Московский государственный технический

университет им. Н.Э. Баумана

Факультет «Радиотехнический»

Кафедра ИУ5 «Информатика и вычислительная техника»

**Курс «Разработка интернет-приложений»**

**Отчет по лабораторной работе № 3**

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы РТ5-51 |  | преподаватель |
| Незаметдинов Т. И. |  | Балашов А. М. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2021 г.

Общее описание задания

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. 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 количество аргументов.

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

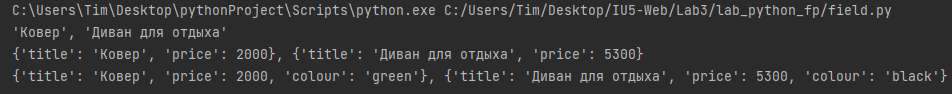
Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно None, то оно пропускается. Если все поля содержат значения None, то пропускается элемент целиком.

Текст программы

fields.py

goods = [  
 {'title': 'Ковер', 'price': 2000, 'colour': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'colour': 'black'}  
]  
  
  
def field(items, \*args):  
 try:  
 assert len(items) != 0 and len(args) != 0  
 except:  
 print("Вы не передали аргументы")  
 return\_list = []  
 for item in items:  
 temp\_dict = {}  
 if len(args) == 1:  
 for key, value in item.items():  
 if key in args and value != None:  
 return\_list.append(value)  
 else:  
 for key, value in item.items():  
 if key in args and value != None:  
 temp\_dict[key] = value  
 if len(temp\_dict) > 0:  
 return\_list.append(temp\_dict)  
  
 return return\_list  
  
  
def main():  
 print(str(field(goods, 'title'))[1:-1])  
 print(str(field(goods, 'title', 'price'))[1:-1])  
 print(str(field(goods, 'title', 'price', 'colour'))[1:-1])  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Экранные формы с примерами выполнения программы



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. Пример:

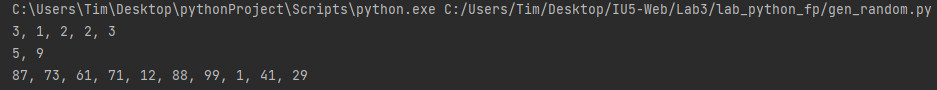
gen\_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

Текст программы

gen\_random.py

from random import randint  
  
def gen\_random(count, begin, end):  
 try:  
 assert count > 0  
 except:  
 print("Вы не указали кол-во чисел")  
 pass  
  
 result\_list = [randint(begin, end) for index in range(count)]  
 return result\_list  
  
def main():  
 print(str(gen\_random(5, 1, 3))[1:-1])  
 print(str(gen\_random(2, 0, 10))[1:-1])  
 print(str(gen\_random(10, 0, 99))[1:-1])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Экранные формы с примерами выполнения программы



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.

Пример:

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

Unique(data) будет последовательно возвращать только 1 и 2.

data = gen\_random(1, 3, 10)

Unique(data) будет последовательно возвращать только 1, 2 и 3.

data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’]

Unique(data) будет последовательно возвращать только a, A, b, B.

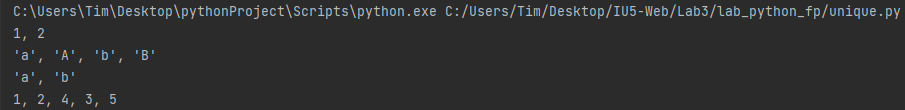
Unique(data, ignore\_case=True) будет последовательно возвращать только a, b.

Текст программы

unique.py

from random import randint  
from gen\_random import gen\_random  
  
class Unique:  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.index = 0  
 self.data = list(items)  
 self.unique\_list = set()  
  
 if 'ignore\_case' in kwargs.keys():  
 self.ignore\_case = kwargs['ignore\_case']  
 else:  
 self.ignore\_case = False  
  
 def \_\_next\_\_(self):  
 while True:  
 if self.index >= len(self.data):  
 raise StopIteration  
  
 current = self.data[self.index]  
 self.index += 1  
  
 if self.ignore\_case == True:  
 if current not in self.unique\_list:  
 if type(current) == str:  
 self.unique\_list.add(current)  
 return current  
 else:  
 self.unique\_list.add(current)  
 return current  
 elif self.ignore\_case == False:  
 if type(current) == str:  
 if current.upper() not in self.unique\_list and current.lower() not in self.unique\_list:  
 self.unique\_list.add(current.lower())  
 self.unique\_list.add(current.upper())  
 return current  
 else:  
 if current not in self.unique\_list:  
 self.unique\_list.add(current)  
 return current  
  
 def \_\_iter\_\_(self):  
 return self  
  
data\_int = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
data\_str = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
  
def main():  
 print(str(list(Unique(data\_int)))[1:-1])  
 print(str(list(Unique(data\_str, ignore\_case = True)))[1:-1])  
 print(str(list(Unique(data\_str, ignore\_case = False)))[1:-1])  
 print(str(list(Unique(gen\_random(100, 1, 5))))[1:-1])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Экранные формы с примерами выполнения программы



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. 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:

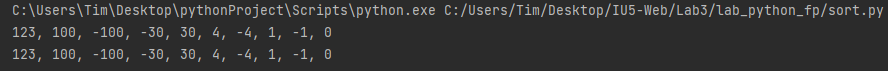
1. С использованием lambda-функции.
2. Без использования lambda-функции.

Текст программы

sort.py

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]  
  
def sort(data):  
 return sorted(data, key=abs, reverse=True)  
  
def sort\_with\_lambda(data):  
 return sorted(data, key=lambda value: value if value > 0 else value\*-1, reverse=True)  
  
def main():  
 print(str(sort(data))[1:-1])  
 print(str(sort\_with\_lambda(data))[1:-1])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Экранные формы с примерами выполнения программы



Задание 5

Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

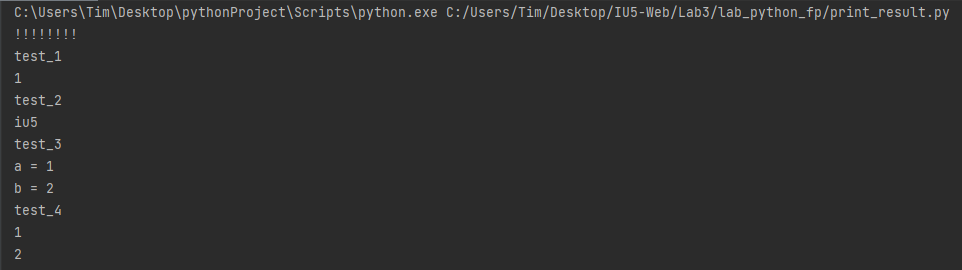
* Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.
* Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.
* Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

Текст программы

print\_result.py

def print\_result(func):  
 def decorated\_func(\*args):  
 print(func.\_\_name\_\_)  
 return\_value = func(\*args)  
 if type(return\_value) == list:  
 for value in return\_value:  
 print (value)  
 elif type(return\_value) == dict:  
 for key, value in return\_value.items():  
 print(f'{key} = {value}')  
 else:  
 print(return\_value)  
 return return\_value  
 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]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

Экранные формы с примерами выполнения программы



Задание 6

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).

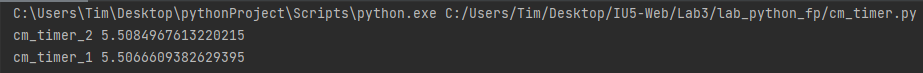
cm\_timer\_1 и cm\_timer\_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки contextlib).

Текст программы

cm\_timer.py

import time  
from contextlib import contextmanager  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):  
 self.start = time.time()  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 print(cm\_timer\_1.\_\_name\_\_, time.time() - self.start)  
  
@contextmanager  
def cm\_timer\_2():  
 start = time.time()  
 yield  
 print(cm\_timer\_2.\_\_name\_\_, time.time() - start)  
  
def main():  
 with cm\_timer\_2():  
 time.sleep(5.5)  
 with cm\_timer\_1():  
 time.sleep(5.5)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Экранные формы с примерами выполнения программы



Задание 7

* В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.
* В файле 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

process\_data.py

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

Экранные формы с примерами выполнения программы

