Студент: Лачина Екатерина
Группа: ИУ5-33Б

Отчет по лабораторной работе №3-4
«Функциональные возможности языка Python.»

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

Постановка задачи:

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 1 (файл field.py)

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.
В качестве первого аргумента генератор принимает список словарей, дальше через *args генератор принимает неограниченное количествово аргументов.
Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.
Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно None, то оно пропускается.
Если все поля содержат значения None, то пропускается элемент целиком.

Задача 2 (файл gen_random.py)

Необходимо реализовать генератор gen_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона.

Задача 3 (файл unique.py)

Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.
Конструктор итератора также принимает на вход именованный bool-параметр ignore_case, в зависимости от значения которого будут считаться

одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.
При реализации необходимо использовать конструкцию **kwargs.
Итератор должен поддерживать работу как со списками, так и с генераторами.
Итератор не должен модифицировать возвращаемые значения.

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.

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.

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.
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.

Python Realisation:

__init__.py

cm_timer.py

```
from contextlib import contextmanager
from typing import Self, Any, Generator
import time


class Timer:
    def __enter__(self) -> Self:
        self.IN = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb) -> None:
        self.OUT = time.time() - self.IN
```

```
        print(self.OUT)


@contextmanager
def cm_timer() -> Generator[Any, Any, Any]:
    start = time.time()
    yield
    print(time.time() - start)


if __name__ == "__main__":
    pass
```

field.py

```
def field(items: list, *args) -> None:
    for item in items:
        for key in args:
            if item[key]:
                print(item[key])


def main():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
        ]
    field(goods, 'title')
    field(goods, 'title', 'price')


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

gen_random.py

```
from random import randint
```

```
def gen_random(num_count: int, begin: int, end: int):
    for i in range(num_count):
        print(randint(begin, end))


def main():
    gen_random()


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

print_result.py

```
from typing import Callable, Any


def print_result(func: Callable) -> Callable:
    def inner(*args, **kwargs) -> Any:
        res = func(*args, **kwargs)
        if isinstance(res, int):
            print(res)
        elif isinstance(res, list):
            for i in res:
                print(i)
        elif isinstance(res, dict):
            for key, value in res.items():
                print(key, " = ", value)
        return res

    return inner


@print_result
def test_1() -> int:
    return 1
```

```python
@print_result
def test_2() -> str:
    return 'iu5'


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


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


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

process_data.py

```python
import json
from print_result import print_result
from random import randint
from cm_timer import Timer


path = "lab_python_fp/data_light.json"
```

```
with open(path) as f:
    data = json.load(f)


@print_result
def f1(data) -> None:
    return list(sorted(i["job-name"].lower() for i in data))


@print_result
def f2(data) -> list:
    return list(filter(lambda x: x.startswith("программист"), data))


@print_result
def f3(data):
    return list(map(lambda x: x + " с опытом Python", data))

@print_result
def f4(data):
    for occupation, salary in zip(data, [randint(100000, 200000) for i in
range(len(data))]):
        print(f"Occupation: {occupation} - with salary {salary} rub")


if __name__ == "__main__":
    with Timer():
        print(f4(f3(f2(f1(data)))))
```

sort.py

```
def sort(items: list) -> list:
    return list(sorted(items))


lambda_sort = lambda items: sorted(items)
```

```
if __name__ == "__main__":
    print(sort(["amanda", "zuck", "bob"]))
    print(lambda_sort([5, 3, 7]))
```

unique.py

```
class Unique:
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case', False)
        self.seen = set()
        self.iterator = iter(items)

    def __next__(self):
        while True:
            item = next(self.iterator)
            comparison_value = item.lower() if self.ignore_case and isinstance(item,
str) else item

            if comparison_value not in self.seen:
                self.seen.add(comparison_value)
                return item

    def __iter__(self):
        return self
```

main.py

```
from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random as random
from lab_python_fp.sort import lambda_sort, sort
from lab_python_fp.print_result import *
from lab_python_fp.cm_timer import *

if __name__ == "__main__":
    pass
```