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

Звіт до Теми №1

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

1. Обернути рядок в зворотному порядку.
2. Виконати тестування функцій, що працюють з рядками: strip(), capitalize(), title(), upper(), lower().
3. Написати функцію пошуку дискримінанту квадратного рівняння.

Хід виконання завдання:

1) Завдання: Обернути рядок в зворотному порядку.

Під час виконання завдання спочатку потрібно отримати рядок від користувача за допомогою функції input(), яка дозволяє ввести текст із клавіатури. Далі, для обертання рядка, використовується спеціальний синтаксис зрізу — [::-1]. Цей метод нарізки рядка бере всі символи рядка, починаючи з кінця до початку, що ефективно обертає його. Після цього результат оберненого рядка зберігається в новій змінній reversed\_line. Нарешті, обернений рядок виводиться на екран за допомогою функції print().

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

Під час виконання завдання спочатку необхідно застосувати метод strip(), який видаляє вказані символи з початку та кінця рядка. Наприклад, у рядку ???hello andrii???? метод видаляє всі знаки питання, залишаючи лише текст без них. Далі, використовується метод capitalize(), який змінює першу літеру рядка на велику, а всі інші робить маленькими, як у випадку з рядком hello andri, що стає Hello andri. Наступним кроком є застосування методу title(), який робить перші літери кожного слова великими, наприклад, перетворюючи hello andri на Hello Andri. Потім використовується метод upper(), який переводить всі літери в рядку у верхній регістр, тобто hello andri перетворюється на HELLO ANDRI. Нарешті, метод lower() перетворює всі літери рядка на маленькі, навіть якщо вони раніше були великими, як у випадку з рядком heSSSSSlo aEFSdri, який стає hesssslo aefsdri.

1. Завдання: Написати функцію пошуку дискримінанту квадратного рівняння.

Під час виконання завдання спочатку визначається функція findD(a, b, c), яка обчислює дискримінант квадратного рівняння за формулою D=b2−4acD = b^2 - 4acD=b2−4ac. Ця функція приймає три параметри: коефіцієнти aaa, bbb і ccc, які вводяться користувачем через функцію input(). Після введення значень ці параметри перетворюються на тип int. Далі, функція викликається з введеними значеннями, і обчислений дискримінант зберігається в змінній result. Нарешті, результат виводиться на екран, що дозволяє користувачу бачити значення дискримінанту, яке вказує на кількість розв'язків квадратного рівняння.

Текст програми:

1) Завдання:

# Отримати рядок від користувача

line = input(" line: ")

# Обернути рядок

reversed\_line = line[::-1]

# Вивести обернутий рядок

print("test:", reversed\_line)

2) Завдання:

text = "???hello andrii????"

print(text.strip("?"))   # strip видаляє символи з початку рядка і кінця

text = "hello andri"

print(text.capitalize())   # capitalize виправляє першу букву на велику всі остальні маленькі тільки перша

text = "hello andri"

print(text.title()) # title початок слова з великої літери  не тільки діє через пробіл а й через інші символи

text = "hello andri"

print(text.upper())  # upper всі літери з великої

text = "heSSSSSlo aEFSdri"

print(text.lower())  # lower всі літери пишуться з маленької

3) Завдання:

def findD(a, b, c):

    D = b\*\*2 - 4\*a\*c

    return D

a = int(input("Please enter start point: "))

b = int(input("Please end point: "))

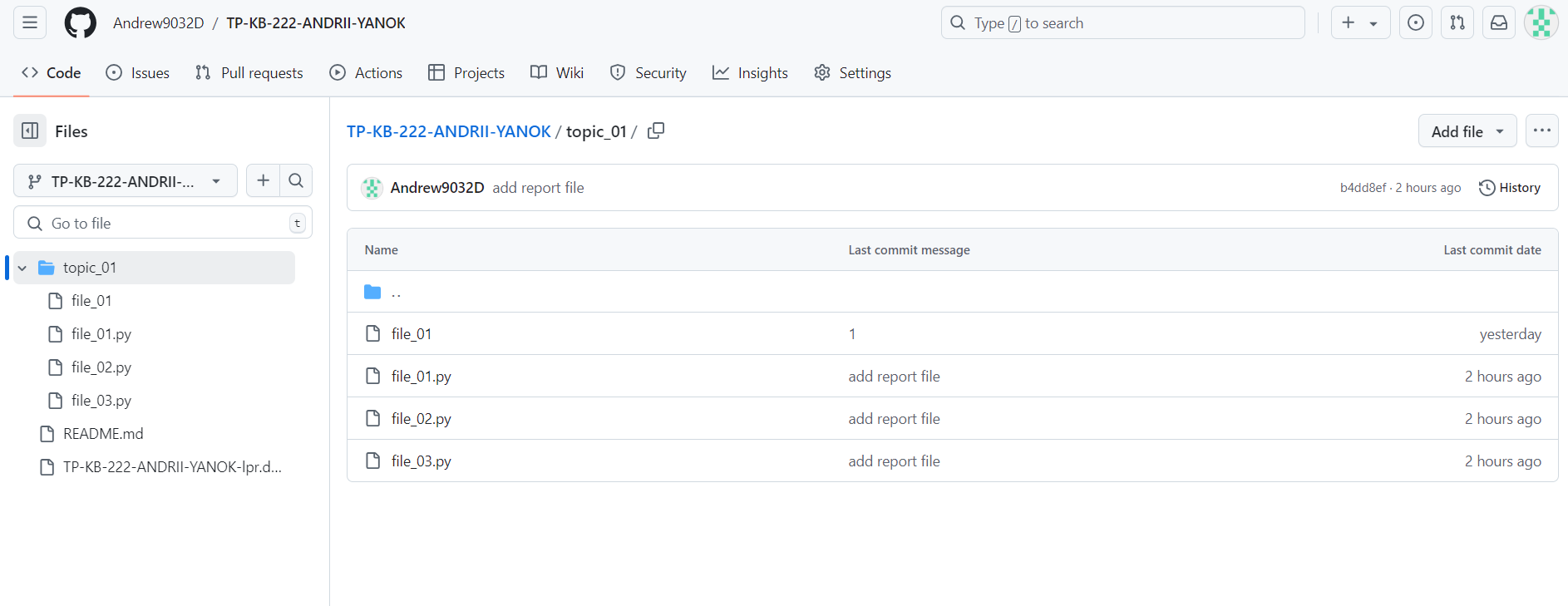
c = int(input("Please enter mult: "))

result = findD(a, b, c)

print("Discriminant (D):", result)

Посилання на github: <https://github.com/Andrew9032D/TP-KB-222-ANDRII-YANOK.git>

Знімок екрану з посилання на github:



Звіт до Теми №2

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

1. Написати функцію пошуку коренів квадратного рівняння використовуючи функцію розрахунку дискримінанту з попередньої теми та умовні переходи.
2. Написати програму калькулятор використовуючи **if else** конструкцію. Кожна операція має бути виконана в окремій функції.
3. Написати програму калькулятор використовуючи **match** конструкцію. Кожна операція має бути виконана в окремій функції.

Хід виконання завдання:

1) Завдання: Написати функцію пошуку коренів квадратного рівняння використовуючи функцію розрахунку дискримінанту з попередньої теми та умовні переходи.

Програма для пошуку коренів квадратного рівняння працює таким чином. Спочатку користувач вводить значення коефіцієнтів aaa, bbb та ccc через функцію input(), що дозволяє вводити дані з клавіатури. Введені значення перетворюються на числа типу float для подальших математичних обчислень. Потім обчислюється дискримінант за допомогою функції discrim(a, b, c), яка реалізує формулу D=b2−4acD = b^2 - 4acD=b2−4ac. Дискримінант визначає кількість і тип коренів квадратного рівняння: якщо дискримінант більший за нуль, рівняння має два різних корені, які обчислюються за допомогою формул x1=−b+D2ax\_1 = \frac{-b + \sqrt{D}}{2a}x1​=2a−b+D​​ та x2=−b−D2ax\_2 = \frac{-b - \sqrt{D}}{2a}x2​=2a−b−D​​. Якщо дискримінант дорівнює нулю, рівняння має один корінь, що обчислюється за формулою x=−b2ax = \frac{-b}{2a}x=2a−b​. Якщо дискримінант менший за нуль, коренів немає. Залежно від значення дискримінанту, програма виводить відповідне повідомлення: два корені, один корінь або відсутність коренів. Це дозволяє користувачеві швидко отримати інформацію про розв'язок квадратного рівняння.

Код:

'''1) Написати функцію пошуку коренів квадратного рівняння

використовуючи функцію розрахунку дискримінанту з попередньої теми та

умовні переходи.'''

import math

def discrim(a, b, c):

    return b\*\*2-4\*a\*c

def roots(a, b, c):

    d = discrim(a, b, c)

    if d > 0:

      x1 = (-b + math.sqrt(d)) / (2 \* a)

      x2 = (-b - math.sqrt(d)) / (2 \* a)

      print("Два корені: x1 =", x1, ", x2 =", x2)

    elif d == 0:

       x = -b / (2 \* a)

       print("Один корінь: x =", x)

    else:

     print("Коренів немає")

a = float(input('a = '))

b = float(input('b = '))

c = float(input('c = '))

result = roots(a, b, c)

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

Програма калькулятора починає свою роботу з того, що отримує два числа від користувача за допомогою функції input(), яка дозволяє ввести значення з клавіатури. Для того щоб ці значення можна було використовувати в математичних операціях, вони конвертуються у цілі числа за допомогою функції int(). Потім програма просить користувача вибрати математичну операцію, яка може бути однією з таких: додавання, віднімання, множення або ділення. Операція вводиться як рядок, який в подальшому використовується для визначення, яку саме функцію викликати.

Для кожної з можливих операцій є окрема функція. Функція addition виконує додавання двох чисел, функція subtraction віднімає одне число від іншого, функція multiplication множить два числа, а функція division ділить одне число на інше, при цьому перевіряється, чи не є дільник нулем, щоб уникнути помилки ділення на нуль. Якщо користувач вводить недійсну операцію, програма виводить повідомлення про помилку.

Виконання операції базується на конструкціях if-elif-else. Спочатку програма перевіряє, яку саме операцію вибрав користувач, і викликає відповідну функцію. Якщо введена операція не є коректною, то програма виводить повідомлення про помилку. Кожен результат операції виводиться на екран за допомогою функції print(). Таким чином, програма виконує математичні операції в залежності від вибору користувача.

Код:

'''2)   Написати програму калькулятор використовуючи if else конструкцію.

Кожна операція має бути виконана в окремій функції.'''

def addition(a, b):

    return a + b

def subtrction(a, b):

    return a - b

def multiplication(a, b):

    return a \* b

def division(a, b): # коли ділення на 0 помилка

   if b == 0:

       return "error"

   return a / b

a = int(input("a = "))

b = int(input("b = "))

operation = input ("What's operation [ + - \* /]")

if operation == "+":

    result = addition(a, b)

    print (result)

elif operation == "-":

    result = subtrction(a, b)

    print (result)

elif operation == "\*":

 result = multiplication(a, b)

 print (result)

elif operation == "/":

    result = division(a, b)

    print (result)

else:

    print ("eror")

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

Цей код реалізує калькулятор, який виконує основні математичні операції (додавання, віднімання, множення, ділення) за допомогою конструкції match. Програма працює таким чином: спочатку запитує у користувача два числа, потім запитує, яку операцію потрібно виконати, і, залежно від вибору користувача, виконується відповідна операція.

У коді є чотири функції: addition (додавання), subtrction (віднімання — є помилка в назві функції, правильніше було б subtraction), multiplication (множення) та division (ділення). Остання функція також перевіряє, чи не намагаються поділити на нуль, у разі чого виводиться повідомлення про помилку.

Після того, як користувач вводить значення для двох чисел та вибір операції, конструкція match перевіряє введену операцію і викликає відповідну функцію для її виконання. Якщо введено некоректну операцію, виводиться повідомлення про помилку.

Код:

'''3) Написати програму калькулятор використовуючи match конструкцію.

Кожна операція має бути виконана в окремій функції.'''

def addition(a, b):

    return a + b

def subtrction(a, b):

    return a - b

def multiplication(a, b):

    return a \* b

def division(a, b): # коли ділення на 0 помилка

   if b == 0:

      return "error"

   return a / b

a = int(input("a = "))

b = int(input("b = "))

operation = input ("What's operation [ + - \* /]")

match operation:

 case  "+":

    result = addition(a, b)

    print (result)

 case  "-":

    result = subtrction(a, b)

    print (result)

 case  "\*":

  result = multiplication(a, b)

  print (result)

 case "/":

    result = division(a, b)

    print (result)

 case \_:

  print("eror")

Посилання на github: <https://github.com/Andrew9032D/TP-KB-222-ANDRII-YANOK.git>

Знімок екрану з посилання на github:

Звіт до Теми №3

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

1. 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.
2. Написати програму тестування функцій списків таких як: extend(), append(), insert(id, val), remove(val), clear(), sort(), reverse(), copy().
3. Написати програму тестування функцій словників таких як: update(), del(), clear(), keys(), values(), items().
4. Маючи відсортований список, написати функцію пошуку позиції для вставки нового елементу в список.

Хід виконання завдання:

1. Завдання: 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.

Цей код реалізує калькулятор, який постійно запитує користувача для введення нових чисел і операцій, доки не буде введена команда для завершення програми. Спочатку визначені функції для основних арифметичних операцій: додавання, віднімання, множення та ділення. Кожна з цих функцій приймає два аргументи і повертає результат відповідної операції. Для ділення додано перевірку на нуль, щоб уникнути помилки ділення на нуль, у такому випадку повертається рядок "error". Далі програма знаходиться в нескінченному циклі, де на кожному етапі запитує користувача для введення першого числа, другого числа і операції. Якщо користувач вводить "exit", програма завершується. Якщо обрана правильна операція, викликається відповідна функція, і результат обчислення виводиться на екран. Якщо операція введена невірно, виводиться повідомлення про помилку. Таким чином, програма дозволяє виконувати нескінченну кількість арифметичних операцій до введення команди для завершення.

Код:

'''1) 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 addition(a, b):

    return a + b

def subtrction(a, b):

    return a - b

def multiplication(a, b):

    return a \* b

def division(a, b): # коли ділення на 0 помилка

   if b == 0:

     return "error"

   return a / b

while True:

 a = input("enter first number (or 'exit' to quit): ")

 if a == "exit":

    break

 b = input("enter second number: ")

 operation = input ("What's operation [ + - \* /]")

 a = float(a)

 b = float(b)

 if operation == "+":

    result = addition(a, b)

    print (result)

 elif operation == "-":

    result = subtrction(a, b)

    print (result)

 elif operation == "\*":

  result = multiplication(a, b)

  print (result)

 elif operation == "/":

    result = division(a, b)

    print (result)

 else:

    print ("eror")

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

Спочатку створюються два списки: fruits (фрукти) та vegetables (овочі). Потім застосовуються різні методи для маніпулювання списками. Метод append() додає елемент в кінець списку. У цьому випадку, до списку fruits додається елемент "orange ". Метод extend() об'єднує два списки, додаючи елементи з одного списку до іншого. Після цього список fruits містить елементи як з оригінального списку, так і з списку vegetables. Метод insert(id, val) дозволяє вставити елемент за вказаним індексом. У прикладі елемент "kiwi" вставляється на друге місце в списку fruits. Метод remove(val) видаляє перше входження елемента з вказаним значенням. У цьому випадку, з списку fruits видаляється елемент "banana". Метод sort() сортує список за алфавітним порядком для рядків. Після його виконання елементи списку fruits впорядковуються. Метод reverse() змінює порядок елементів у списку на зворотний. Після цього елементи списку йдуть у зворотному порядку. Метод copy() створює поверхневу копію списку, тобто новий список, що містить ті ж елементи, але з ним можна працювати окремо від оригінального списку. Останній метод — clear() — видаляє всі елементи зі списку, роблячи його порожнім.

Код:

'''2)   Написати програму тестування функцій списків таких як: extend(), append(),

insert(id, val), remove(val), clear(), sort(), reverse(), copy()'''

fruits = ["apple", "banana", "cherry"]

vegetables = ["carrot", "potato"]

# append()

fruits.append("orange ") # додає в кінець списка

print(fruits)

# extend()

fruits.extend(vegetables) # поєднує два списки

print(fruits)

# insert()

fruits.insert(1, "kiwi") # за індексом вставляє в список елемент

print(fruits)

# remove()

fruits.remove("banana") # видаляє обєкт за індексом

print(fruits)

# sort()

fruits.sort()# сортує список

print(fruits)

# reverse()

fruits.reverse() # в іншому порядку перевертає список

print(fruits)

# copy()

new\_fruits = fruits.copy() # робить копію

print(new\_fruits)

# clear()

fruits.clear() # видаляє всі обєкти зі списку роблячи порожнім

print(fruits)  # []

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

Спочатку створюється словник person, який містить інформацію про особу: її ім'я, вік і місто. Першим методом використовується update(), який дозволяє оновлювати існуючі значення в словнику або додавати нові пари ключ-значення. У даному випадку метод оновлює значення ключа "age" на 26 та додає новий ключ "job" із значенням "Engineer". Далі використовується оператор del, який дозволяє видаляти пари ключ-значення зі словника. У прикладі видаляється ключ "city" разом із його значенням.

Метод keys() дозволяє переглянути всі ключі, що є в словнику. Після його виконання виводяться всі ключі зі словника. Метод values() виводить усі значення зі словника. Метод items() виводить усі пари ключ-значення у вигляді кортежів, що зручно для ітерацій по словнику. Нарешті, метод clear() повністю очищає словник, видаляючи всі пари ключ-значення, що робить словник порожнім.

'''3)   Написати програму тестування функцій словників таких як:

 update(), del(), clear(), keys(), values(), items()'''

# словник

person = {"name": "Alice", "age": 25, "city": "New York"}

# Оновлення словника

person.update({"age": 26, "job": "Engineer"})

print(person)

# Видалення пари ключ-значення

del person["city"]

print(person)

# Перегляд всіх ключів

print(person.keys())

# Перегляд всіх значень

print(person.values())

# Перегляд всіх пар ключ-значення

print(person.items())

# Очистка словника

person.clear()

print(person)

4) Завдання: Маючи відсортований список, написати функцію пошуку позиції для вставки нового елементу в список.

Функція find\_insert\_position() приймає два аргументи: відсортований список sorted\_list та новий елемент new\_element. Вона перебирає всі елементи списку, порівнюючи кожен елемент з новим. Коли знайдено перший елемент, більший за новий елемент, цикл переривається, і повертається індекс позиції для вставки нового елемента.

Функція insert\_and\_print() спочатку викликає функцію find\_insert\_position(), щоб знайти позицію для вставки нового елемента. Після цього за допомогою методу insert() новий елемент вставляється на знайдену позицію. Після цього виводиться оновлений список.

У прикладі, даний список sorted\_list містить елементи [1, 3, 4, 7, 9, 10], а новий елемент — це 11. Функція знаходить, що новий елемент повинен бути вставлений в кінці списку, тому після виконання програми результат буде: Updated list: [1, 3, 4, 7, 9, 10, 11].

'''4)   Маючи відсортований список, написати функцію пошуку

позиції для вставки нового елементу в список.'''

def find\_insert\_position(sorted\_list, new\_element): # пошук потрібної позиції

    position = 0

    for element in sorted\_list: # перебір всіх елементів

        if new\_element < element:

            break

        position += 1

    return position

def insert\_and\_print(sorted\_list, new\_element):

    # Знаходимо позицію для вставки

    position = find\_insert\_position(sorted\_list, new\_element)

    # Вставляємо новий елемент на знайдену позицію

    sorted\_list.insert(position, new\_element)

    # Виводимо оновлений список

    print("Updated list:", sorted\_list)

sorted\_list = [1, 3, 4, 7, 9, 10]

new\_element = 11

insert\_and\_print(sorted\_list, new\_element)

Посилання на github: <https://github.com/Andrew9032D/TP-KB-222-ANDRII-YANOK.git>

Знімок екрану з посилання на github: