IIT Madras
BSc Degree

Home Lesson-3.3

Lesson-3.2

Loops

`for` loop

Let us look at a simple problem of printing numbers. We would like to print the first 5 non-negative integers. We have a different kind of a loop now, the `for` loop:

```
1 for i in range(5):
2     print(i)
3 # A dummy line
```

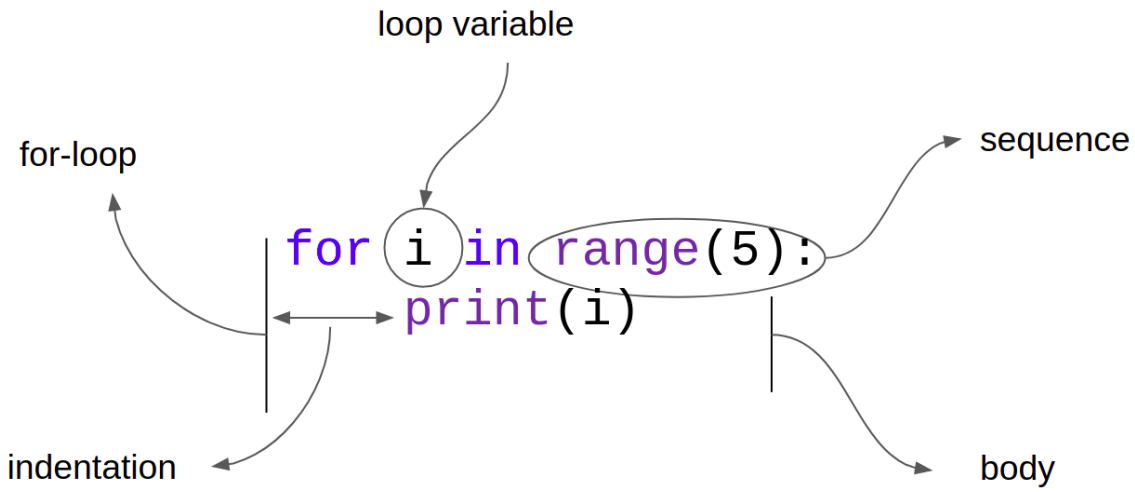
The output is:

```
1 0
2 1
3 2
4 3
5 4
```

`for` and `in` are keywords in Python. `range` is an object that represents a sequence of numbers. Line-2 is the body of the loop. An intuitive understanding of the code given above is as follows:

- In each iteration of the loop, an element *in* the sequence is picked up and is printed to the console.
- Assuming that the sequence is ordered from left to right, the leftmost element is the first to be picked up.
- The sequence is processed from left to right.
- Once the rightmost element has been printed to the console, control returns to line-1 for one last time. Since there are no more elements to be read in the sequence, the control exits the loop and moves to line-3.

A visual representation is given below:



Similar to `while` loops and `if-else` blocks, the body of a `for` loop should be indented.

range()

`range(5)` represents the following sequence: `0, 1, 2, 3, 4`. In general, `range(n)` represents the sequence: `0, 1, ..., n - 1`. `range` is quite versatile. The following code prints all two digit numbers greater than zero:

```
for i in range(10, 100):
    print(i)
```

`range(10, 100)` represents the sequence `10, 11, ..., 99`. In general, `range(start, stop)` represents the sequence `start, start + 1, ..., stop - 1`. Let us add another level of complexity. The following code prints all even two digit numbers greater than 0:

```
for i in range(10, 100, 2):
    print(i)
```

`range(10, 100, 2)` represents the sequence `10, 12, ..., 98`. In general, `range(start, stop, step)` represents the sequence `start, start + step, start + 2 * step, ..., last`, where `last` is the largest element in this sequence that is less than `stop`. This is true when the `step` parameter is positive.

The following are equivalent:

- `range(n)`
- `range(0, n)`
- `range(0, n, 1)`

So far we have seen only increasing sequences. With the help of a negative step size, we can also come up with decreasing sequences. The following code prints all two-digit even numbers greater than zero in descending order:

```
for i in range(98, 9, -2):
    print(i)
```

For a negative `step` value, `range(start, stop, step)` represents the sequence `start, start + step, start + 2 * step, ..., last`, where `last` is the smallest element in the sequence greater than `stop`.

Now, consider the following code:

```
for i in range(5, 5):
    print(i)
```

`range(5, 5)` is an empty sequence. So, the above code will not print anything. Another instance of an empty sequence:

```
for i in range(10, 5):
    print(i)
```

The point to note is that neither of these code snippets produces any error. Finally, try executing the following snippet and observe the output.

```
##### Alarm! Wrong code snippet! #####
for i in range(0.0, 10.0):
    print(i)
##### Alarm! Wrong code snippet! #####
```

Iterating through Strings

Since a string is a sequence of characters, we can use the `for` loop to iterate through strings. The following code will print each character of the string `x` in one line:

```
word = 'good'
for char in word:
    print(char)
```

The output is:

```
g
o
o
d
```

We can add some more code to enrich the output:

```
word = 'good'
count = 1
for char in word:
    print(char, 'occurs at position', count, 'in the string', word)
    count = count + 1
```

The output is:

```
g occurs at position 1 in the string good
o occurs at position 2 in the string good
o occurs at position 3 in the string good
d occurs at position 4 in the string good
```