технології створення користувацьких форм. Розглянуті засоби інтерфейсів прикладного програмування, вбудовані до офісних застосунків Microsoft Word та Microsoft Excel, та способи програмування з їх використанням. Посібник містить багато прикладів програм, на базі яких можуть бути розроблені рішення автоматизації рутинних операцій у офісних застосунках. В кінці кожної теми наведені питання для самоконтролю, тестові завдання для перевірки знань та практичні завдання для самостійного виконання. Текст посібника, приклади програм та скріншоти створені у україномовних застосунках пакету Microsoft Office, які працюють під операційною системою Windows з українською локалізацією. Таким чином автор намагається заповнити прогалини у наявності україномовних підручників та посібників з сучасних інформаційних технологій. Посібник орієнтований на студентів, що навчаються за спеціальністю 121 «Інженерія програмного забезпечення» очної та заочної форм навчання. Також він може бути корисним студентам інших спеціальностей, які вивчають технології та інструменти офісного програмного забезпечення.

Рецензент

С.Л. Хрипко, доктор технічних наук, професор, завідувач кафедри інформаційних технологій та дизайну Класичного приватного університету, м. Запоріжжя

Відповідальна за випуск

Т.В. Критська, доктор технічних наук, професор, завідувач кафедри електроніки, інформаційних систем та програмного забезпечення

ЗМІСТ

ВСТУП

Пакет програм *Microsoft Office* – це середовище, яке забезпечує рішення багатьох завдань людської діяльності, і при цьому є легким у застосуванні і інтуїтивно зрозумілим. Саме ці риси разом з можливістю обміну даними між застосунками зробили цей пакет одним з найпопулярніших у світі. Одночасно розробники пакета Microsoft Office передбачили можливість автоматизувати рутинні та часто повторювані дії, наприклад, однакову обробку текстових документів (форматування, вставку різних об'єктів тощо), перевірку введених користувачами даних до електронних таблиць, аналіз однаково структурованої інформації за різні періоди, додання елементів до слайдів за умовами, роботу з інформацією з бази даних тощо. Для створення власних програм користувачеві надано програмний інструментарій, вбудований в програми пакету Office – редактор мови *Visual Basic for Applications* (*VBA*) та пов'язані з популярними офісними застосунками засоби програмування – *інтерфейси прикладного програмування* (*Application Programming Interface – API*), які являють собою сукупність пов'язаних класів, які відповідають елементам, з якими користувач працює у відповідних застосунках. Такий підхід відкриває дорогу відносно новому напрямку у сучасному програмуванні – *офісному програмуванню*.

Мова програмування VBA обрана для офісного програмування завдяки її простоті, легкості в освоєнні та можливості швидко отримати відчутні результати. Ця мова підтримує технологію візуального програмування, яке дозволяє наочно конструювати екранні форми і елементи керування, а також виконувати запис всієї програми або її частин за допомогою макрорекордеру. Досить часто при вирішенні найпростіших завдань достатньо засобів візуального програмування, але якщо потрібно створити більш складний додаток з змістовною обробкою даних, то на перший план виступає саме мова програмування VBA та API відповідних офісних застосунків.

Мова VBA та засоби програмування вбудовані у такі застосунки пакету Microsoft Office, як Word, Excel, PowerPoint, Outlook, Access, Publisher, Visio, Project, а також у Microsoft Office for Mac. Програмування на VBA підтримується і у популярних застосунках інших виробників: CorelDraw, Corel WordPerfect Office та AutoCAD.

Метою вивчення навчальної дисципліни «Офісне програмування» є підготовка фахівців, що володіють знаннями щодо способів та засобів програмування у застосунках пакету Microsoft Office, а також набули навичок розробки програм на мові програмування Visual Basic for Application для автоматизації виконання задач у текстовому та табличному процесорах офісного пакету Microsoft.

Основними завданнями дисципліни «Офісне програмування» є вивчення мови програмування VBA, засобів програмування, вбудованих у офісні застосунки, інтерфейсів прикладного програмування Microsoft Word та Microsoft Excel, а також опанування способів автоматизації рутинних завдань, які виконуються у основних застосунках пакету Microsoft Office.

У результаті вивчення навчальної дисципліни студент повинен
знати:

- засоби створення програм у застосунках Microsoft Office;
- типи даних, операції та оператори мови програмування VBA;
- способи створення та роботи з підпрограмами та функціям;
- об’єктні моделі Microsoft Word та Microsoft Excel та їх API.

вміти:

- записувати, редагувати та запускати на виконання макроси, що автоматизують рутинні завдання, виконувані у офісних застосунках;
- розробляти підпрограми та функції на мові програмування VBA;
- розробляти користувацькі класи та використовувати об’єкти таких класів у програмах;
- розробляти користувацькі форми та програмувати обробку подій від елементів керування форм VBA;
- працювати із засобами інтерфейсів прикладного програмування Microsoft Word та Microsoft Excel при написанні програм, які працюють з текстовими документами та електронними таблицями.

Згідно з вимогами освітньо-професійної програми «Програмне забезпечення систем» студенти повинні досягти таких компетентностей:

- Здатність до абстрактного мислення, аналізу та синтезу.
- Здатність застосовувати знання у практичних ситуаціях.
- Здатність вчитися і оволодівати сучасними знаннями.
- Здатність накопичувати, обробляти та систематизувати професійні знання щодо створення і супроводження програмного забезпечення та визнання важливості навчання протягом всього життя.
- Здатність обґрунтовано обирати та освоювати інструментарій з розробки та супроводження програмного забезпечення.
- Здатність до алгоритмічного та логічного мислення. забезпечення.

Міждисциплінарні зв’язки. Відповідно до структурно-логічної схеми освітньо-професійної програми, засвоєння навчального матеріалу курсу «Офісне програмування» логічно пов’язане з використанням знань, вмінь та навичок, отриманих у результаті вивчення дисциплін «Основи алгоритмізації» та «Основи програмування».

Текст посібника, приклади програм та скріншоти створені у україномовних застосунках пакету Microsoft Office, які працюють під операційною системою Windows з українською локалізацією. Таким чином автор намагається заповнити прогалини у наявності україномовних підручників та посібників з сучасних інформаційних технологій.

Використане при розробці посібника програмне забезпечення компанії Microsoft отримане в рамках академічної підписки Open Value Subscription замовником Zaporizhzhia National University, номер угоди V8847071.

ТЕМА 1 ОРГАНІЗАЦІЯ ПРОГРАМУВАННЯ У ОФІСНИХ ЗАСТОСУНКАХ

Мета: Вивчити склад та функціонал засобів програмування на мові VBA, вбудованих до основних застосунків пакету Microsoft Office, та отримати практичні навички їх використання.

План

1. Поняття макросу, запис та запуск макросу
2. Формати файлів для збереження проєктів VBA
3. Підключення засобів програмування
4. Вікна і панелі Редактору Visual Basic
5. Вікно Провідника проєктів та елементи проєкту VBA
6. Вікно Редактора вихідного коду та способи написання і запуску програм
7. Робота з довідкою з VBA та API офісних застосунків
8. Налаштування для забезпечення копіювання фрагментів коду з нелатинськими символами.

✎ Основні терміни і поняття

Мова програмування VBA, Макрос, редактор Visual Basic, проєкт, модуль, процедура.

📚 Основні теоретичні положення

1.1 Поняття макросу, запис та запуск макросу

Макрос – це, по суті, невеличка програма на мові VBA, яка дозволяє автоматизувати часто виконувані користувачем дії, шляхом запису таких дії за допомогою *макрорекордера*, вбудованого до основних застосунків Microsoft Office [1].

Алгоритм дій при створенні команди наступний:

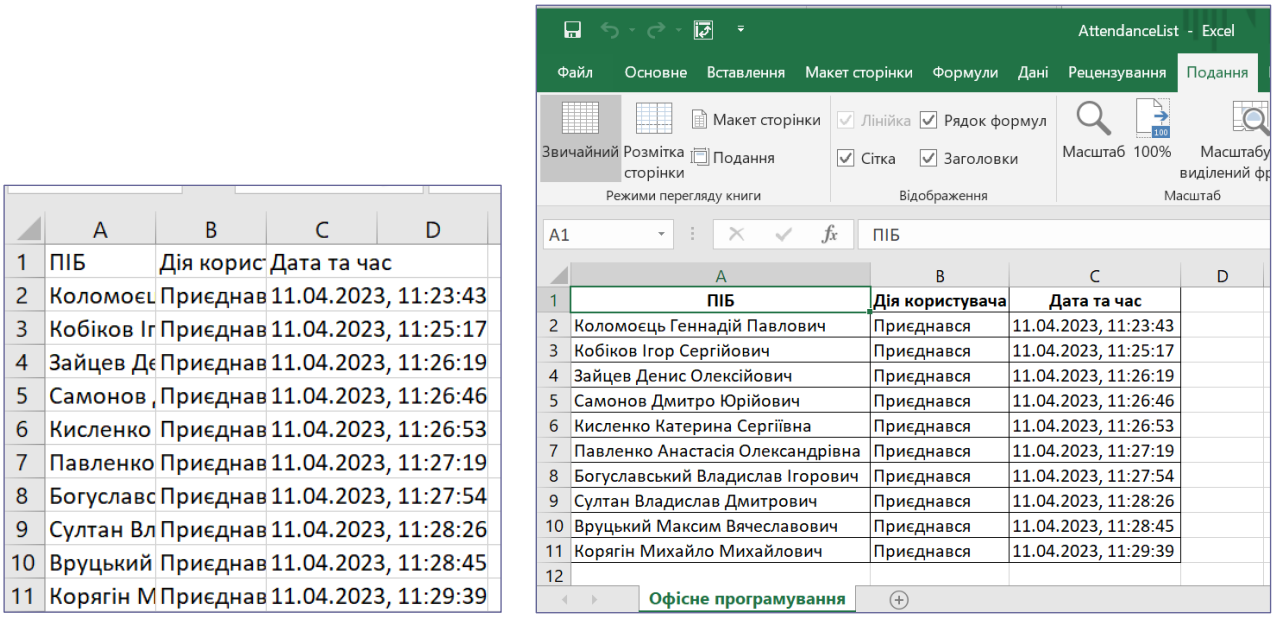
- попередньо продумати та визначити порядок дій;
- включити макрорекордер на запис;
- надати макросу ім'я (також можливо надати необов'язкові параметри – опис, призначення кнопки запуску, призначення клавіатурної комбінації запуску та місця збереження макросу (і, відповідно, області його дії);
- записати дії у макрос у макрорекордері;
- відтворити макрос для перевірки правильності його роботи.

Зверніть увагу, що кроки запису та запуску макросу можуть взагалі не передбачати роботу з програмним кодом, хоча насправді макрорекордер фіксує етапи виконання дій користувачем саме програмними конструкціями VBA і їх можна переглянути та, за необхідності, відредагувати (це буде продемонстровано нижче).

Наприклад, часто виконувана дія – створення списку присутніх на лекції студентів, завантажених з системи підтримки дистанційної роботи Microsoft

Teams починається з файлу Excel вигляду, показаного на Рисунок 1а. Необхідно виконати форматування для нормального вигляду списку. Оскільки це виконується часто, є сенс записати макрос.

Для цього визначимо порядок дій: змінити ширину стовпців, додати границі, відформатувати заголовок таблиці, змінити назву аркушу на назву дисципліни («Офісне програмування») (Рисунок 1б).



	A	B	C	D
1	ПІБ	Дія корис	Дата та час	
2	Коломоєц	Приєднав	11.04.2023, 11:23:43	
3	Кобіков Іг	Приєднав	11.04.2023, 11:25:17	
4	Зайцев Де	Приєднав	11.04.2023, 11:26:19	
5	Самонов ,	Приєднав	11.04.2023, 11:26:46	
6	Кисленко	Приєднав	11.04.2023, 11:26:53	
7	Павленко	Приєднав	11.04.2023, 11:27:19	
8	Богуславс	Приєднав	11.04.2023, 11:27:54	
9	Султан Вл	Приєднав	11.04.2023, 11:28:26	
10	Вруцький	Приєднав	11.04.2023, 11:28:45	
11	Корягін М	Приєднав	11.04.2023, 11:29:39	

AttendanceList - Excel

Файл Основне Вставлення Макет сторінки Формули Дані Рецензування Подання

Звичайний Розмітка сторінки Макет сторінки Подання Лінійка Рядок формул Сітка Заголовки Масштаб 100% Масштабу виділений фр

Режими перегляду книги Відображення Масштаб

A1 ПІБ

	A	B	C	D
1	ПІБ	Дія користувача	Дата та час	
2	Коломоєць Геннадій Павлович	Приєднався	11.04.2023, 11:23:43	
3	Кобіков Ігор Сергійович	Приєднався	11.04.2023, 11:25:17	
4	Зайцев Денис Олексійович	Приєднався	11.04.2023, 11:26:19	
5	Самонов Дмитро Юрійович	Приєднався	11.04.2023, 11:26:46	
6	Кисленко Катерина Сергіївна	Приєднався	11.04.2023, 11:26:53	
7	Павленко Анастасія Олександрівна	Приєднався	11.04.2023, 11:27:19	
8	Богуславський Владислав Ігорович	Приєднався	11.04.2023, 11:27:54	
9	Султан Владислав Дмитрович	Приєднався	11.04.2023, 11:28:26	
10	Вруцький Максим Вячеславович	Приєднався	11.04.2023, 11:28:45	
11	Корягін Михайло Михайлович	Приєднався	11.04.2023, 11:29:39	
12				

Офісне програмування

а) б)

Рисунок 1 – Обробка макросом списку студентів: а) первинний вигляд, б) вигляд після обробки

Далі командою головного меню *Подання-Макроси-Записати макрос...* відкриємо вікно *Запис макросу*, у якому зазначимо ім'я макросу, його короткий опис, клавіатурну комбінацію запуску макросу та місце його збереження (Рисунок 2).

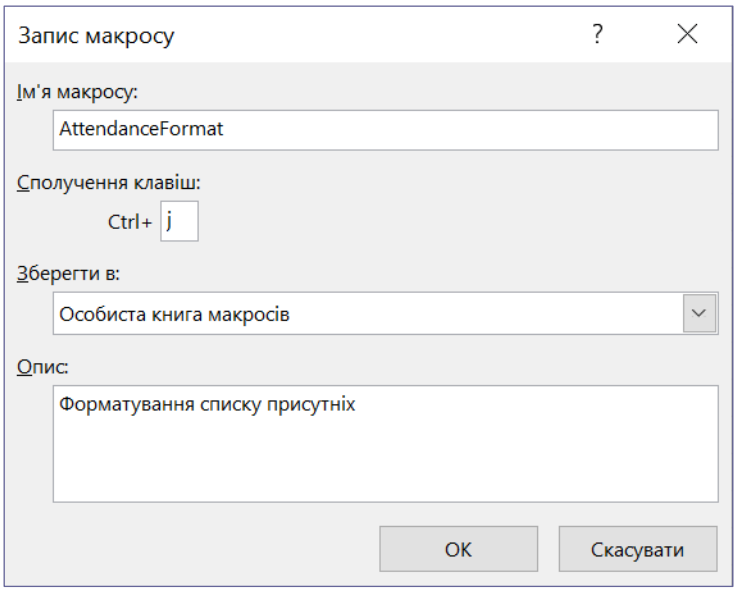
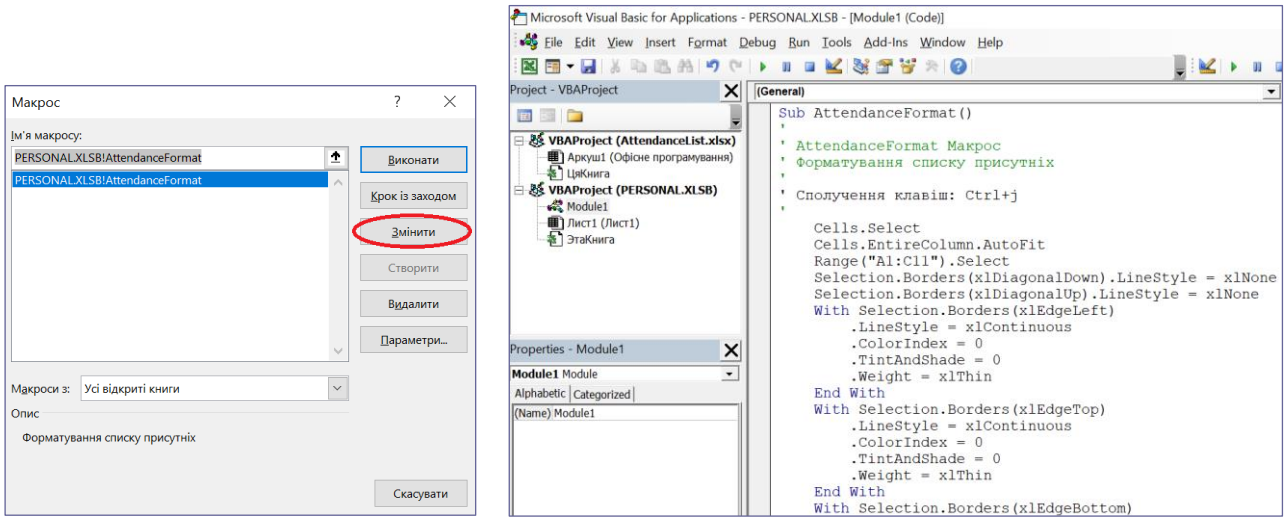


Рисунок 2 – Призначення властивостей макросу

Опція збереження макросів передбачає можливість обрати їх місце збереження: у особистій книзі макросів – у файлі PERSONAL.XLSB, який за замовчуванням зберігається у профілі користувача у підкаталозі AppData\Roaming\Microsoft\Excel\XLSTART, поточній книзі та у новій книзі. Обрання як місця зберігання макросу особистої книги макросів дозволить його використовувати для будь-яких книг (файлів) Excel, натомість інші опції дозволять використовувати макрос тільки у одній книзі (поточній або новій), – тому обрана перша опція [1].

Після клікання на кнопці *OK* вікна *Запис макросу* починається його запис і необхідно виконати визначені вище дії по форматуванню аркуша зі списком присутніх. Після цього необхідно командою головного меню *Подання-Макроси-Зупинити макрос* завершити запис макросу.

Переглянути наявність макросу можливо командою *Подання-Макроси-Переглянути макрос* (Рисунок 3а) (у разі його збереження у персональній книзі макросів – PERSONAL, попередньо необхідно її відобразити командою *Подання-Відобразити* у секції *Вікно*). Виділивши ім'я макросу у списку та клікнувши по кнопці *Змінити* можна побачити код макросу на мові VBA у *Редакторі Visual Basic* (Рисунок 3б) (роботу з цим редактором буде розглянуто нижче).



а) б)

Рисунок 3 – Записаний макрос: а) у списку макросів, б) вихідний код макросу

Запустити макрос для відкритої книги можна з вікна макросів командою *Подання-Макроси-Переглянути макрос*, обравши необхідний макрос зі списку та клікнувши по кнопці *Виконати* (у разі, якщо призначена клавіатурна комбінація для запуску макроса можна її використати прямо у відкритій книзі без відкриття вікна макросів). Результат роботи макросу показаний на Рисунок 1б.

Аналогічно можливо записати макроси для автоматизації виконання різних дій користувача: можна змінювати абзацний відступ, інтервали між рядками, вирівнювати області тексту, виконати розрахунок заданих значень, налаштувати автоматичне заповнення таблиці тощо. Після запуску запису макросу клікайте мишею на відповідних командах меню, введіть з клавіатури необхідні значення тощо. Тільки не виділяйте текст мишкою, для цього краще

використовувати клавіатурні комбінації. Як тільки все буде зроблено, зупиніть запис і макрос буде готовий для використання.

1.2 Формати файлів для збереження проєктів VBA

Код VBA зберігається у *проєктах*. Кожен застосунок Microsoft Office (далі Microsoft будемо писати скорочено – MS) зберігає проєкти по-своєму:

- MS Word зберігає проєкти у документах і шаблонах з форматом Open XML з підтримкою макросів (розширення файлів – DOCM і DOTM) та у документах і шаблонах з бінарним форматом (розширення файлів – DOC і DOT).
- MS Excel зберігає проєкти у робочих книгах і шаблонах з форматом Open XML з підтримкою макросів (розширення файлів – XLSM і XLTM) та у робочих книгах і шаблонах з бінарним форматом (розширення файлів – XLS і XLT).

Рекомендується зберігання проєктів у форматі Open XML.

- MS Access зберігає проєкти у файлах бази даних (розширення файлів – ACCDB та MDB).
- MS PowerPoint зберігає проєкти в презентаціях і шаблонах з підтримкою макросів (PPTM, PPSM і POTM) і в презентаціях і шаблонах з бінарним форматом (PPT, PPS і POT).

1.3 Підключення засобів програмування

Для роботи з VBA у застосунках Microsoft Office, які її підтримують необхідно відобразити у головному меню команду *Розробник*, яка відкриває стрічку з засобами програмування на VBA. Це можна виконати послідовністю команд *Файл-Параметри-Настроювання стрічки* і встановити позначку біля опції *Розробник* у правому списку (Рисунок 4) [3].

Після цього можна буде відкривати редактор Visual Basic, виконуючи клікання по кнопці *Visual Basic* панелі інструментів команди *Розробник* головного меню (Рисунок 5) (клавіатурна комбінація запуску редактора Visual Basic – *Alt+F11*).

1.4 Вікна і панелі Редактору Visual Basic

Редактор Visual Basic містить наступні вікна і панелі (Рисунок 6) [2]:

- Вікно *Project* (*Провідника проєкт*у) – за замовчуванням воно відкрите і знаходиться у лівій верхній частині редактора Visual Basic. У цьому вікні можна додавати компоненти проєкту і виконувати з ними різні операції. Якщо вікно не відображається, воно може бути відкрите командою *View-Project Explorer* або клавіатурною комбінацією *Ctrl+R*, закривається воно стандартною кнопкою закриття вікна.
- Вікно *Properties* (*Властивостей*) – відображає властивості обраного у вікні *Project* компонента або властивості обраного на користувацькій формі

елемента керування, значення властивостей можна змінювати. Якщо вікно не відображається, воно може бути відкрите командою *View-Properties Window* або клавішою *F4*, закривається воно стандартною кнопкою закриття вікна.

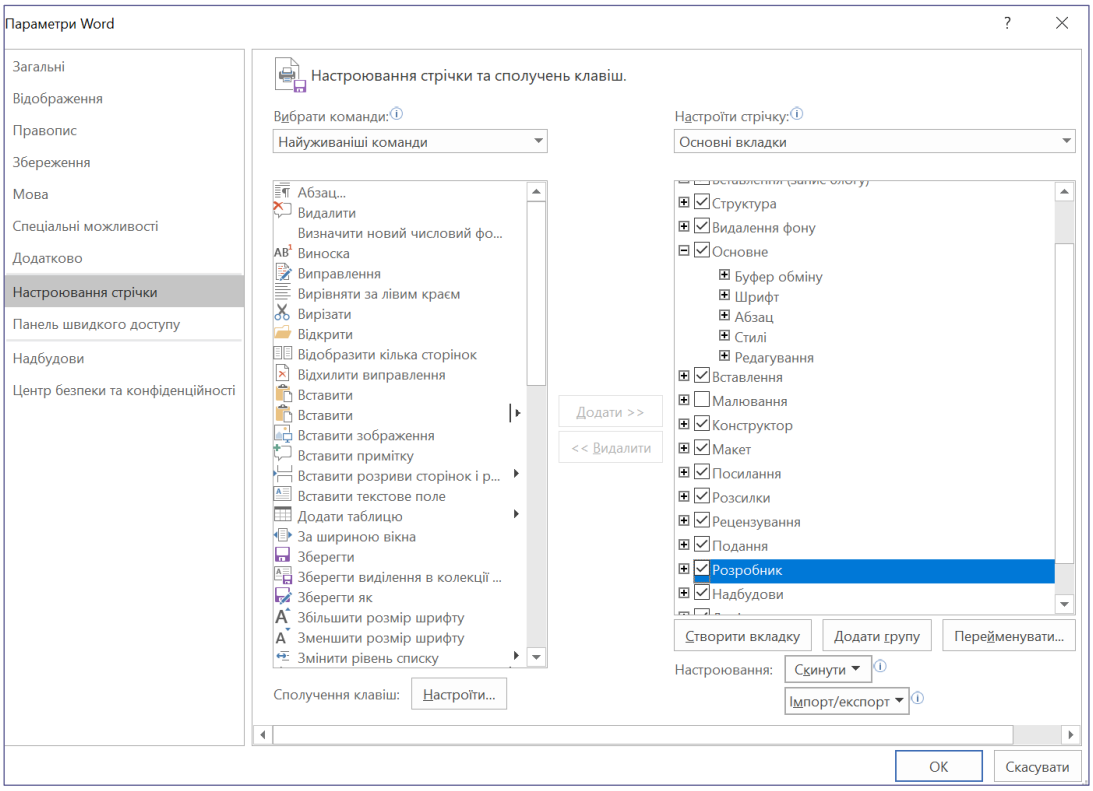


Рисунок 4 – Підключення засобів програмування на VBA

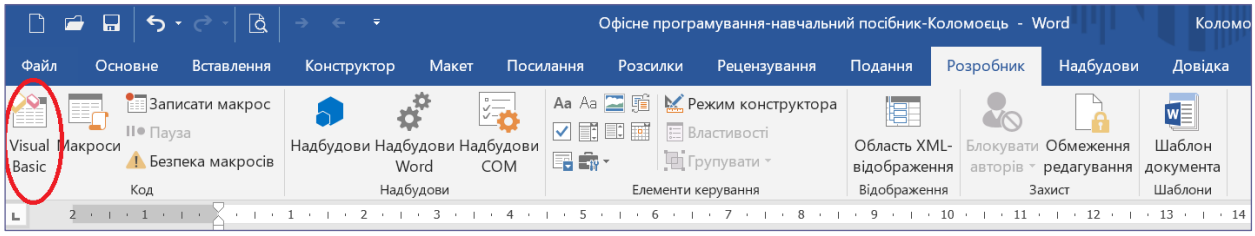


Рисунок 5 – Підключення засобів програмування на VBA

- Вікно *Code* (*Редактору вихідного коду*) – у цьому вікні виконується основна робота з розробки коду програм, прийоми роботи з цим вікном будуть розглянуті у цьому розділі. Якщо вікно не відображається, воно може бути відкрите командою *View-Code* або клавішою *F7*, закривається воно стандартною кнопкою закриття вікна.

- Вікно *Immediate* (*Негайних результатів*) – вікно, у якому можна вводити різні операції, викликати функції та негайно отримувати результат (Рисунок 7). Але зазвичай це вікно виконує роль консолі і відображає результати роботи програми, що виводяться командою `Debug.Print` (її використання буде розглянуте нижче). Додати це вікно можна командою *View-Immediate Window* або клавіатурною комбінацією *Ctrl+G*, закривається воно стандартною кнопкою закриття вікна.

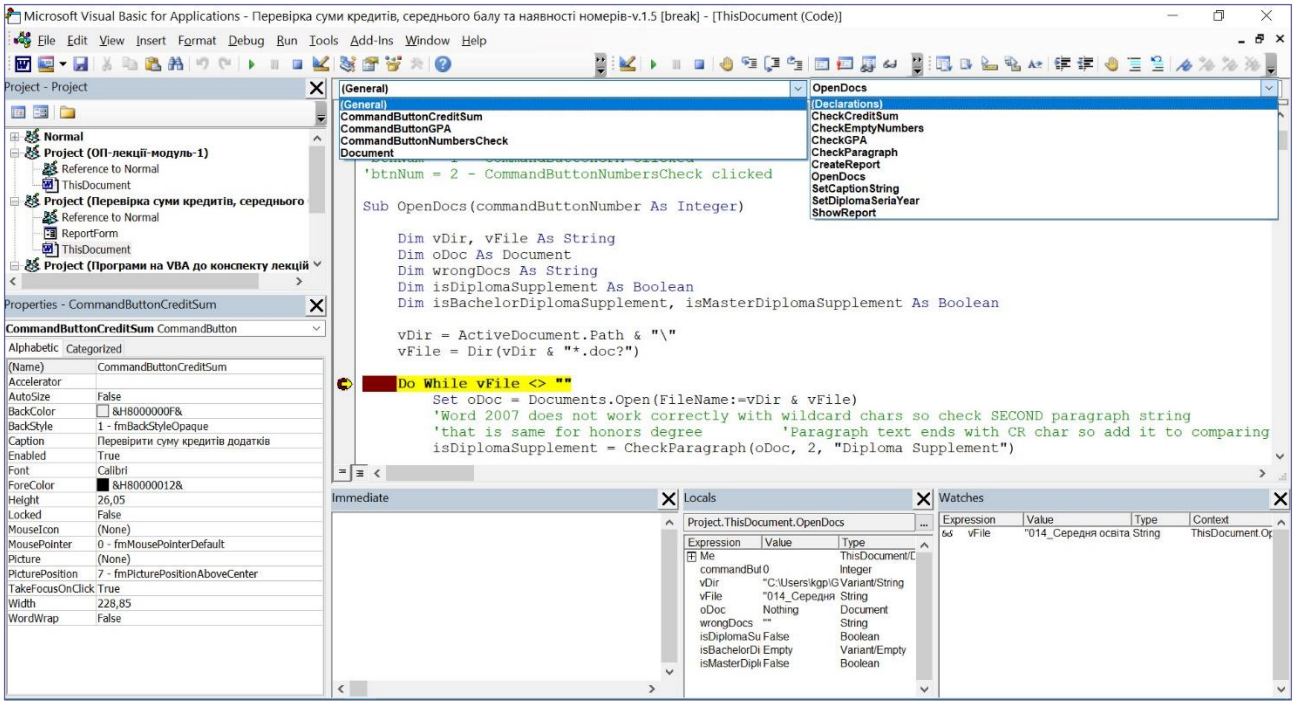


Рисунок 6. Елементи Редактора Visual Basic

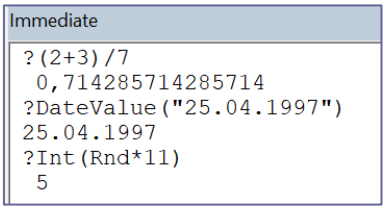


Рисунок 7 – Вікно негайних обчислень Редактора Visual Basic

- Вікно *Locals* (*Локальних змінних*) – відображає при відлагодженні програм значення усіх локальних змінних поточної процедури. Додати це вікно можна командою *View-Locals Window*, закривається воно стандартною кнопкою закриття вікна.

- Вікно *Watch* (*Спостереження*) – відображає при відлагодженні програм значення змінних та операторів/виразів поточної процедури, які обрані для спостереження. Додати це вікно можна командою *View-Watch Window*, закривається воно стандартною кнопкою закриття вікна.

Панелі *Toolbars* (*Інструментів керування*) – містять кнопки, що використовуються для ініціалізації різних дій у редакторі Visual Basic, наприклад, збереження, запуску, зупинки програми, встановлення контрольних точок та покрокового проходження при відлагодженні, доданні та видаленні коментарів тощо. Існують три панелі: *Standard* (*Стандартна*), *Debug* (*Відлагоджувача*) та *Edit* (*Редагування*), додавати та видаляти їх до вікна Редактора Visual Basic можна командою *View-Toolbars* (ми рекомендуємо їх додати та розмістити так, як показано на Рисунок 6.

- Спеціалізованою панеллю є панель *UserForm* (*Користувацької форми*) – вона автоматично з'являється тоді, коли редагується користувацька форма,

додана до проєкту (існує можливість виведення цієї панелі командою *View-Toolbars*).

- Вікно *Object Browser* (*Оглядача об'єктів*) можна відкрити, натискаючи клавішу *F2*, – у ньому відображаються всі доступні для використання модулі, класи, їх властивості, методи та події, а також перечислення та їх елементи (Рисунок 8). Користувацькі класи/перечислення та їх члени/елементи виділені жирним шрифтом. Закривається це вікно стандартною кнопкою закриття вікна.

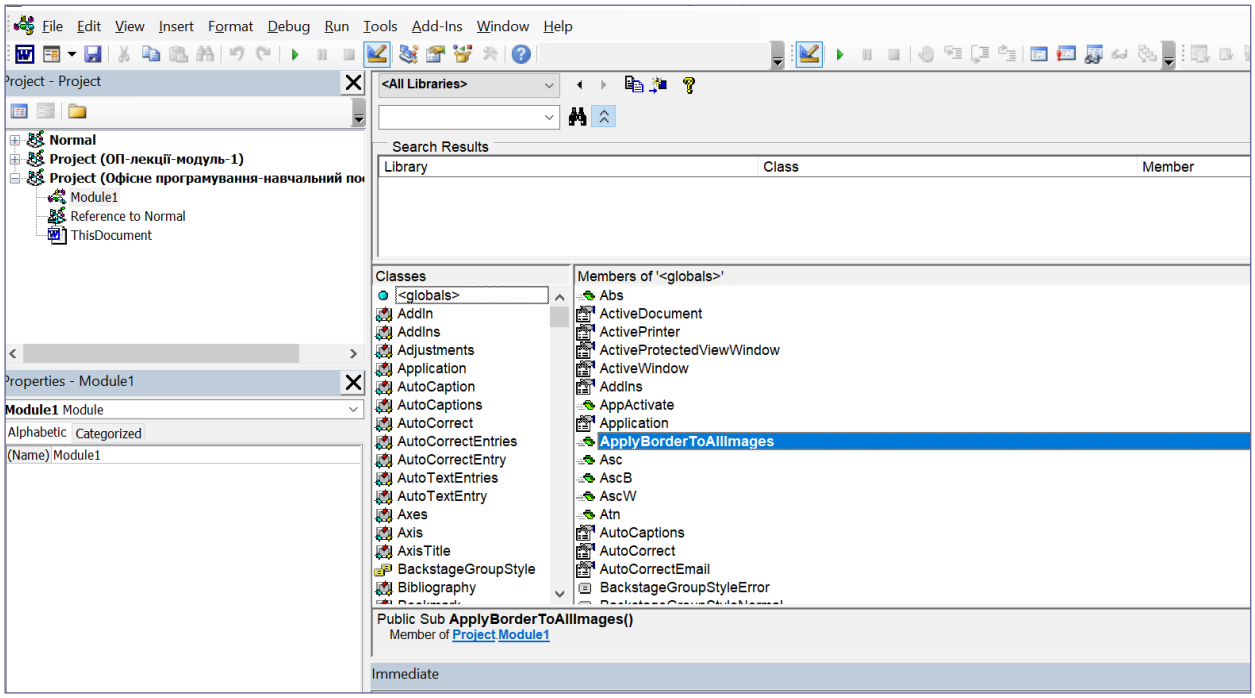


Рисунок 8 – Вікно Оглядача об'єктів

1.5 Вікно Провідника проєктів та елементи проєкту VBA

Вікно *Провідника проєктів (Project Explorer)* при запуску редактора Visual Basic зазвичай відкрито. У цьому вікні *Project Explorer* представлено дерево компонентів застосунків на VBA [3]. Самий верхній рівень - це *проєкт* (*Project*), якому відповідає документ Word, робоча книга Excel, презентація PowerPoint або інші відкриті файли застосунків, які підтримують програмування на VBA. Наприклад, якщо Ви відкрили редактор Visual Basic з Microsoft Word, то у вікні *Провідника проєктів* будуть представлені всі відкриті на даний час документи Microsoft Word плюс шаблон Normal.dot. Якщо редактор Visual Basic відкритий з Microsoft Excel, то у вікні *Провідника проєктів* будуть представлені відкриті книги Microsoft Excel плюс особиста книга макросів PERSONAL.XLSB (див. вище).

Існує можливість налаштувати властивості проєкту: ім'я проєкту, опис проєкту, посилання на файл довідки для проєкту, параметри компілятора, а також захистити проєкт від перегляду, задавши пароль (Рисунок 9). Для цього потрібно клікнути у вікні *Провідника проєктів* правою кнопкою миші по вузлу

Project (*VBAProject* в Excel) та у контекстному меню вибрати *Project Properties* (те саме вікно можна відкрити і через меню *Tools - Project Properties*).

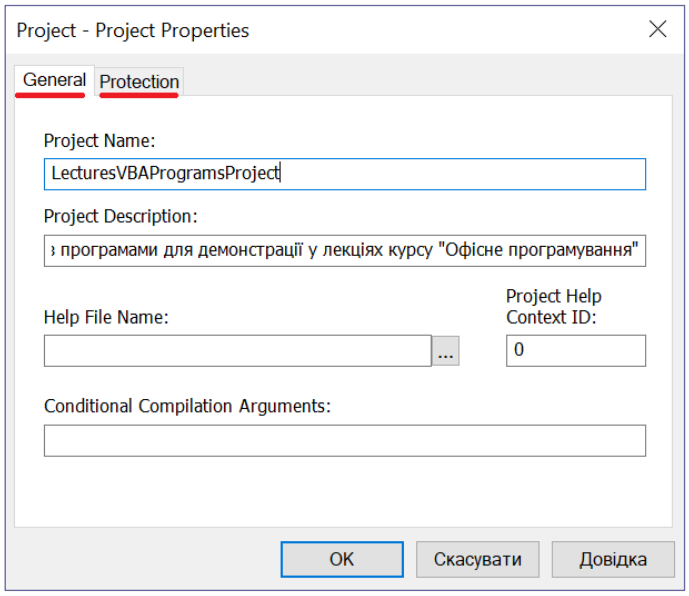


Рисунок 9 – Налаштування властивостей проєкту

Кожен файл застосунків Microsoft Office одночасно є контейнером для зберігання *елементів проєкту VBA*, до яких відносяться [2]:

Модулі – служать для розміщення в них процедур. Зазвичай у одному модулі розміщуються процедури, що забезпечують вирішення однієї прикладної задачі.

Модулі класу – служать для розміщення в них опису класів, з яких породжуються об'єкти, використовувані у програмах. Кожен модуль класу містить опис одного класу. Таким чином, мову VBA можна віднести до мов об’єктно-орієнтованого програмування (ООП), хоча VBA підтримує не всі технології, про що буде викладено у третій частині курсу.

Користувацькі форми – засоби програмування, що дозволяють створювати програми з графічним інтерфейсом користувача і використовувати у формах різноманітні елементи керування: текстові поля, списки, кнопки тощо. Зазвичай розробка таких програм зводиться до написання процедур обробки подій взаємодії користувача з такими елементами керування, детальний розгляд створення форм та роботи з елементами керування буде розглянутий у четвертій частині курсу.

Додати до проєкту кожен з цих компонентів можна за допомогою команди *Insert* головного меню редактора Visual Basic або через контекстне меню у *Провіднику проєктів*. У кожному проєкті VBA в Microsoft Word за замовчуванням міститься посилання на Normal (тобто шаблон normal.dot), що дозволяє, у разі збереження програм у цьому шаблоні, використовувати їх для усіх документів Microsoft Word.

1.6 Вікно Редактора вихідного коду та способи написання і запуску програм

Редактор вихідного коду – це по суті звичайний текстовий редактор, і в ньому Ви можете створювати процедури, вводити до них та редагувати оператори коду, копіювати, вирізати і вставляти код, коментувати/розкоментувати оператори тощо. Редактор відкривається у вікні *Code* для обраного у *Провіднику проєктів* елемента проєкту VBA і містить код процедур цього елемента [2].

Процедури – представляють собою пойменовану послідовність сумісно виконуваних інструкцій (операторів). Існують такі типи процедур: підпрограми (зазначаються `Sub`), функції (зазначаються `Function`) та властивості (зазначаються `Property`), всі вони будуть детально розглянуті нижче. Макроси зберігаються у підпрограмах, які не мають параметрів. Ім'я процедури повинно бути унікальним на рівні елемента проєкту VBA.

Для додавання процедури до обраного елемента проєкту VBA у вікні редактора вихідного коду встановити курсор у місце додання процедури та виконати команду *Insert–Procedure…* Після цього у вікні у вікні *Add Procedure*, що з'явиться, вибрати тип процедури (*Sub*, *Function* або *Property*) та заповнити її властивості: ім'я процедури, модифікатор видимості (*Public* – загальнодоступна або *Private* – доступна лише у поточному модулі), модифікатор *Static* для локальних змінних (за необхідності) (Рисунок 10) [2].

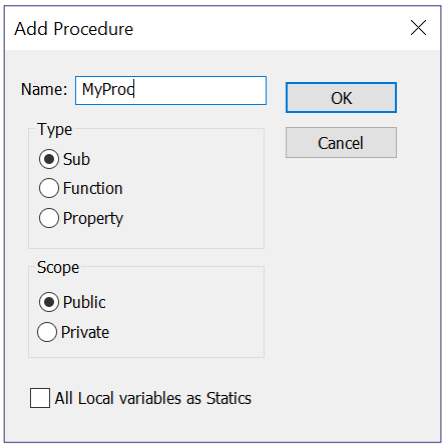


Рисунок 10 – Вікно вставки процедури

В результаті у вікні модуля виводиться оператор оголошення процедури (за замовчуванням підпрограми) – `Sub` з заданим ім'ям:

```
Sub ім'я_процедури()

End Sub
```

У тілі процедури вводяться необхідні оператори, завершальним оператором процедури є `End Sub`.

Запустити процедуру або форму на виконання можна розташувавши курсор миші на одному з операторів процедури (або відкривши форму кліком по ній у вікні *Провідника проєкту*) та натиснувши клавішу *F5* або клікнувши по кнопці  на стандартній панелі інструментів або на панелі відлагодження (також існує команд меню *Run-Sub/UserForm*).

У разі зависання процедури або при її відлагодженні (процес відлагодження буде розглянутий нижче), можна зупинити процедуру кліком по кнопці  на стандартній панелі інструментів або на панелі відлагодження (також існує команд меню *Run-Reset*).

У верхній частині вікна редактора вихідного коду знаходяться два списки: зліва – *список об'єктів* і справа – *список процедур/подій*. Зі списку об’єктів можна обрати об'єкт, до якого буде застосовуватись код, що відображається у вікні редактора.

У разі, якщо відкритий програмний код звичайного модуля, тут буде тільки елемент (*General*). При його обранні Ви побачите у списку *процедур/подій* елемент (*Declarations*) – *оголошення рівня всього модуля* та список усіх процедур модуля. При обранні елементу (*Declarations*) у вікні коду буде виведений код усіх процедур модуля/користувацької форми. При обранні певної процедури зі списку процедур/подій, код буде прокручений саме до цієї процедури. У разі відкриття коду користувацької форми, у списку об'єктів будуть відображатися елементи, що відповідають самій формі та її елементів керування, при обрані якогось з елементів у списку *процедур/подій* будуть знаходитись імена подій для обраного елемента. При обранні необхідної події зі списку автоматично буде створена заготовка необхідної процедури-обробника цієї події.

Редактор вихідного коду, подібно професійним середовищам розробки, надає можливість обрання необхідних властивостей/методів об’єкта при введенні крапки після імені об’єкта. Обравши необхідну властивість/константу/метод (можна надрукувати перші кілька букв для скорочення списку пропозицій), необхідно натиснути клавішу Tab і властивість/метод будуть додані до коду. Ця можливість включається параметром *Auto List Members* діалогового вікна *Options,* яке відкривається командою меню *Tools-Options,* і за замовчуванням є включеною.

Ключові слова VBA допомагає вводити автоматичне доповнення слів, яке реалізується введенням клавіатурної комбінації *Ctrl+Пробіл* після набору декількох перших букв слова. Якщо редактор вихідного коду розпізнає ключове слово, він автоматично робить його першу букву заголовною і виділяє синім кольором.

Корисною можливістю є отримання у всплиаючій підказці опису сигнатури методу та типу результату, що він повертає, яка реалізується клавіатурною комбінацією *Ctrl+I* після розміщення курсору на імені метода. Ця можливість включається параметром *Auto Quick Info* діалогового вікна *Options,* яке відкривається командою меню *Tools-Options* і за замовчуванням є включеною.

У разі, якщо необхідно ввести ім’я константи, яка може бути використана як значення властивості або аргумент метода (константи Visual Basic починаються з префіксу vb), можна скористатися клавіатурною комбінацією *Ctrl+Shift+J*. При цьому відкриється список доступних констант, з якого необхідну можна обрати клавішею *Tab* або кліком миші.

Панель інструментів керування редагуванням (*Edit*) містить кнопки які дозволяють закоментувати (*Comment Block*) та розкоментувати (*Uncomment Block*) виділені рядки коду.

При переході на новий рядок редактор вихідного коду перевіряє синтаксис введеного коду і у разі його порушення видає попередження. Оскільки воно часто видається для незавершеного оператора, що розташовується на декількох рядках, існує можливість відключення попередження відключенням параметру *Auto Syntax Check* у діалоговому вікні *Options*. Навіть при відключенні попередження синтаксично невірні рядки автоматично виділяються червоним кольором.

1.7 Робота з довідкою з VBA та API офісних застосунків

Корисною можливістю є виклик довідки для частини оператора (ключового слова, об’єкта класу, метода, властивості, константи тощо) після встановлення курсору на цей елемент коду – реалізує цю можливість натискання на клавішу *F1*. При цьому відкриється вікно браузера аналогічне представленому на Рисунок 11 [3].

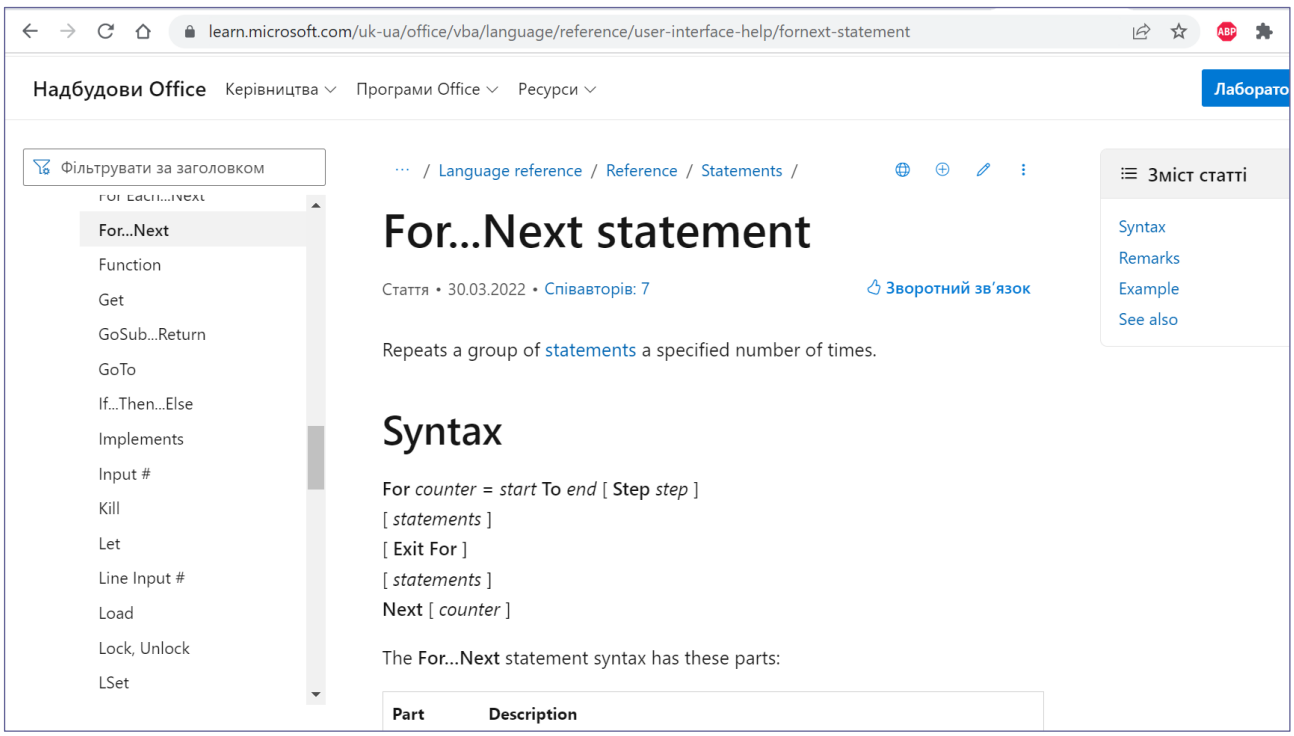


Рисунок 11 – Довідка для ключових слів VBA

Довідка складається з трьох частин:

- довідка по API об'єктних моделей застосунків Microsoft Office – *Excel Reference, Word Reference* тощо, складається з розділів *Concepts (концепції),* у яких розповідається, як програмним чином виконувати найпоширеніші операції у відповідному застосунку, та *Object model reference (довідка по компонентам об'єктної моделі застосунку)*: колекціям, об'єктам, властивостям, методам тощо;
- довідка по синтаксису і вбудованим функціям мови VBA – *Language reference*, найкориснішим розділом є *Reference*, у якому містяться довідки по ключовим словам мови програмування (операторам, вбудованим функціям та структурам даних, опціям тощо);
- довідка по API об'єктної моделі всього Microsoft Office – *Library reference*, у якій приводяться довідки для загальних для застосунків Microsoft Office об’єктів API.

1.7 Налаштування для забезпечення копіювання фрагментів коду з нелатинськими символами

Однією з проблем, яка може виникати при копіюванні з Редактору Visual Basic фрагментів коду, які містять нелатинські символи (що зазвичай виконується при підготовці звітів з лабораторних робіт), є неправильне відображення таких символів внаслідок того, що редактор підтримує стару систему кодування символів ANSI з кодовими таблицями замість сучасної Unicode. Вирішити цю проблему можна налаштуваннями операційної системи [12]. Так, для Windows необхідно з *Панелі керування* обрати *Регіон* і у вікні *Регіональні настройки* на вкладці *Адміністрування* у розділі *Мова програм, які не підтримують Юнікод* клікнути на кнопці *Змінити мовні параметри системи* та обрати мову, якою будуть вводитися нелатинські символи до Редактору Visual Basic (Рисунок 12). Після цього нелатинські символи з редактора будуть копіюватися правильно, але тільки у тому випадку, коли у системі активна відповідна розкладка клавіатури (для наведеного прикладу – українська).

Питання для самоконтролю

1. Що називають макросом?
2. Опишіть алгоритм дій при запису макросу.
3. Які властивості можна призначити макросу?
4. Де можуть зберігатися макроси для Microsoft Word та Microsoft Excel для їх використання з різними файлами цих застосунків?
5. Назвіть формати файлів Microsoft Word, Microsoft Excel, Microsoft Access та Microsoft PowerPoint (їх розширення), які дозволяють зберігати програми на мові VBA.
6. Опишіть спосіб підключення засобів програмування на VBA у застосунках Microsoft Office.
7. Опишіть призначення та способи відкриття вікон та панелей Редактору Visual Basic.

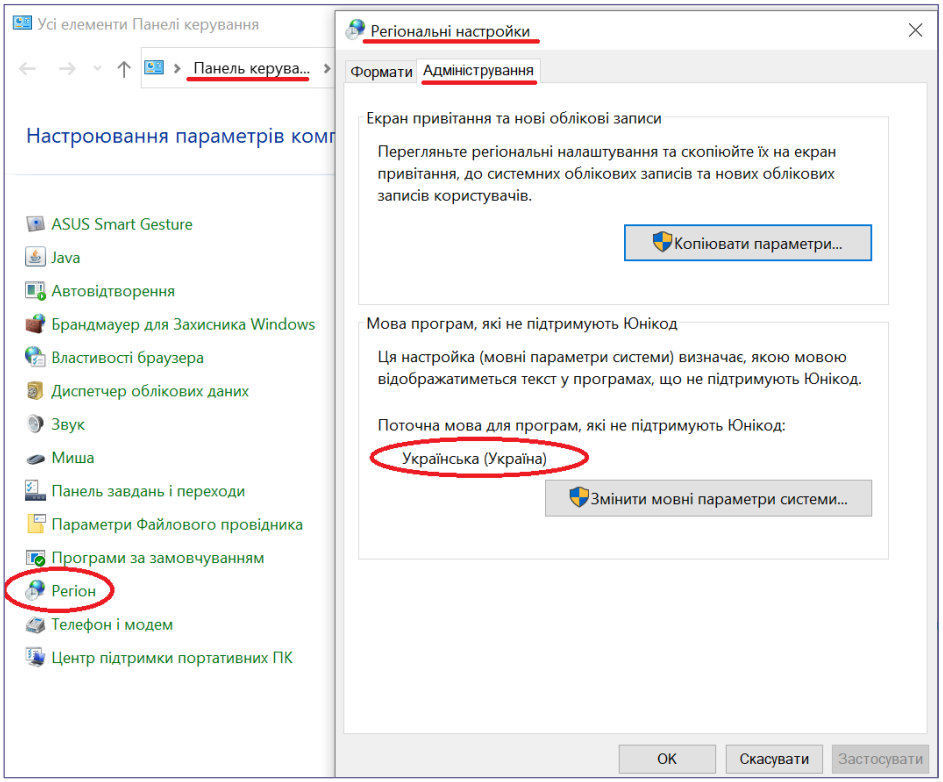


Рисунок 12 – Налаштування копіювання нелатинських символів з Редактора Visual Basic

8. Що являє собою проєкт VBA? Які властивості можна призначити проєкту VBA?
9. Назвіть відомі Вам елементи проєкту VBA та надайте їм коротку характеристику. Назвіть команду додання цих елементів до проєкту.
10. Що являють собою процедури VBA? Які типи процедур Ви знаєте?
11. Якою командою можна додавати процедури до елементів проєкту VBA? Які властивості процедури можуть бути призначені при її доданні?
12. Назвіть способи запуску та зупинки процедури.
13. Як працювати зі списком об’єктів та списком процедур/подій редактора вихідного коду?
14. Назвіть клавіатурні комбінації для автодоповнення ключових слів при введенні коду, отримання опису сигнатури методу та типу результату, що він повертає, та отримання переліку доступних констант. Якою клавішею обирається необхідний елемент з пропонованих?
15. З яких частин складається довідка з VBA та API застосунків Microsoft Office? Якою клавішею викликається довідка?
16. Опишіть спосіб забезпечення правильного копіювання нелатинських символів з вікна Редактору Visual Basic.

Тестові завдання для перевірки знань

1. Визначте властивості, які можна призначити макросу (виберіть одну або декілька відповідей):
   а) ім’я макросу;
   б) клавіатурну комбінацію для запуску макросу;

в) зображення-іконку для макросу;
г) місце збереження макросу;
д) опис макросу;
е) параметри (аргументи), що передаються макросу при запуску.

2. Оберіть розширення файлів документів Microsoft Word та робочих книг Microsoft Excel, які дозволяють зберігати макроси (виберіть одну або декілька відповідей):
а) .docm;
б) .doc;
в) .docx;
г) xlsm;
д) .xls;
е) .xlsx

3. Оберіть назву вікна, яке виконує функції, аналогічно консолі (виберіть одну відповідь):
а) Immediate;
б) Locals;
в) Watch;
г) Console;
д) Output.

4. Оберіть інформацію, яка відображається у вікні Оглядача об’єктів (Object Browser) (виберіть одну або декілька відповідей):
а) класи API застосунків Microsoft Office;
б) користувацькі класи;
в) інтерфейси API застосунків Microsoft Office;
г) користувацькі інтерфейси;
д) перечислення та їх елементи;
е) властивості класів;
ж) модулі проєктів.

5. Визначте властивості, які можна призначити проєкту (виберіть одну або декілька відповідей):
а) ім’я проєкту;
б) опис проєкту;
в) файл довідки для проєкту;
г) параметри компіляції проєкту;
д) зображення-іконку для проєкту;
е) місце збереження проєкту.

6. Оберіть тип процедури, у якій зберігаються макроси (виберіть одну відповідь):
а) підпрограма (`Sub`);
б) функція (`Function`);
в) властивість (`Property`).

7. Оберіть тип елемента проєкту, для якого у списку об’єктів редактора вихідного коду відображається єдина опція (General) (виберіть одну відповідь):
а) звичайний модуль;

б) модуль класу;
в) користувацька форма.

8. Вкажіть клавішу, якою обирається необхідна властивість/константа/метод з пропонованого списку автодоповнення ключових слів (виберіть одну відповідь):
   а) Enter;
   б) Tab;
   в) пробіл.
9. Оберіть клавіатурну комбінацію, що відображає у вспливаючій підказці сигнатуру те тип результату, що повертається методом, на якому встановлений курсор (виберіть одну відповідь):
   а) *Ctrl+Пробіл*;
   б) *Ctrl+I*;
   в) *Ctrl+Shift+I*;
   г) *Ctrl+Shift+J*.
10. Оберіть правильний спосіб вирішення проблеми копіювання нелатинських символів із вікна Редактору Visual Basic (виберіть одну відповідь):
    а) обрання шрифту, що підтримує Unicode кодування, у вікні Options на вкладинці Editor Format;
    б) обрання у регіональних налаштуваннях операційної системи як поточної мови для програм, які не підтримують Unicode кодування, мови, яка застосовується у Редакторі Visual Basic;
    в) обрання шрифту, що підтримує Unicode кодування, у файлі застосунку Microsoft Office, який містить проєкт VBA.

Завдання для самостійного виконання

1. Запишіть макрос, який форматує числове значення у клітинці до як число з трьома знаками після коми. Значення повинно виводитись червоним кольором, ширина стовпця з клітинкою повинна дорівнювати 15 пунктів.
2. Запишіть макрос, який копіює довільний виділений діапазон клітинок на Аркуші1 на Аркуш2, починаючи з клітини з адресою B12, після чого додає усі стандартні границі до скопійованого діапазону на Аркуші2.
3. Запишіть макрос, який додає 3 нових аркуші у поточну робочу книгу і змінює їх назви усіх аркушів на Таблиця-N, де N = 1, 2, ... до кількості аркушів.
4. Запишіть макрос, який, починаючи з поточної позиції, вводить у поточний стовпець геометричну прогресію від 1 до 12400 з кроком 16, розмір шрифту 18, колір червоний, ширину стовпця зробіть мінімальною для відображення усіх чисел прогресії.
5. Запишіть макрос, який, починаючи з поточної позиції, вводить назви дванадцяти місяців зеленим кольором, розмір шрифту 12.
6. Запишіть макрос, який дані у виділеній області приводить до грошового формату з зазначенням грн. як назви валюти, гарнітура шрифту даних Arial, напівжирний, розмір 11, символи світло-зеленого кольору
7. Запишіть макрос, який, починаючи з поточної позиції, вводить у поточний стовпець арифметичну прогресію від 1 до 312 з кроком 37, а у сусідній –

квадрат значення у відповідній клітині зліва, після чого додає усі стандартні границі до доданих значень.

8. Запишіть макрос, який переносить виділену область на десять клітинок вниз і захищає аркуш паролем qwerty від подальшої зміни.
9. Запишіть макрос, який заповнює діапазон клітин з адресою A1:E5 послідовними значеннями від 1 до 25, після чого форматує шрифт доданих значень: колір – червоний, розмір 18, написання – напівжирний та додає усі стандартні границі до вказаного діапазону.
10. Запишіть макрос, який дані у виділеній області виводить червоним кольором (курсив, напівжирний) на світло-блакитному фоні.

ТЕМА 2 БАЗОВІ ЗАСОБИ МОВИ ПРОГРАМУВАННЯ VBA

Мета: ознайомитись з правилами іменування елементів коду програм на VBA, вивчити типи даних, способи оголошення змінних та констант, принципи застосування операторів, способи оголошення та роботи з масивами, типи та правила оголошення і використання процедур, основні вбудовані функції VBA.

План

1. Правила іменування елементів коду на VBA.
2. Типи даних VBA.
3. Оголошення змінних та констант VBA, коментарі у програмі.
4. Арифметичні оператори, оператори порівняння та логічні оператори VBA, пріоритети операторів.
5. Оператори керування програмою на VBA.
6. Організація роботи з масивами.
7. Процедури VBA: підпрограми та функції, передача аргументів та повернення значень.
8. Вбудовані функції VBA.

Основні терміни і поняття

Тип даних, змінна, константа, оператор, масив, процедура, підпрограма, функція.

Основні теоретичні положення

2.1 Правила іменування елементів коду VBA

Приведемо правила VBA щодо іменування процедур, функцій, змінних, констант, перечислень та їх елементів [13]:

- ім'я кожного елементу коду повинно бути унікальним у поточній області видимості;
- не рекомендується обирати імена, що збігаються з ключовими словами VBA;
- імена можуть містити літери алфавіту, цифри та знак підкреслення, деякі інші спеціальні символи можуть використовуватися у іменах але це не рекомендується, оскільки може привести к непередбачуваним проблемам;
- імена повинні починатися з літери алфавіту (можуть починатися зі знаку підкреслення, але це не рекомендується);
- максимальна довжина імені – 255 символів;
- мова VBA нечутлива до регістру символів, однак рекомендується притримуватися однакового стилю іменування: іменувати проєкти, модулі, класи, перечислення, користувацькі форми, елементи керування форм, процедури, властивості з великої літери, у разі коли ім’я складається з декількох слів, кожне слово рекомендується писати з великої літери без пробілів, наприклад, `FindLastDigit()`;

- імена змінних, констант та елементів перечислення рекомендується також писати, обравши єдиний стиль іменування: або починаючи з маленької літери – так звана «верблюжа нотація» або з великої літери – «нотація Pascal» або з префіксом, який вказує на тип змінної або константи – «угорська нотація» [14]. Ми рекомендуємо використовувати для звичайних змінних та констант «верблюжу нотацію», наприклад, `myVar`, а для назв об'єктів елементів керування користувацьких форм – «угорську нотацію», її особливості будуть розглянуті при вивченні користувацьких форм.

2.2 Типи даних VBA

Типи даних VBA можуть бути класифіковані таким чином [1]:

Вбудовані – типи даних, які мають строго визначені у байтах розміри (Таблиця 1), та *визначені користувачем* – типи даних, що, зазвичай, складаються з даних вбудованих типів та/або інших визначених користувачем типів.

Прості – типи даних, які можуть приймати єдине значення, та *складені* – типи даних, які можуть приймати множини значень даних.

Статичні – типи даних, для яких пам'ять відводиться в момент їх оголошення, та *динамічні* – типи даних, для яких розмір даних в момент оголошення не відомий, і пам'ять їм виділяється при виконанні програми [16].

Таблиця 1 – Вбудовані типи даних мови VBA*

Ім'я типу	**Можливі значення**	**Необхідна пам'ять**
`Boolean`	`True, False`	2 байти
`Byte`	0..255	1 байт
`Integer`	-32 768 ... + 32 767	2 байти
`Long`	-2 147 483 648 ... + 2 147 483 647	4 байти
`Decimal`	+/-79 228 162 514 264 337 593 543 950 335 у разі десяткового цілого числа (29 цифр) +/-7.9228162514264337593543950335 у разі десяткового дробу (28 цифр у дробовій частині) Найменше ненульове значення: +/-0.0000000000000000000000000001 (28 цифр у дробовій частині)	14 байтів
`Single`	-3.402823**E38** ... -1.401298**E-45** для негативних значень; 1.401298**E-45** ... 3.402823**E38** для позитивних значень	4 байти
`Double`	-1.79769313486231**E308** ... -4.94065645841247**E-324** для негативних значень; 4.94065645841247**E-324** ... 1.79769313486232**E308** для позитивних значень	8 байтів
`Currency`	-922 337 203 685 477.5808 ... 922 337 203 685 477.5807 (15 десяткових цифр для цілих та 3 – для дробових)	8 байтів

Ім'я типу	Можливі значення	Необхідна пам'ять
`String`	Існує два види рядків: *рядки фіксованої довжини* мають від 1 до приблизно 65 400 символів; *рядки змінної довжини* мають до приблизно 2 мільярдів символів	1 байт на символ 10 байтів + 1 байт на символ
`Date`	Від 1 січня 100 року до 31 грудня 9999 року. Зберігається, як `Double`, ціла частина зберігає число днів від початкової дати, а дробова частина зберігає час від півночі. Початкова дата – 30.12.1899, більш ранні дати зберігаються як від'ємне число.	8 байтів
`Object`	Посилання на об'єкт	4 байти
`Variant`	Універсальний тип, значенням якого можуть бути дані любого з перерахованих вище типів розміром до `Double` для чисел та до максимального рядка `String` змінної довжини для символів, об'єкти, значення NULL і значення помилок ERROR	16 байтів для чисел, 22 байти + довжина рядка для символів

* Існує також тип даних `LongPtr`, але його розмір та діапазон залежить від розрядності Microsoft Office (32 або 64) [15].

2.3 Оголошення змінних та констант VBA, коментарі у програмі

Програми на VBA оформлюють як *процедури*, найпростіші процедури – *підпрограми* починаються з ключового слова `Sub` після якого міститься ім'я підпрограми з круглими дужками після нього, закінчується код процедури оператором `End Sub` [1]:

```
Sub MyProc()
    код підпрограми
End Sub
```

Детальніше типи процедур будуть розглянуті нижче у поточній темі.

Змінною називають елементу коду, пов'язаний з певною областю пам'яті, що зберігає дані вбудованого або користувацького типу, які можуть змінюватися при виконанні програми. Оголошення простої змінної *у процедурі* – такі змінні називають *локальними* – має наступний синтаксис:

```
Dim ім'я_змінної As ім'я_типу
```

Оголошення локальних змінних рекомендується розташовувати на початку процедури, робити відступи у 4 пробіли від лівої границі вікна та відокремлювати пустим рядком від наступного коду, наприклад:

```
Sub MyProc()
    Dim bool As Boolean
    Dim i As Integer, str As String, dat As Date

    наступний код
End Sub
```

Як видно, можна оголошувати одну змінну у кожному рядку, а можна – декілька змінних. VBA ініціалізує змінні при їх оголошенні (до присвоєння їм значень) наступними значеннями:

- `0` – для числових значень та значень типу `Currency`;
- `False` – для значень типу `Boolean`;
- порожній рядок (`""`) – для рядків змінної довжини;
- рядок зазначеної довжини з символами ASCII `0` (ці символи на екран не виводяться) – для рядків фіксованої довжини;
- нульова дата та час, які при друку виглядають як `0:00:00`;
- `Nothing` (немає посилання ні на один з об'єктів) – для типу `Object`;
- `Empty` (значення, яке вказує на відсутність ініціалізації) – для типу `Variant`.

Присвоєння значень змінним у наступному за оголошенням коді виглядає так (зверніть увагу на правила оголошення літералів-рядків та літералів-дат – дата зазначається у форматі місяць/день/рік):

```
bool = True
i = 30
str = “Hello”
dat = #01/15/2021 8:45:00 AM#
```

Далі оголошені змінні можуть змінювати своє значення, наприклад, при використанні арифметичних операторів:

```
i = 30 * 5
```

VBA дозволяє працювати зі змінними без їх оголошення – у такому разі змінним призначається тип `Variant`. Також у разі, якщо після імені змінної конструкція `As` відсутня, змінній буде присвоєний тип `Variant`. Але така стратегія не рекомендується, оскільки, окрім зайвої витрати пам’яті, зменшується спроможність компілятора розпізнавати помилки, пов’язані з використанням неправильного типу. Існує опція

```
Option Explicit
```

зазначення якої в області декларацій модуля буде вимагати обов’язкового оголошення змінних. Ми наполегливо рекомендуємо використовувати таку опцію, і навіть вставляти цю команду автоматично при створенні модулів – встановивши у вікні Редактора Visual Basic прапорець *Require Variable Declarations* (меню *Tools - Options*, вкладка *Editor*).

Також змінні можуть оголошуватися *на рівні модуля* у відповідній області (*Declarations*), до якої можна перейти, обравши цей елемент зі списку процедур/подій редактора коду, – такі змінні можуть використовуватись всіма процедурами модуля і їх називають *глобальними*. Для оголошення глобальних змінних, крім оператора `Dim`, можуть використовуватись ключові слова `Private` або `Public`. Перше ключове слово оголошує глобальну змінну, доступну з усіх процедур тільки поточного модуля, а друге – з усіх процедур усіх модулів усіх відкритих у Редакторі Visual Basic проєктів. Для обмеження

доступу до `Public`-змінних тільки з модулів проєкту, до якого належить модуль зі змінною, в області декларацій цього модуля необхідно додати опцію:

```
Option Private Module
```

Оголошення змінних з використанням цих ключових слів буде розглянуто у темі, присвяченій роботі з класами.

Також зміна на рівні процедури може оголошуватись з використанням ключового слова `Static` – у такому разі змінна не видаляється з пам'яті по завершенню роботи процедури і може використовуватися іншими процедурами цього ж модуля.

Константи, на відміну від змінних, не змінюються в ході виконання VBA-програми, при спробі у тілі процедури змінити значення константи, буде видано повідомлення про помилку. Синтаксис оголошення констант має наступний вигляд, квадратними дужками зазначені необов'язкові частини, а символом | – можливі варіанти:

```
[Public | Private] Const ім'я_константи [As type] = константний_вираз
```

Приклад оголошення константи:

```
Public Const pi As Double = 3.14159
```

Як і змінні, константи можна оголошувати на рівні процедури або модуля. У першому випадку використовується тільки ключове слово `Const`, у другому – додатково можна задати специфікатор `Public` або `Private`. За замовчуванням глобальні константи мають статус `Private`.

В VBA вбудована велика кількість констант: календарних, для роботи з файлами, кольорами, формами тощо. Переглянути їх можна через довідкову систему VBA: *Office VBA language reference - Reference -> Constants* (Рисунок 13).

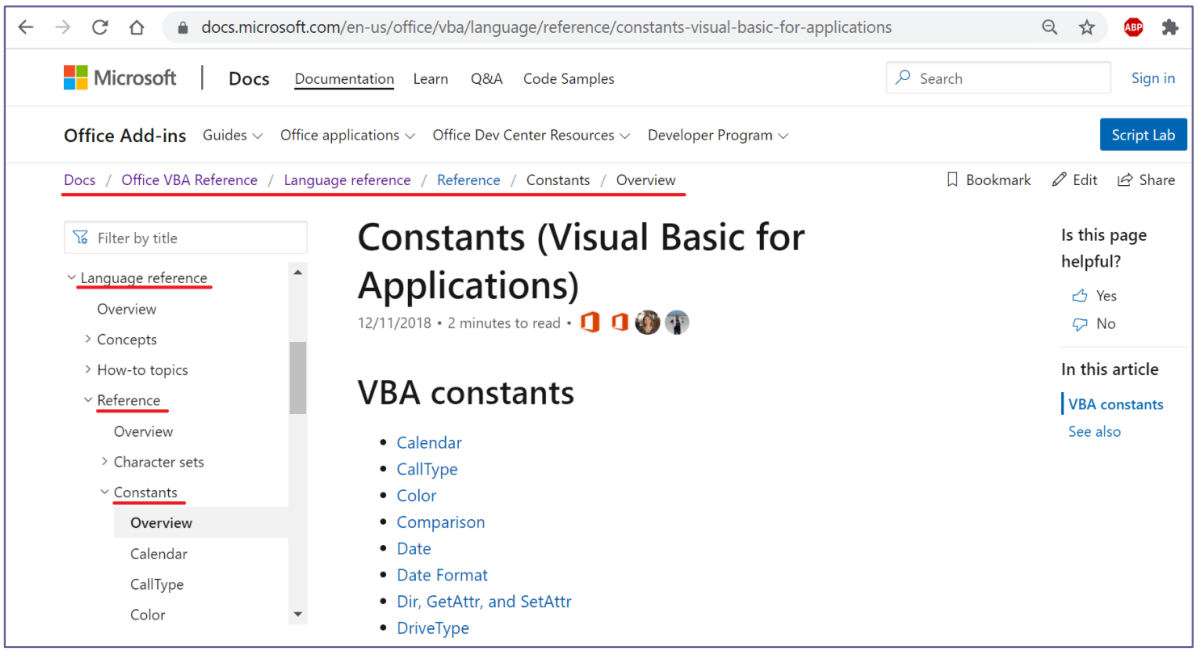


Рисунок 13 – Довідка з констант VBA

Про одну з констант (вона знаходиться в розділі *Miscellaneous*) слід сказати окремо: константа `vbCrLf` дозволяє зробити перехід на новий рядок, наприклад, наступний оператор виводить текст у вікно *Immediate* у два рядки:

```
Debug.Print "Перший рядок" + vbCrLf + "Другий рядок"
```

Велика кількість констант вбудована в об'єктні моделі, які будуть вивчатись у наступних темах.

Коментарі у програмах використовуються для узагальненого опису вмісту модуля, опису процедури, а також пояснення фрагментів коду та оголошень [3]. Окрім того, коментарі можуть використовуватися для тимчасового виключення фрагментів коду. У будь-якому разі коментар починається з символу одинарної верхньої лапки (або ключового слова `REM`) та подовжується до кінця поточного рядка, частіше застосовується одинарна лапка, наприклад:

```
'Демонструє роботу з масивом
Public Sub MyArray()
    подальший код
'   a = a + 1      'тимчасово виключений код
    подальший код
```

2.4 Арифметичні оператори, оператори порівняння та логічні оператори VBA, пріоритети операторів

Оголошені змінні та константи можна використовувати у процедурах, для цього зазвичай використовують *оператори* [3]. Необхідно зазначити, що вище наведені рядки з ключовими словами, які використані при оголошенні змінних та констант, також розглядаються як оператори.

Приведемо особливості написання операторів програм на VBA:

- початок нового оператора – це символ переведення на новий рядок (крапка з комою, як в C або Java для цього не використовується);
- можливо розмістити один оператор на декількох рядках, використовуючи для поділу оператора послідовність символів пробіл та знак підкреслення:

```
Debug.Print "Повідомлення користувачеві " _
       & vUserName
```

- можливо розмістити декілька операторів на одному рядку, розділивши їх символом двокрапки (він фактично імітує переведення на новий рядок):

```
MyVar = 5 : Debug.Print "MyVar=" & MyVar
```

- зазначення символу двокрапки у кінці оператора `Debug.Print` дозволяє виконувати наступний оператор у тому ж рядку (цей символ фактично пригнічує переведення на новий рядок):

```
Debug.Print "Значення myVar="; myVar
```

зазначення символу коми у кінці оператора `Debug.Print` дозволяє виконувати наступний оператор у тому ж рядку (цей символ фактично пригнічує

переведення на новий рядок), вставляючи символ табулятора (Tab) перед виведенням наступного оператора:

```
Debug.Print "Значення myVar=", myVar
```

Розрізняють оператори арифметичні, присвоювання, порівняння, логічні та оператори керування програмою.

Арифметичними операторами є:

- додавання (+);
- віднімання (-);
- множення (*);
- ділення (/) – завжди отримує число з плаваючою комою незалежно від типів чисельника та знаменника, яке виводиться як ціле у разі відсутності дробової частини;
- піднесення до ступеня (^), наприклад, `2 ^ 3 = 8` (результат має тип `Double`);
- цілочисельне ділення (\) - ділить перший операнд на другий, відкидаючи дробову частину, наприклад, `5 \ 2 = 2`;
- ділення по модулю (`Mod`) - ділить перше число на друге, повертаючи тільки залишок від ділення, наприклад, `5 Mod 2 = 1` (операція визначена над даними цілого типу та повертає результат цілого типу);
- унарного мінусу (-), унарного плюсу, як і інкременту, декременту та сумісних операцій +=, -=, *=, /= тощо в VBA не існує.

Оператор присвоювання в VBA зазначається знаком рівності (=), існує можливість використовувати додатково ключове слово `Let`:

```
Let myVar = 10
```

але зазвичай його не використовують:

```
myVar = 10
```

Операторами порівняння є:

- перевірка рівності (=), наприклад, `If (myVar = 10);`
- більше, ніж (>), наприклад, `If (myVar > 10);`
- менше, ніж (<), наприклад, `If (myVar < 10);`
- більше або дорівнює (>=), наприклад, `If (myVar >= 10);`
- менше або дорівнює (<=), наприклад, `If (myVar <= 10);`
- не дорівнює (<>), наприклад, `If (myVar <> 10);`
- оператор порівняння об'єктів (`Is`) – визначає, чи посилаються об'єктні змінні на той самий об'єкт або на різні об’єкти, наприклад, `If (obj1 is obj2);`
- оператор подібності (`Like`) порівнює і визначає, чи збігається рядок з шаблоном.

Оператори порівняння завжди повертають `True` – якщо ствердження істинне, або `False` – якщо ствердження помилкове.

Інформація про порівняння значень-рядків та використання оператору `Like` буде приведена нижче.

Логічні оператори зазвичай використовуються при перевірці комбінованих умов, кожна з яких реалізується оператором порівняння. Логічні оператори також повертають булеве значення `True` або `False`, до логічних операторів відносяться:

- `And` – логічне І – оператор повертає `True`, якщо обидві умови, які він об'єднує, повертають `True`, у іншому разі оператор повертає `False`;
- `OR` – логічне АБО – оператор повертає `False`, якщо обидві умови, які він об'єднує, повертають `False`, у іншому разі оператор повертає `True`;
- `Not` – логічне заперечення, повертає `True`, якщо умова, зазначена після цього оператора, повертає `False` та навпаки;
- `Xor` – логічний виняток, повертає `False`, якщо обидві умови, які він об'єднує, повертають однакове значення (`True` або `False`), у іншому разі оператор повертає `True`;
- `Eqv` – еквівалентність, повертає `True`, якщо обидві умови, які він об'єднує, повертають однакове значення (`True` або `False`), у іншому разі оператор повертає `False`;
- `Imp` – імплікація, повертає `False`, якщо перша умова (зазначена зліва від оператора) повертає `True`, а друга умова (зазначена справа від оператора) повертає `False`, у іншому разі оператор повертає `True`.

Майже в будь-якій програмі VBA використовується *оператор конкатенації*. В VBA цій оператор зазначається символом `&` або `+`, рекомендується використовувати `&`.

У разі, якщо у одному виразі використовуються декілька операторів починають діяти їх пріоритети, зазначені у таблиці 2 [3].

Таблиця 2 – Пріоритети операторів

Пріоритет	**Арифметичні**	**Порівняння**	**Логічні**
1	`^`	`= <> < > <= >= Like Is` – всі операції мають однаковий пріоритет	`Not`
2	унарний мінус `(-)`		`And`
3	`*, /`		`Or`
4	`\`		`Xor`
5	`Mod`		`Eqv`
6	`+, -`		`Imp`
7	`&` (не є арифметичною)		

При побудові виразів необхідно враховувати наступне:

- дужки дозволяють змінити вказаний порядок обчислення виразу, оскільки вирази в дужках мають найвищий пріоритет і обчислюються першими, усередині дужок діють звичайні пріоритети;
- якщо вираз містить оператори різних типів, то спочатку виконуються арифметичні операції, потім операції порівняння, і останніми логічні;
- всі операції порівняння мають однаковий пріоритет і виконуються у виразі зліва направо;

- одна і та ж операція, записана кілька разів поспіль, або операції одного пріоритету виконуються зліва направо;
- операція конкатенації виконується після усіх арифметичних операцій, але до обчислення операцій порівняння;
- оператори `Not`, `And`, `Or` і `Xor` можуть також використовуватись для роботи з числами у двійковому/вісімковому та шістнадцятковому форматах.

2.5 Оператори керування програмою на VBA

Набір операторів керування VBA включає оператори розгалуження та цикли різних типів, які дозволяють організувати нелінійну передачу керування у програмах [2].

Оператор розгалуження `If Then Else` дозволяє направляти обчислення по одній з гілок в залежності від результату умови, заданої оператором порівняння або складеної умови, реалізованої логічними операторами. Синтаксис оператора має вигляд:

```
If умова Then
    [оператори1]
[Else
    оператори2]
End If
```

Існує можливість використовувати вкладені оператори розгалуження, у такому випадку їх синтаксис має вигляд:

```
If умова Then
    [Оператори]
[ElseIf умова-n Then
    [Оператори-n]
...
[Else
    [ІнакшеОператори]]
End If
```

Необхідно зазначити, що тип `умови`, окрім `Boolean`, може бути числовим або рядком, але таким, що асоціюється зі значеннями `True` або `False` (`Null` трактується як `False`). Наведемо приклад підпрограми, яка використовує оператор *`If Then Else`:*

```
'Порівняння значень двох змінних
Public Sub MinMax(ByVal x As Integer, ByVal y As Integer)
    If x > y Then
        Debug.Print "Змінна x більша змінної y"
    ElseIf x < y Then
        Debug.Print "Змінна x менша змінної y"
    Else
        Debug.Print "Змінні x та y дорівнюють одна одній"
    End If
End Sub
```

У наведеному прикладі процедура MinMax має два параметри, які зазначаються ім'ям параметру та його типом, аналогічно оголошенню змінних. Ключове слово ByVal перед кожним з параметрів означає передачу параметрів за значенням (способи передачі параметрів до процедур будуть розглянуті нижче). Оператор Debug.Print в залежності від значень параметрів виводить у вікно *Immediate* повідомлення, який з них більше/менше або, що вони дорівнюють один одному.

Оператор вибору *Select Case* також є оператором розгалуження та направляє виконання програми за однією з гілок, розташованих у блоках Case, в залежності від результату порівняння результату виразу зі значенням-n блоків Case. Його синтаксис:

```
Select Case вираз
    [Case значення-n
        [Оператори-n]]
    [Case Else
        [ІнакшеОператори]]
End Select
```

Вираз має бути присутнім обов'язково. Значення-n можуть бути представлені однією з форм:

- довільний вираз з числовим значенням результату або значенням-рядком;
- діапазон значень у вигляді нижня-межа To верхня-межа (включаючи обидві);
- діапазон значень у вигляді If умова.

Перша форма задає окремі значення, друга та третя – дозволяють задавати діапазони значень. Необхідно зазначити, що у одному блоці Case можна вказати список значень, які відокремлюються комами, оператори блока будуть виконані, якщо результат виразу співпадає з будь яким значенням списку. Після виконання операторів-n блоку Case перевірка на відповідність іншим значенням-n не проводиться, і керування передається на оператор, наступний за End Select. Необов'язковий блок Else виконується, якщо жодне зі значень блоків Case не співпало з результатом виразу. Наведемо приклад використання оператора *Select Case*:

```
Public Sub RememberYears(ByVal before As Integer, ByVal
currentYear As Integer)
    Dim msg As String

    Select Case currentYear - before
        Case 1970 To 1980, 1981 To 1985, 1990 To 1992, 1996
            msg = "Роки навчання"
        Case 1966 To 1985
            msg = "Роки виховання"
        Case Is >= 1963
            msg = "Інші роки"
        Case Else
```

```
            msg = "Мене ще не було"
    End Select

    Debug.Print msg
End Sub
```

Якщо буде введене значення параметру `before = 54`, а значення параметру `currentYear = 2023`, значенням `виразу` буде `1969`, і буде працювати другий блок `Case` (`"Роки виховання"`). При `before = 42` та `currentYear = 2023` значення `виразу` `1981` можна порівняти з двома діапазонами, але для виконання буде обраний лише перший блок `Case` (`"Роки навчання"`).

Цикли є операторами, які дозволяють виконувати блок коду декілька разів в залежності від значення, яке повертає `умова`, яка явно або неявно зазначається у операторах циклу. Існують цикли різних типів, але у кожному типі циклів можна знайти тіло циклу – послідовність операторів, яка повторюється циклом декілька разів, умову завершення циклу, змінну-ітератор, значення якої повинно змінюватись при кожному повторенні тіла циклу (кажуть при кожній ітерації) – у іншому разі тіло циклу буде повторюватись нескінченно, ця змінна, як і умова, може визначатися неявно. Розглянемо основні типи циклів.

Цикл *`For ... Next`* використовується у разі, якщо відома точна кількість ітерацій. Його синтаксис має вигляд:

```
For змінна-ітератор = початок To кінець [Step крок]
    тіло циклу
Next [змінна-ітератор]
```

Тут `змінна-ітератор` – це числова змінна. На початку виконання циклу вона приймає значення, що задається числовим виразом `початок`, числовий вираз `кінець` задає заключне значення `змінної-ітератора` циклу. Зміну значення неявно виконує частина оператора `Next [змінна-ітератор]` – у разі відсутності частини оператора `[Step крок]`, виконується інкремент значення `змінної-ітератора` (збільшення на `1`), у разі наявності частини оператора `[Step крок]`, виконується збільшення змінної на значення `крок`. У разі, якщо значення `змінної-ітератора` перевищує значення, задане числовим виразом `кінець`, цикл завершує свою роботу. Наведемо приклад використання такого типу циклу для програми, як повинна виводити на екран десять випадкових значень змінної від `1` до `10`:

```
Public Sub RandomNumbersFor()
    Dim num As Integer
    Dim i As Integer    'змінна-ітератор

    For i = 1 To 10
        'Функція Rnd повертає значення >=0 та < 1
        'Функція Int повертає ціле значення відкидуванням
        'дробової частини
        num = Int(10 * Rnd) + 1
        'Крапка з комою в кінці оператора Debug.Print
```

```
        'попереджає перехід на новий рядок
        Debug.Print num & " ";
    Next i
        'Оператор Debug.Print без параметрів здійснює перехід
        'на новий рядок
        Debug.Print

End Sub
```

У цій програмі використовується вбудована функція `Rnd`, виклик якої повертає випадкове значення типу `Single` від `0` (включаючи `0`) до `1` (виключаючи `1`). Зверніть увагу на спосіб виведення згенерованих чисел у один рядок.

Існує можливість примусового завершення циклу *`For ... Next`* до виконання усіх ітерацій – це можна організувати за допомогою оператора `Exit For`. Наведемо приклад модифікації попередньої програми із примусовим виходом у разі випадкової генерації числа `5`:

```
Public Sub RandomNumbersExitFor ()
    Dim num As Integer
    Dim i As Integer    'змінна-ітератор

    For i = 1 To 10
        num = Int(10 * Rnd) + 1
        Debug.Print num & " ";
        If num = 5 Then
            Exit For
        End If
    Next i

        Debug.Print
End Sub
```

Існує модифікація циклу типу *`For ... Next`* для послідовної обробки усіх елементів масивів та колекцій – цикл *`For Each`* – він буде розглянутий нижче у цьому розділі.

Цикл *`Do ... Loop`* використовується у разі, якщо невідома точна кількість ітерацій, але відома `умова` завершення циклу. Існує чотири варіанти циклу такого типу: два варіанти з передумовою – коли умова завершення циклу перевіряється до тіла циклу, та два варіанти з постумовою – коли умова завершення циклу перевіряється після тіла циклу.

Варіанти циклу *`Do ... Loop`* з передумовою мають такий синтаксис:

```
Do {While | Until} умова
    тіло циклу
Loop
```

У разі використання `While` у такому операторі ітерації будуть виконуватися, поки умова буде повертати значення `True`, а у разі використання `Until` у такому операторі ітерації будуть виконуватися, поки умова буде повертати значення `False`.

Наведемо приклади програм, що використовують такий тип циклу:

```
Public Sub RandomNumbersWhile()
    Dim num As Integer
    Dim i As Integer

    i = 0
    Do While i < 10
        num = Int(10 * Rnd) + 1
        Debug.Print num & " ";
        i = i + 1
    Loop

        Debug.Print
End Sub
```

Наведена програма має ту ж функціональність, що і програма `RandomNumbersFor()`, зверніть увагу на явну зміну значення змінної-ітератора.

```
Public Sub RandomNumbersUntil()
    Dim num As Integer
    Dim i As Integer

    i = 1
    Do Until num = 5 Or i > 10
        num = Int(10 * Rnd) + 1
        Debug.Print num & " ";
        i = i + 1
    Loop

    Debug.Print
End Sub
```

Наведена програма має ту ж функціональність, що і програма `RandomNumbersExitFor()`, зверніть увагу на комбіновану `умову` завершення циклу.

В інших двох варіантах `умова` перевіряється в кінці циклу, такі типи циклів використовуються коли також невідома точна кількість ітерацій та відома `умова` завершення циклу, але необхідно, щоб у будь-якому разі перша ітерація була виконана. Варіація програми `RandomNumbersWhile()` з циклом з постумовою, ітерації якого виконуються поки умова повертає `True` виглядає наступним чином:

```
Public Sub RandomNumbersDoWhile()
    Dim num As Integer
    Dim i As Integer

    i = 0
    Do
        num = Int(10 * Rnd) + 1
        Debug.Print num & " ";
        i = i + 1
```

```
    Loop While i < 10
    Debug.Print
End Sub
```

Варіація програми `RandomNumbersUntil()` з циклом з постумовою, ітерації якого виконуються поки умова повертає `False`, виглядає наступним чином:

```
Public Sub RandomNumbersDoUntil()
    Dim num As Integer
    Dim i As Integer

    i = 1
    Do
        num = Int(10 * Rnd) + 1
        Debug.Print num & " ";
        i = i + 1
    Loop Until num = 5 Or i > 10

    Debug.Print
End Sub
```

Аналогічно оператору `Exit For` для передчасного завершення циклів *`Do ... Loop`* можна використовувати `Exit Do`, виконання якого передає керування оператору, наступному за `Loop`.

Необхідно зазначити, що крім розглянутих операторів керування, VBA має додаткові, що залишилися для сумісності з Visual Basic та попередніми версіями VBA, зокрема:

- цикл типу *`While ... Wend`*, який повторює функціональність циклу типу *`Do While ... Loop`*;
- оператор переходу по мітці `GoTo`;
- оператор виклику процедури за умовою `On ... GoSub`
- та деякі інші.

Сьогодні їх використання не рекомендується.

2.6 Організація роботи з масивами

Масив – це найпоширеніший складений тип даних (його також називають структурою даних). Масив являє собою упорядковану сукупність даних одного типу, порядок елементів задається цілочисельними індексами – порядковими номерами елементів, що зберігаються у масиві [1].

Масив може бути оголошеним за допомогою тих же ключових слів, які використовуються при оголошенні змінних та констант, включаючи зазначення типу елементів масиву, але після імені масиву зазначається список його вимірів (кількість елементів у кожному вимірі – масив може бути багатовимірним):

```
[Dim | Public | Private | Static]
ім'я_змінної-масиву ([список_вимірів]) [As ім'я_типу]
```

- Кожен вимір в списку відокремлюється комою і визначається завданням значень індексів *нижньої* (*lower bound*) та *верхньої* (*upper bound*) *границі* або тільки верхньої. Аналогічно оголошенню змінних, у разі, якщо не зазначений тип елементів, їм призначується тип `Variant` і масив може містити елементи різних типів. Наведемо приклад оголошення масиву:

```
Dim intArray(0 To 4) As Integer
```

або – для того ж масиву:

```
Dim intArray(4) As Integer
```

Такий масив може зберігати п'ять цілочисельних елемента, оскільки `4` – це значення індексу верхньої границі масиву, а значення індексу нижньої за замовчуванням дорівнює `0`. Якщо необхідно, щоб значення індексів масиву починалися з `1`, то у *розділі оголошень* (*Declarations*) модуля необхідно зазначити опцію:

```
Option Base 1
```

У такому разі зазначений вище масив буде оголошений:

```
Dim intArray(1 To 5) As Integer
```

або

```
Dim intArray(5) As Integer
```

Присвоїти значення елементу масиву можна зазначивши після ім'я масиву значення індексу елементу у круглих дужках. У разі нумерації індексів за замовчуванням (починаючи з `0`) для першого елементу масиву це виглядає так:

```
intArray(0) = 100
```

Аналогічним чином це значення може бути використане, наприклад:

```
Debug.Print intArray(0)
```

Наведемо приклад програми, що створює масив з `5` цілих чисел зі значеннями від `1` до `10`, що генеруються випадковим чином:

```
Sub ArrayCreation()
    Dim intArray(4) As Integer
    Dim i As Integer

    For i = 0 To 4
        intArray(i) = Int(10 * Rnd) + 1
        Debug.Print intArray(i) & " ";
    Next i
    Debug.Print
End Sub
```

У програмі виділені оператори, що призначають значення елементам масиву та використовують їх, у обох випадках використовується доступ до елементу за індексом.

Масиви можуть бути *багатовимірними*, у такому разі вони можуть оголошуватись, наприклад для двовимірного масиву-матриці 4 рядки на 5 стовпців:

```
Dim biIntArray(0 To 3, 0 To 4)
```

або (у разі нумерації індексів за замовчуванням)

```
Dim biIntArray(3, 4)
```

Наведемо варіант попередньої програми, розширеної для заповнення випадковими числами двовимірного масиву:

```
Sub BiArrayCreation()
    Dim biIntArray(3, 4) As Integer
    Dim i As Integer, j As Integer

    For i = 0 To 3
        For j = 0 To 4
            biIntArray(i, j) = Int(10 * Rnd) + 1
            Debug.Print biIntArray(i, j) & " ";
        Next j
        Debug.Print
    Next i
End Sub
```

Програма виводить матрицю 4 рядки на 5 стовпців, заповнену випадковими цілими числами від 1 до 10.

Розглянуті вище приклади демонструють роботу зі *статичним масивом*. Кількість елементів такого масиву визначена в момент оголошення масиву.

Окрім статичних VBA також підтримує роботу з *динамічними масивами* – масивами, для яких при оголошені розмір не визначається, але він може бути визначений та змінений при виконанні програми. Наведемо приклад оголошення динамічного масиву:

```
Dim dynIntArray() As Integer
```

Для визначення розміру оголошеного динамічного масиву використовується оператор `ReDim(список_вимірів)`, де список вимірів може мати вигляд `(нижня_границя To верхня_границя)` або `(верхня_границя)`. Якщо при перевизначенні розміру масиву задати ключове слово `Preserve`, то у масиві з новим розміром будуть збережені всі раніше розміщені елементи (звісно, у разі, якщо новий розмір більше старого; якщо це не так – елементи зі старшими індексами, що не вміщуються у новий масив, будуть втрачені). У разі використання ключового слова `Preserve` можна змінювати тільки верхню межу масиву (зміна нижньої викликає помилку). Також при використанні ключового слова `Preserve` для багатовимірного масиву існує можливість зміни лише останнього виміру масиву. Наведемо приклад оголошення, визначення та перевизначення розміру динамічного масиву:

```
Sub DynArrayCreation()
    Dim dynIntArray() As Integer
```

```
    Dim i As Integer

    'Визначення розміру динамічного масиву
    ReDim dynIntArray(4)
    For i = 0 To 4
        dynIntArray(i) = Int(10 * Rnd) + 1
        Debug.Print dynIntArray(i) & " ";
    Next i
    Debug.Print

    'Перевизначення розміру динамічного масиву
    'зі збереженням елементів
    ReDim Preserve dynIntArray(0 To 7)
    For i = 0 To 7
        dynIntArray(i) = Int(10 * Rnd) + 1
        Debug.Print dynIntArray(i) & " ";
    Next i
    Debug.Print

    'Перевизначення розміру динамічного масиву
    'зі збереженням елементів
    ReDim Preserve dynIntArray(10)
    For i = 0 To 9
        dynIntArray(i) = Int(10 * Rnd) + 1
        Debug.Print dynIntArray(i) & " ";
    Next i
End Sub
```

Виконання цієї програми виведе у вікно *Immediate*, наприклад, такі дані:

```
6 5 3 7 7
3 3 9 9 6 10 10 3
7 10 3 6 2 10 7 1 6 2
```

Масиви можуть створюватись та заповнюватись одночасно (без використання оператору `ReDim`) за допомогою вбудованої функції `Array()`, що приймає список параметрів. Але, оскільки ця функція повертає масив елементів типу `Variant`, оголошуваний масив повинен бути динамічним та типу `Variant`:

```
Dim intArray()
intArray = Array(1, 3, 5, 7, 9)
```

Оскільки розмір масиву, що отримується функцією `Array()` визначається динамічно, для його обходу циклом *`For ... Next`* необхідно визначити його межі. Це можна зробити з використанням вбудованих функцій `LBound()` – визначає нижню межу масиву, та `UBound()` – визначає верхню межу масиву. Синтаксис цих функцій схожий та має вигляд:

```
LBound(ім'я_масиву [, вимір])
UBound(ім'я_масиву [, вимір])
```

параметр `вимір` – ціле число, яке для багатовимірних масивів визначає вимір, для якого треба отримати межу. Наведемо приклад оголошення

заповнення динамічного масиву за допомогою функції `Array()` та його обходу циклом *`For ... Next`*:

```
Sub ArrayCreationFor()
    'Для використання функції Array масив повинен бути
    'динамічним та мати тип Variant
    Dim intArray()
    Dim i As Integer
    Dim intArrayLength As Integer 'кількість елементів масиву

    intArray = Array(1, 3, 5, 7, 9)
    intArrayLength = UBound(intArray) - LBound(intArray) + 1
    Debug.Print "Розмір масиву: " & intArrayLength & _
                " елементів"

    For i = 0 To intArrayLength - 1
        Debug.Print intArray(i) & " ";
    Next i
        Debug.Print
End Sub
```

У цьому прикладі для динамічно створеного та заповненого функцією `Array()` масиву за допомогою функцій `LBound()` та `UBound()` визначаються його межі, які використовуються у циклі. Зверніть увагу на необхідність корекції довжини масиву та верхньої межі у циклі внаслідок використання нумерації індексів масиву за замовчуванням (`Option Base 0`).

Існує більш зручний варіант циклу *`For ... Next`*, який дозволяє виконувати обхід динамічно створеного масиву з неявним визначенням його розміру (виконується обхід усіх елементів масиву від першого по зростанню індексів). Такий цикл називають циклом типу *`For Each ... Next`*. Він має наступний синтаксис:

```
For Each елемент In ім'я_масиву
    тіло циклу
Next [елемент]
```

Тут `елемент` – це змінна, якій по черзі присвоюються значення всіх елементів масиву. При використанні такого типу циклів змінна `елемент` зобов'язана мати тип `Variant` (або `Object`). Вказувати змінну `елемент` після ключового слова `Next` не обов'язково. Такий тип циклу часто використовується для колекцій (ми ознайомимось з ними, вивчаючи API застосунків Microsoft Office – колекції документів, аркушів робочої книги тощо), у такому разі замість `імені_масиву` використовується `ім'я_колекції`. Наведемо приклад попередньої програми з використанням циклу типу *`For Each ... Next`*:

```
Sub ArrayCreationForEach()
    'Для використання функції Array масив повинен бути
    'динамічним та мати тип Variant
    Dim intArray()
    'Змінна-елемент в циклі For Each повинна мати тип Variant
    'або Object
```

```
    Dim i As Variant

    intArray = Array(1, 3, 5, 7, 9)

    For Each i In intArray
        Debug.Print i & " ";
    Next
    Debug.Print
End Sub
```

Реалізація програми таким чином виглядає значно простіше.

У разі необхідності видалити усі елементи масиву зручно використовувати оператор `Erase`, наприклад:

```
Erase intArray
```

Але цей оператор по різному працює для статичного та динамічного масиву: статичний масив заповнюється елементами зі значеннями за замовчуванням для відповідних типів, а динамічний масив видаляється з пам'яті (залишається тільки його ім'я), і у разі необхідності його повторного використання необхідно буде його перевизначати оператором `ReDim`.

Наведемо приклад програми, що виконує видалення масивів різних типів:

```
Sub ArrayErase()
    Dim intArray(2) As Integer
    Dim varStrArray(1) As String
    Dim dynIntArr()
    Dim i As Integer
    Dim el

    'Статичний масив цілих чисел
    For i = 0 To 2
        intArray(i) = i
        Debug.Print intArray(i) & " ";
    Next i
    Debug.Print
    Erase intArray
    For Each el In intArray
        Debug.Print el & " ";
    Next
    Debug.Print

    'Статичний масив рядків змінної довжини
    varStrArray(0) = "yes"
    varStrArray(1) = "no"
    For Each el In varStrArray
        Debug.Print el & " ";
    Next
    Debug.Print
    Erase varStrArray
    For Each el In varStrArray
        Debug.Print el & " ";
    Next
```

```
    Debug.Print

    'Динамічний масив цілих чисел
    dynIntArr = Array(1, 2, 3)
    Erase dynIntArr
End Sub
```

Запуск програми забезпечує наступне виведення у вікно *Immediate*:

```
0 1 2
0 0 0
yes no
```

Видно, що статичний масив цілих чисел після його видалення містить нульові елементи у тій же кількості, аналогічна ситуація і для масиву рядків змінної довжини – після стирання в масиві залишаються два елементи але з пустими рядками, яких не видно при виведенні. У разі динамічного масиву після його стирання ніяких елементів не залишається взагалі – це можна проконтролювати розмістивши контрольну точку відлагоджувача на перед останньому операторі програми та продивившись масив у вікні *Locals* до та після оператора `Erase dynIntArr` (Рисунок 14).

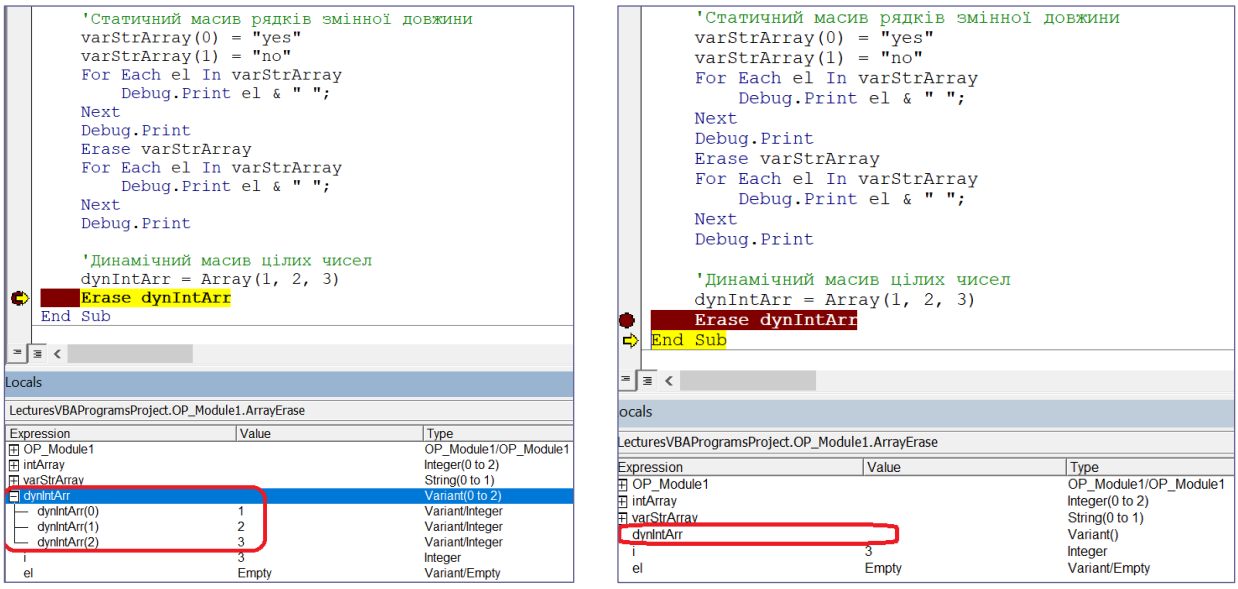


Рисунок 14 – Стирання динамічного масиву

Існують і інші функції для роботи з масивами, але при програмуванні з використанням API застосунків Microsoft Office частіше використовуються колекції, аніж масиви, тому обмежимся основними знаннями про роботу з масивами.

2.7 Процедури VBA: підпрограми та функції, передача аргументів та повернення значень

Процедура – це програмна одиниця VBA, що включає оператори опису її локальних даних і виконувані оператори [3]. Зазвичай процедура містить

послідовність операторів, які вирішують окреме завдання або підзадачу. Особливість процедур VBA полягає у тому, що вони використовувати велику кількість вбудованих методів і функцій, що оперують з різноманітними об'єктами програмної моделі застосунків Microsoft Office.

Процедури VBA можна класифікувати за кількома ознаками (Рисунок 15):

- за типом;
- за способом створення коду процедури;
- за способом запуску процедури на виконання;
- за місцем знаходження процедури в проєкті.

Процедури VBA *за типом* поділяються на *підпрограми, функції* та *властивості*. Перші описуються ключовим словом `Sub`, другі – `Function`, треті – `Property`. У цьому розділі будуть розглянуті підпрограми та функції, процедури `Property` (властивість) будуть викладені у розділі "Засоби VBA для роботи з об’єктами".

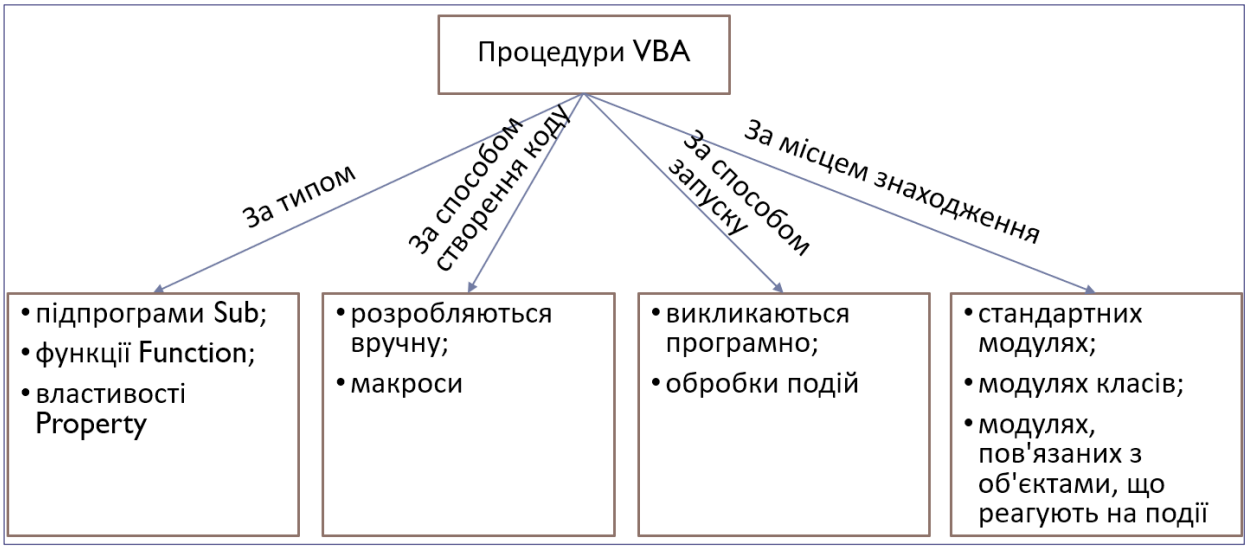


Рисунок 15 – Класифікація процедур VBA

Процедура `Sub` (*підпрограма*) являє собою набір команд, за допомогою яких можна вирішити певне завдання. При запуску процедури виконуються її команди, потім керування передається у додаток, який викликав дану процедуру або у іншу процедуру, з якої викликана дана процедура. У процедурах типу `Sub` зазвичай міститься більшість виконуваних програм.

Процедура `Function` *(функція)* відрізняється від підпрограм тим, що обов’язково повертає одне результуюче значення. Функції зазвичай використовують для отримання значення в результаті обчислень, отримання рядка в результаті операцій над текстом, для визначення логічного значення тощо.

За способом створення коду процедури поділяються на *такі, що розробляються вручну*, і на процедури, код яких створюється автоматично макрорекордером; їх називають *макросами* (способи запису макросів були розглянуті у попередньому розділі).

За способом запуску процедури на виконання їх можна поділити на *такі, що викликаються програмно*, та процедури, що запускаються автоматично при виникненні певної події – це так звані *процедури обробки подій*.

За місцем знаходження в проєкті розрізняють процедури, що знаходяться у *стандартних модулях*, *модулях класів* і *модулях, пов'язаних з об'єктами, що реагують на події*.

Синтаксис *підпрограми* VBA має наступний вигляд:

```
[Private | Public | Static] Sub ім'я ([список-параметрів])
    тіло-підпрограми
End Sub
```

Модифікатор `Public` в заголовку процедури використовується, щоб оголосити процедуру загальнодоступною, тобто дозволити викликати її з усіх інших процедур всіх модулів всіх проектів. Якщо модуль, у якому описана процедура, містить оператор `Option Private Module`, то процедури будуть доступні лише модулям свого проекту. Модифікатор `Private` у оголошенні процедури забороняє доступ до цієї процедури з процедур усіх модулів, крім того, в якому вона описана. За замовчуванням процедура вважається загальнодоступною (`Public`). Модифікатор `Static` забезпечує збереження значень локальних змінних процедури після завершення її роботи (тобто локальні змінні стануть статичними).

Параметр `ім'я` – це ім'я процедури, що задовольняє вимогам VBA до іменування елементів коду. Необов'язковий параметр `список-параметрів` – це послідовність зазначених через кому змінних з їх типами (параметрів), які передають у процедуру при її виклику значення цих змінних (аргументи). Параметри, що задаються при описі процедури, завжди представляють тільки імена змінних, а при виклику процедури її аргументи можуть бути не тільки змінними, але й виразами.

Виклик підпрограм може здійснюватися двома способами:

- перший передбачає зазначення імені підпрограми та списку аргументів, між якими вставляються коми (аргументи передаваються без дужок):

```
MySubroutine 5, "stop", myVar + 1
```

другий використовує оператор `Call` і зазначення списку аргументів у круглих дужках після імені підпрограми:

```
Call MySubroutine(5, "stop", myVar + 1)
```

Наведемо приклад процедури зі списком параметрів та процедури, що її викликає. Якщо подивитись на попередню програму `ArrayErase()`, то можна побачити код виведення масивів циклом *For Each ... Next,* який дублюється. З метою уникнення дублювання коду можна створити підпрограму, якій параметром передається масив (насправді – посилання на масив, це буде розглянуто нижче). Тип параметру обраний `Variant`, що дозволяє передавати масиви з елементами різних типів:

```
Sub PrintArray(arr As Variant)
    Dim el

    For Each el In arr
        Debug.Print el & " ";
    Next
    Debug.Print
End Sub
```

Тепер підпрограму `ArrayErase()` (див. вище) можна переписати так, щоб вона викликала підпрограму `PrintArray()`, передаючи їй у якості аргумента масиви різних типів (зверніть увагу на використання обох способів виклику підпрограм):

```
Sub ArrayErase()
    Dim intArray(2) As Integer
    Dim varStrArray(1) As String
    Dim dynIntArr()
    Dim i As Integer

    'Статичний масив цілих чисел
    For i = 0 To 2
        intArray(i) = i
        Debug.Print intArray(i) & " ";
    Next i
    Debug.Print
    Erase intArray
    Call PrintArray(intArray)

    'Статичний масив рядків змінної довжини
    varStrArray(0) = "yes"
    varStrArray(1) = "no"
    PrintArray varStrArray
    Erase varStrArray
    Call PrintArray(varStrArray)

    'Динамічний масив цілих чисел
    dynIntArr = Array(1, 2, 3)
    Erase dynIntArr
End Sub
```

Програма стала коротшою і дублювання коду видалено.

На відміну від підпрограми, *функція* може використовуватись у операторах, повертаючи у місце її виклику значення, яке буде використовуватись оператором.

Синтаксис функції VBA має наступний вигляд:

```
[Private | Public | Static] Function ім'я ([список-
параметрів]) As тип-значення-що-повертається
    тіло-підпрограми
End Sub
```

Якщо порівняти синтаксис функції з синтаксисом підпрограми (див. вище), можна помітити, що оголошення функції містить тип результуючого значення, яке повертається функцією. Таким чином, підпрограму VBA можна асоціювати з `void`-функцією C++, а функцію VBA – з іншими типам функцій C++.

Розглянемо, як приклад, визначення функції `TriangleSquare()`, яка повертає значення площі трикутника за декартовими координатами трьох вершин:

```
Function TriangleSquare(x1 As Integer, y1 As Integer, _
                        x2 As Integer, y2 As Integer, _
                        x3 As Integer, y3 As Integer) _
                        As Double

    TriangleSquare = Abs((x1 * (y3 - y2) + x2 * (y1 - y3) _
                    + x3 * (y2 - y1)) / 2)
End Function
```

Зверніть увагу, що значення виразу, який розраховує площу трикутника привласнюється локальній змінній `TriangleSquare`, яка явно не оголошена і ім'я якої співпадає з назвою функції. Такі змінні створюються у функціях VBA автоматично і використовуються для повернення результату з функції. Використання функції у підпрограмі виглядає, наприклад, так:

```
Sub Main()
    Dim s As Double
    s = TriangleSquare(1, 5, 6, -4, -2, 1)
    Debug.Print "Площа трикутника дорівнює " & s
End Sub
```

Аналогічно передчасному виходу з циклів, існує можливість передчасно вийти (зазвичай, по виконанню якоїсь умови) з підпрограми або функції за допомогою операторів `Exit Sub` або `Exit Function`.

Існують ситуації, коли функція викликається, але її результат не використовується. Звучить це дивно, але існують вбудовані функції VBA, які можна застосовувати або використовуючи результат, що вона повертає, або ні. Якщо результат функції використовуватись не буде, вона викликається за правилами виклику підпрограми: використовуючи оператор `Call`, після якого зазначається ім'я функції та її аргументи у дужках, або без додаткових ключових слів, але аргументи функції повинні бути зазначені без дужок. Наприклад, дуже часто використовувана функція, що являє собою графічне вікно для виведення повідомлень:

```
MsgBox(prompt, [buttons,] [title,] [helpfile, context ]) _
        As VbMsgBoxResult
```

повертає як результат елемент перечислення `VbMsgBoxResult`, що вказує яка саме кнопка була натиснута у цьому вікні. Вочевидь бувають випадки, коли нам необхідно знати про це, та випадки, коли результат нас не цікавить, а після кліку на кнопці вікно автоматично закривається. Наступна програма демонструє такі випадки:

```
Sub MsgBoxReturnValue()
    Dim returnValue As VbMsgBoxResult
    Dim msg As String

    returnValue = MsgBox("Виконайте клік по одній з кнопок", _
                         vbOKCancel, "Вікно MsgBox")
    If returnValue = VbMsgBoxResult.vbOK Then
        msg = "Ви клікнули по кнопці OK"
    ElseIf returnValue = VbMsgBoxResult.vbCancel Then
        msg = "Ви клікнули по кнопці Скасувати"
    End If
    MsgBox msg, vbOKOnly, "Вікно MsgBox"
    Call MsgBox(msg & " - повторне повідомлення", vbOKOnly, _
                "Вікно MsgBox")
End Sub
```

Детально аргументи функції `MsgBox()` будуть розглянуті нижче у цьому розділі. Для багатьох вбудованих функцій компілятор VBA не дає можливості ігнорувати значення, що повертається, змушуючи укладати аргументи у круглі дужки і приймати значення, що повертається.

Передача аргументів до процедур має певні тонкощі, які ми зараз і розглянемо. Якщо подивитись на аргументи, що передавались до функції `MsgBox()`, то можна побачити серед них і константи (рядки-літерали та елемент перечислення, який фактично є цілим числом), і змінні. У разі передачі до функції у якості аргумента змінної та подальшої зміни значення цієї змінної, виникає питання, чи зміниться значення цієї змінної зовні функції, тобто у підпрограмі, яка цю функцію викликала. Давайте дослідимо це на прикладі. Нехай є функція яка повертає добуток своїх аргументів і у ній виконується зміна значення одного з них:

```
Function Multiply(op1 As Integer, op2 As Integer)
    Multiply = op1 * op2
    'Зміна значення аргумента
    op1 = op1 + 1
End Function
```

Викличемо таку функцію у підпрограмі та перевіримо значення змінної, що передається як аргумент до та після виклику:

```
Sub ArgumentsPass()
    Dim myVar As Integer

    myVar = 3
    Debug.Print "До виконання функції myVar="; myVar
    Debug.Print "Результат функції="; Multiply(myVar, 2)
    Debug.Print "Після виконання функції myVar="; myVar
End Sub
```

Інформація у вікні *Immediate* буде наступною:

```
До виконання функції myVar= 3
Результат функції= 6
Після виконання функції myVar= 4
```

Бачимо, що значення змінної, яка була передана як аргумент, змінилося – у програмуванні таку ситуація називають *побічним ефектом* (*side effect*) – і вона, зазвичай, вважається небажаною, оскільки ускладнює відстеження значень змінних у програмах і призводить до помилок. Якщо аргумент у функції змінює своє значення, краще виконати його копіювання у локальну змінну, оголошену в функції, і використовувати в функції копію:

```
Function Multiply(op1 As Integer, op2 As Integer)
    Dim op1Copy As Integer

    op1Copy = op1
    Multiply = op1Copy * op2
    'Зміна значення аргумента
    op1Copy = op1Copy + 1
End Function
```

Вочевидь, при використанні такої функції, зовнішня змінна залишить своє значення.

Існує простіша реалізація коду функції з усуненням побічного ефекту – оголошення передачі аргумента до функції *по значенню* (*By Value*) – це досягається розміщенням ключового слова `ByVal` перед аргументом, при передачі якого до функції необхідно створити локальну копію. У такому разі код тіла функції не відрізняється від першого варіанту, але насправді змінна `op1`, що використовується в функції є копією однойменної зовнішньої змінної:

```
Function Multiply(ByVal op1 As Integer, op2 As Integer)
    Multiply = op1 * op2
    'Зміна значення аргумента
    op1 = op1 + 1
End Function
```

Повторне виконання підпрограми `ArgumentsPass()` виводить у вікно *Immediate* інформацію:

```
До виконання функції myVar= 3
Результат функції= 6
Після виконання функції myVar= 3
```

яка підтверджує, що зовнішня змінна після відпрацювання функції не змінила своє значення. Передача аргументів за посиланням доцільна, наприклад, коди передається значення складеного типу, наприклад, масив, у такому разі програма буде працювати ефективніше [17]:

```
Sub PopulateArray(ByRef arr() As Integer, _
                  ByVal length As Integer, _
                  ByVal lValue As Integer, _
                  ByVal uValue As Integer)
    Dim i As Integer

    ReDim arr(length - 1)
    If uValue >= lValue And length > 0 Then
        For i = 0 To length - 1
            arr(i) = Int(Rnd * uValue + lValue)
```

```
        Next i
    End If
End Sub
```

Ця підпрограма приймає як аргумент посилання на масив цілих чисел, бажану кількість елементів масиву, мінімальне та максимальне значення діапазону випадкових цілих чисел, з якого будуть надані значення елементам масиву. Підпрограма приймає посилання на масив, який повинен бути оголошений (або створений ззовні підпрограми). Підпрограма, яка викликає підпрограму `PopulateArray()` може бути такою:

```
Sub ArgumentArrayPass()
    Dim myArray() As Integer
    Dim el

    Call PopulateArray(myArray, 5, 1, 10)

    For Each el In myArray
        Debug.Print el & " ";
    Next
    Debug.Print
End Sub
```

У цій підпрограмі оголошений динамічний масив передається до підпрограми `PopulateArray()` для створення масиву цілих чисел з `5` елементів та заповнення їх значень випадковими числами від `1` до `10`.

Ще одна особливість передачі аргументів – обов'язковість їх зазначення. Наприклад, функція

```
Function ThreeArgsMultiply(op1 As Integer, op2 As Integer, _
                           op3 As Integer) As Integer
    ThreeArgsMultiply = op1 * op2 * op3
End Function
```

має два обов'язкових параметра, а спроба її викликати функцію без передачі їй усіх аргументів, наприклад:

```
Debug.Print "Результат функції="; ThreeArgsMultiply(5)
```

призведе до помилки "Argument not optional" ("Аргумент є обов'язковим"). У разі, якщо функція при відсутності аргумента використовує якесь його значення за замовчуванням, параметр оголошують необов'язковим, додаючи ключове слово `Optional`. Для перевірки того, чи був переданий необов'язковий параметр, у разі, якщо параметр має тип `Variant`, використовується функція `IsMissing(ім'я-параметра)`, для параметрів інших типів виконується порівняння їх значень зі значеннями за замовчуванням. Перепишемо вище приведену функцію у новій редакції

```
Function ThreeArgsMultiply(op1 As Integer, _
                           Optional op2 As Integer, _
                           Optional op3 As Variant) As Double
    If op2 = 0 Then
        op2 = 1
```

```
    End If
    If IsMissing(op3) Then
        op3 = 1
    End If
    ThreeArgsMultiply = op1 * op2 * op3
End Function
```

Тепер виклик функції з одним параметром відпрацює без помилок. Зазначимо, що функція може мати всі параметри необов'язковими, але якщо якийсь з параметрів оголошений як `Optional`, всі наступні параметри у списку повинні бути `Optional`.

Розглянемо ще випадок, коли ми хочемо передати третій параметр, залишивши другий параметром за замовчуванням. У такому разі виклик функції буде виглядати трохи дивним – кома зазначається, а аргумент перед нею – ні:

```
Debug.Print "Результат функції="; ThreeArgsMultiply(5, , 2)
```

Більш зрозумілою та корисною формою виклику функції у разі наявності багатьох параметрів, серед яких є необов'язкові, є форма передачі аргументів, вказуючи імена параметрів:

```
Debug.Print "Результат функції="; _
            ThreeArgsMultiply(op1:= 5, op3:= 2)
```

У такому разі можна зазначати тільки ті аргументи, що фактично передаються. Зазначимо, що у довідці з VBA необов'язкові параметри процедур вказуються у квадратних дужках.

2.8 Вбудовані функції VBA

Мова VBA підтримує велику кількість вбудованих функцій, наприклад, `Rnd()`, `MsgBox()`, які були використані в попередніх програмах [1]. Виконаємо короткий огляд найбільш корисних функцій, розбивши їх за типом даних, з якими вони працюють.

Першими розглянемо *функції обробки числових даних*, кожна з них повертає числове значення:

- `Abs(число)` – модуль (абсолютне значення) `числа`;
- `Sqr(число)` – квадратний корінь `числа`;
- `Exp(число)` – експоненту (результат піднесення `числа` до ступеня числа e);
- `Log(число)` – натуральний логарифм `числа`;
- `Sin(число)` – синус кута, заданого `числом` у радіанах;
- `Cos(число)` – косинус кута, заданого `числом` у радіанах;
- `Tan(число)` – тангенс кута, заданого `числом` у радіанах;
- `Atn(число)` – арктангенс (в радіанах) `числа`, що задає тангенс кута;
- `Rnd(число)` – рівномірно розподіленого випадкового значення типу `Single` в інтервалі від `0`, включаючи `0`, до `1`, не включаючи `1`. Якщо аргумент не заданий або більше `0`, то породжується чергове випадкове число. Якщо

аргумент дорівнює 0, то результатом буде попереднє випадкове число, а якщо число менше нуля, то кожен раз породжується одне і те ж випадкове число;

- Sng(число) – знак числа (якщо число більше нуля, повертається 1, якщо дорівнює нулю, повертається 0, якщо менше нуля, повертається -1);

- Int(), Fix() і Round() – дозволяють по різному округляти числа: Int() повертає найближче менше ціле, Fix() відкидає дробову частину, Round() округлює до зазначеної кількості знаків після коми. При цьому Round() округлює значення 5 за так званим "банківським алгоритмом": якщо попередній до 5 розряд парний, 5 округлює попередній розряд до меншого значення, а якщо попередній розряд непарний - то 5 округлює його до більшого значення:

```
Debug.Print Round(3.425, 2)
Debug.Print Round(3.435, 2)
Debug.Print Round(3.445, 2)
```

Роздруковані у вікні *Immediate* значення для цього прикладу:

```
3,42
3,44
3,44
```

На практиці для виведення округленого числа за звичайними правилами округлення використовують функцію Format():

```
Debug.Print Format(3.425, "#,##0.00")
```

наведений приклад поверне 3,43 (детальніше функція Format() буде описана нижче).

Наступна група функцій – *функції конвертації та перевірки типів даних*, більшість таких функції мають ім'я, що починається великою літерою C (від слова *Convert*) та продовжуються назвою типу (або скороченою назвою типу), у який конвертується аргумент функції: CBool(), CByte(), CCur(), CDate(), CDbl(), CDec(), CInt(), CLng(), CSng(), CStr(), CVar(), CVErr(). Таким чином, призначення більшості функцій зрозуміло з їх імені, остання функція використовується для створення визначених користувачем об'єктів-помилок і приймає номер помилки як аргумент. Типи помилок та роботу з ними ми розглянемо у наступному розділі. Існує вбудована функція TypeName(varname), що повертає як рядок назву типу даних аргументу, таку функцію зручно використовувати для визначення фактичного типу змінних, наприклад:

```
Sub ConvertType()
    Dim myVar, myCVar, nullVar

    myVar = 5.12
    Debug.Print myVar & " is "; TypeName(myVar)
    myCVar = CInt(myVar)
    Debug.Print myCVar & " is "; TypeName(myCVar)
    myCVar = CStr(myVar)
```

```
    Debug.Print myCVar & " is "; TypeName(myCVar)
    myCVar = CCur(myVar)
    Debug.Print myCVar & " is "; TypeName(myCVar)
    myCVar = CDate(myVar)
    Debug.Print myCVar & " is "; TypeName(myCVar)
    myCVar = CBool(myVar)
    Debug.Print myCVar & " is "; TypeName(myCVar)
    nullVar = Null
    Debug.Print TypeName(nullVar)
    'Type mismatch
'    myCVar = CDate(CVErr(1009))
End Sub
```

Запуск цієї підпрограми виведе у вікно *Immediate*:

```
5,12 is Double
5 is Integer
5,12 is String
5,12 is Currency
04.01.1900 2:52:48 is Date
True is Boolean
-5,12 is Currency
Null
```

Функція конвертації у останньому операторі (закоментованому) при його розкоментуванні намагається виконати конвертацію об'єкта помилки до дати, таке перетворення неможливе і виникає помилка Type mismatch (невідповідність типу). Окрім вище наведених існують функція конвертації до рядка `Str()` – робить те ж саме, що функція `CStr()`, але для позитивних чисел вставляє пробіл попереду, та функція `Val()` – повертає з рядка, що містить цифри і літери тільки числове значення, функція читає дані зліва направо і зупиняється на першому нечисловому значенні, допускається нечислове значення – точка, яка буде відокремлювати цілу та дробову частину у результаті.

Для уникнення помилок при застосуванні функцій конвертації рекомендується спочатку впевнитись у відповідному значенні аргументів таких функцій. Це можна зробити вбудованими функціями `IsNumeric()`, `IsDate()`, `IsArray()`, `IsEmpty()`, `IsError()`, `IsMissing()`, `IsNull()` та `IsObject()`. Призначення цих функцій зрозуміле з їх імені і вони повертають `True` або `False` залежно від результату перевірки аргументу.

Також існують функції конвертації чисел у десятковій системі до відповідних значень у шістнадцятковій `Hex()` та вісімковій `Oct()` системах.

Досить популярними у використані є *функції обробки рядків*. Але перед їх вивченням розглянемо використання оператора `Like`, який дозволяє виконувати пошук рядка, який відповідає шаблону, у рядку, для якого викликається цей оператор:

```
рядок Like шаблон
```

У разі, якщо така відповідність знайдена, оператор повертає `True`, якщо збігу немає, повертається `False`. Якщо або `рядок`, або `шаблон` є `Null`, оператор

повертає Null. При цьому шаблон – це рядок зі спеціальними символами, які задають логіку порівняння (Таблиця 3).

Таблиця 3. Символи шаблону оператора Like

Символ підстановки	Значення
#	Будь-яка цифра (тільки одна) від 0 до 9
*	Будь-яка кількість будь-яких символів (включаючи нульовий)
?	Будь-який символ (тільки один)
[a, b, c]	Будь-який символ (тільки один) з наведеного списку
[! a, b, c]	Будь-який символ (тільки один), окрім наведених у списку

Наведемо приклад програми, яка перевіряє відповідність рядків шаблону та, у разі їх відповідності, виводить рядки на екран:

```
Sub LikeStudy()
    Dim s As String

    s = "Park ave, 22"
    If (s Like "*##*") Then Debug.Print s
    s = "123-345-6789"
    If (s Like "###[-]###[-]####") Then Debug.Print s
    s = "123 345 6789"
    If (s Like "###[- ]###[- ]####") Then Debug.Print s
    s = "Animal"
    If s Like "[A-B]*" Then Debug.Print s
    s = "test.docx"
    If s Like "*.doc?" Then Debug.Print s
End Sub
```

Проаналізуйте шаблони у цій програмі, орієнтуючись на Таблиця 3 і приймаючи до відому, що усі рядки будуть виведені на екран.

Існують нюанси порівняння рядків: при порівнянні рядків за замовчуванням порівнюються двійкові коди символів (працює опція Option Compare Binary) – при цьому звісно регістр символу впливає на результат порівняння. У разі, якщо при порівнянні враховувати регістр не потрібно, у області декларацій модуля зазначається опція Option Compare Text.

До основних функцій обробки рядків можна віднести:

- Len(string) – повертає довжину (число символів) рядка-аргумента;
- Left(string, length) – повертає з рядка string підрядок з вказаним числом length символів з початку рядка;
- Right(string, length) – повертає з рядка string підрядок з вказаним числом length символів з кінця рядка;
- Mid(string, start [, length]) – повертає з рядка string підрядок довжиною length, починаючи зі start номера позиції символу (номери позиції символу починаються з 1).
- ASC(string) – повертає числовий код для першого символу рядка string – нагадаємо, що символьний тип у VBA відсутній (Таблиця 1);

- `Chr(char_code)` – повертає рядок з одного символу зі значенням коду `char_code`;

Наведемо приклад програми, що реалізує шифр Цезаря, який є простим шифром зсуву символів шляхом додання/віднімання ключа шифру до їх коду [18]. Функція шифрування символів у діапазоні від `0` (код 48) до `z` (код 122) використовує вище наведені функції та може бути реалізована таким чином:

```
Function Encrypt(str As String, key As Integer) As String
    Dim ch As String, encCh As String
    Dim i As Integer, chCode As Integer

    For i = 1 To Len(str)
        ch = Mid(str, i, 1)
        chCode = Asc(ch)
        encCh = Chr((chCode - 48 + key) Mod 75 + 48)
        Encrypt = Encrypt & encCh
    Next i
End Function
```

Завдяки симетричній природі операції ділення по модулю (Mod), функція дешифрування може використовувати функцію шифрування:

```
Function Decrypt(encStr As String, key As Integer) As String
    Decrypt = Encrypt(encStr, 75 - (key Mod 75))
End Function
```

Наступна підпрограма використовує функції шифрування та дешифрування рядка, який містить граничні символи можливого для шифрування-дешифрування діапазону з ключем, що визначає зсув на `3` символи:

```
Sub CaesarCipher()
    Dim str As String, encStr As String, decStr As String
    Dim key As Integer

    str = "0Hellz"
    key = 3
    encStr = Encrypt(str, key)
    Debug.Print "Шифрований рядок: " & encStr
    decStr = Decrypt(encStr, key)
    Debug.Print "Дешифрований рядок: " & decStr
End Sub
```

Запуск програми виводить у вікно *Immediate*:

```
Шифрований рядок: 3Khoo2
Дешифрований рядок: 0Hellz
```

Існують функції, що виконують пошук та заміну підрядків у рядку:

- `InStr` – визначає позицію першого входження підрядка `substring` в рядку `string` при пошуку з початку рядка `string`. У разі якщо підрядок `substring` у рядку `string` не знайдено, повертається `0`, синтаксис:

```
InStr([start,] string, substring [, compare])
```

Необов'язковий параметр start задає позицію, з якої починається пошук (за замовчуванням – з першого символу рядка string). Необов'язковий параметр compare визначає особливості порівняння рядків, при використанні значення compare, що дорівнює 0 (значення за замовчуванням), посимвольне порівняння виконується з урахуванням регістру, значення compare, що дорівнює 1, задає режим посимвольного порівняння без урахування регістру.

- InStrRev – визначає позицію першого входження підрядка substring в рядку string при пошуку з кінця рядка string. У разі якщо підрядок substring у рядку string не знайдено, повертається 0, синтаксис:

```
InStrRev (string, substring [, start] [, compare])
```

Необов'язковий параметр start задає позицію, з якої починається пошук (за замовчуванням – з останнього символу рядка string), інші параметри такі ж, як в InStr.

- Replace повертає рядок, у якому послідовність символів find вихідного рядку string замінена іншою послідовністю символів replace, синтаксис:

```
Replace (string, find, replace [, start] [,count] [, compare])
```

Необов'язковий параметр start задає позицію, з якої починається пошук (за замовчуванням - з першого символу рядка string). Необов'язковий параметр count задає кількість замін, якщо він не вказується замінюються всі входження find. Необов’язковий параметр compare працює так, як у функції InStr. Функція повертає рядок з заміненими символами, або пустий рядок, якщо підрядок find не було знайдено. Якщо підрядок replace пустий, то з вихідного рядка string будуть видалені усі входження find. Наведемо приклад програми, яка використовує функції пошуку та заміни підрядків у текстовому рядку:

```
Sub InStrReplaceStudy()
    Dim s1, s2 As String

    s1 = "Hello"
    s2 = "he"
    Debug.Print InStr(s1, s2)       '0
    s2 = "He"
    Debug.Print InStr(s1, s2)       '1
    s2 = "he"
    Debug.Print InStr(1, s1, s2, vbTextCompare) '1
    s2 = "l"
    Debug.Print InStr(s1, s2)       '3
    Debug.Print InStrRev(s1, s2)    '4
    s2 = "ll"
    Debug.Print Replace(s1, s2, "zzz")  'Hezzzo
End Sub
```

Кінцеві коментарі у операторах Debug.Print зазначають дані, що виводяться на екран.

- `StrComp(string1, string2 [, compare])` – дозволяє порівнювати рядки `string1` та `string2`, повертаючи `-1`, якщо `string1 < string2`, `1`, якщо `string1 > string2`, та `0`, якщо вони однакові. Необов'язковий параметр `compare` працює так, як у функції `InStr()`.

- `LCase(string)` – повертає копію рядка-параметра, усі символи якого перетворені до нижнього регістру;

- `UCase(string)` – повертає копію рядка-параметра, усі символи якого перетворені до верхнього регістру;

- `Ltrim(string)` – повертає копію рядка-параметра, з якого видалені пробіли, що знаходяться на початку рядка;

- `Rtrim(string)` – повертає копію рядка-параметра, з якого видалені пробіли, що знаходяться у кінці рядка;

- `Trim(string)` – повертає копію рядка-параметра, з якого видалені пробіли, що знаходяться на початку і в кінці рядка;

- `Space(number)` – повертає рядок із зазначеною параметром `number` кількістю пробілів;

- `String(number, character)` – повертає рядок із зазначеною кількістю `number` символів `character`;

- `StrReverse(string)` – повертає рядок з символами рядка-параметра у зворотному порядку.

Необхідно зазначити, що у разі, якщо у рядку-літералі наявні подвійні лапки, їх необхідно екранувати, в VBA для цього можна використати ще одні подвійні лапки, наприклад, для рядка-літерала `"Спеціальність "Інженерія програмного забезпечення""` це виглядає так:

```
Dim spec As String

spec = "Сеціальність ""Інженерія програмного забезпечення""
Debug.Print spec
```

Наступна група функцій – *функції для роботи з датами та часом*. Нагадаємо, що VBA має тип даних `Date`, який зберігає дату у числі типу `Double` як цілу частину з кількістю днів від початкової дати – `30.12.1899`, а час – як дробову частину зі значенням долі часу від доби. Літерали дат задаються з використанням обмежуючих символів "#". Після введення дати з якимись роздільниками, наприклад, `#24, Apr, 23#` (можна і без ком), вона автоматично перетворюється до стандартної форми `#4/24/23#`. Існує можливість задавати дати функцією `CDate("4/24/2023")`. Якщо для змінної типу `Date` задати дату без обмежуючих символів `4/24/2023`, цей введення буде сприйняте як арифметичний вираз і результат буде інтерпретуватись як час (оскільки ціла частина результату дорівнює `0`). Літерали часу задаються з використанням обмежуючих символів "#" і використовують, зазвичай, двокрапку як роздільник, наприклад, `#10:25#` буде автоматично переведено до `#10:25:00 AM#`, а `#22:25:30#` – до `#10:25:30 PM#`. Зазначимо, що можна вводити одночасно і

дату, і час, наприклад, #4/24/23 10:25# буде автоматично переведено до #4/24/2023 10:25:00 AM#.

Найчастіше використовуються такі вбудовані функції для роботи з датами та часом:

- Now - повертає поточні дату та час з системного годинника комп'ютера як значення типу Variant (Date);
- Date – повертає поточну дату з системного годинника комп'ютера як значення типу Variant(Date);
- Time – повертає поточний час з системного годинника комп'ютера як значення типу Variant(Date);

Хоча Now, Date, Time не містять після імені круглих дужок, вони все ж вважаються функціями.

- DateValue(date) – повертає значення типу Variant(Date),), що містить дату та час, задані рядком date, наприклад, DateValue("24.04.2023 10:25:30"), DateValue("04.24.2023 10:25:30"), DateValue("24 квітень 2023") тощо;
- TimeValue (time) – повертає значення типу Variant(Date), що містить час, заданий рядком time від 0:00:00 (12:00:00 AM) до 23:59:59 (11:59:59 PM).
- DateSerial(year, month, day) – обчислює значення типу Variant(Date) за цілочисельними значеннями аргументів year – рік у діапазоні 100-9999, month – місяць у діапазоні 1-12, day – день у діапазоні 1-31. Можна використовувати для аргументів числові вирази, наприклад, DateSerial(2023 - 32, 4 + 4, 24) поверне дату 24.08.1991.
- TimeSerial (hour, minute, second) – обчислює значення типу Variant(Date) за цілочисельними значеннями аргументів – години, хвилини і секунди, наприклад, TimeSerial(13, 35, 31) поверне час 13:35:31.
- Day(date) – повертає цілочисельне значення порядкового номеру дня місяця для аргумента-дати, який може мати тип Date або String;
- Month(date) – повертає цілочисельне значення порядкового номеру місяця року для аргумента-дати який може мати тип Date або String;
- Year(date) – повертає цілочисельне значення року для аргумента-дати який може мати тип Date або String;
- Hour(time) – повертає цілочисельне значення години для аргумента-часу, який може мати тип Date або String;
- Minute(time) – повертає цілочисельне значення хвилини для аргумента-часу, який може мати тип Date або String;
- Second (time) – повертає цілочисельне значення секунди для аргумента-часу, який може мати тип Date або String;
- MonthName (month [, abbreviate]) – повертає назву місяця за його цілочисельним номером month, необов'язковий булев параметр abbreviate

при виставленні у True повертає скорочену до 3-х літер назву місяця. Значення, що повертається, залежить від регіональних налаштувань.

DateAdd(interval, number, date) – повертає значення дати/часу з доданою/віднятою цілочисельною кількістю number зазначених як рядок інтервалів дат та/або часу interval зі значення аргументу-дати date. Допустимі значення аргументу interval наведені у Таблиця 4. Наприклад, DateAdd("ww", 2, Now) поверне дату на 2 тижня більше поточної.

- DateDiff(interval, date1, date2 [, firstdayofweek [, firstweekofyear]]) – повертає значення Variant(Long) між двома датами date1 (початок) та date2 (кінець) у одиницях інтервалу дат/часу interval (Таблиця 4). Необов'язковий параметр firstdayofweek може задаватися константою, яка вказує перший день тижня (за замовчуванням вважається, що тиждень починається з неділі), необов'язковий параметр firstweekofyear може задаватися константою, яка вказує перший тиждень року (за замовчуванням першим тижнем вважається тиждень, що містить перше січня). Наприклад, для dat1 = #2/13/2023 8:45:00 AM# та dat2 = #6/5/2023 4:35:00 PM# виклик функції DateDiff("ww", dat1, dat2) поверне 16.

Таблиця 4 – Інтервали дат та часу, що використовуються у функціях дати та часу

Значення	Опис
"yyyy"	рік
"q"	квартал
"m"	місяць
"y"	день року
"d"	день місяця
"w"	день тижня
"ww"	тиждень
"h"	години
"n"	хвилини
"s"	секунди

- DatePart(interval, date [, firstdayofweek, [, firstweekofyear]]) – повертає значення Variant(Integer) з зазначеною параметром interval (Таблиця 4) частиною дати date (необов'язкові параметри такі ж, як у функції DateDiff()). Наприклад, DatePart("y", Now) повертає порядковий номер поточного дня у поточному році.

- Timer – повертає кількість секунд, що пройшли з півночі.

Наступна група функцій – *функції форматування даних*. Для форматування даних використовується функція Format і цілий набір функцій, які починаються з префіксу Format: FormatNumber, FormatCurrency, FormatDateTime тощо.

Функція Format повертає значення Variant(String) для вихідного значення expression, відформатоване у відповідності до інструкцій, зазначених параметром-рядком format:

```
Format(expression [, format[, firstdayofweek
[, firstweekofyear]]])
```

Необов'язкові параметри: format – рядок назва/шаблон іменованого (Таблиця 5)/користувацького (Таблиця 6) форматів, інші параметри – такі ж, як у функції DateDiff(). У разі, якщо необов'язкові параметри не зазначені, повертається рядкове значення expression.

Таблиця 5 – Іменовані формати для використання з функцією Format()

Іменований формат	**Дія**
"General Date"	Форматує інформацію про дату і час в послідовне число дати, використовуючи установки формату дати і часу для даного комп'ютера.
"Long Date"	Форматує в послідовну дату тільки частину, яка містить дату, використовуючи установки комп'ютера для Long-формату дати.
"Medium Date"	Форматує в послідовну дату тільки частину, яка містить дату, використовуючи установки комп'ютера для Medium-формату дати.
"Short Date"	Форматує в послідовну дату тільки частину, яка містить дату, використовуючи установки комп'ютера для Short-формату дати.
"Long Time"	Форматує в послідовну дату тільки частину, яка містить час, використовуючи установки комп'ютера для Long-формату часу.
"Medium Time"	Форматує в послідовну дату тільки частину, яка містить час, використовуючи установки комп'ютера для Medium-формату часу.
"Short Time"	Форматує в послідовну дату тільки частину, яка містить час, використовуючи установки комп'ютера для Short-формату часу.
"General Number"	Форматує число в рядок без будь-яких особливих символів.
"Currency"	Форматує число з символом грошової одиниці, роздільником тисяч і тільки двома десятковими розрядами.
"Fixed"	Форматує число так, щоб завжди була, як найменш, одна цифра перед десятковим роздільником і завжди дві цифри після нього.
"Standard"	Форматує число з роздільником тисяч так, щоб була, як найменш, одна цифра перед десятковим роздільником і завжди дві цифри після нього.
"Percent"	Форматує число як процентне відношення, помножуючи його на 100 і додаючи символ відсотка.
"Scientific"	Форматує число в звичайний експонентний формат.
"Yes/No"	Повертається рядок "Yes", якщо число, що форматується, не дорівнює нулю, інакше – "No".
"True/False"	Повертається рядок "True", якщо число, що форматується, не дорівнює нулю, інакше – "False".
"On/Off"	Повертається рядок "On", якщо число, що форматується, не дорівнює нулю, інакше - "Off".

У наступному прикладі тестуються команди форматування з іменованими форматами:

```
Sub FormatOptions()
    Dim today As Date
    Dim num As Double
    Dim zer As Integer

    today = Now
    num = 1234.567
    zer = 0

    Debug.Print Format(today, "General Date")
    Debug.Print Format(today, "Long Date")
    Debug.Print Format(today, "Medium Date")
    Debug.Print Format(today, "Short Date")
    Debug.Print Format(today, "Long Time")
    Debug.Print Format(today, "Medium Time")
    Debug.Print Format(today, "Short Time")
    Debug.Print
    Debug.Print Format(num, "General Number")
    Debug.Print Format(num, "Currency")
    Debug.Print Format(num, "### ###.00 грн.")
    Debug.Print Format(num, "Fixed")
    Debug.Print Format(num, "Standard")
    Debug.Print Format(num, "Percent")
    Debug.Print Format(num, "Scientific")
    Debug.Print Format(num, "Yes/No")
    Debug.Print Format(zer, "Yes/No")
    Debug.Print Format(num, "True/False")
    Debug.Print Format(zer, "True/False")
    Debug.Print Format(num, "On/Off")
    Debug.Print Format(zer, "On/Off")
End Sub
```

Після запуску програма виводить, наприклад:

```
24.04.2023 16:17:06
24 квітень 2023 р.
24-Кві-23
24.04.2023
16:17:06
04:17
16:17

1234,567
1 234,57 ?
1 234,57 грн.
1234,57
1 234,57
123456,70%
1,23E+03
Yes
No
True
False
On
Off
```

Зверніть увагу, що іменований формат не виводить назву української валюти, це можна виправити, застосувавши користувацький формат, приведений у програмі. Користувацькі формати будуються з використанням символів-заповнювачів, наведених разом з прикладами їх застосування у Таблиця 6.

Таблиця 6 – Користувацькі формати для використання з функцією `Format()`

Символ-заповнювач	**Дія (як приклад використовується число `1234.45`)**
`0`	Цифровий символ, відображає цифру, якщо вона знаходиться у цій позиції, або `0`, якщо - ні. Можна використовувати символ `0` для відображення початкових нулів для цілих чисел і кінцевих нулів у десяткових дробах: `"00000.000"` відображає `01234.450`
`#`	Цифровий символ, відображає цифру, якщо вона знаходиться у цій позиції, інакше – не відображує нічого. Символ-заповнювач `#` еквівалентний `0`, крім того, що початкові і кінцеві нулі не відображаються: `"#####.###"` відображає `1234.45`
`$`	Відображає знак долара та пробіл у позиціях коми, крапка означає роздільник цілої та дробової частини, який виводиться за правилами системної локалі: `"$###,###.00"` відображає `$1 234,45`
`.`	Роздільник цілої та дробової частини, виводиться за правилами системної локалі: `"####.#"` відображає `1234.5`
`%`	Символ відсотка, помножує значення на `100` і додає знак відсотка в позицію, зазначену символом-заповнювачем `%`: `"##0.00%"` відображає число `0.12345` як `12.35%` (відбувається округлення)
`,`	Роздільник тисяч, додає встановлені в регіональних налаштуваннях символи як роздільники тисяч: `"###,###,###.00"` відображає `1 234,45`
`E- e-`	Відображає значення в експоненційному форматі зі знаком порядку тільки для негативних значень: `"#.####E-00"` відображає `1.23445E03`
`E+ e+`	Відображає значення в експоненційному форматі зі знаком порядку як для негативних, так і для позитивних значень: `"#.####E+00"` відображає `1.23445E+03`
`/`	Відокремлює день, місяць і рік для форматування значень дат.
`m`	Вказує, як відображати місяці в датах: `m` відображає `1`, `mm` - `01`, `mmm` - `Січ`, `mmmm` - `січень`
`d`	Вказує, як відображати дні в датах: `d` відображає `1`, `dd` - `01`, `ddd` - `Вс`, `dddd` - `неділя`
`y`	Показує день року як число від `1` до `366`
`yy`	Вказує, як відображати роки в датах: `yy - 23, yyyy - 2023`
`q`	Відображає квартал року як число від `1` до `4`
`w`	Показує день тижня як число (`1` - неділя)
`ww`	Відображає тиждень року як число від `1` до `54`
`:`	Відокремлює години, хвилини і секунди в значеннях формату часу: `hh:mm:ss - 1:01:01, h:m:s - 1:1:1`
`h`	Вказує, як відображати години: `hh - 01, h - 1`
`n`	Вказує, як відображати хвилини: `nn - 01, n - 1`
`s`	Вказує, як відображати секунди: `ss - 01, s - 1`
`AM/PM`	Показує час у 12-годинному форматі часу з доданими `AM` або `PM`
`@`	Символьний заповнювач, відображає пробіл, якщо у поточній позиції немає символу в рядку, що форматується.

Символ-заповнювач	Дія (як приклад використовується число `1234.45`)
<	Показує всі символи в нижньому регістрі.
>	Показує всі символи в верхньому регістрі.

У разі зазначення кількості розрядів у користувацькому форматі, меншого за кількість розрядів числа, що форматується, відбувається округлення числа за арифметичними правилами (`5-9` у більшу сторону, `0-4` – у меншу). Наведемо приклад використання користувацьких форматів для округлення чисел, форматування дати/часу та зміни регістру символів рядка:

```
Sub UserFormat()
    Debug.Print Format(1234.65, "####.#")
    Debug.Print Format(1234.75, "####.#")
    Debug.Print Format(Now, "dddd dd/mmmm/yyyy hh:nn:ss AM/PM")
    Debug.Print Format("hello", ">")
Sub
```

Програма відпрацює, наприклад, так:

```
1234,7
1234,8
понеділок 24.квітень.2023 06:23:15 PM
HELLO
```

Існують вбудовані функції, що дозволяють організувати *взаємодію з користувачем*. Досі ми використовували оператор `Debug.Print` для виведення результатів роботи програм у вікно *Immediate*. Як альтернативу, можна використовувати функцію `MsgBox()`, яка може виводити результат роботи програми у графічному вікні. Ця функція має наступний синтаксис:

```
MsgBox(Prompt [, Buttons] [, Title] [, HelpFile, Context])
```

і повертає інформацію про кнопку, яку натиснув користувач у цьому вікні (всього 7 варіантів) (Таблиця 7).

Таблиця 7 – Значення констант, що повертаються функцією `MsgBox()`

Назва константи	Значення константи	Призначення
vbAbort	3	Користувач обрав кнопку *Перервати*
vbCancel	2	Користувач обрав кнопку *Скасувати*
vbIgnore	5	Користувач обрав кнопку *Пропустити*
vbNo	7	Користувач обрав кнопку *Ні*
vbOK	1	Користувач обрав кнопку *ОК*
vbRetry	4	Користувач обрав кнопку *Повторити*
vbYes	6	Користувач обрав кнопку *Так*

Єдиним обов'язковим аргументом є `Prompt` – рядок, який виводиться у вікні. Необов'язкові аргументи:

- `Buttons` – одна або декілька констант, що визначають кількість і тип кнопок у діалоговому вікні, а також вказують кнопку за замовчуванням та стандартні значки для повідомлень (Таблиця 8). Якщо потрібно використати

одночасно декілька констант, що визначають аргумент `Buttons`, їх вказують через знак конкатенації (`&`).

Таблиця 8 – Константи аргумента `Buttons` функції `MsgBox()`

Константа	Призначення
`vbAbortRetryIgnore`	Відображає командні кнопки *Перервати*, *Повторити*, *Пропустити*
`vbApplicationModal`	Користувач повинен відповісти на запит діалогового вікна для продовження роботи з додатком (до надання відповіді додаток призупиняється)
`vbCritical`	Відображається вікно з попереджувальним значком критичної помилки 

`vbDefaultButton1`
`vbDefaultButton2`
`vbDefaultButton3`
`vbDefaultButton4`
	Номер командної кнопки в діалоговому вікні, яка є кнопкою за замовчуванням.

`vbExclamation`	Відображається вікно з попереджувальним значком  . Використовується у вікні з інформацією, яка не вимагає відповіді.
`vbInformation`	Відображається вікно з попереджувальним значком  . Використовується у вікні з довідковою інформацією.
`vbMsgBoxHelpButton`	Додає до діалогового вікна кнопку *Довідка*. При натисканні на цю кнопку відкривається файл, який заданий в аргументі `HelpFile`, в розділі, заданому аргументом `Context`
`vbOKCancel`	Відображає кнопки *ОК* та *Скасувати*.
`vbOKOnly`	Відображає тільки кнопку *ОК* – аналогічна пропуску аргументу `Buttons`.
`vbQuestion`	Відображається вікно з попереджувальним значком  . Використовується разом з повідомленням-запитом.
`vbRetryCancel`	Відображає кнопки *Повторити* та *Скасувати*.
`vbSystemModal`	Користувач повинен відповісти на запит діалогового вікна для продовження роботи з додатком (до надання відповіді всі відкриті додатки призупиняються, діалогове вікно завжди залишається попереду інших вікон доки не буде закрите).
`vbYesNo`	Відображає кнопки *Так* та *Ні*.
`vbYesNoCancel`	Відображає кнопки *Так, Ні* та *Скасувати.*

- `Title` – рядок для заголовка вікна.
- `HelpFile` – рядок, що містить ім'я файлу довідки (який вже повинен бути створений у форматі Microsoft Compiled HTML Help [19]).
- `Context` – чисельний вираз, який вказує розділ у файлі довідки.

Нижче наведений приклад використання функції `MsgBox()` з повертанням нею константи та вигляд діалогового вікна та значення константи у вікні *Immediate* після натискання кнопки *Перервати* (кнопкою за замовчуванням призначена кнопка *Повторити*) (Рисунок 16):

```
buttonNum = MsgBox("Деяке повідомлення", vbAbortRetryIgnore _
          + vbCritical + vbDefaultButton2, "Заголовок")
Debug.Print "buttonNum="; buttonNum
```

Значення констант, що повертаються діалоговим вікном можуть бути використані далі у програмі.

Іноді (наприклад, при пакетній обробці даних) хотілося б, щоб вікно повідомлення через деякий час закривалося само собою. Це можна зробити за допомогою методу `Popup()` об'єкта `Wscript.Shell`. Для цього в проект через меню *References* потрібно додати посилання на *Windows Script Host Object Model* (файл C:\WINNT\system32\wshom.ocx), а після цього використовувати наступний код:

```
Dim oShell As New WshShell
buttonNum = oShell.Popup("Деяке повідомлення", 3, _
                         "Заголовок", vbAbortRetryIgnore)
Debug.Print "buttonNum="; buttonNum
```

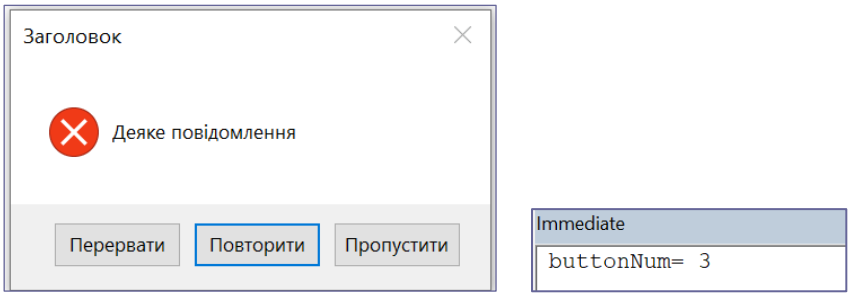


Рисунок 16 – Приклад використання функції `MsgBox()`

Інша функціональність такого вікна однакова з `MsgBox()`. Значення константи, що повертається, якщо користувач не натиснув на жодну кнопку, дорівнює `-1`.

Функція `InputBox()` дозволяє організувати введення даних користувачем через графічну форму з єдиним тестовим полем і має наступний синтаксис:

```
InputBox(Prompt [, Title] [, Default] [, XPos] [, YPos]
         [, HelpFile, Context])
```

Ця функція повертає рядок, що введений у текстове поле (тому може знадобитись перетворення типів у програмі). Єдиний обов'язковий аргумент `Prompt` зазначає рядок з інформацію, яка підказує користувачеві, що потрібно ввести у діалоговому вікні. Необов'язкові аргументи:

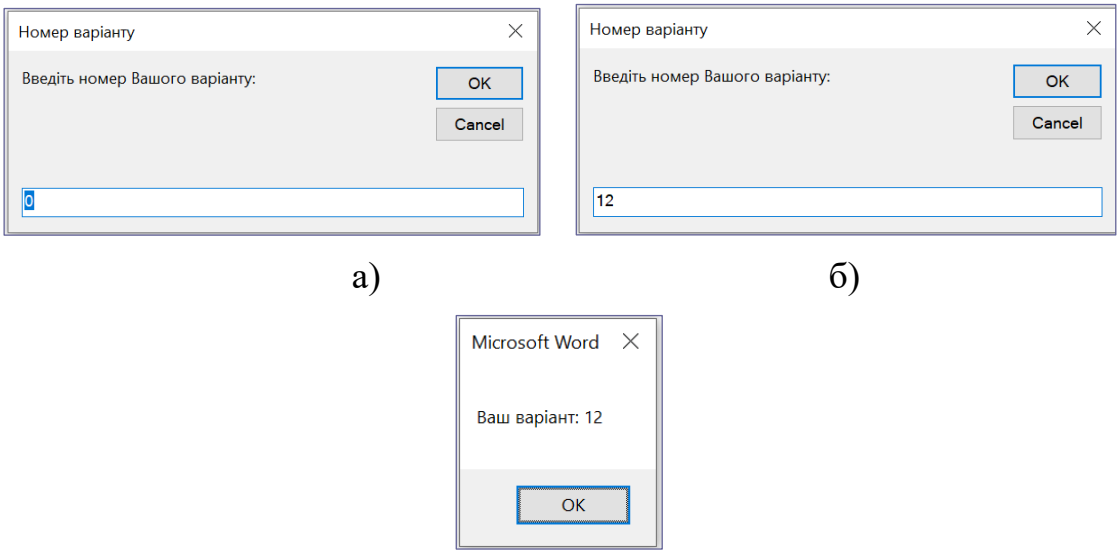
- `Title`, `HelpFile`, `Context` – аналогічні вже розглянутим однойменним аргументам функції `MsgBox()`;
- `Default` – вираз-рядок, що використовується як значення за замовчуванням і повертається функцією, якщо користувач нічого не ввів у текстове поле;
- `XPos`, `YPos` – задають координати виведення діалогового вікна на екрані монітора – `XPos` – горизонтальна відстань від лівого краю вікна. `YPos` – вертикальна відстань від верхнього краю вікна. Обидва відстані вимірюються в

твіпах (1 *твіп* = 1/15 пікселя = 0.0017638889 см). Якщо ці аргументи не вказуються, вікно виводиться по центру екрана.

Приклад коду з використанням `InputBox` (для привертання уваги користувача організується програвання звукового сигналу):

```
Sub InputBoxOptions()
    Dim inp As String

    inp = InputBox("Введіть номер Вашого варіанту: ", "Номер
варіанту", "0", 4000, 2000)
    MsgBox "Ваш варіант: " & inp
End Sub
```

Результат роботи програми показаний на Рисунок 17.



а) б)

в)

Рисунок 17 – Приклад використання функції `InputBox()`

Ще одна група вбудованих функцій VBA – *функції-замінники синтаксичних конструкцій*:

- `Choose` – приймає номер значення і кілька значень, що будуть повертатися функцією в залежності від номера. Є замінником операторів *If Then Else* та *Select Case*;
- `IIF` – розшифровується як *Immediate IF* (*Негайний IF*) – являє собою спрощений варіант *If Then Else*, коли перевіряється умова і повертається одне з двох значень;

`Switch` – приймає необмежену кількість пар типу "вираз – значення", перевіряє кожний вираз на істинність і повертає значення для першого виразу, який виявився істинним.

Нижче наведений приклад, що використовує описані функції-замінники синтаксичних конструкцій:

```
Sub FuncReplacers()
    Dim Num As Integer
    Dim Num As Integer
```

```
    Dim Res, More, CityName, Language As String

    Num = Int(Rnd * 3 + 1)
    Res = Choose(Num, "Перший", "Другий", "Третій")
    More = IIf(Num > 2, "Більше двох", _
                        "Менше або дорівнює двом")
    MsgBox Res & " - " & More
    CityName = InputBox("Введіть місто-столицю: Київ, " _
                      & "Вашингтон, Париж, Берлін")
    Language = Switch(CityName = "Київ", "українська", _
               CityName = "Вашингтон", "американська", _
               CityName = "Париж", "французська", _
               CityName = "Берлін", "німецька")
    MsgBox Language
End Sub
```

Згадаємо ще функцію для роботи з кольором – `RGB(red, green, blue)`, яка часто використовується при форматуванні об'єктів користувацьких форм та застосунків Microsoft Office. Ця функція приймає як аргументи інтенсивність від `0` до `255` основних кольорів палітри, що використовується для формування кольору на екрані: червоного (`red`), зеленого (`green`) та синього (`blue`). Наприклад, самий зелений з можливих кольорів вийде, якщо передані цій функції значення будуть виглядати як `RGB(0, 255, 0)`. Зауважимо, що такий підхід дозволяє формувати `256 x 256 x 256= 16 777 216` кольорів.

Зазначимо, що в VBA існують *функції для роботи з файлами та каталогами*, але ми їх розглядати не будемо, оскільки в API застосунків Microsoft Office існують частіше використовувані функції для роботи з файлами, що містять створений у цих застосунках контент.

Питання для самоконтролю

1. Які символи можна використовувати у іменах елементів коду VBA? З яких символів можуть починатися такі імена?
2. Назвіть рекомендації щодо регістру символів імен процедур та змінних і констант.
3. Назвіть вбудовані типи даних VBA, діапазон їх значень та обсяг пам'яті, яку вони займають.
4. Опишіть оператори оголошення підпрограм та змінних. Як оголошуються і чим відрізняються локальні та глобальні змінні?
5. Як оголошуються змінні типу `Variant`? Опишіть призначення опції `Explicit`.
6. Назвіть вбудовані типи даних VBA та опишіть значення змінних таких типів за замовчуванням.
7. Наведіть приклади присвоєння значень змінним усіх вбудованих типів VBA.
8. Опишіть способи обмеження доступності глобальних змінних модуля. На що впливає оголошення змінної з модифікатором `Static`?
9. Опишіть синтаксис оголошення констант VBA. Для чого використовують константу `vbCrLf`? Як у VBA зазначаються коментарі?
10. Опишіть способи перенесення частини оператора на новий рядок,

розміщення декількох операторів у одному рядку та пригнічення переведення на новий рядок у кінці оператора `Debug.Print`.

11. Назвіть відомі Вам арифметичні оператори VBA. Як записати операцію інкременту?
12. Назвіть відомі Вам оператори порівняння VBA. Що є результатом таких операторів? Як виконати порівняння посилань на два об'єкти?
13. Назвіть відомі Вам логічні оператори VBA. Коли вони використовуються? Опишіть роботу цих операторів в залежності від значень їх операндів.
14. Опишіть пріоритети операторів VBA. Для чого використовують оператор конкатенації?
15. Опишіть оператор передачі керування *If Then Else.* Наведіть приклад вкладення таких операторів.
16. Опишіть оператор передачі керування *Select Case*. Які форми може мати значення у частинах *Case*? Як зазначається гілка, куди передається керування у разі, якщо у жодному *Case* значення не знайдене?
17. Опишіть синтаксис циклу типу *For ... Next.* Коли доцільне його використання?
18. Опишіть синтаксис циклу типу *Do ... Loop.* Коли доцільне його використання? Чим відрізняється цикл з передумовою та з постумовою? Чим відрізняється застосування ключових слів `While` та `Until` у оголошенні циклів?
19. Опишіть синтаксис оголошення масиву. Яка опція дозволяє визначати значення початкового індексу масиву? Як оголошуються багатовимірні масиви?
20. Шо являє собою динамічний масив? Приведіть приклади його оголошення та зміни розміру із збереженням елементів. Опишіть створення та заповнення масиву функцією `Array()`.
21. Опишіть спосіб обходу динамічного масиву циклом *For ... Next* із використанням функцій визначення розміру масиву. Опишіть спосіб обходу динамічного масиву циклом *For Each ... Next*.
22. Назвіть ознаки класифікації процедур VBA. Чим відрізняється підпрограма від функції та як вони оголошуються та працюють?
23. Як виконується виклик підпрограм та функцій із інших процедур?
24. Опишіть відмінності передачі аргументів до процедур по значенню та по посиланню. Як така передача оголошується і працює? Коли доцільно виконувати передачу аргументів за посиланням?
25. Як оголошуються необов'язкові параметри процедур? Яким чином викликаються процедури, до яких не передаються необов'язкові параметри?
26. Назвіть відомі Вам функції VBA для обробки числових даних. Як виконують округлення чисел функції `Int()`, `Fix()` та `Round()`?
27. Назвіть відомі Вам функції VBA для конвертації та перевірки типів даних.
28. Назвіть відомі Вам функції VBA для обробки рядків. Опишіть символи шаблону оператора `Like`. Яка опція дозволяє регулювати врахування регістру символів при порівнянні?
29. Назвіть відомі Вам функції VBA для роботи з датами та часом. Як задаються літералами та як зберігаються дата та час у VBA? Назвіть шаблони інтервалів

дат та часу, які використовуються функціями.

30. Опишіть функції та можливості форматування даних у VBA. Які іменовані формати можуть вони приймати? Назвіть дії символів-заповнювачів користувацьких форматів.
31. Опишіть синтаксис та способи застосування функцій `MsgBox()` та `InputBox()`. Що повертають ці функції?
32. Які функції-замінники синтаксичних конструкцій Вам відомі? Опишіть їх синтаксис та застосування.

Тестові завдання для перевірки знань

1. Оберіть, що можна віднести до визначення макросу: макрос – це:
   а) програма, написана на C++;
   б) команда, вбудована у Microsoft Word;
   в) програма, написана на VBA;
   г) команда, вбудована у Microsoft Excel.
2. Оберіть головну відмінність підпрограми від функції:
   а) підпрограма і функція – це синоніми, відмінностей між ними немає;
   б) функція повертає результат роботи у точку виклику, а підпрограма – ні;
   в) процедури можна оголошувати тільки з модифікатором доступності `Private`, а функції – з будь-якими модифікаторами доступності;
   г) підпрограма повертає результат роботи у точку виклику, а функція - ні.
3. Запис оператора множинного вибору у VBA починається з ключових слів:
   а) `If Case`;
   б) `Select Case`;
   в) `Switch Case`.
4. Оберіть правильний варіант запису операції присвоєння значення змінній:
   а) вираз = ім'я_змінної;
   б) ім'я_змінної : = вираз;
   в) ім'я_змінної = вираз;
5. Оберіть вбудований тип даних, відсутній у VBA:
   а) `Long`;
   б) `Byte`;
   в) `Short`;
   г) `Integer`;
   д) `Single`.
6. Запишіть результат обчислення:
   27\6 =
7. Запишіть результат обчислення:
   27 Mod 6 =
8. Оберіть дозволені компілятором VBA імена змінних (оберіть декілька варіантів):
   а) `1stVar`;
   б) `my_Var`;
   в) `myVar`;

г) `var1`;
д) `var_1`.

9. Оберіть символи, що дозволяють перенести оператор на новий рядок:
   а) :
   б) ;
   в) _
   г) &
   д) +
10. Оберіть допустимі виклики процедур (оберіть декілька варіантів):
   а) `Call MyProc(5,"Y")`
   б) `MyProc 5,"Y"`
   в) `Call MyProc 5,"Y"`
   г) `MyProc(5,"Y")`
   д) `res = MyProc(5,"Y")`

Завдання для самостійного виконання

1. Розробіть програму на мові VBA, використовуючи оператори *If Then Else*: виведіть у вікні `MsgBox()` словесний опис будь-якого цілого числа у діапазоні від 100 до 999 включно, введеного функцією `InputBox()` (наприклад, 159 – "сто п'ятдесят дев'ять"). На основі розробленої програми створіть ще одну, проте використовуйте у ній оператор *Select Case*.
2. Розробіть програму на мові VBA, використовуючи оператори *If Then Else*: виведіть у вікні `MsgBox()` назву року за східним календарем для будь-якого року, введеного функцією `InputBox()`, починаючи з 1984. (У східному календарі прийнятий 60-річний цикл, що складається з 12-річних підциклів, що позначаються назвами кольорів: зелений, червоний, жовтий, білий і чорний. У кожному підциклі роки носять назви тварин: пацюка, корови, тигра, зайця, дракона, змії , коня, вівці, мавпи, курки, собаки і свині. 1984 рік був початком циклу – роком зеленого пацюка). На основі розробленої програми створіть ще одну, проте використовуйте у ній оператор *Select Case*.
3. Розробіть програму на мові VBA, використовуючи оператори *If Then Else*: за допомогою генератора випадкових чисел, що генерує числа від 0 до 1, задайте три цілочисельні координати для точки у просторі. Виведіть у вікні `MsgBox()` інформацію про положення точки:
   - чи лежить вона на початку координат;
   - чи лежить вона на будь-якій з координатних осей;
   - чи лежить вона на осі X;
   - чи лежить вона на осі Y;
   - чи лежить вона на осі Z.

   На основі розробленої програми створіть ще одну, проте використовуйте у ній оператор *Select Case.*
4. Розробіть програму на мові VBA, використовуючи оператори *If Then Else*: за допомогою генератора випадкових чисел, що генерує числа від 2 до 5, задайте оцінки для п'яти студентів. Розрахуйте і виведіть на екран значення відсотка

успішності (враховуються оцінки 3, 4, 5) і значення відсотка якості (враховуються оцінки 4, 5) по групі студентів. На основі розробленої програми створіть ще одну, проте використовуйте у ній оператор *Select Case.*

5. Розробіть програму на мові VBA, використовуючи оператори *If Then Else*: для довільних координат двох точок A і B на площині виведіть на екран назву відрізка (0A або 0B), який утворює більший кут з віссю X (кут вимірюється проти годинникової стрілки). На основі розробленої програми створіть ще одну, проте використовуйте у ній оператор *Select Case.* При необхідності виконайте заміну умови *If Then Else* виразом *Select Case* за допомогою функції-замінника синтаксичної конструкції `IIf()`.
6. Розробіть програму на мові VBA, використовуючи цикл типу *For ... Next*: за допомогою генератора випадкових цілих чисел введіть до масиву 20 цілих чисел у діапазоні від 1 до 100 включно. Розрахуйте та виведіть на екран середнє арифметичне елементів масиву та окремо елементи масиву, значення яких більше або дорівнює середньому арифметичному і окремо елементи масиву, значення яких більше середнього арифметичного. На основі розробленої програми створіть ще одну, проте використовуйте у ній цикли типу *Do ... Loop* з передумовою.
7. Розробіть програму на мові VBA, використовуючи цикл типу *For ... Next*: за допомогою генератора випадкових цілих чисел в діапазоні від 1 до 100 включно задайте значення елементів двох квадратних матриць з порядком 3, обчисліть різницю цих матриць і знайдіть для визначник для матриці-результату.
8. Розробіть програму на мові VBA, використовуючи цикл типу *For ... Next*: для довільного дійсного числа X і цілого числа N > 0 розрахуйте і виведіть на екран суму

$$X - \frac{X^3}{3!} + \frac{X^5}{5!} - ... + \frac{-1^N}{(2 \cdot N + 1)!} X^{2N+1}$$

(вираз N! – "N факторіал" – позначає добуток всіх цілих чисел від 1 до N: N! = 1 · 2 · ... · N). Перевірте, що отримане число є наближеним значенням функції *sin* в точці X.

9. Розробіть програму на мові VBA, використовуючи цикл типу *For ... Next*: за допомогою генератора випадкових чисел, що генерує числа від 1 до 10 включно, задайте елементи квадратної матриці 5 порядку. Виведіть на екран добуток значень елементів з непарними значеннями і суму елементів з парними значеннями.
10. Розробіть програму на мові VBA, використовуючи цикл типу *Do ... Loop*: виведіть на екран послідовність і суму чисел Фібоначчі у діапазоні від 1 до 1000 (числа Фібоначчі – це послідовність цілих чисел, у якій кожне наступне число дорівнює сумі двох попередніх чисел).

ТЕМА 3 ЗАСОБИ VBA ДЛЯ РОБОТИ З ОБ'ЄКТАМИ

Мета: розглянути способи проєктування класів та створення об'єктів класів, технологію програмування та обробки подій у класах, типів помилок та засобів їх знаходження та виправлення, які надає мова VBA та Редактор Visual Basic.

План

1. Основні елементи ООП програм VBA.
2. Розробка модуля класу.
3. Створення та видалення об'єктів.
4. Додання методів до класу.
5. Події об'єктів.
6. Обробка помилок у програмі.

Основні терміни і поняття

Клас, поле, об'єкт, властивість, метод, подія, помилки часу виконання, обробник помилки.

Основні теоретичні положення

3.1 Основні елементи ООП програм VBA

Створення програми на VBA шляхом написання підпрограм та функцій можна віднести до *процедурного програмування*. Такі програми містять основну процедуру (точку входу до програми), з якої можуть викликатися інші процедури з передачею до них як аргументів змінних та результатів виконання операторів та/або третіх процедур. При цьому процедури можуть працювати як з локальними змінними, так і з глобальними змінними, оголошеними, наприклад, на рівні модуля (Рисунок 18).

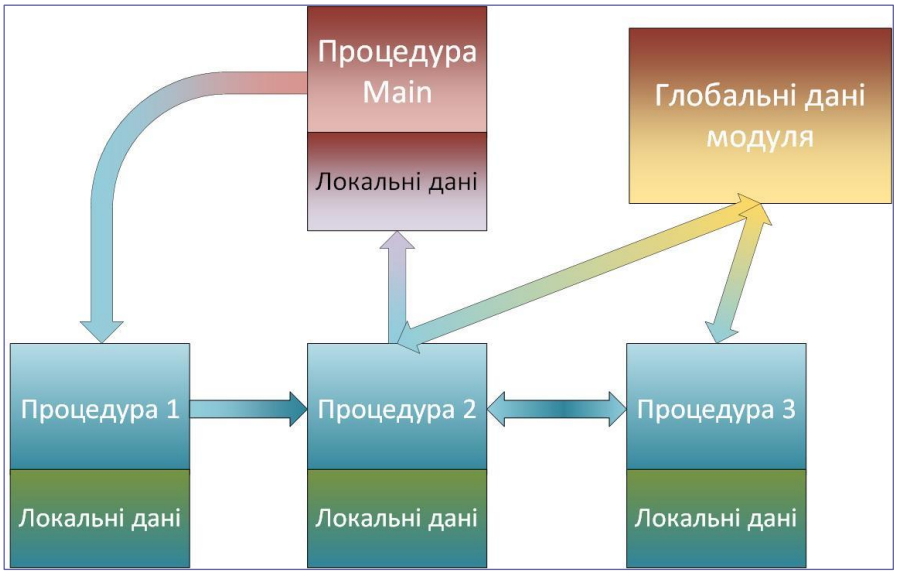


Рисунок 18 – Структура програми при процедурному програмуванні

При *об'єктно-орієнтованому програмуванні* (*ООП*) підхід інший: програмні одиниці – вони називаються *об'єктами класів* зберігають у собі недоступні безпосередньо ззовні дані (кажуть *інкапсулюють*), які характеризують саме ці об'єкти. Модуль класу VBA являє собою програмну одиницю, у якій оголошуються змінні, що характеризують об'єкти класу (їх називають *полями*) та процедури (їх називають *методами*), що забезпечують доступ та зміну значень полів, а також реалізують необхідний функціонал об'єктів класів. Об'єктно-орієнтована програма починає свою роботу з запуска звичайної процедури у звичайному модулі, у якій створюються об'єкти класів. Далі вони викликають методи один одного, можливо передаючи деякі параметри, та отримують результат їх роботи та використовують його (Рисунок 19).

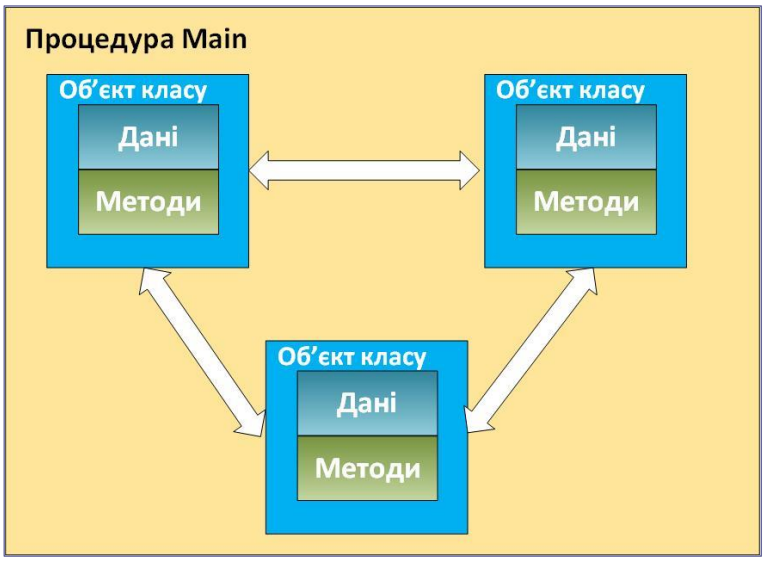


Рисунок 19 – Структура програми при об'єктно-орієнтованому програмуванні

Розглянемо основні елементи ООП програм [3]:

Клас – являє собою складений тип даних, у якому описується структура *об'єкта*. Зазвичай у ООП програмі існує відносно велика кількість класів, що являють собою програмну модель об'єктів реального світу, які приймають участь у роботі програмі (наприклад, студент, дисципліна, спеціальність – у програмі розрахунку рейтингу за результатами навчання; автобус, водій, маршрут – у програмі диспетчеризації рейсів тощо). Інкапсуляція полів об'єкта, що визначають його стан, та методів, що визначають поведінку об'єкта, у класі дозволяє будувати програми, у яких створюються велика кількість класів та велика кількість *екземплярів класів* (це – синонім для *об'єктів класів*), які взаємодіють між собою, відтворюючи взаємодію відповідних об'єктів реального світу. Класи у VBA розміщуються у *модулях класів*, поля у класах тільки оголошуються, але не ініціалізуються – їм не привласнюються значення.

Об'єкт (або *екземпляр класу*) – програмний елемент, що має унікальне у відповідному контексті програми ім'я і який є копією свого класу, але при

створенні його полям обов'язково привласнюються певні значення, які можуть змінюватися при виконанні програми.

У мові VBA одним з типів процедур є *властивість (Property)* – це сукупність методів класу, які забезпечують читання та запис інкапсульованого поля класу (об'єкту класу). Також в ООП програмах використовують *події (Events)*, які можна розуміти як сигнали, що генеруються програмним чином при зміні стану об'єкта або при виклику його метода, що реалізує поведінку об'єкта. Найчастіше у якості прикладу таких подій називають події, що генеруються елементами керування інтерфейсу користувача програми у відповідь на дії користувача над ними (клік по кнопці, обрання елементу зі списку тощо). У відповідь на генерацію події можна написати код, що буде автоматично запускатись і виконувати необхідні дії. Ми розглянемо програмування з орієнтацію на події інтерфейсу користувача у наступному розділі.

3.2 Розробка модуля класу

Розглянемо створення класу на прикладі задачі визначення точки на площині з можливістю розрахунку відстані між двома точками.

Для створення класу в Редакторі VBA необхідно обрати елемент меню *Insert–Class Module*, в результаті до проєкту додається модуль класу з ім'ям Class1. Змінимо йому ім'я на MyPoint, яке описує призначення класу (ім'я Point вже зайняте класом Word API), у вікні *Properties* (Рисунок 20).

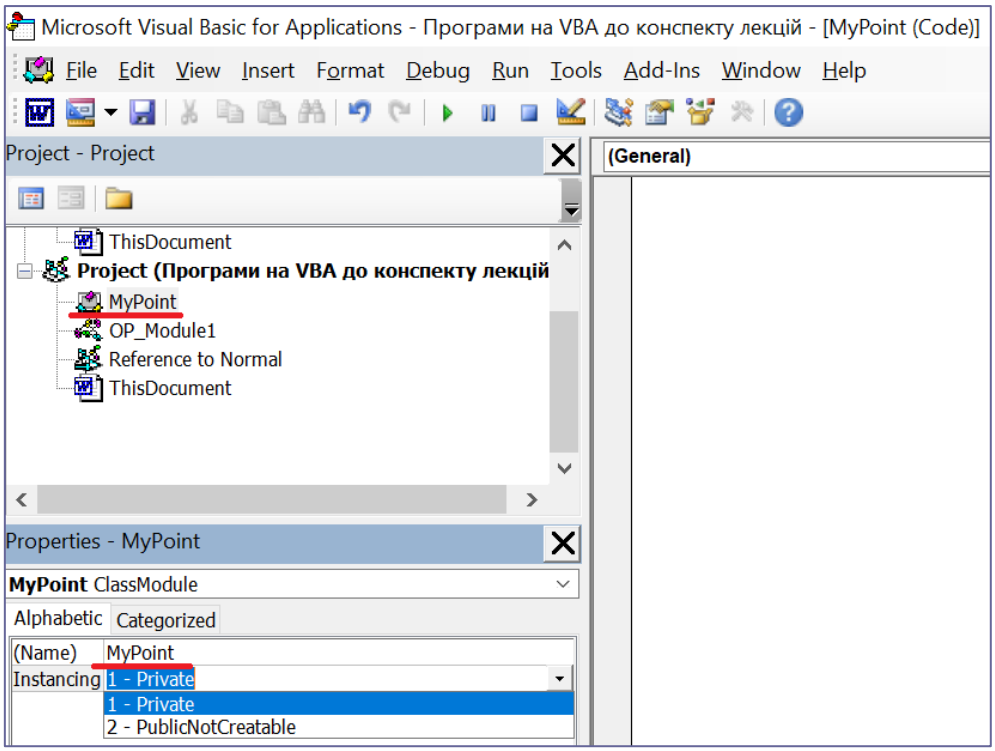


Рисунок 20 – Перейменування модулю класу

Окрім імені, у цьому вікні відображається властивість Instancing. При установці його значення у 1-Private (за замовчуванням) члени класу будуть доступні тільки з модулів даного проекту, при установці його значення у 2-

PublicNotCreatable, члени класу також будуть доступні тільки з модулів даного проекту і екземпляр класу не можна створювати у інших проєктах.

Вочевидь клас MyPoint повинен мати два поля – координату `x` та координату `y` (припустимо, що вони мають тип `Integer`). Відкриємо вікно коду модулю класу MyPoint та введемо до *розділу оголошень* (*Declarations*):

```
'Клас точки на площині
Option Explicit

'Поля класу - координати точки
Public x As Integer
Public y As Integer
```

Найпростіше оголошення класу на цьому завершене і можна створювати об’єкти цього класу.

3.3 Створення та видалення об'єктів

Об’єкти класу зазвичай створюються і з ними працюють у процедурах та функціях звичайного модулю, тому командою *Insert-Module* виконаємо вставку звичайного модуля та надаймо йому ім’я, наприклад, Module3 у вікні *Properties*. Зазначимо раніше розглянуті директиви `Option Explicit` (див. вище) та `Option Private Module` (див. вище) у розділі оголошень модуля:

```
Option Explicit
Option Private Module
```

Створення об’єктів (екземплярів) класу MyPoint будемо виконувати у підпрограмі `TestMyPoint()` модуля Module3:

```
Private Sub TestMyPoint()
    Dim p1 As MyPoint

    Set p1 = New MyPoint
    Debug.Print "x=" & CStr(p1.x), "y=" & CStr(p1.y)
    Set p1 = Nothing

End Sub
```

У цьому прикладі ми оголошуємо змінну з ім’ям **p1** як об'єкт класу **MyPoint**, а потім присвоюємо цій змінній оператором `Set` посилання на об’єкт, що створюється за допомогою оператора `New`. Синтаксис цього оператора наступний:

Set змінна-об’єкт **=** [**New**] вираз-об’єкта | **Nothing**

вираз-об’єкта - вираз, що складається з назви класу об'єкта або з імені іншої оголошеної змінної цього класу або з функції/методу, що повертає об'єкт цього класу.

При запуску наведений вище код виведе у вікно *Immediate* значення полів об'єкту **p1** (перетворені до рядків функцією `str()`), що дорівнюють значенню за замовчуванням для змінних типу `Integer`:

```
x= 0            y= 0
```

Останній оператор підпрограми встановлює значення об’єкту змінної в `Nothing`, тим самим видаляючи його з пам’яті. Об'єкт можна і не видаляти – він буде видалений автоматично після завершення процедури. Однак явне видалення об'єктів – це "правило хорошого тону", яке дозволяє уникнути конфліктів імен та перевитрат ресурсів.

Модуль класу у списку процедур/подій у правій верхній частині вікна Редактора коду містить дві події: Initialize та Terminate (Рисунок 21).

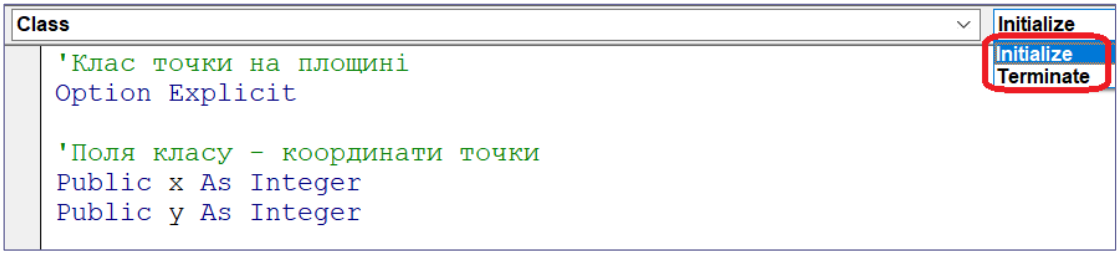


Рисунок 21 – Події класу

Обрання кожної з них додасть до коду модуля класу заготовки двох *обробників подій* (підпрограм, що не мають параметрів): `Class_Initialize()`, яка може використовуватись як *конструктор* – підпрограма для ініціалізації полів об'єкта під час його створення, та `Class_Terminate()`, яка може використовуватись як *деструктор* – підпрограма, що автоматично запускається при видаленні об’єкта з пам’яті.

Визначимо ці обробники подій для нашого класу наступним чином:

```
'Клас точки на площині
Option Explicit

'Поля класу - координати точки
Public x As Integer
Public y As Integer

'Конструктор
Private Sub Class_Initialize()
    x = 1
    y = 1
End Sub

'Деструктор
Private Sub Class_Terminate()
    Debug.Print "Об'єкт MyPoint видаляється з пам'яті."
End Sub
```

Якщо тепер запустити процедуру `TestMyPoint()` (див. вище), у вікно *Immediate* буде виведено:

```
x= 1           y= 1
Об'єкт MyPoint видаляється з пам'яті.
```

Після створення об’єкту класу ми можемо змінювати значення його полів-змінних (Рисунок 22), оскільки вони оголошені у класі MyPoint як `Public`.

Але на практиці часто існують певні обмеження на значення, які присвоюються полям об’єкту (наприклад, якщо в класі, який описує людину – `Person` є поле `age`, що містить кількість повних років людини, то окрім обмеження, що накладає тип поля `Integer`, повинно бути встановлене обмеження на діапазон значень, які можуть присвоюватись полю). Але такі обмеження неможливо додати до поля – воно може обмежуватись тільки зазначенням для нього того або іншого типу даних. Тому використовують підхід, який називається *інкапсуляція*, її суть полягає у тому, що поле закривають для безпосереднього доступу модифікатором `Private`, а в клас вводять методи з модифікатором `Public`, які забезпечують отримання значення поля та встановлення значення для поля. У таких методах можна додавати перевірки на дозволені значення, що присвоюються полю, та/або перевірки можливості отримання поточного значення. У VBA такі методи реалізується через процедури типу *властивості* (*Properties*).

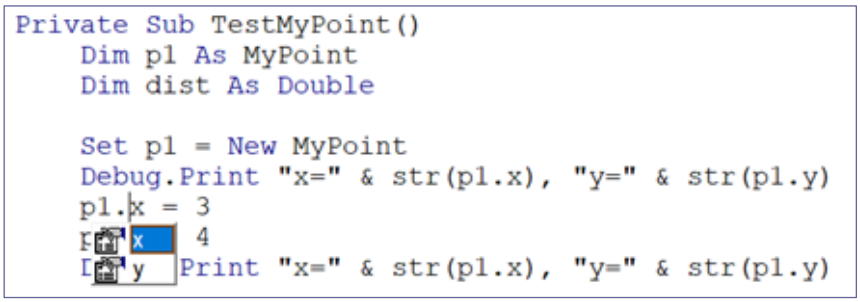


Рисунок 22 – Зміна значення публічного поля об’єкту класу

Виконаємо інкапсуляцію для нашого класу та додаймо властивості отримання значення (їх називають *геттери*) та властивості встановлення значення (їх називають *сеттери*) для полів нашого класу, які інкапсулюємо замінивши модифікатор доступу на `Private` (ми дещо змінимо імена полів, щоб віддати імена `X` та `Y` властивостям, які будуть використовуватися ззовні класу):

```
'Клас точки на площині
'Поля класу - координати точки
Private iX As Integer
Private iY As Integer

'Конструктор
Private Sub Class_Initialize()
    iX = 1
    iY = 1
End Sub

'Деструктор
Private Sub Class_Terminate()
```

```
    Debug.Print "Об'єкт MyPoint з координатами x="; iX; ",
y="; iY; " видаляється з пам'яті."
End Sub
```

Для додання до модуля класу властивості потрібно встановити курсор у необхідне місце класу та обрати команду *Insert-Procedure*. У вікні вставки процедури необхідно задати їй ім'я (ми задаймо `X` та `Y` з великої літери відповідно до узгодження щодо іменування процедур), обрати тип *Property* і модифікатор *Public* (Рисунок 23).

Після клікання по кнопці *OK* до коду буде додана заготовка властивості, що по суті являє собою методи читання значення поля `iX: Public Property Get X() As Variant` та присвоювання значення полю `iX: Public Property Let X(ByVal vNewValue As Variant)`. Виконаємо зміну типів результату, що повертається геттером, та параметру сеттера на `Integer`, оскільки ми маємо саме такий тип полів класу, а також змінимо параметр сеттера на однойменний для поля. Аналогічним чином додаймо та відредагуємо властивість для поля `iY`.

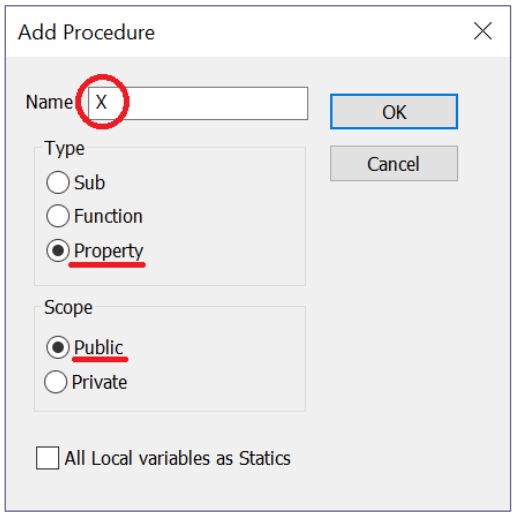


Рисунок 23 – Вставка властивості класу

Для демонстрації можливості організації обмеження на значення, що привласнюються полям об'єкта, додаймо до сеттерів дозвіл встановлювати тільки позитивні значення:

```
'Властивість для поля iX
Public Property Get X() As Integer
    X = iX
End Property

Public Property Let X(ByVal iXValue As Integer)
    If (iXValue >= 0) Then
        Me.X = iXValue
    Else
        Debug.Print "Встановлення негативних значень" _
                    & " неможливе, значення не змінено."
    End If
End Property
```

```
'Властивість для поля iY
Public Property Get Y() As Integer
    Y = iY
End Property

Public Property Let Y(ByVal iYValue As Integer)
    If (iYValue >= 0) Then
        Me.Y = iYValue
    Else
        Debug.Print "Встановлення негативних значень" _
                    & " неможливе, значення не змінено."
    End If
End Property
```

Виконаємо зміни у коді процедури `TestMyPoint()`, щоб перевірити роботу класу з інкапсульованими полями:

```
Private Sub TestMyPoint()
    Dim p1 As New MyPoint

    Debug.Print "x=" & str(p1.X), "y=" & str(p1.Y)
    p1.X = -1
    p1.X = 4
    p1.Y = 5
    Debug.Print "x=" & str(p1.X), "y=" & str(p1.Y)

    Set p1 = Nothing
End Sub
```

У цьому прикладі оголошення та створення об’єкта класу виконуються єдиним оператором. Зверніть увагу, на вигляд доступу до полів класу – вони відбуваються через властивості (оскільки імена полів `iX` та `iY`). У разі намагання присвоїти полю негативне значення, буде виведений рядок з інформацією про неможливість такого значення. Виведення у вікно *Immediate* після запуску процедури буде наступним:

```
x= 1        y= 1
Встановлення негативних значень неможливе, значення не змінено.
x= 4        y= 5
Об'єкт MyPoint з координатами x=4, y=5 видаляється з пам'яті.
```

Зауважимо, що ключове слово `New` не може використовуватись при оголошенні об'єктів з ключовим словом `WithEvents` (таке оголошення буде розглянуте нижче у цьому розділі) та при створенні змінних вбудованих типів (`Integer`, `String` тощо), тому іноді необхідно використовувати оператор `Set`.

Якщо у класі відсутня властивість `Let`, то відповідне поле класу можна тільки читати, а змінювати не можна. Зауважимо, що VBA підтримує ще властивість `Set`, що присвоює полю посилання на інший об’єкт (хоча це можливо зробити і за допомогою `Let`).

3.4Додання методів до класу

До створеного класу можна додавати процедури і функції, які реалізують необхідний функціонал класу (визначають поведінку його об'єктів). Вони будуть доступні для виклику з об'єктів класу як *методи*, у разі якщо вони оголошені з модифікатором доступу `Public`, і не будуть доступні, якщо оголошені з модифікатором доступу `Private`. Всі процедури і функції класу, які не планується використовувати як методи, повинні бути `Private`. Це також вимагається принципом *інкапсуляції*.

Створення методів у модулі класу нічим не відрізняється від створення процедур і функцій у звичайному модулі.

Додаймо до класу MyPoint метод, що приймає як параметри координати іншої точки на площині, розраховує за теоремою Піфагора та повертає значення відстані від поточної точки до точки з координатами-параметрами метода (Рисунок 24):

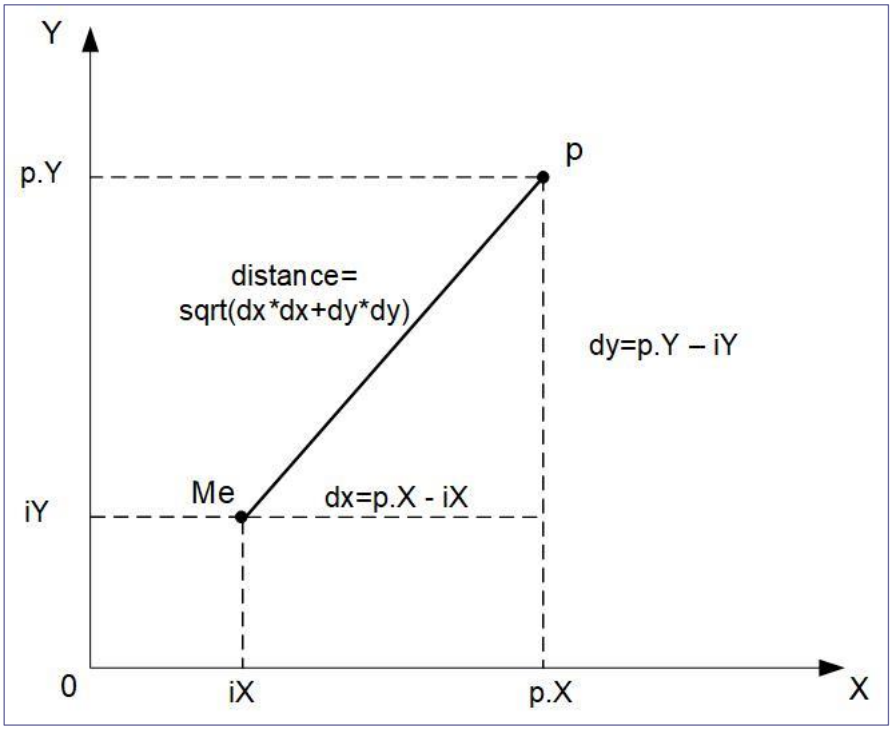


Рисунок 24 – Задача розрахунку відстані між точками на площині

```
'Повертає відстань між поточною точкою та іншою точкою,
'заданою координатами-параметрами
Public Function CalcDistance(x As Integer, y As Integer)
                                    As Double
    Dim dx As Integer, dy As Integer

    dx = x - iX
    dy = y - iY
    CalcDistance = Sqr(dx ^ 2 + dy ^ 2)
End Function
```

Тоді у процедурі `TestMyPoint()` можна додати оператор виклику методу з об’єкту, який розраховує відстань між поточною точкою та точкою, заданою параметрами методу `CalcDistance()`:

```
Private Sub TestMyPoint()
    Dim p1 As New MyPoint

    Debug.Print "x=" & str(p1.X), "y=" & str(p1.Y)
    p1.X = -1
    p1.X = 4
    p1.Y = 5
    Debug.Print "x=" & str(p1.X), "y=" & str(p1.Y)

    Debug.Print "Distance = "; p1.CalcDistance(1, 1)

    Set p1 = Nothing
End Sub
```

При запуску процедури вона між іншим виведе:

```
Distance =  5
```

Ми додали до класу MyPoint метод, що розраховує відстань від поточної точки до іншої, але у якості параметрів ми обрали координати іншої точки, які мають значення вбудованого типу (`Integer`). Але ніщо не заважає у якості параметру використовувати об’єкт (у нашому випадку об’єкт класу MyPoint зі значеннями координат другої точки). Додаймо такий метод `Public Function CalcPointDistance(p As MyPoint) As Double`:

```
'Повертає відстань між поточною точкою та іншою точкою
Public Function CalcPointDistance(p As MyPoint) As Double
    Dim dx As Integer, dy As Integer

    dx = p.X - iX
    dy = p.Y - iY
    CalcPointDistance = Sqr(dx ^ 2 + dy ^ 2)
End Function
```

Тоді в процедурі `TestMyPoint()` можна викликати цей метод з об'єкта `p1`, передавши йому як параметр другу точку, наприклад, `p2` із значеннями полів за замовчуванням (`1, 1`):

```
Private Sub TestMyPoint()
    Dim p1 As New MyPoint
    Dim p2 As New MyPoint

    Debug.Print "x=" & str(p1.X), "y=" & str(p1.Y)
    ap1.X = -1
    ap1.X = 4
    ap1.Y = 5
    Debug.Print "x=" & str(p1.X), "y=" & str(p1.Y)

    Debug.Print "Distance = "; p1.CalcDistance(1, 1)
```

```
    Debug.Print "Distance = "; p1.CalcPointDistance(p2)

    Set p1 = Nothing
    Set p2 = Nothing
End Sub
```

Відстань буде тією ж:

```
Distance =  5
```

На жаль, VBA не підтримує такі технології ООП як перевантаження методів та конструкторів. З цієї причини імена методів, що розраховують відстань, оголошені різними (у мовах програмування що підтримують поліморфізм, їх можна зазначати однаковими, компілятор розуміє, що методи різні, оскільки для них відрізняються типи параметрів та/або кількість параметрів та/або послідовність параметрів). З цієї ж причини VBA не підтримує створення конструктора з параметрами. Тому потрібно створювати об’єкт за допомогою конструктора без параметрів, а потім встановлювати необхідні значення полів властивостями. Також VBA не підтримує технологію успадкування, можливо, тому, що він позиціонується як мова автоматизації документів офісних застосунків із використанням готових об’єктних моделей цих застосунків.

3.5 Події об’єктів

Подія – це спеціальна процедура, яка викликається спеціальним оператором `RaiseEvent` (кажуть, що при цьому *подія генерується*) у разі зміни значення поля об'єкта класу (зміни його стану) або при виклику метода об'єкта класу (що реалізує його поведінку). Фактично цей виклик є сигналом, у відповідь на який у іншому об'єкті, який використовує об'єкт з подією для зміни його стану або виклику методу, автоматично викликається *метод-обробник події*, у якому можна запрограмувати необхідні дії.

Кожен модуль класу має дві вбудованих події (Рисунок 21):

Initialize – вона генерується при створенні об'єкта цього класу і у методі-обробнику цієї події – `Class_Initialize` зручно задавати значення за замовчуванням полям створюваного об’єкту – ми називали такий метод *конструктором*.

Terminate – вона генерується при видаленні об'єкта цього класу з і у методі-обробнику цієї події – `Class_Terminate` зручно програмувати дії, які необхідно виконати перед видаленням об'єкта з пам'яті – ми називали такий метод *деструктором*.

Також можливо оголошувати, генерувати та використовувати власні події, розглянемо як це робиться на прикладі події, що буде генеруватися при зміні значень полів об'єкта класу MyPoint.

Для цього, по-перше, додаймо до класу MyPoint у розділ декларацій оголошення події – процедури з ім'ям, наприклад, `CoordinateChanged`, яка буде приймати як параметр рядок з інформацію яке саме поле змінено:

```
Public Event CoordinateChanged(msg As String)
```

А, по-друге, до методів-сеттерів властивостей додаймо оператор ініціалізації події:

```
Public Property Let x(ByVal iXValue As Integer)
    If (iXValue >= 0) Then
        iX = iXValue
        'Генерування події
        RaiseEvent CoordinateChanged("Координата x змінена на x="
                                     & str(iXValue))
    Else
        Debug.Print "Встановлення негативних значень неможливе," _
                    & " значення не змінено."
    End If
End Property
```

та

```
Public Property Let y(ByVal iYValue As Integer)
    If (iYValue >= 0) Then
        iY = iYValue
        'Генерування події
        RaiseEvent CoordinateChanged("Координата y змінена на y="
                                     & str(iYValue))
    Else
        Debug.Print "Встановлення негативних значень неможливе," _
                    & " значення не змінено."
    End If
End Property
```

Зазначимо, що оголошувати, генерувати та обробляти події можливо тільки у так званих об'єктних модулях – до них, зокрема, відносяться модулі класу та користувацькі форми, але не відносяться звичайні модулі.

Тому, по третє, створимо ще один модуль класу з ім'ям MyPointWrapper (перекладається як "огортка"), який буде мати своїм полем об'єкт класу MyPoint, змінювати його поля у своєму методі та визначати метод-обробник події. Оскільки тепер клас MyPoint має подію, оголошення буде виглядати так:

```
Private WithEvents p As MyPoint
```

А повний код класу MyPointWrapper буде наступним:

```
'Клас, що огортає об'єктом точки на площині з подією
'Оголошення об'єкту класу з подією
Private WithEvents p As MyPoint

'Конструктор
Private Sub Class_Initialize()
    Set p = New MyPoint
End Sub

'Деструктор
Private Sub Class_Terminate()
    Debug.Print "Об'єкт MyPointWrapper видаляється " & _
```

```
                    "з пам'яті."
End Sub

'Виконує зміну координат об'єкту точки з подією
Sub ChangeCoordinates(ByVal newXValue As Integer, _
                      ByVal newYValue As Integer)
    Debug.Print "p.x="; p.x; "p.y="; p.y
    p.x = newXValue
    p.y = newYValue
End Sub

'Обробник події
Private Sub p_CoordinateChanged(msg As String)
    MsgBox msg
End Sub
```

У конструкторі створюється об'єкт з подією, а у методі `ChangeCoordinates()` його полям присвоюються нові значення, що викликає генерування події. Якщо, відкрити у вікні *Редактора коду* для класу MyPointWrapper список об'єктів зліва, у ньому окрім класу буде елемент з ім'ям **p**, яким ми оголосили об'єкт класу MyPoint з подією. Після обрання цього елементу у списку процедур/подій можна побачити подію CoordinateChanged, обрання якої додасть до коду заготовку методу-обробника події `p_CoordinateChanged` (Рисунок 25). Ми запрограмували цей обробник так, щоб він виводив у вікно `MsgBox()` повідомлення події з інформацією, яке поле змінювалося.

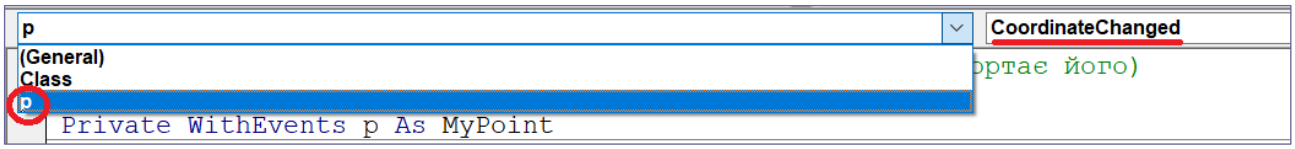


Рисунок 25 – Додання метода-обробника події до модуля класу

Нарешті у звичайному модулі Module3, у якому ми працювали з об'єктами класу MyPoint, додаймо підпрограму `TestMyPointWrapper()`, у якій буде створений об’єкт класу MyPoint з подією, з якого буде викликаний метод `ChangeCoordinates()`:

```
Sub TestMyPointWrapper()
    Dim pw As New MyPointWrapper

    pw.ChangeCoordinates 5, 6
End Sub
```

При запуску цієї процедури при виклику методу `ChangeCoordinates()` змінюються значення полів об'єкта MyPoint унаслідок чого генерується подія виникає подія `CoordinateChanged`, у відповідь на яку обробник події у класі MyPointWrapper виводить по черзі два діалогових вікна з повідомленнями про зміну поля `x` та зміну поля `y` (Рисунок 26).

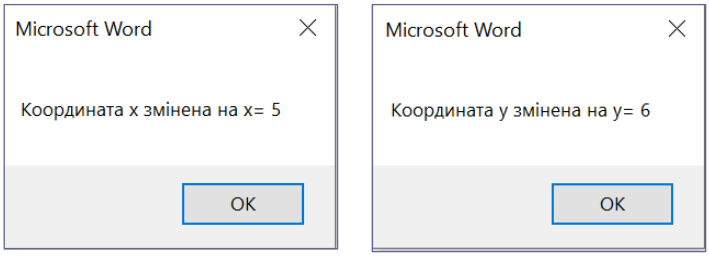


Рисунок 26 – Результат обробки події зміни координат точки

Події можна також генерувати у методах об'єкту класу, наприклад, оголосити подію

```
Public Event DistanceCalculated()
```

яка буде генеруватися у методі (зверніть увагу на відсутність дужок для події без параметрів)

```
Public Function CalcDistance(x As Integer, y As Integer) _
                              As Double
    Dim dx As Integer, dy As Integer

    dx = x - iX
    dy = y - iY
    CalcDistance = Sqr(dx ^ 2 + dy ^ 2)

    'Генерування події
    RaiseEvent DistanceCalculated
End Function
```

і у класі MyPointWrapper у методі, який змінює координати огорненого об'єкту класу MyPoint можна додати виклик методу розрахунку відстані до точки-параметра, у якому буде генеруватися подія:

```
Sub ChangeCoordinates(ByVal newXValue As Integer, _
                      ByVal newYValue As Integer)
    Debug.Print "p.x="; p.x; "p.y="; p.y
    p.x = newXValue
    p.y = newYValue
    p.CalcDistance 5, 6
End Sub
```

Тепер у класі MyPointWrapper у списку процедур\подій Редактору коду буде додатково відображатися подія DistanceCalculated (Рисунок 27), обрання якої додасть коду класу заготовку методу-обробника події, визначимо його код наступним чином:

```
Private Sub p_DistanceCalculated()
    MsgBox "Розрахована відстань до точки."
End Sub
```

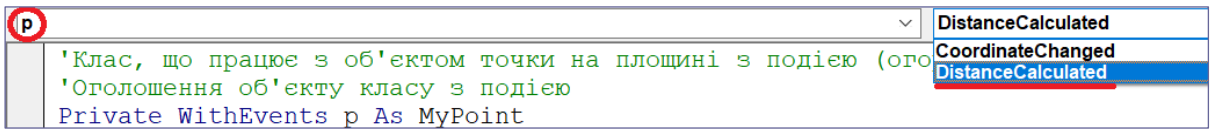


Рисунок 27 – Додання другої події до модуля класу

Тепер запуск методу `TestMyPointWrapper()`, крім виведення інформації у діалогових вікнах про зміну координат точки (Рисунок 26), виведе ще вікно з інформацією про виконання розрахунку відстані (Рисунок 28).

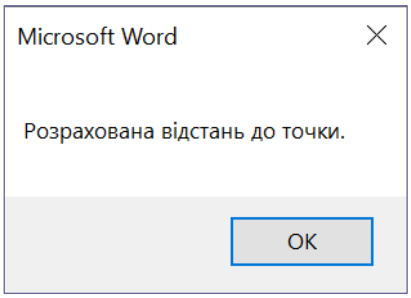


Рисунок 28 – Результат обробки події виконання розрахунку відстані

В якості подій можна використовувати відкриття або закриття документу, клікання мишею по кнопці, натискання клавіші, обирання елементу зі списку тощо. Події вбудовані у об'єкти об'єктних моделей Microsoft Office і рекомендується їх використовувати через вибір потрібного об'єкта та його події у вікні *Редактора коду Visual Basic*. Детально робота з подіями елементів керування – кнопок, прапорців, перемикачів тощо – буде розглянута при вивченні роботи з користувацькими формами у наступному розділі.

3.6 Обробка помилок у програмі

Всі помилки можна розділити на три групи:

- *синтаксичні* – неправильно написаний оператор, використання неоголошеної змінної при зазначенні `Option Explicit`, відсутність обов'язкового аргументу при виклику процедури тощо. Такі помилки автоматично "відловлюються" редактором коду VBA ще в процесі написання коду або при запуску програми. При цьому у діалоговому вікні надається інформація про причину та місце виникнення помилки (Рисунок 29) і автоматично процедура, у якій виявлена помилка, запускається у відлагоджувачі.

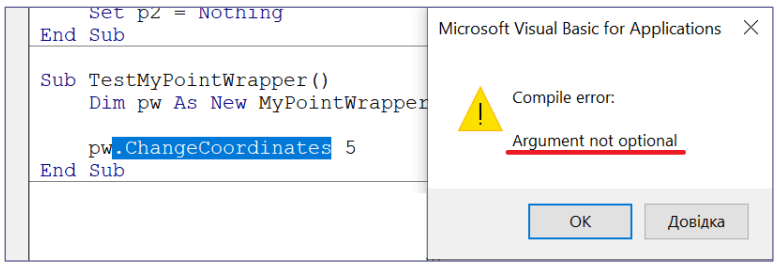


Рисунок 29 – Виявлення синтаксичної помилки відсутності обов'язкового аргумента

- *логічні* – коли програма виконується, але при виконанні програма веде себе не так, як передбачалося, та/або надає не ті результати, які передбачалися, унаслідок невірно спроєктованого коду, помилково передбачених значень та/або типів змінних, неявного перетворення даних тощо. Для усунення таких помилок необхідно зрозуміти, чому програма так себе веде. Для цього виконують запуск

програми у відлагоджувачі та перевірки та аналіз даних на кожному кроці виконання програми (засоби, які надає відлагоджувач Редактору Visual Basic будуть розглянуті нижче у цьому розділі).

- *помилки часу виконання* (*run-time error*) – коли програма виконується, спроєктована правильно, але при виконанні виникають проблеми, вирішити які програма не в змозі унаслідок введення користувачем неправильних даних: літер замість цифр, значень із заборонених діапазонів), зазначення помилкового шляху до файлу, з якого зчитуються дані, або його відсутності, запис до динамічного масиву з невизначеною кількістю елементів тощо. Зазвичай, при виникненні таких помилок програма припиняє свою роботу. Такі помилки повинні передбачатися програмістом та перехоплюватися і оброблятися програмою із виведенням відповідного повідомлення і, зазвичай, без припинення роботи програми. Далі у цьому розділі ми розглянемо способи перехоплення та обробки помилок часу виконання (див. нижче) [3].

Засоби відлагодження програм. Редактор Visual Basic містить відлагоджувач, що дозволяє запустити програму з покроковим виконанням або зупинити програму у необхідному місці (у *контрольній точці* – *breakpoint*) та виконувати її подальше покрокове проходження із аналізом даних на кожному кроці. Запуск програми в режимі покрокового виконання здійснюється командою меню *Debug–Step Into* або натисканням клавіши *F8*. Встановити у програмі *контрольну точку* можна, встановивши курсор у необхідному рядку коду і вибравши в меню *Debug Toggle* команду *Breakpoint* (або натиснувши на кнопку  на панелі інструментів *Debug* або натиснувши на клавішу *F9*), найзручніше – клікнути мишею по сірій рамці зліва від рядка коду (Рисунок 30).

```
'Повертає відстань між поточною точкою та іншою точкою,
'заданою координатами-параметрами
Public Function CalcDistance(x As Integer, y As Integer) As Double
    Dim dx As Integer, dy As Integer

    dx = x - iX
    dy = y - iY
    CalcDistance = Sqr(dx ^ 2 + dy ^ 2)
    'Генерування події
    RaiseEvent DistanceCalculated
End Function
```

Рисунок 30 – Встановлення контрольної точки для відлагоджувача

Видалення контрольної точки виконується повторною вище зазначеною дією. При запуску програма автоматично зупиниться на першій контрольній точці. Часто контрольну точку встановлюють на початку процедури, але після операторів оголошення змінних.

На жаль, точки зупину не зберігаються після закриття документа. Якщо необхідно запам'ятати місце зупинки між сеансами налагодження, то потрібно вставити у це місце рядок з єдиною командою `Stop`. Програма при запуску автоматично зупиниться на цьому рядку.

Після запуску програми з встановленою контрольною точною виконання програми буде призупинено (кажуть, що програма переходить у *режим паузи*) у

відповідному місці і наступний оператор, який повинен бути виконаний, буде виділений жовтим кольором.

У режимі можна виконувати наступні дії:

- продовжити виконання в покроковому режимі. Для цього можна скористатися кнопкою  панелі інструментів *Debug* або виконати команду *Debug–Step Into* або натиснути на клавішу *F8*;
- якщо у рядку програми викликається інша процедура, яка не повинна створити проблем (наприклад, вже відлагоджена), існує можливість виконати таку процедуру без зупинок і перейти до наступного оператору. Для цього використовується кнопка  панелі інструментів *Debug* або команда *Debug–Step Over* або клавіатурна комбінація *Shift +F8*;
- якщо у рядку програми викликається інша процедура і Ви перейшли в неї, виконали там необхідні дії і хочете швидко виконати її до кінця, використовуйте кнопку  панелі інструментів *Debug* або команду *Debug-Step Out* або клавіатурну комбінацію *Ctrl+Shift+F8*;
- якщо Ви хочете виконати код не покроково, а ділянками (наприклад, щоб не проводити послідовне виконання великих циклів), можна натиснути правою кнопкою миші на останньому операторі відповідної ділянки коду і у контекстному меню вибрати команду *Run to Cursor* або скористатися тією ж командою меню *Debug* або натиснути клавіатурну комбінацію *Ctrl+F8*. Програма виконає без зупинки відповідну ділянку коду і зупиниться у рядку, в якому знаходиться курсор;
- якщо необхідно тимчасово "виключити" якусь ділянку коду, можна скористатися командою меню *Debug–Set Next Statement* або натиснути клавіатурну комбінацію *Ctrl+F9*, а потім перетягнути жовту позначку по лівій межі до місця звідки необхідно продовжувати виконувати програму. Альтернативний (і більш поширений) спосіб тимчасового "виключення" ділянки коду – виділити її та закоментувати кнопкою  панелі інструментів *Edit*, а коли буде необхідно, виділити та зняти коментування кнопкою );
- щоб продовжити виконання прогарами після зупинки, можна клікнути на кнопці  панелі інструментів *Debug* (або стандартної панелі інструментів) або скористатися командою *Continue* (вона з'явиться замість команди *Run* в меню *Run*) або натиснути на клавішу *F5*;
- припинити виконання програми можна клікнувши на кнопці  панелі інструментів *Debug* (або стандартної панелі інструментів) або скористатися за командою *Reset* меню *Run* або натиснувши клавіатурну комбінацію *Al+F4*.

На наш погляд, із можливих альтернатив ініціації дій у режимі паузи найзручніше використовувати команди, задаючи їх натисканням відповідних клавіш.

Зазвичай, перехід у режим паузи необхідний, щоб переглянути поточні значення змінних, що використовуються, та/або виправити код. Поточні значення змінних можна переглянути у вікнах *Immediate*, *Locals* та *Watch* (про них буде розказано нижче), а можна скористатися підказками, що з'являються, у самому коді. Для того, щоб отримати інформацію про поточне значення змінної, досить навести на неї курсор миші (Рисунок 31) (якщо включений параметр за замовчуванням у меню *Tools–Options–Auto Data Tips*).

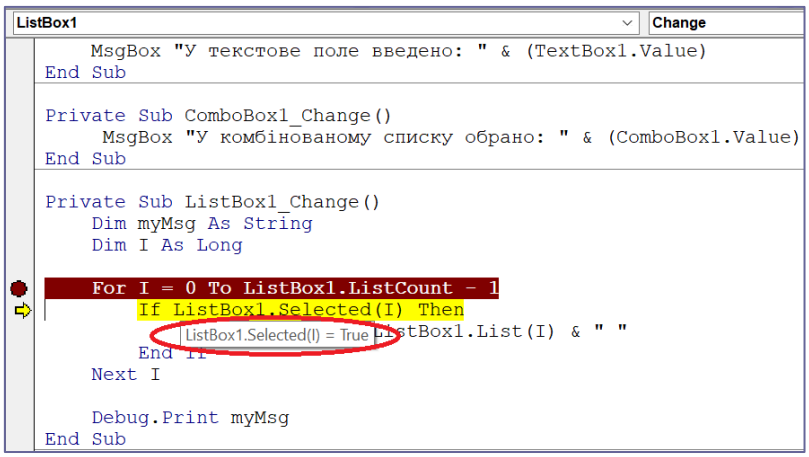


Рисунок 31 – Відображення значень змінних та властивостей при наведенні курсору

Щоб переглянути тип даних змінної та її область видимості, необхідно встановити на неї курсор і натиснути на кнопку  панелі інструментів *Edit* або обрати команду *Quick Info* в меню *Edit* або натиснути клавіатурну комбінацію *Ctrl+I*. При цьому для глобальних змінних у всплиаючій підказці будуть виведені ім'я та тип змінної, а для локальних – слово Local, ім'я та тип змінної. Для процедур та ж команда відображає їх сигнатуру та тип результату, що повертається (якщо він є) (Рисунок 32).

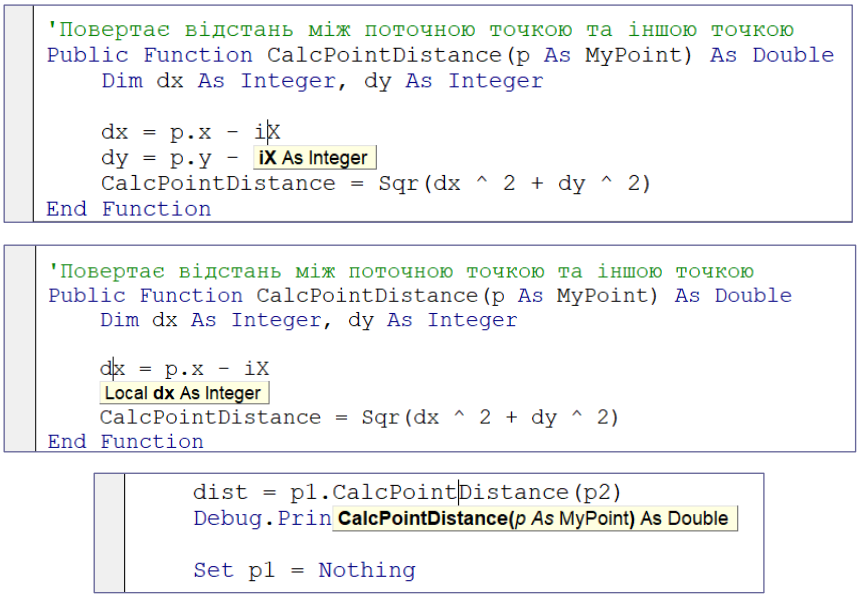


Рисунок 32 – Відображення інформації про змінні та методи

Всі можливості редагування коду в режимі налагодження залишаються. Якщо Ви зрозуміли, що змінній (властивості об'єкта) присвоєно невірне значення, то можна виправити відповідний рядок, повернутися назад за допомогою команди *Set Next Statement* меню *Debug* або клавіатурної комбінації *Ctrl+F9* і продовжити виконання програми з виправленим значенням.

Якщо при відлагодженні необхідно одночасно відстежувати значення багатьох змінних, більш функціональним є використання можливостей спеціалізованих вікон *Immediate*, *Locals* та *Watch* Редактору Visual Basic.

Вікно *Immediate* для програм виконує роль консолі, у яку виводяться результати роботи програм за допомогою оператора `Debug.Print`. У разі відсутності цього вікна на екрані, його можна відкрити командою меню *View–Immediate Window* або клавіатурною комбінацією *Ctrl+G*. У цьому вікні також можна переглядати або визначати значення змінних програми і властивостей об'єктів і навіть виконувати арифметичні та інші операції, для цього змінні, оператори та операції зазначаються після знаку питання (Рисунок 33).

```
'Повертає відстань між поточною точкою та іншою точкою,
'заданою координатами-параметрами
Public Function CalcDistance(x As Integer, y As Integer) As Double
    Dim dx As Integer, dy As Integer

    dx = x - iX
    dy = y - iY
    CalcDistance = Sqr(dx ^ 2 + dy ^ 2)
    'Генерування події
    RaiseEvent DistanceCalculated
End Function

'Повертає відстань між поточною точкою та іншою точкою
```

mmediate

```
?dx
 3
?CalcDistance
 5
Call Class_Terminate
Об'єкт MyPoint з координатами x= 1 , y= 1  видаляється з пам'яті.
?(15.5 * 2) / 4
 7,75
```

Рисунок 33 – Перегляд значень змінних/властивостей, запуск процедур/функцій, результатів обчислень у вікні Immediate

На жаль, для цього вікна відсутня можливість налаштування видалення попередніх результатів перед виведенням нових, тому залишається тільки виділяти вміст цього вікна клавіатурною комбінацією *Ctrl+A* та видаляти його натисканням клавіші *Delete*.

У разі, якщо необхідно переглянути значення змінних та властивостей об'єктів, що використовуються у процедурі, яка відлагоджується, використання засобів вікна *Immediate* буде неефективним. Набагато зручніше для цього використовувати вікно *Locals*. У цьому вікні виводяться значення всіх змінних і властивостей об'єктів процедури, що виконується, та змінних і властивостей модуля (у елементі Me) (Рисунок 34). Також у цьому вікні можна змінювати

значення змінних/властивостей – для цього у стовпці *Value* вікна необхідно ввести потрібне значення для визначеної змінної/властивості. Рядки та дати зазначаються як літерали – у лапках ("") та символах #, відповідно. У вікні *Locals* також можна змінювати значення елементів масивів і колекцій.

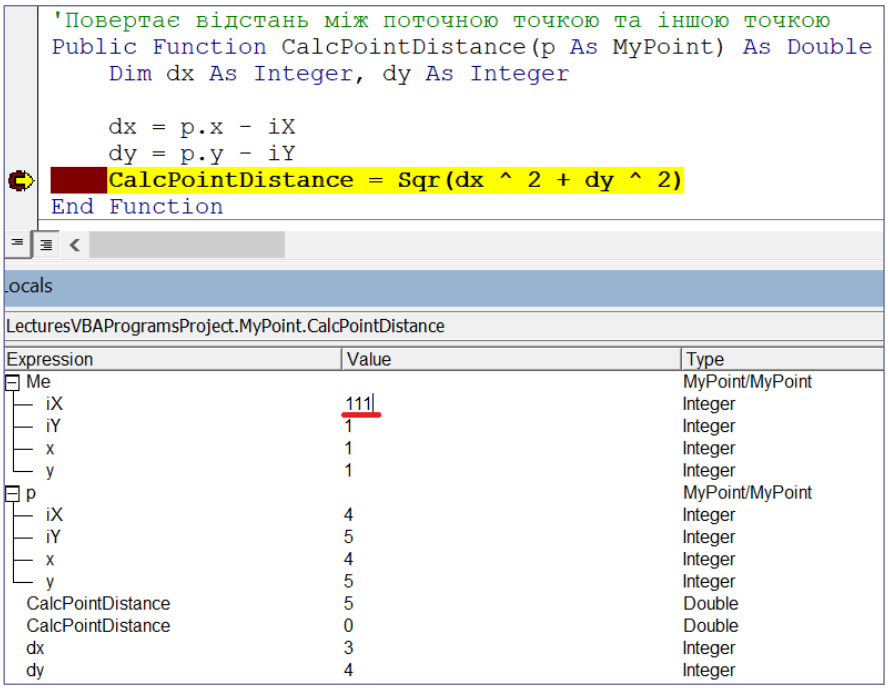


Рисунок 34 – Перегляд-зміна значень змінних/властивостей у вікні Locals

У разі відсутності вікна *Locals* на екрані, його можна відкрити командою меню *View– Locals Window*.

На відміну від вікна *Locals*, засоби вікна *Watch* дозволяють контролювати тільки ті змінні/властивості об'єктів або вирази за їх участю, які Вам цікаві. Для цього визначаються так звані *контрольованих вирази* (*Watch expression*). Створити контрольований вираз можна одним з наступних способів:

- клікнути по змінній/властивості/виразу у вікні Редактора коду правою кнопкою миші і у контекстному меню обрати команду *Add Watch*.
- скористатися командою *Add Watch* меню *Debug* і у однойменному вікні ввести необхідні змінні/властивості об'єктів або вирази у поле *Expression*;
- встановити курсор на необхідній змінній/властивості/виразу та скористатися командою *Quick Watch* меню *Debug*;
- виділити та перетягнути вираз з коду у вікно *Watch*. У разі відсутності вікна *Watch* на екрані, його можна відкрити командою меню *View– Watch Window*.

У перших двох випадках відкриється вікно *Add Watch* (Рисунок 35), у інших вираз буде сформований відповідно до виділеного фрагменту коду.

У вікні *Add Watch* буде зазначений контрольований вираз (його можна вписати/відредагувати) і можна обрати тип контрольованого виразу:

- саме контрольований вираз (*Watch Expression*) – дозволяє спостерігати значення контрольованого виразу при покроковому проходженні програми (можна їх змінювати у стовпці *Value* вікна *Watch*);

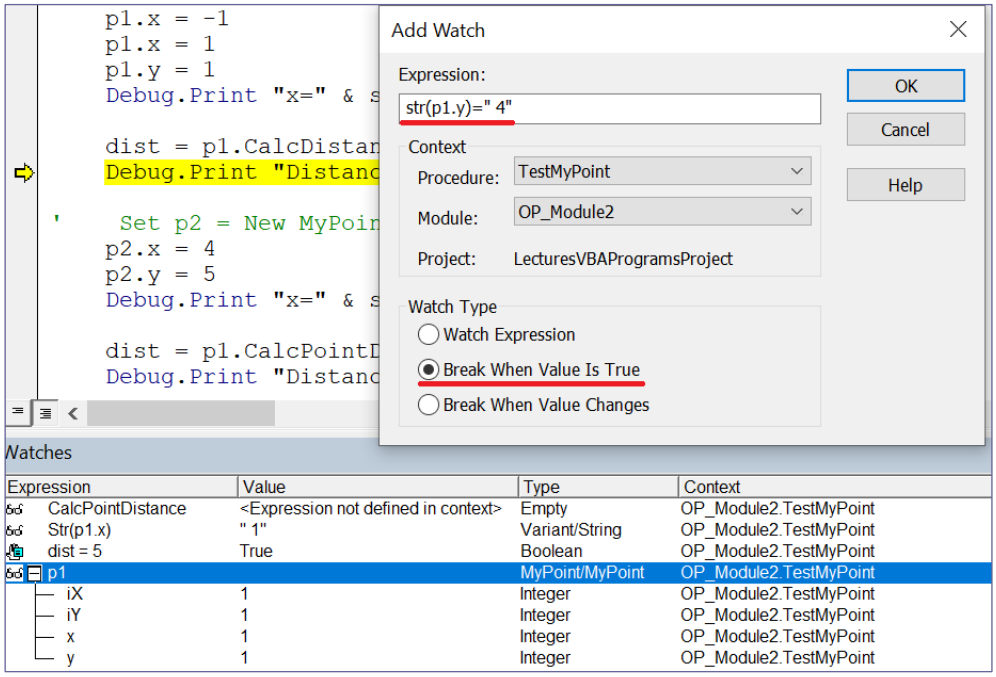


Рисунок 35 – Додання змінних/властивостей/виразів у вікно Watch

- автоматично переводити програму в режим паузи, якщо контрольований вираз повертає значення `True` (*Break When Value is True*) – тобто при виконанні програми значення змінної/властивості об'єкта/виразу стане саме таким, яке зазначено у контрольованому виразі;

- автоматично переводити програму в режим паузи, якщо значення контрольованого виразу змінилося (*Break When Value Changes*).

Перехоплення і обробка помилок часу виконання. Одна з найбільш трудомістких частин при створенні програми – передбачити, які помилки часу виконання можуть виникнути при роботі користувача і реалізувати їх обробку.

Розглянемо як приклад підпрограму, яка виконує ділення двох чисел:

```
Sub RandomDenominator()
    Dim i As Integer, n As Integer

    i = 10
    n = Int(Rnd() * 2)
        Debug.Print i & "/" & n & "=" & i / n
End Sub
```

У цій програмі знаменник арифметичної операції в операторі `Debug.Print` обчислюється випадковим чином і може у деяких випадках приймати значення `0`. У такому разі при виконанні програми виникає помилка (Рисунок 36).

Для її перехоплення та обробки перед небезпечним кодом, у якому потенційно можуть виникнути помилки часу виконання, вставляють оператор **`On Error GoTo позначка_обробника_помилки`**, а далі у коді програми розміщується як позначка рядка коду, на який виконується перехід у разі виникнення помилки, **`позначка_обробника_помилки:`**, за якою слідує програмний код обробки помилки.

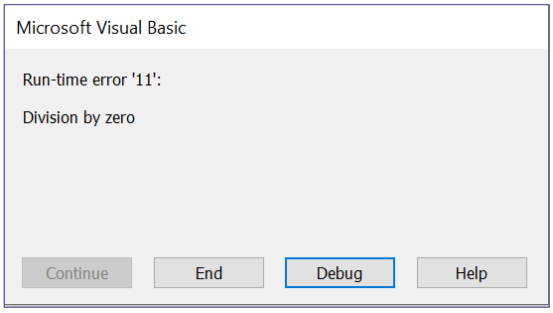


Рисунок 36 – Приклад помилки часу виконання

Наведемо приклад перехоплення та обробки помилки для підпрограми:

```
Sub RandomDenominator()
    Dim i As Integer, n As Integer

    i = 10
    n = Int(Rnd() * 2)
    On Error GoTo DivErrHandler
        Debug.Print i & "/" & n & "=" & i / n
        Debug.Print "Після потенційної помилки"
    Exit Sub
DivErrHandler:
    MsgBox "Ділення на n=" & n
End Sub
```

У разі ділення на 0 у операторі `Debug.Print i & "/" & n & "=" & i / n` виникне помилка і керування буде передано на позначку обробника помилки **`DivErrHandler:`**, який виведе опис помилки у діалогове вікно (Рисунок 37) і програма буде нормально завершена. Оператор **`Debug.Print "Після потенційної помилки"`** виконаний не буде.

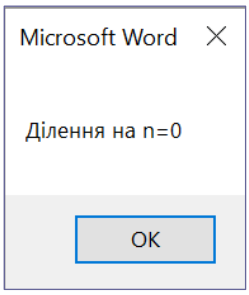


Рисунок 37 – Результат роботи обробника помилки

У разі відсутності помилки, після виведення результату ділення також буде виконаний оператор **`Debug.Print "Після потенційної помилки"`**, а оператор **`Exit Sub`** виконує завершення підпрограми, оминаючи код обробника помилки (у разі перехоплення та обробки помилки у функції або властивості, використовують аналогічні оператори `Exit Function` або `Exit Property`).

Часто після виконання коду обробника помилки необхідно або продовжити виконання тієї процедури, у якій виникла помилка, або припинити її виконання і передати керування процедурі, яка її викликала. Для цього існують такі можливості:

- ще раз виконати оператор, який викликав помилку (після усунення причини помилки у обробнику помилки) – для цього в обробник помилок необхідно вставити команду `Resume`, наприклад:

```
Sub RandomDenominator()
    Dim i As Integer, n As Integer

    i = 10
    n = Int(Rnd() * 2)
    On Error GoTo DivErrHandler
        Debug.Print i & "/" & n & "=" & i / n
        Debug.Print "Після потенційної помилки"
    Exit Sub
DivErrHandler:
    Debug.Print "Ділення на n=" & n
    n = Int(Rnd() * 2)
    Resume
End Sub
```

Тепер програма, у разі виникнення помилки буде *корегувати помилку*: у блоці обробника помилки після виведення інформації про помилку ще раз згенерується значення знаменника `n` і передасть керування на оператор `Debug.Print i & "/" & n & "=" & i / n`, у якому воно використовується, (у разі, якщо для знаменника знову буде згенероване значення `0`, обробка помилки буде повторена);

- пропустити оператор, який викликав помилку – для цього в обробник помилок необхідно вставити команду `Resume Next`, наприклад:

```
Sub RandomDenominator()
    Dim i As Integer, n As Integer

    i = 10
    n = Int(Rnd() * 2)
    On Error GoTo DivErrHandler
'    On Error Resume Next
        Debug.Print i & "/" & n & "=" & i / n
'    On Error GoTo 0
        Debug.Print "Після потенційної помилки"
    Exit Sub
DivErrHandler:
    Debug.Print "Ділення на n=" & n
    Resume Next
End Sub
```

Тепер програма, у разі виникнення помилки, буде *ігнорувати помилку з передачею* у блоці обробника помилки *керування на наступний* за помилковим *оператор* `Debug.Print "Після потенційної помилки"` (після виведення інформації про неї). Зауважимо, що така ж стратегія – ігнорування помилки – реалізується у разі застосування замість `On Error GoTo DivErrHandler` оператора `On Error Resume Next` (у наведеному прикладі цей оператор закоментований), але тепер *передача керування на блок обробника помилки*

взагалі не виконується, а керування передається на наступний за помилковим оператор. Зауважимо, що існує можливість у будь-якому місці коду відключити механізм перехоплення помилок – для цього використовують оператор `On Error GoTo 0` (він закоментований у коді) після потенційно небезпечного коду.

- продовжити виконання з певного місця в програмі – для цього в обробник помилок необхідно вставити команду `Resume позначка_переходу`, а у текст програми позначку `позначка_переходу`:

```
Sub RandomDenominator()
    Dim i As Integer, n As Integer

    i = 10
Start:
    n = Int(Rnd() * 2)
    On Error GoTo DivErrHandler
        Debug.Print i & "/" & n & "=" & i / n
'    On Error GoTo 0
        Debug.Print "Після потенційної помилки"
    Exit Sub
DivErrHandler:
    Debug.Print "Ділення на n=" & n
    Resume Start
End Sub
```

Тепер програма, у разі виникнення помилки, після виведення інформації про неї у блоці обробника помилки *передасть керування на оператор з позначкою_переходу* `Start` у сподіванні отримання відмінного від 0 значення знаменника (у разі, якщо для знаменника знову буде згенероване значення `0`, обробка помилки буде повторена).

Розглянуті вище підходи дозволяють організовувати обробку помилок *невибірково*. Але на практиці у одній процедурі часто можуть виникати помилки різного типу, які необхідно обробляти по-різному. Така можливість в VBA реалізується завдяки автоматично створюваному при виникненні помилки *об'єкту* `Err`, властивості якого вказують на тип помилки:

`Number` – номер помилки, номери найбільш поширених помилок вказані у [20], у разі, якщо номер помилки дорівнює `0`, помилок не було – ця властивість є властивістю об'єкта `Err` за замовчуванням, тому перевіряти його номер можна без зазначення властивості `Number`, наприклад, `If Err = 11 Then`;

`Description` – текстовий опис помилки – він відображається разом з номером у діалоговому вікні при виникненні помилки часу виконання, наприклад (Рисунок 36);

`Source` – містить ім'я об'єкта, який викликає помилку;

`HelpFile` та `HelpContext` – дозволяють зазначити ім'я та розділ файлу довідки, який вже повинен бути створений у форматі Microsoft Compiled HTML Help [19] (ці властивості, зазвичай, не використовують).

Наведемо модифікацію нашої програми-прикладу (змінимо її назву на `MultipleErrors()`), додавши до неї оператор, який генерує помилку запису за межі індексів динамічного масиву:

```
Sub MultipleErrors()
    Dim i As Integer, n As Integer
    Dim arr() As Integer

    i = 10
    n = Int(Rnd() * 2)
    On Error GoTo DivErrHandler
        Debug.Print i & "/" & n & "=" & i / n
        arr(0) = 5
    On Error GoTo 0
        Debug.Print "Після потенційної помилки"
    Exit Sub
DivErrHandler:
    Debug.Print "Помилка номер " & Err & ": " _
              & Err.Description & " в об'єкті: " _
              & Err.Source
End Sub
```

Тепер у програмі може виникнути або помилка ділення на нуль і керування буде передане до обробника помилки, або ділення буде успішним, але виникне помилка запису до масиву. В обох випадках повідомлення, згенероване обробником подій, надасть повну інформацію про помилку. При декількох запусках програми можна побачити у вікні *Immediate*, наприклад, таке виведення:

```
Помилка номер 11: Division by zero в об'єкті: VBAProgramsProject
10/1=10
Помилка номер 9: Subscript out of range в об'єкті: VBAProgramsProject
```

Але, якщо необхідно не просто проінформувати про помилку, а й обробити (скорегувати) її, код обробника помилки зміниться, наприклад, на такий:

```
DivErrHandler:
    Select Case Err
        Case 11
            Debug.Print "Ділення на 0"
            n = Int(Rnd() * 2)
            Resume
        Case 9
            Debug.Print "Запис за межі масиву"
            ReDim arr(3)
            Resume
    End Select
End Sub
```

Тепер при декількох запусках програми можна побачити у вікні *Immediate*, наприклад, таке виведення:

```
Ділення на 0
10/1=10
```

Запис за межі масиву
Після потенційної помилки

Як бачимо, помилки різного типу корегуються по різному.

На останок, опишемо ще одну технологію, пов'язану з помилками часу виконання, – *програмну генерацію помилки*. Ця технологія дозволяє у разі виникнення нештатної ситуації при виконанні програми, наприклад, закінченні пального у програмі моделювання руху автомобіля, отриманні повідомлення від користувача, яке не відповідає визначеному формату тощо, згенерувати власну, користувацьку помилку, зазначивши її властивостям необхідні значення. У разі, якщо код програми написаний з використанням операторів перехоплення та обробки помилок, така помилка буде оброблятися, і код її обробника можна запрограмувати необхідним чином.

Для програмного генерування помилки використовують метод `Raise()` об'єкта `Err`. Синтаксис цього методу наступний:

```
Err.Raise Number [, Source][, Description][, HelpFile]
                 [, HelpContext]
```

Призначення параметрів описане для об'єкту `Err` (див. вище). Єдиний обов’язковий параметр `Number` містить номер помилки у діапазоні від `0` до `65535`, для користувацьких помилок зарезервовані номери `513-65535`. Практика, яка рекомендується для зазначення номеру помилки, – додання номеру користувацької помилки до константи `vbObjectError`. Наприклад, щоб згенерувати номер помилки `513`, призначте параметру `Number` значення `vbObjectError + 513`.

Наведемо програму, яка генерує користувацьку помилку з номером `513` у разі, якщо користувач вводить рік народження зі значенням за межами дозволеного діапазону:

```
Sub UserError()
    Dim birthYear As Integer, birthYearDate As Date

Start:
    On Error GoTo MsgErrHandler
        birthYear = InputBox("Введіть свій рік народження")
        birthYearDate = DateSerial(birthYear, 1, 1)
        If DateDiff("yyyy", birthYearDate, Now) > 120 _
            Or DateDiff("yyyy", birthYearDate, Now) < 1 Then
            Err.Raise Number:=vbObjectError + 513, _
               Description:="Значення року народження хибне!"
        End If
        MsgBox "Введені дані правильні"
    Exit Sub
MsgErrHandler:
    MsgBox Err.Description & vbCr & _
          "Ви повинні ввести число у діапазоні від " & _
           year(Now) - 120 & " до " & year(Now) - 1
    Resume Start
End Sub
```

Зазначимо, що генерування помилки може виконуватись у довільній частині процедури/функції після оператора `On Error GoTo позначка_обробника_помилки`.

У разі введення року народження, який менше поточного більше, аніж на 120 років, або менше, аніж на 1 рік, буде згенерована користувацька помилка, яка буде перехоплена і керування буде передане обробнику помилки, який виведе відповідну інформацію, включаючи властивість `Description` об'єкта користувацької помилки.

Окрім метода `Raise()`, об'єкт `Err` має метод `Clear()`, який можна використовувати для примусового очищення властивостей об'єкта помилки (наприклад, при його повторному використанні) [21]. Наведемо приклад програми з потенційною помилкою переповнення значення типу та з так званою *відкладеною обробкою помилки* (*deferred error handling*), що організується застосуванням оператора `On Error Resume Next` з подальшою перевіркою об'єкта `Err` [22]:

```
Sub ErrClear()
    Dim arr(10) As Integer
    Dim i As Integer

    On Error Resume Next     'Відкладена обробка помилки
        For i = 0 To 5

            'Можливе переповнення Integer
            arr(i) = Int(Rnd * 50000 + 1)
            Debug.Print i; arr(i)
            If Err.Number <> 0 Then
                Debug.Print "Помилка: " & Err & " - " _
                            & Err.Description

'                Err.Clear    'Очищує властивості Err
'                Debug.Print "Після очищення: " & Err _
                          & " - " & Err.Description
            End If
        Next i
End Sub
```

При запуску програми у разі закоментованого оператора очищення властивостей об'єкта `Err`, можемо отримати, наприклад:

```
 0  20639
 1  0
Помилка: 6 - Overflow
 2  16311
Помилка: 6 - Overflow
 3  31659
Помилка: 6 - Overflow
...
```

При значенні ітератора `i=1` згенероване число перебільшує діапазон `Integer` і виникає помилка, але на наступних ітераціях згенероване число відповідає значенням `Integer`, тим не менш виведення помилки залишається.

Для правильної роботи програми необхідно розкоментувати оператор **`Err.Clear`** (а також наступний за ним оператор `Debug.Print,` що покаже значення властивостей об'єкта `Err` після його очищення):

```
 0  29168
 1  4036
 2  22899
 3  0
Помилка: 6 - Overflow
Після очищення: 0 -
 4  13069
 5  0
Помилка: 6 - Overflow
Після очищення: 0 -
```

Як бачимо VBA надає досить розвинені можливості для перехоплення та обробки помилок часу виконання. Зазначимо, що професійні програми активно їх використовують.

Питання для самоконтролю

1. Опишіть відмінності структури програм VBA, написаних у процедурно-орієнтованій парадигмі, від програм об'єктно-орієнтованої парадигми.
2. Які елементи коду може містити модуль класу?
3. Назвіть відмінності між класом та об'єктом класу.
4. Що являє собою властивість VBA?
5. Опишіть оператори, які використовуються для створення об'єктів класу. Які способи існують для видалення об'єкта з пам'яті?
6. Назвіть події класу VBA, які можна використовувати для ініціалізації полів об'єкту класу при його створенні та для виконання дій перед видаленням об'єкта з пам'яті.
7. Опишіть як працює і для чого використовується технологія інкапсуляції полів класу. Як виглядає сигнатури геттерів та сеттерів у реалізації VBA?
8. Що називають методом класу VBA? Наведіть приклад оголошення методу класу.
9. Чи підтримує VBA технології успадкування та поліморфізму – перевантаження методів?
10. Що являє собою подія, коли вона виникає?
11. У якому розділі класу оголошується подія? Який оператор для цього використовується? Назвіть оператор, який генерує подію і можливі місця його розміщення.
12. Назвіть типи модулів, у яких можна оголошувати, генерувати та обробляти події. Яким оператором оголошується об'єкт класу, у якому може бути згенерованою подія? Як додати до коду обробник події?
13. Опишіть три типи помилок, які можуть виникати у програмі. Які засоби для їх знаходження та виправлення надає Редактор Visual Basic?
14. Які помилки знаходять відлагодженням? Опишіть послідовність відлагодження програми та команди і засоби Редактора Visual Basic, що дозволяють його виконувати.

15. Як можна перевести програму у відлагоджувачі у режим паузи? Які дії можна виконувати у цьому режимі? Назвіть відповідні команди.
16. Опишіть призначення вікна *Immediate* Редактору Visual Basic. Які операції можна у ньому виконувати? Як його відкрити?
17. Опишіть призначення вікна *Locals* Редактору Visual Basic. Що виводиться у цьому вікні і що можна у ньому робити?
18. Опишіть призначення вікна *Watch* Редактору Visual Basic. Що називають контрольованим виразом? Як можна його створити? Які типи контрольованих виразів можна створювати?
19. Що являють собою помилки часу виконання? Наведіть приклади таких помилок.
20. Яким оператором можна організувати спостереження за виникненням помилок часу виконання? Як він реагує на виникнення помилки?
21. Які можливі варіанти повернення до процедури з блоку обробника помилок часу виконання?
22. Як можна організувати вибіркову обробку помилок часу виконання? Назвіть властивості об'єкту `Err`.
23. Опишіть технологію генерування та обробки користувацької помилки та оператори, які при цьому використовуються. Для чого генерувати користувацькі помилки? Які номери доступі для користувацьких помилок?
24. Для чого можна використовувати метод `Clear()` об'єкту `Err`?

Тестові завдання для перевірки знань

1. Оберіть, що може входити до структуру класи VBA (виберіть одну або декілька відповідей):
   а) інтерфейс;
   б) подія;
   в) метод;
   г) перерахування;
   д) поле;
   е) властивість.
2. Оберіть, які з технологій об'єктно-орієнтованого програмування підтримуються VBA (виберіть одну або декілька відповідей):
   а) перевантаження конструкторів;
   б) перевантаження методів;
   в) успадкування класів;
   г) інкапсуляція полів;
   д) ініціалізація полів у конструкторі (або його аналогу).
3. Оберіть оператор видалення об'єкту з ім'ям Obj1:
   а) `Obj1 = Null`;
   б) `Obj1 = Nothing`;
   в) `Set Obj1 = NULL`;
   г) `Obj1 = NILL`.
4. Оберіть призначення обробника події `Class_Initialize()` модулю класу:
   а) оголошує типи полів класу;

б) резервує пам'ять під об'єкт класу;
в) успадкування класів;
г) створює об'єкт класу;
д) присвоює вказанні значення полям об'єкту класу.

5. Оберіть призначення обробника події `Class_Terminate()` модулю класу:
а) автоматично виконується при видаленні об'єкту з пам'яті;
б) видаляє об'єкт з пам'яті;
в) присвоює значення за замовчуванням полям об'єкту;
г) присвоює значення аргументів полям об'єкту.

6. Оберіть оператор, який програмно генерує подію в VBA:
а) `ThrowEvent ім'я_події`;
б) `RaiseEvent ім'я_події`;
в) `FireEvent ім'я_події`.

7. Оберіть де можна оголошувати екземпляр класу з подіями (виберіть одну або декілька відповідей):
а) у процедурі звичайного модуля;
б) в області декларацій модуля класу;
в) в області декларацій модуля форми;
г) в області декларацій звичайного модуля;
д) у процедурі модуля класу.

8. Оберіть типи контрольованих виразів, які можна конфігурувати у вікні Watch Редактору Visual Basic (виберіть одну або декілька відповідей):
а) контрольований вираз, який дозволяє спостерігати значення визначеного оператора або його частини при покроковому проходженні програми;
б) контрольований вираз, який дозволяє переводити програму у режим паузи, якщо контрольований вираз повертає значення True;
в) контрольований вираз, який дозволяє виходити з підпрограми при заданому значенні;
г) контрольований вираз, який дозволяє переводити програму у режим паузи, якщо його значення змінилося;
д) контрольований вираз, який дозволяє змінювати значення змінної на вказане.

9. Оберіть оператор, який програмно генерує помилку в VBA:
а) `Err.Fire number`;
б) `Err.Throw number`;
в) `Err.Raise number`.

10. Оберіть оператор, який дозволяє організувати відкладену обробку помилки часу виконання:
а) `On Error GoTo`;
б) `On Error Resume Next`;
в) `On Error Resume`;
г) `On Error GoTo 0`;
д) `On Error GoTo Next`.

Завдання для самостійного виконання

1. Розробіть на мові VBA клас, який представляє вектор на площині, заданий координатами початку і кінця вектору, з методами знаходження модуля вектору і геометричної суми двох векторів. Напишіть код процедури, у якому створіть декілька об'єктів розробленого Вами класу із різними значеннями та виконайте методи об'єктів, задавши довільні значення аргументів.
2. Розробіть на мові VBA клас, який представляє довільний трикутник, заданий координатами його вершин на площині, з методами розрахунку його периметра і площі. Напишіть код процедури, у якому створіть декілька об'єктів розробленого Вами класу із різними значеннями та виконайте методи об'єктів, задавши довільні значення аргументів.
3. Розробіть на мові VBA клас, який представляє дійсне число, з методами розрахунку суми, різниці, добутку і частки двох чисел. Напишіть код процедури, у якому створіть декілька об'єктів розробленого Вами класу із різними значеннями та виконайте методи об'єктів, задавши довільні значення аргументів.
4. Розробіть на мові VBA клас, який представляє табурет, з методом, що виводить на екран його розміри і матеріал, з якого він виготовлений, а також з методом, що порівнює вагу двох табуретів з виведенням на екран інформації, який з них важче. Напишіть код процедури, у якому створіть декілька об'єктів розробленого Вами класу із різними значеннями та виконайте методи об'єктів, задавши довільні значення аргументів.
5. Розробіть на мові VBA клас, який представляє персону, з методом виведення його прізвища і методом розрахунку кількості прожитих повних років, місяців і днів. Напишіть код процедури, у якому створіть декілька об'єктів розробленого Вами класу із різними значеннями та виконайте методи об'єктів, задавши довільні значення аргументів.
6. Розробіть на мові VBA клас, який представляє матрицю з ініціалізацією її елементів випадковими цілими числами та методами знаходження суми та різниці двох матриць. Напишіть код процедури, у якому створіть декілька об'єктів розробленого Вами класу із різними значеннями та виконайте методи об'єктів, задавши довільні значення аргументів.
7. Розробіть на мові VBA клас, який представляє многочлен від однієї змінної, що задається ступенем многочлена і масивом коефіцієнтів, з методами обчислення значення многочлена для заданого аргументу, знаходження суми та різниці многочленів з отриманням нового об'єкта-многочлена. Напишіть код процедури, у якому створіть декілька об'єктів розробленого Вами класу із різними значеннями та виконайте методи об'єктів, задавши довільні значення аргументів.
8. Розробіть на мові VBA клас, який представляє квадрат з методами розрахунку його периметра і площі. Додайте до розробленого класу подію інформування про результат розрахунку, а у методи класу – її генерування. Розробіть клас-огортку для об'єкту класу квадрату та реалізуйте у ньому обробник події. Продемонструйте виникнення та обробку події.

9. Розробіть на мові VBA клас, який представляє коло, з методами розрахунку його периметра і площі. Додайте до розробленого класу подію інформування про результат розрахунку, а у методи класу – її генерування. Розробіть клас-огортку для об'єкту класу кола та реалізуйте у ньому обробник події. Продемонструйте виникнення та обробку події.
10. Розробіть на мові VBA клас, який представляє прямокутник, з методами розрахунку його периметра і площі. Додайте до розробленого класу подію інформування про результат розрахунку, а у методи класу – її генерування. Розробіть клас-огортку для об'єкту класу кола та реалізуйте у ньому обробник події. Продемонструйте виникнення та обробку події.

ТЕМА 4 ЗАСОБИ VBA ДЛЯ РОБОТИ З КОРИСТУВАЦЬКИМИ ФОРМАМИ

Мета: вивчити призначення та способи створення користувацьких форм, додання до них та конфігурування стандартних та додаткових елементів керування та написання програм обробників подій таких елементів.

План

1. Користувацькі форми та їх призначення.
2. Створення форм, їх властивості, методи та події.
3. Стандартні елементи керування, їх властивості, методи та події.
4. Додаткові елементи керування, їх властивості, методи та події.
5. Програмне додання елементів керування та налаштування обробників подій.

Основні терміни і поняття

Графічний інтерфейс користувача, користувацька форма, елемент керування, властивість елемента керування, метод елемента керування, обробник події елемента керування.

Основні теоретичні положення

4.1 Користувацькі форми та їх призначення

Користувацька форма – являє собою *графічний інтерфейс користувача* (*Graphic User Interface – GUI*) у вигляді стандартної екранної форми з розташованими на ній *елементами керування* (*Controls*), такими як *Написи, Текстові поля, Списки, Кнопки* тощо. Наявність в Редакторі Visual Basic засобів проєктування користувацьких форм дозволяє створювати більш ефективні інструменти для введення даних користувачем та виведення результатів роботи програми, аніж застосування вбудованих функцій `MsgBox()` та `InputBox()` [3].

Робота з документами застосунків Microsoft Office, у які засобами VBA додані користувацькі форми, зазвичай, передбачає автоматичний запуск форми на початку роботи з документом/робочою книгою тощо для забезпечення введення різноманітної інформації користувачем: заповнення текстових полів документу, обрання тієї або іншої опції, зазначення дати та введення необхідних числових даних тощо. Після цього виконується клік по одній з кнопок форми, що ініціює відповідний метод-обробник події, у якому прописується код, який використовує введені дані.

Необхідно зазначити, що елементи керування, які додаються до форми, можуть бути додані і до документу/робочої книги тощо безпосередньо – для цього використовуються кнопки секції *Елементи керування* на стрічці меню *Розробник* (Рисунок 38). Після їх додання можливо створення процедур VBA у модулі, пов'язаному з документом/робочою книгою тощо (Рисунок 39). Враховуючи, що прийоми роботи з таким елементами керування та елементами керування форм однакові, ми зосередимось на розгляді особливостей застосування останніх.

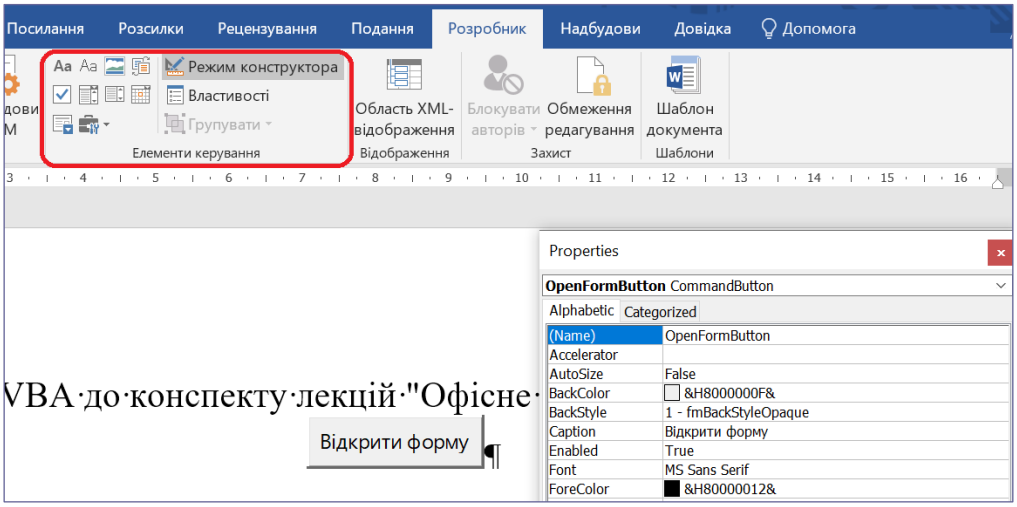


Рисунок 38 – Додання елементів керування безпосередньо до документу

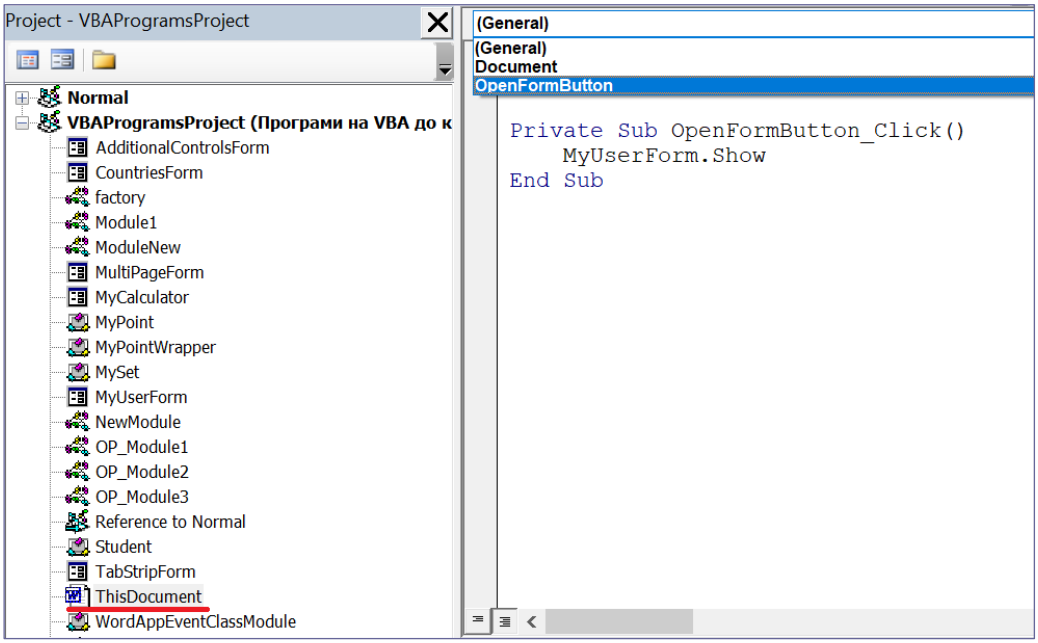


Рисунок 39 – Модуль коду, пов'язаний з документом

4.2 Створення форм, їх властивості, методи та події

Для створення форми необхідно у Редакторі Visual Basic виконати команду *Insert–UserForm*. Відкриється вікно візуального проєктування форми та панель *Toolbox* із вкладкою *Controls* з елементами керування (Рисунок 40). З цією формою пов'язаний модуль, у якому можна створювати процедури, які, зазвичай виконують обробку подій форми та її елементів керування. Перехід до модуля форми виконується командою меню *View–Code* або натисканням клавіші *F7*, повернення до вікна проєктування форми виконується командою меню *View–Object* або натисканням клавіатурної комбінації *Shif+F7*. Запускати форму можна кліком по кнопці *Run Sub/UserForm*  на стандартній панелі інструменті або на панелі інструментів відлагодження Редактора Visual Basic або натисканням клавіші *F5*.

Зазначимо *найважливіші властивості форм*:

- `Name` – визначає ім'я форми, що використовується у коді форми, рекомендується надавати змістовні імена формам, які відображають їх призначення, також рекомендується використовувати в іменах суфікс `Form`, наприклад, `PersonDataForm`. Зазначимо, що якщо у коді необхідно посилатися на поточну форму, використовують посилання `Me`, наприклад:

```
For Each cBox In Me.Controls ...
```

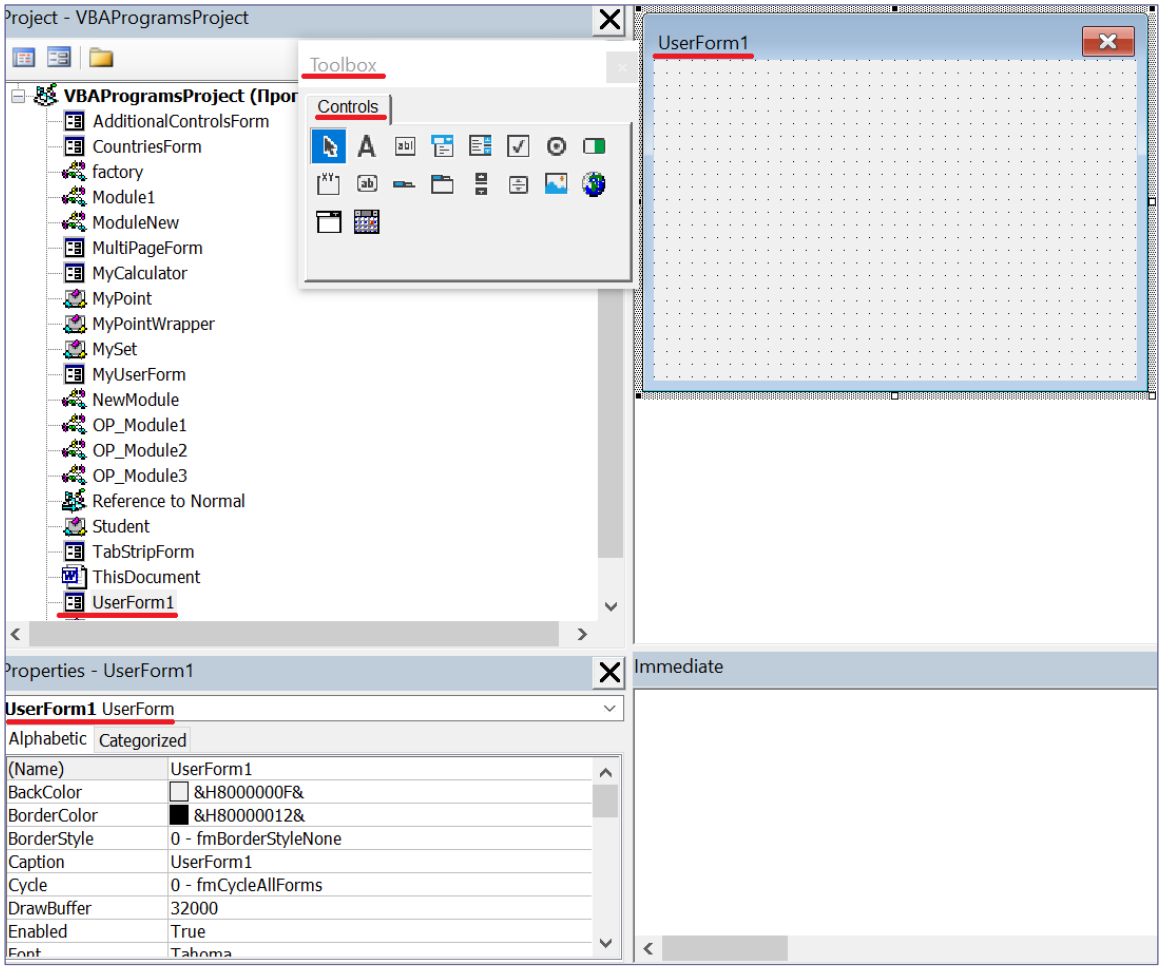


Рисунок 40 – Вікно форми з панеллю з елементами керування

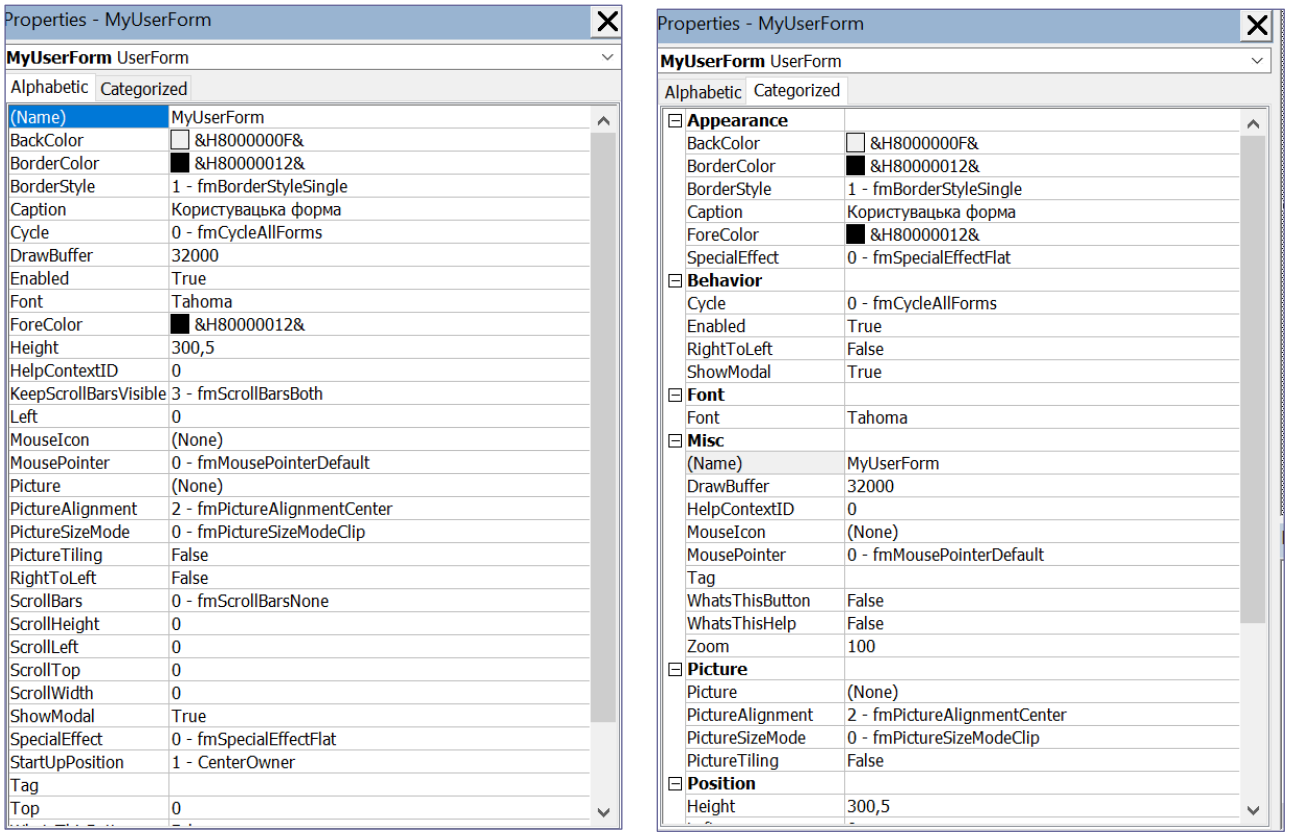
- `Caption` – визначає рядок-заголовок форми, рекомендується визначати його мовою користувачів програми, наприклад, "Персональні дані користувачів";
- `Enabled` – визначає можливість працювати з формою: при виставленні цієї властивості в значення `False`, користувач працювати з формою не зможе;
- `Height`, `Width` – визначають висоту та ширину форми у твіпах (див. вище);
- `Left`, `Top` – визначають зміщення форми від лівої та верхньої границі екрану у твіпах (див. вище);

Зазначені властивості мають і елементи керування, які будуть розглянуті далі у цьому розділі (як і властивості пов'язані з кольорами тощо).

– `ShowModal` – визначає чи буде вікно форми модальним (за замовчуванням) – коли користувач не зможе залишити форму, доки не закриє її.

- `ScrollBars` – дозволяє відображати смуги прокрутки для форми, у разі якщо вона не вміщується у задані розміри – має чотири значення:
  - `fmScrollBarsNone` – не відображає смуги прокрутки;
  - `fmScrollBarsHorizontal` – відображає горизонтальну смугу прокрутки;
  - `fmScrollBarsVertical` – відображає вертикальну смугу прокрутки;
  - `fmScrollBarsBoth` – відображає горизонтальну та вертикальну смугу прокрутки.

Редактор Visual Basic надає зручне *вікно властивостей* (*Properties*) для встановлення значень властивостей форми та виділених у ній елементів керування, у якому властивості зазначені в алфавітному порядку та згруповані за категоріями (Рисунок 41). Для відображення цього вікна необхідно виконати команду меню *View–Properties Window* або натиснути клавішу *F4*.



а) *б)*

Рисунок 41 – Вікно властивостей з а) алфавітним та б) за категоріями поданням

До *найважливіших методів форм* можна віднести методи:

- `Show()` – відображає форму на екрані – якщо форма вже була завантажена у пам'ять, вона стане видимою, якщо ні – то вона буде автоматично завантажена:

```
UserForm1.Show
```

Можливо при відображенні форми вказати її режим роботи, зазначивши як аргумент константу `vbModal` – для модального режиму (за замовчуванням) або `vbModeless` – для немодального, наприклад:

```
MyUserForm.Show vbModeless
```

Способів відкриття форми багато: вона може бути відкрита з процедур модулів, у тому числі обробників подій елементів керування, з користувацьких макросів, з макросів, що автоматично запускаються, наприклад, `AutoExec` або `AutoOpen` для Microsoft Word тощо;

- `Hide()` – прибирає форму з екрану, залишаючи її у пам'яті для забезпечення можливості повторного відображення:

```
MyUserForm.Hide
```

- `Unload()` – прибирає форму з екрану та видаляє її з пам'яті:

```
MyUserForm.Unload
```

Зазначимо, що при закритті документу, що містить модуль форми, форма видаляється з екрану та пам'яті автоматично.

- `Copy()`, `Cut()`, `Paste()` – дозволяють копіювати, вирізати форму до Буферу обміну та вставляти її звідти, відповідно – можуть використовуватися, наприклад, для дублювання вже створеної форми;

- `Move(Left As Single [, Top [, Width [, Height])` – дозволяє програмно переміщувати форму по екрану та/або змінювати її розміри;

- `PrintForm()` – виводить форму на друк;

- `Repaint()` – дозволяє оновити форму на екрані після програмної зміни елементів керування на ній;

- `Scroll([`*`ActionX`* `[,`*`ActionY`*`]])` – дозволяє програмно прокручувати форму, у разі якщо вона не вміщується у зазначені розміри `Height` та `Width`, параметрів визначають відстань прокрутки за відповідними осями [23].

- `UndoAction()`, `RedoAction()` – імітують виконання для форми команд *Скасувати ввід* та *Повернути ввід*, відповідно.

До *найважливіших подій форм* відносяться:

- `Initialize` – відбувається перед відкриттям форми, зазвичай в обробнику цієї події розміщують код ініціалізації елементів керування форми;

- `Activate`, `Deactivate` – відбувається після встановлення фокусу на формі, який робить її активною (зазвичай, клік миші по формі) та втраті фокусу (зазвичай, клік миші поза формою), відповідно;

- `Resize`, `Scroll`– відбуваються при зміні розмірів та прокрутці форми, відповідно;

- `Terminate` – відбувається перед завершенням роботи форми – зазвичай використовується для звільнення ресурсів, протоколювання тощо.

Зазначимо, що хоча форма має події `Click` (її обробник створюється автоматично) та `DblClick`, на практиці для форм вони використовуються не часто.

Наведемо приклад обробника події ініціалізації форми:

```
Private Sub UserForm_Initialize()
    Me.Caption = "Користувацька форма"
    Me.Width = 400
    Me.Height = 300
End Sub
```

4.3 Стандартні елементи керування, їх властивості, методи та події

Елементи керування – це об'єкти спеціальних класів VBA, що мають відтворювані у формі (або у файлах застосунків Microsoft Office) візуальні реалізації і у яких визначений певний перелік властивостей, методів та організоване генерування певних подій. Програми на VBA, зазвичай, містять код обробки необхідних для реалізації функцій програми подій, які ініціюються взаємодією користувача з елементами керування. Таким чином, елементи керування створюють *графічний інтерфейс користувача*, який полегшує йому введення даних та відображення результату роботи програми.

Редактор Visual Basic надає спеціальну панель інструментів *Toolbox*, з якої необхідні елементи керування перетягуються мишею на форму. Також він надає можливість візуального визначення розмірів елементів керування та їх розміщення у визначеній області форми.

Елемент керування Напис (*Label*) використовують для виведення необхідного тексту у певній області форми для пояснення інших елементів керування або виведення повідомлень користувачеві (Рисунок 42).

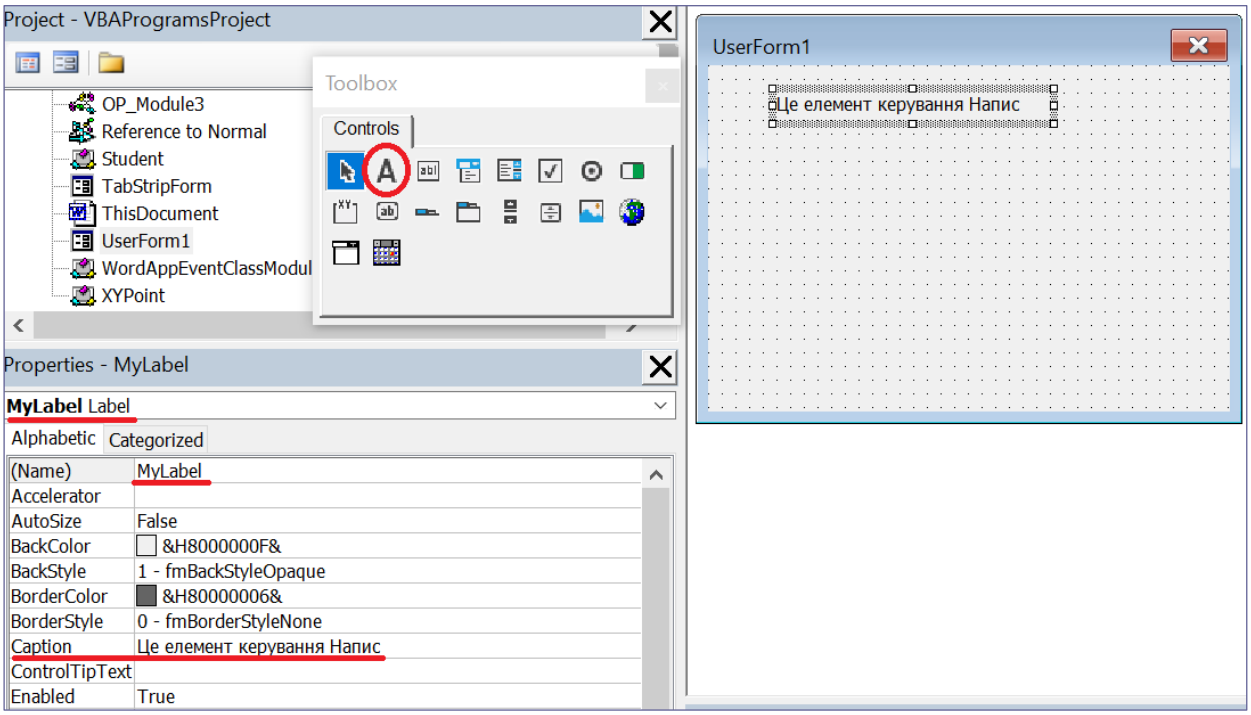


Рисунок 42 – Вікно форми з елементом керування Напис (Label)

Найважливішими властивостями цього елементу керування є:

`Name` – назва елементу, як може використовуватися у коді для посилання на цей елемент;

`Caption` – рядок, який буде виведений в елементі керування на формі.

Визначити `Caption` можна як у вікні *Properties* (Рисунок 42), так і програмно. У останньому випадку до обробника події ініціалізації форми можна додати оператор, наприклад, такий:

```
MyLabel.Caption = "Це елемент керування Напис (Label)"
```

Хоча *Напис* має декілька методів та подій, на практиці вони використовуються не часто.

Елемент керування Текстове поле (TextBox) використовується для введення даних користувачем, які, зазвичай, оброблюються у процедурах модуля форми (Рисунок 43).

До найважливіших властивостей *Текстового поля* можна віднести:

- `Value` – містить рядок, що вводить користувач до цього елементу керування, або рядок, який виводиться у цей елемент програмою. *Текстове поле* має також властивість `Text`, яка дублює властивість `Value`.
- `MaxLength` – містить значення максимальної кількості символів, які можуть бути введені у *Текстове поле*, значення `0` не обмежує кількість символів, що вводяться.
- `ControlTipText` – текст підказки, яка з'являється, коли користувач наводить покажчик миші на текстове поле.
- `MultiLine` – визначає, чи будуть перенесені символи, які вводяться на новий рядок, у разі, якщо вони не вміщуються по ширині поля (`True`) або рядок буде тільки один, а символи, а при вводі довгих рядків перші символи будуть ховатися за межі поля (`False`). У разі обрання значення `True`, необхідно встановити таке ж значення властивості `WordWrap`.
- `ScrollBars` – працює аналогічно однойменній властивості форми (див. вище).
- `PasswordChar` – дозволяє вказати символ, який буде виводитись у *Текстове поле* замість символів, що вводить користувач, зазвичай використовується для введення пароля.
- `Visible` – при виставленні значення цієї властивості у `False`, елемент керування відображатися на формі не буде;
- `Enabled` – при виставленні значення цієї властивості у `False`, текстове поле стане сірим і доступ до його вмісту буде заборонено: не можна буде вводити, виділяти, копіювати, вирізати, замінювати та видаляти символи.
- `Locked` – при виставленні значення цієї властивості у `True`, *Текстове поле* буде виглядати як зазвичай, користувач зможе виділяти і копіювати символи з нього, але не зможе вводити, вирізати, замінювати та видаляти їх.

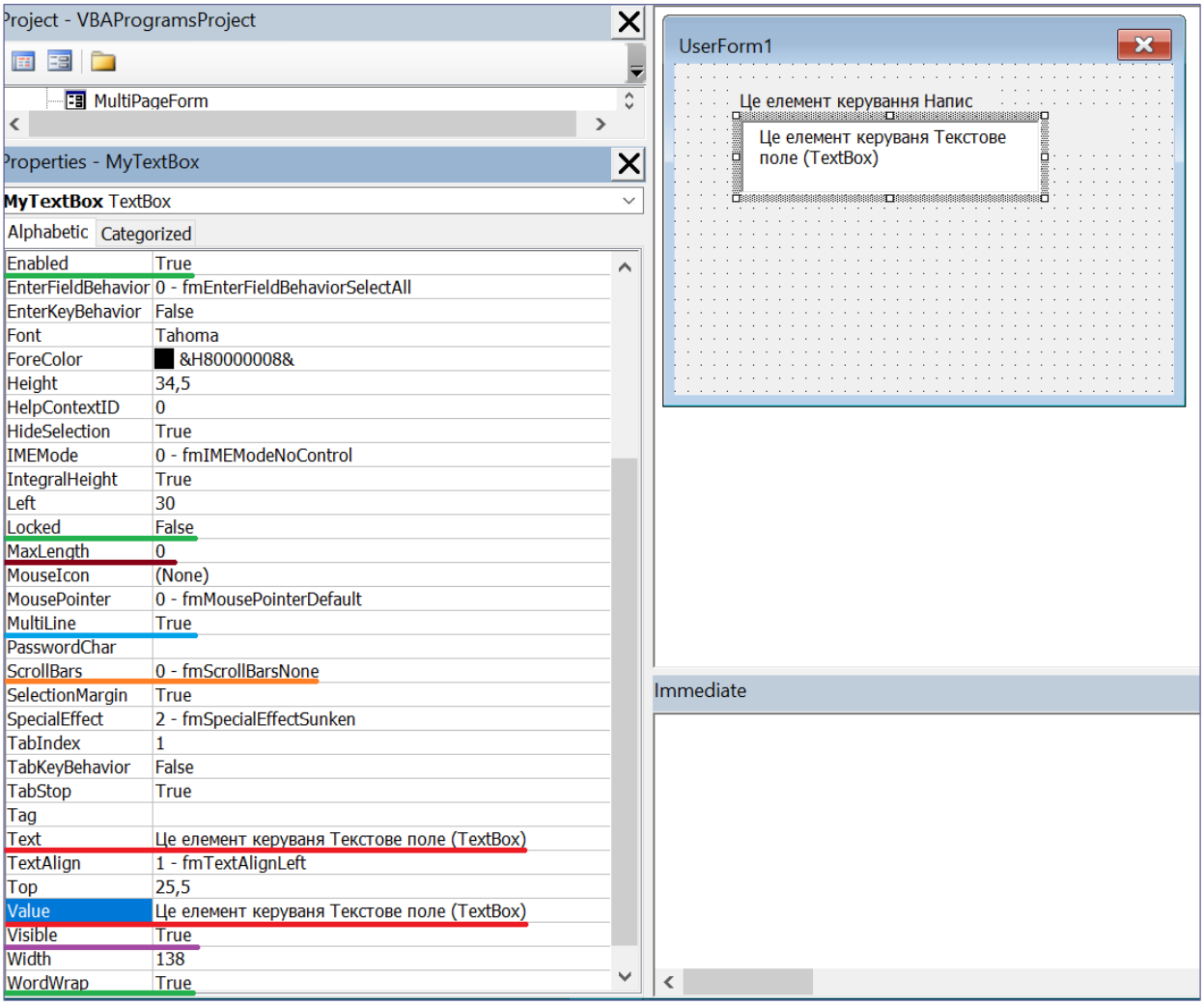


Рисунок 43 – Вікно форми з елементом керування Текстове поле (TextBox)

Методи, які має текстове поле включають методи роботи з Буфером Обміну: `Copy()`, `Cut()`, `Paste()` та ще декілька, які використовуються не часто.

До найважливіших подій *Текстового поля* можна віднести:

- `Change` – виникає кожного разу при зміні вмісту *Текстового поля*. Наведемо приклад обробника такої події, який підраховує та виводить у окремий елемент керування типу *Напис* `LblSymbCount` довжину рядка, що вводиться у текстове поле:

```
Private Sub MyTextBox_Change()
    LblSymbCount.Caption = Len(MyTextBox.Value)
End Sub
```

- `AfterUpdate` – виникає кожного разу, коли після введення або зміни тексту у *Текстовому полі* користувач натискає кнопку *Enter* або залишає *Текстове поле* (обирає інший елемент керування). Наступний приклад виводить текст, введений у *Текстове поле*, у діалоговому вікні:

```
Private Sub MyTextBox_AfterUpdate()
    MsgBox "У текстове поле введено: " & MyTextBox.Value
End Sub
```

Текстове поле має і інші події, але вони використовуються нечасто.

Елемент керування Кнопка (CommandButton) безумовно є найчастіше використовуваним елементом керування. Взаємодія з ним, зазвичай, запускає програму обробки введених користувачем даних у інші елементи керування, а також ініціює різноманітні дії, такі як закриття форми, скасування обробки даних тощо.

Найважливішими властивостями *Кнопки* є:

- `Caption` – текст, що буде виведений на *Кнопці*;
- `Picture` – посилання на Рисунок , що буде виведений на *Кнопці*;
- `Default` – у разі встановлення значення `True`, *Кнопка* буде автоматично натискатись при натисканні клавіші Enter, навіть коли активним буде інший елемент керування (окрім іншої *Кнопки*);
- `Cancel` – у разі встановлення значення `True`, *Кнопка* буде автоматично натискатись при натисканні клавіші Esc;
- `TakeFocusOnClick` – визначає, чи буде передаватися фокус цій *Кнопці* при клікі на ній, за замовчуванням `True`.

Елемент керування *Кнопка* не має жодного метода, але має події, серед яких головною є:

`Click` – запускає при одинарному клікі на *Кнопці* виконання коду, що реалізує функції програми або ініціює виконання таких дій, як закриття форми, відміна від дії тощо;

Наведений код простого калькулятора, побудованого за допомогою користувацької форми та розглянутих вище елементів керування. Інтерфейс програми має вигляд (Рисунок 44):

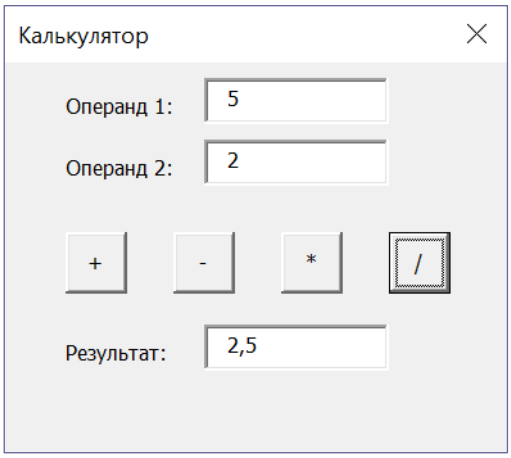


Рисунок 44 – Інтерфейс програми Калькулятор з елементами керування Напис, Текстове поле та Кнопка

Для доданих до форми елементів керування обрані такі імена:

`Op1Label`, `Op2Label` та `ResLabel` – для *Написів* першого, другого операнду та результату, відповідно;

`Op1TextField`, `Op2TextField` та `ResTextField` – для *Текстових полів* першого, другого операнду та результату, відповідно;

`AddButton`, `SubButton`, `MulButton` та `DivButton` – для *Кнопок*, які ініціюють операції додання, віднімання, множення та ділення, відповідно.

Зазначимо, що для `ResTextField` властивість `Locked` встановлена у `True` для попередження можливості видалення або зміни результату.

Найпростіша реалізація функціоналу програми передбачає створення обробників подій Click для чотирьох кнопок, у якій будуть виконуватися відповідні операції над значеннями, отриманими від текстових полів, та записуватися результат у поле `ResTextField`, наприклад для додання:

```
Dim op1 As Double, op2 As Double, res As Double

Private Sub AddButton_Click()
    op1 = Op1TextField.Value
    op2 = Op2TextField.Value
    res = op1 + op2
    ResTextField.Value = res
End Sub
```

При присвоюванні значень-рядків *Текстових полів* до змінних типу `Double` виконується автоматичне перетворення рядків до числових значень. Аналогічно, при присвоюванні результату операції у форматі `Double` як значення властивості `Value` *Текстового поля* результату, виконується автоматичне перетворення числового значення до рядка.

Але, якщо користувач введе у *Текстове поле* замість цифри літеру або введе нуль як значення другого оператора та вибере операцію ділення, програма згенерує помилку часу виконання ("Type mismatch" – код 13 або "Division by zero" – код 11). Тому необхідно додати перехоплення та обробку можливих помилок. Для того, щоб не дублювати код, пов'язаний з перехопленням та обробкою помилок, винесемо операції калькулятора у окрему підпрограму `Calculate()`, яка буде приймати як параметр посилання на *Кнопку*, що була використана для ініціювання операції (також перенесемо до цієї підпрограми оголошення змінних з області декларацій):

```
Private Sub Calculate(btn As CommandButton)
    Dim op1 As Double, op2 As Double, res As Double

    On Error GoTo CalcErrHandler
        op1 = Op1TextField.Value
        op2 = Op2TextField.Value
        Select Case btn.Caption
            Case "+"
                res = op1 + op2
            Case "-"
                res = op1 - op2
            Case "*"
                res = op1 * op2
            Case "/"
                res = op1 / op2
        End Select
        ResTextField.Value = res
```

```
    Exit Sub
CalcErrHandler:
    Select Case Err
        Case 11
            MsgBox "При діленні Операнд 2 не може мати " _
                & "значення 0"
        Case 13
            MsgBox "Операнди повинні бути числами"
    End Select
End Sub
```

Тепер обробники події клік від кнопок будуть викликати цю підпрограму передаючи їм посилання на відповідну кнопку:

```
Private Sub AddButton_Click()
    Calculate AddButton
End Sub
```

Тепер при помилкових даних, введених користувачем програма буде відображати діалогові вікна з відповідними повідомленнями.

Елемент керування Список (ListBox) – використовується для забезпечення обрання одного або декількох елементів серед заздалегідь визначеної множини елементів. *Список* має перевагу перед *Текстовим полем* у тому, що користувач не може зробити помилку у назві елемента, обираючи його зі *Списку,* порівняно з введенням аналогічного рядка у *Текстове поле*. Зазвичай *Список* використовується як проміжний елемент керування для обрання користувачем одного або декількох значень разом з, наприклад, *Кнопкою*, яка ініціює обробник події, у якому обрані елементи *Списку* використовуються.

Елементи *Списку,* зазвичай, визначають у обробнику події `Initialize` форми, у якій *Список* розташований, наприклад, так:

```
Private Sub UserForm_Initialize()
...
    MyListBox.AddItem "Запорізька область"          'ListIndex = 0
    MyListBox.AddItem "Дніпропетровська область"    'ListIndex = 1
    MyListBox.AddItem "Київська область"            'ListIndex = 2
    MyListBox.AddItem "Львівська область"           'ListIndex = 3
End Sub
```

При доданні елементів до списку вони автоматично отримують послідовний номер – `ListIndex`, який використовується для визначення поточного та обраних елементів *Списку*. Метод `AddItem()` має необов'язковий параметр `index`, яким можна визначати положення елемента у *Списку*.

Ще одним зручним способом ініціалізації елементів *Списку* є використання масиву – зазначимо, що *Список* може містити декілька стовпців. Наведемо приклад такої ініціалізації:

```
Private Sub UserForm_Initialize()
    Dim numbers(3, 1) As Integer, i As Integer, j As Integer
...
    For i = 0 To 3
        For j = 0 To 1
```

```
            If j = 0 Then
                numbers(i, j) = i
            Else
                numbers(i, j) = Int(Rnd * 10 + 1)
            End If
        Next j
    Next i

    MyListBox.ColumnCount = 2
    MyListBox.ColumnWidths = "1 cm; 2 cm"
    MyListBox.List = numbers
End Sub
```

У наведеній програмі створений двовимірний масив присвоюється властивості *Списку* `List` після визначення кількості колонок *Списку* за допомогою властивості `ColumnCount`, результуючий список показаний на Рисунок 45.

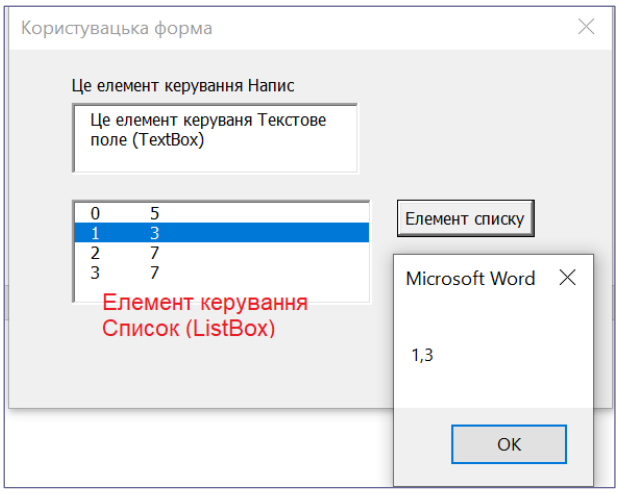


Рисунок 45 – Форма зі Списком, ініціалізованим двовимірним масивом

До основних властивостей *Списку* можна віднести:

- `List[index [, column]]` – дозволяє присвоїти *Списку* масив зі значеннями елементів (використовується без параметрів) або повертає елемент, зазначений параметрами: `index` – номер рядка, `column` – номер стовпця *Списку;*
- `ListCount` – повертає кількість рядків *Списку* (можливо тільки читати цю властивість);
- `ListIndex` – містить індекс (номер) обраного рядка елементів *Списку* (починається з `0`). До форми додана *Кнопка,* обробник події Click якої виводить у діалогове вікно значення виділеного рядка *Списку* (Рисунок 45):

```
Private Sub ListItemBtn_Click()
    Dim itemIndex As Integer

    itemIndex = MyListBox.ListIndex
    MsgBox MyListBox.List(itemIndex, 0) & "," _
                        & MyListBox.List(itemIndex, 1)
End Sub
```

Для списків у одну колонку виділений елемент *Списку* може бути також отриманий за допомогою властивостей `Value` та `Text`.

- `ColumnCount`, `ColumnWidth` – визначають кількість стовпців *Списку* та значення ширини кожного зі стовпців (див. вище);
- `MultiSelect` – визначає спосіб вибору елементів зі *Списку*:
  - дозволяє користувачеві вибирати тільки один елемент (`0 – frmMultiSelectSingle` – значення за замовчуванням);
  - дозволяє користувачеві вибирати кілька елементів, які можуть не знаходитися послідовно – виконується клікання по елементам з утриманням натиснутою клавіши Control (`1 – frmMultiSelectMulti`);
  - дозволяє користувачеві вибирати кілька послідовних елементів – виконується клікання по елементам з утриманням натиснутою клавіши Shift (`2 – frmMultiSelectExtended`).
- `Selected(Index)` – повертає `True`, якщо вказаний параметром `Index` елемент є обраним, ця властивість корисна при налаштуванні можливості обранні декількох елементів, наприклад для *Списку* на Рисунок 46;
- `ListStyle` – визначає спосіб обрання/відображення елементів списку:
  - значення `0 – frmListStylePlain` забезпечує відображення обраних елементів кольором;
  - значення `1 – frmListStyleOption` забезпечує додаткове розміщення елемента керування *Прапорець* (*CheckBox*) поруч з його значенням для можливості обрання елемента (Рисунок 46).

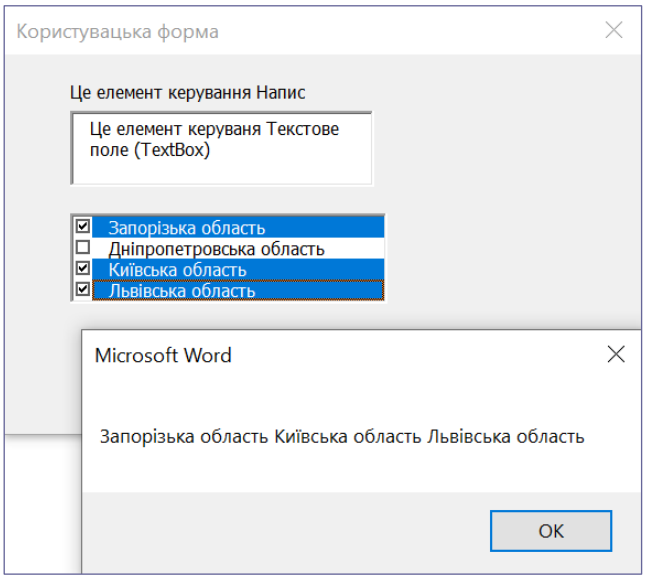


Рисунок 46 – Множинний вибір елементів Списку

Серед найважливіших методів *Списку,* окрім розглянутого `AddItem()` (див. вище), можна назвати:

`RemoveItem(index)` – видаляє зі *Списку* елемент з номером `index;`

`Clear` – видаляє зі *Списку* усі елементи.

Головною подією для *Списку*, як і для *TextBox*, є подія Change. Зазвичай, в обробнику цієї події перевіряються обрані користувачем елементи:

```
Private Sub MyListBox_Change()
    Dim myMsg As String
    Dim i As Long

    For i = 0 To MyListBox.ListCount - 1
        If MyListBox.Selected(i) Then
            myMsg = myMsg & MyListBox.List(i) & " "
        End If
    Next i

    MsgBox myMsg
End Sub
```

Результат роботи обробника події показаний на Рисунок 46.

Елемент керування Поле зі списком (ComboBox). Недоліком *Списку* є те, що він при наявності великої кількості елементів займає багато місця на формі (властивості `ScrollBars` він не має). Такий недолік усувається використанням *Поля зі списком*, який зазвичай відображає тільки один елемент (обраний), а інші з'являються як список після клікання на кнопці з зображенням трикутника (Рисунок 47). Також перевагою цього елементу керування є можливість додання елементів користувачем. Але, на відміну від звичайного *Списку*, *Поле зі списком* не підтримує множинний вибір елементів.

Ініціалізація *Поля зі списком* виконується методом `AddItem()` аналогічно ініціалізації звичайного *Списку* (див. вище).

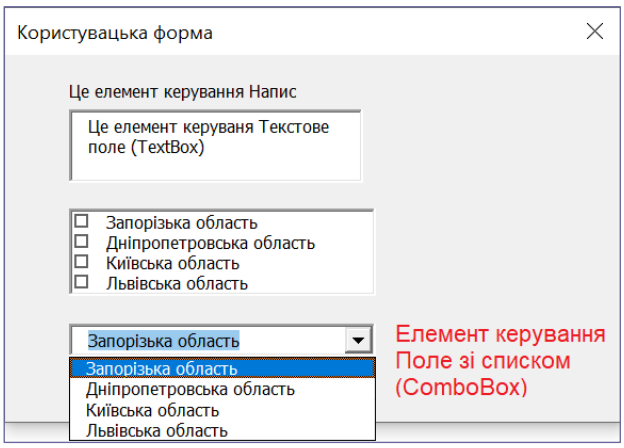


Рисунок 47 – Елемент керування Поле зі списком (ComboBox)

Властивості, методи та події *Поля зі списком* співпадають з такими для звичайного *Списку*.

Зазначимо тільки додаткову властивість `MatchRequired`, що визначає можливість введення користувачем нових елементів, що не співпадають з наявними: при значенні `False` така можливість доступна, а при значенні `True` вводити нові елементи не можна – необхідно обирати тільки існуючі. Наведемо приклад обробника події, що додає введений користувачем елемент до *Поля зі*

списком (для цього зручно використовувати обробник події AfterUpdate, яка генерується у випадках, як і для *Текстового поля* – див. вище):

```
Private Sub MyComboBox_AfterUpdate()
    MyComboBox.AddItem (MyComboBox.Value)
End Sub
```

Елементи керування Прапорець (CheckBox) та *Кнопка з фіксацією (ToggleButton)* використовуються, коли необхідно забезпечити вибір варіантів із деякої їх кількості, при цьому можливий вибір від одного до усіх варіантів одночасно. На відміну від *Списку*, який також може забезпечити таку функціональність, зазначені елементи керування використовуються, якщо кількість варіантів не є великою (Рисунок 48). Особливістю цих елементів є можливість використання трьох станів *Прапорця/Кнопки з фіксацією*: встановлений/включена, знятий/виключена та невизначений стан – відмітка/напис на кнопці відображаються сірим кольором (звичайно, можна використовувати такі елементи і тільки з двома станами – встановлений/включена, знятий/виключена).

До найважливіших властивостей зазначених елементів відносяться:

– `Caption` – напис праворуч від *Прапорця* або на *Кнопці з фіксацією*, який, зазвичай, пояснює, що вибирається;

– `TripleState` – якщо ця властивість встановлена у `False` (за замовчуванням), то *Прапорець/Кнопка з фіксацією* працюють тільки у двох станах: встановлений/включена, знятий/виключена, яким відповідають значення `True` та `False` властивості `Value`. Якщо ця властивість встановлена у `True`, то *Прапорець/Кнопка з фіксацією* працюють у трьох станах, значення властивості `Value` для невизначеного стану: `Null`. Таке значення може використовуватись, як початкове значення для варіантів (з невизначеним станом);

– `Value` – містить значення, що відповідає стану прапорця (див. пояснення властивості `TripleState`).

Методів у *Прапорця* або на *Кнопці з фіксацією* небагато і вони використовуються нечасто.

Найважливішою подією для *Прапорця* можна вважати подію Change, а для *Кнопки з фіксацією* – подію Click.

Наведемо приклад програми, що зчитує стан трьох прапорців та виводить назви встановлених прапорців у вікно *Immediate*, для прапорця зі встановленим у `True` режимом `TripleState`, може виводитись рядок `"Null"` (Рисунок 48).

```
Private Sub CheckCheckBoxes()
    Dim cBox As Control
    Dim msg As String

    For Each cBox In Me.Controls
        If TypeName(cBox) = "CheckBox" Then
            'Прапорець "Red" встановлений у Tripple State
            If IsNull(cBox) Then
                Msg = "Null "
            If cBox.Value = True Then
```

```
                msg = msg & cBox.Caption & " "
            End If
        End If
    Next
    Debug.Print msg
End Sub

Private Sub CheckBox1_Change()
    Call CheckCheckBoxes
End Sub

Private Sub CheckBox2_Change()
    Call CheckCheckBoxes
End Sub

Private Sub CheckBox3_Change()
    Call CheckCheckBoxes
End Sub
```

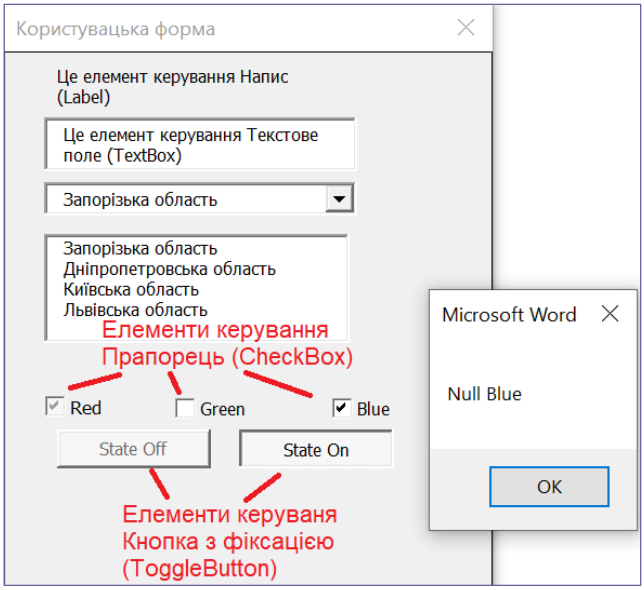


Рисунок 48 – Форма з елементами керування Прапорець (CheckBox) та Кнопка з фіксацією (ToggleButton)

У процедурі `CheckCheckBoxes()` в циклі виконується обхід усіх елементів керування форми - вони автоматично додаються до властивості-колекції `Controls` форми. За допомогою функції `TypeName(varname)`, яка повертає рядок-назву простих типів для змінних простих типів та рядок-назву об'єктних типів для змінних вбудованих у VBA об'єктних типів обираються для подальшої обробки тільки екземпляри *CheckBox*, для яких виводиться значення їх властивості `Caption`.

Кнопки з фіксацією рекомендується використовувати, коли їх перемикання ініціює перехід у якийсь довготривалий стан. Приклад програми, що зчитує стан *Кнопки з фіксацією* та виводить його у напису на кнопці (значення властивості `Caption`):

```
Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        ToggleButton1.Caption = "State On"
    Else
        ToggleButton1.Caption = "State Off"
    End If
End Sub
```

Елементи керування Перемикач (OptionButton) та *Рамка (Frame)*. На відміну від *Прапорців* та *Кнопок з фіксацією*, *Перемикач* використовують для обрання одного з варіантів, при цьому декілька *Перемикачів* об'єднують у групу за допомогою *Рамки* (Рисунок 49).

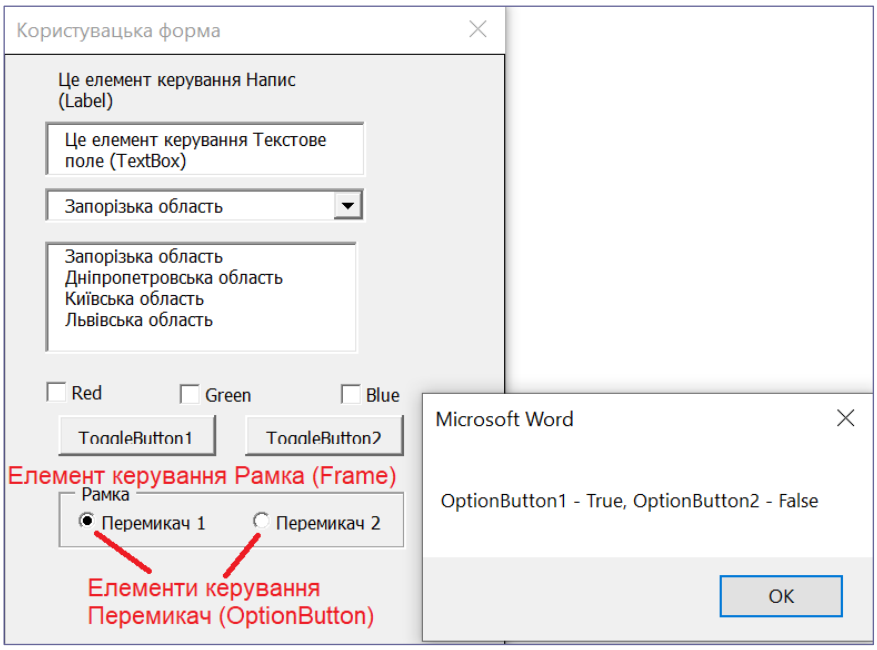


Рисунок 49 – Форма з елементами керування Перемикач (OptionButton) та Рамка (Frame)

Для *Перемикача* можна виділити дві найважливіші властивості:

- `Caption` – напис, пояснюючий варіант вибору;
- `Value` – значення, яке відповідає *Перемикачу* зі встановленою відміткою (`True`) та без такої (`False`).

Для *Рамки* властивості `Value` не існує.

Головною подією *Перемикача* є подія Click. Нижче наводиться код процедури, яка аналізує *Перемикачі*, об'єднані *Рамкою*, та виводить їх назву (значення властивості `Caption`) та стан (значення властивості `Value`) у діалогове при кожному кликанні мишею на любому з *Перемикачів* (Рисунок 49).

```
Private Sub OptionButtonsCheck()
    Dim i As Integer
    Dim msg(0 To 1) As String

    For i = 0 To Me.Frame1.Controls.Count - 1
        Msg(i) = Me.Frame1.Controls(i).Name & " - " _
               & Me.Frame1.Controls(i).Value
    Next i
```

```
    MsgBox msg(0) & ", " & msg(1)
End Sub

Private Sub OptionButton1_Click()
    Call OptionButtonsCheck
End Sub

Private Sub OptionButton2_Click()
    Call OptionButtonsCheck
End Sub
```

Елементи керування Смуга прокрутки (*ScrollBar*) та *Лічильник (SpinButton)*. Зі *Смугами прокрутки* ми вже стикалися при вивченні властивостей *Текстового поля:* його властивість `ScrollBars` дозволяє додавати або горизонтальну, або вертикальну, або обидві *Смуги прокрутки*, що є корисним у разі, якщо введений к поле текст не вміщується на екрані. Але *Смуги прокрутки* можуть бути використані і самостійно для введення до програми якогось значення із заздалегідь визначеного діапазону шляхом переміщення повзунка *Смуги прокрутки* (Рисунок 50).

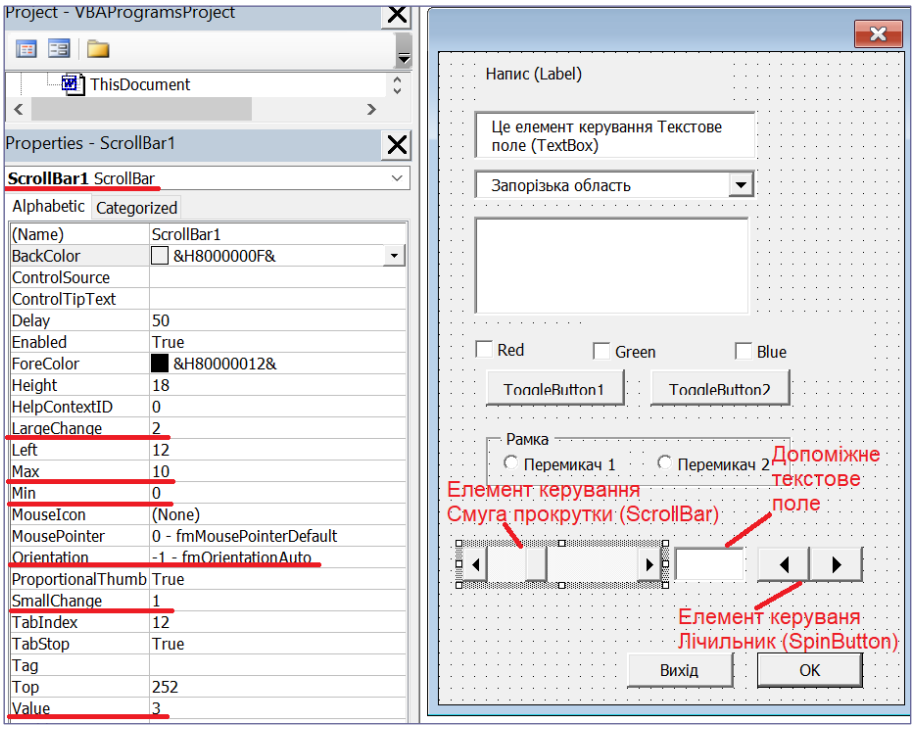


Рисунок 50 – Форма з елементами керування Смуга прокрутки (ScrollBar) та Лічильник (SpinButton)

До найважливіших властивостей *Смуги прокрутки* можна віднести:

- `Max` – максимальне значення діапазону, з якого повзунком обирається значення;
- `Min` – мінімальне значення діапазону, з якого повзунком обирається значення;
- `Value` – містить значення, обране положенням повзунка;

- LargeChange – визначає "крок" зміни значення у разі клікання по області *Смуги прокрутки*, вільної від повзунка;

- SmallChange – визначає "крок" зміни значення у разі клікання по кнопкам *Смуги прокрутки* з напрямом прокрутки;

- Orientation – визначає розташування повзунка: вертикальне – 0 – frmOrientationVertical, горизонтальне – 1 – frmOrientationHorizontal або -1 – frmOrientationAuto (за замовчуванням). Останнє значення конфігурує автоматичне визначення орієнтації у залежності від того, як визначається розмір елемента при доданні на форму.

Методів у цього елемента керування небагато і вони використовується не часто, а головна подія – Change. Нижче буде наведений код обробника такої події.

Елемент керування *Лічильник (SpinButton)* схожий на *Смугу прокрутки*, але не має самої смуги і повзунка, він дозволяє задавати значення з заздалегідь визначеного діапазону тільки за допомогою кнопок з напрямом зміни значення. Властивості *Лічильника*, за виключенням SmallChange та LargeChange, яких у нього не існує, збігаються з властивостями *Смуги прокрутки*, те ж саме стосується методів та подій.

Нижче наведений код обробників подій Change обох елементів керування, що виводить значення від 0 до 10 у допоміжному *Текстовому полі* та синхронізує стани обох елементів керування (мінімальне та максимальне значення однакові у обох елементів керування):

```
Private Sub ScrollBar1_Change()
    TextBox2.Value = ScrollBar1.Value
    SpinButton1.Value = TextBox2.Value
End Sub

Private Sub SpinButton1_Change()
    TextBox2.Value = SpinButton1.Value
    ScrollBar1.Value = TextBox2.Value
End Sub
```

Елементи керування Набір вкладок (TabStrip) та *Набір сторінок (MultiPage)* застосовуються, коли елементи керування багато і їх не вдається розмітити на формі. Обидва елементи додають до форми необхідну кількість вкладок, збільшуючи, таким чином, площу форми для розміщення елементів керування. Відмінність між *Набором вкладок* та *Набором сторінок* полягає у тому, що на різних вкладках *Набору вкладок* завжди розташовуються ті ж самі елементи керування (можливо програмно змінювати їх вміст) (Рисунок 51), а на різних вкладках *Набору сторінок* можуть розміщуватися різні елементи керування (Рисунок 52). Додавати, видаляти, змінювати назву та порядок розміщення вкладки можливо командами контекстного меню, яке з'являється після клікання на будь-якій вкладці правою кнопкою миші.

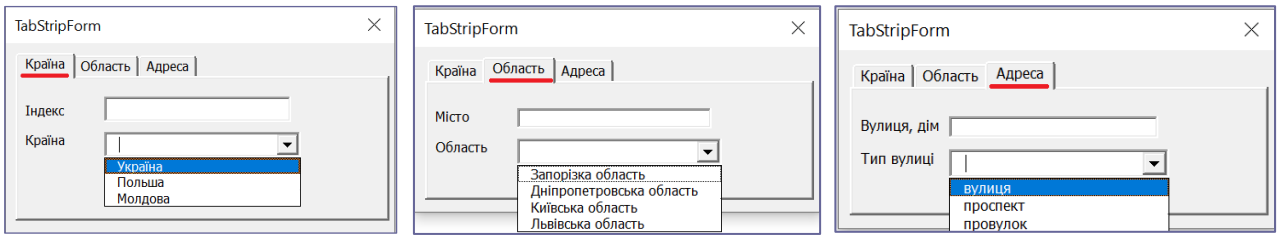


Рисунок 51 – Форма з елементом керування Набір вкладок (TabStrip)

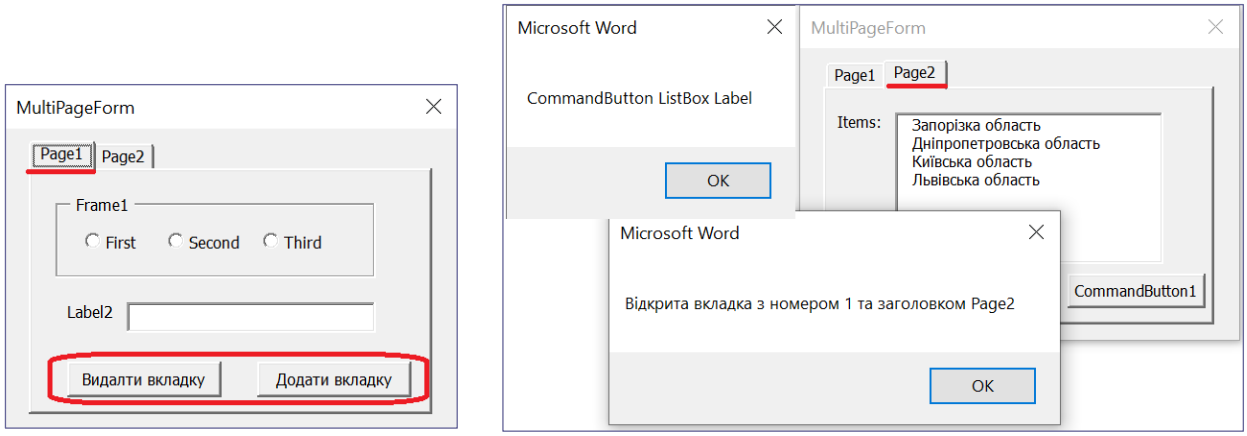


Рисунок 52 – Форма з елементом керування Набір сторінок (MultiPage)

Елементи керування *Набір вкладок (TabStrip) та Набір сторінок* мають більшість найчастіше застосовуваних властивостей однаковими, включаючи:

- `Value` – індекс поточної вкладки (починається з `0`), якщо встановити для цієї властивості необхідний індекс, активується відповідна вкладка;
- `SelectedItem` – повертає поточну (активну) вкладку (`SelectedItem.Caption` – повертає назву активної вкладки), ця властивість *Набору сторінок* має властивістю колекцію `Controls`, що зберігає посилання на елементи керування на поточній сторінці, *Набір вкладок* такої функціональності не має, оскільки кожна вкладка містить ті ж самі елементи керування;
- `MultiRow` – визначає, чи можна буде розміщувати вкладки у кілька рядів.
- `TabOrientation` – визначає місце розміщення вкладок на формі: зверху (за замовчуванням), знизу, зліва або справа.

Елемент керування *Набір вкладок* має властивість `Tabs` – колекцію об'єктів `Tab`, що відповідають вмісту вкладок, окрема вкладка може бути активована передачею її індексу як параметра до цієї властивості (`Tabs(index).Caption` – повертає назву вкладки з індексом `index`). А елемент керування *Набір сторінок* має властивість `Pages` – колекцію об'єктів `Page`, що відповідають вмісту вкладок, окрема вкладка може бути активована передачею її індексу як параметра до цієї властивості (`Pages(index).Caption` – повертає назву вкладки з індексом `index`).

Наведемо код ініціалізації форми та обробника події, що змінює вміст однакових за типом елементів керування на різних вкладках *Набору вкладок*:

```
Private Sub UserForm_Initialize()
    TabStrip1.Value = 0  'Активізується перша вкладка
    ComboBox1.List = Array("Україна", "Польща", "Молдова")
    Debug.Print TabStrip1.Tabs(0).Caption
End Sub

Private Sub TabStrip1_Change()
    Select Case TabStrip1.SelectedItem.Caption
        Case "Країна"
            Label1.Caption = "Індекс"
            Label2.Caption = "Країна"
            ComboBox1.Clear
            ComboBox1.List = Array("Україна", "Польща", "Молдова")
        Case "Область"
            Label1.Caption = "Місто"
            Label2.Caption = "Область"
            ComboBox1.Clear
            ComboBox1.List = Array("Запорізька область", _
                 "Дніпропетровська область", "Київська область", _
                 "Львівська область")
        Case "Адреса"
            Label1.Caption = "Вулиця, дім"
            Label2.Caption = "Тип вулиці"
            ComboBox1.Clear
            ComboBox1.List = Array("вулиця", "проспект", _
                                   "провулок")
    End Select
End Sub
```

Методів у елементів керування *Набір вкладок та Набір сторінок* небагато і вони використовуються не часто, а для властивостей-колекцій `Tabs` та `Pages` існують методи, що дозволяють програмно додавати – `Add`, видаляти – `Remove` вкладки, а також посилатися на елементи керування, що містяться на вкладці – `Item` та видаляти усі елементи керування з вкладки – `Clear`. Також корисною є властивість колекцій `Count`, що містить кількість вкладок.

Головна подія елементів керування *Набір вкладок та Набір сторінок* – `Change` виникає при переходах між вкладками. Наведемо приклад програмного налаштування та роботи з вкладками *Набору сторінок*:

```
Private Sub UserForm_Initialize()
    ListBox1.List = Array("Запорізька область", _
                "Дніпропетровська область", "Київська область", _
                "Львівська область")
End Sub

Private Sub MultiPage1_Change()
    MsgBox "Відкрита сторінка з номером " & MultiPage1.Value _
         & " та заголовком " & MultiPage1.SelectedItem.Caption
    DisplayPageControls
```

```
End Sub

Private Sub AddPageBtn_Click()
    Dim lastPageNumber As Integer

    MultiPage1.Pages.Add
    lastPageNumber = MultiPage1.Pages.Count - 1
    MsgBox "Додано сторінку з номером " & lastPageNumber _
         & " та заголовком " _
         & MultiPage1.Pages(lastPageNumber).Caption
End Sub

Private Sub RemovePageBtn_Click()
    Dim lastPageNumber As Integer, lastPageNumberCaption As String

    lastPageNumber = MultiPage1.Pages.Count - 1
    lastPageNumberCaption = _
                          MultiPage1.Pages(lastPageNumber).Caption
    MultiPage1.Pages.Remove lastPageNumber
    MsgBox "Видалено сторінку з номером " & lastPageNumber _
        & " та заголовком " & lastPageNumberCaption
End Sub

Private Sub DisplayPageControls()
    Dim ctrl As Control, msg As String

    For Each ctrl In MultiPage1.SelectedItem.Controls
        msg = msg & " " & TypeName(ctrl)
    Next
    MsgBox msg
End Sub
```

У обробнику події Change *Набору сторінок* після виведення інформації про відкриту вкладку викликається підпрограма `DisplayTabControls()`, у якій для обраної вкладки виконується ітерація її колекції `Controls` із виведенням назв типів елементів керування, що містить вкладка, до діалогового вікна. У обробниках подій Click *Кнопок* першої вкладки продемонстроване програмне додання та видалення останньої вкладки.

Додання рисунків до форм підвищує естетичність їх сприйняття і може бути використане для реалізації певних функцій. Розміщувати *Рисунок* в одному з поширених форматів (.bmp, .cur, .gif, .ico, .jpg, .wmf) на формі дозволяє *елемент керування Рисунок (Image)* (Рисунок 53).

Основними властивостями *Рисунку* можна назвати:

- `Picture` – містить посилання на файл з зображенням, яке буде відображатися;

- `PictureSizeMode` – дозволяє обрати розмір зображення: `0` - `frmPictureSizeModeClip` – обрізає *Рисунок* за розмірами області для його відтворювання (за замовчуванням), `1` - `frmPictureSizeModeStretch` – розтягує/стискає *Рисунок* за розмірами області для його відтворювання,

заповнюючи усю область (при цьому можливі зміни пропорцій рисунку), `3 – frmPictureSizeModeZoom` – розтягує/стискає *Рисунок*, вміщуючи його до області для його відтворювання та не порушуючи пропорцій *Рисунку*;

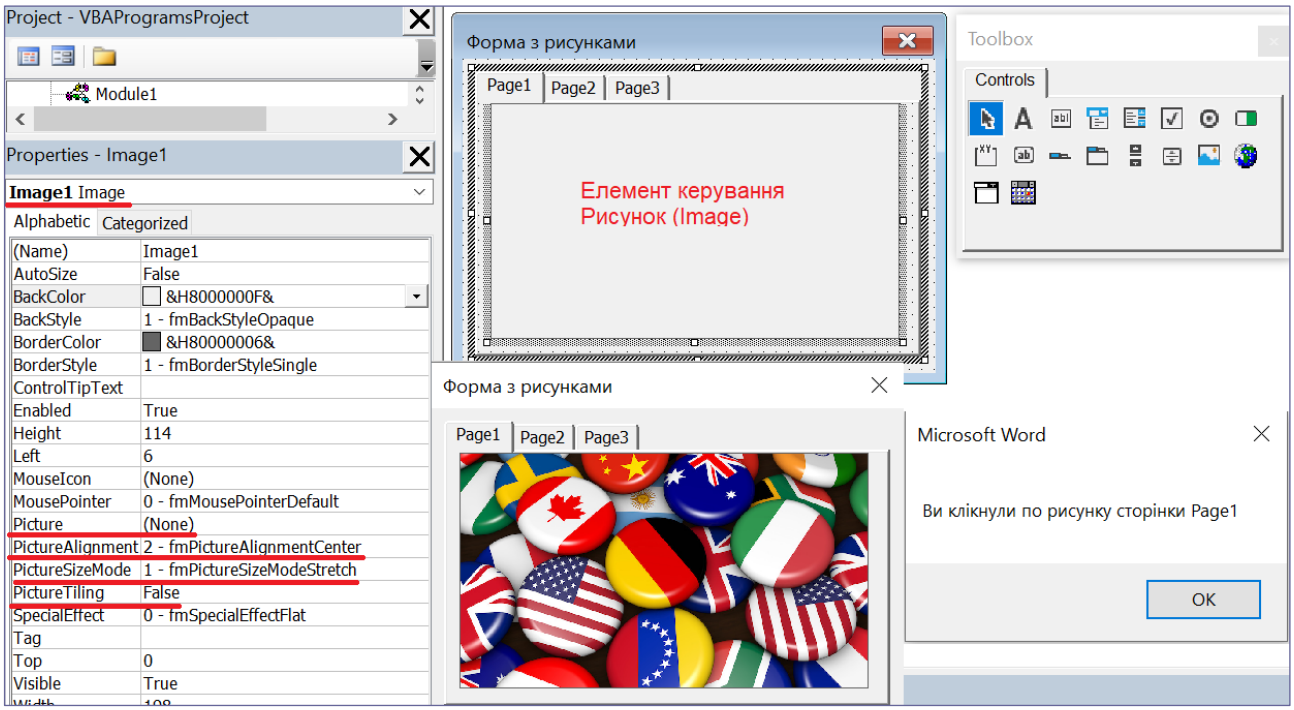


Рисунок 53 – Форма з елементом керування Рисунок (Image)

- `PictureAlignment` – визначає вирівнювання зображення у області для його відтворювання, за замовчуванням – по центру;
- `PictureTiling` – у разі значення `True` дозволяє покривати мозаїкою з рисунку усю область для його відтворювання.

Методів у *Рисунка* немає зовсім, а головною подіє вважається подія `Click`, але вона є схованим членом класу Image, це можна побачити у *Оглядачі об'єктів* (*Object Browser*), обравши у контекстному меню, яке відкриється після кліку на класі Image команду *Show hidden members* (*Показати сховані члени*) (Рисунок 54).

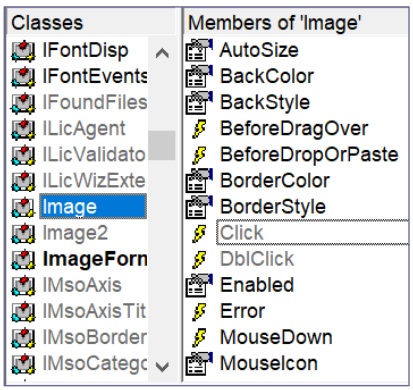


Рисунок 54 – Сховані події елемента керування Рисунок (Image)

Наведемо код програми, яка додає рисунки з файлів з розширенням ,jpg, розташованих у підкаталозі каталогу з файлом, у якому міститься код програми, до сторінок *Набору сторінок*:

```
Private Sub UserForm_Initialize()
    Dim ctrl As Image
    Dim vFile As String
    Dim i As Integer

    ChDir ActiveDocument.Path & "\images"

    i = 0
    vFile = Dir("*.jpg")
    Do While vFile <> ""
        MultiPage1.Value = i
        Set ctrl = MultiPage1.SelectedItem.Controls(0)
        ctrl.Picture = LoadPicture(vFile)
        vFile = Dir()
        i = i + 1
    Loop
    MultiPage1.Value = 0
End Sub

Private Sub Image1_Click()
    MsgBox "Ви кликнули по рисунку сторінки "
          & MultiPage1.SelectedItem.Caption
End Sub

Обробники Image2_Click() та Image3_Click() – аналогічно
```

Програма використовує вбудовані функції для роботи з файлами:

- `ChDir path` – виконує перехід з поточного каталогу до каталогу, вказаному параметром `path`, властивість `Path` об'єкта `ActiveDocument` містить шлях до каталогу з файлом, у якому зберігається програма;

- `Dir [(path [, attributes])]` – повертає рядок, що представляє назву файлу або каталогу, які відповідають вказаному шаблону `path` чи атрибуту `attributes`, у разі наявності декількох файлових об'єктів, що відповідають параметрам, повертається перший з них. Для отримання наступного, викликається `Dir()` без параметрів. У разі, якщо збігів не буде, повертається пустий рядок.

- `LoadPicture(picturename)` – повертає Рисунок , завантажений з файлу, вказаному параметром `picturename`.

Зауважимо, що форма та елементи керування, наприклад, *Напис, Кнопка, Прапорець, Кнопка з фіксацією, Перемикач,* сторінки *Набору сторінок* мають властивість `Picture`, яка дозволяє завантажувати рисунки без використання елементу керування *Рисунок* . Також існують засоби для завантаження та роботи з рисунками до документів Microsoft Office, ми розглянемо їх у наступному розділі.

4.4Додаткові елементи керування, їх властивості, методи та події

Наступні декілька елементів керування є додатковими і для їх відображення на *Панелі інструментів* (*Toolbox*) необхідно їх підключити командою контекстного меню *Панелі інструментів Additional Controls* (або командою головного меню *Tools – Additional Controls*) та обрати потрібний елемент, наприклад, *Microsoft Web Browser* (Рисунок 55).

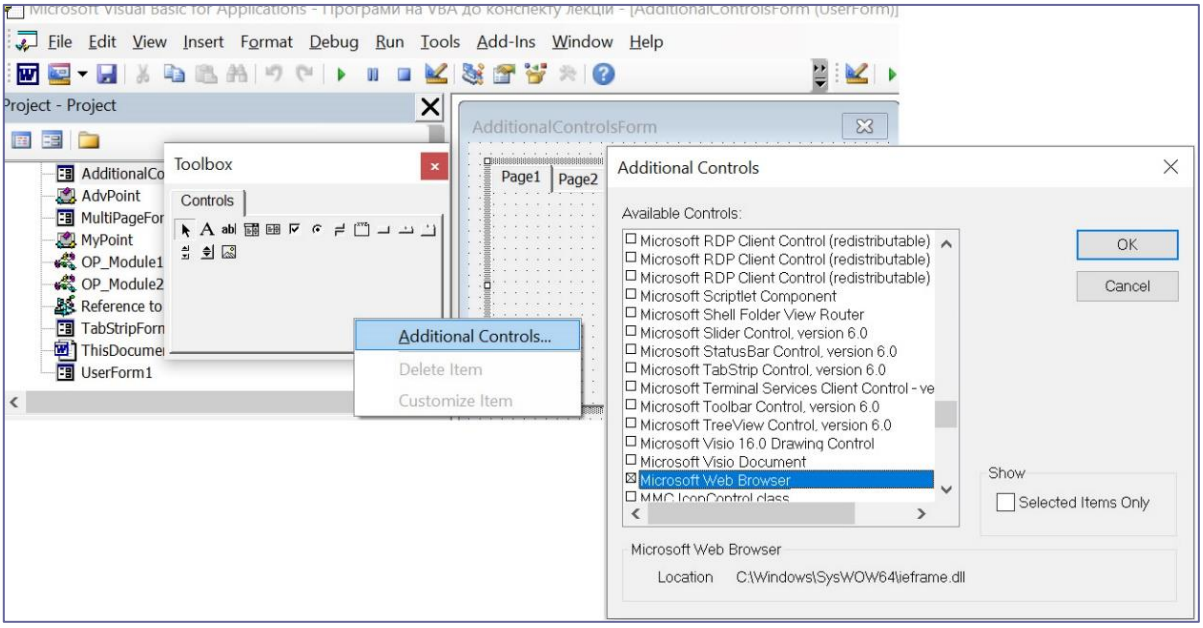


Рисунок 55 – Підключення додаткових елементів керування

Для роботи з елементом керування *WebBrowser* потрібно його додати до форми звичайним чином (Рисунок 56) та активувати доданням до обробнику події, наприклад, ініціалізації форми код завантаження початкової сторінки довільного серверу Інтернет, наприклад:

```
Private Sub UserForm_Initialize()
    WebBrowser1.Silent = True
    WebBrowser1.Navigate "www.google.com"
End Sub
```

Зауважимо, що перед завантаженням сторінки доцільно перевести елемент керування у режим блокування повідомлень про проблеми зі скриптами (`Silent`), стандартні протоколи (http://, https://) та порти (80) в адресі можна не вказувати. Після запуску форми розміщений у ній браузер завантажить відповідну сторінку (Рисунок 57).

Операції введення дати зручно виконувати за допомогою елементів керування *DTPicker* або *MonthView*. Для їх використання необхідно завантажити бібліотеку Microsoft Common Controls 2 Object Library, що містить, зокрема елемент керування *Microsoft Date and Time Picker Control*, наприклад, з https://www.ocxme.com/files/mscomct2_ocx, розпакувати архів та скопіювати файл MSCOMCT2.OCX у каталог Windows\SysWOW64 (для 64-бітних версій Windows) або у каталог Windows\System32 (для 32-бітних версій Windows). Після цього запустіть вікно командного рядка (cmd) з правами адміністратора,

перейдіть до каталогу, до якого Ви скопіювали файл MSCOMCT2.OCX, та виконайте команду:

```
regsvr32 MSCOMCT2.OCX
```

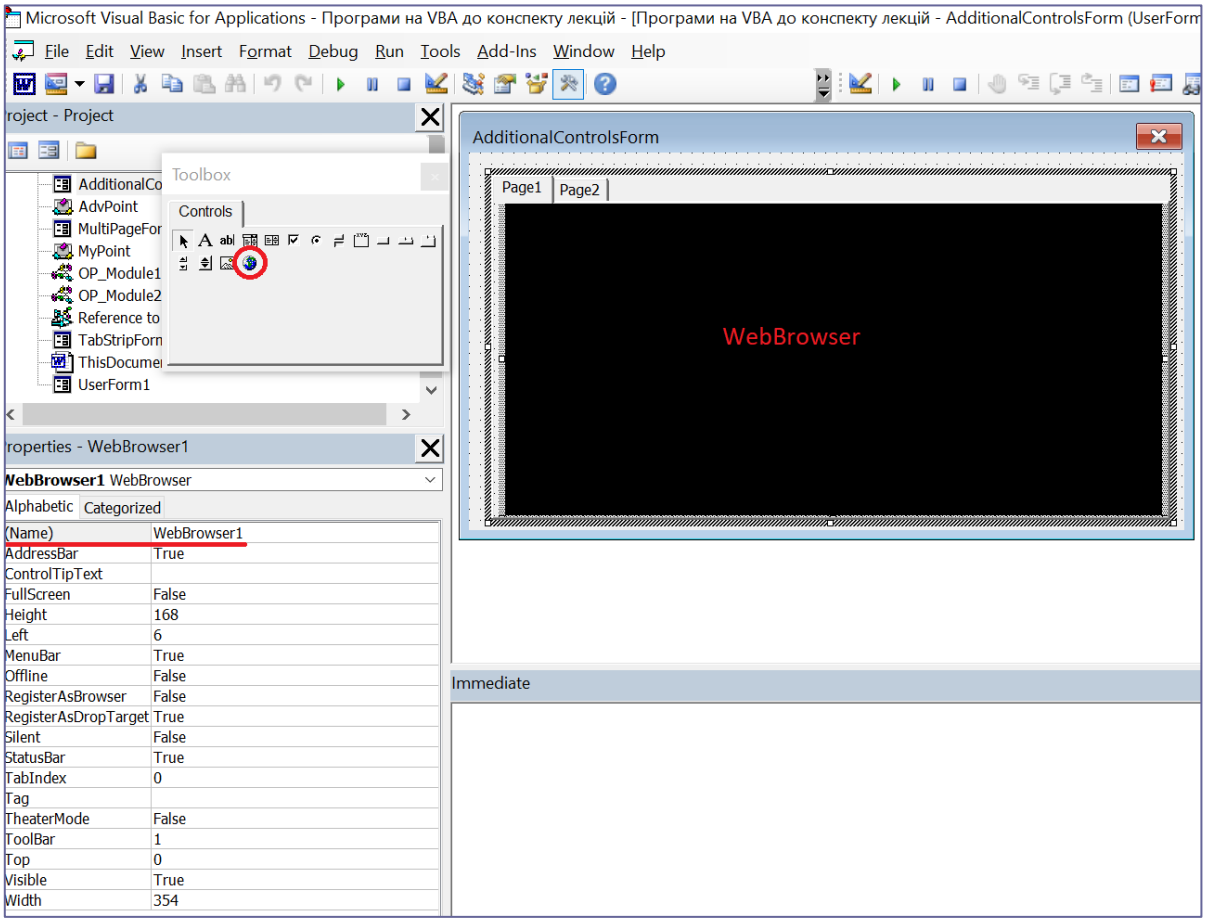


Рисунок 56 – Форма з елементом керування Microsoft Web Browser

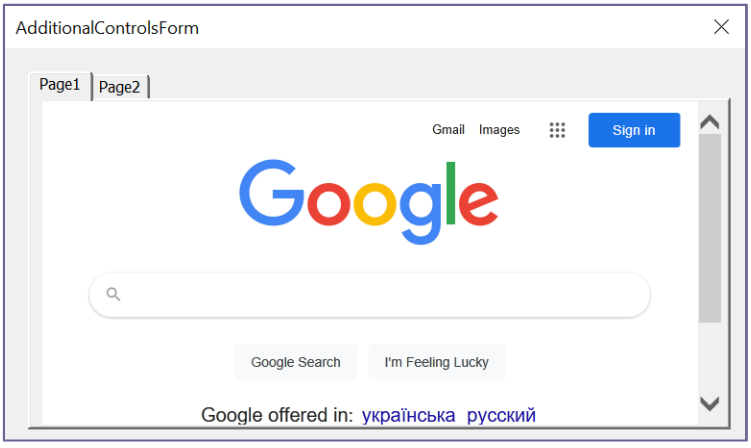


Рисунок 57 – Форма з браузером

Повинно з'явитися діалогове вікно з повідомленням успішної реєстрації бібліотеки. Після цього у вікні *Додаткових елементів керування* (*Additional Controls*) (Рисунок 55) Редактора VBA з'являться елементи керування:

- *Microsoft Animation Control* – анімація виконання будь-якого процесу (копіювання/переміщення файлу з однієї папки в іншу, пошук та інше)

- *Microsoft UpDown Control* – покращений лічильник з додатковими можливостями;

- *Microsoft MonthView Control* (*MonthView*)– вдалий календарик "в чистому вигляді";

- *Microsoft Date and Time Picker Control* (*DTPicker*)– цей же календарик *MonthView*, але у вигляді комбінованого списку, крім дати можна працювати і з часом;

- *Microsoft Flat ScrollBar Control* – смуга прокрутки.

Головною властивістю елементів керування *DTPicker* та *MonthView* є `Value` –дата/час, які обрані користувачем. Решта властивостей призначені для конфігурування зовнішнього вигляду календаря.

Нижче наводиться код обробників події Change для елемента керування *Microsoft Date and Time Picker Control* та події DateClick для елемента керування *Microsoft MonthView Control*, які виставляють обрану користувачем дату у допоміжне *Текстове поле,* синхронізуючи один одного (Рисунок 58):

```
Private Sub DTPicker1_Change()
    TextBox1.Value = DTPicker1.Value
    MonthView1.Value = DTPicker1.Value
End Sub

Private Sub MonthView1_DateClick(ByVal DateClicked As Date)
    TextBox1.Value = MonthView1.Value
    DTPicker1.Value = MonthView1.Value
End Sub
```

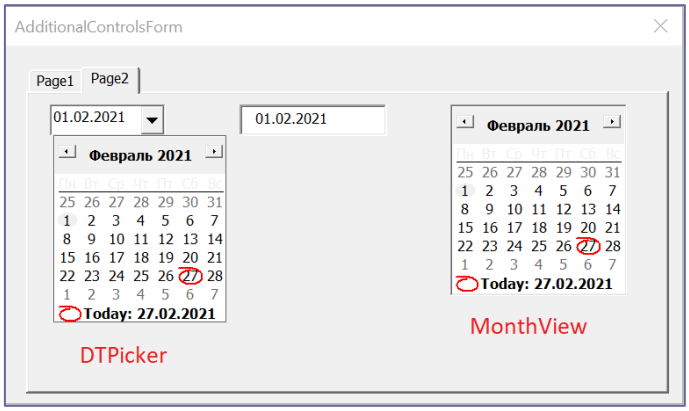


Рисунок 58 – Елементи керування Microsoft Date and Time Picker та Microsoft MonthView

Зверніть увагу, що бібліотеку MSCOMCT2.OCX потрібно встановлювати не тільки на комп'ютері, на якому розробляється програма VBA з додатковими елементами керування, але і на тих комп'ютерах, де ця програма буде запускатись – в іншому випадку виникне повідомлення про помилку.

4.5 Програмне додання елементів керування та налаштування обробників подій

Додавати елементи керування також можна програмно, для цього використовується метод `Add()`, що додає елемент до колекції форми `Controls`, у якій містяться усі елементи керування. Сигнатура цього методу для більшості елементів наступна:

```
Set ControlName = object.Controls.Add(ProgID [, Name [,
Visible]]),
```

де `object` – ім'я форми (для поточної форми зручно використовувати `Me`),

`ProgID` – рядок-ідентифікатор елемента керування, що має вигляд: `назва_колекції_елементів.назва_елемента.назви_версії_елемента`, значення параметру `ProgID` для інших елементів керування можна подивитись тут [24].

`Name` – ім'я доданого елемента,

`Visible` – визначає, чи відображати елемент (`True` – значення за замовчуванням) або – ні (`False`).

Наприклад, для програмного додання кнопки до поточної форми можна додати до обробника події ініціалізації форми наступний код:

```
Private Sub UserForm_Initialize()
    Dim MyCmd As MSForms.CommandButton

    Me.Caption = "Користувацька форма"
    Me.Width = 300
    Me.Height = 200

    Set MyCmd = Me.Controls.Add("Forms.CommandButton.1", _
                                "MyCmd", True)
    MyCmd.Top = 60
    MyCmd.Left = 100
    MyCmd.Caption = "Моя кнопка"
End Sub
```

Тоді запуск цієї форми викличе цей обробник і форма буде мати вигляд (Рисунок 59).

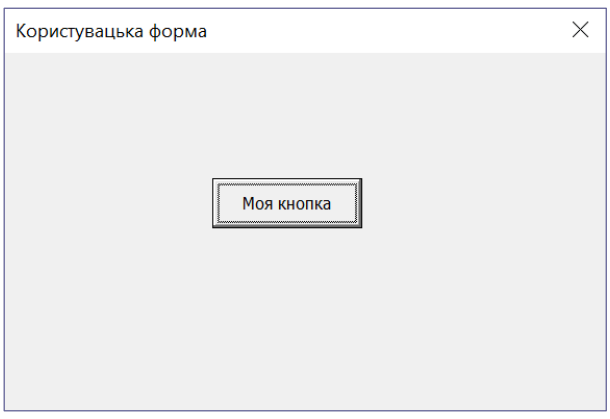


Рисунок 59 – Користувацька форма з програмно доданою командною кнопкою

Але є проблема – з програмно доданого елементу керування неможливо викликати обробник його подій (Рисунок 60).

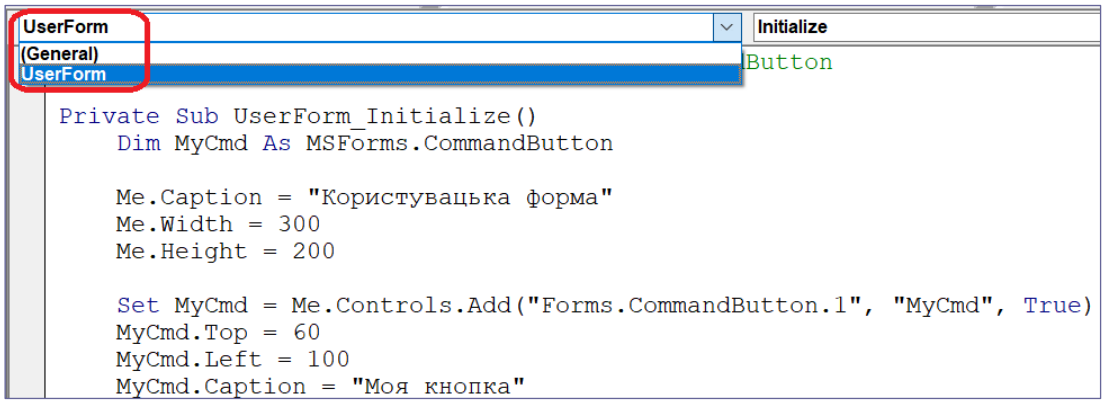


Рисунок 60 – Відсутність програмно доданого елемента керування у списку доступних об'єктів

Вирішується ця проблема доданням до форми об'єкта `CommandButton` з бібліотеки `MSForms` як об'єкта класу з подіями та присвоєння посилання на програмно доданий елемент цьому об'єкту [6]:

```
Public WithEvents CustomBtn As MSForms.CommandButton

Private Sub UserForm_Initialize()
    Dim MyCmd As MSForms.CommandButton

    Me.Caption = "Користувацька форма"
    Me.Width = 300
    Me.Height = 200

    Set MyCmd = Me.Controls.Add("Forms.CommandButton.1", _
                                "MyCmd", True)
    MyCmd.Top = 60
    MyCmd.Left = 100
    MyCmd.Caption = "Моя кнопка"

    Set CustomBtn = MyCmd
End Sub
```

В результаті у списку об'єктів з'явиться об'єкт `CustomBtn` (Рисунок 61) і для нього можна буде обрати необхідний обробник подій, наприклад, Click:

```
Private Sub CustomBtn_Click()
    MsgBox "Привіт від " & CustomBtn.Caption
End Sub
```

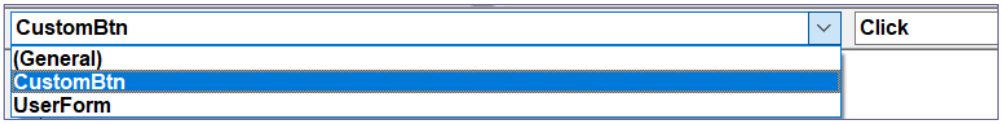


Рисунок 61 – Об'єкт кнопки, що посилається на програмно доданий елемент керування у списку доступних об'єктів

Аналогічним чином можуть бути програмно додані інші елементи керування та використані обробники їх подій.

Питання для самоконтролю

1. Назвіть основне призначення користувацьких форм. Опишіть типовий сценарій їх використання при автоматизованій обробці документів/робочих книг.
2. Назвіть команду додання користувацької форми до проєкту та переходів між вікном проєктування форми та модулем форми та назад.
3. Назвіть найважливіші властивості користувацьких форм. Як вони можуть задаватися для форми? Коли може бути використане ключове слово `Me`?
4. Назвіть найважливіші методи користувацьких форм. Як можна конфігурувати модальну поведінку форми?
5. Назвіть найважливіші події користувацьких форм. Обробник якої події форми, зазвичай, використовується для початкового налаштування її елементів керування?
6. Що називають елементами керування форм. Дайте визначення графічного інтерфейсу користувача.
7. Назвіть призначення та основні властивості елемента керування *Напис*. Чи має такий елемент методи та події?
8. Назвіть призначення, основні властивості, методи та події елемента керування *Текстове поле*. Як можна організувати введення з перенесенням на нові рядки?
9. Назвіть призначення, основні властивості та події елемента керування *Кнопка*.
10. Назвіть призначення, основні властивості, методи та події елементів керування *Список* та *Поле зі списком.* Опишіть відмінності між цими елементами керування. Як можна посилатися на елемент списку?
11. Назвіть призначення, основні властивості та події елементів керування *Прапорець* та *Кнопка з фіксацією.* Як можна налаштувати відстеження трьох станів для цих елементів?
12. Назвіть призначення, основні властивості та події елементу керування *Перемикач.* Для чого використовується елемент керування *Рамка (Frame)*?
13. Назвіть призначення, основні властивості та події елементів керування *Смуга прокрутки* та *Лічильник.* Опишіть відмінності між цими елементами керування.
14. Назвіть призначення, основні властивості, методи та події елементів керування *Набір вкладок* та *Набір сторінок.* Опишіть відмінності між цими елементами керування. Як можна отримати доступ до елементів керування сторінок *Набору сторінок*?
15. Назвіть призначення, основні властивості та події елементу керування *Рисунок* . Яка функція призначена для завантаження рисунку за файлу? Які елементи керування мають властивість `Picture`, яка дозволяє завантажувати рисунки до них безпосередньо?
16. Як підключити додаткові елементи керування, які входять до Microsoft Office та які завантажуються з додаткових бібліотек? До якого каталогу необхідно

завантажувати такі бібліотеки і як зареєструвати їх?

17. Опишіть спосіб налаштування елементу керування *WebBrowser* для відкриття ним довільного Інтернет-серверу при завантаженні форми. Для чого використовується його властивість `Silent`?
18. Назвіть призначення, основні властивості та події елементів керування *DTPicker* та *MonthView.* Опишіть відмінності між цими елементами керування.
19. Опишіть способи програмного додання елементів керування до форми. Наведіть формат та приклад застосування методу `Add()` для додання довільного елемента керування.
20. Опишіть спосіб забезпечення роботи обробників подій від програмно доданих елементів керування.

Тестові завдання для перевірки знань

1. Оберіть властивість, яка визначає заголовок користувацької форми:
   а) `Title`;
   б) `Caption`;
   в) `Header`;
   г) `Name`.
2. Оберіть метод, що запускає користувацьку форму:
   а) `Load`;
   б) `Show`;
   в) `Run`;
   г) `Exec`.
3. Оберіть подію, яка відбувається при підготовці користувацької форми до відкриття:
   а) Layout;
   б) Click;
   в) Initialize;
   г) Activate.
4. Оберіть властивість елементу керування *Текстове поле*, яка дозволяє заборонити змінювати дані в елементі керування, але дозволяє їх копіювати:
   а) Enabled;
   б) Accessed;
   в) Locked;
   г) Opened.
5. Оберіть подію елементу керування *Текстове поле*, яка відбувається після натискання клавіші Enter у полі з текстом:
   а) Exit;
   б) Enter;
   в) Change;
   г) AfterUpdate.
6. Оберіть подію елементу керування *Поле зі списком*, яка відбувається після обрання елементу списку:
   а) Change;

б) AfterUpdate;
в) Enter;
г) Exit.

7. Оберіть назву колекції, у якій зберігаються посилання на елементи керування форми:
а) Elements;
б) ManagedElements;
в) Handlers;
г) Controls.

8. Оберіть елементи керування, які мають властивість `Picture`, що дозволяє завантажувати рисунки до них безпосередньо (оберіть декілька відповідей):
а) *Напис*;
б) *Список*;
в) *Кнопка*;
г) *Прапорець*,
д) *Текстове поле*.

9. Оберіть додаткові елементи керування, які можна використовувати у формах VBA (оберіть декілька відповідей):
а) ToggleButton;
б) Date and Time Picker;
в) SpinButton;
г) Web Browser,
д) MonthView.

10. Оберіть правильний формат зазначеного у процедурі модуля форми з ім'ям MyUserForm оператора програмного додання елемента керування *Список* до цієї форми:
а) `MyUserForm.Add("Forms.ListBox.1", "MyList", True);`
б) `Me.Controls.Add("Forms.ListBox.1", "MyList", True);`
в) `MyUserForm.Controls.Add("Forms.ListBox", "MyList", True);`
г) `Me.Add("Forms.ListBox.1", "MyList", True);`
д) `MyUserForm.Controls.Add("ListBox", "MyList", True).`

Завдання для самостійного виконання

1. Розробіть користувацьку форму "Тестування", яка відображає питання і кілька відповідей на нього (передбачити одноваріантні і багатоваріантні відповіді). Форма повинна пропонувати кілька питань, і в кінці тестування виводити розрахований бал.

2. Розробіть користувацьку форму "Геометричні фігури", що обчислює та виводить значення периметра та площі для квадрата, трикутника, прямокутника і трапеції, які обираються зі списку, за необхідними даними, які вводяться користувачем у необхідну кількість текстових полів (зайві поля на формі відображатися не повинні).

3. Розробіть користувацьку форму "Перетворення величин", яка дозволяє перетворювати введені значення градусів Цельсія в градуси Фаренгейта, кілометрів у милі, кілограмів у фунти, барів у Паскалі та літрів у галони.

4. Розробіть користувацьку форму "Щоденник" для додання, редагування та видалення нотаток. Нотатка відноситься до однієї з категорій, що обираються зі списку. Нотатка має тему, дату/час додання/останнього редагування та текст.

5. Розробіть користувацьку форму "Телефонний довідник", який дозволяє додавати, редагувати і видаляти записи: прізвище, ім’я, по-батькові – номер телефону. Передбачити можливість пошуку записів по прізвищу і по номеру телефону.

6. Розробіть користувацьку форму гри "Прапори країн", що відображає Рисунок прапора країни, випадково виведений з переліку файлів із зображеннями прапорів країн, і пропонує обрати зі списку назву країну, чий прапор відображений. Завдання має повторюватися десять разів з виведенням після закінчення кількості вгаданих варіантів.

7. Розробіть користувацьку форму "Тригонометричні функції", що дозволяє обчислювати тригонометричні функції *sin, cos, tg*, які обираються зі списку, для діапазону кутів від 0 до 90º з кроком 5º. Інтерфейс повинен відображати в першому стовпці значення градусів або відповідні значення радіанів (вибираються перемикачем), а у другому стовпці - значення функцій.

8. Розробіть користувацьку форму "Конвертор валют", що дозволяє для обраної з двох списків пари валют виконати конвертування введеної у текстове поле суми з першої валюти до другої валюти.

9. Розробіть користувацьку форму "Геометричні тіла", що обчислює площу поверхні та об'єм для куба, прямокутного паралелепіпеда, кулі і прямого кругового конуса, які обираються зі списку, за необхідними даним, які вводяться користувачем у необхідну кількість текстових полів (зайві поля на формі відображатися не повинні).

10. Розробіть користувацьку форму "Бінарний калькулятор", що дозволяє виконувати операції NOT, AND, OR і XOR над двома бітовими числами розмірів 8 або 16 або 32 біта (розрядність задається перемикачем).

ТЕМА 5 ОБ’ЄКТНА МОДЕЛЬ ТА ПРОГРАМУВАННЯ В MICROSOFT WORD

Мета: розглянути об'єктну модель Microsoft Word та вивчити найважливіші властивості та методи класів, що відображають сам текстовий процесор, та документи, з якими він працює, вивчити способи програмної навігації та редагування вмісту документу, пошуку і заміни його фрагментів, роботи з закладками, колонтитулами, таблицями та рисунками, доданими до документу.

План

1. Об’єктна модель Microsoft Word.
2. Клас застосунку Microsoft Word (`Application`), його основні властивості, методи та події
3. Робота з колекцією документів (клас `Documents`) та документами (клас `Document`)
4. Робота з діапазонами (клас `Range`), виділеними областями (клас `Selection`)
5. Робота з закладками (клас `Bookmark`) та колонтитулами (клас `HeaderFooter`)
6. Пошук та заміна фрагментів документу (класи `Find` та `Replacement`)
7. Робота з таблицями (класи `Tables`, `Table`, `Row`, `Column` та `Cell`)
8. Робота з рисунками (колекція `Shapes`, класи `Shape` та `InlineShape`)

Основні терміни і поняття

Об'єктна модель, застосунок Microsoft Word, колекція документів, документ, діапазон документу, виділена область документу, закладка, колонтитул, пошук фрагменту документу, таблиця документу, Рисунок документу.

Основні теоретичні положення

5.1 Об’єктна модель Microsoft Word

Microsoft Word – є найпопулярнішим текстовим процесором внаслідок наявності його розвиненого функціоналу зі створення та роботі з текстовими документами. Програми на VBA, створювані для обробки документів Microsoft Word, можуть застосовуватись (але не обмежуються) для наступних дій:

- автоматизація створення великої кількості однотипових документів шляхом заповнення полів шаблону даними, які отримані ззовні: зі списку/таблиці у іншому документі, з бази даних, з файлу табличного процесора Microsoft Excel тощо – наприклад, виклик на сесію для кожного студента групи, заяви працівників підприємства на оформлення щорічної відпустки тощо;
- валідація даних, зазначених у однотипових документах – наприклад, перевірка суми кредитів вивчених студентом дисциплін та його середнього балу

у додатках до диплому [7], перевірка заповнення полів документів, створених за допомогою шаблонів з закладками тощо;

- генерація на основі шаблонів звітів з результатами розрахунків, отриманих у інших програмах Microsoft Office, перевагами таких звітів є можливість корекції шаблону звичайним користувачем без допомоги програміста;

- автоматизоване конвертування сукупності документів з одного формату до іншого – наприклад, усіх файлів формату .docx у певному каталозі до формату .pdf.

- однотипове форматування документу або декількох документів – програмне застосування стилів до певних частин тексту, додання рамки для усіх рисунків документу, форматування тексту в усіх таблицях документу тощо.

Програмування автоматизованої обробки документів Microsoft Word виконується зміною значень властивостей та виконанням методів таких об'єктів, як розділ, сторінка, абзац, слово, діапазон, виділена область тексту тощо та застосуванням обробників подій таких об'єктів як сам застосунок Microsoft Word або документ [4]. Зазначені об'єкти, як і багато інших є об'єктами класів, що складають *об'єктну модель Microsoft Word* [25]. Ця модель містить більше 300 класів, але найчастіше використовуються зазначені на Рисунок 62.

5.2 Клас застосунку Microsoft Word (Application)

Клас `Application` – відповідає екземпляру застосунку Microsoft Word, створення об'єкту цього класу призводить до запуску Microsoft Word, для відображення вікна застосунку на екрані необхідно встановити властивості `Visible` створеного об'єкта значення `True` (також змінений стандартний заголовок застосунку для його ідентифікації) (Рисунок 63):

```
Dim myWord As Word.Application

Set myWord = new Word.Application
myWord.Visible = True
myWord.Caption = "Програмно запущений Microsoft Word"
```

Перевірити наявність двох екземплярів запущеного застосунку Microsoft Word (одного з файлом з програмами, а другого – запущеного вище вказаними операторами) можливо у *Диспетчері завдань* операційної системи, відкривши його відповідною командою після натискання клавіатурної комбінації *Ctrl+Alt+Delete* (Рисунок 64).

Після запуску вікно Microsoft Word не містить жодного документу. Способи додання та роботи з документами будуть викладені нижче у цьому розділі. Завершити роботу екземпляру Microsoft Word (це, зазвичай, виконують у кінці програми) можна викликом методу `Quit()` з цього екземпляру:

```
myWord.Quit
```

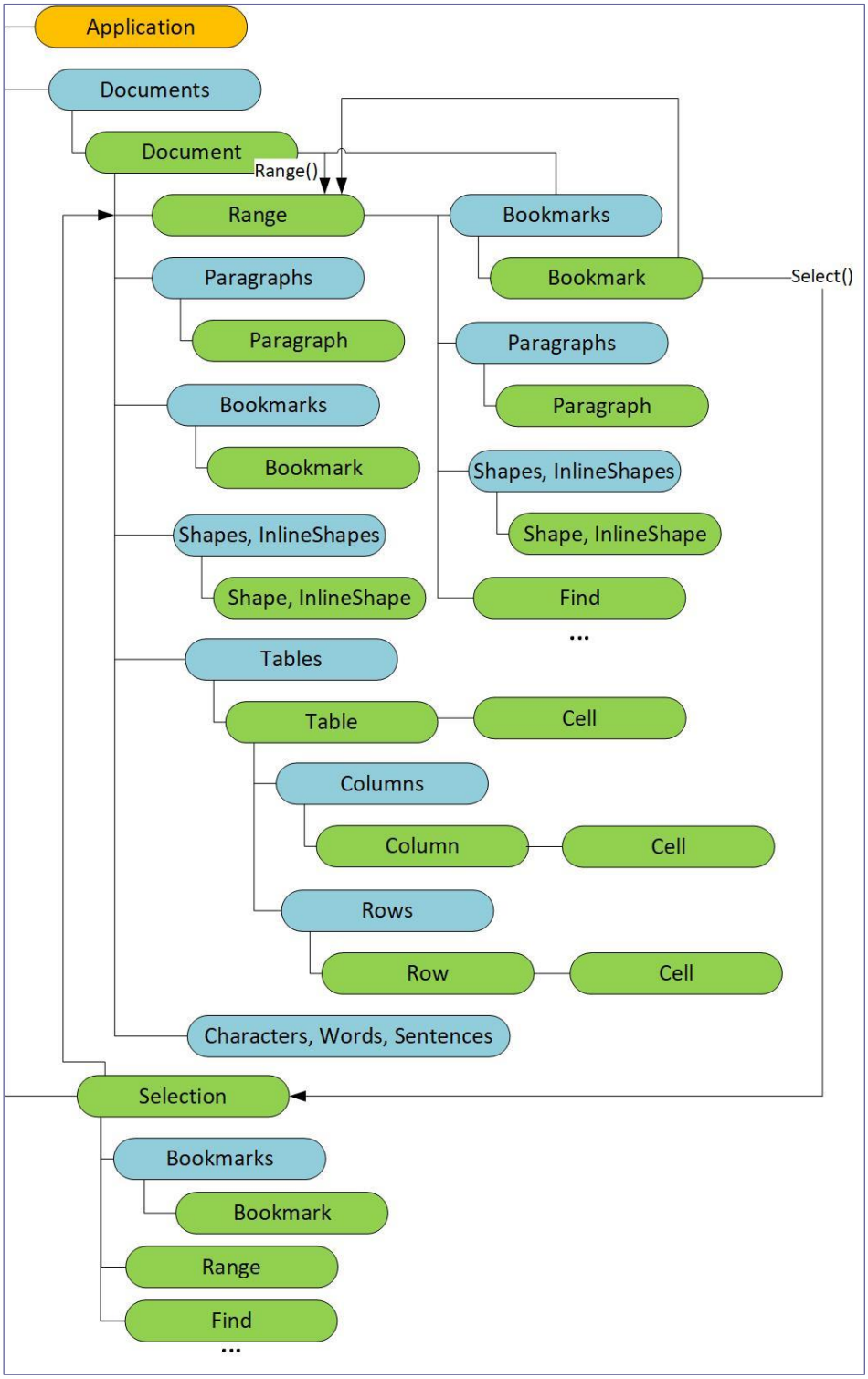


Рисунок 62 – Об'єктна модель Microsoft Word

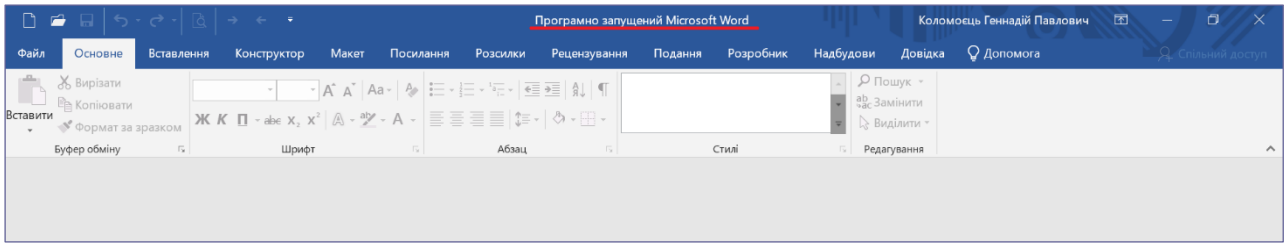


Рисунок 63 – Програмно запущений екземпляр Microsoft Word

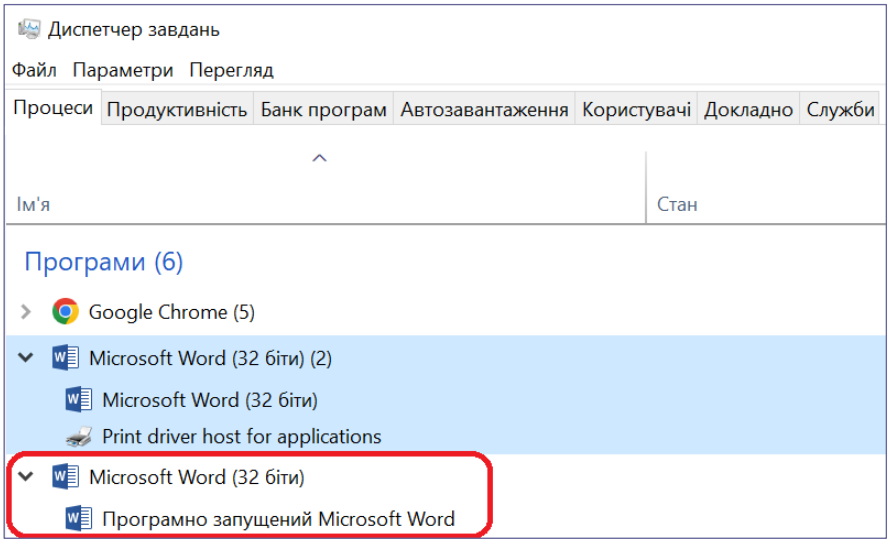


Рисунок 64 – Вікно Диспетчера завдань з двома екземплярами Microsoft Word

Досить часто програми містяться у документах Microsoft Word їх запуску Microsoft Word вже запущений. Тоді для доступу до екземпляру запущеного застосунку можна використати вбудовану функцію

```
GetObject([path] [, class]),
```

що повертає посилання на об'єкт, вказаний аргументом `class` у форматі `appname.objecttype`, де `appname` – назва застосунку, який надає об'єкт, `objecttype` – тип або клас об'єкту, на який функція повертає посилання. Аргумент `path` містить повний шлях та ім'я файлу, який містить об'єкт. Якщо аргумент `path` не вказується, то аргумент `class` є обов'язковим. Для запущеного застосунку Microsoft Word виклик цієї функції виглядає:

```
Set myWord = GetObject(, "Word.Application")
```

Ще один спосіб отримання посилання на вже запущений екземпляр Microsoft Word – використання назви класу `Application`, що автоматично посилається на запущений застосунок і ми можемо працювати з властивостями застосунку та викликати методи класу `Application`. Більш того, якщо навіть не вказувати назву класу `Application`, а зразу зазначати його властивості або методи, вони також будуть відноситися до запущеного застосунку Microsoft Word. Наведемо приклади такого неявного отримання посилання на запущений екземпляр Microsoft Word:

```
Debug.Print Application.Name
```

або

```
Debug.Print Name
```

Обидва оператори виведуть "Microsoft Word" у вікні *Immediate*.

До *найважливіших властивостей класу* `Application` можна віднести:

- `Documents` – колекція документів, які відкриті у поточному об'єкті `Application` – роботу з цією колекцією буде розглянуто нижче у цьому розділі;

- Windows – колекція об'єктів Window, пов'язаних з вікнами Microsoft Word, у яких відкриті документи;

- ActiveDocument – повертає посилання на об'єкт активного документа (вікно якого є активним на цей момент), відкритого у поточному екземплярі Microsoft Word;

ActiveWindow – повертає посилання на об'єкт вікна активного документа (об'єкт класу Window), відкритого у поточному екземплярі Microsoft Word (у одному екземплярі можуть бути відкриті декілька документів – кожен у своєму вікні). Властивості цього об'єкту дозволяють програмно керувати вертикальною та горизонтальною прокруткою та виглядом вікна, у якому відкритий документ. Отримати доступ до вікна відкритого документа можна за його ім'ям, наприклад:

```
Set myWindow = Windows("Документ-зразок.docx")
```

Зауважимо, що для документів, створених у попередніх версіях Microsoft Word (*.doc), до ім'я додається суфікс [Режим сумісності] – це видно у заголовку вікна. У цьому разі ім'я документу-параметр повинно включати такий суфікс.

- FileDialog(fileDialogType) – містить посилання на об'єкт FileDialog, що відповідає діалоговому вікну обрання файлів/каталогів для їх подальшого використання або відкриття чи збереження файлу. Значення параметру fileDialogType:

  - msoFileDialogFilePicker – дозволяє користувачу обрати файл;
  - msoFileDialogFolderPicker – дозволяє користувачу обрати каталог;
  - msoFileDialogOpen - дозволяє користувачу відкрити файл;
  - msoFileDialogSaveAs - дозволяє користувачу зберегти файл.

Для відкриття цього вікна необхідно скористатися методом Show() цього об'єкта, який повертає цілочисельне значення відповідно до кнопки вікна, по якій користувач виконав клікання: -1 – у разі клікання по кнопці *Відкрити*/*Зберегти* та 0 – у разі клікання по кнопці *Скасувати*. У разі обрання користувачем одного або декількох файлів/каталогів їх повні імена збережуться у колекції SelectedItems об'єкту FileDialog. Наприклад, обрання файлу для відкриття можна організувати таким чином:

```
Dim fDialog As FileDialog

Set fDialog = Application.FileDialog(msoFileDialogOpen)
If fDialog.Show = -1 Then
    MsgBox fDialog.SelectedItems(1)
End If
```

Зауважимо, що обрані файлові об'єкти розміщуються у колекції SelectedItems, починаючи з індексу 1. У разі встановлення значення параметру fileDialogType у msoFileDialogFilePicker або msoFileDialogFolderPicker, можливо у діалоговому вікні обрати декілька

файлів/каталогів, переключення у такий режим реалізується встановленням властивості AllowMultiSelect об'єкта FileDialog у значення True:

```
Dim selectedFile As Variant

With Application.FileDialog(msoFileDialogFilePicker)
    .AllowMultiSelect = True
    If .Show = -1 Then
        Debug.Print "Ви обрали файл(и): "
        For Each selectedFile In .SelectedItems
            Debug.Print selectedFile
        Next
    End If
End With
```

Наведена програма виведе у вікно *Immediate* список повних імен усіх файлів, обраних у діалоговому вікні. Необхідно зауважити, що використання властивості FileDialog не відкриває та не зберігає файли (тобто не створює посилання типу Document, з яким далі можна працювати), а лише повертає повний шлях до файлів\каталогів, забезпечуючи інтерактивний спосіб їх визначення.

- Name – містить назву застосунку ("Microsoft Word"), тільки для читання;
- Caption – містить назву застосунку в заголовку вікна ("Microsoft Word" за замовчуванням), може бути змінена на довільний рядок;
- Version – містить посилання на версію Microsoft Word (менш детальну, ніж Build). Для Word 2016/2019 значення цієї властивості - 16.0);
- StatusBar – дозволяє вивести текст у рядку в нижній частині вікна програми;
- Visible – дозволяє сховати вікно Microsoft Word.

До *найважливіших методів класу* *Application* можна віднести:

- PrintOut() – друкує документ або його частину на принтері за замовчуванням, має багато параметрів, у тому числі:
  - FileName – файл, що буде роздрукований, якщо не зазначається – друкується активний документ;
  - Pages – сторінки документу, що будуть друкуватись, наприклад, 2, 6-10;
  - Copies – кількість копій документу, які будуть роздруковані;
  - Background – у разі значення True – документ буде друкуватися разом з продовженням виконання програми.

Наступний приклад коду організує друк усіх документів Microsoft Word каталогу, у якому знаходиться документ з програмою:

```
myDoc = Dir("*.doc?")
Do While myDoc <> ""
    Application.PrintOut FileName:= myDoc
    myDoc = Dir()
Loop
```

- `Quit()` –закриває поточний екземпляр Microsoft Word, його параметр `SaveChanges` може приймати один з елементів енумератора `WdSaveOptions`:
  - `wdDoNotSaveChanges` – Microsoft Word буде закритий без збереження відкритих документів;
  - `wdPromptToSaveChanges` – при закритті Microsoft Word буде виведене діалогове вікно з запитом про необхідність збереження змінених документів;
  - `wdSaveChanges` – змінені документи зберігаються автоматично без запиту дій користувача.

Клас `Application` може генерувати велику кількість *подій* – при створенні, відкритті, перед збереженням, перед закриттям, перед друком, після зміни документа, перед активацією вікна з документом, перед закриттям екземпляру Microsoft Word тощо. Але працювати з подіями (визначати код методів-обробників подій) можна тільки у об'єктному модулі (модулі класу або форми) після оголошення у його розділі Declarations об'єкта `Application` з подіями:

```
Public WithEvents App As Word.Application

Private Sub App_NewDocument(ByVal doc As Document)
    MsgBox "Документ " & doc.Name & " створено."
End Sub
```

5.3 Робота з колекцією документів (клас Documents) та документами (клас Document)

Якщо об'єкт `Application` асоціюється з запущеним застосунком Microsoft Word, то колекція `Documents` – пов'язана з відкритими у застосунку документами, кожному з яких відповідає екземпляр класу `Document`. Колекція `Documents` та екземпляри класу `Document` надають способи створення, відкриття, збереження документа тощо [4]. Наприклад, створення документу забезпечується методом `Add` колекції `Documents` запущеного об'єкта `Application`:

```
Dim myWord As Word.Application
Dim standardDoc As Word.Document

myWord.Caption = "Програмно запущений Microsoft Word"
Set standardDoc = myWord.Documents.Add
```

Виконання цього коду призведе до створення у програмно запущеному екземплярі Microsoft Word документу зі стандартним ім'ям – Документ 1 (Рисунок 65).

Також існує можливість створення документу при оголошенні його змінної, у цьому разі він буде створений у поточному екземплярі Microsoft Word (у якому відкритий файл з програмою):

```
Dim myDoc As New Word.Document
```

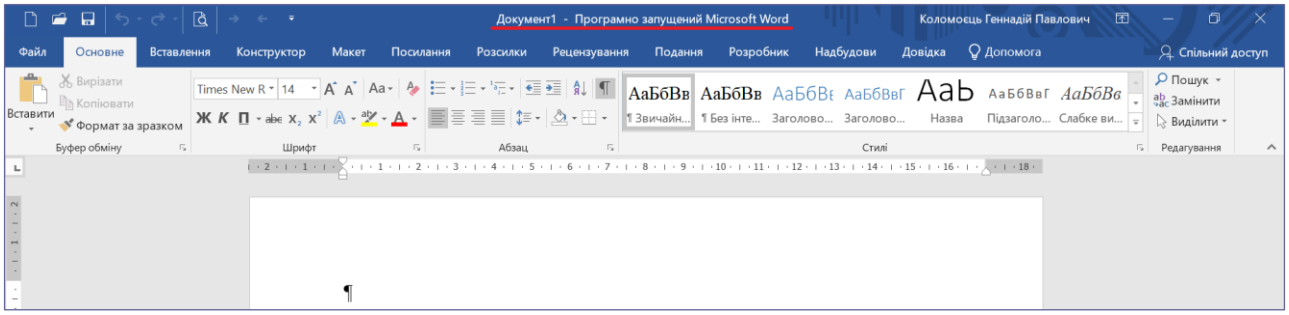


Рисунок 65 – Створення документу у програмно запущеному Microsoft Word

До *найважливіших властивостей колекції* `Documents` можна віднести тільки властивість `Count`, яка містить кількість відкритих документів, а *найважливіших методів* ця колекція має більше:

- `Add([Template] [, NewTemplate] [, DocumentType] [, Visible])` – додає новий документ до колекції і тим самим створює новий документ у екземплярі `Application` (Microsoft Word), який містить цю колекцію, метод має опціональні параметри:

  - `Template` – повний шлях до шаблону (файлу .dotx або .dot), на основі якого створюється документ, у разі невизначення цього параметру, документ створюється на основі шаблону normal.dotx;
  - `NewTemplate` – у разі значення `True` створює новий шаблон, за замовчуванням – `False`;
  - `DocumentType` – дозволяє зазначити тип створюваного документу: `wdNewBlankDocument` – новий документ, `wdNewEmailMessage` – нове повідомлення електронної пошти, `wdNewWebPage` – нова Веб-сторінка тощо, за замовчуванням – `wdNewBlankDocument`;
  - `Visible` – при встановленні цього параметру у `False`, документ відкривається але не відображається на екрані (вікну, у якому він відображається встановлюється властивість `Visible=False`).

Нижче наводиться код програми, яка запускає екземпляр Microsoft Word, та створює у ньому документ з ім'ям за замовчуванням Документ1 на основі користувацького шаблону:

```
Sub CreateDocument()
    Dim myWord As Word.Application
    Dim myDoc As Word.Document

    Set myWord = New Word.Application
    myWord.Visible = True
    myWord.Caption = "Програмно запущений Microsoft Word"

    Set myDoc = myWord.Documents.Add(
           "c:\MyTemplates\LR-Otchet-Template-ESEI-ZNU.dotx")
    Debug.Print myWord.Caption; " - "; myDoc.Name

    myDoc.Close
```

```
    myWord.Quit
End Sub
```

Запуск програми виводить у вікні *Immediate*:

```
Програмно запущений Microsoft Word - Документ1
```

А у разі виставлення контрольної точки для відлагодження, можна побачити створений документ на основі шаблону звіту з лабораторної роботи (Рисунок 66).

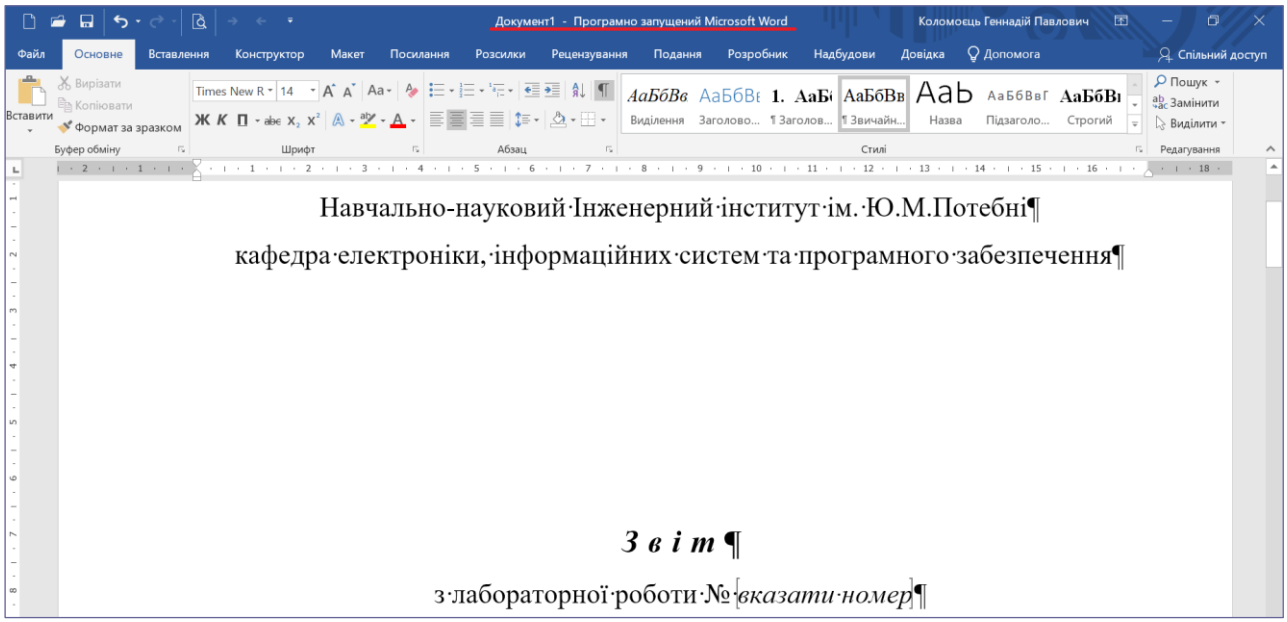


Рисунок 66 – Створення документу на основі користувацького шаблону

- `Open()` – дозволяє відкрити документ з файлу та додати до колекції `Documents`, окрім інших, має один обов'язковий параметр – повне ім'я файлу, що містить документ. Приклад відкриття документу з файлу:

```
Dim myWord As New Word.Application
Dim myDoc As Word.Document

Set myDoc = myWord.Documents.Open("C:\myDocument.docx")

myDoc.Close
myWord.Quit
```

- `Save()` – зберігає усі змінені (нові та відкриті з файлів) документи колекції `Documents`, виводить діалогове вікно збереження файлів для нових документів:

```
Dim myWord As New Word.Application
Dim myDoc As Word.Document
Dim stdDoc As Word.Document

Set myDoc = myWord.Documents.Open("C:\myDocument.docx")
Set stdDoc = myWord.Documents.Add

myWord.Documents.Save
```

```
myWord.Documents.Close wdDoNotSaveChanges

myWord.Quit
```

Якщо встановити контрольну точку для оператора `myWord.Documents.Save`, та внести зміни до відкритого та створеного документу, для створеного буде виведене діалогове вікно збереження, у якому необхідно задати ім'я документу та місце збереження, а відкритий документ буде збережений у файлі, з якого він відкритий.

- `Close()` – закриває усі документи колекції `Documents`, має серед інших необов'язковий параметр `SaveChanges`, який може приймати як значення елементи енумератора `WdSaveOptions`, аналогічно методу `Close()` класу `Application` (див. вище).

- `Item(index)` – дозволяє знайти потрібний документ в колекції `Documents` за його індексом `index`, який може бути або числом `Long`, або рядком `String`. Внаслідок зворотного порядку збереження документів за індексами, порівняно до порядку їх додання до `Documents`, частіше використовують останній тип даних. У наступному прикладі документи створюються та відкриваються у поточному екземплярі Microsoft Word (у якому відкритий документ з програмою):

```
Sub DocumentsMethods()
    Dim standardDoc As Word.Document
    Dim myDoc As Word.Document
    Dim openDoc As Word.Document
    Dim foundDoc As Word.Document

    Set openDoc = Documents.Open("E:\myDocument.docx")
    Set standardDoc = Documents.Add
    Set myDoc = Documents.Add(
           "c:\MyTemplates\LR-Otchet-Template-ESEI-ZNU.dotx")

    Set foundDoc = Documents.Item("myDocument.docx")

    Debug.Print foundDoc.FullName

    myWord.Documents.Close
    myWord.Quit
End Sub
```

Зазначимо, що можлива безпосередня передача аргументу `index` колекції `Documents`:

```
Set foundDoc = Documents("myDocument.docx")
```

Події для колекції `Documents` не передбачені.

Розглянуті вище методи `Add()`, `Open()`, `Item()` колекції `Documents` повертають посилання на об'єкт `Document`, з яким можна далі працювати. Окрім цих методів, отримати доступ до об'єкту `Document` можна

використовуючи властивість `ActiveDocument` об'єкта `Application` (якщо об'єкту `Document` обробляється у поточному екземплярі Microsoft Word, можна опустити `Application`), наприклад:

```
ActiveDocument.Name
```

або використовувати ключове слово `ThisDocument`, за допомогою якого можна отримати посилання на об'єкт документа, якому належить виконуваний програмний модуль, наприклад:

```
ThisDocument.Name
```

До *найважливіших властивостей класу* `Document` можна віднести:

- `Name` – містить ім'я файлу, у якому знаходиться документ (без шляху до нього) – доступна тільки для читання;
- `Path` – містить шлях у файловій системі до файлу, у якому знаходиться документ (без імені файлу) – доступна тільки для читання;
- `FullName` – містить шлях у файловій системі та ім'я файлу (повне ім'я), у якому знаходиться документ – доступна тільки для читання;
- `Type` – повертає тип документа (звичайний – `0` – `wdTypeDocument`, шаблон – `1` – `wdTypeTemplate` або сторінка з фреймами – `2` – `wdTypeFrameset`) – доступна тільки для читання;
- `SaveFormat` – дозволяє отримати інформацію про формат документа, підтримується 27 форматів, з яких основними є:
  - `16 - wdFormatDocumentDefault` – Документ Word (.docx) – значення за замовчуванням;
  - `0 - wdFormatDocument` – Документ Word 97-2003 (.doc);
  - `13 - wdFormatXMLDocumentMacroEnabled` – Документ Word з підтримкою роботи макросів (.docm);
  - `14 - wdFormatXMLTemplate` – Шаблон Word (.dotx);
  - `15 - wdFormatXMLTemplateMacroEnabled` – Шаблон Word з підтримкою роботи макросів (.dotm);
  - `1 - wdFormatTemplate` – Шаблон Word 97-2003 (.dot);
  - `17 - wdFormatPDF` – PDF;
  - `8 - wdFormatHTML` – Веб-сторінка;
  - `9 - wdFormatWebArchive` – Веб-сторінка в одному файлі;
  - `6 - wdFormatRTF` –Текст у форматі RTF;
  - `2 - wdFormatText` – Звичайний текст,

  доступна тільки для читання.
- `Content` – повертає об'єкт `Range`, що відповідає повному тексту документу (але без колонтитулів, коментарів тощо) – так званий *головний ланцюжок документа* (*main document story*);
- `Sections` – містить посилання на колекцію об'єктів `Section`, що відповідають розділам документа;

- Paragraphs – містить посилання на колекцію об'єктів Paragraph, що відповідають абзацам документа;

- Words – містить посилання на колекцію об'єктів Range, що відповідають словам документа;

- Characters – містить посилання на колекцію об'єктів Range, що відповідають символам документа;

- Tables – містить посилання на колекцію об'єктів Table, що відповідають таблицям документа;

- Shapes – містить посилання на колекцію об'єктів Shape, що відповідають таким рисункам документа, як, наприклад, автофігури, вставленим до документа за допомогою об'єкта Canvas;

- InlineShapes – містить посилання на колекцію об'єктів InlineShape, що відповідають рисункам, вставленим безпосередньо у текст документа;

- AttachedTemplate – містить посилання на шаблон, на базі якого створений поточний документ, дозволяє призначити для документу інший шаблон (для використання його стилів тощо);

- BuiltInDocumentProperties – містить посилання на колекцію DocumentProperties з парами "ім'я – значення", що відповідають властивостям документа (назва, автор, кількість сторінок, дата та час створення та останньої зміни тощо). Код доступу до цих властивостей:

```
Dim myDoc As Word.Document
Dim myDocProp As DocumentProperty

For Each myDocProp In myDoc.BuiltInDocumentProperties
    Debug.Print myDocProp.Name; " - "; myDocProp.Value
    On Error Resume Next
Next
```

До *найважливіших методів класу* *Document* можна віднести:

- Activate() – дозволяє зробити документ активним (використовується при роботі з декількома документами);

- Save() – зберігає документ, відкритий з файлу, для нового документу відкривається діалогове вікно збереження файлу;

- SaveAs2() – зберігає документ, метод має велику кількість оціональних параметрів, які дозволяють при збереженні документу задавати ім'я та місце збереження файлу (FileName), формат файлу (FileFormat – підтримуються формати, які доступні як значення властивості SaveFormat – див. вище), пароль для відкриття (Password) та запису (WritePassword) тощо, наведемо приклад оператора збереження документу у форматі *Portable Data Format – PDF*:

```
myDoc.SaveAs2 FileName:="e:\mydoc.pdf",
              FileFormat:=wdFormatPDF
```

- Close() – дозволяє закрити документ, необов'язковий параметр SaveChanges визначає необхідність збереження документу при закритті, може

мати ті ж значення, що однойменний метод колекції `Documents` (див. вище), наведемо приклад оператора, що закриває документ без збереження його змін:

```
myDoc.Close wdDoNotSaveChanges
```

- `FollowHyperlink()` – дозволяє відкрити гіпертекстову сторінку за вказаним у єдиному обов'язковому серед інших параметрі `Address` URL, наприклад:

```
myDoc.FollowHyperlink Address:="https://www.google.com"
```

- `GoTo()` – забезпечує навігацію (переміщення) по документу, повертаючи об'єкт `Range`, який використовується для подальшої обробки, має опціональні параметри:

  - `What` – об'єкт переходу документу, на певну кількість яких виконується переміщення – можливі об'єкти переходу наведені у Таблиця 9;
  - `Which` – спосіб переміщення до об'єктів переходу документу – можливі способи переміщення наведені у Таблиця 10;
  - `Count` – кількість об'єктів переходу (`Long`), на яку виконуватиметься переміщення (значення за замовчуванням дорівнює `1`);
  - `Name` – ім'я закладки, коментаря, поля або об'єкта (`String`), до якого виконуватиметься переміщення.

Таблиця 9 – Об'єкти переходу документу, які використовуються при навігації (елементи енумератора `WdGoToItem`)

Об'єкт переходу	Значення	Пояснення
wdGoToSection	0	Розділ
wdGoToPage	1	Сторінка
wdGoToLine	3	Рядок тексту
wdGoToBookmark	-1	Закладка
wdGoToComment	6	Коментар
wdGoToEndnote	5	Кінцева виноска
wdGoToEquation	10	Формула
wdGoToField	7	Поле
wdGoToFootnote	4	Виноска
wdGoToGrammaticalError	14	Граматична помилка
wdGoToGraphic	8	Рисунок
wdGoToHeading	11	Заголовок
wdGoToObject	9	Об'єкт
wdGoToPercent	12	Процент від розміру документу
wdGoToProofreadingError	15	Коректорська помилка
wdGoToSpellingError	13	Орфографічна помилка
wdGoToTable	2	Таблиця

Таблиця 10 – Способи переміщення до об'єктів переходу документу (елементи енумератора `WdGoToDirection`)

Спосіб переміщення	Значення	Пояснення
`wdGoToAbsolute`	1	Переміщення до об'єкту переходу з абсолютним значенням його порядкового номера
`wdGoToFirst`	1	Переміщення до першого об'єкту переходу
`wdGoToLast`	-1	Переміщення до останнього об'єкту переходу
`wdGoToNext`	2	Переміщення до наступного об'єкту переходу
`wdGoToPrevious`	3	Переміщення до попереднього об'єкту переходу
`wdGoToRelative`	2	Переміщення до об'єкту переходу, який знаходиться на вказану кількість об'єктів переходу попереду або позаду поточного

Наведемо приклад програми, яка виконує навігацію з використанням різних об'єктів переходу, для відображення переміщення використовується вставка тексту та/або виділення частини документу:

```
Sub DocumentGoToUsing()
    Dim myWord As Word.Application
    Dim openDoc As Word.Document
    Dim navRange As Range

    Set myWord = New Word.Application
    myWord.Visible = True
    myWord.Caption = "Програмно запущений Microsoft Word"

    Set openDoc = myWord.Documents.Open(
                                "e:\Документ-зразок.docx")
    'Переміщення до розділу 5
    Set navRange = openDoc.GoTo(wdGoToSection, _
                                wdGoToAbsolute, 5)
    navRange.Text = "ДОДАНИЙ ТЕКСТ" & vbCrLf
    navRange.Select

    'Переміщення на наступну (2) після початкової сторінку
    Set navRange = openDoc.GoTo(wdGoToPage, wdGoToNext)
    navRange.Text = "ДОДАНИЙ ТЕКСТ" & vbCrLf
    navRange.Select

    'Переміщення до рядка, на 4 рядки нижче першого
    Set navRange = openDoc.GoTo(What:=wdGoToLine, _
                   Which:=wdGoToRelative, Count:=3)
    navRange.Text = "ДОДАНИЙ ТЕКСТ" & vbCrLf
    navRange.Select

    'Переміщення до закладки з ім'ям BeforeQuadraticEquation
    Set navRange = openDoc.GoTo(What:=wdGoToBookmark, _
                            Name:="BeforeQuadraticEquation")
    navRange.Text = "ДОДАНИЙ ТЕКСТ" & vbCrLf
    navRange.Select
```

```
    'Переміщення до першої таблиці
    Set navRange = openDoc.GoTo(What:=wdGoToTable, _
                                Which:=wdGoToFirst)
    navRange.Select

    'Переміщення до третього рисунку
    Set navRange = openDoc.GoTo(What:=wdGoToGraphic, _
                                Count:=1)
    navRange.Select

    openDoc.Close wdDoNotSaveChanges
    myWord.Quit
End Sub
```

Зауважимо, що усі переміщення виконуються відносно початку документу, що не завжди зручно, тому метод `GoTo()` викликають з документу тільки як перший крок навігації, після чого використовують методи навігації об'єктів `Range` або `Selection`, які будуть розглянуті нижче.

- `PrintOut()` – виводить на друк весь документ або його частину, має велику кількість необов'язкових параметрів;
- `Range()` – повертає об'єкт `Range` – діапазон (частина) документу, може приймати як необов'язкові параметри номер початкового та кінцевого символів діапазону (роботу з `Range` буде описано нижче);
- `Select()` – виділяє увесь текст документу.

До *найбільш використовуваних подій класу* `Document` можна віднести наступні:

- `New` – виникає при створенні нового документу (є сенс використовувати тільки для шаблону – спрацьовує, коли на основі цього шаблону створюється новий документ);
- `Open` – виникає при відкритті існуючого документу;
- `Close()` – виникає при закритті документу.

Для використання обробників подій об'єкту `Document` необхідно його оголосити як об'єкт з подіями у модулі класу або форми, аналогічно об'єкту `Application` (див. вище). Але для документа існує ще одна можливість: для кожного документа у вікні *Провідника проєктів* Редактора Visual Basic відображається спеціальний ***модуль класу*** поточного документа ThisDocument, який містить значення властивостей поточного документу та у якому можна розміщувати обробники подій поточного документу (Рисунок 67).

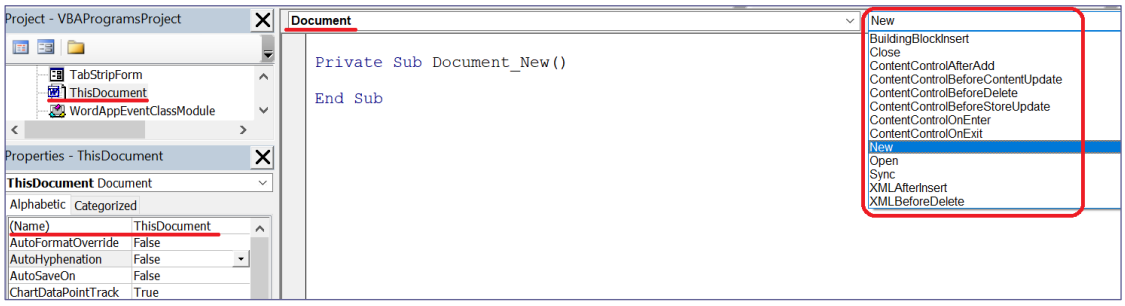


Рисунок 67 – Організація доступу к обробникам подій об'єкту `Document`

5.4 Робота з діапазонами (клас Range) та виділеними областями (клас Selection)

Об'єкти `Range` (діапазон) та `Selection` (виділена область) використовуються для навігації по документу та додання/редагування/видалення/форматування тексту або об'єктів документу [4]. Головна відмінність між цими об'єктами полягає у тому, що об'єкт `Range` можна визначити тільки програмно (він представляє безперервний фрагмент тексту документу, включаючи фрагмент нульової довжини, або доданий до документа об'єкт), він не залежить від поточного положення курсору або дій користувача і ніяк не відображається на екрані, а об'єкт `Selection` можна визначити і програмно, і користувачем (найчастіше встановленням курсору у певне місце документу або виділенням якогось фрагменту документа), і ми, звісно, бачимо це виділення на екрані. Також, одночасно може існувати скільки завгодно різних об'єктів `Range`, а об'єкт `Selection` може бути тільки один – створення нового об'єкту `Selection` призводить до видалення попереднього. Використання об'єктів `Range` унеможливлює негативний вплив переміщення курсору та виділення фрагментів документу користувачем на виконання програми. У той же час клас `Selection` надає більше методів для роботи і, у разі, коли дії користувача записуються у макрос, навігація та додання/редагування/видалення тексту або об'єктів документу виконуються за допомогою об'єкта `Selection` (Рисунок 3б).

Розглянемо основні способи створення об'єктів `Range` та `Selection`. *Об'єкт `Range` може бути створений*:

1) викликом методу `Range(Start, End)` об'єкта `Document`, при цьому як аргументи методу можуть бути передані номер (від початку документа) початкового символу створюваного діапазону (`Start`) та номер кінцевого символу (`End`). У разі, якщо параметри не зазначаються, створюваний об'єкт `Range` відповідає усьому документу. Наприклад, створити діапазон, який буде включати в себе перші `9` символів документа, можна так:

```
Dim myRange As Range

Set myRange = ActiveDocument.Range(0, 9)
```

2) використовуючи властивість `Range` класів `Selection`, `Section`, `Paragraph`, `Bookmark`, `Table`, `InlineShape`, тощо:

```
Dim myRange As Range

Set myRange = ActiveDocument.Paragraphs(1).Range
```

у цьому випадку ми отримуємо об'єкт `Range`, що описує відповідну частину документу;

На відміну від об'єкту `Range`, об'єкт `Selection` можна не оголошувати, оскільки він існує тільки один.

Об'єкт `Selection` *може бути створений:*

1) використанням властивості `Selection` класу `Application`, при цьому `Application` можна і не зазначати. Наприклад, для вставки тексту у місце, визначене користувачем шляхом встановлення курсору або виділення фрагменту документу, можна використати оператор:

```
Selection.Text = "текст, що додається"
```

Такий підхід не рекомендується, оскільки користувач може випадково виділити не той фрагмент або встановити курсор в інше, аніж передбачалося програмістом, місце.

2) отримуючи відповідний об'єкт `Range`, з якого викликають метод `Select()`, що повертає об'єкт `Selection`, наприклад:

```
ActiveDocument.Paragraphs(3).Range.Select
Selection.Text = "текст, що додається"
```

Звичайно, можна вставляти текст і без використання метода `Select()` і об'єкта `Selection` безпосереднім використанням властивості `Text` об'єкта `Range`:

```
ActiveDocument.Paragraphs(3).Range.Text = _
                                          "текст, що додається"
```

але у попередньому випадку при відлагодженні можна впевнитись, що обране правильне місце вставки.

Основні властивості класів `Range` *та* `Selection` співпадають і до них можна віднести:

- `Start` і `End` – визначають номер першого і останнього символу фрагменту документа відносно до початку документу, можуть бути змінені, що веде до зміни фрагменту, наприклад, після визначення як діапазону першого параграфу документу, діапазон буде зменшений на 1 символ у кінці:

```
Dim myRange As Range

Set myRange = ActiveDocument.Paragraphs(1).Range
myRange.End = myRange.End - 1
myRange.Select
```

Якщо значення `End` буде меншим, аніж значення `Start`, останнє виставляється у значення `End`. Якщо значення `Start` буде більшим, аніж значення `End`, останнє виставляється у значення `Start`;

- `Text` – дозволяє ввести текст замість визначеного/виділеного фрагмента (у тому числі нульової довжини) – приклади див. вище;

- `FormattedText` – повертає або встановлює поточному діапазону/виділеній області значення іншого діапазону/виділеної області, зберігаючи форматування останнього/останньої, наприклад, копіювання виділеної області з ознаками форматування у створюваний новий документ можна виконати таким чином:

```
Dim myRange As Range

Set myRange = Selection.FormattedText
Documents.Add.Content.FormattedText = myRange
```

- Font – містить посилання на об'єкт класу Font, за допомогою якого можна встановлювати властивості шрифту, наприклад, гарнітуру, розмір, написання та колір (останній за допомогою функції RGB – див. вище), наприклад:

```
Dim myFont As New Font

With myFont
    .Name = "Arial"
    .Size = 12
    .Bold = True
    .Color = RGB(255, 0, 0)
End With
ActiveDocument.Paragraphs(3).Range.Font = myFont
```

Зауважимо, що колір шрифту також може бути заданий як значення констант кольору VBA, які є основними кольорами палітр RGB, що використовується для формування кольору на екрані та CMYK, що використовується при формуванні кольору при друку: vbBlack, vbRed, vbGreen, vbYellow, vbBlue, vbMagenta, vbCyan, vbWhite.

Іншими найчастіше використовуваними властивостями об'єкту Font можна назвати Italic – курсивне написання, Underline – написання з підкресленням, Subscript – верхній індекс, Superscript – нижній індекс (всі вони активуються значенням True і деактивуються значенням False);

- ParagraphFormat – містить посилання на об'єкт класу ParagraphFormat, за допомогою якого можна встановлювати властивості абзацу, наприклад, вирівнювання, відступ першого рядка абзацу, міжрядковий інтервал, інтервали перед та після абзацу тощо:

```
Dim myFormat As New ParagraphFormat

With myFormat
    .Alignment = wdAlignParagraphLeft
    .FirstLineIndent = InchesToPoints(0.5)
    .LineSpacingRule = wdLineSpaceDouble
    .SpaceBefore = 36
    .SpaceAfter = 18
End With

ActiveDocument.Paragraphs(1).Range.ParagraphFormat = myFormat
```

У наведеному прикладі створюється об'єкт ParagraphFormat, якому призначаються ознаки форматування: вирівнювання абзацу по лівій межі тексту, відступ першого рядка абзацу у 0.5 дюйма, подвійний міжрядковий інтервал та відступи перед параграфом у 36 пунктів (1 пункт дорівнює 1/72 дюйма) та після параграфу у 18 пунктів. Далі цей об'єкт призначається однойменній властивості

об’єкта Range першого параграфу. Для властивості ParagraphFormat.Alignment також можливі найбільш використовувані значення: wdAlignParagraphRight, wdAlignParagraphCenter та wdAlignParagraphJustify (останнє вирівнює текст за шириною сторінки). Для властивості ParagraphFormat.LineSpacingRule також можливі найбільш використовувані значення: wdLineSpaceSingle, wdLineSpace1pt5 та wdLineSpaceMultiple (останнє вимагає використання додаткової властивості ParagraphFormat.LineSpacing з встановленим значенням у пунктах);

- Find – містить посилання на об'єкт класу Find з результатами пошуку у документі, детальніше про цей клас і пов'язаний з ним клас Replace буде викладено нижче у цьому розділі.

Об'єкт Selection також має властивість Range, що надає можливість з користувацького виділення створити об'єкт Range:

```
Dim myRange As Range

Set myRange = Selection.Range
```

та властивість HeaderFooter, що відповідає колонтитулам розділу документу (робота з колонтитулами буде описана нижче у цьому розділі).

Серед найважливіших методів класів Range та Selection можна виділити:

а) методи навігації можна поділити на такі, що повертають новий діапазон (навіть для поточної виділеної області) та такі, що переміщують або змінюють межі поточного діапазону/виділеної області:

- GoTo() – універсальний метод, що забезпечує навігацію (переміщення) по фрагменту, повертаючи об'єкт Range, який використовується для подальшої обробки, у разі виклику з об'єкту Selection – додатково встановлюючи місце вставки безпосередньо перед об'єктом переходу, що використовується методом як параметр. Параметри методу та їх можливі значення повністю співпадають з однойменним методом класу Document (див. вище);

- Next(Unit, Count) – повертає об'єкт Range, що знаходиться далі від поточного діапазону/виділеної області на зазначену параметром Count кількість одиниць даних, вказаних параметром Unit, – можливі одиниці даних наведені у Таблиця 11;

- Previous(Unit, Count) – повертає об'єкт Range, що знаходиться попереду від поточного діапазону/виділеної області на зазначену параметром Count кількість одиниць даних, вказаних параметром Unit, – можливі одиниці даних наведені у Таблиця 11;

- GoToNext(What) – повертає об'єкт Range, що відповідає наступному від поточного діапазону/виділеної області об'єкту переходу, зазначеному обов'язковим параметром What (можливі його значення наведені у Таблиця 9);

- GotoPrevious(What) – повертає об'єкт Range, що відповідає попередньому від поточного діапазону/виділеної області об'єкту переходу,

зазначеному обов'язковим параметром What (можливі його значення наведені у Таблиця 9);

- Move(Unit, Count) – дозволяє перевизначити початок та кінець поточного діапазону/виділеної області шляхом їх переміщення на зазначену параметром Count кількість зазначених параметром Unit одиниць даних (Таблиця 11);

- MoveEnd(Unit, Count) – дозволяє перевизначити кінець поточного діапазону/виділеної області, залишаючи незмінним початок, параметри методу такі ж, як у метода Move();

Таблиця 11 – Одиниці даних, які використовуються об'єктами Range та Selection для навігації (елементи енумератора WdUnits)

Одиниця даних	Значення	Пояснення
wdCell	12	Клітинка таблиці
wdCharacter	1	Символ
wdCharacterFormatting	13	Ознаки форматування символа
wdColumn	9	Стовпець таблиці
wdItem	16	Виділений елемент
wdLine	5	Рядок тексту
wdParagraph	4	Абзац
wdParagraphFormatting	14	Ознаки форматування абзацу
wdRow	10	Рядок таблиці
wdScreen	7	Значення розподільчої здатності екрану
wdSection	8	Розділ
wdSentence	3	Речення
wdStory	6	Ланцюжок документа
wdTable	15	Таблиця
wdWindow	11	Вікно
wdWord	2	Слово

Кольором виділені одиниці даних, що збігаються з об'єктами переходу (Таблиця 9).

- MoveStart(Unit, Count) – дозволяє перевизначити початок поточного діапазону/виділеної області, залишаючи незмінним кінець, параметри методу такі ж, як у метода Move();

- EndOf(Unit, Extend) – дозволяє перемістити позицію останнього символу поточного діапазону/виділеної області на кінець наступної одиниці даних, визначеної параметром Unit (Таблиця 11) та створити виділення до цього місця (Extend:=wdExtend), або встановити у це місце точку вставки (параметр Extend:= wdMove);

- StartOf(Unit, Extend) – дозволяє перемістити позицію першого символу поточного діапазону/виділеної області на початок попередньої одиниці даних, визначеної параметром Unit (Таблиця 11) та створити виділення до цього місця (Extend:=wdExtend), або встановити у це місце точку вставки (параметр Extend:= wdMove);

- MoveUntil(Cset, Count) – виконує стискання діапазону/виділеної області до позиції вставлення та переміщення до першого з послідовності символів, вказаної обов'язковим параметром Cset, але не далі, аніж на кількість символів, вказану опціональним параметром Count (якщо параметр позитивний, пошук ведеться вперед, якщо негативний – назад, якщо параметр не зазначається, пошук ведеться до кінця документа). Значення Count можуть дорівнювати константам wdForward – для пошуку до кінця документу (значення за замовчуванням) або wdBackward – для пошуку до початку документу, метод повертає число символів, на яке виконано переміщення: для Count > 0 – це число символів + 1, для Count < 0 – це число символів – 1, у разі, якщо жоден з символів Cset не знайдений, повертається 0 (переміщення не відбувається);

- Існують методи навігації, похідні від методу MoveUntil(Cset, Count): MoveStartUntil(Cset, Count) та MoveEndUntil(Cset, Count), які не стискають діапазон/виділену область, а переміщують їх початок або кінець до першого з послідовності символів Cset, параметр Count працює аналогічно такому ж параметру методу MoveUntil(Cset, Count);

- MoveWhile(Cset, Count) – стискання діапазону/виділеної області до позиції вставлення та переміщення до наступного за послідовністю символів, вказаною обов'язковим параметром Cset, у разі, якщо за діапазоном/виділеною областю зразу ж розміщена ця послідовність символів. Значення параметрів Cset та Count такі ж, як у метода MoveUntil(Cset, Count).

- Існують методи навігації, похідні від методу MoveWhile(Cset, Count): MoveStartWhile(Cset, Count) та MoveEndWhile(Cset, Count), які не стискають діапазон/виділену область, а переміщують їх початок або кінець до наступного за послідовністю символів, вказаною обов'язковим параметром Cset, у разі, якщо за діапазоном/виділеною областю зразу ж розміщена ця послідовність символів, параметр Count працює аналогічно такому ж параметру методу MoveWhile(Cset, Count);

б) <u>методи, що виконують зміну розміру діапазону/виділеної області:</u>

Як видно, декі методи навігації виконують зміну розміру діапазону/виділеної області, також існують окремі методи, які виконують аналогічні дії:

- SetRange(Start, End) – виконує зміну розміру поточного діапазону/виділеної області заданням послідовних від початку документу номерів першого і останнього символу нового діапазону/виділеної області, наведемо приклад його використання:

```
Dim MyRange As Range
Set MyRange = ActiveDocument.Range(Start:=0, End:=0)
MyRange.SetRange Start:=MyRange.Start, End:=Selection.End
```

- Collapse(Direction) – перетворює поточний діапазон/виділену область у місце вставлення, параметр Direction визначає напрям стискання діапазону/виділеної області: до початку (wdCollapseStart) – за замовчуванням або до кінця (wdCollapseStart);

- Expand(Unit) – дозволяє розширити поточний діапазон/виділену область на одиницю даних, зазначену параметром Unit (Таблиця 11), повертає кількість символів (типу Long), доданих до діапазону/виділеної області.

Наведемо код підпрограми, яка демонструє використання методів навігації для діапазонів та виділеної області, коментарі пояснюють використання методів (методи Select викликаються з діапазонів для візуальної демонстрації їх розміру):

```
Sub RangeSelectionNavigation()
    Dim myDoc As Word.Document
    Dim firstRange As Range, secondRange As Range

    Set myDoc = Documents.Open(ThisDocument.Path _
        & Application.PathSeparator & "Документ-зразок.docx")

    'Отримання першого діапазону з точкою вставки на початку
    'сторінки 2, додання тексту та вставка абзацу після нього
    Set firstRange = myDoc.GoTo(wdGoToPage, wdGoToAbsolute, 2)
    firstRange.Text = "ДОДАНИЙ ТЕКСТ 1"
    firstRange.InsertParagraphAfter
    firstRange.Select

    'Отримання другого діпазону, що знаходиться на 2 абзаци вище
    'поточного, додання тексту, отримання виділеної області
    'з діапазону, стиснення діапазону до місця вставки
    'та вставка абзацу
    Set secondRange = firstRange.Previous(wdParagraph, 2)
    secondRange.Select
    secondRange.InsertAfter "ДОДАНИЙ ТЕКСТ 2"
    secondRange.Select
    Selection.Collapse wdCollapseEnd
    Selection.TypeParagraph

    'Переміщення першого діапазону на 2 абзаци вгору
    'та додання тексту
    firstRange.Move wdParagraph, -2
    firstRange.Text = "ДОДАНИЙ ТЕКСТ 3"
    'Переміщення початку першого діапазону на 2 слова вправо
    'та вставка абзацу
    firstRange.MoveStart wdWord, 2
    firstRange.Select
    firstRange.InsertParagraphAfter
    firstRange.Select
    'Розширення виділення до початку розділу
    Selection.StartOf wdSection, wdExtend

    'Отримання другого діапазону для першого абзацу документа
    Set secondRange = myDoc.Paragraphs(1).Range
    secondRange.Select

    'Стискання діапазону до точки вставлення
    'та переміщення його до першого входження символів "КОМП"
```

```
    secondRange.MoveWhile ("КОМП")
    secondRange.Text = "ДОДАНИЙ ТЕКСТ 4"
    secondRange.Select

    'Розширення поточного діапазону до поточного абзацу
    '- повертає кількість символів на які розширений діапазон
    Debug.Print secondRange.Expand(wdParagraph)
    secondRange.Select

    'Стискання діапазону до точки вставлення та переміщення його
    'до першого входження одного з символів "Ш" або "Х"
    secondRange.MoveUntil "ШХ"
    secondRange.Text = "ДОДАНИЙ ТЕКСТ 5"
    secondRange.Select

    'Видаляє останній символ виділеної області
    Selection.Collapse wdCollapseEnd
    Selection.TypeBackspace

    myDoc.Close wdDoNotSaveChanges
End Sub
```

в) методи вставки/видалення тексту та об'єктів:

Класи `Range` та `Selection` мають методи вставки/видалення тексту та об'єктів, серед яких найчастіше використовуються:

- `InsertBefore(Text)` – виконує вставлення тексту, зазначеного параметром `Text`, перед діапазоном/виділеною областю, розширюючи їх на цей текст;
- `InsertAfter(Text)` – виконує вставлення тексту, зазначеного параметром `Text`, після діапазону/виділеної області, розширюючи їх на цей текст;
- `InsertBreak(Type)` – виконує вставлення розриву у залежності від опціонального параметру `Type`, найчастіше використовуваними значеннями якого є: `wdPageBreak - 7` – розрив сторінки, `wdLineBreak - 6` – розрив рядка, `wdColumnBreak - 8` – розрив стовпця, `wdSectionBreakNextPage - 2` – розрив розділу з переходом на нову сторінку, `wdSectionBreakContinuous - 3` – розрив розділу на поточній сторінці;
- `InsertDateTime` – виконує вставлення поточної дати та/або часу, серед інших має опціональні параметри: `DateTimeFormat` – зазначає формат дати/часу (Таблиця 4), `InsertAsField` – значення `True` (за замовчуванням) забезпечує вставлення дати/часу як поля, що може оновлюватись, значення `False` – як текстового рядка;
- `InsertParagraph` – виконує заміну поточного діапазону/виділеної області на порожній абзац, параметрів не має;
- `InsertParagraphAfter` – виконує вставку порожнього абзацу після поточного діапазону/виділеної області, параметрів не має;

- `InsertParagraphBefore` – виконує вставку порожнього абзацу перед поточним діапазоном/виділеною областю, параметрів не має;

- `InsertSymbol(CharacterNumber, Font, Unicode, Bias)` – виконує вставку символу замість поточного діапазону/виділеної області, має один обов'язковий параметр `CharacterNumber` – код символу в таблиці символів відповідного шрифту, зазначеного параметром `Font`. У разі значення `False` параметру `Unicode` (за замовчуванням) символи беруться з ASCII таблиць шрифтів і максимальний код символу дорівнює `255`, у разі значення `True` параметру `Unicode` можна використовувати Unicode шрифти з відповідними десятковими значеннями кодів символів;

- `Delete(Unit, Count)` – видаляє вказану параметром `Count` кількість одиниць даних (Таблиця 11), вказаних параметром `Unit`.

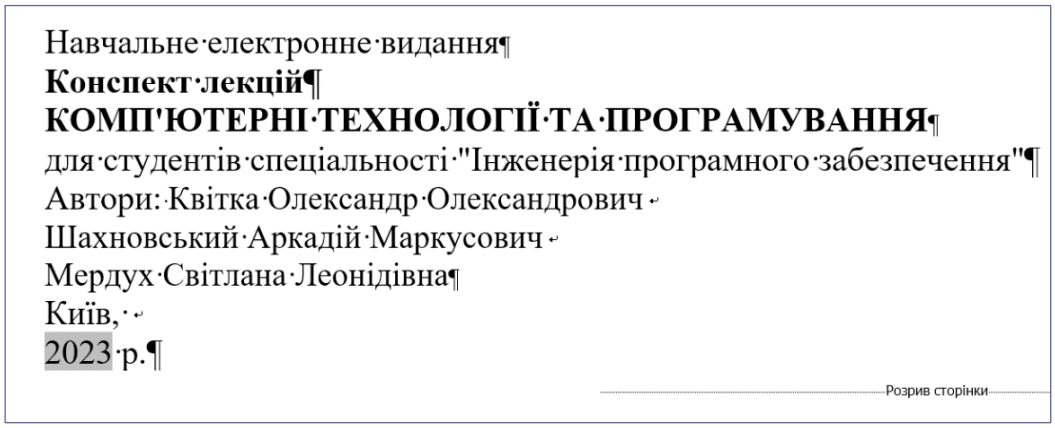
Попередній приклад програми для додання тексту до документу використовував властивість `Text` об'єкту `Range` (див. вище), насправді існують більш зручні методи для виконання цієї дії. Наведемо код підпрограми, яка демонструє використання методів вставки тексту для діапазону, коментарі пояснюють використання методів (методи `Select` викликаються з діапазонів для візуальної демонстрації їх розміру):

```
Sub RangeSelectionInsert()
    Dim myDoc As Word.Document
    Dim myRange As Range

    Set myDoc = Documents.Open(ActiveDocument.Path _
            & Application.PathSeparator & "Документ-
зразок.docx")

    'Додання тексту перед та після визначеного діапазону
    Set myRange = myDoc.Paragraphs(2).Range
    With myRange
        .Select
        .InsertParagraphBefore
        .InsertBefore "Конспект лекцій"
        .InsertAfter "для студентів спеціальності " _
                  & """Інженерія програмного забезпечення"""
        .Select
        .InsertParagraphAfter

        'Вставка розриву сторінки для перенесення
        'року видання на новий рядок
        .Move wdParagraph, 2
        .Select
        .Move wdWord, 2
        .InsertBreak wdLineBreak

        'Заміна значення року на поточний
        .Expand wdWord
        .Select
        .InsertSymbol 32
```

```
            .InsertDateTime "yyyy"
        End With

        'Видалення рисунку
        myRange.Select
        Selection.GoTo wdGoToGraphic, wdGoToAbsolute, 2
        Selection.Delete

        myDoc.Close wdDoNotSaveChanges
    End Sub
```

На Рисунок 68 наведений вигляд документу до та після виконання підпрограми `RangeSelectionInsert()`.

а)



б)

Рисунок 68 – Вигляд документа: а) до та б) після виконання програми вставлення тексту

г) <u>методи роботи з Буфером обміну:</u>

Класи `Range` та `Selection` мають методи роботи з Буфером обміну:

- `Copy()` – копіює поточний діапазон/виділену область до Буферу обміну;
- `Cut()` – видаляє поточний діапазон/виділену область до Буферу обміну;
- `Paste()` – виконує вставку з Буферу обміну до поточного діапазону/виділеної області, замінюючи його;

Наведемо код підпрограми, яка демонструє використання методів роботи з Буфером обміну для діапазону та конвертування діапазону до виділеної області та навпаки, коментарі пояснюють використання методів:

```
Sub SelectionCopyPaste()
    Dim myDoc As Word.Document
    Dim myRange As Range

    Debug.Print "Опція ""Заміняти виділений текст"" " _
        & "встановлена у " & Options.ReplaceSelection

    Set myDoc = Documents.Open(ActiveDocument.Path _
            & Application.PathSeparator & "Документ-зразок.docx")

    Set myRange = myDoc.Paragraphs(2).Range
    'Отримання виділенної області з діапазону
    myRange.Select

    With Selection
        'Копіювання-вставка тексту
        .Copy
        .Move wdParagraph, 3
        .InsertBreak wdPageBreak
        .Paste

        'Вирізання-вставка тексту
        .Move wdSection, -1
        .EndKey Unit:=wdLine, Extend:=wdExtend
        .Cut
        .Move wdParagraph, 5
        .Paste

        .GoTo wdGoToPage, wdGoToFirst
        .Expand wdLine
    End With

    'Отримання діапазону з виділенної області
    Set myRange = Selection.Range
    Debug.Print myRange

    myDoc.Close wdDoNotSaveChanges
End Sub
```

На Рисунок 69 наведений вигляд документу до та після виконання підпрограми `SelectionCopyPaste()`.

Рисунок 69 – Вигляд документа після виконання програми роботи з Буфером обміну

д) інші методи:

Інші методи, загальні для класів `Range` та `Selection`:

- `Select()` – надає можливість створити об'єкт `Selection` для поточного об'єкта `Range` або виділяє фрагмент документа, що входить до поточного об'єкта `Selection` – це часто використовується для візуалізації об'єктів `Range` та `Selection`;

- `ClearFormatting()` – видаляє форматування (шрифту та абзаців) поточного діапазону/виділеної області;

- `Calculate()` – надає можливість виконати арифметичні обчислення, записані як текст у поточному діапазоні/виділеній області з поверненням результату як значення типу `Single` до поточного діапазону/виділеної області (результати правильні тільки для операцій +, -, *, / та ^, можна використовувати дужки):

```
Dim myRange As Range

Set myRange = ActiveDocument.Range(0, 0)
myRange.InsertBefore "(8 + 2/4 - 2^3) * 2"
myRange.InsertAfter " = " & myRange.Calculate
```

- методи `Sort()`, `SortAscending()`, `SortDescending()` – надають можливість сортування абзаців та даних таблиць, яким відповідає поточний діапазон/виділена область. Враховуючи, що найчастіше ці методи використовуються для сортування даних таблиць, розглянемо їх разом з об'єктом `Table` нижче у цьому розділі;

- `WholeStory()` – дозволяє виділити увесь текст документа (*main document story*), у наведеному прикладі довільний діапазон розширюється на весь документ і у ньому змінюється шрифт:

```
Dim myRange As Range

Set myRange = ActiveDocument.Range(0, 0)
myRange.WholeStory
myRange.Font.Name = "Arial"
```

е) унікальні методи класу `Range`:

Клас `Range` також має методи, які відсутні у класі `Selection`, серед яких найважливішим можна назвати:

- `Relocate(Direction)` – надає можливість переставити місцями поточний діапазон з наступним (`Direction:=wdRelocateDown`) або попереднім (`Direction:=wdRelocateUp`) абзацами.

ж) унікальні методи класу `Selection`:

Клас `Selection` також має методи, які відсутні у класі `Range`, серед яких найважливіші:

- методи навігації, похідні від методу `Move()`: `MoveLeft(Unit, Count, Extend)`, `MoveRight(Unit, Count, Extend)`, `MoveUp(Unit, Count, Extend)`, `MoveDown(Unit, Count, Extend)`, які дозволяють переміщувати виділену

область на зазначену кількість зазначених одиниць даних вліво, вправо, вгору та вниз, відповідно. Параметр `Extend` дозволяє або розширити виділену область до цього місця (`Extend:=wdExtend`), або встановити у це місце точку вставки (`Extend:=wdMove`);

- Методи `SelectCell()`, `SelectRow()` та `SelectColumn()` – вконують виділення клітинки, рядка та стовпця таблиці, які містять поточний об'єкт `Selection` – часто використовуються при роботі з таблицями об'єктами класу `Table` (див. нижче);

- Методи `SelectCurrentAlignment()`, `SelectCurrentColor()`, `SelectCurrentFont()`, `SelectCurrentIndent()`, `SelectCurrentSpacing()`, `SelectCurrentTabs` – дозволяють виділити фрагмент документа, який має однакові з поточною виділеною областю відповідні ознаки форматування до місця зміни будь якої з ознак для вирівнювання, шрифту, відступу, зміни кольору тощо;

- `EndKey(Unit, Extend)` – дозволяє при значенні параметра `Extend:=wdExtend` розширити поточну виділену область, перемістивши її кінець на вказану параметром `Unit` одиницю даних (Таблиця 11), або при значенні параметра `Extend:= wdMove` перейти на кінець наступної за виділеною областю одиницю даних, вказану параметром `Unit`, наприклад, для переміщення у кінець документу:

```
Selection.EndKey Unit:=wdStory, Extend:=wdMove
```

- `HomeKey(Unit, Extend)` – дозволяє при значенні параметра `Extend:=wdExtend` розширити поточну виділену область, перемістивши її початок на вказану параметром `Unit` одиницю даних (Таблиця 11), або при значенні параметра `Extend:= wdMove` перейти на початок попередньої перед виділеною областю одиницю даних, вказану параметром `Unit`;

- `TypeParagraph()` – виконує вставлення абзацу замість поточного об'єкта `Selection`, аналогічно методу `InsertParagraph()` (див. вище);

- `TypeText(Text)` – виконує вставку параметру-рядка `Text`. У разі, якщо значення налаштування `Options.ReplaceSelection=True` (значення за замовчуванням, виставлене в *Файл–Параметри–Додатково–Параметри редагування–Замінити виділений текст*), вставка виконується замість поточного об'єкта `Selection`, якщо `Options.ReplaceSelection=False`, виконується вставка перед поточним об'єктом `Selection`;

- `TypeBackspace()` – видаляє символ перед поточним об'єктом `Selection` (у стиснутому до місця вставки стані).

Для об'єктів `Range` та `Selection` генерування подій не передбачено.

5.5 Робота з закладками (клас Bookmark) та колонтитулами (клас HeaderFooter)

Як ми побачили, Microsoft Word API надає багато гнучких методів для навігації по документу, але є суттєва проблема: у разі редагування документу користувачем змінюються координати розділів, сторінок, параграфів тощо, і прив'язаний до них код буде працювати неправильно (звісно, це не стосується програмного редагування, яке може врахувати відповідні зміни координат). Вирішенням цієї проблеми є робота з *закладками – об'єктами* `Bookmark`, що являють собою додані у необхідних місцях документу іменовані об'єкти, до яких можна додавати необхідний текст та об'єкти. Замість обрахування координат необхідного місця вставки, підраховуючи порядкові номери розділів, сторінок, параграфів, речень або слів у параграфі тощо, можна у це місце засобами Microsoft Word додати закладку з визначеним ім'ям і в подальшому звертатися до неї (у тому числі і програмно) для вставлення/редагування/видалення тексту або об'єктів, що є значно зручнішим. Також, закладки зберігаються у документі (або шаблоні) і можуть бути використані у наступних сеансах роботи з ним, у той час, як об’єкти `Selection` і `Range` існують тільки в пам'яті і після закриття документа видаляються [4].

Для додання закладки до документу необхідно виділити місце у документі, де буде додана закладка та виконати команду *Вставлення–Посилання–Закладка*. Після цього у вікні *Закладка* необхідно додати ім'я закладки (рекомендуємо використовувати латинські літери, пробіли не допускаються) та виконати клік по кнопці *Додати* (Рисунок 70).

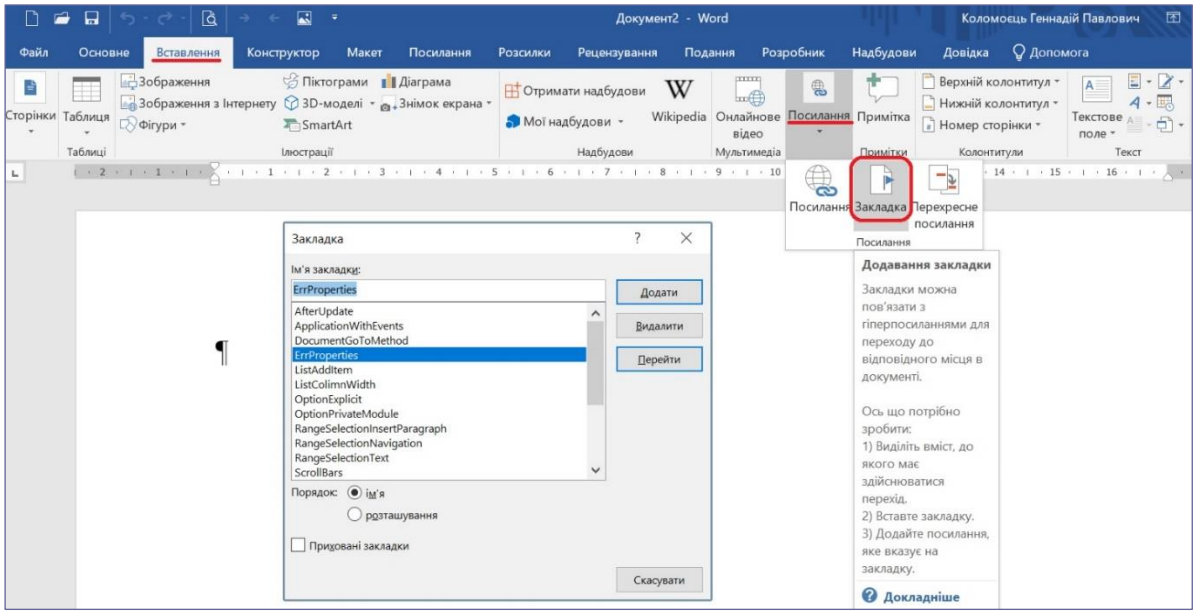


Рисунок 70 – Вставка закладок

Досить зручним є відображення закладок у документі, яке можна включити командою *Файл–Параметри–Додатково–Відображення вмісту документа–Відображати закладки* (Рисунок 71).

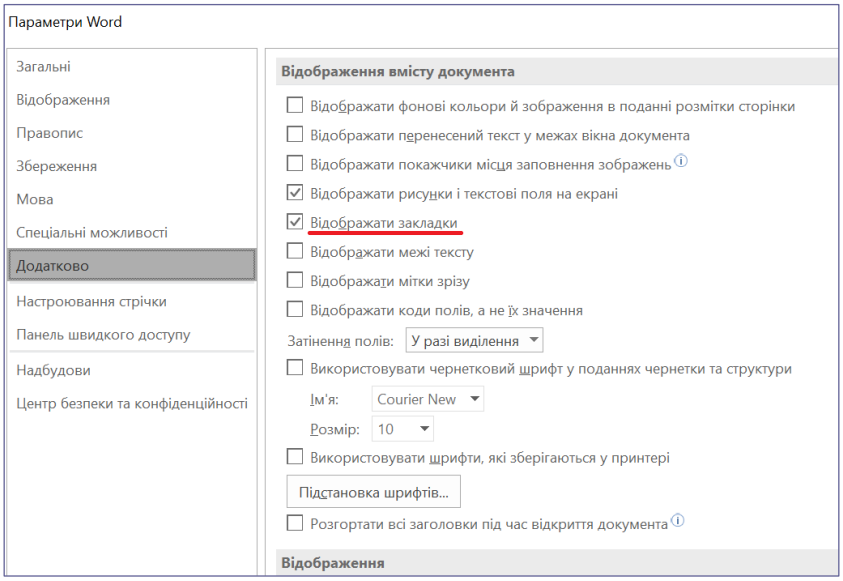


Рисунок 71 – Налаштування відображення закладок

Після встановлення цієї опції можна побачити закладки у документі або шаблоні (Рисунок 72).

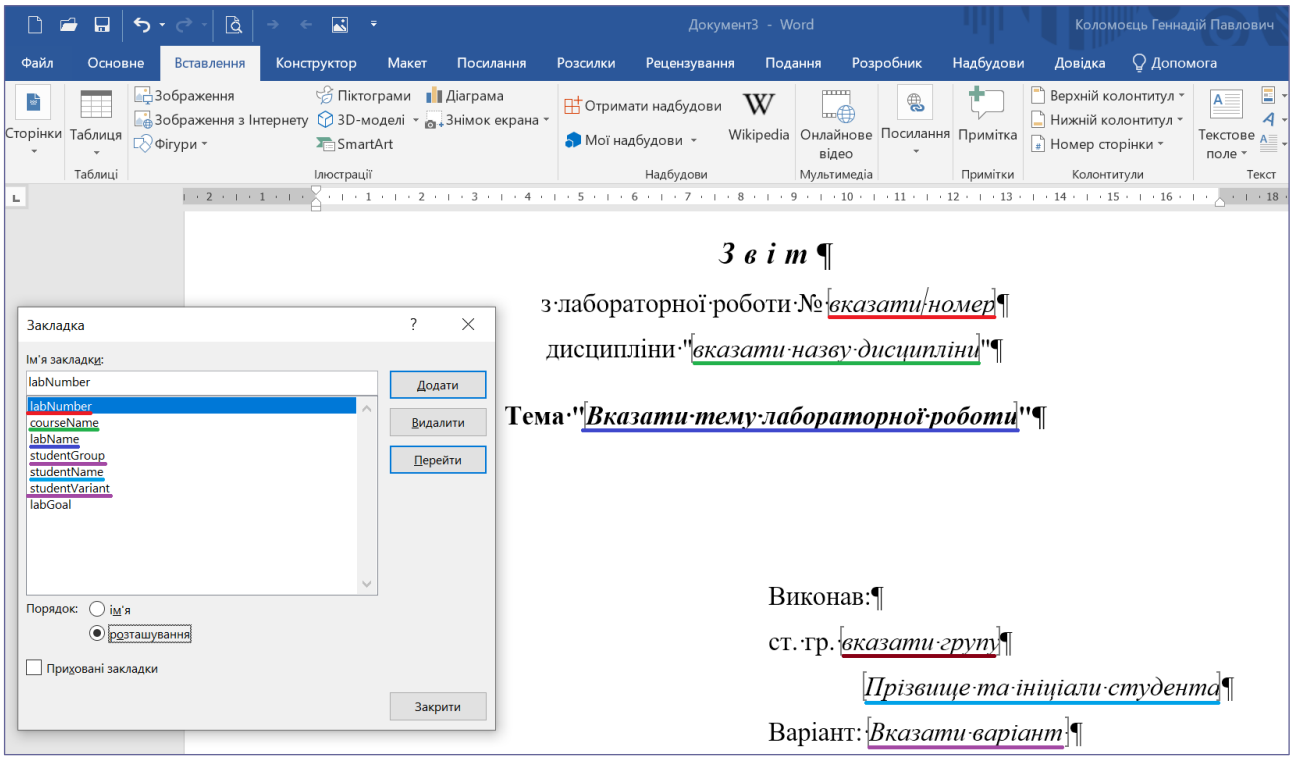


Рисунок 72 – Шаблон з закладками

Властивостей і методів у об'єкта `Bookmark` набагато менше, аніж у об'єктів `Range` та `Selection`. Однак, зазвичай, немає потреби використовувати об'єкт `Bookmark` для роботи з текстом безпосередньо. Об'єкт `Bookmark` має властивість `Range`, що посилається на діапазон з вмістом заклади, та метод `Select()`, що дозволяє отримати виділену область з вмістом закладки. Отримавши, таким чином, об'єкт `Range` або об'єкт `Selection`, можна використати їх властивості та методи, наприклад:

```
Dim myRange As Range

Set myRange = ActiveDocument.Bookmarks("Bookmark1").Range
myRange.Text = "ДОДАНИЙ ДО ЗАКЛАДКИ ТЕКСТ"
```

З об'єктами Bookmark пов'язана колекція Bookmarks, яка є властивістю таких класів як Document, Range, Selection тощо (Рисунок 62).

Із найчастіше використовуваних *властивостей колекції Bookmarks* назвемо тільки Count – кількість створених закладок (властивість типу Long, тільки для читання).

Щодо найважливіших *методів колекції Bookmarks*, то це:

- Add(Name, Range) – додає закладку, має обов'язковий параметр Name, що визначає ім'я закладки та необов'язковий параметр Range, що визначає діапазон (може бути точкою вставлення), до якого прив'язується закладка, у разі невикористання Range, закладка прив'язується до поточного діапазону. Додати програмним способом закладку можна наступним чином:

```
ActiveDocument.Bookmarks.Add "FirstBookmark", myRange
```

- Exists(Name) – перевіряє наявність закладки з ім'ям Name, метод повертає True у разі наявності закладки та False – у протилежному випадку:

```
Debug.Print ActiveDocument.Bookmarks.Exists("FirstBookmark")
```

Item(index) – дозволяє знайти потрібний документ в колекції Bookmarks за його індексом index, який може бути або числом Long, або рядком String, зауважимо, що посилання на конкретну закладку може бути як за її індексом, так і за її ім'ям (останнє – зручніше):

```
Debug.Print ActiveDocument.Bookmarks(1)
Debug.Print ActiveDocument.Bookmarks("FirstBookmark")
```

Найважливішими *властивостями об'єкта Bookmark* можна назвати:

- Name – ім'я закладки – властивість тільки для читання;
- Range – повертає об'єкт Range, який відповідає вмісту закладки;
- Empty – повертає True у разі, якщо закладка не має вмісту (при цьому вона ж точкою вставлення) – властивість тільки для читання;
- Start – містить позицію початкового символу вмісту закладки (тип Long), починаючи від початку документу;
- End – містить позицію кінцевого символу вмісту закладки (тип Long), починаючи від початку документу.

Методів у об'єкта Bookmark всього три:

- Copy(Name) – копіює поточну закладку у нову з ім'ям, вказаним обов'язковим параметром Name;
- Delete() – видаляє поточну закладку;
- Select() – виділяє вміст закладки.

Наступна програма демонструє створення документу на основі шаблону з закладками та заповнення закладок у документі даними з масиву (також

демонструється робота з властивостями закладки), на Рисунок 73 показаний результат її роботи.

```
Sub BookmarkUsing()
    Dim myDoc As Word.Document
    Dim lrData As Variant
    Dim myBookmark As Bookmark
    Dim i As Long
    Dim bkmkRange As Range, bkmkName As String

    lrData = Array("1", "Офісне програмування",
          "Створення простого застосунку на VBA та виконання
          арифметичних операцій", "6.1210-пзс-1", "Петренко М.І.",
          "7", "Опанувати основи роботи з VBA")

    Set myDoc = Documents.Add("e\LabWork-Report-Template.dotx")

    'Індекс масиву VBA за замовчуваннм починається з 0
    For i = 0 To myDoc.Bookmarks.Count - 1
        'Індекс колекцій VBA починається з 1
        Set bkmkRange = myDoc.Range.Bookmarks(i + 1).Range
        bkmkName = myDoc.Range.Bookmarks(i + 1).Name
        Debug.Print bkmkName; "-"; bkmkRange
        bkmkRange.Text = lrData(i)
        'Необхідно повторно вставити закладку, оскільки оператор
        'bkmkRange.Text її видаляє
        myDoc.Bookmarks.Add bkmkName, bkmkRange
        Debug.Print "Count="; myDoc.Bookmarks.Count
    Next i

    'Виводить дані закладки з ім'ям "courseName"
    If (myDoc.Bookmarks.Exists("courseName")) Then
        Set myBookmark = myDoc.Bookmarks("courseName")
        Debug.Print myBookmark.Name & ": " & myBookmark.Start _
                & " - " & myBookmark.End
    End If

    myDoc.Close wdDoNotSaveChanges
End Sub
```

Зверніть увагу, що додання тексту до закладки за допомогою властивості `Text` об'єкту `Range` виконується замість закладки і, таким чином, закладка видаляється. Тому для правильної роботи програми потрібно ще раз додати видалену закладку до колекції `Bookmarks`.

Як і для класів `Range` та `Selection`, події для класу `Bookmark` не передбачені.

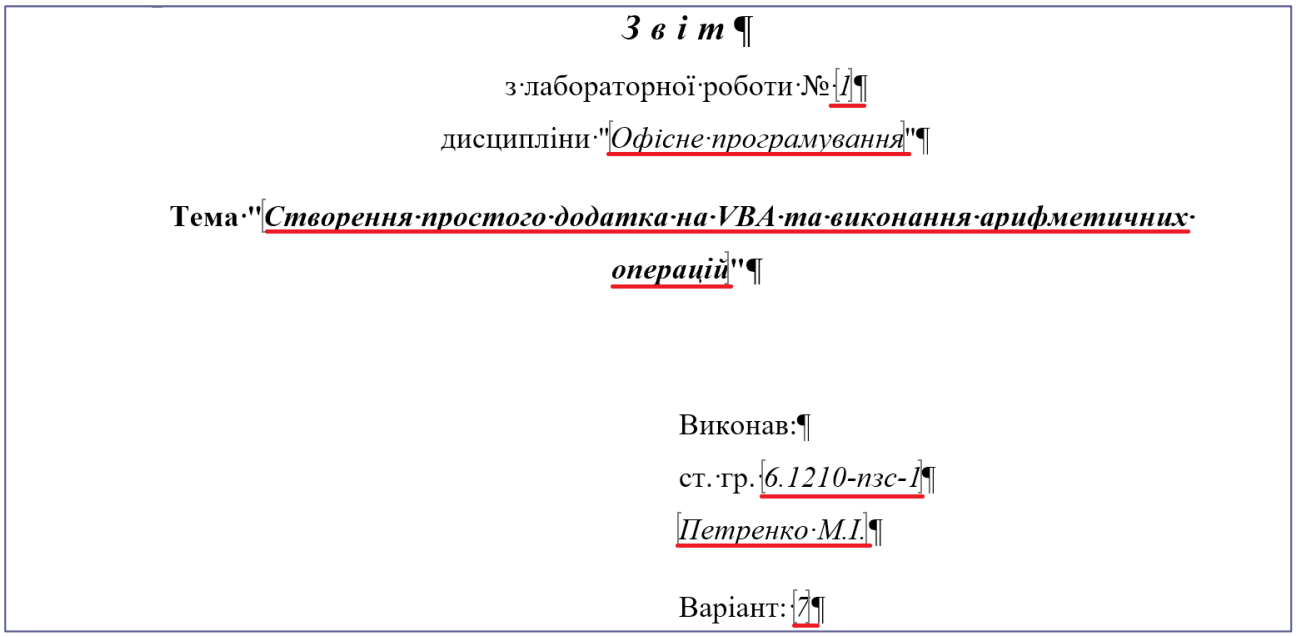


Рисунок 73 – Створений на основі шаблону документ з заповненими закладками

Розглянемо засоби роботи з верхніми та нижніми колонтитулами, які надає об'єктна модель Microsoft Word. Враховуючи, що колонтитули можуть незалежно налаштовуватись для різних розділів документу, у класі `Section`, що відповідає розділу, передбачені дві колекції `Footers` та `Headers`, які містять об'єкти класу `HeaderFooter`, що відповідають колонтитулами розділу документу. Кожна з колекцій приймає як параметр індекс, що може мати значення елемента енумератора `WdHeaderFooterIndex`:

- `wdHeaderFooterPrimary – 1` – налаштовує однакові колонтитули для усіх сторінок розділу, окрім першої;
- `wdHeaderFooterFirstPage – 2` – налаштовує колонтитул для першої сторінки розділу, якщо властивості `DifferentFirstPageHeaderFooter` об'єкту класу `PageSetup` (який у свою чергу є властивістю класу `Document`) встановлено значення `True` – альтернативно може бути застосований параметр `FirstPage` методу `Add()` колекції `PageNumbers` (див. нижче);
- `wdHeaderFooterEvenPages – 3` – налаштовує однакові колонтитули для усіх сторінок розділу з парними послідовними номерами, якщо властивості `OddAndEvenPagesHeaderFooter` об'єкту класу `PageSetup` (який у свою чергу є властивістю класу `Document`) встановлено значення `True` (для налаштування сторінок розділу з непарними послідовними номерами у цьому випадку використовується колекція з індексом `wdHeaderFooterPrimary`).

Клас `HeaderFooter`, що відповідає колонтитулам, має тільки *властивості, найважливіші з яких*:

- `Index` – містить тип колонтитула, зазначений елементом енумератора `WdHeaderFooterIndex` (тільки для читання);

- IsHeader – містить True, якщо поточний об'єкт HeaderFooter є верхнім колонтитулом, та False – якщо нижнім (тільки для читання);

- PageNumbers – повертає посилання на колекцію PageNumbers з об'єктами PageNumber, що відповідають за нумерацію сторінок у поточному колонтитулі, ця колекція має *корисні властивості*:

  - StartingNumber – дозволяє задати початковий номер для нумерації сторінок;
  - ShowFirstPageNumber – у разі встановлення значення False не виводить номер на першій сторінці розділу з поточним колонтитулом;
  - RestartNumberingAtSection – у разі встановлення значення True починає нумерацію знову з 1 у колонтитулі поточної секції;
  - LinkToPrevious – у разі значення True, поточний колонтитул буде пов'язаний з відповідним колонтитулом попереднього розділу, наприклад, буде продовжена нумерація сторінок;

- Range – повертає діапазон для поточного колонтитула.

та метод Add (PageNumberAlignment, FirstPage) – для поточного розділу додає нумерацію у визначений колонтитул, приймає як параметри налаштування вирівнювання номеру сторінки PageNumberAlignment – елемент енумератора WdPageNumberAlignment (найчастіше використовують: wdAlignPageNumberLeft - 0 – вирівнювання по лівій межі, wdAlignPageNumberCenter - 1 – вирівнювання по центру та wdAlignPageNumberRight - 2 – вирівнювання по правій межі – за замовчуванням) та ознаки (True/False), чи необхідно виводити номер на першій сторінці розділу з поточним колонтитулом.

Наведемо код програми, яка встановлює у верхньому колонтитулі усіх сторінок першого розділу документу текст "Електронне видання", а у нижньому колонтитулі – номери сторінок без виведення номера на першій сторінці:

```
Sub HeaderFooterUsing()
    Dim myDoc As Word.Document

    Set myDoc = Documents.Open(ActiveDocument.Path _
            & Application.PathSeparator & "Документ-зразок.docx")

    With myDoc.Sections(1)
        .Headers(wdHeaderFooterPrimary).Range.Text = _
                                            & "Електронне видання"
        .Headers(wdHeaderFooterFirstPage).Range.Text = _
                                            & "Електронне видання"
        .Footers(wdHeaderFooterPrimary).PageNumbers.Add _
                                                FirstPage:=False
    End With

    myDoc.Close wdDoNotSaveChanges
End Sub
```

5.6 Пошук та заміна фрагментів документу (класи Find та Replacement)

У разі, коли документ коригується користувачем, прив'язатися до певних місць документу, окрім засобів класу `Bookmark`, можна за допомогою пошуку фрагментів, які напевне не змінюватимуться. Такий пошук забезпечує клас `Find`, об'єкти якого містять критерії пошуку, які можуть включати текст або ознаки форматування [4]. Класи `Range` та `Selection` мають властивість `Find`, яка може містити посилання на об'єкти класу `Find` з критеріями пошуку у відповідному діапазоні або виділеній області (Рисунок 62).

До *найважливіших властивостей класу `Find`* можна віднести:

- `Text` – містить рядок, який буде шукатися;
- `Font` – містить посилання на об'єкт `Font` з певними налаштуваннями шрифту, що використовуватиметься для пошуку;
- `ParagraphFormat` – містить посилання на об'єкт `ParagraphFormat` з певними налаштуваннями абзацу, що використовуватиметься для пошуку;
- `Style` – містить ім'я стилю, який використовуватиметься для пошуку;
- `LanguageID` – містить елемент енумератора `WdLanguageID`, що вказує на мову написання фрагментів тексту, шо шукатимуться (найпоширеніші значення елементів: `wdUkrainian` – `1058` – українська мова, `wdEnglishUS` – `1033` – англійська мова, значення інших мов – див. [26]);
- `Found` – повертає `True` у разі, якщо знайдений збіг з шаблоном пошуку, (властивість тільки для читання);
- `MatchCase` – у разі значення `True` виконує пошук з врахуванням збігу регістру символів (значення за замовчуванням – `False`).
- `Replacement` – містить посилання на об'єкт `Replacement` з критеріями заміни знайденого фрагменту документа.

Серед *найважливіших методів класу `Find`* можна назвати:

- `Execute()` – ініціює операцію пошуку, повертає `True` у разі знайдення збігу, має велику кількість опціональних параметрів, більшість яких встановлює зазначені вище властивості об'єкту `Find`, що формують критерій пошуку, наприклад:
  - `FindText` – встановлює властивість `Text`, у разі пошуку виключно за ознаками форматування, цей параметр встановлюється у пустий рядок `""`;
  - `MatchCase` – встановлює властивість `MatchCase`, у разі значення `False` та використання заміни тексту у верхньому регістрі, він буде замінений текстом також у верхньому регістрі, у разі значення `True` – він буде замінений символами у регістрі, що відповідає тексту, що заміняє знайдений;
  - `Format` – у разі значення `True`, виконує пошук лише за ознаками форматування при `FindText:=""` або за ознаками форматування та текстом пошуку, коли він зазначений, у разі значення `True` ознаки форматування при пошуку не враховуються;

- `ReplaceWith` – містить текст, яким буде замінюватись знайдений;
- `Replace` – визначає кількість замін знайдених фрагментів тексту, містить елемент енумератора `WdReplace`: `wdReplaceAll – 2` – виконує заміну всіх входжень знайдених фрагментів, `wdReplaceOne – 1` – виконує заміну першого знайденого фрагменту, `wdReplaceNone – 0` – не виконує заміну знайдених фрагментів.

- `ClearFormatting()` – видаляє ознаки форматування тексту та абзацу з рядка-критерія пошуку (є альтернативним застосуванню параметру `Format` метода `Execute()`).

Для організації пошуку тексту необхідно:

1) визначити діапазон/виділену область документу, у якій буде виконуватися пошук та отримати відповідний об'єкт `Range` або `Selection`;

2) отримати для визначеного діапазону/виділеної області об'єкт `Find`;

3) задати значення властивостей об'єкту `Find`, що визначають критерії пошуку (альтернативно вони можуть бути зазначені як аргументи методу `Execute()`);

4) викликати метод `Execute()` об’єкта `Find` та перевірити результат, що він повертає – у разі значення `True`, збіг тексту, що шукається у визначеному діапазоні/виділеній області документу знайдений, повторні виклики цього методу будуть виконувати пошук далі по тексту.

Наведемо приклад програми, що виконує пошук текстового рядка у документі та підраховує кількість його входжень:

```
Sub FindUsing()
    Dim myDoc As Word.Document
    Dim myRange As Range
    Dim textFind As String
    Dim count As Integer

    Set myDoc = Documents.Open(ActiveDocument.Path _
            & Application.PathSeparator & "Документ-зразок.docx")
    count = -1
    Set myRange = myDoc.Content
    textFind = "програмування"

    With myRange.Find
        .Text = textFind
'        .MatchCase = True
        Do
            count = count + 1
            .Execute
        Loop While .Found

        If count > 0 Then
            Debug.Print "Текст "; Chr(34); textFind; Chr(34); _
                        " знайдений "; count; " разів."
        Else
            Debug.Print "Текст "; Chr(34); textFind; Chr(34); _
```

```
                        " не знайдений."
        End If
    End With

    myDoc.Close wdDoNotSaveChanges
End Sub
```

Запуск програми виведе:

Текст "програмування" знайдений 20 разів.

Якщо ж замінити значення змінної `textFind` на `"КИЇВ"` (у документі слово `"Київ"` – саме у вказаному регістрі – зустрічається тільки один раз), то пошук поверне один збіг, але при розкоментуванні оператора встановлення значення властивості `MatchCase` об'єкту `Find` у `True`, збігів знайдено не буде.

Окрім кількості входжень рядка, що шукається, можна отримати його координати у тексті – порядковий номер першого символу, наведемо фрагмент коду, який це реалізує – організуємо пошук з використанням параметрів метода `Execute()`:

```
Set myRange = myDoc.Content
With myRange.Find
     Do
         .Execute FindText:=textFind, MatchCase:=True, _
                  Format:=False
        If .Found Then
            myRange.MoveStartUntil Cset:=textFind
            Debug.Print "Текст "; Chr(34); textFind; Chr(34); _
                        " знайдений на позиції " _
                    & myRange.Start + 1
        End If
    Loop While .Found
End With
```

Програма буде виводити на екран порядкові номери першого символу усіх знайдених фрагментів тексту. Зауважимо, якщо пошук для визначеного діапазону/виділеної області пройшов до їх кінця і необхідно повторити пошук (можливо, за іншими критеріями), слід повторно отримати посилання на цей діапазон/виділену область – `Set myRange = myDoc.Content` у наведеному прикладі.

Наведемо також приклад пошуку фрагментів тексту за ознаками форматування – назвою застосованої гарнітури шрифту та його розміру:

```
     With ActiveDocument.Content.Find
         .Font.Name = "Courier New"
         .Font.size = 12
         .Text = ""
         .Execute Format:=True
     End With
```

На практиці більш корисною, аніж отримання підтвердження збігу рядка, що шукається у тексті документу (або його кількості збігів), є можливість заміни знайдених фрагментів документу на інші (та/або заміни їх ознак форматування).

Таку можливість реалізують об'єкти класу Replacement, які використовуються разом з об'єктами класу Find. Посилання на цей об’єкт можна отримати з властивості Replacement об’єкта Find.

Найважливіші властивості класу Replacement збігаються з властивостями об'єкту Find (включаючи Text, Font, ParagraphFormat, Style, LanguageID – див. вище) встановлюються у значення, на які потрібно замінити знайдений текст та/або ознаки його форматування.

Метод у класі Replacement тільки один:

- ClearFormatting() – видаляє ознаки форматування тексту та абзацу з рядка-заміни, що використовується.

Організація пошуку фрагментів документу з їх заміною містить перші три з чотирьох кроки, вказані вище для пошуку тексту (див. вище) та додаткові кроки:

4) отримати об'єкт класу Replacement з властивості Replacement об’єкта Find;

5) задати значення властивостей об'єкту Replacement, що визначають текст та/або ознаки форматування фрагменту, яким будуть замінені знайдені фрагменти документу (альтернативно текст може бути зазначений як аргумент ReplaceWith методу Execute());

6) викликати метод Execute() об’єкта Find (за необхідністю встановлюючи кількість замін значенням параметру Replace цього методу – див. вище) та перевірити результат, що він повертає – у разі значення True, збіг тексту, що шукається у визначеному діапазоні/виділеній області документу знайдений і заміна його виконана.

Наведений нижче приклад виконує заміну усіх входжень текстового рядка "програмування" на "!програмування!" у документі:

```
Set myRange = ActiveDocument.Content
With myRange.Find
    .Text = "програмування"
    With .Replacement
        .Text = "!програмування!"
    End With
    .Execute Replace := wdReplaceAll
End With
```

Зауважимо, що у разі організації заміни фрагментів документу, параметр Replace методу Execute() є обов'язковим і може приймати значення елементів енумератора WdReplace (див. вище).

5.7 Робота з таблицями (класи Tables, Table, Row, Column та Cell)

Одними з часто використовуваних об'єктів у документах Microsoft Word є таблиці, для роботи з ними об'єктна модель Microsoft Word передбачає класи Table, Row, Column, Cell та колекцію Tables, яка є властивістю класів Document, Selection та Range (Рисунок 62) [4]. Як і більшість інших

колекцій, *колекція* `Tables` *має основну властивість* `Count` – кількість таблиць у документі та *два методи*:

- `Add()` – виконує додання нової таблиці до документу та повертає посилання на неї, має параметри:
  - `Range` – діапазон, у який (якщо він стиснений) або замість якого (якщо не стиснений) буде вставлена таблиця (обов'язковий параметр);
  - `NumRows` – кількість рядків створюваної таблиці (значення `Long`, обов'язковий параметр);
  - `NumColumns` – кількість стовпців створюваної таблиці (значення `Long`, обов'язковий параметр);
  - `DefaultTableBehavior` – визначає, чи буде автоматично змінюватися розмір клітинок для вміщення їх вмісту, може мати значення: `wdWord8TableBehavior` – автоматична зміна розміру клітинок заборонена (за замовчуванням) або `wdWord9TableBehavior` – автоматична зміна розміру клітинок дозволена (необов'язковий параметр);
  - `AutoFitBehavior` – визначає правила автоматичного визначення розміру таблиці, може бути елементом енумератора `WdAutoFitBehavior` з одним зі значень: `wdAutoFitWindow - 2` – таблиця вміщується по ширині вікна документа (за замовчуванням), `wdAutoFitContent - 1` – ширина таблиці визначається вмістом її клітинок, `wdAutoFitFixed - 0` – автоматичне визначення ширини таблиці заборонено.
- `Item(Index)` – повертає посилання на таблицю за її індексом (типу `Long`, обов'язковий параметр).

Для додання таблиці до документа необхідно:

1) Визначити діапазон/виділену область, зазвичай у стисненому вигляді (як місце вставлення), у яку буде додана таблиця;

2) Викликати метод `Add()` колекції `Tables`, яка є властивістю об'єкта `Document`, передаючи йому визначений діапазон та кількість рядків та стовпців створюваної таблиці;

3) Присвоїти посилання на створену таблицю, що повертається методом `Add()`, об'єкту `Table`, який буде використовуватися для роботи з цією таблицею.

Наведемо приклад програми, що створює таблицю у новому документі:

```
Sub TableCreate()
    Dim myDoc As Word.Document
    Dim myRange As Range
    Dim myTable As Table

    Set myDoc = Documents.Add()
    Set myRange = myDoc.Range(Start:=0, End:=0)
    Set myTable = myDoc.Tables.Add(Range:=myRange, _
                                   NumRows:=3, NumColumns:=4)
```

```
    myDoc.Close wdDoNotSaveChanges
End Sub
```

Клас `Table` має такі найважливіші властивості:

- `Rows` – колекція, що містить рядки таблиці – об'єкти `Row` (будуть розглянуті нижче у цьому розділу);
- `Columns` – колекція, що містить стовпці таблиці – об'єкти `Column` (будуть розглянуті нижче у цьому розділу);
- `Range` – діапазон документу, який містить таблицю;
- `Borders` – містить посилання на колекцію об'єктів `Border`, що визначають вигляд меж таблиці через властивості колекції такі, як `InsideLineStyle` та `OutsideLineStyle`, що приймають значення елементів енумератора – найпоширеніші: `wdLineStyleSingle - 1` – одинарна суцільна лінія, `wdLineStyleDouble - 7` – подвійна суцільна лінія, `wdLineStyleNone - 0` – відсутність ліній;

Клас `Table` має такі найважливіші методи:

- `Cell(Row, Column)` – мабуть найважливіший метод, що повертає посилання на клітинку таблиці – об'єкт `Cell` (буде розглянутий нижче у цьому розділі), зазначаючи аргументом `Row` послідовний номер її рядка, а аргументом `Column` – послідовний номер її стовпця (номери починаються з `1` і мають тип `Long`);
- `Select()` – повертає об'єкт `Selection`, який містить таблицю;
- `Sort()` – виконує сортування даних таблиці, має велику кількість опціональних параметрів, включаючи:
  - `ExcludeHeader` – визначає, чи виключати рядок заголовку з сортування (`True`), чи ні (`False` – значення за замовчуванням);
  - `FieldNumber`, `FieldNumber2`, `FieldNumber3` – містять послідовний номер стовпця (починаючи з 1), за яким виконується сортування – підтримується трирівневе сортування;
  - `SortFieldType`, `SortFieldType2`, `SortFieldType3` – тип сортування для відповідних стовпців, за якими виконується сортування, може приймати значення елементів енумератора `WdSortFieldType` (найчастіше: `wdSortFieldAlphanumeric - 0` – сортування за кодами символів, `wdSortFieldNumeric - 1` – сортування за цифровими значеннями, `wdSortFieldDate - 2` – сортування за значеннями дат/часу);
  - `SortOrder`, `SortOrder2`, `SortOrder3` – порядок сортування для відповідних стовпців, за якими виконується сортування, може приймати значення елементів енумератора `WdSortOrder`: `wdSortOrderAscending - 0` – сортування за зростанням (за замовчуванням), `wdSortOrderDescending - 1` – сортування за зменшенням;

- `CaseSensitive` – визначає, чи враховується регістр символів при сортуванні (`True`), чи ні (`False` – значення за замовчуванням);
- `LanguageID` – визначає мову тексту, що сортується, приймає значення енумератора `WdLanguageID` (див. вище).

- `AutoFormat()` – дозволяє застосувати до таблиці один зі стандартних форматів, які визначаються елементом енумератора `WdTableFormat` [27], наприклад, `wdTableFormatGrid1` – для таблиці з усіма межами у вигляді суцільної лінії;

- `AutoFitBehavior()` – визначає правила автоматичного визначення розміру таблиці, приймає обов'язковий параметр `Behavior` зі значеннями елементів енумератора `WdAutoFitBehavior` (див. вище).

- `Delete()` – видаляє поточну таблицю з документу.

Для заповнення таблиці даними необхідно:

1) отримати посилання на об’єкт `Cell`, що представляє клітинку таблиці, викликом методу `Cell(Row, Column)` об’єкта `Table` із зазначенням координат клітинки;

2) отримати посилання на об’єкт `Range` клітинки, який визначає її вміст;

3) додати необхідний вміст клітинки за допомогою метода `InsertAfter()` об’єкта `Range`.

Наведемо приклад програми, у якій створюється таблиця, форматується стандартним форматуванням, заповнюється даними, а потім ці дані зчитуються у двовимірний масив:

```
Sub TableDataInsertAndRead()
    Dim myDoc As Word.Document
    Dim myRange As Range
    Dim myTable As Table
    Dim tableCell As Cell
    Dim i As Integer, j As Integer
    Dim cellValues() As String

    'Створення таблиці
    Set myDoc = Documents.Add()
    Set myRange = myDoc.Range(Start:=0, End:=0)
    Set myTable = myDoc.Tables.Add(Range:=myRange, _
                                   NumRows:=3, NumColumns:=4)
        'Форматування таблиці
    myTable.AutoFormat Format:=wdTableFormatGrid1

    'Заповнення таблиці даними
    i = 1
    For Each tableCell In myTable.Range.Cells
        tableCell.Range.InsertAfter "Cell-" & i
        i = i + 1
    Next tableCell

    'Читання даних таблиці у двовимірний масив
    ReDim cellValues(myTable.Rows.count, _
```

```
                    myTable.Columns.count)
    For i = 1 To myTable.Rows.count
        For j = 1 To myTable.Columns.count
            Set myRange = myTable.Cell(i, j).Range

            'Видалення з діапазону вмісту клітинки
            'маркеру кінця клітинки
            myRange.MoveEnd Unit:=wdCharacter, count:=-1
            cellValues(i, j) = myRange
            Debug.Print cellValues(i, j); " ";
        Next j
        Debug.Print vbCrLf;
    Next

    myDoc.Close wdDoNotSaveChanges
End Sub
```

Зверніть увагу, що при зчитуванні вмісту клітинки виконується зміщення кінцевої межі діапазону на один символ вліво – це необхідно, щоб не зчитувати маркер кінця клітинки '¤'.

Наведемо приклад коду, який демонструє організацію багаторівневого сортування та сортування за датами та часом:

```
Sub TableSorting()
    Dim myDoc As Word.Document
    Dim myTable As Table

    Set myDoc = Documents.Open(ActiveDocument.Path _
            & Application.PathSeparator & "Документ-зразок.docx")

    'Багаторівневе сортування
    Set myTable = myDoc.Tables(3)
    myTable.Sort ExcludeHeader:=True, FieldNumber:=2, _
       SortFieldType:=wdSortFieldNumeric, _
       SortOrder:=wdSortOrderAscending, _
       FieldNumber2:=1, SortFieldType2:=wdSortFieldAlphanumeric, _
       SortOrder2:=wdSortOrderAscending

    'Сортування дат та часу
    Set myTable = myDoc.Tables(5)
    myTable.Select
    myTable.Sort ExcludeHeader:=True, FieldNumber:=2, _
                 SortFieldType:=wdSortFieldDate, _
                 SortOrder:=wdSortOrderDescending

    myDoc.Close wdDoNotSaveChanges
End Sub
```

На Рисунок 74а наведений результат дворівневого сортування даних таблиці: спочатку за розміром у байтах (стовпець 2), а потім за кодами символів типу даних (стовпець 1), а на Рисунок 74б наведений результат сортування за часом за зменшенням значень.

Тип даних	Розмір (байт)	Діапазон значень
String (змінної довжини)	10 + 1*n	*Від 0 до* **приблизно** *2 млрд. символів*
String (фіксованої довжини)	Довжина рядка	*Від 0 до* **приблизно** *65 400 символів*
Variant (універсальний текстовий)	22 + 1*n	Той же діапазон, що і для строкового типу із змінною довжиною
Byte	1	Цілі числа від **0** до **255**
Boolean	2	**ІСТИНА (TRUE)** або **ХИБНІСТЬ (FALSE)**
Integer	2	Цілі числа від **–32 768** до **32 767**
Long	4	Цілі числа від **–2 147 483 648** до **2 147 483 647**
Object	4	***Посилання на будь-який об'єкт***
Single	4	Від **–3,402823E38** до **–1,401298E-45** (від'ємні) та від **1,401298E-45** до **3,402823E38** (додатні)
Currency	8	Від **–922 337 203 685 477,5808** до **922 337 203 685 477,5807**

а)

Дата	Час	Рядок
03.07.2022	21:10	третій етап
01.06.2021	18:55	другий етап
25.04.2020	05:15	перший етап

б)

Рисунок 74 – Сортування таблиць: а) багаторівневе; б) дат та часу

Microsoft Word API для роботи з таблицями також пропонує класи `Row` – рядок таблиці та `Column` – стовпець таблиці, *основними властивостями* яких можна назвати:

- `Cells` – колекція клітинок рядка або стовпця (зауважимо, що таку ж властивість мають класи `Range` та `Selection`);
- `Range` – містить посилання на діапазон з клітинками рядка або стовпця;
- `Index` – послідовний номер рядка або стовпця в колекції рядків (`Rows`) або стовпців (`Columns`);
- `isFirst/isLast` – повертають `True`, якщо рядок або стовпець є першим/останнім;
- `Next/Previous` – повертають наступний/попередній від поточного рядка або стовпця.

Основні методи класів `Row` та `Column`:

- `Select` – повертає посилання на об'єкт `Selection`, який містить рядок або стовпець;
- `Delete` – видаляє поточний рядок або стовпець з таблиці.

Клас `Cell` має аналогічні класам `Row` та `Column` властивості та методи, але й корисні додаткові методи:

- `Merge(MergeTo)` – об'єднує поточну клітинку з клітинкою, визначеною параметром `MergeTo`;
- `Split(NumRows, NumColumns)` – поділяє поточну клітинку на `NumRows` по вертикалі та `NumColumns` по горизонталі;
- `AutoSum` – виконує вставлення поля `=(Formula)`, яке виконує розрахунок суми значень у поточному стовпці вище поточної клітинки або у поточному рядку зліва від поточної клітинки у залежності, де знаходяться дані, які можуть бути підраховані (поточний стовпець має перевагу);
- `Formula(Formula, NumFormat)` – виконує вставлення поля `=(Formula)`, яке виконує розрахунок над даними таблиці відповідно до

значення параметру `Formula`, що визначає функцію та область даних, що обраховуються. Параметр `NumFormat` визначає формат результату (Таблиця 6).

Наведемо приклад програми, у якій створюється таблиця, до неї додаються випадкові дані до всіх клітинок, окрім останнього рядка, а до останнього рядка заносяться функції-агрегації:

```
Sub TableCalculate()
    Dim myDoc As Word.Document
    Dim myTable As Table
    Dim tableCell As Cell
    Dim i As Integer, j As Integer

    Set myDoc = Documents.Open(ActiveDocument.Path _
            & Application.PathSeparator & "Документ-зразок.docx")
    'Перехід до кінця документа
    Selection.EndKey Unit:=wdStory
    Selection.TypeParagraph

    'Створення таблиці та заповнення випадковими даними
    'з 1 по 4 рядок
    Set myTable = myDoc.Tables.Add(Selection.Range, 5, 6)
    myTable.AutoFormat Format:=wdTableFormatGrid1
    For i = 1 To myTable.Rows.count - 1
        For j = 1 To myTable.Columns.count
            myTable.Cell(i, j).Range.InsertAfter Int(10 * Rnd()+1)
        Next j
    Next i

    'Додання функцій агрегації до останнього рядка
    With myTable
        'Якщо спочатку вставляти формулу у перший стовпець,
        'у нього будуть вставлені усі формули
'        .Cell(5, 1).Formula "=SUM(ABOVE)"
        .Cell(5, 2).Formula "=AVERAGE(ABOVE)"
        .Cell(5, 3).Formula "=MIN(ABOVE)"
        .Cell(5, 4).Formula "=Max(Above)"
        .Cell(5, 5).Formula "=Count(Above)"
        .Cell(5, 6).Formula "=Product(Above)"
        .Cell(5, 1).Formula "=SUM(ABOVE)"
    End With

    myDoc.Close wdDoNotSaveChanges
End Sub
```

Зверніть увагу, що у разі, якщо формули починають додаватися з першого стовпця, метод `Formula()` відпрацьовує некоректно – усі формули додаються у перший стовпець, тому рекомендується додавати формулу до першого стовпця останнім [28].

5.8 Робота з рисунками (колекція Shapes, класи Shape та InlineShape)

Об’єкти `Document` (а також об'єкти `Range`) мають властивість `Shapes`, яка є колекцію об’єктів `Shape` (Рисунок 62). Кожен об'єкт `Shape` представляє об'єкт у *області рисунку* (*Canvas*), такий як автофігура (існує перерахування `MsoAutoShapeType`, яке визначає більше 180 автофігур – лінія, прямокутник, еліпс, зірки, стрілки тощо) [29], Рисунок вільної форми який складається з автофігур, OLE-об'єкт або Рисунок, що завантажується з файлу. Виконувати програмне додання автофігур є досить складним і навряд чи доцільним, тому далі ми розглянемо роботу з рисунками, які додаються до документу із файлів.

Роботу з такими рисунками забезпечує колекція `InlineShapes` (властивість класів `Document`, `Range` та `Selection`) та клас `InlineShape` (Рисунок 62). Особливістю додання рисунків через ці засоби є їх безпосереднє завантаження до документу без використання області рисунку (Canvas). При цьому вони додаються як об’єкти `InlineShape` у колекцію `InlineShapes`. Клас `InlineShapes` має стандартні для колекцій *властивості*, з яких згадаємо `Count` – кількість елементів колекції, а *найважливішим методом* можна назвати:

- `AddPicture(FileName, LinkToFile, SaveWithDocument, Range)`, який приймає один обов’язковий параметр `FileName` – повний шлях до файлу рисунку. Необов’язковий параметр `LinkToFile` дозволяє встановлювати зв’язок з файлом, з якого завантажується зображення (`msoTrue`) так, щоб при зміні файлу автоматично змінилося зображення у документі. За замовчуванням для цього параметру використовується `msoFalse`, що робить вставлений у документ Рисунок незалежною копією. Значення `msoTrue` параметру `SaveWithDocument` дозволяє зберігати пов’язаний Рисунок разом з документом (значення за замовчуванням - `msoFalse`). Параметр `Range` вказує на діапазон, у якому буде розміщений Рисунок. Якщо діапазон не стиснутий (не використаний його метод `Collapse()`), при вставці Рисунок замінює діапазон, у противному разі – Рисунок вставляється у діапазон. Якщо параметр `Range` не вказаний Рисунок вставляється автоматично у поточний діапазон, який містить посилання на поточну колекцію `InlineShapes`. Метод повертає посилання на об’єкт `InlineShape`, та додає цей об'єкт до колекції `InlineShapes`.

Наступний приклад демонструє вставку рисунків з підкаталогу images каталогу, у якому знаходиться файл з програмою, на початок кожної сторінки нового документу:

```
Sub PicturesInsert()
    Dim myDoc As Word.Document
    Dim myRange As Range
    Dim pathToImgDir As String
    Dim filename As String
    Dim i As Integer

    i = 1
```

```
    pathToImgDir = ThisDocument.Path & Application.PathSeparator _
                 & "images" & Application.PathSeparator

    Set myDoc = Documents.Add()
    Set myRange = myDoc.Paragraphs(i).Range
    ChDir (pathToImgDir)

    filename = Dir("*.jpg")
    Do While filename <> ""
        myRange.InlineShapes.AddPicture fileName:=pathToImgDir _
          & vFile, LinkToFile:=msoFalse, SaveWithDocument:=msoTrue
        myRange.MoveStart wdParagraph, 1
        myRange.InsertBreak
        i = i + 1
        filename = Dir()
    Loop

    myDoc.Close wdDoNotSaveChanges
End Sub
```

Після вставки рисунку виконується зміщення початку діапазону на один абзац далі, щоб наступні вставки не замінювали рисунок, і вставка розриву сторінки для створення нової сторінки.

Питання для самоконтролю

1. Назвіть задачі, які можуть бути автоматизовані при роботі з документами Microsoft Word.
2. Назвіть основні класи об'єктної моделі Microsoft Word.
3. Назвіть клас, що відповідає екземпляру застосунку Microsoft Word. Назвіть його найважливіші властивості.
4. Як запустити програмним чином Microsoft Word із появою його на екрані?
5. Яким чином можна отримати посилання на вже запущений застосунок Microsoft Word?
6. Як отримати доступ до вікна вже відкритого документа?
7. Як організувати виведення діалогового вікна відкриття та збереження документу?
8. Як організувати друк певних сторінок документу у певній кількості копій?
9. Як закрити Microsoft Word без збереження змін у документі?
10. Для чого використовують колекцію `Documents`? Які параметри може приймати метод `Add()` цієї колекції?
11. Надайте характеристику найважливішим методам колекції `Documents`.
12. Чим відрізняються властивості `ThisDocument` та `ActiveDocument` класу `Application`?
13. Назвіть основні формати, у яких може зберігатися документ Microsoft Word.
14. Що називають головним ланцюжком документа (main document story), яка властивість класу `Document` його повертає та у якому вигляді?
15. Назвіть основні властивості-колекції класу `Document` та надайте їм характеристику.

16. Назвіть найважливіші методи класу `Document` та надайте їм характеристику.
17. Опишіть параметри методу `GoTo()` класу `Document`. Які об'єкти переходу та способи переміщення до них він підтримує?
18. Назвіть відмінності об'єктів класів `Range` (діапазон) та `Selection` (виділена область).
19. Назвіть способи створення об'єктів `Range` та способи створення об'єктів `Selection`.
20. Назвіть однакові основні властивості класів `Range` та `Selection` та надайте їм характеристику. Які властивості дозволяють додавати/редагувати текст а які виконувати форматування тексту та абзаців?
21. Охарактеризуйте методи навігації класів `Range` та `Selection` та надайте їм характеристику. Назвіть спільні та відмінні елементи об'єктів переходу, що використовуються методом `GoTo()`, та одиниць даних, які використовуються об'єктами `Range` та `Selection` для навігації у інших методах.
22. Охарактеризуйте методи класів `Range` та `Selection`, шо виконують зміну розміру діапазону/виділеної області та надайте їм характеристику.
23. Охарактеризуйте методи класів `Range` та `Selection`, шо виконують вставку/видалення тексту та об'єктів та надайте їм характеристику.
24. Назвіть методи класу `Selection`, відсутні у класі `Range`, які виконують навігацію та зміну виділеної області.
25. Що являють собою закладки документу Microsoft Word та яким чином вони можуть бути використані при програмуванні? Назвіть переваги навігації за допомогою закладок порівняно з навігацією по об'єктам документу.
26. Назвіть основні властивості класу `Bookmark` та надайте їм характеристику. Яким чином можна програмно додати закладку до документу?
27. Назвіть основні методи класу `Bookmark` та надайте їм характеристику. Яку особливість необхідно враховувати при доданні закладки за допомогою властивості `Text` пов'язаного з закладкою об'єкта `Range`?
28. Який об'єкт документу Microsoft Word зберігає колонтитули та які колекції існують для цього? Об'єкти якого класу вони зберігають?
29. Охарактеризуйте значення елементів енумератора `WdHeaderFooterIndex`, які можуть приймати як аргументи параметра `Index` колекції, що зберігають колонтитули.
30. Назвіть основні властивості класу `HeaderFooter` та надайте їм характеристику. Охарактеризуйте властивості класу `PageNumber`, об'єкти якого зберігаються у властивості-колекції `PageNumbers` класу `HeaderFooter`.
31. Яким чином організується програмний пошук фрагментів документа або ознак форматування? Об'єкти якого класу містять критерії пошуку?
32. Назвіть основні властивості класу `Find` та надайте їм характеристику.
33. Назвіть найважливіші методи класу `Find` та надайте їм характеристику. Охарактеризуйте параметри метода `Execute()`.
34. Перерахуйте кроки програмної організації пошуку фрагменту тексту у

документі. Як отримати координати знайденого фрагменту?

35. Перерахуйте кроки програмної організації пошуку ознак форматування у документі.
36. Яким чином організується заміна знайдених фрагментів тексту або ознак форматування, об'єкти якого класу забезпечують таку функціональність?
37. Перерахуйте кроки програмної організації пошуку та заміни фрагменту тексту у документі. Як керувати кількістю замін знайдених фрагментів тексту?
38. Перерахуйте класи, що забезпечують можливість роботи з таблицями документу Microsoft Word. Які класи містять властивість-колекцію `Tables`?
39. Як додати таблицю до документу програмним чином? Охарактеризуйте параметри методу `Add()` колекції `Tables`.
40. Назвіть основні властивості класу `Table` та надайте їм характеристику.
41. Назвіть найважливіші методи класу `Table` та надайте їм характеристику.
42. Яким чином можна додати та налаштувати межі таблиці? Які класи та методи для цього використовуються?
43. Перерахуйте кроки програмного заповнення таблиці даними та їх зчитування до двовимірного масиву. Яку особливість при зчитуванні вмісту клітинки необхідно враховувати?
44. Охарактеризуйте параметри методу `Sort()` класу `Table`. Опишіть кроки програмної організації багаторівневого сортування даних таблиці.
45. Назвіть основні властивості та методи класів `Row` та `Column` і надайте їм характеристику.
46. Назвіть основні властивості та методи класу `Cell` та надайте їм характеристику.
47. Яким чином виконується програмна організація розрахунків даних таблиці Microsoft Word? Яку особливість необхідно враховувати при використання методу `Formula()`?
48. Які засоби надає Microsoft Word API для роботи з рисунками? Чим відрізняються об'єкти `Shape` та `InlineShape`?
49. Які класи містять властивість-колекцію `InlineShapes`? Охарактеризуйте параметри методу `AddPicture()` цієї колекції.
50. Опишіть кроки програмної реалізації додання до документу рисунку з файлу формату JPEG.

Тестові завдання для перевірки знань

1. Оберіть оператори, виконання яких призводить до запуску Microsoft Word (оберіть декілька відповідей):

а) `Set myWord = GetObject(, "Word.Application");`

б) `Dim myWord As New Word.Application;`

в) `Dim myWord As Word.Application;`

г)
```
Dim oWord As Word.Application
Set oWord = New Word.Application.
```

2. Оберіть способи звернення до об'єкту `Document` без створення змінної (оберіть декілька відповідей):

а) `Documents.Item(i),i - індекс документа;`

б) `Application.newDocument();`

в) `ThisDocument;`

г) `ActiveDocument;`

д) `GetObject(,Word.Document).`

3. Оберіть властивість класів `Range` та `Selection`, що дозволяє ввести до об'єктів цих класів текстовий рядок:

а) `Type`;

б) `Information`;

в) `Bookmark;`

г) `Text;`

д) `Data.`

4. Оберіть з переліку об'єкти-колекції, які **не містять** відповідні їх назві об'єкти, а натомість містять об'єкти `Range` (оберіть декілька відповідей):

а) `Sections`;

б) `Paragraphs`;

в) `Sentences`;

г) `Words`;

д) `Characters.`

5. Оберіть однакові об'єкти, які використовуються як об'єкти переходу методом `GoTo()`, та як одиниці даних, які використовуються методом `Move()` класів `Range` та `Selection` (оберіть декілька відповідей):

а) Розділ;

б) Абзац;

в) Речення;

г) Рядок тексту;

д) Таблиця.

6. Вкажіть клас, об'єкти якого визначають колонтитули документу:

а) `Header`;

б) `Footer`;

в) `HeaderFooter`;

г) `Headline`.

7. Оберіть ситуації, що підтверджують знаходження фрагменту документа або ознаки форматування у документі (оберіть декілька відповідей):

а) Властивість `Text` об'єкту `Find` повертає текстовий рядок, що шукався;

б) Метод `Execute()` об'єкту `Find` повертає `True`;

в) Властивість `FindText` об'єкту `Find` повертає текстовий рядок, що шукався;

г) Властивість `Found` об'єкту `Find` повертає `True`.

8. Оберіть класи, які мають властивість-колекцію `Tables` (оберіть декілька відповідей):

а) `Document`;
б) `Section`;
в) `Selection`;
г) `Paragraph`;
д) `Range`.

9. Яку максимальну кількість рівнів сортування підтримує метод `Sort()` класу `Table`:
а) два;
б) три;
в) чотири;
г) п'ять.

10. Оберіть клас, який відповідає рисунку, створеному з автофігур, таких як лінія. прямокутник, еліпс тощо:
а) `Drawing`;
б) `Shape`;
в) `Figure`;
г) `InlineShape`.

Завдання для самостійного виконання

1. Розробити програму, що дозволяє до документу зі списком студентів з даними (ПІБ студента, назва групи, ПІБ директора інституту, дата пропуску, причина пропуску) на першій сторінці додати на кожну сторінку, починаючи з другої, пояснювальні записки пропуску занять такого змісту: "Директору ІННІ ЗНУ ПІБ директора інституту студента групи назва групи ПІБ студента. Прошу вважати пропуск занять дата пропуску з поважної причини: причина пропуску. поточна дата, підпис". Причини пропуску: за сімейними обставинами, внаслідок поганого самопочуття, повістка до військкомату тощо.

2. Розробити програму, що дозволяє конвертувати усі файли формату *.docx з обраного за допомогою діалогового вікна каталогу у файли формату *.pdf.

3. Розробити програму, що дозволяє з обраного за допомогою діалогового вікна обрання каталогу для усіх документів витягнути та створити у документі з програмою список назв документів разом з назвами усіх рисунків кожного документу (назви рисунків знаходяться у наступному після рисунку абзаці).

4. Розробити програму, що дозволяє скопіювати вміст усіх файлів формату *.docx до нового документу Word з обраного за допомогою діалогового вікна обрання каталогу.

5. Розробити програму, що дозволяє розділити вміст документу у декілька нових документів так, щоб у кожному зберігався окремий абзац, та зберігати всі нові документи з іменами оригінального файлу та з суфіксами-послідовними номерами абзаців.

6. Розробити програму, що виконує додання до документу, починаючи з третьої сторінки таблиць розміром 8 стовпців на 10 рядків, для кожної вставляє у клітинці першого рядку однакові заголовки зі списку на першій сторінці і

розміщує перед таблицею її назву зі списку на другій сторінці у вигляді: Табл. X. Назва таблиці (X - послідовний номер таблиці).

7. Розробити програму, що виконує додання до документу, починаючи з другої сторінки усіх рисунків формату *.jpg з обраного за допомогою діалогового вікна обрання каталогу та додає під кожним рисунком підпис зі списку підписів до рисунків на першій сторінці документу у вигляді: Рисунок X підпис (X - послідовний номер рисунку).

8. Розробити програму, що дозволяє конвертувати всі файли формату *.docx у файли формату *.doc а всі файли формату *.doc у файли формату *.docx з обраного за допомогою діалогового вікна обрання каталогу.

9. Розробити програму, що дозволяє з обраного за допомогою діалогового вікна обрання каталогу для усіх документів витягнути та створити у документі з програмою список назв документів разом з назвами усіх таблиць кожного документу (назви таблиць знаходяться у попередньому перед таблицею абзаці).

10. Розробити програму, що дозволяє для документу з таблицею виконати переміщення її стовпців так, щоб перший стовпець став останнім, другий – передостаннім тощо.

ТЕМА 6 ОБ’ЄКТНА МОДЕЛЬ ТА ПРОГРАМУВАННЯ В MICROSOFT EXCEL

Мета: розглянути об'єктну модель Microsoft Excel та вивчити найважливіші властивості та методи класів, що відображають сам табличний процесор, файл робочої книги, робочі аркуші та діапазони клітинок, вивчити способи програмної навігації та редагування клітинок аркуша, пошуку і заміни його фрагментів, роботи з закладками, колонтитулами, таблицями та рисунками, доданими до документу.

План

1. Об’єктна модель Microsoft Excel
2. Клас застосунку Microsoft Excel (`Application`)
3. Робота з колекцією робочих книг (клас `Workbooks`) та робочими книгами (клас `Workbook`)
4. Робота з колекцією робочих аркушів (класи `Sheets`, `Worksheets`) та робочими аркушами (клас `Worksheet`)
5. Робота з діапазонами (клас `Range`)
6. Робота з об'єктами інших класів об’єктної моделі Microsoft Excel

Основні терміни і поняття

Об'єктна модель, застосунок Microsoft Excel, колекція робочих книг, робоча книга, колекція аркушів, колекція робочих аркушів, робочий аркуш, діапазон документу.

Основні теоретичні положення

6.1 Об’єктна модель Microsoft Excel

Microsoft Excel – є найпопулярнішим табличним процесором внаслідок наявності його розвиненого функціоналу зі створення та роботі з електронними таблицями. Зазвичай, Microsoft Excel використовується для виконання різноманітних розрахунків, проведення аналізу їх результатів та їх відображення, наприклад, у вигляді діаграм. Microsoft Excel містить дані у *робочих книгах* (*Workbooks*), що зберігаються у файлах, кожна робоча книга містить *аркуші* (*Sheets*) з електронними таблицями, у яких є велика кількість *стовпців* (*Columns*) та *рядків* (*Rows*) (16 384 стовпця та 1 048 576 рядків у сучасних версіях застосунку), на перетині кожного з кожним знаходяться *клітинки* (*Cells*), саме у якій розміщуються дані. Така організація даних забезпечує природну адресацію клітинок у вигляді адреса_стовпця адреса_рядка (адреса стовпця, зазвичай, передається літерами латинської абетки, а адреса рядка – послідовним номером), наприклад: A3 – клітинка у третьому рядку першого стовпця (Рисунок 75).

Зауважимо, що при адресації клітинки аргументами властивостей та методів об'єктів Microsoft Excel API першою вказується адреса рядка, а другою – адреса стовпця, зазвичай, як послідовними номерами [5].

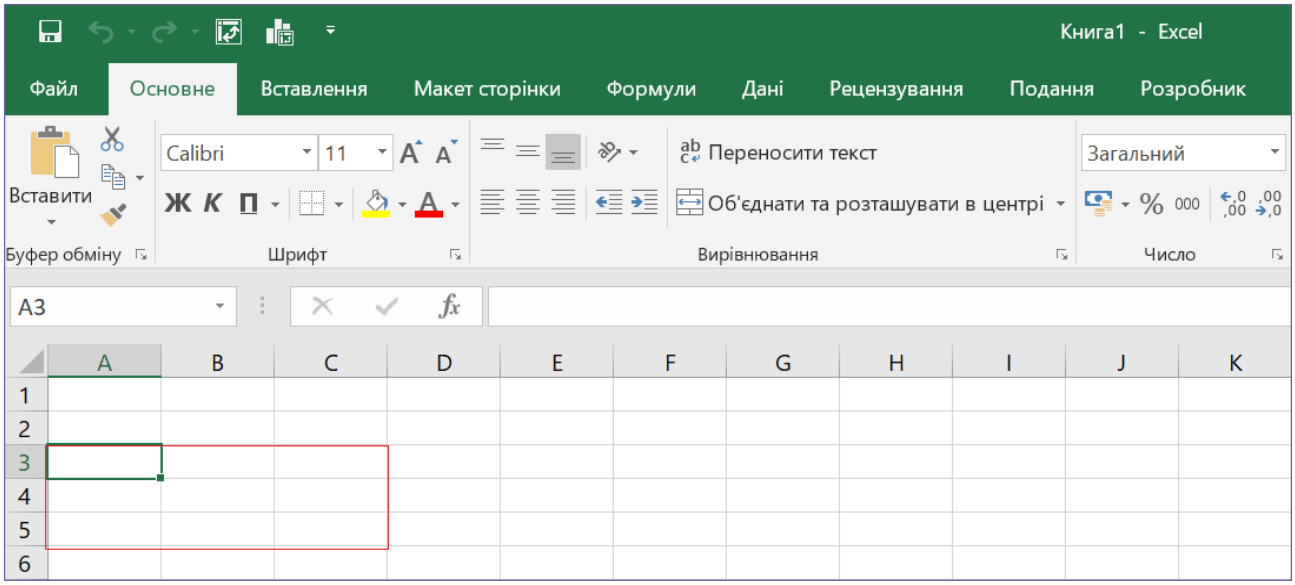


Рисунок 75 – Електронна таблиця аркушу робочої книги Microsoft Excel з виділеною клітинкою з адресою A3

У разі, якщо адресується діапазон суміжних клітинок, його адреса вказується як адреса_лівої_верхньої_клітинки:адреса_правої_нижньої клітинки, наприклад (Рисунок 75),

A3:C5

або

R3C1:R5C3

У разі, якщо адресується клітинка (або діапазон клітинок) на іншому аркуші (наприклад, як операнд формули), адреса такої клітинки (або діапазону клітинок) виглядає:

Аркуш2!A5 або Аркуш2!R5C1

та

Аркуш2!A5:C8 або Аркуш2!R5C:R8C3

У разі, якщо адресується клітинка (або діапазон клітинок) у іншій відкритій робочій книзі, адреса такої клітинки (або діапазону клітинок) виглядає:

[КнигаЗразок.xlsx]Аркуш1!B4 або [КнигаЗразок.xlsx]Аркуш1!R4C2

та

[КнигаЗразок.xlsx]Аркуш1!B4:D7 або [КнигаЗразок.xlsx]Аркуш1!R4C2:R7C4

У якості прикладів можна навести такі області застосування, для яких доцільне програмування на VBA у Microsoft Excel:

- повторювані для великої кількості об'єктів та повторювані у часі розрахунки, наприклад, планування навчального навантаження, складання кошторисів, обрахування податків тощо [8];
- опрацювання даних з зовнішніх джерел (інших електронних таблиць, баз даних тощо), їх аналіз та зручне представлення, наприклад, у вигляді діаграм;

- підготовка даних для експорту до текстового процесору з метою їх використанні у різноманітних звітах.

Зазначені вище об'єкти робочої книги, аркушу, стовпця. рядка, клітини разом з багатьма іншими об'єктами, що використовуються при програмуванні у Microsoft Excel, є об'єктами класів, що складають *об'єктну модель Microsoft Excel* [30]. Ця модель містить більше 200 класів, але найчастіше використовуються зазначені на Рисунок 76.

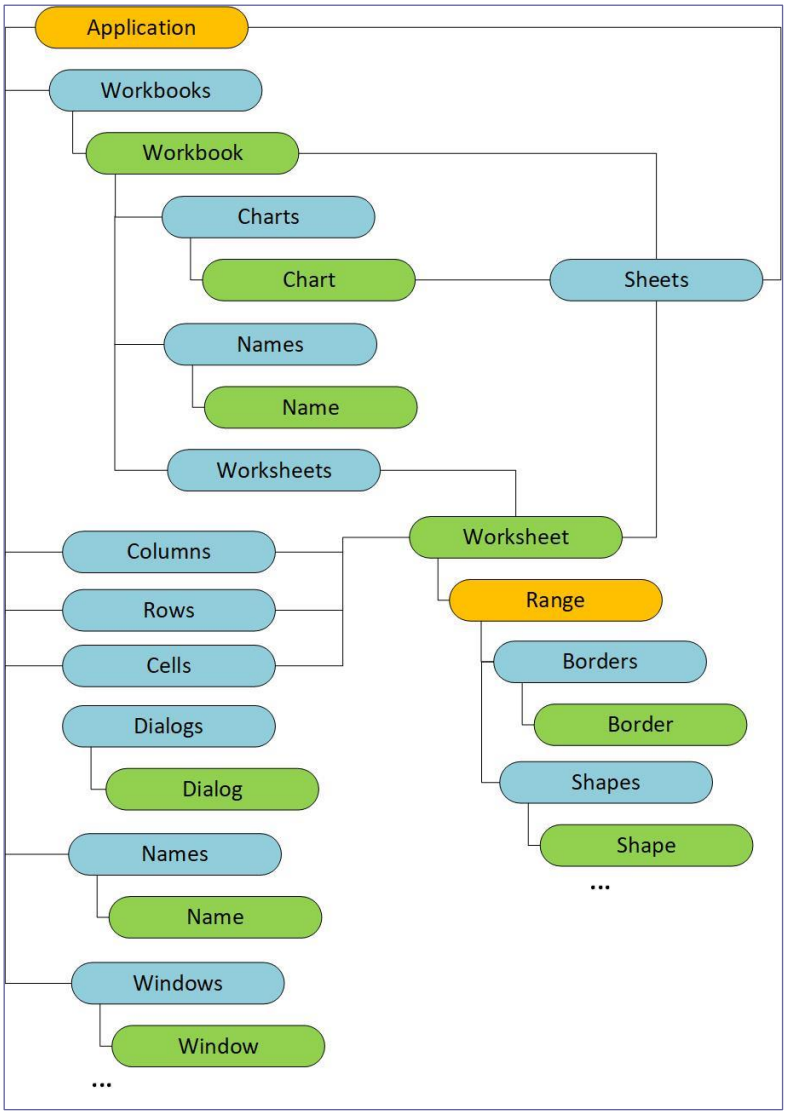


Рисунок 76 – Об'єктна модель Microsoft Excel

6.2 Клас застосунку Microsoft Excel (Application)

Клас `Application` об'єктної моделі Microsoft Excel відповідає екземпляру застосунку Microsoft Excel [5]. Створення об'єкту такого класу означає запуск екземпляра Microsoft Excel, він запускається як фоновий процес, який може спостерігатися у *Диспетчері завдань* операційної системи. Для відображення застосунку на екрані, як і у випадку з Microsoft Word, необхідно встановити значення `True` для його властивості `Visible`. Наведена далі програма

демонструє запуск екземпляра Microsoft Excel, його відображення та завершення роботи екземпляра застосунку Microsoft Excel:

```
Sub StartExcel()
    Dim myExcel As Excel.Application

    Set myExcel = New Excel.Application
    myExcel.Visible = True

    myExcel.Quit
End Sub
```

У разі, якщо застосунок Microsoft Excel вже запущений і потрібно отримати посилання на нього, так як і для Microsoft Word використовується вбудована функція `GetObject()` (див. вище), однак цього разу їй передається аргумент `class`, який дорівнює `"Excel.Application"`:

```
Sub GetExcel()
    Dim myExcel As Excel.Application

    Set myExcel = GetObject(, "Excel.Application")
    myExcel.Caption = "Мій Microsoft Excel"
End Sub
```

Ця програма також змінює заголовок вікна поточного застосунку Microsoft Excel (у якому запущена програма) (Рисунок 77).

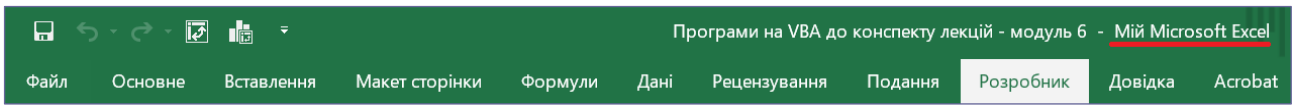


Рисунок 77 – Застосунок Microsoft Excel зі зміненою назвою у заголовку

Найважливішими властивостями класу `Application` є:

- `ActiveWorkbook` – містить посилання на активну робочу книгу – ту, з якою на даний час працює програма;
- `ActiveSheet` – містить посилання на активний аркуш активної робочої книги – той, з яким на даний час працює програма;
- `ActiveCell` – містить посилання на активну клітинку активного аркуша активної робочої книги – клітинка, з якою на даний час працює програма;
- `ActiveChart` – містить посилання на активну діаграму – розташовану на окремій вкладинці активної робочої книги діаграму, з якою на даний час працює програма;
- `ActiveWindow` – містить посилання на активне вікно – вікно, у якому відображена активна робоча книга;

Всі `Active`... властивості є властивостями тільки для читання. Наведений нижче фрагмент програми демонструє використання `Active`... властивостей У цьому прикладі також демонструється відкриття існуючої робочої книги (методи класу-колекції `Workbooks` детальніше будуть описані нижче):

```
    Dim myWbk As Workbook
```

```
    Set myWbk = Workbooks.Open(ThisWorkbook.Path _
    & Application.PathSeparator & "Робоча книга-зразок.xlsx")

    Debug.Print "Активна робоча книга: "; ActiveWorkbook.Name
    Debug.Print "Активний аркуш: "; ActiveSheet.Name
    Debug.Print "Активна клітинка: "; ActiveCell.Address
'    Debug.Print "Активна діаграма: "; ActiveChart.Name
    Debug.Print "Активне вікно: "; ActiveWindow.Caption
```

При виконанні для активного робочого аркушу програма виводить, наприклад:

```
Активна робоча книга:Робоча книга-зразок.xlsx
Активний аркуш:Аркуш1
Активна клітинка:$A$1
Активне вікно:Робоча книга-зразок.xlsx
```

Оскільки виводяться дані для активного аркушу (останнього, з яким працював користувач перед збереженням файлу робочої книги), а ним може бути або робочий аркуш, або діаграма (приклад демонструє випадок активного робочого аркушу), оператор виведення імені діаграми закоментований. У разі, якщо буде активна діаграма, його слід розкоментувати, а закоментувати оператор виведення адреси клітинки, оскільки клітинок на діаграмі немає.

- `ThisWorkbook` – містить посилання на поточну робочу книгу (у якій знаходиться виконувана програма);

- `Cells` – повертає об'єкт `Range`, який представляє усі клітинки активного аркушу активної робочої книги. Існує можливість визначати необхідні клітинки зазначенням аргументів для цієї властивості – послідовний номер рядка та послідовний номер стовпця, наприклад:

```
Application.Cells(1, 2).Value = "Привіт"
Application.Cells(1, 2).Font.Bold = True
```

Ці оператори вводять текстовий рядок у клітинку на перетині першого рядка та другого стовпця (B1) та задають їх напівжирний шрифт.

- `Columns` – повертає об'єкт `Range`, який представляє усі стовпці активного аркушу активної робочої книги. Існує можливість визначати необхідний стовпець зазначенням його послідовного номеру як аргументу;

- `Rows` – повертає об'єкт `Range`, який представляє усі рядки активного аркушу активної робочої книги. Існує можливість визначати необхідний рядок зазначенням його послідовного номеру як аргументу;

Наступний приклад задає напівжирний шрифт для усіх клітинок другого стовпця (B) та курсивний шрифт – для усіх клітинок другого рядка (2):

```
Application.Columns(2).Font.Bold = True
Application.Rows(2).Font.Italic = True
```

- `Range(Cell1, Cell2)` – повертає об'єкт `Range`, який являє собою клітинку або діапазон клітинок. Обов'язковий аргумент `Cell1` визначає адресу клітинки або діапазону клітинок у вигляді текстового рядка з адресами у вигляді

літера стовпця-номер рядка, необов'язковий параметр `Cell2` може містити адресу клітинки, що визначає розмір діапазону, наприклад:

```
Application.Range("A1", "B1").Select
Application.Range("A1:B1").Font.Color = vbRed
```

Наведений код виділяє та змінює колір шрифту для діапазону, що містить клітинки першого та другого стовпця першого рядка. Зауважимо, що у якості адрес також можуть використовуватися імена діапазонів (про них буде викладено нижче).

Існує можливість неявного виклику властивості `Range` – для цього використовуються квадратні дужки, у яких зазначається аргумент у вигляді тексту:

```
Application.["A1:B1"].Interior.ColorIndex = 37
```

Наведений код змінює колір фону для діапазону, що містить клітинки першого та другого стовпця першого рядка;

- `Selection` – повертає поточний виділений об'єкт на активному робочому аркуші, повертає `Nothing`, якщо нічого не виділено, ця властивість зазвичай використовується макрорекордером і майже не використовується при написанні програм;
- `Workbooks` - містить посилання на однойменну колекцію, яка включає робочі книги, відкриті у застосунку Microsoft Excel;
- `Sheets` – містить посилання на однойменну колекцію, яка включає робочі аркуші та аркуші діаграмам активної робочої книги;
- `Worksheets` – містить посилання на однойменну колекцію, яка включає робочі аркуші активної робочої книги;
- `Charts` – містить посилання на однойменну колекцію, яка включає аркуші діаграмам активної робочої книги;

Наведемо для прикладу фрагмент коду, що виводить імена усіх відкритих робочих книг у поточному екземплярі Microsoft Excel:

```
Dim tmpWbk As Workbook

Debug.Print "Відкриті робочі книги: "
For Each tmpWbk In Application.Workbooks
    Debug.Print tmpWbk.Name
Next
```

Програма виводить, наприклад:

```
Відкриті робочі книги:
PERSONAL.XLSB
Програми на VBA до конспекту лекцій - модуль 6.xlsm
Робоча книга-зразок.xlsx
```

Зверніть увагу, що окрім створених та відкритих робочих книг автоматично відкривається так *Особиста книга макросів* PERSONAL.XLSB (див. вище).

- Names – містить посилання на однойменну колекцію, що представляє всі іменовані діапазони у активній робочій книзі. За допомогою методу Add() колекції Names можливо визначати у робочій книзі іменовані діапазони, наприклад:

```
Names.Add Name:="ВхідніДані", RefersTo:="=Аркуш1!$a$1:$a$10"
Debug.Print "Адреса іменованого діапазону "; Chr(34); _
            Names("ВхідніДані").Name; Chr(34); Chr(32); _
            Names("ВхідніДані").RefersToRange.Address
```

Другий оператор виводить ім'я та адресу іменованого діапазону:

```
Адреса іменованого діапазону "ВхідніДані" $A$1:$A$10
```

Зверніть увагу на спосіб зазначення імені аркушу у текстовому рядку, що вказує адресу діапазону. Іменовані діапазони можуть використовуватися у формулах та вбудованих функціях Microsoft Excel.

- Dialogs – повертає посилання на однойменну колекцію об’єктів Dialog, яку можна використовувати для відображення діалогових вікон Excel (їх передбачено кілька сотень) і визначати реакцію на дії в них користувачів. Часто використовується властивість FileDialog, що представляє вікна, призначені для роботи з файлами (наприклад, вікно відкриття файлу). Працювати з цією властивістю можна так, як і у Microsoft Word (див. вище).

- Windows – містить посилання на однойменну колекцію, яка включає вікна (об'єкти Window), у яких відкриті робочі книги, часто використовується для активізації іншої відкритої робочої книги, наприклад:

```
Dim myWindow As Window

Set myWindow = Application.Windows("Інша робоча книга.xlsx")
```

Властивості цього об'єкту Window дозволяють програмно керувати вертикальною та горизонтальною прокруткою та виглядом вікна, у якому відкрита робоча книга.

Як і у випадку з Microsoft Word, якщо код VBA виконується в уже запущеному екземплярі Microsoft Excel і у коді зазначається звертання до властивості або метода без вказівки об'єкта, то вважається, що у якості об'єкта використовується об'єкт Application. Тому ці два рядки коду будуть рівнозначні:

```
Application.Workbooks.Add
```

та

```
Workbooks.Add
```

- WorkSheetFunction – дозволяє використовувати у програмі вбудовані функції Microsoft Excel, наприклад, наступний фрагмент програми заповнює діапазон випадковими числами та використовує вбудовані функції пошуку мінімального значення та отримання суми значень у діапазоні:

```
Dim myRange As Range
Dim tmpCell As Range
```

```
Set myRange = Worksheets("Аркуш1").Range("A2:A10")
For Each tmpCell In myRange
    tmpCell.Value = Int(Rnd * 10) + 1
Next

Debug.Print "Мінімальне значення у диапазоні " & myRange.Address _
          & " = " & WorksheetFunction.Min(myRange)
Debug.Print "Сума значень діапазону A2:A10 = "; _
            WorksheetFunction.Sum(Range("A2:A10"))
```

- `ReferenceStyle` – повертає або встановлює стиль адресації клітинок як елемент енумератора `XlReferenceStyle`: `xlA1=1` – стиль адресації за замовчуванням (літера стовпця-номер рядка) або `xlR1C1=-4150` – стиль адресації `R`-номер рядка-`C`-номер стовпця, наприклад `R1C1`.

- `DisplayAlerts` – дозволяє відключати (значення `False`) показ діалогових вікон з попередженнями, після завершення програми Microsoft Excel автоматично включає показ попереджень (встановлює значення `True`). Часто використовується для тимчасового відключення вікон попередження про необхідність збереження даних.

- `Interactive` – дозволяє заблокувати введення даних користувачем (як з клавіатури, так і за допомогою миші), окрім введення даних у вікна, що виводяться програмою. Зазвичай використовується, щоб користувач не зміг перешкодити роботі програмі.

Найважливішими методами класу `Application` можна назвати:

- `ActivateMicrosoftApp(Index)` – виконує запуск зазначеного застосунку Microsoft Office (або його активацію, якщо він вже запущений) і деяких інших застосунків Microsoft. Приймає як аргумент елемент енумератора `XlMSApplication`: `xlMicrosoftWord = 1`, `xlMicrosoftPowerPoint = 2`, `xlMicrosoftMail = 3`, `xlMicrosoftAccess = 4`, `xlMicrosoftFoxPro = 5`, `xlMicrosoftProject = 6`, `xlMicrosoftSchedulePlus = 7`.

- `Calculate()` – ініціює обрахування значень у клітинках усіх робочих аркушів усіх робочих книг (аналогічно працює метод `CalculateFull()`) або у клітинках зазначеного робочого аркушу поточної робочої книги або у зазначеному діапазоні зазначеного робочого аркушу поточної робочої книги. Цей метод використовується при вимкнутому автоматичному перерахунку виразів та функцій оператором:

```
Application.Calculation = xlManual
```

Приклади виклику цього методу:

```
Application.Calculation = xlManual
ActiveSheet.Cells(1, 1) = 10
ActiveSheet.Cells(2, 1) = 5
Worksheets("Аркуш1").Range("A3").Formula = "=A1+A2"
Worksheets("Аркуш1").Range("A4").Formula = "=A1-A2"
ActiveSheet.Cells(2, 1) = 7
```

```
'Обрахування значення у клітинці A3
ActiveSheet.Cells(3, 1).Calculate
'Обрахування значень у клітинках рядка 4
ActiveSheet.Rows(4).Calculate
ActiveSheet.Cells(2, 1) = 9

'Обрахування значень в усіх клітинках усіх робочих аркушів
Application.Calculate
Application.Calculation = xlCalculationAutomatic
```

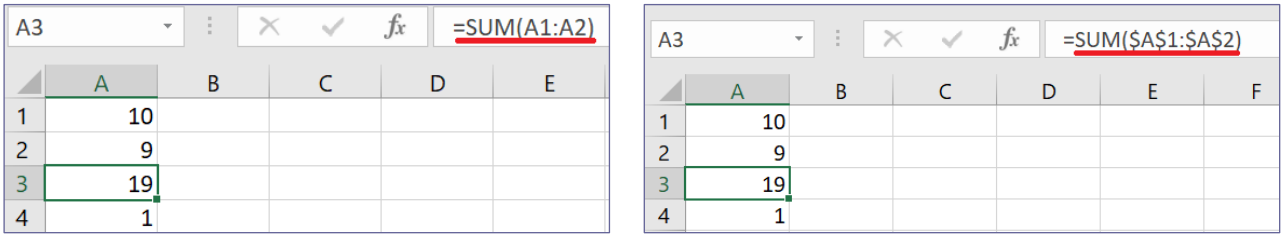
Зверніть увагу на повернення режиму автоматичного перерахунку у кінці програми. Зауважимо, що ручне обчислення використовують, зазвичай, коли воно займає багато часу і його часті виконання недоцільні, наприклад, при відлагодженні інших частин програми;

- `CalculateFullRebuild()` – ініціює обрахування значень у клітинках усіх аркушів усіх робочих книг та виконує перебудову формул (аналогічно занесенню всіх формул заново);

- `CheckAbort()` – зупиняє ініційоване `Calculate`...-методами обрахування;

- `ConvertFormula(Formula, FromReferenceStyle, ToReference Style, ToAbsolute, RelativeTo)` – дозволяє змінити тип посилань – абсолютні посилання в адресах на відносні або навпаки та/або стиль адресації (замість `xlA1` в `xlR1C1` або навпаки) у клітинках з формулами. Як обов'язкові параметри приймає `Formula` – формула у текстовому вигляді (починається з символу =) та `FromReferenceStyle`, що приймає як значення елемент енумератора `XlReferenceStyle`: `xlA1=1` – стиль адресації за замовчуванням (літера стовпця-номер рядка) або `xlR1C1=-4150` – стиль адресації `R`-номер рядка-`C`-номер стовпця, наприклад `R1C1`. Необов'язковий аргумент `ToReferenceStyle` може містити елемент енумератора `XlReferenceStyle` зі стилем адресації на який необхідно виконати зміну, у разі якщо він не зазначається, зміни стилю не відбувається. Необов'язковий аргумент `ToAbsolute` може містити елемент енумератора `XlReferenceType`: `xlAbsolute=1` – виконує конвертацію до абсолютних типів адресації рядка та стовпця, `xlAbsRowRelColumn=2` – виконує конвертацію до абсолютного типу адресації для рядка та відносного для стовпця, `xlRelRowAbsColumn=3` – виконує конвертацію до відносного типу адресації для рядка та абсолютного для стовпця, `xlRelative=4` – виконує конвертацію до відносних типів адресації рядка та стовпця. Найчастіше цей метод використовують для зміни типу посилань у формулах, так зміна на абсолютний тип посилань (Рисунок 78) дозволить копіювати формулу без зміни адрес, що вона використовує:

```
Worksheets("Аркуш1").Range("A3").Formula = "=SUM(A1:A2)"
Range("A3").Formula = Application.ConvertFormula("=SUM(A1:A2)", _
        XlReferenceStyle.xlA1, ToAbsolute:=XlReferenceStyle.xlA1)
```



а) *б)*

Рисунок 78 – Зміна типу адресації методом `ConvertFormula()`*: а) відносний стиль, б) абсолютний стиль*

`Evaluate(Name)` – виконує пошук різних об'єктів/значень за їх текстовими назвами та повертає ці об'єкти/значення. Існує неявний спосіб виклику методу `Evaluate(Name)` – вказати текстовий рядок-ім'я об'єкта у квадратних дужках, далі будуть наведені приклади для обох форм виклику методу. Обов'язковий параметр `Name` може приймати:

- ім'я формули або функції, записані як рядок, наприклад:

```
Worksheets("Аркуш1").Range("B1") = 0
Worksheets("Аркуш1").Range("B2") = 1
Worksheets("Аркуш2").Range("A1") = 10
Debug.Print "B1+B2 = "; Application.Evaluate("B1+B2")
Debug.Print "Cos(0) = "; Application.[COS(B1)]
Debug.Print "Аркуш1!B2 + Аркуш2!A1="; _
            [Аркуш1!B2 + Аркуш2!A1]
```

зверніть увагу на спосіб адресації клітинок на різних робочих аркушах. Якщо формула містить змінні-параметри, вони повинні бути рядками і додаватись через конкатенацію (`&`) – у такому разі неявний виклик методу `Evaluate(Name)` неможливий:

```
Dim myRange As Range

Set myRange = Worksheets("Аркуш1").Cells(2, 3)
myRange.Value = 5
Worksheets("Аркуш1").Cells(3, 3) = _
          Application.Evaluate("FACT(" & myRange.Value & ")")
```

- адресу клітинки в стилі `xlA1`, задану як рядок (повертається об'єкт `Range`):

```
Application.Evaluate("B5").Value = "Привіт"
Application.[B5].Value = "Привіт"
```

- адресу діапазону, задану як рядок (повертається об'єкт `Range`);

```
Application.Evaluate("B1:B5").Font.Italic = True
Application.[B1:B5].Font.Italic = True
```

- визначені у макросі імена (найчастіше – імена змінних) – повинні бути рядками:

```
Dim PiVal As String
```

```
PiVal = Str(WorksheetFunction.Pi() / 4)
Debug.Print "Pi/4 = "; Application.Evaluate(PiVal)
Debug.Print "Pi/4 = "; Application.[PiVal]
```

- визначені поіменовані діапазони:

```
Application.Names.Add "cellName", [C1]
Application.Evaluate("cellName") = 333
Debug.Print [cellName]
```

Наведений вище код оголошує поіменований діапазон з ім'ям `cellName` для діапазону з однієї клітинки `C1`, призначає йому значення та зчитує його до вікна *Immediate*.

- посилання на дані у інших робочих книгах (вони повинні бути відкритими на момент виклику методу `Evaluate(Name)`), наприклад:

```
Debug.Print Application.Evaluate("_
                        [Інша книга.xlsx]Аркуш2!A1:A5"));
Debug.Print Application.[[КнигаЗразок.xlsx]Аркуш1!A5]
```

Зауважимо, що метод не буде знаходити відповідні об'єкти, якщо в іменах файлів та аркушів не будуть відповідати правилам іменування елементів коду VBA (див. вище), що є незручним (не допускаються пробіли, дефіси тощо), тому частіше для доступу до даних у інших книгах вони спочатку відкриваються, повертаючи посилання оголошеній змінній типу `Workbook`, а потім виконується навігація по цій книзі (програмування таких дій буде розглянуте далі у цьому розділі).

- вказані їх ім'ям об’єкти діаграм, такі, як "Назва діаграми", "Легенда" тощо, наприклад:

```
ActiveWorkbook.Charts("Діаграма1")_
              .Evaluate("Назва діаграми").Font.Size = 22
```

Наведений вище код встановлює розмір шрифту, що використовується у діаграмі з ім’ям `Діаграма1`, яка міститься на окремому аркуші активної робочої книги.

- імена або послідовні номери елементів керування форми, доданих на робочий аркуш (існує можливість додавати елементи керування з меню *Розробник–Вставити–Елементи керування форми*) (Рисунок 79), наприклад,

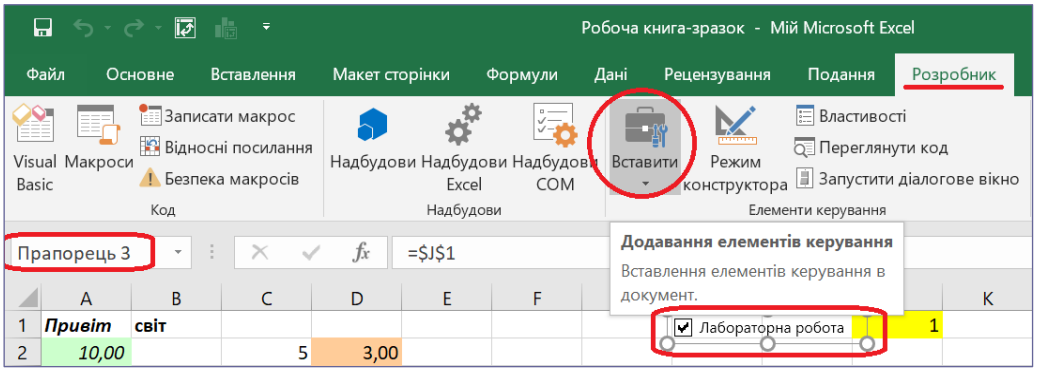


Рисунок 79 – Вставка елемента керування форми на робочий аркуш

Такий елемент керування може бути знайдений за його ім'ям, яке відображається у списку іменованих діапазонів вище рядка з назвами стовпців:

```
Debug.Print Evaluate("Прапорець 3").Caption; "="; _
            [Прапорець 3].Value
```

Для ситуації, показаній на Рисунок 79, у вікно *Immediate* буде виведено:

Лабораторна робота= 1

- значення елементів масиву (у тому числі багатовимірного), наприклад, код:

```
Dim myRange As Range

Set myRange = Worksheets("Аркуш1").Range("A1:D1")
myRange.Value = Application.Evaluate("{1, 2, 3, 4}")
Set myRange = Worksheets("Аркуш1").Range("A2:D4")
myRange.Value = [{1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12}]
```

встановлює значення елементів одновимірного масиву (елементи зазначаються у фігурних дужках через кому) як значення клітинок діапазону A1:D1 та значення елементів двовимірного масиву (елементи вимірів розділяються крапкою з комою) як значення клітинок діапазону A2:D4.

- `GetOpenFilename(FileFilter, FilterIndex, Title, ButtonText, MultiSelect)` – надає спрощений спосіб відображення діалогового вікна відкриття файлу, повертає повне ім'я обраного файлу без його фактичного відкриття або `False`, якщо файл не був обраним (був виконаний клік по кнопці *Скасувати*). Метод приймає необов'язкові параметри, які дозволяють налаштувати фільтрацію файлів, що відображаються у вікні – `FileFilter` та `FilterIndex`, кастомізувати заголовок вікна – `Title` та напис на кнопці вибору `ButtonText` (тільки для Macintosh-версії Microsoft Office), керувати можливістю множинного вибору файлів – `MultiSelect`. На Рисунок 80 показане вікно, яке відкривається при виконанні наступного коду:

```
Dim openWbk As Workbook
Dim filename As Variant

filename = Application.GetOpenFilename FileFilter:=_
  "Книга Excel (*.xlsx), *.xlsx, Книга Excel 97-2003 (*.xls), *.xls", _
  FilterIndex:=2, Title:="Відкриття робочої книги")

If Filename <> False Then
    Debug.Print filename
    Set openWbk = Workbooks.Open(filename)
End If
```

- `GetSaveAsFilename(InitialFilename, FileFilter, FilterIndex, Title, ButtonText)` – надає спрощений спосіб відображення діалогового вікна збереження файлу, повертає повне ім'я обраного файлу без його фактичного збереження або `False`, якщо файл не був обраним (був виконаний клік по кнопці *Скасувати*). Метод приймає необов'язкові параметри,

які дозволяють задати ім'я файлу під яким пропонується зберігати робочу книгу – `InitialFilename` (якщо не зазначене – використовується ім'я робочої книги), налаштувати фільтрацію файлів, що відображаються у вікні – `FileFilter` та `FilterIndex`, кастомізувати заголовок вікна – `Title` та напис на кнопці вибору `ButtonText` (тільки для Macintosh-версії Microsoft Office).

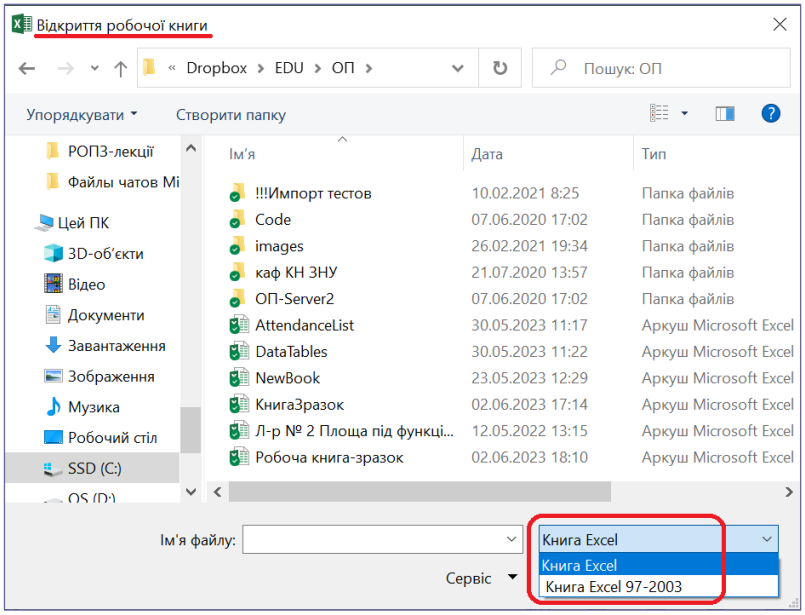


Рисунок 80 – Вікно відкриття файлу, налаштоване методом GetOpenFilename()

На Рисунок 81 показане вікно, яке відкривається при виконанні наступного коду:

```
Dim openWbk As Workbook
Dim filename As Variant

filename = Application.GetSaveAsFilename(InitialFileName:= _
        "Моя робоча книга", FileFilter:="Книга Excel
(*.xlsx), *.xlsx", _
        Title:="Збереження робочої книги")

If Filename <> False Then
    Debug.Print filename
    ActiveWorkbook.SaveAs(filename)
End If
```

- `GoTo(Reference, Scroll)` – виділяє і активізує зазначений необов'язковим параметром `Reference` діапазон (`Range`) або запускає зазначену цим же параметром процедуру (`Reference` приймає або об'єкт `Range`, або рядок з адресою діапазону в форматі `xlR1C1`, або ім'я-рядок процедури). Аргумент `True` для необов'язкового параметру `Scroll` виконує прокрутку аркушу так, щоб діапазон, зазначений параметром `Reference` відобразився у верхньому лівому куті вікна. У разі, якщо параметри не зазначені, активується останній використаний з цим методом діапазон без виконання прокрутки.

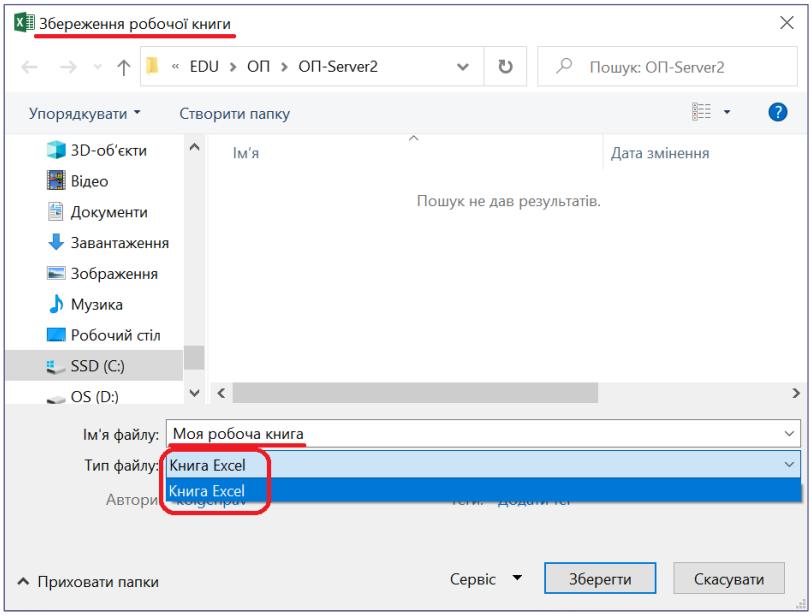


Рисунок 81 – Вікно збереження файлу, налаштоване методом GetSaveAsFilename()

Наприклад, оператор:

```
Application.Goto Reference:=ThisWorkbook.Worksheets("Аркуш1"). _
                 Range("A10"), Scroll:=True
```

виконує прокрутку аркушу `Аркуш1` поточної робочої книги до клітинки `A10`.

- `InputBox(Prompt, Title, Default, Left, Top, HelpFile, HelpContextID, Type)` – відображає діалогове вікно для введення даних користувачем, більшість параметрів співпадає з вбудованою функцією `InputBox()` VBA (див. вище). Параметри `Left`, `Top` відповідають параметрам `XPos`, `YPos`, параметр `HelpContextID` – параметру `Context`, параметр `Type` задає тип результату, що повертається методом (у разі, якщо він не зазначений, повертається рядок).

- `Intersect(Arg1, Arg2, ..., Arg30)` – повертає діапазон (`Range`), який представляє прямокутну область перетину двох або більше (до 30) інших діапазонів, перші два параметри обов'язкові. Якщо діапазони, які передаються як параметри, не перетинаються, метод повертає `Nothing`. Наприклад, у разі заповнення клітинок, як показано на Рисунок 82, оператор

```
Dim intersectRange As Variant

intersectRange = Intersect(Range("A2:A10"), _
                           Range("A5:D7")).Address
```

поверне `$A$5:$A$7`.

- `Union(Arg1, Arg2, ..., Arg30)` – дозволяє об'єднати два або більше (до 30) діапазонів у один, перші два параметри обов'язкові. Наприклад, у разі заповнення клітинок, як показано на Рисунок 82, оператор

```
Dim unionRange As Variant
```

```
unionRange = Union(Range("A2:A10"), _
                   Range("A5:D6")).Address
```

поверне

$A$2:$A$10,$A$5:$D$6.

	A	B	C	D
1	*Привіт*	світ		
2	*10*	11		3
3	*5*			4
4	*4*			5
5	*9*			6
6	*5*			7
7	*7*			8
8	*1*			9
9	*-3*			10
10	*0*			11

Рисунок 82 – Знаходження перетину та об'єднання двох діапазонів

- `Run()` – дозволяє виконати: процедуру VBA, макрос Microsoft Excel або процедуру або функцію в модулі DLL або XLL (і передати їй до 30 параметрів). Параметром методу може бути або рядок з ім'ям макросу, або об'єкт `Range`, що вказує, де знаходиться функція, або ідентифікатор зареєстрованої DLL або XLL-функції. Якщо використовується рядок, він буде оцінюватися у контексті активного робочого аркушу. Наприклад:

```
Run("GetExcel")
```

- `OnTime(EarliestTime, Procedure, LatestTime, Schedule)` – дозволяє запланувати запуск процедури у зазначений час або за розкладом або після спливання зазначеного часу. Обов'язковий параметр `EarliestTime` зазначає час, коли процедура буде запущена, обов'язковий параметр `Procedure` зазначає ім'я процедури. Необов'язковий параметр `LatestTime` визначає проміжок часу (зазвичай `EarliestTime` + кількість секунд) впродовж якого процедура може запускатися, якщо Microsoft Excel не може запустити процедуру у зазначений параметром `EarliestTime` час. Необов'язковий параметр `Schedule` може приймати значення `True` для активізації налаштованого методом `OnTime()` таймера (значення за замовчуванням), значення `False` деактивує таймер. Наведемо приклад коду, який буде запускати процедуру о 9 ранку кожного дня, якщо Microsoft Excel запущений та зможе її запустити. Якщо це йому не вдасться, він буде намагатися її запустити впродовж 30 хвилин:

```
Application.OnTime "09:00:00", "09:30:00", "MyMorningTask"
```

- `SendKeys(Keys, Wait)` – дозволяє емулювати натискання клавіш і передати їх в активне вікно програми. Обов'язковий параметр `Keys` визначає клавішу або клавіатурну комбінацію, яка емулюється, а необов'язковий параметр `Wait` дозволяє налаштувати, чи очікувати макросу результату емуляції (`True`), чи ні (`False`). Наступний код відкриває меню *File* (надається комбінація *Alt+F*):

```
Application.SendKeys() "%{f}"
```

Для клавіш *Ctrl* та *Shift* використовують символи ^ та +, відповідно.

- `Wait(Time)` – дозволяє призупинити роботу Microsoft Excel на зазначений обов'язковим параметром `Time` час (рядком або функціями VBA), метод повертає `True` після спливання часу призупинки. Наприклад, щоб призупинити Microsoft Excel на п'ять секунд, можна скористатися кодом:

```
If Application.Wait(Now + TimeValue("0:00:5")) Then
    MsgBox "П’ять секунд минуло"
End If
```

- `Quit()` – завершує роботу поточного екземпляру Microsoft Excel.

Зауважимо, що зазначення посилання `Application` при використанні більшості властивостей і методів цього об'єкта не є обов'язковим, тобто оператори:

```
Application.Evaluate("C1") = 333
```

та

```
Evaluate("C1") = 333
```

рівнозначні, у той же час зазначення посилання `Application` при використанні методу `GoTo()` є обов'язковим.

Клас `Application` підтримує велику кількість подій. Для їх використання необхідно у розділі Declarations об’єктного модуля (модуля класу або форми) оголосити об'єкт `Excel.Application` з ключовим словом `WithEvents`, наприклад, так:

```
Public WithEvents App As Excel.Application
```

У списку об'єктів з'явиться новий об'єкт `App` (тобто `Application`), для якого можна вибрати події і додавати код в обробники подій (Рисунок 83).

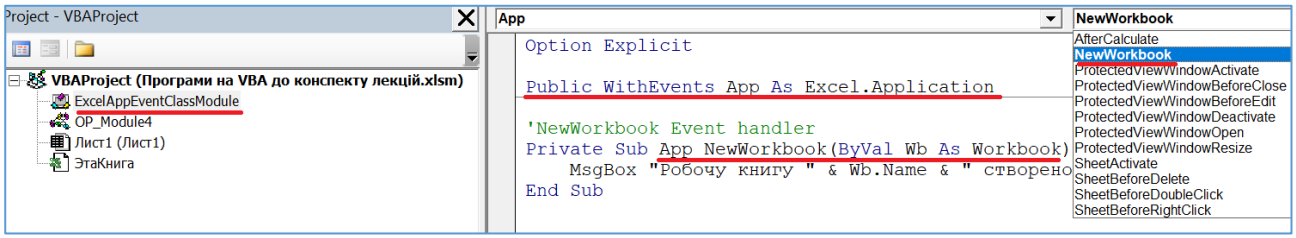


Рисунок 83 – Додавання об’єкту Excel з подіями до модуля класу

Тепер у звичайному модулі можна використовувати об’єкт класу, у якому оголошений об’єкт `Excel.Application` з подіями:

```
Sub CreateDocWithEvent()
    Dim eventApp As New ExcelAppEventClassModule
    Dim myWbk As Excel.Workbook

    Set eventApp.App = Excel.Application
    Set oWbk = eventApp.App.Workbooks.Add
    oWbk.Close
End Sub
```

Виконання цього коду приведе до створення нової робочої книги екземпляром Microsoft Excel з подіями `eventApp.App`, після чого автоматично буде запущений обробник події, який виведе інформаційне повідомлення (Рисунок 84).

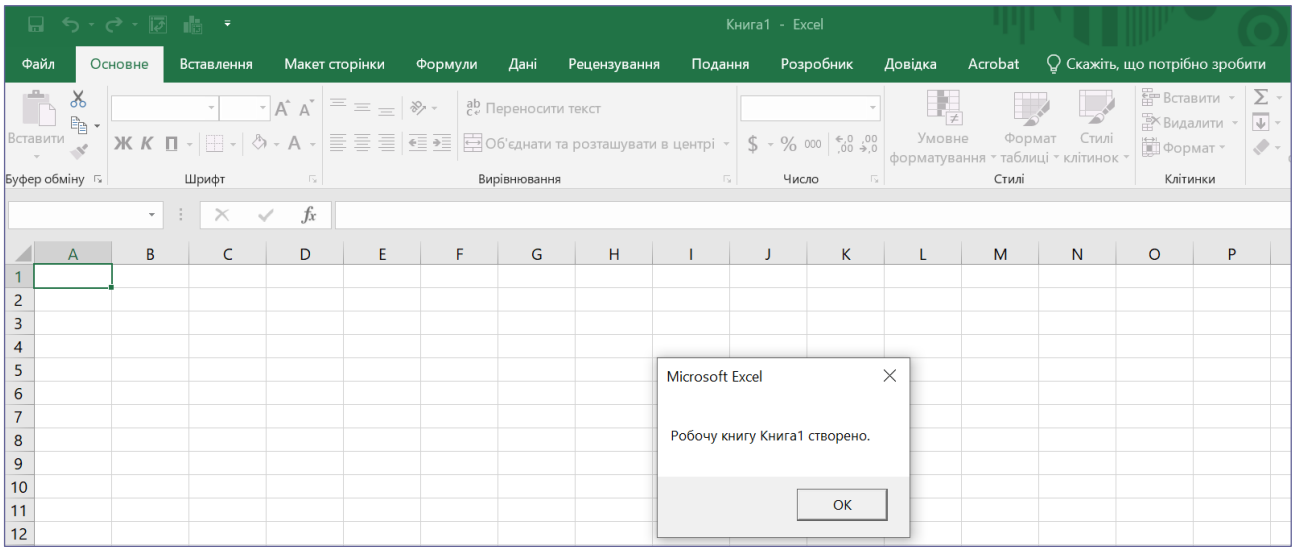


Рисунок 84 – Обробка події створення нової робочої книги

6.3 Робота з колекцією робочих книг (клас Workbooks) та робочими книгами (клас Workbook)

Якщо об'єкт `Application` асоціюється з запущеним застосунком Microsoft Excel, то кожній з яких відповідає екземпляр класу `Workbook`. Колекція `Workbooks` та екземпляри класу `Workbook` надають способи створення, відкриття, збереження робочої книги тощо [5].

До *найважливіших властивостей колекції* `Workbooks` можна віднести тільки властивість `Count`, яка містить кількість відкритих робочих книг, а *найважливіших методів* ця колекція має більше:

- `Add(Template)` – додає нову робочу книгу до колекції і тим самим створює її у поточному екземплярі `Application` (Microsoft Excel), який містить цю колекцію, метод має необов'язковий параметр `Template`, який приймає значення елемента енумератора `XlWBATemplate`, з яких основні: `xlWBATWorksheet=-4167` – створює робочу книгу з одним робочим аркушем, `xlWBATChart=-4109` - створює робочу книгу з одним аркушем діаграми. Також параметр `Template` може приймати повне ім'я вже існуючої робочої книги, у такому випадку створюється її копія. Якщо параметр не зазначається створюється нова робоча книга з кількістю робочих аркушів, зазначених у налаштуванні *Файл–Параметри–Загальні–Під час створення нових книг–Включити цю кількість аркушів*. Наприклад, код

```
Dim myExcel As New Excel.Application
Dim newWbk As Workbook, copyWbk As Workbook
Dim oneWkshtWbk As Workbook, oneChartWbk As Workbook
```

```
myExcel.Visible = True
Set newWbk = myExcel.Workbooks.Add()
Set copyWbk = myExcel.Workbooks.Add(ThisWorkbook.Path _
        & Application.PathSeparator & "AttendanceList.xlsx")
Set oneWkshtWbk = myExcel.Workbooks.Add(xlWBATWorksheet)
Set oneChartWbk = myExcel.Workbooks.Add(xlWBATChart)

oneChartWbk.Close SaveChanges:=False
myExcel.Workbooks.Close
myExcel.Quit
```

створює чотири робочих книги: зі стандартною кількістю робочих аркушів (`newWbk`), копію існуючої книги (`copyWbk`), з одним робочим аркушем (`copyWbk`) та з одним аркушем діаграми (`oneChartWbk`) також продемонстровано роботу методу `Worksheets Close()`.

- `Close()` – закриває усі робочі книги колекції `Workbooks`, не має параметрів, для книг зі змінами виводиться діалогове вікно збереження файлу.

- `Open()` – відкриває робочу книгу з файлу та додає її до колекції `Workbooks`, окрім інших, має один обов'язковий параметр – повне ім'я файлу, що містить робочу книгу. Приклад відкриття існуючої робочої книги:

```
Dim myWbk As Workbook

Set myWbk = Workbooks.Open(ThisWorkbook.Path _
    & Application.PathSeparator & "Робоча книга-зразок.xlsx")
```

Окрім стандартного методу відкриття робочої книги `Open()` клас `Workbooks` містить додаткові методи [5]:

- `OpenText()` – відкриває текстовий файл та виконує його парсинг до робочої книги Microsoft Excel. Метод має один обов'язковий параметр `FileName` – повне ім'я текстового файлу, серед інших параметрів найчастіше використовуються: `DataType` – вказує тип поділу даних, задається елементом енумератора `XlTextParsingType` – `xlDelimited=1` – у разі використання символу-роздільника між даними, або `xlFixedWidth=2` – у разі, коли в файлі дані розташовані у стовпцях фіксованої ширини. У разі використання символу-роздільника між даними, необхідно встановити значення `True` параметру, що його визначає: `Tab` (табулятор) або `Semicolon` (крапка з комою) або `Comma` (кома) або `Space` (пробіл) або `Other` (для іншого, аніж зазначені параметрами символу). В останньому випадку параметру `OtherChar` необхідно передати символ, який використовується як роздільник. Наведений нижче код відкриває файл з даними у форматі *даних, розділених комою (Comma Separated Values – CSV)*. Оскільки метод `OpenText()` не повертає посилання на створену робочу книгу, отримати його можна за допомогою властивості `ActiveWorkbook` класу `Application`:

```
Dim txtWbk As Workbook

Workbooks.OpenText FileName:=ActiveWorkbook.Path _
```

```
            & Application.PathSeparator & "TextData.csv", _
            DataType:=xlDelimited, Comma:=True
    Set txtWbk = ActiveWorkbook
```

- `OpenXML(FileName, Stylesheets, LoadOption)` – відкриває файл даних у форматі *розширеної мови розмітки (Extensible Markup Language, скорочено – XML)* та виконує їх парсинг до робочої книги Microsoft Excel. Метод має один обов'язковий параметр `FileName` – повне ім'я текстового файлу та необов'язкові: `Stylesheets` – визначає інструкції *мови трансформації даних XML (Extensible Stylesheet Language Transformations – XSLT)* та `LoadOption` – визначає яким чином Microsoft Excel буде представляти дані XML-файлу у робочій книзі, може бути одним з елементів енумератора `XlXmlLoadOption`: `xlXmlLoadPromptUser=0` – запитує користувача, як представляти дані, `xlXmlLoadOpenXml=1` – відкриває дані у рядках робочого аркушу, заголовки формує з тегів XML, `xlXmlLoadImportToList=2` – відкриває дані на робочому аркуші у форматованій таблиці з заголовками та автофільтром, `xlXmlLoadMapXml=3` – відкриває на робочому аркуші вікно структури з назвами тегів з можливістю вручну сформувати таблицю з необхідними даними перетягуванням необхідних тегів на аркуш та виконанням команди *Оновити* з контекстного меню для заповнення таблиці даними.

Для XML файлу [31], дані якого показані на Рисунок 85, наведений нижче код виконує їх імпорт у робочу книгу, показану на Рисунок 86.

Books.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <catalog>
3     <book id="bk101">
4        <author>Gambardella, Matthew</author>
5        <title>XML Developer's Guide</title>
6        <genre>Computer</genre>
7        <price>44.95</price>
8        <publish_date>2000-10-01</publish_date>
9        <description>An in-depth look at creating applications
10       with XML.</description>
11    </book>
12    <book id="bk102">
13       <author>Ralls, Kim</author>
14       <title>Midnight Rain</title>
15       <genre>Fantasy</genre>
16       <price>5.95</price>
17       <publish_date>2000-12-16</publish_date>
18       <description>A former architect battles corporate zombies,
19       an evil sorceress, and her own childhood to become queen
```

Рисунок 85 – XML-файл, що використовується методом OpenXML()

```
Dim xmlWbk As Workbook

Set xmlWbk = Workbooks.OpenXML(FileName:=ActiveWorkbook.Path _
             & Application.PathSeparator & _
             "Books.xml", LoadOption:=xlXmlLoadImportToList)
```

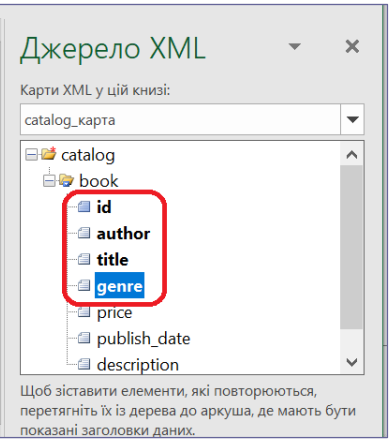
та

```
Set xmlWbk = Workbooks.OpenXML(FileName:=ActiveWorkbook.Path _
             & Application.PathSeparator & _
             "Books.xml", LoadOption:=xlXmlLoadMapXml)
```

	A	B	C	D	E	F	G	H
1	id	author	title	genre	price	publish_date	description	
2	bk101	Gambardella, Matthew	XML Developer's Guide	Computer	44,95	01.10.2000	An in-depth look at creating applications with XML.	
3	bk102	Ralls, Kim	Midnight Rain	Fantasy	5,95	16.12.2000	A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.	
4	bk103	Corets, Eva	Maeve Ascendant	Fantasy	5,95	17.11.2000	After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.	
5	bk104	Corets, Eva	Oberon's Legacy	Fantasy	5,95	10.03.2001	In post-apocalypse England, the mysterious agent known only as Oberon helps to create a new life for the inhabitants of London. Sequel to Maeve Ascendant.	
							The two daughters of Maeve, half-sisters,	

а)

	A	B	C	D	E
1	id	author	title	genre	
2	bk101	Gambardella, Matthew	XML Developer's Guide	Computer	
3	bk102	Ralls, Kim	Midnight Rain	Fantasy	
4	bk103	Corets, Eva	Maeve Ascendant	Fantasy	
5	bk104	Corets, Eva	Oberon's Legacy	Fantasy	
6	bk105	Corets, Eva	The Sundered Grail	Fantasy	
7	bk106	Randall, Cynthia	Lover Birds	Romance	
8	bk107	Thurman, Paula	Splish Splash	Romance	
9	bk108	Knorr, Stefan	Creepy Crawlies	Horror	
10	bk109	Kress, Peter	Paradox Lost	Science Fiction	
11	bk110	O'Brien, Tim	Microsoft .NET: The Programming Bible	Computer	
12	bk111	O'Brien, Tim	MSXML3: A Comprehensive Guide	Computer	
13	bk112	Galos, Mike	Visual Studio 7: A Comprehensive Guide	Computer	
14					



б)

Рисунок 86 – Робоча книга з імпортованими даними XML файлу: а) при LoadOption:=xlXmlLoadImportToList, б) при LoadOption:=xlXmlLoadMapXml

- `OpenDatabase (FileName, CommandText, CommandType, BackgroundQuery, ImportDataAs)` – завантажує дані з файлу бази даних Microsoft Access та виконує їх парсинг до робочої книги Microsoft Excel. Метод має один обов'язковий параметр `FileName` – повне ім'я файлу бази даних. Необов'язкові параметри включають:
  - `CommandText` – записаний як рядок оператор SQL запиту, що буде надавати дані або як ім'я об'єкта бази даних (таблиці, запиту);
  - `CommandType` – тип запиту, приймає елементи енумератора `XlCmdType`, найчастіше: `xlCmdSql` – у разі, якщо параметр `CommandText` приймає оператор SQL запиту, або `xlCmdTable` – у разі, якщо параметр `CommandText` приймає ім'я об'єкта бази даних (таблиці, запита);
  - `BackgroundQuery` – при значенні `True` дозволяє налаштувати фонове виконання запиту до бази даних (значення за замовчуванням – `False`);
  - `ImportDataAs` – визначає формат результату імпорту даних до робочої книги, приймає як значення елемент енумератора

`XlImportDataAs: xlQueryTable=0` – результат у вигляді звичайної таблиці та `xlPivotTableReport=1` – результат у вигляді зведеної таблиці.

Наведений нижче код виконує імпорт даних запиту Замовлення товару з навчальної бази даних Northwind.accdb СУБД Microsoft Access [32]:

```
Dim dbWbk As Workbook

Set dbWbk = Workbooks.OpenDatabase(FileName:=ActiveWorkbook.Path _
      & Application.PathSeparator & "Northwind.accdb", _
      CommandText:="Замовлення товару", CommandType:=xlCmdTable, _
      BackgroundQuery:=True, ImportDataAs:=xlPivotTableReport)
```

Результат виконання цього оператору з подальшою побудовою звіту зведеної таблиці представлений на Рисунок 87.

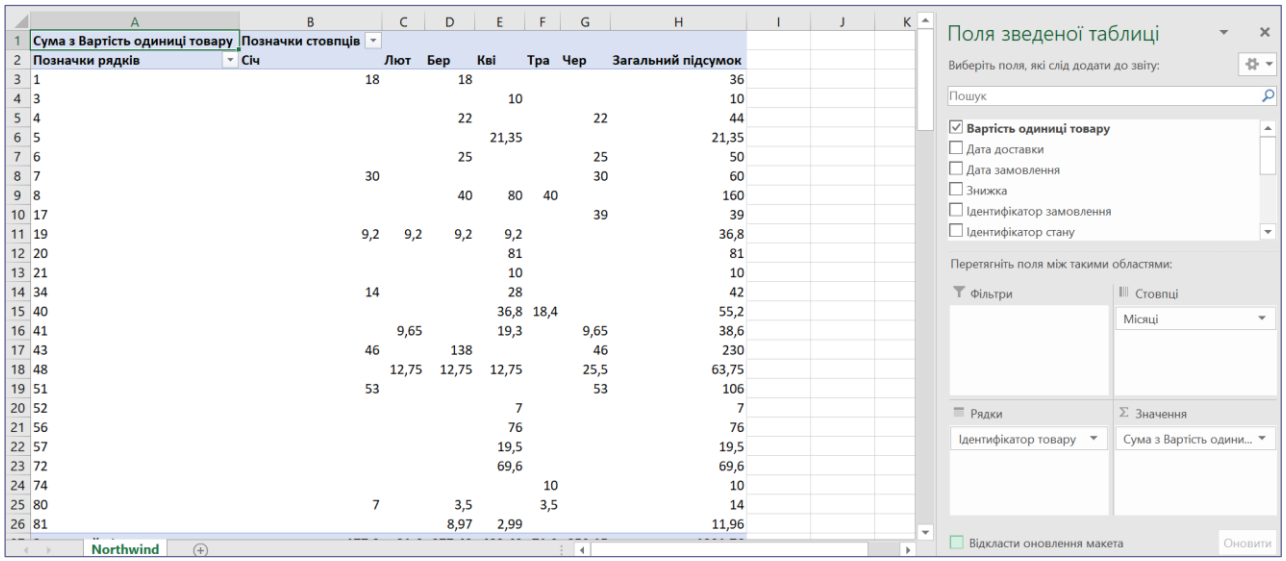


Рисунок 87 – Робоча книга з імпортованими даними запиту бази даних у вигляді звіту зведеної таблиці

Подій для колекції `Workbooks` не передбачено.

Розглянемо клас `Workbook`, який представляє робочу книгу Microsoft Excel. Отримати посилання на об'єкт цього класу можна кількома способами:

- зазначивши номер або ім'я необхідної робочої книги як індекс колекції `Workbooks`, наприклад:

```
Debug.Print Workbooks("Робоча книга-зразок.xlsx").FullName
```

- використовуючи властивість `Application.ActiveWorkbook` після відкриття або активації робочої книги За її допомогою ми звертаємося до активної в цей момент книги:

```
Workbooks("Робоча книга-зразок.xlsx").Activate
Debug.Print ActiveWorkbook.Name
```

- використовувати властивість `Application.ThisWorkbook` у разі, якщо необхідна робоча книга містить програму, що виконується:

```
Debug.Print ThisWorkbook.Name
```

До найважливіших властивостей класу `Workbook` можна віднести [5]:

- `ActiveSheet` – містить посилання на активний аркуш активної робочої книги (об'єкт `Workbook`);
- `ActiveChart` – містить посилання на активну діаграму активної робочої книги (об'єкт `Chart`);
- `WorkSheets` – містить послання на однойменну колекцію робочих аркушів у поточній робочій книзі;
- `Charts` – містить послання на однойменну колекцію діаграм на окремих аркушах у поточній робочій книзі;
- `Sheets` – містить послання на однойменну колекцію робочих аркушів та діаграм на окремих аркушах у поточній робочій книзі;
- `Names` - повертає колекцію усіх іменованих діапазонів усіх робочих аркушів поточної робочої книги;
- `Name` – містить рядок – ім'я файлу робочої книги без шляху до нього;
- `FullName` – містить рядок – повне ім'я файлу робочої книги з шляхом до нього;
- `Path` – містить рядок – шлях до файлу книги, для нової робочої книги ця властивість дорівнює пустому рядку (" ");
- `CodeName` – містить рядок – значення властивості `Name` об'єкта, яке зазначається у вікні *Properties* цього об'єкта (Рисунок 88), може використовуватися у коді програм як посилання на відповідний об'єкт:

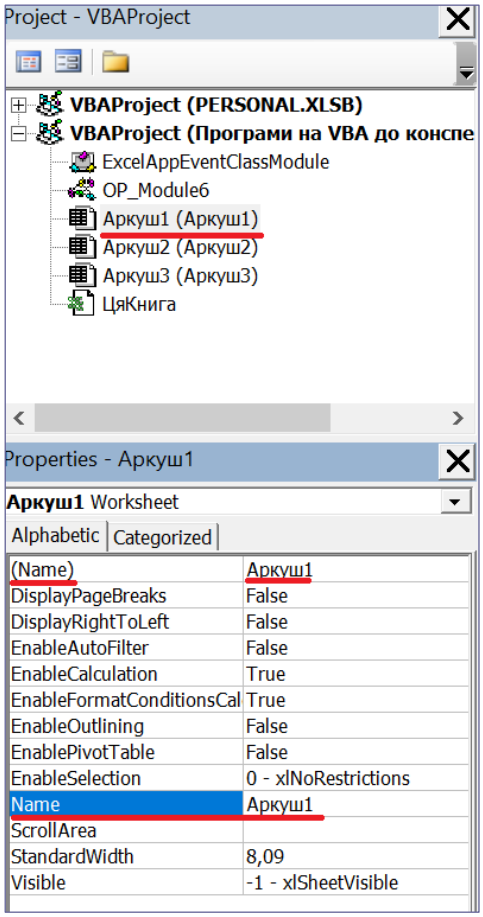


Рисунок 88 – Кодове ім'я робочої книги у вікні проекту

```
Dim myWbk As Workbook

Set myWbk = Workbooks.Open(ThisWorkbook.Path _
        & Application.PathSeparator & "Робоча книга-зразок.xlsx")
Debug.Print Аркуш1.Range("A1")
```

- `Saved` – повертає `True` у разі, якщо поточна робоча книга не була змінена після останнього зберігання, якщо встановити для цієї властивості `True`, змінена робоча книга не буде збережена і не буде виводитись діалогове вікно з пропозицією її збереження:

```
ActiveWorkbook.Saved = True
ActiveWorkbook.Close
```

До *найважливіших методів об'єкта* `Workbook` можна віднести:

- `Activate()` – робить активною робочу книгу, для якої викликається цей метод;
- `Save()` – зберігає поточну робочу книгу у тому ж файлі, з якого вона була відкриті, для нової робочої книги відкриває діалогове вікно її збереження;
- `SaveAs()` – дозволяє зберегти створену або відкриту робочу книгу у новому файлі. Обов'язковим параметром є `Filename` – повне ім'я файлу під яким буде збережена книга. Найбільш використовувані необов'язкові додаткові параметри:
  - `FileFormat` – формат файлу, у якому буде збережена робоча книга, найбільш використовувані формати наведені у Таблиця 12.

Таблиця 12 – Формати файлу, у якому зберігаються дані робочої книги (елементи енумератора `XlFileFormat`)

Елемент	Значення	Опис	Розширення файлів
xlOpenXMLWorkbook	51	Open XML Workbook	*.xlsx
xlExcel8	56	Excel 97-2003 Workbook	*.xls
xlWorkbookNormal	-4143	Workbook normal	*.xls
xlOpenXMLWorkbookMacroEnabled	52	Open XML Workbook Macro Enabled	*.xlsm
xlOpenXMLTemplate	54	Open XML Template	*.xltx
xlTemplate	17	Excel Template format	*.xlt
xlOpenXMLTemplateMacroEnabled	53	Open XML Template Macro Enabled	*.xltm
xlCSV	6	CSV	*.csv
xlHtml	44	HTML format	*.htm; *.html

  - `ConflictResolution` – визначає, як будуть вирішуватися конфлікти зміни даних, якщо книга відкрита декількома користувачами відразу (shared workbook). Є можливість зробити так, щоб локальні зміни вносились автоматично (`xlLocalSessionChanges`), зміни віддалених користувачів вносились автоматично

(xlOtherSessionChanges) або з'являлось діалогове вікно з можливістю визначити пріоритет змін (xlUserResolution). Однак для того, щоб не з'являлося діалогове вікно підтвердження перезапису файлу, необхідно відключити перед викликом методу SaveAs() виведення діалогових вікон, а після - включити:

```
Dim myWbk As Workbook

Application.DisplayAlerts = False
myWbk.SaveAs filename:=ThisWorkbook.Path _
    & Application.PathSeparator & "NewBook.xlsx", _
      FileFormat:=xlOpenXMLWorkbook, _
      ConflictResolution:=xlLocalSessionChanges
Application.DisplayAlerts = True
```

- Close(SaveChanges, FileName, RouteWorkbook) – закриває поточну робочу книгу, має необов'язкові параметри: SaveChanges – у разі значення True, зміни робочої книги записуються при її закритті, у разі значення False – зміни ігноруються, FileName – при закритті робочої книги, вона зберігається у файлі, зазначеному цим параметром, RouteWorkbook – дозволяє налаштовувати черговість пересилки робочої книги користувачам для її обробки.

Для використання обробників подій об'єкту Workbook необхідно його оголосити як об'єкт з подіями у модулі класу або форми, аналогічно об'єкту Application (див. вище). Але для робочої книги існує ще одна можливість: для кожної робочої книги у вікні *Провідника проєктів* Редактора Visual Basic відображається спеціальний *модуль класу* поточної робочої книги ЦяКнига, який містить значення властивостей поточної робочої книги та у якому можна розміщувати обробники подій поточної робочої книги (Рисунок 89).

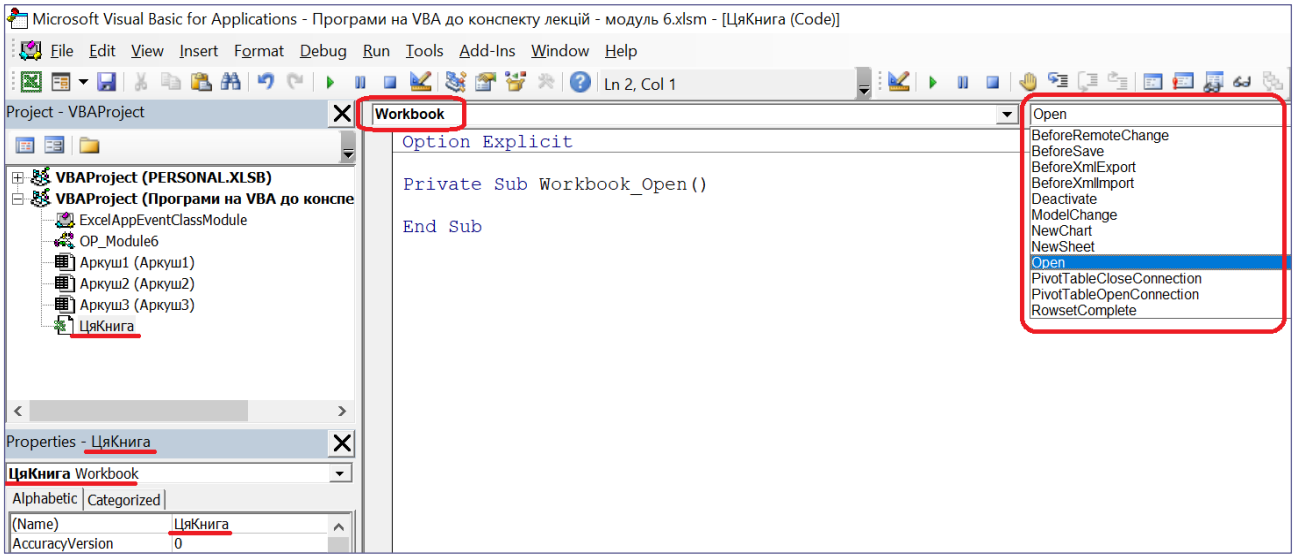


Рисунок 89 – Організація доступу к обробникам подій поточної робочої книги

6.4 Робота з колекцією робочих аркушів (класи Sheets, Worksheets) та робочими аркушами (клас Worksheet)

Як зазначалося вище, властивості-колекції об'єкта `Application` `Sheets` та `Worksheets` відрізняються тим, що елементами `Sheets` можуть бути не тільки робочі аркуші (екземпляри класу `Worksheet`), але й діаграми на окремих аркушах (екземпляри класу `Chart`). Зауважимо, що класу `Sheet` і, відповідно, його об'єктів моделлю Microsoft Excel не передбачено [5].

До *найважливіших властивостей класів `Sheets` та `Worksheets`* можна віднести стандартні для усіх колекцій: `Item` – індекс елемента колекції та `Count` – кількість елементів колекції.

До *найважливіших методів класів `Sheets` та `Worksheets`* можна віднести:

- `Add(Before, After, Count, Type)` – додає новий робочий аркуш (або діаграму на окремому аркуші для `Sheets`), приймає необов'язкові параметри: `Before` – об'єкт, шо ідентифікує аркуш, перед яким додається новий, `After` – об'єкт, шо ідентифікує аркуш, після якого додається новий, `Count` – кількість аркушів, що додаються, `Type` – тип аркушу, що додається – приймає елементи енумератора `XlSheetType`: найчастіше – значення `xlWorksheet= -4167` – для звичайного аркушу та `xlChart= -4109` – для аркушу діаграми. У разі, якщо аргументу методу не передаються, новий аркуш буде розміщений першим;
- `Delete()` – видаляє поточний аркуш, перед видаленням виводить вікно підтвердження видалення. У разі підтвердження видалення, метод повертає `True`, у разі скасування підтвердження, методо повертає `False`;
- `Copy(Before, After)` – виконує копіювання поточного робочого аркушу (або діаграми на окремому аркуші для `Sheets`) до іншого місця у переліку аркушів поточної робочої книги або до нової робочої книги. Необов'язкові параметри `Before` та `After` ідентифікують аркуш, перед/після якого копіюється поточний. Якщо метод викликається без параметрів, поточний аркуш копіюється до нової робочої книги;
- `Move(Before, After)` – виконує переміщення поточного робочого аркушу (або діаграми на окремому аркуші для `Sheets`) до іншого місця у переліку аркушів поточної робочої книги або до нової робочої книги. Необов'язкові параметри `Before` та `After` ідентифікують аркуш, перед/після якого переміщується поточний. Якщо метод викликається без параметрів, поточний аркуш переміщується до нової робочої книги;
- `FillAcrossSheets(Range, Type)` – копіює діапазон `Range` з визначеного робочого аркушу поточної робочої книги до таких самих діапазонів усіх інших робочих аркушів цієї книги. Параметр `Type` приймає як значення елементи енумератора `XlFillWith`: `xlFillWithAll=-4104` – копіює вміст діапазону та ознаки форматування, `xlFillWithContents=2` – копіює тільки

вміст діапазону, xlFillWithFormats=-4122 – копіює тільки ознаки форматування діапазону.

Наведемо приклад коду, що використовує методи класів Sheets та Worksheets:

```
Sub WorksheetsMethods()
    Dim myWbk As Workbook, newWbk As Workbook
    Dim mySheet As Worksheet
    Dim i As Integer

    Set myWbk = Workbooks.Open(ThisWorkbook.Path _
         & Application.PathSeparator & "Робоча книга-зразок.xlsx")
    'Додання нового аркушу та встановлення йому імені
    Set mySheet = myWbk.Worksheets.Add()
    mySheet.Name = "Новий аркуш"

    'Активація іншого аркушу
    Sheets("Аркуш1").Activate
    Debug.Print ActiveSheet.Name


    'Переміщення аркушу у кінець переліку аркушів
    myWbk.Worksheets("Новий аркуш").Move _
                    After:=myWbk.Sheets(myWbk.Sheets.Count)
    'Копіювання аркушу до нової робочої книги
    myWbk.Worksheets("Новий аркуш").Copy
    Set newWbk = ActiveWorkbook

    For i = 1 To ActiveWorkbook.Worksheets.Count
        Debug.Print ActiveWorkbook.Worksheets.Item(i).Name
    Next i

   'Копіювання діапазону на інші робочі аркуші
    myWbk.Activate
    Worksheets.FillAcrossSheets _
            Worksheets("Test").Range("A1:D10"), xlFillWithContents

    'Видалення робочого аркушу з відключенням діалогових вікон
    Application.DisplayAlerts = False
    mySheet.Delete
    Application.DisplayAlerts = True

    newWbk.Close SaveChanges:=False
    myWbk.Close SaveChanges:=False
End Sub
```

Якщо перейти до характеристики робочих аркушів, то нагадаємо, що об'єкт Worksheet одночасно є елементом двох колекцій: Sheets та Worksheets. До *найважливіших властивостей класу Worksheet* можна віднести:

- Cells – повертає об'єкт Range, який представляє усі клітинки поточного робочого аркушу та дозволяє конкретизувати клітинку зазначенням номеру рядка та номеру стовпця, наприклад, оператор:

```
Worksheets("Новий аркуш").Cells(1, 2).Font.Name = "Arial"
```

встановлює шрифт Arial для клітинки з адресою B1 на робочому аркуші Новий аркуш;

- `Rows` – повертає об'єкт `Range`, який представляє рядок поточного робочого аркушу та дозволяє конкретизувати його зазначенням номеру рядка, наприклад, оператор:

```
Worksheets("Новий аркуш").Rows(1).Font.Italic = True
```

встановлює курсивний шрифт для клітинок рядка з номером 1;

- `Columns` – повертає об'єкт `Range`, який представляє стовпець поточного робочого аркушу та дозволяє конкретизувати його зазначенням номеру стовпця, наприклад, оператор:

```
Worksheets("Новий аркуш").Columns(2).Font.Bold = True
```

встановлює напівжирний шрифт для клітинок стовпця B;

- `Range` - повертає діапазон клітинок (об'єкт `Range`), за допомогою якого можна виконувати маніпуляції з вмістом клітинок діапазону, клас `Range` буде розглянутий нижче у цій темі;

- `Name` – містить ім'я робочого аркушу;

- `Index` – повертає порядковий номер робочого аркушу у колекції `Worksheets`, наприклад:

```
Dim mySheet As Worksheet

Set mySheet = myWbk.Worksheets.Add()
mySheet.Name = "Новий аркуш"
Debug.Print Worksheets("Новий аркуш").Index
```

- `Next` – містить посилання на наступний робочий аркуш у поточній робочій книзі, наприклад, оператор

```
Worksheets("Новий аркуш").Next.Activate
```

активує наступний робочий аркуш за робочим аркушем Новий аркуш;

- `Previous` – повертає посилання на попередній робочий аркуш у поточній робочій книзі, наприклад, оператор:

```
Debug.Print ActiveSheet.Previous.Name
```

виводить ім'я попереднього робочого аркушу до активного аркушу поточної робочої книги;

- `Type` – містить тип поточного аркушу як елемент енумератора `XlSheetType`: найчастіше – значення `xlWorksheet= -4167` – для звичайного аркушу та `xlChart= -4109` – для аркушу діаграми;

- `UsedRange` – повертає об'єкт `Range`, який представляє максимальний за розміром діапазон, утворений клітинками робочого аркушу, у яких є дані, наприклад, оператор:

```
Debug.Print Worksheets("Test").UsedRange.Address
```

виведе до вікна *Immediate* $A$1:$J$10 для робочого аркушу Test активної робочої книги, наведеного на Рисунок 90;

- `Visible` – дозволяє приховувати/відображати поточний робочий аркуш у переліку аркушів робочої книги, може приймати як значення `False/True` або елемент енумератора `XlSheetVisibility`: `xlSheetVisible=-1` – відображає робочий аркуш, `xlSheetHidden=0` – приховує робочий аркуш, але користувач зможе його відобразити командою контекстного меню, `xlSheetVeryHidden=2` – приховує робочий аркуш з забороною його відображення користувачем (він може бути відображений тільки встановленням у відповідне значення цієї властивості програмою). Зазначимо, що можна перед виконанням програми сховати робочий аркуш, щоб користувач не бачив маніпуляції, до яких приводить виконання програми, а перед завершенням програми відобразити його вже у результуючому вигляді.

	A	B	C	D	E	F	G	H	I	J	K
1	***Привіт***	**світ**						☑ Лабораторна робота		1	
2	*10,00*		5	3,00							
3	*5,00*		120	4,00							
4	*4,00*			5,00							
5	*9,00*			6,00							
6	*5,00*			7,00							
7	*7,00*			8,00							
8	*1,00*			9,00							
9	*-3,00*			10,00							
10	*0,00*			11,00							
11											

Рисунок 90 – Приклад робочого аркушу з даними, які обробляються методом UsedRange()

До *найважливіших методів класу* `Worksheet` можна віднести:

- `Activate()` – робить активним робочий аркуш, для якого викликається цей метод;

- `Calculate()` – ініціює обрахування значень у клітинках усіх аркушів усіх відкритих робочих книг або у клітинках зазначеного робочого аркушу робочої книги або у зазначеному діапазоні зазначеного робочого аркушу робочої книги. Цей метод використовується при вимкнутому автоматичному перерахунку виразів та функцій на робочому аркуші оператором:

```
mySheet.EnableCalculation = False
```

Наступний код демонструє "ручне" перерахування значення формули =FACT(C2), що розраховує факторіал, у клітинці з адресою C3 після зміни значення її аргументу у клітинці C2:

```
ActiveSheet.EnableCalculation = False
ActiveSheet.Cells(2, 3) = 6
'Дивись на результат формули у клітинці C3
ActiveSheet.EnableCalculation = True
```

```
    ActiveSheet.Calculate
    Debug.Print "Факторіал від 6="; mySheet.Cells(3, 3)
```

Оскільки автоматичний перерахунок відключається, ми не побачимо зміни значення формули після виконання другого оператора. Нагадаємо, що клас `Application` також містить метод `Calculate()` (див. вище), але у такому разі метод застосовується для перерахування у клітинках усіх робочих аркушів усіх робочих книг.

- `Evaluate(Name)` – виконує пошук об'єктів/значень за їх текстовими назвами у контексті поточного робочого аркушу та повертає ці об'єкти/значення. Аргументами параметра `Name` можуть бути такі ж, як і для методу `Evaluate(Name)` класу `Application` (див. вище). Наведемо код програми, яка демонструє використання методу з різними типами параметру `Name`:

```
Sub WorksheetEvaluateMethod()
    Dim myWbk As Workbook
    Dim mySheet As Worksheet
    Dim PiVal As String
    Dim myRange As Range

    Set myWbk = Workbooks.Open(ThisWorkbook.Path _
         & Application.PathSeparator & "Робоча книга-зразок.xlsx")
    Set mySheet = myWbk.Worksheets("Аркуш1")

    mySheet.Range("B1") = 0
    mySheet.Range("B2") = 1
    'Метод працює і для клітинок іншого робочого аркушу
    mySheet.[Аркуш2!A1] = 10
    Debug.Print "B1+B2 = "; mySheet.Evaluate("Аркуш2!A1")
    'Метод Evalute для формули (вбудованої функції)
    Debug.Print "B1+B2 = "; mySheet.Evaluate("B1+B2")
    Debug.Print "Cos(0) = "; mySheet.[COS(B1)]
    Debug.Print "Аркуш1!B2 + Аркуш2!A1="; _
                mySheet.[Аркуш1!B2 + Аркуш2!A1]

    'Метод Evalute для клітинки
    mySheet.Evaluate("B1").Font.Bold = True
    mySheet.[B5].Value = "Привіт"

    'Метод  Evalute для діапазону
    mySheet.Evaluate("A1:A2").Font.Italic = True
    mySheet.[A1:A2].Font.Italic = True

    'Метод Evaluate для змінної - змінна повинна бути рядком!
    PiVal = Str(WorksheetFunction.Pi() / 4)
    Debug.Print "Pi/4 = "; mySheet.Evaluate(PiVal)

    'Метод Evaluate для поіменованого діапазону
    mySheet.Names.Add "Аркуш1!cellName", mySheet.Cells(2, 3)
    mySheet.Evaluate("Аркуш1!cellName") = 333
    Debug.Print mySheet.[Аркуш1!cellName]
```

```
    'Метод Evalute для масивів (у тому числі багатовимірних)
    Set mySheet = myWbk.Worksheets("Аркуш1")
    Set myRange = mySheet.Range("A1:D1")
    myRange.Value = mySheet.Evaluate("{1, 2, 3, 4}")
    Set myRange = mySheet.Range("A2:D4")
    'Використовуємо неявний виклик методу Evaluate
    myRange.Value = [{1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12}]

    myWbk.Close SaveChanges:=False
End Sub
```

Відміна від використання цього методу у разі застосування об'єкта `Application` зводиться тільки до зазначення об'єкта робочого аркушу, з якого викликається метод. Також підтримується і неявний виклик метода за допомогою квадратних дужок.

- `ChartObjects(Index)` – містить посилання на додану до поточного робочого аркушу діаграму (об'єкт `ChartObject`) або колекцію таких об'єктів, якщо додано більше однієї діаграми. Наприклад, для робочого аркушу, показаного на Рисунок 91, наступний код:

```
Dim mySheet As Worksheet

Set mySheet = ActiveWorkbook.Worksheets("Test")
Debug.Print mySheet.ChartObjects(1).Chart.ChartTitle.Text
```

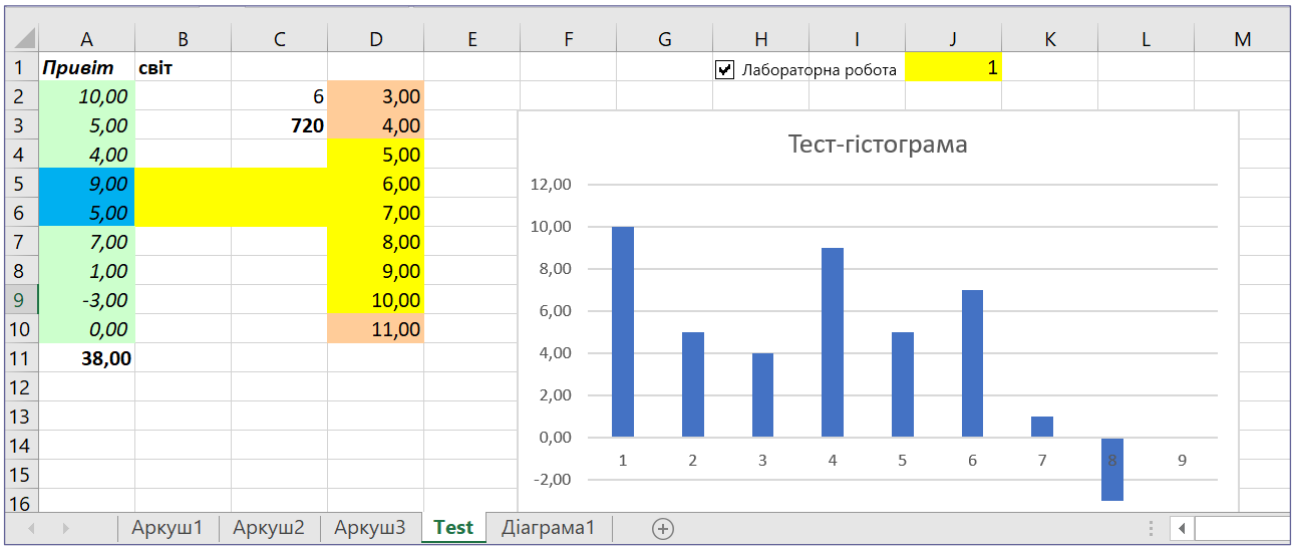


Рисунок 91 – Приклад робочого аркушу з даними, які обробляються методом ChartObjects(Index)

виведе у вікно *Immediate* назву доданої до аркушу діаграми – Тест-гістограма

- `Select(Replace)` – виділяє поточний робочий аркуш, параметр `Replace` дозволяє регулювати виділення: при значенні `True` (за замовчуванням) буде виділений робочий аркуш, з об'єкта якого викликався метод, при значенні `False` – разом з активним аркушем буде виділений у переліку аркушів аркуш, з

якого викликався метод. Наприклад, якщо був активним робочий аркуш Аркуш1, то після виконання оператора:

```
Worksheets("Test").Select False
```

виділені аркуші будуть виглядати так, як показано на Рисунок 92.

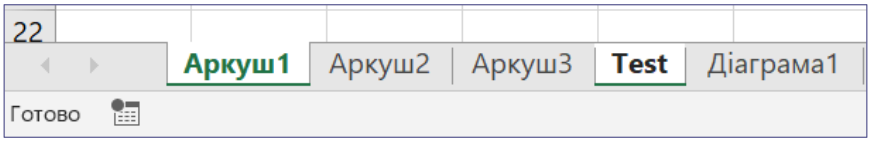


Рисунок 92 – Перелік аркушів після виконання методу Select False

- `Move(Before, After)` – переміщує поточний аркуш у інше місце у переліку аркушів робочої книги. Необов'язкові параметри `Before` та `After` містять посилання на аркуш перед або після якого буде розміщений поточний. Одночасно може зазначатися тільки один з параметрів, якщо вони не будуть зазначені, метод створює нову робочу книгу, до якої переміщується аркуш, з об'єкта якого був викликаний метод. Наприклад, код:

```
ActiveSheet.Move Before:=Worksheets("Аркуш1")
```

преміщує активний аркуш перед робочим аркушем Аркуш1;

- `Delete()` – видаляє поточний аркуш, перед видаленням виводиться діалогове вікно його підтвердження. У разі підтвердження видалення, метод повертає `True`, у разі скасування – `False`, наприклад, код:

```
If Worksheets("Аркуш2").Delete Then
        Debug.Print "Аркуш2 видалений"
    Else
        Debug.Print "Видалення скасоване"
    End If
```

виведе у вікно *Immediate* інформацію про видалення аркушу Аркуш2 або скасування видалення.

- `Copy(Before, After)` – копіює поточний аркуш у інше місце у переліку аркушів робочої книги. Необов'язкові параметри `Before` та `After` містять посилання на аркуш перед або після якого буде розміщена копія поточного аркушу. Одночасно може зазначатися тільки один з параметрів, якщо вони не будуть зазначені, метод створює нову робочу книгу, до якої копіюється аркуш, з об'єкта якого був викликаний метод. Наприклад, код:

```
Dim mySheet As Worksheet

Set mySheet = ActiveWorkbook.Worksheets("Test")
mySheet.Copy
With ActiveWorkbook
    .SaveAs filename:=ThisWorkbook.Path _
                & Application.PathSeparator & "New.xlsx", _
            FileFormat:=xlOpenXMLWorkbook
    .Close SaveChanges:=False
End With
```

виконує копіювання активного аркушу до нової робочої книги New.xlsx , яка зберігається у тому ж каталозі, у якому знаходиться файл з програмою, збереження виконується методом:

- `SaveAs()` – зберігає поточний аркуш у іншому файлі, має багато параметрів, серед яких один обов'язковий: `FileName` – повний шлях до файлу робочої книги, у якій буде збережений поточний аркуш. Серед необов'язкових параметрів часто вживається `FileFormat` – формат файлу робочої книги, у якій буде збережений поточний аркуш, він приймає елементи енумератора `XlFileFormat` (Таблиця 12).

- `Paste(Destination, Link)` – виконує додання вмісту Буферу обміну до поточного аркушу, може бути використаний один з необов'язкових параметрів: `Destination` – містить посилання на діапазон (об'єкт `Range`), до якого буде виконано додання, `Link` – при значенні `True` дозволяє встановити зв'язок з джерелом даних, що додаються (за замовчуванням значення – `False`). У разі не використання параметрів додання вмісту Буферу обміну виконується до поточного виділення (необхідно, щоб воно було виконане перед використанням методу). Наступний код виконує копіювання діапазону за допомогою Буферу обміну з одного робочого аркушу до іншого методами `Copy()` та `Paste()`:

```
Dim mySheet As Worksheet

Set mySheet = ActiveWorkbook.Worksheets("Test")
mySheet.Range("A2:A10").Copy
ActiveSheet.Paste _
            Destination:=Worksheets("Аркуш1").Range("D1:D9")
```

- `ExportAsFixedFormat()` – дозволяє зберегти поточний аркуш у файлі формату *Portable Data Format* (*PDF*) або *XML Paper Specification* (*XPS*), зауважимо, що цей метод є доповненням до методу метод `SaveAs()` (див. вище), який не дозволяє зберігання у таких форматах. Метод має серед інших один обов'язковий параметр – `Type`, який може приймати як аргументи елементи енумератора `XlFixedFormatType`: `xlTypePDF=0` – для збереження у PDF-файлі та `xlTypeXPS=1` – для збереження у XPS-файлі. Серед необов'язкових параметрів, які використовуються найчастіше можна назвати: `FileName` – повне ім'я файлу, у якому буде збережений аркуш (у разі невизначення цього параметру, він зберігається у файлі з ім'ям робочої книги аркушу у каталозі за замовчуванням, зазвичай, Документи у профілі користувача). Наступний код демонструє збереження поточного аркушу у файлі формату PDF з попереднім встановленням орієнтації сторінки у альбомну, вигляд результату наведений на Рисунок 93:

```
Dim mySheet As Worksheet

Set mySheet = ActiveWorkbook.Worksheets("Test")
mySheet.PageSetup.Orientation = xlLandscape
mySheet.ExportAsFixedFormat Type:=xlTypePDF, _
```

```
        filename:=ThisWorkbook.Path & Application.PathSeparator _
                & "New.pdf"
```

- `ShowAllData()` – дозволяє показати всі приховані за допомогою команд контекстного меню рядки та стовпці і відфільтровані Автофільтром дані на поточному робочому аркуші.

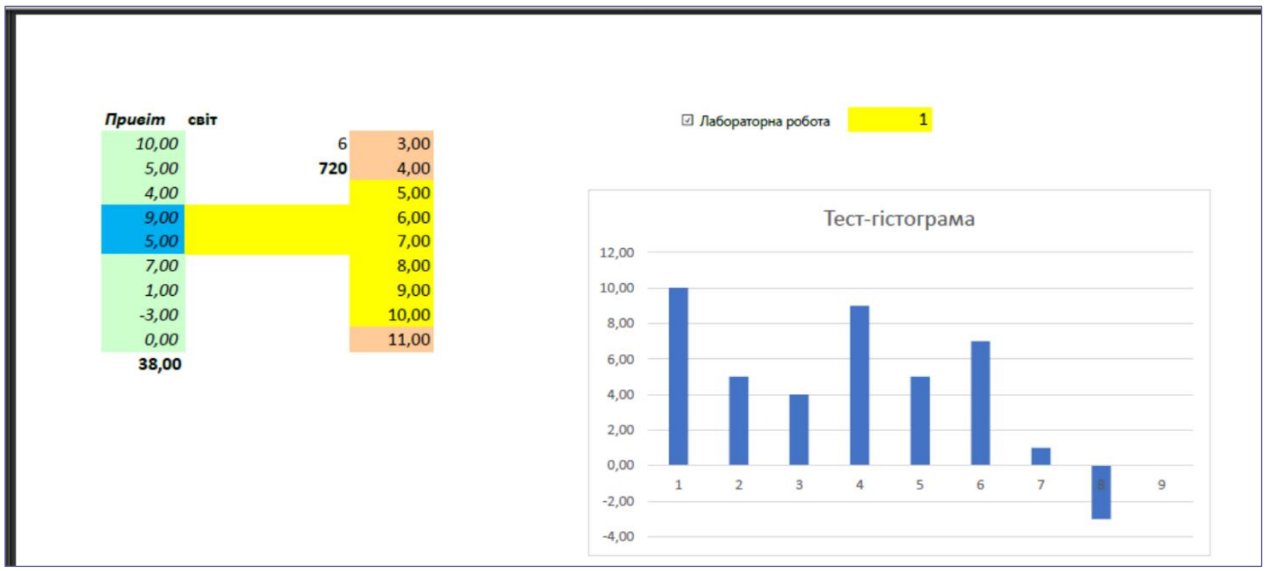


Рисунок 93 – Робочий аркуш, екпортований до файлу формату PDF методом `ExportAsFixedFormat()`

Клас `Worksheet` має більш, аніж півтора десятка *подій, з яких найважливішими* можна назвати:

- `Activate` – виникає перед активацією визначеного аркуша. Так, у разі програмування певних дій при активації робочого аркушу Аркуш1 робочої книги з програмою, необхідно відкрити у вікні проєктів Редактора Visual елемент Аркуш1 (Аркуш1) проєкту цієї робочої книги, обрати у вікні редактора коду у списку зверху зліва елемент Worksheet, а у списку справа зверху – подію Activate, після чого написати код обробника події (Рисунок 94).

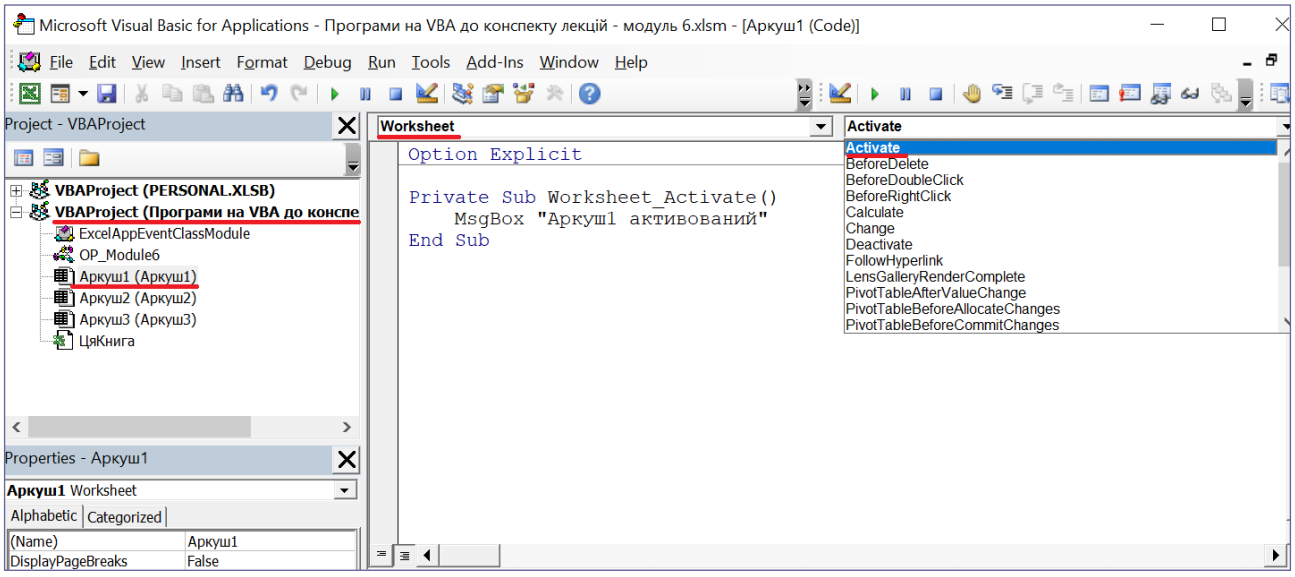


Рисунок 94 – Організація обробки події активації робочого аркушу

- Deactivate – виникає перед деактивацією визначеного аркуша (активацією іншого аркуша в переліку аркушів робочої книги);

- Calculate – виникає після завершення розрахунків на визначеному аркуші, цю подію використовують при організації ініціалізації "ручного" перерахування значень, після якого, наприклад, може бути виконана автоматична підгонка ширини стовпців, так, щоб були видимі отримані значення:

```
Private Sub Worksheet_Calculate()
    Columns("A:E").AutoFit
End Sub
```

- Change(Target) – виникає при зміні вмісту будь-якої клітинки робочого аркуша, для якого викликається подія. Параметр Target (об'єкт класу Range) дозволяє контролювати зміни у необхідному діапазоні, наприклад, обробник події:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    If Target.Address = "$A$1" Then
        MsgBox "Значення клітинки A1 змінено"
    End If
End Sub
```

перевіряє, чи відбулася зміна вмісту у клітинці з адресою A1 робочого аркушу, для якого виникає подія (у модулі якого програмується її обробник – Рисунок 94).

6.5 Робота з діапазонами (клас Range)

На відміну від Microsoft Word, об'єктна модель якого для таблиць передбачає класи Table, Row, Column, Cell та колекцію Tables (Рисунок 62), об'єктна модель Microsoft Excel і рядки, і стовпці, і клітинки, і діапазони клітинок визначає через єдиний об'єкт – об'єкт класу Range (Рисунок 76) [5].

Розглянемо основні способи отримання об'єкта Range. По-перше, можна використати властивість Range, яку мають об'єкти класів Application, Worksheet, наприклад, код:

```
Dim mySheet As Worksheet
Dim myRange As Range

Set mySheet = ActiveWorkbook.Worksheets("Аркуш1")
Set myRange = mySheet.Range("A1")
```

визначає у якості діапазону (об'єкта класу Range) клітинку за адресою A1 робочого аркушу Аркуш1, код:

```
Dim myRange As Range

ActiveWorkbook.Worksheets("Аркуш1").Activate
Set myRange = Application.Range("A1")
```

виконує те ж саме (властивість `Range` об'єкта `Application` застосовується для активного робочого аркуша).

Отримати об'єкт `Range` для діапазону клітинок можна аналогічним чином, наприклад:

```
Dim myRange As Range

Set myRange = Worksheets("Аркуш1").Range("A1:A3")
```

Нагадаємо, що посилання `ActiveWorkbook` можна не зазначати, колекція `Worksheets` буде автоматично використана саме для активної робочої книги.

Необхідно зазначити, що властивість `Range` має і сам клас `Range` – це дозволяє створювати піддіапазони з діапазонів клітинок, але при цьому адресація клітинок піддіапазону буде відрізнятися від стандартної (зазначеної літерою стовпця та номером рядка), наприклад, код:

```
Dim myRange As Range, subRange As Range

Set myRange = Worksheets("Аркуш1").Range("B2")
Set subRange = myRange.Range("C2")
subRange.Value = 5
```

вставляє значення `5` не до клітинки з адресою C2, а до клітинки з адресою D3, оскільки створений піддіапазон являє собою новий віртуальний робочий аркуш з початком його адресації (тобто клітинки з віртуальною адресою піддіапазону A1) у клітинці з фактичною адресою B2 (Рисунок 95).

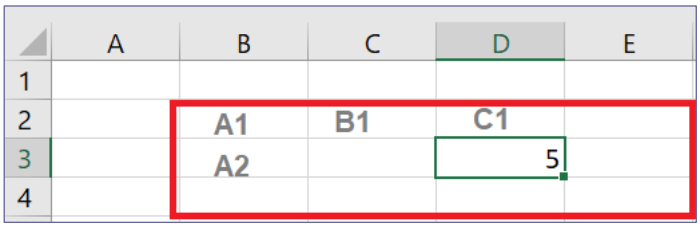


Рисунок 95 – Створення піддіапазону з діапазону клітинок

По-друге, для отримання об'єкта `Range` можна використати властивість `Cells`, яку мають об'єкти класів `Application` та `Worksheet`. Наприклад, код:

```
Dim mySheet As Worksheet
Dim myRange As Range

Set mySheet = ActiveWorkbook.Worksheets("Аркуш1")
Set myRange = mySheet.Cells(1, 4)
```

визначає у якості діапазону (об'єкта класу `Range`) клітинку за адресою D1 робочого аркушу Аркуш1, а отримання діапазону виконується з застосуванням обох властивостей `Range` і `Cells`:

```
Dim myRange As Range

Set myRange = Worksheets("Аркуш1").Range(Cells(1, 3), Cells(2, 4))
```

Наведений код створює діапазон клітинок з адресою C1:D2.

Клас `Range` також має властивість `Cells`, що є альтернативним способом створення піддіапазонів, при цьому також створюється віртуальний робочий аркуш і фактична адреса клітинок буде відрізнятися від стандартної (зазначеної літерою стовпця та номером рядка). Наведемо код, який відповідає вище наведеному коду, результат виконання якого представлений на Рисунок 95.

```
Dim myRange As Range, subRange As Range

Set myRange = Worksheets("Аркуш1").Cells(2, 2)
Set subRange = myRange.Cells(2, 3)
subRange.Value = 5
```

Ще одним способом отримання об'єкту `Range` є властивості та методи самого класу `Range` та інших класів, що повертають цей об'єкт для інших об'єктів Microsoft Excel API, наприклад, властивості `Rows` і `Columns` класів `Application` та `Worksheet Range`, або методи `Evaluate`, `Intersect`, `Union` класу `Application` та метод `Evaluate` класу `Worksheet`.

Розглянемо *найбільш вживані властивості класу `Range`*:

- `Value` – представляє вміст клітинок діапазону. У разі, якщо діапазон містить тільки одну клітинку завдання її значення через цю властивість передбачувано та використано у прикладах коду, наведеного вище. Але коли діапазон містить більше однієї клітинки цю властивість можна використати для завдання їх значень двома способами: якщо задати властивості діапазону `Value` одне значення, воно буде задане усім клітинкам діапазону, якщо ж присвоїти цій властивості посилання на двовимірний масив, клітинкам будуть задані значення відповідних елементів масиву (Рисунок 96):

```
'Заповнюємо діапазон клітинок однаковим значенням
Set myRange = Worksheets("Аркуш1").Range("A5:B6")
myRange.Value = 5

'Заповнюємо клітинки діапазону даними двовимірного масиву
Set myRange = Worksheets("Аркуш1").Range("E1:H3")
'Використовуємо неявний виклик методу Evaluate
myRange.Value = [{1, 2, 3, 4; 6, 6, 7, 8; 9, 10, 11, 12}]
```

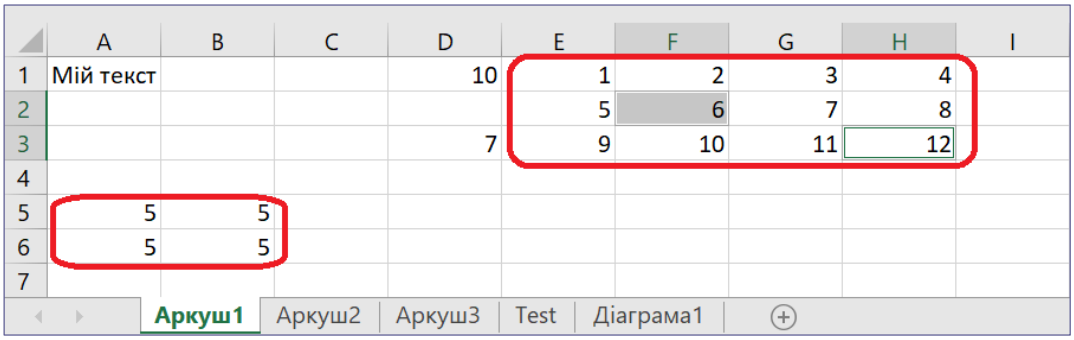


Рисунок 96 – Заповнення діапазонів даними

Зауважимо, що встановлення значення діапазону може виконуватися без явного зазначення властивості `Value`, тому оператори:

```
myRange.Value = 5
myRange = 5
```

фактично однакові;

- `Text` – повертає вміст поточного діапазону, який містить тільки одну клітинку, як текст разом з ознаками форматування. У разі, якщо діапазон містить більше однієї клітинки і вони мають однаковий міст та однакове форматування, повертає цей вміст. Якщо клітинок більше однієї і вони мають різний вміст/форматування, повертається `Null`.

- `Count` – повертає кількість клітинок діапазону;

- `Address` – містить адресу поточного об'єкту `Range`, може приймати необов'язкові параметри, серед яких виділимо: `ReferenceStyle` – визначає стиль адресу, приймає елементи енумератора `XlReferenceStyle`: `xlA1=1` – стиль адресації за замовчуванням (літера стовпця-номер рядка, наприклад, A1) або `xlR1C1=-4150` – стиль адресації R-номер рядка-C-номер стовпця, наприклад R1C1, параметр `External` при значенні `True` повертає адресу діапазону. Наприклад, код:

```
Dim myRange As Range

Set myRange = Worksheets("Аркуш1").Range("B2")
Debug.Print "Адреса діапазону клітинок:"; _
      myRange.Address(ReferenceStyle:=xlR1C1, External:=True)
```

виводить до вікна *Immediate*

```
Адреса діапазону клітинок:'[Робоча книга-зразок.xlsx]Аркуш1'!R2C2
```

- `Item(RowIndex, ColumnIndex)` – дозволяє отримати як об'єкт `Range` клітинку поточного об'єкту `Range`. У разі зазначення тільки одного параметру `RowIndex` (він обов'язковий), повертається об'єкт `Range` з однієї клітинки, обраної за її послідовним номером (індексом у діапазоні). У разі зазначення обох параметрів, повертається об'єкт `Range` з однієї клітинки, обраної послідовним номером рядка в межах діапазону (`RowIndex`) та клітинки у цьому рядку (`ColumnIndex`). Наприклад, для діапазону E1:H3, представленого на Рисунок 96, код

```
Debug.Print myRange.Item(6).Address
Debug.Print myRange.Item(3, 4).Address
```

виводить до вікна *Immediate*:

```
$F$2
$H$3
```

Зауважимо, що отримання клітинки властивістю `Item(RowIndex, ColumnIndex)` може виконуватись без явного зазначення цієї властивості, тому оператори

```
Debug.Print myRange.Item(3, 4).Address
```

та

```
Debug.Print myRange(3, 4).Address
```

фактично однакові;

- `End(Direction)` – повертає як об'єкт `Range` граничну клітинку діапазону, який містить поточний діапазон (найчастіше об'єкт `Range` клітинки). Параметр `Direction` може приймати елементи енумератора `XlDirection`, які визначають напрям, де шукається границя: `xlDown=-4121` – нижня границя, `xlToLeft=-4159` – ліва границя, `xlToRight=-4161` – права границя, `xlUp=-4162` – верхня границя. Фактично ця властивість емулює клавіатурні комбінації *Ctrl + відповідна стрілка*. Наступний код виводить адреси граничних клітинок діапазону E1:H3, представленого на Рисунок 96, для клітинки G2:

```
Set cellRange = myRange.Item(2, 3)
Debug.Print "Для клітинки діапазону з адресою ";
cellRange.Address
Debug.Print "Крайня ліва клітинка діапазону:";
cellRange.End(xlToLeft).Address
Debug.Print "Крайня права клітинка діапазону:";
cellRange.End(xlToRight).Address
Debug.Print "Крайня верхня клітинка діапазону:";
cellRange.End(xlUp).Address
Debug.Print "Крайня нижня клітинка діапазону:";
cellRange.End(xlDown).Address
```

Цей код виводить:

```
Для клітинки діапазону з адресою $G$2
Крайня ліва клітинка діапазону:$E$2
Крайня права клітинка діапазону:$H$2
Крайня верхня клітинка діапазону:$G$1
Крайня нижня клітинка діапазону:$G$3
```

- `CurrentRegion` – повертає об'єкт `Range`, який представляє діапазон, оточений порожніми рядками та стовпцями. Наприклад, щоб отримати адресу непорожньої області навколо клітинки `cellRange` з попереднього прикладу, можна скористатися кодом:

```
Debug.Print cellRange.CurrentRegion.Address
```

цей код виведе адресу діапазону, більшого, аніж початковий діапазон, оскільки є непусті клітинки у стовпці D:

```
$D$1:$H$3
```

- `Next` – повертає як об'єкт `Range` наступну у рядку клітинку для поточної клітинки (або першої клітинки діапазону). Наступна клітинка може не належати вихідному діапазону. У разі використання цієї властивості для клітини останнього стовпця (XFD для Microsoft Office 2019) виникне помилка;

- `Previous` – повертає як об'єкт `Range` попередню у рядку клітинку для поточної клітинки (або першої клітинки діапазону). Попередня клітинка може не належати вихідному діапазону. У разі використання цієї властивості для клітини першого стовпця (A) виникне помилка;

- `Offset(RowOffset, ColumnOffset)` – повертає як об'єкт `Range` новий об'єкт `Range` зі зміщенням від початкового, вказаним параметрами `RowOffset` – зміщення по рядкам (позитивні числа – до низу, негативні – до гори) та `ColumnOffset` – зміщення по стовпцям (позитивні числа – вправо, негативні – вліво). Наступний код демонструє використання методів `Next`, `Previous` та `Offset` для діапазону `myRange` з адресою E1:H3, представленого на Рисунок 96:

```
Debug.Print myRange.Next.Address     '$F$1
Debug.Print myRange.Item(1, 4).Next.Address     '$I$1
' Debug.Print Range("$XFD$1").Next.Address       'Помилка

Debug.Print myRange.Previous.Address     '$D$1
Debug.Print myRange.Item(1, 4).Previous.Address     '$G$1
' Debug.Print Range("$A$1").Previous.Address       'Помилка

'Адреса клітинки на 2 рядки вище та 3 стовпця лівіше
'останньої клітинки діапазону
Debug.Print myRange.Item(3, 4).Offset(-2, -3).Address    '$E$1
```

Результати роботи цього коду представлені у коментарях.

- `Areas` – повертає посилання на колекцію несуміжних піддіапазонів об'єкту `Range`, якщо у нього такі є (існує можливість виділення їх за допомогою миші з натиснутою клавішею *Ctrl*), у разі, якщо виділений діапазон суміжних клітинок, колекція містить поточний об'єкт `Range`. Наприклад, при виділенні трьох піддіапзонів, як показано на Рисунок 97, код:

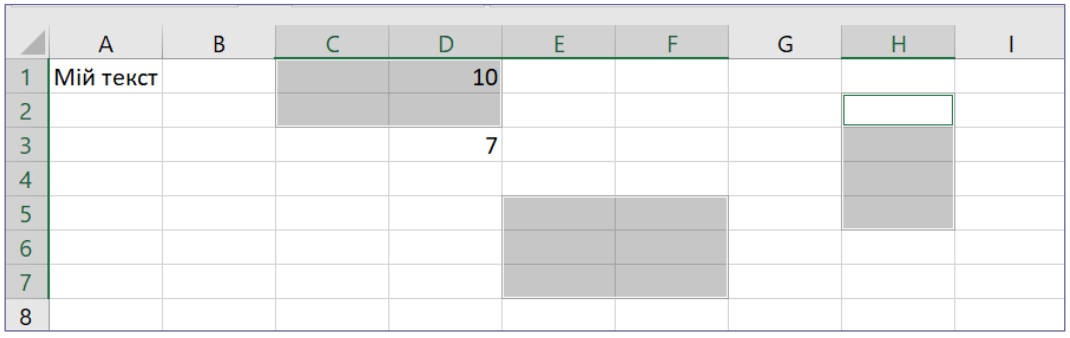


Рисунок 97 – Виділення діапазону, що складається з трьох несуміжних піддіапазонів

```
Dim myRange As Range

If Selection.Areas.Count > 1 Then
        For Each myRng In Selection.Areas
            Debug.Print myRng.Address
        Next
    End If
```

виведе до вікна *Immediate* адреси піддіапазонів:

```
$C$1:$D$2
$E$5:$F$7
$H$2:$H$5
```

які у подальшій програмі можуть оброблятися як окремі діапазони.

- `Range(Cell1, Cell2)` – повертає піддіапазон (об'єкт `Range`) поточного діапазону, властивість була розглянута в описі створення об'єктів `Range` (див. вище);

- `Cells` – повертає піддіапазон (об'єкт `Range`) поточного діапазону, властивість була розглянута в описі створення об'єктів `Range` (див. вище);

- `Rows` – містить посилання на колекцію об'єктів `Range`, що являють собою рядки поточного діапазону, властивість `Item(Index)` цієї колекції дозволяє отримати посилання на необхідний рядок діапазону за значенням його індексу;

- `Columns` – містить посилання на колекцію об'єктів `Range`, що являють собою стовпці поточного діапазону, властивість `Item(Index)` цієї колекції дозволяє отримати посилання на необхідний стовпець діапазону за значенням його індексу;

- `Row` – повертає абсолютний номер рядка у діапазоні;

- `Column` – повертає абсолютний номер стовпця у діапазоні;

Наступний код демонструє використання властивостей `Rows`, `Columns`, `Row` та `Column`: для діапазону `myRange` з адресою E1:H3, представленого на Рисунок 96:

```
Debug.Print myRange.Rows.Item(2).Address
Debug.Print myRange.Columns.Item(2).Address
For Each tmpRange In myRange.Rows
    Debug.Print tmpRange.Row;
Next
Debug.Print
For Each tmpRange In myRange.Columns
    Debug.Print tmpRange.Column;
Next
Debug.Print
```

Наведений код виведе до вікна *Immediate*:

```
$E$2:$H$2
$F$1:$F$3
 1 2 3
 5 6 7 8
```

- `Resize(RowSize, ColumnSize)` – змінює поточний діапазон, повертає об'єкт `Range`, що відповідає зміненому діапазону. Необов'язкові параметри `RowSize` та `ColumnSize` вказують кількість рядків та стовпців у оновленому діапазоні. Наприклад, збільшення діапазону `myRange` на один рядок вниз та один стовпець вправо може виглядати так:

```
myRange.Resize(myRange.Rows.Count + 1, _
               myRange.Columns.Count + 1).Select
```

Розглянемо властивості, що пов'язані з розміщенням у клітинках діапазону формул та функцій:

- Formula – дозволяє вводити/зчитувати формулу, записану з використанням стилю адресації xlA1 як об'єкт типу Variant. Записується формула до діапазону як рядок, починаючи з символу "дорівнює", наприклад:

```
Dim myRange As Range

Set myRange = Worksheets("Аркуш1").Range("A7")
myRange.Formula = "=$E$1+$F$1"
```

Зчитується з діапазону формула також як рядок, наприклад, код:

```
Debug.Print "Формула у діапазоні "; myRange.Address; _
            myRange.Formula
```

виводить до вікна *Immediate*:

```
Формула у діапазоні $A$7=$E$1+$F$1
```

Зазвичай формула вводиться у діапазон, що містить одну клітинку, але якщо це не так, формула буде скопійовану у всі клітинки діапазону. Якщо формула містить функцію з параметрами, які передаються посиланнями, то значення параметрів повинно конкатенуватись з ім'ям функції у вигляді рядка, аналогічно використанню метода Evaluate(Name) класів Application та Worksheet (див. вище):

```
Dim myRange As Range, subRange As Range

Set myRange = Worksheets("Аркуш1").Range("B5")
myRange.Value = 5
Set subRange = Worksheets("Аркуш1").Range("B6")
subRange.Formula = "=FACT(" & myRange.Value & ")"
Debug.Print "Факторіал від"; myRange.Value; "="; _
            subRange.Value
```

Результатом виконання наведеного коду буде:

```
Факторіал від 5 = 120
```

Звернемо увагу на відміну використання властивості Formula класу Range та метода Evaluate(Name) класів Application та Worksheet (див. вище). Властивість Formula класу Range додає до діапазону саме формулу, а Microsoft Excel виконує її обрахунок і виводить результуюче значення, у той час як метод Evaluate(Name) передає Microsoft Excel формулу як текстовий параметр, який оцінюється застосунком (тобто обчислюється) і у клітинку записується не формула, а результат.

- FormulaR1C1 – працює аналогічно властивості Formula, але використовує стиль адресації xlR1C1, наприклад:

```
Worksheets("Аркуш1").Range("B8").FormulaR1C1 = "=R1C5+R1C6"
```

Зауважимо, що до діапазону буде вставлена у тому стилі адресації, який налаштований командою *Файл–Параметри–Формули–Стиль посилань R1C1* або значенням властивості ReferenceStyle класу Application (див. вище).

- `HasFormula` – перевіряє поточний діапазон на наявність формул, повертає `True`, у разі знаходження формул у кожній клітинці діапазону, повертає `False`, у разі відсутності формул у кожній клітинці діапазону, інакше – повертає `null`.

- `Dependents` – дозволяє отримати як об'єкт `Range` клітинки, які залежать від клітинок поточного діапазону, з якого викликається ця властивість. Наприклад, якщо поточний діапазон зазначає клітинку, у якій зберігається аргумент, що використовується у формулі в іншій клітинці, то властивість `Dependents` повертає посилання на клітинку з формулою:

```
Dim myRange As Range

'Клітинка з аргументом
Set myRange = Worksheets("Аркуш1").Range("E1")
'Клітинки з формулами
Debug.Print myRange.Dependents.Address
```

Цей код для робочого аркушу, представленого на Рисунок 98, зазначає адреси клітинок, у яких знаходяться формули, що залежать від вмісту діапазону E1:

```
$B$8,$A$7:$A$10
```

- `Precedents` – дозволяє отримати як об'єкт `Range` клітинки, які впливають на клітинки поточного діапазону, з якого викликається ця властивість. Наприклад, якщо поточний діапазон зазначає клітинку з формулою, то властивість `Precedents` повертає посилання на клітинки, де містяться її аргументи:

```
Dim myRange As Range

'Клітинка з формулою
Set myRange = Worksheets("Аркуш1").Range("B8")
'Клітинки з аршументами
Debug.Print myRange.Precedents.Address
```

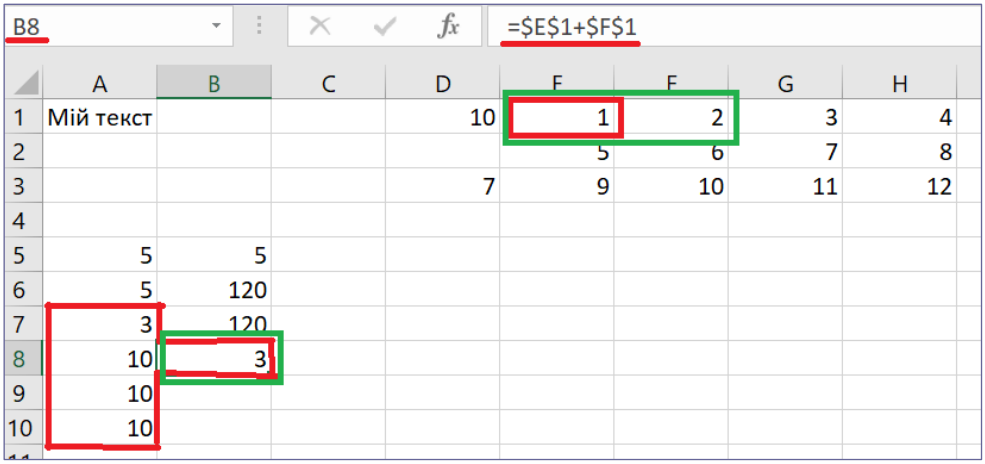


Рисунок 98 – Знаходження залежних від вмісту діапазону та впливових на вміст діапазону клітинок

Цей код для робочого аркушу, представленого на Рисунок 98, зазначає адреси клітинок, у яких знаходяться аргументи, що впливають на вміст діапазону B8 з формулою:

```
$E$1:$F$1
```

Розглянемо властивості, що виконують форматування діапазонів:

- `NumberFormat` – надає можливість виконати форматування даних у клітинках діапазону, приймає як своє значення рядок з символів підстановки, що визначають числовий формат або слово `"General"` для формату за замовчуванням [33]. Зауважимо, що секція *Число* панелі інструментів команди *Основне* дозволяє відкрити вікно *Формат клітинок*, у якому на вкладинці *Число* при обранні елемента *(усі формати)* у списку *Числові формати* можна сконструювати необхідний формат, орієнтуючись на наявні зразки (Рисунок 99).

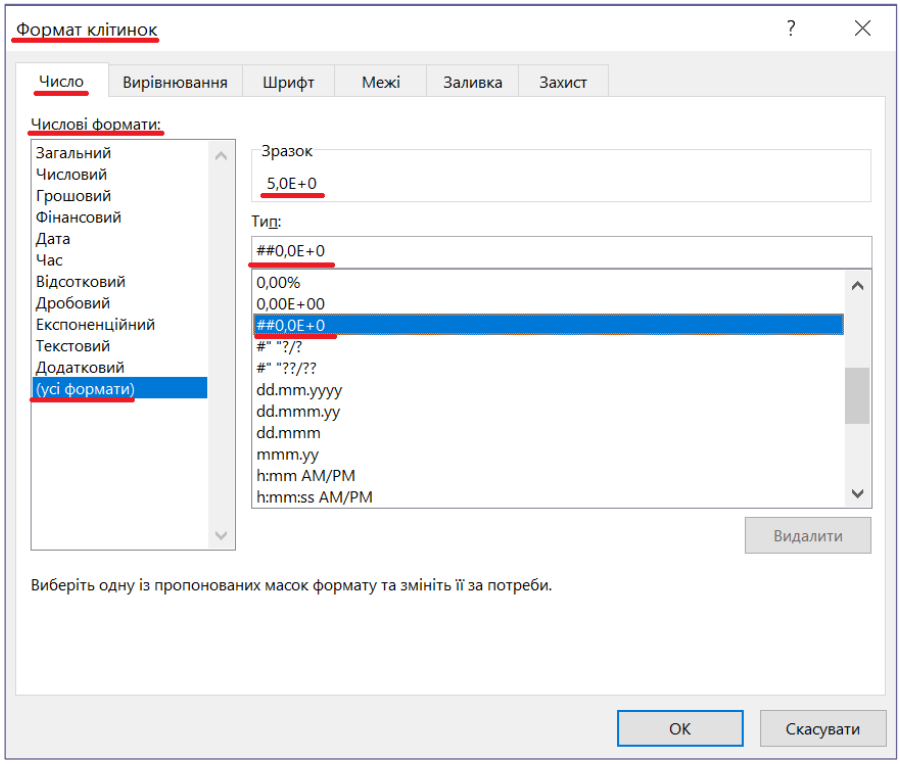


Рисунок 99 – Конструювання користувацького числового формату

Наприклад, код:

```
Debug.Print myRange.NumberFormat
myRange.NumberFormat = "###.00 грн."
```

спочатку виводить назву формату за замовчуванням: `"General"`, а потім у клітинках, що складають діапазон `myRange` числа форматуються як суми грошей з необов'язковими максимум трьома розрядами у частині гривень та обов'язковими двома – у частині копійок.

- `Font` – містить посилання на об'єкт класу `Font`, за допомогою якого встановлювати властивості шрифту, наприклад, гарнітуру, розмір, написання та колір (див. вище).

- Characters(Start, Length) – містить посилання на колекцію з символами текстового вмісту діапазону, необов'язкові параметри Start та Length дозволяють зазначити початковий символ та довжину підрядка, який повертається цією властивістю. Ця властивість дозволяє змінювати формат частини тексту в клітинках діапазону. Наприклад, щоб замінити у введеному тексті у клітинці A1 колір першої літери, можна скористатися кодом:

```
Dim myRange As Range

Set myRange = Range("A1")
myRange.Value = "Мій текст"
myRange.Characters(1, 1).Font.Color = vbRed
```

- Interior – містить посилання на об’єкт класу Interior, що задає ознаки форматування поточного об'єкту Range, насамперед – колір фону. Колір фону задається як значення індексу одного з 56 кольорів у поточній палітрі кольорів ColorIndex (Рисунок 100) або як елементи енумератора XlColorIndex: xlColorIndexAutomatic = -4105 – при кольорі, що обирається автоматично, та xlColorIndexNone = -4142) – при відсутності кольору.

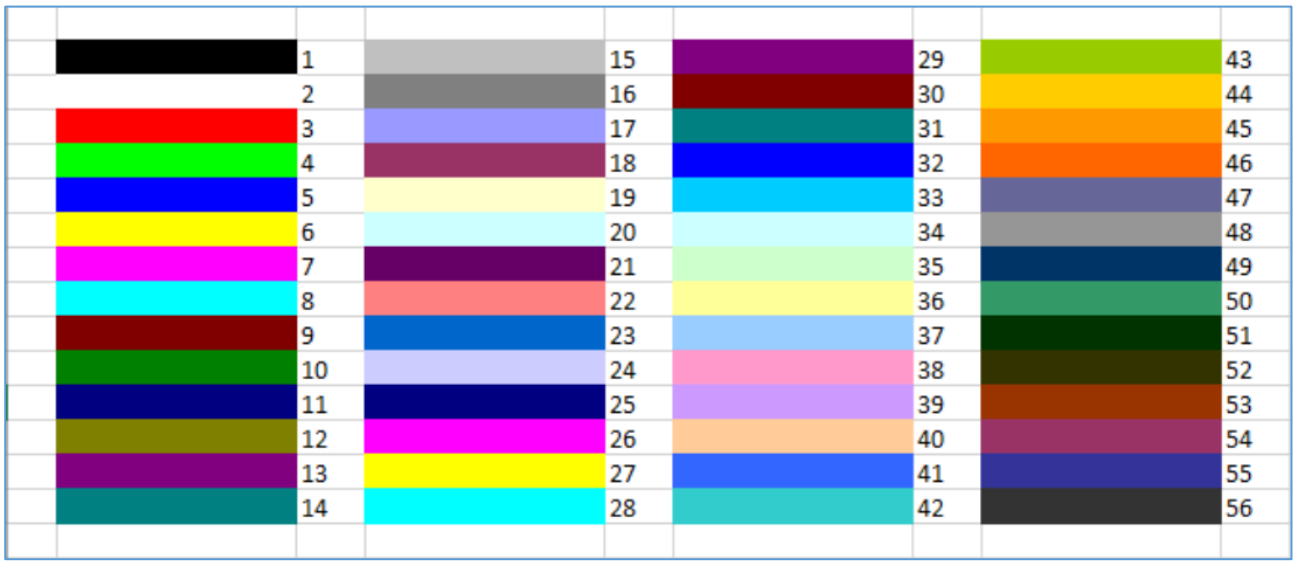


Рисунок 100 – Значення параметру ColorIndex властивості Interior

- Borders – містить посилання на колекцію Borders, за допомогою якої можна налаштовувати усі межі для клітинок діапазону, які представляються об'єктами класу Border. Можна отримати доступ до кожної з меж, якщо колекції передати аргумент Index, значення якого є елементом енумератора XlBordersIndex: xlEdgeTop=8 – верхня межа діапазону, xlEdgeRight=10 – права межа діапазону, xlEdgeBottom=9 – нижня межа діапазону, xlEdgeLeft=7 – ліва межа діапазону, xlInsideHorizontal=12 – горізонтальні межі для усіх клітинок діапазону, окрім зовнішніх меж діапазону, xlInsideVertical=11 – вертикальні межі для усіх клітинок діапазону, окрім зовнішніх меж діапазону, xlDiagonalDown=5 – межі від лівого верхнього кута

до правого нижнього кута кожної клітинки діапазону, xlDiagonalUp=6 – межі від лівого нижнього кута до правого верхнього кута кожної клітинки діапазону. Властивість LineStyle класу Border визначає тип лінії межі, який задається елементом енумератора XlLineStyle, серед яких найчастіше використовуються: xlContinuous=1 – безперервна лінія, xlDouble=-4119 – подвійна безперервна лінія, xlDash=-4115 – переривчаста лінія з символів "тире", xlLineStyleNone – відсутність лінії або константи xlGray25, xlGray50, xlGray75, що встановлюють сірий колір різної інтенсивності, або константа xlAutomatic, що забезпечує автоматичне визначення стилю. межі Товщина лінії межі задається властивістю Weight класу Border, що приймає елемент енумератора XlBorderWeight: xlHairline=1 – найтонша лінія, xlThin=2 – тонка лінія, xlMedium=-4238 – лінія середньої товщини або xlThick - найтовща лінія. Зауважимо, що властивості LineStyle та Weight впливають одна на іншу при застосуванні їх до одного об'єкта і результати можуть бути непередбачувані [34].

Колір межі може задаватись за допомогою властивостей Color або ColorIndex класу Border. Перша з них приймає результат функції RGB (див. вище), а друга – значення індексу кольору (Рисунок 100).

Наведений нижче код демонструє використання властивостей Font, Interior, а також властивостей елементів колекції Borders для форматування діапазону:

```
Set myRange = Worksheets("Аркуш1").Range("E1:H3")
myRange.Interior.ColorIndex = xlColorIndexNone
With Worksheets("Аркуш1").Range("E1")
    .Font.Color = vbYellow
    .Interior.ColorIndex = 5
End With
With myRange.Borders
    .LineStyle = xlDouble
'    .Weight = xlMedium   'Змінює LineStyle на одинарну лінію
    .Color = RGB(0, 0, 255)
End With
With myRange.Borders(xlInsideHorizontal)
    .LineStyle = xlContinuous
    .Weight = xlThick
    .ColorIndex = 4
End With
With myRange.Borders(xlInsideVertical)
    .LineStyle = xlContinuous
    .Weight = xlThin
    .ColorIndex = 7
End With
```

Результат виконання коду наведений на Рисунок 101.

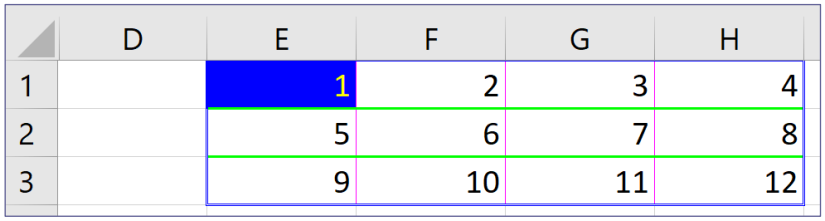


Рисунок 101 – Форматування вмісту та меж клітинок діапазону

- `HorizontalAlignment` – містить посилання на елемент енумератора `XlHAlign`, який визначає горизонтальне вирівнювання вмісту діапазону, найбільш вживані значення: `xlHAlignLeft=-4131` – вирівнювання за лівим краєм, `xlHAlignCenter=-4108` – вирівнювання по центру, `xlHAlignRight=-4152` – вирівнювання за лівим краєм, `xlHAlignJustify=-4130` – вирівнювання за шириною, `xlHAlignGeneral=1` – вирівнювання за значенням (типом даних: числа, дати, час – за правим краєм, рядки – за лівим краєм).

- `VerticalAlignment` – містить посилання на елемент енумератора `XlVAlign`, який визначає вертикальне вирівнювання вмісту діапазону, найбільш вживані значення: `xlVAlignTop=-4160` – за верхнім краєм, `xlVAlignCenter=-4108` – посередині, `xlVAlignBottom=-4107` – за нижнім краєм.

- `Orientation` – дозволяє відображати вміст клітинок діапазону під зазначеним як параметр кутом. Значення куту вказується у градусах від -90 до 90. Параметром також може бути елемент енумератора `XlOrientation`: Вказується кут нахилу в градусах. Наприклад, щоб розташувати текст по діагоналі, можна використовувати код:

```
oRange.Orientation = -45
```

- `WrapText` - надає можливість включити/відключити перенесення тексту на наступний рядок у клітинках діапазону.

- `RowHeight` – містить значення висоти першого рядка діапазону, вказане у пунктах (1 пункт = 1/72 дюйма);

- `ColumnWidth` – містить значення ширини кожного зі стовпців діапазону, одиницею ширини є ширина символу шрифту стилю *Звичайний* (за замовчуванням шрифту *Calibri*, 11 пунктів нормального написання), для автоматичної зміни ширини стовпців за їх максимальним вмістом використовують метод `Autofit()` для відповідної колекції `Columns` (нагадаємо, що вона, як і `Rows`, і `Cells` містить об'єкти класу `Range`);

- `ShrinkToFit` – дозволяє автоматично налаштувати розмір вмісту діапазону так, щоб він умістився у наявну ширину стовпця, наприклад:

```
Rows(1).ShrinkToFit = True
```

Зауважимо, що на практиці частіше виникає потреба вмістити дані до стовпця, не змінюючи розміру шрифту, таким чином метод `Autofit()` використовується частіше аніж властивість `ShrinkToFit`.

Наведемо приклад форматування даних, імпортованих з системи обліку навантаження. Імпортовані до Microsoft Excel дані мають такий вигляд (Рисунок 102). Бачимо, що не усі дані вмішуються у стовпці B та C, не вміщуються назви заголовків у другому рядку, ширина стовпців для більшості фактичних даних завелика.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1				Контингент				Передбачено робочим навчальним планом						
2	№ п/п	Назва нав	Факульте	К-ть студе	Кількість	Кількість	Форма на	Курс	Семестр	Лекції	Лаборато	Практичн	Заліки	Екзамени
3	5	Корпорат	ФЕЕІТ 121	34	1	1	ден	1	1	26	28		1	
4	10	Кваліфіка	ФЕЕІТ 121	10	1		ден	2	1					
5	13	Кваліфіка	ФЕЕІТ 121	10	1		ден	2	1					
6	25	Сучасні си	ФЕЕІТ 121	10	1	1	ден	2	1	22	22		1	
7	38	Технології	ФЕЕІТ 121	13	1	1	ден	4	1	28	14		1	
8	40	Виробнич	ФЕЕІТ 121	16	1		ден	1	2					
9	41	Групова д	ФЕЕІТ 121	25	1		ден	1	2	16		16	1	
10	46	Офісне пр	ФЕЕІТ 121	25	1	1	ден	1	2	16	32		1	

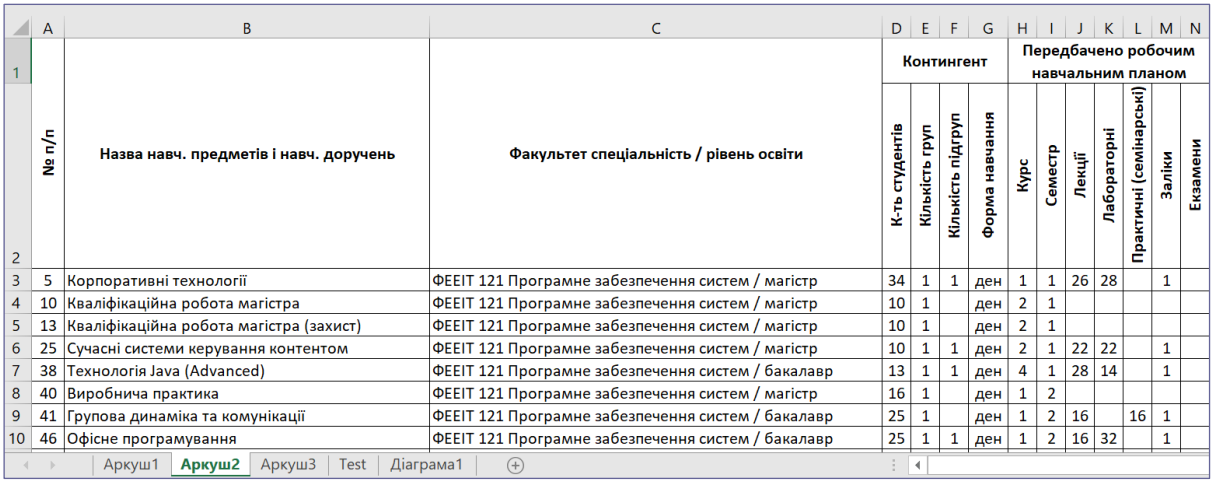
Аркуш1 | Аркуш2 | Аркуш3 | Test | Діаграма1

Рисунок 102 – Вихідний вигляд даних, імпортованих до Microsoft Excel

У разі, якщо такий імпорт з подальшим опрацюванням даних виконується регулярно, є сенс написати програму, що буде виконувати автоматичне форматування таблиці з даними (більшість використаних методів описані вище, метод `Merge()` дозволяє об'єднувати клітинки і буде детально описаний нижче):

```
Worksheets("Аркуш2").Activate
Range("A1:N2").HorizontalAlignment = xlHAlignCenter
Range("A1:N2").VerticalAlignment = xlVAlignCenter
Range("A2").Orientation = xlUpward
Range("D2:N2").Orientation = xlUpward
Range("H1:N1").WrapText = True
Range("A1:A2").Merge
Range("B1:B2").Merge
Range("C1:C2").Merge
Rows(1).RowHeight = Rows(1).RowHeight * 2
Range("A1").CurrentRegion.Columns.AutoFit
```

В результаті виконання програми таблиця з даними буде відформатована з оптимальним розміщенням даних (Рисунок 103).



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1				Контингент				Передбачено робочим навчальним планом						
2	№ п/п	Назва навч. предметів і навч. доручень	Факультет спеціальність / рівень освіти	К-ть студентів	Кількість груп	Кількість підгруп	Форма навчання	Курс	Семестр	Лекції	Лабораторні	Практичні (семінарські)	Заліки	Екзамени
3	5	Корпоративні технології	ФЕЕІТ 121 Програмне забезпечення систем / магістр	34	1	1	ден	1	1	26	28		1	
4	10	Кваліфікаційна робота магістра	ФЕЕІТ 121 Програмне забезпечення систем / магістр	10	1		ден	2	1					
5	13	Кваліфікаційна робота магістра (захист)	ФЕЕІТ 121 Програмне забезпечення систем / магістр	10	1		ден	2	1					
6	25	Сучасні системи керування контентом	ФЕЕІТ 121 Програмне забезпечення систем / магістр	10	1	1	ден	2	1	22	22		1	
7	38	Технологія Java (Advanced)	ФЕЕІТ 121 Програмне забезпечення систем / бакалавр	13	1	1	ден	4	1	28	14		1	
8	40	Виробнича практика	ФЕЕІТ 121 Програмне забезпечення систем / магістр	16	1		ден	1	2					
9	41	Групова динаміка та комунікації	ФЕЕІТ 121 Програмне забезпечення систем / бакалавр	25	1		ден	1	2	16		16	1	
10	46	Офісне програмування	ФЕЕІТ 121 Програмне забезпечення систем / бакалавр	25	1	1	ден	1	2	16	32		1	

Аркуш1 | Аркуш2 | Аркуш3 | Test | Діаграма1

Рисунок 103 – Вигляд даних, імпортованих до Microsoft Excel, після форматування

Розглянемо тепер *найважливіші методи об'єкта* `Range`:

- `Activate()` – активує поточний діапазон (робить активною першу клітинку діапазону), використовується при одночасній роботі з декількома діапазонами, наприклад:

```
Dim myRange As Range
Worksheets("Аркуш1").Activate
Set myRange = Range("A1:D10")
myRange.Interior.ColorIndex = 8
Range("E1").Activate
ActiveCell.Value = "Мій текст"
myRange.Activate
ActiveCell.Borders.Color = RGB(255, 0, 0)
Range("E1").Activate
```

Наведений код переключається між двома діапазонами: діапазоном `myRange` з адресою A1:D10 та анонімним діапазоном з адресою E1;

- `Select()` – надає можливість виділити зазначений діапазон (який може складатися з несуміжних областей), наприклад, код:

```
Set myRange = Range("A1:A10, C1:C10")
myRange.Select
myRange.Activate
Debug.Print ActiveCell.Address
```

створює діапазон, який складається з двох несуміжних областей та виділяє його, але при виклику методу `Activate()` для цього діапазону, активується тільки перша його клітинка (Рисунок 104), це також підтверджується виведенням до вікна *Immediate* її адреси: $A$1;

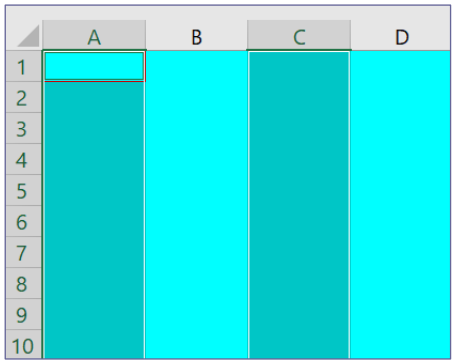


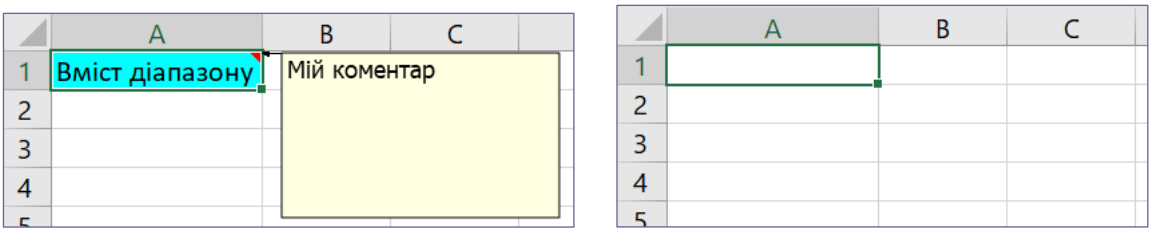
Рисунок 104 – Відмінність роботи методів Select та Activate

- `AddComment(Text)` – додає текстовий коментар до діапазону;
- `Clear()` – видаляє вміст, форматування, коментарі тощо діапазону;
- `ClearContents()` – видаляє тільки вміст діапазону;
- `ClearFormats()` – видаляє тільки форматування діапазону;
- `ClearComments()` – видаляє тільки коментарі діапазону;
- `Autofit()` – виконує автоматичну зміну ширини стовпців та висоти рядків поточного діапазону за їх максимальним вмістом, викликається з колекцій `Columns` або `Rows`, які зазначають стовпці або рядки діапазону, відповідно;

Наступний код демонструє використання наведених вище методів:

```
Dim myRange As Range
'Заповнення діапазону
Set myRange = Range("A1")
myRange.Value = "Вміст діапазону"
myRange.Columns.AutoFit
myRange.Interior.ColorIndex = 8
myRange.AddComment ("Мій коментар")
'Очищення діапазону
myRange.ClearComments
myRange.ClearFormats
myRange.ClearContents
```

На Рисунок 105 продемонстрований вигляд робочого аркушу а) після заповнення та б) після очищення діапазону.



а) *б)*

Рисунок 105 – Вигляд робочого аркушу а) після заповнення діапазону б) після очищення діапазону

- методи копіювання значення однієї клітинки поточного діапазону до усіх інших клітинок цього діапазону: `FillDown()` – копіювання найверхньої клітинки поточного діапазону до усіх інших клітинок цього діапазону, `FillLeft()` – копіювання найправішої клітинки поточного діапазону до усіх інших клітинок цього діапазону, `FillRight()` – копіювання найлівішої клітинки поточного діапазону до усіх інших клітинок цього діапазону, `FillUp()` – копіювання найнижньої клітинки поточного діапазону до усіх інших клітинок цього діапазону;

- `Autofill(Destination, Type)` – виконує автоматичне заповнення клітинок діапазону, зазначеного обов'язковим параметром `Destination` (він повинен містити поточний діапазон, з якого викликається метод) на основі значень клітинок поточного діапазону. Необов'язковий параметр `Type` приймає як значення елемент енумератора `XlAutoFillType`, який визначає тип заповнення цільового діапазону, найчастіше: `xlFillSeries=2` – заповнюється значеннями прогресії, `xlFillCopy=1` – значення поточного діапазону послідовно копіюються у вихідний, `xlFillDays=5` – заповнюється назвами послідовних днів тижня, `xlFillMonths=7` – заповнюється назвами послідовних місяців, `xlFillYears=8` – заповнюється значеннями послідовних років. Елемент `xlFillDefault=0` – значення за замовчуванням (коли аргумент `Type` не вказується) – у такому разі Microsoft Excel обирає тип заповнення на основі

типів даних та значень поточного діапазону. Наведемо приклад коду, який використовує методи `Fill` та `Autofill(Destination, Type)`:

```
Dim myRange As Range

Range("A1").Value = 1
Range("A1:B1").FillRight
Range("A2").Value = 3
Range("A2:B2").FillRight
Set myRange = Range("A1:A2")
myRange.AutoFill Range("A1:A10"), xlFillCopy
Set myRange = Range("B1:B2")
myRange.AutoFill Range("B1:B10"), xlFillSeries
Set myRange = Range("C1")
myRange.Cells(1, 1).Value = MonthName(DatePart("m", Date))
myRange.AutoFill Range("C1:C5"), xlFillMonths
```

та результат його роботи, представлений на Рисунок 106.

	A	B	C
1	1	1	червень
2	3	3	липень
3	1	5	серпень
4	3	7	вересень
5	1	9	жовтень
6	3	11	
7	1	13	
8	3	15	
9	1	17	
10	3	19	

Рисунок 106 – Вигляд робочого аркушу після автозаповнення діапазонів

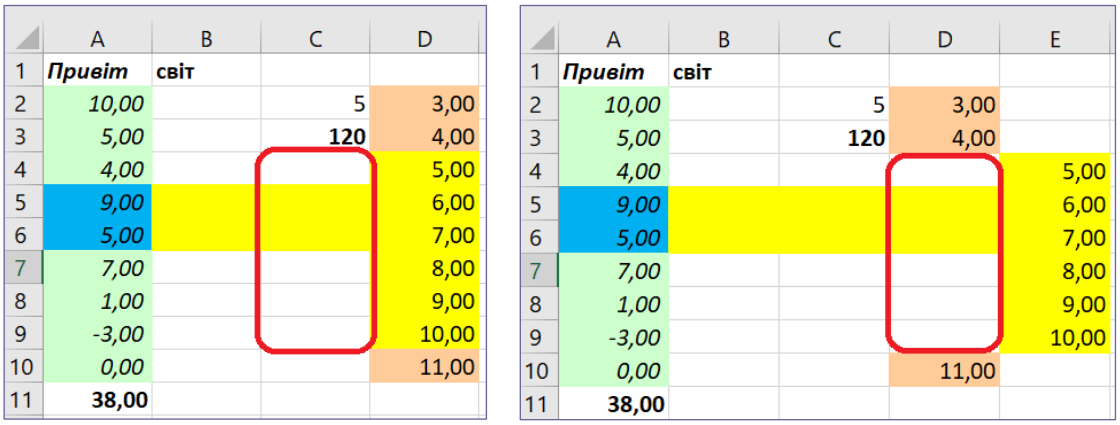
Зауважимо, що у разі невизначення параметру `Type` при виклику методів `Autofill(Destination, Type)` у наведеному прикладі, результат буде той самий.

- `Insert(Shift, CopyOrigin)` – виконує вставку діапазону до поточного діапазону, необов'язковий параметр `Shift` приймає значення елементів енумератора `XlInsertShiftDirection`, які вказують напрям зсуву клітинок, на місце яких виконується вставлення: `xlShiftDown=-4121` – вниз або `xlShiftToRight=-4161` – вправо. Необов'язковий параметр `CopyOrigin` вказує звідки копіювати формат для вставленого діапазону, приймає як значення елементи енумератора `XlInsertFormatOrigin`: `xlFormatFromLeftOrAbove =0` – копіює формат з клітинок, розміщених вище або зліва від вставлених, `xlFormatFromRightOrBelow=1` – копіює формат з клітинок, розміщених нижче або справа від вставлених;

- `Delete(Shift)` – видаляє вміст клітинок поточного діапазону, необов'язковий параметр `Shift` приймає значення елементів енумератора `XlDeleteShiftDirection`, які вказують напрям зсуву клітинок, які заміщують видалені: `xlShiftUp=-4162` – вгору або `xlShiftToLeft=-4159` – вліво;

Наведемо приклад використання методів `Insert(Shift, CopyOrigin)` та `Delete(Shift)` та їх результат, представлений на Рисунок 107.

```
Dim myRange As Range

Set myRange = Range("C4:C9")
myRange.Insert Shift:=xlShiftToRight, CopyOrigin:=True
myRange.Delete Shift:=xlShiftToLeft
```



	A	B	C	D
1	***Привіт***	**світ**		
2	*10,00*		5	3,00
3	*5,00*		**120**	4,00
4	*4,00*			5,00
5	*9,00*			6,00
6	*5,00*			7,00
7	*7,00*			8,00
8	*1,00*			9,00
9	*-3,00*			10,00
10	*0,00*			11,00
11	**38,00**			

	A	B	C	D	E
1	***Привіт***	**світ**			
2	*10,00*		5	3,00	
3	*5,00*		**120**	4,00	
4	*4,00*				5,00
5	*9,00*				6,00
6	*5,00*				7,00
7	*7,00*				8,00
8	*1,00*				9,00
9	*-3,00*				10,00
10	*0,00*			11,00	
11	**38,00**				

а) *б)*

Рисунок 107 – Вигляд робочого аркушу а) до вставки та після видалення діапазону
б) після вставки діапазону

- `Merge(Across)` – створює об'єднану клітинку з усіх клітинок поточного діапазону, у разі наявності різних даних у клітинах діапазону після їх об'єднання залишиться значення однієї верхньої лівої клітинки (про що буде сповіщено у діалоговому вікні). Необов'язковий параметр `Across` може приймати значення `True` – при цьому при наявності декількох рядків у поточному діапазоні, клітинки кожного рядка будуть об'єднані у окремі клітинки, або `False` – у такому разі буде створена одна об'єднана клітинка (`False` – значення за замовчуванням), приклад використання методу `Merge()` (див. вище);
- `UnMerge()` – поділяє об'єднану методом `Merge(Across)` клітинку на звичайні клітинки, що складали об'єднану;
- `Consolidate(Sources, Function, TopRow, LeftColumn, CreateLinks)` – консолідує (сумісно обробляє) дані декількох діапазонів (у тому числі на різних робочих аркушах) агрегатною функцією з розміщенням її результатів у єдиному діапазоні. Параметри метода (всі необов'язкові): `Sources` – масив адрес діапазонів у `R1C1`-стилі, записаних як рядки, `Function` – агрегатна функція, визначена елементом енумератора `XlConsolidationFunction`, найбільш популярними з яких є: `xlSum=-4157` – сума значень даних клітинок діапазонів, `xlAverage=-4106` – середнє значень даних клітинок діапазонів, `xlCount=-4112` – кількість значень даних клітинок діапазонів, `xlMax=-4136` – максимальне зі значень даних клітинок діапазонів, `xlMin=-4139` – мінімальне зі значень даних клітинок діапазонів, `xlProduct=-4149` – добуток значень

даних клітинок діапазонів), `TopRow – True` – консолідує дані на основі заголовків стовпців у верхньому рядку діапазонів консолідації, `False` (за замовчуванням) – консолідує дані на основі їх розміщення, `LeftColumn – True` – консолідує дані на основі заголовків рядків у лівій колонці діапазонів консолідації, `False` (за замовчуванням) – консолідує дані на основі їх розміщення, `CreateLinks – True` - консолідація використовує посилання на робочий аркуш, `False` (за замовчуванням) – консолідація копіює дані. Приклад коду консолідації двох діапазонів на основі заголовків стовпців та заголовків рядків:

```
Dim myRange As Range

Set myRange = Range("G1")
myRange.Consolidate Sources:=Array("Consolidate!R1C1:R10C2", _
       "Consolidate!R1C4:R10C5"), Function:=xlSum, TopRow:=True, _
       LeftColumn:=True, CreateLinks:=False
```

Приклад коду консолідації трьох незалежних діапазонів з отриманням середнього для кожного з рядків:

```
Dim myRange As Range

Set myRange = Range("G12")
myRange.Consolidate Array("R12C1:R16C1", "R12C3:R16C3", _
"R12C5:R16C5"), xlAverage, False, False
```

Результат роботи цих двох прикладів коду наведений на Рисунок 108, кольоровими межами означені згенеровані діапазони. Синьою межею обведений результуючий діапазон при значенні параметрів `TopRow:=False, LeftColumn:=True`, зеленою – при значенні параметрів `TopRow:=True, LeftColumn:= False`, при обраних у коді значеннях бачимо їх об'єднання. Коричневою межею обведений результуючий діапазон консолідації без врахування верхнього рядка та лівого стовпця вихідних діапазонів.

	A	B	C	D	E	F	G	H	I
1	**Країна**	**Площа, кв. км**		**Країна**	**Населення, млн.**			**Площа, кв. км**	**Населення, млн.**
2	Україна	603 628		Україна	41,98		Україна	603 628	41,98
3	Канада	9 984 670		Канада	37,59		Канада	9 984 670	37,59
4	Китай	9 598 962		Китай	1404,328		Китай	9 598 962	1404,328
5	Іспанія	505 990		Іспанія	47,35		Іспанія	505 990	47,35
6	США	9 519 431		США	328,2		США	9 519 431	328,2
7	Бразилія	8 515 767		Бразилія	209,5		Бразилія	8 515 767	209,5
8	Австралія	7 692 000		Австралія	24,99		Австралія	7 692 000	24,99
9	Індія	3 287 263		Індія	1 383,81		Індія	3 287 263	1 383,81
10									
11							**Середнє**		
12	9		6		5		6,666667		
13	9		10		10		9,666667		
14	10		1		7		6		
15	10		6		6		7,333333		
16	3		6		8		5,666667		

Рисунок 108 – Результат консолідації даних діапазонів

- `Copy(Destination)` – виконує копіювання поточного діапазону до іншого діапазону, зазначеного необов'язковим параметром `Destination`, у який

буде виконано копіювання поточного діапазону (тобто фактично виконується операцію копіювання та вставки), або до Буферу обміну (у разі, якщо параметр не зазначається). Наступний приклад коду виконує копіювання діапазону зі вставкою на поточний робочий аркуш та на інший робочий аркуш:

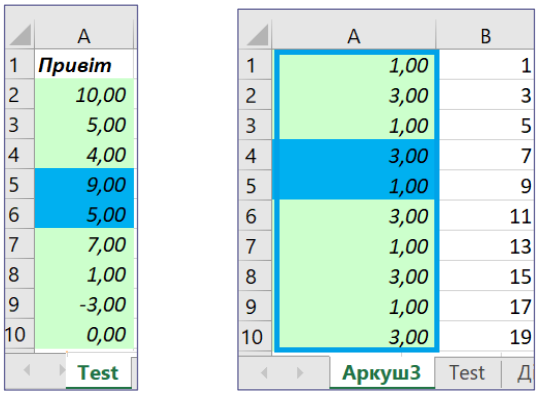
```
Dim myRange As Range

Worksheets("Test").Activate
Set myRange = Range("A2:A10")
myRange.Copy Range("B2:B10")
myRange.Copy Worksheets("Аркуш1").Range("A2:A10")
```

- `Cut()` – вирізає поточний діапазон до Буферу обміну та вставляє його у діапазон, вказаний необов'язковий параметром `Destination`, якщо він зазначений (тобто фактично виконується операцію копіювання та вставки). Якщо параметр `Destination` не вказаний, вихідний діапазон тільки вирізається до Буферу обміну. Наступний приклад коду виконує вирізання діапазону зі вставкою на інший робочий аркуш:

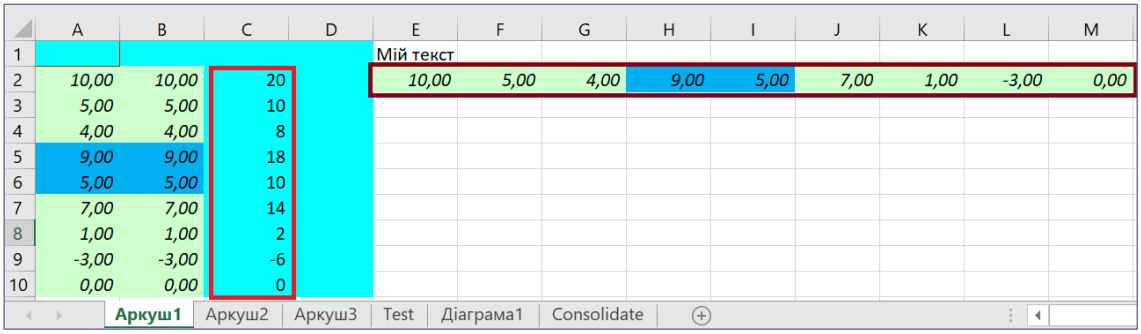
```
Dim myRange As Range

Worksheets("Test").Activate
Set myRange = Range("B2:B10")
myRange.Cut Worksheets("Аркуш1").Range("B2:B10")
```

- `PasteSpecial(Paste, Operation, SkipBlanks, Transpose)` – дозволяє вставити вміст Буферу обміну до поточного діапазону. Може приймати необов'язкові параметри: `Paste` – визначає, що буде вставлятися, приймає значення елементів енумератора `XlPasteType`, серед яких найбільш поширені – `xlPasteValues=-4163` – вставка тільки значень без ознак форматування, `xlPasteFormats=-4122` – вставка тільки ознак форматування без значень, `xlPasteAll=-4104` – вставка і значень, і ознак форматування – значення за замовчуванням); `Operation` – визначає тип операції над даними, які вставляються та даними діапазону, до якого виконується вставка, приймає значення елементів енумератора `XlPasteSpecialOperation`, серед яких найбільш поширені – `xlPasteSpecialOperationAdd=2` – виконується додавання до даних діапазону, `xlPasteSpecialOperationMultiply=4` – виконується множення з даними діапазону, `xlPasteSpecialOperationSubtract=3` – виконується віднімання від даних діапазону, `xlPasteSpecialOperationDivide=5` – виконується ділення даних діапазону, `xlPasteSpecialOperationNone=-4142` – виконується вставка без виконання операцій. Параметр `SkipBlanks` дозволяє пропускати пусті клітинки у поточному діапазоні і не копіювати їх у цільовий діапазон (значення `True`). Параметр `Transpose` зі значенням `True` дозволить дані діапазону, розміщені у стовпцях, розмістити у рядках і навпаки. Наступний код демонструє використання методу `PasteSpecial` з різними параметрами, коментарі пояснюють його частини, а на Рисунок 109 показаний вихідний діапазон та результат виконання коду:

```
Dim myRange As Range

Worksheets("Test").Activate

Set myRange = Range("A2:A10")
myRange.Copy
'Вставка тільки значень
Worksheets("Аркуш1").Activate
Set myRange = Range("C2:C10")
myRange.PasteSpecial xlPasteValues

'Вставка тільки ознак форматування
Worksheets("Аркуш3").Activate

'Цільовий діапазон більше аніж той, що копіюється
Set myRange = Range("A1:A10")
myRange.PasteSpecial xlPasteFormats

'Вставка з операцією додавання
Worksheets("Аркуш1").Activate
Set myRange = Range("C2:C10")
myRange.PasteSpecial xlPasteValues, xlPasteSpecialOperationAdd

'Вставка з транспозицією
Range("E2").PasteSpecial Transpose:=True
```



а) *б)*



в)

Рисунок 109 – Застосування методу PasteSpecial: а) вихідний діапазон, що копіюється, б) вставка ознак форматування, в) вставка з додаванням та вставка з транспозицією

Зауважимо, що перше використання методу PasteSpecial вставляє у діапазон C2:C10 тільки значення вихідного діапазону, на Рисунок 109 видно результат додавання цих значень до даних вихідного діапазону, який вставляється з операцією додавання.

- Find(What, After, LookIn, LookAt, SearchOrder, SearchDirection, MatchCase, MatchByte, SearchFormat) – виконує пошук зазначених даних у клітинках поточного діапазону, повертає об'єкт Range, який представляє першу клітинку, у якій було знайдені дані, що шукалися. Обов'язковий параметр What зазначає критерій пошуку, він має тип Variant. Інші параметри необов'язкові: After – дозволяє почати пошук в діапазоні, починаючи з клітинки цього діапазону, вказаної як значення цього параметру у вигляді об'єкту Range; LookIn – визначає тип критерія пошуку, приймає як значення елементи енумератора XlFindLookIn, найчастіше – xlValues=-4163 – значення (за замовчуванням), xlFormulas=-4123 - формули, xlComments=-4144 –коментарі; LookAt – дозволяє визначити шаблон пошуку, приймає як значення елементи енумератора XlLookAt – xlWhole=1 – критерій пошуку повинен співпадати з повним вмістом клітинки або xlPart=2 – критерій пошуку повинен співпадати з частиною вмісту клітинки (за замовчуванням); SearchOrder – визначає спосіб пошуку, приймає як значення елементи енумератора XlSearchOrder – xlByRows=1 – пошук виконується за рядками (за замовчуванням) або xlByColumns=2 – пошук виконується за стовпцями; SearchDirection – визначає напрямок пошуку, приймає як значення елементи енумератора XlSearchDirection – xlNext=1 – шукає збіги у наступних клітинках діапазону (за замовчуванням) або xlPrevious=2 – шукає збіги у попередніх клітинках діапазону; MatchCase – визначає чутливість до регістру символів при пошуку – True – шукати з врахуванням регістру символів, False – шукати без врахування регістру символів (за замовчуванням); MatchByte – використовується тільки у разі використання двобайтових символів, у разі значення True, шукається співпадіння текстових даних-зразка, записаних двобайтовими символами з даними клітинок, записаних двобайтовими символами, у разі значення False, шукається співпадіння текстових даних, записаних двобайтовими символами з їх однобайтовими еквівалентами (цей параметр не впливає на пошук при використанні латинських та кириличних символів); SearchFormat – дозволяє виконувати пошук шаблону з зазначеними ознаками форматування, також регулюється значеннями True/False, спочатку шаблон пошуку повинен бути встановленим як властивість Application.FindFormat. На Рисунок 110 показаний робочий аркуш, для даних якого застосовані наведені нижче приклади використання методу Find з пояснюючими коментарями та коментарями з результатами пошуку:

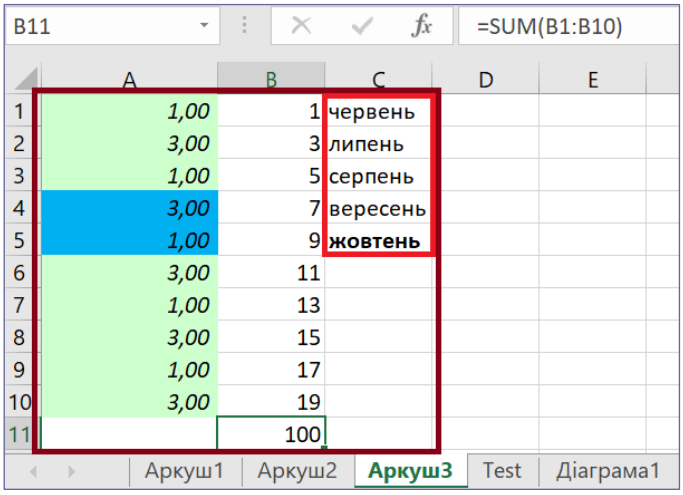


Рисунок 110 – Робочий аркуш з даними, на якому виконується пошук методами Find та FindNext

```
Dim myRange As Range, foundedange As Range

Worksheets("Аркуш3").Activate
Set myRange = Range("C1:C5")
'Простий пошук (за замовчуванням - за рядками, зі співпадінням
'критерія пошуку з частиною вмісту клітинки, без врахування
'регістру символів)
With myRange
    Set foundedRange = .Find(What:="ВЕРЕСЕНЬ")
    RangeSearchResult foundedRange   'Дані знайдені у клітинці$C$4
End With

'Пошук по частині співпадіння з опцією пошуку по повному вмісту
'клітинки
With myRange
    Set foundedRange = .Find(What:="верес", LookAt:=xlWhole)
    RangeSearchResult foundedRange    'Дані не знайдені
End With

'Пошук з урахуванням регістру символів
With myRange
    Set foundedRange = .Find(What:="ВЕРЕСЕНЬ", MatchCase:=True)
    RangeSearchResult foundedRange    'Дані не знайдені
End With

'Пошук за стовпцями
With myRange
    Set foundedRange = .Find(What:="Вересень", _
                             SearchOrder:=xlByColumns)
    RangeSearchResult foundedRange   'Дані знайдені у клітинці$C$4
End With

'Пошук формули
Set myRange = Range("A1:C11")
With myRange
    Set foundedRange = .Find(What:="=SUM(B1:B10)", _
```

```
                            After:=Range("A10"), LookIn:=xlFormulas)
    RangeSearchResult foundedRange  'Дані знайдені у клітинці$B$11
End With

'Пошук ознак форматування
Range("C5").Font.Bold = True
Application.FindFormat.Clear
Application.FindFormat.Font.Bold = True
With myRange
    Set foundedRange = .Find(What:="*", SearchFormat:=True)
    RangeSearchResult foundedRange   'Дані знайдені у клітинці$C$5
End With
```

Код використовує допоміжну процедуру, яка, у разі знайдення даних, виводить адресу першої клітинки зі знайденими даними:

```
Sub RangeSearchResult(rng As Range)
    If rng Is Nothing Then
        Debug.Print "Дані не знайдені"
    Else
        Debug.Print "Дані знайдені у клітинці" & rng.Address
    End If
End Sub
```

Зверніть увагу, на шаблон пошуку `"*"`, що використовується при пошуку ознак форматування.

Метод `Find` знаходить тільки першу у діапазоні клітинку, вміст якої відповідає критеріям пошуку. У разі, якщо необхідно знайти всі такі клітинки, разом з методом `Find` використовують методи `FindNext(After)` та `FindPrevious(Before)`, які дозволяють продовжити пошук, розпочатий методом `Find`, у відповідних напрямках, починаючи з найденого попереднім пошуком діапазону, що передається їм як аргумент `After` або `Before`, відповідно. Наведемо приклад коду, який знаходить адреси усіх клітинок діапазону A1:C11 робочого аркушу, представленого на Рисунок 110, що відповідають критерію пошуку:

```
Dim myRange As Range, foundedange As Range
Dim firstFoundAddress As String

Worksheets("Аркуш3").Activate
Set myRange = Range("C1:C5")

Set foundedRange = myRange.Find(What:=3)
If Not foundedRange Is Nothing Then
    firstFoundAddress = foundedRange.Address
    Do
        Debug.Print "Дані знайдені у клітинці " _
                    & foundedRange.Address
        Set foundedRange = myRange.FindNext(foundedRange)
    Loop While Not foundedRange Is Nothing _
                And firstFoundAddress <> foundedRange.Address
Else
```

```
    Debug.Print "Дані не знайдені"
End If
```

Особливістю роботи методів `FindNext(After)`/ `FindPrevious(Before)` є те, що після знаходження останньої/першої клітинки діапазону, яка відповідає критеріям пошуку, пошук продовжується знову з початку/кінця діапазону. Тому у наведеному вище прикладі адреса першої знайденої клітинки порівнюється з адресою кожної наступної знайденої клітинки і у разі, якщо вони співпадають, програма завершує роботу. Результат роботу наведеного фрагменту коду:

Дані знайдені у клітинці $A$2
Дані знайдені у клітинці $A$4
Дані знайдені у клітинці $A$6
Дані знайдені у клітинці $A$8
Дані знайдені у клітинці $A$10
Дані знайдені у клітинці $B$2

- `Replace(What, Replacement, LookAt, SearchOrder, MatchCase, MatchByte, SearchFormat, ReplaceFormat)` – виконує пошук та заміну значень у клітинках діапазону, повертає значення `Boolean`, яке вказує, чи відбулась хоча б одна заміна (`True` – якщо так). Обов'язкові параметри – `What` – зазначає критерій пошуку та `Replacement` – містить дані, на які будуть замінені знайдені за критерієм дані, обидва параметри мають тип `Variant`. Необов'язкові параметри: `LookAt`, `SearchOrder`, `MatchCase`, `MatchByte`, `SearchFormat` працюють так, як у методі `Find()` (див. вище). Необов'язковий параметр `ReplaceFormat` – значеннями `True` або `False` зазначає, чи необхідно змінювати ознаки форматування знайдених за критерієм пошуку даних, попередньо шаблон їх ознак форматування, на які відбудеться заміна, повинен бути встановленим до властивості `Application.ReplaceFormat`. Наведемо приклади використання методу `Replace` з пояснюючими коментарями для робочого аркушу за даними, представленого на Рисунок 110:

```
Dim myRange As Range

Worksheets("Аркуш3").Activate
'Проста заміна
Set myRange = Range("C1:C5")
isReplaced = myRange.Replace(What:="вересень",
Replacement:="вересень-початок семестру")
Debug.Print "Заміна виконана: " & isReplaced

'Заміна з врахуванням та заміною ознак форматування
Application.FindFormat.Clear
Application.FindFormat.Font.Bold = True
Application.ReplaceFormat.Font.Bold = False
isReplaced = myRange.Replace(What:="жовтень", _
                        Replacement:="ЖОВТЕНЬ", MatchCase:=True, _
                        SearchFormat:=True, ReplaceFormat:=True)
```

Результат виконання цього фрагменту коду показаний на Рисунок 111 , до вікна *Immediate* буде виведено: Заміна виконана: True

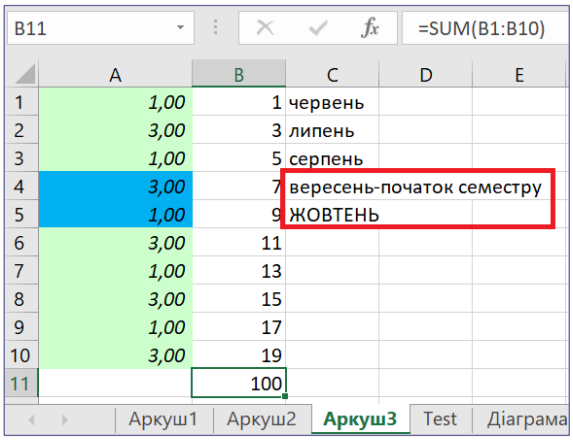


Рисунок 111. Робочий аркуш з даними, зміненими методом Replace

- `Sort(Key1, Order1, Key2, Type, Order2, Key3, Order3, Header, OrderCustom, MatchCase, Orientation, SortMethod, DataOption1, DataOption2, DataOption3)` – виконує сортування (максимум трьохрівневе) вмісту клітинок поточного діапазону. Приймає досить багато необов'язкових параметрів для налаштування сортування, опишемо найбільш використовувані. Параметри `KeyN (N=1-3)` – вказують поля (найчастіше стовпці) як об'єкти `Range` для вкладеного (максимум трьохрівневого) сортування, параметри `OrderN` вказують порядок сортування як елементи енумератора `XlSortOrder`: `xlAscending=1` – за зростанням (за замовчуванням) або `xlDescending=2` – за зменшенням значень у клітинках. Багаторівневе сортування – це сортування за значеннями у наступному полі, якщо в попередньому полі є однакові значення. Параметр `Type` відноситься до налаштувань сортування даних зведеної таблиці (ми не будемо розглядати цей функціонал, оскільки його корисніше використовувати вручну, без програмування на VBA). Параметр `Header` визначає, чи вважати перший рядок діапазону, що сортується, рядком заголовків, приймає як значення елементи енумератора `XlYesNoGuess`: `xlYes=1` – так, перший рядок – це заголовок і він не приймає участі у сортуванні, `xlNo` – ні, перший рядок не є заголовком і сортується разом з іншими рядками (значення за замовчуванням), `xlGuess` – Microsoft Excel автоматично визначає, чи є рядок заголовку у діапазоні і застосовує для нього відповідні правила сортування. Параметр `MatchCase` визначає, чи враховувати регістр символів при сортуванні (його значення `True/False`). Параметр `Orientation` визначає чи будуть сортуватися рядки – значення `xlSortRows=2` енумератора `XlSortOrientation` (значення за замовчуванням), чи будуть сортуватися стовпці – значення `xlSortColumns=1`. Параметри `OrderCustom`, `SortMethod`, `DataOption1`, `DataOption2`, `DataOption3` використовуються не часто. Наведемо приклад багаторівневого сортування списку уявних клієнтів уявної фірми, представленого на Рисунок 112. Нехай, необхідно їх відсортувати спочатку за назвою штату, потім за назвою міста, а потім за прізвищем.

	A	B	C	D	E	F	G
1	first_name	last_name	company_name	address	city	zip	state
2	Nicolette	Brossart	Goulds Pumps Inc Slurry Pump	1 Midway Rd	Westborough	1581	MA
3	Donte	Kines	W Tc Industries Inc	3 Aspen St	Worcester	1602	MA
4	Lucy	Treston	Franz Inc	57254 Brickell Ave #372	Worcester	1602	MA
5	Levi	Munis	Farrell & Johnson Office Equip	2094 Ne 36th Ave	Worcester	1603	MA
6	Annabelle	Boord	Corn Popper	523 Marquette Ave	Concord	1742	MA
7	Katina	Polidori	Cape & Associates Real Estate	5 Little River Tpke	Wilmington	1887	MA
8	Jina	Briddick	Grace Pastries Inc	38938 Park Blvd	Boston	2128	MA
9	Luisa	Jurney	Forest Fire Laboratory	25 Se 176th Pl	Cambridge	2138	MA
10	Oretha	Menter	Custom Engineering Inc	8 County Center Dr #647	Boston	2210	MA
11	Beatriz	Corrington	Prohab Rehabilitation Servs	481 W Lemon St	Middleboro	2346	MA
12	Nobuko	Halsey	Goeman Wood Products Inc	8139 I Hwy 10 #92	New Bedford	2745	MA
13	Corinne	Loder	Local Office	4 Carroll St	North Attleboro	2760	MA
14	Hermila	Thyberg	Chilton Malting Co	1 Rancho Del Mar Shopping C	Providence	2903	RI
15	Blondell	Pugh	Alpenlite Inc	201 Hawk Ct	Providence	2904	RI
16	Lawrence	Lorens	New England Sec Equip Co Inc	9 Hwy	Providence	2906	RI
17	Delmy	Ahle	Wye Technologies Inc	65895 S 16th St	Providence	2909	RI
18	Caitlin	Julia	Helderman, Seymour Cpa	5 Williams St	Johnston	2919	RI
19	Dalene	Riden	Silverman Planetarium	66552 Malone Rd	Plaistow	3865	NH
20	Jolanda	Hanafan	Perez, Joseph J Esq	37855 Nolan Rd	Bangor	4401	ME
21	Goldie	Schirpke	Reuter, Arthur C Jr	34 Saint George Ave #2	Bangor	4401	ME
22	Ceola	Setter	Southern Steel Shelving Co	96263 Greenwood Pl	Warren	4864	ME
23	Ma	Layous	Development Authority	78112 Morris Ave	North Haven	6473	CT
24	Yoko	Fishburne	Sams Corner Store	9122 Carpenter Ave	New Haven	6511	CT
25	Harrison	Haufler	John Wagner Associates	759 Eldora St	New Haven	6515	CT

Аркуш1 | Аркуш2 | Аркуш3 | Test | Діаграма1 | Consolidate | Customers

Рисунок 112 – Робочий аркуш з даними, які необхідно сортувати

Код, який виконує таке сортування, може виглядати так:

```
Dim myRange As Range

Worksheets("Customers").Activate
Set myRange = Range("A1").CurrentRegion
myRange.Sort Key1:=Columns("G"), Key2:=Columns("E"), _
     Order2:=xlDescending, Key3:=Columns("B"), Header:=xlYes
```

Результат сортування наведений на Рисунок 113. Як видно, клієнти відсортовані за штатом – за зростанням, за містом – з зменшенням та за прізвищем – за зростанням. При цьому перший рядок діапазону вважається заголовком та не приймає участі у сортуванні.

	A	B	C	D	E	F	G
1	first_name	last_name	company_name	address	city	zip	state
2	Roxane	Campain	Rapid Trading Intl	1048 Main St	Fairbanks	99708	AK
3	Erick	Ferencz	Cindy Turner Associates	20 S Babcock St	Fairbanks	99712	AK
4	Wilda	Giguere	Mclaughlin, Luther W Cpa	1747 Calle Amanecer #2	Anchorage	99501	AK
5	Gail	Kitty	Service Supply Co Inc	735 Crawford Dr	Anchorage	99501	AK
6	Lenna	Paprocki	Feltz Printing Service	639 Main St	Anchorage	99501	AK
7	Penney	Weight	Hawaiian King Hotel	18 Fountain St	Anchorage	99515	AK
8	Carin	Deleo	Redeker, Debbie	1844 Southern Blvd	Little Rock	72202	AR
9	Regenia	Kannady	Ken Jeter Store Equipment Inc	10759 Main St	Scottsdale	85260	AZ
10	Herminia	Nicolozakes	Sea Island Div Of Fstr Ind Inc	4 58th St #3519	Scottsdale	85254	AZ
11	Keneth	Borgman	Centerline Engineering	86350 Roszel Rd	Phoenix	85012	AZ
12	Christiane	Eschberger	Casco Services Inc	96541 W Central Blvd	Phoenix	85034	AZ
13	Arminda	Parvis	Newtec Inc	1 Huntwood Ave	Phoenix	85017	AZ
14	Mattie	Poquette	Century Communications	73 State Road 434 E	Phoenix	85013	AZ
15	Elke	Sengbusch	Riley Riper Hollin & Colagreco	9 W Central Ave	Phoenix	85013	AZ
16	Helene	Rodenberger	Bailey Transportation Prod Inc	347 Chestnut St	Peoria	85381	AZ
17	Iluminada	Ohms	Nazette Marner Good Wendt	72 Southern Blvd	Mesa	85204	AZ
18	Shenika	Seewald	East Coast Marketing	4 Otis St	Van Nuys	91405	CA
19	Gail	Similton	Johnson, Wes Esq	62 Monroe St	Thousand Palms	92276	CA
20	Melodie	Knipp	Fleetwood Building Block Inc	326 E Main St #6496	Thousand Oaks	91362	CA
21	Shawna	Palaspas	Windsor, James L Esq	5 N Cleveland Massillon Rd	Thousand Oaks	91362	CA
22	Alyce	Arias	Fairbanks Scales	3196 S Rider Trl	Stockton	95207	CA
23	Mozell	Pelkowski	Winship & Byrne	577 Parade St	South San Francisco	94080	CA
24	Rasheeda	Sayaphon	Kummerer, J Michael Esq	251 Park Ave #979	Saratoga	95070	CA
25	Charlene	Hamilton	Oshins & Gibbons	985 E 6th Ave	Santa Rosa	95407	CA

Аркуш1 | Аркуш2 | Аркуш3 | Test | Діаграма1 | Consolidate | Customers

Рисунок 113 - Робочий аркуш з відсортованими даними

- SpecialCells(Type, Value) – повертає об'єкт Range, який містить усі клітинки поточного діапазону з вказаними аргументами типом та значенням. Обов'язковий параметр Type визначає тип клітинок діапазону, що будуть додані до результуючого діапазону, як елемент енумератора XlCellType, найчастіше – xlCellTypeConstants=2 – клітинки з постійними значеннями, xlCellTypeFormulas=-4123 – клітинки з формулами, xlCellTypeBlanks=4 – порожні клітинки, xlCellTypeComments – клітинки з коментарями, xlCellTypeLastCell – остання у діапазоні клітинка. Необов'язковий параметр Value який уточнює значення клітинок з типом xlCellTypeConstants або xlCellTypeFormulas, приймає як значення елемент енумератора XlSpecialCellsValue: xlNumbers=1 – клітинки з числами, xlTextValues=2 – клітинки з текстом, xlLogical=4 – клітинки з булевими значеннями (True або False), xlErrors=16 – клітинки з помилками. Наприклад, щоб для робочого аркушу, показаного на Рисунок 111, повернути об'єкт Range, що складається тільки з чисел, можна використати код:

```
Worksheets("Аркуш3").Activate
Set myRange = Range("A1:C11")
Set foundedRange = myRange.SpecialCells(xlCellTypeConstants, _
                                        xlNumbers)
Debug.Print "Діапазон з числами: " & foundedRange.Address
```

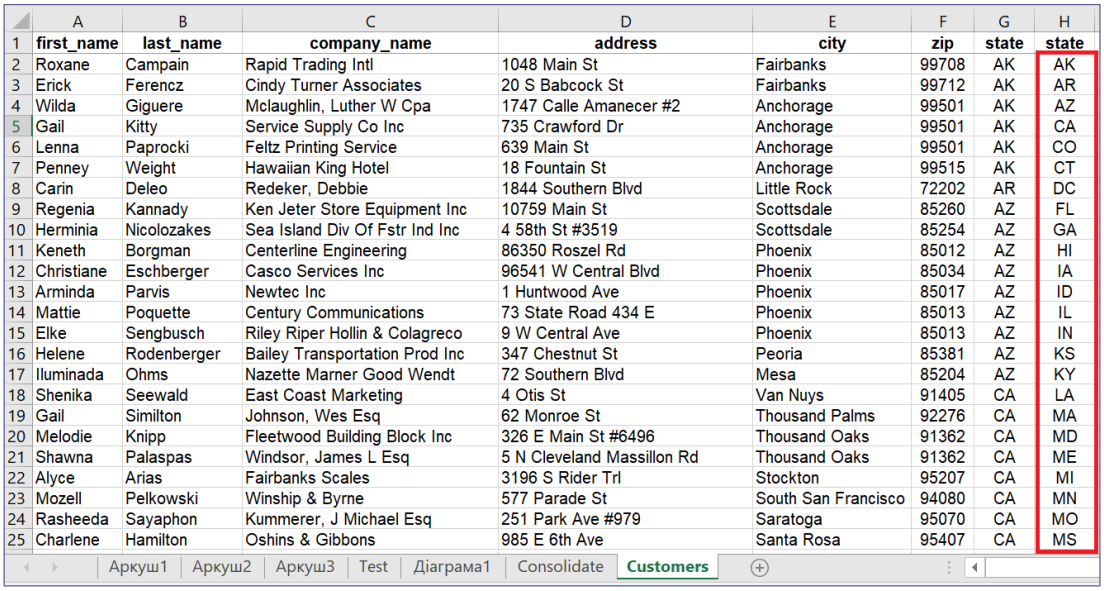
Код виводить до вікна *Immediate* адресу відповідного діапазону:

```
Діапазон з числами: $A$1:$B$10
```

- RemoveDuplicates(Columns, Header) – видаляє дубльовані значення з поточного діапазону, цей метод часто використовується для отримання словників зі списків даних. Обов'язковий параметр Columns зазначає стовпці, у яких будуть шукатися та видалятися дублікати, вони зазначаються як масив з індексами стовпців у поточному діапазоні, з якого викликається метод. Необов'язковий параметр Header елементом енумератора XlYesNoGuess (див. вище) зазначає, чи вважати перший рядок заголовком. Наведемо приклад створення словника з назвами штатів зі списку даних на Рисунок 113:

```
Worksheets("Customers").Activate
Columns("G").Copy Columns("H")
Set myRange = Columns("H")
myRange.RemoveDuplicates Columns:=Array(1), Header:=xlYes
```

Спочатку стовпець з назвами штатів копіюється, щоб вихідні дані залишилися незмінними, а потім з нього видаляються дублікати, також приймається, що перший рядок є заголовком. Результат виконання коду показаний на Рисунок 114.



	A	B	C	D	E	F	G	H
1	first_name	last_name	company_name	address	city	zip	state	state
2	Roxane	Campain	Rapid Trading Intl	1048 Main St	Fairbanks	99708	AK	AK
3	Erick	Ferencz	Cindy Turner Associates	20 S Babcock St	Fairbanks	99712	AK	AR
4	Wilda	Giguere	Mclaughlin, Luther W Cpa	1747 Calle Amanecer #2	Anchorage	99501	AK	AZ
5	Gail	Kitty	Service Supply Co Inc	735 Crawford Dr	Anchorage	99501	AK	CA
6	Lenna	Paprocki	Feltz Printing Service	639 Main St	Anchorage	99501	AK	CO
7	Penney	Weight	Hawaiian King Hotel	18 Fountain St	Anchorage	99515	AK	CT
8	Carin	Deleo	Redeker, Debbie	1844 Southern Blvd	Little Rock	72202	AR	DC
9	Regenia	Kannady	Ken Jeter Store Equipment Inc	10759 Main St	Scottsdale	85260	AZ	FL
10	Herminia	Nicolozakes	Sea Island Div Of Fstr Ind Inc	4 58th St #3519	Scottsdale	85254	AZ	GA
11	Keneth	Borgman	Centerline Engineering	86350 Roszel Rd	Phoenix	85012	AZ	HI
12	Christiane	Eschberger	Casco Services Inc	96541 W Central Blvd	Phoenix	85034	AZ	IA
13	Arminda	Parvis	Newtec Inc	1 Huntwood Ave	Phoenix	85017	AZ	ID
14	Mattie	Poquette	Century Communications	73 State Road 434 E	Phoenix	85013	AZ	IL
15	Elke	Sengbusch	Riley Riper Hollin & Colagreco	9 W Central Ave	Phoenix	85013	AZ	IN
16	Helene	Rodenberger	Bailey Transportation Prod Inc	347 Chestnut St	Peoria	85381	AZ	KS
17	Iluminada	Ohms	Nazette Marner Good Wendt	72 Southern Blvd	Mesa	85204	AZ	KY
18	Shenika	Seewald	East Coast Marketing	4 Otis St	Van Nuys	91405	CA	LA
19	Gail	Similton	Johnson, Wes Esq	62 Monroe St	Thousand Palms	92276	CA	MA
20	Melodie	Knipp	Fleetwood Building Block Inc	326 E Main St #6496	Thousand Oaks	91362	CA	MD
21	Shawna	Palaspas	Windsor, James L Esq	5 N Cleveland Massillon Rd	Thousand Oaks	91362	CA	ME
22	Alyce	Arias	Fairbanks Scales	3196 S Rider Trl	Stockton	95207	CA	MI
23	Mozell	Pelkowski	Winship & Byrne	577 Parade St	South San Francisco	94080	CA	MN
24	Rasheeda	Sayaphon	Kummerer, J Michael Esq	251 Park Ave #979	Saratoga	95070	CA	MO
25	Charlene	Hamilton	Oshins & Gibbons	985 E 6th Ave	Santa Rosa	95407	CA	MS

Аркуш1 | Аркуш2 | Аркуш3 | Test | Діаграма1 | Consolidate | Customers

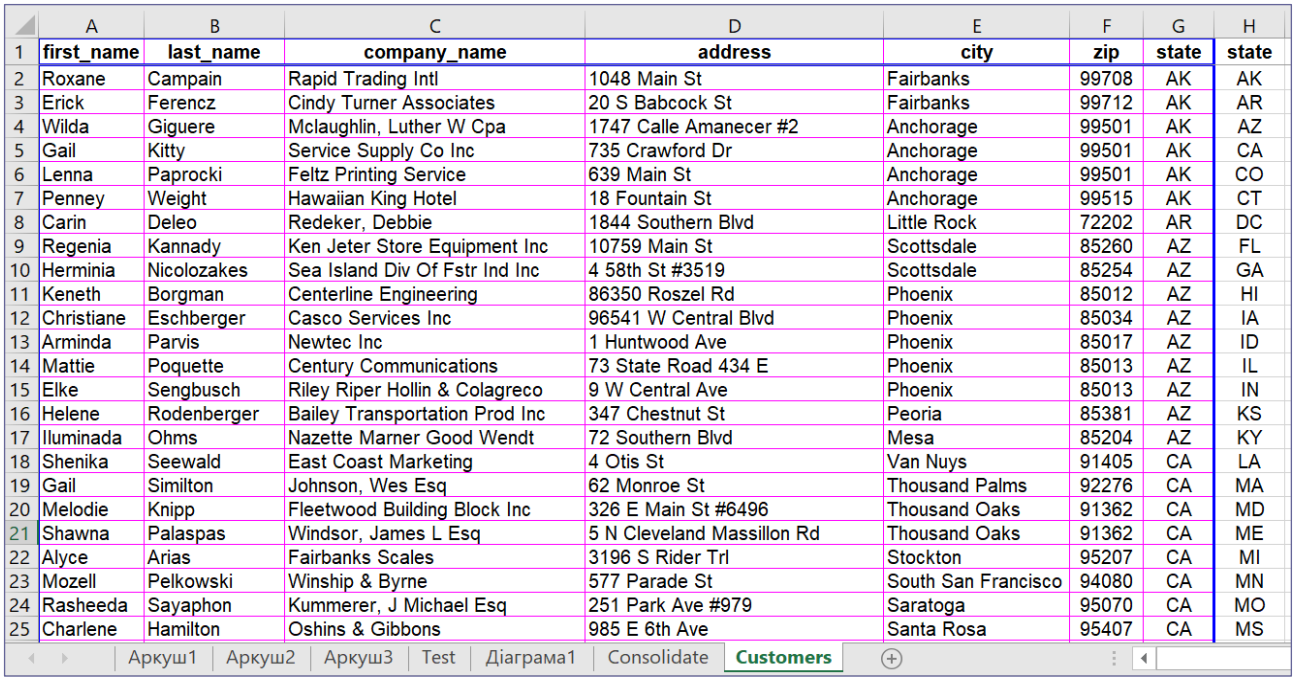
Рисунок 114 – Робочий аркуш зі словником штатів, отриманим видаленням дублікатів

Розглянемо ще метод, який часто використовується при форматуванні діапазонів разом із їх властивістю `Borders` (див. вище):

- `BorderAround(LineStyle, Weight, ColorIndex, Color, ThemeColor)` – додає межу навколо діапазону з визначеними необов'язковими параметрами: `LineStyle` – стиль границі – приймає значення, як властивість `LineStyle` об'єкту `Border` (див. вище); `Weight` – товщина границі – приймає значення, як властивість `Weight` об'єкту `Border` (див. вище), `ColorIndex` – значення індексу кольору для фону (Рисунок 100), `Color` – визначає колір межі, що задається вбудованою функцією RGB (див. вище), `ThemeColor` – колір теми, може зазначатися як елемент енумератора `XlThemeColor` [35], оскільки колір зазначається тільки одним з параметрів `ColorIndex`, `Color` або `ThemeColor`, останній використовується рідко, також товщину границі можна задати або параметром `LineStyle`, або параметром `Weight`. Наступний приклад демонструє створення меж для списку клієнтів, показаного на Рисунок 114:

```
myWbk.Worksheets("Customers").Activate
Set myRange = Range("A1:G501")
With myRange.Borders
    .Color = RGB(255, 0, 255)
    .Weight = xlThin
End With
myRange.BorderAround Weight:=xlThick, Color:=RGB(0, 0, 255)
With Range("A1:G1").Borders(xlEdgeBottom)
    .Weight = xlThick
    .Color = RGB(0, 0, 255)
End With
```

Результат роботи наведеного коду показаний на Рисунок 115.



	A	B	C	D	E	F	G	H
1	first_name	last_name	company_name	address	city	zip	state	state
2	Roxane	Campain	Rapid Trading Intl	1048 Main St	Fairbanks	99708	AK	AK
3	Erick	Ferencz	Cindy Turner Associates	20 S Babcock St	Fairbanks	99712	AK	AR
4	Wilda	Giguere	Mclaughlin, Luther W Cpa	1747 Calle Amanecer #2	Anchorage	99501	AK	AZ
5	Gail	Kitty	Service Supply Co Inc	735 Crawford Dr	Anchorage	99501	AK	CA
6	Lenna	Paprocki	Feltz Printing Service	639 Main St	Anchorage	99501	AK	CO
7	Penney	Weight	Hawaiian King Hotel	18 Fountain St	Anchorage	99515	AK	CT
8	Carin	Deleo	Redeker, Debbie	1844 Southern Blvd	Little Rock	72202	AR	DC
9	Regenia	Kannady	Ken Jeter Store Equipment Inc	10759 Main St	Scottsdale	85260	AZ	FL
10	Herminia	Nicolozakes	Sea Island Div Of Fstr Ind Inc	4 58th St #3519	Scottsdale	85254	AZ	GA
11	Keneth	Borgman	Centerline Engineering	86350 Roszel Rd	Phoenix	85012	AZ	HI
12	Christiane	Eschberger	Casco Services Inc	96541 W Central Blvd	Phoenix	85034	AZ	IA
13	Arminda	Parvis	Newtec Inc	1 Huntwood Ave	Phoenix	85017	AZ	ID
14	Mattie	Poquette	Century Communications	73 State Road 434 E	Phoenix	85013	AZ	IL
15	Elke	Sengbusch	Riley Riper Hollin & Colagreco	9 W Central Ave	Phoenix	85013	AZ	IN
16	Helene	Rodenberger	Bailey Transportation Prod Inc	347 Chestnut St	Peoria	85381	AZ	KS
17	Iluminada	Ohms	Nazette Marner Good Wendt	72 Southern Blvd	Mesa	85204	AZ	KY
18	Shenika	Seewald	East Coast Marketing	4 Otis St	Van Nuys	91405	CA	LA
19	Gail	Similton	Johnson, Wes Esq	62 Monroe St	Thousand Palms	92276	CA	MA
20	Melodie	Knipp	Fleetwood Building Block Inc	326 E Main St #6496	Thousand Oaks	91362	CA	MD
21	Shawna	Palaspas	Windsor, James L Esq	5 N Cleveland Massillon Rd	Thousand Oaks	91362	CA	ME
22	Alyce	Arias	Fairbanks Scales	3196 S Rider Trl	Stockton	95207	CA	MI
23	Mozell	Pelkowski	Winship & Byrne	577 Parade St	South San Francisco	94080	CA	MN
24	Rasheeda	Sayaphon	Kummerer, J Michael Esq	251 Park Ave #979	Saratoga	95070	CA	MO
25	Charlene	Hamilton	Oshins & Gibbons	985 E 6th Ave	Santa Rosa	95407	CA	MS

Аркуш1 | Аркуш2 | Аркуш3 | Test | Діаграма1 | Consolidate | **Customers**

Рисунок 115 – Робочий аркуш зі списком з доданими методом BorderAround та властивістю Borders межами

Питання для самоконтролю

1. Назвіть задачі, які можуть бути автоматизовані при роботі з документами Microsoft Excel.
2. Опишіть способи зберігання даних та координати для їх пошуку у файлі робочої книги Microsoft Excel.
3. Назвіть основні класи об'єктної моделі Microsoft Excel.
4. Назвіть клас, що відповідає екземпляру застосунку Microsoft Excel. Назвіть його найважливіші властивості.
5. Як запустити програмним чином Microsoft Excel із появою його на екрані?
6. Яким чином можна отримати посилання на вже запущений застосунок Microsoft Excel?
7. Як отримати доступ до робочого аркушу вже відкритої робочої книги?
8. Назвіть властивість класу `Application`, яка дозволяє посилатися на всі клітинки активного робочого аркушу, або на зазначену параметрами цієї властивості клітинку.
9. Наведіть приклади посилання на певний стовпець та певний рядок активного робочого аркушу через властивість класу `Application`.
10. Наведіть приклади адресації клітинки та діапазону клітинок за допомогою властивості `Range` класу `Application`. Яким чином організується неявний виклик цієї властивості?
11. Яка властивість класу `Application` відповідає виділеному на активному робочому аркуші діапазону? Чому дорівнює ця властивість, якщо нічого не виділено?
12. Чим відрізняються властивості-колекції `Sheets` та `Worksheets` класу `Application`?
13. Що представляє собою *Особиста книга макросів*? Назвіть ім'я файлу, у якому

вона знаходиться та її місце зберігання за замовчуванням.

14. Назвіть властивість класу `Application`, яка дозволяє використовувати у програмі вбудовані функції Microsoft Excel та наведіть приклад її використання.
15. Які стилі адресації клітинок підтримує Microsoft Excel, яка властивість відповідає за їх переключення?
16. Наведіть приклад адресації клітинки на певному робочому аркуші з певним ім'ям та у робочій книзі з певним ім'ям.
17. Назвіть приклади практичного застосування властивості `DisplayAlerts` класу `Application`.
18. Опишіть призначення методу `Calculate()` класу `Application`. Разом з якою властивістю цього класу він зазвичай використовується?
19. Наведіть приклад перетворення типу посилання діапазону з відносного на абсолютний за допомогою методу класу `Application`.
20. Назвіть призначення та принцип роботи методу `Evaluate(Name)` класу `Application`. Які аргументи може приймати її параметр? Наведіть приклади. Як можна організувати неявний виклик цього методу?
21. Для чого використовують метод `GetOpenFilename()` класу `Application`? Охарактеризуйте параметри цього методу.
22. Для чого використовують метод `GetSaveAsFilename()` класу `Application`? Охарактеризуйте параметри цього методу.
23. Для чого використовують метод `GoTo()` класу `Application`? Охарактеризуйте параметри цього методу.
24. Охарактеризуйте методи `Intersect(Arg1, Arg2, ..., Arg30)` та `Union(Arg1, Arg2, ..., Arg30)` класу `Application`. Наведіть приклади їх використання.
25. Охарактеризуйте методи `OnTime(EarliestTime, Procedure, LatestTime, Schedule)`, `SendKeys(Keys, Wait)` та `Wait(Time)` класу `Application`. Наведіть приклади їх використання. Які символи відповідають натиснутим клавішам *Ctrl* та *Shift*?
26. Охарактеризуйте колекцію `Workbooks`. Які параметри може приймати метод `Add()` цієї колекції?
27. Надайте характеристику найважливішим методам колекції `Workbooks`. Охарактеризуйте параметри методів.
28. Якими методами можна завантажити до робочої книги дані текстового файлу, XML файлу та бази даних Microsoft Access? Охарактеризуйте параметри методів та наведіть приклади їх використання.
29. Чим відрізняються властивості `ThisWorkbook` та `ActiveWorkbook` класу `Application`?
30. Назвіть основні властивості-колекції класу `Workbook` та надайте їм характеристику.
31. Назвіть найважливіші методи класу `Workbook` та надайте їм характеристику.
32. Назвіть основні формати, у яких може зберігатися робоча книга Microsoft

Excel.

33. Що являє собою програмний модуль ЦяКнига у проєкті відкритої робочої книги і як він може бути використаний при організації обробки подій?
34. Назвіть найважливіші властивості колекцій `Sheets` та `Worksheets`. Наведіть приклади їх використання.
35. Назвіть методи класів `Sheets` та `Worksheets`, які дозволяють виконувати додання, видалення, копіювання та переміщення аркушів робочої книги. Охарактеризуйте параметри цих методів та наведіть приклади їх використання.
36. Назвіть властивості класу `Worksheet`, які дозволяють виконувати адресацію клітинок, діапазонів, рядків та стовпців робочого аркушу. Наведіть приклади використання таких властивостей.
37. Назвіть методи класу `Worksheet`, які є однойменними з методами класу `Application`. Поясніть, чим вони відрізняються один від іншого.
38. Що містить колекція `ChartObjects()`, яка повертається однойменним методом класу `Worksheet`? Чим вона відрізняється від колекції `Charts` – властивості класу `Application`?
39. Назвіть методи класу `Worksheet`, які дозволяють виконувати виділення, видалення, копіювання та переміщення аркушів робочої книги. Охарактеризуйте параметри цих методів та наведіть приклади їх використання.
40. Опишіть метод `Paste()` класу `Worksheet` та наведіть приклади його використання. Яким методом можна зберегти робочий аркуш у форматі PDF? Наведіть приклад використання цього методу.
41. Назвіть відомі Вам події класу `Worksheet` та наведіть приклад обробки однієї з них.
42. Назвіть відмінності об'єктів класів `Range` (діапазон) та `Selection` (виділена область).
43. Порівняйте об'єктну модель таблиці документу Microsoft Word з робочим аркушем Microsoft Excel, які класи відповідають клітинкам, діапазонам клітинок, рядкам та стовпцям у першому та другому випадках?
44. Опишіть можливі способи отримання об'єктів класу `Range`, використовуючи об'єкти різних класів об'єктної моделі Microsoft Excel. Наведіть відповідні приклади отримання об'єктів класу `Range`.
45. Назвіть властивості класу `Range`, які дозволяють вводити до діапазонів значення та формули. Наведіть приклади використання таких властивостей.
46. Назвіть властивості класу `Range`, які дозволяють отримувати адресу поточного діапазону, адресу діапазону, що містить клітинку, яку представляє поточний діапазон, та адреси несуміжних піддіапазонів, виділених на робочому аркуші. Наведіть приклади використання таких властивостей.
47. Назвіть властивості класу `Range`, які дозволяють виконувати адресацію клітинок, діапазонів, рядків та стовпців об'єкта цього класу. Наведіть приклади використання таких властивостей.

48. Чим відрізняється робота властивості `Formula` класу `Range` та метода `Evaluate(Name)` класів `Application` та `Worksheet`?
49. Опишіть використання властивості `NumberFormat` класу `Range`. Наведіть приклад використання метасимволів для визначення користувацького формату.
50. Яким чином можна додати межі до клітинок діапазону? Які ознаки форматування можна їм задавати? Які існують варіанти завдання кольору меж?
51. Назвіть властивості класу `Range`, які відповідають за форматування вмісту клітинок діапазону. Чим відрізняється властивість `ShrinkToFit` від методу `Autofit()`?
52. Опишіть методи класу `Range`, які відповідають за заповнення клітин діапазону даними та за їх видалення. Що ще окрім вмісту може бути видалено цими методами?
53. Опишіть методи класу `Range`, які відповідають за вставку та видалення діапазонів даних. Охарактеризуйте їх параметри та наведіть приклади використання.
54. Опишіть методи класу `Range`, які дозволяють об'єднувати клітинки та роз'єднувати об'єднану клітинку. Чим від цих методів відрізняється метод `Consolidate()`? Наведіть приклади використання цих методів.
55. Опишіть методи класу `Range`, які працюють з Буфером обміну. Які особливості має метод, що виконує вставку даних з Буферу обміну?
56. Які методи дозволяють організувати пошук та заміну вмісту клітинок у Microsoft Excel? Опишіть їх параметри та наведіть приклади використання.
57. Який метод класу `Range` відповідає за організацію сортування вмісту клітинок? Який максимальний рівень вкладеного сортування він підтримує? Опишіть його параметри та наведіть приклади використання.
58. Яку функціональність реалізують методи `SpecialCells()`, `RemoveDuplicates()`, та `BorderAround()` класу `Range`? Чим останній метод відрізняється від властивості `Borders` цього класу? Наведіть приклади використання цих методів.

Тестові завдання для перевірки знань

1. Оберіть об'єкти об'єктної моделі Microsoft Excel: (оберіть декілька відповідей):
   а) `Chart` з колекцією `Charts`;
   б) `Sheet` з колекцією `Sheets`;
   в) `Workbook` з колекцією `Workbooks`;
   г) `Worksheet` з колекцією `Worksheets`;
   д) `Cell` з колекцією `Cells`.

2. Оберіть оператори, які виведуть значення першої клітини діапазону 5 (оберіть декілька відповідей):

```
Dim myRange As Range
```

```
Set myRange = Worksheets("Аркуш1").Range("A1:D10")
Range("A1") = 5
```

а) `Debug.Print myRange.Value`
б) `Debug.Print myRange.Text`
в) `Debug.Print myRange.Cells.Item(1).Value`
г) `Debug.Print myRange.Cells(1,1).Value`
д) `Debug.Print myRange.Range("A1").Value`
е) `Debug.Print myRange.Range("A1")`

3. Оберіть адресу діапазону, заданого наступними операторами:

```
Dim myRange As Range

Set myRange = ActiveWorksheet.Range(Cells(1, 3), Cells(2, 4))
```

а) A1:D4;
б) A3:B4;
в) A2:C4;
г) C1:D2.

4. Оберіть адресу клітинки робочого аркушу, у яку буде вставлене число 5 при виконанні коду:

```
Dim myRange As Range, mySubRange As Range
Set myRange = ActiveWorksheet.Range("B1")
Set mySubRange = myRange.Range("B2")
mySubRange.Value = 5
```

а) `A1`
б) `B1`
в) `B2`
г) `C2`
д) `D2`

5. Оберіть визначення для властивості `Areas` об'єкту `Range`:
   а) повертає посилання на колекцію несуміжних піддіапазонів поточного діапазону;
   б) повертає адресу діапазону, який може складатися з несуміжних піддіапазонів;
   в) повертає адресу лівої верхньої та правої нижньої клітинки діапазону несуміжних клітинок
   г) дозволяє обчислити площу діапазону.
6. Оберіть, що зберігає властивість `UsedRange` об'єкта `Worksheet`:
   а) об'єкт `Range`, який представляє прямокутну область без порожніх клітинок;
   б) об'єкт `Range`, який представляє діапазон, який містить тільки порожні клітинки;
   в) об'єкт `Range`, який представляє всі клітинки активного робочого аркушу;
   г) об'єкт `Range`, який представляє прямокутну область, що включає всі непусті клітинки.

7. У разі, якщо у робочій книзі з поточною програмою наявні у вказаному порядку робочі аркуші з такими назвами: "Аркуш1", "Аркуш2", "Діаграма1", "Аркуш3", оберіть куди буде скопійований перший аркуш оператором:

```
ThisWorkbook.Worksheets("Аркуш1").Copy _
       After:=ThisWorkbook.Sheets(ThisWorkbook.Worksheets.Count)
```

а) після "Аркуш1";
б) після "Аркуш2";
в) після "Аркуш3";
г) після "Діаграма1".

8. Оберіть оператори, виконання яких призводить до запуску нового екземпляру Microsoft Excel (оберіть декілька відповідей):

а) `Dim myExcel As New Excel.Application`
б)
```
Dim myExcel As Excel.Application
Set myExcel = GetObject(,"Excel.Application")
```
в) `Dim myExcel As Excel.Application`
г)
```
Dim myExcel As Excel.Application
Set myExcel = New Excel.Application
```

9. Оберіть оператор, який відключає попередження Microsoft Excel при роботі програми:

а) `Application.Interactive = False`
б) `Application.DisplayPromts = False`
в) `Application.EnableEvents = False`
г) `Application.DisplayAlerts = False`

10. Оберіть правильні оператори, які обчислюють значення функції на прикладі функції-факторіалу для діапазону `myRange`, якщо (оберіть декілька відповідей):

```
Dim myRange As Range
Set myRange = ThisWorkbook.Worksheets("Аркуш1").Cells(1, 1)
myRange.Value = 5
```

а) `Debug.Print WorksheetFunction.Fact(myRange)`
б) `Debug.Print Application.Evaluate(Fact(myRange.Value))`
в)
```
ThisWorkbook.Worksheets("Аркуш1").Cells(1,2).Formula = _
                          "=FACT(" & myRange.Value & ")"
```
г) `Debug.Print Application.Evaluate("FACT(myRange.Value)")`
д)
```
ThisWorkbook.Worksheets("Аркуш1").Cells(1,3).Formula = _
                          "=FACT(myRange.Value)"
```
е)
```
Debug.Print Application.Evaluate("FACT(" _
                                 & myRange.Value & ")")
```

Завдання для самостійного виконання

1. Розробити програму, що для десяти студентів, занесених програмою у рядки стовпця A як Студент-N, де N=1-10, генерує випадковим чином у стовпці B оцінку за п'ятибальною шкалою від 2 до 5 включно. Для згенерованих даних програма повинна обрахувати відсоток абсолютної успішності студентів (тих,

хто отримав 3, 4 та 5) та якості навчання (тих, хто отримав 4 та 5) і занести результати у рядки нижче таблиці студентів з оцінками.

2. Розробити програму, що для днів червня минулого року, номери яких заносяться до рядка 1, згенерує цілочисельні значення середньодобової температури від 16 до 31 градусу Цельсія. Програма повинна розрахувати середню температуру за тиждень для повних тижнів місяця та занести отримані дані на окремий робочий аркуш у вигляді таблиці з заголовками тиждень-N, середня за тиждень температура.

3. Розробити програму, що дозволяє з обраного за допомогою діалогового вікна каталогу для усіх файлів робочих книг цього каталогу скопіювати їх робочі аркуші до нової робочої книги під іменами книгаN-оригінальне ім'я робочого аркушу, де N – кількість файлів робочих книг у каталозі. Результуючу таблицю необхідно відформатувати для оптимального відображення даних та додати межі.

4. Розробити програму, що виконує завантаження XML даних щодо офіційного курсу гривні до долара США, отриманих запитом: https://bank.gov.ua/NBU_Exchange/exchange_site?start=yyyymmdd&end=yyyymmdd&valcode=usd&sort=exchangedate&order=desc

де шаблоном yyyymmdd зазначається поточна дата як кінець проміжку часу та за два роки до поточної дати – як початок проміжку часу, та збережених у файл currency.xml. Програма повинна обробити дані, отримавши середні значення курсу за кожен повний місяць, та занести їх на окремий робочий аркуш у вигляді таблиці з заголовками рік, місяць, середній курс долара США. Результуючу таблицю необхідно відформатувати для оптимального відображення даних та додати межі.

5. Розробити програму, що створює список студентів, ідентифікованих у стовпці A, як Студент-N, де N=1-20 та приписує випадковим чином кожному студенту у стовпці B назву групи, як група-1, група-2 та група-3. На окремому аркуші необхідно додати вручну таблицю-довідник груп з полями Назва групи та Курс, у якій для група-1 та група-2 зазначений курс 1, а для група-3 – курс 2. Програма повинна додати до таблиці студентів відповідний групі курс та виконати її сортування за курсом, назвою групи, номером студента.

6. Розробити програму, що створює список співробітників, ідентифікованих у стовпці A, як Співробітник-N, де N=1-20 та приписує випадковим чином кожному співробітнику у стовпці B значення заробітної платні від 12 до 25 тис. грн на місяць (з точністю до копійок). Програма повинна вибрати у окремі таблиці на новому аркуші співробітників, які отримують заробітну платню до 15 тис. грн включно, до 20 тис. грн. включно та інші. До таблиць необхідно додати заголовки, межі та виконати їх форматування для оптимального розміщення даних.

7. Виконайте запит до RANDOM USER GENERATOR API: https://randomuser.me/api/?results=200&format=csv
для отримання даних 200 псевдо користувачів у файлі формату CSV. Розробіть програму, яка виконує завантаження даних користувачів до робочої книги з CSV файлу . Програма повинна виконато багаторівневе сортування даних: за країною,

за містом та за прізвищем користувача. Результуючу таблицю необхідно відформатувати для оптимального відображення даних, виділити заголовок напівжирним шрифтом та додати межі.

8. Розробіть програму, яка буде виконувати парсинг даних зі сторінки сайту https://hotline.ua/ua/computer/noutbuki-netbuki
та занесе до робочої книги у рядки стовпців A, B та C модель ноутбуку, його специфікацію та ціну, відповідно. Програма повинна додати заголовок до таблиці, перетворити рядкове значеня ціни на числове та виконати сортування за зменшеням ціни. Результуючу таблицю необхідно відформатувати для оптимального відображення даних, виділити заголовок напівжирним шрифтом та додати межі (див. нижче).

9. Розробити програму, що дозволяє конвертувати всі файли формату *.xlsx у файли формату *.xls а всі файли формату *.xls у файли формату *.xlsx з обраного за допомогою діалогового вікна обрання каталогу.

10. Розробити програму, що створює список студентів, ідентифікованих у стовпці A, як Студент-N, де N=1-20 та виконує копіювання до 20 абзаців документу Microsoft Word їх порядкового номеру та текстового ідентифікатора.

ЗАМІСТЬ ЕПІЛОГУ

Завершуючи викладення матеріалу цього посібника автор хотів би звернути увагу читача на три моменти:

1. Суттєвою перевагою офісного програмування є можливість написання єдиних програм, які будуть працювати з даними різних застосунків. Ми вже наводили приклад зчитування даних до робочих книг Microsoft Excel з бази даних Microsoft Access (див. вище). Зауважимо, що програми, які працюють з даними різних застосунків можна розташовувати у файлі будь-якого застосунку, що підтримує збереження макросів. Також можливе розширення функціональних можливостей Microsoft Excel засобами бібліотек, які постачаються разом з пакетом Microsoft Office та операційною системою Windows (прив'язку до однієї операційної системи можна вважати недоліком програмування на VBA). Наприклад, при підключенні до проєкту бібліотек Microsoft HTML Object Library та Microsoft Internet Controls командою меню Редактора Visual Basic *Tools–References* (Рисунок 116), до функціоналу програмування застосунків Microsoft Office додається можливість працювати з Веб-ресурсами за допомогою браузеру Internet Explorer.

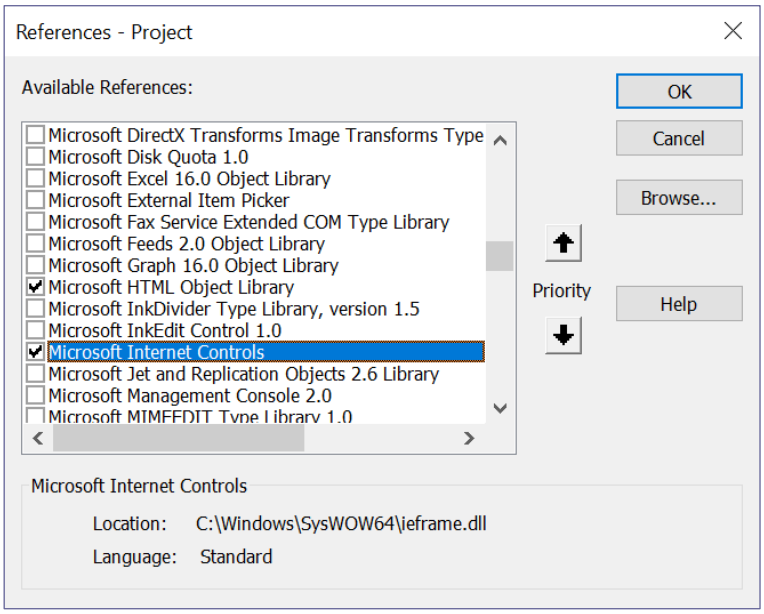


Рисунок 116 – Додання до проєкту бібліотек, що забезпечують роботу з Веб-ресурсами

У якості прикладу програми, що працює з декількома застосунками пакету Microsoft Office, наведемо програму, яка виконує зчитування новин з головної Веб-сторінки Запорізького національного університету до робочої книги Microsoft Excel, виконує там їх обробку (сортування за жатою та часом публікації), а потім публікує як список у документі Microsoft Word, зберігатися програма буде у документі Microsoft Word із підтримкою роботи макросів. Попередньо необхідно вивчити структуру HTML сторінки для новин, це виконується, наприклад, у браузері Chrome кліком по необхідному елементу Веб-сторінки (новині у даному випадку) та обранням з контекстного меню

команди *Перевірити*. Відкриється вікно *Chrome Developer Tools* з HTML-кодом сторінки на вкладці *Elements* з розгорненою структурою коду для обраного Веб-елемента (Рисунок 117).

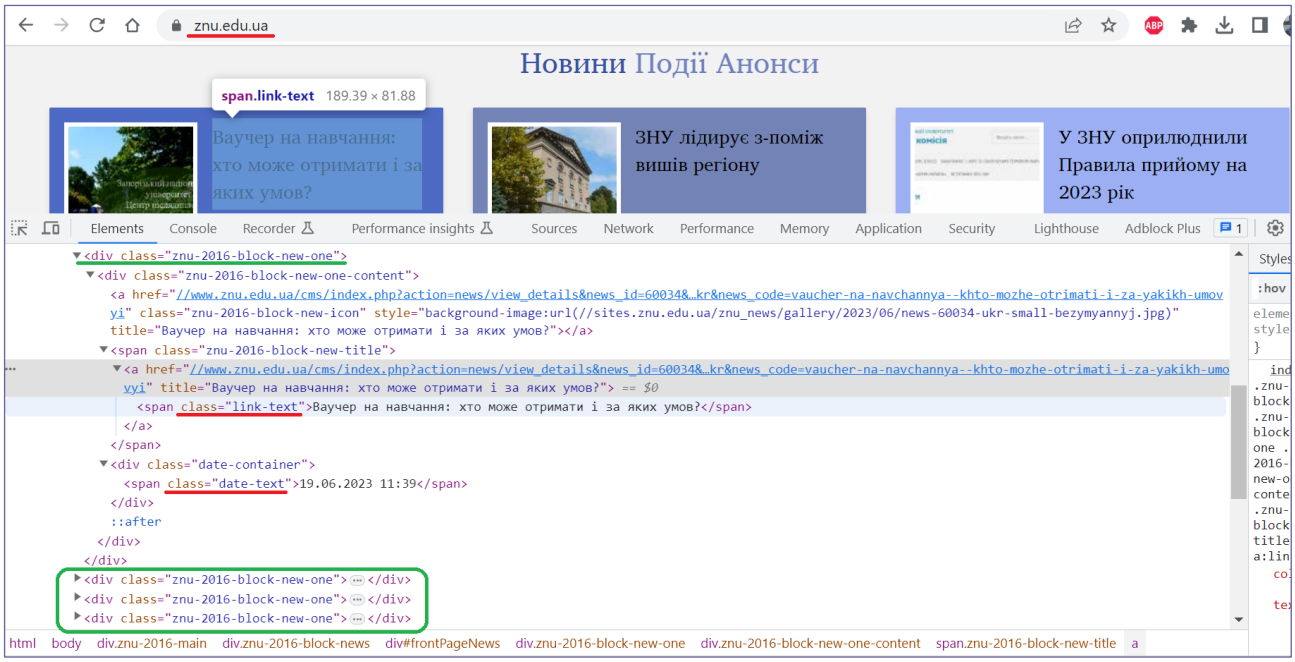


Рисунок 117 – HTML-код сторінки, з якої будуть зчитуватися дані

Видно, що текст новин є вмістом тегу `<span>` з CSS-класом `"link-text"`, а дата та час публікації новин – вмістом тегу `<span>` з CSS-класом `"date-text"`.

Архітектура програми буде наступною (зауважимо, що поділ функціональності програми так, щоб кожна її функція виконувалась у окремій процедурі, відповідає концепції "чистого коду" [11]):

1) головна підпрограма `MainProgram()` буде запускати екземпляр Microsoft Excel (або отримувати посилання на нього, якщо він запущений) та викликати функцію `HTMLToExcelImport(url As String) As Integer`, що виконує зчитування даних зазначеної аргументом Веб сторінки та повертає кількість зчитаних новин, а потім – підпрограму `ExcelToWordImport(num As Integer)`, що виконує імпорт Microsoft Word зазначеної аргументом кількості зчитаних новин.

2) функція `HTMLToExcelImport(url As String) As Integer` створює об'єкт класу `InternetExplorer` та відкриває у ньому методом `Navigate()` сторінку https://www.znu.edu.ua, після цього методом `Document()` отримує вміст сторінки як об'єкт `HTMLDocument`. Далі методом `getElementsByClassName(v)`, де `v` – назва класу CSS отримується колекція об'єктів `IHTMLElement` для тексту новин та дати та часу їх опублікування. Після цього створюється нова робоча книга і елементи отриманих колекцій вставляються до рядків стовпців A та B робочого аркушу "Аркуш1". Далі

виконується оптимізації ширини стовпців з даними та їх сортування за датами публікації у зворотному порядку та збереження робочої книги у файлі.

3) підпрограма `ExcelToWordImport(num As Integer)` відкриває файл робочої книги з новинами, виконує їх додання як параграфи документу Microsoft Word та зберігає документ у файлі.

Наведемо код описаних процедур, зауважимо, що оскільки об'єкт-застосунок Microsoft Excel `xlApp` використовується у всіх процедурах, він оголошений на рівні модуля:

```
Option Explicit

Dim xlApp As Object

'Програма імпорту даних з Веб сторінки до робочої книги Excel, їх
'обробки та імпорту з робочої книги до документу Word
Sub MainProgram()
    Dim xlWbk As Object
    Dim num As Integer

    On Error Resume Next
    Set xlApp = GetObject(, "Excel.Application")
    If Err Then
        Set xlApp = CreateObject("Excel.Application")
    End If
    On Error GoTo 0
    xlApp.Visible = True

    num = HTMLToExcelImport("https://www.znu.edu.ua")
    ExcelToWordImport (num)

    xlApp.Quit
End Sub

'Завантаження даних Веб-сторінки до робочої книги
Function HTMLToExcelImport(url As String) As Integer
    Dim browser As New InternetExplorer
    Dim page As New HTMLDocument
    Dim news As Object
    Dim newsDates As Object
    Dim num As Integer
    Dim myWbk As Object

    browser.Visible = True
    browser.Navigate url

    Do While browser.Busy: Loop  'Очікування завантаження сторінки

    Set page = browser.Document
    Set news = page.getElementsByClassName("link-text")
    Set newsDates = page.getElementsByClassName("date-text")

    Set myWbk = xlApp.Workbooks.Add
```

```
    With myWbk.Worksheets("Аркуш1")
        For num = 1 To news.Length
            .Cells(num, 1).Value = news.Item(num - 1).innerText
            .Cells(num, 2).Value = _
                        CDate(newsDates.Item(num - 1).innerText)
        Next num
        With .Range(.Cells(1, 1), .Cells(2, num)).CurrentRegion
            .Columns.AutoFit
            .Sort Key1:=.Columns("B"), Order1:=2
        End With
    End With


    browser.Quit
    myWbk.SaveAs ThisDocument.Path & Application.PathSeparator _
              & "Новини.xlsx"
    myWbk.Close
    HTMLToExcelImport = num
End Function

'Завантаження даних робочої книги до документу Word
Sub ExcelToWordImport(num As Integer)
    Dim myWbk As Object
    Dim myDoc As Document
    Dim i As Integer
    Dim myRange As Range

    Set myWbk = xlApp.Workbooks.Open(ThisDocument.Path _
                    & Application.PathSeparator & "Новини.xlsx")

    Set myDoc = Documents.Add
    For i = 1 To num
         Set myRange = myDoc.Paragraphs(i).Range
         myRange.InsertAfter _
                 myWbk.Worksheets("Аркуш1").Cells(i, 1) & " - " _
               & myWbk.Worksheets("Аркуш1").Cells(i, 2)
         myRange.InsertParagraphAfter
    Next i

    myDoc.SaveAs2 ThisDocument.Path & Application.PathSeparator _
                & "Новини.docx"
    myDoc.Close
    myWbk.Close
End Sub
```

Зауважимо, що окрім методу `getElementsByClassName(v)`, для отримання даних Веб-сторінки можуть бути корисними методи [36]:

`getElementsByTagName`

`getElementById`

`getAttribute`

```
innerText
innerHTML
```

Також можливе отримання даних за вкладеними локаторами різного типу для необхідних Веб-елементів, наприклад [36], для елементу, вказаного на Рисунок 118, відповідний фрагмент коду буде:

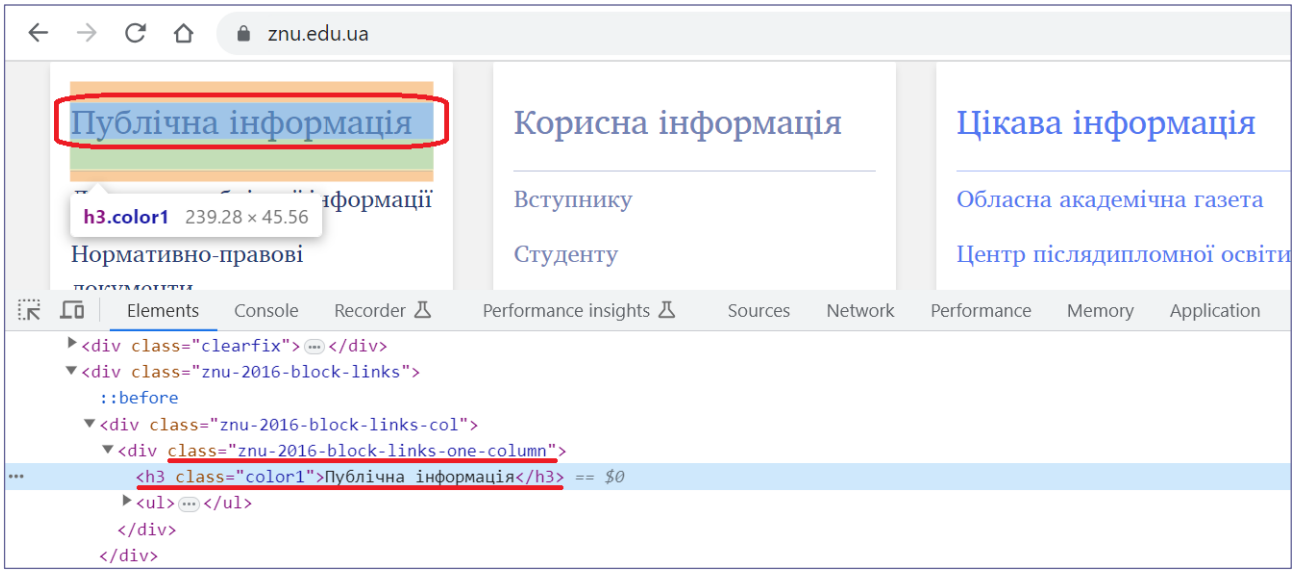


Рисунок 118 – HTML-код сторінки вкладеного елемента дані якого будуть зчитуватись

```
Dim columnData As Object

...
Set columnData = _
 page.getElementsByClassName("znu-2016-block-links-one-column")(0)
    Debug.Print columnData.getElementsByTagName("H3")(0).innerText
...
```

2. Сумісне використання макрорекордеру для запису основи користувацької програми з подальшим її рефакторингом (зміною з метою оптимізації) у Редакторі Visual Basic дозволяє спростити програмування та збільшити його ефективність. Наведемо для прикладу повний код макросу списку присутніх, записаного у першому розділі першого модуля (Рисунок 3б):

```
Sub AttendanceFormat()
'
' AttendanceFormat Макрос
' Форматування списку присутніх
'
' Сполучення клавіш: Ctrl+j
'
    Cells.Select
    Cells.EntireColumn.AutoFit
    Range("A1:C11").Select
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone
    With Selection.Borders(xlEdgeLeft)
```

```
        .LineStyle = xlContinuous
        .ColorIndex = 0
        .TintAndShade = 0
        .Weight = xlThin
    End With
    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlContinuous
        .ColorIndex = 0
        .TintAndShade = 0
        .Weight = xlThin
    End With
    With Selection.Borders(xlEdgeBottom)
        .LineStyle = xlContinuous
        .ColorIndex = 0
        .TintAndShade = 0
        .Weight = xlThin
    End With
    With Selection.Borders(xlEdgeRight)
        .LineStyle = xlContinuous
        .ColorIndex = 0
        .TintAndShade = 0
        .Weight = xlThin
    End With
    With Selection.Borders(xlInsideVertical)
        .LineStyle = xlContinuous
        .ColorIndex = 0
        .TintAndShade = 0
        .Weight = xlThin
    End With
    With Selection.Borders(xlInsideHorizontal)
        .LineStyle = xlContinuous
        .ColorIndex = 0
        .TintAndShade = 0
        .Weight = xlThin
    End With
    Rows("1:1").Select
    With Selection
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlBottom
        .WrapText = False
        .Orientation = 0
        .AddIndent = False
        .IndentLevel = 0
        .ShrinkToFit = False
        .ReadingOrder = xlContext
        .MergeCells = False
    End With
    Selection.Font.Bold = True
    Sheets("meetingAttendanceList (21)").Select
    Sheets("meetingAttendanceList (21)").Name = "Офісне
програмування"
End Sub
```

Код цього макросу може бути оптимізований. По-перше, можна винести повторюваний код форматування меж у окрему підпрограму і її викликати для різних меж, по-друге, можна видалити встановлення властивостей у значення за замовчуванням і по-третє, можна додати код автоматичного визначення заповненого діапазону та замінити звернення до робочого аркушу за ім'ям на звернення за індексом, щоб зробити цю програму універсальною. Після виконання описаного рефакторингу код буде таким:

```
Sub AttendanceFormatOptimized()
'
' AttendanceFormat ОПТИМІЗОВАНИЙ Макрос
' Форматування списку присутніх
'
' Сполучення клавіш: Ctrl+j
'
    Cells.Select
    Cells.EntireColumn.AutoFit
    Range("A1").CurrentRegion.Select
    BorderFormat xlEdgeLeft
    BorderFormat xlEdgeTop
    BorderFormat xlEdgeBottom
    BorderFormat xlEdgeBottom
    BorderFormat xlEdgeRight
    BorderFormat xlInsideVertical
    BorderFormat xlInsideHorizontal

    Rows("1:1").Select
    With Selection
        .HorizontalAlignment = xlCenter
        .Font.Bold = True
    End With
    Sheets(1).Select
    Sheets(1).Name = "Офісне програмування"
End Sub

Sub BorderFormat(border As XlBordersIndex)
    With Selection.Borders(border)
        .LineStyle = xlContinuous
        .Weight = xlThin
    End With
End Sub
```

3. Цей посібник обмежений вивченням програмування тільки для застосунків Microsoft Word та Microsoft Excel, оскільки освітньою програмою передбачений відносно невеликий обсяг часу на відповідну дисципліну (4 кредити), яка викладається на першому курсі бакалаврату та, окрім безпосереднього програмування для офісних застосунків, має додаткову мету ознайомлення студентів з процедурним програмуванням, основами об'єктно-орієнтованого програмування, основами програмування людино-машинної взаємодії тощо. У той же час існують інші застосунки, для яких доцільне вивчення програмування на VBA. Насамперед, це Microsoft Access, написання

програм на VBA для баз даних якого дозволяє створювати професійні застосунки, що вирішують різні завдання обліку, планування, прогнозування тощо. Вивчення засобів цього застосунку варте окремої навчальної дисципліни і пов'язане з наданням студентам знань щодо організації та функціонування реляційних баз даних та об'єктних моделей доступу до баз даних. Для бажаючих вивчити засоби програмування для Microsoft Access можна порекомендувати книгу [9]. Автору також доводилось програмувати завдання автоматизації для презентацій Microsoft PowerPoint, таких як форматування фрагментів коду, анімація рисунків, зазначення мови для перевірки правопису тощо. Для бажаючих вивчити засоби програмування для Microsoft PowerPoint можна порекомендувати книгу [10].

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна:

1. Меркулов В.С., Бізюк І.Г., Чаленко О.В. Програмування в середовищі Visual Basic : Конспект лекцій. Харків : УкрДАЗТ, 2019. 78 с.
2. Mansfield R. Mastering VBA for Microsoft Office 365, 2019 Edition. Sybex, 2019. 944 p.
3. Mueller J.P. VBA For Dummies, 5th Edition. Wiley Publishing, Inc., 2007. 428 p.

Додаткова:

1. Квітка О.О., Шахновський А.М., Мердух С.Л. Комп'ютерні технології та програмування 1. Основи алгоритмізації : метод. вказівки до викон. лаб. і практ. робіт та самост. роботи для студ. напряму підготовки 6.050202 «Автоматизація та комп’ютерно-інтегровані технології» (Навч. електронне видання). Київ : НТУУ «КПІ», 2014. 94 с.
2. Korol J. Excel 2021/Microsoft 365 Programming by Example. Mercury Learning and Information, 2023. 1072 p.
3. Korol J. Access 2021/Microsoft 365 Programming by Example. Mercury Learning and Information LLC, 2022. 1171 p.
4. Sanchez E.N. Programming PowerPoint with VBA Straight to the Point. Holy Macro! Books, 2022. 65 p.

Інформаційні джерела:

1. Visual Basic Naming Conventions. URL: https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/program-structure/naming-conventions (дата звернення 17.04.2023).
2. Data type summary. URL: https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/data-type-summary (дата звернення 17.04.2023).
3. Microsoft Word Object Model. URL: https://learn.microsoft.com/en-us/previous-versions/office/developer/office-xp/aa165321(v=office.10) (дата звернення 30.04.2023).
4. Microsoft Excel Object Model. URL: https://learn.microsoft.com/en-us/previous-versions/office/developer/office-xp/aa141044(v=office.10)?redirectedfrom=MSDN) (дата звернення 30.05.2023).
5. Parse HTML in Excel VBA – Learn by parsing hacker news home page (Updated 2022). URL: https://codingislove.com/parse-html-in-excel-vba (дата звернення 30.05.2023).

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Mansfield R. Mastering VBA for Microsoft Office 365, 2019 Edition. Sybex, 2019. 944 p.
2. Квітка О.О., Шахновський А.М., Мердух С.Л. Комп'ютерні технології та програмування 1. Основи алгоритмізації : метод. вказівки до викон. лаб. і практ. робіт та самост. роботи для студ. напряму підготовки 6.050202

«Автоматизація та комп'ютерно-інтегровані технології» (Навч. електронне видання). Київ : НТУУ «КПІ», 2014. 94 с.
3. Mueller J.P. VBA For Dummies, 5th Edition. Wiley Publishing, Inc., 2007. 428 p.
4. Wyatt A. L. Word VBA Guidebook. Sharon Parq Associates, Inc., 2011. 247 p.
5. Korol J. Excel 2021/Microsoft 365 Programming by Example. Mercury Learning and Information, 2023. 1072 p.
6. Коломоєць Г.П. Програмне додання елементів керування та налаштування обробників подій користувацьких форм Microsoft Office. Геостратегічні трансформації та траєкторія національної безпеки в контексті відбудови і сталого розвитку України : матеріали міжнар. наук.-практ. конф., м. Запоріжжя, 25-26 трав. 2023 р. Запоріжжя, 2023. С.137-139.
7. Коломоєць Г.П. Автоматизація перевірки додатків до дипломів про вищу освіту *Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки*, Київ, 2021, Т. 32 (71) № 1, Ч. 1. С. 85-92.
8. Коломоєць Г.П., Кісельов Є.М., Таранець А.В. Інформаційна система обліку навчального навантаження з урахуванням малочисельних груп. *Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки*, Київ, 2019, Т. 30 (69) №2, Ч. 1. С. 118-128.
9. Korol J. Access 2021/Microsoft 365 Programming by Example. Mercury Learning and Information LLC, 2022. 1171 p.
10. Sanchez E.N. Programming PowerPoint with VBA Straight to the Point. Holy Macro! Books, 2022. 65 p.
11. Мартін Р. Чистий код: створення і рефакторинг за допомогою Agile. Харків : Фабула, 2019, 416 с.
12. MS Access VBA code editor character encoding and copy/paste. *StackOverflow site*. URL: https://stackoverflow.com/questions/22560121/ms-access-vba-code-editor-character-encoding-and-copy-paste (дата звернення 17.04.2023).
13. Visual Basic Naming Conventions. URL: https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/program-structure/naming-conventions (дата звернення 17.04.2023).
14. Naming Convention. URL: https://bettersolutions.com/vba/variables/naming-convention.htm (дата звернення 17.04.2023).
15. Data type summary. URL: https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/data-type-summary (дата звернення 17.04.2023).
16. Static statement. URL: https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/static-statement (дата звернення 17.04.2023).
17. Passing Arguments by Value and by Reference (Visual Basic). URL: https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/procedures/passing-arguments-by-value-and-by-reference (дата звернення 17.04.2023).
18. Шифр Цезаря. URL: https://uk.wikipedia.org/wiki/Шифр_Цезаря (дата звернення 17.04.2023).

19. Скомпільований файл довідки в форматі HTML. URL: https://uk.wikipedia.org/wiki/Скомпільований_файл_довідки_в_форматі_HTML (дата звернення 17.04.2023).
20. Trappable errors. URL: https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/trappable-errors (дата звернення 30.04.2023).
21. Clear method. URL: https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/clear-method-visual-basic-for-applications (дата звернення 30.04.2023).
22. VBA Error Handling with Resume Next, GoTo 0. The VBA Tutorials Blog. URL: https://wellsr.com/vba/2021/excel/vba-error-handling-resume-next-goto-0 (дата звернення 30.04.2023).
23. Scroll method (Microsoft Forms). URL: https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/scroll-method (дата звернення 30.04.2023).
24. Add method (Microsoft Forms). URL: https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/add-method-microsoft-forms (дата звернення 30.04.2023).
25. Microsoft Word Object Model. URL: https://learn.microsoft.com/en-us/previous-versions/office/developer/office-xp/aa165321(v=office.10) (дата звернення 30.04.2023).
26. WdLanguageID enumeration (Word). URL: https://learn.microsoft.com/en-us/office/vba/api/word.wdlanguageid (дата звернення 20.05.2023).
27. WdTableFormat enumeration (Word). URL: https://learn.microsoft.com/en-us/office/vba/api/word.wdtableformat (дата звернення 20.05.2023).
28. Kolomoets G. Strange behavior of the first column when adding by VBA multiple formulas to the cells of the MS Word table. *StackOverflow site*. URL: https://stackoverflow.com/questions/76341483/strange-behavior-of-the-first-column-when-adding-by-vba-multiple-formulas-to-the/76341755#76341755 (дата звернення 22.06.2023).
29. MsoAutoShapeType enumeration (Office). URL: https://learn.microsoft.com/en-us/office/vba/api/office.msoautoshapetype (дата звернення 20.05.2023).
30. Microsoft Excel Object Model. URL: https://learn.microsoft.com/en-us/previous-versions/office/developer/office-xp/aa141044(v=office.10)?redirectedfrom=MSDN) (дата звернення 30.05.2023).
31. Sample XML File (books.xml). URL: https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ms762271(v=vs.85) (дата звернення 30.05.2023).
32. Get the Northwind sample database for Microsoft Access. URL: https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/downloading-sample-databases#northwind_access (дата звернення 30.05.2023).
33. Number format codes. URL: https://support.microsoft.com/en-us/office/number-format-codes-5026bbd6-04bc-48cd-bf33-80f18b4eae68?ui=en-us&rs=en-us&ad=us (дата звернення 30.05.2023).

34. Border object (Excel). URL: https://learn.microsoft.com/en-us/office/vba/api/excel.border(object) (дата звернення 30.05.2023).
35. XlThemeColor enumeration (Excel). URL: https://learn.microsoft.com/en-us/office/vba/api/excel.xlthemecolor (дата звернення 30.05.2023).
36. Parse HTML in Excel VBA – Learn by parsing hacker news home page (Updated 2022). URL: https://codingislove.com/parse-html-in-excel-vba (дата звернення 30.05.2023).

Навчальне видання

(*українською мовою*)

Коломоєць Геннадій Павлович

ОФІСНЕ ПРОГРАМУВННЯ МОВОЮ VBA

Навчальний посібник
для здобувачів ступеня вищої освіти бакалавра
спеціальності 121 «Інженерія програмного забезпечення»
освітньо-професійної програми
«Програмне забезпечення систем»

Рецензент *С.Л. Хрипко*

Відповідальний за випуск *Т.В. Критська*

Коректор *А.І. Безверхий*