Московский государственный технический университет им. Н.Э.Баумана

Отчет по лабораторной работе №4
по курсу «Разработка Интернет-приложений»

Вариант № <21>

ИСПОЛНИТЕЛЬ:
студент группы ИУ5-52
Ромичева Е.В.
09.09.2016 г.

Москва, МГТУ - 2016

Содержание

Описание задания лабораторной работы

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

В ex_1.py нужно вывести на экран то, что они выдают одной строкой

Генераторы должны располагаться в librip/gen.py

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

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

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]

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

Пример:

```
@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()
```

На консоль выведется:

```
test_1
1
test_2
iu
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

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.

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.

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

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.

Исходный код

ex_1.py

```
from librip.gen import field, gen_random

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
    {'title': None, 'price': 800, 'color': 'white'}
]

# Реализация задания 1
print(list(field(goods,'title1')))
print(list(field(goods,'title','color')))
print(list(gen_random(1,3,5)))
```

ex_2.py

```
from librip.gen import gen_random
from librip.iterators import Unique

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 5, 8)
data3 = ['A','a','b','v','B','A']
```

```
# Реализация задания 2

print(list(Unique(data1)))
print(list(Unique(list(data2))))
print(list(Unique(data3, ignore_case=True)))
```

ex_3.py

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3
  print(sorted(data, key=lambda x: abs(x), data))
```

ex_4.py

```
# Необходимо верно реализовать print_result
# и задание будет выполнено

@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()
```

ex_5.py

```
from time import sleep
from librip.ctxmngrs import timer

with timer():
    sleep(5.5)
```

ex_6.py

```
#!/usr/bin/env python3
import json
```

```python
import sys
import os.path
from librip.ctxmngrs import timer
from librip.decorators import print_result
from librip.gen import field, gen_random
from librip.iterators import Unique as unique

path = path = "data_light.json"

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

with open(path, encoding="utf8") as f:
    data = json.load(f)
    #print(type(data))


# 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 sorted(unique(field(arg, 'job-name'), ignore_case=True), key=lambda x: x.lower())


@print_result
def f2(arg):
    #return list(filter(lambda x: 'программист' in str(x),arg)) #все, что содержит 'программист'
    return list(filter(lambda x: str(x).startswith('Программист'),arg)) #все, что начинается с 'программист'


@print_result
def f3(arg):
    return list(map(lambda x: str(x)+' с опытом Python', arg))


@print_result
def f4(arg):
    res = gen_random(100000, 200000, len(arg))
    #return list(map(lambda x: x+', зарплата '+str(next(res)) + ' руб.', arg))
    return list(x+', зарплата '+str(next(res)) + ' руб.' for x in arg)
```

```
with timer():
      f4(f3(f2(f1(data))))
```

ctxmngrs.py

```
import  time

class timer(object):
    def __enter__(self):
        self._startTime = time.time()

    def __exit__(self, type, value, traceback):
        #print ('my time = ', format(float(time.time())-
self._startTime))
          print ('my time = ', float(time.time())-self._startTime)
```

decorators.py

```
def print_result(func):
    def decorated_func(*args):
        print(func.__name__)
        res = func(*args)
        if type(res) is list:
            print('\n'.join(str(i) for i in res))
            # for i in res:
            #     print (i, sep = '\n')
        elif type(res) is dict:
            # for i in res:
            #     print (i, res[i], sep = ' = ')
            print('\n'.join(str(i) + '=' + str(res[i]) for i in res))

        else:
            print(res)
        return (res)
      return decorated_func
```

iterators.py

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = iter(items) if isinstance(items, list) else items
        self.num = 0
        self.fl=False
        self.unique=[]
        if (kwargs.get('ignore_case')==True):
           #self.items=[str(it).lower() for it in items]
           self.fl=True

    def __next__(self):
        # Нужно реализовать __next__
        for it in self.items:
            if (self.fl==True and type(it)!=int):
```

```
                if (it.lower() not in self.unique):
                    buf=str(it).lower()
                    self.unique.append(str(buf))
                    return it
            else:
                if (it not in self.unique):
                    self.unique.append(it)
                    return it
        raise StopIteration()


    def __iter__(self):
        return self
```

gen.py

```
from random import  randint
def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
    if  len(args)==1:
        for it in items:
            if (it.get(args[0])):
                yield it[args[0]]
    else:
        for it in items:
            b={}
            for arg in args:
                if (it.get(arg)):   # если не None
                    b[arg]=it.get(arg)

            if len(b)>0:
                yield b

def gen_random(begin, end, num_count):
    pass
    # Необходимо реализовать генератор
    for i in range(num_count):
        yield randint(begin,end)
```

Результат работы

ex_1.py

```
[]
[{'title': 'Ковер', 'color': 'green'}, {'title': 'Диван для отдыха',
'color': 'black'}, {'title': 'Стелаж', 'color': 'white'}, {'color':
'white'}]
[3, 3, 2, 1, 2]
```

ex_2.py

```
[1, 2]
```

```
[5, 4, 1]
['A', 'b', 'v']
```

ex_3.py

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

ex_4.py

```
test_1
1
test_2
iu
test_3
b=2
a=1
test_4
1
2
```

ex_5.py

```
my time =  5.500744104385376
```

ex_6.py

```
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
автослесарь
Автослесарь – моторист
. . .
Юрисконсульт. Контрактный управляющий
Юрист
Юрист (специалист по сопровождению международных договоров, английский
- разговорный)
Юрист волонтер
Юристконсульт
f2
Программист
Программист / Senior Developer
```

Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик информационных систем
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С с опытом Python
Программист C# с опытом Python
Программист C++ с опытом Python
Программист C++/C#/Java с опытом Python
Программист/ Junior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программистр-разработчик информационных систем с опытом Python
f4
Программист с опытом Python, зарплата 192871 руб.
Программист / Senior Developer с опытом Python, зарплата 103406 руб.
Программист 1С с опытом Python, зарплата 196091 руб.
Программист C# с опытом Python, зарплата 129945 руб.
Программист C++ с опытом Python, зарплата 151179 руб.
Программист C++/C#/Java с опытом Python, зарплата 182901 руб.
Программист/ Junior Developer с опытом Python, зарплата 140093 руб.
Программист/ технический специалист с опытом Python, зарплата 130464 руб.
Программистр-разработчик информационных систем с опытом Python, зарплата 177696 руб.
my time = 0.30472302436828613