

## Module-2

**Conditional statements:** if, if-else, elif, nested if;

**Looping statements:** while, for, for each, nested loops; break, continue; Else statement with loop;

**Functions Basics:** Built-in Functions, Declaring and calling user defined functions, Parameters and default arguments, Fruitful functions and void functions, recursive functions.

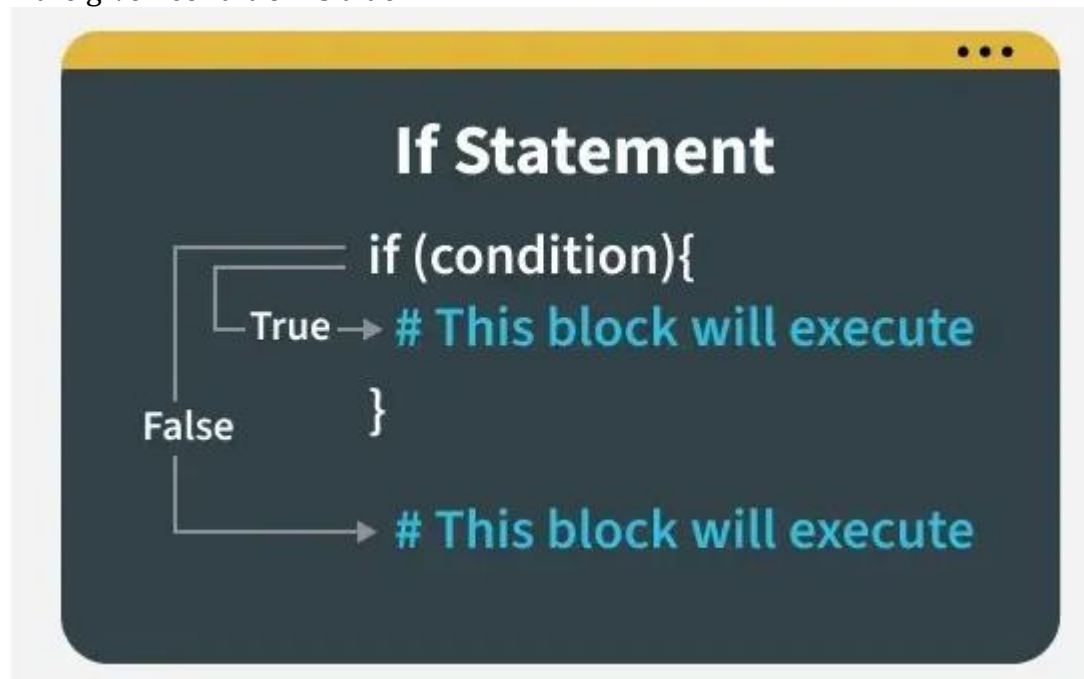
### Chapter 1

#### Conditional Statements in Python

Conditional statements in Python are used to execute certain blocks of code based on specific conditions. These statements help control the flow of a program, making it behave differently in different situations.

#### If Conditional Statement

If statement is the simplest form of a conditional statement. It executes a block of code if the given condition is true.



```
age = 20
if age >= 18:
    print("Eligible to vote.")
```

### Output

```
Eligible to vote.
```

### Short Hand if

Short-hand if statement allows us to write a single-line if statement.

```
age = 19
if age > 18: print("Eligible to Vote.")
```

### Output

```
Eligible to Vote.
```

This is a compact way to write an if statement. It executes print statement if the condition is true.

### If else Conditional Statement:

If Else allows us to specify a block of code that will execute if the condition(s) associated with an if or elif statement evaluates to False. Else block provides a way to handle all other cases that don't meet the specified conditions.

# If...else Statement



*If Else Statement*

```
age = 10
if age <= 12:
    print("Travel for free.")
else:
    print("Pay for ticket.")
```

## Output

Travel for free.

## Short Hand if-else

The short-hand if-else statement allows us to write a single-line if-else statement.

```
marks = 45
res = "Pass" if marks >= 40 else "Fail"
print(f"Result: {res}")
```

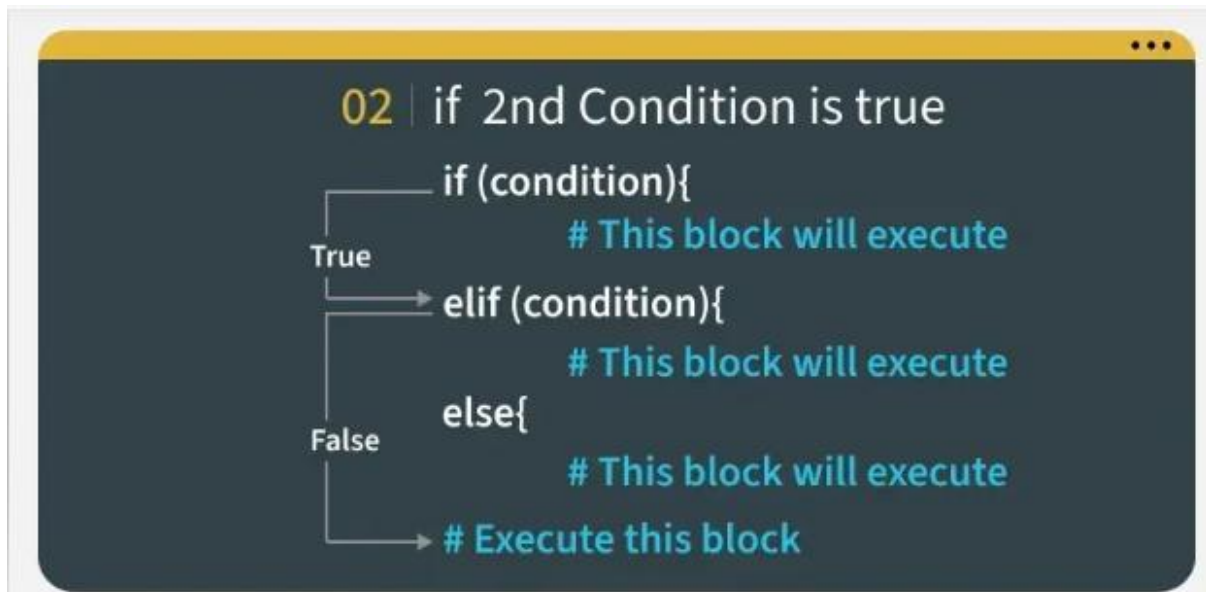
## Output

Result: Pass

**Note:** This method is also known as ternary operator. Ternary Operator essentially a shorthand for the if-else statement that allows us to write more compact and readable code, especially for simple conditions.

### elif Statement:

elif statement in Python stands for "else if." It allows us to check multiple conditions, providing a way to execute different blocks of code based on which condition is true. Using elif statements makes our code more readable and efficient by eliminating the need for multiple nested if statements.



```
age = 25
```

```
if age <= 12:  
    print("Child.")  
elif age <= 19:  
    print("Teenager.")  
elif age <= 35:  
    print("Young adult.")  
else:  
    print("Adult.")
```

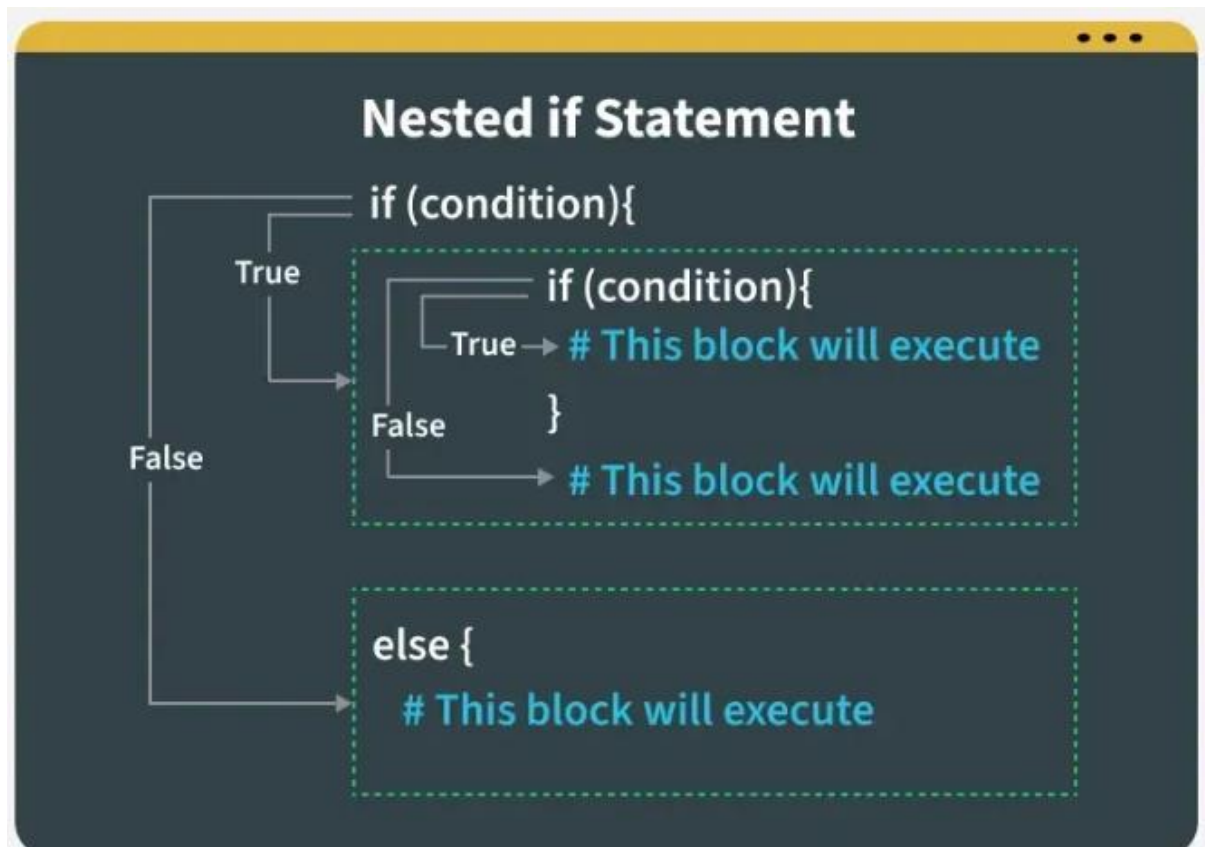
### Output

```
Young adult.
```

The code checks the value of age using if-elif-else. Since age is 25, it skips the first two conditions (age <= 12 and age <= 19), and the third condition (age <= 35) is True, so it prints "Young adult."

### Nested if..else Conditional Statement:

Nested if..else means an if-else statement inside another if statement. We can use nested if statements to check conditions within conditions.



```
age = 70
is_member = True

if age >= 60:
    if is_member:
        print("30% senior discount!")
    else:
        print("20% senior discount.")
else:
    print("Not eligible for a senior discount.")
```

### Output

```
30% senior discount!
```

## Chapter 2

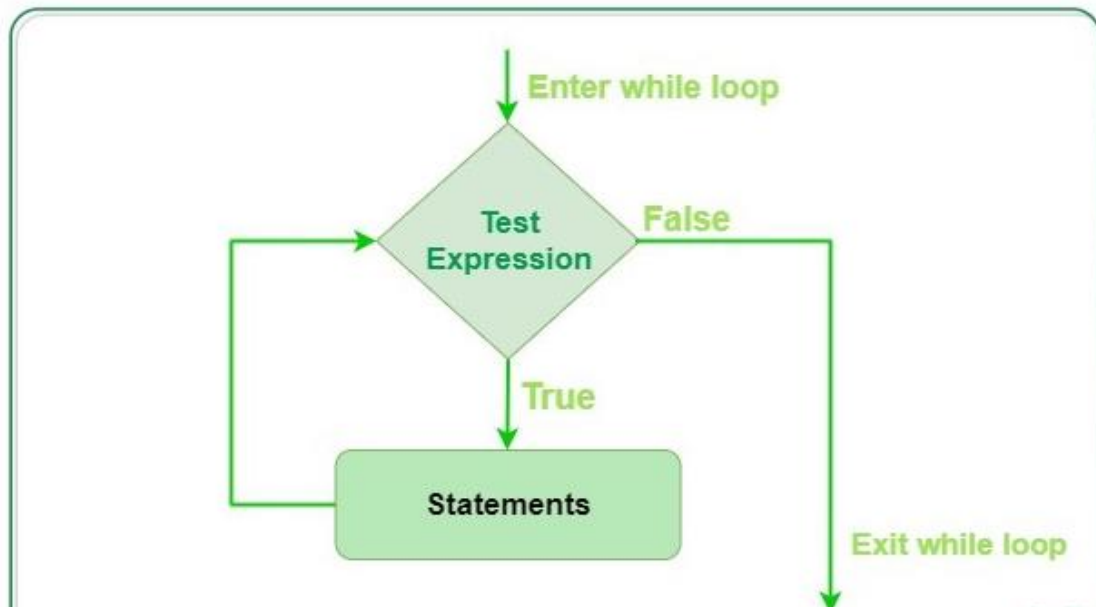
**Looping statements:** while, for, for each, nested loops; break, continue; Else statement with loop;

Loops in Python are used to repeat actions efficiently. The main types are For loops (counting through items) and While loops (based on conditions). In this article, we will look at Python loops and understand their working with the help of examples.

### While Loop

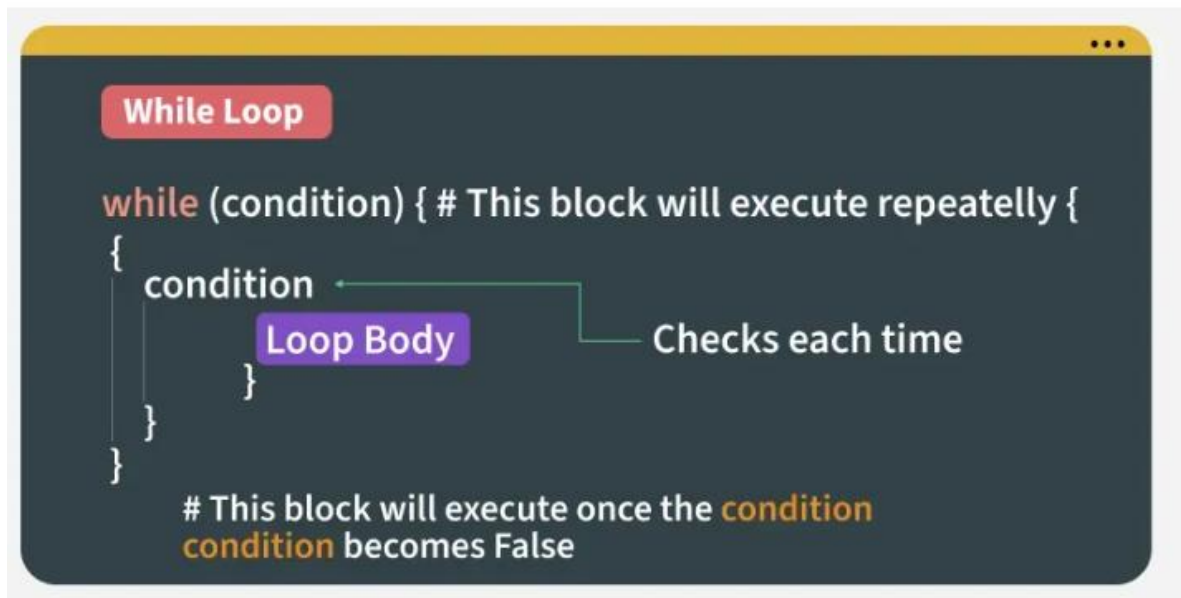
In Python, a while loop is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the line immediately after the loop in the program is executed.

### While Loop Flowchart



## Syntax:

```
while expression:  
    statement(s)
```



while loop explanation

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.

In below code, loop runs as long as the condition `cnt < 3` is true. It increments the counter by 1 on each iteration and prints "Hello Students" three times.

```
cnt = 0  
while (cnt < 3):  
    cnt = cnt + 1  
    print("Hello Students")
```



```
Hello Students  
Hello Students  
Hello Students
```

## For Loop

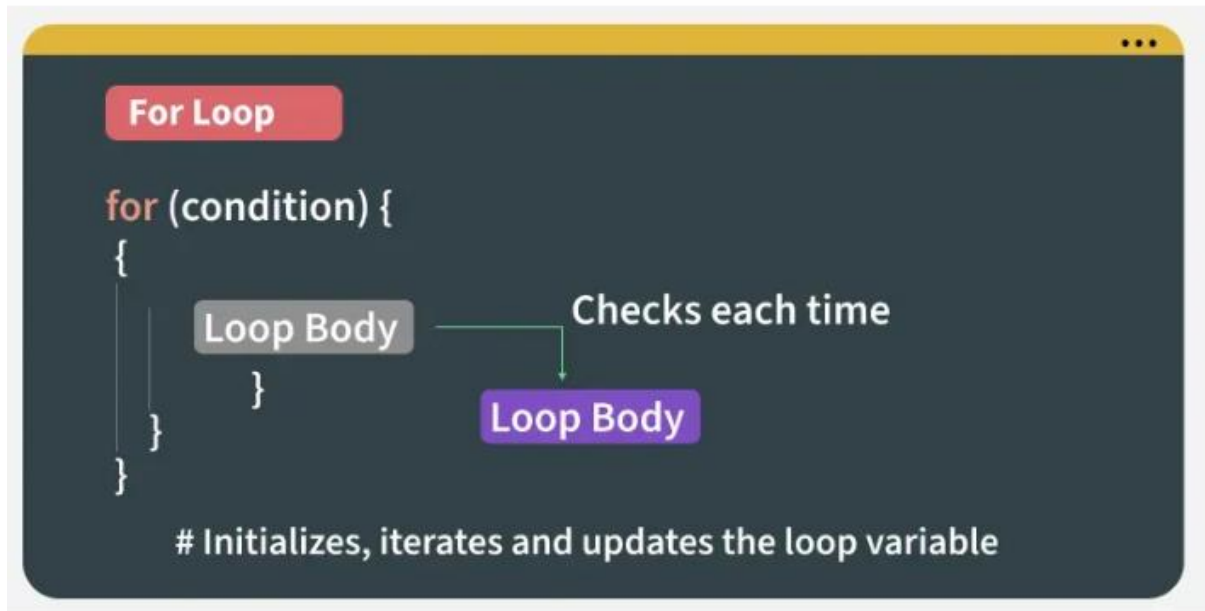
For loops is used to iterate over a sequence such as a list, tuple, string or range. It allow to execute a block of code repeatedly, once for each item in the sequence.

Python **for loops** are used for iterating over sequences like lists, tuples, strings and ranges.

- A for loop allows you to apply the same operation to every item within the loop.
- Using a for loop avoids the need to manually manage the index.
- A for loop can iterate over any iterable object, such as a dictionary, list or custom iterator.

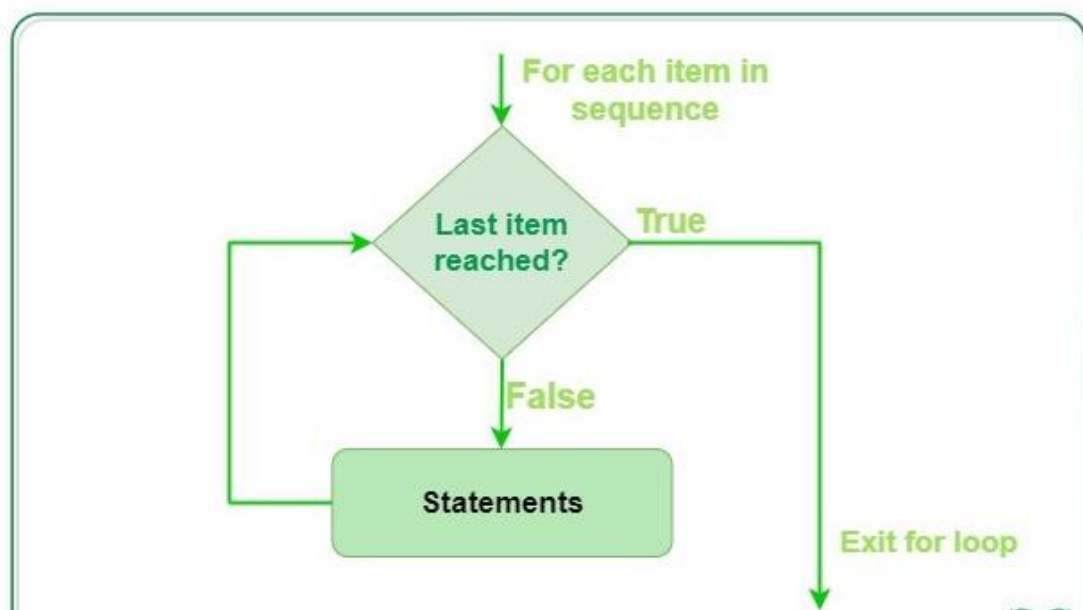
## Syntax:

```
for iterator_var in sequence:  
    statements(s)
```



*For loop explanation*

## Flowchart of For Loop



### Example:

```
n = 4
for i in range(0, n):
    print(i)
```

### Output

```
0
1
2
3
```

### Nested Loops:

Python programming language allows to use one loop inside another loop which is called nested loop. Following section shows few examples to illustrate the concept.

#### Syntax:

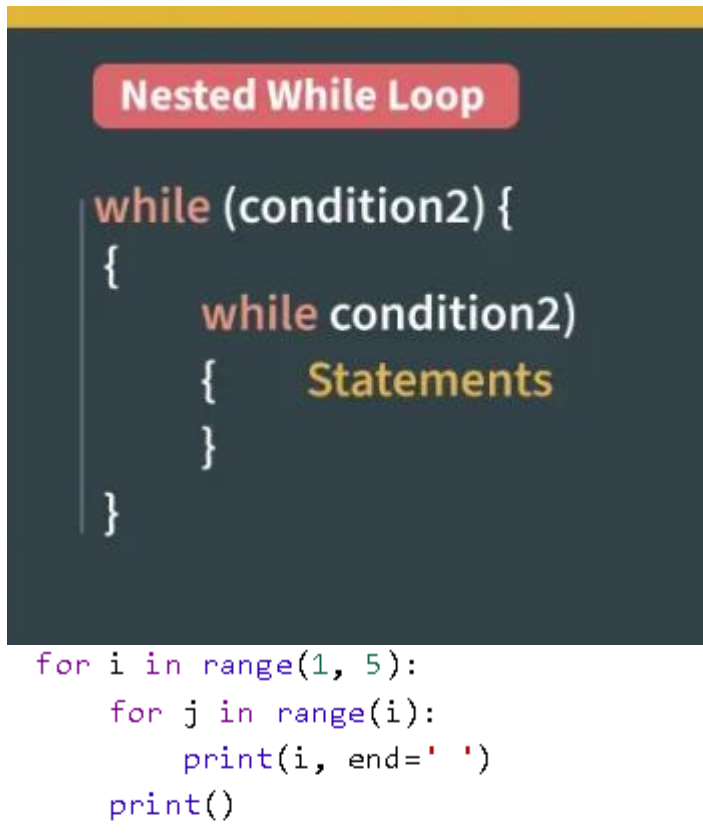
```
for iterator_var in sequence:
    for iterator_var in sequence:
        statements(s)
    statements(s)
```

#### Nested for Loop

```
for (condition 1)
    └───> # This is the
          # This is the inner loop
for (condition 2)
    └───> Outer print statement
False}
}
```

The syntax for a nested while loop statement in the Python programming language is as follows:

```
while expression:
    while expression:
        statement(s)
    statement(s)
```



## Output

```
1
2 2
3 3 3
4 4 4 4
```

A final note on loop nesting is that we can put any type of loop inside of any other type of loops in Python. For example, a for loop can be inside a while loop or vice versa.

**Explanation:** In the above code we use nested loops to print the value of *i* multiple times in each row, where the number of times it prints *i* increases with each iteration of the outer loop. The `print()` function prints the value of *i* and moves to the next line after each row.

## Loop Control Statements

Loop control statements in Python are special statements that help control the execution of loops (for or while). They let you modify the default behavior of the loop, such as stopping it early, skipping an iteration, or doing nothing temporarily.

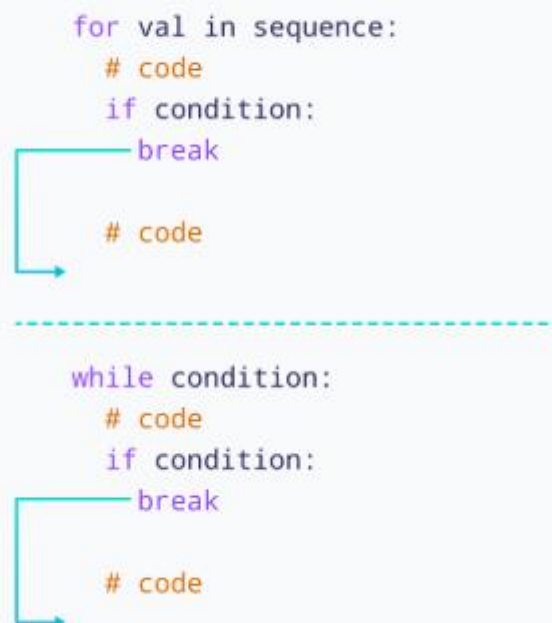
. Python supports the following control statements:

- Break statement
- Continue statement

### Break Statement in Python

The break statement in Python is used to exit or “break” out of a loop (either a for or while loop) prematurely, before the loop has iterated through all its items or reached its condition. When the break statement is executed, the program immediately exits the loop, and the control moves to the next line of code after the loop.

## Working of Python break Statement



Example:

```
for i in range(5):  
    if i == 3:  
        break  
    print(i)
```

## Output

```
0  
1  
2
```

### Continue Statement in Python

Python Continue statement is a loop control statement that forces to execute the next iteration of the loop while skipping the rest of the code inside the loop for the current iteration only, i.e. when the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped for the current iteration and the next iteration of the loop will begin.

### Working of continue Statement in Python

```
→ for val in sequence:  
    # code  
    if condition:  
        continue
```

```
    # code
```

---

```
→ while condition:  
    # code  
    if condition:  
        continue
```

```
    # code
```

Example:

```
for i in range(5):  
    if i == 3:  
        continue  
    print(i)
```

## Output

```
0  
1  
2  
4
```

### for each:

Python does not have a separate "foreach" keyword or construct like some other programming languages. Instead, Python's standard for loop functions as a "foreach" loop, designed to iterate directly over the elements of an iterable object.

### How it works:

The for loop in Python iterates through each item in a sequence (such as a list, tuple, string) or any other iterable object, executing a block of code for each item.

### Syntax:

```
for item in iterable:  
    # Code to be executed for each item
```

### Explanation:

- **item:** This is a variable that takes on the value of each element in the iterable during each iteration of the loop.
- **in:** This keyword specifies that the loop will iterate over the elements of the iterable.
- **iterable:** This can be any object that can be iterated over, such as a list, tuple, string, dictionary, or a custom iterable object.

### Example with a list:

Python

```
my_list = ["apple", "banana", "cherry"]

for fruit in my_list:
    print(fruit)
```

### Output:

Code

```
apple
banana
cherry
```

Python

```
my_list = ["apple", "banana", "cherry"]

for fruit in my_list:
    print(fruit)
```

### Output:

Code

```
apple
banana
cherry
```

## Example with a string:

Python

```
my_string = "Python"

for char in my_string:
    print(char)
```

## Output:

Code

```
p
y
t
h
o
n
```

## Accessing index and value (using `enumerate`):

If you need to access both the index and the value of each element during iteration, you can use the built-in `enumerate()` function.

Python



```
my_list = ["apple", "banana", "cherry"]

for index, fruit in enumerate(my_list):
    print(f"Index: {index}, Fruit: {fruit}")
```

## Output:

Code



```
Index: 0, Fruit: apple
Index: 1, Fruit: banana
Index: 2, Fruit: cherry
```

## Else Statement with For Loops:

Python also allows us to use the [else condition for loops](#). The else block just after for/while is executed only when the loop is NOT terminated by a break statement.

```
for i in range(1, 4):
    print(i)
else: # Executed because no break in for
    print("No Break\n")
```

## Output

```
1
2
3
No Break
```

## while loop with else:

As discussed above, while loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed. The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won't be executed.

**Note:** The else block just after for/while is executed only when the loop is NOT terminated by a break statement.

```
i = 0
while i < 4:
    i += 1
    print(i)
else: # Executed because no break in for
    print("No Break\n")
```

```
i = 0
while i < 4:
    i += 1
    print(i)
    break
else: # Not executed as there is a break
    print("No Break")
```

---