|  |  |  |
| --- | --- | --- |
| Защищено:  Папин А.В..    "\_\_" \_\_\_\_\_\_\_\_\_\_\_2022 г. |  | Демонстрация:  Папин А.В..    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_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 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Гапанюк Ю.Е. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. |

Москва, МГТУ - 2022

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ ОТЧЕТА**

[1. Цель лабораторной работы 2](#_Toc120621285)

[2. Описание задания. 2](#_Toc120621286)

[2.1. Задача 1 (файл field.py) 2](#_Toc120621287)

[2.2. Задача 2 (файл gen\_random.py) 3](#_Toc120621288)

[2.3. Задача 3 (файл unique.py) 3](#_Toc120621289)

[2.4. Задача 4 (файл sort.py) 5](#_Toc120621290)

[2.5. Задача 5 (файл print\_result.py) 5](#_Toc120621291)

[2.6. Задача 6 (файл cm\_timer.py) 6](#_Toc120621292)

[2.7. Задача 7 (файл process\_data.py) 7](#_Toc120621293)

[3. Листинг программы: 9](#_Toc120621294)

[3.1. Lab\_03.py 9](#_Toc120621295)

[3.2. cm\_timer.py 13](#_Toc120621296)

[3.3. filed.py 13](#_Toc120621297)

[3.4. gen\_random.py 14](#_Toc120621298)

[3.5. print\_result.py 15](#_Toc120621299)

[3.6. sort.py 17](#_Toc120621300)

[3.7. unique.py 17](#_Toc120621301)

[3.8. process\_data.py 18](#_Toc120621302)

[4. Результаты работы программы: 21](#_Toc120621303)

[4.1. В IDE JetBrains PyCharm 21](#_Toc120621304)

[4.1.1.Задание №1 21](#_Toc120621305)

[4.1.2.Задание №2 21](#_Toc120621306)

[4.1.3.Задание №3 22](#_Toc120621307)

[4.1.4.Задание №4 23](#_Toc120621308)

[4.1.5.Задание №5 24](#_Toc120621309)

[4.1.6.Задание №6 24](#_Toc120621310)

[4.1.7.Задание №7 25](#_Toc120621311)

[4.2. Через cmd / powershell 26](#_Toc120621312)

[4.2.1.Задание №1 26](#_Toc120621313)

[4.2.2.Задание №2 27](#_Toc120621314)

[4.2.3.Задание №3 27](#_Toc120621315)

[4.2.4.Задание №4 28](#_Toc120621316)

[4.2.5.Задание №5 28](#_Toc120621317)

[4.2.6.Задание №6 29](#_Toc120621318)

[4.2.7.Задание №7 29](#_Toc120621319)

# **Цель лабораторной работы**

Изучение возможностей функционального программирования в языке Python.

# **Описание задания.**

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fp. 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. 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', '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**

# **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 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**

# **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, Aбв и АБВ - разные строки**

**# ignore\_case = False, Aбв 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**

# **Задача 4 (файл sort.py)**

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. 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)**

# **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 выводить 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**

# **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).

# **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))))**

# **Листинг программы:**

# **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-фнукции:\n {result\_with\_lambda}')  
  
 result = sorted(data\_input, key=abs, reverse=True)  
 print(f'Отсортированный список без применении 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-фнукции:\n {result\_with\_lambda}')  
  
 result = sorted(data\_argv, key=abs, reverse=True)  
 print(f'Отсортированный список без применении 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()

# **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 \_\_int\_\_(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 библитоеки 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)

# **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

# **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

# **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 выводить в столбик через знак равенства.  
  
Шаблон реализации:  
  
Результат выполнения:  
test\_1  
1  
test\_2  
iu5  
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()

# **sort.py**

*'''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.  
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-фнукции:\n {result\_with\_lambda}')  
  
 result = sorted(data, key=abs, reverse = True)  
 print(f'Отсортированный список без применении lambda-функции:\n {result}')

# **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, Aбв и АБВ - разные строки  
 # ignore\_case = False, Aбв 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 присвоем 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

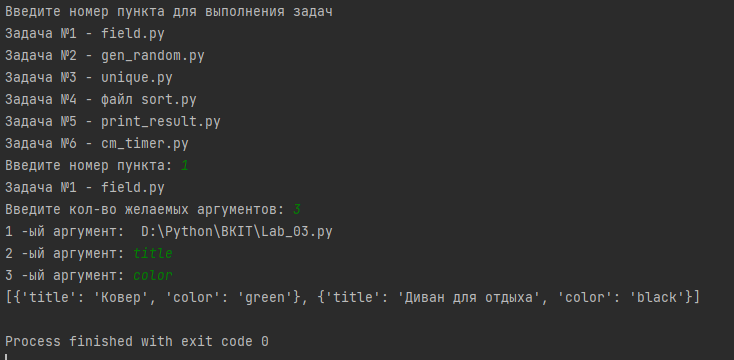
# **process\_data.py**

*'''  
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  
# Сделаем другие необходимые импорты  
from operator import concat  
from lab\_python\_fp.filed 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, ' c 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()

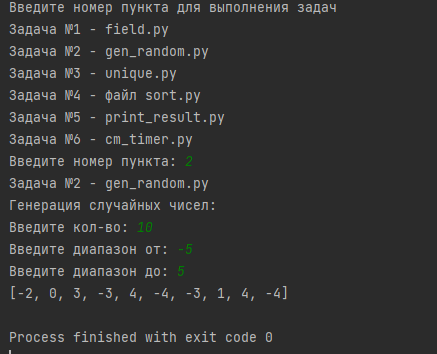
# **Результаты работы программы:**

# **В IDE JetBrains PyCharm**

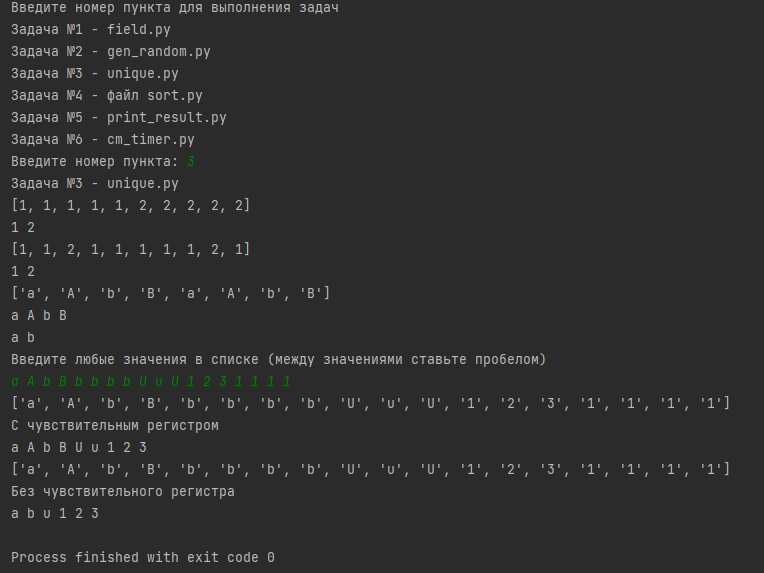
# **Задание №1**



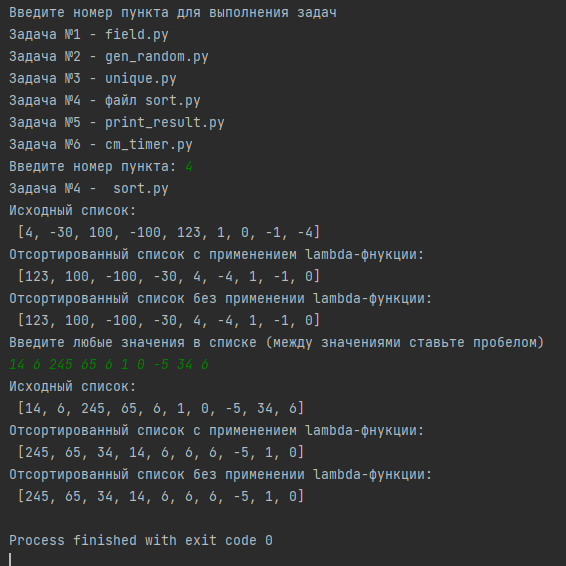
# **Задание №2**



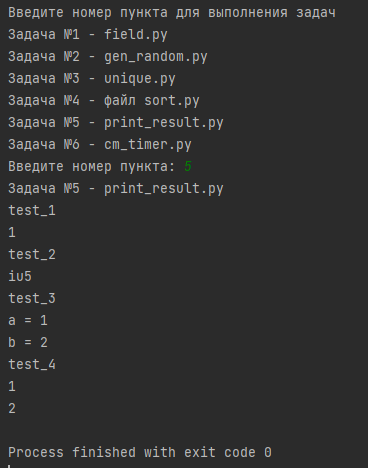
# **Задание №3**



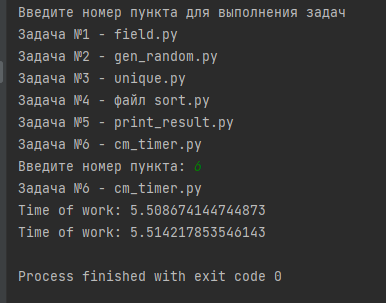
# **Задание №4**



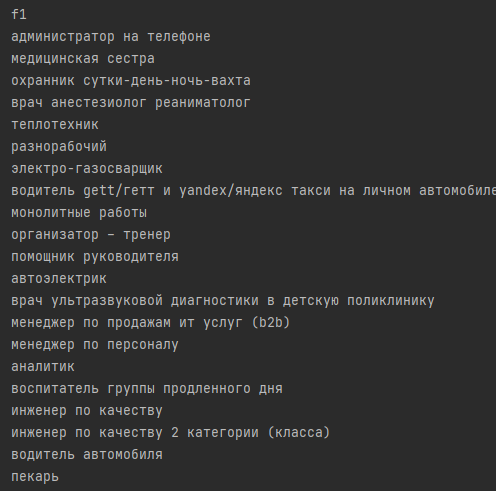
# **Задание №5**



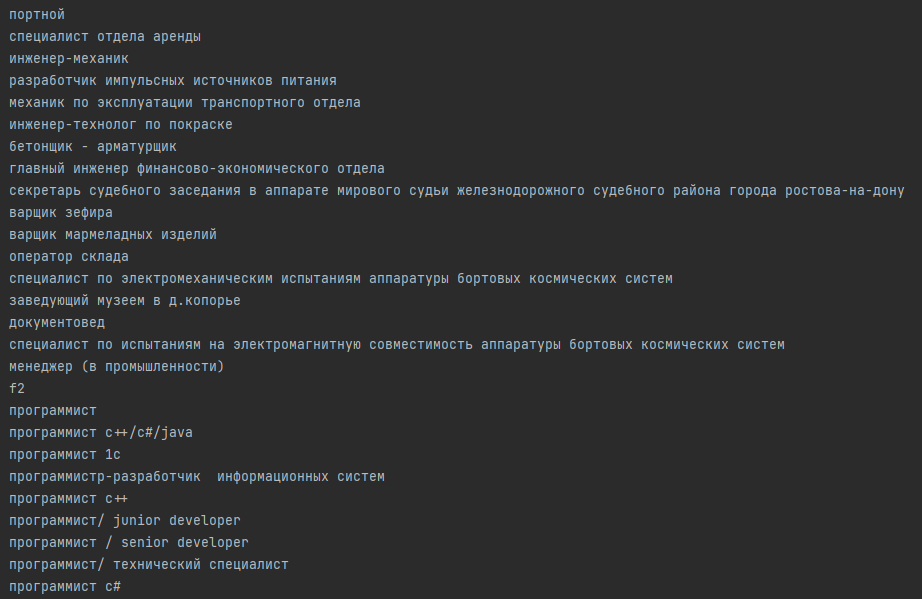
# **Задание №6**

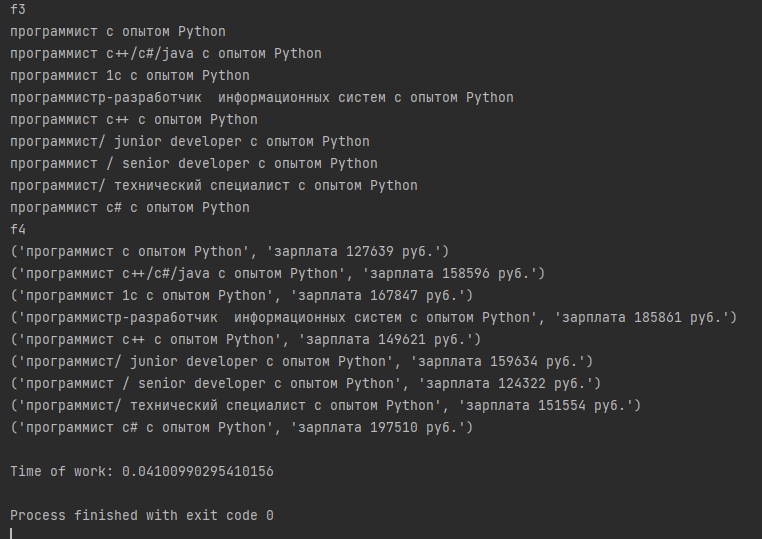


# **Задание №7**



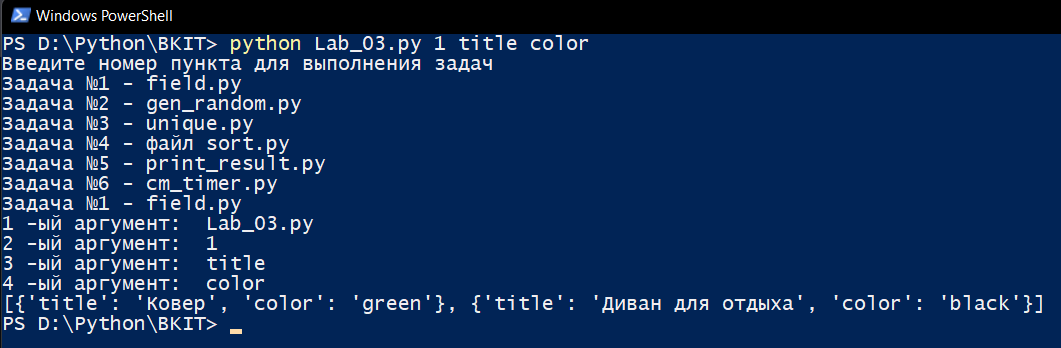
**…………………………………..**



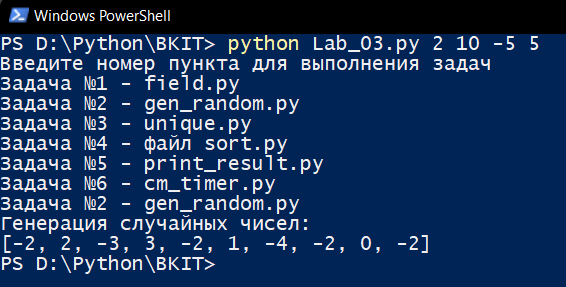


# **Через cmd / powershell**

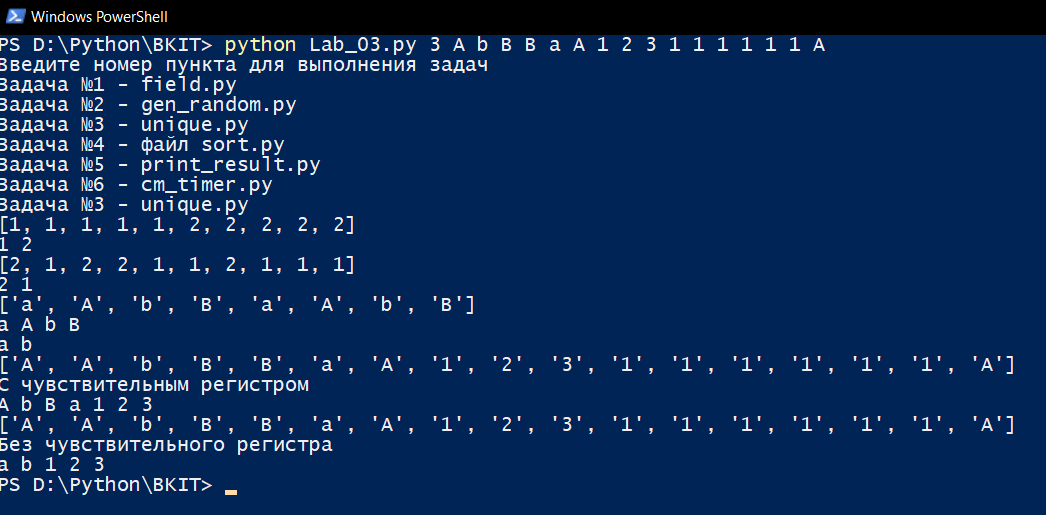
# **Задание №1**



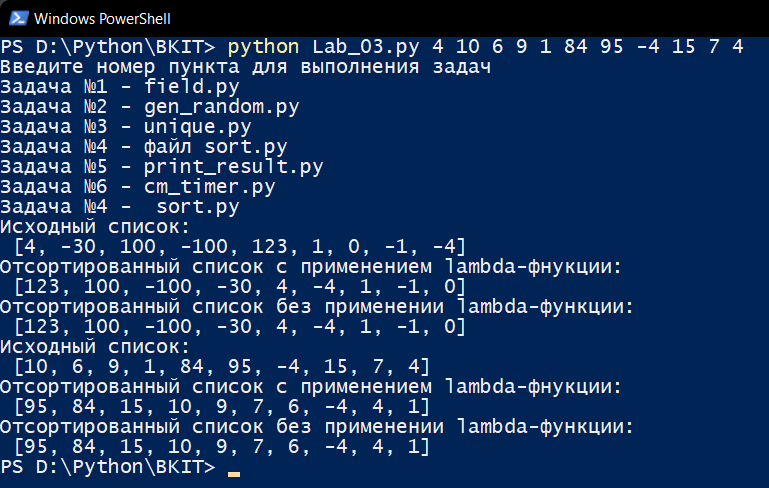
# **Задание №2**



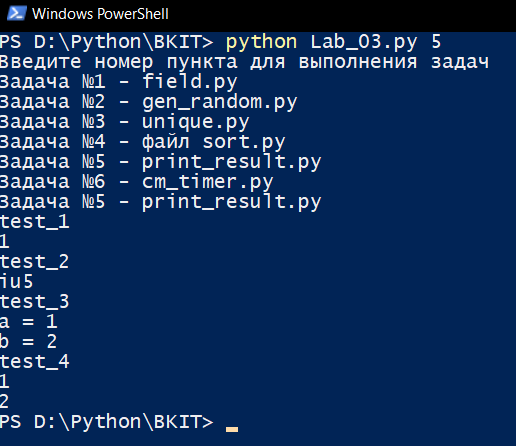
# **Задание №3**



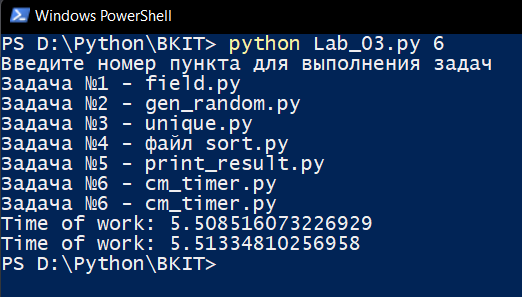
# **Задание №4**



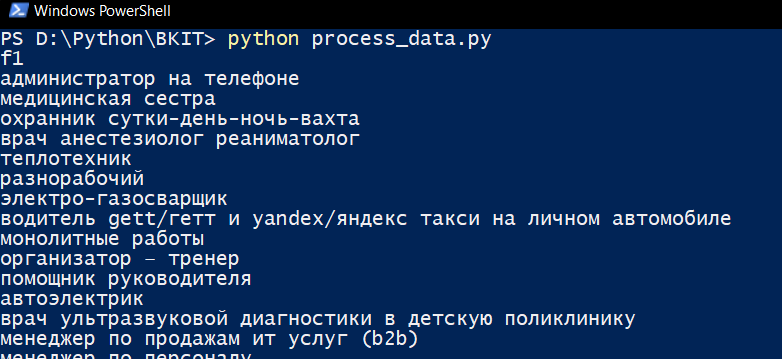
# **Задание №5**



# **Задание №6**



# **Задание №7**



**…………………………………..**

