Московский государственный технический университет имени Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5

Отчёт по
лабораторной работе № 3
«Разработка интернет приложений»

Подготовил:

Кан Андрей Дмитриевич

Группа ИУ5-54Б

Подпись_____

Дата_____

Москва
2020г.

1.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.

2. Задача 1 (файл field.py)
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.

Текст программы:

```
def field(items, *args): # генератор, возвращающий поля с заданным ключом и словари
    assert len(items) > 0 and isinstance(items, list)
    if len(args) == 1:
        for i in range(len(items)):
            for key, value in items[i].items():
                if key == args[0]:  # если передан один аргумент, возвращаем поля с ним
                    yield value

    else:
        for i in range(len(items)):
            dictionary = {}
            for key, value in items[i].items():
                for arg in args:    # если передано много аргументов, то возвращаем словари
```

```
сзаданными ключами
                if key == arg:
                    dictionary[key] = items[i][key]
                    break   # как только нужный аргумент подошел к ключу, выходим из
цикла (оптимизация)
            yield dictionary
```

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

Текст программы:

```
import random


def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield random.randrange(begin, end+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.

Пример:

```
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
```

Unique(data) будет последовательно возвращать только 1 и 2.

```
data = gen_random(1, 3, 10)
```

Unique(data) будет последовательно возвращать только 1, 2 и 3.

```
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
```

Unique(data) будет последовательно возвращать только a, A, b, B.
Unique(data, ignore_case=True) будет последовательно возвращать только a, b.

Текст программы:

```
class Unique:
    def __init__(self, data, **kwargs):
        self.used_elements = set()
        self.data = list(data)
        self.index = 0
        self.ignore_case = False
        for key, value in kwargs.items():
            if key == 'ignore_case' and isinstance(value, bool):
                self.ignore_case = value

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            if self.index >= len(self.data):
                raise StopIteration
            else:
                current = self.data[self.index]
                self.index = self.index + 1
                flag = False
                if self.ignore_case and isinstance(current, str):
                    for elem in self.used_elements:
                        if current.upper() == elem.upper():
                            flag = True
                            break
                    if not flag:
                        self.used_elements.add(current)
                        return current

                else:
                    if current not in self.used_elements:
                        self.used_elements.add(current)
                        return current
```

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. 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.

Текст программы:

```
def sort(arr):
    return sorted(arr, reverse=True)
```

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.

Текст программы:

```
def print_result(func):
    def decorator(*args, **kwargs):
        print("Имя функции = "+func.__name__)
        res = func(*args, **kwargs)
        if isinstance(res, list):
            for i in res:
                print(i)
        elif isinstance(res, dict):
```

```
            for key, value in res.items():
                print("{}={}".format(key, value))
        else:
            print(res)
        return res
    return decorator
```

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:

```
with cm_timer_1():
    sleep(5.5)
```

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.

Текст программы:

```
import time
from contextlib import contextmanager

# Пока мы в блоке with будет работать yield, после выхода из блока мы вызовем метод stop
@contextmanager
def cm_timer2():
    t = cm_timer1()
    t.start()
    yield t
    t.stop()

class TimerError(Exception):
    """A custom exception used to report errors in use of Timer class"""

# Пока мы в блоке with object будет иметь значение, возвращенное с метода __enter__
# После выхода из блока автоматически вызовется метод __exit__
class cm_timer1:
    def __init__(self):
        self._start_time = None

    def start(self):
        """Start a new timer"""
        if self._start_time is not None:
            raise TimerError(f"Timer is running. Use .stop() to stop it")

        self._start_time = time.perf_counter()
```

```
    def stop(self):
        """Stop the timer, and report the elapsed time"""
        if self._start_time is None:
            raise TimerError(f"Timer is not running. Use .start() to start it")

        elapsed_time = time.perf_counter() - self._start_time
        self._start_time = None
        print(f"Elapsed time: {elapsed_time:0.4f} seconds")

    def __enter__(self):
        self.start()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.stop()
```

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:

```
from lab_python_fp.print_result import *
from lab_python_fp.sort import sort
from lab_python_fp.field import field
from lab_python_fp.unique import Unique
from lab_python_fp.gen_random import gen_random
import sys


# 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(data):
    return sort(Unique(field(data, "job-name"), ignore_case=True)) # удаление дубликатов и сортировка по алфавиту

@print_result
def f2(f1arr):
    return list(filter(lambda x: x[:11].upper() == "программист".upper(), f1arr))

@print_result
def f3(f2arr):
    return list(map(lambda x: x + " с опытом Python", f2arr))

@print_result
def f4(f3arr):
    pairs = list(zip(f3arr, [" с зарплатой {} рублей".format(zp) for zp in gen_random(len(f3arr), 100000, 200000)]))
    return ["{}{}".format(pair[0], pair[1]) for pair in pairs]
```

Текст программы main.py:

```
from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.sort import sort
from lab_python_fp.print_result import print_result
from lab_python_fp.cm_timer import *
import time
from lab_python_fp.process_data import *
import json

@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]


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]
    print("Example Gen ")
    print(list(field(goods, 'title')))
    print(list(field(goods, 'title', 'price')))
    count = 5
    print("Example Gen_Random")
    for i in gen_random(count, 1, 3):
        print(i)
    print("Example Unique iterator")
    data = [1, 1, 1, 1, 2, 2, 2, 2]
    data1 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for i in Unique(data, ignore_case=True):
        print(i)
    for i in Unique(data1, ignore_case=True):
        print(i)
    data = gen_random(4, 1, 10)
    for i in Unique(data, ignore_case=True):
        print(i)
    print("Example sort")
    data2 = [4, -30, 100, -100, 123, 1, 0, -1, -4]
    res = sort(data2)
    print(res)
    res_lambda = (lambda arr: sort(arr))(data2)
    print(res_lambda)
```

```
    print("Example decorator")
    test_1()
    test_2()
    test_3()
    test_4()

    print("Example timer")
    with cm_timer1():
        time.sleep(0.1)
    with cm_timer2():
        time.sleep(0.1)

    path = "./package.json"

    # Необходимо в переменную path сохранить путь к файлу, который был передан при
запуске сценария

    with open(path) as f:
        data = json.load(f)  # список словарей

    with cm_timer1():
        print(f4(f3(f2(f1(data)))))
```

3. Результат работы (для краткости в пункте с декоратором опустим декоратор для функции f1, так как результат работы занимает слишком много места)

```
Example Gen
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]
Example Gen_Random
3
2
2
3
2
Example Unique iterator
1
2
a
b
6
3
2
5
Example sort
[123, 100, 4, 1, 0, -1, -4, -30, -100]
[123, 100, 4, 1, 0, -1, -4, -30, -100]
Example decorator
Имя функции = test_1
1
Имя функции = test_2
iu5
Имя функции = test_3
```

a=1
b=2
Имя функции = test_4
1
2
Example timer
Elapsed time: 0.1004 seconds
Elapsed time: 0.1002 seconds
Имя функции = f2
Программистр-разработчик информационных систем
Программист/ технический специалист
Программист/ Junior Developer
Программист C++/C#/Java
Программист C++
Программист C#
Программист 1C
Программист / Senior Developer
Программист
Имя функции = f3
Программистр-разработчик информационных систем с опытом Python
Программист/ технический специалист с опытом Python
Программист/ Junior Developer с опытом Python
Программист C++/C#/Java с опытом Python
Программист C++ с опытом Python
Программист C# с опытом Python
Программист 1C с опытом Python
Программист / Senior Developer с опытом Python
Программист с опытом Python
Имя функции = f4
Программистр-разработчик информационных систем с опытом Python с зарплатой 133519 рублей
Программист/ технический специалист с опытом Python с зарплатой 155590 рублей
Программист/ Junior Developer с опытом Python с зарплатой 106920 рублей
Программист C++/C#/Java с опытом Python с зарплатой 148517 рублей
Программист C++ с опытом Python с зарплатой 134452 рублей
Программист C# с опытом Python с зарплатой 186513 рублей
Программист 1C с опытом Python с зарплатой 190786 рублей
Программист / Senior Developer с опытом Python с зарплатой 111457 рублей
Программист с опытом Python с зарплатой 182060 рублей
['Программистр-разработчик информационных систем с опытом Python с зарплатой 133519 рублей', 'Программист/ технический специалист с опытом Python с зарплатой 155590 рублей', 'Программист/ Junior Developer с опытом Python с зарплатой 106920 рублей', 'Программист C++/C#/Java с опытом Python с зарплатой 148517 рублей', 'Программист C++ с опытом Python с зарплатой 134452 рублей', 'Программист C# с опытом Python с зарплатой 186513 рублей', 'Программист 1C с опытом Python с зарплатой 190786 рублей', 'Программист / Senior Developer с опытом Python с зарплатой 111457 рублей', 'Программист с опытом Python с зарплатой 182060 рублей']
Elapsed time: 1.5208 seconds

Process finished with exit code 0