*Федеральное государственное бюджетное образовательное учреждение* *высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(МГТУ им. Н.Э. Баумана)*** |

Факультет Радиотехнический

Кафедра ИУ5

**Отчет**

**по лабораторной работе № 2**

**Дисциплина: Разработка интернет приложений**

**Подготовил:**

**Студент группы РТ5-51Б**

**Курьянов А.И.**

Москва, 2019

Задание:

Важно выполнять все задачи последовательно. С 1 по 5 задачу формируется модуль librip, с помощью которого будет выполняться задание 6 на реальных данных из жизни. Весь вывод на экран (даже в столбик) необходимо запрограммировать одной строкой.

Задача 1 (ex\_1.py)

Необходимо реализовать генераторы field и gen\_random

Генератор field последовательно выдает значения ключей словарей массива

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'}

1. В качестве первого аргумента генератор принимает list, дальше через \*args генератор принимает неограниченное кол-во аргументов.
2. Если передан один аргумент, генератор последовательно выдает только значения полей, если поле равно None, то элемент пропускается
3. Если передано несколько аргументов, то последовательно выдаются словари, если поле равно None, то оно пропускается, если все поля None, то пропускается целиком весь элемент

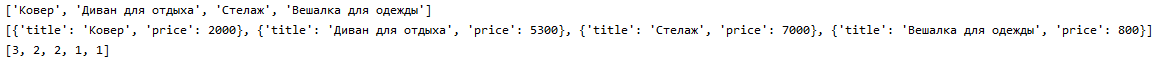
Генератор gen\_random последовательно выдает заданное количество случайных чисел в заданном диапазоне

Пример:

gen\_random(1, 3, 5)должен выдать 5 чисел от 1 до 3, т.е. примерно 2, 2, 3, 2, 1

Итоговый код и вывод:

from librip.gen import field  
from librip.gen import gen\_random  
  
goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},  
 {'title': 'Стелаж', 'price': 7000, 'color': 'white'},  
 {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}  
]  
# Реализация задания 1  
print(list(field(goods, 'title')))  
print(list(field(goods, 'title', 'price')))  
print(list(gen\_random(1, 3, 5)))



Задача 2 (ex\_2.py)

Необходимо реализовать итератор, который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты. Конструктор итератора также принимает на вход именной bool-параметр ignore\_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False. 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(gen\_random(1, 3, 10))будет последовательно возвращать только 1, 2 и 3

data = [‘a’, ‘A’, ‘b’, ‘B’]

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

data = [‘a’, ‘A’, ‘b’, ‘B’]

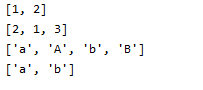
Unique(data, ignore\_case=True) будет последовательно возвращать только a, b

В ex\_2.py нужно вывести на экран то, что они выдают *одной строкой*. **Важно** продемонстрировать работу как с массивами, так и с генераторами (gen\_random).

Итератор должен располагаться в librip/iterators.py

Итоговый код и вывод:

from librip.gen import gen\_random  
from librip.iterators import Unique  
# Реализация задания 2  
  
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
print(list(Unique(data))) #будет последовательно возвращать только 1 и 2  
  
data = gen\_random(1, 3, 10)  
print(list(Unique(gen\_random(1, 3, 10)))) #будет последовательно возвращать только 1, 2 и 3  
  
data = ["a", "A", "b", "B"]  
print(list(Unique(data))) #будет последовательно возвращать только a, A, b, B  
  
data = ["a", "A", "b", "B"]  
print(list(Unique(data, ignore\_case=True))) #будет последовательно возвращать только a, b



Задача 3 (ex\_3.py)

Дан массив с положительными и отрицательными числами. Необходимо одной строкой вывести на экран массив, отсортированный по модулю. Сортировку осуществлять с помощью функции sorted

Пример:

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

Вывод: [0, 1, -1, 4, -4, -30, 100, -100, 123]

Итоговый код и вывод:

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
# Реализация задания 3  
#data = sorted(data, key=abs())  
data = sorted(data, key=lambda x: abs(x)) #key- функция, вызываемая на каждом элементе перед сравнением, то есть функция модуля  
print(data)



Задача 4 (ex\_4.py)

Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции. Файл ex\_4.py **не нужно** изменять.

Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции, печатать результат и возвращать значение.

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

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

Итоговый код и вывод:

from librip.decorators import print\_result  
  
# Необходимо верно реализовать print\_result  
# и задание будет выполнено  
  
@print\_result  
def test\_1():  
 return 1  
  
  
@print\_result  
def test\_2():  
 return 'YAP'  
  
  
@print\_result  
def test\_3():  
 return {'a': 1, 'b': 2}  
  
  
@print\_result  
def test\_4():  
 return [1, 2]  
  
  
test\_1()  
test\_2()  
test\_3()  
test\_4()

test\_1

1

test\_2

YAP

test\_3

a = 1

b = 2

test\_4

1

2

Декоратор должен располагаться в librip/decorators.py

Задача 5 (ex\_5.py)

Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран

Пример:

with timer():

    sleep(5.5)

После завершения блока должно вывестись в консоль примерно 5.5

Итоговый код:

from time import sleep  
from librip.сontextmanager import timer  
  
with timer():  
 sleep(5.5)

Задача 6 (ex\_6.py)

Мы написали все инструменты для работы с данными. Применим их на реальном примере, который мог возникнуть в жизни. В репозитории находится файл data\_light.json. Он содержит облегченный список вакансий в России в формате json (ссылку на полную версию размером ~ 1 Гб. в формате xml можно найти в файле README.md).

Структура данных представляет собой массив словарей с множеством полей: название работы, место, уровень зарплаты и т.д.

В ex\_6.py дано 4 функции. В конце каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print\_result печатается результат, а контекстный менеджер timer выводит время работы цепочки функций.

Задача реализовать все 4 функции по заданию, ничего не изменяя в файле-шаблоне. Функции f1-f3 должны быть реализованы в 1 строку, функция f4 может состоять максимум из 3 строк.

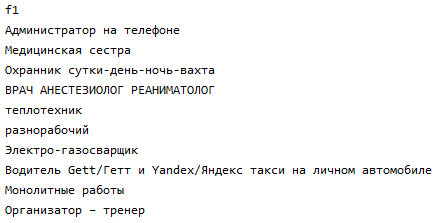
Что функции должны делать:

1. Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна **игнорировать регистр**. Используйте наработки из предыдущих заданий.
2. Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Иными словами нужно получить все специальности, связанные с программированием. Для фильтрации используйте функцию filter.
3. 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.

Итоговый код и вывод:

import json  
import sys  
from librip.сontextmanager import timer  
from librip.decorators import print\_result  
from librip.gen import field, gen\_random  
from librip.iterators import Unique  
  
path = "data\_light.json"  
import json  
import sys  
# Здесь необходимо в переменную path получить  
# путь до файла, который был передан при запуске  
  
with open(path, encoding="utf8") as f: #Без кодировки не работает  
 data = json.load(f)  
  
  
# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`  
# Важно!  
# Функции с 1 по 3 дожны быть реализованы в одну строку  
# В реализации функции 4 может быть до 3 строк  
# При этом строки должны быть не длиннее 80 символов  
  
@print\_result  
def f1(arg):  
 return list(Unique(list(field(arg, "job-name")), ignore\_case=True))  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda s: "программист" in s[0:12], arg))  
  
  
@print\_result  
def f3(arg): # map(func, arr)  
 return list(map(lambda s: s + " с опытом Python", arg))  
  
  
@print\_result  
def f4(arg):  
 Sal = gen\_random(100000, 200000, len(arg))  
 return list(map(lambda s: '{}, зарплата {} руб.'.format(  
 s[0], s[1]), zip(arg, Sal)))  
  
  
with timer():  
 f4(f3(f2(f1(data))))



Модуль LIBRIP:

Реализация декоратора:

def print\_result(fn): #Получаем функцию  
 def inside(\*args): #Получаем аргументы  
 print(fn.\_\_name\_\_) #Выводим название функции  
 if len(args) == 0: #Для вызова функции без аргументов  
 fun = fn()  
 else: #Дле нескольких аргументов  
 fun = fn(args[0])  
 if type(fun) == list: #Если список  
 for i in fun:  
 print(i)  
 elif type(fun) == dict: #Если словарь  
 for i in fun:  
 print(i, "=", fun[i])  
 else:  
 print(fun) #Иначе выводим функцию  
 return fun  
 return inside

Реализация генератора списка случайных чисел:

import random  
  
def field(items, \*args):  
 assert len(args) > 0 # Проверка на ошибку  
 if len(args) == 1: #Если передан один аргумент  
 for i in items:  
 for a in args:  
 yield i[a]  
 else: # Для нескольких аргументов  
 for i in items: # Для каждого словаря  
 dict = {} # Создаем новый словарь  
 for arg in args:  
 if arg in i is not None:  
 dict[arg] = i[arg] # Заполняем словарь  
 if len(dict) > 0 and len(args) > 1:  
 yield dict  
  
  
# Генератор списка случайных чисел  
# Пример:  
# gen\_random(1, 3, 5) должен выдать примерно 2, 2, 3, 2, 1  
# Hint: реализация занимает 2 строки  
def gen\_random(begin, end, num\_count):  
 for \_ in range(num\_count):  
 yield random.randint(begin, end)

Реализация итератора:

from types import GeneratorType  
# Итератор для удаления дубликатов  
class Unique(object):  
 IGNORE\_CASE = False  
 INDEX = 0  
 OBJECTS = []  
 VOID = []  
  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 if 'ignore\_case' in kwargs.keys(): #Проверка на регистр  
 self.IGNORE\_CASE = kwargs['ignore\_case']  
 if type(items == GeneratorType): #Для генератора  
 self.OBJECTS = list(items)  
 else:  
 self.OBJECTS = items  
 # self.ITEMS = len(items)  
  
 def \_\_next\_\_(self):  
 while True:   
 if self.INDEX == (len(self.OBJECTS) - 1):  
 raise StopIteration  
 self.INDEX += 1  
  
 val = self.OBJECTS[self.INDEX]  
 val2 = str(val).lower()  
 if self.IGNORE\_CASE: #Если регистр игнорируется, то ставим второе значение  
 val = val2  
 if val not in self.VOID: #Если не находится значение в списке, туда его добавляем  
 self.VOID.append(val)  
 return val  
  
 def \_\_iter\_\_(self): #Возвращает итератор для объекта  
 del self.VOID[:]  
 self.INDEX = -1  
 return self

Реализация контекстного менеджера timer:

import time  
# Здесь необходимо реализовать  
# контекстный менеджер timer  
# Он не принимает аргументов, после выполнения блока он должен вывести время выполнения в секундах  
# Пример использования  
# with timer():  
# sleep(5.5)  
#  
# После завершения блока должно вывестись в консоль примерно 5.5  
  
class timer:  
 def \_\_enter\_\_(self):  
 self.start = time.time() #Начало подсчета  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 ti = (time.time()) - self.start #Получаем общее время как время- время начала  
 print(ti)