МГТУ им. Н.Э. Баумана

Факультет «Информатика и системы управления»

ДИСЦИПЛИНА:

«Базы данных»

Отчет по лабораторной работе №2

«Python. Функциональные возможности»

Выполнила:

Студент 3 курса

Факультет ИУ

Группа ИУ5-51Б

Мартынова Д.П.

04.10.19

**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 запрограммировать 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\_2

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, с помощью кода в *одну строку*

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 *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**. 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.

**Исходный код:**

**ex\_1.py**

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': 'white', 'sz' : 1}

]

# Реализация задания 1

print([x for x in field(goods, "title")],

[x for x in field(goods, "title", "price")],

[x for x in gen\_random(1, 3, 5)],

sep = '\n')

**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

print([x for x in Unique(data1)],

[x for x in Unique(data2)],

[x for x in Unique(data3)],

[x for x in Unique(data3, ignore\_case=True)],

sep = '\n')

**ex\_3.py**

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

# Реализация задания 3

def srt(num):

return(abs(num))

print(sorted(data, key = srt))

**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()

**ex\_5.py**

from time import sleep

from librip.ctxmngrs import timer

with timer():

sleep(5.5)

**ex\_6.py**

import json

import sys

from sys import argv

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 = argv[1]

with open(path) as f:

data = json.load(f)

@print\_result

def f1(arg):

return sorted(unique([x for x in field(arg, 'job-name')],ignore\_case = True))

@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):

return['{0}, зарплата {1} рублей'.format(x[0],x[1])for x in zip(arg,gen\_random(100000,200000,len(arg)))]

with timer():

f4(f3(f2(f1(data))))

**ctxmngrs.py**

from time import time

class timer(object):

def \_\_enter\_\_(self):

self.start = time()

def \_\_exit\_\_(self, exp\_type, exp\_value, exp\_tr):

print(time() - self.start)

**decorators.py**

def print\_result(func):

def wrapper(\*args):

print(func.\_\_name\_\_)

res = func(\*args)

if isinstance(res, list):

print('\n'.join([str(x) for x in res]))

elif isinstance(res, dict):

print('\n'.join(['{0} = {1}'.format(key, value) for key, value in res.items()]))

else:

print(res)

return res

return wrapper

**gens.py**

import random

def field(items, \*args):

assert len(args) > 0

for i in items:

if len(args) == 1:

res = i.get(args[0])

if res is None:

continue

yield res

else:

dict = {}

for a in args:

res = i.get(a)

if res is None:

continue

dict[a] = res

yield dict

# Генератор списка случайных чисел

def gen\_random(begin, end, num\_count):

for i in range(num\_count):

yield(random.randint(begin,end))

**iterators.py**

# Итератор для удаления дубликатов

from itertools import groupby

from types import GeneratorType

class Unique(object):

def \_\_init\_\_(self, items, \*\*kwarg):

ignore\_case = kwarg.get('ignore\_case')

if ignore\_case is None:

ignore\_case = False

self.res = []

self.ignore\_case = ignore\_case

if isinstance(items, GeneratorType):

self.items = items

else:

self.items = iter(items)

def \_\_next\_\_(self):

self.this\_el = self.items.\_\_next\_\_()

if self.ignore\_case:

self.this\_el = self.this\_el.lower()

if self.this\_el in self.res:

self.\_\_next\_\_();

else:

self.res.append(self.this\_el)

return self.this\_el

def \_\_iter\_\_(self):

return self