Московский государственный технический университет

имени Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

**Отчёт по лабораторной работе** **№3**

**«Функциональные возможности языка Python.»**

Выполнила:

Рыжкова Юлия Николаевна

Группа ИУ5-31Б

Проверил:

Канев Антон Игоревич

Кафедра ИУ5

Москва 2021г.

**Задание**

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 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.

### **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

### **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(1, 3, 10)

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.

### **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.

### **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.

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.

**Текст программы**

Файл **field.py**

goods = [

{'title': 'Ковер', 'price': 2000, 'color': 'green'},

{'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}

]

def field(items, \*args):

assert len(args) > 0

# Необходимо реализовать генератор

if len(args) == 1:

for dict in items:

if args[0] in dict.keys():

yield dict[args[0]]

else:

for dict in items:

checked = set(args) & set(dict.keys())

if bool(checked) == True:

yield {arg:dict[arg] for arg in args if arg in checked}

if \_\_name\_\_ == '\_\_main\_\_':

list1 = field(goods, 'title')

for i in list1:

print(i)

dict1 = field(goods, 'title', 'price')

for i in dict1:

print(i)

Файл **gen\_random.py**

from random import randint

def gen\_random(num\_count, begin, end):

for i in range(num\_count):

yield randint(begin, end)

if \_\_name\_\_ == '\_\_main\_\_':

nums = gen\_random(5, 1, 3)

for i in nums:

print(i)

Файл **unique.py**

class Unique:

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

if 'ignore\_case' in kwargs.keys():

self.ignore\_case = kwargs['ignore\_case']

else:

self.ignore\_case = False

self.it = iter(items)

self.rep = set()

def \_\_next\_\_(self):

while True:

next\_ = next(self.it)

if self.ignore\_case and isinstance(next\_, str):

rep\_low = (i.lower() for i in self.rep)

if next\_.lower() not in rep\_low:

self.rep.add(next\_)

return next\_

elif next\_ not in self.rep:

self.rep.add(next\_)

return next\_

def \_\_iter\_\_(self):

return self

if \_\_name\_\_ == '\_\_main\_\_':

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

for i in Unique(data1):

print(i)

data2 = ['a','A','B','b']

for i in Unique(data2, ignore\_case=True):

print(i)

Файл **sort.py**

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

result = sorted(data, reverse = True)

print(result)

result\_with\_lambda = sorted(data, key=lambda x: x, reverse = True)

print(result\_with\_lambda)

Файл **print\_result.py**

# Здесь должна быть реализация декоратора

def print\_result(func):

def wrap(\*args, \*\*kwargs):

print(func.\_\_name\_\_)

var = func(\*args, \*\*kwargs)

if type(var) == list:

for i in var:

print(i)

elif type(var) == dict:

for i in var:

print(i, " = ", var[i])

else:

print(var)

return func(\*args, \*\*kwargs)

return wrap

@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()

Файл **cm\_timer.py**

from contextlib import contextmanager

import time

class cm\_timer\_1:

def \_\_init\_\_(self):

self.\_start\_time = None

def start(self):

self.\_start\_time = time.perf\_counter()

def stop(self):

elapsed\_time = time.perf\_counter() - self.\_start\_time

self.\_start\_time = None

print("time: ", elapsed\_time)

def \_\_enter\_\_(self):

self.start()

return self

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

self.stop()

@contextmanager

def cm\_timer\_2():

try:

start\_time = time.perf\_counter()

yield start\_time

finally:

elapsed\_time = time.perf\_counter() - start\_time

print("time: ", elapsed\_time)

if \_\_name\_\_ == '\_\_main\_\_':

with cm\_timer\_1():

time.sleep(5.5)

with cm\_timer\_2():

time.sleep(5.5)

Файл **process\_data.py**

import json

from cm\_timer import cm\_timer\_1

import unique

from print\_result import print\_result

from gen\_random import gen\_random

path = "data\_light.json"

with open(path) as f:

data = json.load(f)

@print\_result

def f1(arg):

jobs = list(sorted(unique.Unique((i['job-name'] for i in data), ignore\_case=True)))

return jobs

@print\_result

def f2(arg):

jobs\_filtered = list(filter(lambda x: str(x)[0:11].lower()=='программист', arg))

return jobs\_filtered

@print\_result

def f3(arg):

moddified = list(map(lambda x: str(x)+' с опытом Python', arg))

return moddified

@print\_result

def f4(arg):

salary = list(gen\_random(len(arg), 100000, 200000))

salary = list(map(lambda x: " зарплата " + str(x) + " руб.", salary))

return list(zip(arg, salary))

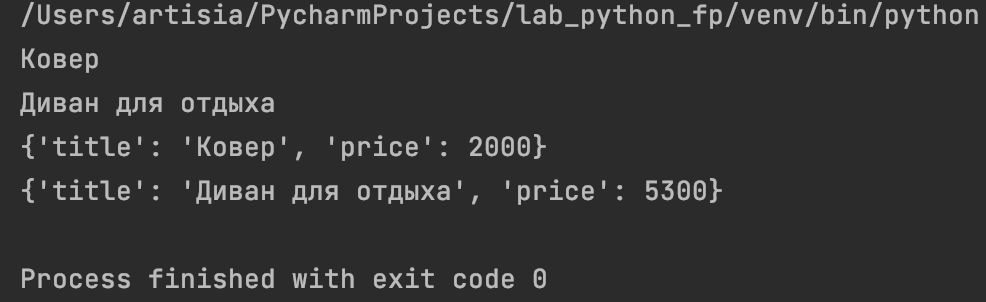
if \_\_name\_\_ == '\_\_main\_\_':

with cm\_timer\_1():

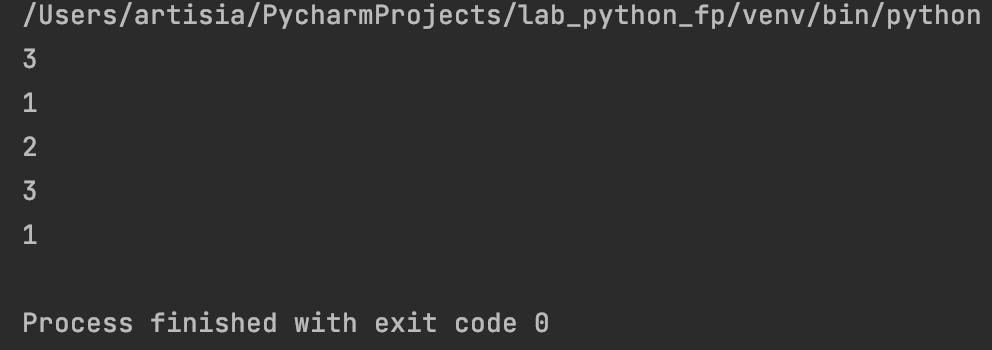
f4(f3(f2(f1(data))))

**Экранные формы с примерами выполнения программы**

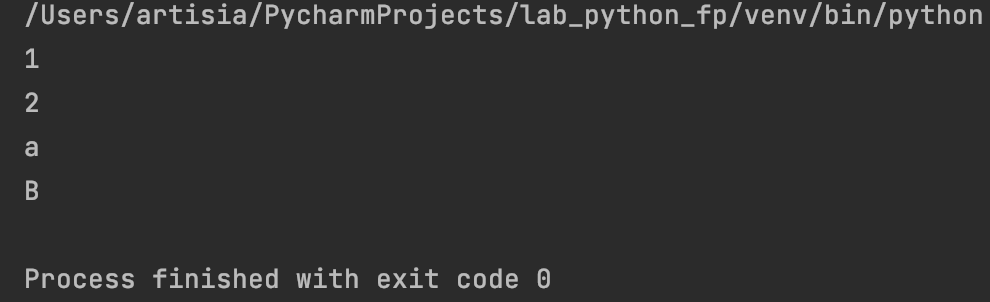
Файл **field.py**



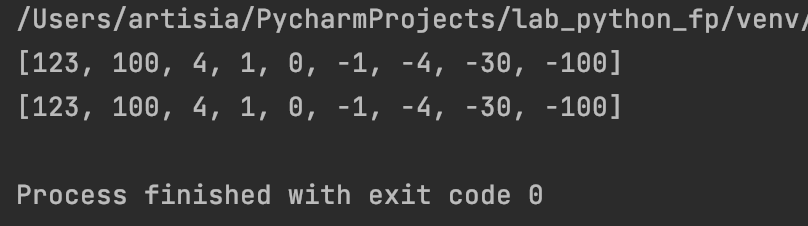
Файл **gen\_random.py**

****

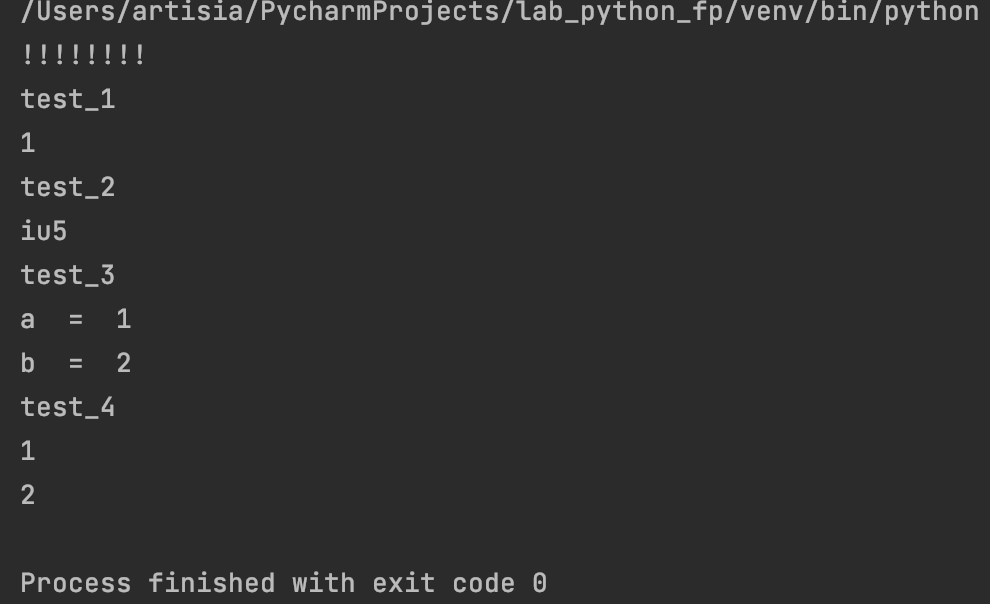
Файл **unique.py**

****

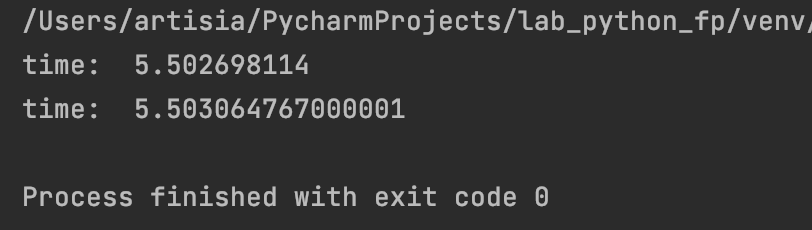
Файл **sort.py**

****

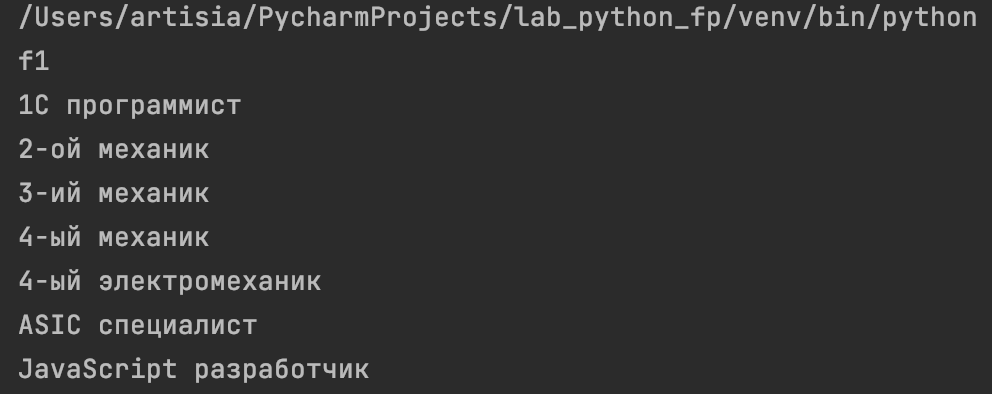
Файл **print\_result.py**

****

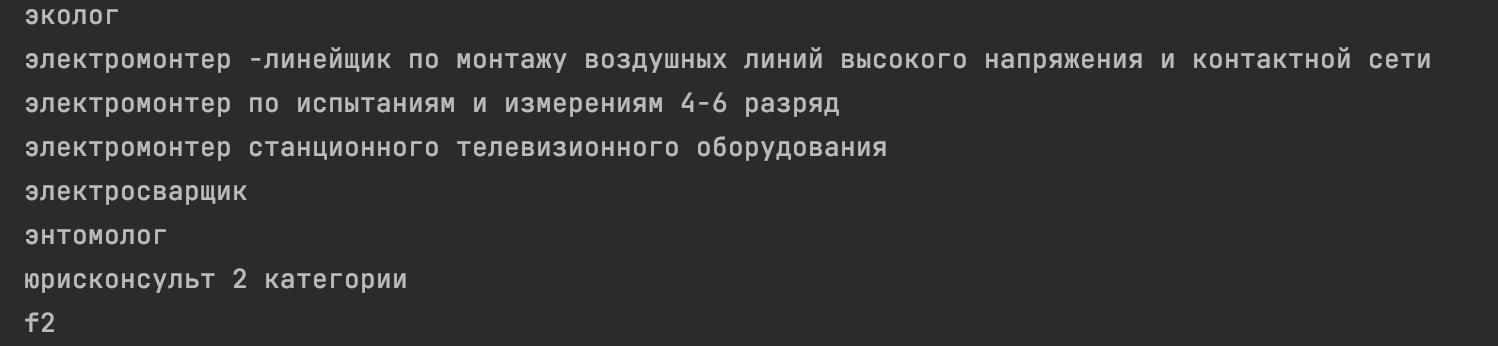
Файл **cm\_timer.py**

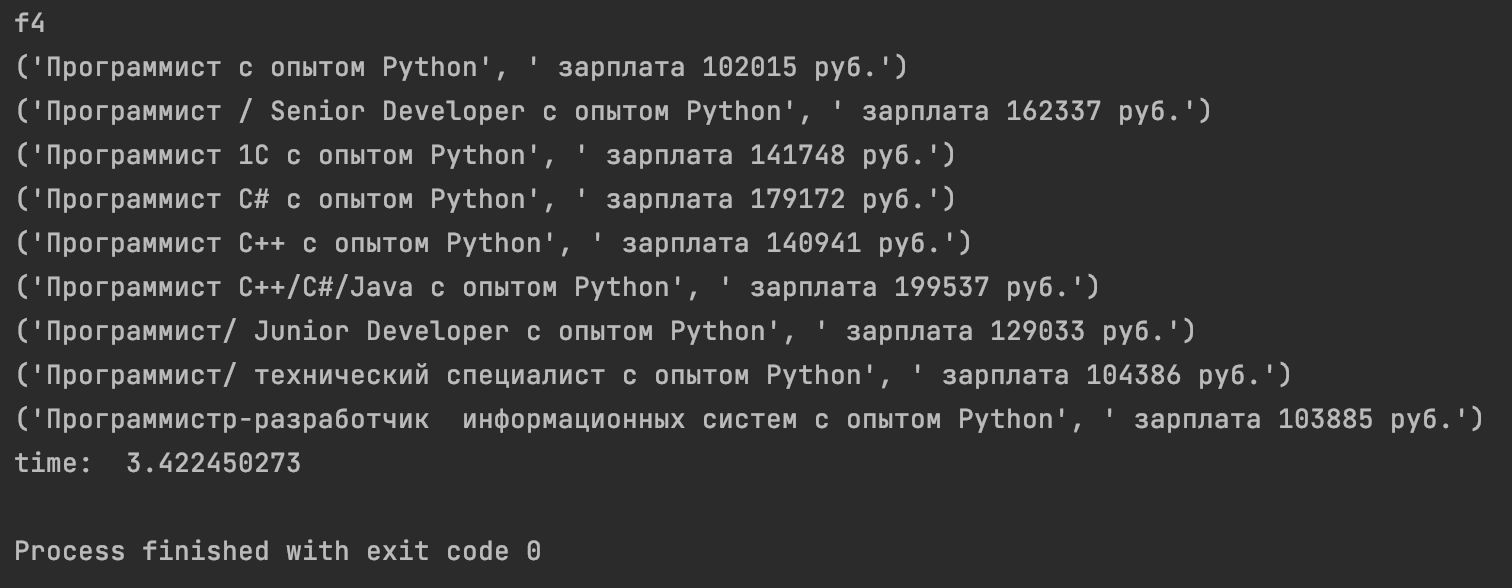
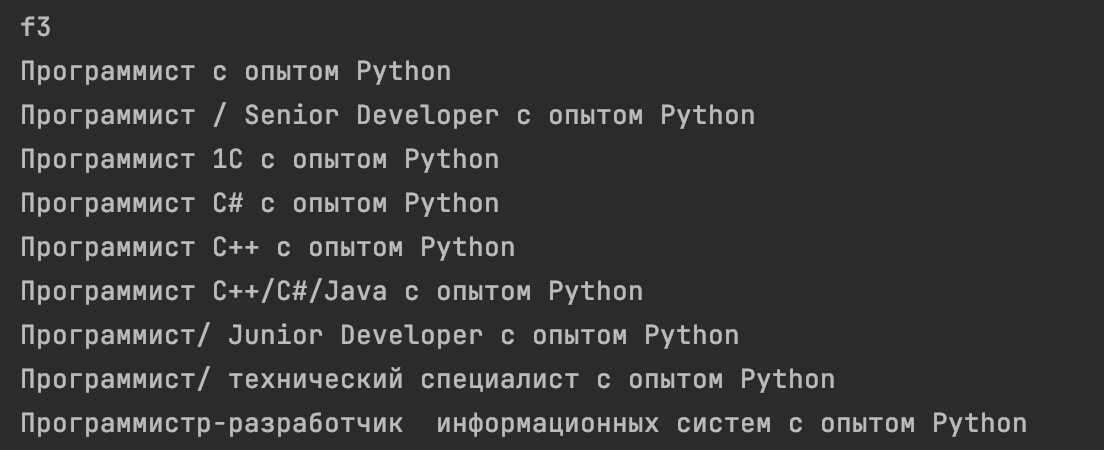
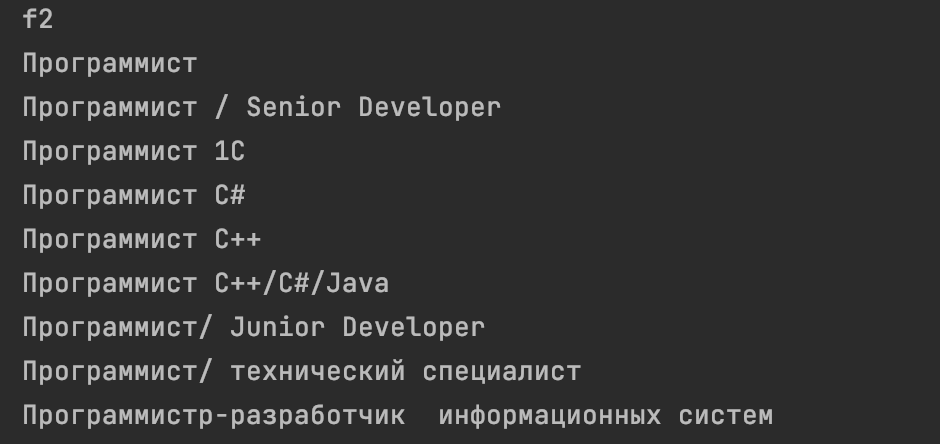
****

Файл **process\_data.py**

****

**…**

****

****