Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

ОТЧЁТ ПО

Лабораторной работе №6

Выполнил: Богуславский Андрей

студент группы ИУ5-31Б

_____

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Нардид А.Н.

Подпись и дата:

Москва

2024

Лабораторная работа №6

Функциональные возможности языка Python.
Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

Задание:

Задача 4 (файл sort.py)

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо **одной строкой кода** вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted.
Пример:

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
Вывод: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

Шаблон реализации:

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

if __name__ == '__main__':
    result = ...
    print(result)

    result_with_lambda = ...
    print(result_with_lambda)
```

Задача 5 (файл print_result.py)

Необходимо реализовать декоратор print_result, который выводит на экран результат выполнения функции.

- Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.
- Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.
- Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

Шаблон реализации:

```
# Здесь должна быть реализация декоратора

@print_result
def test_1():
    return 1


@print_result
def test_2():
    return 'iu5'


@print_result
def test_3():
    return {'a': 1, 'b': 2}


@print_result
def test_4():
    return [1, 2]


if __name__ == '__main__':
    print('!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

Результат выполнения:

```
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

Задача 6 (файл cm_timer.py)

Необходимо написать контекстные менеджеры cm_timer_1 и cm_timer_2, которые считают время работы блока кода и выводят его на экран. Пример:

```
with cm_timer_1():
```

sleep(5.5)

После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

cm_timer_1 и cm_timer_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки contextlib).

Текст программы:

Файл sort.py:

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

if __name__ == '__main__':
    result =  sorted(data, key=lambda x: abs(x), reverse=True)
    print(result)

result_with_lambda = sorted(data, key=abs, reverse=True)
print(result_with_lambda)
```

Файл print_result.py:

```
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)

        if isinstance(result, list):
            print(func.__name__)
            for item in result:
                print(item)

        elif isinstance(result, dict):
            print(func.__name__)
            for key, value in result.items():
                print(f'{key} = {value}')

        else:
            print(func.__name__)
            print(result)

        return result
    return wrapper

@print_result
def test_1():
    return 1

@print_result
```

```python
def test_2():
    return 'iu5'

@print_result
def test_3():
    return {'a': 1, 'b': 2}

@print_result
def test_4():
    return [1, 2]


test_1()
test_2()
test_3()
test_4()
```

Файл cm_timer.py:

```python
import time
from contextlib import contextmanager

class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        elapsed_time = time.time() - self.start_time
        print(f"time: {elapsed_time}")

@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    elapsed_time = time.time() - start_time
    print(f"time: {elapsed_time}")

# Использование cm_timer_1
with cm_timer_1():
    time.sleep(5.5)

# Использование cm_timer_2
with cm_timer_2():
    time.sleep(5.5)
```

Экранные формы с выводом:

```
PS C:\Users\Andrey\Desktop\инфа> & C:/Users/Andrey/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Andrey/Desktop/инфа/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
PS C:\Users\Andrey\Desktop\инфа>
```

```
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
PS C:\Users\Andrey\Desktop\инфа>
```

```
PS C:\Users\Andrey\Desktop\инфа> & C:/Users/Andrey/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Andrey/Desktop/инфа/cm_timer.py
time: 5.500959873199463
time: 5.500692367553711
PS C:\Users\Andrey\Desktop\инфа>
```