**Звіт про виконання практичних завдань до лекцій з курсу Технології програмування на мові Python**

Звіт до Теми №1

Функції та змінні

Під час виконання практичного завдання до Теми №1 було надано варіанти рішення до наступних задач:

**Перетворення рядка**

Необхідно рядок, що має вигляд "abcdefg123" перетворити наступним чином "321gfedcba", вважаючи сталою довжину рядку в 10 символів.

***Хід виконання завдання:***

**Input\_string = “abcdefg123”**: тут ми задаємо змінну **input\_string**, в якій зберігається вихідна строка **"abcdefg123"**. Передбачається, що довжина рядка завжди буде 10 символів, як зазначено в умові.

If len(input\_string) == 10: : ця строка перевіряє, чи має input\_string довжину строки 10 символів. Ми використовуємо функцію len(), яка повертає кількість символів у рядку. Якщо довжина рівна 10, програма продовжує виконування, якщо менше, або більше — видає помилку.

**letters = input\_string[:7]**

**numbers = input\_string[7:]**

Тут ми розділяємо строку на дві частини:

**letters**— це перші 7 символів строки, отримані зрізом **input\_string[:7]**. Зріз **[:7]**вказує, що ми беремо символи з початку рядка (індекс 0) до індексу 7 (не включаючи його).

**numbers**— це останні 3 символи рядка, які ми отримуємо за допомогою зрізу **input\_string[7:]**. Тут **[7:]**означає, що беруться всі символи, починаючи з індексу 7 і до кінця строки.

**result\_string = numbers[::-1] + letters[::-1]**

Тут ми створюємо новий рядок result\_string, перевертаючи lettersі numbersоб'єднуючи їх у новому порядку:

**numbers[::-1]**— перевертає строку numbers. [::-1]вказує на зріз, який іде з кінця до початку, таким чином інвертуючи строку.

**letters[::-1]**— аналогічно перевертає строку letters.

Потім ми об'єднуємо перевернуті частини, спочатку вставляючи перевернуті числа, а потім перевернуті букви.

**print("Result:", result\_string)**

Ця строка виводить результат на екран, показуючи перетворену строку.

**else:**

**print("The line must contain exactly 10 chracters")**

Цей блок else виконується, якщо довжина рядка input\_stringне дорівнює 10 символам. В цьому випадку програма виводить повідомлення про помилку і завершує виконання.

|  |
| --- |
| input\_string = "abcdefg123"  if len(input\_string) == 10:      letters = input\_string[:7]      numbers = input\_string[7:]      result\_string = numbers[::-1] + letters[::-1]      print("Result", result\_string)  else:      print("The line must contain exactly 10 characters") |

Текст програми:

**Тестування функцій**

Як працюють дані функції:

**strip() -** видаляє пробіли на початку і в кінці рядка.

**capitalize() -** робить першу букву рядка з великої літери, а інші з маленької.

**title() -** робить великими перші букви всіх слів у рядку.

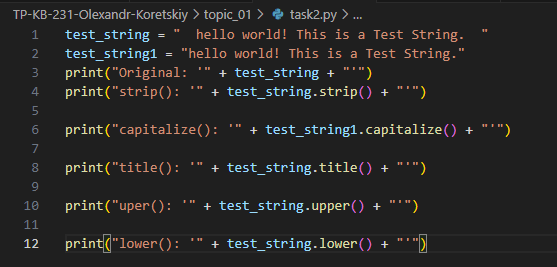
**upper() –** усі букви в рядку робить з великої літери.

**lower() –** усі букви в рядку робить з маленької літери.

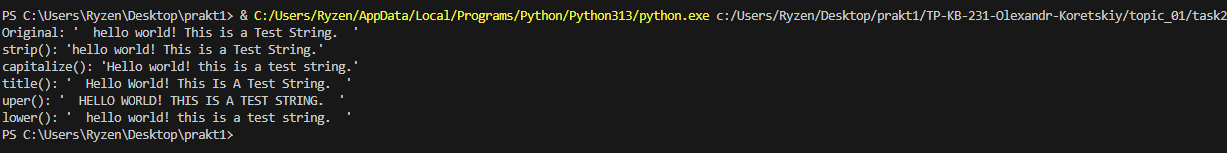
Як протестувати надані функції: **strip(), capitalize(), title(), upper(), lower().**

Створюємо код, у якому ми протестуємо кожну функцію.

**Код**



**Результат**

****

**Пояснення:**

**test\_string = " hello world! This is a Test String. " -** Ми задаємо рядок test\_string з пробілами на початку та в кінці та з буквами в різному реєстрі, щоб перевірити поведінку кожної функції, а також окремо для функції **capitalize()** зробив ще один рядок test\_string:

**test\_string = " hello world! This is a Test String. " –** так як капіталайз не реагує на пробіли

**print("Оригинал: '" + test\_string + "'") –** оригінальний рядок тексту

**print("strip(): '" + test\_string.strip() + "'") –** виконнаня функції **strip()**

**print("capitalize(): '" + test\_string.capitalize() + "'") –** виконання функції **capitalize()**

**print("title(): '" + test\_string.title() + "'") -** виконання функції **title()**

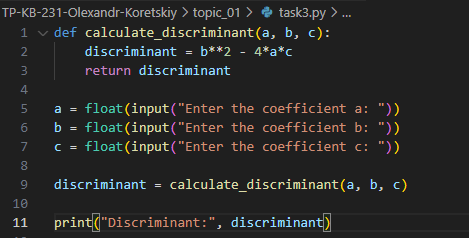
**print("upper(): '" + test\_string.upper() + "'") -** виконання функції **upper()**

**print("lower(): '" + test\_string.lower() + "'") -** виконання функції **lower()**

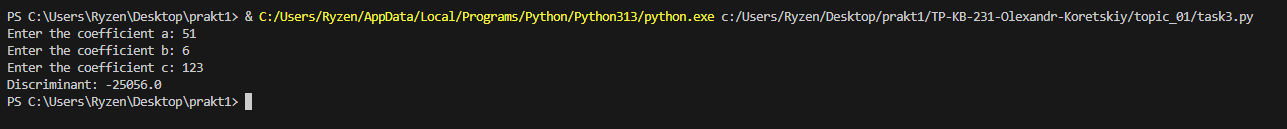
**Функція пошуку дискримінанту квадратного рівняння.**

Напишемо код який буде обчислювати дискримінант:

**Код**



**Результат**



**Пояснення:**

**Функція calculate\_discriminant(a, b, c)**:

Це функція, яка приймає три аргументи: a, b, c (коефіцієнти квадратного рівняння).

Формула для дискримінанта квадратного рівняння виглядає так: D=b2−4acD = b^2 - 4acD=b2−4ac

У коді вираз b\*\*2 - 4\*a\*c обчислює дискримінант.

Після обчислення значення дискримінанта повертається за допомогою return discriminant.

**Запит введення від користувача**:

a = float(input("Введіть коефіцієнт a: ")): Користувача просять ввести значення коефіцієнта a. Введене значення перетворюється на число з плаваючою комою за допомогою функції float().

Аналогічно запитуються і перетворюються на числа коефіцієнти b та c.

**Обчислення дискримінанта**:

Викликається функція calculate\_discriminant(a, b, c), у яку передаються введені користувачем значення a, b, c.

Функція обчислює дискримінант і повертає результат.

**Виведення результату**:

print("Дискримінант:", discriminant) виводить значення дискримінанта, яке було обчислене функцією.

Силка на гітхаб: <https://github.com/tmpsty/TP-KB-231-Olexandr-Koretskiy>

Скріншот силки

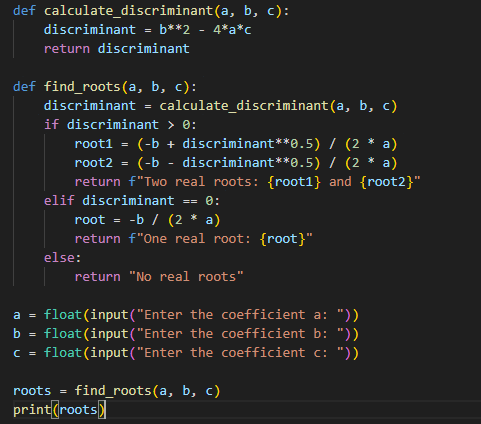
Звіт до Теми №2

Умовний перехід

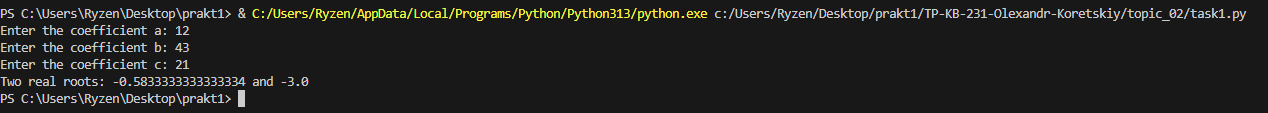
**Написати функцію пошуку коренів квадратного рівняння використовуючи функцію розрахунку дискримінанту з попередньої теми та умовні переходи.**

Напишемо код використовуючи функцію розразхунку з попередньої теми:

**Код**



**Результат**

****

**Пояснення:**

**def calculate\_discriminant(a, b, c):**

Оголошується функція **calculate\_discriminant**, яка приймає три параметри: коефіцієнти a, b, і c квадратного рівняння.

**discriminant = b\*\*2 - 4\*a\*c**  
Обчислюється дискримінант

**return discriminant**  
Значення дискримінанта повертається з функції.

**def find\_roots(a, b, c):**  
Оголошується функція **find\_roots**, яка також приймає коефіцієнти **a, b, і c.**

**discriminant = calculate\_discriminant(a, b, c)**  
Викликається функція **calculate\_discriminant**, щоб знайти дискримінант D.

**if discriminant > 0:**  
Перевіряється, чи D>0. Якщо так, квадратне рівняння має два різних дійсних кореня.

**root1 = (-b + discriminant\*\*0.5) / (2 \* a)**  
Обчислюється перший корінь

**root2 = (-b - discriminant\*\*0.5) / (2 \* a)**  
Обчислюється другий корінь

**return f"Two real roots: {root1} and {root2}"**  
Повертається рядок з двома дійсними коренями.

**elif discriminant == 0:**  
Перевіряється, чи D=0. Якщо так, рівняння має один дійсний корінь.

**root = -b / (2 \* a)**  
Обчислюється єдиний корінь

**return f"One real root: {root}"**  
Повертається рядок з єдиним дійсним коренем.

**else:**  
Якщо D<0, рівняння не має дійсних коренів.

**return "No real roots"**  
Повертається повідомлення, що дійсних коренів немає.

**a = float(input(...))**  
Користувач вводить коефіцієнт a

**b = float(input(...))**  
Користувач вводить коефіцієнт b.

**c = float(input(...))**  
Користувач вводить коефіцієнт c.

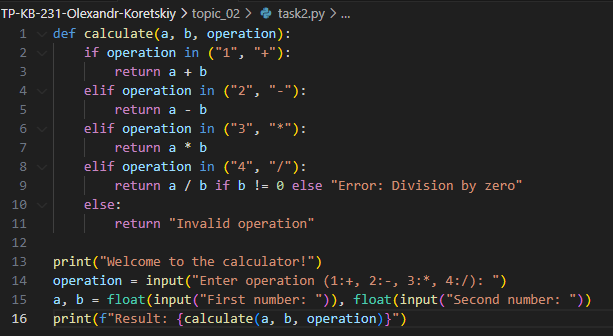
**roots = find\_roots(a, b, c)**  
Викликається функція **find\_roots**, і результат зберігається у змінній **roots**.

**print(roots)**  
Виводить результат (корені рівняння або повідомлення про їх відсутність).

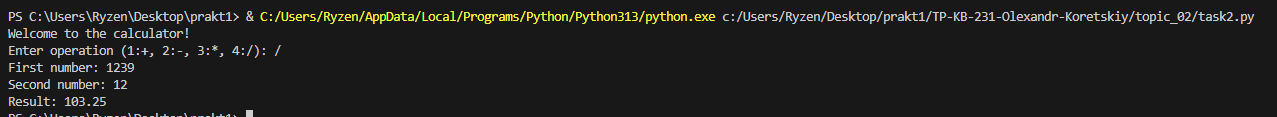
**Написати програму калькулятор використовуючи *if else* конструкцію. Кожна операція має бути виконана в окремій функції.**

Напишемо код використовуючи **if else** конструкцію:

**Код**



**Результат**

****

**Хід роботи програми:**

**1.Привітання користувача:**

Програма виводить повідомлення "Welcome to the calculator!" (Вітаємо у калькуляторі!).

**2.Вибір операції:**

Користувачу пропонується вибрати математичну операцію. Можливі варіанти:

[+] для додавання.

[-] для віднімання.

[\*] для множення.

[/] для ділення.

**3.Введення чисел:**

Користувач вводить два числа

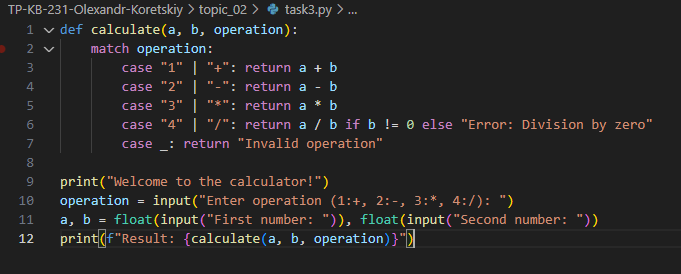
4.**Перевірка на коректність операції:**

Якщо користувач ввів неправильну операцію, програма виводить повідомлення "Invalid operation" (Невірна операція).

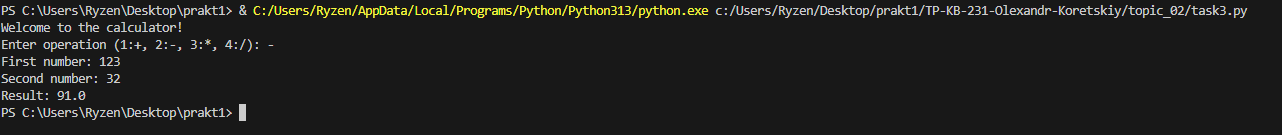
**5.Виведення результату:**

Програма відображає результат обчислень або повідомлення про помилку.

**Код**

****

**Результат**

****

**Хід роботи програми:**

**1.Привітання користувача:**

Програма виводить повідомлення "Welcome to the calculator!" (Вітаємо у калькуляторі!).

**2.Вибір операції:**

Користувач вводить номер операції або її символ:

[+] для додавання.

[-] для віднімання.

[\*] для множення.

[/] для ділення.

**3.Введення чисел:**

Користувач вводить два числа

4.**Перевірка на коректність операції:**

Якщо жоден із вказаних варіантів не збігається, програма повертає повідомлення "Invalid operation" (Невірна операція).

**5.Виведення результату:**

Програма виводить результат обчислень або повідомлення про помилку залежно від введених даних.

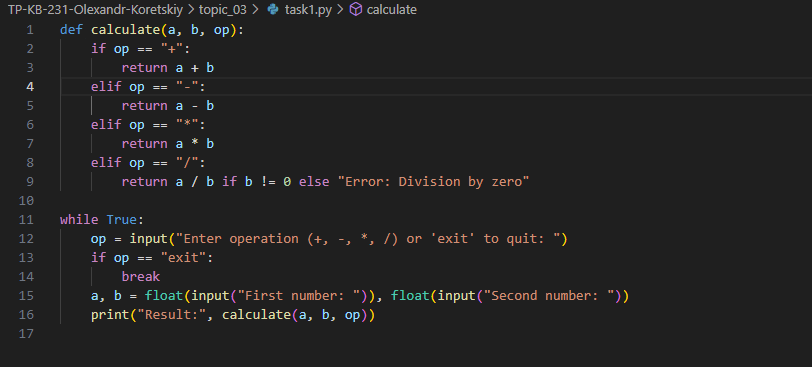
Звіт до Теми №3

Цикли

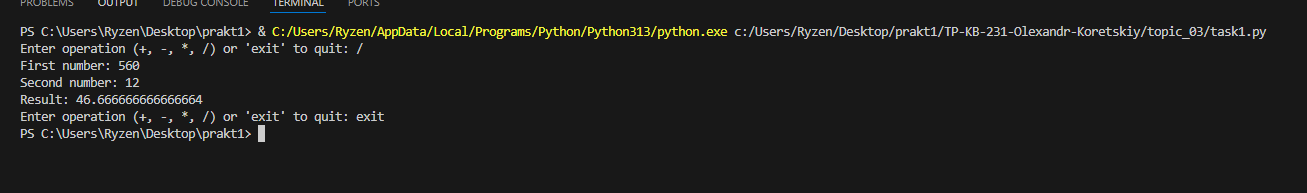
**Написати програму калькулятор з постійними запитами на введення нових даних та операцій. За основу взяти програму калькулятор з попередньої теми. Реалізувати механізм завершення програми після отримання відповідної команди.**

Напишемо код який буде виконувати дані запити

**Код**



**Результат**



**Пояснення:**

**1.Функція calculate:**

Приймає два числа **(a і b)** та операцію **(op)**.

Перевіряє, яка операція вказана, і виконує відповідне обчислення.

Якщо операція — ділення, перевіряє, чи знаменник не дорівнює нулю.

**2.Цикл while True:**

Постійно працює, доки користувач не введе **exit**.

Запитує операцію, яку хоче виконати користувач.

**3.Перевірка на вихід:**

Якщо користувач вводить **exit**, цикл переривається **(break)**, і програма завершується.

**4.Обчислення та вивід результату:**

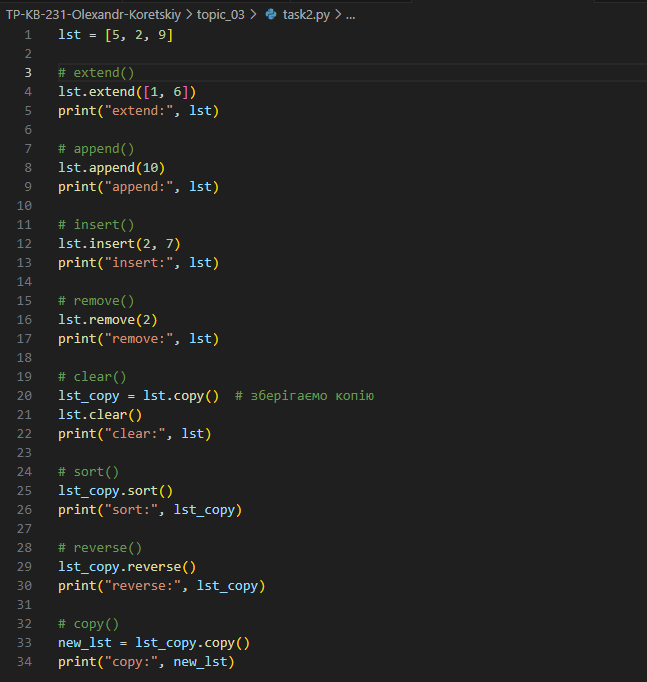
Користувач вводить два числа.

Результат обчислень виводиться на екран.

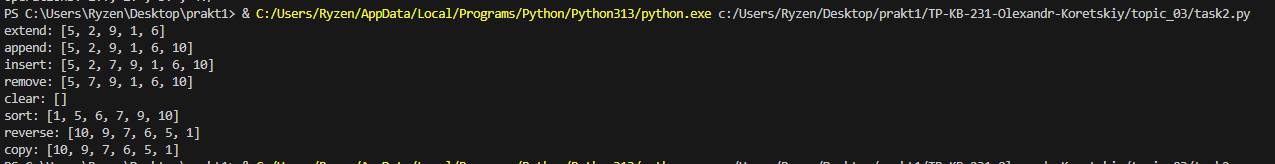
**Написати програму тестування функцій списків таких як: extend(), append(), insert(id, val), remove(val), clear(), sort(), reverse(), copy()**

Напишемо код, який протестує надані функції:

**Код**



**Результат**

****

Пояснення:

**1.Створення списку:**

Програма починає з базового списку, який використовується для тестування функцій.

**2.Додавання кількох елементів:**

Функція ***extend*** додає до списку всі елементи іншого ітерабельного об'єкта, наприклад, списку.

**3.Додавання одного елемента:**

Функція ***append*** додає один новий елемент у кінець списку.

**4.Вставка елемента на певну позицію:**

Функція ***insert*** додає елемент на задану позицію, зсуваючи інші елементи вправо.

**5.Видалення елемента:**

Функція ***remove*** видаляє перше входження зазначеного елемента зі списку.

**6.Очищення списку:**

Функція ***clear*** повністю очищає список, видаляючи всі його елементи.

**7.Копіювання списку:**

Перед очищенням створюється копія списку за допомогою ***copy***, щоб зберегти дані для подальших операцій.

**8.Сортування елементів:**

Функція ***sort*** впорядковує елементи списку у зростаючому порядку.

**9.Зворотний порядок:**

Функція ***reverse*** змінює порядок елементів у списку на протилежний.

**10.Створення нової копії списку:**

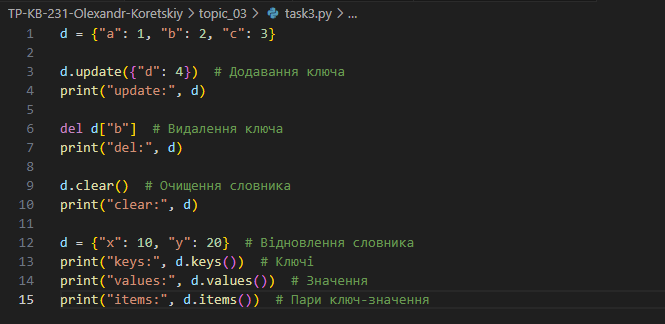
Копія списку створюється за допомогою ***copy***, щоб продемонструвати, як отримати дубль списку для окремої роботи з ним.

Усі функції викликаються послідовно, а результати після кожної операції виводяться, щоб можна було побачити, як змінюється список на кожному етапі.

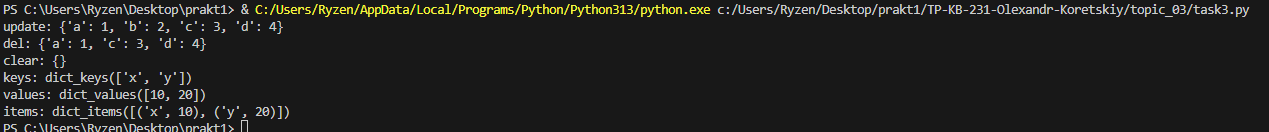
**Написати програму тестування функцій словників таких як: update(), del(), clear(), keys(), values(), items()**

Напишемо код для перевірки цих функцій:

**Код**



**Результат:**

****

Пояснення:

**1.Оновлення словника (update):**

Ця функція додає нові ключі зі значеннями до словника або оновлює значення існуючих ключів.

**2.Видалення елемента (del):**

За допомогою цієї команди можна видалити конкретний ключ зі словника разом із його значенням.

**3.Очищення словника (clear):**

Видаляє всі елементи зі словника, залишаючи його порожнім.

**4.Отримання ключів (keys):**

Повертає всі ключі словника у вигляді спеціального об’єкта, який можна використовувати, наприклад, для ітерації.

**5.Отримання значень (values):**

Повертає всі значення, що зберігаються в словнику, у вигляді спеціального об’єкта.

**6.Отримання пар ключ-значення (items):**

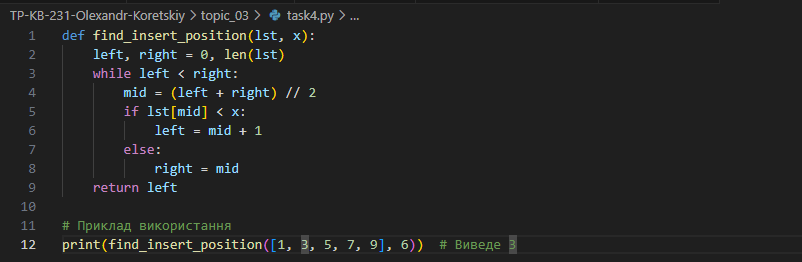
Повертає список кортежів, де кожен кортеж — це пара ключ-значення.

Програма демонструє роботу кожної функції по черзі, а результат кожної операції виводиться для перегляду змін у словнику.

**Маючи відсортований список, написати функцію пошуку позиції для вставки нового елементу в список:**

Напишемо код:

**Код**



**Результат:**

****

Пояснення:

**1.Ініціалізація меж:**

Задаємо дві межі: **left** (початок списку) та **right** (кінець списку).

**right** дорівнює довжині списку, бо новий елемент може бути більшим за всі інші.

**2.Двійковий пошук:**

Поки межі не збігаються **(left < right)**, обчислюється середина списку **(mid)**.

Порівнюємо елемент у позиції **mid** зі значенням, яке потрібно вставити:

Якщо середній елемент менший за потрібне значення, звужуємо пошук до правої половини списку **(left = mid + 1)**.

Інакше звужуємо пошук до лівої половини, включаючи **mid**.

**3.Результат:**

Коли межі збігаються **(left == right)**, змінна **left**вказує на позицію, де потрібно вставити новий елемент.

Звіт до Теми №4

Виняткові ситуації

**1.Розширити програму калькулятор функцією запитів даних для виконання операцій від користувача, що обробляє виняткові ситуації.**

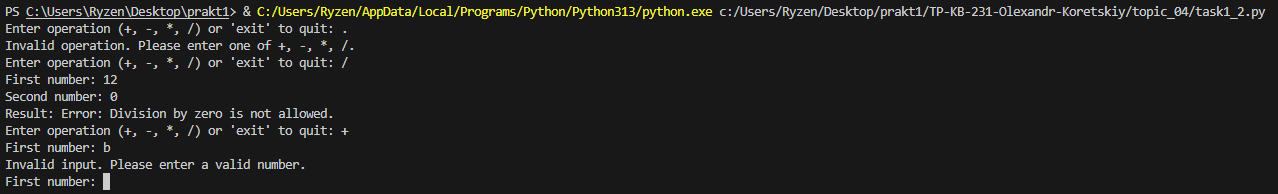
**2.Розширити функцію ділення обробкою виняткової ситуації ділення но нуль**

Модернізуємо код калькулятора з 3-ої практичнох роботи:

**Код**



**Результат**



**Що нового в коді:**

**1.Функція get\_user\_input:**

Перевіряє, чи введення користувача є числом.

Якщо введено некоректні дані, користувач отримує повідомлення про помилку і запит повторюється.

**2.Обробка винятків у calculate:**

Використовується **try-except**для обробки ділення на нуль і невірних операцій.

**3.Перевірка введеного оператора:**

Якщо введено щось, окрім дозволених операторів, програма попереджає про це і запитує повторно.

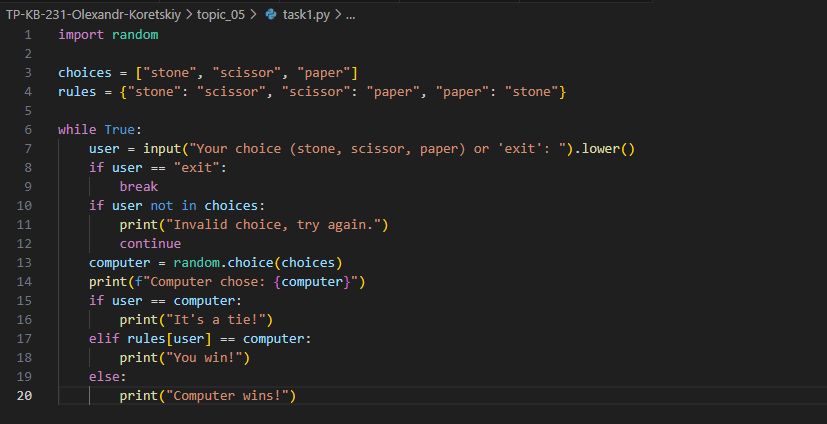
Звіт до Теми №5

Бібліотеки

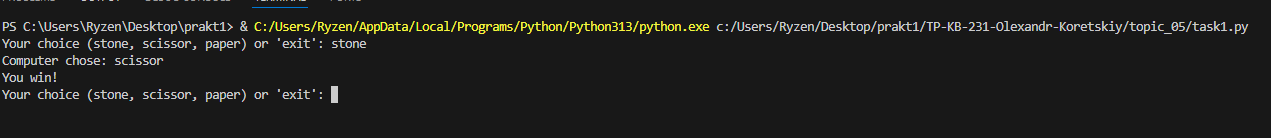
Гра з комп’ютером: камінь, ножиці, папір. Програма виконує запит від користувача на введення одного із значень ["stone", "scissor", "paper"]. Наступним кроком, використовуючи модуль random, програма у випадковому порядку вибирає одне із значень ["stone", "scissor", "paper"]. В залежності від умови, що камінь перемагає ножиці, ножиці перемагають папір, а папір перемагає камінь визначити переможця.

Напишемо код який буде здійснювати дані маніпуляції:

**Код**



**Результат**

****

Пояснення:

**Імпорт бібліотеки:**

Використовується модуль **random** для випадкового вибору ходу комп'ютера.

**Список можливих виборів:**

Містить три елементи: **stone**, **scissor**, **paper**. Це всі можливі варіанти для вибору користувача і комп'ютера.

**Словник правил:**

Визначає, який вибір перемагає інший. Наприклад, ключ stone має значення scissor, що означає: камінь перемагає ножиці.

**Основний цикл гри:**

Користувач вводить свій вибір.

Якщо користувач вводить exit, гра завершується.

Якщо вибір неправильний (не входить у список можливих варіантів), програма виводить повідомлення і пропонує повторити спробу.

**Хід комп'ютера:**

Використовуючи функцію випадкового вибору, комп'ютер обирає один із варіантів зі списку.

**Визначення переможця:**

Якщо вибір користувача і комп'ютера збігаються, це нічия.

В іншому випадку, якщо значення зі словника правил для вибору користувача відповідає вибору комп'ютера, виграє користувач.

Інакше перемагає комп'ютер.

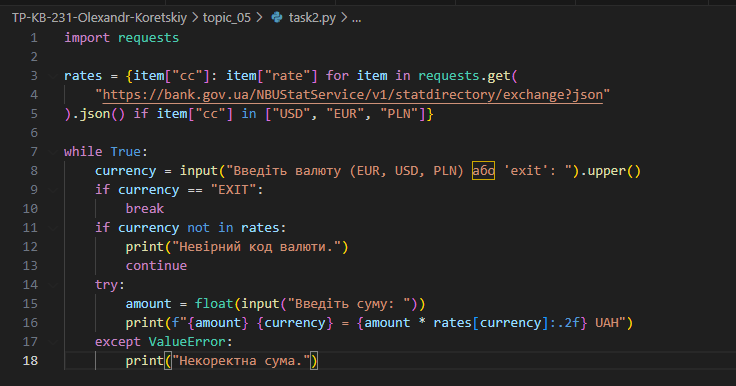
**Виведення результатів:**

Програма повідомляє вибір комп'ютера і оголошує результат гри: перемога користувача, комп'ютера або нічия.

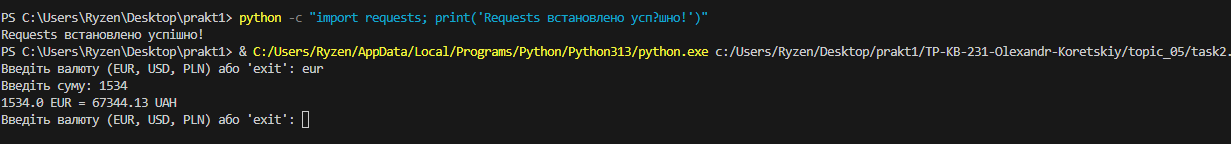
Програма конвертування іноземної валюти в українську гривню. Для отримання актуальних курсів валют необхідно використовувати API НБУ та модуль, що надає можливість виконувати запити до сторонніх сервісів requests. Достатня умова роботи – можливість конвертації для трьох іноземних валют EUR, USD, PLN. Користувачу надається можливість введення кількості та типу валюти, результат роботи програми – конвертоване значення в українських гривнях.

Напишемо код:

**Код**

****

**Результат**

****

**Імпорт бібліотеки:**

Програма використовує модуль requests для надсилання HTTP-запитів до API НБУ.

**Отримання курсів валют:**

Програма звертається до API НБУ за допомогою requests.get і отримує дані про курси валют у форматі JSON.

З отриманих даних відбираються лише курси валют для трьох валют: USD, EUR, PLN.

Результат зберігається у словнику rates, де ключ — це код валюти (наприклад, "USD"), а значення — її курс відносно гривні.

**Основний цикл програми:**

Програма працює в циклі, поки користувач не введе "exit".

**Введення валюти:**

Користувач вводить код валюти (EUR, USD, PLN).

Якщо введений код валюти некоректний (не входить до списку обраних валют), програма виводить повідомлення про помилку та пропонує спробувати знову.

**Введення суми:**

Якщо валюта введена правильно, програма запитує суму для конвертації.

Якщо користувач вводить некоректне значення (наприклад, текст замість числа), програма повідомляє про це й повертає користувача до початку циклу.

**Розрахунок конвертації:**

Програма обчислює еквівалент введеної суми в гривнях, використовуючи курс із словника rates.

Результат виводиться у форматі:  
[сума] [валюта] = [еквівалент у гривнях] UAH.

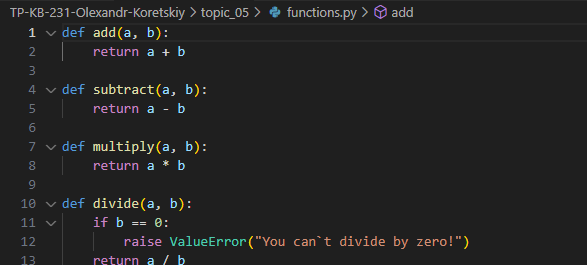
**Завершення роботи:**

Якщо користувач вводить "exit", програма припиняє роботу.

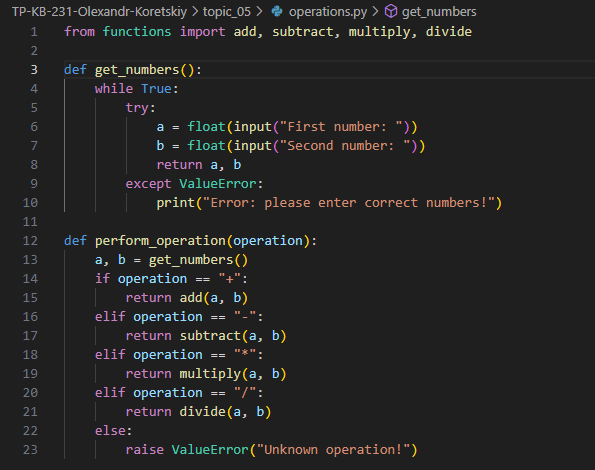
Використання модулів для програми калькулятор. Функції додавання, віднімання, множення та ділення перенести в файл functions.py. Функції запиту на введення даних для операцій та самих операцій перемістити в файл operations.py. Програму калькулятор реалізувати в файлі calc.py, до якого підключають файл functions.py та operations.py.

Напишемо коди:

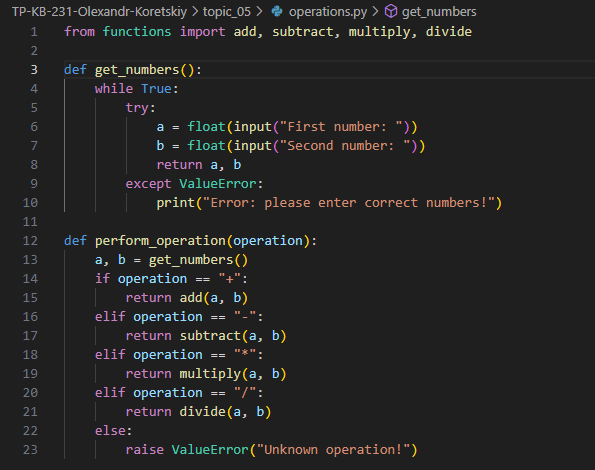
**Код *functions.py***

****

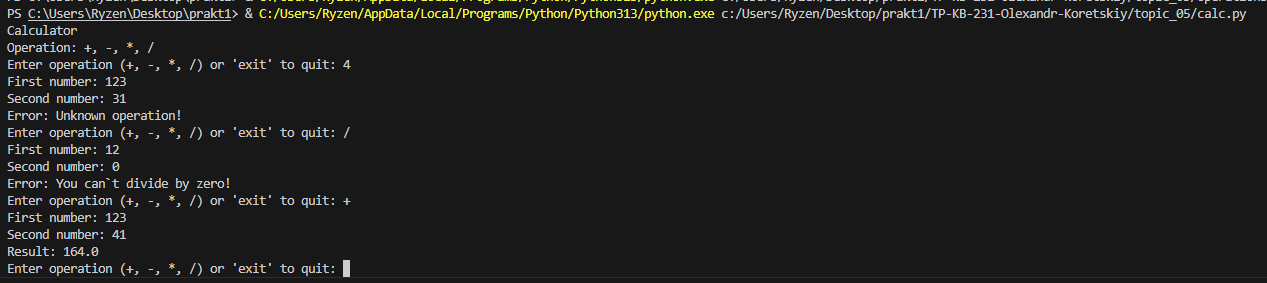
**Код *operations.py***

****

**Код *calc.py***



**Результат**

****

**Пояснення:**

**1. *functions.py* (Мінімальний функціонал арифметичних операцій)**

Цей файл відповідає за виконання базових математичних операцій. У ньому є 4 функції: додавання, віднімання, множення та ділення. Якщо ділення відбувається на нуль, функція повертає текстове повідомлення "Ділення на нуль!", замість того щоб кидати помилку.

**2. *operations.py* (Обробка введення та виконання операцій)**

**У цьому файлі:**

Функція **get\_numbers** запитує у користувача два числа, конвертує їх у тип **float** і повертає як пару.

Функція **perform\_operation** приймає код операції (наприклад, "+", "-", "\*", "/"). Вона викликає функцію **get\_numbers**, отримує два числа, а потім вибирає відповідну математичну операцію за допомогою словника. Якщо код операції не відповідає жодній з підтримуваних, повертається повідомлення "Невідома операція!".

**3. *calc.py* (Головний файл програми)**

Це основний файл програми:

Він працює у циклі, поки користувач не введе команду "**exit**".

Користувач вводить код операції (наприклад, "+", "-", "\*", "/"). Якщо введено "**exit**", програма завершується.

Якщо операція правильна, викликається функція **perform\_operation** із **operations.py**. Результат обчислення виводиться на екран.

Цей файл об'єднує всі модулі (**functions.py та operations.py**) для виконання функцій калькулятора.

Звіт до Теми №6

Робота з файлами

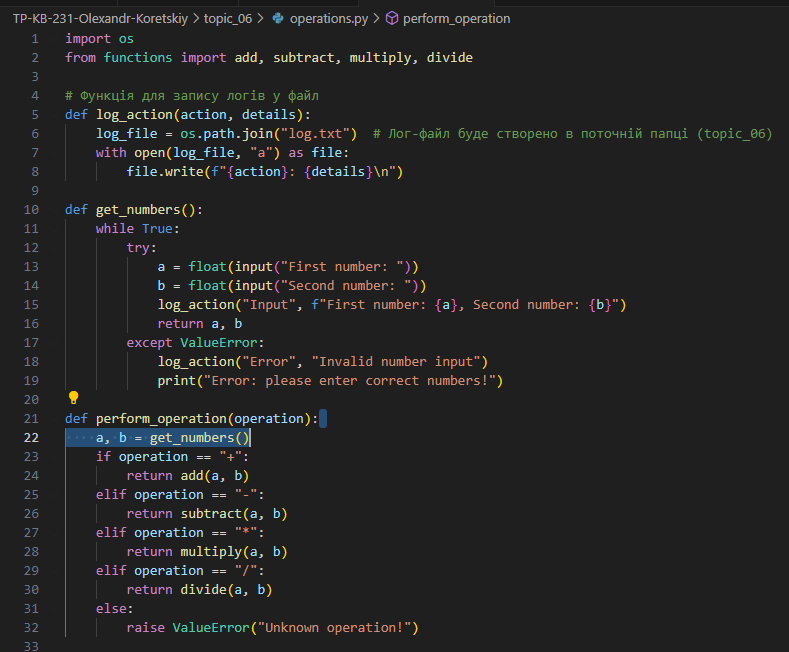
**Розробити механізм логування всіх дій, що виконує програма. Забезпечити зберігання інформації про введені данні, виконану операцію та результат виконання операції над даними.**

Модернізуємо калькулятор з попередньої теми:

**calc.py**

****

**operations.py**

****

Пояснення:

**Фіксація введення даних:** Коли користувач вводить числа, ця інформація записується у файл. Це дозволяє відслідковувати, які саме числа використовувалися для обчислень.

**Збереження результатів:** Коли виконуються арифметичні операції, результат також записується. Це допомагає побачити, які дії виконувалися і з якими результатами.

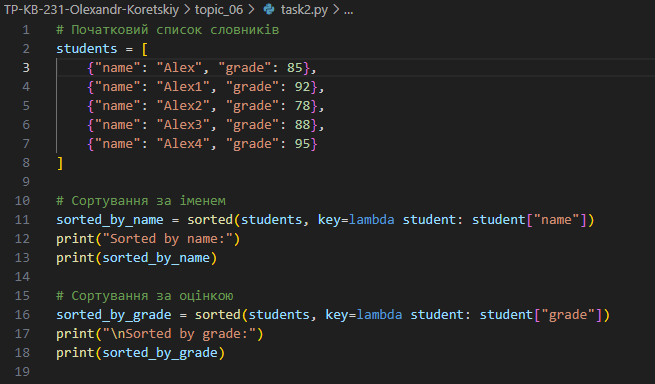
**Фіксація помилок:** Якщо під час роботи трапляються помилки (наприклад, введення некоректних даних або спроба поділити на нуль), ці помилки також записуються в лог. Це спрощує діагностику та пошук проблем.

**Вихід з програми:** Коли користувач завершує роботу, це також фіксується, щоб відзначити кінець сесії.

**Маючи не відсортований список, елементами якого є словники з двома параметрами (ім’я та оцінка) виконати сортування списку, використовуючи стандартну функцію sorted(). Другим параметром для функції sorted() має бути lambda функція, що повертає ім’я або оцінку із елемента словника.**

Напишемо код:

**Код**



Пояснення:

**Сортування за іменем:**

За допомогою функції sorted() я сортував список студентів за іменами. Для цього використовую ключ (key) — анонімну функцію, яка витягує значення за ключем "name" кожного студента.

Результат сортування буде таким, що студенти будуть розташовані в алфавітному порядку за їх іменами.

**Сортування за оцінкою:**

Я використав ту ж саму функцію sorted(), але цього разу за критерієм "grade", тобто оцінка кожного студента.

Це дозволяє впорядкувати студентів від найменшої до найбільшої оцінки.

**Виведення результатів:**

Після сортування я виводжу два результати на екран: перший — це список студентів, відсортований за іменем, а другий — відсортований за оцінкою.

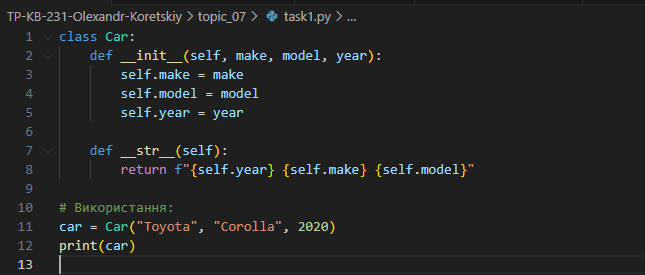
Звіт до Теми №7

Об'єктно-орієнтоване програмування

Ознайомитись з існуючими за замовченням методами класу по типу **\_\_init\_\_(self) \_\_str\_\_(self)\_\_** та надати приклади використання.

Напишемо код для тестування:

**Код**

****

Пояснення:

**Метод *\_\_init\_\_***

Цей метод я використовую для ініціалізації нового об'єкта класу. Кожен раз, коли я створюю новий об'єкт (наприклад, автомобіль), цей метод автоматично викликається. Я передаю в нього параметри, такі як марка, модель та рік випуску автомобіля. У результаті ці дані зберігаються всередині об'єкта, і я можу працювати з ними.

**Метод *\_\_str\_\_***

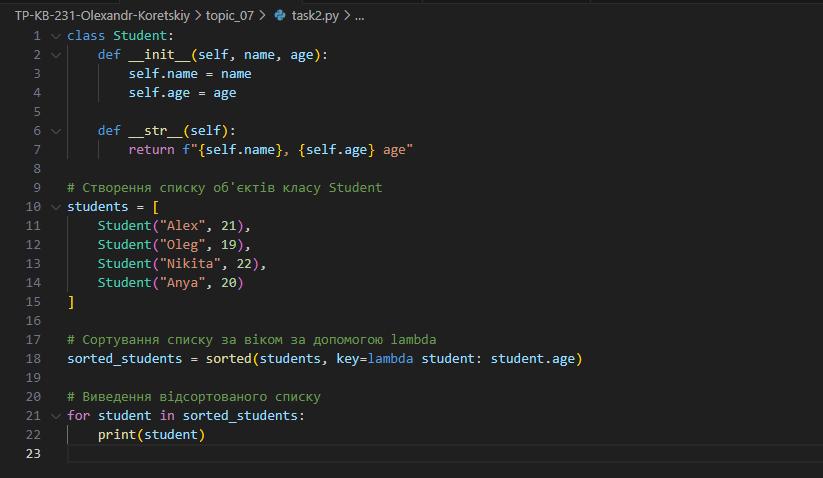
Метод ***\_\_str\_\_*** дозволяє визначити, як об'єкт буде виглядати, коли я спробую його вивести на екран (наприклад, використовуючи функцію ***print***). Я формую зручний для мене рядок, який містить інформацію про рік, марку та модель автомобіля. Коли я виводжу об'єкт на екран, **Python** автоматично викликає цей метод і відображає результат.

Таким чином, метод ***\_\_init\_\_*** дозволяє встановити початкові значення для нового об'єкта, а метод ***\_\_str\_\_*** відповідає за зручне представлення цього об'єкта при виведенні на екран.

**Розробити клас Student атрибутами якого э два параметра name та age. Створити список елементами якого є об'єкти класу Student. Написати цикл який виводить на екран елементи списку у відсортованому порядку. Для сортування використати стандартну функцію sorted. Функція sorted має використовувати lambda функцію для визначення ключа сортування.**

Напишемо код:

**Код**



Створюю клас **Student**, який має два атрибути: ім'я **(name)** та вік **(age)**. Це дозволяє створювати об'єкти класу, кожен з яких буде представляти студента з певними даними.

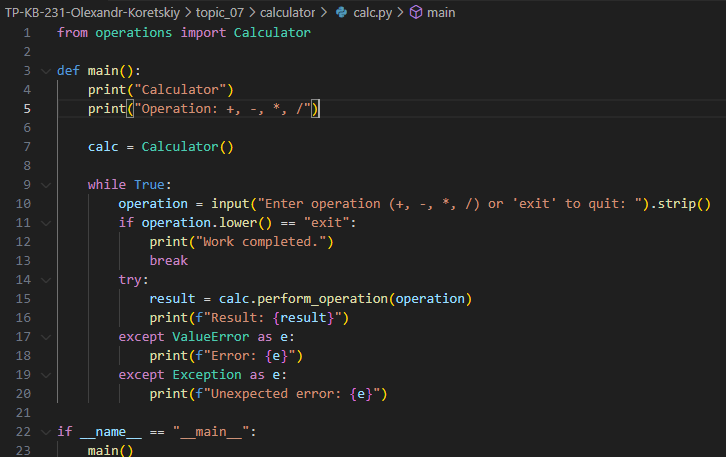
Далі, я створюю список студентів, де кожен елемент списку — це об'єкт класу **Student**. Наприклад, кожен студент має своє ім'я та вік, і ці дані зберігаються в списку у вигляді об'єктів.

Використовую функцію sorted, щоб відсортувати список студентів. Для цього вказую, що сортувати потрібно за віком студентів. Щоб визначити критерій сортування, я використовую **lambda**-функцію, яка дозволяє швидко створити анонімну функцію, що витягує вік студента (тобто, значення атрибута **age**) для сортування.

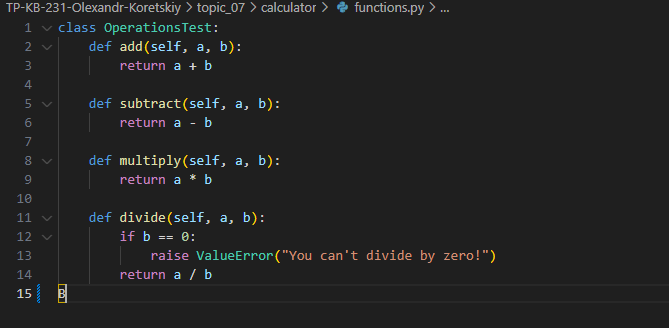
Використовую цикл для виведення на екран кожного студента після сортування. Завдяки використанню методу **\_\_str\_\_**, кожен студент буде виводитись у зручному для сприйняття форматі, наприклад, як рядок з ім'ям студента та його віком.

Таким чином, результатом буде виведення списку студентів у порядку їхнього віку, від наймолодшого до найстаршого.

**Використовуючи принципи ООП переписати програму Калькулятор. Завдання має бути виконано використовуючи модульний підхід.**Напишемо код:  
Код calc.py



Код Functions.py



Код Operations.py

****

**Пояснення:**

**Як усе працює:**

***OperationsTest***  
Це "серце" обчислень. У цьому класі зібрані всі базові операції: додавання, віднімання, множення, ділення. Логіка кожної операції прихована всередині методів, тому інші частини коду можуть просто викликати їх, не турбуючись, як вони працюють.

***Calculator***  
Це "мозок" програми. Він відповідає за взаємодію з користувачем: запитує числа, операцію та викликає відповідні методи з OperationsTest. Тут також обробляються помилки, наприклад, якщо користувач вводить неправильні дані або намагається ділити на нуль.

Функція main (з першого коду)  
Це "обличчя" програми. Вона запускає калькулятор і керує процесом: показує меню, приймає команди та завершує роботу, коли користувач цього хоче.

**Чому це зручно:**

Кожна частина відповідає за своє завдання.  
Наприклад, OperationsTest тільки виконує обчислення, а Calculator працює з користувачем. Це називається розділенням відповідальності.

Код легко змінювати.  
Якщо потрібно додати нову операцію, наприклад, піднесення до степеня, достатньо додати метод в OperationsTest і внести зміни в Calculator. Інші частини програми залишаться без змін.

Код можна використовувати повторно.  
Наприклад, клас OperationsTest можна взяти для іншого проекту, і він буде працювати без змін.

Усі три коди разом показують, як правильно організувати програму за допомогою ООП, щоб зробити її зрозумілою, зручною для використання та легкою для підтримки.