**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 вычислительная техника»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №3

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-52Б |  | преподаватель каф. ИУ5 |
| Веревкина Диана В. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2021 г.

# ****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 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.

1. В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово аргументов.
2. Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.
3. 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)

1. Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.
2. 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.
3. При реализации необходимо использовать конструкцию \*\*kwargs.
4. Итератор должен поддерживать работу как со списками, так и с генераторами.
5. Итератор не должен модифицировать возвращаемые значения.

# 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.

Необходимо решить задачу двумя способами:

С использованием lambda-функции.

Без использования lambda-функции.

# Задача 5 (файл print\_result.py)

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

1. Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.
2. Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.
3. Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

# 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.

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)

1. В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.
2. В файле data\_light.json содержится фрагмент списка вакансий.
3. Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.
4. 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.
5. Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.
6. Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.
7. Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.
8. 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.
9. 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

**main**

from lab\_pythone\_fp.field import \*  
from lab\_pythone\_fp.gen\_random import \*  
from lab\_pythone\_fp.unique import \*  
from lab\_pythone\_fp.sort import \*  
from lab\_pythone\_fp.print\_result import \*  
from lab\_pythone\_fp.cm\_timer import \*  
from lab\_pythone\_fp.process\_data import \*  
  
  
def ex1():  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'color': 'black'},  
 {'color': 'wr'},  
 {'title': None, 'price': 25, 'color': 'red'},  
 {'color': 'blue'},  
 ]  
 print('Задание 1')  
 print(list(field(goods, 'title')))  
 print(list(field(goods, 'title', 'price')))  
  
  
def ex2():  
 print('\n\nЗадание 2')  
 g = gen\_random(5, 1, 3)  
 for item in g:  
 print(item, end=', ')  
 print('\n' , list(gen\_random(6, 2, 10)), sep='')  
  
  
def ex3():  
 print('\n\nЗадание 3')  
 print('Обработка разного списка')  
 data = [1, 1, 1, 1, 1, 2, 2, 2, '2', 2, 2, 'F', 'f', 'A']  
 mycl = Unique(data, ignore\_case=True)  
 myit = iter(mycl)  
 while True:  
 try:  
 print(next(myit))  
 except StopIteration:  
 break  
  
 print('Обработка генератора')  
 mycl1 = Unique(gen\_random(10, 1, 3))  
 myit1 = iter(mycl1)  
 while True:  
 try:  
 print(next(myit1))  
 except StopIteration:  
 break  
  
  
def ex4():  
 print('\n\nЗадание 4')  
 data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]  
 srt(data)  
  
  
def ex5():  
 print('\n\nЗадание 5')  
 all\_test()  
  
  
def ex6():  
 print('\n\nЗадание 6')  
 test\_cm\_timer()  
  
  
def ex7():  
 print('\n\nЗадание 7')  
 start()  
  
  
def main():  
 a = (ex1, ex2, ex3, ex4, ex5, ex6, ex7)  
 while True:  
 print('Выберите задание 1-7:')  
 i = int(input())  
 if i != 0:  
 a[i-1]()  
 else:  
 return  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**field.py**

def field(lst: list, \*keys):  
 if len(keys) == 1: # если 1 аргумент  
 for i in lst: # проходим список  
 a = i.get(keys[0]) # берем такое значение словаря  
 if a is not None: # если оно есть  
 yield a # выводим  
  
 elif len(keys) > 1: # если больше 1 аргумента  
 d: dict = {} # создаем блуждающий словарь  
 for i in lst: # проходим лист  
 for j in keys: # и для каждого входного аргумента  
 a = i.get(j) # получаем значение  
 if a is not None: # если оно есть  
 d[j] = a # добавляем в блуждающий словарь пару ключ из аргумента и значение  
  
 if d: # если словарь не пуст  
 yield d.copy() # копируем его в список  
 d.clear() # чистим блужд. словарь

**gen\_random.py**

from random import randint  
  
def gen\_random(n: int, st: int, fin: int):  
 l: list = []  
 for i in range(n):# n раз выдаем значение рандома  
 yield randint(st, fin)  
 return l.append(randint(st, fin))

**unique.py**

class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 if kwargs.get('ignore\_case'): # если дан флаг  
 for i, val in enumerate(items): # переводим все str в нижний регистр  
 if type(val) == str:# если взяли стр  
 items[i] = val.lower()#опускаем текст  
 self.items = list(items)  
  
 def \_\_next\_\_(self):  
 a = self.l[-1]#сохраняем последний аргумент списка  
 while self.n < len(self.items) and self.items[self.n] in self.l:#если мы не прошли весь входной список и данный аргумент в нашем списке  
 self.n += 1#пропускаем его  
 if self.n > len(self.items):#если вышли за предел  
 raise StopIteration('Достигнут предел итерации')#кидаем ошибку  
 elif self.n < len(self.items):#если пока не вышли и не на последнем аргументе  
 self.l.append(self.items[self.n])#добавляем не повт. аргумент в наш список  
 self.n += 1#берем млед аргумент  
 return a#выдаем сохраненный ранее последний аргумент списка  
  
 def \_\_iter\_\_(self):#при созд итератора  
 self.l = []#пустой список  
 self.n = 1#номер след аргумента для проверки  
 self.l.append(self.items[0])#добавляем 0 аргумент, тк его еще не было  
 return self

**sort.py**

def srt(data):  
 print('С лямбдой:')  
 print(sorted(data, key=lambda x: abs(x), reverse=True))#используем ключ как лямбду  
 print('Без лямбды:')  
 print(sorted(data, key= abs, reverse=True)) # используем ключ без лямбды

**print\_result.py**

def print\_result(func):  
 def wrapper(\*arg):  
 print('-----\nИмя оборачиваемой функции: {}'.format(func.\_\_name\_\_))  
 print('Результат выполнения:')  
 a = func(\*arg)#выполняем функцию и сохр результат  
 if type(a) == list:#если это список  
 print(\*a, sep='\n')#выводим в столбик  
 elif type(a) == dict:#если это словарь  
 for key, value in a.items():  
 print("{} = {}".format(key, value))#выводим в столбик в цикле  
 else:  
 print(a)#иначе просто выводим  
 print('-----')  
 return a  
  
 return wrapper  
  
  
@print\_result  
def test\_1():  
 return 1  
  
  
@print\_result  
def test\_2():  
 return 'iu5'  
  
  
@print\_result  
def test\_3():  
 return {'a': 1, 'b': 2, 'c': 3, 'd': 4}  
  
  
@print\_result  
def test\_4():  
 return [1, 2, 'a', 28]  
  
  
def all\_test():  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

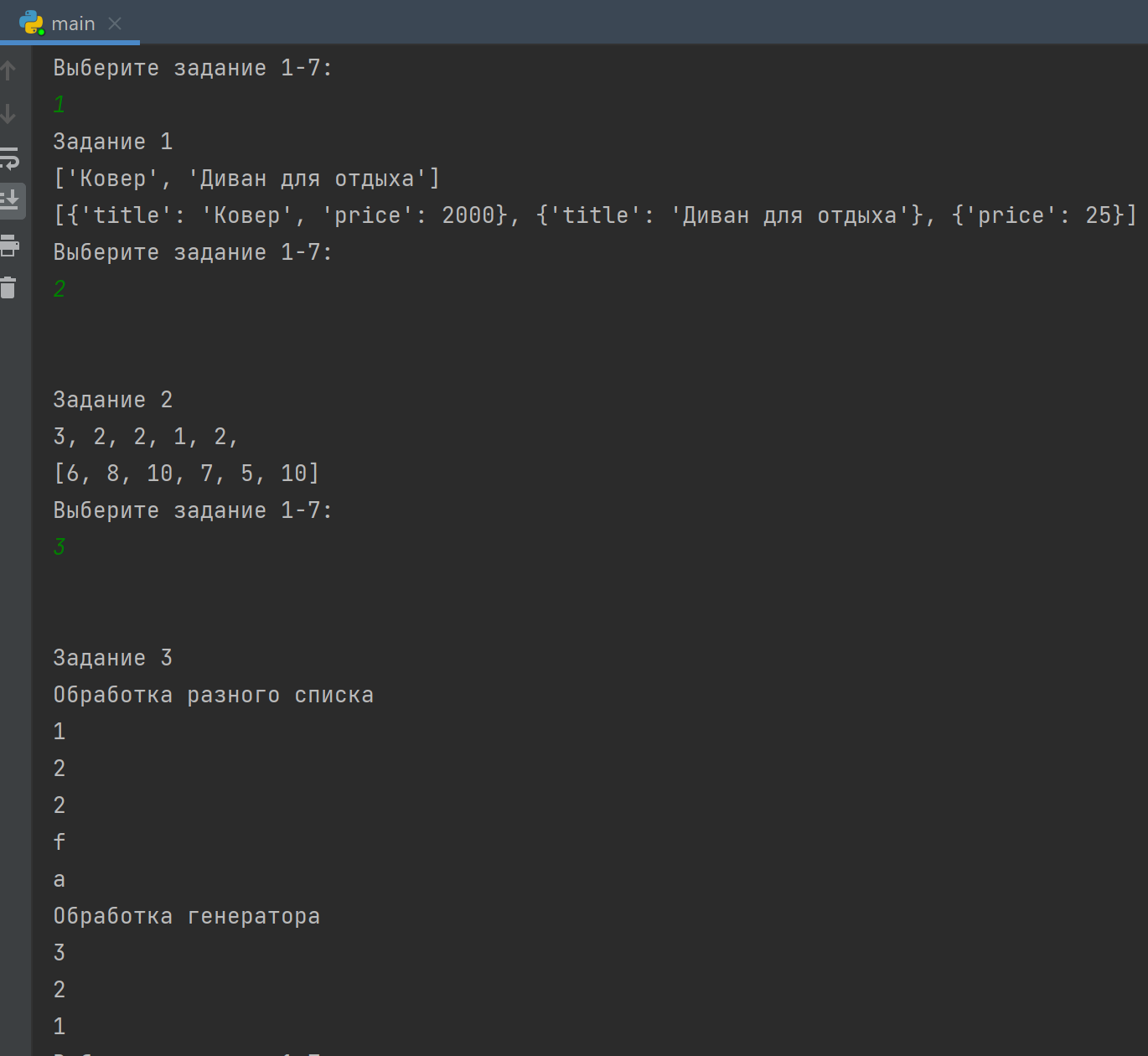
**cm\_timer.py**

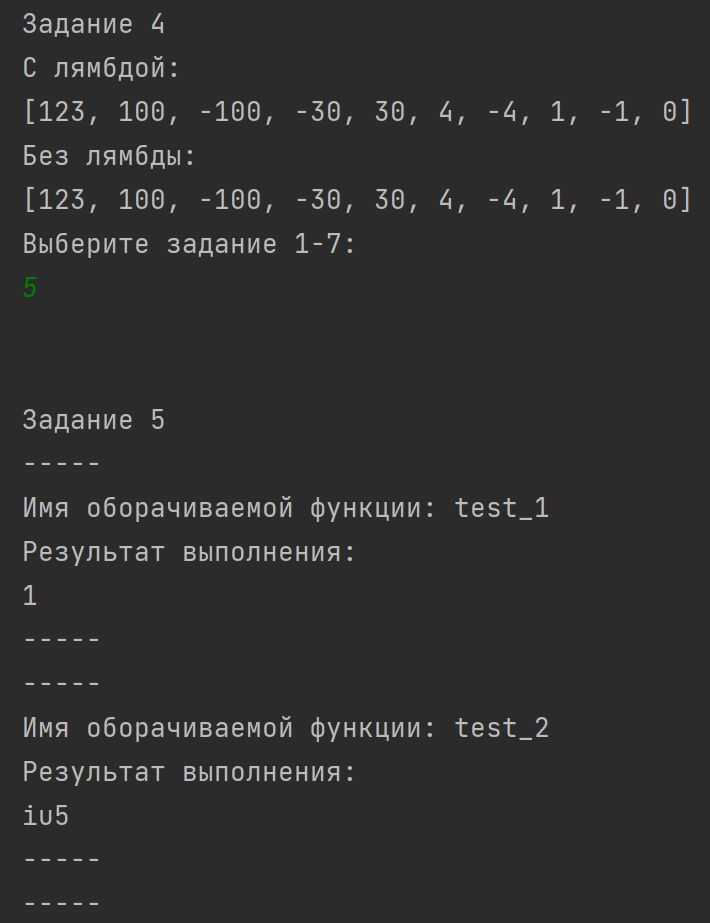
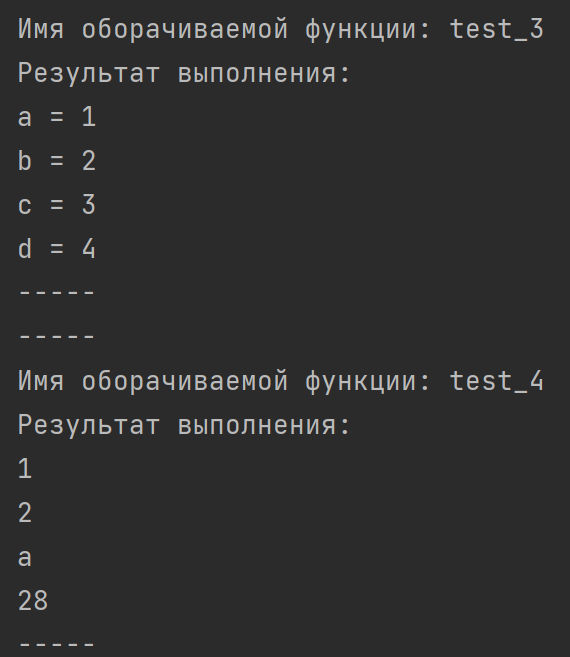
import time  
from contextlib import contextmanager  
  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):#запуск  
 self.start\_time = time.time()#сохр время  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):#выход  
 print("Время выполнения {} секунд".format(time.time() - self.start\_time))#выводим время  
 if exc\_val:# если получили экз иисключ  
 raise  
  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()#запуск  
 yield  
 print("Время выполнения {} секунд".format(time.time() - start\_time))#выход  
  
  
def test\_cm\_timer():  
 print('Вызываем конт. мен. как класс')  
 with cm\_timer\_1():  
 time.sleep(1)  
  
 print('Вызываем конт. мен. как contextmanager')  
 with cm\_timer\_2():  
 time.sleep(1)

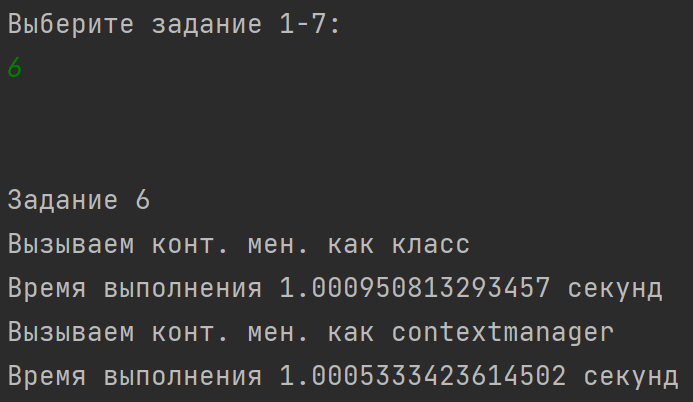
**process\_data.py**

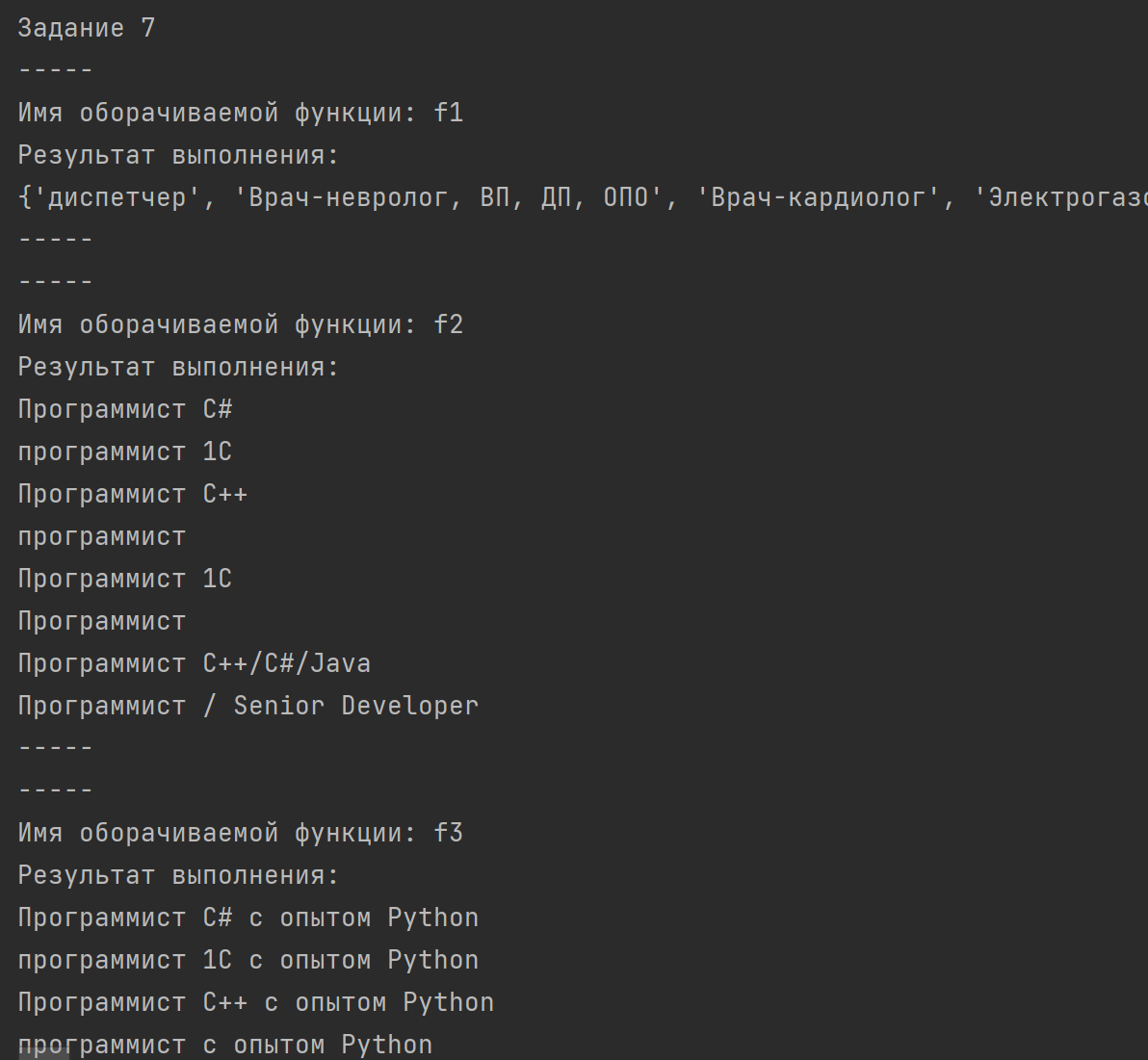
import json  
from random import randint  
from lab\_pythone\_fp.print\_result import print\_result  
from lab\_pythone\_fp.cm\_timer import cm\_timer\_1  
from lab\_pythone\_fp.gen\_random import gen\_random  
  
  
def start():  
 with open('lab\_pythone\_fp\data\_light.json', 'r',  
 encoding='utf-8') as f: # открываем файл personal.json и указываем его кодировку — что бы можно было работать с русскими буквами  
 data = json.load(f) # загоняем в переменную все, что получилось в результате работы библиотеки  
  
 with cm\_timer\_1():#запускаем последовательность  
  
 print(f4(f3(f2(f1(data)))))  
 #f4(f3(f2(f1(data))))  
  
  
@print\_result  
def f1(arg):  
 return set([x.get("job-name") for x in arg])#возвр список без повторений ключа job-name  
  
@print\_result  
def f2(arg):  
 return list(filter((lambda x: x.split()[0].lower() in 'программист'), arg))#возвр список профессий, который мы делим на слова по пробелу и смотрим, чтобы 1 слово было программист  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + ' с опытом Python', arg))# добавляем ко все мпрофессиям "с опытом Python"  
  
@print\_result  
def f4(arg):  
 #return list(zip(arg, list(map(lambda x: 'с зарплатой ' + x + ' рублей', [str(randint(100000, 200000)) for x in range(len(arg))]))))#проходим по всем профессиям и создаем словари, в которых указываем профессию - рандомную зарплату.Зарплата это список, размером с длину сп. профессий  
 return list(zip(arg, list(map(lambda x: 'с зарплатой ' + str(x) + ' рублей', list(gen\_random(len(arg),100000,200000 ))))))#проходим по всем профессиям и создаем словари, в которых указываем профессию - рандомную зарплату.Зарплата это список, размером с длину сп. профессий

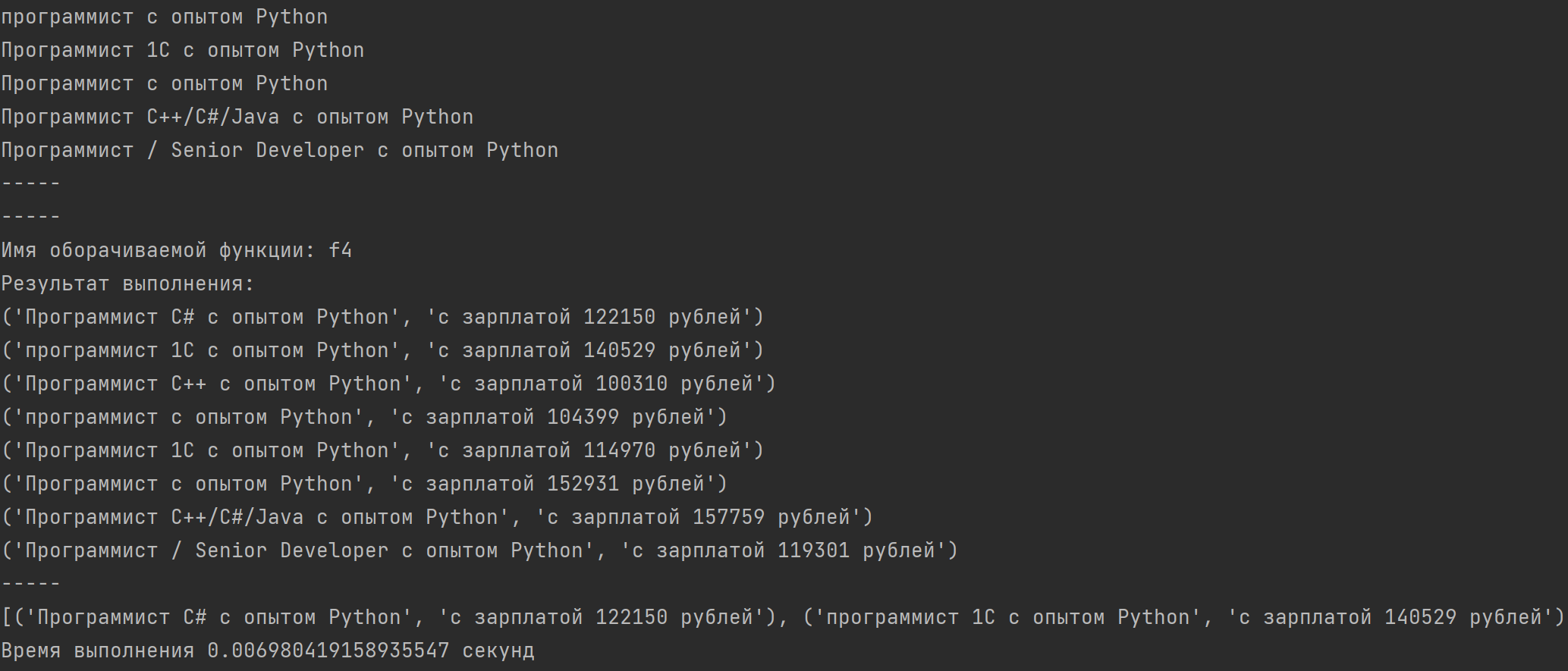
# Экранные формы с примерами выполнения программы









# Вывод

В ходе выполнения данной лабораторной работы были повторены основные конструкции языка Python. Также были изучены возможности языка функционального программирования Python я.