**Московский государственный технический университет им. Н.Э. Баумана.**

### Факультет «Информатика и системы управления»

### Кафедра ИУ5. Курс «Разработка интернет приложений»

### Отчет по лабораторной работе №3

**«Python. Функциональные возможности»**

### Выполнил:

### студент группы ИУ5-54

### Геништа М.А. Подпись и дата:

### Проверил:

### доцент каф. ИУ5 Гапанюк Ю. Е.

### Подпись и дата:

### Москва, 2017 г.

#### Оглавление

[Задание 3](#_TOC_250018)

[Задача 1 (ex\_1.py) 3](#_TOC_250017)

[Задача 2 (ex\_2.py) 3](#_TOC_250016)

[Задача 3 (ex\_3.py) 4](#_TOC_250015)

[Задача 4 (ex\_4.py) 4](#_TOC_250014)

[Задача 5 (ex\_5.py) 4](#_TOC_250013)

[Задача 6 (ex\_6.py) 5](#_TOC_250012)

[Исходный код 5](#_TOC_250011)

[librib/gens.py 5](#_TOC_250010)

[ex\_1.py 6](#_TOC_250009)

[librip/iterators.py 7](#_TOC_250008)

[ex\_2.py 7](#_TOC_250007)

[ex\_3.py 8](#_TOC_250006)

[librip/decorators.py 8](#_TOC_250005)

[ex\_4.py 9](#_TOC_250004)

[librip/ctxmngrs.py 9](#_TOC_250003)

[ex\_5.py 9](#_TOC_250002)

[ex\_6.py 9](#_TOC_250001)

[Результаты работы 11](#_TOC_250000)

# 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 еѐ, 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 еѐ к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб.

#### Используйте zip для обработки пары специальность — зарплата

# Исходный код

## librib/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 if(len(args)==1):

for item in items: try:

if item[args[0]] is not None: yield item[args[0]]

except: pass

else:

for item in items: line = {}

for arg in args: try:

if item[arg] is not None: line[arg] = item[arg]

except: pass

if len(line) > 0: yield line

# 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): # Необходимо реализовать генератор for x in range(0, num\_count):

yield random.randint(begin, end)

## ex\_1.py

#!/usr/bin/env python3 import os

import sys

from librip.gens import \* 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': None}#, 'color': 'white'}

]

# Реализация задания 1

#for f in field(goods,'color'):#,'title'): # print(f)

#print(', '.join(map(str,field(goods,'color')))) print('field generator:')

print(', '.join(map(str,field(goods,'color','title')))) print()

print('random generator:')

print(', '.join(map(str,gen\_random(1,3,5))))

## 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

self.items = items self.ignore\_case = False if 'ignore\_case' in kwargs:

self.ignore\_case = kwargs['ignore\_case'] self.returned = set()

def next (self):

# Нужно реализовать \_\_next for item in self.items:

if type(item) == str and self.ignore\_case is True: if item.lower() not in self.returned:

self.returned.add(item.lower()) return item

else:

if item not in self.returned: self.returned.add(item) return item

raise StopIteration()

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) data3 = ['a','A','b','B']

# Реализация задания 2 print('data:',data1)

print('unique:',', '.join(map(str,Unique(data1)))) print()

print('data:',data2)

print('unique:',', '.join(map(str,Unique(data2)))) print()

print('data:',data3,'ignore\_case:',True)

print('unique:',', '.join(map(str,Unique(data3,ignore\_case=True)))) print()

print('data:',data3,'ignore\_case:',False)

print('unique:',', '.join(map(str,Unique(data3,ignore\_case=False)))) print()

## 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)))

## librip/decorators.py

# Здесь необходимо реализовать декоратор, print\_result который принимает на вход функцию, # вызывает еѐ, 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

# Пример из ex\_4.py: # @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

def print\_result(func): def new\_func(\*args):

res = func(\*args) print(func. name ) if type(res) == list:

[print(x) for x in res] elif type(res) == dict:

[print(key,'=',value) for key,value in res.items()] else:

print(res) return(res)

return new\_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()

## librip/ctxmngrs.py

# 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 from time import time

class timer():

def enter (self): self.start = time()

def exit (self, exc\_type, exc\_val, exc\_tb): print(time()-self.start)

## ex\_5.py

from time import sleep

from librip.ctxmngrs import timer

with timer(): sleep(5.5)

## 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

try:

path = sys.argv[1] except IndexError:

raise ValueError('Path to data file is not specified')

# 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

#for k,v in data[0].items(): # print(k,v)

@print\_result def f1(arg):

return sorted(unique(field(arg,'job-name'),ignore\_case=True),key=lambda x: x.lower()) # raise NotImplemented

@print\_result def f2(arg):

return list(filter(lambda x:x.lower().startswith('программист'),arg)) # raise NotImplemented

@print\_result def f3(arg):

return list(map(lambda x: x+" с опытом Python",arg)) # raise NotImplemented

@print\_result def f4(arg):

rand = gen\_random(100000,200000,len(arg))

return list(map(lambda s:s[0]+', с зарплатой '+str(s[1]),zip(arg,rand))) # raise NotImplemented

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

# Результаты работы









