Лабораторная работа №4 (продолжение ЛР№3)

Функциональные возможности языка Python.

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

Задание:

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab_python_fp. Решение каждой задачи должно раполагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

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).

Задача 7 (файл process_data.py)

- В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.
- В файле data_light.json содержится фрагмент списка вакансий.
- Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.
- Необходимо реализовать 4 функции - f1, f2, f3, f4. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print_result печатается результат, а контекстный менеджер cm_timer_1 выводит время работы цепочки функций.
- Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.
- Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.

- Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.
- Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: Программист C# с опытом Python. Для модификации используйте функцию map.
- Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб. Используйте zip для обработки пары специальность — зарплата.

Шаблон реализации:

```
import json

import sys

# Сделаем другие необходимые импорты


path = None


# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария


with open(path) as f:

    data = json.load(f)


# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`

# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку

# В реализации функции f4 может быть до 3 строк


@print_result
```

```
def f1(arg):
    raise NotImplemented


@print_result
def f2(arg):
    raise NotImplemented


@print_result
def f3(arg):
    raise NotImplemented


@print_result
def f4(arg):
    raise NotImplemented


if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

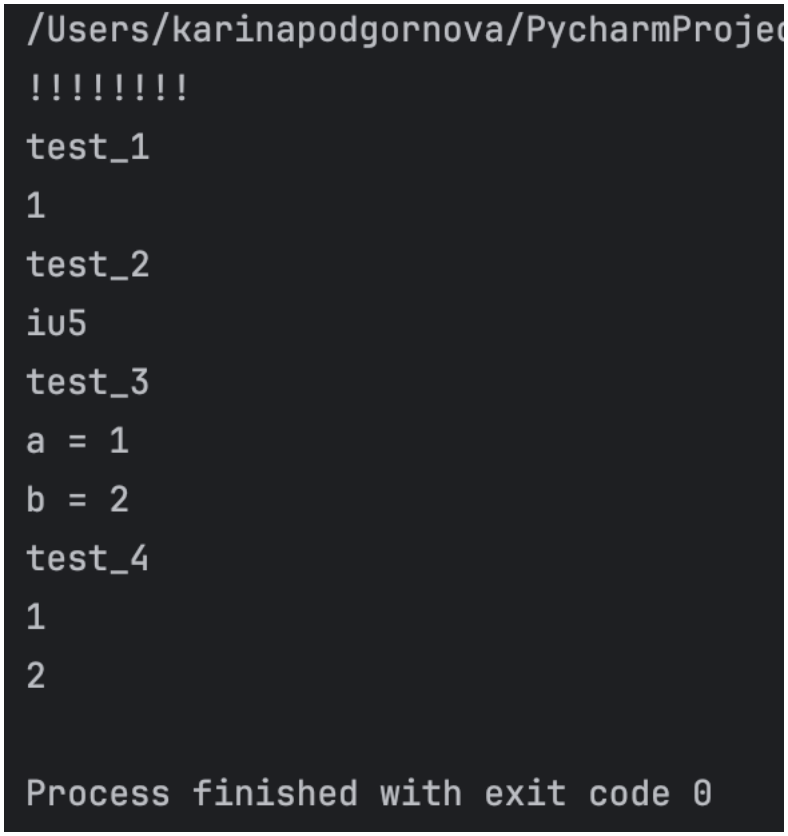
Задача 5 (файл print_result.py)

Текст программы

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

```
        print(func.__name__)
        if isinstance(result, list):
            for item in result:
                print(item)
        elif isinstance(result, dict):
            for key, value in result.items():
                print(f"{key} = {value}")
        else:
            print(result)
        return result
    return wrapper


# Примеры использования 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()
```

Пример выполнения программы



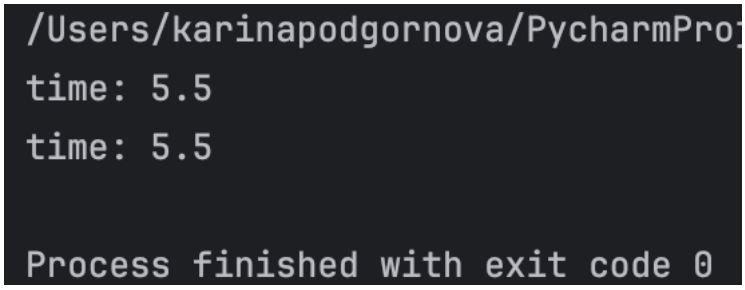
Задача 6 (файл cm_timer.py)

Текст программы

```
import time
from contextlib import contextmanager


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

    def __exit__(self, exc_type, exc_value, traceback):
        end_time = time.time()
        elapsed_time = end_time - self.start_time
        print(f'time: {elapsed_time:.1f}')

# Контекстный менеджер с использованием contextlib
@contextmanager
def cm_timer_2():
    start_time = time.time()  # Запоминаем время начала
```

```
    try:
        yield
    finally:
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f'time: {elapsed_time:.1f}')

if __name__ == "__main__":
    from time import sleep

    with cm_timer_1():
        sleep(5.5)

    with cm_timer_2():
        sleep(5.5)
```

Пример выполнения программы



Задача 7 (файл process_data.py)

Текст программы

```
import json
import sys
import time
from contextlib import contextmanager

path = "/Users/karinapodgornova/Desktop/data_light1.json"

with open(path, encoding="utf8") as f:
    data = json.load(f)

@contextmanager
def cm_timer_1():
    start_time = time.time()
```

```
    yield
    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"time: {elapsed_time}")

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

@print_result
def f1(arg):
    return sorted(list(set([job['job-name'].lower() for job in arg])))

@print_result
def f2(arg):
    return list(filter(lambda job: job.startswith('программист'), arg))

@print_result
def f3(arg):
    return list(map(lambda job: job + ', с опытом Python', arg))

@print_result
def f4(arg):
    salary = [str(i) for i in range(100000, 200001)]
    return list(zip(arg, salary))

if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Вывод

Я изучила возможности функционального программирования в языке Python.