Московский государственный
технический университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчёт по лабораторной работе № 1

Выполнил:
студент группы РТ5-31Б
Сафонов Федор
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

2021

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. Пример:

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. Пример:

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.

Текст программы

cm_timer.py

```
import time, sys
from contextlib import contextmanager

class cm_timer_1:
    def __init__(self):
        return
    def __enter__(self):
        self.time = time.time()
        return self.time
    def __exit__(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
```

```
        else:
            print(f'{time.time() - self.time:.2f}')

@contextmanager
def cm_timer_2():
    start = time.time()
    yield
    print(f'{time.time() - start:.2f}')
if __name__ == '__main__':
    with cm_timer_1() as obj:
        time.sleep(5.5)

    with cm_timer_2() as obj:
        time.sleep(5.5)
```

filed.py

```
goods = [                       # список словарей для теста
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title' : None, 'price' : 100, 'color' : 'white'},
    {'title' : None, 'price' : None, 'color' : None}
]

def field(items, *args):
    assert len(args) > 0
    key = args[0]
    if len(args) == 1:
        for dict in items: # последовательно выдаем значения полей
            if dict[key]:  # элемент не None
                yield dict[key]
    else:
        for dict in items: # фиксируем словарь
            res_dict = {}
            for key in args: # находим в нем все нужные ключи
                if dict[key]: # элемент не None
                    res_dict[key] = dict[key]
            if len(res_dict) != 0:
                yield res_dict # последовательно возвращаем нужный нам подсловарь
            res_dict.clear()

print('TEST 1')
for elem in field(goods, 'title'):
    print(elem)
print('TEST 2')
for elem in field(goods, 'title', 'price'):
    print(elem)
```

gen_random.py

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

```
def main():
    for elem in gen_random(5, 1, 3):
        print(elem)

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

print_result.py

```
def print_result(funct):
    def wrapper():
        print(funct.__name__)
        res = funct()
        if isinstance(res, list):
            for elem in res:
                print(elem)
            return
        if isinstance(res, dict):
            for key, val in res.items():
                print(f'{key} = {val}')
            return
        print(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()
```

process_data.py

```
from cm_timer import *
from unique import *
from gen_random import *
```

```python
import json

def print_result(funct):
    def wrapper(arg):
        print(funct.__name__)
        res = funct(arg)
        if isinstance(res, list):
            for elem in res:
                print(elem)
            return res
        if isinstance(res, dict):
            for key, val in res.items():
                print(f'{key} = {val}')
            return res
        return res
    return wrapper

path = 'data_light.json'

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария
with open('data_light.json') 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(Unique([x['job-name'].lower() for x in arg]))

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

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

@print_result
def f4(arg):
    return list(zip(arg, [rnd for rnd in gen_random(len(arg), 100_000, 200_000)]))

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

sort.py

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

if __name__ == '__main__':
    print("TEST 1")
```

```
    result = list(zip(*sorted([(-x**2, x) for x in data], key=operator.itemgetter(0))))[1]
    print(*result)
    print("DATA")
    print(data)
    print("TEST 2")
    result_with_lambda = sorted(data, key=lambda x : -x**2)
    print(*result_with_lambda)
```

unique.py

```
from Lab3.lab3_package.gen_random import gen_random
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        if 'ignore_case' in kwargs.keys():
            self.ignore_case = kwargs['ignore_case']
        else:
            self.ignore_case = False

        if self.ignore_case:
            self.data = list(map(lambda s : str(s).lower(), items)) # приведение элементов к строке без
учета регистра
        else:
            self.data = list(map(str, items)) # приведение элементов к строке с учетом регистра

        self.used_elements = set()
        self.index = 0
        pass

    def __next__(self):
        while True:
            if self.index >= len(self.data):
                raise StopIteration
            else:
                current = self.data[self.index]
                self.index += 1
                if current not in self.used_elements: # возвращаем все эл-ты не встреющиеся ранее
                    self.used_elements.add(current)
                    return current
        pass

    def __iter__(self):
        return self

# Тестирование на примерах
if __name__ == '__main__':
    print("TEST 1")
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for elem in Unique(data):
        print(elem)
    print("TEST 2")
    data = gen_random(10, 1, 3)
    for elem in Unique(data):
        print(elem)
    print("TEST 3")
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
```

```
for elem in Unique(data):
    print(elem)
print("TEST 4")
for elem in Unique(data, ignore_case=True):
    print(elem)
```

Результат работы программы

filed.py:

TEST 1
Ковер
Диван для отдыха
TEST 2
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха', 'price': 5300}
{'price': 100}

gen_random.py:

2
3
3
2
1

print_result.py:

test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

sort.py

TEST 1
123 100 -100 -30 4 -4 1 -1 0

DATA
[4, -30, 100, -100, 123, 1, 0, -1, -4]
TEST 2
123 100 -100 -30 4 -4 1 -1 0

unique.py

TEST 1
1
2
TEST 2
3
1
2
TEST 3
a
A
b
B
TEST 4
a
b