

Защищено:
Гапанюк Ю.Е.

Демонстрация:
Гапанюк Ю.Е.

"__" _____ 2016 г.

"__" _____ 2016 г.

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

7
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-51

Дробышева В.О.

(подпись)

"__" _____ 2016 г.

Москва, МГТУ - 2016

1. Задача 1.

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', 'price': 5300}

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

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
```

```
    if len(args) == 1:
```

```
        for i in items:
```

```
            if args[0] in i: yield i[args[0]]
```

```
    else:
```

```
        for i in items:
```

```
            res = {}
```

```
            for j in args:
```

```
                if j in i:
```

```
                    res[j] = i[j]
```

```
            yield res
```

```
# 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 range(num_count):
```

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

ex_1.py

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

```
from librip.gens import field
```

```
from librip.gens import 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'}
]

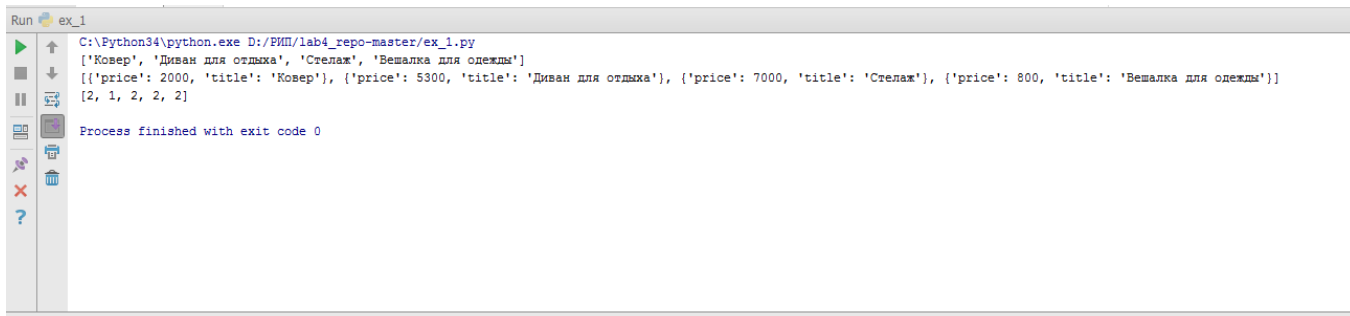
```

Реализация задания 1

```

print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price'))))
print(list(gen_random(1, 3, 5)))

```



2. Задача 2.

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`

Ex_2.py

```

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

```

for element in Unique(data1):
    print(element, end=' ')

print('\n')

for element in Unique(list(data2)):
    print(element, end=' ')

print('\n')

for element in Unique(list(data3)):
    print(element, end=' ')

```

```
print('\n')
```

```
for element in Unique(list(data3), ignore_case=True):
    print(element, end=' ')
```

iterators.py

```
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, Абв и АБВ разные строки
        # ignore case = False, Абв и АБВ одинаковые строки, одна из них
        удалится
        # По-умолчанию ignore_case = False

        # Проверка наличия ключевого аргумента 'ignore_case' и его значения
        if ('ignore_case' in kwargs.keys()) and (kwargs['ignore_case']):
            # Игнорирование регистра - приведение всех строк из списка к нижнему
            регистру
            self.items = [str(i).lower() for i in items]
        else:
            self.items = items
        self.index = 0
        self.used = []

    def __next__(self):
        # Нужно реализовать __next__
        # Проходим по списку использованных элементов - если текущего элемента в нём
        нет, то добавляем его и выводим
        while self.items[self.index] in self.used:
            if self.index == len(self.items) - 1:
                raise StopIteration
            self.index += 1

        self.used.append(self.items[self.index])
        return self.items[self.index]

    def __iter__(self):
        return self
```



3. Задача 3.

Дан массив с положительными и отрицательными числами. Необходимо одной строкой вывести на экран массив,

отсортированный по модулю. Сортировку осуществлять с помощью функции sorted

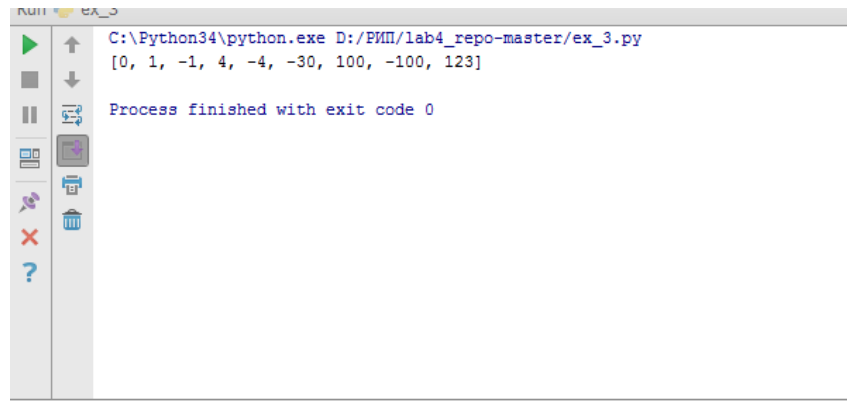
Пример:

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

Вывод: [0, 1, -1, 4, -4, -30, 100, -100, 123]

Ex_3.py

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3
print(sorted(data, key=lambda x: abs(x)))
```



4. Задача 4.

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.

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()
```

decorators.py

```
def print_result(func_arg):  
    def decorated_func(*args):  
        print(func_arg.__name__)  
        # Если возвращает список - печатать в столбик  
        if type(func_arg(*args)) == list:  
            for i in func_arg(*args):  
                print(i)  
        # Если словарь - печатать парами ключ-значение  
        elif type(func_arg(*args)) == dict:  
            for key, val in func_arg(*args).items():  
                print('{} = {}'.format(key, val))  
        # Во всех прочих случаях - выводить результат как есть
```

```

else:
    print(func_arg(*args))
return decorated_func

```

```

Run ex_4
C:\Python34\python.exe D:/РМП/lab4_repo-master/ex_4.py
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2
Process finished with exit code 0

```

5. Задача 5.

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

Ctxmgrs.py

```
import time
```

```

class timer:
    def __enter__(self):
        self.t = time.time()

    def __exit__(self, exp_type, exp_value, traceback):
        print(time.time() - self.t)

```

ex_5.py

```

from time import sleep
from librip.ctxmgrs import timer

with timer():
    sleep(5.5)

```

```

Run ex_5
C:\Python34\python.exe D:/РМП/lab4_repo-master/ex_5.py
5.511810064315796
Process finished with exit code 0

```

6. Задача 6.

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 os.path
import json
import sys
from librip.ctxmgrs import timer
from librip.decorators import print_result
from librip.gens import field, gen_random
from librip.iterators import Unique as unique

path = os.path.abspath(sys.argv[1])

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

with open(path, encoding="utf8") as f:
    data = json.load(f)

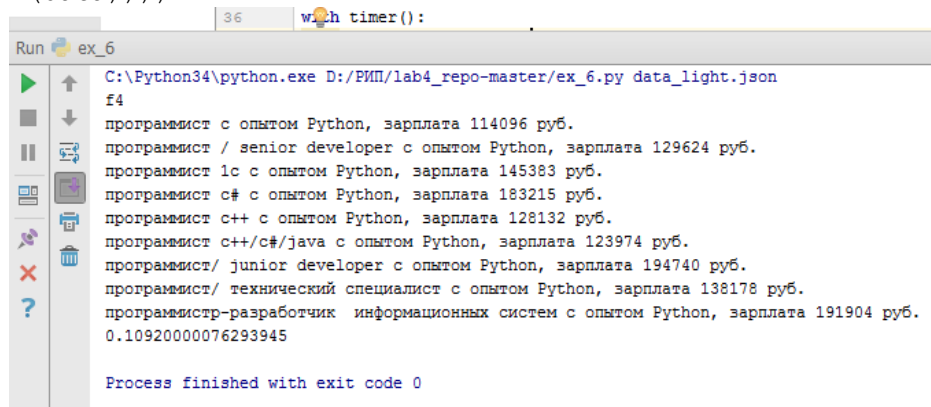
# Функция для вывода отсортированного списка профессий без повторений
def f1(arg):
    return(sorted([i for i in unique([j['job-name'] for j in arg], ignore_case =
True)]))

# Функция для отбора профессий со словом "программист" в начале
def f2(arg):
    return(filter(lambda x: (x.lower().find('программист') == 0), arg))

# Функция модификации профессии
def f3(arg):
    return(["{} {}".format(x, "с опытом Python") for x in arg])

# Функция генерации размера зарплаты для профессий
@print_result
def f4(arg):
    return(["{}", {} {} {}".format(x, "зарплата", y, "руб.") for x, y in zip(arg,
list(gen_random(100000, 200000, len(arg))))])

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



```
Run ex_6
C:\Python34\python.exe D:/РИП/lab4_repo-master/ex_6.py data_light.json
f4
программист с опытом Python, зарплата 114096 руб.
программист / senior developer с опытом Python, зарплата 129624 руб.
программист 1с с опытом Python, зарплата 145383 руб.
программист c# с опытом Python, зарплата 183215 руб.
программист c++ с опытом Python, зарплата 128132 руб.
программист c++/c#/java с опытом Python, зарплата 123974 руб.
программист/ junior developer с опытом Python, зарплата 194740 руб.
программист/ технический специалист с опытом Python, зарплата 138178 руб.
программист-разработчик информационных систем с опытом Python, зарплата 191904 руб.
0.10920000076293945

Process finished with exit code 0
```