Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и управления»

Лабораторная работа по курсу РИП №4

«Python. Функциональные возможности»

Студент группы ИУ5-53:
Барашкова Е.С. _____

Преподаватель:
Гапанюк Ю.Е. _____

Москва 2016

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.

Программы

Исходный код
ex_1.py

```
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'}
]
getr = gen_random(1,5,7)
print(*list(getr), sep=' ')

getf = field(goods, 'title')
print(*list(getf), sep=', ')
```

Результат

```
['Ковер', 'Кресло', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Кресло', 'price': 1500}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стелаж', 'price': 7000}, {'price': 100},
[1, 2, 1, 2, 1]
```

Исходный код
ex_2.py

```
data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 3, 10)
#Реализация задания 2

print(list(Unique(data1)))
print(list(Unique(data2)))
print(list(Unique([a', 'A', 'b', 'B'])))
print(list(Unique(([a', 'A', 'b', 'B'], ignore_case=True)))
```

Результат

```
[1, 2]
[3, 1, 2]
['a', 'A', 'b', 'B']
['a', 'b']
```

Исходный код
ex_3.py

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

Результат

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

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()
```

Результат

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

Исходный код
ex_5.py

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

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

Результат

```
5.512930393218994
```

Исходный код

ex_6.py

```
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 = None

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске
path = sys.argv[1]
print(path)

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

# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`
# Важно!
# Функции с 1 по 3 дожны 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(data):
   return sorted(list(unique(list(field(data, 'job-name')),ignore_case = True)))

@print_result
def f2(data):
   return list(filter(lambda x: x[0:11].lower() == 'программист', data))

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

@print_result
def f4(data):
      return(list('{} с зарплатой {} рублей'.format(prof, salary) for prof,salary in zip(data,
gen_random(100000, 200000, len(data)))))

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

Результат

```
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
```

```
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
```

```
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
```

```
f4
Программист с опытом Python, зарплата 108608 руб.
Программист / Senior Developer с опытом Python, зарплата 184920 руб.
Программист 1С с опытом Python, зарплата 145069 руб.
Программист C# с опытом Python, зарплата 113173 руб.
Программист C++ с опытом Python, зарплата 160225 руб.
Программист C++/C#/Java с опытом Python, зарплата 197812 руб.
Программист/ Junior Developer с опытом Python, зарплата 137866 руб.
Программист/ технический специалист с опытом Python, зарплата 106606 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 143440 руб.
0.10937809944152832
```

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
from time import *
```

```
import contextlib
@contextlib.contextmanager
def timer():
    time_start = time();
    yield
    time_end = time();
    print(time_end - time_start)
```

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_to_decorate):
    def decorated_func(*args):
        res = func_to_decorate(*args)
        if type(res) == list:
```

```
            print(func_to_decorate.__name__, *list(res), sep='\n')
        elif type(res) == dict:
            print(func_to_decorate.__name__ , '\n'.join('{}={}'.format(k[0],k[1]) for k in res.items()),
sep='\n')
        else:
            print(func_to_decorate.__name__, res, sep='\n')
        return res
    return decorated_func
```

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
    for item in items:
        if len(args) == 1:
            val = item.get(args[0])
            if val != None:
                yield val
        else:
            res = {};
            for item_arg in item:
                if (item_arg in args) and (item[item_arg] != None):
                    res[item_arg] = item[item_arg]
            if res != {}:
                yield res
    # Необходимо реализовать генератор
# 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 list(range(num_count)):
        yield random.randint(begin,end)
```

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.ignore_case = False if (kwargs.get('ignore_case') == None) else kwargs.get('ignore_case')
        self.seen = []
        self.items = iter(items)

    def __next__(self):
        # Нужно реализовать __next__
        while True:
            val = self.items.__next__()
            val_cmp = val if self.ignore_case == False else str(val).lower()
            if val_cmp not in self.seen:
                self.seen.append(val_cmp)
                return val

    def __iter__(self):
        return self
```