Защищено:

Гапанюк Ю. Е.

"__"_____2016 г.

Демонстрация:

Гапанюк Ю. Е.

"__"_____2016 г.

Отчет по лабораторной работе №4
по курсу «Разработка интернет приложений»

10

(количество листов)

ИСПОЛНИТЕЛЬ:

Студент группы ИУ5-54 _____

Повираева М. Л. (подпись)

"__"_____2016 г.

Москва, МГТУ - 2016

Оглавление

Описание задания лабораторной работы

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

Задача 1 (ex_1.py)

Необходимо реализовать генераторы field и gen_random Генератор field последовательно выдает значения ключей словарей массива.

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.

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. Итератор не должен модифицировать возвращаемые значения.

В ex_2.py нужно вывести на экран то, что они выдают одной строкой. 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.

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.

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.

Задача 6 (ex_6.py)

Реализовать 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”. Для модификации использовать функцию map.

4. Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Использовать zip для обработки пары специальность — зарплата.

Реализация поставленной задачи

Листинг программы gen.py

```
import random


# Генератор вычленения полей из массива словарей
# Пример:
# goods = [
#    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
#    {'title': 'Диван для отдыха', 'price': 5300, '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', 'price': 5300}

def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
    for elem in items:
        r = {}
        for arg in args:
            if arg in elem:
                if len(args) == 1:
                    yield elem[arg]
                else:
                    if elem[arg] is not None:
                        r[arg] = elem[arg]
        if len(r) > 0 and len(args) > 1:
            yield r


# 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):
    # Необходимо реализовать генератор
    for i in range(num_count):
        yield random.randint(begin, end)
```

```
def gen(begin, end, num_count):
    # Необходимо реализовать генератор
    isFirst = True
    for i in range(num_count):
        if (isFirst):
            yield begin
        else:
            yield end
        isFirst = not isFirst
```

Листинг программы iterators.py

```
# Итератор для удаления дубликатов
import types

class Unique(object):
    IGNORE_CASE = False
    ITEMS = []
    PASSED = []

    def __init__(self, items, **kwargs):
        # Нужно реализовать конструктор
        # В качестве ключевого аргумента, конструктор должен принимать bool-
параметр ignore_case,
        # в зависимости от значения которого будут считаться одинаковые
строки в разном регистре
        # Например: ignore_case = True, Абв и АБВ разные строки
        #           ignore case = False, Абв и АБВ одинаковые строки, одна из
них удалится
        # По-умолчанию ignore_case = False
        if 'ignore_case' in kwargs.keys():
            self.IGNORE_CASE = kwargs['ignore_case']
        if not isinstance(items, types.GeneratorType):
            self.ITEMS = iter(items)
        else:
            self.ITEMS = items

    def __next__(self):
        # Нужно реализовать __next__
        while True:
            val = next(self.ITEMS)
            val2 = val
            if self.IGNORE_CASE:
                val2 = val2.lower()
            if val2 not in self.PASSED:
                self.PASSED.append(val2)
                return val

    def __iter__(self):
        del self.PASSED[:]
        return self
```

Листинг программы decorators.py

```
# Здесь необходимо реализовать декоратор, print_result который принимает на
вход функцию,
# вызывает её, печатает в консоль имя функции, печатает результат и
возвращает 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 из ex_4.py:
# @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

def print_result(func):
    def decorated(*args,**kwargs):
        print(func.__name__)
        result = func(*args,**kwargs)
        if type(result) is list:
            print("\n".join([str(x) for x in result]))
        elif type(result) is dict:
            print("\n".join([str(x) + "=" + str(result[x]) for x in result]))
        else:
            print(result)
        return result
    return decorated
```

Листинг программы ctxmngrs.py

```
# 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
import time

class timer:
    def __enter__(self):
        self.start = (time.time())
```

```
    def __exit__(self, exp_type, exp_value, traceback):
        tstop = (time.time())
        tstop -= self.start
        print("Время выполнения: " + str(round(tstop,1)))
```

Листинг программы ex_1.py

```
#!/usr/bin/env python3
from librip.gen import *

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, 'title1', 'price')))
print(list(gen_random(1, 3, 5)))
```

Листинг программы ex_2.py

```
#!/usr/bin/env python3
from librip.gen import gen_random, gen
from librip.iterators import Unique

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 3, 10)

data3 = gen(1,2,10)

# Реализация задания 2
print(list(Unique(data1)))
print(list(Unique(data2)))
data = ['a', 'A', 'b', 'B']
print(list(Unique(data)))
data = ['a', 'A', 'B', 'b']
print(list(Unique(data, ignore_case=True)))

print(list(data3))
```

Листинг программы ex_3.py

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

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

Листинг программы ex_4.py

```
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 'iu'


@print_result
def test_3():
    return {'a': 1, 'b': 2}


@print_result
def test_4():
    return [1, 2]

@print_result
def test_5(a,b,c=0):
    return a+b+c

test_1()
test_2()
test_3()
test_4()
test_5(2,2,c=2)
```

Листинг программы ex_5.py

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

with timer():
    sleep(1.5)
```

Листинг программы ex_6.py

```
#!/usr/bin/env python3
import json
import sys
from librip.ctxmngrs import timer
from librip.decorators import print_result
from librip.gen import *
from librip.iterators import Unique as unique


path=sys.argv[1]
# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

with open(path) as f:
    data = json.load(f)


# Далее необходимо реализовать все функции по заданию, заменив `raise
NotImplemented`
# Важно!
# Функции с 1 по 3 должны 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 _: "Программист" in _, arg))


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


@print_result
def f4(arg):
    return list(map(lambda x: "{}, зарплата {} руб.".format(x[0], x[1]),
                    zip(arg, gen_random(100000, 200000, len(arg)))))


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

Результаты работы программы

ex_1:

```
C:\Users\Mary\AppData\Local\Programs\Python\Python35-32\python.exe "
['Ковер', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']
[{'price': 2000}, {'price': 5300}, {'price': 7000}, {'price': 800}]
[2, 3, 1, 2, 2]

Process finished with exit code 0
```

ex_2:

```
C:\Users\Mary\AppData\Local\Progra
[1, 2]
[2, 1, 3]
['a', 'A', 'b', 'B']
['a', 'B']
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]

Process finished with exit code 0
```

ex_3:

```
C:\Users\Mary\AppData\Local\Programs\Py
[0, 1, -1, 4, -4, -30, 100, -100, 123]

Process finished with exit code 0
```

ex_4:

```
C:\Users\Mary\AppData\Local\Progra
test_1
1
test_2
iu
test_3
b=2
a=1
test_4
1
2
test_5
6

Process finished with exit code 0
```

ex_5:

```
C:\Users\Mary\AppData\Local\Progra
Время выполнения: 1.5

Process finished with exit code 0
```