Отчет №5-6

Cm\_timer.py

import time  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):  
 self.start\_time = time.time()  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):  
 end\_time = time.time()  
 print(f'time: {end\_time - self.start\_time}')

Field.py

def field(items, \*args):  
 assert len(args) > 0  
 for item in items:  
 if len(args) == 1:  
 # Если передан один ключ, выводим значения ключей  
 value = item.get(args[0])  
 if value is not None:  
 yield value  
 else:  
 # Если передано несколько ключей, возвращаем словарь  
 result = {key: item.get(key) for key in args if item.get(key) is not None}  
 if result:  
 yield result

Gen\_random.py

import random  
  
def gen\_random(num\_count, begin, end):  
 for \_ in range(num\_count):  
 yield random.randint(begin, end)

Print\_result.py

def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 result = func(\*args, \*\*kwargs)  
 print(func.\_\_name\_\_)  
  
 if isinstance(result, list):  
 for item in result:  
 print(item)  
 elif isinstance(result, dict):  
 for key, value in result.items():  
 print(f'{key} = {value}')  
 else:  
 print(result)  
 return result  
  
 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\_\_':  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

Sort.py

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

Unicque.py

class Unique:  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.items = items  
 self.ignore\_case = kwargs.get('ignore\_case', False)  
 self.seen = set()  
  
 def \_\_iter\_\_(self):  
 for item in self.items:  
 # Преобразуем в нижний регистр, если ignore\_case=True  
 check\_item = item.lower() if self.ignore\_case and isinstance(item, str) else item  
 if check\_item not in self.seen:  
 self.seen.add(check\_item)  
 yield item

Process\_data.py

import json  
import sys  
from cm\_timer import cm\_timer\_1  
from gen\_random import gen\_random  
from print\_result import print\_result  
from unique import Unique  
  
path = "D:/Загрузки Chrome/data\_light.json"  
  
with open(path, encoding='utf-8') as f:  
 data = json.load(f)  
  
def print\_result(func\_to\_decorate):  
 def decorated\_func(arg):  
 result = func\_to\_decorate(arg)  
 print()  
 if type(result) == list:  
 for i in result:  
 print(i)  
 elif type(result) == dict:  
 for i in result.keys():  
 print('{} = {}'.format(i, result[i]))  
 else:  
 print(result)  
 return result  
 return decorated\_func  
  
  
def field(items, \*args):  
 assert len(args) > 0  
 for i in range(len(items)):  
 if len(args) == 1:  
 if items[i][args[0]] != None:  
 yield "{}".format(items[i][args[0]])  
 else:  
 flag = True  
 map = dict()  
 for j in range(len(args)):  
 if items[i][args[j]] != None:  
 flag = False  
 map[args[j]] = items[i][args[j]]  
 if not flag:  
 yield map  
  
  
@print\_result  
def f1(arg):  
 return sorted(list(Unique(field(arg, "job-name"), ignore\_case=True)))  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda x: (x.startswith('программист') or x.startswith('Программист')), arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + ' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 return [i + ", зарплата {} руб.".format(j) for i, j in zip(arg, gen\_random(len(arg), 100\_000, 200\_000))]  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))