**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:

**Обернути рядок в зворотному порядку.**

Хід виконання роботи:

Для реалізації функції, яка обертає рядок, використовується зріз (s[::-1]). Цей метод дозволяє швидко отримати новий рядок, що містить символи оригінального рядка в зворотному порядку.

Текст програми :

|  |
| --- |
| # Обернути рядок у зворотному порядку:  def reverse\_string(s):      return s[::-1]  print(reverse\_string("Hello, world!")) |

**Виконати тестування функцій, що працюють з рядками: strip(), capitalize(), title(), upper(), lower().**

Хід виконання роботи:

Було проведено тестування стандартних функцій Python для обробки рядків:

strip() видаляє зайві пробіли з початку і кінця рядка.

capitalize() перетворює першу літеру рядка у велику.

title() перетворює кожне слово в рядку на "титульний" формат (з великої літери).

upper() переводить всі літери рядка у верхній регістр.

lower() переводить всі літери у нижній регістр.

Текст програми:

|  |
| --- |
| # Виконати тестування функцій,  # що працюють з рядками: strip(), capitalize(), title(), upper(), lower().  def test\_string\_functions(s):      stripped = s.strip()      capitalized = s.capitalize()      titled = s.title()      uppercased = s.upper()      lowercased = s.lower()      return {          "strip": stripped,          "capitalize": capitalized,          "title": titled,          "upper": uppercased,          "lower": lowercased      }  string = "  hello world  "  results = test\_string\_functions(string)  for func, result in results.items():      print(f"{func}: {result}") |

**Написати функцію пошуку дискримінанту квадратного рівняння.**

Хід виконання роботи:

Функція **find\_discriminant(a, b, c)** реалізована згідно з формулою для знаходження дискримінанту: **D = b^2 - 4ac** . При тестуванні на коефіцієнтах a = 1, b = -3, c = 2 дискримінант виводиться коректно і дорівнює 1.

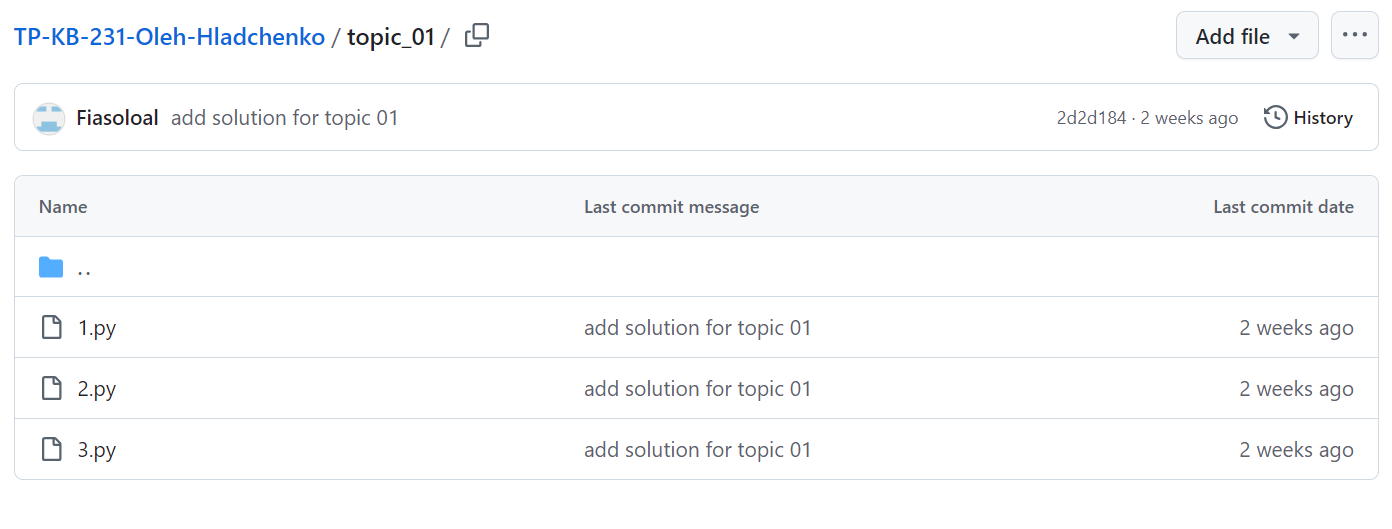
Текст програми:

|  |
| --- |
| # Написати функцію пошуку дискримінанту квадратного рівняння.  def find\_discriminant(a, b, c):      discriminant = b\*\*2 - 4\*a\*c      return discriminant  a, b, c = 1, -3, 2  d = find\_discriminant(a, b, c)  print(f"Дискримінант: {d}") |

Посилання на github:

<https://github.com/Fiasoloal/TP-KB-231-Oleh-Hladchenko/tree/main/topic_01>

Знімок екрану з посилання на github:



Звіт до Теми №2

Функції та змінні

Під час виконання практичного завдання до Теми №2 було надано варіанти рішення до наступних задач:

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

Хід виконання роботи:

Квадратне рівняння: **ax^2 + bx + c = 0**. Щоб знайти корені цього рівняння, нам потрібно спочатку обчислити дискримінант **D = b^2 - 4ac**. Якщо дискримінант більше 0, є два корені. Якщо дискримінант дорівнює 0, є один корінь. Якщо дискримінант менше 0, коренів немає.

Текст програми:

|  |
| --- |
| #1) Написати функцію пошуку коренів квадратного рівняння використовуючи функцію  #розрахунку дискримінанту з попередньої теми та умовні переходи.  import math  def discriminant(a, b, c):      return b\*\*2 - 4 \* a \* c  def find\_roots(a, b, c):      d = discriminant(a, b, c)      if d > 0:          root1 = (-b + math.sqrt(d)) / (2 \* a)          root2 = (-b - math.sqrt(d)) / (2 \* a)          return f"Два корені: x1 = {root1}, x2 = {root2}"      elif d == 0:          root = -b / (2 \* a)          return f"Один корінь: x = {root}"      else:          return "Коренів немає"  a = 1  b = -3  c = 2  print(find\_roots(a, b, c)) |

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

Хід виконання роботи:

Калькулятор з використанням if-else. Ця програма дозволяє додавати, віднімати, множити і ділити два числа. Використовуючи умови для перевірки, яку операцію користувач хоче виконати.

Текст програми:

|  |
| --- |
| #2) Написати програму калькулятор використовуючи if else конструкцію.  #Кожна операція має бути виконана в окремій функції.  def add(a, b):      return a + b  def subtract(a, b):      return a - b  def multiply(a, b):      return a \* b  def divide(a, b):      if b == 0:          return "Помилка: не можна ділити на нуль!"      return a / b  def calculator():      a = float(input("Введіть перше число: "))      b = float(input("Введіть друге число: "))      operation = input("Введіть операцію (+, -, \*, /): ")      if operation == '+':          print(f"Результат: {add(a, b)}")      elif operation == '-':          print(f"Результат: {subtract(a, b)}")      elif operation == '\*':          print(f"Результат: {multiply(a, b)}")      elif operation == '/':          print(f"Результат: {divide(a, b)}")      else:          print("Помилка: невідома операція")  calculator() |

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

Хід виконання роботи:

Калькулятор з використанням match. Ця програма дозволяє додавати, віднімати, множити і ділити два числа. Використовуючи операцію яка викликає окрему функцію для обчислень.

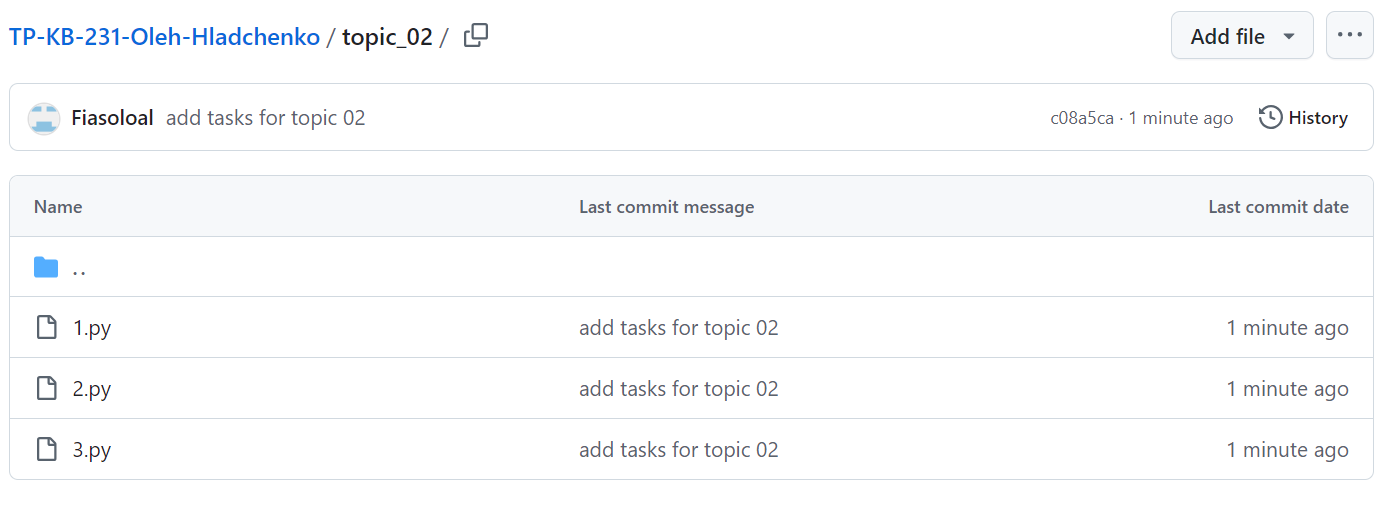
Текст програми:

|  |
| --- |
| #3) Написати програму калькулятор використовуючи match конструкцію.  #Кожна операція має бути виконана в окремій функції.  def add(a, b):      return a + b  def subtract(a, b):      return a - b  def multiply(a, b):      return a \* b  def divide(a, b):      if b == 0:          return "Помилка: не можна ділити на нуль!"      return a / b  def calculator():      a = float(input("Введіть перше число: "))      b = float(input("Введіть друге число: "))      operation = input("Введіть операцію (+, -, \*, /): ")      match operation:          case '+':              print(f"Результат: {add(a, b)}")          case '-':              print(f"Результат: {subtract(a, b)}")          case '\*':              print(f"Результат: {multiply(a, b)}")          case '/':              print(f"Результат: {divide(a, b)}")          case \_:              print("Помилка: невідома операція")  calculator() |

Посилання на github:

<https://github.com/Fiasoloal/TP-KB-231-Oleh-Hladchenko/tree/main/topic_02>

Знімок екрану з посилання на github:



Звіт до Теми №3

Функції та змінні

Під час виконання практичного завдання до Теми №3 було надано варіанти рішення до наступних задач:

Хід виконання роботи:

**Програма калькулятор з постійними запитами**

Програма калькулятора, що працює в нескінченному циклі та виконує основні арифметичні операції: додавання, віднімання, множення та ділення. Користувач може ввести дві змінні та вибрати операцію для виконання. В програмі також реалізовано механізм завершення роботи при введенні ключового слова "вихід".

Текст програми :

|  |
| --- |
| #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.  def calculator():      print("Калькулятор. Введіть 'вихід', щоб завершити програму.")      while True:          num1 = input("Введіть перше число (або 'вихід' для завершення): ")          if num1.lower() == 'вихід':              break          num2 = input("Введіть друге число: ")          operation = input("Введіть операцію (+, -, \*, /): ")          try:              num1 = float(num1)              num2 = float(num2)              if operation == '+':                  result = num1 + num2              elif operation == '-':                  result = num1 - num2              elif operation == '\*':                  result = num1 \* num2              elif operation == '/':                  result = num1 / num2              else:                  print("Неправильна операція.")                  continue              print(f"Результат: {result}")          except ValueError:              print("Будь ласка, введіть правильні числа.")  calculator() |

**Програма тестування функцій списків**

Ця програма перевіряє різні методи роботи зі списками, такі як extend(), append(), insert(), remove(), clear(), sort(), reverse(), copy(). Кожен метод застосовується до списку, а результати виводяться на екран.

Текст програми:

|  |
| --- |
| #Написати програму тестування  #функцій списків таких як: extend(), append(), insert(id, val), remove(val), clear(), sort(), reverse(), copy()  def test\_list\_functions():      my\_list = [1, 2, 3]      print("Початковий список:", my\_list)      my\_list.extend([4, 5])      print("Після extend():", my\_list)      my\_list.append(6)      print("Після append():", my\_list)      my\_list.insert(0, 0)      print("Після insert():", my\_list)      my\_list.remove(3)      print("Після remove():", my\_list)      my\_list.clear()      print("Після clear():", my\_list)      my\_list = [3, 1, 2]      my\_list.sort()      print("Після sort():", my\_list)      my\_list.reverse()      print("Після reverse():", my\_list)      copied\_list = my\_list.copy()      print("Скопійований список:", copied\_list)  test\_list\_functions() |

**Програма тестування функцій словників**

Програма тестує методи для роботи зі словниками, включаючи update(), del, clear(), keys(), values(), та items(). Використовується базовий словник, і кожна функція викликається окремо з виведенням результатів.

Текст програми:

|  |
| --- |
| #Написати програму тестування функцій словників таких як: update(), del(), clear(), keys(), values(), items()  def test\_dict\_functions():      my\_dict = {'a': 1, 'b': 2}      print("Початковий словник:", my\_dict)      my\_dict.update({'c': 3})      print("Після update():", my\_dict)      del my\_dict['a']      print("Після del():", my\_dict)      my\_dict.clear()      print("Після clear():", my\_dict)      my\_dict = {'a': 1, 'b': 2}      print("Ключі:", my\_dict.keys())      print("Значення:", my\_dict.values())      print("Пари ключ-значення:", my\_dict.items())  test\_dict\_functions() |

**Пошук позиції для вставки елементу в відсортований список**

Функція приймає відсортований список та новий елемент і знаходить позицію, куди слід вставити цей елемент для збереження сортування.

Текст програми:

|  |
| --- |
| #Маючи відсортований список, написати функцію пошуку позиції для вставки нового елементу в список.  def find\_insert\_position(sorted\_list, new\_element):      for index, element in enumerate(sorted\_list):          if new\_element < element:              return index      return len(sorted\_list)  sorted\_list = [1, 3, 5, 7]  new\_element = 4  position = find\_insert\_position(sorted\_list, new\_element)  print(f"Позиція для вставки {new\_element} в {sorted\_list}: {position}") |

Посилання на github:

<https://github.com/Fiasoloal/TP-KB-231-Oleh-Hladchenko/tree/main/topic_03>

Знімок екрану з посилання на github: