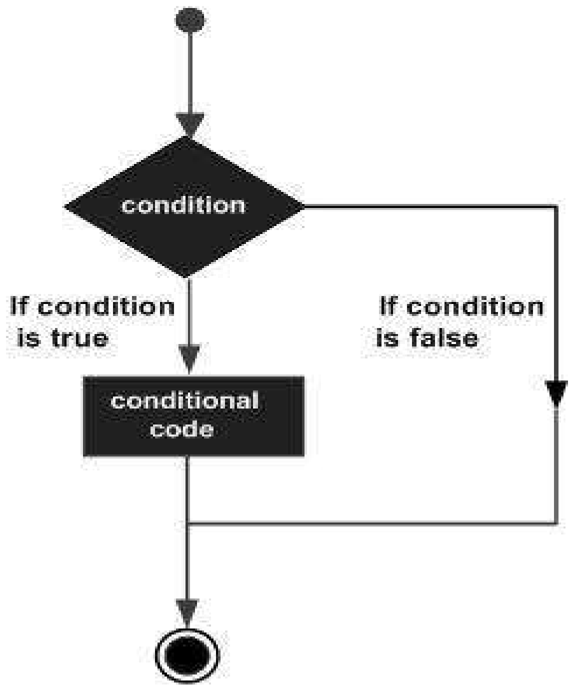
Conditional Statement

Conditional expressions, involving keywords such as **if, elif,** and **else**, provide Python programs with the ability to perform different actions depending on a boolean condition: **True** or **False**. This section covers the use of Python conditionals, boolean logic, and ternary statements.



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

The general form of a typical decision making structure is shown in figure, 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**.

Python programming language provides the following 3 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).

Truth Values

The following values are considered falsey, in that they evaluate to False when applied to a boolean operator.

- None
- False
- 0 , or any numerical value equivalent to zero, for example 0L , 0.0 , 0j
- Empty sequences: '' , "" , () , []
- Empty mappings: {}
- User-defined types where the __bool__ or __len__ methods return 0 or False

All other values in Python evaluate to True .

Boolean Logic Expressions

Boolean logic expressions, in addition to evaluating to True or False , return the value that was interpreted as True or False .

And operator

The and operator evaluates all expressions and returns the last expression if all expressions evaluate to True . Otherwise it returns the first value that evaluates to False :

```
>>> 1 and 2
2
>>> 1 and 0
0
>>> 1 and "Hello World"
"Hello World"
>>> "" and "Pancakes"
""
```

Or operator

The or operator evaluates the expressions left to right and returns the first value that evaluates to True or the last value (if none are True).

```
>>> 1 or 2
1
>>> None or 1
1
>>> 0 or []
[ ]
```

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.

Example

Code:

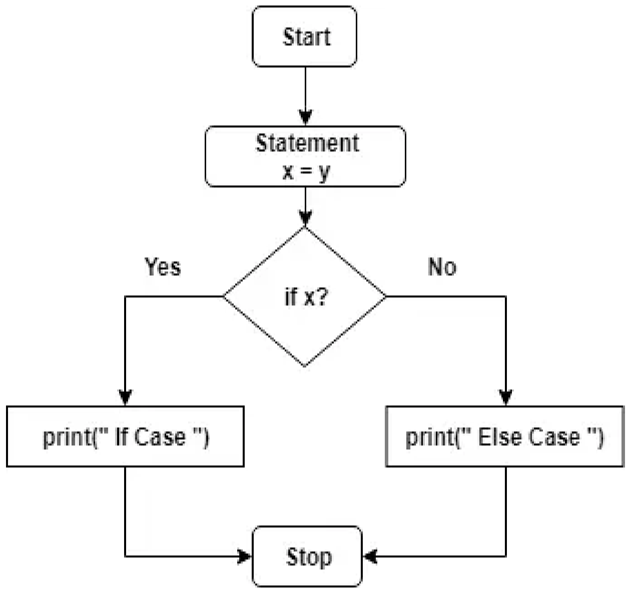
```
var1 = 100
if var1:
    print ("1 - Got a true expression value")
    print (var1)
var2 = 0
if var2:
    print ("2 - Got a true expression value")
    print (var2)
print ("Good bye!")
```

Output

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

IF...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)
```

Example

Code:	**Output**:

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)
	Enter amount: 600
	Discount 30.0
	Net payable: 570.0
	
	Enter amount: 1200
	Discount 120.0
	Net payable: 1080.0

Testing for multiple conditions

A common mistake when checking for multiple conditions is to apply the logic incorrectly. This example is trying to check if two variables are each greater than 2. >>>

```
a = 1
b = 6
if a and b > 2:
        print('yes')
else:
       print('no')
```

Output will be yes. The statement [**if a and b>2**] is evaluated as - **if (a) and (b > 2)** . This produces an unexpected result because bool(a) i.e. bool(1) evaluates as True when a is not zero.

Each variable needs to be compared separately.

```
 if a > 2 and b > 2:
        print('yes')
else:
        print('no')
```

Now the **output will be "no"**

Another, similar, mistake is made when checking if a variable is one of multiple values. The statement in this example is evaluated as - if (a == 3) or (4) or (6) . This produces an unexpected result because bool(4) and bool(6) each evaluate to True

```
 a = 1
if a == 3 or 4 or 6:
        print('yes')
else:
        print('no')
```

Here the **Output will be "yes"**

Again each comparison must be made separately

```
if a == 3 or a == 4 or a == 6:
        print('yes')
else:
        print('no')
```

Now the **output will be "no"**

Using the in operator is the canonical way to write this.

```
if a in (3, 4, 6):
        print('yes')
else:
        print('no')
```

In this case **output will be "no"**

Nested IF Statements

There may be a situation when we want to check for another condition after a condition resolves to true or false. In such a situation, we can use the nested if construct.
In a nested if construct, we can have an **if...elif...else** construct inside another **if...elif...else** construct.

Syntax

The syntax of one 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

Code:

```
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")
```

Output:

```
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
```

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...else** statements to simulate switch case as follows-

Example

Code:

```
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)
```

Output:

```
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
```

Single Statement Suites

If the number of statements present in 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!
```

```
Statement_when_value_true if condition else
statement_value_when_false
```