МГТУ им. Н.Э. Баумана

Факультет «Информатика и системы управления»

ДИСЦИПЛИНА:

«Разработка интернет-приложений»

Отчет по лабораторной работе №2

«Python. Функциональные возможности»

Выполнил:

Студент 3 курса

Факультет ИУ

Группа ИУ5-51Б

Сукиасян В.М.

Преподаватель:

Гапанюк Ю.Е.

## **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

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

test\_1()

На консоль выведется:

test\_1

1

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 данного задания**

### **Задача 1**

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):

assert len(args) > 0

for item in items:

if len(args) == 1 and args[0] in item:

yield item[args[0]]

else:

result = {}

for arg in args:

if arg in item:

result[arg] = item[arg]

if len(result) != 0:

yield result

# 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)

**# Реализация задания 1**

[print(res) for res in field(goods, 'title', 'color')]

### **Задача 2**

# Итератор для удаления дубликатов

class Unique(object):

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

self.items = iter(items)

self.ignore\_case = kwargs['ignore\_case'] if 'ignore\_case' in kwargs else False

self.unique\_items = []

# 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, Aбв и АБВ разные строки

# ignore\_case = False, Aбв и АБВ одинаковые строки, одна из них удалится

# По-умолчанию ignore\_case = False

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

while True:

item = self.items.\_\_next\_\_()

if not self.ignore\_case and type(item) is str:

tmp\_item = item.lower()

else:

tmp\_item = item

if tmp\_item not in self.unique\_items:

self.unique\_items.append(tmp\_item)

return item

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

return self

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']

[print(res) for res in Unique(data1, ignore\_case=True)]

print(list(Unique(data3)))

# or

# [print(res) for res in Unique(data3, ignore\_case=False)]

### **Задача 3**

#!/usr/bin/env python3  
  
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
# Реализация задания 3  
print(sorted(map(lambda x: abs(x), data)))

### **Задача 4**

def print\_result(func):

def decorated\_func(data):

result = func(data)

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

if type(result) is list:

print(\*result, sep='\n')

elif type(result) is dict:

for key, value in result.items():

print('{}={}'.format(key, value))

else:

print(result)

return result

return decorated\_func

### **Задача 5**

import time

class timer:

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

self.time = time.time()

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

print(time.time() - self.time)

### **Задача 6**

#!/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) 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(

list(unique(field(arg, 'job-name'))),

key=lambda s: s.lower()

)

@print\_result

def f2(arg):

return list(filter(lambda s: s.lower().startswith('программист'), arg))

@print\_result

def f3(arg):

return list(map(lambda s: s + ' c опытом Python', arg))

@print\_result

def f4(arg):

return list(map(lambda s: s + ', зарплата {} рублей'.format(list(gen\_random(100000, 200000, 1))[0]), arg))

with timer():

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

### **Скриншоты с результатами выполнения**





