



**Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ

ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА

СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

**Отчет по лабораторной работе № 3
«Python. Функциональные возможности»
по курсу “Разработка интернет-приложений”**

Исполнитель:
Студент группы ИУ5-53
Желанкина А.С.
_____ 08.10.2018

Москва, 2018

Задание 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_3`
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, 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.

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

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). 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

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')))
print(list(field(goods, 'title', 'price')))
print(list(field(goods, 'title', 'price', 'color')))

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

ex_3.py

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

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3

print(sorted(data, key = lambda i : math.fabs(i)))
```

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

path = "C:/learning/data_light_cp1251.json"

with open(path) 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(Unique(sorted(list(field(arg, "job-name"))),
ignore_case=True))

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

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

@print_result
def f4(arg):
    salaries = list(gen_random(100000, 200000, len(list(arg))))
    return list(zip(list(arg), list(map(lambda x: "зарплата " + str(x) + "
руб.", salaries))))

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

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 __init__(self):
        pass

    def __enter__(self):
        self.time = time.clock()
```

```
def __exit__(self, type, value, traceback):
    print(time.clock() - self.time)
```

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 print_result(func, *arg):
    def decorated_function(*arg):
        result = func(*arg)
        print(func.__name__)
        if type(result) is dict:
            for k, v in result.items():
                print('{} = {}'.format(k, v))
            elif type(result) is list:
                for i in result:
                    print(i)
            else:
                print(result)
        return result
    return decorated_function
```

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
    if len(args) == 1:
        for i in items:
            for a in args:
                if i[a] is not None:
                    yield i[a]
    else:
        for i in items:
            if i.values() is not None:
                for a in args:
                    if i[a] is not None:
                        yield {a: i[a]}

```

```

# 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 n in range(num_count):
        yield random.randint(begin, end)
    pass
    # 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.unique = []
        self.items = iter(items)
        if len(kwargs) == 0:
            self.ignore_case = False
        else:
            self.ignore_case = kwargs.values()
    pass

    def __next__(self):
        # Нужно реализовать __next__
        while True:
            item = self.items.__next__()
            smth = None
            if self.ignore_case and type(item) is str:
                smth = item.lower()

```

```

else:
    smth = item
if smth not in self.unique:
    self.unique.append(smth)
return smth

pass

def __iter__(self):
    return self

```

Результаты выполнения

```

ex_1 x
C:\Users\Анечка\PycharmProjects\lab3_func_prop\venv\Scripts\python.exe C:/Users/Анечка/PycharmProjects/lab3_func_prop/ex_1.py
['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'}, {'title': 'Ковер'}, {'price': 2000}, {'color': 'green'}, {'title': 'Диван для отдыха'}, {'price': 5300}, {'color': 'black'}, {'title': 'Стелаж'}, {'color': 'black'}]
[1, 1, 2, 2, 3]

Process finished with exit code 0

```

```

ex_2 x
C:\Users\Анечка\PycharmProjects\lab3_func_prop\venv\Scripts\python.exe C:/Users/Анечка/PycharmProjects/lab3_func_prop/ex_2.py
[1, 2]
[1, 3, 2]
['A', 'a', 'b', 'B']
['a', 'b']

Process finished with exit code 0

```

```

ex_3 x
C:\Users\Анечка\PycharmProjects\lab3_func_prop\venv\Scripts\python.exe C:/Users/Анечка/PycharmProjects/lab3_func_prop/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]

Process finished with exit code 0

```

```

ex_4 x
C:\Users\Анечка\PycharmProjects\lab3_func_prop\venv\Scripts\python.exe C:/Users/Анечка/PycharmProjects/lab3_func_prop/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

```

```

ex_5 x
C:\Users\Анечка\PycharmProjects\lab3_func_prop\venv\Scripts\python.exe C:/Users/Анечка/PycharmProjects/lab3_func_prop/ex_5.py
5.48888465900529

Process finished with exit code 0

```

```

ex_6 x
('программист с опытом Python', 'зарплата 100000 руб.')
('программист / senior developer с опытом Python', 'зарплата 107422 руб.')
('программист lcc с опытом Python', 'зарплата 147654 руб.')
('программист c# с опытом Python', 'зарплата 120150 руб.')
('программист c++ с опытом Python', 'зарплата 102057 руб.')
('программист c++/c#/java с опытом Python', 'зарплата 181601 руб.')
('программист/ junior developer с опытом Python', 'зарплата 139835 руб.')
('программист/ технический специалист с опытом Python', 'зарплата 174756 руб.')
('программист-разработчик информационных систем с опытом Python', 'зарплата 183111 руб.')
('системный программист (с, linux) с опытом Python', 'зарплата 150395 руб.')
('старший программист с опытом Python', 'зарплата 199096 руб.')
('инженер - программист с опытом Python', 'зарплата 103568 руб.')
('педагог программист с опытом Python', 'зарплата 191090 руб.')
0.0931467021603664

Process finished with exit code 0

```