

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

Факультет «Информатика и системы управления»  
Кафедра «Автоматизированные системы обработки  
информации и управления»  
Дисциплина «Разработка интернет-приложений»

## Отчёт по лабораторной работе №3

Выполнил:  
Студент группы ИУ5-53Б  
Аникин Ф.А.

Проверил:  
Преподаватель  
Гапанюк Ю.Е.

Москва, 2020 г.

## **Постановка задачи**

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)**

Необходимо реализовать генератор field. Генератор field 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(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for item in items:
            for arg in args:
                if arg in item:
                    yield item[arg]
    else:
        for item in items:
            new_item = {}
            arg_c = 0
            for arg in item:
                if arg in args:
                    arg_c += 1
            if arg_c == len(args):
                for i in item:
                    new_item[i] = item[i]
                yield new_item

if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
        {'title': 'Кресло', 'color': 'brown', 'm': 'meow'}
    ]

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

### Пример работы:

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
['green', 'black', 'brown']
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]
Press any key to continue . . .
```

## Задача 2 (файл gen\_random.py)

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(count, start, end):
    for _ in range(count):
        yield random.randint(start, end)

if __name__ == '__main__':
    print(list(gen_random(10, 2, 8)))
```

### Пример работы:

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
[3, 7, 8, 6, 7, 3, 6, 2, 5, 5]
Press any key to continue . . .
```

## Задача 3 (файл unique.py)

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:

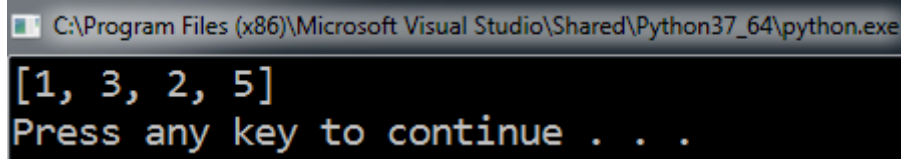
```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.unique_items = []
        self.items = iter(items)
        if 'ignore_case' not in kwargs:
            self.ignore_case = False
        else:
            self.ignore_case = kwargs['ignore_case']

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            item = self.items.__next__()
            current_item = None
            if self.ignore_case and type(item) is str:
                current_item = item.lower()
            else:
                current_item = item
            if current_item not in self.unique_items:
                self.unique_items.append(current_item)
            return item

if __name__ == '__main__':
    data = [1, 3, 1, 3, 1, 2, 2, 5, 2, 2]
    print(list(Unique(data)))
```

Пример работы:



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
[1, 3, 2, 5]
Press any key to continue . . .
```

## Задача 4 (файл sort.py)

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо одной строкой кода вывести на экран массив 2, который содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции `sorted`. Необходимо решить задачу с использованием `lambda`-функции и без.

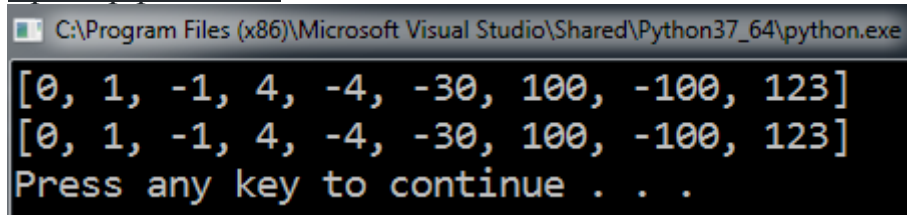
Текст программы:

```
from math import fabs

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

if __name__ == '__main__':
    print(sorted(data, key=fabs, reverse=False))
    print(sorted(data, key=lambda i: fabs(i), reverse=False))
```

Пример работы:



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
[0, 1, -1, 4, -4, -30, 100, -100, 123]
[0, 1, -1, 4, -4, -30, 100, -100, 123]
Press any key to continue . . .
```

## Задача 5 (файл print\_result.py)

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, *args):
    def decorator(*args):
        print(func.__name__)
        result = func(*args)
        if type(result) is list:
            for item in result:
                print(item)
        elif type(result) is dict:
            for key, item in result.items():
                print(str(key) + ' = ' + str(item))
        else:
            print(result)
        return result
    return decorator

if __name__ == '__main__':
    @print_result
    def test_1():
        return 1

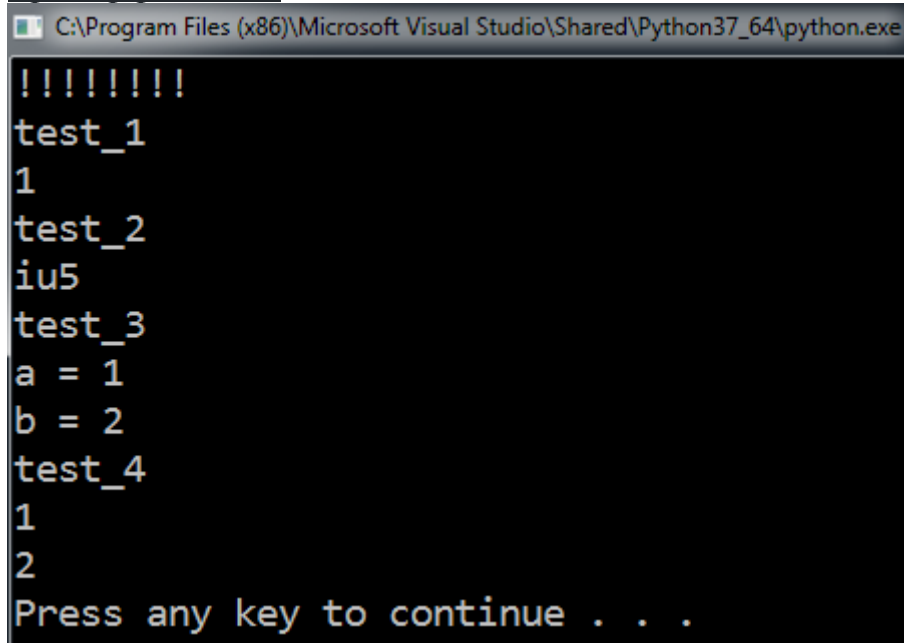
    @print_result
    def test_2():
        return 'iu5'

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

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

```
print('!!!!!!!')
test_1()
test_2()
test_3()
test_4()
```

*Пример работы:*



```
!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
Press any key to continue . . .
```

## Задача 6 (файл cm\_timer.py)

Необходимо написать контекстные менеджеры cm\_timer\_1 и cm\_timer\_2, которые считают время работы блока кода и выводят его на экран.

*Текст программы:*

```
import time

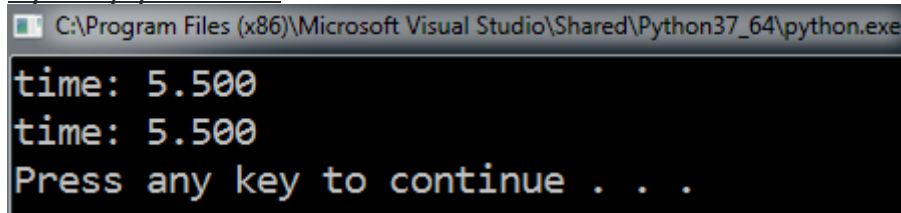
class cm_timer_1:
    def __enter__(self):
        self.time = time.time()
    def __exit__(self, value, key, traceback):
        print(f"time: {time.time()-self.time:0.3f}")

class cm_timer_2:
    def __init__(self):
        self._start_time = None
    def __enter__(self):
        self._start_time = time.perf_counter()
    def __exit__(self, value, key, traceback):
        elapsed_time = time.perf_counter() - self._start_time
        self._start_time = None
        print(f"time: {elapsed_time:0.3f}")

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(5.5)
```

```
with cm_timer_2():  
    time.sleep(5.5)
```

*Пример работы:*



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe  
time: 5.500  
time: 5.500  
Press any key to continue . . .
```

## Задача 7 (файл process\_data.py)

- В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.
- В файле data\_light.json содержится фрагмент списка вакансий.
- Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.
- 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.
- Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.
- Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.
- Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова "программист". Для фильтрации используйте функцию filter.
- Функция f3 должна модифицировать каждый элемент массива, добавив строку "с опытом Python" (все программисты должны быть знакомы с Python). Пример: Программист С# 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 С# с опытом Python, зарплата 137287 руб. Используйте zip для обработки пары специальность — зарплата.

### Код программы:

```
from lab_python_fp.field import field
from lab_python_fp.unique import Unique
from lab_python_fp.print_result import print_result
from lab_python_fp.gen_random import gen_random
from lab_python_fp.cm_timer import cm_timer_1, cm_timer_2
import json
import sys

path = "C:\\Users\\FIL\\Desktop\\Python\\5 семестр\\Lab_3\\data_light.json"

with open(path, encoding='utf8') as f:
    data = json.load(f)

@print_result
def f1(arg):
    return sorted(list(Unique(field(arg, 'job-name'), ignore_case=True)))

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

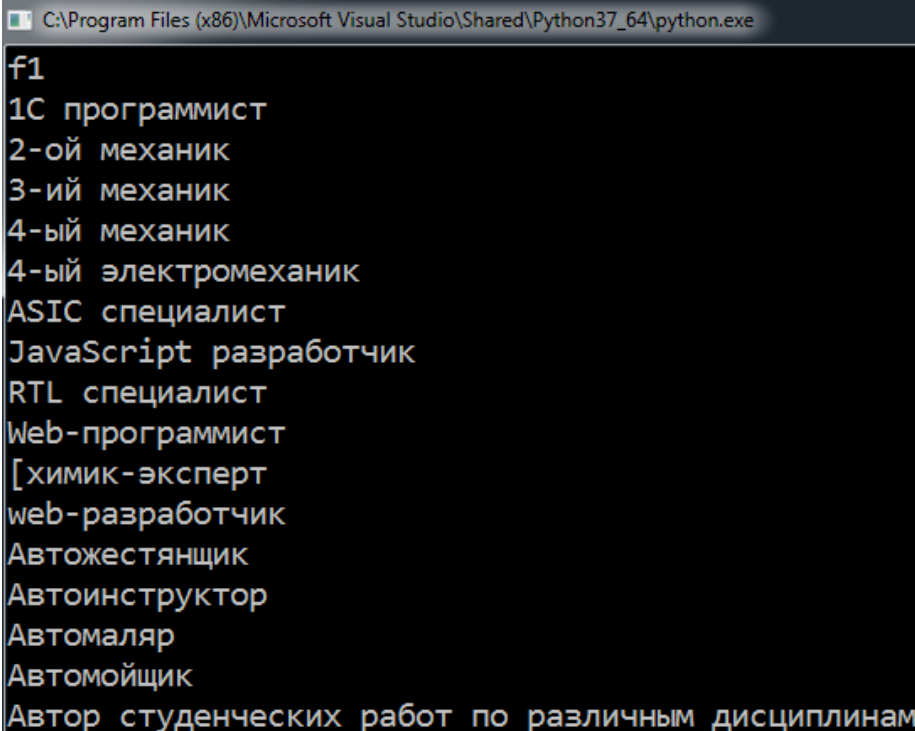
@print_result
def f3(arg):
    return list(map(lambda x: x + " с опытом Python", arg))

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

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

### Результат работы программы:

1) фрагмент работы *f1* по сортировке списка профессий без повторений



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
[химик-эксперт
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
```



2) фрагмент работы *f2* по фильтрации массива по наличию в названии слова “программист”

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
f2
1С программист
Web-программист
Веб - программист (PHP, JS) / Web разработчик
Веб-программист
Ведущий инженер-программист
Ведущий программист
Инженер - программист АСУ ТП
Инженер-программист (Клинский филиал)
Инженер-программист (Орехово-Зуевский филиал)
Инженер-программист 1 категории
Инженер-программист ККТ
Инженер-программист ПЛИС
Инженер-программист САПОУ (java)
Инженер-электронщик (программист АСУ ТП)
```

3) фрагмент работы *f3* по модификации массива с добавлением строки “с опытом Python”

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
f3
1С программист с опытом Python
Web-программист с опытом Python
Веб - программист (PHP, JS) / Web разработчик с опытом Python
Веб-программист с опытом Python
Ведущий инженер-программист с опытом Python
Ведущий программист с опытом Python
Инженер - программист АСУ ТП с опытом Python
Инженер-программист (Клинский филиал) с опытом Python
Инженер-программист (Орехово-Зуевский филиал) с опытом Python
Инженер-программист 1 категории с опытом Python
Инженер-программист ККТ с опытом Python
Инженер-программист ПЛИС с опытом Python
Инженер-программист САПОУ (java) с опытом Python
Инженер-электронщик (программист АСУ ТП) с опытом Python
Помощник веб-программиста с опытом Python
Программист с опытом Python
```

4) работа *f4* по генерации для каждой специальности зарплаты, а также вывод в конце времени работы цепочки функций через контекстный менеджер

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
f4
1С программист с опытом Python = 161496
Web-программист с опытом Python = 173550
Веб - программист (PHP, JS) / Web разработчик с опытом Python = 162960
Веб-программист с опытом Python = 134754
Ведущий инженер-программист с опытом Python = 149935
Ведущий программист с опытом Python = 165883
Инженер - программист АСУ ТП с опытом Python = 184061
Инженер-программист (Клинский филиал) с опытом Python = 172994
Инженер-программист (Орехово-Зуевский филиал) с опытом Python = 103446
Инженер-программист 1 категории с опытом Python = 132802
Инженер-программист ККТ с опытом Python = 108873
Инженер-программист ПЛИС с опытом Python = 133192
Инженер-программист САПОУ (java) с опытом Python = 180171
Инженер-электронщик (программист АСУ ТП) с опытом Python = 115037
Помощник веб-программиста с опытом Python = 195693
Программист с опытом Python = 145888
Программист / Senior Developer с опытом Python = 130122
Программист 1С с опытом Python = 117434
Программист C# с опытом Python = 129835
Программист C++ с опытом Python = 165063
Программист C++/C#/Java с опытом Python = 140540
Программист/ Junior Developer с опытом Python = 189410
Программист/ технический специалист с опытом Python = 106688
Программист-разработчик информационных систем с опытом Python = 161808
Системный программист (C, Linux) с опытом Python = 167302
Старший программист с опытом Python = 166578
инженер - программист с опытом Python = 177679
инженер-программист с опытом Python = 181612
педагог программист с опытом Python = 173253
time: 0.314
Press any key to continue . . .
```