Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**з дисципліни**

**Кросплатформні засоби програмування**

**Звіт з Лабораторної роботи № 8**

**ФАЙЛИ ТА ВИКЛЮЧЕННЯ У PYTHON**

**Виконав:**

КІ-307

Герега Р. О.

**Перевірив:**

Іванов Ю.С

**2023**

**Мета роботи:** навиками використання засобів мови Python для роботи з

файлами.

**Теоретичний вступ**

Функції

Функції у мові python не відрізняються за своєю суттю від функцій С/С++.

Синтаксис оголошення функцій:

def function\_name({параметри}):

[оператори]

Приклади оголошення функцій:

def my\_output():

print("Hello world")

def my\_output(txt1, txt2, delimitator = " "):

"""

Outputs concatenated string

:param txt1: the first text string

:param txt2: the second text string

:param delimitator: Delimitator. Space by default.

:return: concatenated string

"""

merged\_text = delimitator.join((txt1, txt2))

return merged\_text

Приклади виклику функцій:

my\_output("Hello ", "world")

my\_output(txt1 = "Hello ", txt2 = "world") # Передача параметрів за назвою параметра

Функції з довільної кількістю параметрів

У Python функції можуть мати довільну кількість параметрів. У цьому випадку їм

можна передавати неіменовані або іменовані параметри, або їх комбінацію. Неіменовані

параметри довільної кількості передаються за допомогою конструкції:

\*args

При цьому всі передані аргументи поміщаються у артумент args, який є списком,

що містить значення переданих аргументів.

Приклад оголошення функції з довільною кількістю неіменованих аргументів

def my\_output(\*args):

"""

Outputs concatenated string

:param args: the tuple of text strings

:return: concatenated string

"""

merged\_text = str()

for arg in args:

merged\_text = merged\_text + " " + arg

return merged\_text

Приклад виклику функції з довільною кількістю неіменованих аргументів

my\_output("Hello ", "world")

**Варіант 6**

y=tg(x)/sin(2x)

**Завдання:**

1. Створити параметризований клас, що реалізує предметну область задану варіантом. Клас має містити мінімум 4 методи опрацювання даних включаючи розміщення та виймання елементів. Парні варіанти реалізують пошук мінімального елементу, непарні – максимального. Написати на мові Java та налагодити програму-драйвер для розробленого класу, яка мстить мінімум 2 різні класи екземпляри яких розмішуються у екземплярі розробленого класу-контейнеру.
2. Програма має розміщуватися в пакеті Група.Прізвище.Lab6 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету. Автоматично згенерувати документацію до розробленого пакету.
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату їївиконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

**Код**

import math

import pickle

def calculate\_expression(x):

try:

result = math.tan(x) / math.sin(2 \* x)

return result

except ZeroDivisionError:

print("Error: sin(2x) is undefined when tan(x) = 0.")

return None

def save\_to\_text\_file(filename, data):

with open(filename, 'w') as file:

for item in data:

file.write(str(item) + '\n')

def save\_to\_binary\_file(filename, data):

with open(filename, 'wb') as file:

pickle.dump(data, file)

def read\_from\_text\_file(filename):

data = []

try:

with open(filename, 'r') as file:

for line in file:

data.append(float(line.strip()))

except FileNotFoundError:

print(f"Error: File '{filename}' not found.")

return data

def read\_from\_binary\_file(filename):

data = []

try:

with open(filename, 'rb') as file:

data = pickle.load(file)

except FileNotFoundError:

print(f"Error: File '{filename}' not found.")

return data

if \_\_name\_\_ == "\_\_main\_\_":

x = float(input("Enter x:"))

res = calculate\_expression(x)

print(f"result: {res}")

x\_values = [0.1, 0.2, 0.3, 0.4, 0.5]

print(f"x\_values: {x\_values}")

results = [calculate\_expression(x) for x in x\_values]

print(f"results: {results}")

save\_to\_text\_file("results.txt", results)

save\_to\_binary\_file("results.dat", results)

text\_data = read\_from\_text\_file("results.txt")

binary\_data = read\_from\_binary\_file("results.dat")

print("Results from text file:", text\_data)

print("Results from binary file:", binary\_data)

**Результат:**

Enter x:1

result: 1.71275941040738

x\_values: [0.1, 0.2, 0.3, 0.4, 0.5]

results: [0.5050335232112475, 0.5205456792479637, 0.5478444576612735, 0.5893770529054876, 0.6492232052047624]

Results from text file: [0.5050335232112475, 0.5205456792479637, 0.5478444576612735, 0.5893770529054876, 0.6492232052047624]

Results from binary file: [0.5050335232112475, 0.5205456792479637, 0.5478444576612735, 0.5893770529054876, 0.6492232052047624]

**Короткий висновок**

Під час виконання даної лабораторної роботи я оволодів навиками використання засобів мови Python для роботи з

файлами.