|  |  |
| --- | --- |
|  | **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»**

Вопияшин Никита ИУ5-35Б

Парадигмы и конструкции языков программирования

**ОТЧЁТ ПО**

**Лабораторной работе №3-4**

Москва

2023

**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.

* field.py: необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.
* 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.
* unique.py: необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.
* 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.
* print\_result.py: необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.
* 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.
* process\_data.py: 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.

**Текст программы.**

**field.py**

def field(dicts, \*args):  
 assert len(args) > 0  
  
 if len(args) == 1:  
 key = args[0]  
 for i in dicts:  
 if i.get(key):  
 yield i[key]  
 else:  
 for dict\_ in dicts:  
 if all(not dict\_.get(i) for i in args):  
 continue  
 res = {}  
 for i in args:  
 if dict\_.get(i):  
 res[i] = dict\_.get(i)  
 yield res  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'color': 'black'}  
 ]  
 a = field(goods, 'title', 'price') # должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}  
 print(next(a))  
 print(next(a))

**gen\_random.py**

import random  
  
  
# 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):  
 random.seed()  
  
 for \_ in range(num\_count):  
 yield random.randint(begin, end)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 a = gen\_random(5, 1, 3)  
 print(next(a))  
 print(next(a))  
 print(next(a))  
 print(next(a))  
 print(next(a))

**unique.py**

class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 ignore\_case = bool(kwargs.get('ignore\_case'))  
 self.items = []  
 used\_items = set()  
 for i in items:  
 if ignore\_case and i.lower() not in used\_items:  
 self.items.append(i)  
 used\_items.add(i.lower())  
 elif not ignore\_case and i not in used\_items:  
 self.items.append(i)  
 used\_items.add(i)  
 self.index = 0  
  
 def \_\_next\_\_(self):  
 if self.index < len(self.items):  
 res = self.items[self.index]  
 self.index += 1  
 return res  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 self.index = 0  
 return self  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
  
 a = Unique(data, ignore\_case=True)  
 print(next(a))  
 print(next(a))

**sort.py**

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 result = sorted(data, key=abs, reverse=True)  
 print(result)  
  
 result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)  
 print(result\_with\_lambda)

**print\_result.py**

# Здесь должна быть реализация декоратора  
def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 res = func(\*args, \*\*kwargs)  
 print(func.\_\_name\_\_)  
  
 if type(res) == list:  
 print(\*res, sep='\n')  
 elif type(res) == dict:  
 for k, w in res.items():  
 print(f'{k} = {w}')  
 else:  
 print(res)  
 return res  
 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}  
  
  
@print\_result  
def test\_4():  
 return [1, 2]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

**cm\_timer.py**

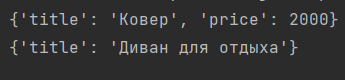
import contextlib  
import time  
from datetime import timedelta  
  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):  
 self.start\_time = time.time()  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 end\_time = time.time()  
 print("Execution time in seconds:", timedelta(seconds=end\_time - self.start\_time).seconds)  
 return False  
  
  
@contextlib.contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield start\_time  
 end\_time = time.time()  
 print("Execution time in seconds: {}".format(timedelta(seconds=end\_time - start\_time).seconds))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 time.sleep(5)  
  
 with cm\_timer\_2() as start\_time:  
 time.sleep(5)

**process\_data.py**

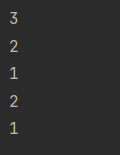
import json  
  
import unique  
import field  
import gen\_random  
from print\_result import print\_result  
from cm\_timer import cm\_timer\_1  
  
path = './data\_light.json'  
  
with open(path, encoding='utf-8') as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg):  
 return sorted(unique.Unique(field.field(data, 'job-name'), ignore\_case=True))  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda x: x.lower().startswith("программист"), arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: (x + " с опытом Python"), arg))  
  
  
@print\_result  
def f4(arg):  
 salaries = gen\_random.gen\_random(len(arg), 100\_000, 200\_000)  
 result = []  
 for job, salary in zip(arg, salaries):  
 result.append(f"{job}, зарплата {salary} руб")  
 return result  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

Экранные формы:

**field.py**



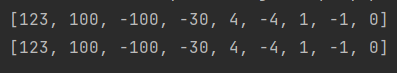
**gen\_random.py**



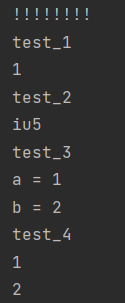
**unique.py**

****

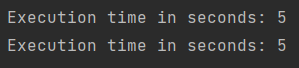
**sort.py**



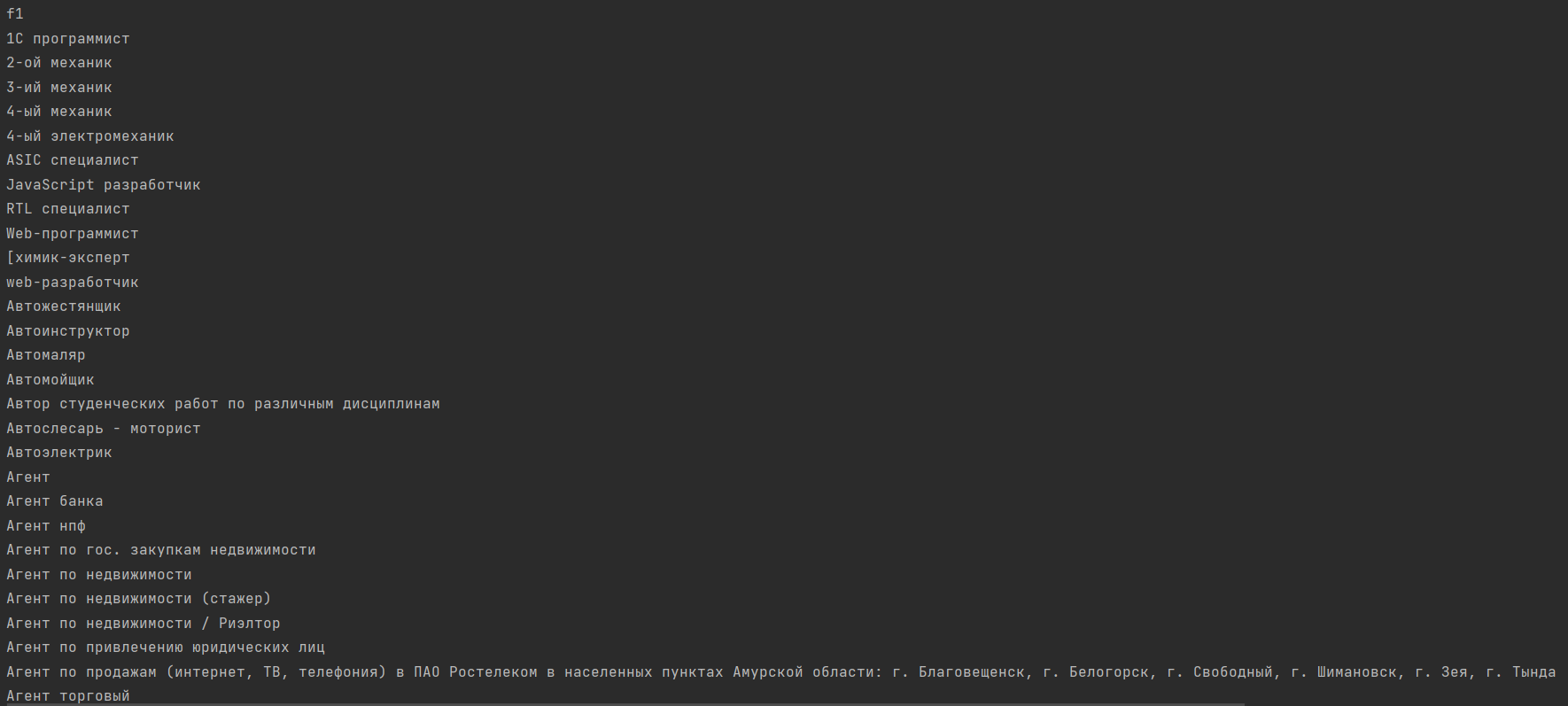
**print\_result.py**

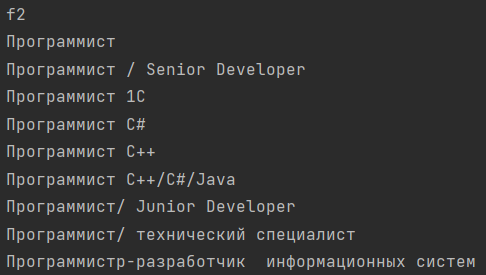


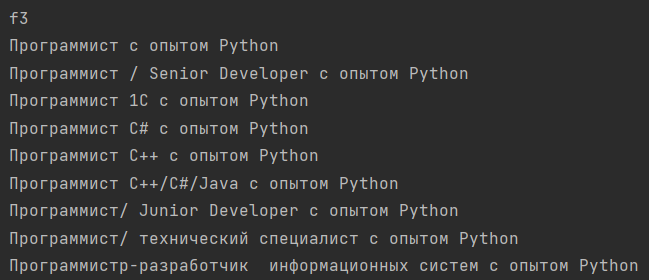
**cm\_timer.py**

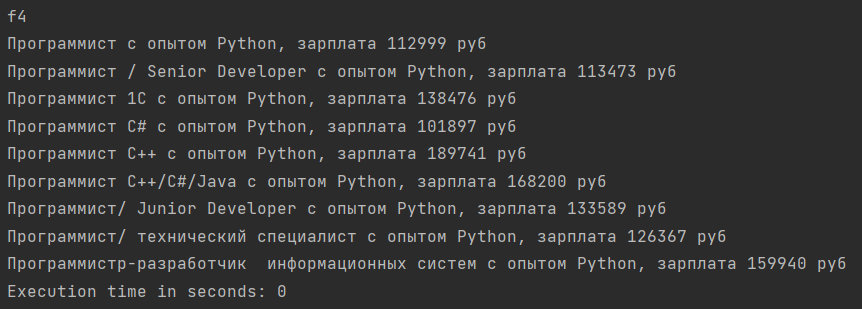
****

**process\_data.py**

****



****

****