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

Факультет «Информатика и управление»

Отчёт по лабораторной работу № по курсу «Разработка интернет-приложений»

“Функциональные возможности”

Выполнил: Проверил:

студент группы ИУ5-52 преподаватель каф. ИУ5

Сергеев В.Р.

Подпись и дата: Подпись и дата:

Москва, 2017 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 реализовывать ​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\_4

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

gens.py – файл с функциями генеротора

**import random**

**def field(items, \*args):**

**assert len(args) > 0**

**for keys in items:**

**if len(args) <= 1 and keys.get(args[0]) != None:**

**yield keys[args[0]]**

**elif len(args) <= 1 and keys.get(args[0]) == None:**

**continue**

**else:**

**for zn in args:**

**if keys.get(zn) is not None:**

**yield {zn: keys.get(zn)}**

**def gen\_random(begin, end, num\_count):**

**for i in range(num\_count):**

**yield random.randint(begin,end)**

ex\_1.py – файл с проверкой работы

**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': 'white'}**

**]**

**# Реализация задания 1**

**for i in field(goods,'title','price'):**

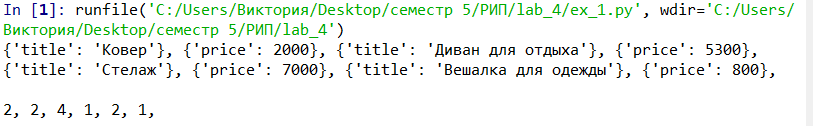
**print (i, end=', ')**

**print ('\n')**

**for i in gen\_random(1, 4, 6):**

**print(i, end=', ')**

Результат работы:



### 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.​

Iterators.py

**class Unique(object):**

**def \_\_init\_\_(self, items, \*\*kwargs):**

**self.lst = items**

**self.ignore\_case = False**

**if 'ignore\_case' in kwargs:**

**self.ignore\_case = kwargs['ignore\_case']**

**self.newlst = set()**

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

**for i in self.lst:**

**if type(i) != int:**

**if self.ignore\_case is True:**

**if i.lower() not in self.newlst:**

**self.newlst.add(i.lower())**

**return i**

**elif self.ignore\_case is False :**

**if i not in self.newlst:**

**self.newlst.add(i)**

**return i**

**else:**

**if i not in self.newlst:**

**self.newlst.add(i)**

**return i**

**raise StopIteration()**

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

**return self**

ex\_2.py - проверка работы итератора

**from librip.gens import gen\_random**

**from librip.iterators import Unique**

**data0 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]**

**data1 = gen\_random(1, 3, 10)**

**# Реализация задания 2**

**data2= ['a', 'A', 'A', 'a', 'C', 'a', 'a', 'b', 'b', 'b', 'b', 'b']**

**d0 = Unique(data0)**

**for x in d0:**

**print(x, end=', ')**

**print('\n')**

**d1 = Unique(list(data1))**

**for x in d1:**

**print(x, end=', ')**

**print('\n')**

**d2 = Unique(data2, ignore\_case=True)**

**for x in d2:**

**print(x, end=', ')**

**print('\n')**

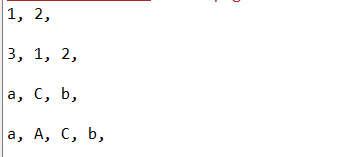
**d3 = Unique(data2)**

**for x in d3:**

**print(x, end=', ')**

**print('\n')**

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

Ex\_3.py

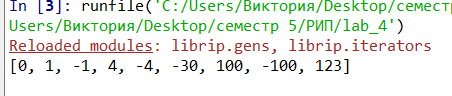
**import math**

**data = [4, -30, 100, -100, 123, 1, 0, -1, -4]**

**newdata=sorted(data, key = lambda x: math.fabs(x))**

**print (newdata)**

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

decorators.py

**def func\_print(items):**

**if type(items) == list:**

**for i in items:**

**print(i)**

**elif type(items) == dict:**

**for key in items:**

**print('%s = %s' % (key, items[key]))**

**else:**

**print(items)**

**return items**

**def print\_result(your\_func):**

**def get\_arg(\*args, \*\*kwargs):**

**print(your\_func.\_\_name\_\_)**

**res = func\_print(your\_func(\*args, \*\*kwargs))**

**return res**

**return get\_arg**

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]**

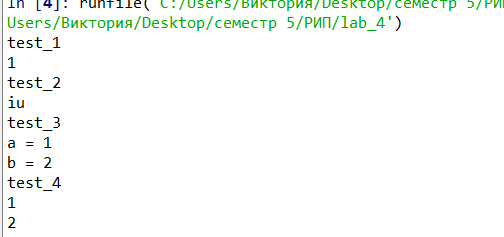
**test\_1()**

**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)

**ctxmngrs.py**

**import time**

**class timer:**

**def \_\_enter\_\_(self):**

**self.start = time.time()**

**self.date=self.start**

**return self**

**def \_\_exit\_\_(self, \*args):**

**self.end = time.time()**

**self.sec = self.end - self.start**

**print("with timer(): %s" % self.sec)**

ex\_5.py

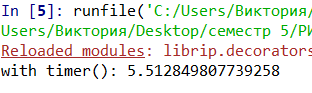
**from time import sleep**

**from librip.ctxmngrs import timer**

**with timer():**

**sleep(5.5)**

Результат работы:



### 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

**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\_cp1251.json"**

**with open(path) as f:**

**data = json.load(f)**

**@print\_result**

**def f1(arg):**

**return sorted(list(unique(list(field(arg, "job-name")), ignore\_case=True)))**

**@print\_result**

**def f2(arg):**

**return list(filter(lambda x: x.startswith("Программист"), arg))**

**@print\_result**

**def f3(arg):**

**return list(map(lambda x: x + " с опытом Python", arg))**

**@print\_result**

**def f4(arg):**

**salaries = list(gen\_random(100000, 200000, len(arg)))**

**for name, salary in zip(arg, salaries):**

**print(name, ", зарплата ", salary)**

**with timer():**

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

Результат работы:

