



Лекции. Практические занятия

Солдатов Е.Ю.

2022 г.

СПИСКИ

Это аналог массивов в Python.

Создадим список:

```
list = [0, 1, 2, 3, 4]
print(list[3])
D:\python>python code.py
3
```

В качестве элемента список может содержать список:

```
list = ['a', 'b', ['c', 'd', 'e']]
print(list[1] + list[0] + list[2][1])
D:\python>python code.py
bad
```

В список можно добавлять элементы, можно его умножить на числа:

```
print(list + ['f', 'g'])
print(list * 2)
['a', 'b', ['c', 'd', 'e'], 'f', 'g']
['a', 'b', ['c', 'd', 'e'], 'a', 'b', ['c', 'd', 'e']]
```

Кстати **строка** тоже является списком:

```
word = 'Коллайдер'
print(word[5]+word[1]+word[6])
D:\python>python code.py
йод
```

СПИСКИ

Поиск по списку (строке) осуществляется с помощью оператора **in**:

```
list = [ 'Ivanov', 'Petrov', 'Sidorov', 'Kuznetsov' ]
```

- *if 'Petrov' in list:*

```
    print('Petrov is in the list!')
```

```
D:\python>python code.py  
Petrov is in list!
```

```
list = [ 'Ivanov', 'Petrov', 'Sidorov', 'Kuznetsov' ]
```

if 'Sokolov' not in list:

```
    print('Sokolov isn\'t in the list!')
```

```
D:\python>python code.py  
Sokolov isn't in list!
```

Добавление элементов в список (в конец списка):

```
list = []
```

```
list.append('Слово')
```

```
list.append(123)
```

```
list.append([3, 2, 1])
```

```
print(list)
```

```
D:\python>python code.py  
['Слово', 123, [3, 2, 1]]
```

Интересная особенность python: любая переменная или строка являются объектами. В данном случае список *list* – объект, а *append()* – метод.

СПИСКИ

Длину списка можно получить с помощью функции `len()`

```
list = [0, 1, 2, 3, 4]
```

```
print("В списке " + str(len(list)) + " элементов.")
```

```
D:\python>python code.py
В списке 5 элементов.
```

Удаление элементов возможно методом `remove()`

```
list = [0, 1, 2, 3, 4, 5]
```

```
list.remove(1)
```

```
print("В списке " + str(len(list)) + " элементов. Сам список: " + str(list))
```

```
В списке 5 элементов. Сам список: [0, 2, 3, 4, 5]
```

Если нужно удалить элемент именно по номеру, то `del list[index]`

Добавлять элементы можно и методом `insert(позиция, элемент)`. Он позволяет элемент добавить на любую (заданную) позицию:

```
list = [0, 1, 2, 3, 4]
```

```
list.insert(2, 7)
```

```
print("Список: " + str(list))
```

```
D:\python>python code.py
Список: [0, 1, 7, 2, 3, 4]
```

Ещё функции для списков:

`max(list)`, `min(list)`

И методы:

`list.count(7)` – программно посчитать сколько раз есть объект “7” в списке

`list.reverse()` – переворачивает массив (**не возвращает!**)

ЗАДАЧИ

Есть следующий список:

```
list1=[1, 2, 3, 5, 4, 4, 35, 44, 79, 91]
```

Выведите все чётные элементы

Есть 2 списка:

```
list2=[1, 2, 3, 5, 4, 4, 35, 44, 79, 91]
```

```
list3=[1, 3, 5, 5, 13, 27, 44, 59, 61, 88]
```

Создайте и выведите список, который состоит из элементов, общих для этих двух списков.

ЦИКЛЫ

Цикл с условием **while**:

```
i = 0
while i <= 3:
    print (i)
    i += 1
```

```
D:\python>python code.py
0
1
2
3
```

Команда **break** может принудительно остановить цикл на текущей операции. Остановим бесконечный цикл:

```
i = 1
while 3 == 3:
    print (i)
    i += 1
    if i > 100:
        break
```

Команда **continue** прерывает текущую итерацию цикла и переводит программу к следующей. Синтаксис такой же.

Слово **else**, примененное в цикле for или while, проверяет, был ли произведен выход из цикла инструкцией break, или же "естественным" образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

ДИАПАЗОНЫ И ЦИКЛЫ

Список можно заполнять автоматически с помощью диапазонов `range()`

```
numbers=list(range(100))
```

```
print(numbers)
```

Нужно преобразование типа, т.к. функция `range()` сама по себе выдаёт т.н. «итерируемый объект диапазона».

```
numbers=list(range(100))  
print(numbers)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,  
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,  
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,  
98, 99]
```

Диапазон может иметь несколько аргументов, если их 2, то 1-ый – начало, второй – конец. Также можно задавать шаг 3-им аргументом.

```
numbers=list(range(50,101,2))
```

```
print(numbers)
```

range работает только с int числами!

```
D:\python>python code.py
```

```
[50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88,  
90, 92, 94, 96, 98, 100]
```

Чтобы обойти список с помощью цикла:

Можно с помощью цикла `while` и итерационной переменной:

```
i=0
```

```
length=len(numbers)-1
```

И далее итерировать `while` по условию: `i<=length`

ЦИКЛЫ

Либо можно использовать цикл **for**:

```
array=[0, 1, 2, 3, 4, 5]
for element in array:
    print(element)
```



```
D:\python>python code.py
0
1
2
3
4
5
```

Чтобы исполнить код определённое число раз делаем так:

```
for test in range(5):
    print (test)
```



Итерируемый
объект диапазона



```
D:\python>python code.py
0
1
2
3
4
```

Можно итерировать по списку:

```
array3=list(range(50,55,2))
for element in array3:
    print(str(element)+"!")
```



```
D:\python>python code.py
50!
52!
54!
```

Кстати, так как строка тоже список, то можно итерировать и по строке.

Это делается так:

```
for element in "Hello":
    print(element+"!")
```



```
D:\python>python code.py
H!
e!
l!
l!
o!
```


ЗАДАЧИ

x имеет значения от 0 до 100 через каждые 5

$$y=x^2-3x+2$$

Заполнить массивы значений для графика.

Если выписать все натуральные числа меньше 10, кратные 3 или 5, то получим 3, 5, 6 и 9. Сумма этих чисел равна 23.

Найдите сумму всех чисел меньше 1000, кратных 3 или 5.

*) Каждый следующий элемент ряда Фибоначчи получается при сложении двух предыдущих. Начиная с 1 и 2, первые 10 элементов будут:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Найдите сумму всех четных элементов ряда Фибоначчи, которые не превышают четыре миллиона.

ФУНКЦИИ

Функция – определённый блок кода.

Создание и вызов своих функций:

```
def func_print():  
    for el in range(3):  
        print('polundra')
```

В основной программе:

```
func_print()
```



```
D:\python>python code.py  
polundra  
polundra  
polundra
```

Функцию нельзя сначала вызывать и только потом определять...

Кстати, `print()`, `int()`, `str()` – тоже функции, просто не пользовательские.

Функция с параметрами:

```
def maximum(num1, num2):
```

 ← тут параметры

```
    if num1 > num2:
```

```
        return num1
```

```
    else:
```

```
        return num2
```

```
print(maximum(2,3))
```

 ← а тут аргументы (значения)

Переменные внутри функции – локальны и уничтожаются при выходе из неё.

```
D:\python>python code.py  
3
```

В данном случае функция возвращает значение.

Функция также является переменной

```
new_max = maximum
```

```
print(new_max(4, 6))
```



```
D:\python>python code.py  
6
```

И ещё функции можно передавать в качестве аргументов в др. функции.

Хорошей практикой является создание основной функции `main()`

КОММЕНТАРИИ, ДОКУМЕНТАЦИЯ

Закомментировать одну строку можно с помощью #

Много строк закомментировать можно с помощью:

```
““
```

```
Текст
```

```
““
```

Строки документирования функций и их вывод:

```
def func_print():  
    "Definition"  
    for el in range(3):  
        print('spam')
```

```
print(func_print.__doc__)
```

```
D:\python>python code.py  
Definition
```

Делается потому что исходный код может быть не всегда доступен, чтобы прочитать комментарии, а тут текст документирования можно вывести программно.

Используйте комментарии! Документируйте функции!

(Вы сами скажете себе потом спасибо).

**Real programmers
don't comment
their codes.**

**If it was hard to write... it must
be hard to read!**

МОДУЛИ

Модуль – python файл, содержащий различные сторонние функции. Аналог библиотек, внешних классов в других языках программирования.

Чтобы подключить используем инструкцию `import`:

```
import random  
print( random.random() )
```

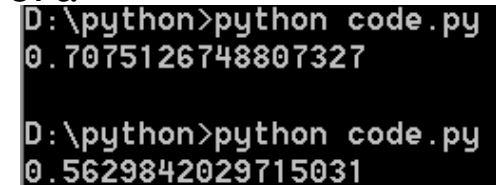
Будет выводить случайные числа от 0 до 1.

В этом случае мы получали объект `random` для использования в коде.

А если хочется только получить функцию из модуля:

```
from random import random  
print( random() )
```

Выдаст то же самое.



```
D:\python>python code.py  
0.7075126748807327  
  
D:\python>python code.py  
0.5629842029715031
```

Если хочется импортировать все функции из модуля делаем так:

```
from math import *
```

Или несколько:

```
from math import sqrt, pi
```

А можно при импорте и переименовать функцию для собственных целей:

```
from math import sqrt as my_sqrt
```

Это нужно, чтобы избегать конфликтов с одинаковыми названиями функций.

УСТАНОВКА СТОРОННИХ МОДУЛЕЙ

Часть модулей предустановлена: *math, random, os, sys*

Они называются STL (стандартная библиотека)

- Например, модуль *datetime*:

```
from datetime import datetime
```

```
now = datetime.now()
```

```
print("Сегодняшняя дата: " + str(now.date()) + ", время:" + str(now.time()))
```

```
D:\python>python code.py  
Сегодняшняя дата: 2019-04-15, время:19:54:27.263780
```

Вторая часть модулей – те, что написаны Вами

И третья часть – загруженные из Интернет.

Загрузить можно из репозитория **PyPi**. Там люди публикуют свои модули.

Установка пакета с помощью команды `pip` в командной строке:

```
> pip install module_name
```

И потом в программе:

```
import module_name
```

...

Большие модули в Windows могут не установиться, то можно найти т.н. **Prebuild binaries**, скачать, установить их в системе и после этого модуль станет вам доступен в **python**.

СТОРОННИЙ МОДУЛЬ: ПРИМЕР

Попробуем взаимодействовать со сторонним модулем. Например, который сообщает погоду в любой точке мира `pyowm`

<https://pypi.org/project/pyowm/>

Установим:

```
> pip install pyowm
```

Подробная информация об использовании:

<https://github.com/csparpa/pyowm/blob/master/README.md>

Напишем программу:

```
import pyowm
owm = pyowm.OWM('6d00d1d4e704068d70191bad2673e0cc')
mgr = owm.weather_manager()
```

API-ключ, доступный при
регистрации на сайте сервиса



```
city=input("Введите город: ")
observation = mgr.weather_at_place(city)
w = observation.weather
```

```
print("Температура: " + str(w.temperature('celsius')['temp']))
print("Также в этом городе "+w.detailed_status)
```

```
Введите город: Moscow
Температура: 10.57
Также в этом городе few clouds
```

ЗАДАЧА

Сделайте свой модуль, включающий функции:

А) пересчёта футов в метры;

Б) пересчёта градусов Цельсия в градусы Фаренгейта.

В главной программе организуйте выбор нужного конвертера и ввода исходных данных.