

Защищено:  
Папин А.В..

Демонстрация:  
Папин А.В..

"\_\_" \_\_\_\_\_ 2022 г.

"\_\_" \_\_\_\_\_ 2022 г.

## **Отчет по лабораторной работе №3-4 по курсу базовые компоненты интернет-технологий (БКИТ)**

**Тема работы: "Функциональные возможности языка Python"**

31

(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5Ц-54Б  
Папин Алексей

Гапанюк Ю.Е.

\_\_\_\_\_  
(подпись)

"\_\_" \_\_\_\_\_ 2022 г.

## СОДЕРЖАНИЕ ОТЧЕТА

1. Цель лабораторной работы.....	2
2. Описание задания.....	2
2.1. Задача 1 (файл field.py).....	2
2.2. Задача 2 (файл gen_random.py).....	3
2.3. Задача 3 (файл unique.py).....	3
2.4. Задача 4 (файл sort.py).....	5
2.5. Задача 5 (файл print_result.py).....	5
2.6. Задача 6 (файл cm_timer.py).....	6
2.7. Задача 7 (файл process_data.py).....	7
3. Листинг программы:.....	9
3.1. Lab_03.py.....	9
3.2. cm_timer.py.....	13
3.3. filed.py.....	13
3.4. gen_random.py.....	14
3.5. print_result.py.....	15
3.6. sort.py.....	17
3.7. unique.py.....	17
3.8. process_data.py.....	18
4. Результаты работы программы:.....	21
4.1. В IDE JetBrains PyCharm.....	21
4.1.1.Задание №1.....	21
4.1.2.Задание №2.....	21
4.1.3.Задание №3.....	22
4.1.4.Задание №4.....	23
4.1.5.Задание №5.....	24
4.1.6.Задание №6.....	24
4.1.7.Задание №7.....	25
4.2. Через cmd / powershell.....	26
4.2.1.Задание №1.....	26
4.2.2.Задание №2.....	27
4.2.3.Задание №3.....	27
4.2.4.Задание №4.....	28
4.2.5.Задание №5.....	28
4.2.6.Задание №6.....	29
4.2.7.Задание №7.....	29

## 1. Цель лабораторной работы

Изучение возможностей функционального программирования в языке Python.

## 2. 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.

### 2.1. Задача 1 (файл `field.py`)

Необходимо реализовать генератор `field`. Генератор `field` последовательно выдает значения ключей словаря. 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'}

- 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.

Шаблон для реализации генератора:

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

```
# {'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
```

## 2.2.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. Пример:

```
gen_random(5, 1, 3)
```

должен выдать 5 случайных чисел в диапазоне от 1 до 3, например

```
2, 2, 3, 2, 1
```

Шаблон для реализации генератора:

```
# Пример:
# gen_random(5, 1, 3) должен выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки
def gen_random(num_count, begin, end):
    pass
    # Необходимо реализовать генератор
```

## 2.3.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.

Пример:

```
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
```

`Unique(data)` будет последовательно возвращать только 1 и 2.

```
data = gen_random(10, 1, 3)
```

`Unique(data)` будет последовательно возвращать только 1, 2 и 3.

```
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
```

`Unique(data)` будет последовательно возвращать только a, A, b, B.

`Unique(data, ignore_case=True)` будет последовательно возвращать только a, b.

Шаблон для реализации класса-итератора:

```
# 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 одинаковыми
        строки в разном регистре
        # Например: ignore_case = True, Абв и АБВ - разные строки
        # ignore_case = False, Абв и АБВ - одинаковые строки, одна
        из которых удалится
        # По-умолчанию ignore_case = False
        pass

    def __next__(self):
        # Нужно реализовать __next__
        pass

    def __iter__(self):
        return self
```

## 2.4.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. Пример:

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

Необходимо решить задачу двумя способами:

- С использованием lambda-функции.
- Без использования lambda-функции.

Шаблон реализации:

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

if __name__ == '__main__':
    result = ...
    print(result)

    result_with_lambda = ...
    print(result_with_lambda)
```

## 2.5.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.

Шаблон реализации:

```
# Здесь должна быть реализация декоратора

@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()

```

Результат выполнения:

```

test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

```

## 2.6. 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. Пример:

```

with cm_timer_1():
    sleep(5.5)

```

После завершения блока кода в консоль должно вывестись **time: 5.5** (реальное время может несколько отличаться).

`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`).

## 2.7.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 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.

Шаблон реализации:

```
import json
import sys
# Сделаем другие необходимые импорты

path = None

# Необходимо в переменную path сохранить путь к файлу, который был передан
при запуске сценария

with open(path) as f:
    data = json.load(f)
```



```
# Далее необходимо реализовать все функции по заданию, заменив `raise  
NotImplemented`  
# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку  
# В реализации функции f4 может быть до 3 строк
```

```
@print_result  
def f1(arg):  
    raise NotImplemented
```

```
@print_result  
def f2(arg):  
    raise NotImplemented
```

```
@print_result  
def f3(arg):  
    raise NotImplemented
```

```
@print_result  
def f4(arg):  
    raise NotImplemented
```

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

### 3. Листинг программы:

#### 3.1.Lab\_03.py

```
import sys
from lab_python_fp.filed import *
from lab_python_fp.gen_random import *
from lab_python_fp.unique import *
from lab_python_fp.sort import *
from lab_python_fp.print_result import *
from lab_python_fp.cm_timer import *

def get_argv(index, prompt):
    try:
        # Получение значения из командной строки
        coef_str = sys.argv[index]
        print(index + 1, prompt, coef_str)
    except:
        # Вводим с клавиатуры
        print(index + 1, prompt, end='')
        coef_str = input()
    # По умолчанию строки
    return coef_str

def get_argv_value(index, prompt):
    try:
        # Получение значения из командной строки
        coef_str = sys.argv[index]
    except:
        # Вводим с клавиатуры
        print(prompt, end='')
        coef_str = input()
    # Переводим строку в целое число
    coef = int(coef_str)
    return coef

def into_tuple_from_str_in_value(str):
    tuple_buff = []
    str_buff = ''
    for i in range(len(str)):
        if (str[i] == ' '):
            tuple_buff.append(int(str_buff))
            str_buff = ''
        else:
            str_buff += str[i]

    tuple_buff.append(int(str_buff))

    return tuple_buff

def main():
    print('Введите номер пункта для выполнения задач')
    print('Задача №1 - field.py')
    print('Задача №2 - gen_random.py')
```

```

print('Задача №3 - unique.py')
print('Задача №4 - файл sort.py')
print('Задача №5 - print_result.py')
print('Задача №6 - cm_timer.py')
print('Задача №7 - process_data.py')

# switch = int(input('Введите номер пункта: '))
switch = get_argv_value(1, 'Введите номер пункта: ')

if(switch == 1):
    print('Задача №1 - field.py')
    mas = ''
    if (len(sys.argv) == 1):
        count_argv = int(input('Введите кол-во желаемых аргументов: '))
        if(count_argv > 1):
            for i in range(0, count_argv):
                # print('{}-ый аргумент'.format(i + 1))
                mas += (get_argv(i, '-ый аргумент: '))
                # print('{}-ый аргумент: {}'.format(i + 1, mas[i]))
                mas += ' '
            print(field(goods, mas))
        else:
            print('Ошибка введения кол-во аргументов!')
    elif(len(sys.argv) > 1):
        for i in range(0, len(sys.argv)):
            # print('{}-ый аргумент'.format(i + 1))
            mas += (get_argv(i, '-ый аргумент: '))
            # print('{}-ый аргумент: {}'.format(i + 1, mas[i]))
            mas += ' '
        print(field(goods, mas))
    else:
        print('Ошибка введения аргументов!')

elif(switch == 2):
    print('Задача №2 - gen_random.py')
    print('Генерация случайных чисел:')
    size = get_argv_value(2, 'Введите кол-во: ')
    if(size < 1):
        print('Ошибка! Разер больше 0 должен быть')
    else:
        value = gen_random(size, get_argv_value(3, 'Введите диапазон от: '), get_argv_value(4, 'Введите диапазон до: '))
        print(value)

elif(switch == 3):
    print('Задача №3 - unique.py')
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print(data)
    a = Unique(data)
    for i in Unique(a):
        print(i, end=' ')
    print()

    data1 = gen_random(10, 1, 3)
    print(data1)

```

```

b = Unique(data1)
for i in Unique(b):
    print(i, end=' ')
print()

data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
print(data2)
c = Unique(data2)
for i in Unique(c):
    print(i, end=' ')
print()

d = Unique(data2, ignore_case=True)
for i in Unique(d):
    print(i, end=' ')
print()

if (len(sys.argv) == 1):
    data_input = into_tuple_from_str(input('Введите любые значения в
списке (между значениями ставьте пробелом)\n'))

    print(data_input)
    c = Unique(data_input)
    print('С чувствительным регистром')
    for i in Unique(c):
        print(i, end=' ')
    print()

    print(data_input)
    c = Unique(data_input, ignore_case=True)
    print('Без чувствительного регистра')
    for i in Unique(c):
        print(i, end=' ')
    print()
elif (len(sys.argv) > 1):
    buff = sys.argv[2]
    for i in range(2, len(sys.argv)):
        buff = buff + ' ' + sys.argv[i]

    data3 = into_tuple_from_str(buff)

    print(data3)
    c = Unique(data3)
    print('С чувствительным регистром')
    for i in Unique(c):
        print(i, end=' ')
    print()

    print(data3)
    c = Unique(data3, ignore_case=True)
    print('Без чувствительного регистра')
    for i in Unique(c):
        print(i, end=' ')
    print()
else:

```

```

        print('Ошибка введения аргументов!')

    elif(switch == 4):
        print('Задача №4 - sort.py')
        exercise_4_sort()

    if (len(sys.argv) == 1):
        data_input = into_tuple_from_str_in_value(input('Введите любые значения в списке (между значениями ставьте пробелом)\n'))

        print(f'Исходный список:\n {data_input}')

        result_with_lambda = sorted(data_input, key=lambda i: -abs(i))
        print(f'Отсортированный список с применением lambda-функции:\n {result_with_lambda}')

        result = sorted(data_input, key=abs, reverse=True)
        print(f'Отсортированный список без применения lambda-функции:\n {result}')
    elif (len(sys.argv) > 1):
        buff = sys.argv[2]
        for i in range(3, len(sys.argv)):
            buff = buff + ' ' + sys.argv[i]

        data_argv = into_tuple_from_str_in_value(buff)

        print(f'Исходный список:\n {data_argv}')

        result_with_lambda = sorted(data_argv, key=lambda i: -abs(i))
        print(f'Отсортированный список с применением lambda-функции:\n {result_with_lambda}')

        result = sorted(data_argv, key=abs, reverse=True)
        print(f'Отсортированный список без применения lambda-функции:\n {result}')
    else:
        print('Ошибка введения аргументов!')

    elif(switch == 5):
        print('Задача №5 - print_result.py')
        exercise_5_print_result()

    elif(switch == 6):
        print('Задача №6 - cm_timer.py')
        exercise_6_cm_timer()

    else:
        print('Нет такого пункта')

if __name__ == '__main__':
    main()

```

### 3.2.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.  
Пример:

```
with cm_timer_1():  
    sleep(5.5)
```

После завершения блока кода в консоль должно вывестись time: 5.5  
(реальное время может несколько отличаться).

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).  
'''

```
from time import time, sleep  
from contextlib import contextmanager
```

# С использованием класса

```
class cm_timer_1:  
    def __init__(self):  
        self._start = 0  
        self._end = 0  
  
    def __enter__(self):  
        self._start = time()  
  
    def __exit__(self, the_type, the_value, the_backing):  
        self._end = time()  
        print(f'Time of work: {self._end - self._start}')
```

# С использованием библиотеки contextlib

```
@contextmanager  
def cm_timer_2():  
    start_time = time()  
    yield None  
    end_time = time()  
    print(f'Time of work: {end_time - start_time}')
```

```
def exercise_6_cm_timer():  
    with cm_timer_1():  
        sleep(5.5)  
  
    with cm_timer_2():  
        sleep(5.5)
```

### 3.3.filed.py

```
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}

# items = goods, *args = 'title' => len(args) = 1;
# items = goods, *args = ('title', 'price') => len(args) = 2;

def field(items, *args):
    try:
        # Преобразование в кортеж из строки
        argv = into_tuple_from_str(*args)
        assert len(argv) > 0, 'Ошибка! Отсутствуют аргументы!\nПримечание
аргументы не должны быть пустыми!'
        # assert len(args) > 0, 'Ошибка! Отсутствуют аргументы!\nПримечание
аргументы не должны быть пустыми!'
        r = [{ } for i in range(len(items))]
        for i in range(len(items)):
            for j in items[i]:
                #if j in argv:
                if j in argv:
                    r[i].update({j: items[i][j]})
            return r
    except:
        print('Ошибка! Отсутствует список в качестве переданного аргумента!')

# Преобразование в строку из кортежа
def into_tuple_from_str(str):
    tuple_buff = []
    str_buff = ''
    for i in range(len(str)):
        if (str[i] == ' '):
            tuple_buff.append(str_buff)
            str_buff = ''
        else:
            str_buff += str[i]

    tuple_buff.append(str_buff)

    return tuple_buff

```

### 3.4.gen\_random.py

```

# Пример:
# gen_random(5, 1, 3) должен выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки
import random

def gen_random(num_count, begin, end):
    arr = []
    for i in range(0, num_count):
        arr.append(int(random.uniform(begin, end)))
    return arr

```

### 3.5.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.

Шаблон реализации:

Результат выполнения:
test_1
1
test_2
iv5
test_3
a = 1
b = 2
test_4
1
2
'''

# Здесь должна быть реализация декоратора

# Синтаксис для обертывания функции в декоратор

def print_result(function):
    def control(arr=[], *args, **kwargs):
        # печатает название вызываемой функции
        print(function.__name__)
        if len(arr) == 0:
            result = function(*args, **kwargs)
        else:
            result = function(arr, *args, **kwargs)

        '''if type(result) is not int and type(result) is not float:
            if type(result[0]) is tuple:
                for arr_item_index in range(len(result)):
                    word = ''
                    for item in result[arr_item_index]:
                        word = word + ' ' + item
                    word = word.replace(' ', '', 1)
                    result[arr_item_index] = word'''

        if type(result) == int or type(result) == str:
            print(result)
        elif type(result) is list:
```



```

'''
Можно заменить функцию MAP на это
result = []
    for a in iterable_a:
        for b in iterable_b:
            result.append((a, b))
join - принимает итеративные объекты, последовательно
присоединяет и возвращает их в виде строки.

Итог:
С применением join и map можно заменить на это, но будет длинным
кодом:

flexiple = ["Hire", "the", "top", 10, "python","freelancers"]

f1 = ""

for i in flexiple:
    f1 += str(i)+ " "

print(f1)
'''

print('\n'.join(map(str, result)))
elif type(result) is dict:
    for key, el in result.items():
        print(f'{key} = {el}')
# Функция zip () в Python создает итератор, который объединяет
элементы из нескольких источников данных.
''' >>> list (range(3))
[0, 1, 2]
>>> list(zip(range(3), 'abc'))
[(0, 'a'), (1, 'b'), (2, 'c')] '''
elif type(result) == zip:
    for name, number in result:
        print(name, number)
else:
    print(result)
return result
return control

@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]

def exercise_5_print_result():
    test_1()
    test_2()
    test_3()
    test_4()

```

### 3.6.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. Пример:

```

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

```

Необходимо решить задачу двумя способами:

С использованием lambda-функции.

Без использования lambda-функции.

Шаблон реализации:

```

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

def exercise_4_sort():
    print(f'Исходный список:\n {data}')
    result_with_lambda = sorted(data, key=lambda i: -abs(i))
    print(f'Отсортированный список с применением lambda-функции:\n {result_with_lambda}')

    result = sorted(data, key=abs, reverse = True)
    print(f'Отсортированный список без применения lambda-функции:\n {result}')
'''

```

### 3.7.unique.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
        # одинаковыми строки в разном регистре
        # Например: ignore_case = True, Абв и АБВ - разные строки
        #             ignore_case = False, Абв и АБВ - одинаковые строки,
        #             одна из которых удалится
        # По-умолчанию ignore_case = False

```

```

self.arr = []

# используя кортежи, получаем ключ и значения
for key, value in kwargs.items():
    # если ключ пустой и значение ИСТИНА, то
    if key == 'ignore_case' and value == True:
        # в текущем списке все символы преобразуем в нижний регистр
        # через функции lower
        items = [i.lower() for i in items]

    for index in items:
        # Если текущее значение с списка item не совпадает / не
        # существует в созданном списке arr
        if index not in self.arr:
            # то присвоим несуществующее значение в созданном списке arr
            self.arr.append(index)
        pass

def __next__(self):
    try:
        x = self.arr[self.begin]
        self.begin += 1
        return x
    except:
        raise StopIteration

def __iter__(self):
    self.begin = 0
    return self

```

### 3.8.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 sm_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.

Шаблон реализации:

```
'''
```

```
import json
```

```
# Сделаем другие необходимые импорты
```

```
from operator import concat
```

```
from lab_python_fp.field import field
```

```
from lab_python_fp.unique import Unique
```

```
from lab_python_fp.sort import exercise_4_sort
```

```
from lab_python_fp.print_result import print_result
```

```
from lab_python_fp.cm_timer import cm_timer_1
```

```
from lab_python_fp.gen_random import gen_random
```

```
path = 'data_light.json'
```

```
# Необходимо в переменную path сохранить путь к файлу, который был передан  
# при запуске сценария
```

```
# Преобразуем в UTF-8 кодировку, иначе программа неправильно прочтет файл
```

```
with open(path, 'r', encoding='UTF-8') as f:
```

```
    data = json.load(f)
```

```
    # print(data)
```

```
# Далее необходимо реализовать все функции по заданию, заменив `raise  
NotImplemented`
```

```
# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку
```

```
# В реализации функции f4 может быть до 3 строк
```

```
@print_result
```

```
def f1(arg):
```

```
    # Подбираем названия работы, которые не повторяют друг друга, в список
```

```
    # info_job_name = Unique([i['job-name'] for i in field(data, 'job-  
name')], ignore_case=True)
```

```
    # Отсортируем
```

```
    # info_job_name_sorted = sorted(info_job_name, key=str, reverse = False)
```

```
    # return info_job_name.arr.sort()
```

```
    # return sorted(info_job_name, key=str, reverse = False)
```

```
    # return sorted(list(set([el['job-name'] for el in arg])), key=lambda a:
```

```

a.lower())
    return list(Unique([i['job-name'] for i in field(data, 'job-name')],
ignore_case=True))

@print_result
def f2(arg):
    return list(filter(lambda i: i.startswith('программист'), arg))

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

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

if __name__ == '__main__':
    with cm_timer_1():
        # ex_1 = f1(data)
        # ex_2 = f2(f1(data))
        # ex_3 = f3(f2(f1(data)))
        ex_4 = (f4(f3(f2(f1(data)))))
        # (f4(f3(f2(f1(data)))))
        print()

```

## 4. Результаты работы программы:

### 4.1.В IDE JetBrains PyCharm

#### 4.1.1. Задание №1

```
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - sm_timer.py
Введите номер пункта: 1
Задача №1 - field.py
Введите кол-во желаемых аргументов: 3
1 -ый аргумент: D:\Python\BKIT\Lab_03.py
2 -ый аргумент: title
3 -ый аргумент: color
[{'title': 'Ковер', 'color': 'green'}, {'title': 'Диван для отдыха', 'color': 'black'}]

Process finished with exit code 0
```

#### 4.1.2. Задание №2

```
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - sm_timer.py
Введите номер пункта: 2
Задача №2 - gen_random.py
Генерация случайных чисел:
Введите кол-во: 10
Введите диапазон от: -6
Введите диапазон до: 5
[-2, 0, 3, -3, 4, -4, -3, 1, 4, -4]

Process finished with exit code 0
```

### 4.1.3. Задание №3

```
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - sm_timer.py
Введите номер пункта: 3
Задача №3 - unique.py
[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
1 2
[1, 1, 2, 1, 1, 1, 1, 1, 2, 1]
1 2
['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
a A b B
a b
Введите любые значения в списке (между значениями ставьте пробелом)
a A b B b b b b U u U 1 2 3 1 1 1 1
['a', 'A', 'b', 'B', 'b', 'b', 'b', 'b', 'U', 'u', 'U', '1', '2', '3', '1', '1', '1', '1']
С чувствительным регистром
a A b B U u 1 2 3
['a', 'A', 'b', 'B', 'b', 'b', 'b', 'b', 'U', 'u', 'U', '1', '2', '3', '1', '1', '1', '1']
Без чувствительного регистра
a b u 1 2 3

Process finished with exit code 0
```

#### 4.1.4. Задание №4

```
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - sm_timer.py
Введите номер пункта: 4
Задача №4 - sort.py
Исходный список:
[4, -30, 100, -100, 123, 1, 0, -1, -4]
Отсортированный список с применением lambda-функции:
[123, 100, -100, -30, 4, -4, 1, -1, 0]
Отсортированный список без применения lambda-функции:
[123, 100, -100, -30, 4, -4, 1, -1, 0]
Введите любые значения в списке (между значениями ставьте пробелом)
14 6 245 65 6 1 0 -5 34 6
Исходный список:
[14, 6, 245, 65, 6, 1, 0, -5, 34, 6]
Отсортированный список с применением lambda-функции:
[245, 65, 34, 14, 6, 6, 6, -5, 1, 0]
Отсортированный список без применения lambda-функции:
[245, 65, 34, 14, 6, 6, 6, -5, 1, 0]

Process finished with exit code 0
|
```



#### 4.1.5. Задание №5

```
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - cm_timer.py
Введите номер пункта: 5
Задача №5 - print_result.py
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

#### 4.1.6. Задание №6

```
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - cm_timer.py
Введите номер пункта: 6
Задача №6 - cm_timer.py
Time of work: 5.508674144744873
Time of work: 5.514217853546143

Process finished with exit code 0
```

#### 4.1.7. Задание №7

f1

администратор на телефоне

медицинская сестра

охранник сутки-день-ночь-вахта

врач анестезиолог реаниматолог

теплотехник

разнорабочий

электро-газосварщик

водитель gett/гетт и yandex/яндекс такси на личном автомобиле

монолитные работы

организатор – тренер

помощник руководителя

автоэлектрик

врач ультразвуковой диагностики в детскую поликлинику

менеджер по продажам ит услуг (b2b)

менеджер по персоналу

аналитик

воспитатель группы продленного дня

инженер по качеству

инженер по качеству 2 категории (класса)

водитель автомобиля

пекарь

.....

портной

специалист отдела аренды

инженер-механик

разработчик импульсных источников питания

механик по эксплуатации транспортного отдела

инженер-технолог по покраске

бетонщик - арматурщик

главный инженер финансово-экономического отдела

секретарь судебного заседания в аппарате мирового судьи железнодорожного судебного района города ростова-на-дону

варщик зефира

варщик мармеладных изделий

оператор склада

специалист по электромеханическим испытаниям аппаратуры бортовых космических систем

заведующий музеем в д.копорье

документовед

специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических систем

менеджер (в промышленности)

f2

программист

программист c++/c#/java

программист 1с

программист-разработчик информационных систем

программист c++

программист/ junior developer

программист / senior developer

программист/ технический специалист

программист c#

```

f3
программист с опытом Python
программист с++/с#/java с опытом Python
программист 1с с опытом Python
программист-разработчик информационных систем с опытом Python
программист с++ с опытом Python
программист/ junior developer с опытом Python
программист / senior developer с опытом Python
программист/ технический специалист с опытом Python
программист с# с опытом Python
f4
('программист с опытом Python', 'зарплата 127639 руб.')
('программист с++/с#/java с опытом Python', 'зарплата 158596 руб.')
('программист 1с с опытом Python', 'зарплата 167847 руб.')
('программист-разработчик информационных систем с опытом Python', 'зарплата 185861 руб.')
('программист с++ с опытом Python', 'зарплата 149621 руб.')
('программист/ junior developer с опытом Python', 'зарплата 159634 руб.')
('программист / senior developer с опытом Python', 'зарплата 124322 руб.')
('программист/ технический специалист с опытом Python', 'зарплата 151554 руб.')
('программист с# с опытом Python', 'зарплата 197510 руб.')

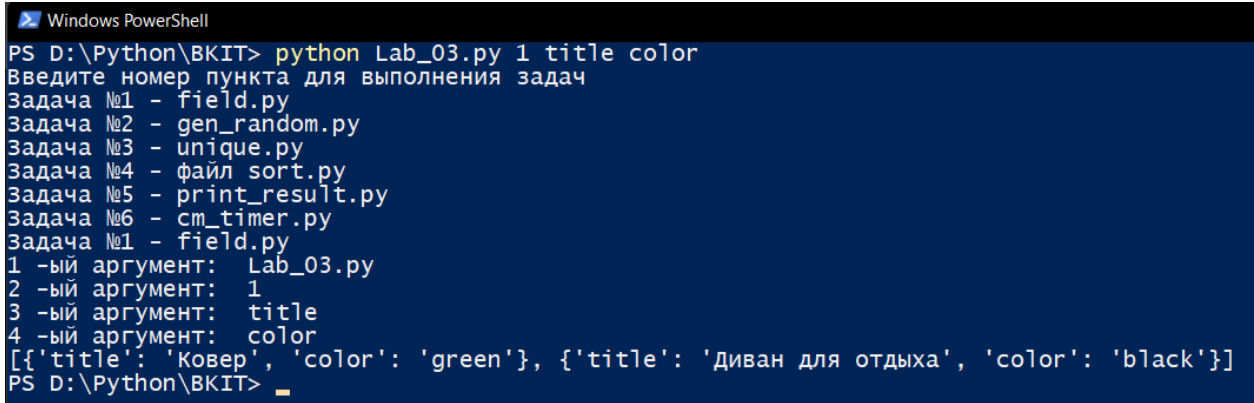
Time of work: 0.04100990295410156

Process finished with exit code 0
|

```

## 4.2. Через cmd / powershell

### 4.2.1. Задание №1



```

Windows PowerShell
PS D:\Python\BKIT> python Lab_03.py 1 title color
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - cm_timer.py
Задача №1 - field.py
1 -ый аргумент: Lab_03.py
2 -ый аргумент: 1
3 -ый аргумент: title
4 -ый аргумент: color
[{'title': 'Ковер', 'color': 'green'}, {'title': 'Диван для отдыха', 'color': 'black'}]
PS D:\Python\BKIT>

```

#### 4.2.2. Задание №2

```
Windows PowerShell
PS D:\Python\BKIT> python Lab_03.py 2 10 -5 5
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - cm_timer.py
Задача №2 - gen_random.py
Генерация случайных чисел:
[-2, 2, -3, 3, -2, 1, -4, -2, 0, -2]
PS D:\Python\BKIT>
```

#### 4.2.3. Задание №3

```
Windows PowerShell
PS D:\Python\BKIT> python Lab_03.py 3 A b B a A 1 2 3 1 1 1 1 1 A
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - cm_timer.py
Задача №3 - unique.py
[1, 1, 1, 1, 1, 2, 2, 2, 2]
1 2
[2, 1, 2, 2, 1, 1, 2, 1, 1, 1]
2 1
['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
a A b B
a b
['A', 'A', 'b', 'B', 'B', 'a', 'A', '1', '2', '3', '1', '1', '1', '1', '1', '1', 'A']
С чувствительным регистром
A b B a 1 2 3
['A', 'A', 'b', 'B', 'B', 'a', 'A', '1', '2', '3', '1', '1', '1', '1', '1', '1', 'A']
Без чувствительного регистра
a b 1 2 3
PS D:\Python\BKIT>
```

#### 4.2.4. Задание №4

```
Windows PowerShell
PS D:\Python\BKIT> python Lab_03.py 4 10 6 9 1 84 95 -4 15 7 4
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - cm_timer.py
Задача №4 - sort.py
Исходный список:
[4, -30, 100, -100, 123, 1, 0, -1, -4]
Отсортированный список с применением lambda-функции:
[123, 100, -100, -30, 4, -4, 1, -1, 0]
Отсортированный список без применения lambda-функции:
[123, 100, -100, -30, 4, -4, 1, -1, 0]
Исходный список:
[10, 6, 9, 1, 84, 95, -4, 15, 7, 4]
Отсортированный список с применением lambda-функции:
[95, 84, 15, 10, 9, 7, 6, -4, 4, 1]
Отсортированный список без применения lambda-функции:
[95, 84, 15, 10, 9, 7, 6, -4, 4, 1]
PS D:\Python\BKIT> _
```

#### 4.2.5. Задание №5

```
Windows PowerShell
PS D:\Python\BKIT> python Lab_03.py 5
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - cm_timer.py
Задача №5 - print_result.py
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
PS D:\Python\BKIT> _
```

#### 4.2.6. Задание №6

```
Windows PowerShell
PS D:\Python\BKIT> python Lab_03.py 6
Введите номер пункта для выполнения задач
Задача №1 - field.py
Задача №2 - gen_random.py
Задача №3 - unique.py
Задача №4 - файл sort.py
Задача №5 - print_result.py
Задача №6 - cm_timer.py
Задача №6 - cm_timer.py
Time of work: 5.508516073226929
Time of work: 5.51334810256958
PS D:\Python\BKIT>
```

#### 4.2.7. Задание №7

```
Windows PowerShell
PS D:\Python\BKIT> python process_data.py
f1
администратор на телефоне
медицинская сестра
охранник сутки-день-ночь-вахта
врач анестезиолог реаниматолог
теплотехник
разнорабочий
электро-газосварщик
водитель gett/гетт и yandex/яндекс такси на личном автомобиле
монолитные работы
организатор – тренер
помощник руководителя
автоэлектрик
врач ультразвуковой диагностики в детскую поликлинику
менеджер по продажам ит услуг (b2b)
менеджер по персоналу
```

.....

```
бетонщик – арматурщик
главный инженер финансово-экономического отдела
секретарь судебного заседания в аппарате мирового судьи железнодорожного судебного района города ростова-на-дону
варщик зефира
варщик мармеладных изделий
оператор склада
специалист по электромеханическим испытаниям аппаратуры бортовых космических систем
заведующий музеем в д.копорье
документовед
специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических систем
менеджер (в промышленности)
f2
программист
программист c++/c#/java
программист 1с
программист-разработчик информационных систем
программист c++
программист/ junior developer
программист / senior developer
программист/ технический специалист
программист c#
```

```
программист с/
f3
программист с опытом Python
программист с++/с#/java с опытом Python
программист 1с с опытом Python
программист-разработчик информационных систем с опытом Python
программист с++ с опытом Python
программист/ junior developer с опытом Python
программист / senior developer с опытом Python
программист/ технический специалист с опытом Python
программист с# с опытом Python
f4
('программист с опытом Python', 'зарплата 125347 руб.')
('программист с++/с#/java с опытом Python', 'зарплата 199084 руб.')
('программист 1с с опытом Python', 'зарплата 175167 руб.')
('программист-разработчик информационных систем с опытом Python', 'зарплата 199230 руб.')
('программист с++ с опытом Python', 'зарплата 105428 руб.')
('программист/ junior developer с опытом Python', 'зарплата 157566 руб.')
('программист / senior developer с опытом Python', 'зарплата 154019 руб.')
('программист/ технический специалист с опытом Python', 'зарплата 114503 руб.')
('программист с# с опытом Python', 'зарплата 179465 руб.')
Time of work: 0.07500290870666504
```