



**«Московский государственный технический университет  
имени Н.Э. Баумана»  
(МГТУ им. Н.Э. Баумана)**

---

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления (ИУ5)

**О т ч е т**

**по лабораторной работе №3**

**Дисциплина: Разработка Интернет-Приложений**

Студент гр. ИУ5-53Б

\_\_\_\_\_  
(Подпись, дата)

Ваксина И.Р.

(И.О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Гапанюк Ю.Е.

(И.О. Фамилия)

Москва, 2020

## **1. Цель работы**

Изучение возможностей функционального программирования в языке Python.

## **2. Задание**

- 1 Задание лабораторной работы состоит из решения нескольких задач.
- 2 Файлы, содержащие решения отдельных задач, должны располагаться в пакете `lab_python_fr`. Решение каждой задачи должно располагаться в отдельном файле.
- 3 При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

## **3. Задача 1**

### **3.1. Условие**

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.

### **3.2. Текст программы**

```

lab_python_fp > field.py
1  def field(items, *args):
2      assert len(args) > 0
3      if len(args) == 1:
4          for item in items:
5              for arg in args:
6                  if arg in item:
7                      yield item[arg]
8      else:
9          for item in items:
10             new_item = {}
11             for arg in args:
12                 if arg in item:
13                     new_item[arg] = item[arg]
14             if len(new_item.keys()) > 0:
15                 yield new_item
16
17
18  goods = [
19      {'title': 'Ковер', 'price': 2000, 'color': 'green'},
20      {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
21      {'color': 'black'}
22  ]
23  if __name__ == '__main__':
24      for i in field(goods, 'title'):
25          print (i)
26      #field(goods, 'title', 'price')

```

### 3.3. Результат выполнения программы

```

PS C:\Users\Ия\OneDrive\Рабочий стол\lab3\lab_python_fp> python field.py
Ковер
Диван для отдыха

```

## 4. Задача 2

### 4.1. Условие

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.

### 4.2. Текст программы

```

lab_python_fp > gen_random.py
1  import random
2  # Пример:
3  # gen_random(5, 1, 3) должен выдать 5 случайных чисел
4  # в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
5  # Hint: типовая реализация занимает 2 строки
6  def gen_random(num_count, begin, end):
7      for i in range(num_count):
8          yield random.randint(begin, end)
9
10 if __name__ == '__main__':
11     a = gen_random(5, 1, 3)
12     for i in a:
13         print(i)

```

#### 4.3. Результат выполнения программы

```

PS C:\Users\Ия\OneDrive\Рабочий стол\lab3\lab_python_fp> python gen_random.py
2
3
1
1
2

```

### 5. Задача 3

#### 5.1. Условие

- 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.

#### 5.2. Текст программы

```

lab_python_fp > unique.py
1  from gen_random import *
2
3  class Unique(object):
4      def __init__(self, items, **kwargs):
5          self.used_elements = set()
6          self.case = False
7          for key in kwargs:
8              if key == "ignore_case":
9                  if kwargs[key] is True:
10                     self.case = True
11             self.data = iter(items)
12         def __next__(self):
13             while True:
14                 current = self.data.__next__()
15                 if self.case is True and type(current) is str:
16                     new_item = current.lower()
17                 else:
18                     new_item = current
19                 if new_item not in self.used_elements:
20                     self.used_elements.add(new_item)
21                 return new_item
22
23         def __iter__(self):
24             return self
25     if __name__ == '__main__':
26         data = ['c', 'a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
27         #data = gen_random(6, 3, 10)
28         for i in Unique(data, ignore_case=True):
29             print(i)

```

### 5.3. Результат выполнения программы

```

PS C:\Users\Ия\OneDrive\Рабочий стол\lab3\lab_python_fp> python unique.py
c
a
b

```

## 6. Задача 4

### 6.1. Условие

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`.

## 6.2. Текст программы

```
lab_python_fp > sort.py
1  from math import fabs
2  data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
3
4  if __name__ == '__main__':
5      result = sorted(data, reverse=True)
6      print(result)
7
8      result_with_lambda = sorted(data, key=lambda i: i, reverse = True)
9      print(result_with_lambda)
```

## 6.3. Результат выполнения программы

```
PS C:\Users\Ия\OneDrive\Рабочий стол\lab3\lab_python_fp> python sort.py
[123, 100, 4, 1, 0, -1, -4, -30, -100]
[123, 100, 4, 1, 0, -1, -4, -30, -100]
```


## 7. Задача 5

### 7.1. Условие

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.

### 7.2. Текст программы

lab\_python\_fp >  print\_result.py

```
1  def print_result(func_to_decorate):
2
3      def decorated_func(*arg):
4          print(func_to_decorate.__name__)
5          a = func_to_decorate(*arg)
6          if type(a) is list:
7              for i in a:
8                  print(i)
9          elif type(a) is dict:
10             for key in a:
11                 print("{} = {}".format(key, a[key]))
12             else:
13                 print(a)
14             return a
15         return decorated_func
16
17 @print_result
18 def test_1():
19     return 1
20
21
22 @print_result
23 def test_2():
24     return 'iu5'
25
26
27 @print_result
28 def test_3():
29     return {'a': 1, 'b': 2}
```

```
30
31
32 @print_result
33 def test_4():
34     return [1, 2]
35
36
37 if __name__ == '__main__':
38     print('!!!!!!!')
39     test_1()
40     test_2()
41     test_3()
42     test_4()
```

### 7.3. Результат выполнения программы

```
PS C:\Users\Ия\OneDrive\Рабочий стол\lab3\lab_python_fp> python print_result.py
!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

## 8. Задача 6

### 8.1. Условие

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

### 8.2. Текст программы

```
lab_python_fp > cm_timer.py
1  import time
2  from contextlib import contextmanager
3  class cm_time_1:
4      def __enter__(self):
5          self.time = time.time()
6
7      def __exit__(self, exp_type, exp_value, traceback):
8          if exp_type is not None:
9              print(exp_type, exp_value, traceback)
10         else:
11             print(time.time() - self.time)
12
13  @contextmanager
14  def cm_time_2():
15      timer = time.time()
16      yield
17      print(time.time() - timer)
18
19  if __name__ == '__main__':
20      with cm_time_1():
21          time.sleep(1)
22      with cm_time_2():
23          time.sleep(2)
```



### 8.3. Результат выполнения программы

```
PS C:\Users\Ия\OneDrive\Рабочий стол\lab3\lab_python_fp> python cm_timer.py
1.0101001262664795
2.0030691623687744
```

## 9. Задача 7

### 9.1. Условие

- 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 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.

## 9.2. Текст программы

```
lab_python_fp > process_data.py
1  import json
2  from unique import Unique
3  from print_result import print_result
4  from cm_timer import cm_time_1
5  from field import field
6  from gen_random import gen_random
7  import sys
8  # Сделаем другие необходимые импорт
9
10 path = "/Users/Ия/Downloads/data_light.json"
11
12 # Необходимо в переменную path сохранить путь к файлу, который был передан
13
14 with open(path) as f:
15     data = json.load(f)
16
17 # Далее необходимо реализовать все функции по заданию, заменив `raise NotI
18 # Предполагается, что функции f1, f2, f3 будут реализованы в одну строку
19 # В реализации функции f4 может быть до 3 строк
20
21 @print_result
22 def f1(arg):
23     return sorted(list(Unique(field(arg, 'job-name'), ignore_case=True)),
```

```
25
26 @print_result
27 def f2(arg):
28     return list(filter(lambda x: "программист" in x.lower(), arg))
29
30
31 @print_result
32 def f3(arg):
33     return list(map(lambda x: x + " с опытом Python", arg))
34
35
36 @print_result
37 def f4(arg):
38     return dict((zip(arg, gen_random(len(arg), 100000, 200000))))
39
40
41 if __name__ == '__main__':
42     with cm_time_1():
43         f4(f3(f2(f1(data))))
```