Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»

****

**Лабораторная работа по курсу РИП №3**

**«Python. Функциональные возможности»**

Студент группы ИУ5 -53:

Алтунин Н.С. \_\_\_\_\_\_\_\_\_

Преподаватель:

Гапанюк Ю.Е. \_\_\_\_\_\_\_\_\_\_

**Москва 2016**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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 реализовывать, одной строкой.

**Исходный код**

**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'}

]

getr = gen\_random(1,5,7)

print(\*list(getr), sep=' ')

getf = field(goods, 'title')

print(\*list(getf), sep=', ')

**ex\_2.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.ignore\_case = False if (kwargs.get('ignore\_case') == None) else kwargs.get('ignore\_case')

self.seen = []

self.items = iter(items)

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

# Нужно реализовать \_\_next\_\_

while True:

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

val\_cmp = val if self.ignore\_case == False else str(val).lower()

if val\_cmp not in self.seen:

self.seen.append(val\_cmp)

return val

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

return self

**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 item: abs(item)))

**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**

#!/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 = None

# Здесь необходимо в переменную path получить

# путь до файла, который был передан при запуске

path = sys.argv[1]

print(path)

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(data):

return sorted(list(unique(list(field(data, 'job-name')),ignore\_case = True)))

@print\_result

def f2(data):

return list(filter(lambda x: x[0:11].lower() == 'программист', data))

@print\_result

def f3(data):

return list(map(lambda x: x+' с опытом Python', data))

@print\_result

def f4(data):

return(list('{} c зарплатой {} рублей'.format(prof, salary) for prof,salary in zip(data, gen\_random(100000, 200000, len(data)))))

with timer():

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

**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 \*

import contextlib

@contextlib.contextmanager

def timer():

time\_start = time();

yield

time\_end = time();

print(time\_end - time\_start)

**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

# 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\_to\_decorate):

def decorated\_func(\*args):

res = func\_to\_decorate(\*args)

if type(res) == list:

print(func\_to\_decorate.\_\_name\_\_, \*list(res), sep='\n')

elif type(res) == dict:

print(func\_to\_decorate.\_\_name\_\_ , '\n'.join('{}={}'.format(k[0],k[1]) for k in res.items()), sep='\n')

else:

print(func\_to\_decorate.\_\_name\_\_, res, sep='\n')

return res

return decorated\_func

**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

for item in items:

if len(args) == 1:

val = item.get(args[0])

if val != None:

yield val

else:

res = {};

for item\_arg in item:

if (item\_arg in args) and (item[item\_arg] != None):

res[item\_arg] = item[item\_arg]

if res != {}:

yield res

# Необходимо реализовать генератор

# 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 list(range(num\_count)):

yield random.randint(begin,end)

**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.ignore\_case = False if (kwargs.get('ignore\_case') == None) else kwargs.get('ignore\_case')

self.seen = []

self.items = iter(items)

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

# Нужно реализовать \_\_next\_\_

while True:

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

val\_cmp = val if self.ignore\_case == False else str(val).lower()

if val\_cmp not in self.seen:

self.seen.append(val\_cmp)

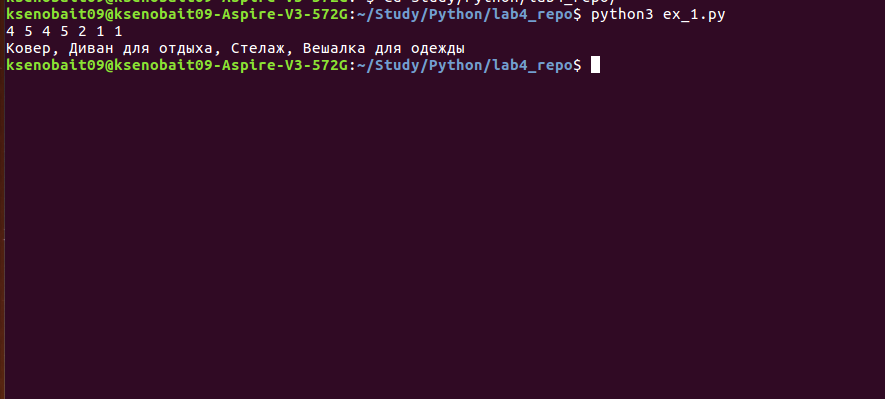
return val

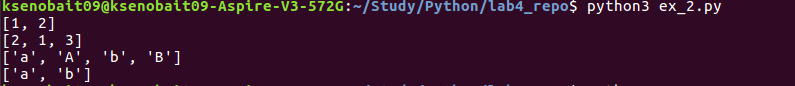
def \_\_iter\_\_(self):

return self

**Скриншоты с результатами выполнения**

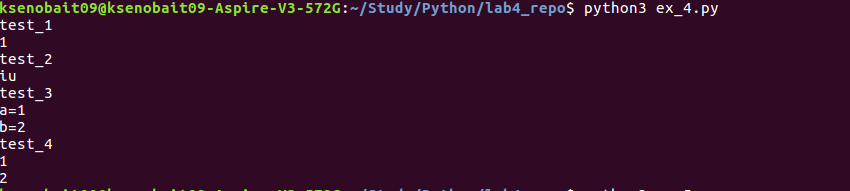
**ex\_1.py**

**ex\_2.py**

****

**ex\_3.py**

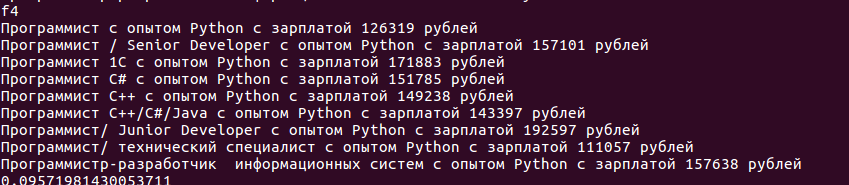
****

**ex\_4.py**

**ex\_5.py**

****

**ex\_6.py**

****