**ITERATION**

**Iteration** is the act of repeating a process, either to generate an unbounded sequence of outcomes, or with the aim of approaching a desired goal, target or result. Each repetition of the process is also called an "iteration", and the results of one iteration are used as the starting point for the next iteration.

**While**

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a **while** loop in Python programming language is –

**while** expression:

Statement(s)

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

count=0

**while** (count<9):

Print’The count is:’, count

count=count+1

Print “Good bye!”

More formally, here is the flow of execution for a while statement:

1. Evaluate the condition, yielding True or False.
2. If the condition is false, exit the while statement and continue execution at the next statement.
3. If the condition is true, execute the body and then go back to step 1.

This type of flow is called a **loop** because the third step loops back around to the top. Each time we execute the body of the loop, we call it an **iteration**. For the above loop, we would say, "It had five iterations" which means that the body of of the loop was executed five times.

**For Loop:**

The for loop in Python is used to iterate over a sequence ([list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple" \o "Python tuple), [string](https://www.programiz.com/python-programming/string)) or other iterable objects. Iterating over a sequence is called traversal.

## Syntax of for Loop

for val in sequence:

Body of for

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Example:

numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

sum = 0

**for** val in numbers:

sum = sum+val

print("The sum is", sum)

Output: The sum is 48

## The range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

To force this function to output all the items, we can use the function list().

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate though a sequence using indexing. Here is an example.

genre = ['pop', 'rock', 'jazz']

# iterate over the list using index

**for** i in range(len(genre)):

print("I like", genre[i]

## Break Statement

In Python, the break statement provides you with the opportunity to exit out of a loop when an external condition is triggered. You’ll put the break statement within the block of code under your loop statement, usually after a [conditional if statement](https://www.digitalocean.com/community/tutorials/how-to-write-conditional-statements-in-python-3-2).

Let’s look at an example that uses the break statement in a for loop:

number = 0

**for** number **in** range(10):

number = number + 1

**if** number == 5:

**break** # break here

print('Number is ' + str(number))

print('Out of loop')

In this small program, the variable number is initialized at 0. Then a for statement constructs the loop as long as the variable number is less than 10.

Within the for loop, the number increases incrementally by 1 with each pass because of the line number = number + 1.

Then, there is an if statement that presents the condition that if the variable number is equivalent to the integer 5, then the loop will break.

Within the loop is also a print() statement that will execute with each iteration of the for loop until the loop breaks, since it is after the break statement.

To see when we are out of the loop, we have included a final print() statement outside of the for loop.

When we run this code, our output will be the following:

Output

Number is 1

Number is 2

Number is 3

Number is 4

Out of loop

This shows that once the integer number is evaluated as equivalent to 5, the loop breaks, as the program is told to do so with the break statement.

The break statement causes a program to break out of a loop.

## Continue Statement

The continue statement gives you the option to skip over the part of a loop where an external condition is triggered, but to go on to complete the rest of the loop. That is, the current iteration of the loop will be disrupted, but the program will return to the top of the loop.

The continue statement will be within the block of code under the loop statement, usually after a conditional if statement.

Using the same for loop program as in the [Break Statement](https://www.digitalocean.com/community/tutorials/how-to-use-break-continue-and-pass-statements-when-working-with-loops-in-python-3#break-statement) section above, we’ll use a continue statement rather than a break statement:

number = 0

**for** number **in** range(10):

number = number + 1

**if** number == 5:

**continue** # continue here

print('Number is ' + str(number))

print('Out of loop')

The difference in using the continue statement rather than a break statement is that our code will continue despite the disruption when the variable number is evaluated as equivalent to 5. Let’s look at our output:

Output

Number is 1

Number is 2

Number is 3

Number is 4

Number is 6

Number is 7

Number is 8

Number is 9

Number is 10

Out of loop

Here we see that the line Number is 5 never occurs in the output, but the loop continues after that point to print lines for the numbers 6-10 before leaving the loop.

You can use the continue statement to avoid deeply nested conditional code, or to optimize a loop by eliminating frequently occurring cases that you would like to reject.

The continue statement causes a program to skip certain factors that come up within a loop, but then continue through the rest of the loop.

## Pass Statement

When an external condition is triggered, the pass statement allows you to handle the condition without the loop being impacted in any way; all of the code will continue to be read unless a break or other statement occurs.

As with the other statements, the pass statement will be within the block of code under the loop statement, typically after a conditional if statement.

Using the same code block as above, let’s replace the break or continue statement with a pass statement:

number = 0

**for** number **in** range(10):

number = number + 1

**if** number == 5:

**pass** # pass here

print('Number is ' + str(number))

print('Out of loop')

The pass statement occurring after the if conditional statement is telling the program to continue to run the loop and ignore the fact that the variable number evaluates as equivalent to 5 during one of its iterations.

We’ll run the program and take a look at the output:

Output

Number is 1

Number is 2

Number is 3

Number is 4

Number is 5

Number is 6

Number is 7

Number is 8

Number is 9

Number is 10

Out of loop

By using the pass statement in this program, we notice that the program runs exactly as it would if there were no conditional statement in the program. The pass statement tells the program to disregard that condition and continue to run the program as usual.

The pass statement can create minimal classes, or act as a placeholder when working on new code and thinking on an algorithmic level before hammering out details.