

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

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Разработка Интернет-приложений»

Отчет по лабораторной работе №2
«Python. Функциональные возможности»

Выполнил:
студент группы ИУ5-54Б
Сергеев Илья

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.

Подпись и дата:

Подпись и дата:

Москва, 2019 г.

Описание задания лабораторной работы

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_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`.
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.

Исходный код

- Модуль *librip/ctxmgrs.py*

```
import time

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

    def __exit__(self, type, value, traceback):
        print('Block has been executed during {:.g}
s'.format(time.clock()))
```

- Модуль *librip/decorators.py*

```
def print_result(any_func):
    def wrapper():
        print(any_func.__name__)
        res = any_func()
        if isinstance(res, list):
            for i in range(len(res)):
                print(res[i])
        elif isinstance(res, dict):
            for k in res:
                print('{} = {}'.format(k, res[k]))
        else:
            print(any_func())
    return wrapper()
```

- Модуль *librip/gens.py*

```
import random

def field(items, *args):
    assert len(args) > 0
    res = ""
    if len(args) == 1:
        for i in range(len(items)):
            if i != 0:
                res += ", "
            res += "{}".format(items[i][args[0]])
    else:
        for i in range(len(items)):
            if i != 0:
                res += ", "
            res += "{"
            f = 0
            for item in args:
                if f != 0:
                    res += ", "
                f += 1
            res += "{}{}: {}".format(item, items[i][item])
            res += "}"
        print(res)

def gen_random(begin, end, num_count):
    res = [random.randrange(begin, end + 1) for i in range(num_count)]
    return res
```

- Модуль *librip/iterators.py*

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = items
        self.used = []
        self.index = 0

    def __next__(self):
        if self.index == len(self.items):
            raise StopIteration
        if self.items[self.index] not in self.used:
            self.used.append(self.items[self.index])
            self.index += 1
        return self.items[self.index - 1]
        self.index += 1

    def __iter__(self):
        return self
```

- Модуль *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', 'price': 5300, 'color': 'black'},
    {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
    {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
]

field(goods, 'title', 'price')

rand_list = gen_random(1, 3, 5)
print(rand_list)
```

- Модуль *ex_2.py*

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

data1 = [1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 3, 10)

a = Unique(data1)
print([i for i in a if i != None])

data = ['a', 'B', 'A', 'b']
c = Unique(data)
print([i for i in c if i != None])
```

- Модуль *ex_3.py*

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

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
print(sorted(data, key=abs))
```

- Модуль ex_4.py

```
from librip.decorators import print_result
```

```
@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()
```

- Модуль ex_5.py

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

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

- Модуль ex_6.py

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

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

```
# Helpful functions
```

```
def __prog(arr):
    if 'программист' in arr:
        return arr
```

```
def python(arr):
    return str(arr) + ' с опытом Python'
```

```

@print_result
def f1(arg):
    return sorted(list(field(data, 'job-name')))

@print_result
def f2(arg):
    return list(filter(__prog, f1()))

@print_result
def f3(arg):
    return list(map(python, f2()))

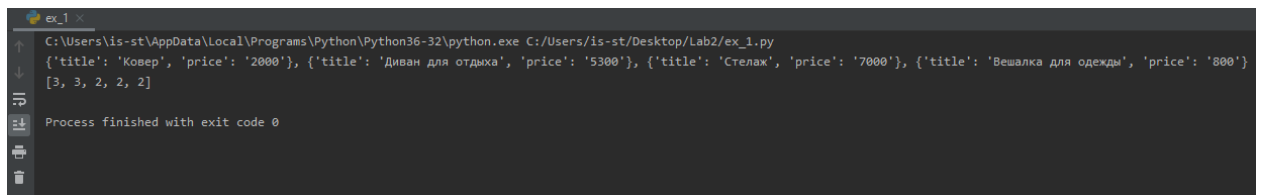
@print_result
def f4(arg):
    return list(zip(f3(), gen_random(100000, 200000, len(f3()))))

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

```

Скриншоты с результатами выполнения

Скриншот с результатами выполнения ex_1.py:

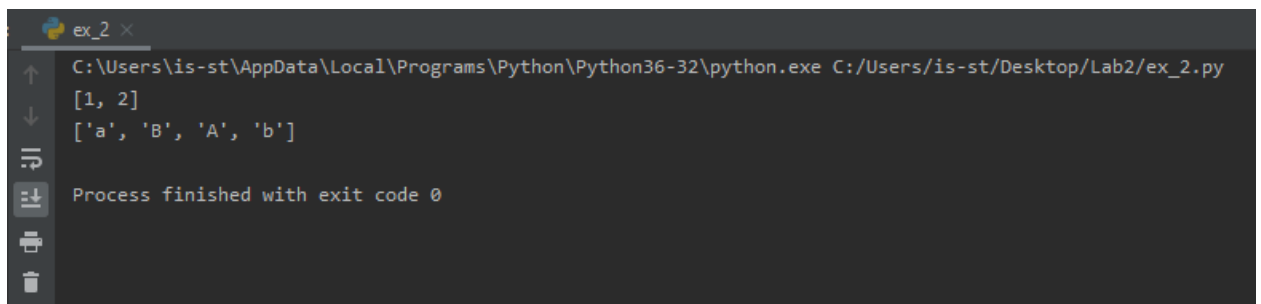


```

ex_1 x
C:\Users\is-st\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/is-st/Desktop/Lab2/ex_1.py
{'title': 'Ковер', 'price': '2000'}, {'title': 'Диван для отдыха', 'price': '5300'}, {'title': 'Стелаж', 'price': '7000'}, {'title': 'Вешалка для одежды', 'price': '800'}
[3, 3, 2, 2, 2]
Process finished with exit code 0

```

Скриншот с результатами выполнения ex_2.py:

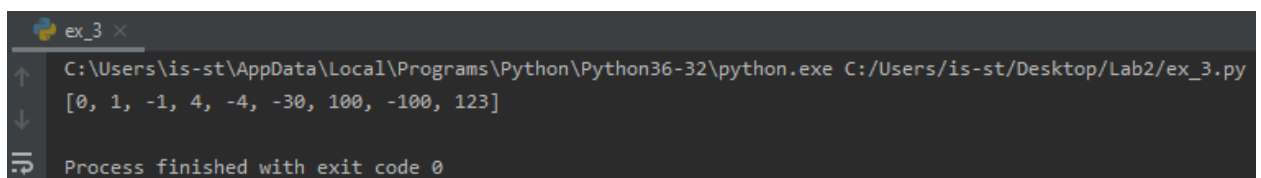


```

ex_2 x
C:\Users\is-st\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/is-st/Desktop/Lab2/ex_2.py
[1, 2]
['a', 'B', 'A', 'b']
Process finished with exit code 0

```

Скриншот с результатами выполнения ex_3.py:

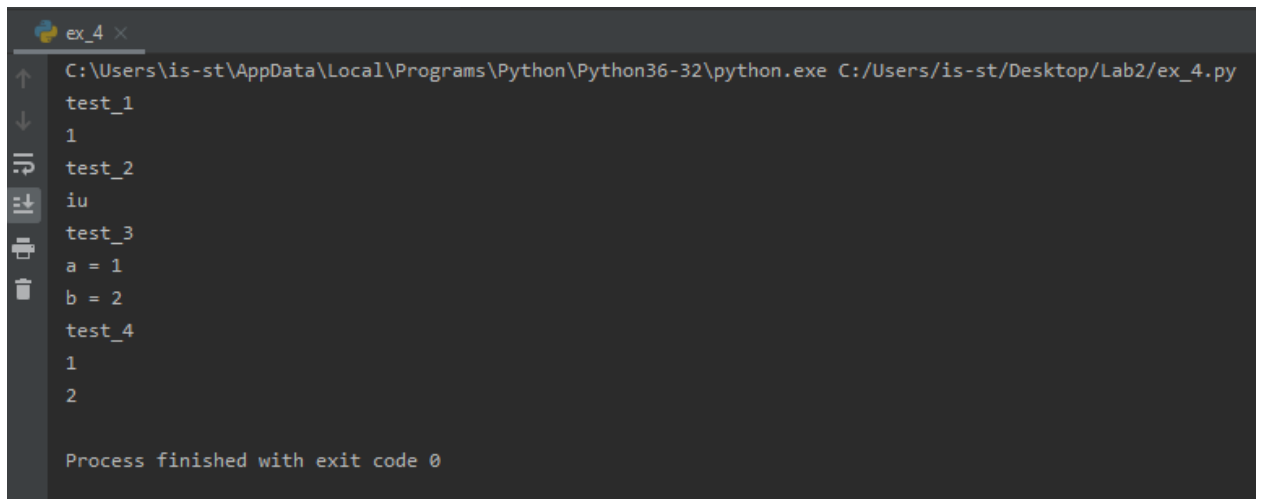


```

ex_3 x
C:\Users\is-st\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/is-st/Desktop/Lab2/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]
Process finished with exit code 0

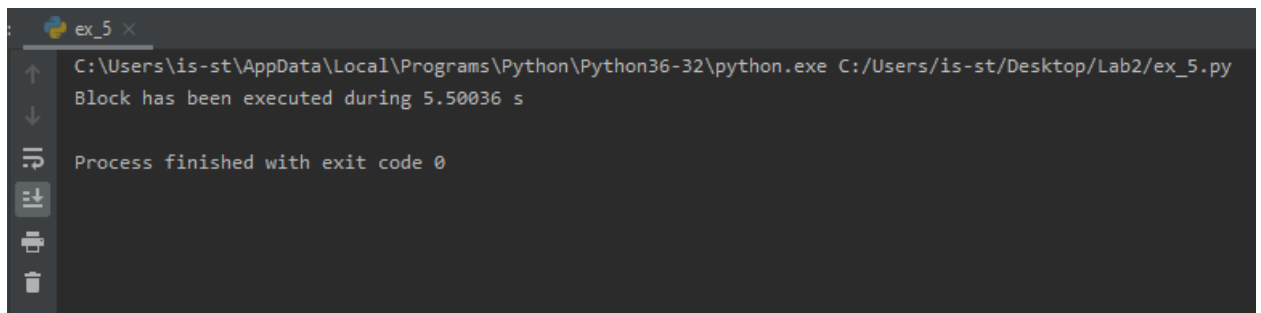
```


Скриншот с результатами выполнения ex_4.py:



```
ex_4 x
C:\Users\is-st\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/is-st/Desktop/Lab2/ex_4.py
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2
Process finished with exit code 0
```

Скриншот с результатами выполнения ex_5.py:



```
ex_5 x
C:\Users\is-st\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/is-st/Desktop/Lab2/ex_5.py
Block has been executed during 5.50036 s
Process finished with exit code 0
```