**Unit-III**

**Conditional Statement**

**Conditional Statements: -**

Conditional structures of a programming language perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false. In Python, conditional statements are handled by if statements.

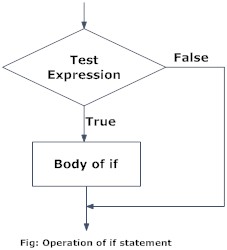
**if statement:-**

We will start with the if statement, which will evaluate whether a statement is true or false, and run code only in the case that the statement is true.

**Syntax**

if expression:

statement(s)



Example

grade = 70

**if** grade >= 65:

print("Passing grade")

**Output:-**

Passing code

**if else statement:-**

An else statement can be combined with an if statement.

An else statement contains the block of code that executes if the conditional

expression in the if statement resolves to 0 or a false value.

The else statement is an optional statement and there could be at most only one

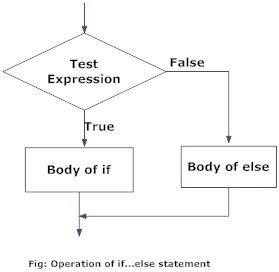
**Syntax:-**

if expression:

statement(s)

else:

statement(s)

****

**Example:-**

grade = 60

**if** grade >= 65:

print("Passing grade")

**else**:

print("Failing grade")

**Output:-**

failing grade

**Example:-**

balance = 522

**if** balance < 0:

print("Balance is below 0, add funds now or you will be charged a penalty.")

**else**:

print("Your balance is 0 or above.")

**Output :-**

Your balance is 0 or above.

**Nested if statements :-**

We can have a if...elif...else statement inside another if...elif...elsestatement. This is called nesting in computer programming.Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Syntax

The syntax of the nested *if...elif...else* construct may be −

if expression1:

statement(s)

if expression2:

statement(s)

elif expression3:

statement(s)

else:

statement(s)

elif expression4:

statement(s)

else:

statement(s)

**Example**

#!/usr/bin/python

var = 100

if var < 200:

print "Expression value is less than 200"

if var == 150:

print "Which is 150"

elif var == 100:

print "Which is 100"

elif var == 50:

print "Which is 50"

elif var < 50:

print "Expression value is less than 50"

else:

print "Could not find true expression"

print "Good bye!"

**Example:-**

num = float(input("Enter a number: "))

if num >= 0:

if num == 0:

print("Zero")

else:

print("Positive number")

else:

print("Negative number")

**Output 1**

Enter a number: 5

Positive number

**Output 2**

Enter a number: -1

Negative number

**Output 3**

Enter a number: 0

Zero

**Looping Statements:-**

statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times.

**Types of Loop:-**

|  |  |
| --- | --- |
| Sr.No. | Loop Type & Description |
| 1 | [while loop](https://www.tutorialspoint.com/python/python_while_loop.htm)  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](https://www.tutorialspoint.com/python/python_for_loop.htm)  Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| 3 | [nested loops](https://www.tutorialspoint.com/python/python_nested_loops.htm)  You can use one or more loop inside any another while, for or do..while loop. |

**while Loop :-**

Loops are used in programming to repeat a specific block of code. In this article, you will learn to create a while loop in Python.

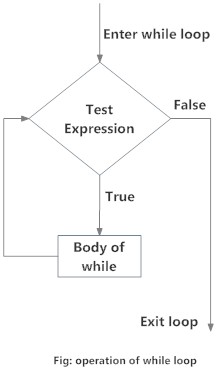
The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

## Syntax :-

while test\_expression:

Body of while

**Flowchart of while Loop**



**Example:-**

n = 10

sum = 0

i = 1

while i <= n:

sum = sum + i

i = i+1

print("The sum is", sum)

**Output:-**

Enter n: 10

The sum is 55

**2)** [**for loop**](https://www.tutorialspoint.com/python/python_for_loop.htm) **:-**

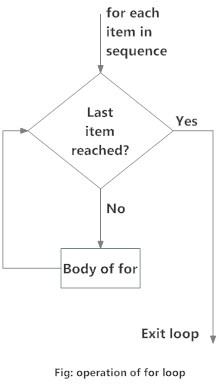
The for loop in Python is used to iterate over a sequence ([list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [string](https://www.programiz.com/python-programming/string)) or other iterable objects. Iterating over a sequence is called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

**Syntax:-**

for val in sequence:

Body of for

**Flowchart :-**

****

**Example:-**

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

# variable to store the sum

sum = 0

# iterate over the list

for val in numbers:

sum = sum+val

# Output: The sum is 48

print("The sum is", sum)

**Output:-**

The sum is 48

**3)** [**nested loops**](https://www.tutorialspoint.com/python/python_nested_loops.htm) **:-**

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

**Syntax :-**

for iterating\_var in sequence:

for iterating\_var in sequence:

statements(s)

statements(s)

**The syntax for a nested while loop statement**

while expression:

while expression:

statement(s)

statement(s)

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

**Example**

i = 2

while(i < 100):

j = 2

while(j <= (i/j)):

if not(i%j): break

j = j + 1

if (j > i/j) : print i, " is prime"

i = i + 1

print "Good bye!"

**Example:-**

count = 1

for i in range(10):

for j in range(0, i):

print(count, end='')

count = count +1

print()

input()

**Loop manipulation:-**

You might face a situation in which you need to exit a loop completely when an external condition is triggered or there may also be a situation when you want to skip a part of the loop and start next execution.

Python provides break and continue statements to handle such situations and to have good control on your loop.

***break, continue* and *pass* statements available in Python.**

**The *break* Statement:**

The break statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both *while* and *for* loops.

**Example:**

for letter in 'Python':

if letter == 'h':

break

print 'Current Letter :', letter

**Outout:-**

Current Letter : P

Current Letter : y

Current Letter : t

**Example:-**

var = 10

while var > 0:

print 'Current variable value :', var

var = var -1

if var == 5:

break

print "Good bye!"

**Outout:-**

Current variable value : 10

Current variable value : 9

Current variable value : 8

Current variable value : 7

Current variable value : 6

Good bye!

**The *continue* Statement:**

The continue statement in Python returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

The continue statement can be used in both *while* and *for* loops.

**Example:**

for letter in 'Python':

if letter == 'h':

continue

print 'Current Letter :', letter

**Output:-**

Current Letter : P

Current Letter : y

Current Letter : t

Current Letter : o

Current Letter : n

**Example:-**

var = 10

while var > 0:

var = var -1

if var == 5:

continue

print 'Current variable value :', var

print "Good bye!"

**Output:-**

Current variable value : 10

Current variable value : 9

Current variable value : 8

Current variable value : 7

Current variable value : 6

Current variable value : 4

Current variable value : 3

Current variable value : 2

Current variable value : 1

Good bye!

**The *else* Statement Used with Loops**

Python supports to have an else statement associated with a loop statements.

* If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.
* If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

**Example:**

#!/usr/bin/python

for num in range(10,20): #to iterate between 10 to 20

for i in range(2,num): #to iterate on the factors of the number

if num%i == 0: #to determine the first factor

j=num/i #to calculate the second factor

print '%d equals %d \* %d' % (num,i,j)

break #to move to the next number, the #first FOR

else: # else part of the loop

print num, 'is a prime number'

**Output:-**

10 equals 2 \* 5

11 is a prime number

12 equals 2 \* 6

13 is a prime number

14 equals 2 \* 7

15 equals 3 \* 5

16 equals 2 \* 8

17 is a prime number

18 equals 2 \* 9

19 is a prime number

Similar way you can use else statement with while loop.

**The *pass* Statement:**

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

The pass statement is a *null* operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

Example:

#!/usr/bin/python

for letter in 'Python':

if letter == 'h':

pass

print 'hiiiiiiiiiiiiiiii'

print 'Current Letter :', letter

print "Good bye!"

**Output:-**

Current Letter : P

Current Letter : y

Current Letter : t

This is pass block

Current Letter : h

Current Letter : o

Current Letter : n

Good bye!

**assert :-**

An assertion is a sanity-check that you can turn on or turn off when you are done with your testing of the program.

The easiest way to think of an assertion is to liken it to a raise-ifstatement (or to be more accurate, a raise-if-not statement). An expression is tested, and if the result comes up false, an exception is raised.

Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.

## *assert* Statement

When it encounters an assert statement, Python evaluates the accompanying expression, which is hopefully true. If the expression is false, Python raises an *AssertionError exception*.

**Syntax**

assert Expression[, Arguments]

If the assertion fails, Python uses ArgumentExpression as the argument for the AssertionError. AssertionError exceptions can be caught and handled like any other exception using the try-except statement, but if not handled, they will terminate the program and produce a traceback.

**Exampl**e:-

def avg(marks):

assert len(marks) != 0

return sum(marks)/len(marks)

mark1 = []

print("Average of mark1:",avg(mark1))e:-

**Output:-**

Assertion error

**Exampl**e:-

def avg(marks):

assert len(marks) != 0,"List is empty."

return sum(marks)/len(marks)

mark2 = [55,88,78,90,79]

print("Average of mark2:",avg(mark2))

mark1 = []

print("Average of mark1:",avg(mark1))

**Output:-**

Average of mark2: 78.0

AssertionError: List is empty.