|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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)** |

**Лабораторная работа №2**

**По курсу “Разработка интернет приложений”**

**Группа: РТ5-51Б**

**Выполнила: Трофимова Ю.О.**

**Задача 1.**

Задание:

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

В ex\_1.py нужно вывести на экран то, что они выдают, с помощью кода в *одну строку*

Генераторы должны располагаться в librip/gen.py

Код:

Файл gens.py

**import** random  
**def** field(items, \*args):  
 **assert** len(args) > 0  
 **if** len(args) == 1:  
 **for** item **in** items:  
 **if** item[args[0]]:  
 **yield** item[args[0]]  
 **else**:  
 **for** item **in** items:  
 **yield** {arg: item[arg] **for** arg **in** args **if** item[arg]}

**def** gen\_random(begin, end, num\_count):  
 res =[]  
 **for** i **in** range(num\_count):  
 **yield** random.randint(begin, end)

Файл ex\_1.py

*#!/usr/bin/env python3***from** librip.gens **import** field  
  
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'**,**'price'**)))

Результат работы:

[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стелаж', 'price': 7000}, {'title': 'Вешалка для одежды', 'price': 800}]

**Задача 2.**

Задание:

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

Код:

Файл iterators.py

**class** Unique(object):  
 **def** \_\_init\_\_(self, items, \*\*kwargs):

self.unique = []  
 **if 'ignore\_case' not in** kwargs.keys():  
 self.ignore\_case = **False  
 else**:  
 self.ignore\_case = kwargs[**'ignore\_case'**]  
 self.items = iter(items)  
  
 **def** \_\_next\_\_(self):  
 **while True**:  
 item = self.items.\_\_next\_\_()  
 compare\_item = **None  
  
 if** self.ignore\_case **and** type(item) **is** str:  
 compare\_item = item.lower()  
 **else**:  
 compare\_item = item  
  
 **if** compare\_item **not in** self.unique:  
 self.unique.append(compare\_item)  
 **return** item  
  
 **def** \_\_iter\_\_(self):  
 **return** self

Файл ex\_2.py

*#!/usr/bin/env python3***from** librip.gens **import** gen\_random  
**from** librip.iterators **import** Unique  
  
data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
data2 = gen\_random(1, 3, 10)  
  
*# Реализация задания 2*data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
data2 = gen\_random(1, 3, 10)  
print(\*Unique(data1))  
print(\*Unique(data2))

Результат работы:

1 2

2 1 3

**Задача 3.**

Задание:

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]

Код:

Файл ex\_3.py

*#!/usr/bin/env python3***import** math  
  
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
*# Реализация задания 3*print(sorted(data, key=**lambda** x: abs(x)))

Результат работы:

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

**Задача 4.**

Задание:

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

Декоратор должен располагаться в librip/decorators.py

Код:

Файл decorators.py

**def** print\_result(func):  
 **def** wrapper(\*arg):  
 res = func(\*arg)  
 print(func.\_\_name\_\_)  
 **if** type(res) **is** list:  
 **for** i **in** res:  
 print(i)  
 **elif** type(res) **is** dict:  
 **for** i, j **in** res.items():  
 print(**'{} = {}'**.format(i,j))  
 **else**:  
 print(res)  
 **return** res  
 **return** wrapper

Файл ex\_4.py

**from** librip.decorators **import** print\_result  
@print\_result  
**def** test\_1():  
 **return** 1  
  
  
@print\_result  
**def** test\_2():  
 **return 'iu'**@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()

Результат работы:

test\_1

1

test\_2

iu

test\_3

a = 1

b = 2

test\_4

1

2

**Задача 5.**

Задание:

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

Код:

Файл ctxmngrs.py

**import** time  
  
**class** timer():  
 **def** \_\_init\_\_(self):  
 self.start = time.time()  
  
 **def** \_\_enter\_\_(self):  
 **pass  
  
 def** \_\_exit\_\_(self, \*args):  
 print (time.time() - self.start)

Файл ex\_5.py

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

Результат работы:

5.500298023223877

**Задача 6.**

Задание:

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.
4. 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:

Файл ex\_6.py

*#!/usr/bin/env python3***import** json  
**import** sys  
**from** librip.ctxmngrs **import** timer  
**from** librip.decorators **import** print\_result  
**from** librip.gens **import** field, gen\_random  
**from** librip.iterators **import** Unique **as** unique  
  
path = **'D:\Программирование\lab2\_rip\data\_light.json'****with** open(path, encoding=**"utf8"**) **as** f:  
 data = json.load(f)  
@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: s.lower().startswith(**"программист"**), arg))  
  
  
@print\_result  
**def** f3(arg):  
 **return** list(map(**lambda** s: s + **" с опытом Python"**, arg))  
  
  
@print\_result  
**def** f4(arg):  
 salary = gen\_random(100000, 200000, len(arg))  
 **return** list(map(**lambda** s: **'{}, зарплата {} руб.'**.format(  
 s[0], s[1]), zip(arg, salary)))  
  
  
**with** timer():  
 f4(f3(f2(f1(data))))

Результат работы:

f1

Администратор на телефоне

Медицинская сестра

Охранник сутки-день-ночь-вахта

ВРАЧ АНЕСТЕЗИОЛОГ РЕАНИМАТОЛОГ

теплотехник

разнорабочий

Электро-газосварщик

Водитель Gett/Гетт и Yandex/Яндекс такси на личном автомобиле

Монолитные работы

…

f2

Программист

Программист С++/С#/Java

Программист 1С

Программистр-разработчик информационных систем

Программист С++

Программист/ Junior Developer

Программист / Senior Developer

Программист/ технический специалист

Программист C#

f3

Программист с опытом Python

Программист С++/С#/Java с опытом Python

Программист 1С с опытом Python

Программистр-разработчик информационных систем с опытом Python

Программист С++ с опытом Python

Программист/ Junior Developer с опытом Python

Программист / Senior Developer с опытом Python

Программист/ технический специалист с опытом Python

Программист C# с опытом Python

f4

Программист с опытом Python, зарплата 148223 руб.

Программист С++/С#/Java с опытом Python, зарплата 165015 руб.

Программист 1С с опытом Python, зарплата 161130 руб.

Программистр-разработчик информационных систем с опытом Python, зарплата 114569 руб.

Программист С++ с опытом Python, зарплата 151791 руб.

Программист/ Junior Developer с опытом Python, зарплата 180131 руб.

Программист / Senior Developer с опытом Python, зарплата 123209 руб.

Программист/ технический специалист с опытом Python, зарплата 174323 руб.

Программист C# с опытом Python, зарплата 101299 руб.

0.07114315032958984