Python Loops

The following loops are available in Python to fulfil the looping needs. Python offers 3 choices for running the loops. The basic functionality of all the techniques is the same, although the syntax and the amount of time required for checking the condition differ.

We can run a single statement or set of statements repeatedly using a loop command.

The following sorts of loops are available in the Python programming language.

1	**While loop**	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	**For loop**	This type of loop executes a code block multiple times and abbreviates the code that manages the loop variable.
3	**Nested loops**	We can iterate a loop inside another loop.

Loop Control Statements

Statements used to control loops and change the course of iteration are called control statements. All the objects produced within the local scope of the loop are deleted when execution is completed.

Python provides the following control statements. We will discuss them later in detail.

Let us quickly go over the definitions of these loop control statements.

Sr.No.	Name of the control statement	Description
1	**Break statement**	This command terminates the loop's execution and transfers the program's control to the statement next to the loop.
2	**Continue statement**	This command skips the current iteration of the loop. The statements following the continue statement are not executed once the Python interpreter reaches the continue statement.
3	**Pass statement**	The pass statement is used when a statement is syntactically necessary, but no code is to be executed.

The for Loop

Python's for loop is designed to repeatedly execute a code block while iterating through a list, tuple, dictionary, or other iterable objects of Python. The process of traversing a sequence is known as iteration.

Syntax of the for Loop

```
1. for value in sequence:
2.     { code block }
```

In this case, the variable value is used to hold the value of every item present in the sequence before the iteration begins until this particular iteration is completed.

Loop iterates until the final item of the sequence are reached.

Code

```
# Python program to show how the for loop works

# Creating a sequence which is a tuple of numbers
numbers = [4, 2, 6, 7, 3, 5, 8, 10, 6, 1, 9, 2]

# variable to store the square of the number
square = 0

# Creating an empty list
squares = []

# Creating a for loop
for value in numbers:
    square = value ** 2
    squares.append(square)
print("The list of squares is", squares)
```

Output:

The list of squares is [16, 4, 36, 49, 9, 25, 64, 100, 36, 1, 81, 4]

Using else Statement with for Loop

As already said, a for loop executes the code block until the sequence element is reached. The statement is written right after the for loop is executed after the execution of the for loop is complete.

Only if the execution is complete does the else statement comes into play. It won't be executed if we exit the loop or if an error is thrown.

Here is a code to better understand if-else statements.

Code

```
# Python program to show how if-else statements work

string = "Python Loop"

# Initiating a loop
for s in a string:
    # giving a condition in if block
```

```
    if s == "o":
        print("If block")
# if condition is not satisfied then else block will be executed
else:
    print(s)
```

Output:

```
P
y
t
h
If block
n

L
If block
If block
p
```

Now similarly, using else with for loop.

Syntax:

```
for value in sequence:
    # executes the statements until sequences are exhausted
else:
    # executes these statements when for loop is completed
```

Code

```
# Python program to show how to use else statement with for loop

# Creating a sequence
tuple_ = (3, 4, 6, 8, 9, 2, 3, 8, 9, 7)

# Initiating the loop
for value in tuple_:
    if value % 2 != 0:
        print(value)
# giving an else statement
else:
    print("These are the odd numbers present in the tuple")
```

Output:

```
3
```

9
3
9
7
These are the odd numbers present in the tuple

The range() Function

With the help of the range() function, we may produce a series of numbers. range(10) will produce values between 0 and 9. (10 numbers).

We can give specific start, stop, and step size values in the manner range(start, stop, step size). If the step size is not specified, it defaults to 1.

Since it doesn't create every value it "contains" after we construct it, the range object can be characterized as being "slow." It does provide in, len, and __getitem__ actions, but it is not an iterator.

The example that follows will make this clear.

Code

```
1. # Python program to show the working of range() function
2.
3. print(range(15))
4.
5. print(list(range(15)))
6.
7. print(list(range(4, 9)))
8.
9. print(list(range(5, 25, 4)))
```

Output:

range(0, 15)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[4, 5, 6, 7, 8]
[5, 9, 13, 17, 21]
To iterate through a sequence of items, we can apply the range() method in for loops. We can use indexing to iterate through the given sequence by combining it with an iterable's len() function. Here's an illustration.

Code

```
1. # Python program to iterate over a sequence with the help of indexing
```

```
2.
3.  tuple_ = ("Python", "Loops", "Sequence", "Condition", "Range")
4.
5.  # iterating over tuple_ using range() function
6.  for iterator in range(len(tuple_)):
7.      print(tuple_[iterator].upper())
```

Output:

PYTHON
LOOPS
SEQUENCE
CONDITION
RANGE

While Loop

While loops are used in Python to iterate until a specified condition is met. However, the statement in the program that follows the while loop is executed once the condition changes to false.

Syntax of the while loop is:

```
1.  while <condition>:
2.      { code block }
```

All the coding statements that follow a structural command define a code block. These statements are intended with the same number of spaces. Python groups statements together with indentation.

Code

```
1.  # Python program to show how to use a while loop
2.  counter = 0
3.  # Initiating the loop
4.  while counter < 10: # giving the condition
5.      counter = counter + 3
6.      print("Python Loops")
```

Output:

Python Loops
Python Loops
Python Loops
Python Loops

Using else Statement with while Loops

As discussed earlier in the for loop section, we can use the else statement with the while loop also. It has the same syntax.

Code

```
1. #Python program to show how to use else statement with the while loop
2. counter = 0
3. 
4. # Iterating through the while loop
5. while (counter < 10):
6.     counter = counter + 3
7.     print("Python Loops") # Executed untile condition is met
8. # Once the condition of while loop gives False this statement will be executed
9. else:
10.    print("Code block inside the else statement")
```

Output:

Python Loops
Python Loops
Python Loops
Python Loops
Code block inside the else statement

Single statement while Block

The loop can be declared in a single statement, as seen below. This is similar to the if-else block, where we can write the code block in a single line.

Code

```
1. # Python program to show how to write a single statement while loop
2. counter = 0
3. while (count < 3): print("Python Loops")
```

Loop Control Statements

Now we will discuss the loop control statements in detail. We will see an example of each control statement.

Continue Statement

It returns the control to the beginning of the loop.

Code

```
# Python program to show how the continue statement works

# Initiating the loop
for string in "Python Loops":
    if string == "o" or string == "p" or string == "t":
        continue
    print('Current Letter:', string)
```

Output:

Current Letter: P
Current Letter: y
Current Letter: h
Current Letter: n
Current Letter:
Current Letter: L
Current Letter: s

Break Statement

It stops the execution of the loop when the break statement is reached.

Code

```
# Python program to show how the break statement works

# Initiating the loop
for string in "Python Loops":
    if string == 'L':
        break
    print('Current Letter: ', string)
```

Output:

Current Letter: P
Current Letter: y
Current Letter: t
Current Letter: h
Current Letter: o
Current Letter: n
Current Letter:

Pass Statement

Pass statements are used to create empty loops. Pass statement is also employed for classes, functions, and empty control statements.

Code

```
1. # Python program to show how the pass statement works
2. for a string in "Python Loops":
3.     pass
4. print( 'Last Letter:', string)
```

Output:

Last Letter: s

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

```
for value in sequence:
    {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

```
# Code to find the sum of squares of each element of the list using for loop

# creating the list of numbers
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]

# initializing a variable that will store the sum
sum_ = 0

# using for loop to iterate over the list
for num in numbers:

sum_ = sum_ + num ** 2

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

```
my_list = [3, 5, 6, 8, 4]
for iter_var in range( len( my_list ) ):
    my_list.append(my_list[iter_var] + 2)
print( my_list )
```

Output:

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

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

```
# Code to find the sum of squares of each element of the list using for loop

# creating the list of numbers
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]

# initializing a variable that will store the sum
sum_ = 0

# using for loop to iterate over list
for num in range( len(numbers) ):

sum_ = sum_ + numbers[num] ** 2

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

```
# code to print marks of a student from the record
student_name_1 = 'Itika'
student_name_2 = 'Parker'


# Creating a dictionary of records of the students
records = {'Itika': 90, 'Arshia': 92, 'Peter': 46}
def marks( student_name ):
    for a_student in record: # for loop will iterate over the keys of the dictionary
        if a_student == student_name:
            return records[ a_student ]
            break
    else:
        return f'There is no student of name {student_name} in the records'

# giving the function marks() name of two students
print( f"Marks of {student_name_1} are: ", marks( student_name_1 ) )
print( f"Marks of {student_name_2} are: ", marks( student_name_2 ) )
```

Output:

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

Python While Loops

In coding, loops are designed to execute a specified code block repeatedly. We'll learn how to construct a while loop in Python, the syntax of a while loop, loop controls like break and continue, and other exercises in this tutorial.

Introduction of Python While Loop

In this article, we are discussing while loops in Python. The Python while loop iteration of a code block is executed as long as the given Condition, i.e., conditional_expression, is true.

If we don't know how many times we'll execute the iteration ahead of time, we can write an indefinite loop.

Syntax of Python While Loop

Now, here we discuss the syntax of the Python while loop. The syntax is given below -

```
Statement
while Condition:
        Statement
```

The given condition, i.e., conditional_expression, is evaluated initially in the Python while loop. Then, if the conditional expression gives a boolean value True, the while loop statements are executed. The conditional expression is verified again when the complete code block is executed. This procedure repeatedly occurs until the conditional expression returns the boolean value False.

- The statements of the Python while loop are dictated by indentation.
- The code block begins when a statement is indented & ends with the very first unindented statement.
- Any non-zero number in Python is interpreted as boolean True. False is interpreted as None and 0.

Example

Now we give some examples of while Loop in Python. The examples are given in below -

Program code 1:

Now we give code examples of while loops in Python for printing numbers from 1 to 10. The code is given below -

```
i=1
while i<=10:
    print(i, end=' ')
```

```
    i+=1
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

1 2 3 4 5 6 7 8 9 10

Program Code 2:

Now we give code examples of while loops in Python for Printing those numbers divisible by either 5 or 7 within 1 to 50 using a while loop. The code is given below -

```
i=1
while i<51:
    if i%5 == 0 or i%7==0 :
        print(i, end=' ')
    i+=1
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

5 7 10 14 15 20 21 25 28 30 35 40 42 45 49 50

Program Code:

Now we give code examples of while loops in Python, the sum of squares of the first 15 natural numbers using a while loop. The code is given below -

```
# Python program example to show the use of while loop

num = 15

# initializing summation and a counter for iteration
summation = 0
c = 1

while c <= num: # specifying the condition of the loop
    # begining the code block
    summation = c**2 + summation
    c = c + 1    # incrementing the counter

# print the final sum
print("The sum of squares is", summation)
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

The sum of squares is 1240
Provided that our counter parameter i gives boolean true for the condition, i less than or equal to num, the loop repeatedly executes the code block i number of times.

Next is a crucial point (which is mostly forgotten). We have to increment the counter parameter's value in the loop's statements. If we don't, our while loop will execute itself indefinitely (a never-ending loop).

Finally, we print the result using the print statement.

Exercises of Python While Loop

Prime Numbers and Python While Loop

Using a while loop, we will construct a Python program to verify if the given integer is a prime number or not.

Program Code:

Now we give code examples of while loops in Python for a number is Prime number or not. The code is given below -

```
num = [34, 12, 54, 23, 75, 34, 11]

def prime_number(number):
    condition = 0
    iteration = 2
    while iteration <= number / 2:
        if number % iteration == 0:
            condition = 1
            break
        iteration = iteration + 1

    if condition == 0:
        print(f"{number} is a PRIME number")
    else:
        print(f"{number} is not a PRIME number")
for i in num:
```

```
17.     prime_number(i)
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
34 is not a PRIME number
12 is not a PRIME number
54 is not a PRIME number
23 is a PRIME number
75 is not a PRIME number
34 is not a PRIME number
11 is a PRIME number
```

2. Armstrong and Python While Loop

We will construct a Python program using a while loop to verify whether the given integer is an Armstrong number.

Program Code:

Now we give code examples of while loops in Python for a number is Armstrong number or not. The code is given below -

```
1.  n = int(input())
2.  n1=str(n)
3.  l=len(n1)
4.  temp=n
5.  s=0
6.  while n!=0:
7.      r=n%10
8.      s=s+(r**1)
9.      n=n//10
10. if s==temp:
11.     print("It is an Armstrong number")
12. else:
13.     print("It is not an Armstrong number ")
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

It is not an Armstrong number

Multiplication Table using While Loop

In this example, we will use the while loop for printing the multiplication table of a given number.

Program Code:

In this example, we will use the while loop for printing the multiplication table of a given number. The code is given below -

```
num = 21
counter = 1
# we will use a while loop for iterating 10 times for the multiplication table
print("The Multiplication Table of: ", num)
while counter <= 10: # specifying the condition
    ans = num * counter
    print (num, 'x', counter, '=', ans)
    counter += 1 # expression to increment the counter
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

The Multiplication Table of: 21
21 x 1 = 21
21 x 2 = 42
21 x 3 = 63
21 x 4 = 84
21 x 5 = 105
21 x 6 = 126
21 x 7 = 147
21 x 8 = 168
21 x 9 = 189
21 x 10 = 210

Python While Loop with List

Program Code 1:

Now we give code examples of while loops in Python for square every number of a list. The code is given below -

```
# Python program to square every number of a list
```

```
# initializing a list
list_ = [3, 5, 1, 4, 6]
squares = []
# programing a while loop
while list_: # until list is not empty this expression will give boolean True after that F
alse
    squares.append( (list_.pop())**2)
#  Print the squares of all numbers.
print( squares )
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

[36, 16, 1, 25, 9]
In the preceding example, we execute a while loop over a given list of integers that will repeatedly run if an element in the list is found.

Program Code 2:

Now we give code examples of while loops in Python for determine odd and even number from every number of a list. The code is given below -

```
list_ = [3, 4, 8, 10, 34, 45, 67,80]       # Initialize the list
index = 0
while index < len(list_):
    element = list_[index]
    if element % 2 == 0:
        print('It is an even number')      # Print if the number is even.
    else:
        print('It is an odd number')       # Print if the number is odd.
    index += 1
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

It is an odd number
It is an even number
It is an even number
It is an even number
It is an even number
It is an odd number

It is an odd number
It is an even number

Program Code 3:

Now we give code examples of while loops in Python for determine the number letters of every word from the given list. The code is given below -

```
1. List_= ['Priya', 'Neha', 'Cow', 'To']
2. index = 0
3. while index < len(List_):
4.     element = List_[index]
5.     print(len(element))
6.     index += 1
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

5
4
3
2

Python While Loop Multiple Conditions

We must recruit logical operators to combine two or more expressions specifying conditions into a single while loop. This instructs Python on collectively analyzing all the given expressions of conditions.

We can construct a while loop with multiple conditions in this example. We have given two conditions and a and keyword, meaning the Loop will execute the statements until both conditions give Boolean True.

Program Code:

Now we give code examples of while loops in Python for multiple condition. The code is given below -

```
1. num1 = 17
2. num2 = -12
3.
4. while num1 > 5 and num2 < -5 : # multiple conditions in a single while loop
5.     num1 -= 2
6.     num2 += 3
```

```
7.      print( (num1, num2) )
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

(15, -9)
(13, -6)
(11, -3)

Let's look at another example of multiple conditions with an OR operator.

Code

```
1. num1 = 17
2. num2 = -12
3.
4. while num1 > 5 or num2 < -5 :
5.     num1 -= 2
6.     num2 += 3
7.     print( (num1, num2) )
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

(15, -9)
(13, -6)
(11, -3)
(9, 0)
(7, 3)
(5, 6)

We can also group multiple logical expressions in the while loop, as shown in this example.

Code

```
1. num1 = 9
2. num = 14
3. maximum_value = 4
4. counter = 0
5. while (counter < num1 or counter < num2) and not counter >= maximum_value: # gr
   ouping multiple conditions
6.     print(f"Number of iterations: {counter}")
7.     counter += 1
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

Number of iterations: 0
Number of iterations: 1
Number of iterations: 2
Number of iterations: 3

Single Statement While Loop

Similar to the if statement syntax, if our while clause consists of one statement, it may be written on the same line as the while keyword.

Here is the syntax and example of a one-line while clause -

```
1. # Python program to show how to create a single statement while loop
2. counter = 1
3. while counter: print('Python While Loops')
```

Loop Control Statements

Now we will discuss the loop control statements in detail. We will see an example of each control statement.

Continue Statement

It returns the control of the Python interpreter to the beginning of the loop.

Code

```
1. # Python program to show how to use continue loop control
2.
3. # Initiating the loop
4. for string in "While Loops":
5.     if string == "o" or string == "i" or string == "e":
6.         continue
7.     print('Current Letter:', string)
```

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

Output:

Current Letter: W
Current Letter: h
Current Letter: l
Current Letter:
Current Letter: L
Current Letter: p
Current Letter: s

Break Statement

It stops the execution of the loop when the break statement is reached.

Code

```
1. # Python program to show how to use the break statement
2.
3. # Initiating the loop
4. for string in "Python Loops":
5.     if string == 'n':
6.         break
7.     print('Current Letter: ', string)
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

Current Letter: P
Current Letter: y
Current Letter: t
Current Letter: h
Current Letter: o

Pass Statement

Pass statements are used to create empty loops. Pass statement is also employed for classes, functions, and empty control statements.

Code

```
1. # Python program to show how to use the pass statement
2. for a string in "Python Loops":
3.     pass
4. print( 'The Last Letter of given string is:', string)
```

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

Output:

The Last Letter of given string is: s

Python break statement

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition. The syntax of the break statement in Python is given below.

Syntax:

```
1. #loop statements
2. break;
```

Example 1 : break statement with for loop

Code

```
1. # break statement example
2. my_list = [1, 2, 3, 4]
3. count = 1
4. for item in my_list:
5.     if item == 4:
6.         print("Item matched")
7.         count += 1
8.         break
9. print("Found at location", count)
```

Output:

Item matched
Found at location 2
In the above example, a list is iterated using a for loop. When the item is matched with value 4, the break statement is executed, and the loop terminates. Then the count is printed by locating the item.

Example 2 : Breaking out of a loop early

Code

```
1. # break statement example
2. my_str = "python"
3. for char in my_str:
4.     if char == 'o':
5.         break
6.     print(char)
```

Output:

p
y
t
h

When the character is found in the list of characters, break starts executing, and iterating stops immediately. Then the next line of the print statement is printed.

Example 3: break statement with while loop

Code

```
1. # break statement example
2. i = 0;
3. while 1:
4.     print(i," ",end=""),
5.     i=i+1;
6.     if i == 10:
7.         break;
8. print("came out of while loop");
```

Output:

0 1 2 3 4 5 6 7 8 9 came out of while loop

It is the same as the above programs. The while loop is initialised to True, which is an infinite loop. When the value is 10 and the condition becomes true, the break statement will be executed and jump to the later print statement by terminating the while loop.

Example 4 : break statement with nested loops

Code

```
1. # break statement example
2. n = 2
3. while True:
4.     i = 1
5.     while i <= 10:
```

```
        print("%d X %d = %d\n" % (n, i, n * i))
        i += 1
    choice = int(input("Do you want to continue printing the table? Press 0 for no: "))
    if choice == 0:
        print("Exiting the program...")
        break
    n += 1
print("Program finished successfully.")
```

Output:

2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
Do you want to continue printing the table? Press 0 for no: 1
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
Do you want to continue printing the table? Press 0 for no: 0
Exiting the program...
Program finished successfully.

There are two nested loops in the above program. Inner loop and outer loop The inner loop is responsible for printing the multiplication table, whereas the outer loop is responsible for incrementing the n value. When the inner loop completes execution, the user will have to continue printing. When 0 is entered, the break statement finally executes, and the nested loop is terminated.

Python continue Statement

Python continue keyword is used to skip the remaining statements of the current loop and go to the next iteration. In Python, loops repeat processes on their own in an efficient way. However, there might be occasions when we wish to leave the current loop entirely, skip iteration, or dismiss the condition controlling the loop.

We use Loop control statements in such cases. The continue keyword is a loop control statement that allows us to change the loop's control. Both Python while and Python for loops can leverage the continue statements.

Syntax:

```
1. continue
```

Python Continue Statements in for Loop

Printing numbers from 10 to 20 except 15 can be done using continue statement and for loop. The following code is an example of the above scenario:

Code

```
1. # Python code to show example of continue statement
2. 
3. # looping from 10 to 20
4. for iterator in range(10, 21):
5. 
6.     # If iterator is equals to 15, loop will continue to the next iteration
7.     if iterator == 15:
8.         continue
9.     # otherwise printing the value of iterator
10.    print( iterator )
```

Output:

```
10
11
12
13
14
16
```

17
18
19
20

Explanation: We will execute a loop from 10 to 20 and test the condition that the iterator is equal to 15. If it equals 15, we'll employ the continue statement to skip to the following iteration displaying any output; otherwise, the loop will print the result.

Python Continue Statements in while Loop

Code

```
# Creating a string
string = "JavaTpoint"
# initializing an iterator
iterator = 0

# starting a while loop
while iterator < len(string):
    # if loop is at letter a it will skip the remaining code and go to next iteration
    if string[iterator] == 'a':
        continue
    # otherwise it will print the letter
    print(string[ iterator ])
    iterator += 1
```

Output:

J
v
T
p
o
i
n
t

Explanation: We will take a string "Javatpoint" and print each letter of the string except "a". This time we will use Python while loop to do so. Until the value of the iterator is less than the string's length, the while loop will keep executing.

Python Continue statement in list comprehension

Let's see the example for continue statement in list comprehension.

Code

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Using a list comprehension with continue
sq_num = [num ** 2 for num in numbers if num % 2 == 0]
```

```
# This will skip odd numbers and only square the even numbers
print(sq_num)
```

Output:

[4, 16, 36, 64, 100]

Explanation: In the above code, list comprehension will square the numbers from the list. And continue statement will be encountered when odd numbers come and the loop will skip the execution and moves to the next iterartion.

Python Continue vs. Pass

Usually, there is some confusion in the pass and continue keywords. So here are the differences between these two.

Definition	The continue statement is utilized to skip the current loop's remaining statements, go to the following iteration, and return control to the beginning.	The pass keyword is used when a phrase is necessary syntactically to be placed but not to be executed.
Action	It takes the control back to the start of the loop.	Nothing happens if the Python interpreter encounters the pass statement.
Application	It works with both the Python while and Python for loops.	It performs nothing; hence it is a null operation.
Syntax	It has the following syntax: -: continue	Its syntax is as follows:- pass
Interpretation	It's mostly utilized within a loop's condition.	During the byte-compile stage, the pass keyword is removed.