FLOW CONTROL

Types Of Flow Control

- Sequential Flow
- Conditional Flow
- Iterative Flow
- Transfer Flow

--

Conditional or Selection Statements

- if

  Syntax :

```
  if(condition):
    [statements inside if block]
  [statements outside if block]
```

- if else

  Syntax :

```
if(condition):
     [statements inside if block]
else:
     [statements inside else block]
[statements outside if block]
```

- if elif else

  Syntax:

```
if(condition):
     [statements inside if block]
elif(condition):
     [statements inside elif block]
else:
     [statements inside else block]
[statements outside if block]
```

- if elif

Syntax :

```
if(condition):
     [statements inside if block]
elif(condition):
     [statements inside elif block]
[statements outside if block]
```

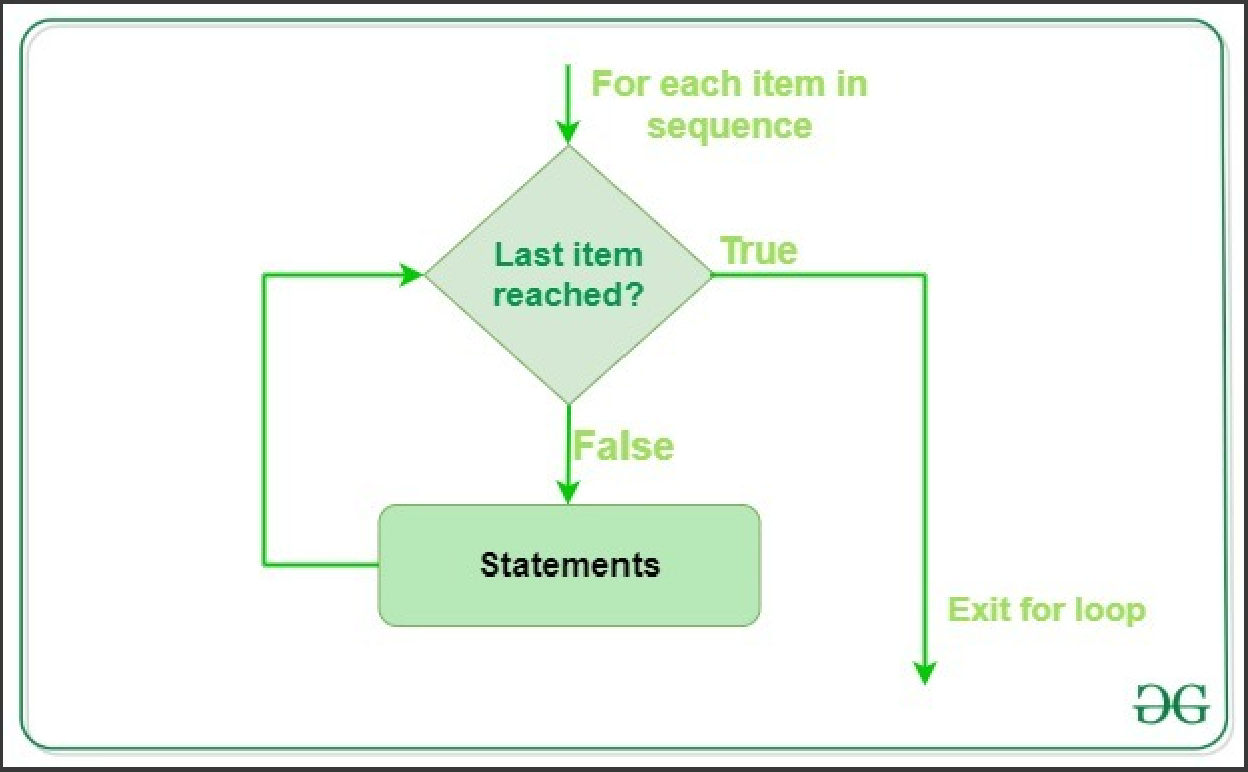
--

Iterative Staements

- for loop
- while loop

for loop Syntax:

```
for iterator_var in sequence:
     [statements inside for loop block]
[statements outside for loop block]
```



Python program

```
# Iterating over a list
print("List Iteration")
```

```
l = ["geeks", "for", "geeks"]
for i in l:
    print(i)

# Iterating over a tuple (immutable)
print("\nTuple Iteration")
t = ("geeks", "for", "geeks")
for i in t:
    print(i)

# Iterating over a String
print("\nString Iteration")
s = "Geeks"
for i in s:
    print(i)

# Iterating over dictionary
print("\nDictionary Iteration")
d = dict()
d['xyz'] = 123
d['abc'] = 345
for i in d:
    print("%s %d" % (i, d[i]))

# Iterating over a set
print("\nSet Iteration")
set1 = {1, 2, 3, 4, 5, 6}
for i in set1:
    print(i),
```

```
# Output

List Iteration
geeks
for
geeks

Tuple Iteration
geeks
for
geeks

String Iteration
G
e
e
```

```
k
s

Dictionary Iteration
xyz  123
abc  345

Set Iteration
1
2
3
4
5
6
```

Using else statement with for loop in Python

We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate so the else block will be executed immediately after for block finishes execution.

Python program

```
# combining else with for

list = ["geeks", "for", "geeks"]
for index in range(len(list)):
    print(list[index])
else:
    print("Inside Else Block")
```

```
# Output
geeks
for
geeks
Inside Else Block
```

Python For Loop with Zip()

This code uses the zip() function to iterate over two lists (fruits and colors) in parallel. The for loop assigns the corresponding elements of both lists to the variables fruit and color in each iteration. Inside the loop, the print() function is used to display the message “is” between the fruit and color values. The output will display each fruit from the list of fruits along with its corresponding color from the colours list.

```
fruits = ["apple", "banana", "cherry"]
colors = ["red", "yellow", "green"]
for fruit, color in zip(fruits, colors):
    print(fruit, "is", color)
Output :

apple is red
banana is yellow
cherry is green
```

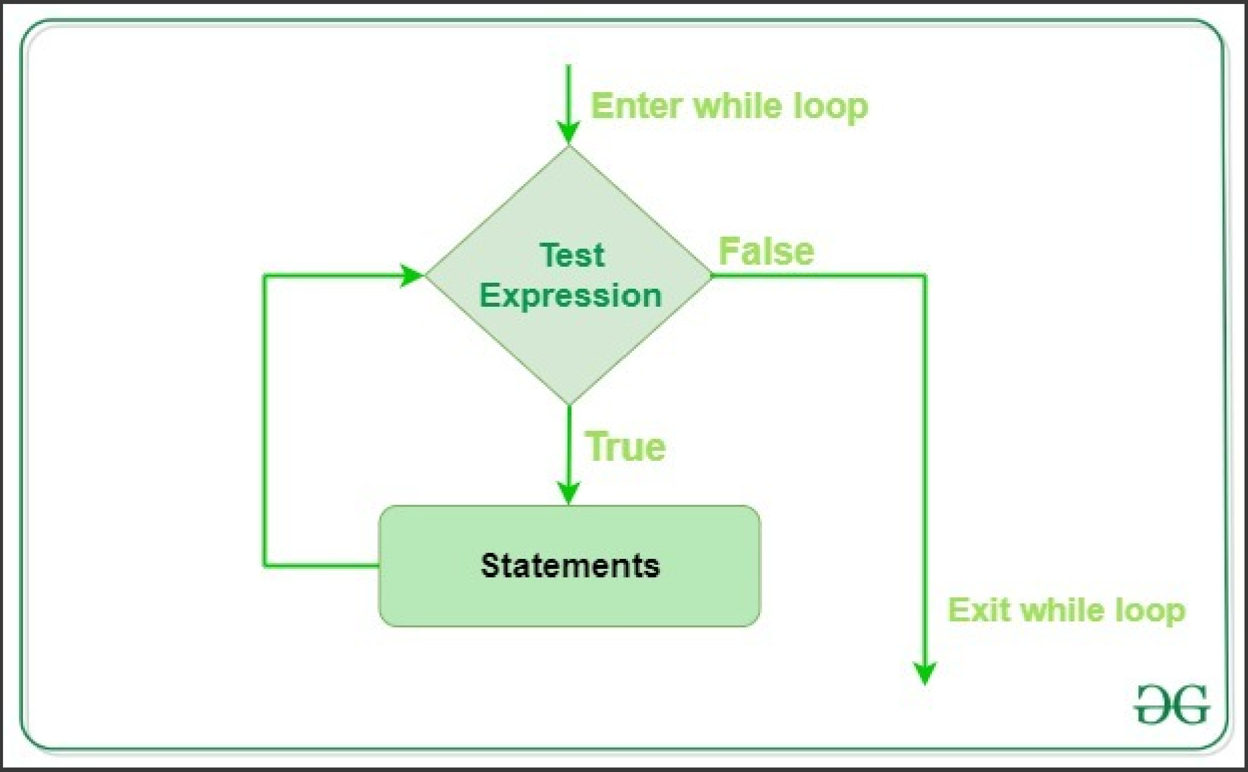
while loop Syntax:

Syntax:

```
while expression:
    [statements inside while loop block]
[statements outside while loop block]



while condition:
    [statements inside while loop block]
else:
    [statements inside else block]
[statements outside else block]
```



Jump Statements or Control Statements

Using loops in Python automates and repeats the tasks in an efficient manner. But sometimes, there may arise a condition where you want to exit the loop completely, skip an iteration or ignore some statements of the loop before continuing further in the loop. These can be done by loop control statements called jump statements. Loop control or jump statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control/jump statements.

- break
- continue
- pass

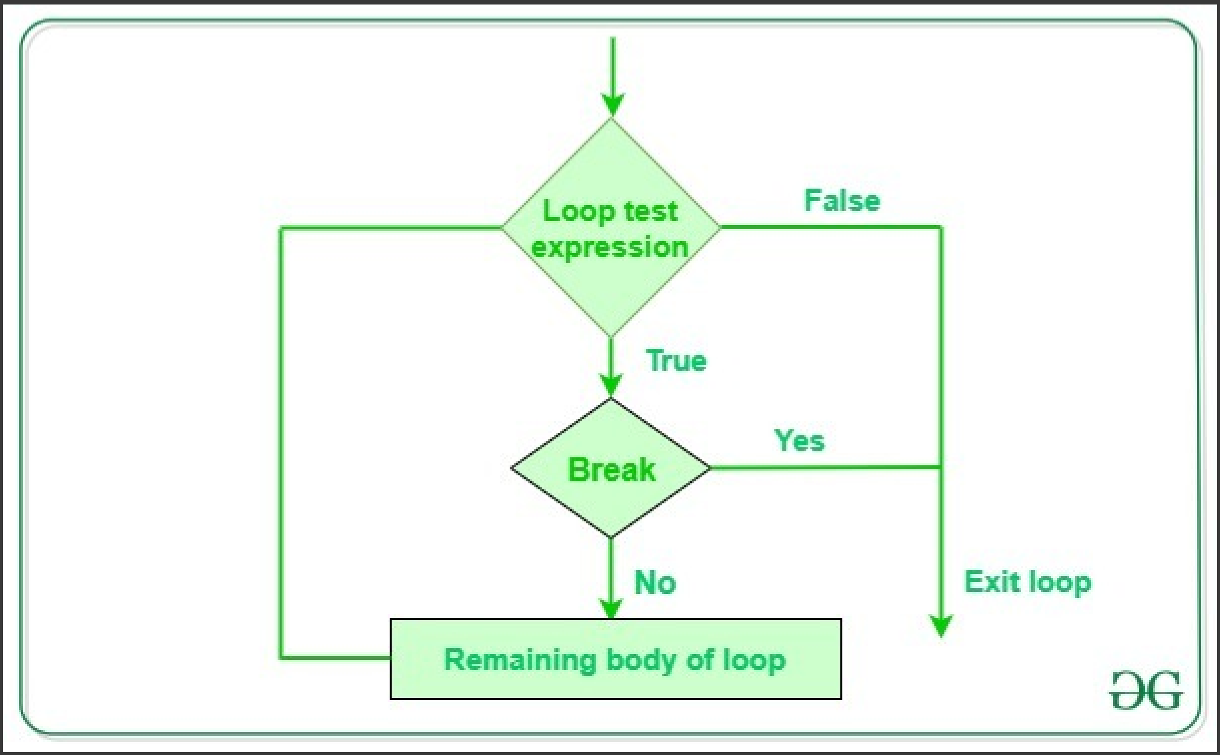
--

break statement

Python break is used to terminate the execution of the loop.

break statement Syntax:

```
Loop{
    Condition:
        break
    }
```



```
for i in range(10):
    print(i)
    if i == 2:
        break

# output
0
1
2
```

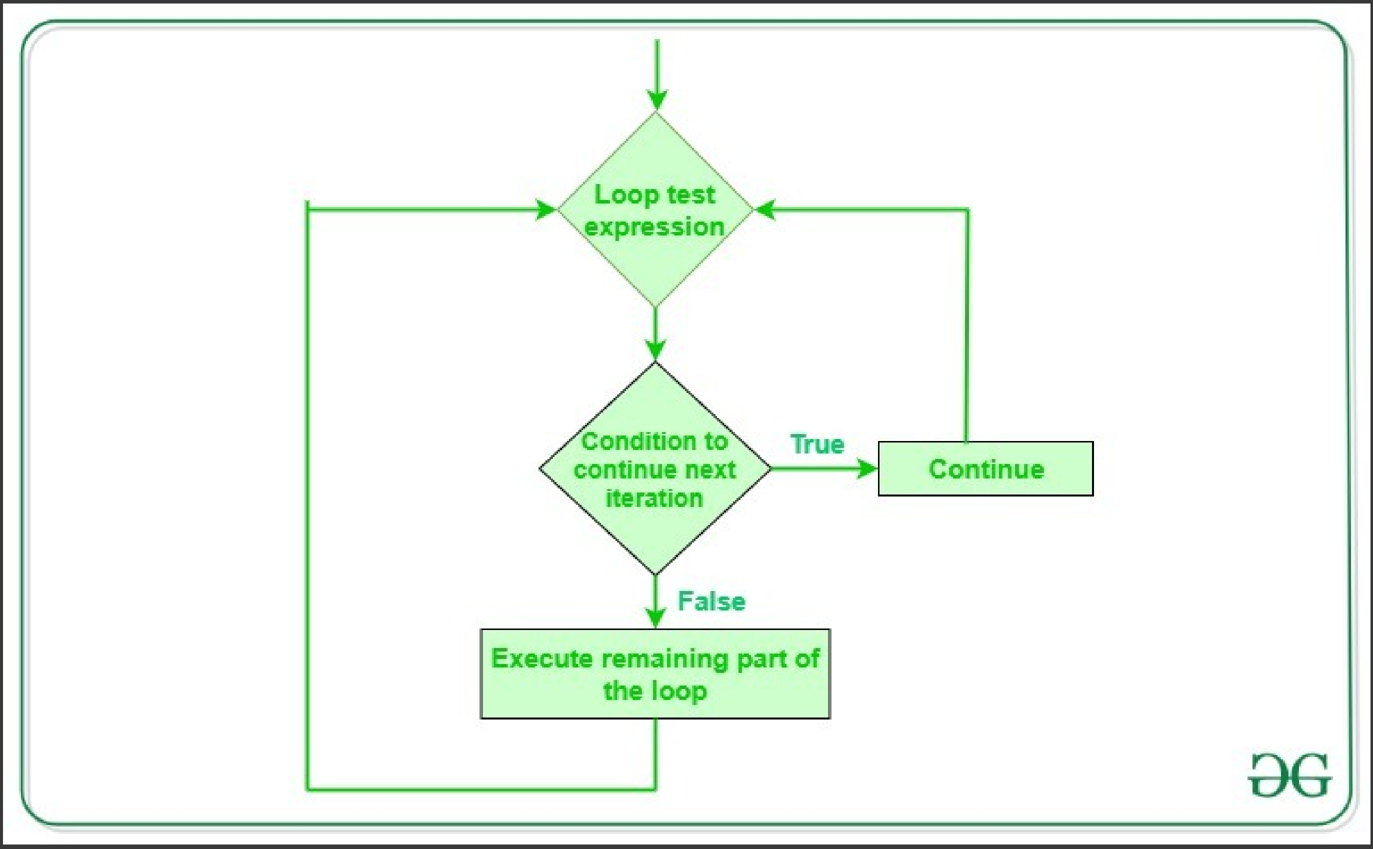
--

continue statement

Python Continue Statement skips the execution of the program block after the continue statement and forces the control to start the next iteration.

continue Statement Syntax

```
while True:
    ...
    if x == 10:
        continue
    print(x)
```


Loop test expression
Condition to continue next iteration
True
Continue
False
Execute remaining part of the loop

```
for var in "Geeksforgeeks":
    if var == "e":
        continue
    print(var)
# output
G
k
s
f
o
r
g
k
s
```

—

pass statement

When the user does not know what code to write, So user simply places a pass at that line. Sometimes, the pass is used when the user doesn’t want any code to execute. So users can simply place a pass where empty code is not allowed, like in loops, function definitions, class definitions, or in if statements. So using a pass statement user avoids this error.

```
n = 26

if n > 26:
    # write code your here
print('Geeks')

# Output
IndentationError: expected an indented block after 'if' statement
```

Use of pass keyword in Python Loop

The pass keyword can be used in Python for loop, when a user doesn’t know what to code inside the loop in Python.

```
n = 10
for i in range(n):
  # pass can be used as placeholder
  # when code is to added later
  pass
```

Use of pass keyword in Conditional statement

Python pass keyword can be used with conditional statements.

```
a = 10
b = 20

if(a<b):
  pass
else:
  print("b<a")
```

Let’s take another example in which the pass statement gets executed when the condition is true.

```


li =['a', 'b', 'c', 'd']

for i in li:
    if(i =='a'):
        pass
    else:
        print(i)
Output:

b
c
d
```

Python The pass Keyword in If

In the first example, the pass statement is used as a placeholder inside an if statement. If the condition in the if statement is true, no action is taken, but the program will not raise a syntax error because the pass statement is present.

In the second example, the pass statement is used inside a function definition as a placeholder for implementation. This is useful when defining a function that will be used later, but for which the implementation has not yet been written.

In the third example, the pass statement is used inside a class definition as a placeholder for implementation. This is useful when defining a class that will be used later, but for which the implementation has not yet been written.

Note that in all cases, the pass statement is followed by a colon (:) to indicate the start of a code block, but there is no code inside the block. This is what makes it a placeholder statement.

```
# Using pass as a placeholder inside an if statement
x = 5
if x > 10:
    pass
else:
    print("x is less than or equal to 10")

# Using pass in a function definition as a
# placeholder for implementation
def my_function():
    pass

# Using pass in a class definition as
# a placeholder for implementation

class MyClass:
    def __init__(self):
        pass
# Output
x is less than or equal to 10
```