**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Системы обработки информации и управления»

Кафедра ИУ5. Курс «РИП»

Отчет по лабораторным работе №4

«Python. Функциональные возможности»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-53 |  | преподаватель каф. ИУ5 |
| Белков А.Д. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2017 г.

**Задание и порядок выполнения ЛР №3**

**Важно** выполнять все задачи последовательно . С 1 по 5 задачу формируется модуль l ibrip , 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 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 *о* *дной строкой* Генераторы должны располагаться в l ibrip/ 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 **н** **е 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

*МГТУ им. Н. Э. Баумана, кафедра ИУ5, курс РИП*

*ЛР No4: Python, функциональные возможности*

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 *о* *дной строкой*. **Важно** продемонстрировать работу как с массивами, так и с генераторами (g en\_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

*МГТУ им. Н. Э. Баумана, кафедра ИУ5, курс РИП*

*ЛР No4: Python, функциональные возможности*

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:

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 m ap .
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.

**Код программы**

**ctxmngrs.py**

from time import time  
# 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  
  
  
class timer:  
 def \_\_init\_\_(self):  
 pass  
  
 def \_\_enter\_\_(self):  
 self.time = time()  
  
 def \_\_exit\_\_(self, type, value, traceback):  
 print(time() - self.time)

**decoratos.py**

def print\_result(func, \*arg):  
  
 def decorated\_function(\*arg):  
 result = func(\*arg)  
  
 print(func.\_\_name\_\_)  
  
 if type(result) is dict:  
 for key, value in result.items():  
 print("%s=%s" % (str(key), str(value)))  
 elif type(result) is list:  
 for i in result:  
 print(i)  
 else:  
 print(result)  
 return result  
  
 return decorated\_function

**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):  
 # Необходимо реализовать генератор  
 if len(args) == 1:  
 for item in items:  
 for arg in args:  
 if arg in item:  
 yield item[arg]  
 else:  
 for item in items:  
 new\_item = {}  
 for arg in args:  
 if arg in item:  
 new\_item[arg] = item[arg]  
 if len(new\_item.keys()) > 0:  
 yield new\_item  
  
# 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  
  
 if begin > end:  
 begin, end = end, begin  
  
 for i in range(1, num\_count):  
 yield random.randint(begin, end)

**iterators.py**

# Итератор для удаления дубликатов  
class Unique(object):  
 def \_\_init\_\_(self, items, ignore\_case=False, \*\*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.unique\_items = []  
 self.ignore\_case = ignore\_case  
 self.items = iter(items)  
  
 def \_\_next\_\_(self):  
 # Нужно реализовать \_\_next\_\_  
  
 while True:  
 item = self.items.\_\_next\_\_()  
 compare\_item = None  
  
 if self.ignore\_case and type(item) is str:  
 compare\_item = item.lower()  
 else:  
 compare\_item = item  
  
 if compare\_item not in self.unique\_items:  
 self.unique\_items.append(compare\_item)  
 return item  
  
 def \_\_iter\_\_(self):  
  
 return self

**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)))

**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 = ['A', 'a', 'b', 'B']  
data3 = ['A', 'a', 'b', 'B']  
  
# Реализация задания 2  
for i in Unique(data1):  
 print(i, end=" ")  
print(" ")  
for i in Unique(data2):  
 print(i, end=" ")  
print(" ")  
for i in Unique(data3, ignore\_case=True):  
 print(i, end=" ")  
print(" ")

**ex\_3.py**

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

**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()

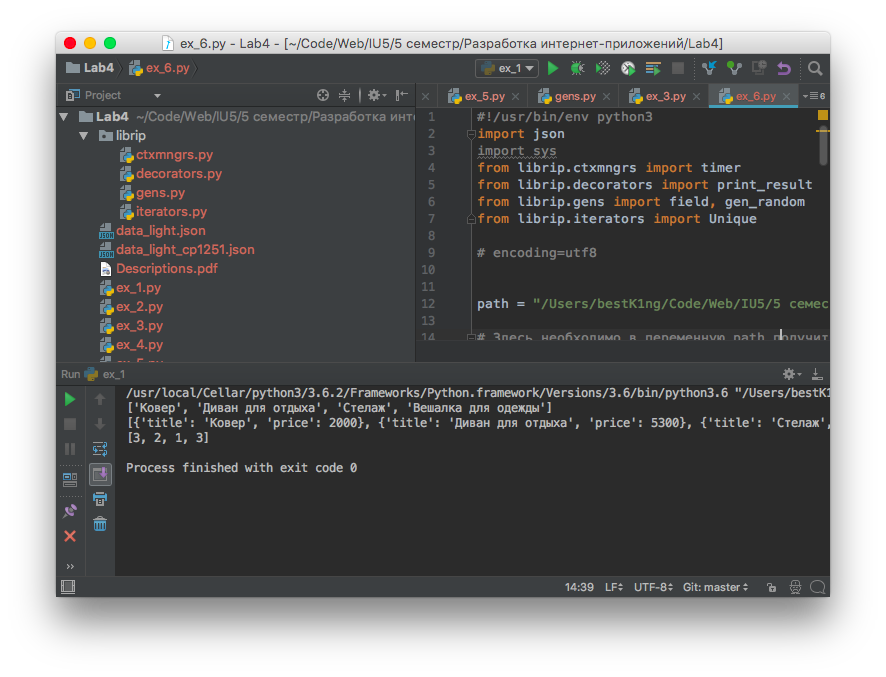
**ex\_5.py**

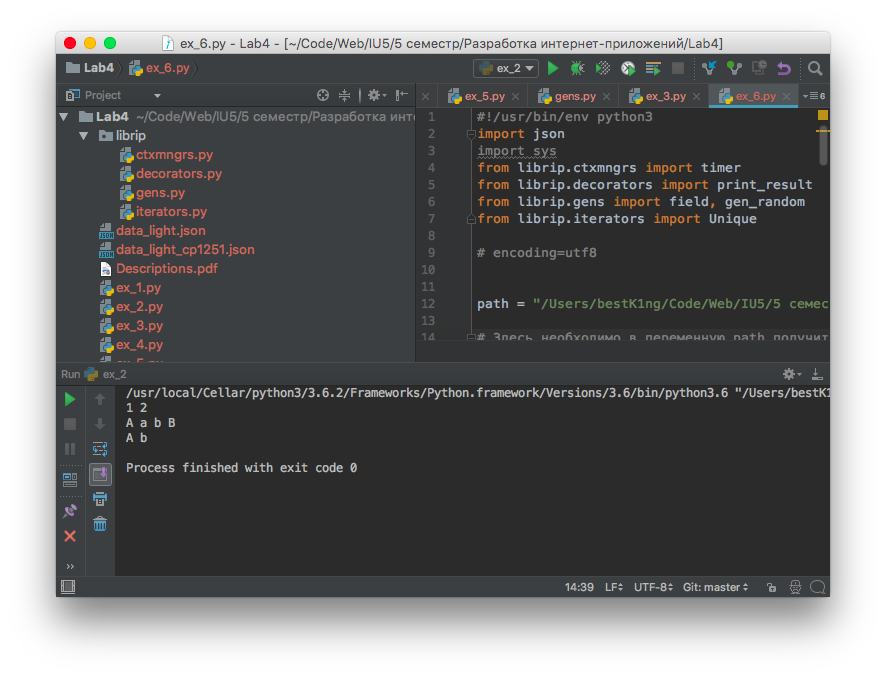
from time import sleep  
from librip.ctxmngrs import timer  
  
with timer():  
 sleep(3.23)

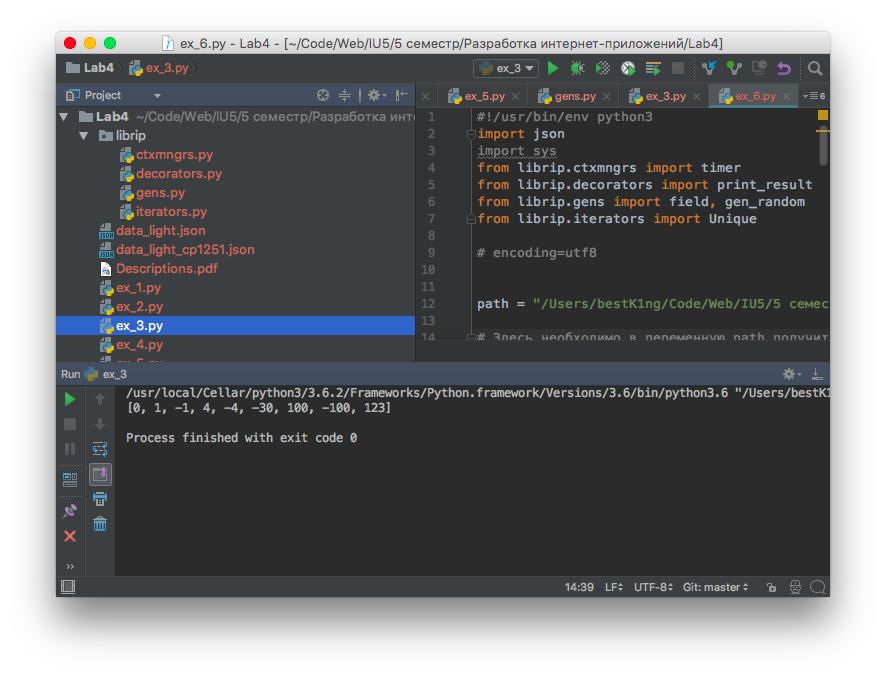
**ex\_6.py**

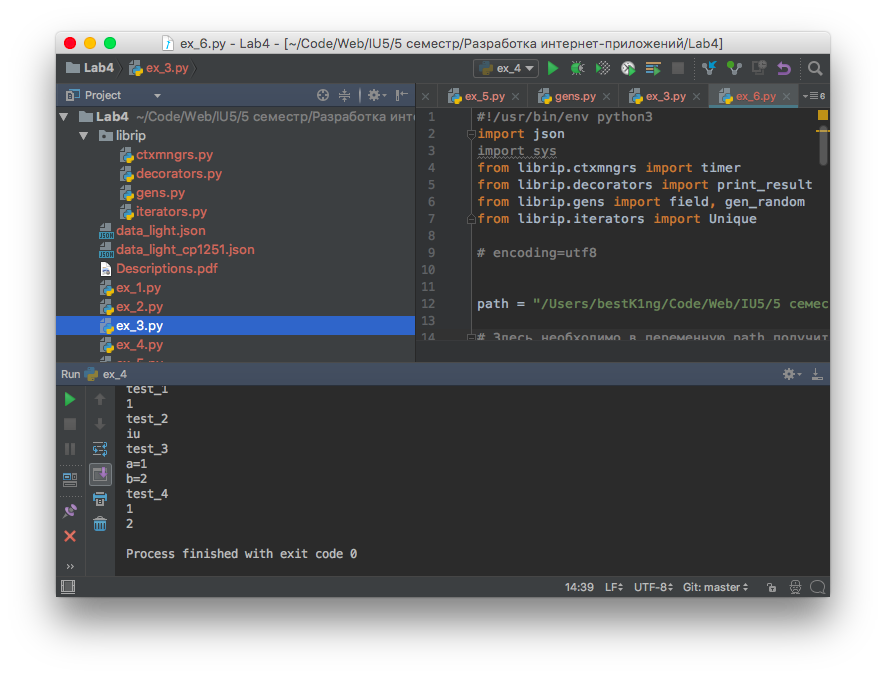
#!/usr/bin/env python3  
import json  
import sys  
from librip.ctxmngrs import timer  
from librip.decorators import print\_result  
from librip.gens import field, gen\_random  
from librip.iterators import Unique  
  
# encoding=utf8  
  
  
path = "/Users/bestK1ng/Code/Web/IU5/5 семестр/Разработка интернет-приложений/Lab4/data\_light.json"  
  
# Здесь необходимо в переменную path получить  
# путь до файла, который был передан при запуске  
  
with open(path, "r", 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):  
 jobs = list(field(arg, "job-name"))  
 jobs = Unique(jobs, ignore\_case=True)  
 jobs = sorted(jobs)  
  
 return jobs  
  
  
@print\_result  
def f2(arg):  
 jobs = list(filter(lambda x: "программист" in x.lower(), arg))  
  
 return jobs

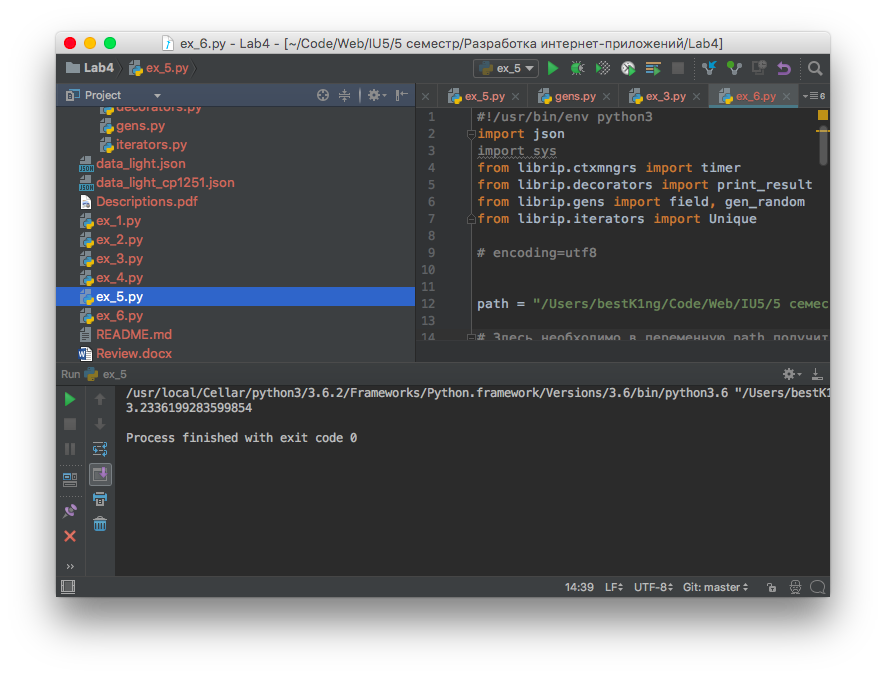
**Скришоты выполнения**

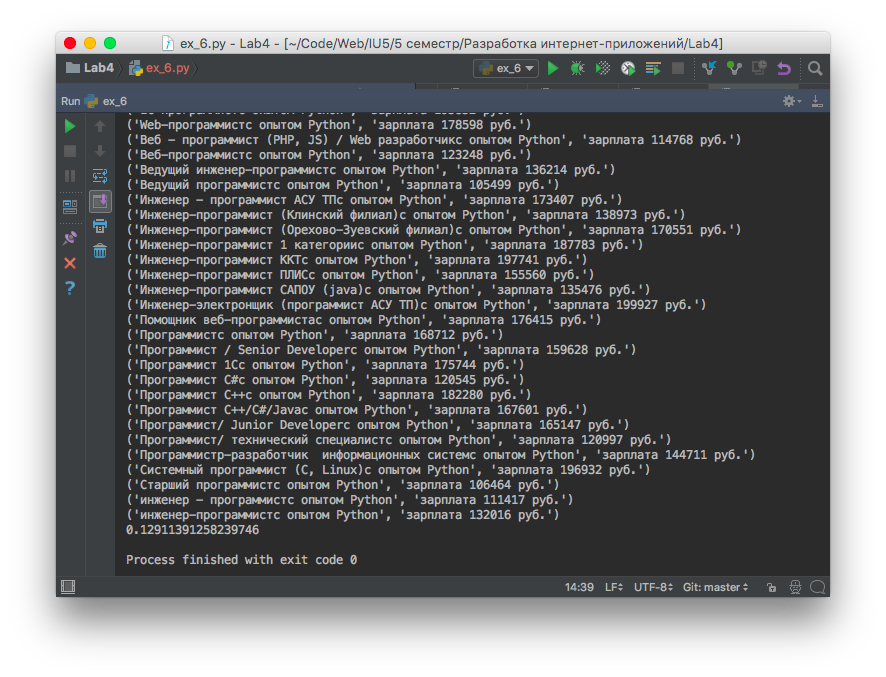
****

****

****

****

****

****