Отчет по лабораторной работе
№ 4 по курсу
"Разработка интернет приложений"

" Python. Функциональные возможности "

Дмитриев Н.А., ИУ5-53

Москва, МГТУ - 2016 год

Задание и порядок выполнения:

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

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

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

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'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)))
print (list(field(goods, 'title')))
print (list(field(goods, 'title', 'price')))
print (list(gen_random(1, 5, 10)))
```

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):
    # Необходимо реализовать генератор
    if len(args) == 1:
        for i in items:
            if args[0] in i:
                yield i[args[0] ]
    else:
        for i in items:
            tmp = {}
            for key in args:
                if key in i:
                    tmp[key] =
            i[key] if tmp != {}:
                yield tmp

# 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.randrange(begin, end)
```

ex_2.py

```
#!/usr/bin/env python3
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, 3, 10)
data3 = ['a', 'b', 'A', 'B']
# Реализация задания 2

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

iterator.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.data = iter(items)
        self.set = set()
        self.ignore_case = kwargs.get('ignore_case', False)


    def __next__(self):
        # Нужно реализовать __next__

        while( True ):
            nxt = next(self.data)
            tmp = nxt
            if self.ignore_case :
                tmp = nxt.lower()
            if tmp in self.set:
                continue
            else:
                break

        self.set.add(tmp)
        return nxt


    def __iter__(self):
        return self
```

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 num: abs(num) ))
```

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()
```

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 some_fun(*args, **kwargs ):
        print(func.__name__)
        res = func(*args, **kwargs)
        if type(res) == type(list()):
            print ("\n".join (map(str, res)))
        elif type(res) == type(dict()):
            print ("\n".join (map(lambda x: "{} = {}".format(x[0], x[1]) ,
        res.items()))) else:
            print (res)
        return res

    return some_fun
```

ex_5.py

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

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

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 __init__(self):
        pass

    def __enter__(self):
        self.start = time.time()

    def __exit__(self, *args):
        print ("Elapsed %s" % (time.time() - self.start,))
```

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 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, encoding="utf-8") as f:
    data = json.load(f)


# 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 : not x.lower().find('программист') , arg))


@print_result
def f3(arg):
    return list( "%s с опытом Python" % (i, ) for i in arg)


@print_result
def f4(arg):
    return list("%s, зарплата %s" % (i, next(gen_random(100000, 200000, 1))) for i in arg)


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

Результаты выполнения:

Ex_1.py

```
[]

['Ковер', 'Стелаж', 'Вешалка для одежды']

[{'price': 2000, 'title': 'Ковер'}, {'price': 5300}, {'price': 7000, 'title': 'Стелаж'}, {'price': 800, 'title': 'Вешалка для одежды'}]
```

[1, 4, 3, 4, 3, 4, 4, 3, 2, 2]

Ex_2.py

[1, 2] [1,

2] ['a',

'b']

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

Elapsed 5.500242233276367

Ex_6.py

…

Энергетик литейного производства

энтомолог Юрисконсульт

юрисконсульт 2 категории 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 142689

Программист / Senior Developer с опытом Python, зарплата 172461

Программист 1С с опытом Python, зарплата 196823

Программист C# с опытом Python, зарплата 106433

Программист C++ с опытом Python, зарплата 195062

Программист C++/C#/Java с опытом Python, зарплата 177046

Программист/ Junior Developer с опытом Python, зарплата 138553

Программист/ технический специалист с опытом Python, зарплата 134898

Программистр-разработчик информационных систем с опытом Python, зарплата 175787

Elapsed 0.01890873908996582