

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

**Кафедра ИУ5 «Системы обработки информации и управления»**

**Отчет по лабораторной работе №4**

**«Python. Функциональные возможности»**

**по дисциплине «Разработка Интернет-приложений»**

**Выполнил: студент  
группы ИУ5-53  
Виноградов Андрей**

**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.

### Подготовительный этап

1. Зайти на [github.com](https://github.com/iu5team/ex-lab4) 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

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 *о 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.
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
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*
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.*

## Исходный код

Файл `ex_1.py`

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

```
from librip.gen 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'}  
]
```

```
# Реализация задания 1
```

```
print(list(field(goods, 'title')))
```

```
print(list(field(goods, 'title', 'price')))
```

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

Файл ex\_2.py

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

```
from librip.gen 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)
```

```
# 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)))
```

Файл ex\_3.py

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

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

```
# Реализация задания 3
```

```
print(sorted(data, key=abs))
```

Файл 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()
```

Файл ex\_5.py

```
from time import sleep
```

```
from librip.ctxmgrs import timer
```

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

Файл ex\_6.py

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

```
import json  
import sys  
from librip.ctxmgrs import timer  
from librip.decorators import print_result from librip.gen  
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) 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(unique(field(arg, 'job-name'), ignore_case=True))
```

```
@print_result def  
f2(arg):  
    return list(filter(lambda x: str(x).startswith('программист'), arg))
```

```
@print_result def  
f3(arg):  
    return ["{} с опытом Python".format(i) for i in arg]
```

```
@print_result def  
f4(arg):  
    return ["{}, зарплата {} руб.".format(work, salary) for (work, salary) in zip(arg,  
gen_random(100000, 200000, len(arg)))]
```

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

Файл librip/ctxmgrs.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:  
    @staticmethod def  
    get_time():  
        return time.time()  
  
    def __enter__(self):  
        self.time_start = __class__.get_time()  
  
    def __exit__(self, exc_type, exc_val, exc_tb):  
        print(__class__.get_time() - self.time_start)
```

Файл `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 result(*args, **kwargs): val =
    func(*args, **kwargs)

    print(func.__name__)
    if isinstance(val, list): print("\n".join(map(str,
        val)))
    elif isinstance(val, dict):
        print("\n".join(map(lambda x: "{} = {}".format(x[0], x[1]), val.items())))

    else: print(val)

    return val

return result

```

Файл librip/gen.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: assert(isinstance(item,
        dict)) if len(args) == 1:

        val = item.get(args[0]) if val is
        not None:
            yield val
        else:
            res_dict = {key: item.get(key) for key in (set(item.keys()) & set(args)) if
item.get(key) is not None}
            if len(res_dict) > 0: yield
                res_dict

# 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(0, num_count):
        yield random.randint(begin, end)

```

Файл librip/iterators.py

# Итератор для удаления дубликатов

```

class Unique(object):
    def __init__(self, items, **kwargs):

```

```

        self.ignore_case = kwargs.get('ignore_case') is not None and
kwargs.get('ignore_case')
        self.items = iter(items)
        self.seen = set()

    def __next__(self):
        val = self.items.__next__()
        val = str(val).lower()
        if self.ignore_case else val

        if val not in self.seen:
            self.seen.add(val)
            return val

        return self.__next__()

    def __iter__(self):
        return self

```

## Результаты выполнения

Файл ex\_1.py

```

> python3 ex_1.py
['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, 1, 3, 1, 1]

```

Файл ex\_2.py

```

> python3 ex_2.py
[1, 2]
[3, 1, 2]
['a', 'A', 'b', 'B']
['a', 'b']

```

Файл ex\_3.py

```

> python3 ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]

```

Файл ex\_4.py

```

> python3 ex_4.py
test_1
1
test_2
iu
test_3
b = 2
a = 1
test_4
1
2

```

Файл ex\_5.py

```
> python3 ex_5.py  
5.504544019699097
```

Файл ex\_6.py

```
f2  
программист  
программист / senior developer  
программист 1с  
программист c#  
программист c++  
программист c++/c#/java  
программист/ junior developer  
программист/ технический специалист  
программист-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  
программист-разработчик информационных систем с опытом Python  
f4  
программист с опытом Python, зарплата 177337 руб.  
программист / senior developer с опытом Python, зарплата 167943 руб.  
программист 1с с опытом Python, зарплата 137777 руб.  
программист c# с опытом Python, зарплата 186532 руб.  
программист c++ с опытом Python, зарплата 170609 руб.  
программист c++/c#/java с опытом Python, зарплата 156323 руб.  
программист/ junior developer с опытом Python, зарплата 133773 руб.  
программист/ технический специалист с опытом Python, зарплата 123420 руб.  
программист-разработчик информационных систем с опытом Python, зарплата 122272 руб.  
0.07739806175231934
```