НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ»

Звiт з лабораторної роботи №7

Тема: «Декоратори»

Виконав:   
ст. гр. КІТ-118в  
Міщенко Д.С.

Перевірив:  
Поворознюк О.А.

Харків – 2021

***Мета:*** Ознайомлення та отримання практичних навичок у використанні декораторів

***Загальне завдання***

*Напишіть декоратори:*

1. Для відстежування часу початку та завершення виконання програми

(використовуйте модуль datetime); для симуляції довгих обчислень можна

використовувати sleep().

2. Для декількох запусків функції у разі її неуспішного виконання (функція

виконується успішно з деякою вірогідністю р);

3. Декоратор, що буде додавати у змінну REGISTERED усі продекоровані

функції програми.

*Умови виконання:*

1. Передбачити, що декоратори можуть приймати аргументи.

2. Продемонструвати виклики декораторів за допомогою звичайних викликів

та нотації @. Приклад декорування: @measure\_time(timezone=”US/Eastern”).

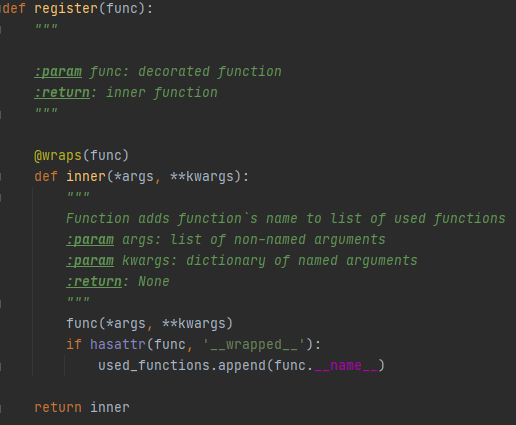
3. Передбачити, що функції, які будуть декоровані приймають аргументи.

4. Зробити описи Doc strings.

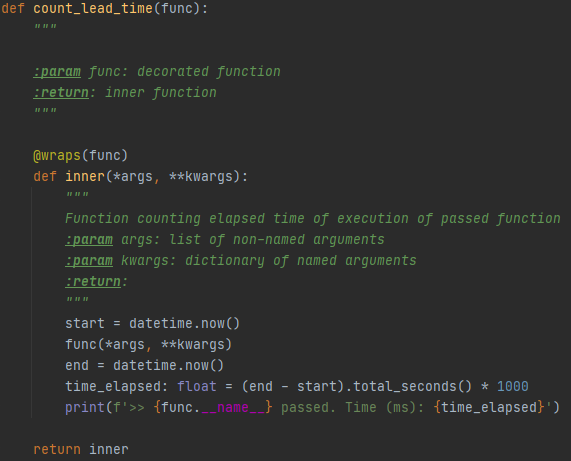
5. Перевірити, що після декорування метадані початкової функції збережені.

***Хід роботи***

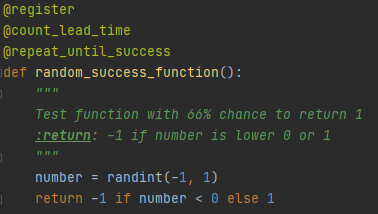
***Фрагменти програми***

******

**Рисунок 1** ─ Декоратор реєстрування викликаної функції

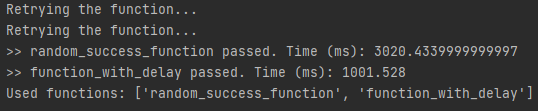
******

**Рисунок 2** ─ Декоратор підрахунку часу виконання програми

******

**Рисунок 3** ─ Декорування анотаціями

***Результати роботи***



**Рисунок 3** ─ Отримання століття з заданого року

***Код програми***

from datetime import datetime  
import time  
from random import randint  
from functools import wraps  
  
used\_functions = list()  
  
  
def register(func):  
 *"""* ***:param*** *func: decorated function* ***:return****: inner function  
 """* @wraps(func)  
 def inner(\*args, \*\*kwargs):  
 *"""  
 Function adds function`s name to list of used functions* ***:param*** *args: list of non-named arguments* ***:param*** *kwargs: dictionary of named arguments* ***:return****: None  
 """* func(\*args, \*\*kwargs)  
 if hasattr(func, '\_\_wrapped\_\_'):  
 used\_functions.append(func.\_\_name\_\_)  
  
 return inner  
  
  
def count\_lead\_time(func):  
 *"""* ***:param*** *func: decorated function* ***:return****: inner function  
 """* @wraps(func)  
 def inner(\*args, \*\*kwargs):  
 *"""  
 Function counting elapsed time of execution of passed function* ***:param*** *args: list of non-named arguments* ***:param*** *kwargs: dictionary of named arguments* ***:return****:  
 """* start = datetime.now()  
 func(\*args, \*\*kwargs)  
 end = datetime.now()  
 time\_elapsed: float = (end - start).total\_seconds() \* 1000  
 print(f'>> {func.\_\_name\_\_} passed. Time (ms): {time\_elapsed}')  
  
 return inner  
  
  
def repeat\_until\_success(func):  
 *"""* ***:param*** *func: decorated function* ***:return****: inner function  
 """* @wraps(func)  
 def inner(\*args, \*\*kwargs):  
 *"""  
 Function will repeat call to passed function until it returns -1* ***:param*** *args: list of non-named arguments* ***:param*** *kwargs: dictionary of named arguments* ***:return****: None  
 """* i = 1  
 while func(\*args, \*\*kwargs) == -1:  
 print('Retrying the function...')  
 time.sleep(i)  
 i += 1  
  
 return inner  
  
  
@register  
@count\_lead\_time  
@repeat\_until\_success  
def random\_success\_function():  
 *"""  
 Test function with 66% chance to return 1* ***:return****: -1 if number is lower 0 or 1  
 """* number = randint(-1, 1)  
 return -1 if number < 0 else 1  
  
  
def function\_with\_delay(delay=0):  
 *"""* ***:param*** *delay: Delay time in sec* ***:return****: None  
 """* time.sleep(delay)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 random\_success\_function()  
 function\_with\_delay = register(count\_lead\_time(function\_with\_delay))  
 function\_with\_delay(1)  
 print(f"Used functions: {used\_functions}")

**Висновки:** в ході виконання лабораторної роботи були набуті практичні навички з роботи з декораторами в мові Python.