Московский Государственный Технический Университет им. Н. Э. Баумана
Факультет «Информатика и системы управления»

Отчет по лабораторной работе № 4
Курс «Разработка интернет-приложений»

Выполнил:

студент группы ИУ5-54 _____
(подпись)

Харлашкин А. И.

"__"_____2016 г.

Проверил:
Преподаватель каф. ИУ5

_____
(подпись)

Гапанюк Ю. Е.

"__"_____2016 г.

Москва, МГТУ – 2016 г.

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 реализовывать 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_4`
3. Выполнить `git clone` проекта из вашего репозитория

Задача 1 (`ex_1.py`)

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'}
```

1. В качестве первого аргумента генератор принимает `list`, дальше через `*args` генератор принимает неограниченное кол-во аргументов.
2. Если передан один аргумент, генератор последовательно выдает только значения полей, если поле равно `None`, то элемент пропускается
3. 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 *одной строкой*
Генераторы должны располагаться в `librip/gen.py`

Задача 2 (`ex_2.py`)

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`

Задача 3 (`ex_3.py`)

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]`

Задача 4 (`ex_4.py`)

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`

Задача 5 (`ex_5.py`)

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

Задача 6 (`ex_6.py`)

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:

1. Функция `f1` должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна **игнорировать регистр**. Используйте наработки из предыдущих заданий.
2. 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`.
3. 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`.
4. 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.

Формат отчета

1. Титульный лист
2. Задание (копируется из данной методички)

Обязательные условия

1. Форк репозитория на github
2. `ex_1.py`: использование `*args`
3. `ex_2.py`: ипользование `**kwargs`, поддержка работы как со списками, так и с генераторами
4. `ex_3.py`: использование `lambda`-выражения
5. `ex_4.py`: внутри декоратора печать должна быть реализована в одну строчку, запрещается использовать циклы. Печать словарей и массивов должна выполняться в столбик
6. `ex_6.py`: соблюдения кол-ва строк и использование функций, указанных в задании
7. Путь до файла передается как аргумент при запуске скрипта
8. Код на github

ex_1.py

```
#!/usr/bin/env python3
from librip.gens import field
from librip.gens import 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'}
]

# Реализация задания 1

print (list(gen_random(1, 3, 5)))
print (list(field(goods)))
print (list(field(goods, 'title')))
print (list(field(goods, 'title', 'price')))
```

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, 'Aaa', 'aaa']
data2 = gen_random(1, 3, 10)

# Реализация задания 2
data_1 = [1, 4, 1, 1, 90, 5, 2, 3, 2, 2]
for i in sorted(Unique(data_1)):
    print(i, end = ' ')

print()

for i in Unique(list(gen_random(1, 3, 10))):
    print(i, end = ' ')

print()

data_2 = ['a', 'A', 'b', 'c', 'B', 'C']
print(list(Unique(data_2)))

print(list(Unique(data_2, ignore_case=True)))
```

ex_3.py

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

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3
print(sorted(data, key = lambda x: abs(x)))
```

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()
```

ex_5.py

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

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

ex_6.py

```
#!/usr/bin/env python3
import os.path
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.argv[1]
# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

with open(path, encoding='utf8') 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
# 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([i for i in unique([j['job-name'] for j in arg],
ignore_case=True)]))
    #raise NotImplemented

@print_result
def f2(arg):
    return ([x for x in arg if 'программист' in x])
```

```
    #raise NotImplemented


@print_result
def f3(arg):
    return (["{} {}".format(x, "с опытом Python") for x in arg])
    #raise NotImplemented


@print_result
def f4(arg):
    return (["{}, {} {} {}".format(x, "зарплата", y, "руб.") for x, y in
zip(arg, list(gen_random(100000, 200000, len(arg))))])
    #raise NotImplemented


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

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 __init__(self):
      self.t = time.clock()
   def __enter__(self):
      t2 = time.clock()
   def __exit__(self, exp_type, exp_value, traceback):
      print(time.clock()-self.t)
```

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(function_to_decorate):
    def wrapper_function(*args):
        print(function_to_decorate.__name__)
        if (type(function_to_decorate(*args)) == type(list())):
            a = function_to_decorate(*args)
            for i in range(len(a)):
                print(a[i])
        elif (type(function_to_decorate(*args)) == type(dict())):
            b = function_to_decorate(*args)
            for key in b:
                print(key,' = ', b[key])
        else:
            print(function_to_decorate(*args))
        return function_to_decorate(*args)
    return wrapper_function
```

gens.py

```
import 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'}
]

def field(items, *args):
     if (len(args) == 1):
         for n in items:            #перебор словарей
             if (args[0] in n):
                 yield n[args[0]]

     if (len(args) > 1):
         for n in items:
             buf = {}
             #for i in range(len(args)):     #i-номер текущего аргумента
             i=0
             while (i < len(args)):
                 if (args[i] in n):
                     buf[args[i]] = n[args[i]]          #!!??если {'price':
7000, 'color': 'white'}, то 'price':, 'title':
                 i+=1
             if (buf != {}):
                yield buf
```

```
def gen_random(begin, end, quantity):    #quantity-количество
    for i in range(quantity):
       yield random.randint(begin, end)
```

iterators.py

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case', False)

        self.items = items
        self.i = 0
        self.unique_data = []

    def __next__(self):
        if (self.ignore_case == True):
            self.items = [str(i).lower() for i in self.items]
        while self.items[self.i] in self.unique_data:
            if (len(self.items) == self.i + 1):
                raise StopIteration
            self.i += 1

        self.unique_data.append(self.items[self.i])
        return self.items[self.i]

    def __iter__(self):
        return self
```

Скриншот результата:

Ex_1:

```
[1, 1, 2, 3, 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}]

Process finished with exit code 0
```

Ex_2:

```
1 2 3 4 5 90
1 3 2
['a', 'A', 'b', 'c', 'B', 'C']
['a', 'b', 'c']

Process finished with exit code 0
```

Ex_3:

```
[0, 1, -1, 4, -4, -30, 100, -100, 123]

Process finished with exit code 0
```

Ex_4:

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

Process finished with exit code 0
```

Ex_5:

```
5.4921751708832565

Process finished with exit code 0
```

Ex_6:

f2
1с программист
web-программист
веб - программист (php, js) / web разработчик
веб-программист
ведущий инженер-программист
ведущий программист

инженер - программист
инженер - программист асу тп
инженер-программист
инженер-программист (клинский филиал)
инженер-программист (орехово-зуевский филиал)
инженер-программист 1 категории
инженер-программист ккт
инженер-программист плис
инженер-программист сапоу (java)
инженер-электронщик (программист асу тп)
педагог программист
помощник веб-программиста
программист
программист / senior developer
программист 1с
программист c#
программист c++
программист c++/c#/java
программист/ junior developer
программист/ технический специалист
программистр-разработчик информационных систем
системный программист (c, linux)
старший программист
f3
1с программист с опытом Python
web-программист с опытом Python
веб - программист (php, js) / web разработчик с опытом Python
веб-программист с опытом Python
ведущий инженер-программист с опытом Python
ведущий программист с опытом Python
инженер - программист с опытом Python
инженер - программист асу тп с опытом Python
инженер-программист с опытом Python
инженер-программист (клинский филиал) с опытом Python
инженер-программист (орехово-зуевский филиал) с опытом Python
инженер-программист 1 категории с опытом Python
инженер-программист ккт с опытом Python
инженер-программист плис с опытом Python
инженер-программист сапоу (java) с опытом Python
инженер-электронщик (программист асу тп) с опытом Python
педагог программист с опытом Python
помощник веб-программиста с опытом Python
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
системный программист (c, linux) с опытом Python
старший программист с опытом Python
f4
1с программист с опытом Python, зарплата 180906 руб.
web-программист с опытом Python, зарплата 101448 руб.
веб - программист (php, js) / web разработчик с опытом Python, зарплата 106226 руб.
веб-программист с опытом Python, зарплата 124884 руб.
ведущий инженер-программист с опытом Python, зарплата 108310 руб.
ведущий программист с опытом Python, зарплата 197170 руб.
инженер - программист с опытом Python, зарплата 124394 руб.
инженер - программист асу тп с опытом Python, зарплата 123929 руб.
инженер-программист с опытом Python, зарплата 111896 руб.

инженер-программист (клинский филиал) с опытом Python, зарплата 120097 руб.
инженер-программист (орехово-зуевский филиал) с опытом Python, зарплата 113568 руб.
инженер-программист 1 категории с опытом Python, зарплата 150495 руб.
инженер-программист ккт с опытом Python, зарплата 108407 руб.
инженер-программист плис с опытом Python, зарплата 188350 руб.
инженер-программист сапоу (java) с опытом Python, зарплата 164465 руб.
инженер-электронщик (программист асу тп) с опытом Python, зарплата 141596 руб.
педагог программист с опытом Python, зарплата 139131 руб.
помощник веб-программиста с опытом Python, зарплата 168681 руб.
программист с опытом Python, зарплата 106860 руб.
программист / senior developer с опытом Python, зарплата 106552 руб.
программист 1с с опытом Python, зарплата 121555 руб.
программист c# с опытом Python, зарплата 189119 руб.
программист c++ с опытом Python, зарплата 186849 руб.
программист c++/c#/java с опытом Python, зарплата 108196 руб.
программист/ junior developer с опытом Python, зарплата 126354 руб.
программист/ технический специалист с опытом Python, зарплата 163722 руб.
программистр-разработчик информационных систем с опытом Python, зарплата 191461 руб.
системный программист (c, linux) с опытом Python, зарплата 129068 руб.
старший программист с опытом Python, зарплата 106545 руб.
63.01157031479225

Process finished with exit code 0