

Отчёт по лабораторной работе №4  
по курсу «Разработка интернет-приложений»  
Python. Функциональные возможности

Выполнил:

Волобуев В.Н., ИУ5-54

---

Преподаватель:

Гапанюк Ю.Е.

---

2016 г.

# 1) Задание лабораторной работы

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` 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

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

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.

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`

### 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

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

### 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.

## 2) ЛИСТИНГ

**# librip/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
```

```
    for item in items:
```

```
        if len(args) == 1:
```

```
            if args[0] not in item:
```

```
                pass
```

```
            else:
```

```
                yield item[args[0]]
```

```
        else:
```

```
            if len(set(args) & set(item)) == 0:
```

```
                pass
```

```
            else:
```

```
                yield {arg: item[arg] for arg in  
                      args if arg in item}
```

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

### **# 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
        pass
        self.ignore_case = kwargs.get('ignore_case', False)
        self.uniq_items = list()
        self.items = iter(items)
```

```

def __next__(self):
    # Нужно реализовать __next__
    pass
    while self.items:
        value = next(self.items) #will Raise StopIteration
        if self.ignore_case and type(value) == str:
            value = str(value).lower()
        if value not in self.uniq_items:
            self.uniq_items.append(value)
        return value

def __iter__(self):
    return self

```

### **# 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

```

```

def print_result(f):
    def decor(*args, **kwargs):
        inresult = f(*args, **kwargs)
        print(f.__name__)
        if type(inresult) == dict:
            for key in sorted(inresult):
                print(key+' = '+str(inresult[key]))

```

```
        elif type(inresult) == list:
            for item in inresult:
                print(item)
        else:
            print(inresult)
        return inresult
    return decor
```

### **# 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():
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(time.time() - self.start_time)
        return False    # no errors, could have been zero
```



## **# ex\_1.py**

```
#!/usr/bin/env python3
from librip.gens import field
from librip.gens import gen_random

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'},
    {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
    {'title': 'Вешалка для одежды', 'color': 'white'}
]

def one_liner(arg):
    print(', '.join(str(x) for x in arg))

# Реализация задания 1
if __name__ == '__main__':
    test = field(goods, 'title')
    print(', '.join(str(x) for x in test))
    one_liner(field(goods, 'title', 'price'))
    #print(', '.join(str(x) for x in gen_random(1, 5, 5)))
    one_liner(gen_random(1, 5, 5))
```

## **# 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_str = ['a', 'A', 'b', 'B']

def one_liner(arg):
    print(', '.join(str(x) for x in arg))
```

```
# Реализация задания 2
if __name__ == '__main__':
    test = Unique(data1) # pre-generated list
    one_liner(test)
    test = Unique(data2) # our generator test
    one_liner(test)
    test = Unique(data_str) # cased string
    one_liner(test)
    test = Unique(data_str, ignore_case=True) # caseless string
    one_liner(test)
```

### **# 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 item: abs(item)))
```

### **# 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()
```

#### **# ex\_5.py**

```
from time import sleep
from librip.ctxmgrs 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
```

```
path = None
```

```
# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске
path = sys.argv[1]

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

# Далее необходимо реализовать все функции по заданию, заменив
# `raise NotImplementedError`
# 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 list(map(lambda x: x+' с опытом Python', arg))

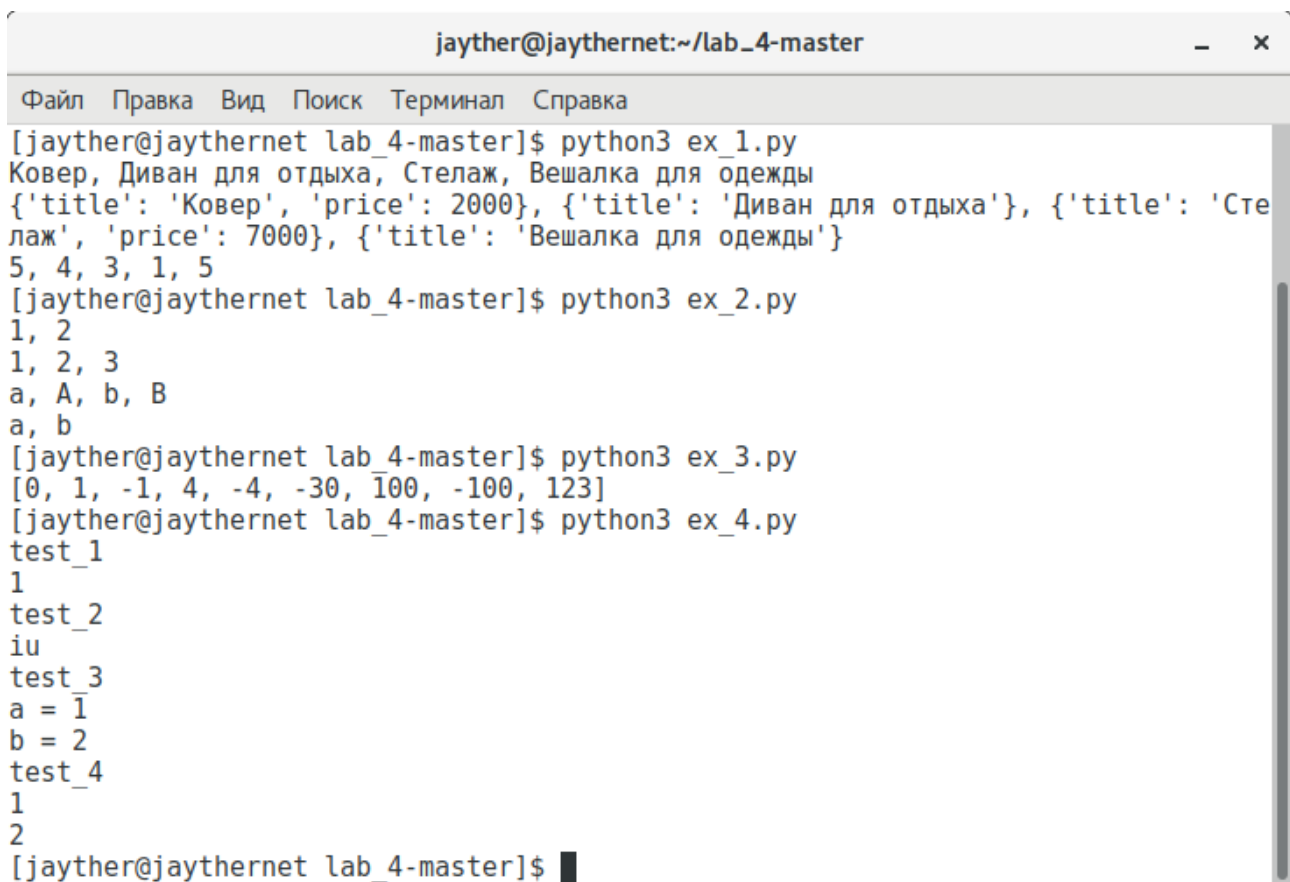
@print_result
def f4(arg):
    salaries = gen_random(100000, 200000, len(arg))
```

```
        return [job+', запплата '+str(salary)+' руб.' for (job,salary)
in
        zip(arg, salaries)]
```

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

### 3) Результаты работы

ex\_1 – ex\_4



```
jayther@jaythernet:~/lab_4-master
Файл  Правка  Вид  Поиск  Терминал  Справка
[jayther@jaythernet lab_4-master]$ python3 ex_1.py
Ковер, Диван для отдыха, Стелаж, Вешалка для одежды
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}, {'title': 'Сте
лаж', 'price': 7000}, {'title': 'Вешалка для одежды'}
5, 4, 3, 1, 5
[jayther@jaythernet lab_4-master]$ python3 ex_2.py
1, 2
1, 2, 3
a, A, b, B
a, b
[jayther@jaythernet lab_4-master]$ python3 ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]
[jayther@jaythernet lab_4-master]$ python3 ex_4.py
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2
[jayther@jaythernet lab_4-master]$
```

ex\_5, ex\_6

```
jayther@jaythernet:~/lab_4-master
Файл  Правка  Вид  Поиск  Терминал  Справка

[jayther@jaythernet lab_4-master]$ python3 ex_5.py
5.505188703536987
[jayther@jaythernet lab_4-master]$ python3 ex_6.py data_light.json
f1
1с программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
asic специалист
javascript разработчик
rtl специалист
web-программист
web-разработчик
автожестянщик
автоинструктор
автомаляр
автомойщик
автор студенческих работ по различным дисциплинам
автослесарь
автослесарь - моторист
автоэлектрик
--
```

```
jayther@jaythernet:~/lab_4-master
Файл  Правка  Вид  Поиск  Терминал  Справка

программистр-разработчик информационных систем
f3
программист с опытом Python
программист / senior developer с опытом Python
программист 1с с опытом Python
программист с# с опытом Python
программист с++ с опытом Python
программист с++/с#/java с опытом Python
программист/ junior developer с опытом Python
программист/ технический специалист с опытом Python
программистр-разработчик информационных систем с опытом Python
f4
программист с опытом Python, зарплата 189777 руб.
программист / senior developer с опытом Python, зарплата 160975 руб.
программист 1с с опытом Python, зарплата 181098 руб.
программист с# с опытом Python, зарплата 169041 руб.
программист с++ с опытом Python, зарплата 104824 руб.
программист с++/с#/java с опытом Python, зарплата 107648 руб.
программист/ junior developer с опытом Python, зарплата 144890 руб.
программист/ технический специалист с опытом Python, зарплата 191357 руб.
программистр-разработчик информационных систем с опытом Python, зарплата 189955
руб.
0.11742424964904785
[jayther@jaythernet lab_4-master]$ █
```