МГТУ им. Н. Э. Баумана, кафедра ИУ5,

курс “Разработка интернет-приложений”

Лабораторная работа №4 Python. Функциональные возможности

Выполнил:

Студент гр. ИУ5-53

Юркевич Алексей

Москва 2017

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 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 .

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'}
]

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

Ex_2.py

```
data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 3, 10)
data3 = ["DGR", "asd","RGH", "Asd", "Asd", "wtw", "Rgh", "WTW", "wt4w4",
"dgr"]

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

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

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

Ex_3.py

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
new_data = sorted(data, key=lambda i: abs(i))
print(new_data)
```

Ex_6.py

```
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


# encoding=utf8

path = "C:/Users/Олег/PycharmProjects/lab_4/data_light.json"

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

with open(path, "r", encoding='utf8') 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):
    jobs = list(field(arg, "job-name"))
    jobs = Unique(jobs, ignore_case=True)
    jobs = sorted(jobs)
    return jobs


@print_result
def f2(arg):
    jobs = list(filter(lambda x: "программист" in x.lower(), arg))
```

```
    return jobs


@print_result
def f3(arg):
    jobs = list(map(lambda x: x + "с опытом Python", arg))
    return jobs


@print_result
def f4(arg):
    jobs = list(arg)
    salaries = list(gen_random(100000, 200000, len(jobs)))
    salaries = list(map(lambda x: "зарплата " + str(x) + " руб.", salaries))
    full_info = list(zip(jobs, salaries))
    return full_info


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

ctxmngrs.py

```
class timer:

    def __init__(self):
        pass

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

    def __exit__(self, type, value, traceback):
        print(time() - self.time)
```

decorators.py

```
def print_result(func, *arg):

    def decorated_function(*arg):
        result = func(*arg)
        print(func.__name__)
        if type(result) is dict:
            for key, value in result.items():
                print("%s=%s" % (str(key), str(value)))
        elif type(result) is list:
            for i in result:
                print(i)
        else:
            print(result)
        return result

    return decorated_function
```

gens.py

```
def field(items, *args):
    # Необходимо реализовать генератор
    assert len(args) > 0
    if len(args) == 1:
        for item in items:
            if args[0] in item:
                yield item[args[0]]
```

```
    else:
        for item in items:
            new_item = {}
            for arg in args:
                if arg in item:
                    new_item[arg] = item[arg]
            if len(new_item.keys()) > 0:
                yield new_item


    # Необходимо реализовать генератор


# 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):
    if begin > end:
        begin, end = end, begin

    for i in range(1, num_count):
        yield random.randint(begin, end)
```

iterators.py

```
def __init__(self, items, ignore_case=False, **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.unique_items = []
    self.ignore_case = ignore_case
    self.items = iter(items)

def __next__(self):
    # Нужно реализовать __next__
    while True:
        item = self.items.__next__()
        compare_item = None
        if self.ignore_case and type(item) is str:
            compare_item = item.lower()
        else:
            compare_item = item
        if compare_item not in self.unique_items:
            self.unique_items.append(compare_item)
            return item

def __iter__(self):

    return self
```

Результаты

f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
[химик-эксперт
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
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: г. Благовещенск, г. Белогорск, г. Свободный, г. Шимановск, г. Зея, г. Тында
Агент торговый
Агроном-полевод
Администратор
Администратор (удаленно)
Администратор Active Directory
Администратор в парикмахерский салон
Администратор зала (предприятий общественного питания)

f2
1С программист
Web-программист
Веб - программист (PHP, JS) / Web разработчик
Веб-программист
Ведущий инженер-программист
Ведущий программист
Инженер - программист АСУ ТП
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)
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
инженер - программист
инженер-программист
педагог программист

f3
1С программистс опытом Python
Web-программистс опытом Python
Веб - программист (PHP, JS) / Web разработчикс опытом Python
Веб-программистс опытом Python
Ведущий инженер-программистс опытом Python
Ведущий программистс опытом Python
Инженер - программист АСУ ТПс опытом Python
Инженер-программист (Клинский филиал)с опытом Python
Инженер-программист (Орехово-Зуевский филиал)с опытом Python
Инженер-программист 1 категориис опытом Python
Инженер-программист ККТс опытом Python
Инженер-программист ПЛИСс опытом Python
Инженер-программист САПОУ (java)с опытом Python
Инженер-электронщик (программист АСУ ТП)с опытом Python
Помощник веб-программистас опытом Python
Программистс опытом Python
Программист / Senior Developerс опытом Python
Программист 1Сс опытом Python
Программист C#с опытом Python
Программист C++с опытом Python
Программист C++/C#/Javac опытом Python
Программист/ Junior Developerс опытом Python
Программист/ технический специалистс опытом Python
Программистр-разработчик информационных системс опытом Python
Системный программист (C, Linux)с опытом Python
Старший программистс опытом Python
инженер - программистс опытом Python
инженер-программистс опытом Python
педагог программистс опытом Python

```
f4
('1С программистс опытом Python', 'зарплата 110952 руб.')
('Web-программистс опытом Python', 'зарплата 195239 руб.')
('Веб - программист (PHP, JS) / Web разработчикс опытом Python', 'зарплата 132894 руб.')
('Веб-программистс опытом Python', 'зарплата 100691 руб.')
('Ведущий инженер-программистс опытом Python', 'зарплата 192671 руб.')
('Ведущий программистс опытом Python', 'зарплата 177476 руб.')
('Инженер - программист АСУ ТПс опытом Python', 'зарплата 130843 руб.')
('Инженер-программист (Клинский филиал)с опытом Python', 'зарплата 180743 руб.')
('Инженер-программист (Орехово-Зуевский филиал)с опытом Python', 'зарплата 159130 руб.')
('Инженер-программист 1 категориис опытом Python', 'зарплата 129699 руб.')
('Инженер-программист ККТс опытом Python', 'зарплата 174029 руб.')
('Инженер-программист ПЛИСс опытом Python', 'зарплата 110138 руб.')
('Инженер-программист САПОУ (java)с опытом Python', 'зарплата 124683 руб.')
('Инженер-электронщик (программист АСУ ТП)с опытом Python', 'зарплата 103686 руб.')
('Помощник веб-программистас опытом Python', 'зарплата 197124 руб.')
('Программистс опытом Python', 'зарплата 173332 руб.')
('Программист / Senior Developerс опытом Python', 'зарплата 115193 руб.')
('Программист 1Сс опытом Python', 'зарплата 111206 руб.')
('Программист C#с опытом Python', 'зарплата 142334 руб.')
('Программист C++с опытом Python', 'зарплата 118014 руб.')
('Программист C++/C#/Javac опытом Python', 'зарплата 197043 руб.')
('Программист/ Junior Developerс опытом Python', 'зарплата 134952 руб.')
('Программист/ технический специалистс опытом Python', 'зарплата 165752 руб.')
('Программистр-разработчик  информационных системс опытом Python', 'зарплата 107873 руб.')
('Системный программист (C, Linux)с опытом Python', 'зарплата 141893 руб.')
('Старший программистс опытом Python', 'зарплата 143768 руб.')
('инженер - программистс опытом Python', 'зарплата 158769 руб.')
('инженер-программистс опытом Python', 'зарплата 175407 руб.')
0.3160698413848877
```