**Python’s range() Function Explained**

**What is Python's range() Function?**

## Definition and Usage

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

**range() function syntax and arguments**

range(start, stop[, step])

It takes three arguments. Out of the three 2 arguments are optional. I.e., start and step are the optional arguments.

1. A **start** argument is a starting number of the sequence. i.e., lower limit. By default, it starts with 0 if not specified.
2. A **stop** argument is an upper limit. i.e., generate numbers up to this number, The range() doesn’t include this number in the result.
3. The **step** is a difference between each number in the result. The default value of the step is 1 if not specified.

**Python's range() Parameters**

The range() function has two sets of parameters, as follows:

range(stop)

* stop: Number of integers (whole numbers) to generate, starting from zero. eg. range(3) == [0, 1, 2].

range([start], stop[, step])

* start: Starting number of the sequence.
* stop: Generate numbers up to, but not including this number.
* step: Difference between each number in the sequence.

Note that:

* All parameters must be integers.
* All parameters can be positive or negative.
* range() (and Python in general) is 0-index based, meaning list indexes start at 0, not 1. eg. The syntax to access the first element of a list is mylist[0]. Therefore the last integer generated by range() is up to, but not including, stop. For example range(0, 5) generates integers from 0 up to, but not including, 5.

**Python's range() Function Examples**

**Simple Usage**

|  |  |
| --- | --- |
| 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 | >>> # One parameter  >>> for i in range(5):  ...     print(i)  ...  0  1  2  3  4  >>> # Two parameters  >>> for i in range(3, 6):  ...     print(i)  ...  3  4  5  >>> # Three parameters  >>> for i in range(4, 10, 2):  ...     print(i)  ...  4  6  8  >>> # Going backwards  >>> for i in range(0, -10, -2):  ...     print(i)  ...  0  -2  -4  -6  -8 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | >>> my\_list = ['one', 'two', 'three', 'four', 'five']  >>> my\_list\_len = len(my\_list)  >>> for i in range(0, my\_list\_len):  ...     print(my\_list[i])  ...  one  two  three  four  five |

**Iterating Lists**

**99 Bottles of Beer on the Wall...**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | for i in range(99, 0, -1):     if i == 1:       print('1 bottle of beer on the wall, 1 bottle of beer!')       print('So take it down, pass it around, no more bottles of beer on the wall!')         elif i == 2:            print('2 more bottles of beer on the wall, 2 more bottles of beer!')            print('So take one down, pass it around, 1 more bottle of beer on the wall!')          else:            print('{0} bottles of beer on the wall, {0} bottles of beer!'.format(i))            print('So take it down, pass it around, {0} more bottles of beer on the wall!'.format(i - 1)) |

With the following code:

We get the following output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 99 bottles of beer on the wall, 99 bottles of beer!  So take one down, pass it around, 98 more bottles of beer on the wall!  98 bottles of beer on the wall, 98 bottles of beer!  So take one down, pass it around, 97 more bottles of beer on the wall!  97 bottles of beer on the wall, 97 bottles of beer!  So take one down, pass it around, 96 more bottles of beer on the wall!  ...  3 bottles of beer on the wall, 3 bottles of beer!  So take one down, pass it around, 2 more bottles of beer on the wall!  2 more bottles of beer on the wall, 2 more bottles of beer!  So take one down, pass it around, 1 more bottle of beer on the wall!  1 bottle of beer on the wall, 1 bottle of beer!  So take it down, pass it around, no more bottles of beer on the wall! |

**Python's range() vs xrange() Functions**

You may have heard of a function known as xrange(). This is a function that is present in Python 2.x, however it was renamed to range() in Python 3.x, and the original range() function was deprecated in Python 3.x. So what's the difference? Well, in Python 2.x range() produced a list, and xrange() returned an iterator - a sequence object. We can see this in the following example:

|  |  |
| --- | --- |
| 1  2  3  4 | >>> range(1)  range(0, 1)  >>> type(range(1))  <class 'range'> |

So in Python 3.x, the range() function got its own type. In basic terms, if you want to use range() in a for loop, then you're good to go. However you can't use it purely as a list object. For example you cannot slice a range type.

When you're using an iterator, every loop of the for statement produces the next number on the fly. Whereas the original range() function produced all numbers instantaneously, before the for loop started executing. The problem with the original range() function was that it used a very large amount of memory when producing a lot of numbers. However it tends to be quicker with a small amount of numbers. Note that in Python 3.x, you can still produce a list by passing the generator returned to the list() function. As follows:

|  |  |
| --- | --- |
| 1  2  3 | >>> list\_of\_ints = list(range(3))  >>> list\_of\_ints  [0, 1, 2] |

**Using floats with Python's range() function**

Unfortunately the range() function doesn't support the float type. However don't get upset too soon! We can easily implement it with a function. There's several ways it can be done, however here's one.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | >>> # Note: All arguments are required.  >>> # We're not fancy enough to implement all.  >>> def frange(start, stop, step):  ...     i = start  ...     while i < stop:  ...         yield i  ...         i += step  ...  >>> for i in frange(0.5, 1.0, 0.1):  ...         print(i)  ...  0.5  0.6  0.7  0.8  0.9  1.0 |