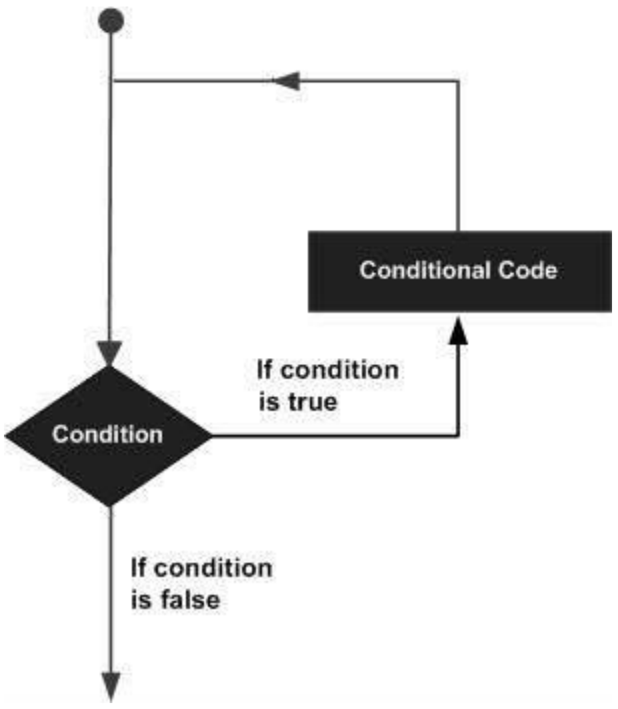
# Python - Loops

In general, 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. The following diagram illustrates a loop statement −



Python programming language provides following types of loops to handle looping requirements.

| Sr.No. | Loop Type & Description |
|---|---|
| 1 | while loop<br><br>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<br><br>Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |

|   |   |
|---|---|
| 3 | nested loops<br>You can use one or more loop inside any another while, for or do..while loop. |

## For Loop

The for loop in Python is used to iterate over a sequence (list, tuple, 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**: Python for Loop

```
# Program to find the sum of all numbers stored in a list

# List of numbers
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

print("The sum is", sum)
```

When you run the program, the output will be:

```
The sum is 48
```

## for loop with else

A `for` loop can have an optional `else` block as well. The `else` part is executed if the items in the sequence used in for loop exhausts.
The `break` keyword can be used to stop a for loop. In such cases, the else part is ignored.
Hence, a for loop's else part runs if no break occurs.

Here is an example to illustrate this.

```
digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

When you run the program, the output will be:

```
0
1
5
No items left.
```

Here, the for loop prints items of the list until the loop exhausts. When the for loop exhausts, it executes the block of code in the `else` and prints `No items left.`
This `for...else` statement can be used with the `break` keyword to run the `else` block only when the `break` keyword was not executed. Let's take an example:

```
# program to display student's marks from record
student_name = 'Soyuj'

marks = {'James': 90, 'Jules': 55, 'Arthur': 77}

for student in marks:
    if student == student_name:
        print(marks[student])
        break
else:
    print('No entry with that name found.')
```

**Output**

```
No entry with that name found.
```

## While Loop

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

We generally use this loop when we don't know the number of times to iterate beforehand.

### Syntax of while Loop in Python

```
while test_expression:

    Body of while
```

In the while loop, test expression is checked first. The body of the loop is entered only if the `test_expression` evaluates to `True`. After one iteration, the test expression is checked again. This process continues until the `test_expression` evaluates to `False`.
In Python, the body of the while loop is determined through indentation.

The body starts with indentation and the first unindented line marks the end.

Python interprets any non-zero value as `True`. `None` and `0` are interpreted as `False`.

## **Example**: Python while Loop

```
# Program to add natural
# numbers up to
# sum = 1+2+3+...+n

# To take input from the user,
# n = int(input("Enter n: "))

n = 10

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

When you run the program, the output will be:

```
Enter n: 10
The sum is 55
```

In the above program, the test expression will be `True` as long as our counter variable `i` is less than or equal to `n` (10 in our program).
We need to increase the value of the counter variable in the body of the loop. This is very important (and mostly forgotten). Failing to do so will result in an infinite loop (never-ending loop).

## While loop with else

Same as with for loops, while loops can also have an optional `else` block.
The `else` part is executed if the condition in the while loop evaluates to `False`.
The while loop can be terminated with a break statement. In such cases, the `else` part is ignored. Hence, a while loop's `else` part runs if no break occurs and the condition is false.
Here is an example to illustrate this.

```
'''Example to illustrate
the use of else statement
with the while loop'''

counter = 0

while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
    print("Inside else")
```

### Output

```
Inside loop
Inside loop
Inside loop
Inside else
```