**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.

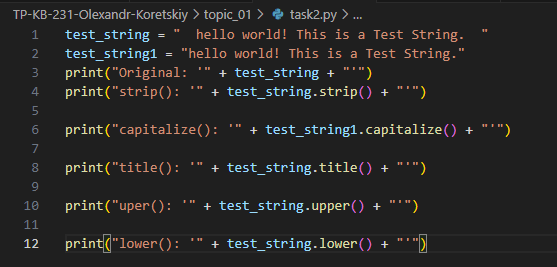
**upper() –** усі букви в рядку робить з великої літери.

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

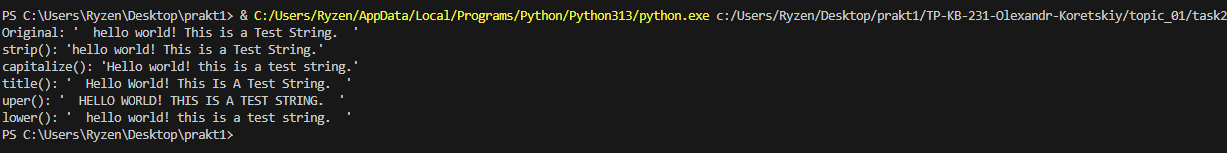
Як протестувати надані функції: **strip(), capitalize(), title(), upper(), lower().**

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()**

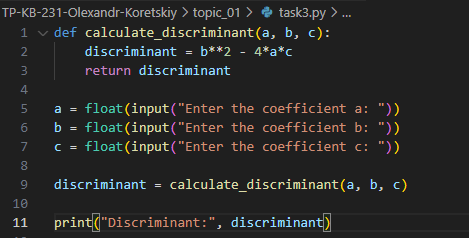
**print("upper(): '" + test\_string.upper() + "'") -** виконання функції **upper()**

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

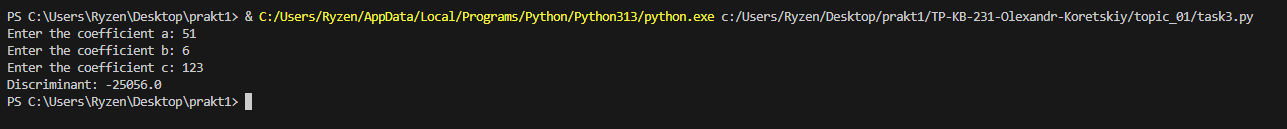
**Функція пошуку дискримінанту квадратного рівняння.**

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

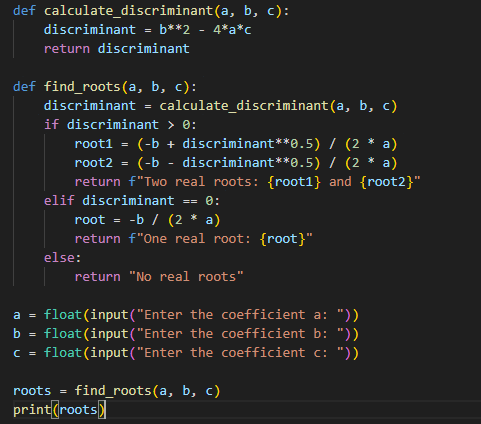
Звіт до Теми №2

Умовний перехід

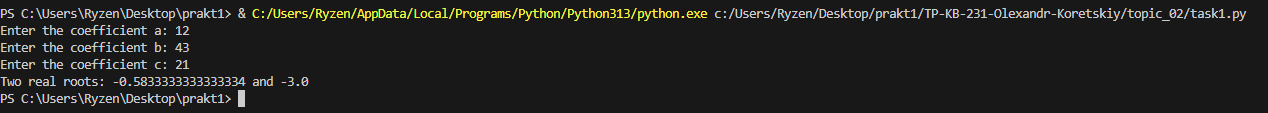
**Написати функцію пошуку коренів квадратного рівняння використовуючи функцію розрахунку дискримінанту з попередньої теми та умовні переходи.**

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.

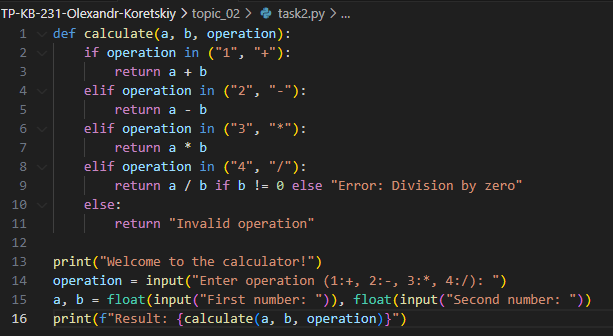
**roots = find\_roots(a, b, c)**  
Викликається функція **find\_roots**, і результат зберігається у змінній **roots**.

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

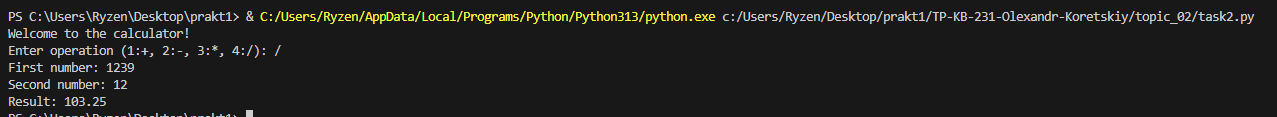
**Написати програму калькулятор використовуючи *if else* конструкцію. Кожна операція має бути виконана в окремій функції.**

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

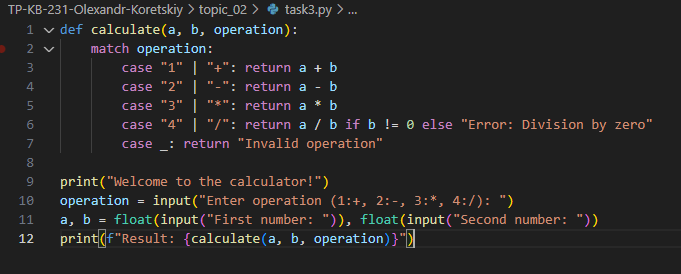
4.**Перевірка на коректність операції:**

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

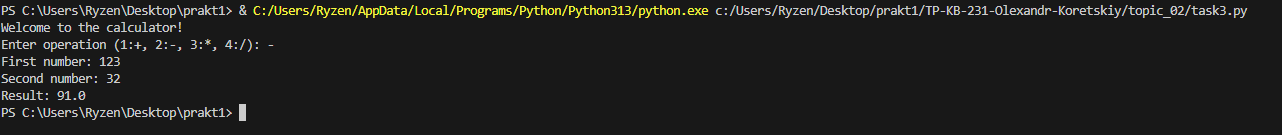
**5.Виведення результату:**

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.

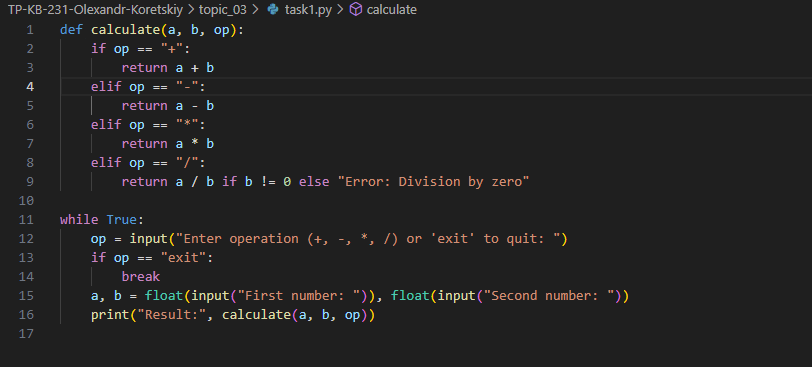
Звіт до Теми №3

Цикли

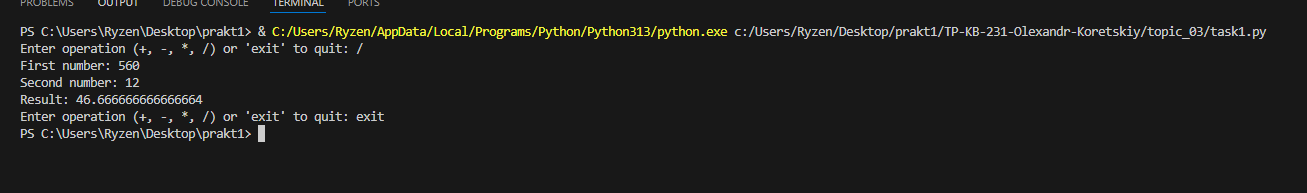
**Написати програму калькулятор з постійними запитами на введення нових даних та операцій. За основу взяти програму калькулятор з попередньої теми. Реалізувати механізм завершення програми після отримання відповідної команди.**

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:**

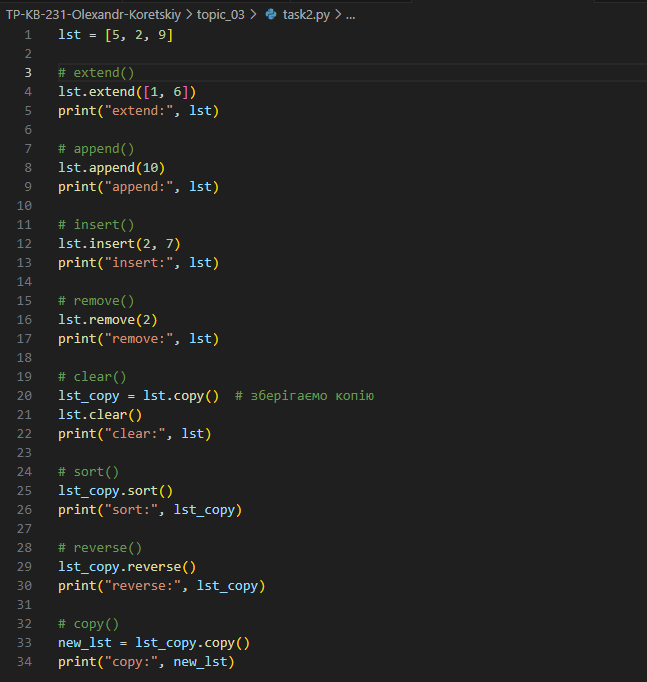
Користувач вводить два числа.

Результат обчислень виводиться на екран.

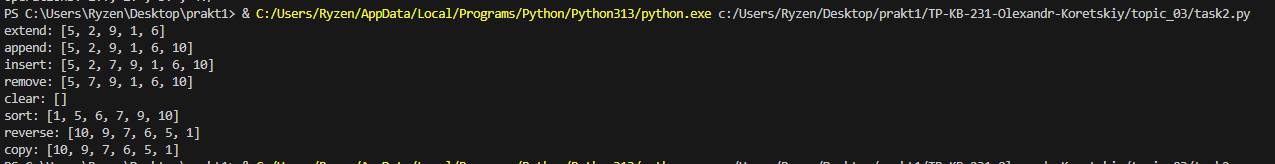
**Написати програму тестування функцій списків таких як: extend(), append(), insert(id, val), remove(val), clear(), sort(), reverse(), copy()**

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:**

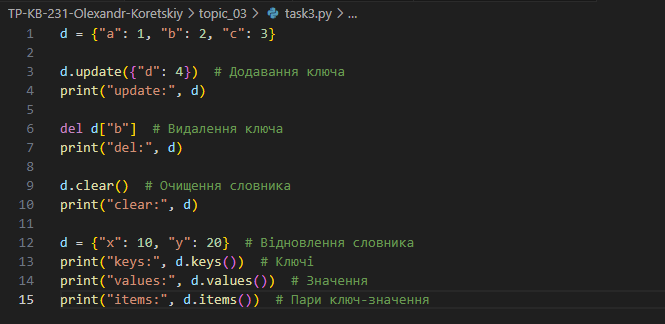
Копія списку створюється за допомогою ***copy***, щоб продемонструвати, як отримати дубль списку для окремої роботи з ним.

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

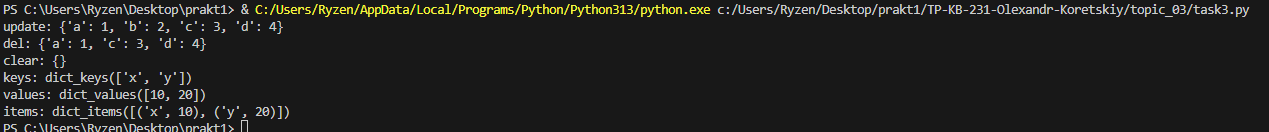
**Написати програму тестування функцій словників таких як: update(), del(), clear(), keys(), values(), items()**

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):**

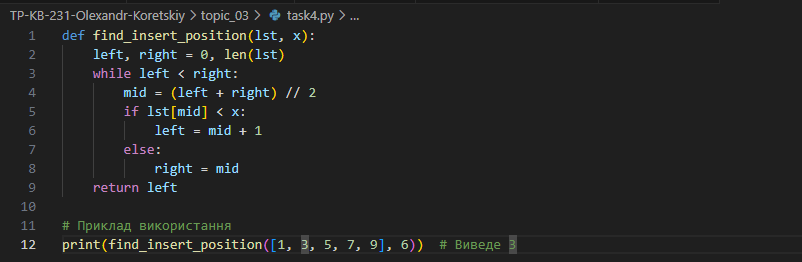
Повертає список кортежів, де кожен кортеж — це пара ключ-значення.

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.