

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:

Я вирішив трохи ускладнити собі задачу, і зробив можливість вводу будь якого тексту, але з умовою, що цей текст буде мати 10 символів, якщо ні, то спрацюють обробники помилки і скажуть більше треба, чи менше, якщо ж текст має 10 символів, то за допомогою слайсів, де можна вибрати з якого елементу масива почати, де зупинитись і який крок, в моєму випадку я зазначив лише який крок -1, щоб він йшов з кінця.

Текст програми:

```
text = input("Введіть текст з 10 знаками: ")
```

```
if len(text) > 10:
```

```
    print("Знаків більше ніж 10")
```

```
elif len(text) < 10:
```

```
    print("Знаків менше ніж 10")
```

```
else:
```

```
reserved = text[::-1]
```

```
print(reserved)
```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar titled 'Files' with a dropdown set to 'main'. Below it are search and filter buttons. Under 'topic_01', there are four files: 'Task1.py' (selected), 'Task2.py', 'Task3.py', and 'TP-CS-241-Puhoviy-Serhiy.pdf'. On the right, the main area shows the code for 'Task1.py'. The code is as follows:

```
text = input("Введіть текст з 10 знаками: ")
if len(text) > 10:
    print("Знаків більше ніж 10")
elif len(text) < 10:
    print("Знаків менше ніж 10")
else:
    reserved = text[::-1]
```

Рис.1 Скріншот першого завдання з GitHub

Посилання:https://github.com/sasegas/TP-CS-241-Puhoviy-Serhiy/blob/main/topic_01/Task1.py

Стилі для тексту

Виконати деякі тести на strip, capitalize, title, upper, lower

Хід виконання завдання:

В цьому завданні я написав текст в змінній string, щоб його можна було перевірити на всі вищезазначені функції,

strip - прибирає зайві пробіли на початку і в кінці,

capitalize - в рядку тільки перша літера, першого слова велика,

title - в кожному слові великі літери, тільки перші, а інші в нижньому регістрі,

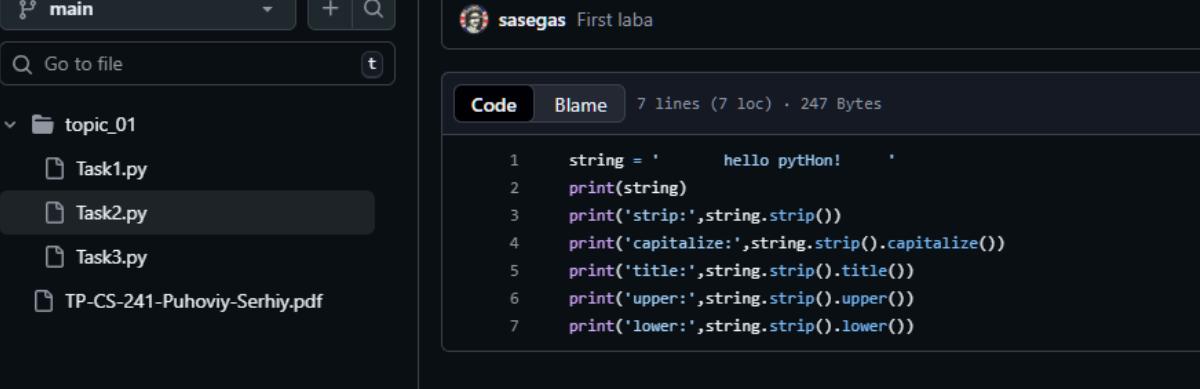
upper - всі букви у верхньому регістрі,

lower - всі букви у нижньому регістрі

Текст програми:

```
string = '    hello pytHon!    '
```

```
print(string)
print('strip:',string.strip())
print('capitalize:',string.strip().capitalize())
print('title:',string.strip().title())
print('upper:',string.strip().upper())
print('lower:',string.strip().lower())
```



```
string = 'hello python!'
print(string)
print('strip:',string.strip())
print('capitalize:',string.strip().capitalize())
print('title:',string.strip().title())
print('upper:',string.strip().upper())
print('lower:',string.strip().lower())
```

Рис.2 Скріншот другого завдання з GitHub

Посилання:https://github.com/sasegas/TP-CS-241-Puhoviy-Serhiy/blob/main/topic_01/Task2.py

Квадратне рівняння

Знайти відповідь для квадратного рівняння за допомогою функцій

Хід виконання завдання:

Спочатку я створив функцію quadraticFn в якій задав 3 змінні, a,b,c відповідно як у рівнянні і після цього задав змінну дискримінанта, де його обрахував відповідно до формули і після цього використав умовний оператор if, для того щоб правильно порахувати x, після чого ми маємо три варіанти відповіді, або x не існує, або є тільки один x, або є x1 і x2. Для того щоб порахувати квадратний корінь дискримінанту мені знадобилася бібліотека math.

Текст програми:

```
import math

def quadraticFn():

    a = int(input('Введіть перший коефіцієнт:'))

    b = int(input('Введіть другий коефіцієнт:'))

    c = int(input('Введіть вільний член:'))

    discriminator = b**2 - 4*a*c

    if discriminator == 0:

        print ("Рівняння має один корінь")

        x = -b/(2*a)

        print(x)

    elif discriminator > 0:

        print ("Рівняння має два корінь")

        x1 = (-b - math.sqrt(discriminator))/(2*a)

        x2 = (-b + math.sqrt(discriminator))/(2*a)

        print('x1=',x1)

        print('x2=',x2)

    else:

        print("Рівняння не має дійсних коренів")

quadraticFn()
```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a folder icon labeled 'topic_01' containing files: 'Task1.py', 'Task2.py', 'Task3.py' (which is highlighted), and a PDF file 'TP-CS-241-Puhoviy-Serhiy.pdf'. The main area is titled 'Code' and shows the content of 'Task2.py'. The code is a Python script for solving quadratic equations. It imports the 'math' module, defines a function 'quadraticFn()', and uses it to calculate discriminant values. It then prints the results based on the discriminant's value: one root if zero, two roots if positive, or no real roots if negative.

```
1 import math
2
3 def quadraticFn():
4     a = int(input('Введіть перший коефіцієнт:'))
5     b = int(input('Введіть другий коефіцієнт:'))
6     c = int(input('Введіть вільний член:'))
7     discriminator = b**2 - 4*a*c
8
9     if discriminator == 0:
10        print ("Рівняння має один корінь")
11        x = -b/(2*a)
12        print(x)
13    elif discriminator > 0:
14        print ("Рівняння має два коріні")
15        x1 = (-b - math.sqrt(discriminator))/(2*a)
16        x2 = (-b + math.sqrt(discriminator))/(2*a)
17        print('x1=',x1)
18        print('x2=',x2)
19    else:
20        print("Рівняння не має дійсних коренів")
```

Рис.3 Скріншот третього завдання з GitHub

Посилання:https://github.com/sasegas/TP-CS-241-Puhoviy-Serhiy/blob/main/topic_01/Task2.py

Звіт до Теми №2

Умовний перехід

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

Функція пошуку коренів

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

Хід виконання завдання:

Використав код для знаходження дискримінанта з попередньої теми, який я додав, після нього я написав функцію, для розрахунку дискримінанта, за допомогою умовних операторів if, elif, else, я додав три умови по яким і знаходяться корені.

Текст програми:

```
import math

def discriminant(a,b,c):
    return b**2 - 4*a*c

def quadratic_roots(a, b, c):
    d = discriminant(a, b, c)

    if d > 0:
        print("Рівняння має два корені")
        x1 = (-b - math.sqrt(d)) / (2*a)
        x2 = (-b + math.sqrt(d)) / (2*a)
        print("x1 =", x1)
```

```

        print("x2 =", x2)

    return x1, x2

elif d == 0:

    print("Рівняння має один корінь")

    x = -b / (2*a)

    print("x =", x)

    return x,

else:

    print("Рівняння не має дійсних коренів")

    return None

```

a, b, c = 3, -18, 27

roots = quadratic_roots(a, b, c)

The screenshot shows a code editor interface with a file tree on the left and a code editor window on the right.

File Tree:

- main
- lab_01
- topic_01
- topic_02
 - Task1.py
 - Task2.py
 - Task3.py
- TP-CS-241-Puhoviy-Serhiy.pdf

Code Editor Window:

File: Task1.py (sasegas Second topic)

Code Blame 29 lines (22 loc) · 666 Bytes

```

1 import math
2
3 def discriminant(a,b,c):
4     return b**2 - 4*a*c
5
6
7
8 def quadratic_roots(a, b, c):
9     d = discriminant(a, b, c)
10
11    if d > 0:
12        print("Рівняння має два корені")
13        x1 = (-b - math.sqrt(d)) / (2*a)
14        x2 = (-b + math.sqrt(d)) / (2*a)
15        print("x1 =", x1)
16        print("x2 =", x2)
17        return x1, x2
18    elif d == 0:
19        print("Рівняння має один корінь")
20        x = -b / (2*a)
21        print("x =", x)
22        return x,
23    else:
24        print("Рівняння не має дійсних коренів")
25        return None
26
27
28 a, b, c = 3, -18, 27
29 roots = quadratic_roots(a, b, c)

```

Рис.1 Скріншот першого завдання з GitHub

Калькулятор використовуючи if

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

Хід виконання завдання:

В цьому завданні я спочатку додав `input` для кожного значення, а саме першого числа, другого і знаку, потім створив функцію калькулятор, яки за допомогою умовних операторів робить певну дію, чи то множення, ділення, чи інші, а потім значення цієї функції поміщається в змінну `result`, яку, потім показую за допомогою `print`

Текст програми:

```
a = float(input("введіть перше число: "))

b = float(input("введіть друге число: "))

sign = input("введіть дію(+ - * /): ")

def calculator (a, b, sign):

    if sign == "+":
        return a + b

    elif sign == "-":
        return a - b

    elif sign == "*":
        return a * b

    elif sign == "/":
        if b != 0:
            return a / b
        else:
            return "Ділити на 0 неможна"
    else:
```

```

        return "Була задана некоректна дія"

result = calculator(a,b,sign)
print("Відповідь: ", result)

```

```

1  a = float(input("введіть перше число: "))
2  b = float(input("введіть друге число: "))
3  sign = input("введіть дію(+ - * /): ")
4
5  def calculator (a, b, sign):
6      if sign == "+":
7          return a + b
8      elif sign == "-":
9          return a - b
10     elif sign == "*":
11         return a * b
12     elif sign == "/":
13         if b != 0:
14             return a / b
15         else:
16             return "Ділити на 0 неможна"
17     else:
18         return "Була задана некоректна дія"
19
20 result = calculator(a,b,sign)
21 print("Відповідь: ", result)

```

Рис.2 Скріншот другого завдання з GitHub

Калькулятор використовуючи match

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

Хід виконання завдання:

В цьому завданні я зробив 4 окремі функції, які відповідають за конкретну дію і за допомогою match запускає певну дію в залежності від вибору користувача, а потім виводив

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 "Ділити на 0 неможна"

a = float(input("введіть перше число: "))
b = float(input("введіть друге число: "))
sign = input("введіть дію(+ - * /): ")

match(sign):
    case "+":
        result = plus(a,b)

    case "-":
        result = minus(a,b)

    case "*":
        result = multiply(a,b)

    case "/":
        result = divide(a,b)
    case _:
        result = "Була задана некоректна дія"

print("Відповідь: ", result)
```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a search bar and a list of files: 'lab_01', 'topic_01', 'topic_02', 'Task1.py', 'Task2.py', 'Task3.py' (which is highlighted), and 'TP-CS-241-Puhoviy-Serhiy.pdf'. The main area is titled 'sasegas Second topic' and shows the content of 'Task3.py'. The code is as follows:

```
1  def plus(a,b):
2      return a+b
3  def minus(a,b):
4      return a-b
5  def multiply(a,b):
6      return a*b
7  def divide(a,b):
8      if b!= 0:
9          return a/b
10     else:
11         return "Ділити на 0 неможна"
12
13 a = float(input("введіть перше число: "))
14 b = float(input("введіть друге число: "))
15 sign = input("введіть дію(+ - * /): ")
16
17
18 match(sign):
19     case "+":
20         result = plus(a,b)
21
22     case "-":
23         result = minus(a,b)
24
25     case "*":
26         result = multiply(a,b)
27
28     case "/":
29         result = divide(a,b)
30     case _:
31         result = "Була задана некоректна дія"
32
33
34 print("Відповідь: ", result)
```

Рис.3 Скріншот третього завдання з GitHub

Звіт до Теми №3

Цикли

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

Програма калькулятор з використанням циклів

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:

В цьому завданні я використав минулий код з калькулятором, тільки додав ще цикл while який дає змогу працювати калькулятору, поки його не вимкнуть.

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 "Ділити на 0 не можна"
```

```
print("Щоб вийти, введіть 'exit' замість знака дії.")

while True:

    try:

        a = float(input("Введіть перше число: "))

        b = float(input("Введіть друге число: "))

    except ValueError:

        print("Потрібно вводити числа!")

        continue

    sign = input("Введіть дію (+ - * /) або 'exit' для виходу: ")

    if sign.lower() == "exit":

        print("Роботу завершено.")

        break

    match sign:

        case "+":

            result = plus(a, b)

        case "-":

            result = minus(a, b)

        case "*":

            result = multiply(a, b)

        case "/":

            result = divide(a, b)

        case _:

            result = "Була задана некоректна дія"

    print(result)
```

```
print("Відповідь:", result, "\n")
```

The screenshot shows a GitHub repository interface. The repository name is 'main'. The file tree on the left includes 'lab_01', 'topic_01', 'topic_02', and 'topic_03'. 'topic_03' is expanded, showing files 'Task1.py', 'Task2.py', 'Task3.py', 'Task4.py', and 'TP-CS-241-Puhoviy-Serhiy.pdf'. 'Task1.py' is selected and highlighted. The code in 'Task1.py' is displayed on the right:

```
1  def plus(a, b):
2      return a + b
3
4  def minus(a, b):
5      return a - b
6
7  def multiply(a, b):
8      return a * b
9
10 def divide(a, b):
11     if b != 0:
12         return a / b
13     else:
14         return "Ділити на 0 не можна"
15
16 print("Щоб вийти, введіть 'exit' замість знака дії.")
17
18 while True:
19     try:
20         a = float(input("Введіть перше число: "))
21         b = float(input("Введіть друге число: "))
22     except ValueError:
23         print("Потрібно вводити числа!")
24         continue
25
26     sign = input("Введіть дію (+ - * /) або 'exit' для виходу")
27
```

Рис.1 Скріншот першого завдання з GitHub

Програма тестування функцій списків

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

Хід виконання завдання:

У процесі виконання завдання я ознайомився з основними функціями роботи зі списками в Python. Зокрема, використав метод append() для додавання одного елемента в кінець списку, extend() – для об'єднання списків, insert() – для вставки елемента за вказаним індексом, remove() – для видалення першого входження заданого елемента, clear() – для повного очищення списку, sort() – для сортування елементів у списку, reverse() – для зміни порядку елементів на зворотній, а також copy() – для створення копії списку.

Текст програми:

```
print("==== Тестування методів списків ===")  
  
list = [1, 5, 7]  
  
print('Початковий список: ', list)  
  
list.extend([4, 2])  
  
print('extend([4, 2]): ', list)  
  
list.append(3)  
  
print('append(3): ', list)  
  
list.insert(1,1)  
  
print('insert(1,1): ', list)  
  
list.remove(1)  
  
print('remove(1): ', list)  
  
list.sort()  
  
print('sort(): ', list)  
  
list.reverse()  
  
print('reverse(): ', list)  
  
list_copy = list.copy()  
  
print("copy()): ", list_copy)  
  
list.clear()  
  
print('clear(): ', list)
```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a search bar and a 't' icon. Below it is a tree view of files and folders: 'lab_01', 'topic_01', 'topic_02', and 'topic_03'. 'topic_03' is expanded, showing 'Task1.py', 'Task2.py' (which is highlighted with a grey background), 'Task3.py', 'Task4.py', and 'TP-CS-241-Puhoviy-Serhiy.pdf'. On the right, a code editor window is open. At the top of the editor, there are tabs for 'Code' (which is selected) and 'Blame', followed by statistics: '19 lines (19 loc) · 552 Bytes'. The code itself is a Python script demonstrating various list methods:

```
1 print("== Тестування методів списків ==")
2 list = [1, 5, 7]
3 print('Початковий список: ', list)
4 list.extend([4, 2])
5 print('extend([4, 2]): ', list)
6 list.append(3)
7 print('append(3): ', list)
8 list.insert(1,1)
9 print('insert(1,1): ', list)
10 list.remove(1)
11 print('remove(1): ', list)
12 list.sort()
13 print('sort(): ', list)
14 list.reverse()
15 print('reverse(): ', list)
16 list_copy = list.copy()
17 print("copy()): ", list_copy)
18 list.clear()
19 print('clear()): ', list)
```

Рис.2 Скріншот другого завдання з GitHub

Програма тестування функцій словника

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

Хід виконання завдання:

У процесі виконання завдання я ознайомився з основними функціями роботи зі словниками в Python. Зокрема, використав метод update() для додавання або оновлення пар ключ-значення, оператор del – для видалення ключів зі словника, clear() – для повного очищення словника, keys() – для отримання всіх ключів, values() – для отримання всіх значень, а також items() – для отримання всіх пар ключ-значення.

Текст програми:

```
print("== Тестування методів словників ==")
dict = {'a':1,'b':2,'c':3}
print("Початковий словник: ", dict)
```

```

dict.update({'d':4, 'b':20})

print("update({'d':4, 'b':20}):",dict)

del dict['a']

print("del dict['a']:      ",dict)

print("keys():           ", list(dict.keys()))

print("values():          ", list(dict.values()))

print("items():           ", list(dict.items()))

dict.clear()

print("clear():           ",dict)

```

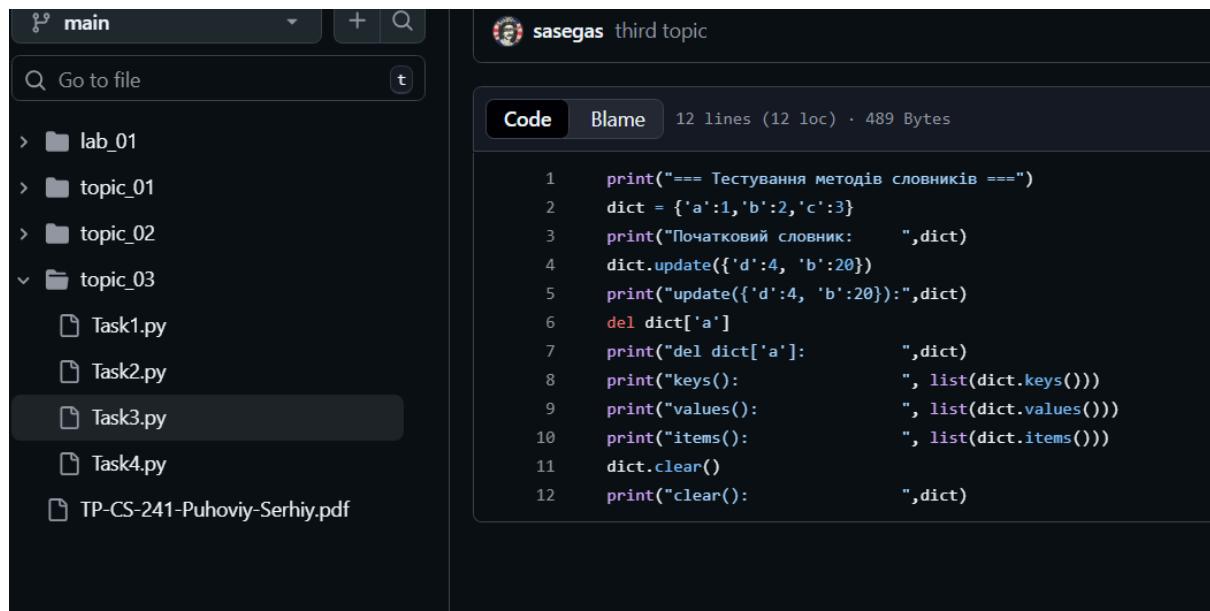


Рис.3 Скріншот третього завдання з GitHub

Пошук позиції для вставки

Маючи відсортований список, написати функцію пошуку позиції для вставки нового елементу в список.

Хід виконання завдання:

В цьому завданні я зробив цикл, який вираховує куди треба вставити елемент у відсортований список, суть циклу в перебиренні списку з

середини і в залежності від значення рухається назад, або вперед, після сортування за допомогою метода `include` вставляється новий елемент в список.

Текст програми:

```
def find_insert_position(sorted_list, value):

    left = 0

    right = len(sorted_list)

    while left < right:

        mid = (left + right) // 2

        if sorted_list[mid] < value:

            left = mid + 1

        else:

            right = mid

    return left

my_list = [1, 3, 5, 7, 9]

new_value = 6

position = find_insert_position(my_list, new_value)

print(f"Новий елемент {new_value} слід вставити на позицію {position + 1}")

my_list.insert(position, new_value)

print("Список після вставки:", my_list)
```

```
Code Blame 19 lines (16 loc) · 562 Bytes
1  def find_insert_position(sorted_list, value):
2      left = 0
3      right = len(sorted_list)
4
5      while left < right:
6          mid = (left + right) // 2
7          if sorted_list[mid] < value:
8              left = mid + 1
9          else:
10             right = mid
11
12     return left
13
14 my_list = [1, 3, 5, 7, 9]
15 new_value = 6
16 position = find_insert_position(my_list, new_value)
17
18 my_list.insert(position, new_value)
19 print("Список після вставки:", my_list)
```

Рис.4 Скріншот четвертого завдання з GitHub

Звіт до Теми № 4

Виняткові ситуації

Обробка виняткових ситуацій в калькуляторі

Розширити програму калькулятор функцією запитів даних для виконання операцій від користувача, що обробляє виняткові ситуації.

Хід виконання завдання:

Для виконання цього завдання я додав обробку помилки `ValueError`, щоб програма не зупинялась, якщо був введений неправильний тип даних

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):
    return a / b

print("Щоб вийти, введіть 'exit' замість знака дії.")

while True:
    try:
        a = float(input("Введіть перше число: "))
        b = float(input("Введіть друге число: "))
    except ValueError:
        print("Потрібно вводити числа!")
        continue

    sign = input("Введіть дію (+ - * /) або 'exit' для виходу: ")

    if sign.lower() == "exit":
        print("Роботу завершено.")
        break

    match sign:
        case "+":
            result = plus(a, b)
        case "-":
            result = minus(a, b)
        case "*":
            result = multiply(a, b)
```

```

        case "/":
            result = divide(a, b)

        case _:
            result = "Була задана некоректна дія"

print("Відповідь:", result, "\n")

```

```

Code Blame 44 lines (32 loc) · 1.01 KB
1 def plus(a, b):
2     return a + b
3
4 def minus(a, b):
5     return a - b
6
7 def multiply(a, b):
8     return a * b
9
10 def divide(a, b):
11     return a / b
12
13
14
15 print("Щоб вийти, введіть 'exit' замість знака дії.")
16
17 while True:
18     try:
19         a = float(input("Введіть перше число: "))
20         b = float(input("Введіть друге число: "))
21     except ValueError:
22         print("Потрібно вводити числа!")

```

Рис.1 Скріншот першого завдання з GitHub

Обробка виняткової ситуації ділення на нуль в калькуляторі

Розширити функцію ділення обробкою виняткової ситуації ділення на нуль

Хід виконання завдання:

Для виконання цього завдання я додав обробку помилки `ZeroDivisionError` в функцію ділення, щоб замість відповіді повертається текст помилки.

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):
    try:
        return a / b
    except ZeroDivisionError:
        return "Не можна ділити на нуль"

print("Щоб вийти, введіть 'exit' замість знака дії.")

while True:
    try:
        a = float(input("Введіть перше число: "))
        b = float(input("Введіть друге число: "))
    except ValueError:
        print("Потрібно вводити числа!")
        continue

    sign = input("Введіть дію (+ - * /) або 'exit' для виходу: ")

    if sign.lower() == "exit":
        print("Роботу завершено.")
        break

    match sign:
        case "+":
            result = plus(a, b)
        case "-":
```

```

        result = minus(a, b)

    case "*":
        result = multiply(a, b)

    case "/":
        result = divide(a, b)

    case _:
        result = "Була задана некоректна дія"

print("Відповідь:", result, "\n")

```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a tree view of the repository structure:

- > lab_01
 - topic_01
 - topic_02
 - topic_03
 - topic_04
 - Task1.py
 - Task2.py**
 - TP-CS-241-Puhoviy-Serhiy.pdf

The main area displays the content of `Task2.py`:

```

1  def plus(a, b):
2      return a + b
3
4  def minus(a, b):
5      return a - b
6
7  def multiply(a, b):
8      return a * b
9
10 def divide(a, b):
11     try:
12         return a / b
13     except ZeroDivisionError:
14         return "Не можна ділити на нуль"
15
16
17 print("Щоб вийти, введіть 'exit' замість знака дії.")
18
19 while True:
20     try:
21         a = float(input("Введіть перше число: "))
22         b = float(input("Введіть друге число: "))

```

Рис.2 Скріншот другого завдання з GitHub

Звіт до Теми № 5

Бібліотеки

Гра з комп'ютером: камінь, ножиці, папір

Програма виконує запит від користувача на введення одного із значень ["stone", "scissor", "paper"]. Наступним кроком, використовуючи модуль random, програма у випадковому порядку вибирає одне із значень ["stone", "scissor", "paper"]. В залежності від умови, що камінь перемагає ножиці, ножиці перемагають папір, а папір перемагає камінь визначити переможця.

Хід виконання завдання:

Для виконання цього завдання спочатку я створив масив з варіантами значень, а потім створив функцію game, яка вираховує всі можливі варіанти закінчення раунду і створив головну функцію де задаються значення гравця і комп'ютера, для того щоб комп'ютер обирав випадково варіанти я підключив бібліотеку random і передав змінні в функцію game.

Текст програми:

```
import random

options = ["stone", "scissor", "paper"]

def game(user, bot):
    if user == bot:
        print("Нічия")
    elif (
        (user == "stone" and bot == "scissor") or
        (user == "paper" and bot == "stone") or
        (user == "scissor" and bot == "paper")):
        print("Ви перемогли!")
    else:
        print("Комп'ютер переміг")

def main():
    while True:
        user_choise = input("Введіть своє значення (stone, scissor, paper) або 'exit' для виходу: ").lower()
        if user_choise == "exit":
            print("Гру завершено.")
            break
        elif user_choise not in options:
            print("Неправильне значення! Оберіть stone, scissor або paper.")
```

```

        continue

    else:

        computer_choice = random.choice(options)

        print(f"Комп'ютер обрав: {computer_choice}")

        game(user_choice, computer_choice)

        continue

main()

```

```

Code Blame 30 lines (27 loc) · 891 Bytes
1 import random
2
3 options = ["stone", "scissor", "paper"]
4
5 def game(user, bot):
6     if user == bot:
7         print("Нічия")
8     elif (
9         (user == "stone" and bot == "scissor") or
10        (user == "paper" and bot == "stone") or
11        (user == "scissor" and bot == "paper")):
12         print("Ви перемогли!")
13     else:
14         print("Комп'ютер переміг")
15

```

Рис.1 Скріншот першого завдання з GitHub

Програма конвертування іноземної валюти в українську гривню

Для отримання актуальних курсів валют необхідно використовувати API НБУ та модуль, що надає можливість виконувати запити до сторонніх сервісів requests. Достатня умова роботи – можливість конвертації для трьох іноземних валют EUR, USD, PLN. Користувачу надається можливість введення кількості та типу валюти, результат роботи програми – конвертоване значення в українських гривнях.

Хід виконання завдання:

В цьому завданні спочатку я підключив модуль requests для обробки API, після чого підключив шлях по якому буду отримувати курс і створив функцію яка робить запит по API і повертає курси валют в форматі Json і записав за допомогою циклу потрібні значення в словник rates, після чого в

функції main вивів поточний курс, після чого зробив конвертор цих валют в гривні.

Текст програми:

```
import requests

URL = "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json"

def get_exchange_rates():

    response = requests.get(URL)

    if response.status_code != 200:

        print("Помилка отримання даних з НБУ!")

        return None

    data = response.json()

    rates = { }

    for item in data:

        if item["cc"] in ["USD", "EUR", "PLN"]:

            rates[item["cc"]] = item["rate"]

    return rates

def main():

    rates = get_exchange_rates()

    if not rates:

        return

    print("Актуальні курси валют (за 1 одиницю):")
```

```
for code, rate in rates.items():

    print(f"{code}: {rate} грн")

while True:

    currency = input("\nВведіть валюту (USD, EUR, PLN) або 'exit' для
виходу: ").upper()

    if currency == "EXIT":

        print("Програму завершено.")

        break

    if currency not in rates:

        print("Неправильна валюта! Оберіть USD, EUR або PLN.")

        continue

    try:

        amount = float(input("Введіть суму: "))

        result = amount * rates[currency]

        print(f"{amount} {currency} = {result:.2f} грн")

    except ValueError:

        print("Введіть числове значення суми!")

main()
```

```

1 import requests
2
3 URL = "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json"
4
5 def get_exchange_rates():
6     response = requests.get(URL)
7     if response.status_code != 200:
8         print("Помилка отримання даних з НБУ!")
9     return None
10
11 data = response.json()
12 rates = {}
13
14 for item in data:
15     if item["cc"] in ["USD", "EUR", "PLN"]:
16         rates[item["cc"]] = item["rate"]
17
18 return rates
19
20 def main():
21     rates = get_exchange_rates()
22     if not rates:

```

Рис.2 Скріншот другого завдання з GitHub

Використання модулів для програми калькулятор

Функції додавання, віднімання, множення та ділення перенести в файл functions.py. Функції запиту на введення даних для операцій та самих операцій перемістити в файл operations.py. Програму калькулятор реалізувати в файлі calc.py, до якого підключають файл functions.py та operations.py.

Хід виконання завдання:

Я створив 3 файли functions.py, operations.py, calc.py, після чого в першому файлі створив функції які виконують математичний обрахунок, в другому файлі я імпортував всі функції з первого і створив дві функції, перша функція отримує числа від користувача, а друга отримує знак і виконує функції з минулого файлу, а в головному файлі calc я імпортую функцію perform_operation

Текст програми:

functions.py

```

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):
    try:
        return a / b
    except ZeroDivisionError:
        return "Не можна ділити на нуль"
```

operations.py

```
from functions import plus, minus, divide, multiply

def get_number():
    try:
        a = float(input("Введіть перше число: "))
        b = float(input("Введіть друге число: "))
        return a, b
    except ValueError:
        print("Помилка: введіть числове значення!")
        return None, None

def perform_operation():
    while True:
        a, b = get_number()
        if a is None or b is None:
            continue
        sign = input("Введіть дію (+ - * /) або 'exit' для виходу: ")
```

```
if sign == "exit":  
  
    print("Вихід з програми.")  
  
    break  
  
  
match sign:  
  
    case "+":  
  
        print(f"Результат: {plus(a, b)}")  
  
    case "-":  
  
        print(f"Результат: {minus(a, b)}")  
  
    case "*":  
  
        print(f"Результат: {multiply(a, b)}")  
  
    case "/":  
  
        print(f"Результат: {divide(a, b)}")  
  
    case _:  
  
        print("Була задана некоректна дія.")
```

calc.py

```
from operations import perform_operation  
  
  
def main():  
    perform_operation()  
  
  
if __name__ == "__main__":  
    main()
```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a tree view of directory structures:

- lab_01
- topic_01
- topic_02
- topic_03
- topic_04
- topic_05
 - Task3
 - __pycache__
 - calc.py
 - functions.py
 - operations.py

The file `calc.py` is highlighted with a dark grey bar at the bottom of its listing. On the right, the main area displays the following Python code:

```
Code Blame / lines (5 loc) • 110 Bytes
1 from operations import perform_operation
2
3 def main():
4     perform_operation()
5
6 if __name__ == "__main__":
7     main()
```

Рис.3 Скріншот третього завдання з GitHub

Звіт до Теми № 6

Робота з файлами

Розробити механізм логування всіх дій, що виконує програма

Забезпечити зберігання інформації про введені дані, виконану операцію та результат виконання операції над даними.

Хід виконання завдання:

Спочатку я створив файл в якому будуть зберігатись логи, потім в файлі `functions.py` задав змінну з назвою файлу, після чого створив функцію `log_custom` в яку передаю п'ять значень, 3 обов'язкових і 2 додаткових, в залежності від додаткових значень `result` або `errlog` визначається який матиме статус лог, після чого роблю рядок тексту, який потім вставляю в файл з логами і додаю доожної дії обробку цієї функції. В файлі `operations.py` роблю логування початку і завершення роботи, а також обробку декількох помилок.

Текст програми:

`functions.py`

```
LOG_FILE = "calculator_log.txt"
```

```
def log_custom(sign, a, b, result=None, error=None):
    if error:
        log_level = "ERROR"
        status_info = f"Помилка: {error}"
    else:
        log_level = "INFO"
        status_info = f"Результат: {result}"

    log_entry = f"{log_level} - Операція: {a} {sign} {b} , {status_info}\n"

    with open(LOG_FILE, "a", encoding="utf-8") as file:
        file.write(log_entry)

def plus(a, b):
    result = a + b
    log_custom("+", a, b, result)
    return result

def minus(a, b):
    result = a - b
    log_custom("-", a, b, result)
    return result

def multiply(a, b):
```

```

    result = a * b
    log_custom("*", a, b, result)
    return result

def divide(a, b):
    try:
        result = a / b
        log_custom("/", a, b, result)
        return result
    except ZeroDivisionError:
        error_msg = "Не можна ділити на нуль"
        log_custom("/", a, b, error=error_msg)
        return error_msg

```

operations.py

```

from functions import plus, minus, divide, multiply, log_custom

def get_number():
    try:
        a = float(input("Введіть перше число: "))
        b = float(input("Введіть друге число: "))
        return a, b
    except ValueError:
        error_msg = "Помилка: введіть числове значення!"
        print(error_msg)
        log_custom("err", "N/A", "N/A", error="Введено не числове
значення")
        return None, None

```

```
def perform_operation():

    log_custom("?", "N/A", "N/A", result="Початок роботи")

    while True:

        a, b = get_number()

        if a is None or b is None:
            continue

        sign = input("Введіть дію (+ - * /) або 'exit' для виходу:")

        if sign == "exit":
            print("Вихід з програми.")

            log_custom("?", "N/A", "N/A", result="Завершення
роботи")

            break

        result = None

        match sign:

            case "+":
                result = plus(a, b)

            case "-":
                result = minus(a, b)

            case "*":
                result = multiply(a, b)

            case "/":
                result = divide(a, b)

            case _:
```

```

        error_msg = "Була задана некоректна дія."

        print(error_msg)

        log_custom("err", a, b, error=f"Невідома дія - 
{sign}")

        continue

    print(f"Результат: {result}")

```

The screenshot shows a GitHub repository interface. On the left, there's a file tree for a repository named 'TP-CS-241-Puhoviy-Serhiy'. The tree includes folders like 'lab_01', 'topic_01' through 'topic_06', and 'Task1' which contains 'calc.py', 'functions.py', and 'operations.py'. It also includes 'Task2.py', '.gitignore', 'TP-CS-241-Puhoviy-Serhiy.pdf', and 'calculator_log.txt'. On the right, there's a code editor window titled 'Code' with a tab for 'Blame'. The code shown is a Python script for a calculator. It defines a function 'log_custom' that logs operations based on a sign ('+', '-', '*', '/') and handles errors. It also defines a 'plus' function that adds two numbers and logs the result. The code editor shows line numbers from 1 to 23.

```

1     LOG_FILE = "calculator_log.txt"
2
3     def log_custom(sign, a, b, result=None, error=None):
4
5         if error:
6             log_level = "ERROR"
7             status_info = f"Помилка: {error}"
8         else:
9             log_level = "INFO"
10            status_info = f"Результат: {result}"
11
12            log_entry = f"{log_level} - Операція: {a} {sign} {b} , {status_info}\n"
13
14
15            with open(LOG_FILE, "a", encoding="utf-8") as file:
16                file.write(log_entry)
17
18
19            def plus(a, b):
20                result = a + b
21                log_custom("+", a, b, result)
22
23                return result

```

Рис.1 Скріншот першого завдання з GitHub

Сортування списку

Маючи не відсортований список, елементами якого є словники з двома параметрами (ім'я та оцінка) виконати сортування списку, використовуючи стандартну функцію `sorted()`. Другим параметром для функції `sorted()` має бути `lambda` функція, що повертає ім'я або оцінку із елемента словника.

Хід виконання завдання:

Спочатку я імпортував csv, потім створив пустий масив для майбутніх даних, далі дістав з файлу table.csv дані про студентів та додав їх до масиву і виконав сортування використовуючи lambda

Текст програми:

```
import csv

studentList = []

with open("table.csv") as file:
    reader = csv.DictReader(file, skipinitialspace=True)
    for row in reader:
        studentList.append({"name":row["StudentName"], "mark":int(row["StudentMark"])})

print("---- Сортування ----")

for elem in sorted(studentList, key=lambda x: x['mark']):
    print(f"Name = {elem['name']} mark = {elem['mark']}")
```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a tree view of the project structure, including folders like lab_01, topic_01, topic_02, topic_03, topic_04, topic_05, topic_06, and a Task1 folder containing a Task2.py file. The .gitignore file is also listed. On the right, the main area displays the contents of Task2.py. The code is presented in a dark-themed code editor with syntax highlighting. At the top of the code area, there are tabs for 'Code' and 'Blame', and a status bar indicating '13 lines (9 loc) · 386 Bytes'. The code itself is identical to the one shown in the previous text block.

Рис.2 Скріншот другого завдання з GitHub

Звіт до Теми № 7

Об'єктно-орієнтоване програмування

Ознайомитись з існуючими за замовченням методами класу

Ознайомитись з існуючими за замовченням методами класу по типу `__init__(self)` `__str__(self)` та надати приклади використання.

Хід виконання завдання:

Для прикладу я використав три методи, перший це `__init__` - це конструктор, який запускається автоматично при створенні нового екземпляра класу, `__str__` - відповідає за те щоб повернати зрозумілий і читабельний опис об'єкта для користувача і `__repr__` - відповідає за представлення об'єкта для розробника, я зробив клас Book який спочатку отримує дані, потім обробляє їх для клієнта і для розробника, потім я задав змінну з цим класом в якій я задав об'єкт.

Текст програми:

```
class Book:

    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return f"Книга '{self.title}', автор: {self.author}"

    def __repr__(self):
        return f"Book(title='{self.title}', author='{self.author}')"
```

```

my_book = Book("Дюна", "Герберт Френк")

def main():

    print(my_book)

    print(repr(my_book))

main()

```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a tree view of the project structure:

- lab_02
- topic_01
- topic_02
- topic_03
- topic_04
- topic_05
- topic_06
- topic_07** (selected)
- Task3
 - Task1.py**
 - Task2.py
- .gitignore

The main area displays the contents of `Task1.py`:

```

1  class Book:
2      def __init__(self, title, author):
3          self.title = title
4          self.author = author
5
6      def __str__(self):
7          return f"Книга '{self.title}', автор: {self.author}"
8
9      def __repr__(self):
10         return f"Book(title='{self.title}', author='{self.author}')"
11
12 my_book = Book("Дюна", "Герберт Френк")
13
14 def main():
15     print(my_book)
16     print(repr(my_book))
17
18 main()

```

Рис.1 Скріншот першого завдання з GitHub

Розробити клас Student

Розробити клас `Student` атрибутами якого є два параметра `name` та `age`. Створити список елементами якого є об'єкти класу `Student`. Написати цикл який виводить на екран елементи списку у відсортованому порядку. Для сортування використати стандартну функцію `sorted`. Функція `sorted` має використовувати `lambda` функцію для визначення ключа сортування.

Хід виконання завдання:

Спочатку я створив клас `Student` в якому ініціалізував ім'я та вік студента, після чого задав коректний вивід даних, потім створив масив зі

студентами, та в main функції виконав 2 сортування в яких ключ визначала lambda, та одне сортування виконувалось за віком, інше за ім'ям, потім за допомогою циклу виводив відсортовані списки.

Текст програми:

```
class Student:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"Студент: {self.name}, Вік: {self.age}"


students = [
    Student("Андрій", 21),
    Student("Олена", 19),
    Student("Богдан", 22),
    Student("Вікторія", 20)
]

def main():

    sorted_by_age = sorted(students, key=lambda student: student.age)

    print("--- Сортування за віком (від найменшого) ---")
    for student in sorted_by_age:
        print(student)

    sorted_by_name = sorted(students, key=lambda student: student.name)
```

```

print("\n--- Сортування за ім'ям (алфавіт) ---")

for student in sorted_by_name:

    print(student)

main()

```

The screenshot shows a GitHub repository interface. On the left, there is a sidebar with a tree view of files and folders. The visible items include: lab_01, lab_02, topic_01, topic_02, topic_03, topic_04, topic_05, topic_06, topic_07 (which contains Task3, Task1.py, and Task2.py), and a main.py file. The main.py file is displayed on the right in a code editor. The code defines a Student class with __init__ and __str__ methods, a list of Student objects named students, and a main function that sorts the students by age.

```

> lab_01
> lab_02
> topic_01
> topic_02
> topic_03
> topic_04
> topic_05
> topic_06
> topic_07
>   Task3
>     Task1.py
>     Task2.py
> main.py

1  class Student:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6      def __str__(self):
7          return f"Студент: {self.name}, Вік: {self.age}"
8
9  students = [
10     Student("Андрій", 21),
11     Student("Олена", 19),
12     Student("Богдан", 22),
13     Student("Вікторія", 20)
14 ]
15
16 def main():
17     sorted_by_age = sorted(students, key=lambda student: student.age)
18

```

Рис.2 Скріншот другого завдання з GitHub

Калькулятор з використанням ООП

Використовуючи принципи ООП переписати програму Калькулятор. Завдання має бути виконано використовуючи модульний підхід.

Хід виконання завдання:

Спочатку я в файлі function.py створив клас MathOperations, в якому виконуються всі доступні операції, потім в файлі operation.py імпортував клас MathOperations і створив інший клас Operator де задав змінну self.engine яка має клас MathOperations, після чого виконую функцію де в залежності від вибору знаку буде виконуватись певна дія і в головному файлі calc.py в функції main створюється зміна з класом Operator і виконується введення даних і виведення результату.

Текст програми:

function.py

```
class MathOperations:
    def plus(self, a, b):
        return a + b

    def minus(self, a, b):
        return a - b

    def multiply(self, a, b):
        return a * b

    def divide(self, a, b):
        if b != 0:
            return a / b
        else:
            raise ValueError("Ділення на нуль неможливе")
```

operation.py

```
from function import MathOperations

class Operator:
    def __init__(self):
        self.engine = MathOperations()

    def perform_calculation(self, a, b, sign):
        match sign:
            case "+":
                return self.engine.plus(a, b)
            case "-":
                return self.engine.minus(a, b)
            case "*":
                return self.engine.multiply(a, b)
            case "/":
                return self.engine.divide(a, b)
            case _:
                error_msg = "Була задана некоректна дія."
                print(error_msg)
```

calc.py

```
from operation import Operator

def main():
    app = Operator()

    try:
        num1 = float(input("Введіть перше число: "))
        num2 = float(input("Введіть друге число: "))
        sign = input("Введіть дію (+ - * /): ")

        result = app.perform_calculation(num1, num2, sign)

        print(f"Відповідь: {result}")

    except ValueError as e:
        print(f"Виникла помилка: {e}")

if __name__ == "__main__":
    main()
```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a tree view of the project structure:

- lab_02
- topic_01
- topic_02
- topic_03
- topic_04
- topic_05
- topic_06
- topic_07
- Task3
 - calc.py
 - function.py

On the right, the content of the `calc.py` file is displayed:

```
1  class MathOperations:
2      def plus(self, a, b):
3          return a + b
4
5      def minus(self, a, b):
6          return a - b
7
8      def multiply(self, a, b):
9          return a * b
10
11     def divide(self, a, b):
12         if b != 0:
13             return a / b
14         else:
15             raise ValueError("Ділення на нуль неможливе")
```

Рис.3 Скріншот третього завдання з GitHub