

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
ИМЕНИ Н.Э. БАУМАНА

Кафедра ИУ-5  
«Системы обработки информации и управления»

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

Выполнил: Лузин Д.С.  
Группа: ИУ5-53

## 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).<sup>1</sup>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:<sup>1</sup>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.<sup>1</sup>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.<sup>1</sup>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.<sup>1</sup> 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.<sup>1</sup> 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<sup>1</sup>
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*<sup>1</sup>
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.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'}
]

# 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.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)

# 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=lambda n: abs(n)))
```

### 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(1.5)
```

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.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="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  
# 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), key=  
lambda x: x.lower())  
  
@print_result  
def f2(arg):  
    return list(filter(lambda x: 'программист' in x.lower(), arg))  
  
@print_result  
def f3(arg):  
    return list(map(lambda x: x + ' с опытом Python', arg))
```

```

@print_result
def f4(arg):
    return list(zip(arg, gen_random(100000, 200000, len(arg))))

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

```

Файл decorators.py

```

def print_result(func):
    def some_fun(*args, **kwargs):
        print(func.__name__)
        data = func(*args, **kwargs)
        if isinstance(data, list) == True:
            print("\n".join(map(str, data)))
        elif type(data) == dict:
            print("\n".join(map(lambda x: "{} = {}".format(x[0], x[1]),
data.items()))))
        else:
            print(data)
        return data

    return some_fun

```

Файл gen.py

```

def field(items, *args):
    #assert len(args) > 0
    # Необходимо реализовать генератор
    flag = 0
    l = ''
    for item in items:
        assert (type(item) == dict)
        if len(args) == 1:
            yield item.get(args[0])
        else:
            yield { arg : item.get(arg) for arg in args}

# 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 i in range(num_count):
        yield random.randint(begin, end)

```

Файл ctxmgrs.py

```

import time

class timer:
    def __enter__(self):
        print(time.time())
        self.enter = time.time()

```



```
def __exit__(self, *args):
    print(time.time())
    print("Время работы: " + str((time.time() - self.enter))+ " секунд")
```

Файл iterators.py

```
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 = iter(items)
        self.set = set()
        self.ignore_case = kwargs.get('ignore_case')

    def __next__(self):
        # Нужно реализовать __next__
        while (True):
            nex = next(self.items)
            cor = nex
            if self.ignore_case == True:
                cor = nex.lower()
            if cor in self.set:
                continue
            else:
                break

            self.set.add(cor)
        return nex

    def __iter__(self):
        return self
```

## Результаты выполнения

### Файл ex\_1.py

```
"D:\Предметы\3 курс\РПП\Env_Python35\Scripts\python.exe" "D:/Предметы/3 курс/РПП/GitHub/lab4/lab_4/ex_1.py"
['Ковер', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']
[{'price': 2000, 'title': 'Ковер'}, {'price': 5300, 'title': 'Диван для отдыха'}, {'price': 7000, 'title': 'Стелаж'}, {'price': 800, 'title': 'Вешалка для одежды'}]
[1, 2, 1, 1, 3]
```

### Файл ex\_2.py

```
"D:\Предметы\3 курс\РПП"
[1, 2]
[1, 3, 2]
['a', 'A', 'b', 'B']
['a', 'b']
```

### Файл ex\_3.py

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

### Файл ex\_4.py

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

### Файл ex\_5.py

```
1480060329.8565936
1480060331.3567002
Время работы: 1.5001065731048584 секунд
```

### Файл ex\_6.py

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  
Программист-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  
Программист-разработчик информационных систем с опытом Python  
Системный программист (C, Linux) с опытом Python  
Старший программист с опытом Python

f4

```
('1С программист с опытом Python', 175447)
('Web-программист с опытом Python', 135688)
('Веб - программист (PHP, JS) / Web разработчик с опытом Python', 106452)
('Веб-программист с опытом Python', 178912)
('Ведущий инженер-программист с опытом Python', 158353)
('Ведущий программист с опытом Python', 163701)
('инженер - программист с опытом Python', 104831)
('Инженер - программист АСУ ТП с опытом Python', 112802)
('инженер-программист с опытом Python', 139599)
('Инженер-программист (Клинский филиал) с опытом Python', 111583)
('Инженер-программист (Орехово-Зуевский филиал) с опытом Python', 194522)
('Инженер-программист 1 категории с опытом Python', 171999)
('Инженер-программист ККТ с опытом Python', 187118)
('Инженер-программист ПЛИС с опытом Python', 105091)
('Инженер-программист САПОУ (java) с опытом Python', 125818)
('Инженер-электронщик (программист АСУ ТП) с опытом Python', 112942)
('педагог программист с опытом Python', 111570)
('Помощник веб-программиста с опытом Python', 105505)
('Программист с опытом Python', 162318)
('Программист / Senior Developer с опытом Python', 149042)
('Программист 1С с опытом Python', 175818)
('Программист С# с опытом Python', 112746)
('Программист С++ с опытом Python', 182663)
('Программист С++/С#/Java с опытом Python', 137947)
('Программист/ Junior Developer с опытом Python', 180383)
('Программист/ технический специалист с опытом Python', 118325)
('Программист-разработчик информационных систем с опытом Python', 142389)
('Системный программист (C, Linux) с опытом Python', 184903)
('Старший программист с опытом Python', 199451)
1480060398.3653114
Время работы: 0.031099796295166016 секунд
```