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 РИП

Вариант № <26>

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-52

_____

(подпись)

Чекулина М.Ю.

"__"_____2016 г.

Москва, МГТУ - 2016

1.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.__

2. Код программы

ex_1.py

```
#!/usr/bin/env python3
from librip.gens import field
from librip.gens 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': 'brown'}
]

# Реализация задания 1

for item in gen_random(1, 7, 9):
    print (item, end=" ")
print()
for item in field(goods, "title", "price", "color"):
    print(item, end=" ")
    print()
```

ex_2.py

```
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']

# Реализация задания 2

for i in Unique(list(data1)):
    print(i, end=" ")
print()

for i in Unique(list(data2)):
    print(i, end=" ")
print()
```

```
for i in Unique(list(data3)):
    print(i, end=" ")
print()

for i in Unique(list(data3), ignore_case=True):
    print(i, end=" ")
print()
```

ex_3.py

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

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3
#abs-модуль
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]


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
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 = 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)))
   return sorted(unique(field(arg, "job-name"), ignore_case=1), key=lambda x:x.lower())


# startswith-проверяет(точнее возвращает флаг) начинается ли строка с такого-то слова
@print_result
def f2(arg):
    return list(filter(lambda x: x.startswith("Программист"), arg))


#map-функция для преобразования коллекции
@print_result
def f3(arg):
    return list(map(lambda x: x + " с опытом Python", arg))


#zip- для слияния по парам (например, a=[1,2], b=[3,4], print zip(a,b), [(1,3),(2,4)]
@print_result
def f4(arg):
    #return ["{}, зарплата {} руб.".format(work, salary) for (work, salary) in zip(arg,
gen random(100000, 200000, len(arg)))]
    a = list(gen_random(100000, 200000, len(arg)))
    return list('{}, зарплата {} руб.'.format(arg, a) for arg, a in zip(arg, a))


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

ctxmngrs.py

```
import time

class timer:
    def __enter__(self):
        self.begin_time = time.clock()
    def __exit__(self, exc_type, exc_val, exc_tb):
        print(time.clock() - self.begin_time)
```

iterators.py

```
# Итератор для удаления дубликатов
class Unique(object):
    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
        # self.ignore_case = False

        #isinstance- проверка принадлежности объекта указанному классу(принадлежит ли
items list )
        self.items = iter(items) if isinstance(items, list) else items
        self.ignore_case = False
        self.unique = []
        if (kwargs.get('ignore_case') == True):
            self.ignore_case = True

    def __next__(self):
        # Нужно реализовать __next__
        for element in self.items:
            if self.ignore_case == True:
                #lower()- преобразование строки к нижнему регистру
                if (element.lower() not in self.unique):
                    #str()- строковое представление объекта
                    str_buf = str(element).lower()
                    self.unique.append(str(str_buf))
                    return element
            else:
                if (element not in self.unique):
                    self.unique.append(element)
                    return element
        raise StopIteration()

    def __iter__(self):
        return self
```

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
def print_result(print_func):
    def decorated_func(*args, **kwargs):
        print(print_func.__name__)
        result = print_func(*args, **kwargs)
        if type(result) is list:
            #"\n".join()- сборка строки из списка с разделителем "\n"
            #join(map())- для списка с числами
            print("\n".join(map(str, result)))
        elif type(result) is dict:
                print("\n".join([str(k)+"="+str(v)for k,v in result.items()]))
        else:
            print(result)
        return result
    return decorated_func
```

gens.py

```
#Модуль random предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности.
#random.randint(A, B) - случайное целое число N, A ≤ N ≤ B.
from random import randint

# реализовать генератор,который последовательно выведет значения ключей словарей массива
def field(list,*args):
    #assert позволяет производить проверки истинности утверждений, что может быть использовано в отладочных целях
    #Если проверка не пройдена, возбуждается исключение
    #для ситуаций, которые не должны происходить вовсе, которые нельзя обработать или это не имеет смысла
    assert len(args) > 0, "There are no input arguments"
    if len(args) == 1:
        for item in list:
            if item.get(args[0]):
                yield item[args[0]]
    else:
        for item in list:
            dictionary = {}
            for element in args:
                if item.get(element):
                    dictionary[element] = item[element]
            if dictionary:
                # вызывающему коду выдается значение dictionary, так как дошли до yield
                yield dictionary

#реализовать генератор, который последовательно выдает заданное
#количество случайных чисел в заданном диапазоне

def gen_random(begin, end, num_count):
    for item in range(num_count):
        yield randint(begin, end)
```

Результаты работы:

ex_1.py

```
"C:\Program Files (x86)\Python 3
7 3 6 3 1 6 2 5 3
Шкаф
```

ex_2.py

```
"C:\Program Files (x86)\Python 35\Python\Python35\python.exe" C:/Users/Марина/PycharmProjects/lab_4/lab_4/ex_2.py
1 2
1 3 2
a A b B
a b
```

ex_3.py

```
"C:\Program Files (x86)\Python 35\Python\Python35\python.exe
[0, 1, -1, 4, -4, -30, 100, -100, 123]
```

ex_4.py

```
"C:\Program Files (x86)\Python 35\Python\Python35\python.exe" C:/Use
test_1
1
test_2
iu
test_3
b=2
a=1
test_4
1
2
```

ex_5.py

```
"C:\Program Files (x86)\Python 35\Pyth
5.499974546099478
```

ex_6.py

```
"C:\Program Files (x86)\Python 35\Python\Python35\python.exe" C:/Users/Марина/PycharmProjects/lab_4/
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
Автоэлектрик
Агент
Агент банка
Агент нпф
Агент по гос. закупкам недвижимости
Агент по недвижимости
Агент по недвижимости (стажер)
Агент по недвижимости / Риэлтор
Агент по привлечению юридических лиц
Агент по продажам (интернет, ТВ, телефония) в ПАО Ростелеком в населенных пунктах Амурской области:
Агент торговый
агрегатчик-топливник KOMATSU
агроном
агроном по защите растений
Агроном-полевод
агрохимик почвовед
```

```
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 189652 руб.
Программист / Senior Developer с опытом Python, зарплата 152044 руб.
Программист 1С с опытом Python, зарплата 163283 руб.
Программист C# с опытом Python, зарплата 133783 руб.
Программист C++ с опытом Python, зарплата 127331 руб.
Программист C++/C#/Java с опытом Python, зарплата 162321 руб.
Программист/ Junior Developer с опытом Python, зарплата 155113 руб.
Программист/ технический специалист с опытом Python, зарплата 179625 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 196492 руб.
0.32436069655009325
```