**Звіт про виконання завдань для практики з курсу «Технології програмування»**

[1. Звіт до теми №1 2](#_Toc147356045)

[1.1. Обернення рядка. 2](#_Toc147356046)

[1.2. Тестування різних методів для роботи зі строками. 2](#_Toc147356047)

[1.3. Функція для знаходження дискримінанту. 4](#_Toc147356048)

[2. Звіт до теми №2 6](#_Toc147356049)

[2.1. Написати функцію для знаходження коренів рівняння, використовуючи умови. 6](#_Toc147356050)

[2.2. Написання калькулятора за допомогою умов if, elif. Написання калькулятора за допомогою оператора match. 8](#_Toc147356051)

[3. Звіт до теми №3 13](#_Toc147356052)

[3.1. Постійно працююча програма за допомогою циклу while. 13](#_Toc147356053)

[3.2. Експерименти з методами масиву. 15](#_Toc147356054)

[3.3. Експерименти з методами для словарів. 18](#_Toc147356055)

[3.4. Написання функції для вставки нових елементів у відсортований масив. 20](#_Toc147356056)

# Звіт до теми №1

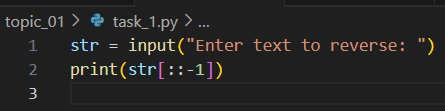
## Обернення рядка.

Завдання: Необхідно перевернути рядок **abcdefg123** так, щоб він був таким: **321gfedcba**.

Виконання завдання:

Створив змінну із запитом користувача, який запитує, яку строку обернути. Далі виводимо через print, у середині за допомогою оператора [::-1] я повернув обернений рядок. Перше та друга крапки в операторі [::-1] означають, що потрібно брати всі елементи від початку до кінця, а число «-1» означає, що відлік має починатися з кінця.

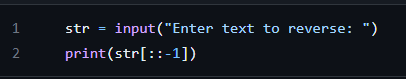
Код завдання:



GitHub:

Посилання на GitHub на цей файл: <https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_01/task_1.py>

Скріншот коду з GitHub:



## Тестування різних методів для роботи зі строками.

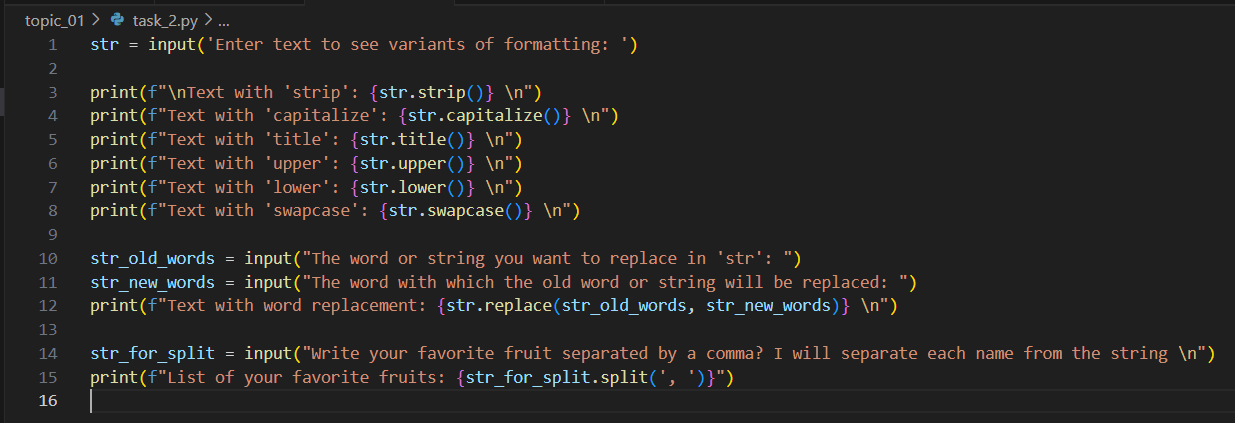
Завдання: Перевірити методи для роботи зі строками strip(), capitalize(), title(), upper(), lower(), swapcase(), replace, split().

Виконання завдання:

Створив змінну із запитом про текст у користувача, потім через функцію print виводжу відформатований текст різними методами(strip(), capitalize(), title(), upper(), lower(), swapcase()) і таким чином показую, як ці методи працюють. Далі для методу replace () я створив дві змінні з запитом до користувача. У першій я запитую, яке слово чи текст хоче замінити в строці, яка є в змінній **str**, а в другій – на яке слово чи текст буде замінений попереднє слово чи текст. Далі виводжу відформатований текст через print, у ньому всередині replace() заміняє старі слова або текст на нові слова або текст.

Далі я вирішив випробувати метод split(). Для цього створив змінну із запитом до користувача про його улюблені фрукти і попрохав перечислити його їх через кому. Потім через print виводжу список улюблених фруктів, список цей видає метод split(), за допомогою нього я виділив із строки назви фруктів, вказавши в split параметр ‘, ‘, тобто, шукати кому з пробілом. У підсумку цей метод повертає список елементів – улюблених фруктів користувача.

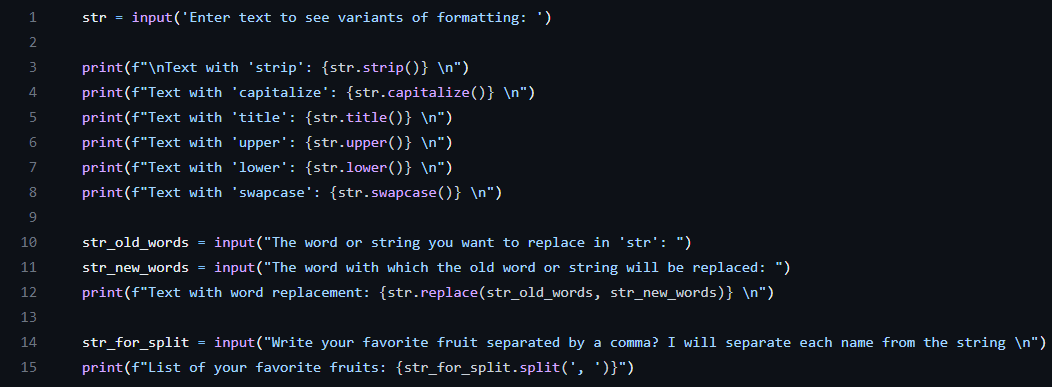
Код завдання:



GitHub:

Посилання на GitHub на цей файл: <https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_01/task_2.py>

Скріншот коду з GitHub:



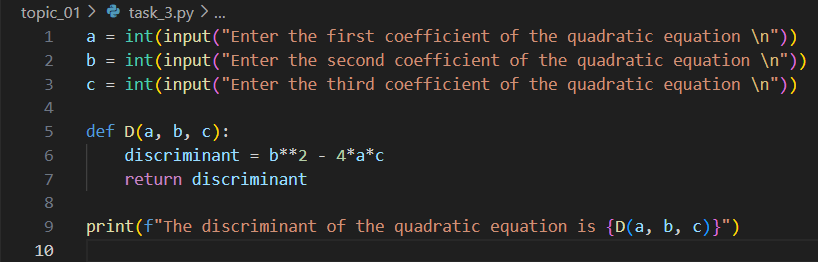
## Функція для знаходження дискримінанту.

Завдання: Потрібно написати функцію, яка приймає три параметри a, b, c – коєфіцієнти квадратного рівняння та повертає значення дискримінанту.

Виконання завдання:

Я створив три змінні із запитом до користувача, щоб отримати коєфіцієнти рівняння а, b, c. Далі створив функцію D, що приймає три аргументи a, b, c. У всередині функції створив змінну discriminant, якій присвоїв формулу b2-4ac. І через return повернув результат функції. Потім виводжу через print, при цьому всередині неї викликаю функцію D і передаю в якості параметрів змінні зі значеннями, що надав користувач. І тоді ми отримуємо результат.

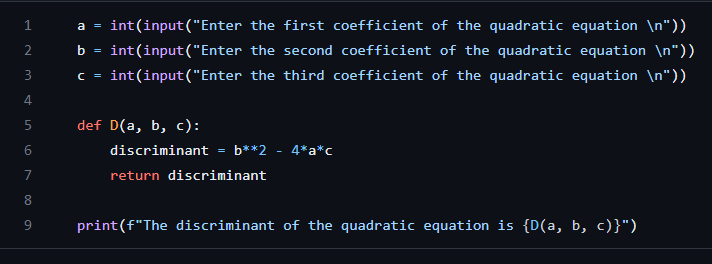
Код завдання:



GitHub:

Посилання на GitHub на цей файл: <https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_01/task_3.py>

Скріншот коду з GitHub:



# Звіт до теми №2

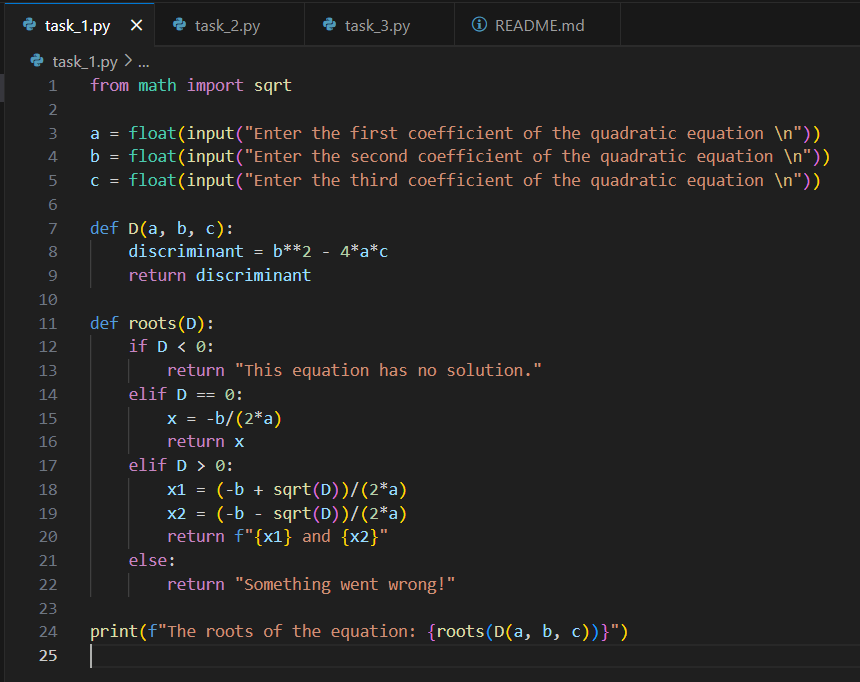
## Написати функцію для знаходження коренів рівняння, використовуючи умови.

Завдання: Необхідно написати функцію для знаходження коренів рівняння, використовуючи умови. Умови ми використовуємо для дискримінанту(далі буде показано як). Функцію знаходження дискримінанту ми візьмемо з розділу 1. завдання 1.3.

Виконання завдання:

Я імпортував з модулі math функцію sqrt. Далі переніс частину коду з розділу 1. завдання 1.3. Дещо переписав: отримання числа не в int, а вже у float для того, щоб можна було не тільки цілі числа використовувати, але й дробові. Потім написав функцію roots, яка отримує параметр – дискримінант. За допомогою умов if, elif прописав умови для різних випадків: якщо дискримінант менше нуля, повертаємо результат - стрічку, що рівняння немає коренів(стрічки 12-13), якщо дискримінант дорівнює нулю, то маємо один корінь(стрічки 14-16), обчислюємо його і виводим, якщо дискримінант більше нуля, то маємо два корені і обчислюємо і виводим. І за допомогою else, якщо піде щось не так, то виводили повідомлення про те, що щось не так. Далі через print виводимо результат, у print викликаємо функцію roots, у якості параметра передаємо функцію D з параметрами a, b, c.

Код завдання:

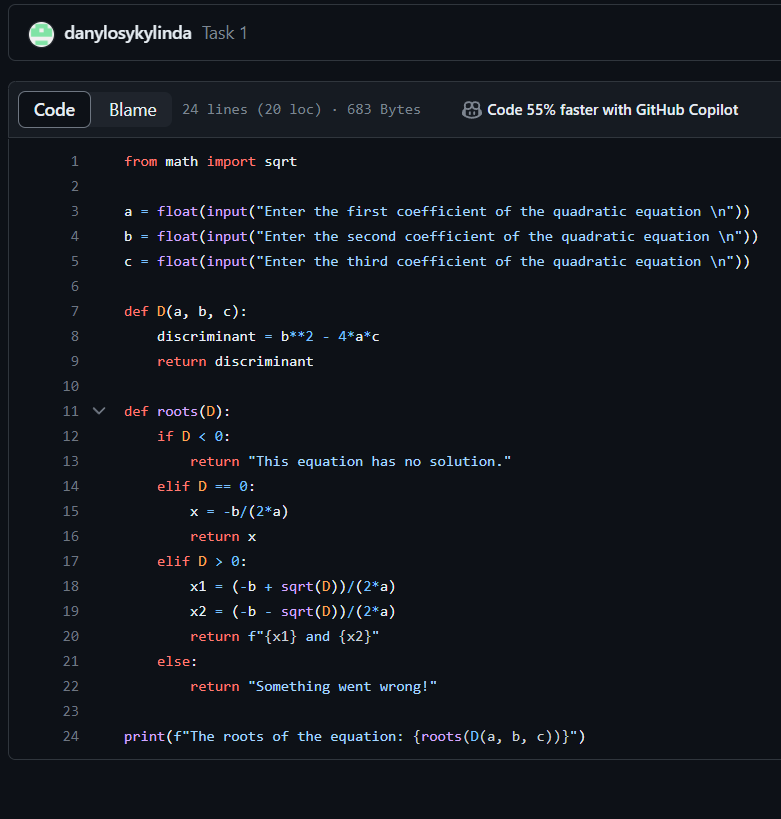


GitHub:

Посилання на GitHub на цей файл:

<https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_02/task_1.py>

Скріншот коду з GitHub:

****

## Написання калькулятора за допомогою умов if, elif. Написання калькулятора за допомогою оператора match.

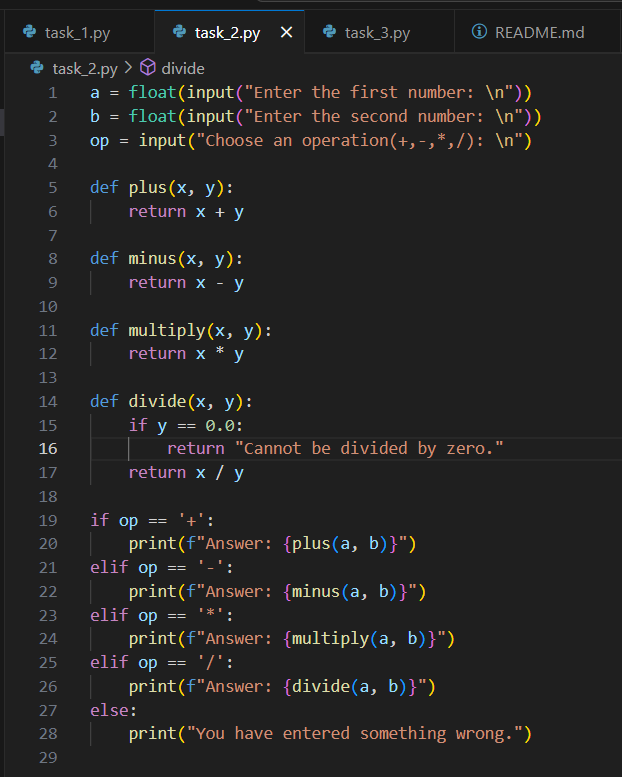
Завдання: Потрібно написати калькулятор, у якому ми запитуємо два числа і дію(+,-,/,\*) від користувача, при цьому коді використовуємо умови if, elif. Потім написати калькулятор, у коді якого ми використаємо оператор match.

Виконання завдання:

Частина 1.

Створив три змінні із запитом до користувача: a та b – числа, op – операція над цими числами. Потім написав функції, які виконують операції(plus – додавання, minus – віднімання, multiply – множення, divide - ділення). У функції divide ми ще умовою перевіряємо, чи другий аргумент є нулем, бо якщо другий дорівнює нулю, то не можна поділити, на нуль не ділиться. Потім написав умову, якщо у змінну користувач написав ‘+’, то ми виводимо результат в print, у ньому викликаємо функцію plus і передаємо два числа їй – a та b. Аналогічні дії робимо для інших операцій. У кінці дописуємо else з повідомленням, якщо користувач написав щось неправильно.

Код завдання:

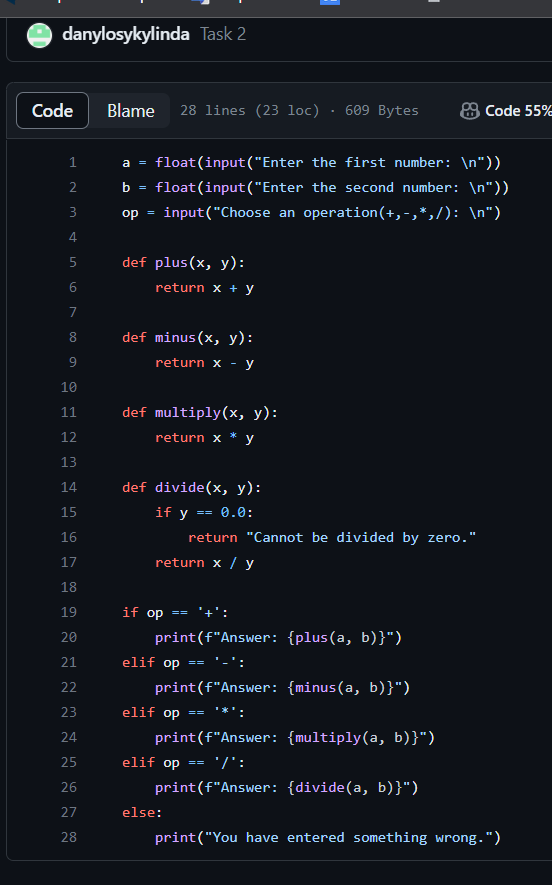


GitHub:

Посилання на GitHub на цей файл:

<https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_02/task_2.py>

Скріншот коду з GitHub:

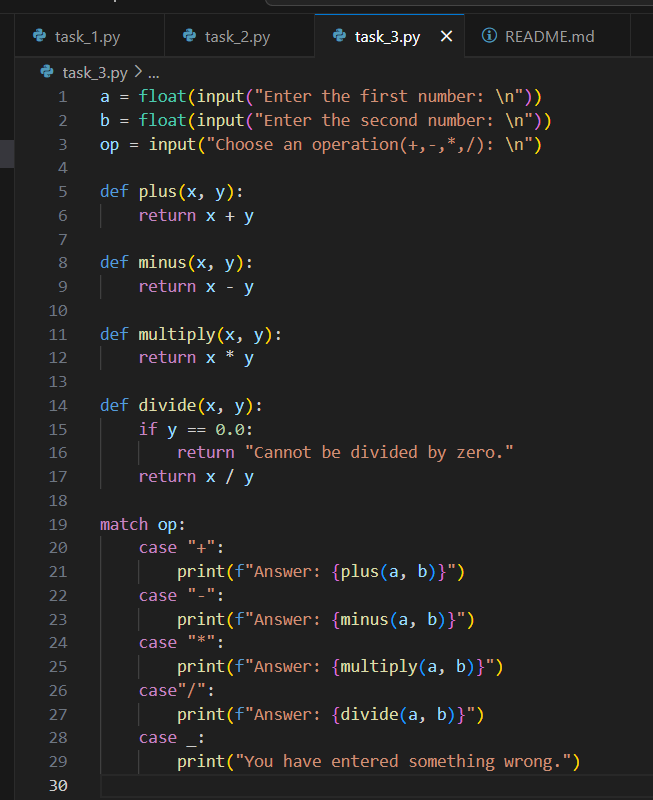
****

Виконання завдання:

Частина 2.

Стрічки 1-17 скопійовані з частини 1 цього завдання. Далі ми за допомогою оператора match прописуємо випадки(case) для (+, -, \*, /). І аналогічно, як частині 1 цього завдання, ми викликаємо функцію print, у ній викликаємо функцію plus і передаємо два аргументи a та b. Аналогічно з іншими математичними діями. У кінці прописуємо case \_: - це означає, що якщо будуть уведені інші символи відмінні, що зазначені в інших case, то реагувати певним чином, у даному case ми виводимо повідомлення, що щось пішло не так.

Код завдання:

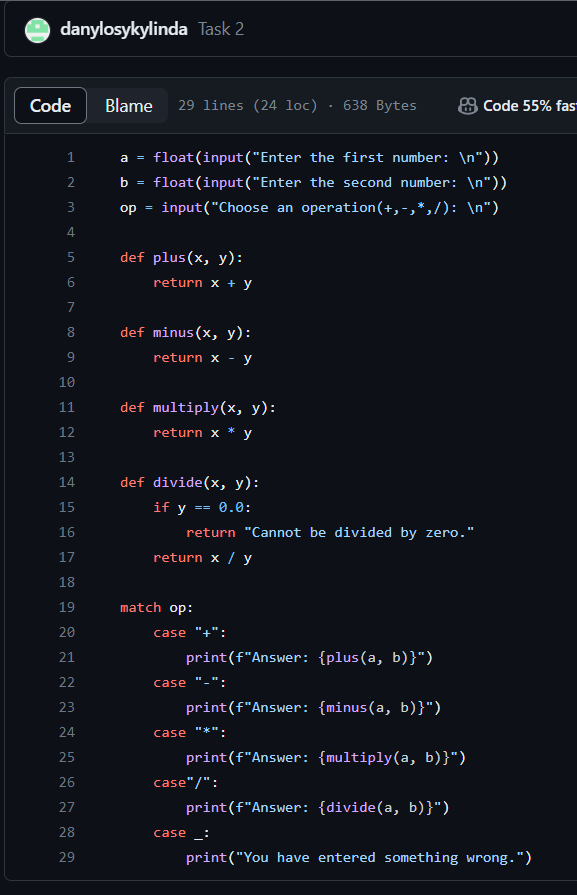


GitHub:

Посилання на GitHub на цей файл:

<https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_02/task_3.py>

Скріншот коду з GitHub:

****

# Звіт до теми №3

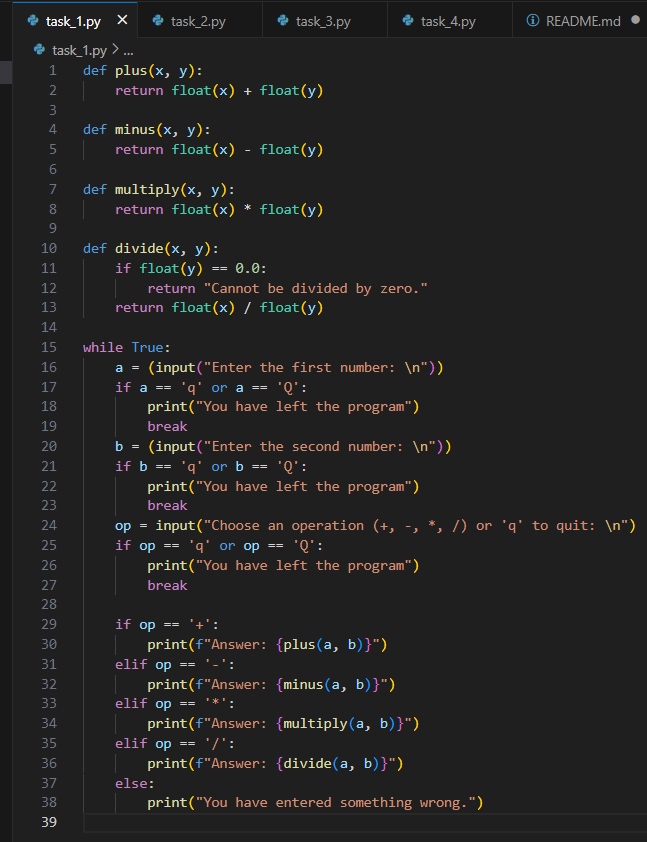
## Постійно працююча програма за допомогою циклу while.

Завдання: Необхідно переписати калькулятор із з розділу 2 пункту 2 так, щоб він безперервно працював, тобто, після введення даних користувачем він знову запитував користувача про числа та дії, також потрібно реалізувати, щоб користувач міг завжди вийти із циклу.

Виконання завдання:

Для початку беремо код з попереднього розділу пункту 2 (тут візьмемо калькулятор на if/elif), для більшої читальбельності винесемо функції окремо, а все інше обернемо в цикл while. while True: означає, що поки не буде break програма буде працювати. Після кожного вводу користувача перевіряємо, чи користувач не увів “Q” або “q”(ці букви у нас будуть позначати вихід). Якщо “q” чи “Q” було уведено, то значить виводимо повідомлення про вихід, далі break і цикл зупинено, програма завершила роботу, якщо ж не уведено, то програма починає знову запитувати користувача про числа та операцію над ними.

Код завдання:

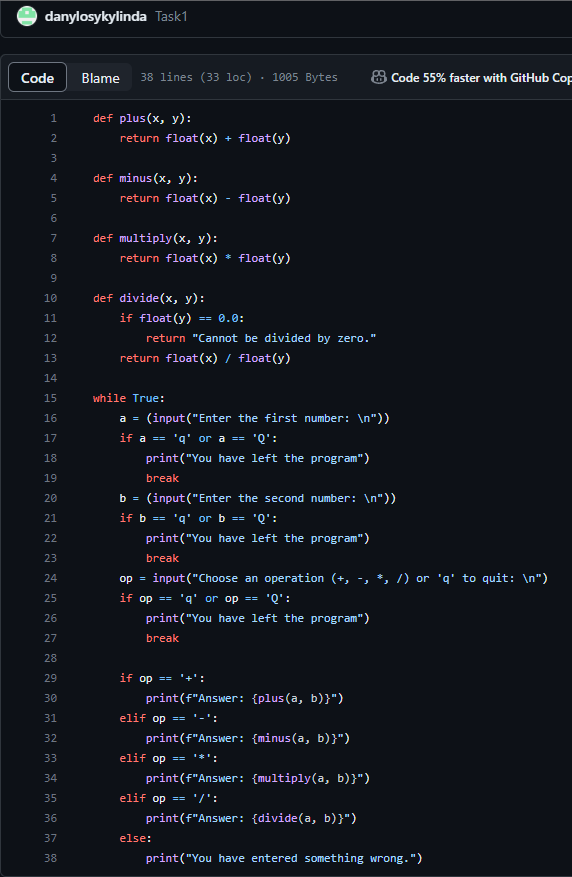


GitHub:

Посилання на GitHub на цей файл:

<https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_03/task_1.py>

Скріншот коду з GitHub:



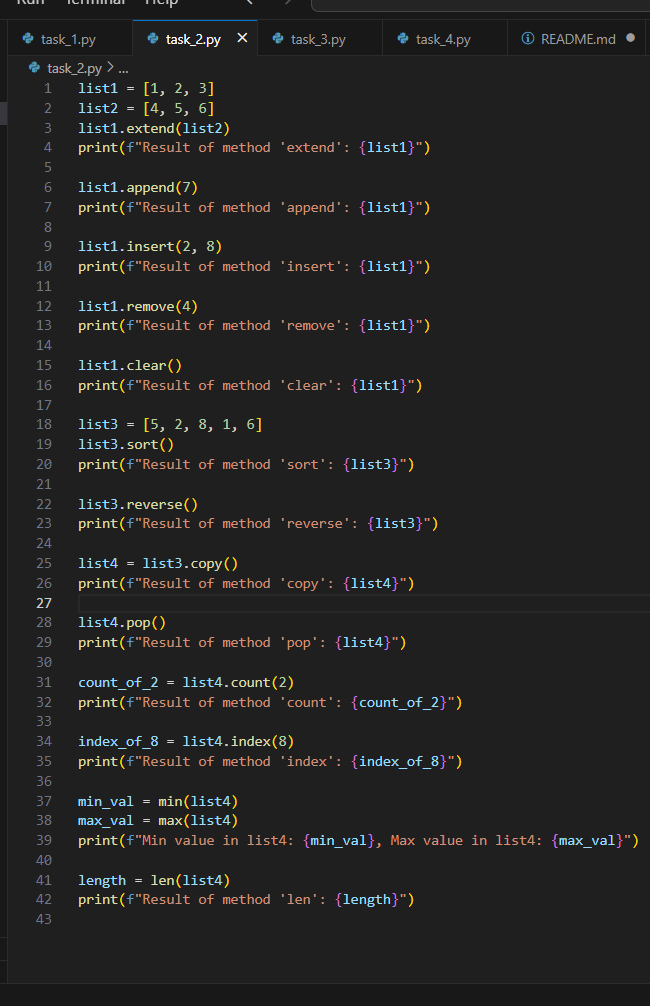
## Експерименти з методами масиву.

Завдання: Поекспериментувати з методами масиву такими як: extend(), append(), insert(id, val), remove(val), clear(), sort(), reverse(), copy() та з іншими методами масиву.

Виконання завдання:

Я створив два масиви list1 та list2 із різними числами. Застосувавши метод extend() з аргументом масив list2 для list1, я доповнив масив list1 масивом list2. Далі методом append() я додав число 7 в кінець масиву list1. Потім за допомогою методу insert() я поставив число 8 в масив list1 на позицію 2. Далі я видалив число 4 з масиву list1 за допомогою методу remove(). Потім за допомогою методу clear() я очистив масив list1 – тепер він пустий. Далі я створив масив list3 з різними числами, використавши метод sort(), я відсортував масив з числами в порядку зростання. Потім за допомогою методу reverse() я повернув всі значення масиву у зворотньому порядку. Далі я створив змінну, в яку копіював list3 за допомогою методу copy(). Потім для list4 я використав метод pop(), цей метод видалив останній елемент в масиві. Далі я використав метод count() з аргументом число 2 в масиві list4, цей метод допомагає знайти нам, скільки чисел 2 знаходиться в цьому масиві. Потім я використав метод index() з аргументом число 8 в масиві list4, цей метод знаходить індекс цього значення у масиві, тобто в цьому випадку він шукає на якій позиції в масиві знаходиться число 8. Далі я використав вбудовані функції min() і max() та передав в якості аргументу масив list4, відповідно ці функції знайшли найменше та найбільше значення в масиві. Далі використав метод len() для масиву list4, щоб дізнатися, яка кількість елементів знаходиться в масиві.

Код завдання:

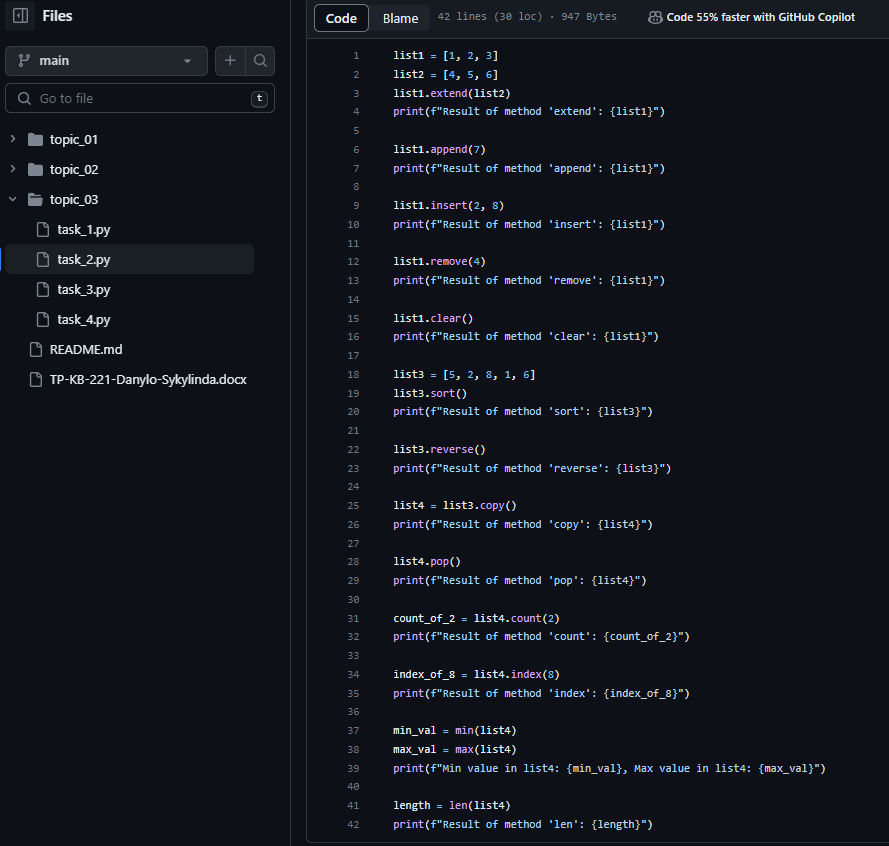


GitHub:

Посилання на GitHub на цей файл:

<https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_03/task_2.py>

Скріншот коду з GitHub:



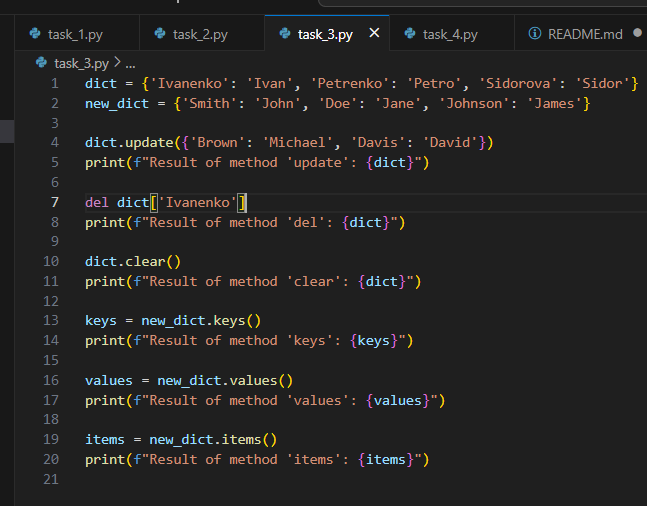
## Експерименти з методами для словарів.

Завдання: Поекспериментувати з методами для словарів такі як: update(), del, clear, keys(), values(), items().

Виконання завдання:

Я створив два словарі з іменами та прізвищами. Далі за допомогою методу update() я додав в масив dict два ім’я-прізвища (ім’я як ключ, а прізвище як значення). Потім я видалив з словара dict одне ім’я-прізвище за допомогою del, звертаючись по ключу. Потім я за допомогою методу clear() я очистив словар dict. Далі за допомогою методу keys() я дізнався про всі наявні ключі в словарі new\_dict. За допомогою методу values() я дізнався про всі значення ключів в словарі new\_dict. Потім за допомогою методу items() я дізнався про кожну пару ключ-значення в словарі new\_dict, тобто, відразу дізнався про кожен ключ і його значення, що є словарі new\_dict.

Код завдання:

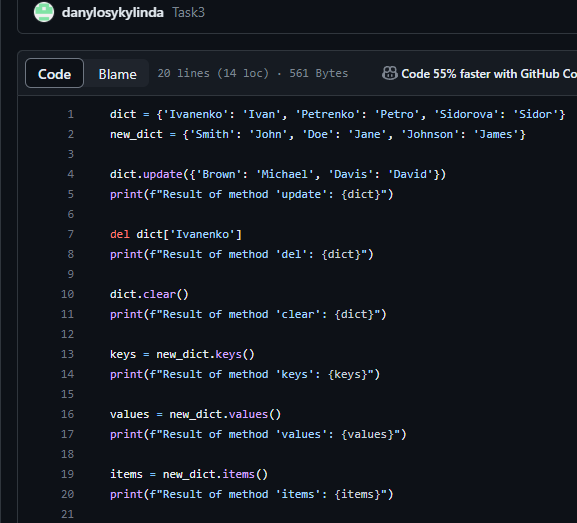


GitHub:

Посилання на GitHub на цей файл:

<https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_03/task_3.py>

Скріншот коду з GitHub:



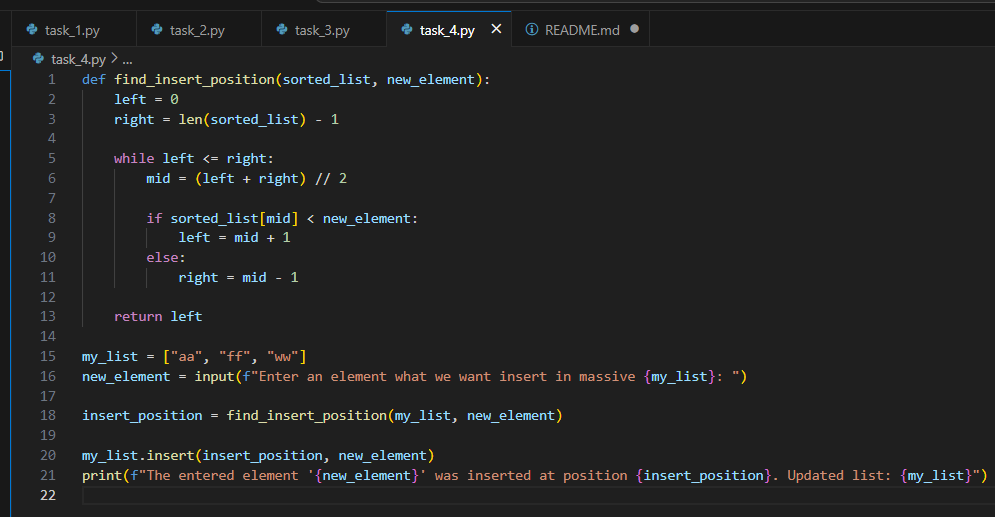
## Написання функції для вставки нових елементів у відсортований масив.

Завдання: Написати функцію, яка буде шукати позицію для вставки в масив нового елемента, позиція для нового елементу має бути така, щоб масив лишився сортованим. При цьому не можна використовувати вбудовані функції.

Виконання завдання:

Щоб реалізувати правильну вставку вже у відсортований масив my\_list, потрібно використати алгоритм бінарного пошуку. Наша функція find\_insert\_position прийматиме два аргументи: перший – відсортований масив, другий – значення, яке потрібно правильно поставити в масив так, щоби він залишився відсортованим. Як це буде працювати? Суть алгоритму бінарного пошуку це ділення масиву на дві частини. Спочатку створимо дві змінні left та right. left = 0, а right = len(sorted\_list) – 1. Чому -1? Тому що right це буде індексом останнього елемента, а останній елемент справді довжина масиву – 1, оскільки масиви починаються з нуля, а функція len рахує кількість елементів. Далі пишемо цикл «допоки left менше або дорівнює right», у ньому визначаємо середину постійно, бо за цим алгоритмом ми масив постійно ділимо навпіл. Середина визначається сумою лівої частини і правої частини індексів ділене на два на виході цілим числом. І далі за ситуацією, порівнюємо, якщо елемент, що знаходиться посередині менше за новий елемент, який ми хочемо вставити, то лівому індексу присвоюємо середину + 1, якщо ж навпаки то правому індексу присвоюємо середину – 1. І в кінці ми повертаємо лівий індекс, який і буде позицією для вставки. І далі вже вставляємо елемент за допомогою методу insert() у відсортований масив і виводимо масив, щоб переконатися, що все вийшло правильно.

Код завдання:



GitHub:

Посилання на GitHub на цей файл:

<https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/blob/main/topic_03/task_4.py>

Скріншот коду з GitHub:

