

Защищено:
Гапанюк Ю.Е.

"__" _____ 2016 г.

Демонстрация ЛР:
Гапанюк Ю.Е.

"__" _____ 2016 г.

Отчет по лабораторной работе № 4 по курсу
Разработка интернет- приложений

Вариант № 14

ИСПОЛНИТЕЛЬ:

студент группы **ИУ5-54**

Маршанцева М.В

(подпись)

"__" _____ 2016 г.

Задание

Важно выполнять все задачи последовательно . С 1 по 5 задачу формирует 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](https://github.com/iu5team/ex-lab4) 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 *одной строкой*

Генераторы должны располагаться в `librip/ gen.py`

Исходный код:

```
#!/usr/bin/env python3
from librip.gen 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)))
```

```
Run ex_1
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab4/ex_1.py
['Ковер', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']
[{'price': 2000, 'title': 'Ковер'}, {'price': 5300, 'title': 'Диван для отдыха'}, {'price': 7000, 'title': 'Стелаж'}, {'price': 800, 'title': 'Вешалка для одежды'}]
[2, 1, 3, 2, 3]
Process finished with exit code 0
```

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`

Исходный код:

```
#!/usr/bin/env python3
from librip.gen 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)

# Реализация задания 2
print(list(Unique(data1)))
print(list(Unique(data2)))
data = ['a', 'A', 'b', 'B']
print(list(Unique(data)))
data = ['a', 'A', 'b', 'B']
print(list(Unique(data, ignore_case=True)))
```

```
Run ex_2
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab4/ex_2.py
[1, 2]
[2, 3, 1]
['a', 'A', 'b', 'B']
['a', 'b']
Process finished with exit code 0
```

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]

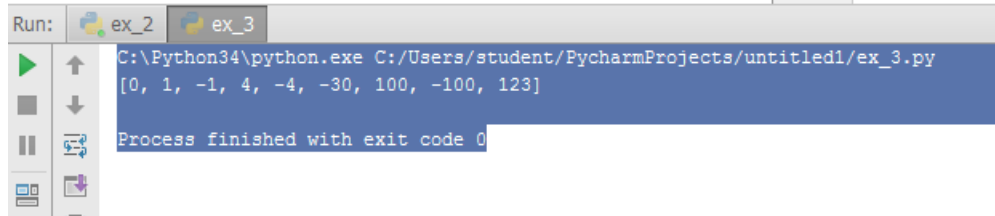
Исходный код:

```
#!/usr/bin/env python3
```

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

```
# Реализация задания 3
```

```
print(sorted(data, key = lambda *args: abs(*args)))
```



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 выводиться 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`

Исходный Код:

```
from librip.decorators import print_result

# Необходимо верно реализовать print_result
# и задание будет выполнено

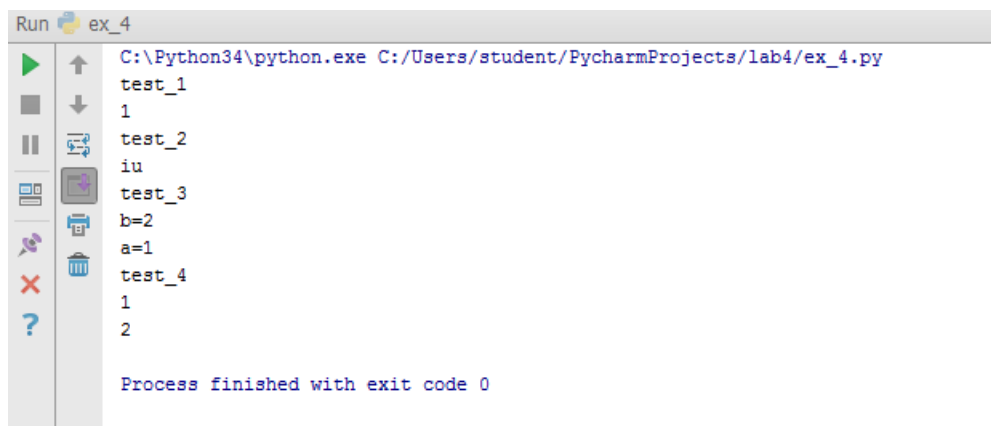
@print_result
def test_1():
    return 1

@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()
```



Задача 5 (ex_5.py)

Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран

Пример:

```
with timer():
```

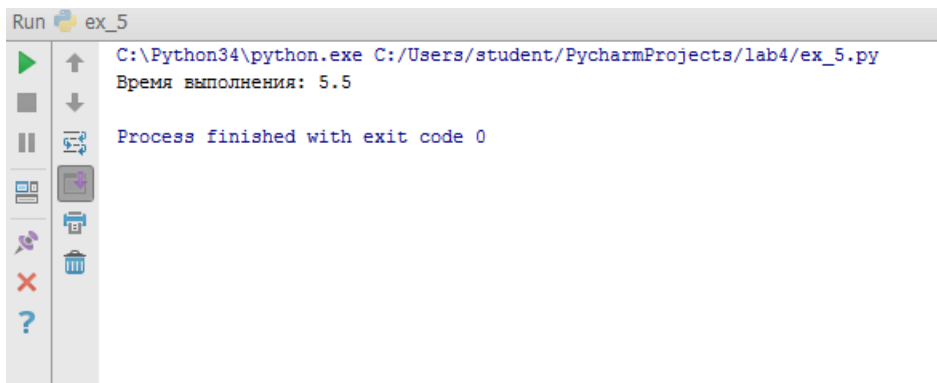
```
    sleep(5.5)
```

После завершения блока должно вывестись в консоль примерно 5.5

Исходный код:

```
from time import sleep
from librip.ctxmgrs import timer

with timer():
    sleep(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.

Исходный код:

```
#!/usr/bin/env python3
import json
import sys
from librip.ctxmgrs import timer
from librip.decorators import print_result
from librip.gen import *
from librip.iterators import Unique as unique
```

```

path = "data_light.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 должны 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 _: "Программист" in _, arg))

@print_result
def f3(arg):
    return list(map(lambda x: x + " с опытом Python", arg))

@print_result
def f4(arg):
    return list(map(lambda x: "{} зарплата {} руб.".format(x[0], x[1]),
                    zip(arg, gen_random(100000, 200000, len(arg)))))

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

```

```
Run ex_6
Специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических
Менеджер (в промышленности)
f2
Программист
Программист C++/C#/Java
Программист 1C
Программист-разработчик информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
Программист C#
f3
Программист с опытом Python
Программист C++/C#/Java с опытом Python
Программист 1C с опытом Python
Программист-разработчик информационных систем с опытом Python
Программист C++ с опытом Python
Программист/ Junior Developer с опытом Python
Программист / Senior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программист C# с опытом Python
f4
Программист с опытом Python, зарплата 115768 руб.
Программист C++/C#/Java с опытом Python, зарплата 118652 руб.
Программист 1C с опытом Python, зарплата 188795 руб.
Программист-разработчик информационных систем с опытом Python, зарплата 159630 руб.
Программист C++ с опытом Python, зарплата 137476 руб.
Программист/ Junior Developer с опытом Python, зарплата 192377 руб.
Программист / Senior Developer с опытом Python, зарплата 127423 руб.
Программист/ технический специалист с опытом Python, зарплата 166414 руб.
Программист C# с опытом Python, зарплата 119096 руб.
Время выполнения: 0.1

Process finished with exit code 0
```