**Московский Государственный Технический Университет имени Н. Э. Баумана**



**Машинное обучение**

**Лабораторная работа №3**

**«Python. Функциональные возможности»**

ИСПОЛНИТЕЛЬ:

ФИО Гладова Анастасия

ИУ5-51 (ИУ5Ц- 71Б)

ПРЕПОДАВАТЕЛЬ:

ФИО Гапанюк Ю.Е.

**Москва 2018**

**Задание и порядок выполнения:**

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 запрограммировать 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\_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

**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 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

**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 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.

# **Обязательные условия:**

1. Форк репозитория на github
2. ex\_1.py: использование \*args
3. ex\_2.py: ипользование \*\*kwargs, поддержка работы как со списками, так и с генераторами
4. ex\_3.py: использование lambda-выражения
5. ex\_4.py: внутри декоратора печать должна быть реализована в одну строчку. Печать словарей и массивов должна выполняться в столбик
6. ex\_6.py: соблюдения кол-ва строк и использование функций, указанных в задании
7. Путь до файла передается как аргумент при запуске скрипта
8. Код на github

**Исходный код:**

**ctxmngrs.py**

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 \_\_enter\_\_(self):  
 self.start = time.time()  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 ti = (time.time()) - self.start  
 print(ti)

**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):  
 def decorated\_func(\*args):  
 if len(args) == 0:  
 result = func()  
 else:  
 result = func(args[0])  
 print(func.\_\_name\_\_)  
 if type(result) == list:  
 for i in result:  
 print(i)  
 elif type(result) == dict:  
 for key in result:  
 print(str(key) + " = " + str(result[key]))  
 else:  
 print(result)  
 return result  
  
 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}  
  
# Генератор вычленения полей из массива словарей  
def field(arr, \*args):  
 assert len(args) > 0  
 # Необходимо реализовать генератор  
 for el in arr: # где el - словарь  
 slovar = {}  
 for arg in args:  
 if (arg in el.keys()) and (len(args) == 1):  
 yield el[arg] # генератор выдает только значения полей  
 elif arg in el is not None:  
 slovar[arg] = el[arg] # формируем новый словарь,  
 # где пропускаем элементы равные None  
 if len(slovar) > 0 and len(args) > 1:  
 yield slovar  
  
# 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 i in range(num\_count):  
 yield random.randint(begin, end)

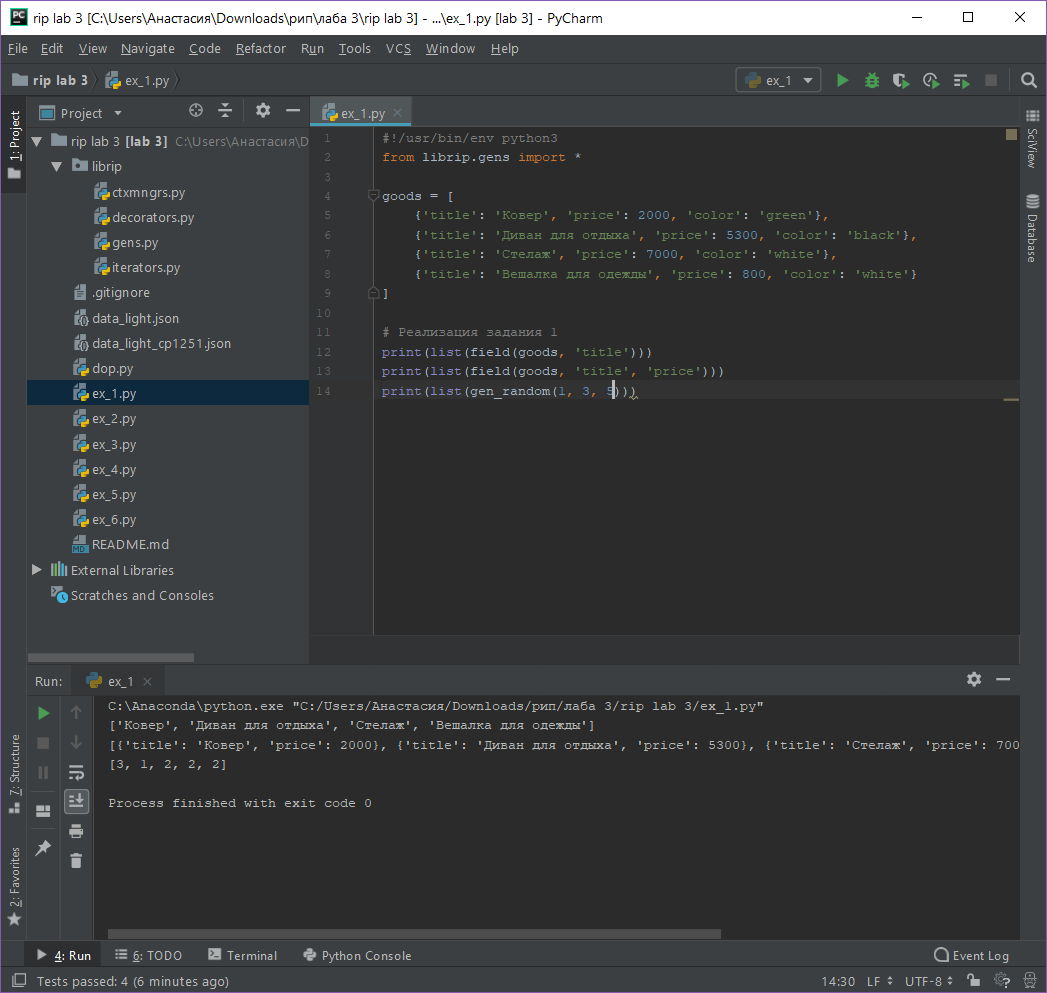
**iterators.py**

from types import GeneratorType  
  
# 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 \*  
  
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)))

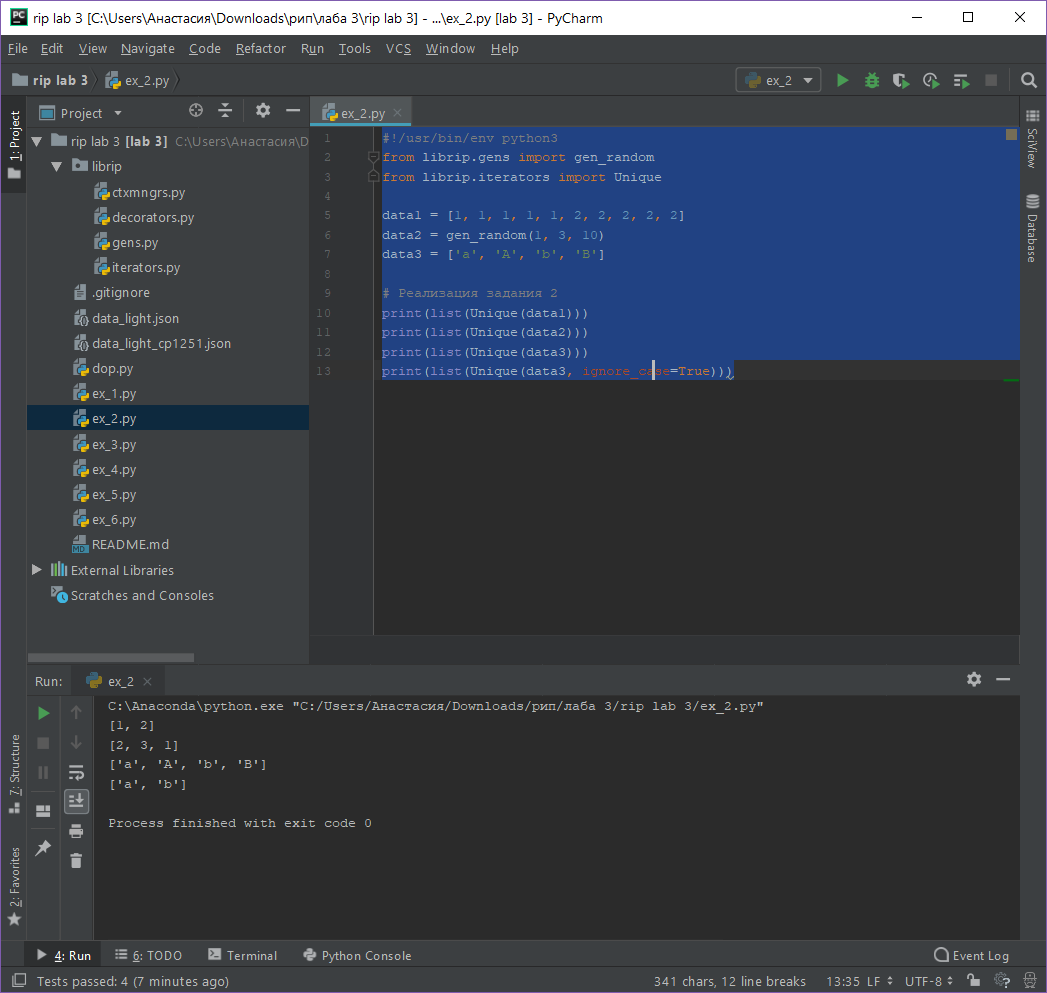
**Результат работы ex\_1.py:**



**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(list(Unique(data1)))  
print(list(Unique(data2)))  
print(list(Unique(data3)))  
print(list(Unique(data3, ignore\_case=True)))

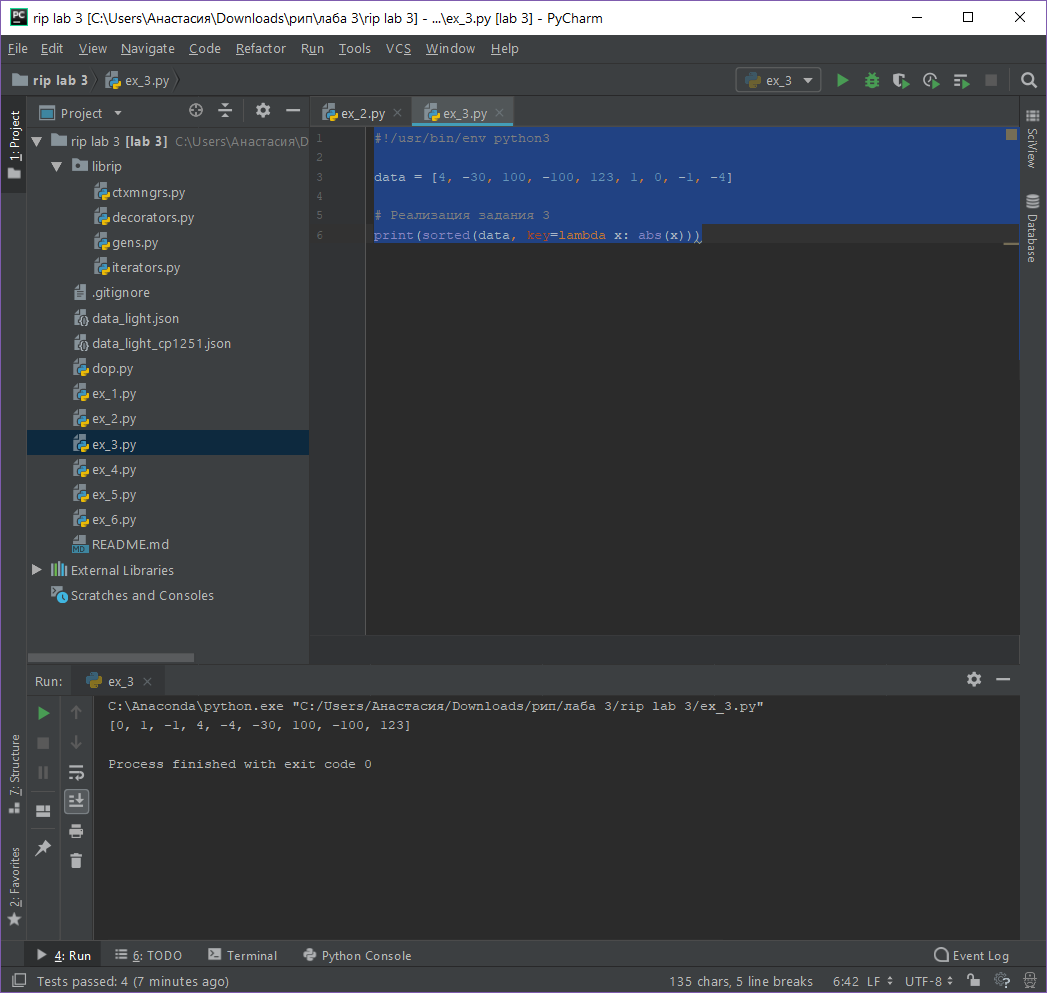
**Результат работы ex\_2.py:**



**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)))

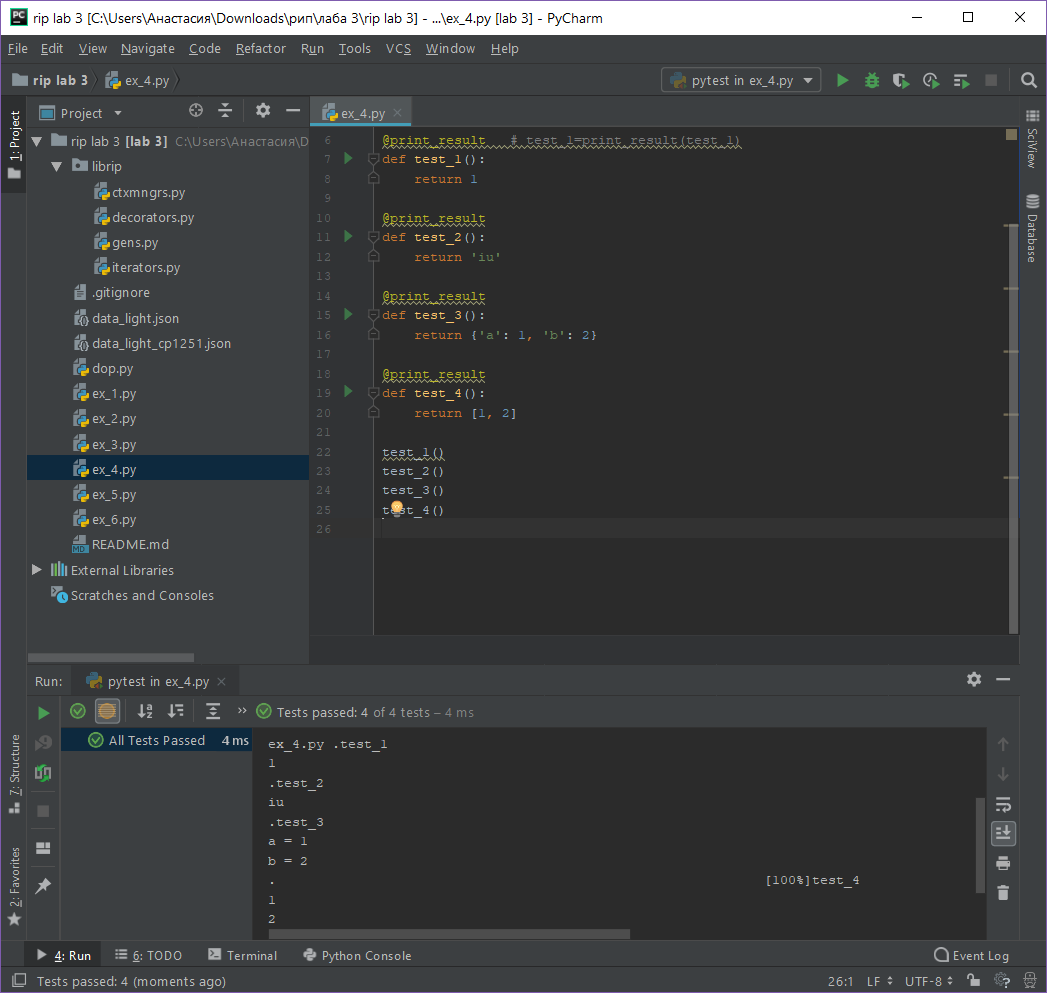
**Результат работы ex\_3.py:**



**ex\_4.py**

from librip.decorators import print\_result  
  
# Необходимо верно реализовать print\_result  
# и задание будет выполнено  
  
@print\_result # test\_1=print\_result(test\_1)  
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()

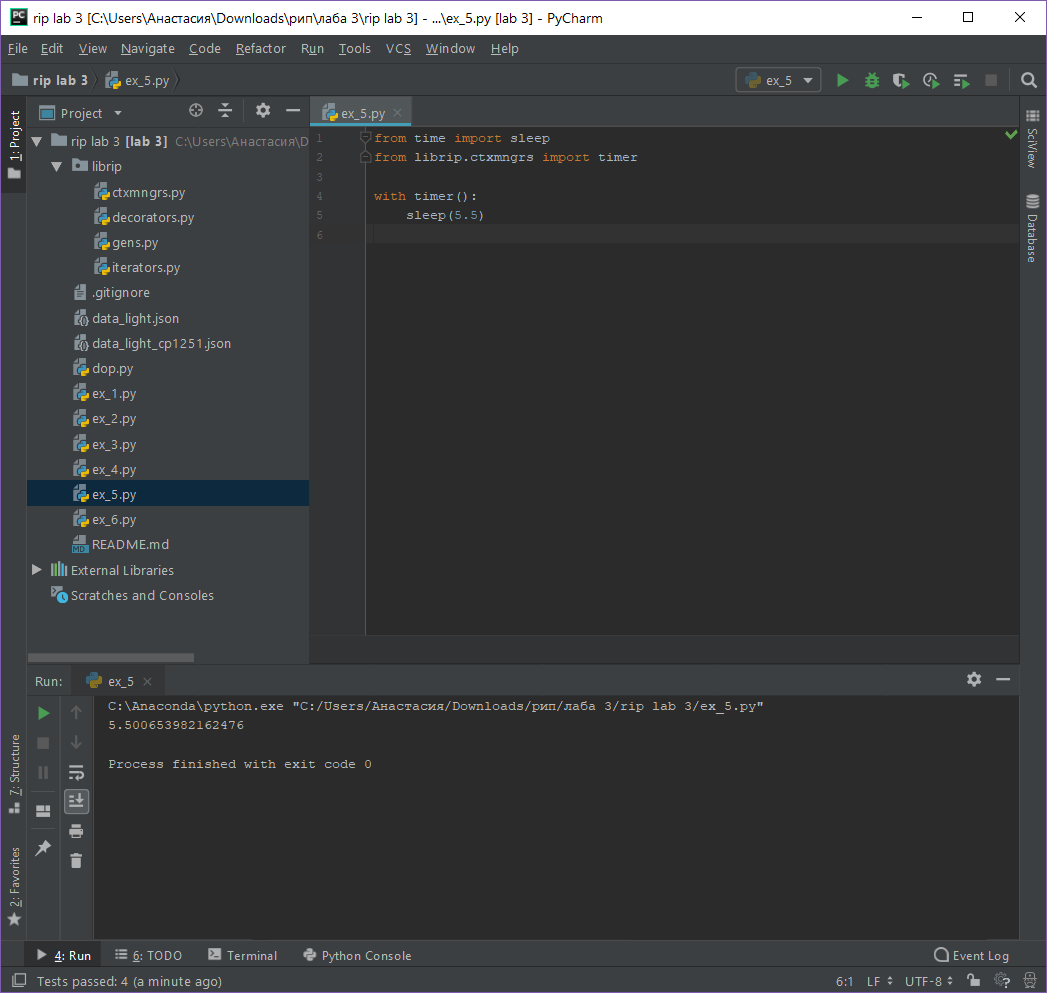
**Результат работы ex\_4.py:**



**ex\_5.py**

from time import sleep  
from librip.ctxmngrs import timer  
  
with timer():  
 sleep(5.5)

**Результат работы ex\_5.py:**



**ex\_6.py**

#!/usr/bin/env python3  
import json  
from librip.ctxmngrs import timer  
from librip.decorators import print\_result  
from librip.gens import field, gen\_random  
from librip.iterators import Unique  
  
path = "data\_light\_cp1251.json"  
  
# 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  
  
@print\_result  
def f1(arg):  
 return list(Unique(list(field(arg, "job-name")), ignore\_case=True))  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda s: "программист" in s[0:12], arg))  
  
@print\_result  
def f3(arg): # map(func, arr)  
 return list(map(lambda s: s + " с опытом Python", arg))  
  
@print\_result  
def f4(arg):  
 Sal = gen\_random(100000, 200000, len(arg))  
 return list(map(lambda s: '{}, зарплата {} руб.'.format(  
 s[0], s[1]), zip(arg, Sal)))  
  
with timer():  
 f4(f3(f2(f1(data))))

**Результат работы ex\_6.py:**

