МГТУ им. Н. Э. Баумана, кафедра ИУ5
курс «Разработка интернет-приложений»

Лабораторная работа №3
Python. Функциональные возможности

Выполнила:
Нурлыева Дана Джалилевна
Группа ИУ5-53

Москва 2018

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_3
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'}

В качестве первого аргумента генератор принимает list, дальше через *args генератор принимает неограниченное кол-во аргументов.

Если передан один аргумент, генератор последовательно выдает только значения полей, если поле равно None, то элемент пропускается

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:

Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих заданий.

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.

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.

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

gens.py

```
import random
# Генератор вычленения полей и 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):
    assert len(args) > 0
    if len(args) == 1:
        for i in items:
            for a in args:
                yield i[a]
    else:
        for i in items:  # идём по словарю
            dict = {}  # новый словарь для записи
            for arg in args:
                if arg in i is not None:
                    dict[arg] = i[arg]  # формируем новый словарь
            if len(dict) > 0 and len(args) > 1:
                yield dict
# 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):
    for _ in range(num_count):
        yield random.randint(begin, end)
```

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':
'white'}
]
# Реализация задания 1
print (list(field(goods, 'title')))
print (list(field(goods, 'title', 'price')))
print (list(gen_random(1, 3, 5)))
```

iterators.py

```
from types import GeneratorType
# Итератор для удаления дубликатов
class Unique(object):
    IGNORE_CASE = False
    INDEX = 0
    OBJECTS = []
    PUSTOTA = []
    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, то Абв и АБВ разные строки
        #           ignore_case = False, то Абв и АБВ одинаковые
строки, одна из них удалится
        # По-умолчанию ignore_case = False(то есть регистры важны)
        if 'ignore_case' in kwargs.keys():
            self.IGNORE_CASE = kwargs['ignore_case']
        if type(items == GeneratorType):
            self.OBJECTS = list(items)
        else:
            self.OBJECTS = items
        # self.ITEMS = len(items)
    def __next__(self):
        while True:
            if self.INDEX == (len(self.OBJECTS) - 1):
                raise StopIteration
            self.INDEX += 1
            val = self.OBJECTS[self.INDEX]
            val2 = str(val).lower()
            if self.IGNORE_CASE:
```

```
                val = val2
            if val not in self.PUSTOTA:
                self.PUSTOTA.append(val)
                return val
    def __iter__(self):
        del self.PUSTOTA[:]
        self.INDEX = -1
        return self
```

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']
# Реализация задания 2
print(list(Unique(data1)))
print(list(Unique(data2)))
print(list(Unique(data3)))
print(list(Unique(data3, ignore_case=True)))
```

ex_3.py

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

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3
data = sorted(data, key=lambda x: abs(x))
print(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
# 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(fn):
    def inside(*args):
        print(fn.__name__)
        if len(args) == 0:
            fun = fn()
        else:
            fun = fn(args[0])
        if type(fun) == list:
            for i in fun:
                print(i)
        elif type(fun) == dict:
            for i in fun:
                print(i, "=", fun[i])
        else:
            print(fun)
        return fun
    return inside
```

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()
```

ctxmngrs.py

```
import time
# 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
class timer:
    def __enter__(self):
        self.start = time.time()
    def __exit__(self, exc_type, exc_val, exc_tb):
        ti = (time.time()) - self.start
        print(ti)
```

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

path = "data_light.json"
```

```
# 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 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 list(Unique(list(field(arg, "job-name")),
ignore_case=True))


@print_result
def f2(arg):
    return list(filter(lambda s: "программист" in s[0:12], arg))


@print_result
def f3(arg):  # map(func, arr)
    return list(map(lambda s: s + " с опытом Python", arg))


@print_result
def f4(arg):
    Sal = gen_random(100000, 200000, len(arg))
    return list(map(lambda s: '{}, зарплата {} руб.'.format(
        s[0], s[1]), zip(arg, Sal)))


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

Скриншоты с результатами выполнения
Задача 1

```
/Users/user/PycharmProjects/chat/env/bin/python /Users/user/Desktop/RIP/RIP-lab_3/ex_1.py
['Ковер', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стелаж', 'price': 7000}, {'title': 'Вешалка для одежды', 'price': 800}]
[2, 3, 1, 3, 2]

Process finished with exit code 0
```

Задача 2

```
/Users/user/PycharmProjects/chat/env/bin/python /Users/user/Desktop/RIP/RIP-lab_3/ex_2.py
[1, 2]
[3, 1, 2]
['a', 'A', 'b', 'B']
['a', 'b']

Process finished with exit code 0
```

Задача 3

```
/Users/user/PycharmProjects/chat/env/bin/python /Users/user/Desktop/RIP/RIP-lab_3/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]

Process finished with exit code 0
```

Задача 4

```
/Users/user/PycharmProjects/chat/env/bin/python /Users/user/Desktop/RIP/RIP-lab_3/ex_4.py
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2
```

Задача 5

```
/Users/user/PycharmProjects/chat/env/bin/python /Users/user/Desktop/RIP/RIP-lab_3/ex_5.py
5.501401662826538

Process finished with exit code 0
```

Задача 6

```
/Users/user/PycharmProjects/chat/env/bin/python /Users/user/Desktop/RIP/RIP-lab_3/ex_6.py
f1
администратор на телефоне
медицинская сестра
охранник сутки-день-ночь-вахта
врач анестезиолог реаниматолог
теплотехник
разнорабочий
электро-газосварщик
водитель gett/гетт и yandex/яндекс такси на личном автомобиле
монолитные работы
организатор – тренер
```

```
f2
программист
программист c++/c#/java
программист 1c
программистр-разработчик  информационных систем
программист c++
программист/ junior developer
программист / senior developer
программист/ технический специалист
программист c#
```

```
f3
программист с опытом Python
программист c++/c#/java с опытом Python
программист 1c с опытом Python
программистр-разработчик  информационных систем с опытом Python
программист c++ с опытом Python
программист/ junior developer с опытом Python
программист / senior developer с опытом Python
программист/ технический специалист с опытом Python
программист c# с опытом Python
f4
```

```
f4
программист с опытом Python, зарплата 106925 руб.
программист c++/c#/java с опытом Python, зарплата 172250 руб.
программист 1c с опытом Python, зарплата 130118 руб.
программистр-разработчик  информационных систем с опытом Python, зарплата 148990 руб.
программист c++ с опытом Python, зарплата 110392 руб.
программист/ junior developer с опытом Python, зарплата 185931 руб.
программист / senior developer с опытом Python, зарплата 141535 руб.
программист/ технический специалист с опытом Python, зарплата 142147 руб.
программист c# с опытом Python, зарплата 167057 руб.
0.060626983642578125
```