Python for Loop

In Python, we use a `for` loop to iterate over sequences such as lists, strings, dictionaries, etc.

```
languages = ['Swift', 'Python', 'Go']

# access elements of the list one by one
for lang in languages:
    print(lang)
```

Output

```
Swift
Python
Go
```

In the above example, we have created a list named *languages*. As the list has 3 elements, the loop iterates **3** times.

The value of `lang` is

- `Swift` in the first iteration.
- `Python` in the second iteration.
- `Go` in the third iteration.

---

for loop Syntax

```
for val in sequence:
    # body of the loop
```

The `for` loop iterates over the elements of *sequence* in order. In each iteration, the body of the loop is executed.

The loop ends after the last item in the sequence is reached.

Indentation in Loop

In Python, we use indentation to define a block of code, such as the body of a loop. For example,

```
languages = ['Swift', 'Python', 'Go']

# Start of loop
for lang in languages:
    print(lang)
    print('-----')
# End of for loop

print('Last statement')
```

Example: Loop Through a String

```
language = 'Python'

# iterate over each character in language
for x in language:
    print(x)
```

Output

```
P
y
t
h
o
n
```

Here, we have printed each character of the string *language* using a `for` loop.

---

for Loop with Python range()

In Python, the range() function returns a sequence of numbers. For example,

```
values = range(4)
```

Here, `range(4)` returns a sequence of **0**, **1**, **2** ,and **3**.

Since the `range()` function returns a sequence of numbers, we can iterate over it using a `for` loop. For example,

```
# iterate from i = 0 to i = 3
for i in range(4):
    print(i)
```

Output

```
0
1
2
3
```

Here, we used the `for` loop to iterate over a range from **0** to **3**.

This is how the above program works.

Iteration	**Value of `i`**	**`print(i)`**	**Last item in sequence?**
1st	`0`	Prints `0`	No
2nd	`1`	Prints `1`	No
3rd	`2`	Prints `2`	No
4th	`3`	Prints `3`	Yes The loop terminates.

Tip: We can end a `for` loop before iterating through all the items by using a break statement.

More on Python for Loop

Python for loop with `else` clause

A `for` loop can have an optional `else` clause. This `else` clause executes after the iteration completes.

```
digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

Output

```
0
1
5
No items left.
```

Here, the `for` loop prints all the items of the *digits* list. When the loop finishes, it executes the `else` block and prints `No items left.`

Note: The `else` block will not execute if the for loop is stopped by a break statement.

Using for loop without accessing items

We can also use `for` loop to repeat an action a certain number of times. For example,

```
languages = ['Swift', 'Python', 'Go']

# looping to repeat an action without using the list elements
for language in languages:
    print('Hi')
```

Output

```
Hi
Hi
Hi
```

Here, we used the list *languages* to run the loop three times. However, we didn't use any of the elements of the list.

In such cases, it is clearer to use the _ (underscore) as the loop variable. The _ indicates that a loop variable is a placeholder and its value is intentionally being ignored.

For example,

```
languages = ['Swift', 'Python', 'Go']

# using _ for placeholder variable
for _ in languages:
    print('Hi')
```

Here, the loop still runs three times because there are three elements in the `languages` list. Using _ indicates that the loop is there for repetition and not for accessing the elements.

Nested for loops

A `for` loop can also have another `for` loop inside it. For each cycle of the outer loop, the inner loop completes its entire sequence of iterations. For example,

```
# outer loop
for i in range(2):
    # inner loop
    for j in range(2):
        print(f"i = {i}, j = {j}")
```

Output

```
i = 0, j = 0
i = 0, j = 1
i = 1, j = 0
i = 1, j = 1
```

Also Read:

- Python while loop
- Python break and continue

Table of Contents

- Introduction
- for loop Syntax
- Example: Loop Through a String
- for Loop with Python range()