Московский государственный технический университет им. Н.Э.Баумана

Защищено:
Гапанюк Ю. Е.

"__"_____201**6** г.

Демонстрация ЛР:
Гапанюк Ю. Е.

"__"_____201**6** г.

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

11
(количество листов)

ИСПОЛНИТЕЛЬ:

студентка группы **ИУ5-51** _____

(подпись)

Мишенёва Е.Ю.

"__"_____201**6** г.

Москва, МГТУ - 201**6**

СОДЕРЖАНИЕ

1. Задание лабораторной работы

Задачи выполняются 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
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`

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 *о дной 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). П ример: *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.__

2. Исходный код

ctxmngrs.py

```
import time

class timer:
    def __enter__(self):
        self.start = time.clock()
    def __exit__(self, exp_type, exp_value, traceback):
        print(time.clock() - self.start)
```

decorators.py

```
def print_result(printable_func):

    def decorated(*args):
        print(printable_func.__name__)
        if type(printable_func(*args)) == list:
            for i in printable_func(*args):
                print(i)
        elif type(printable_func(*args)) == dict:
            for key, val in printable_func(*args).items():
                print('{} = {}'.format(key, val))
        else:
            print(printable_func(*args))

    return decorated
```

gen.py

```
def field(items, *args):
    assert len(args) > 0, 'No args'
    if len(args) == 1:
        for elem in items:
            yield elem[args[0]]
    else:
        dict = {}
        for elem in items:
            for arg in args:
                dict[arg] = elem[arg]
            yield dict

def gen_random(begin, end, num_count):
    pass
    for i in range(num_count):
     yield randint(begin, end)
```

iterators.py

```
class Unique(object):
    def __init__(self, items, **kwargs):
        if ('ignore_case' in kwargs.keys()) and (kwargs['ignore_case']):
```

```
            self.data = [str(i).lower() for i in items]
        else:
            self.data = items

        self.index = 0
        self.mas = []

    def __next__(self):
        while self.data[self.index] in self.mas:
            self.index += 1
            if self.index == len(self.data):
                raise StopIteration

        self.mas.append(self.data[self.index])

        return self.data[self.index]

    def __iter__(self):
        return self
```

ex_1.py

```
#!/usr/bin/env python3
from librip.gen import field
from librip.gen 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('Генератор 1: ')

for i in field(goods, 'color', 'title'):
    print(i, end = ', ')

print()
print('Генератор 2: ')

for j in gen_random(1, 6, 7):
    print(j, end = '', '')
```

ex_2.py

```
#!/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)
data3 = ['a', 'b', 'A', 'B', 'c', 'a', 'B']

# Реализация задания 2

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

print()

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

print()

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

print()

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

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)))
```

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(5.5)
```

ex_6.py

```
#!/usr/bin/env python3
import os.path
import json
import sys
from librip.ctxmngrs import timer
from librip.decorators import print_result
from librip.gen import field, gen_random
```

```
from librip.iterators import Unique as unique


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

with open(path) as f:
    data = json.load(f)

def f1(arg):
    return(sorted([i for i in unique([j['job-name'] for j in arg], ignore_case = True)]))


def f2(arg):
     return([x for x in arg if 'программист' in x])


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


@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))))
```

3. Результат

Ex_1.py

```
/usr/bin/python3.5 /home/sergey/Документы/V_ubuntu/Python_course/Labs/Lab_4_n/lab_4/ex_1.py
Генератор 1:
{'title': 'Ковер', 'color': 'green'}, {'title': 'Диван для отдыха', 'color': 'black'}, {'title': 'Стелаж', 'color': 'white'}, {'title': 'Вешалка для одежды', 'color': 'white'},
Генератор 2:
2, 2, 6, 3, 1, 2, 6,
Process finished with exit code 0
```

Ex_2.py

```
/usr/bin/python3.5 /home/sergey/Документы/V_ubuntu/Python_course/Labs/Lab_4_n/lab_4/ex_2.py
1, 2,
1, 2, 3,
a, b, A, B, c,
a, b, c,
Process finished with exit code 0
```

Ex_3.py

```
/usr/bin/python3.5 /home/sergey/Документы/V_ubuntu/Python_course/Labs/Lab_4_n/lab_4/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]

Process finished with exit code 0
```

Ex_4.py

```
/usr/bin/python3.5 /home/sergey/Документы/V_ubuntu/Python_course/Labs/Lab_4_n/lab_4/ex_4.py
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2
```

Ex_5.py

```
/usr/bin/python3.5 /home/sergey/Документы/V_ubuntu/Python_course/Labs/Lab_4_n/lab_4/ex_5.py
6.300000000000056e-05

Process finished with exit code 0
```

Ex_6.py

```
sergey@sergey-Lenovo-G780:~/Документы/V_ubuntu/Python_course/Labs/Lab_4_n/lab_4$ python3 ex_6.py data_light.json

f4
1с программист с опытом Python, зарплата 180281 руб.
web-программист с опытом Python, зарплата 121872 руб.
веб - программист (php, js) / web разработчик с опытом Python, зарплата 100847 руб.
веб-программист с опытом Python, зарплата 157515 руб.
ведущий инженер-программист с опытом Python, зарплата 129510 руб.
ведущий программист с опытом Python, зарплата 182287 руб.
инженер - программист с опытом Python, зарплата 110364 руб.
инженер - программист асу тп с опытом Python, зарплата 124673 руб.
инженер-программист с опытом Python, зарплата 192706 руб.
инженер-программист (клинский филиал) с опытом Python, зарплата 178004 руб.
инженер-программист (орехово-зуевский филиал) с опытом Python, зарплата 196653 руб.
инженер-программист 1 категории с опытом Python, зарплата 128546 руб.
инженер-программист ккт с опытом Python, зарплата 184273 руб.
инженер-программист плис с опытом Python, зарплата 188231 руб.
инженер-программист сапоу (java) с опытом Python, зарплата 143091 руб.
инженер-электронщик (программист асу тп) с опытом Python, зарплата 135381 руб.
педагог программист с опытом Python, зарплата 191746 руб.
помощник веб-программиста с опытом Python, зарплата 198299 руб.
программист с опытом Python, зарплата 177799 руб.
программист / senior developer с опытом Python, зарплата 143044 руб.
программист 1с с опытом Python, зарплата 101164 руб.
программист c# с опытом Python, зарплата 168143 руб.
программист c++ с опытом Python, зарплата 142613 руб.
программист c++/c#/java с опытом Python, зарплата 148549 руб.
программист/ junior developer с опытом Python, зарплата 120019 руб.
программист/ технический специалист с опытом Python, зарплата 107690 руб.
программистр-разработчик  информационных систем с опытом Python, зарплата 161524 руб.
системный программист (c, linux) с опытом Python, зарплата 107789 руб.
старший программист с опытом Python, зарплата 173068 руб.
0.19406199999999996
```