Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Базовые компоненты интернет-технологий»

Отчет по Лабораторной работе 3-4.

Выполнил:
студент группы ИУ5-33Б
Комаров Дмитрий

Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Юрий Евгеньевич Гапанюк

Подпись и дата:

Москва, 2022 г.

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.

Код программы:

```
def field(items, *args):
    stri = ''
    assert len(args) > 0
    if len(args) == 1:
        for dic in items:
            value = dic.get(args[0])
            if value is not None:
                 yield value
    else:
        for item in items:
            res = {}
            for key in args:
                value = item.get(key)
                if value is not None:
                    res[key] = value
            yield res

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'
    for el in field(goods, 'title', 'price'):
        print(el) # {'title': 'Ковер', 'price': 2000}, {'title': 'Диван
для отдыха', 'price': 5300}
```

Тесты:

```
userdead@DESKTOP-439N993:/mnt/c/Users/dimka/B
['Ковер', 'Диван для отдыха']
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха', 'price': 5300}
```

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.

Код программы:

```python
from random import randint


def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield randint(begin, end)

if __name__ == '__main__':
    print(*gen_random(5, 1, 3), sep=', ')
```

Тесты:

```
userdead@DESKTOP-439N993:/mnt/c/Users/dimka/BKIT_Labs/lab3-4/lab_python_fp$ python3 gen_random.py
3, 1, 2, 1, 3
```

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.

Код программы:

```
from gen_random import gen_random



class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = set(items)
        self.used_elenets = set()
        self.iter = iter(self.items)
        self.ignore_case = kwargs.get('ignore_case', False)
        if self.ignore_case == True:
            self.items={i.lower() for i in items}
        self.iter = iter(self.items)
    def __next__(self):
        for current in self.iter:
                return current
        raise StopIteration
    def __iter__(self):
        return self

if __name__ == '__main__':
    data = [
        [1, 1, 1, 1, 1, 2, 2, 2, 2, 2],
        gen_random(10, 1, 3),
        ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    ]
    for one in data:
        print(list(Unique(one)))
    print(list(Unique(data[-1], ignore_case=True)))
```

Тесты:

```
userdead@DESKTOP-439N993:/mnt/c/Users/dimka/BKIT_Labs/lab3-4/lab_python_fp$ python3 unique.py
[1, 2]
[1, 2, 3]
['a', 'B', 'b', 'A']
['a', 'b']
```

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.

Код программы:

```
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)
```

Тесты:

```
userdead@DESKTOP-439N993:/mnt/c/Users/dimka/BKIT_Labs/lab3-4/lab_python_fp$ python3 sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
```

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.

Код программы:

```
def print_result(f):
    def wrapper(*args, **kwargs):
        res = f(*args, **kwargs)
        print(f.__name__)
```

```
        if type(res) == list:
            for el in res:
                print(el)
        elif type(res) == dict:
            for key, value in res.items():
                print(key, '=', value)
        else:
            print(res)
        return res
    return wrapper

@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()
```

Тесты:

```
userdead@DESKTOP-439N993:/mnt/c/Users/dimka/BKIT_Labs/lab3-4/lab_python_fp$ python3 print_result.py
!!!!!!!!
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

Код программы:

```python
from contextlib import contextmanager
from time import perf_counter, sleep, time

class cm_timer2():
    def __init__(self):
        self.start = 0
        self.end = 0

    def __enter__(self):
        self.start = time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.end = time()
        print(f'time: {self.end-self.start:0.4f}')


@contextmanager
def cm_timer_1():
    start = time()
    yield
    end = time()
    print(f'time: {end-start:0.4f}')

def main():
    with cm_timer_1():
        sleep(5.5)

    with cm_timer2():
```

```
        sleep(5.5)


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

Тесты:

```
time: 5.5002
time: 5.5005
```

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.

Код программы:

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

@print_result
def f1(arg):
    return sorted(Unique(list(field(arg, 'job-name')),
ignore_case=True))

@print_result
def f2(arg):
    return list(filter(lambda x: x[:11] == 'программист', arg))

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

@print_result
def f4(arg):
    return list(zip(arg, map(lambda x: 'зарплата ' +str(x) + str('
руб.') ,gen_random(len(arg), 100000, 200000))))

def main():
    with open('data_light.json') as f:
        data = json.load(f)
    with cm_timer_1():
       f4(f3(f2(f1(data))))

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

Тесты:

```
f4
('программист с опытом Python', 'зарплата 106058 руб.')
('программист / senior developer с опытом Python', 'зарплата 150333 руб.')
('программист 1с с опытом Python', 'зарплата 174935 руб.')
('программист c# с опытом Python', 'зарплата 175364 руб.')
('программист c++ с опытом Python', 'зарплата 137999 руб.')
('программист c++/c#/java с опытом Python', 'зарплата 182271 руб.')
('программист/ junior developer с опытом Python', 'зарплата 191308 руб.')
('программист/ технический специалист с опытом Python', 'зарплата 194905 руб.')
('программистр-разработчик  информационных систем с опытом Python', 'зарплата 123868 руб.')
time: 0.2095
```