**Отчёт по лабораторной работе №3 по курсу «Разработка интернет приложений»**

Тема: «Python. Функциональные возможности»

Выполнил: студент группы ИУ5-53

Андреев Алексей Андреевич

Дата: 08/10/2018 Подпись:\_\_\_\_\_\_\_

Проверил: Гапанюк Ю. Е.

Дата: \_\_\_\_\_\_ Подпись:\_\_\_\_\_\_\_

Москва, 2018

Задание лабораторной работы

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'}

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

Исходный код

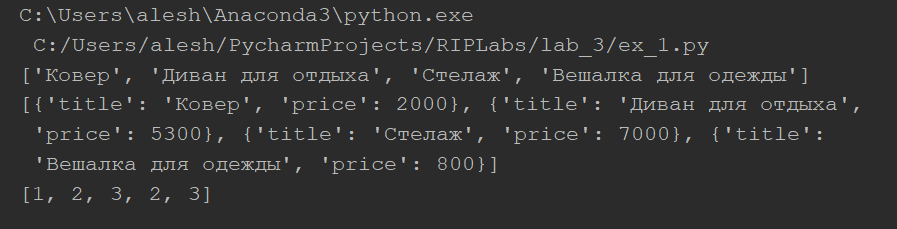
librip/gens.py

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  
 # Необходимо реализовать генератор  
 for x in items:  
 if(len(args) == 1):  
 yield x.get(args[0])  
 else:  
 bass = {}  
 for y in args:  
 bass.update({y : x[y]})  
 yield bass  
  
  
# 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):  
 pass  
 # Необходимо реализовать генератор  
 for i in range(num\_count):  
 yield random.randint(begin, end)  
  
#print(list(gen\_random(1,3,5)))

ex\_1.py

#!/usr/bin/env python3  
from librip.gens import field, 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)))

Результат выполнения



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

Исходный код

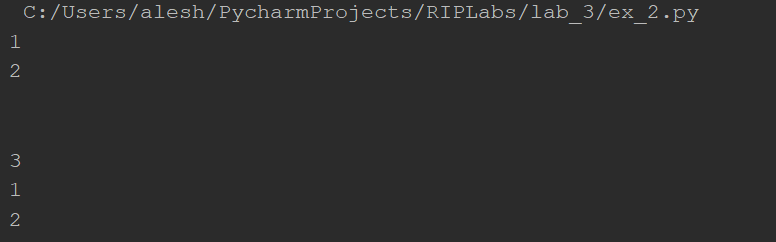
librip/iterators.py

# Итератор для удаления дубликатов  
import itertools  
  
from setuptools.package\_index import unique\_everseen  
  
  
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  
 self.counter = 0  
 if not "ignore\_case" in kwargs:  
 kwargs["ignore\_case"] = False  
 if "ignore\_case" not in kwargs:  
 self.bass = list(unique\_everseen(items, str.lower))  
 pass  
 else:  
 self.bass = list(unique\_everseen(items))  
  
  
  
 def \_\_next\_\_(self):  
 if self.counter != len(self.bass):  
 self.counter += 1  
 return self.bass[self.counter - 1]  
 else:  
 raise StopIteration  
 pass  
  
 def \_\_iter\_\_(self):  
 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)  
  
# Реализация задания 2  
for i in Unique(data1):  
 print(i)  
  
print('\n')  
for i in Unique(data2):  
 print(i)

Результат работы



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]

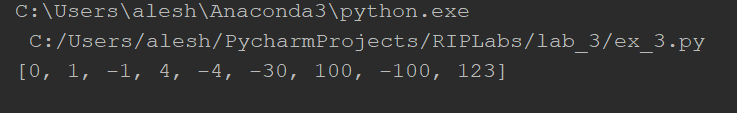
Вывод: [0, 1, -1, 4, -4, -30, 100, -100, 123]

Исходный код

ex\_3.py

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

Результат работы



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

Исходный код

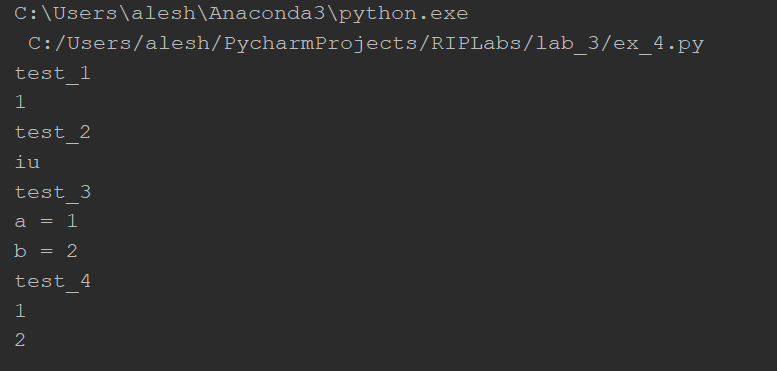
librip/decorators.py

def print\_result(func):  
 def inner\_func(\*args, \*\*kwargs):  
 print(func.\_\_name\_\_)  
 result = func(\*args, \*\*kwargs)  
 if type(result) is list:  
 for i in result:  
 print(i)  
 elif type(result) is dict:  
 for i in result.keys():  
 print(f'{i} = {result[i]}')  
 else:  
 print(result)  
 return result  
 return inner\_func

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 работы



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

ctxmngts.py

from time import time  
  
  
class timer:  
 def \_\_init\_\_(self):  
 pass  
  
 def \_\_enter\_\_(self):  
 self.time = time()  
 pass  
  
 def \_\_exit\_\_(self, \*args):  
 print(time() - self.time)  
 pass

ex\_5.py

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

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 as unique  
  
path = 'data\_light.json'  
  
# Здесь необходимо в переменную path получить  
# путь до файла, который был передан при запуске  
  
with open(path, encoding="utf8") 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(field(arg, 'job-name')), key=(lambda s: s.lower()))  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda x: "Программист" in x, arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + " с опытом Python", arg))  
  
  
@print\_result  
def f4(arg):  
 return [str(i) + ", зарплата " + str(x) + " рублей." for i, x in list(zip(arg, list(gen\_random(100000, 200000, len(arg)))))]  
  
  
with timer():  
 f4(f3(f2(f1(data))))

Результат выполнения

f1

1С программист

2-ой механик

3-ий механик

4-ый механик

4-ый электромеханик

[химик-эксперт

ASIC специалист

JavaScript разработчик

RTL специалист

Web-программист

web-разработчик

web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

web-разработчик

web-разработчик

web-разработчик

web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

web-разработчик

web-разработчик

web-разработчик

web-разработчик

Web-разработчик

web-разработчик

web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

Web-разработчик

web-разработчик

web-разработчик

web-разработчик

Web-разработчик

Web-разработчик

Автожестянщик

Автоинструктор

Автомаляр

Автомойщик

автомойщик

Автомойщик

Автор студенческих работ по различным дисциплинам

автослесарь

Автослесарь

Автослесарь

Автослесарь - моторист

Автоэлектрик

Автоэлектрик

Агент

Агент

Агент

Агент банка

Агент нпф

Агент по гос. закупкам недвижимости

агент по гос. закупкам недвижимости

Агент по 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

агрегатчик-топливник KOMATSU

агроном

Агроном

Агроном

Агроном

агроном по защите растений

Агроном по защите растений

Агроном-полевод

агрохимик почвовед

Администратор

Администратор

Администратор

Администратор

администратор

Администратор

Администратор

Администратор

Администратор

Администратор (удаленно)

Администратор Active Directory

Администратор в парикмахерский салон

Администратор зала (предприятий общественного питания)

Администратор кофейни

Администратор на ресепшен

Администратор на телефоне

Администратор по информационной безопасности

Администратор ресторана

Администратор сайта

Администратор ярмарок выходного дня

Администратор-кассир

Аккомпаниатор на 0,5 ст.

аккумуляторщик 4 разряда

Акушерка

Акушерка

Акушерка

Акушерка

Акушерка

[Много-много строк спустя]

юрист

Юрист

Юрист

Юрист

Юрист

Юрист (cпециалист по сопровождению международных договоров, английский - разговорный)

Юрист волонтер

Юристконсульт

f2

Программист

Программист

Программист

Программист / Senior Developer

Программист 1С

Программист 1С

Программист C#

Программист С++

Программист С++/С#/Java

Программист/ Junior Developer

Программист/ технический специалист

Программистр-разработчик информационных систем

f3

Программист с опытом Python

Программист с опытом Python

Программист с опытом Python

Программист / Senior Developer с опытом Python

Программист 1С с опытом Python

Программист 1С с опытом Python

Программист C# с опытом Python

Программист С++ с опытом Python

Программист С++/С#/Java с опытом Python

Программист/ Junior Developer с опытом Python

Программист/ технический специалист с опытом Python

Программистр-разработчик информационных систем с опытом Python

f4

Программист с опытом Python, зарплата 159715 рублей.

Программист с опытом Python, зарплата 120477 рублей.

Программист с опытом Python, зарплата 113965 рублей.

Программист / Senior Developer с опытом Python, зарплата 102645 рублей.

Программист 1С с опытом Python, зарплата 196559 рублей.

Программист 1С с опытом Python, зарплата 106132 рублей.

Программист C# с опытом Python, зарплата 116504 рублей.

Программист С++ с опытом Python, зарплата 108626 рублей.

Программист С++/С#/Java с опытом Python, зарплата 198083 рублей.

Программист/ Junior Developer с опытом Python, зарплата 161018 рублей.

Программист/ технический специалист с опытом Python, зарплата 141059 рублей.

Программистр-разработчик информационных систем с опытом Python, зарплата 160506 рублей.

0.025973081588745117