Защищено:
Гапанюк Ю.Е.

"__"_____2016 г.

Демонстрация:
Гапанюк Ю.Е.

"__"_____2016 г.

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

8
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-51 _____
(подпись)

Тодосиев Н.Д.

"__"_____2016 г.

Москва, МГТУ - 2016

СОДЕРЖАНИЕ

1. Описание задания лабораторной работы

Задача 1 (ex_1.py)

Необходимо реализовать генераторы field и gen_random
Генератор field последовательно выдает значения ключей словарей массива
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
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.
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

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
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
После завершения блока должно вывестись в консоль примерно 5.5

Задача 6 (ex_6.py)

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. librip/decorators.py

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

3. librip/gen.py

```
from random import randint

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

```
            if out:
                yield out

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

4. librip/iterators.py

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):

        if ('ignore_case' in kwargs.keys()) and (kwargs['ignore_case']):
            self.items = [str(i).lower() for i in items]
        else:
            self.items = items
        self.index = 0
        self.used = []

    def __next__(self):
        # Нужно реализовать   next
        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
```

5. librip/ctxmngrs.py

```
import time

class timer:
    def __enter__(self):
        self.start = time.clock()
    def __exit__(self, exc_type, exc_val, exc_tb):
        print(time.clock() - self.start)
```

6. ex_1.py

```
from librip.gen 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'}
]
for i in gen_random(2,5, 8):
    print(i, end = " ")
print (" ")
# Реализация задания 1
for i in field(goods, 'title'):
    print(i, end = " ")
```

Результат программы:

```
5 3 2 4 3 5 2 4
```

Ковер Диван для отдыха Стелаж Вешалка для одежды

7. ex_2.py

```
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', 'A', 'b', 'B', 'c', 'C', 'D', 'd']
# 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=' ')
```

Результат программы:

```
1 2
2 3 1
a A b B c C D d
a b c d
```

8. ex_3.py

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4, 1000, -2000]
# Реализация задания 3

print(sorted(data, key=lambda x: abs(x)), end=" ")
```

Результат программы:

```
[0, 1, -1, 4, -4, -30, 100, -100, 123, 1000, -2000]
```

9. 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()
```

Результат программы:

```
test_1
1
test_2
iu
test_3
b = 2
a = 1
test_4
1
2
```

10. ex_5.py

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

with timer():
    sleep(3.5)
```

Результат программы:

```
3.499463232338907
```

11. ex_6.py

```
import json
import os.path
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

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__))) +
"\\lab_4\\data_light.json"

with open(BASE_DIR, encoding="utf8") 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))))
```

Результат программы:

```
f4
1с программист с опытом Python, зарплата 181844 руб.
web-программист с опытом Python, зарплата 139116 руб.
веб - программист (php, js) / web разработчик с опытом Python, зарплата 141025 руб.
веб-программист с опытом Python, зарплата 171280 руб.
ведущий инженер-программист с опытом Python, зарплата 163167 руб.
ведущий программист с опытом Python, зарплата 125435 руб.
инженер - программист с опытом Python, зарплата 188548 руб.
инженер - программист асу тп с опытом Python, зарплата 183778 руб.
инженер-программист с опытом Python, зарплата 142803 руб.
инженер-программист (клинский филиал) с опытом Python, зарплата 153455 руб.
инженер-программист (орехово-зуевский филиал) с опытом Python, зарплата 120621 руб.
инженер-программист 1 категории с опытом Python, зарплата 101795 руб.
инженер-программист ккт с опытом Python, зарплата 180270 руб.
инженер-программист плис с опытом Python, зарплата 148091 руб.
инженер-программист сапоу (java) с опытом Python, зарплата 108379 руб.
инженер-электронщик (программист асу тп) с опытом Python, зарплата 135892 руб.
педагог программист с опытом Python, зарплата 176429 руб.
помощник веб-программиста с опытом Python, зарплата 172318 руб.
программист с опытом Python, зарплата 146163 руб.
программист / senior developer с опытом Python, зарплата 111650 руб.
программист 1с с опытом Python, зарплата 188231 руб.
программист c# с опытом Python, зарплата 185573 руб.
программист c++ с опытом Python, зарплата 189782 руб.
программист c++/c#/java с опытом Python, зарплата 159024 руб.
программист/ junior developer с опытом Python, зарплата 140568 руб.
программист/ технический специалист с опытом Python, зарплата 132210 руб.
программистр-разработчик  информационных систем с опытом Python, зарплата 154708
руб.
системный программист (с, linux) с опытом Python, зарплата 193942 руб.
старший программист с опытом Python, зарплата 163510 руб.
0.15574513819868574
```