Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторным работам №3-4 «Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-32Б
Мажитов В.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Описание задания

Задание лабораторной работы состоит из решения нескольких задач.

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

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Задача 1

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

Пример:

goods = [
{'title': 'Ковер', 'price': 2000, 'color': 'green'},
{'title': 'Диван для отдыха', 'color': 'black'}
]

field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'

field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

- 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.

Файл field.py

```python
def field(items, *args):
    assert len(args) > 0
    for item in items:
        result = {}
        for key in args:
            if key in item and item[key] is not None:
                result[key] = item[key]

        if len(result) == 1:
            yield list(result.values())[0]
        elif len(result) > 1:
            yield result


if __name__ == "__main__":
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]
    print(list(field(goods, 'title')))  # 'Ковер', 'Диван для отдыха'
    print(list(field(
        goods,
        'title',
```

```
        'price',
    ))  # {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха',
'price': 5300}
```

```
❯ python3 field.py
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]
```

Задача 2

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. Пример:

gen_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

Файл gen_random.py

```
import random


def gen_random(num_count, begin, end):
    for _ in range(num_count):
        yield random.randint(begin, end)


if __name__ == "__main__":
    print(list(gen_random(5, 1, 3)))  # [3, 2, 2, 2, 3]
```

```
❯ python3 gen_random.py
[3, 1, 1, 1, 1]
```

Задача 3

- 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.

Файл unique.py

```
from gen_random import gen_random


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

    def _get_element_for_comparing(self, el):
        if isinstance(el, str):
            return el.lower() if self.ignore_case else el
```

```
        return el

    def __next__(self):
        while True:
            element = next(self.items)

            if self._get_element_for_comparing(element) not in self.returned_items:
                found = True
                break

        if found:
            self.returned_items.add(self._get_element_for_comparing(element))
            return element

        raise StopIteration

    def __iter__(self):
        return self


if __name__ == "__main__":
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print(list(Unique(data)))  # [1, 2]

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print(list(Unique(data)))  # ['a', 'A', 'b', 'B']

    data = gen_random(10, 1, 3)
    print(list(Unique(data)))  # [2, 1, 3]

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print(list(Unique(data, ignore_case=True)))  # ['a', 'b']
```

```
❯ python3 unique.py
[1, 2]
['a', 'A', 'b', 'B']
[3, 1, 2]
['a', 'b']
```

Задача 4

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.

Файл sort.py

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

if __name__ == '__main__':
    result = sorted(data, key=abs, reverse=True)  # [123, 100, -100, -30, 4, -4, 1,
-1, 0]
    print(result)

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)  # [123,
100, -100, -30, 4, -4, 1, -1, 0]
    print(result_with_lambda)
```

```
❯ python3 sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
```

Задача 5

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.

Файл print_result.py

```
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


@print_result
def test_1():
    return 1


@print_result
def test_2():
    return 'iu5'
```

```python
@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()
```

```
❯ python3 print_result.py
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

Задача 6

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

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).

Файл cm_timer.py

```python
import time
from contextlib import contextmanager


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

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.end = time.time()
        delta = self.end - self.begin
        print(f"time: {delta}")


@contextmanager
def cm_timer_2():
```

```
    begin = time.time()
    yield
    delta = time.time() - begin
    print(f"time: {delta}")


if __name__ == "__main__":
    with cm_timer_1():  # time: 5.505076169967651
        time.sleep(5.5)

    with cm_timer_2():  # time: 5.504739761352539
        time.sleep(5.5)
```

Задача 7

- 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.

Файл process_data.py

```
import json

from cm_timer import cm_timer_1
from field import field
from gen_random import gen_random
from print_result import print_result
from unique import Unique

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

```python
# 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):
    return list(Unique(field(arg, "job-name")))


@print_result
def f2(arg):
    return list(filter(lambda x: x.startswith("Программист"), arg))


@print_result
def f3(arg):
    return list(map(lambda s: f"{s} с опытом Python", arg))


@print_result
def f4(arg):
    salaries = gen_random(len(arg), 100000, 200000)
    return [f"{profession}, зарплата {salary} руб." for profession, salary in
zip(arg, salaries)]


if __name__ == '__main__':
    with open(path) as f:
        data = json.load(f)

    with cm_timer_1():
        f4(f3(f2(f1(data))))

    # Вывод в файле process_data.output.txt
```

```
Программист C++/C#/Java с опытом Python
Программист 1С с опытом Python
Программистр-разработчик  информационных систем с опытом Python
Программист C++ с опытом Python
Программист/ Junior Developer с опытом Python
Программист / Senior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программист C# с опытом Python
f4
Программист с опытом Python, зарплата 117224 руб.
Программист C++/C#/Java с опытом Python, зарплата 196903 руб.
Программист 1С с опытом Python, зарплата 130847 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 117240 руб.
Программист C++ с опытом Python, зарплата 157738 руб.
Программист/ Junior Developer с опытом Python, зарплата 167587 руб.
Программист / Senior Developer с опытом Python, зарплата 123653 руб.
Программист/ технический специалист с опытом Python, зарплата 101665 руб.
Программист C# с опытом Python, зарплата 116638 руб.
time: 0.0062541961669921875
```