



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

Факультет «Информатика и системы управления»  
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3  
«Функциональные возможности языка Python»

Выполнил:

студент группы ИУ5-32Б

Еремихин Владислав

Москва, 2021 г.

## Общее описание задания

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fr. Решение каждой задачи должно располагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

## Задача 1 (файл field.py)

### Описание задания:

Необходимо реализовать генератор `field`, который последовательно выдает значения ключей словаря.

- В качестве первого аргумента генератор принимает список словарей, дальше через `*args` генератор принимает неограниченное количество аргументов.
- Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно `None`, то элемент пропускается.
- Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно `None`, то оно пропускается. Если все поля содержат значения `None`, то пропускается элемент целиком.

### Текст программы:

```
def field(dicts, *keys):  
  
    assert len(keys) > 0  
  
    if len(keys) == 1:  
        for d in dicts:  
            val = d.get(keys[0])  
            if val != None:  
                yield val  
    else:  
        for d in dicts:  
            res_dict = dict()  
            for key in keys:  
                val = d.get(key)  
                if val != None:  
                    res_dict[key] = val  
            if len(res_dict) != 0:  
                yield res_dict  
  
if __name__ == '__main__':
```

```

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
]

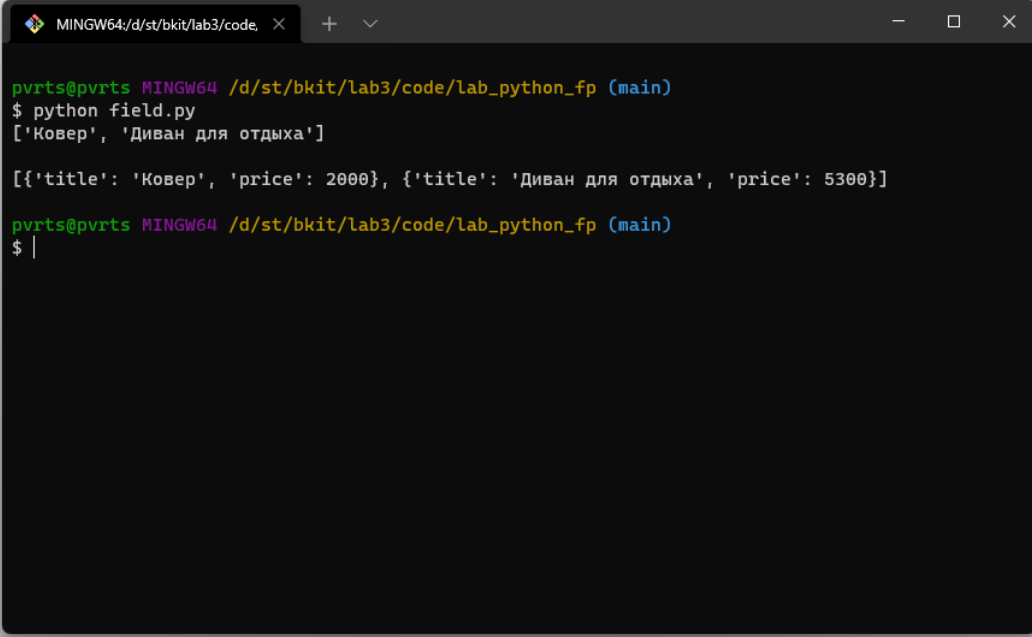
ls1 = list()
ls2 = list()

for i in field(goods, 'title'):
    ls1.append(i)
print(str(ls1))
print()

for i in field(goods, 'title', 'price'):
    ls2.append(i)
print(ls2)

```

### Примеры выполнения программы:



```

MINGW64/d/st/bkit/lab3/code, x + v - □ x
pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$ python field.py
['Ковер', 'Диван для отдыха']

[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]

pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$ |

```

## Задача 2 (файл gen\_random.py)

### Описание задания:

Необходимо реализовать генератор `gen_random(количество, минимум, максимум)`, который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона.

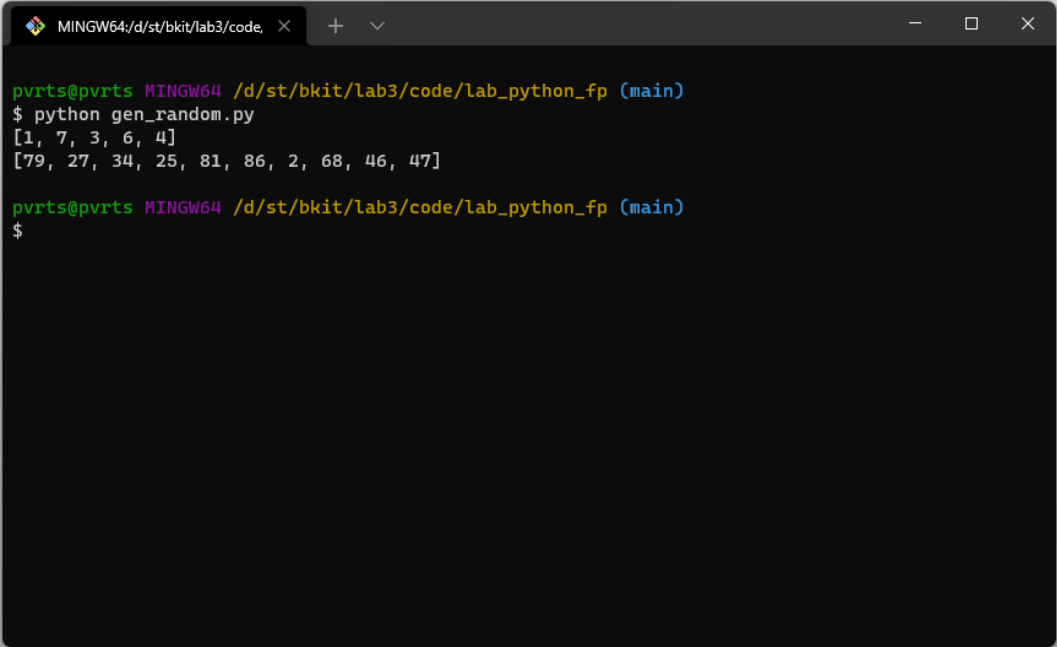
### Текст программы:

```
import random

def gen_random(amount, begin, end):
    for i in range(amount):
        yield random.randint(begin, end)

if __name__ == '__main__':
    print(list(gen_random(5, 1, 10)))
    print(list(gen_random(10, 1, 100)))
```

### Примеры выполнения программы:



```
MINGW64/d/st/bkit/lab3/code, x + v
pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$ python gen_random.py
[1, 7, 3, 6, 4]
[79, 27, 34, 25, 81, 86, 2, 68, 46, 47]

pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$
```

## Задача 3 (файл unique.py)

### Описание задания:

- Необходимо реализовать итератор `Unique(данные)`, который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.
- Конструктор итератора также принимает на вход именованный `bool`-параметр `ignore_case`, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен `False`.
- При реализации необходимо использовать конструкцию `**kwargs`.
- Итератор должен поддерживать работу как со списками, так и с генераторами.

- Итератор не должен модифицировать возвращаемые значения.

#### Текст программы:

```
from .gen_random import gen_random

class Unique(object):
    def __init__(self, items, ignore_case = False, **kwargs):
        self.seen = set()
        self.items = items
        self.ic = ignore_case
        self.kwargs = kwargs

    def __next__(self):
        it = iter(self.items)
        while True:
            try:
                current = next(it)
            except StopIteration:
                raise
            else:
                if self.ic == True and isinstance(current, str):
                    temp = current[:]
                    if temp.lower() not in self.seen:
                        self.seen.add(temp.lower())
                    return current
                elif current not in self.seen:
                    self.seen.add(current)
                    return current

    def __iter__(self):
        return self

if __name__ == '__main__':
    lst = ['a', "B", 'c', 'd', 'c', "A", 'b', "C", 'c', 'b', 1, 2, 2, 3, 3, 1,
2, 3, 4]

    print(list(Unique(gen_random(50, 1, 4))))

    print(list(Unique(lst)))

    print(list(Unique(lst, ignore_case = True)))

    # Исходный список
    print(lst)
```

#### Примеры выполнения программы:

```
MINGW64:/d/st/bkit/lab3/code x + -
pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$ python unique.py
[2, 4, 3, 1]
['a', 'B', 'c', 'd', 'A', 'b', 'C', 1, 2, 3, 4]
['a', 'B', 'c', 'd', 1, 2, 3, 4]
['a', 'B', 'c', 'd', 'c', 'A', 'b', 'C', 'c', 'b', 1, 2, 2, 3, 3, 1, 2, 3, 4]

pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$
```

## Задача 4 (файл sort.py)

### Описание задания:

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо **одной строкой кода** вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции `sorted`.

Необходимо решить задачу двумя способами:

1. С использованием `lambda`-функции.
2. Без использования `lambda`-функции.

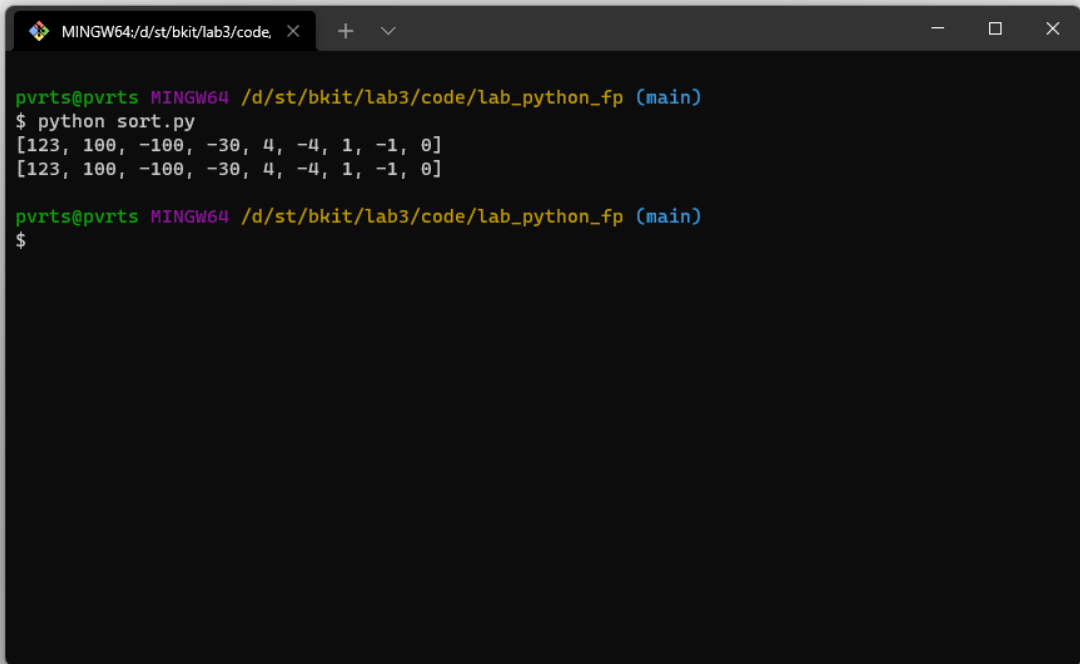
### Текст программы:

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

if __name__ == '__main__':
    result = sorted(data, key=abs, reverse=True)
    print(result)

    result_with_lambda = sorted(data, key=lambda x: x if x > 0 else -
x, reverse=True)
    print(result_with_lambda)
```

## Примеры выполнения программы:



A screenshot of a terminal window with a dark background. The window title bar shows 'MINGW64: d:/st/bkit/lab3/code' and standard window controls. The terminal text shows a user prompt 'pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab\_python\_fp (main)' followed by the command '\$ python sort.py'. The output consists of two identical lines: '[123, 100, -100, -30, 4, -4, 1, -1, 0]'. The prompt '\$' appears again on the next line.

```
MINGW64: d:/st/bkit/lab3/code
pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$ python sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$
```

## Задача 5 (файл print\_result.py)

### Описание задания:

Необходимо реализовать декоратор `print_result`, который выводит на экран результат выполнения функции.

- Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.
- Если функция вернула список (`list`), то значения элементов списка должны выводиться в столбик.
- Если функция вернула словарь (`dict`), то ключи и значения должны выводиться в столбик через знак равенства.

### Текст программы:

```
def print_result(func):
    def decorated_func(*args, **kwargs):
        res = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(res, list):
            for i in res:
                print(i)
```

```

        elif isinstance(res, dict):
            for k, v in res.items():
                print('{} = {}'.format(k, v))
        else:
            print(res)
        return res;
    return decorated_func

@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'iu5'

@print_result
def test_3():
    return {'a': 1, 'b': 2}

@print_result
def test_4():
    return [1, 2]

if __name__ == '__main__':
    print('!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()

```

**Примеры выполнения программы:**

```

pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$ python print_result.py
!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

pvrts@pvrts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$

```



## Задача 6 (файл cm\_timer.py)

### Описание задания:

Необходимо написать контекстные менеджеры `cm_timer_1` и `cm_timer_2`, которые считают время работы блока кода и выводят его на экран.

После завершения блока кода в консоль должно выводиться `time: 5.5`.

`cm_timer_1` и `cm_timer_2` реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки `contextlib`).

### Текст программы:

```
from contextlib import contextmanager
import time

class cm_timer_1():
    def __init__(self):
        self.start_time = None

    def __enter__(self):
        self.start_time = time.time()

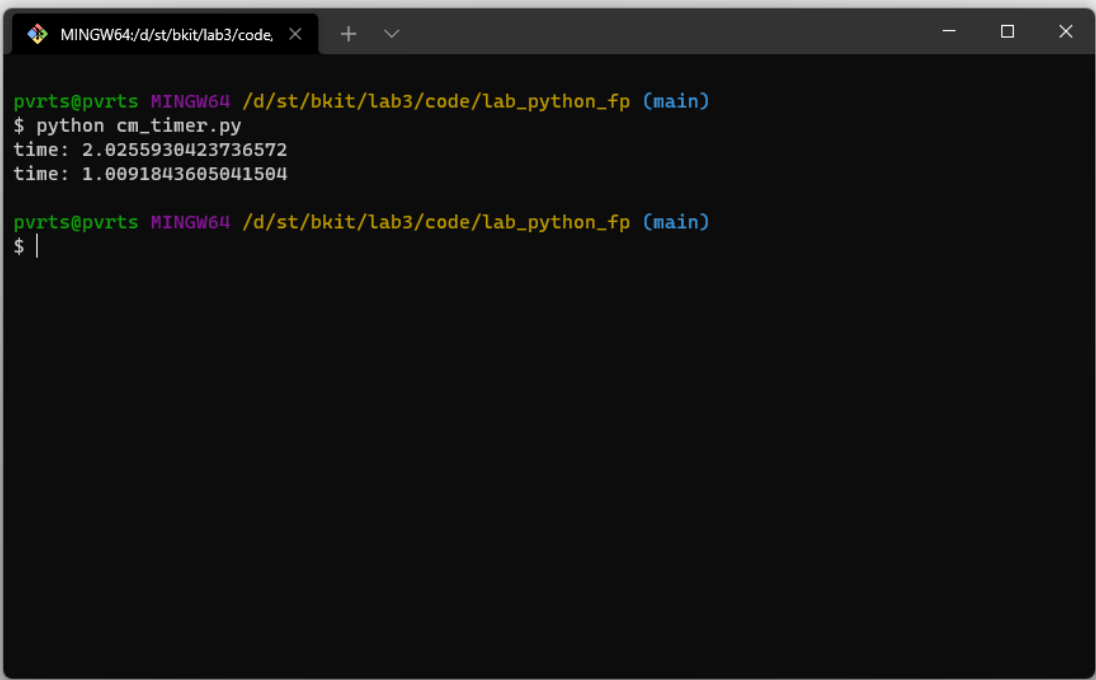
    def __exit__(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
        else:
            print('time: {}'.format(time.time() - self.start_time))

@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    print('time: {}'.format(time.time() - start_time))

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(1)
        time.sleep(1)

    with cm_timer_2():
        time.sleep(1)
```

## Примеры выполнения программы:



```
MINGW64:/d/st/bkit/lab3/code, x + -
pvrts@pvirts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$ python cm_timer.py
time: 2.0255930423736572
time: 1.0091843605041504

pvrts@pvirts MINGW64 /d/st/bkit/lab3/code/lab_python_fp (main)
$ |
```

## Задача 7 (файл process\_data.py)

### Описание задания:

- Необходимо реализовать 4 функции - f1, f2, f3, f4. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print\_result печатается результат, а контекстный менеджер cm\_timer\_1 выводит время работы цепочки функций.
- Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр.
- Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова "программист".
- Функция f3 должна модифицировать каждый элемент массива, добавив строку "с опытом Python" (все программисты должны быть знакомы с Python).
- Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности.

### Текст программы:

```
import sys
import json
from lab_python_fp import cm_timer_1, Unique, field, print_result, gen_random

try:
    path = sys.argv[1]
except IndexError:
    raise ValueError("Не указан путь к файлу.")
else:
    with open(path, encoding='utf-8') as f:
        data = json.load(f)

@print_result
def f1(lst):
    return sorted(list(Unique(field(lst, 'job-
name'), ignore_case=True)), key=str.lower)

@print_result
def f2(lst):
    return list(filter(lambda s: str.startswith(str.lower(s), 'программист'),
lst))

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

@print_result
def f4(lst):
    return dict(zip(lst, list('зарплата {} руб.'.format(num) for num in gen_r
andom(len(lst), 1000000, 2000000))))

if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

### Пример выполнения программы:

```
MINGW64/d/st/bkit/lab3/code x + v - □ x

pvrts@pvrts MINGW64 /d/st/bkit/lab3/code (main)
$ python process_data.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
...
Юрист
Юрист (специалист по сопровождению международных договоров, английский – разговорный)
Юрист волонтер
Юристконсульт
f2
Программист
Программист / Senior Developer
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 1492992 руб.
Программист / Senior Developer с опытом Python = зарплата 1389128 руб.
Программист 1С с опытом Python = зарплата 1708989 руб.
Программист C# с опытом Python = зарплата 1956494 руб.
Программист C++ с опытом Python = зарплата 1443300 руб.
Программист C++/C#/Java с опытом Python = зарплата 1000173 руб.
Программист/ Junior Developer с опытом Python = зарплата 1664465 руб.
Программист/ технический специалист с опытом Python = зарплата 1978257 руб.
Программист-разработчик информационных систем с опытом Python = зарплата 1445516 руб.
time: 0.19203686714172363

pvrts@pvrts MINGW64 /d/st/bkit/lab3/code (main)
$ |
```