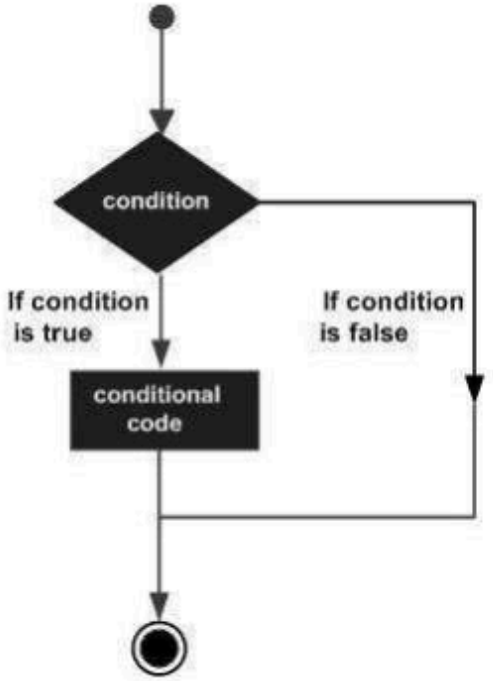
7. Python 3 – Decision Making

Decision-making is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions.

Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outcome. You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages-



Python programming language assumes any **non-zero** and **non-null** values as TRUE, and any **zero** or **null values** as FALSE value.

Python programming language provides the following types of decision-making statements.

Statement	Description
if statements	An if statement consists of a Boolean expression followed by one or more statements.
if...else statements	An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.

nested if statements	You can use one if or else if statement inside another if or else if statement(s).

Let us go through each decision-making statement quickly.

IF Statement

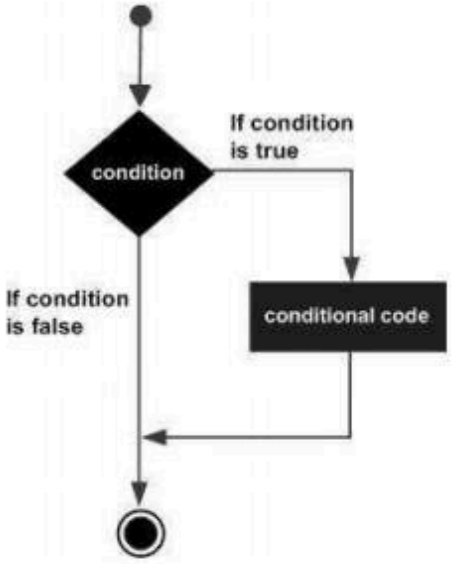
The IF statement is similar to that of other languages. The **if** statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison.

Syntax

```
if expression:
   statement(s)
```

If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed. In Python, statements in a block are uniformly indented after the : symbol. If boolean expression evaluates to FALSE, then the first set of code after the end of block is executed.

Flow Diagram



Example

```
#!/usr/bin/python3
var1 = 100
if var1:
   print ("1 - Got a true expression value")
   print (var1)
```

```
var2 = 0
if var2:
   print ("2 - Got a true expression value")
   print (var2)
print ("Good bye!")
```

When the above code is executed, it produces the following result –

```
1 - Got a true expression value
100
Good bye!
```

IF...ELIF...ELSE Statements

An **else** statement can be combined with an **if** statement. An **else** statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

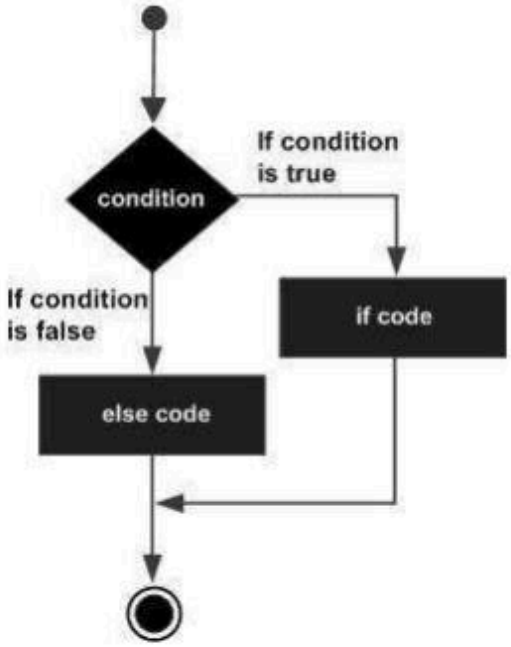
The else statement is an optional statement and there could be at the most only one **else** statement following **if**.

Syntax

The syntax of the **if...else** statement is-

```
if expression:
   statement(s)
else:
   statement(s)
```

Flow Diagram



Example

```
#!/usr/bin/python3
amount=int(input("Enter amount: "))
if amount<1000:
    discount=amount*0.05
    print ("Discount",discount)
else:
    discount=amount*0.10
    print ("Discount",discount)

print ("Net payable:",amount-discount)
```

In the above example, discount is calculated on the input amount. Rate of discount is 5%, if the amount is less than 1000, and 10% if it is above 10000. When the above code is executed, it produces the following result-

```
Enter amount: 600
Discount 30.0
Net payable: 570.0
Enter amount: 1200
Discount 120.0
```

```
Net payable: 1080.0
```

The elif Statement

The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

Similar to the **else**, the **elif** statement is optional. However, unlike **else**, for which there can be at the most one statement, there can be an arbitrary number of **elif** statements following an **if**.

Syntax

```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
```

Core Python does not provide switch or case statements as in other languages, but we can use if..elif...statements to simulate switch case as follows-

Example

```
#!/usr/bin/python3
amount=int(input("Enter amount: "))

if amount<1000:
    discount=amount*0.05
    print ("Discount",discount)
elif amount<5000:
    discount=amount*0.10
    print ("Discount",discount)
else:
    discount=amount*0.15
    print ("Discount",discount)
print ("Net payable:",amount-discount)
```

When the above code is executed, it produces the following result-

```
Enter amount: 600
Discount 30.0
Net payable: 570.0

Enter amount: 3000
Discount 300.0
Net payable: 2700.0

Enter amount: 6000
Discount 900.0
Net payable: 5100.0
```

Nested IF Statements

There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested **if** construct.

In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

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/python3
num=int(input("enter number"))
```

```
if num%2==0:
    if num%3==0:
        print ("Divisible by 3 and 2")
    else:
        print ("divisible by 2 not divisible by 3")
else:
    if num%3==0:
        print ("divisible by 3 not divisible by 2")
    else:
        print  ("not Divisible by 2 not divisible by 3")
```

When the above code is executed, it produces the following result-

```
enter number8
divisible by 2 not divisible by 3

enter number15
divisible by 3 not divisible by 2

enter number12
Divisible by 3 and 2

enter number5
not Divisible by 2 not divisible by 3
```

Single Statement Suites

If the suite of an **if** clause consists only of a single line, it may go on the same line as the header statement.

Here is an example of a **one-line if** clause-

```
#!/usr/bin/python3
var = 100
if ( var  == 100 ) : print ("Value of expression is 100")
print ("Good bye!")
```

When the above code is executed, it produces the following result-

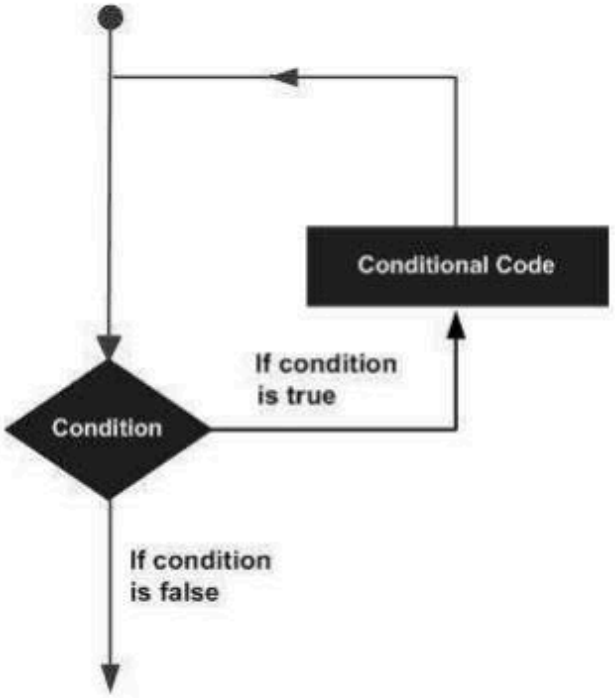
```
Value of expression is 100
Good bye!
```

8. Python 3 – 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 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 the following types of loops to handle looping requirements.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

nested loops	You can use one or more loop inside any another while, or for loop.

while Loop Statements

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a **while** loop in Python programming language is-

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

Here, **statement(s)** may be a single statement or a block of statements with uniform indent. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

In Python, 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.

Flow Diagram