Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №3

Выполнил:
студент группы ИУ5-54Б

Ли М.В.

Преподаватель:

Гапанюк Ю.Е

2020

Цель работы

Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

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. 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.

Код программы:

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
]


def field(items, *args):
            assert len(args) > 0
            if len(args) == 1:
```

```
                for dic in items:
                    for arg in args:
                        if arg in dic:
                            yield dic[arg]
            else:
                for dic in items:
                    new_item = {}
                    for arg in args:
                        if arg in dic:
                                new_item[arg] = dic[arg]
                    if len(new_item.keys()) > 0:
                        yield new_item


def main():
    print(list(field(goods, 'title')))
    print(list(field(goods, 'title', 'price')))
    print(list(field(goods, 'title', 'price', 'color')))


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

Вывод программы:

```
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]

Process finished with exit code 0
```

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`
Шаблон для реализации генератора:

Код программы:

```
import random



def gen_random(num_count, begin, end):
    return [random.randint(begin, end) for x in range(num_count)]


print(gen_random(5, 1, 3))
```

Вывод программы:

```
[2, 2, 1, 1, 2]

Process finished with exit code 0
```

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:

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = list(items)
        self.uniqItems = list()
        self.index = 0
        try:
            self.ignore_case = bool(kwargs['ignore_case'])
        except (KeyError, TypeError):
            self.ignore_case = False

    def __next__(self):
        while True:
            if self.index >= len(self.items):
                raise StopIteration
            else:
                current = self.items[self.index]
                if (self.ignore_case == True) & (type(current) == str):
                    current = current.upper()
                if current not in self.uniqItems:
                    self.uniqItems.append(current)
                    return self.items[self.index]
                self.index = self.index + 1

    def __iter__(self):
        return self
```

Вывод программы:

```
а
б
в
а
б
в
в
```

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.

Код программы:

```python
def sort(x):
    return abs(x)


def main():
    data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
    print('Исходный массив:', data)
    result = sorted(data, key=sort, reverse=True)
    print('Отсортированный массив:', result)

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


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

Вывод программы:

```
Исходный массив: [4, -30, 100, -100, 123, 1, 0, -1, -4]
Отсортированный массив: [123, 100, -100, -30, 4, -4, 1, -1, 0]
Отсортированный массив: [123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 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(func):
    def decor(*arg):
        result = func(*arg)
        print(func.__name__)
        if type(result) is list:
            for i in result:
                print(i)
        elif type(result) is dict:
            for key, value in result.items():
                print(str(key) + ' = ' + str(value))
        else:
            print(result)
        return result

    return decor


@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()
```

Вывод программы:

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

Process finished with exit code 0
```

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.

Код программы:

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


class cm_timer_1():

    def init(self):
        self.timer = 0

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

    def __exit__(self, exp_type, exp_value, traceback):
        print(time() - self.timing)


@contextmanager
def cm_timer_2():
    timer = time()
    yield
    print(time() - timer)


if __name__ == "__main__":
```

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

    with cm_timer_2():
        sleep(5.5)
```

Вывод программы:

```
5.515727996826172
5.5065062046051025

Process finished with exit code 0
```

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:

```
import json
from Task6.gen_random import gen_random
from Task6.print_result import print_result
from Task6.unique import Unique
from Task6.cm_timer import cm_timer_1
from Task6.field import field



path = "C:/Lab3/data_light.json"


with open(path, encoding='utf8') as f:
    data = json.load(f)


@print_result
def f1(arg):
    return sorted(list(Unique(list(field(arg, 'job-name')),
ignore_case=True)), key=lambda x: str.casefold(x))


@print_result
def f2(arg):
    return list(filter(lambda x: "программист" in x.lower(), arg))


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


@print_result
def f4(arg):
    return dict(zip(arg, gen_random(len(arg), 100000, 200000)))


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

Вывод программы:

```
Программист C++/C#/Java с опытом Python = 133273
Программист/ Junior Developer с опытом Python = 191546
Программист/ технический специалист с опытом Python = 141065
Программистр-разработчик  информационных систем с опытом Python = 103368
Системный программист (C, Linux) с опытом Python = 103810
Старший программист с опытом Python = 169013
0.10931754112243652

Process finished with exit code 0
```