

**Московский государственный технический
университет им. Н.Э. Баумана.**

Факультет «Информатика и системы управления»

Кафедра ИУ5. Курс «Разработка интернет приложений»

Отчет по лабораторной работе №3

«Python. Функциональные возможности»

Выполнил:

студент группы ИУ5-54

Арьков К.А.

Подпись и дата:

Проверил:

доцент каф. ИУ5

Гапанюк Ю. Е.

Подпись и дата:

Москва, 2017 г.

Оглавление

Задание.....	3
Задача 1 (ex_1.py).....	3
Задача 2 (ex_2.py).....	3
Задача 3 (ex_3.py).....	4
Задача 4 (ex_4.py).....	4
Задача 5 (ex_5.py).....	4
Задача 6 (ex_6.py).....	5
Исходный код.....	5
librib/gens.py	5
ex_1.py.....	6
librip/iterators.py.....	7
ex_2.py.....	7
ex_3.py.....	8
librip/decorators.py	8
ex_4.py.....	9
librip/ctxmgrs.py.....	9
ex_5.py.....	9
ex_6.py.....	9
Результаты работы.....	11

Задание

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 выводиться 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

Исходный код

librib/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
    # Необходимо реализовать генератор
    if(len(args)==1):
        for item in items:
            try:
                if item[args[0]] is not None:
                    yield item[args[0]]
            except:
                pass
    else:
        for item in items:
            line = {}
            for arg in args:
                try:
                    if item[arg] is not None:
                        line[arg] = item[arg]
                except:
                    pass
            if len(line) > 0:
                yield line
```

```
# 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):
    # Необходимо реализовать генератор
    for x in range(0, num_count):
        yield random.randint(begin, end)
```

ex_1.py

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

import sys

from librip.gens import *

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': None},
]

# Реализация задания 1
#for f in field(goods, 'color'):#, 'title'):
#    print(f)
#print(', '.join(map(str, field(goods, 'color'))))
print('field generator:')
print(', '.join(map(str, field(goods, 'color', 'title'))))
print()

print('random generator:')
```

```
print(', '.join(map(str,gen_random(1,3,5))))
```

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 = items
```

```
        self.ignore_case = False
```

```
        if 'ignore_case' in kwargs:
```

```
            self.ignore_case = kwargs['ignore_case']
```

```
        self.returned = set()
```

```
    def __next__(self):
```

```
        # Нужно реализовать __next__
```

```
        for item in self.items:
```

```
            if type(item) == str and self.ignore_case is True:
```

```
                if item.lower() not in self.returned:
```

```
                    self.returned.add(item.lower())
```

```
                    return item
```

```
            else:
```

```
                if item not in self.returned:
```

```
                    self.returned.add(item)
```

```
                    return item
```

```
        raise StopIteration()
```

```
    def __iter__(self):
```

```
        return self
```

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, 3, 10)
```

```
data3 = ['a','A','b','B']
```

```
# Реализация задания 2
```

```
print('data:',data1)
```

```
print('unique:',', '.join(map(str,Unique(data1))))
```

```
print()
```

```
print('data:',data2)
```

```
print('unique:',', '.join(map(str,Unique(data2))))
```

```
print()
```

```
print('data:',data3,ignore_case=True)
```

```
print('unique:',', '.join(map(str,Unique(data3,ignore_case=True))))
```

```
print()
```

```
print('data:',data3,ignore_case=False)
```

```
print('unique:',', '.join(map(str,Unique(data3,ignore_case=False))))
```

```
print()
```

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)))
```

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 выводиться 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 new_func(*args):
        res = func(*args)
        print(func.__name__)
        if type(res) == list:
            [print(x) for x in res]
        elif type(res) == dict:
            [print(key, '=', value) for key, value in res.items()]
        else:
            print(res)
        return res
    return new_func
```


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()

```

librip/ctxmngers.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 time

class timer():
    def __enter__(self):
        self.start = time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(time()-self.start)

```

ex_5.py

```

from time import sleep
from librip.ctxmngers import timer

with timer():
    sleep(5.5)

```

ex_6.py

```

#!/usr/bin/env python3
import json
import sys

```

```

from librip.ctxmgrs import timer
from librip.decorators import print_result
from librip.gens import field, gen_random
from librip.iterators import Unique as unique

try:
    path = sys.argv[1]
except IndexError:
    raise ValueError('Path to data file is not specified')

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

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 должны быть реализованы в одну строку
# В реализации функции 4 может быть до 3 строк
# При этом строки должны быть не длиннее 80 символов

#for k,v in data[0].items():
#    print(k,v)

@print_result
def f1(arg):
    return sorted(unique(field(arg,'job-name'),ignore_case=True),key=lambda x: x.lower())
#    raise NotImplemented

@print_result
def f2(arg):
    return list(filter(lambda x:x.lower().startswith('программист'),arg))
#    raise NotImplemented

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

@print_result
def f4(arg):
    rand = gen_random(100000,200000,len(arg))
    return list(map(lambda s:s[0]+' с зарплатой '+str(s[1]),zip(arg,rand)))
#    raise NotImplemented

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

```

Результаты работы

X_1

```
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab3py/ex_1.py
Ковер, Диван для отдыха, Стелаж, Вешалка для одежды

2, 3, 3, 1, 1

Process finished with exit code 0
```

X_2

```
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab3py/ex_2.py
data: [4, 1, 2, 1, 1, 2, 1, 2, 3, 2]
unique: 4, 1, 2, 3

data: <generator object gen_random at 0x0000000002A45EA0>
unique: 10, 5, 7, 9

data: ['a', 'A', 'b', 'B', 'c', 'c', 'C', 'f'] ignore_case: True
unique: a, b, c, f

data: ['a', 'A', 'b', 'B', 'c', 'c', 'C', 'f'] ignore_case: False
unique: a, A, b, B, c, C, f

['a', 'A', 'b', 'B', 'c', 'c', 'C', 'f']

Process finished with exit code 0
```

X_3

```
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab3py/ex_3.py
[0, 1, -1, 4, -4, -90, 100, -100, 123]

Process finished with exit code 0
```

4

```
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab3py/ex_4.py
test_1
1
test_2
10
test_3
b = 2
a = 1
test_4
f
f
k

Process finished with exit code 0
```

X_5

```
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab3py/ex_5.py
5.50780987739563

Process finished with exit code 0
```

```
портной
специалист отдела аренды
Инженер-механик
Разработчик импульсных источников питания
Механик по эксплуатации транспортного отдела
Инженер-технолог по покраске
Бетонщик - арматурщик
главный инженер финансово-экономического отдела
Секретарь судебного заседания в аппарате мирового судьи Железнодорожного судебного района города Ростова-на-Дону
варщик зефира
варщик мармеладных изделий
Оператор склада
Специалист по электромеханическим испытаниям аппаратуры бортовых космических систем
Заведующий музеем в д.Копорье
Документовед
Специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических систем
Менеджер (в промышленности)
f2
Программист
Программист C++/C#/Java
Программист 1С
Программист-разработчик информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
Программист C#
f3
Программист с опытом Python
Программист C++/C#/Java с опытом Python
Программист 1С с опытом Python
Программист-разработчик информационных систем с опытом Python
Программист C++ с опытом Python
Программист/ Junior Developer с опытом Python
Программист / Senior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программист C# с опытом Python
f4
Программист с опытом Python с зарплатой 196694
Программист C++/C#/Java с опытом Python с зарплатой 128507
Программист 1С с опытом Python с зарплатой 157602
Программист-разработчик информационных систем с опытом Python с зарплатой 140333
Программист C++ с опытом Python с зарплатой 137105
Программист/ Junior Developer с опытом Python с зарплатой 195318
Программист / Senior Developer с опытом Python с зарплатой 158972
Программист/ технический специалист с опытом Python с зарплатой 195673
Программист C# с опытом Python с зарплатой 176931
0.031199932098388672

Process finished with exit code 0
```