|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ**

**Отчет по лабораторным работам №3-4**

**по дисциплине Базовые компоненты интернет технологии**

#### Тема работы: "[Функциональные возможности языка Python](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp)"

**Выполнил:**

Студент группы ИУ5Ц-54Б

Перевощиков Н.Д. 23.11.22 г.

(дата, подпись)

**Проверил:**

Преподаватель

Канев А.И.

(дата, подпись)

Москва, 2022

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

**СОДЕРЖАНИЕ ОТЧЕТА**

[1. Цель лабораторной работы 4](#_Toc121153275)

[2. Общее описание задания 4](#_Toc121153276)

[Задача 1 (файл field.py) 4](#_Toc121153277)

[Задача 2 (файл gen\_random.py) 5](#_Toc121153278)

[Задача 3 (файл unique.py) 5](#_Toc121153279)

[Задача 4 (файл sort.py) 6](#_Toc121153280)

[Задача 5 (файл print\_result.py) 7](#_Toc121153281)

[Задача 6 (файл cm\_timer.py) 8](#_Toc121153282)

[Задача 7 (файл process\_data.py) 8](#_Toc121153283)

[3. Описание каждой задачи 11](#_Toc121153284)

[3.1. Основная программа 11](#_Toc121153285)

[3.2. Задача №1 (файл filed.py) 14](#_Toc121153286)

[3.2.1. Текст программы 15](#_Toc121153287)

[3.2.2. Результат Задача №1 (Pycharm) 16](#_Toc121153288)

[3.2.3. Результат Задача № 1 (Командная строка) 16](#_Toc121153289)

[3.3. Задача №2 (файл gen\_random.py) 16](#_Toc121153290)

[3.3.1. Текст программы 16](#_Toc121153291)

[3.3.2. Результат Задача № 2 (Pycharm) 17](#_Toc121153292)

[3.3.3. Результат Задача № 2 (Командная строка) 17](#_Toc121153293)

[3.4. Задача №3 (файл unigue.py) 17](#_Toc121153294)

[3.4.1. Текст программы 18](#_Toc121153295)

[3.4.2. Результат Задача № 3 (Pycharm) 19](#_Toc121153296)

[3.4.3. Результат Задача № 3 (Командная строка) 20](#_Toc121153297)

[3.5. Задача №4 (файл sort.py) 20](#_Toc121153298)

[3.5.1. Текст программы 20](#_Toc121153299)

[3.5.2. Результат Задача № 4 (Pycharm) 21](#_Toc121153300)

[3.5.3. Результат Задача №4 (Командная строка) 21](#_Toc121153301)

[3.6. Задача №5 (файл print\_result.py) 22](#_Toc121153302)

[3.6.1. Текст программы 22](#_Toc121153303)

[3.6.2. Результат Задача № 5 (Pycharm) 23](#_Toc121153304)

[3.6.3. Результат Задача № 5 (Командная строка) 24](#_Toc121153305)

[3.7. Задача №6 (файл cm\_timer.py) 24](#_Toc121153306)

[3.7.1. Текст программы 24](#_Toc121153307)

[3.7.2. Результат Задача № 6 (Pycharm) 25](#_Toc121153308)

[3.7.3. Результат Задача № 6 (Командная строка) 25](#_Toc121153309)

[3.8. Задача №7 (файл process\_data.py) 25](#_Toc121153310)

[3.8.1. Текст программы 26](#_Toc121153311)

[3.8.2. Результат Задача №7 (Pycharm) 27](#_Toc121153312)

[3.8.3. Результат Задача №7 (Командная строка) 30](#_Toc121153313)

## Цель лабораторной работы

Изучение возможностей функционального программирования в языке Python.

## Общее 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 раполагаться 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 количествово 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:

1. С использованием lambda-функции.
2. 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](https://github.com/ugapanyuk/BKIT_2021/blob/main/notebooks/fp/files/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))))

## Описание каждой задачи

## Основная программа

**LAB\_03\_04.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 \*  
# from lab\_python\_fp.process\_data 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('МЕНЮ')  
 print('1. Задача 1 (файл field.py)')  
 print('2. Задача 2 (файл gen\_random.py)')  
 print('3. Задача 3 (файл unique.py)')  
 print('4. Задача 4 (файл sort.py)')  
 print('5. Задача 5 (файл print\_result.py)')  
 print('6. Задача 6 (файл cm\_timer.py)')  
 # print('7. Задача 7 (файл process\_data.py)')  
 # Переключатель  
 # switch = int(input('Введите номер пункта: '))  
 switch = get\_argv\_value(1, 'Введите номер пункта: ')  
  
 if(switch == 1):  
 print('Задача 1 (файл field.py)')  
 mas = ''  
 if(len(sys.argv)==1):  
 count\_argv = int(input('Введите количество 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 количество 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 (файл 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:  
 numbers = gen\_random(size, get\_argv\_value(3, 'Введите диапазон от: '),  
 get\_argv\_value(4, 'Введите диапазон до: '))  
 print(numbers)  
  
 elif (switch == 3):  
 print('Задача 3 (файл unique.py)')  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 print(data)  
 mas1 = Unique(data)  
 for i in Unique(mas1):  
 print(i, end=' ')  
 print()  
  
 data\_1 = gen\_random(10, 1, 3)  
 print(data\_1)  
 mas2 = Unique(data\_1)  
 for i in Unique(mas2):  
 print(i, end=' ')  
 print()  
  
 data\_2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 print(data\_2)  
 mas3 = Unique(data\_2)  
 for i in Unique(mas3):  
 print(i, end=' ')  
 print()  
  
 mas4 = Unique(data\_2, ignore\_case=True)  
 for i in Unique(mas4):  
 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)  
 mas3 = Unique(data\_input)  
 print('С чувствительным регистром')  
 for i in Unique(mas3):  
 print(i, end=' ')  
 print()  
  
 print(data\_input)  
 mas3 = Unique(data\_input, ignore\_case=True)  
 print('Без чувствительного регистра')  
 for i in Unique(mas3):  
 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]  
  
 data\_3 = into\_tuple\_from\_str(buff)  
  
 print(data\_3)  
 c = Unique(data\_3)  
 print('С чувствительным регистром')  
 for i in Unique(c):  
 print(i, end=' ')  
 print()  
  
 print(data\_3)  
 c = Unique(data\_3, 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 (файл sort.py)')  
 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 (файл print\_result.py)')  
 fun\_print\_result()  
  
 elif (switch == 6):  
 print('Задача 6 (файл cm\_timer.py)')  
 cm\_timer()  
  
 # elif (switch == 7):  
 # print('Задача 7 (файл process\_data.py)')  
 # print('Переходите в директорию lab\_python\_fp в файл process\_data.py, чтобы выполнить это задание')  
  
 else:  
 print('Ошибка! Нет такого пункта!')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

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

## Задача №1 (файл filed.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'

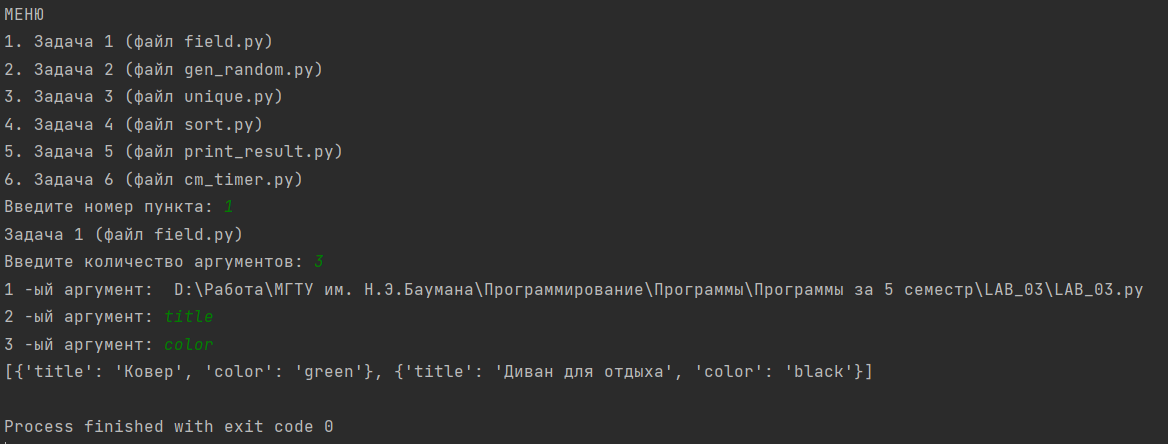
field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

* В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово 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.

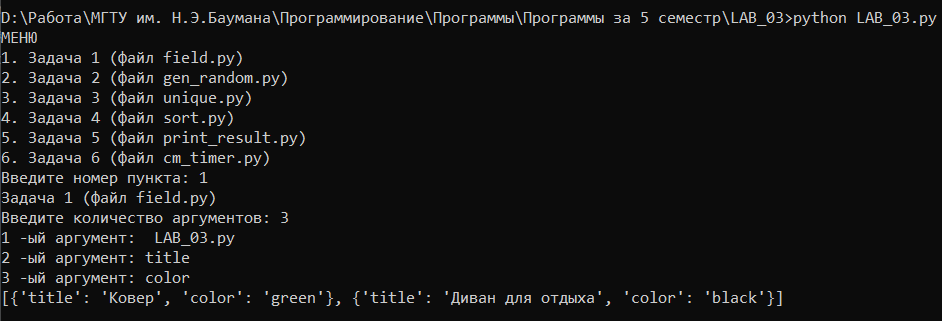
## Текст программы

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):  
 try:  
 # Преобразование в кортеж из строки  
 argv = into\_tuple\_from\_str(\*args)  
 # Необходимо реализовать генератор  
 # len () возвращает длину (количество элементов) в объекте.  
 assert len(argv) > 0, 'Ошибка! Нет аргументов! \nПримечание аргументы не должны быть пустыми!'  
 #range - диапазон, len - длина списка  
 #items - это переменная, в которой на каждой итерации оказывается элемент списка.  
 r = [{} for i in range(len(items))]  
 for i in range(len(items)):  
 for j in items[i]:  
 if j in argv:  
 # updatе - метод обновления словаря элементами из другого объекта словаря.  
 r[i].update({j: items[i][j]})  
 # возврат значения  
 return r  
 except:  
 print('Ошибка! Нет списка в качестве переданного аргумента!')  
  
# Преобразование в строку из кортежа  
def into\_tuple\_from\_str(str):  
 cortes\_buf = []  
 str\_buf = ''  
 for i in range(len(str)):  
 if (str[i] == ' '):  
 cortes\_buf.append(str\_buf)  
 str\_buf = ''  
 else:  
 str\_buf += str[i]  
 # append - метод добавления элементов  
 cortes\_buf.append(str\_buf)  
 # возврат значения  
 return cortes\_buf

## Результат Задача №1 (Pycharm)



## Результат Задача № 1 (Командная строка)



## Задача №2 (файл gen\_random.py)

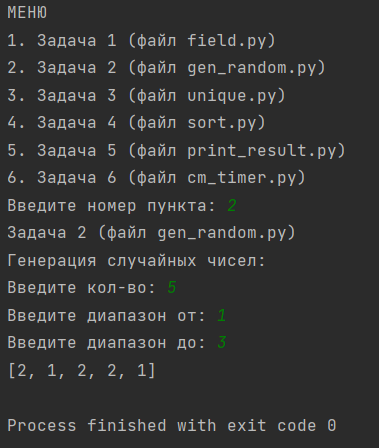
Необходимо реализовать генератор gen\_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона. Пример:

gen\_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

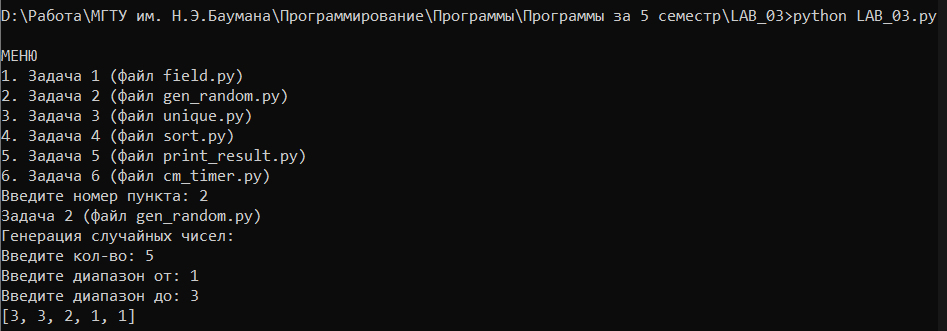
## Текст программы

1. # 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  
     
   import random  
   # gen\_random(количество, минимум, максимум)  
   def gen\_random(num\_count, begin, end):  
    mas = []  
    # range - диапазон  
    for i in range(0, num\_count):  
    # генерация случайных чисел  
    mas.append(int(random.randint(begin, end)))  
    # Возврат значения  
    return mas

## Результат Задача № 2 (Pycharm)



## Результат Задача № 2 (Командная строка)



## Задача №3 (файл unigue.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’]

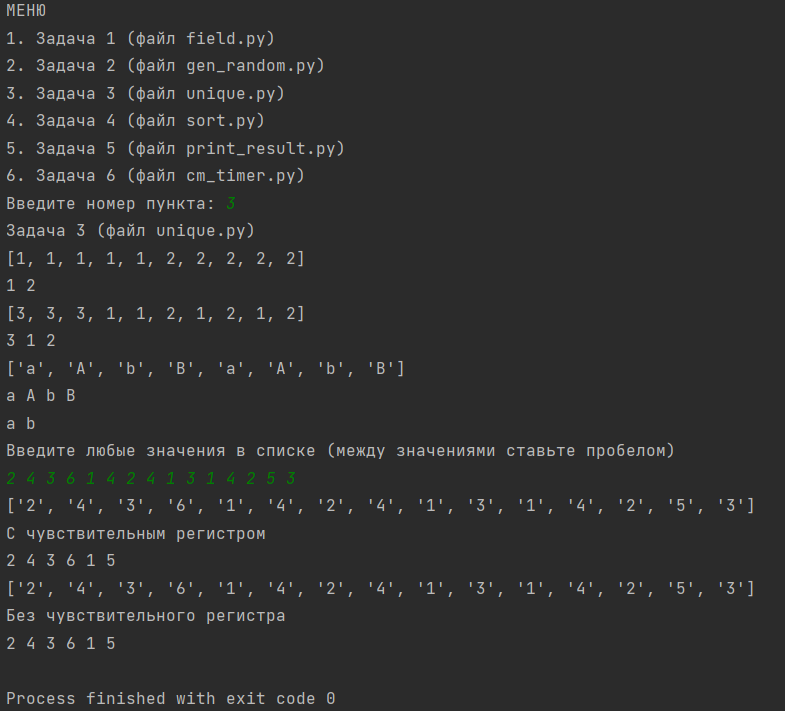
Unique(data) будет последовательно возвращать только a, A, b, B.

Unique(data, ignore\_case=True) будет последовательно возвращать только a, b.

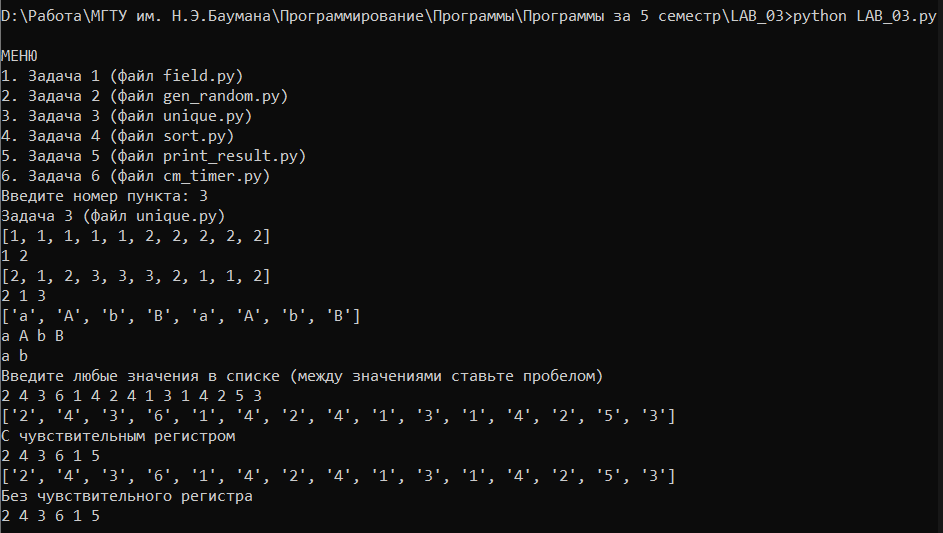
## Текст программы

# 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" - это объект экземпляра, автоматически передаваемый методу экземпляра класса при вызове,  
 # чтобы идентифицировать экземпляр, который его вызвал.  
 self.mas = []  
  
 # Используя кортежи, получаем ключ и значения  
 for key, value in kwargs.items():  
 # Если ключ пустой и значение TRUE, то  
 if key == 'ignore\_case' and value == True:  
 # Методы lower () возвращают строку в нижнем регистре из заданной строки.  
 # Он преобразует все заглавные символы в строчные.  
 items = [i.lower() for i in items]  
  
 for index in items:  
 # Если текущее значение из списка item не совпадает/не существует в созданном списке mas  
 if index not in self.mas:  
 # То присвоем несуществующее значение в созданном списке mas  
 self.mas.append(index)  
 pass  
  
 # Для перехода к следующему элементу используется метод \_\_next\_\_.  
 def \_\_next\_\_(self):  
 try:  
 x = self.mas[self.begin]  
 self.begin += 1  
 return x  
 except:  
 # Оператор raise позволяет принудительно породить исключение. (Заверщение работы итератора)  
 raise StopIteration  
  
 #\_\_iter\_\_(self) метод, который возвращает объект итератора;  
 def \_\_iter\_\_(self):  
 self.begin = 0  
 return self

## Результат Задача № 3 (Pycharm)



## Результат Задача № 3 (Командная строка)



## 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]

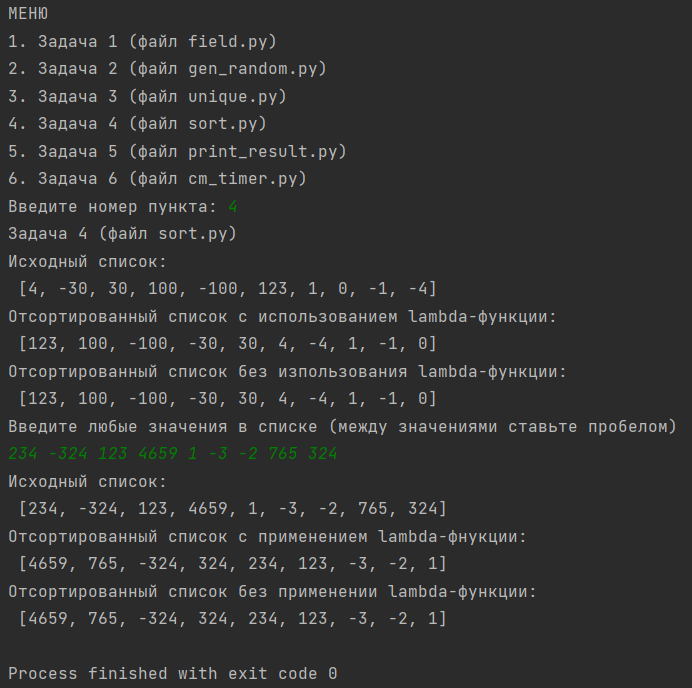
Необходимо решить задачу двумя способами:

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

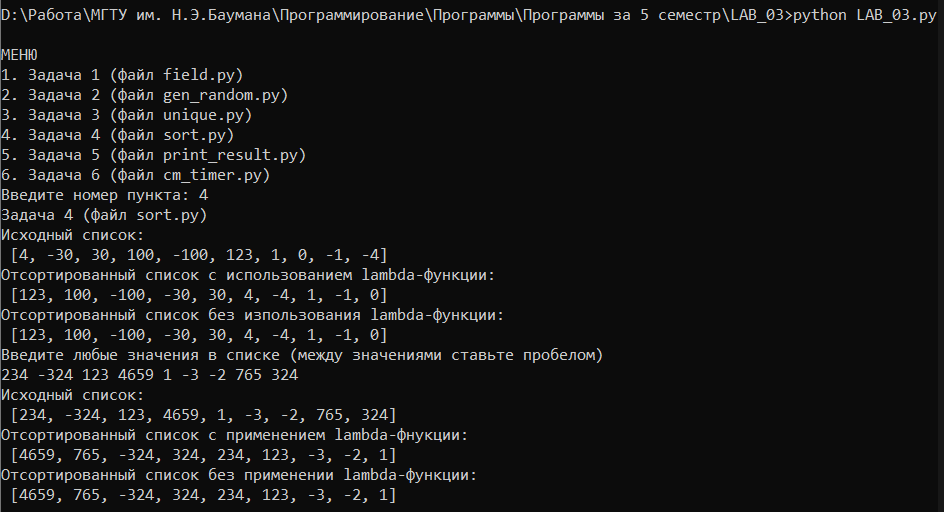
## Текст программы

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]  
  
def sort():  
 print(f'Исходный список:\n {data}')  
 result\_with\_lambda = sorted(data, key=lambda i: -abs(i))  
 print(f'Отсортированный список с использованием lambda-функции:\n {result\_with\_lambda}')  
 # sorted - функция, который упорядочивает значения  
 result = sorted(data, key=abs, reverse=True)  
 print(f'Отсортированный список без изпользования lambda-функции:\n {result}')

## Результат Задача № 4 (Pycharm)



## Результат Задача №4 (Командная строка)



## Задача №5 (файл print\_result.py)

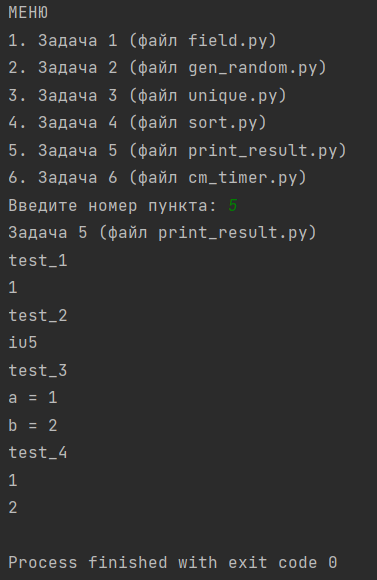
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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.

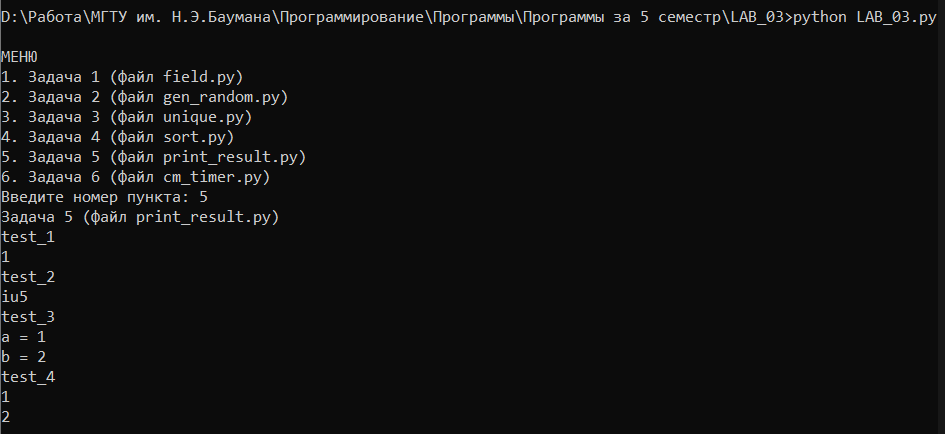
## Текст программы

# Здесь должна быть реализация декоратора  
*'''  
\*args - неименованные аргументы (arguments), для которых важен порядок передачи.  
Также используется название "позиционные аргументы".  
\*\*kwargs - именованные аргументы (keywords), для которых не важен порядок передачи.  
Также используется название "аргументы, передаваемые по ключевым словам".  
'''*# # Декоратор - это функция, ожидающая ДРУГУЮ функцию в качестве параметра.  
# указываем имя декоратора и то, что он принимает function в качестве своей переменной.  
def print\_result(function):  
 #объявление функции-обёртки realization(). Тело обёртки описывает, что именно мы будем делать с функцией, ранее принятой декоратором.  
 def realization(mas=[], \*args, \*\*kwargs):  
 # Печать название вызываемой функции  
 print(function.\_\_name\_\_)  
 if len(mas) == 0:  
 result = function(\*args, \*\*kwargs)  
 else:  
 result = function(mas, \*args, \*\*kwargs)  
  
 if type(result) == int or type(result) == str:  
 print(result)  
 # Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.  
 elif type(result) is list:  
 print('\n'.join(map(str, result)))  
 # Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.  
 elif type(result) is dict:  
 for key, meaning in result.items():  
 print(f'{key} = {meaning}')  
 # Функция zip создает итератор, который объединяет элементы из нескольких источников данных.  
 elif type(result) == zip:  
 for name, chislo in result:  
 print(name, chislo)  
 else:  
 print(result)  
 #функция result возвращает переменную  
 return result  
 #декоратор возвращает нам уже саму функцию realization, точнее, результат её работы над функцией function.  
 return realization  
  
# Синтаксис для обертывания функции в декоратор  
@print\_result  
def test\_1():  
 return 1  
  
# Синтаксис для обертывания функции в декоратор  
@print\_result  
def test\_2():  
 return 'iu5'  
  
# Синтаксис для обертывания функции в декоратор  
@print\_result  
def test\_3():  
 return {'a': 1, 'b': 2}  
  
# Синтаксис для обертывания функции в декоратор  
@print\_result  
def test\_4():  
 return [1, 2]  
  
# Синтаксис для обертывания функции в декоратор  
def fun\_print\_result():  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

## Результат Задача № 5 (Pycharm)



## Результат Задача № 5 (Командная строка)



## 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)

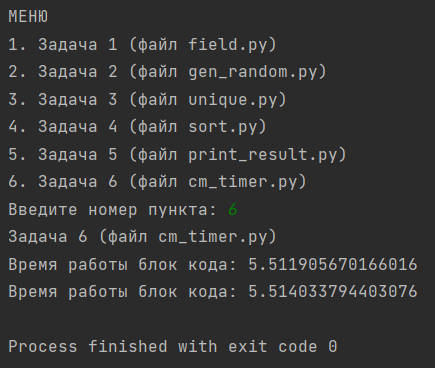
После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

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).

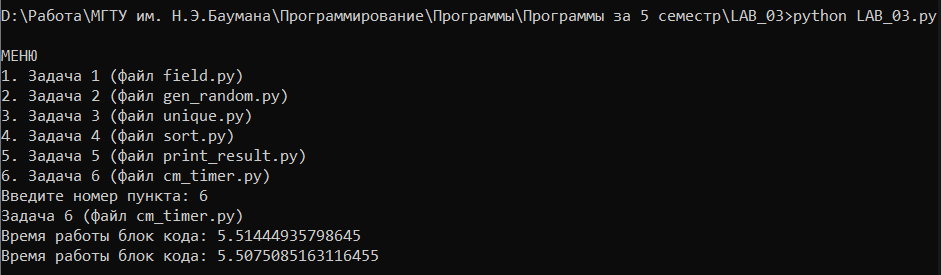
## Текст программы

#time - время, sleep - задержка  
from time import time, sleep  
from contextlib import contextmanager  
  
# На основе 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'Время работы блок кода: {self.\_end - self.\_start}')  
  
# С использованием библиотеки contextlib  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time()  
 # Yield – ключевое слово, которое используется вместо return. С его помощью функция возвращает значение  
 # без уничтожения переменных, кроме того, при каждом последующем вызове функция начинает своё  
 # выполнение с оператора yield.  
 yield None  
 end\_time = time()  
 print(f'Время работы блок кода: {end\_time - start\_time}')  
  
def cm\_timer():  
 with cm\_timer\_1():  
 sleep(5.5)  
  
 with cm\_timer\_2():  
 sleep(5.5)

## Результат Задача № 6 (Pycharm)



## Результат Задача № 6 (Командная строка)



## 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](https://github.com/ugapanyuk/BKIT_2021/blob/main/notebooks/fp/files/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.

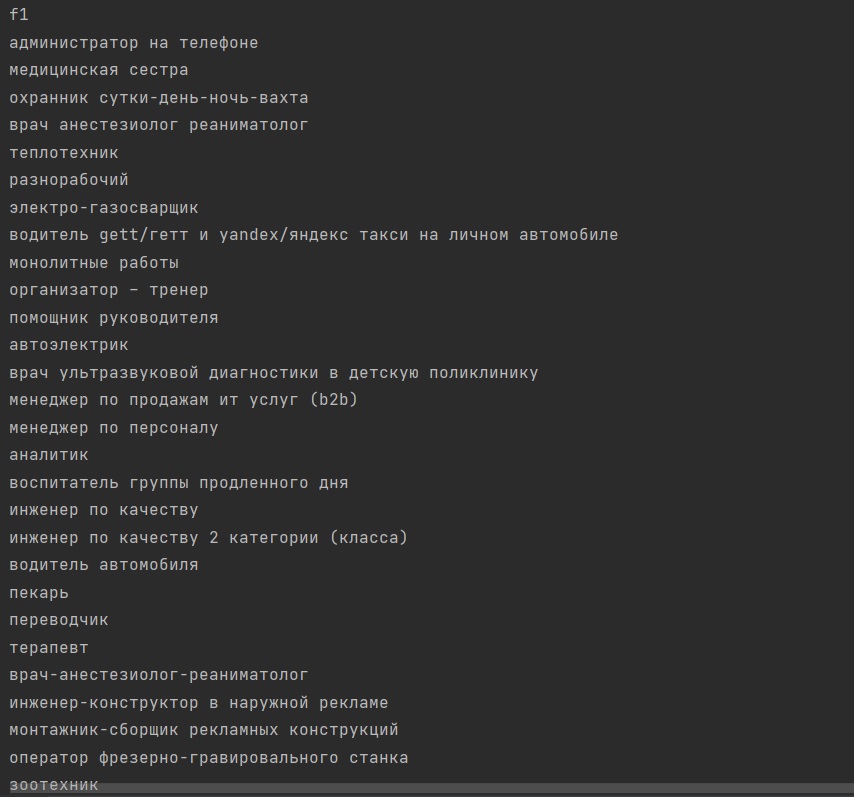
## Текст программы

import json  
# Сделаем другие необходимые импорты  
from operator import concat  
from filed import field  
from unique import Unique  
from sort import sort  
from gen\_random import gen\_random  
from cm\_timer import cm\_timer\_1  
from gen\_random import gen\_random  
from print\_result import print\_result  
# путь файла  
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):  
 # Подборка названий работы, которые не совпадают друг друга в списке  
 # info\_name\_work = Unique([i['name-work'] for i in field(data, 'name-work')], ignore\_case=True)  
 # Отсортируем  
 # info\_name\_work\_sorted = sorted(info\_name\_work, key=str, reverse = False)  
 # return info\_name\_work.mas.sort()  
 # return sorted(info\_job\_name, key=str, reverse = False)  
 # return sorted(list(set([el['name-work'] for meaning 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):  
 # list() - создает или преобразует переданный объект в список.  
 # filter() - применяет другую функцию к заданному итерируемому объекту (список, строка, словарь и так далее),  
 # проверяя, нужно ли сохранить конкретный элемент или нет.  
 # startswith() – начинается ли строка с определенного шаблона или нет.  
 return list(filter(lambda i: i.startswith('программист'), arg))  
  
@print\_result  
def f3(arg):  
 # list() - создает или преобразует переданный объект в список.  
 # map() — это функция, позволяющая обрабатывать и преобразовывать все элементы в итерируемом объекте без цикла for.  
 # concat - сложение строк.  
 return list(map(lambda x: concat(x, ' c опытом Python'), arg))  
  
@print\_result  
def f4(arg):  
 # list() - создает или преобразует переданный объект в список.  
 # zip - создает итератор, который объединяет элементы из нескольких источников данных.  
 return list(zip(arg, ['зарплата ' + str(meaning) + ' руб.' for meaning 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)))))  
 print()

## Результат Задача №7 (Pycharm)

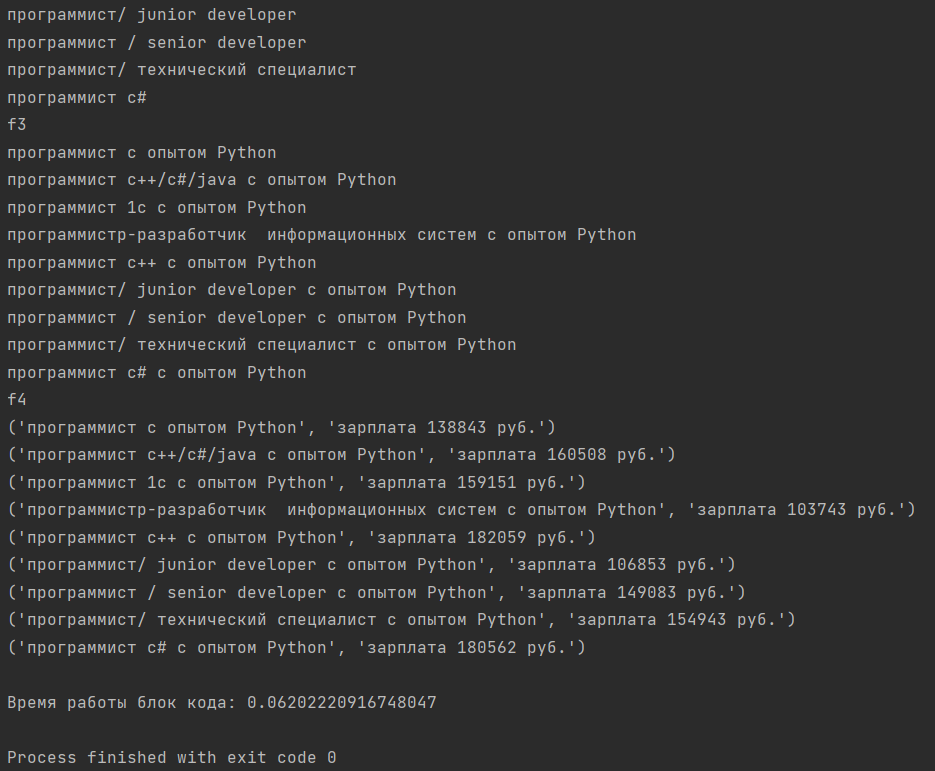
Для 7 пункта включаем: “Curret File”:

Начало результата 7 пункта:

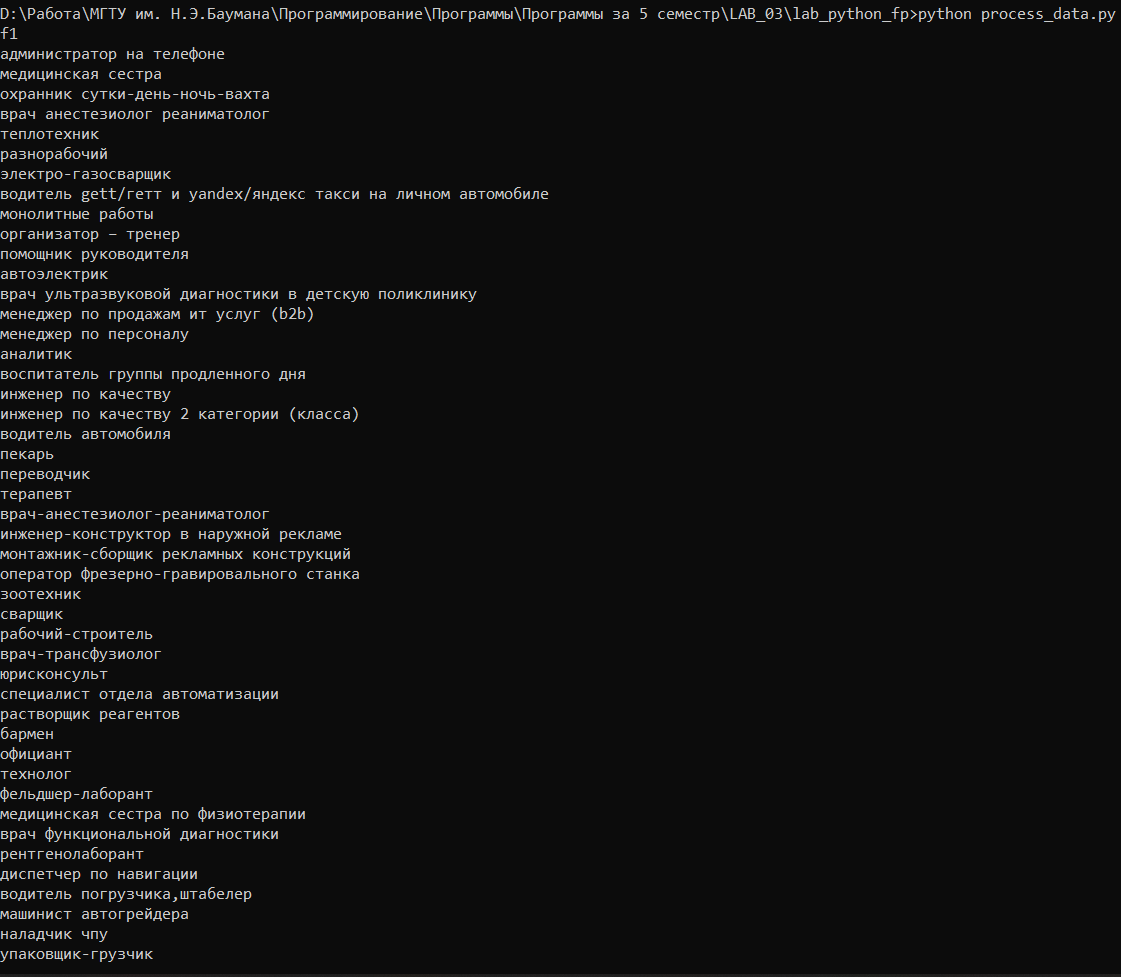


…

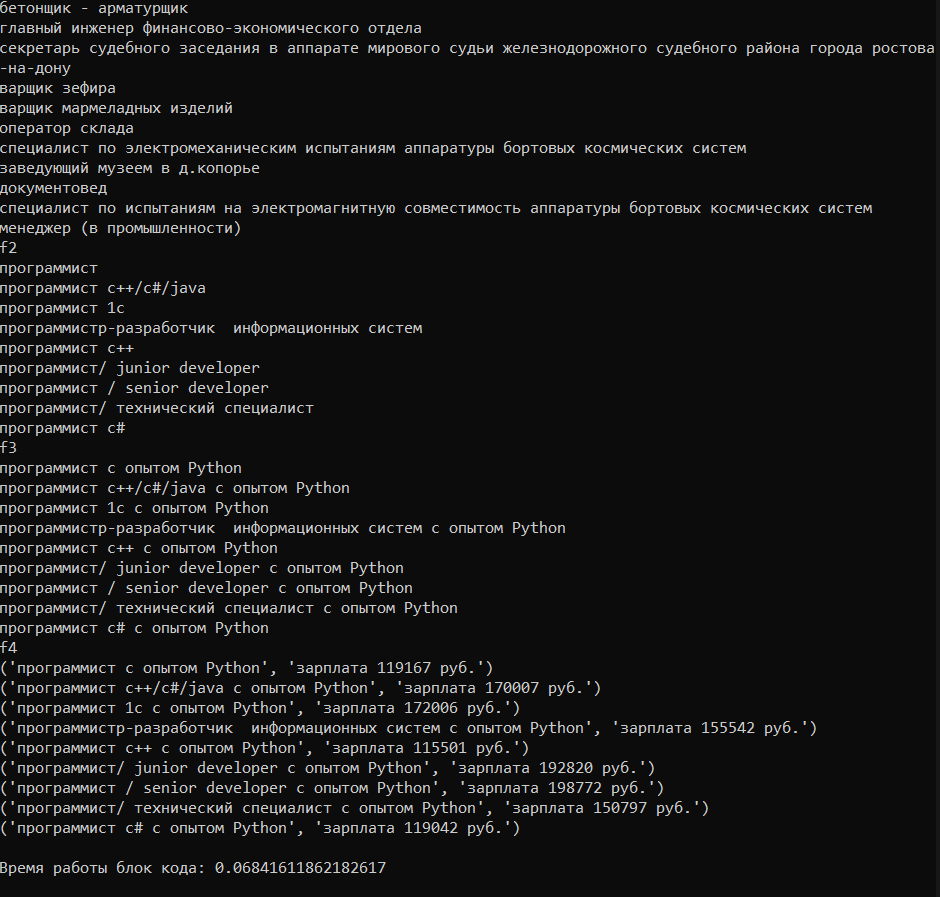
Конец результата программы 7 пункта:



## Результат Задача №7 (Командная строка)

****

**…**

****