|  |  |
| --- | --- |
| 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 ИУ5 «Системы обработки информации и управления»**

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №3

Выполнила:

студент группы ИУ5-51Б

Павловская А.А.

27.10.2020

Проверил:

|  |
| --- |
| преподаватель каф. ИУ5 |
| Гапанюк Ю.Е. |

Москва, 2020 г.

**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 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.

**Текст программы**

def field(items, \*args):

assert len(args) > 0

d1 = {}

for i in range(len(items)):

for j in range (len(args)):

d1[args[j]] = items[i][args[j]]

print(d1)

def main():

print('Задание 1:')

goods = [

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

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

]

print('Пример 1:')

field(goods, 'title')

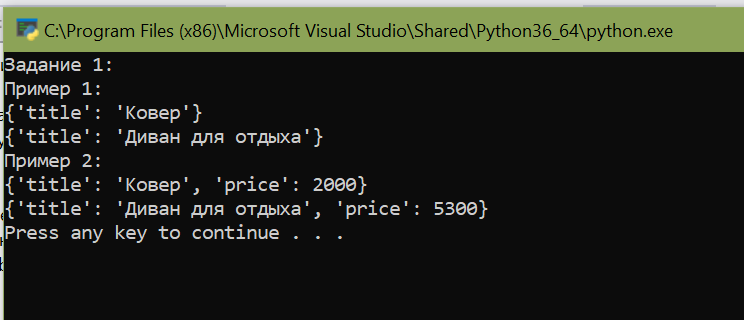
print('Пример 2:')

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

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Экранная форма с примерами выполнения программы**

****

**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**

import random

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

A =[0]\*num\_count

for i in range(num\_count):

A[i] = random.randint(begin, end)

return A

def main():

print('Задание 2:')

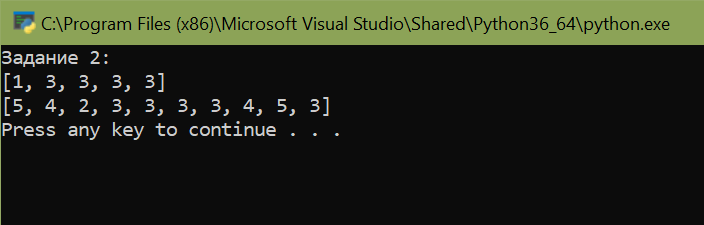
print(gen\_random(5, 1, 3))

print(gen\_random(10, 2, 5))

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Экранная форма с примерами выполнения программы**

****

**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.

**Текст программы**

from gen\_random import gen\_random

class Unique:

"""Итератор, оставляющий только уникальные значения."""

def \_\_init\_\_(self, data, \*\*kwargs):

self.used\_elements = set()

self.data = data

self.index = 0

self.ignore\_case = kwargs.get('ignore\_case')

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

return self

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

while True:

if self.index >= len(self.data):

raise StopIteration

else:

current = self.data[self.index]

if ((self.ignore\_case == True)and(type(current)==str)):

current = current.lower()

self.index = self.index + 1

if current not in self.used\_elements:

self.used\_elements.add(current)

return current

def main():

print('Задание 3:')

print('Пример 1:')

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

for i in Unique(data):

print(i)

print('Пример 2:')

data = gen\_random(6, 1, 3)

for i in Unique(data):

print(i)

print('Пример 3:')

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

for i in Unique(data):

print(i)

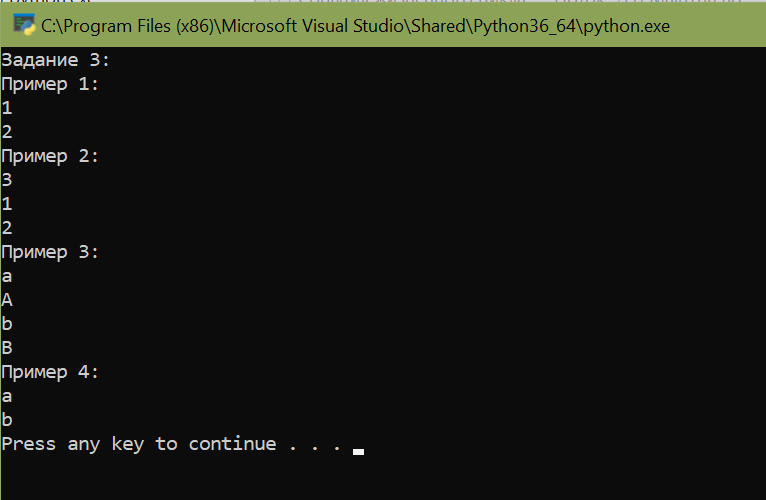
print('Пример 4:')

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

print(i)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Экранная форма с примерами выполнения программы **

**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, 100, -100, 123, 1, 0, -1, -4]

def main():

print('Задание 4:')

result = sorted(data,key = abs, reverse = True)

print(result)

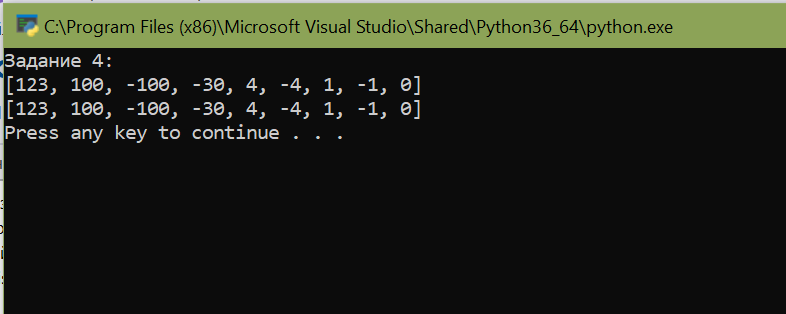
result\_with\_lambda = sorted(data, key=lambda n: abs(n), reverse = True)

print(result\_with\_lambda)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Экранная форма с примерами выполнения программы**



**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**

def print\_result(func\_to\_decorate):

def decorated\_func(\*args):

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

obj = func\_to\_decorate(\*args)

if isinstance(obj, dict):

for i in obj.items():

print('{} = {}'.format(i[0], i[1]))

return obj

elif isinstance(obj, list):

for i in range(len(obj)):

print('{}'.format(obj[i]))

return obj

else:

print(obj)

return obj

return decorated\_func

@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 main():

print('Задание 5:')

test\_1()

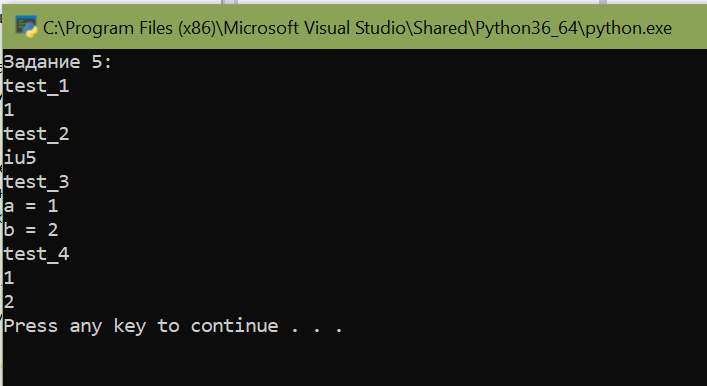
test\_2()

test\_3()

test\_4()

if \_\_name\_\_ == "\_\_main\_\_":

main()  
**Экранная форма с примерами выполнения программы**

****

**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.

**Текст программы**

from contextlib import contextmanager

import time

@contextmanager

def cm\_timer\_1():

start\_time = time.clock()

yield

print ('time: ',time.clock()-start\_time)

class Cm\_Timer\_2:

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

self.all\_time = 0

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

self.all\_time = time.clock()

def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):

if exp\_type is not None:

print(exp\_type, exp\_value, traceback)

else:

self.all\_time = time.clock()-self.all\_time

print ('time: ',self.all\_time)

def main():

print('Задание 6:')

print('cm\_timer\_1:')

with cm\_timer\_1():

try:

time.sleep(5.5)

except:

pass

print('Cm\_timer\_2:')

try:

with Cm\_Timer\_2() as cm\_object:

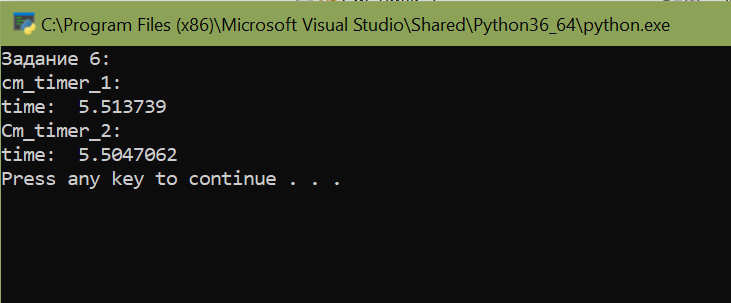
time.sleep(5.5)

except:

pass

if \_\_name\_\_ == "\_\_main\_\_":

main()  
**Экранная форма с примерами выполнения программы**

****

**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/iu5team/iu5web-fall-2020/tree/master/labs/lab3/data/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

import sys

from lab\_python\_fp.gen\_random import gen\_random

from lab\_python\_fp.cm\_timer import cm\_timer\_1

from lab\_python\_fp.print\_result import print\_result

from lab\_python\_fp.Unique import Unique

path = 'C:\\Users\\Настя\\source\\repos\\LabsRIP\\Lab3RIP\\data\_light1.json'

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

with open(path,encoding='utf-8') as f:

data = json.load(f)

# Сортировка списка профессий без повторений

#(строки в разном регистре считаются равными)

@print\_result

def f1(arg):

result11 = []

for i in arg:

result11.append(i['job-name'])

result12 = list(sorted(Unique(result11,ignore\_case = True)))

return result12

# Вывод входного массива только с теми элементами,

# которые начинаются со слова “программист”.

@print\_result

def f2(arg):

result2 = list(filter(lambda i: i.startswith('программист'), arg))

#result2 = list(filter(lambda i:'программист' in i, arg))

return result2

# Добавление к каждому элементу массива строки “с опытом Python”

@print\_result

def f3(arg):

result3 = list(map(lambda x: x+' с опытом Python', arg))

return result3

#Генерирование для каждой специальности зарплаты от 100 000 до 200 000 рублей

#и присоединение её к названию специальности

@print\_result

def f4(arg):

salary = list(map(lambda x:'зарплата '+str(x)+' руб.' , gen\_random(len(arg), 100000, 200000)))

result4 = list(zip(arg, salary))

return result4

def main():

print('Задание 7:')

with cm\_timer\_1():

try:

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

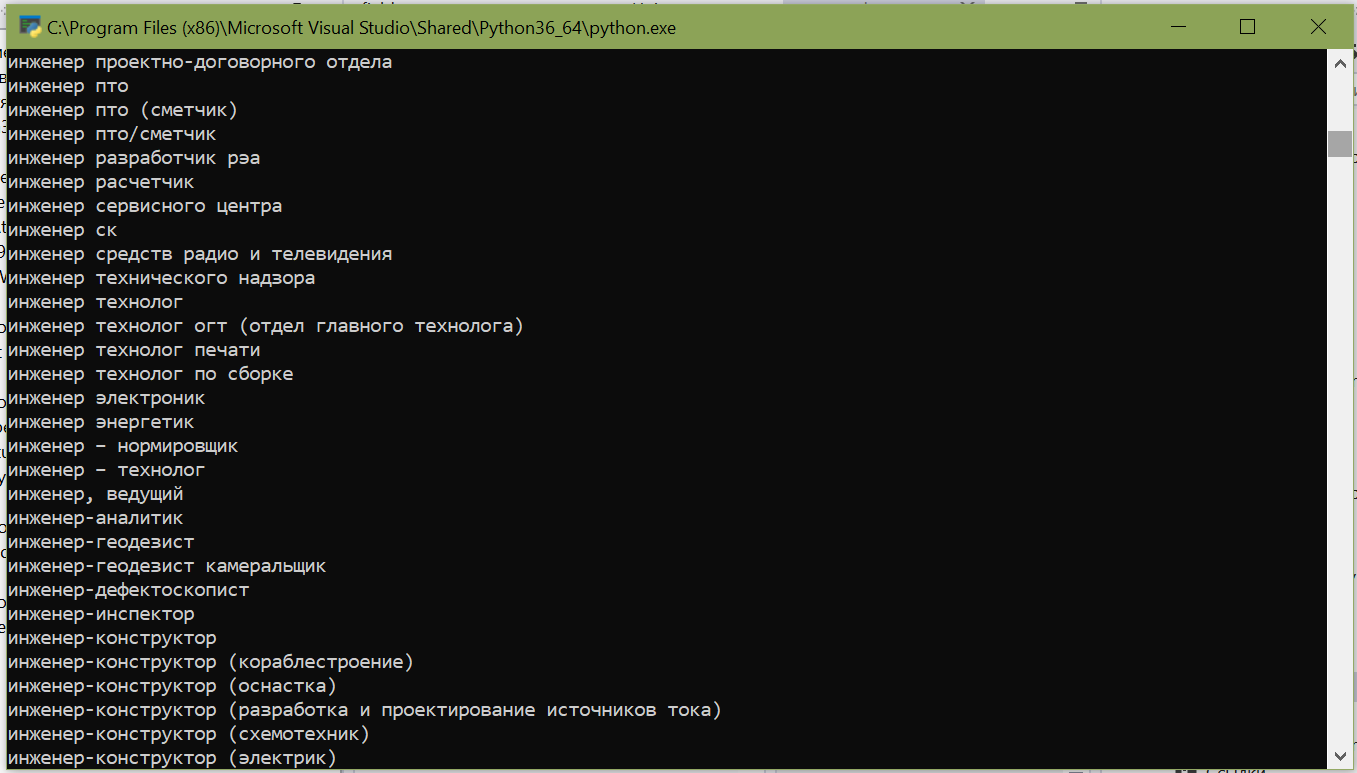
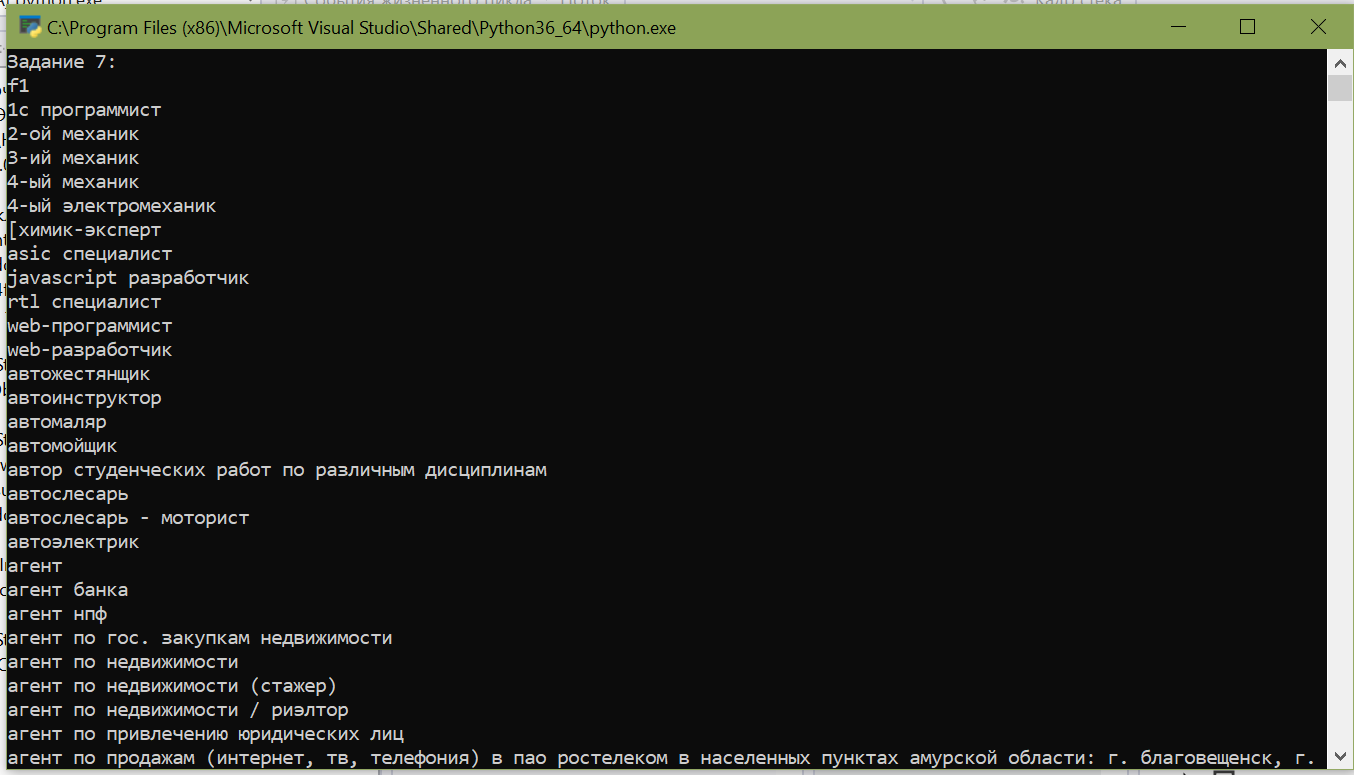
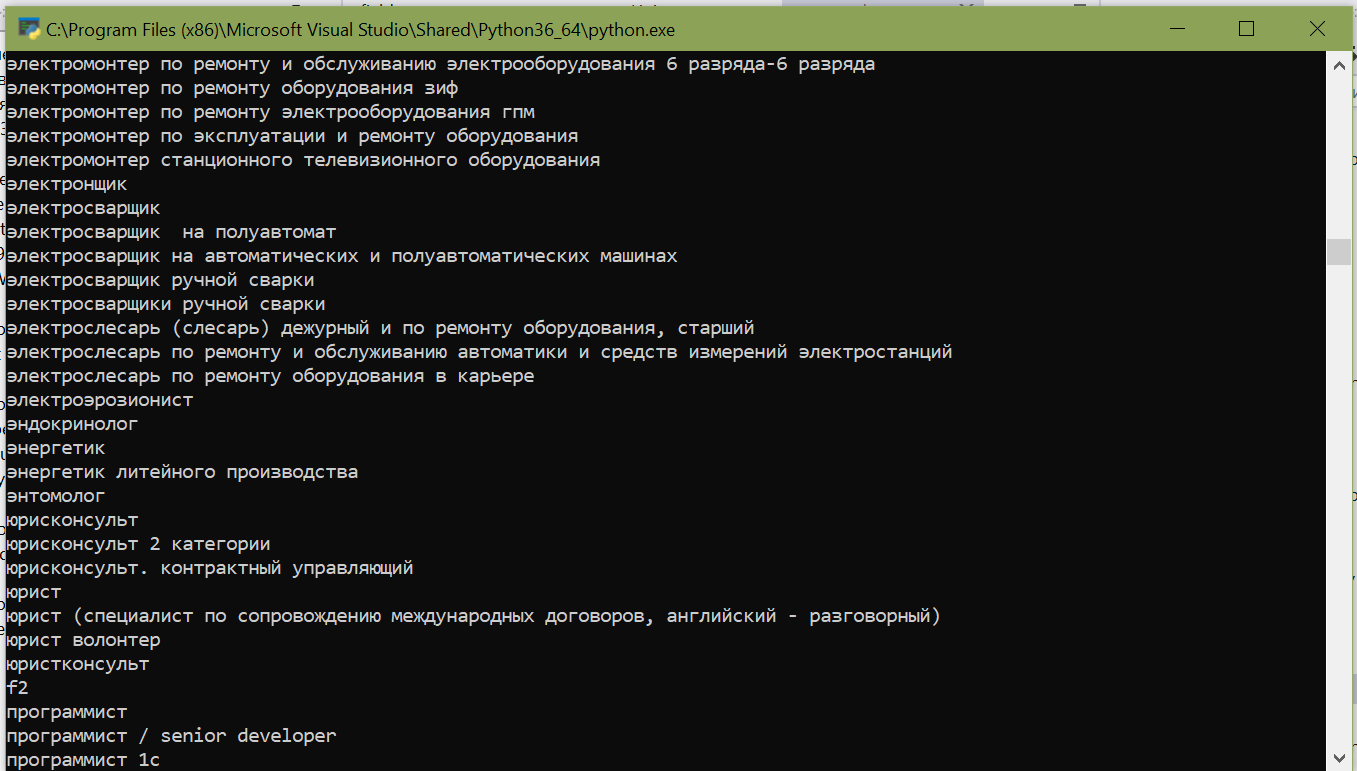
except:

pass

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Экранные формы с примерами выполнения программы**

**** **** 