Московский государственный технический университет им. Н.Э. Баумана Факультет «Информатика и системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»



**Отчет по Лабораторной работе 4**

**По курсу**

**“Разработка интернет-приложений”**

Выполнила:

Бекбусинова Б.Б.

Студентка группы ИУ5-52

Москва 2017

**Задание:**

***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 ​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

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.

**Код программы:**

Модуль librip/gens.py

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  
 if len(args) == 1:  
 for el in items:  
 for key, value in el.items():  
 if (key == args[0]) and (type(value) is not None):  
 yield value  
 else:  
 for el in items:  
 dict1 = dict()  
 for key, value in el.items():  
 for arg in args:  
 if (key == arg) and (type(value) is not None):  
 dict1[key] = value  
 yield dict1  
  
  
# 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 i in range(num\_count):  
 yield random.randint(begin, end)

Модуль librip/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  
 # 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  
 if type(items) == list:  
 self.items = items  
 else:  
 self.items = list(items)  
 if len(kwargs) == 0:  
 self.ignore\_case = False  
 else:  
 self.ignore\_case = kwargs['ignore\_case']  
  
 i = 0  
 while i < len(self.items):  
 j = i+1  
 while (j >= i) and (j < len(self.items)):  
 if self.items[i] == self.items[j]:  
 self.items.pop(j)  
 j -= 1  
 elif (type(self.items[i]) == str) and (type(self.items[j] == str)) and self.ignore\_case:  
 a = str(self.items[i]).lower()  
 b = str(self.items[j]).lower()  
 if a == b:  
 self.items.pop(j)  
 j -= 1  
 j += 1  
 i += 1  
 self.index = -1  
  
 def \_\_next\_\_(self):  
 if self.index == len(self.items)-1:  
 raise StopIteration  
 else:  
 self.index += 1  
 return self.items[self.index]  
  
 def \_\_iter\_\_(self):  
 return self

Модуль librip/decorators.py:

def print\_result(func\_to\_decorate):  
 def decorated\_func(\*args):  
 print(func\_to\_decorate.\_\_name\_\_)  
 res = func\_to\_decorate(\*args)  
 if type(res) == list:  
 print('\n'.join([str(x) for x in res]))  
 elif type(res) == dict:  
 print('\n'.join([str(x) + ' = ' + str(res[x]) for x in res.keys()]))  
 else:  
 print(res)  
 return res  
 return decorated\_func

Модуль librip/ctxmngrs.py

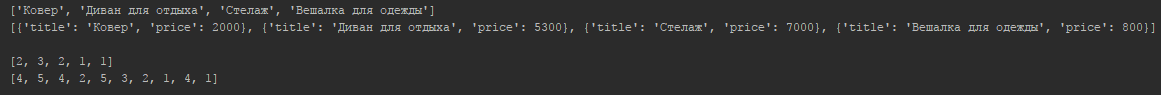
import contextlib  
import time  
  
@contextlib.contextmanager  
def timer():  
 t = time.clock()  
 yield  
 print(time.clock()-t)

Модули для проверки работы и результаты выполнения:

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()  
print(list(gen\_random(1, 3, 5)))  
print(list(gen\_random(1, 5, 10)))

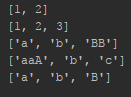
Результат:



ex\_2.py

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, 4, 10)  
data3 = ['a', 'A', 'b', 'BB', 'bB']  
data4 = ['aaA', 'aAa', 'Aaa', 'b', 'c', 'C']  
data5 = ['a', 'b', 'a', 'B']  
  
# Реализация задания 2  
print([x for x in Unique(data1)])  
print([x for x in Unique(data2)])  
print([x for x in Unique(data3, ignore\_case=True)])  
print([x for x in Unique(data4, ignore\_case=True)])  
print([x for x in Unique(data5)])

Результат:



ex\_3.py

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

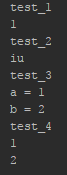
Результат:



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()

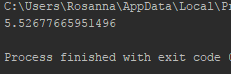
Результат:



ex\_5.py

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

Результат:



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 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)  
  
@print\_result  
def f1(data):  
 return sorted(list(unique(field(data, 'job-name'), ignore\_case=True)), key=str.lower)  
  
@print\_result  
def f2(f1):  
 return list(filter(lambda s: (s.startswith('программист') or s.startswith('Программист')), f1))  
  
@print\_result  
def f3(f2):  
 return list(map(lambda x: x + ' с опытом Python', f2))  
  
@print\_result  
def f4(f3):  
 zp = [', зарплата ' + str(x) + ' руб.' for x in gen\_random(100000, 200000, len(f3))]  
 return ['{0}{1}'.format(a, b) for a, b in zip(f3, zp)]  
  
  
with timer():  
 f4(f3(f2(f1(data))))

Результат:

