Защищено:
Гапанюк Ю.Е.

"__"_____2016 г.

Демонстрация:
Гапанюк Ю.Е.

"__"_____2016 г.

Отчет по лабораторной работе № 4 по курсу

«Разработка интернет-приложений»

«Python. Функциональные возможности»

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-53 _____
(подпись)

Ореликов М.Г.

"__"_____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
МГТУ им. Н. Э. Баумана, кафедра ИУ5, курс РИП
ЛР №4: Python, функциональные возможности
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). 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

```
#!/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'}
]

# Реализация задания 1

print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))

print(list(gen_random(1, 3, 5)))
```

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)

# Реализация задания 2

print(list(Unique(data1)))
print(list(Unique(data2)))
print(list(Unique(['a', 'A', 'b', 'B'])))
print(list(Unique(['a', 'A', 'b', 'B'], 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=lambda n: abs(n)))
```

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, 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 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 x: 'программист' in x.lower() , arg))
```

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

@print_result
def f4(arg):
    return list(zip(arg, gen_random(100000, 200000, len(arg))))


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

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

    return some_fun
```

gens.py

```
import random

# Генератор вычленения полей из массива словарей
def field(items, *args):
    #assert len(args) > 0
    # Необходимо реализовать генератор
    flag =0
    l = ''
    for item in items:
        assert (type(item) == dict)
        if len(args) == 1:
            yield item.get(args[0])
        else:
            yield { arg : item.get(arg) for arg in args}

# Генератор списка случайных чисел
def gen_random(begin, end, num_count):
    # Необходимо реализовать генератор
    for i in range(num_count):
        yield random.randint(begin, end)
```

ctxmngrs.py

```
# Здесь необходимо реализовать
# контекстный менеджер timer
# Он не принимает аргументов, после выполнения блока он должен вывести время выполнения в секундах
import time

class timer:
    def __enter__(self):
        print(time.time())
        self.enter = time.time()

    def __exit__(self, *args):
        print(time.time())
        print("Время работы: " + str((time.time() - self.enter))+ " секунд")
```

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
        self.items = iter(items)
        self.set = set()
        self.ignore_case = kwargs.get('ignore_case')

    def __next__(self):
        # Нужно реализовать __next__
        while (True):
            nex = next(self.items)
            cor = nex
            if self.ignore_case == True:
                cor = nex.lower()
            if cor in self.set:
                continue
            else:
                break

        self.set.add(cor)
        return nex

    def __iter__(self):
        return self
```

Результаты выполнения

ex_1.py

```
C:\Users\LENOVO\AppData\Local\Programs\Python\Python35-32\python.exe C:/Users/LENOVO/PycharmProjects/lab4/ex_1.py
['Ковер', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']
[{'price': 2000, 'title': 'Ковер'}, {'price': 5300, 'title': 'Диван для отдыха'}, {'price': 7000, 'title': 'Стелаж'}, {'price': 800, 'title': 'Вешалка для одежды'}]
[2, 2, 2, 1, 3]

Process finished with exit code 0
```

ex_2.py

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

Process finished with exit code 0
```

ex_3.py

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

Process finished with exit code 0
```

ex_4.py

```
C:\Users\LENOVO\AppData\Local\Programs\Python\
test_1
1
test_2
iu
test_3
b = 2
a = 1
test_4
1
2

Process finished with exit code 0
```

ex_5.py

```
C:\Users\LENOVO\AppData\Local\Programs\
1481294228.8544445
```

ex_6.py

```
C:\Users\LENOVO\AppData\Local\Programs\Python\Python
1481294368.0207326
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
f2
1С программист
Web-программист
Веб - программист (PHP, JS) / Web разработчик
Веб-программист
Ведущий инженер-программист
Ведущий программист
инженер - программист
Инженер - программист АСУ ТП
инженер-программист
Инженер-программист (Клинский филиал)
Инженер-программист (Орехово-Зуевский филиал)
Инженер-программист 1 категории
Инженер-программист ККТ
Инженер-программист ПЛИС
Инженер-программист САПОУ (java)
Инженер-электронщик (программист АСУ ТП)
педагог программист
Помощник веб-программиста
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
Системный программист (C, Linux)
Старший программист
```

```
f3
1С программист с опытом Python
Web-программист с опытом Python
Веб - программист (PHP, JS) / Web разработчик с опытом Python
Веб-программист с опытом Python
Ведущий инженер-программист с опытом Python
Ведущий программист с опытом Python
инженер - программист с опытом Python
Инженер - программист АСУ ТП с опытом Python
инженер-программист с опытом Python
Инженер-программист (Клинский филиал) с опытом Python
Инженер-программист (Орехово-Зуевский филиал) с опытом Python
Инженер-программист 1 категории с опытом Python
Инженер-программист ККТ с опытом Python
Инженер-программист ПЛИС с опытом Python
Инженер-программист САПОУ (java) с опытом Python
Инженер-электронщик (программист АСУ ТП) с опытом Python
педагог программист с опытом Python
Помощник веб-программиста с опытом Python
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 Pythor
Системный программист (C, Linux) с опытом Python
Старший программист с опытом Python
f4
('1С программист с опытом Python', 166859)
('Web-программист с опытом Python', 138392)
('Веб - программист (PHP, JS) / Web разработчик с опытом Python', 146000)
('Веб-программист с опытом Python', 126624)
('Ведущий инженер-программист с опытом Python', 120528)
('Ведущий программист с опытом Python', 141965)
('инженер - программист с опытом Python', 189001)
('Инженер - программист АСУ ТП с опытом Python', 178189)
('инженер-программист с опытом Python', 184786)
('Инженер-программист (Клинский филиал) с опытом Python', 136486)
('Инженер-программист (Орехово-Зуевский филиал) с опытом Python', 164394)
('Инженер-программист 1 категории с опытом Python', 148580)
('Инженер-программист ККТ с опытом Python', 188230)
('Инженер-программист ПЛИС с опытом Python', 122170)
('Инженер-программист САПОУ (java) с опытом Python', 193316)
('Инженер-электронщик (программист АСУ ТП) с опытом Python', 120794)
('педагог программист с опытом Python', 181443)
('Помощник веб-программиста с опытом Python', 150878)
('Программист с опытом Python', 165311)
('Программист / Senior Developer с опытом Python', 119770)
('Программист 1С с опытом Python', 183902)
('Программист C# с опытом Python', 116378)
('Программист C++ с опытом Python', 141240)
('Программист C++/C#/Java с опытом Python', 150723)
('Программист/ Junior Developer с опытом Python', 168322)
('Программист/ технический специалист с опытом Python', 133888)
('Программистр-разработчик  информационных систем с опытом Python', 154223)
('Системный программист (C, Linux) с опытом Python', 196516)
('Старший программист с опытом Python', 101135)
1481294368.098863
Время работы: 0.07813024520874023 секунд

Process finished with exit code 0
```