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)

ФАКУЛЬТЕТ _____ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ (ИУ)_____

КАФЕДРА _____СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)___

О Т Ч Е Т

по лабораторной работе № 2

по дисциплине: Разработка интернет-приложений_____

на тему:__Python. Функциональные возможности_____
_____
_____
_____
_____
_____
_____
_____

Студент ____ИУ5-53____ _____ **А.С. Волков**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

2019 г.

1. Задание

Важно выполнять все задачи последовательно. С 1 по 5 задачу формируется модуль librip, с помощью которого будет выполняться задание 6 на реальных данных из жизни. Весь вывод на экран (даже в столбик) необходимо запрограммировать одной строкой.

Подготовительный этап:

1. Зайти на github.com и выполнить fork проекта с заготовленной структурой https://github.com/iu5team/ex-lab4

2. Переименовать репозиторий в lab_2

3. Выполнить git clone проекта из вашего репозитория

2. Задача 1 (ex_1.py)

2.1. Формулировка задания

Необходимо реализовать генераторы field и gen_random

Генератор field последовательно выдает значения ключей словарей массива

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 list, дальше через *args генератор принимает неограниченное кол-во аргументов.

Если передан один аргумент, генератор последовательно выдает только значения полей, если поле равно None, то элемент пропускается

Если передано несколько аргументов, то последовательно выдаются словари, если поле равно None, то оно пропускается, если все поля None, то пропускается целиком весь элемент

Генератор gen_random последовательно выдает заданное количество случайных чисел в заданном диапазоне

Пример:

gen_random(1, 3, 5)должен выдать 5 чисел от 1 до 3, т.е. примерно 2, 2, 3, 2, 1

В ex_1.py нужно вывести на экран то, что они выдают, с помощью кода в одну строку

Генераторы должны располагаться в librip/gen.py

2.2. Исходный код librip/gens.py

```
import random


# Генератор вычленения полей из массива словарей
# Пример:
# goods = [
#    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
#    {'title': 'Диван для отдыха', 'price': 5300, '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', 'price': 5300}


def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for item in items:
            if args[0] in item:
                yield item[args[0]]
    else:
        for item in items:
            d = {arg: item[arg] for arg in args if arg in item}
            if len(d) > 0:
                yield d


# Генератор списка случайных чисел
# Пример:
# gen_random(1, 3, 5) должен выдать примерно 2, 2, 3, 2, 1
# Hint: реализация занимает 2 строки
def gen_random(begin, end, num_count):
    for i in range(num_count):
        yield random.randint(begin, end)
```

2.3. Исходный код ex_1.py

```
#!/usr/bin/env python3
from librip.gens import field, gen_random

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
    {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
]

print(list(gen_random(1, 11, 20)))
print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))
```

2.4. Скриншоты с результатами выполнения

```
C:\Anaconda3\python.exe "C:/Users/Артём/Google Диск/Учёба/5 сем/РИП/lab_2/ex_1.py"
[4, 6, 6, 2, 1, 10, 8, 9, 8, 6, 2, 7, 5, 7, 11, 10, 6, 1, 11, 2]
['Ковер', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стелаж', 'price': 7000}, {'title': 'Вешалка для одежды', 'price': 800}]
```

3. Задача 2 (ex_2.py)

3.1. Формулировка задания

Необходимо реализовать итератор, который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты. Конструктор итератора также принимает на вход именной bool-параметр ignore_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False. 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(gen_random(1, 3, 10))будет последовательно возвращать только 1, 2 и 3

data = [‘a’, ‘A’, ‘b’, ‘B’]

Unique(data) будет последовательно возвращать только a, A, b, B

data = [‘a’, ‘A’, ‘b’, ‘B’]

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

В ex_2.py нужно вывести на экран то, что они выдают одной строкой. Важно продемонстрировать работу как с массивами, так и с генераторами (gen_random).

Итератор должен располагаться в librip/iterators.py

3.2. Исходный код librip/iterators.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 одинаковые строки в разном регистре
```

```
        # Например: ignore_case = True, Абв и АБВ разные строки
        #           ignore_case = False, Абв и АБВ одинаковые строки, одна из них
удалится
        # По-умолчанию ignore_case = False
        self.items = list(items)
        self.index = 0
        self.unique_items = []
        if 'ignore_case' not in kwargs:
            self.ignore_case = False
        else:
            self.ignore_case = kwargs['ignore_case']

        pass

    def __next__(self):
        if self.index == len(self.items):
            raise StopIteration

        result = self.items[self.index]
        self.index += 1

        if self.ignore_case and type(result) == str:
            if str.casefold(result) not in self.unique_items:
                self.unique_items.append(str.casefold(result))
                return result
            else:
                return next(self)
        elif result not in self.unique_items:
            self.unique_items.append(result)
            return result
        else:
            return next(self)

    def __iter__(self):
        return self
```

3.3. Исходный код ex_2.py

```
#!/usr/bin/env python3
from librip.gens import gen_random
from librip.iterators import Unique

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 10, 50)
data3 = ['abC', 'aBc', 'ABC']

for item in Unique(data1):
    print(item)

print(list(Unique(data2)))
print(list(Unique(data3)))
```

3.4. Скриншоты с результатами выполнения

```
C:\Anaconda3\python.exe "C:/Users/Артём/Google Диск/Учёба/5 сем/РИП/lab_2/ex_2.py"
1
2
[5, 10, 6, 8, 9, 3, 2, 1, 7, 4]
['abC', 'aBc', 'ABC']
```

4. Задача 3 (ex_3.py)

4.1. Формулировка задания

Дан массив с положительными и отрицательными числами. Необходимо одной строкой вывести на экран массив, отсортированный по модулю. Сортировку осуществлять с помощью функции sorted

Пример:

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

Вывод: [0, 1, -1, 4, -4, -30, 100, -100, 123]

4.2. Исходный код ex_3.py

```
#!/usr/bin/env python3

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

print(sorted(data, key=lambda x: abs(x)))
```

4.3. Скриншоты с результатами выполнения

```
C:\Anaconda3\python.exe "C:/Users/Артём/Google Диск/Учёба/5 сем/РИП/lab_2/ex_3.py"
[0, 1, -1, 4, -4, -30, 100, -100, 123]
```

5. Задача 4 (ex_4.py)

5.1. Формулировка задания

Необходимо реализовать декоратор print_result, который выводит на экран результат выполнения функции. Файл ex_4.py не нужно изменять.

Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции, печатать результат и возвращать значение.

Если функция вернула список (list), то значения должны выводиться в столбик.

Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равно

Пример:

```
@print_result
def test_1():
    return 1
@print_result
def test_2():
    return 'iu'
@print_result
def test_3():
    return {'a': 1, 'b': 2}
@print_result
def test_4():
    return [1, 2]
test_1()
test_2()
test_3()
test_4()
```

На консоль выведется:

```
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2
```

Декоратор должен располагаться в librip/decorators.py

5.2. Исходный код librip/decorators.py

```
# Здесь необходимо реализовать декоратор, print_result который принимает на вход функцию,
# вызывает её, печатает в консоль имя функции, печатает результат и возвращает
```

```
значение
# Если функция вернула список (list), то значения должны выводиться в столбик
# Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик
через знак равно
# Пример из ex_4.py:
# @print_result
# def test_1():
#     return 1
#
# @print_result
# def test_2():
#     return 'iu'
#
# @print_result
# def test_3():
#     return {'a': 1, 'b': 2}
#
# @print_result
# def test_4():
#     return [1, 2]
#
# test_1()
# test_2()
# test_3()
# test_4()
#
# На консоль выведется:
# test_1
# 1
# test_2
# iu
# test_3
# a = 1
# b = 2
# test_4
# 1
# 2


def print_result(func):
    def decorated_func(*args):
        print(func.__name__)
        result = func(*args)
        if isinstance(result, list):
            for item in result:
                print(item)
        elif isinstance(result, dict):
            for k, v in result.items():
                print(k, "=", v)
        else:
            print(result)
        return result

    return decorated_func
```

5.3. Исходный код ex_4.py

```
from librip.decorators import print_result

# Необходимо верно реализовать print_result
```

```
# и задание будет выполнено


@print_result
def test_1():
    return 1


@print_result
def test_2():
    return 'iu'


@print_result
def test_3():
    return {'a': 1, 'b': 2}


@print_result
def test_4():
    return [1, 2]


test_1()
test_2()
test_3()
test_4()
```

5.4. Скриншоты с результатами выполнения

```
Launching pytest with arguments C:/Users/Артём/Google Диск/Учёба/5 сем/РИП/lab_2/ex_4.py in C:\Users\Артём\Google Диск\Учёба\5 сем\РИП\lab_2

============================= test session starts =============================
platform win32 -- Python 3.7.4, pytest-5.2.1, py-1.8.0, pluggy-0.13.0 -- C:\Anaconda3\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Артём\Google Диск\Учёба\5 сем\РИП\lab_2
plugins: arraydiff-0.3, doctestplus-0.4.0, openfiles-0.4.0, remotedata-0.3.2
collecting ... collected 4 items

ex_4.py::test_1 <- librip\decorators.py PASSED                           [ 25%]test_1
1

ex_4.py::test_2 <- librip\decorators.py PASSED                           [ 50%]test_2
iu

ex_4.py::test_3 <- librip\decorators.py PASSED                           [ 75%]test_3
a = 1
b = 2

ex_4.py::test_4 <- librip\decorators.py PASSED                           [100%]test_4
1
2
```

6. Задача 5 (ex_5.py)

6.1. Формулировка задания

Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран

Пример:

```
with timer():
    sleep(5.5)
```

После завершения блока должно вывестись в консоль примерно 5.5

6.2. Исходный код контекстного менеджера librip/ctxmngrs.py

```
# Здесь необходимо реализовать
# контекстный менеджер timer
# Он не принимает аргументов, после выполнения блока он должен вывести время
выполнения в секундах
# Пример использования
# with timer():
#   sleep(5.5)
#
# После завершения блока должно вывестись в консоль примерно 5.5
import time


class timer:

    def __enter__(self):
        self.t = time.clock()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print (time.clock() - self.t)
```

6.3. Исходный код ex_5.py

```
from time import sleep
from librip.ctxmngrs import timer

with timer():
    sleep(5.5)
```

6.4. Скриншоты с результатами выполнения

```
C:\Anaconda3\python.exe "C:/Users/Артём/Google Диск/Учёба/5 сем/РИП/lab_2/ex_5.py"
5.508278477
```

7. Задача 6 (ex_6.py)

7.1. Формулировка задания

Мы написали все инструменты для работы с данными. Применим их на реальном примере, который мог возникнуть в жизни. В репозитории находится файл data_light.json. Он содержит облегченный список вакансий в России в формате json (ссылку на полную версию размером ~ 1 Гб. в формате xml можно найти в файле README.md).

Структура данных представляет собой массив словарей с множеством полей: название работы, место, уровень зарплаты и т.д.

В ex_6.py дано 4 функции. В конце каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print_result печатается результат, а контекстный менеджер timer выводит время работы цепочки функций.

Задача реализовать все 4 функции по заданию, ничего не изменяя в файле-шаблоне. Функции f1-f3 должны быть реализованы в 1 строку, функция f4 может состоять максимум из 3 строк.

Что функции должны делать:

Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих заданий.

Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Иными словами нужно получить все специальности, связанные с программированием. 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.

7.2. Исходный код

```
#!/usr/bin/env python3
import json
import sys
from librip.ctxmngrs import timer
from librip.decorators import print_result
from librip.gens import field, gen_random
from librip.iterators import Unique as unique

path = sys.path[0] + "\\data_light.json"

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

with open(path, encoding='UTF-8') as f:
    data = json.load(f)


# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`
# Важно!
# Функции с 1 по 3 должны быть реализованы в одну строку
```

```python
# В реализации функции 4 может быть до 3 строк
# При этом строки должны быть не длиннее 80 символов

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


@print_result
def f2(arg):
    return list(filter(lambda x: x.casefold().startswith("программист"), 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(100000, 200000, len(arg))))


with timer():
    f4(f3(f2(f1(data))))
```

7.3. Скриншоты с результатами выполнения

```
C:\Anaconda3\python.exe "C:/Users/Артём/Google Диск/Учёба/5 сем/РИП/lab_2/ex_6.py"
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
автослесарь
Автослесарь - моторист
Автоэлектрик
Агент
Агент банка
Агент нпф
Агент по гос. закупкам недвижимости
Агент по недвижимости
Агент по недвижимости (стажер)
Агент по недвижимости / Риэлтор
```

...

```
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 = 162349
Программист / Senior Developer с опытом Python = 157840
Программист 1С с опытом Python = 179665
Программист C# с опытом Python = 158290
Программист C++ с опытом Python = 199998
Программист C++/C#/Java с опытом Python = 147979
Программист/ Junior Developer с опытом Python = 166941
Программист/ технический специалист с опытом Python = 168284
Программистр-разработчик  информационных систем с опытом Python = 174053
0.07707422600000002
```