**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции

языков программирования»

Отчет по лабораторной работе №2

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Нестерова Екатерина |  | Гапанюк Ю.Е. |
|  |  |  |

Москва, 2024 г.

**Постановка задачи:**

Задание:

Задание лабораторной работы состоит из решения нескольких задач.

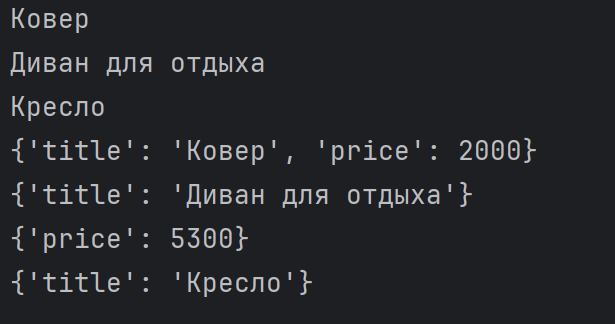
Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fp. Решение каждой задачи должно раполагаться в отдельном файле.

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.

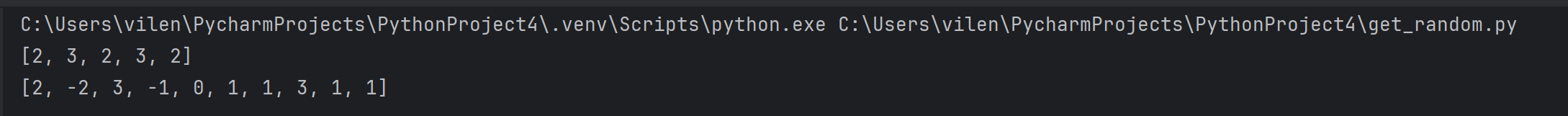
**Код:**goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'color': 'black'},  
 {'title': None, 'price': 5300, 'color': None},  
 {'title': 'Кресло', 'price': None, 'color': 'red'},  
 {'title': None, 'price': None, 'color': None}  
]  
def field(items, \*args):  
 assert len(args) > 0  
  
 if len(args) == 1:  
 field\_name = args[0]  
 for item in items:  
 value = item.get(field\_name)  
 if value is not None:  
 yield value  
 else:  
 for item in items:  
 result\_dict = {}  
 all\_none = True  
 for field\_name in args:  
 value = item.get(field\_name)  
 if value is not None:  
 result\_dict[field\_name] = value  
 all\_none = False  
 if not all\_none:  
 yield result\_dict  
  
  
def main():  
  
 for i in field(goods, 'title'):  
  
 print(i)  
  
 for i in field(goods, 'title', 'price'):  
  
 print(i)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 main()

****

**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.

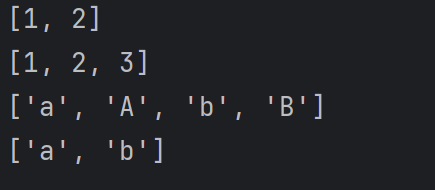
Код:  
import random  
  
def gen\_random(num\_count, begin, end):  
 for \_ in range(num\_count):  
 yield random.randint(begin, end)  
def main():  
 print(list(gen\_random(5, 1, 3)))  
  
 print(list(gen\_random(10, -5, 5)))  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 main()



**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.

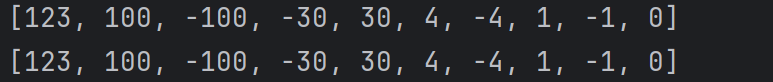
Код:  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.items = iter(items)  
 self.seen = set()  
 self.ignore\_case = kwargs.get('ignore\_case', False)  
  
 def \_\_next\_\_(self):  
 while True:  
 try:  
 item = next(self.items)  
 if self.ignore\_case and isinstance(item, str):  
 item = item.lower()  
 if item not in self.seen:  
 self.seen.add(item)  
 return item  
 except StopIteration:  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
  
  
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
print(list(Unique(data)))  
  
import random  
data = (random.randint(1, 3) for \_ in range(10)) # генератор  
print(list(Unique(data)))  
  
  
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
print(list(Unique(data)))  
print(list(Unique(data, ignore\_case=True)))



**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.

Код:  
data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Sorting without lambda function using `key=abs`  
 result = sorted(data, key=abs, reverse=True)  
 print(result)  
 # Sorting with a lambda function  
 result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)  
 print(result\_with\_lambda)



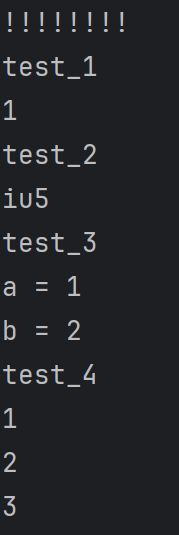
**Задача 5 (файл print\_result.py)**

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.

Код:

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, 3]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

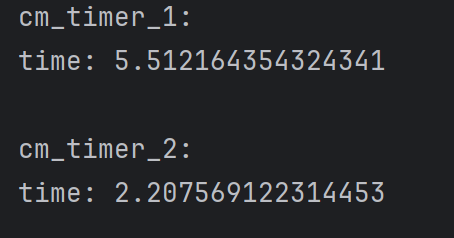


**Задача 6 (файл cm\_timer.py)**

Необходимо написать контекстные менеджеры cm\_timer\_1 и cm\_timer\_2, которые считают время работы блока кода и выводят его на экран.

Код:

import time  
from contextlib import contextmanager  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):  
 self.start = time.time()  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 elapsed = time.time() - self.start  
 print(f"time: {elapsed}")  
  
  
@contextmanager  
def cm\_timer\_2():  
 start = time.time()  
 try:  
 yield  
 finally:  
 elapsed = time.time() - start  
 print(f"time: {elapsed}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("cm\_timer\_1:")  
 with cm\_timer\_1():  
 time.sleep(5.5)   
  
 print("\ncm\_timer\_2:")  
 with cm\_timer\_2():  
 time.sleep(2.2)



**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.

Код:  
import json  
from cm\_timer import cm\_timer\_1  
from print\_result import print\_result  
from get\_random import gen\_random  
  
path = "C:/Users/vilen/PycharmProjects/PythonProject2/data\_light.json"  
with open(path, 'r', encoding='utf-8') as f:  
  
 data = json.load(f)  
  
  
# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`  
  
# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку  
  
# В реализации функции f4 может быть до 3 строк  
  
  
@print\_result  
  
def f1(arg):  
  
 return list(sorted(list(map(lambda x : x["job-name"], arg))))  
  
  
@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):  
  
 salary = zip(arg,gen\_random(len(arg),100000,200000))  
  
 return [f"{spec}, зарплата {sal} руб." for spec, sal in salary]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 with cm\_timer\_1():  
  
 f4(f3(f2(f1(data))))

