**Московский государственный технический**

**университет им. Н. Э. Баумана**

**Факультет «Информатика и системы управления»**

Кафедра «Системы обработки информации и управления»

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №2

Группа: ИУ5-52Б

Студент: Брусникина М.И.

Преподаватель: Гапанюк Ю.Е.

Москва, 2019 г.

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

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. 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

**gen.py**

import random  
  
  
def field(items, \*args):  
 assert len(args) > 0  
 if len(args) == 1:  
 for dict in items:  
 if args[0] in dict.keys() and not dict[args[0]] is None:  
 yield dict[args[0]]  
 else:  
 for dict in items:  
 elem = {}  
 for arg in args:  
 if arg in dict.keys() and not dict[arg] is None:  
 elem[arg] = dict[arg]  
 if len(elem) != 0:  
 yield elem  
  
  
def gen\_random(begin, end, num\_count):  
 for i in range(num\_count):  
 yield random.randint(begin, end)

**iterators.py**

class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 # 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  
 self.data = []  
 strings = []  
 for i in items:  
 if isinstance(i, str):  
 if **'ignore\_case'** in kwargs.keys() and kwargs[**'ignore\_case'**] is True:  
 if i.lower() not in strings:  
 strings.append(i.lower())  
 self.data.append(i)  
 else:  
 if i not in self.data:  
 self.data.append(i)  
 else:  
 if i not in self.data:  
 self.data.append(i)  
  
 def \_\_next\_\_(self):  
 if not self.data:  
 raise StopIteration  
 return self.data.pop(0)  
  
 def \_\_iter\_\_(self):  
 return self

**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 wrap(\*args):  
 print(func.\_\_name\_\_)  
 return\_value = func(\*args)  
 if isinstance(return\_value, list):  
 print(**'**\n**'**.join(str(value) for value in return\_value))  
 elif isinstance(return\_value, dict):  
 print(**'**\n**'**.join((str(key) + **' = '** + str(return\_value[key])) for key in return\_value.keys()))  
 else:  
 print(return\_value)  
 return return\_value  
 return wrap

**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.time = time.clock()  
 return self.time  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 print(time.clock() - self.time)

**ex\_1.py**

from liprip.gen import field  
from librip.gen import gen\_random  
  
goods = [  
 {**'title'**: **'Ковер'**, **'price'**: 2000, **'color'**: **'green'**},  
 {**'title'**: **'Диван для отдыха'**, **'price'**: 5300, **'color'**: **'black'**},  
 {**'title'**: **'Кресло'**, **'price'**: 5000, **'color'**: **'grey'**},  
 {**'title'**: **'Книжная полка'**, **'price'**: 1200, **'color'**: **'brown'**}  
]  
  
# Реализация задания 1  
# str(...)[1:-1] - срез нач. с индекса 1 по -1 не включая его (убираем [])  
print(str(list(field(goods, **'title'**, **'price'**)))[1:-1])  
print(str(list(gen\_random(1, 4, 10)))[1:-1])

**ex\_2.py**

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(str(list(Unique(data1)))[1:-1])  
print(str(list(Unique(data2)))[1:-1])  
print(str(list(Unique(data3)))[1:-1])  
print(str(list(Unique(data3, ignore\_case=True)))[1:-1])

**ex\_3.py**

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
# Реализация задания 3  
print(sorted(data, key=lambda x: abs(x)))

**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**

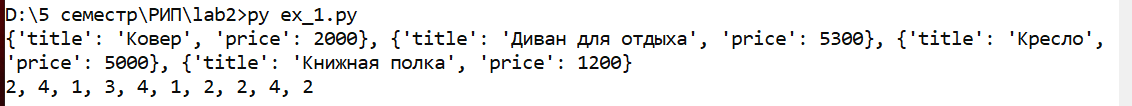
from time import sleep  
from librip.ctxmngrs import timer  
  
with timer():  
 sleep(5.5)

**ex\_6.py**

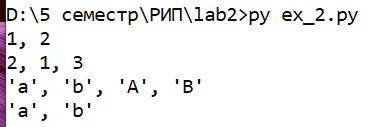
import json  
import sys  
from librip.ctxmngrs import timer  
from librip.decorators import print\_result  
from librip.gen import field, gen\_random  
from librip.iterators import Unique  
  
path = sys.argv[1]  
  
# Здесь необходимо в переменную path получить  
# путь до файла, который был передан 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(Unique(field(arg, **'job-name'**), ignore\_case=True))  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda x: x.lower().startswith(**'программист'**), arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + **' с опытом Python'**, arg))  
  
  
@print\_result  
def f4(arg):  
 salary = list(gen\_random(100000, 200000, len(arg)))  
 jobs = list(zip(arg, salary))  
 return list(map(lambda x: x[0] + **', зарплата '** + str(x[1]) + **' руб.'**, jobs))  
  
  
with timer():  
 f4(f3(f2(f1(data))))

Результаты выполнения

Выполнение ex\_1.py:



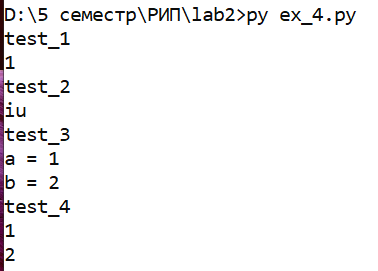
Выполнение ex\_2.py:



Выполнение ex\_3.py:



Выполнение ex\_4.py:

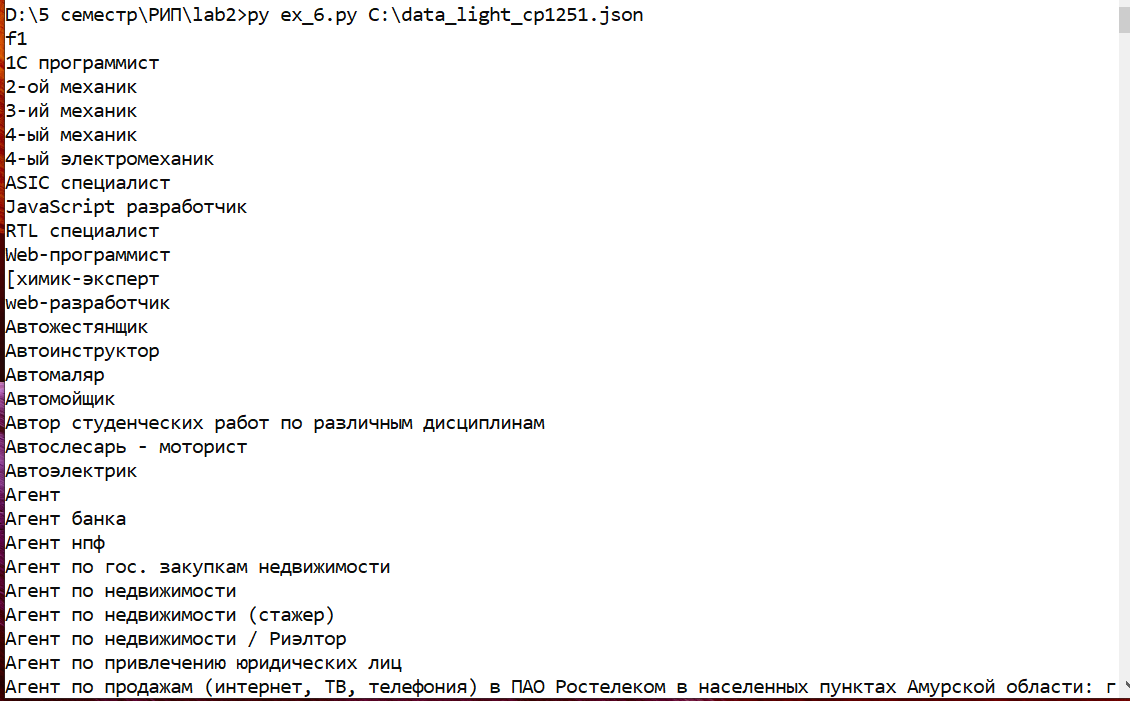


Выполнение ex\_5.py:

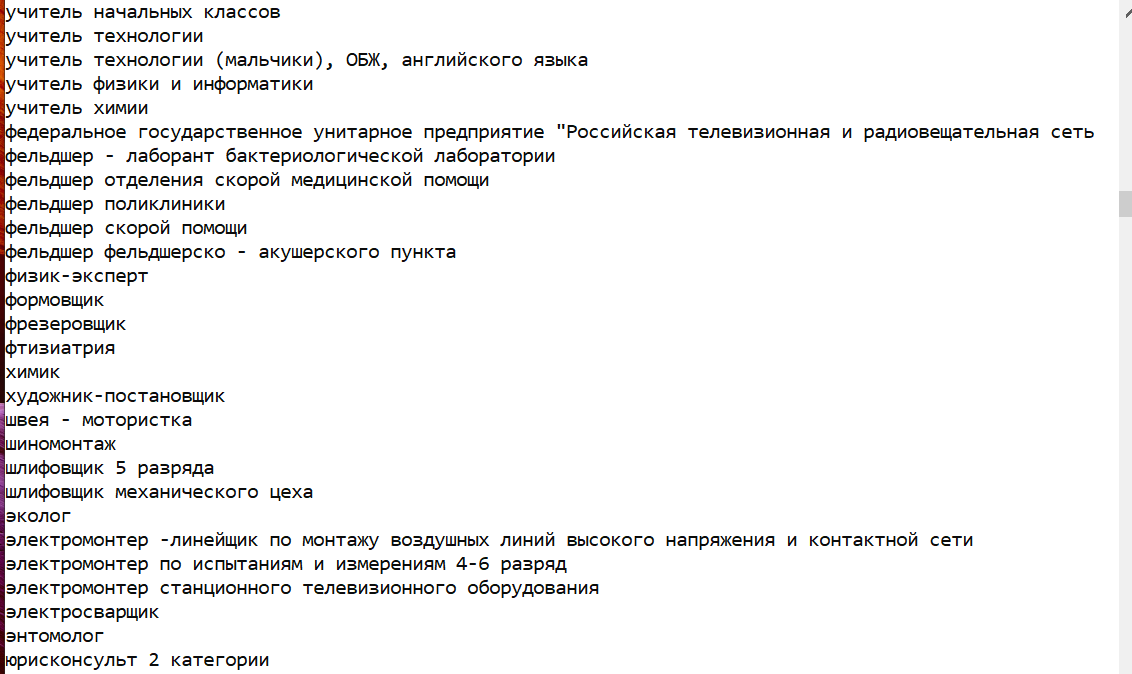


Выполнение ex\_6.py:

Начало вывода функции f1:



Конец вывода функции f1:



Функции f2-f4:

