Московский государственный технический университет им. Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5. Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №3

Выполнил:

Студент группы ИУ5-51Б

Ноздрова Валентина

Дата: 10.10.2021

Подпись:

Проверил:

Гапанюк Ю. Е.

Дата:

Подпись:

Москва, 2021 г.

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 1 (файл field.py)

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

`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.

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. Пример:

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

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.

Необходимо решить задачу двумя способами:

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

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.
`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.

Листинг программы:

```
----------------------------------------field.py----------------------------------------
def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
    if len(args) == 1:
        for j in items:
            a = j.get(args[0])
            if a is not None:
                yield a
    else:
        for j in items:
            a = {i: j.get(i) for i in args if j.get(i) is not None}
            if a is not None:
                yield a


-----------------------------------gen_random.py-----------------------------------------
import random
# Пример:
# gen_random(5, 1, 3) должен выдать выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки


def gen_random(num_count, begin, end):
    # Необходимо реализовать генератор
    for i in range(0, num_count):
        yield random.randint(begin, end)


-----------------------------------unique.py-----------------------------------------

# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        # Нужно реализовать конструктор
```

```
        # В качестве ключевого аргумента, конструктор должен принимать bool-параметр
ignore_case,
        # в зависимости от значения которого будут считаться одинаковыми строки в
разном регистре
        # Например: ignore_case = False, Абв и АБВ - разные строки
        #           ignore_case = True, Абв и АБВ - одинаковые строки, одна из которых
удалится
        # По-умолчанию ignore_case = False
        self.case_sens = kwargs.get('ignore_case')
        self.set = set()
        self.items = iter(items)

    def __next__(self):
        # Нужно реализовать __next__
        a = next(self.items)
        if self.case_sens:
            if a.lower() not in self.set:
                self.set.add(a.lower())
            else:
                a = self.__next__()
        else:
            if a not in self.set:
                self.set.add(a)
            else:
                a = self.__next__()
        return a

    def __iter__(self):
        return self

----------------------------------------sort.py----------------------------------------
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

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

    result_with_lambda = sorted(data, reverse=True, key=lambda x: abs(x))
    print(result_with_lambda)

----------------------------------print_result.py-----------------------------------

def print_result(myfunc):
    def wrapper(arg):
        print(myfunc.__name__)
        res = myfunc(arg)
        if type(res) == dict:
            for i in res:
                print('{} = {}'.format(i, res[i]))
        elif type(res) == list:
            for i in res:
                print(i)
        else:
            print(res)
        return res
    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()
```

----------cm_timer.py----------

```python
import time
from contextlib import contextmanager


class cm_timer_1():
    def __init__(self):
        pass

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('time:', time.perf_counter()-self.start)


@contextmanager
def cm_timer_2():
    a = time.perf_counter()
    yield
    print('time:', time.perf_counter()-a)
```

----------process_data.py----------

```python
import json
import sys
from lab_python_fp.unique import Unique
from lab_python_fp.field import field
from lab_python_fp.cm_timer import cm_timer_1
from lab_python_fp.gen_random import gen_random
from lab_python_fp.print_result import print_result


path = r'C:\Users\user\PycharmProjects\RIP\LR 3\data_light.json'
# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

with open(path, encoding='utf8') 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):
    return sorted(list(Unique(field(arg, 'job-name'), ignore_case=True)),
key=str.casefold)
```

```python
@print_result
def f2(arg):
    return list(filter(lambda x: x.lower().startswith('программист'), arg))


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


@print_result
def f4(arg):
    a = gen_random(len(arg), 100000, 200000)
    return [i+', зарплата {} руб.'.format(j) for i, j in zip(arg, a)]


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

Результаты выполнения:

```
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем

f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С с опытом Python
Программист C# с опытом Python
Программист C++ с опытом Python
Программист C++/C#/Java с опытом Python
Программист/ Junior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программистр-разработчик  информационных систем с опытом Python
f4
Программист с опытом Python, зарплата 175975 руб.
Программист / Senior Developer с опытом Python, зарплата 103175 руб.
Программист 1С с опытом Python, зарплата 106186 руб.
Программист C# с опытом Python, зарплата 119276 руб.
Программист C++ с опытом Python, зарплата 151091 руб.
Программист C++/C#/Java с опытом Python, зарплата 118691 руб.
Программист/ Junior Developer с опытом Python, зарплата 111666 руб.
Программист/ технический специалист с опытом Python, зарплата 107965 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 159100 руб.
time: 0.03943390000000002

Process finished with exit code 0
```