



**Московский государственный технический университет**

**им. Н.Э. Баумана**

**(МГТУ им. Н.Э. Баумана)**

**Радиотехнический факультет (РТ)**

Отчёт по лабораторной работе № 2

По дисциплине

«Разработка интернет-приложений»

Проверил:

Преподаватель кафедры ИУ-5

Гапанюк Ю.Е.

Подпись: \_\_\_\_\_

«\_\_» \_\_\_\_\_ 201\_ г.

Выполнил:

студент группы РТ5-51Б

Ануров Н.С.

Подпись: \_\_\_\_\_

«\_\_» \_\_\_\_\_ 201\_ г.

Москва, 2019

# 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 запрограммировать 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_2`
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, с помощью кода в *одну строку*

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 выводиться 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`.

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

### Ex1

```
#!/usr/bin/env python3
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': 'white'}
    ]

    # Реализация задания 1 (генераторы field и gen_random)
    print(list(field(goods, 'title')))
    print(list(field(goods, 'title', 'price')))

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

```

## Ex2

```

#!/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(list(Unique(data1)))
print(list(Unique(data2)))
print(list(Unique(data3)))
print(list(Unique(data3, ignore_case=True)))

```

## Ex3

```

#!/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)))

```

## Ex4

```

from librip.decorators import print_result

# Необходимо верно реализовать print_result
# и задание будет выполнено

@print_result # test_1=print_result(test_1)
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()

```

## Ex5

```

from time import sleep
from librip.ctxmgrs import timer

with timer():
    sleep(5.5)

```

## Ex6

```

#!/usr/bin/env python3
import json
from librip.ctxmgrs import timer
from librip.decorators import print_result
from librip.gens import field, gen_random
from librip.iterators import Unique

path = "data_light_cp1251.json"

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

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

@print_result
def f1(arg):
    return list(Unique(list(field(arg, "job-name")), ignore_case=True))

@print_result
def f2(arg):
    return list(filter(lambda s: "программист" in s[0:12], arg))

@print_result
def f3(arg): # map(func, arr)
    return list(map(lambda s: s + " с опытом Python", arg))

@print_result
def f4(arg):
    Sal = gen_random(100000, 200000, len(arg))
    return list(map(lambda s: '{}, зарплата {} руб.'.format(
        s[0], s[1]), zip(arg, Sal)))

```

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

## Скриншоты

1

```
['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, 2, 2, 1, 1]
```

2

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

3

```
С. Юсупов написал программу (создал программу) у себя
[0, 1, -1, 4, -4, -30, 100, -100, 123]
```

4

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

6

```
f1
Администратор на телефоне
Медицинская сестра
Охранник сутки-день-ночь-вахта
ВРАЧ АНЕСТЕЗИОЛОГ РЕАНИМАТОЛОГ
теплотехник
разнорабочий
Электро-газосварщик
Водитель Gett/Гетт и Yandex/Яндекс такси на личном автомобиле
Монолитные работы
Организатор – тренер
```