Московский государственный технический университет им. Н.Э.Баумана

Защищено:
Гапанюк Ю.Е.

"__"_____2016 г.

Демонстрация ЛР:
Гапанюк Ю.Е.

"__"_____2016 г.

Отчет по лабораторной работе № 4 по курсу Разработка интернет приложений

"Лабораторная работа №3. Функциональные возможности."

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-52 _____
(подпись)

Смирнов А. И.

"__"_____2016 г.

Москва, МГТУ - 2016

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 реализовывать одной строкой .

Подготовительный этап

1. Зайти на github.com и выполнить fork проекта с заготовленной структурой https://github.com/iu5team/ex-lab4
2. Переименовать репозиторий в lab_4
3. Выполнить git clone проекта из вашего репозитория

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 . 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 *о дной 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** . 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). П ример: *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.__

Текст программы:

Ctxmngrs.py:

```
import time


class timer:
    def __enter__(self):
        time.clock()
```

```
    def __exit__(self, exp_type, exp_value, traceback):
        print(time.clock())
```

decorators.py:

```
def print_result(func_to_decorate):
    def decorated_func(*args):
        res = func_to_decorate(*args)
        print(func_to_decorate.__name__)
        if type(res) is str or type(res) is int:
            print(res)
        if type(res) is list:
            list(map(lambda x: print(x), res))
        if type(res) is dict:
            for k, v in res.items():
                print('{} = {}'.format(k, v))
        return res


    return decorated_func
```

gens.py:

```
import random


def field(items, *args):
    assert len(args) > 0
    for item in items:
        if len(args) == 1:
            if item.get(args[0]) is None:
                continue
            yield item[args[0]]
        else:
            dictionary = {}
            for name in args:
                if item.get(name) is None:
                    continue
                dictionary[name] = item.get(name)
            if dictionary:
                yield dictionary
            else:
                continue


def gen_random(begin, end, num_count):
    for i in range(num_count):
        yield random.randint(begin, end)
```

iterators.py:

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = list(items)
        self.index = -1
```

```
        self.lst = []
        self.len = len(self.items)
        self.ignore_case = kwargs.get('ignore_case')

    def __next__(self):
        self.index += 1
        if self.index == self.len:
            raise StopIteration
        buffer = self.items[self.index]
        buf_str = str(buffer)
        if self.ignore_case:
            buf_str = buf_str.lower()
            while buf_str in self.lst:
                self.index += 1
                if self.index == self.len:
                    raise StopIteration
                buffer = self.items[self.index]
                buf_str = str(buffer).lower()
            self.lst.append(buf_str)
            return buffer
        else:
            while buffer in self.lst:
                self.index += 1
                if self.index == self.len:
                    raise StopIteration
                buffer = self.items[self.index]
            self.lst.append(buffer)
            return buffer

    def __iter__(self):
        return self
```

Ex_1.py:

```
#!/usr/bin/env python3
from librip.gens 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': 'Вешалка для одежды', 'price': 800, 'color': 'white'},
    {'title': None, 'price': 7000},
    {'title': None, 'price': None}
]

# Реализация задания 1
print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price1')))
print(list(gen_random(1, 3, 10)))
```

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)
```

```
data3 = ['a', 'A', 'b', 'B']
data4 = ['Aab', 'aAB', 'BBB', 'bbb', 'bbb']

# Реализация задания 2

print(list(Unique(data4, ignore_case=False)))
print(list(Unique(data3, ignore_case=True)))
print(list(Unique(data2, ignore_case=True)))
print(list(Unique(data1, ignore_case=True)))
```

Ex_3.py:

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

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

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, 3]


test_1()
test_2()
test_3()
test_4()
```

ex_5.py:

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

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

ex_6.py:

```
# 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=str.lower)


@print_result
def f2(arg):
    return list(filter(lambda x: x.startswith('Программист'), arg))


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


@print_result
def f4(arg):
    salary = list(gen_random(100000, 200000, len(arg)))
    return list('{}, зарплата {} руб'.format(x, y) for x, y in zip(arg,
salary))


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

Результаты работы программы: