Python for loop

Python is a strong, universally applicable prearranging language planned to be easy to comprehend and carry out. It is allowed to get to because it is open-source. In this tutorial, we will learn how to use Python for loops, one of the most fundamental looping instructions in Python programming.

Introduction to for Loop in Python

Python frequently uses the Loop to iterate over iterable objects like lists, tuples, and strings. Crossing is the most common way of emphasizing across a series, for loops are used when a section of code needs to be repeated a certain number of times. The for-circle is typically utilized on an iterable item, for example, a rundown or the in-fabricated range capability. In Python, the for Statement runs the code block each time it traverses a series of elements. On the other hand, the "while" Loop is used when a condition needs to be verified after each repetition or when a piece of code needs to be repeated indefinitely. The for Statement is opposed to this Loop.

Syntax of for Loop

1. **for** value **in** sequence:
2. {loop body}

The value is the parameter that determines the element's value within the iterable sequence on each iteration. When a sequence contains expression statements, they are processed first. The first element in the sequence is then assigned to the iterating variable iterating\_variable. From that point onward, the planned block is run. Each element in the sequence is assigned to iterating\_variable during the statement block until the sequence as a whole is completed. Using indentation, the contents of the Loop are distinguished from the remainder of the program.

Example of Python for Loop

**Code**

1. # Code to find the sum of squares of each element of the list using for loop
3. # creating the list of numbers
4. numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
6. # initializing a variable that will store the sum
7. sum\_ = 0
9. # using for loop to iterate over the list
10. **for** num **in** numbers:
12. sum\_ = sum\_ + num \*\* 2
14. **print**("The sum of squares is: ", sum\_)

**Output:**

The sum of squares is: 774

The range() Function

Since the "range" capability shows up so habitually in for circles, we could erroneously accept the reach as a part of the punctuation of for circle. It's not: It is a built-in Python method that fulfills the requirement of providing a series for the for expression to run over by following a particular pattern (typically serial integers). Mainly, they can act straight on sequences, so counting is unnecessary. This is a typical novice construct if they originate from a language with distinct loop syntax:

**Code**

1. my\_list = [3, 5, 6, 8, 4]
2. **for** iter\_var **in** range( len( my\_list ) ):
3. my\_list.append(my\_list[iter\_var] + 2)
4. **print**( my\_list )

**Output:**

[3, 5, 6, 8, 4, 5, 7, 8, 10, 6]

AD

Iterating by Using Index of Sequence

Another method of iterating through every item is to use an index offset within the sequence. Here's a simple illustration:

**Code**

1. # Code to find the sum of squares of each element of the list using for loop
3. # creating the list of numbers
4. numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
6. # initializing a variable that will store the sum
7. sum\_ = 0
9. # using for loop to iterate over list
10. **for** num **in** range( len(numbers) ):
12. sum\_ = sum\_ + numbers[num] \*\* 2
14. **print**("The sum of squares is: ", sum\_)

**Output:**

The sum of squares is: 774

The len() worked in a technique that profits the complete number of things in the rundown or tuple, and the implicit capability range(), which returns the specific grouping to emphasize over, proved helpful here.

Using else Statement with for Loop

A loop expression and an else expression can be connected in Python.

After the circuit has finished iterating over the list, the else clause is combined with a for Loop.

The following example demonstrates how to extract students' marks from the record by combining a for expression with an otherwise statement.

**Code**

1. # code to print marks of a student from the record
2. student\_name\_1 = 'Itika'
3. student\_name\_2 = 'Parker'

6. # Creating a dictionary of records of the students
7. records = {'Itika': 90, 'Arshia': 92, 'Peter': 46}
8. **def** marks( student\_name ):
9. **for** a\_student **in** record: # for loop will iterate over the keys of the dictionary
10. **if** a\_student == student\_name:
11. **return** records[ a\_student ]
12. **break**
13. **else**:
14. **return** f'There is no student of name {student\_name} in the records'
16. # giving the function marks() name of two students
17. **print**( f"Marks of {student\_name\_1} are: ", marks( student\_name\_1 ) )
18. **print**( f"Marks of {student\_name\_2} are: ", marks( student\_name\_2 ) )

**Output:**

AD

Marks of Itika are: 90

Marks of Parker are: There is no student of name Parker in the records

Nested Loops

If we have a piece of content that we need to run various times and, afterward, one more piece of content inside that script that we need to run B several times, we utilize a "settled circle." While working with an iterable in the rundowns, Python broadly uses these.

**Code**

1. **import** random
2. numbers = [ ]
3. **for** val **in** range(0, 11):
4. numbers.append( random.randint( 0, 11 ) )
5. **for** num **in** range( 0, 11 ):
6. **for** i **in** numbers:
7. **if** num == i:
8. **print**( num, end = " " )

**Output:**

0 2 4 5 6 7 8 8 9 10

Next Topic[Python while loop](https://www.javatpoint.com/python-while-loop)