

Московский государственный технический университет им. Н.Э. Баумана  
Факультет «Информатика и системы управления» Кафедра «Системы  
обработки информации и управления»



Студент Андреев К.А.

Отчет

по выполнению лабораторной работы По курсу  
“Разработка интернет-приложений”

Лабораторная работа № 3

## 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) 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`.

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
t
e
s
t

_
2
i
u
t
e
s
t

_
3
a
=
1
b
=
2
t
e
s
```

t  
—  
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 `f1f3` 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

файлы из **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 datetime import contextlib

# @contextlib.contextmanager
# def timer():
#     t1 = datetime.datetime.now() #
yield
#     t2 = datetime.datetime.now()
#     res = t2 - t1
#     res1 = str(res.seconds) + '.' + str(res.microseconds)
#     print('Execution time '+res1)
class timer:
    t1 = 0
def __enter__(self):
    self.t1 = datetime.datetime.now()
def
__exit__(self, exp_type, exp_value, traceback):
    t2 = datetime.datetime.now()
res = t2 - self.t1
res1 = str(res.seconds) + '.' + str(res.microseconds)
print('Execution time ' + res1)
def 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
# Пример из ex_4.py: #
@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 def print2(func,
res):
print(func.__name__)      a
= type(res).__name__
if a == 'list':
    [print(i) for i in res]
elif a == 'dict':
    [print('{0}={1}'.format(k,v)) for k,v in res.items()]
else:
    print(res)
def
print_result(func):
    def decfunc(*args,**kwargs):
res = func(*args,**kwargs)
print2(func, res)        return
res        return decfunc 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 get_data(item,
key):
    try:
        res = item[key]
return res    except

```

```

Exception:         return
None
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        key = args[0]
        for item in items:
            res = get_data(item, key)
            yield res
    else:
        for item in items:
            res = {
                key: get_data(item, key) for key in args }
            res.copy()
            for key in res:
                if res[key] == None:
                    y.pop(key)
            yield y

```

*# Необходимо реализовать генератор*

```

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

# Необходимо реализовать генератор 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
        self.items = iter(items) if isinstance(items, list) else items
        self.ignore_case = False
        self.ignore_case = kwargs.get('ignore_case',
        False)
        self.dupl =
        []
    def
    __next__(self):
        # Нужно реализовать __next__
    while True:
        try:
            cur = next(self.items)
            if isinstance(cur, str) and self.ignore_case is
            True:
                check = cur.upper()
            else:
                check = cur

```



```

        if not check in self.dupl:
self.dupl.append(check)
return cur        except Exception:
        raise StopIteration

    def
__iter__(self):
return self
Основные
файлы задания: ex_1.py:
#!/usr/bin/env python3
from librip.gens import field, gen_random
goods =
[
    {'title': 'Ковер', '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([x for x in field(goods, 'title', 'price')])
print([ x for x in field(goods, 'titl', 'price')]) print()
print([x for x in 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)
print([x for x in Unique(['A', 'a', 'B', 'b'], ignore_case=True)]) print([x
for x in Unique(data1)])

# Реализация задания 2

ex_3.py:
#!/usr/bin/env python3
data = [4, -30, 100, -100, 123, 1, 0, -1, -
4]
# Реализация задания 3
res = sorted(data, key=lambda x: abs(x)) print(res)

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(1)
```

**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 = 'data_light.json' path
= sys.argv[1]
```

```
# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске
with open(path, encoding="utf8") 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([x for x in unique(field(arg, 'job-name'), ignore_case=True)])
@print_result def f2(arg):
    return list(filter(lambda x: x.startswith("Программист") , arg))
```

```

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

@print_result def
f4(arg):
    salary = gen_random(100000, 200000, len(arg))
    return [i + ', зарплата ' + str(j) + ' руб.' for i,j in zip(arg, salary)]

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

```

```

C:\Users\kiril\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/kiril/Desktop/GITHUB/lab3/ex_1.py
[{'title': 'Ковер'}, {'title': 'диван для отдыха', 'price': 5300}, {'title': 'Стелаж', 'price': 7000}, {'title': 'Вешалка для одежды', 'price': 800}]

[3, 1, 4, 5, 5]

Process finished with exit code 0

C:\Users\kiril\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/kiril/Desktop/GITHUB/lab3/ex_2.py
['A', 'B']
[1, 2]

Process finished with exit code 0

C:\Users\kiril\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/kiril/Desktop/GITHUB/lab3/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]

Process finished with exit code 0

C:\Users\kiril\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/kiril/Desktop/GITHUB/lab3/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

C:\Users\kiril\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/kiril/Desktop/GITHUB/lab3/ex_5.py
Execution time 1.33

Process finished with exit code 0

```

```

C:\WINDOWS\system32\cmd.exe
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программист-разработчик информационных систем
f3
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
f4
Программист с опытом Python, зарплата 180290 руб.
Программист / Senior Developer с опытом Python, зарплата 163862 руб.
Программист 1С с опытом Python, зарплата 108603 руб.
Программист C# с опытом Python, зарплата 147348 руб.
Программист C++ с опытом Python, зарплата 154727 руб.
Программист C++/C#/Java с опытом Python, зарплата 142116 руб.
Программист/ Junior Developer с опытом Python, зарплата 190298 руб.
Программист/ технический специалист с опытом Python, зарплата 149811 руб.
Программист-разработчик информационных систем с опытом Python, зарплата 113520 руб.
Execution time 1.580519
C:\Users\kiril\Desktop\GITHUB\lab3>

```

## Результаты