

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

**Разработка интернет-приложений  
Лабораторная работа № 4**

**“Python. Функциональные возможности”**

Выполнил:  
студент группы ИУ5-53  
Пирмамедов М. Э.  
Подпись:  
Дата:

Москва 2017г.

# Задание

**Важно** выполнять все задачи последовательно . С 1 по 5 задачу формирует 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

`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

*МГТУ им. Н. Э. Баумана, кафедра ИУ5, курс РИП*

*ЛР №4: Python, функциональные возможности*

```
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
МГТУ им. Н. Э. Баумана, кафедра ИУ5, курс РИП
ЛР №4: Python, функциональные возможности
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). П ример: *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.\_\_

## Скриншоты исходников

librip:

Файл ctxmngers.py

```
import time

class timer:
    def __enter__(self):
        self.t = time.clock()


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

class print_one:
    def __init__(self, arg):
        self.arg = arg

    def __enter__(self):
        print(self.arg)

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(self.arg)
```

Файл decorators.py

```

def print_result(some_func):
    def decorated_func(*arg, **kwargs):
        result = some_func(*arg, **kwargs)
        print(some_func.__name__)
        if type(result) == list:
            for el in result:
                print(el)
        elif type(result) == dict:
            for k, v in result.items():
                print("{} = {}".format(k, v))
        else:
            print(result)
        return result
    return decorated_func

#####
```

Файл iterators.py

```
# Итератор для удаления дубликатов

class Unique:
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case') and kwargs.get("ignore_case") is not None
        self.items = iter(items)
        self.seen = set()

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

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

    def __iter__(self):
        return self
```

## Файл gens.py

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for el in items:
            if el.get(args[0]) is not None:
                yield el.get(args[0])
    else:
        for el in items:
            if len(set(args) & set(el.keys())) != 0:
                yield {key: el.get(key) for key in args if el.get(key) is not None}

def gen_random(begin, end, num_count):
    for x in range(num_count):
        yield randint(begin, end)
```

## 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")),
      list(field(goods, 'title', 'price')),
      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)
data = ['A', 'a', 'B', 'b']
#print(Unique(data1))

# Реализация задания 2
print(list(Unique(data1)), list(Unique(data2)), list(Unique(data)))
```

## Файл ex\_3.py

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

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

#### 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, print_one

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 = None
if len(sys.argv) > 1:
    path = sys.argv[1]
else:
    path = input("введите путь к файлу \n")
# 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 list(sorted(unique(field(arg, "job-name"), ignore_case=True)))

@print_result
def f2(arg):
    return list(filter(lambda x: "программист" in x, arg))

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

@print_result
def f4(arg):
    sal_list = list(gen_random(100000, 200000, len(arg)))
    worklist = iter(map(lambda x: x + " зарплата", arg))
    return [{"{} {}".format(work, sal) for (work, sal) in zip(worklist, sal_list)}]

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

Результат выполнения программы ex\_6.py



f4

1С программист с опытом Python зарплата 173899  
Web-программист с опытом Python зарплата 190148  
Веб - программист (PHP, JS) / Web разработчик с опытом Python зарплата 105832  
Веб-программист с опытом Python зарплата 174961  
Ведущий инженер-программист с опытом Python зарплата 162117  
Ведущий программист с опытом Python зарплата 175523  
Инженер - программист АСУ ТП с опытом Python зарплата 105645  
Инженер-программист (Клинский филиал) с опытом Python зарплата 119763  
Инженер-программист (Орехово-Зуевский филиал) с опытом Python зарплата 141744  
Инженер-программист 1 категории с опытом Python зарплата 191169  
Инженер-программист ККТ с опытом Python зарплата 153272  
Инженер-программист ПЛИС с опытом Python зарплата 186285  
Инженер-программист САПОУ (java) с опытом Python зарплата 118748  
Инженер-электронщик (программист АСУ ТП) с опытом Python зарплата 125511  
Помощник веб-программиста с опытом Python зарплата 100013  
Системный программист (C, Linux) с опытом Python зарплата 118567  
Старший программист с опытом Python зарплата 100099  
инженер - программист с опытом Python зарплата 100944  
инженер-программист с опытом Python зарплата 137907  
педагог программист с опытом Python зарплата 141682  
0.22277286456281445