**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: Для змінення рядка фіксованої довжини (10 символів) так, щоб символи розташовувалися у зворотному порядку, текст написаний через функцію input(), після чого застосовувався зріз text[:fixed\_len][::-1], де fixed\_len = 10. Отриманий результат виводився на екран.

Текст програми:

|  |
| --- |
| text = input("Введи текст: ")  fixed\_len = 10  result = text[:fixed\_len ][::-1]  print(result) |

**Тестування функцій**

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

Хід виконання завдання: Було реалізовано функцію test\_string\_methods(), яка демонструє роботу базових методів для обробки рядків:

strip() — видаляє зайві пробіли з кінців рядка;

capitalize() — перетворює перший символ у верхній регістр;

title() — робить великі літери на початку кожного слова;

upper() — переводить усі символи у верхній регістр;

lower() — переводить усі символи у нижній регістр.

Текст програми:

|  |
| --- |
| def test\_string\_methods():      original = "hello worLd!          "      print("Original:", repr(original))      print("strip():", repr(original.strip()))      print("capitalize():", original.capitalize())      print("title():", original.title())      print("upper():", original.upper())      print("lower():", original.lower())  test\_string\_methods() |

**Пошук дискримінанта**

Необхідно написати функцію пошуку дискримінанту квадратного рівняння.

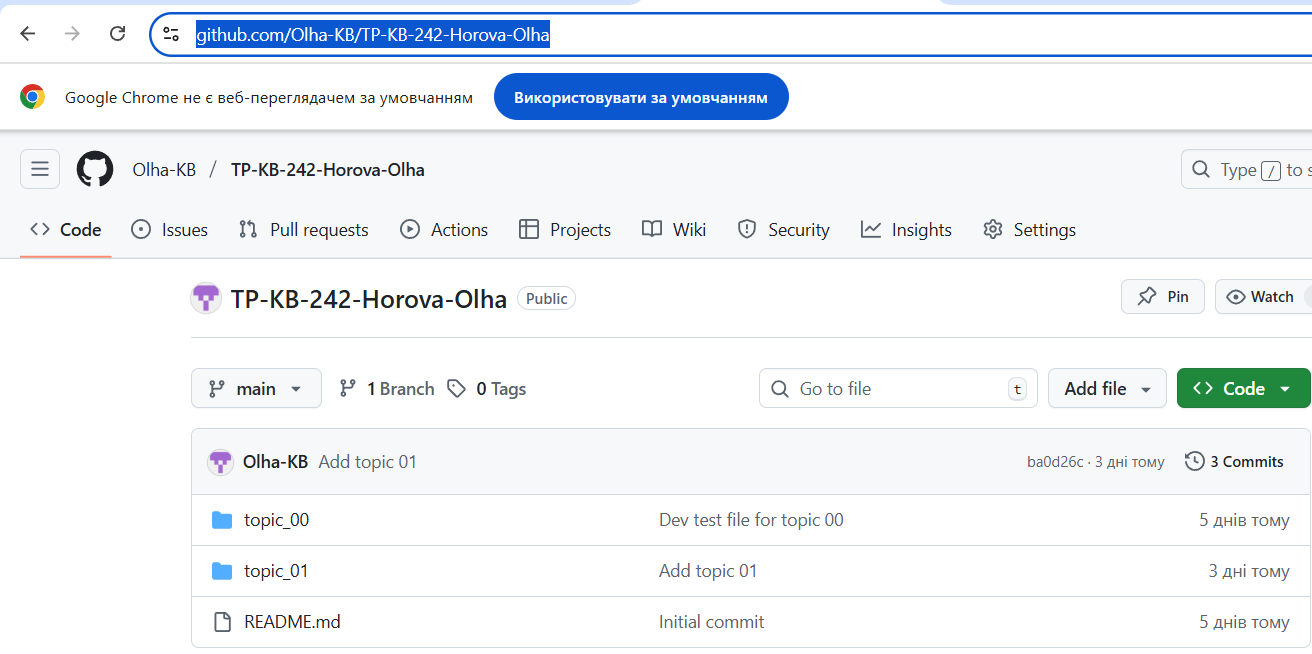
Хід виконання завдання: Створила функцію discr(a, b, c) для обчислення дискримінанта за формулою: D = b\* b - 4ac. При введенні коефіцієнтів a, b, c через input(), отримані значення перетворюються на цілі числа за допомогою int(). Функція повертає результат обчислення, який виводиться на екран.

Текст програми:

|  |
| --- |
| def discr(a, b, c):      return b \*b - 4 \* a \* c  a = int(input("What's a: "))  b = int(input("What's b: "))  c = int(input("What's c: "))  D = discr(a, b, c)  print("Discriminant (D) =", D) |

Посилання на github:  <https://github.com/Olha-KB/TP-KB-242-Horova-Olha>

Знімок екрану з посилання на github:



Звіт до Теми №2

Умовний перехід

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

**Пошук коренів квадратного рівняння**

Необхідно написати функцію пошуку коренів квадратного рівняння використовуючи функцію розрахунку дискримінанту з попередньої теми та умовні переходи.

Хід виконання завдання: Спочатку створено функцію discr(a, b, c) для обчислення дискримінанта. Далі реалізовано функцію coreny(a, b, D), яка за допомогою умовних операторів визначає кількість коренів. Якщо дискримінант більше нуля — обчислюються два корені, якщо дорівнює нулю — один, а якщо менше нуля — виводиться повідомлення про відсутність дійсних коренів. Результат виводиться на екран.

Текст програми:

|  |
| --- |
| import math  def discr(a, b, c):      return b \* b - 4 \* a \* c  a = int(input("What's a: "))  b = int(input("What's b: "))  c = int(input("What's c: "))  D = discr(a, b, c)  print("Discriminant (D) =", D)  def coreny(a, b, D):      if D > 0:          x1 = (-b - math.sqrt(D)) / (2 \* a)          x2 = (-b + math.sqrt(D)) / (2 \* a)          print("x1 =", x1)          print("x2 =", x2)      elif D == 0:          x = -b / (2 \* a)          print("x =", x)      else:          print("Коренів немає")  coreny(a, b, D) |

**Програма калькулятор**

Необхідно написати програму калькулятор використовуючи if else конструкцію. Кожна операція має бути виконана в окремій функції.

Хід виконання завдання: Було створено програму-калькулятор з окремими функціями для додавання, віднімання, множення та ділення. Передбачено перевірку ділення на нуль. Користувач вводить два числа і знак операції, програма виконує дію та виводить результат, або повідомляє про помилку при некоректному вводі.

Текст програми:

|  |
| --- |
| def plus(a, b):      return a + b  def minus(a, b):      return a - b  def multiply(a, b):      return a \* b  def divide(a, b):      if b != 0:          return a / b      else:          return "Помилка: ділення на нуль!"  a = int(input("Введіть перше число: "))  b = int(input("Введіть друге число: "))  op = input("Введіть дію (+, -, \*, /): ")  if op == "+":      print("Результат:", plus(a, b))  elif op == "-":      print("Результат:", minus(a, b))  elif op == "\*":      print("Результат:", multiply(a, b))  elif op == "/":      print("Результат:", divide(a, b))  else:      print("Помилка: невідома операція!") |

**Програма калькулятор через match**

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

Хід виконання завдання: Програма-калькулятор реалізована через match-case з окремими функціями для кожної операції, передбачено перевірку ділення на нуль та обробку невідомих дій.

Текст програми:

|  |
| --- |
| def plus(a, b):      return a + b  def minus(a, b):      return a - b  def multiply(a, b):      return a \* b  def divide(a, b):      if b != 0:          return a / b      else:          return "Помилка: ділення на нуль!"    a = int(input("Введіть перше число: "))  b = int(input("Введіть друге число: "))  op = input("Введіть дію (+, -, \*, /): ")  match op:      case "+":          print("Результат:", plus(a, b))      case "-":          print("Результат:", minus(a, b))      case "\*":          print("Результат:", multiply(a, b))      case "/":          print("Результат:", divide(a, b))      case \_:          print("Помилка: невідома операція!") |

Звіт до Теми №3

Цикли

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

**Програма калькулятор**

Необхідно 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.

Хід виконання завдання: У програмі реалізовано калькулятор з чотирма функціями: додавання, віднімання, множення та ділення (з перевіркою ділення на нуль). Основна робота виконується в циклі while True, де користувач вводить операцію або команду stop для завершення. Використано конструкції if/elif/else для вибору дії, try/except для перевірки введення чисел та оператор break для виходу з циклу.

Текст програми:

|  |
| --- |
| def plus(a, b):      return a + b  def minus(a, b):      return a - b  def multiply(a, b):      return a \* b  def divide(a, b):      if b != 0:          return a / b      else:          return "Помилка: ділення на нуль!"  print("Калькулятор. Для завершення введіть команду 'stop'.")  while True:      op = input("Введіть дію (+, -, \*, /) або 'stop' для виходу: ")      if op == "stop":          print("Завершення програми.")          break  # вихід із циклу      try:          a = int(input("Введіть перше число: "))          b = int(input("Введіть друге число: "))      except ValueError:          print("Помилка: потрібно вводити числа!")          continue      if op == "+":          print("Результат:", plus(a, b))      elif op == "-":          print("Результат:", minus(a, b))      elif op == "\*":          print("Результат:", multiply(a, b))      elif op == "/":          print("Результат:", divide(a, b))      else:          print("Помилка: невідома операція!") |

**Програма тестування функцій списків**

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

Хід виконання завдання: цій програмі використано методи списків:

append() — додає елемент у кінець списку,

extend() — додає кілька елементів одразу,

insert() — вставляє елемент у задану позицію,

remove() — видаляє елемент за значенням,

copy() — створює копію списку,

sort() — сортує список,

reverse() — змінює порядок елементів на зворотний,

clear() — очищає список.

Текст програми:

|  |
| --- |
| def testing():      list = ["A", "B", "C"]      print("Початковий список:", list)      list.append("D")      print("append:", list)      list.extend(["E", "F"])      print("extend:", list)      list.insert(1, "X")      print("insert:", list)      list.remove("B")      print("remove:", list)      copy\_list = list.copy()      print("Копія списку:", copy\_list)      list.sort()      print("sort:", list)      list.reverse()      print("reverse:", list)      list.clear()      print("clear:", list)  testing() |

**Програма тестування функцій словників**

Необхідно написати програму тестування функцій словників таких як: update(), del(), clear(), keys(), values(), items().

Хід виконання завдання: застосовано операції над словниками:

update() — оновлення значення існуючого ключа та додавання нового,

del — оператор для видалення елемента за ключем,

copy() — створення копії словника,

clear() — повне очищення словника,

keys() — отримання списку ключів,

values() — отримання значень,

items() — отримання пар ключ–значення.

Текст програми:

|  |
| --- |
| def testing():      original = {          "A": 1,          "B": 2,          "C": 3,          "D": 4,          "F": 5      }        print("Початковий словник:", original)      original.update({"B": 20, "E": 6})      print("update:", original)      del original["C"]      print("del original:", original)      temp\_dict = original.copy()      temp\_dict.clear()      print("clear:", temp\_dict)      print("Ключі словника:", original.keys())      print("Значення словника:", original.values())      print("Пари ключ-значення:", original.items())  testing() |

**Функція пошуку**

Необхідно написати функцію пошуку позиції для вставки нового елементу в список.

Хід виконання завдання: У цій функції використано:

цикл for з enumerate() — для проходження по списку з індексами, оператор if — для перевірки умови порівняння, return — для повернення знайденої позиції, len() — для визначення довжини списку, якщо елемент потрібно вставити в кінець.

Текст програми:

|  |
| --- |
| def find\_position(sorted\_list, new\_element):      for i, item in enumerate(sorted\_list):          if new\_element <= item:              return i      return len(sorted\_list)  my\_list = ["A", "B", "C", "D", "E", "F"]  new\_elem = "D"  position = find\_position(my\_list, new\_elem)  print(f"Список: {my\_list}")  print(f"Нова позиція для '{new\_elem}': {position}") |

Звіт до Теми №4

Виняткові ситуації

Під час виконання практичного завдання до Теми №4 було надано варіанти рішення до наступних задач:

**Розширення калькулятора**

Необхідно розширити програму калькулятор функцією запитів даних для виконання операцій від користувача, що обробляє виняткові ситуації. Розширити функцію ділення обробкою виняткової ситуації ділення на нуль

Хід виконання завдання: розширено функцію ділення, додавши обробку виняткової ситуації ZeroDivisionError для випадку ділення на нуль через try/except. Також реалізовано обробку помилки ValueError під час введення чисел, щоб уникнути аварійного завершення програми при некоректному вводі.

Текст програми:

