|  |  |
| --- | --- |
|  | **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 |

Отчет по лабораторной работе №3

Студент Компаниец Андрей Александрович

*фамилия, имя, отчество*

Группа ИУ5-51Б.

Студент 15.01.2022 **Компаниец А.А.**

*подпись, дата фамилия, и.о.*

Преподаватель   *подпись, дата фамилия, и.о.*

2022г.

Общее 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.

Задание №1:

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово аргументов.

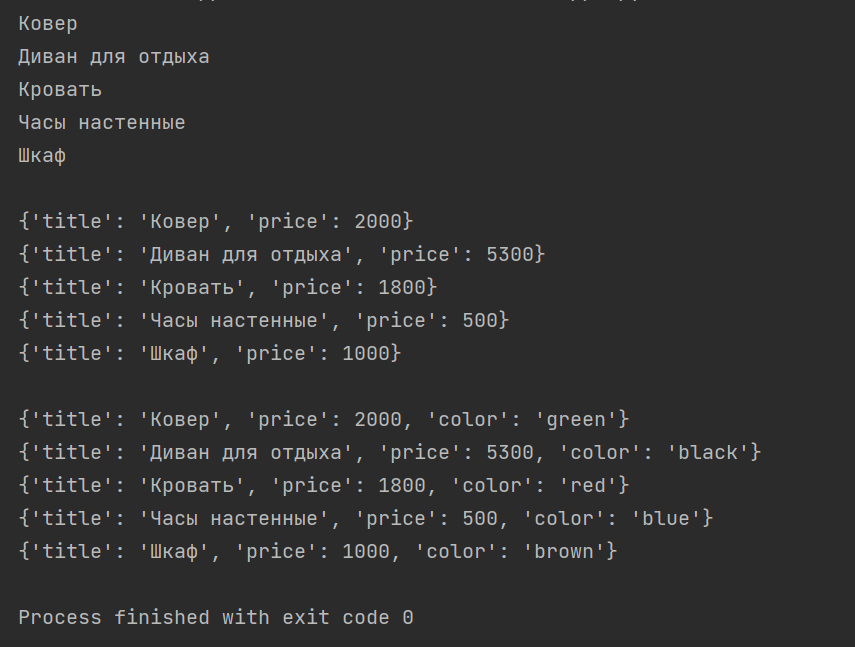
Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.

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  
 if len(args) == 1:  
 for i in range(0, len(items)):  
 print(items[i][args[0]])  
 else:  
 items\_1 = {}  
 for i in range(0, len(items)):  
 for j in range(0, len(args)):  
 items\_1[args[j]] = (items[i][args[j]])  
 print(items\_1)  
  
def main():  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},  
 {'title': 'Кровать', 'price': 1800, 'color': 'red'},  
 {'title': 'Часы настенные', 'price': 500, 'color': 'blue'},  
 {'title': 'Шкаф', 'price': 1000, 'color': 'brown'}  
 ]  
  
 field(goods, 'title')  
 print('')  
 field(goods, 'title', 'price')  
 print('')  
 field(goods, 'title', 'price', 'color')  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Скриншот:



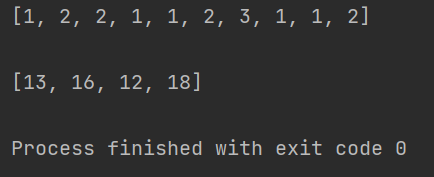
Задание №2:

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.

Текст программы:

import random  
  
def gen\_random(num\_count, begin, end):  
 rand = []  
 for i in range(0, num\_count):  
 rand.append(random.randint(begin, end))  
 # print(\*rand)  
 return(rand)  
  
def main():  
 print(gen\_random(10, 1, 3))  
 print('')  
 print(gen\_random(4, 5, 20))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Скриншот:



Задание №3:

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.

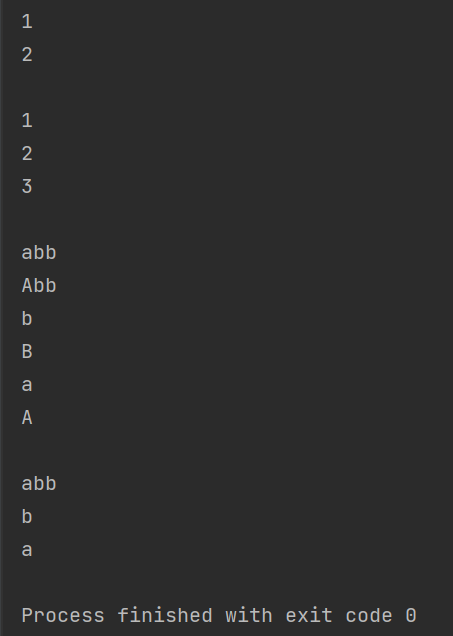
Итератор должен поддерживать работу как со списками, так и с генераторами.

Итератор не должен модифицировать возвращаемые значения.

Текст программы:

import random  
# import gen\_random  
  
def gen\_random(begin, end, num\_count):  
 rand = []  
 for i in range(0, num\_count):  
 rand.append(random.randint(begin, end))  
 return rand  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.elements = {}  
 ignore\_case = kwargs.get('ignore\_case')  
  
 if ignore\_case is None:  
 ignore\_case = False  
 elif not isinstance(ignore\_case, bool):  
 raise Exception("ignore\_case parameter is not Boolean")  
  
 for item in items:  
 if ignore\_case is True and isinstance(item, str):  
 item = item.lower()  
  
 self.elements[item] = item  
  
 def \_\_next\_\_(self):  
 return next(self.elements)  
  
 def \_\_iter\_\_(self):  
 return iter(self.elements)  
   
def main():  
 data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 data2 = gen\_random(1, 3, 10)  
 data3 = ['abb', 'Abb', 'b', 'B', 'a', 'A', 'b', 'B']  
  
 for val in Unique(data1):  
 print(val)  
 print('')  
  
 for val in Unique(data2):  
 print(val)  
 print('')  
  
 for val in Unique(data3, ignore\_case=False):  
 print(val)  
 print('')  
  
 for val in Unique(data3, ignore\_case=True):  
 print(val)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Скриншот:



Задание №4:

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.

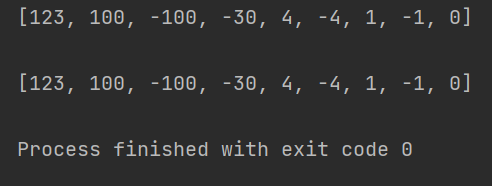
Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

Текст программы:

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

Скриншот:



Задание №5:

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.

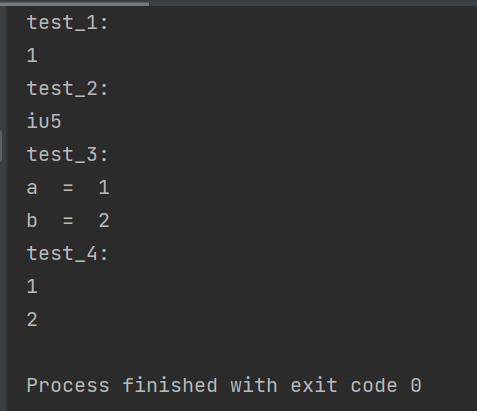
Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.

Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

Текст программы:

def print\_result(func):  
 def pr(\*args):  
 func(\*args)  
 print(func.\_\_name\_\_ + ":")  
 a = func(\*args)  
 if type(a) is dict:  
 for key, value in a.items():  
 print(key, ' = ', value)  
 elif type(a) is list:  
 for i in range(0, len(a)):  
 print(a[i])  
 else:  
 print(a)  
 return a  
 return pr  
  
  
@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(b):  
 return b  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('!!!!!!!!')  
 b = [1, 2]  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4(b)

Скриншот:



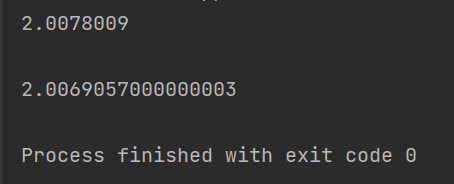
Задание №6:

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

Текст программы:

from contextlib import contextmanager  
  
class TimerError(Exception):  
 pass  
  
class cm\_timer\_1:  
 def \_\_init\_\_(self):  
 self.\_start\_time = None  
  
 def start(self):  
 if self.\_start\_time is not None:  
 raise TimerError(f"Таймер уже работает. Используйте .stop() чтобы его остановить")  
  
 self.\_start\_time = time.perf\_counter()  
  
 def stop(self):  
 if self.\_start\_time is None:  
 raise TimerError(f"Таймер не работает. Используйте .start() для его запуска")  
  
 elapsed\_time = time.perf\_counter() - self.\_start\_time  
 self.\_start\_time = None  
 print(elapsed\_time)  
  
 def \_\_enter\_\_(self):  
 self.start()  
 return self  
  
 def \_\_exit\_\_(self, \*exc\_info):  
 self.stop()  
  
@contextmanager  
def cm\_timer\_2() -> float:  
 start = time.perf\_counter()  
 yield lambda: time.perf\_counter() - start  
 #Yield это ключевое слово, которое используется примерно как return —  
 # отличие в том, что функция вернёт генератор.  
 print(time.perf\_counter() - start)  
  
with cm\_timer\_1():  
 time.sleep(2)  
  
print('')  
  
with cm\_timer\_2():  
 time.sleep(2)

Скриншот:



Задание №7:

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.

Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр.

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”.

Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности.

Текст программы:

import json  
  
from cm\_timer import cm\_timer\_1  
from gen\_random import gen\_random  
from print\_result import print\_result  
from unique import Unique  
  
path = './data\_light.json'  
  
with open(path, encoding='utf-8') as f:  
 data = json.load(f)  
  
@print\_result  
def f1(data):  
 return sorted((Unique((i["job-name"] for i in data), ignore\_case = True)), key=lambda x: x.lower())  
  
@print\_result  
def f2(data):  
 return list(filter(lambda x: x.startswith('программист'), data))  
  
@print\_result  
def f3(data):  
 return list(map(lambda x: x + ' с опытом Python', data))  
  
@print\_result  
def f4(data):  
 data\_sal = zip(data, gen\_random(len(data), 100000, 200000))  
 return list(map(lambda x: x[0] + ', зарплата: ' + str(x[1]) + 'руб', data\_sal))  
  
def main():  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Скриншот:

