

Functions

We need to execute different sections of code under different circumstances. For example, in a billing situation a credit card may attract additional charges or a bill amount higher than a fixed amount may get discount. In order to check different conditions and execute different code we need conditional execution. This is almost always achieved using `if` in all programming languages. Python is no different.

Booleans: Overview

A boolean expression is an expression that is either true or false. One way to write the boolean expression is to use the operator `==`, which compares two values and produces a boolean value:

```
>>> 6 == 6
True
>>> 5 == 6
False
```

In the first statement, the two operands are equal, so the value of the expression is `True`, in the second statement, 5 is not equal to 6, so we get `False`. `True` and `False` are special values that are built into Python.

The `==` operator is one of the comparison operators; the others are:

Expression	Description
<code>x != y</code>	x is not equal to y
<code>x > y</code>	x is greater than y
<code>x < y</code>	x is less than y
<code>x >= y</code>	x is greater than or equal to y
<code>x <= y</code>	x is less than or equal to y

Although these operations are probably familiar to you, the Python symbols are different from the mathematical symbols. A common error is to use a single equal sign(=) instead of a double equal sign(==). Remember that `=` is an assignment operator and `==` is a comparison operator.

Simple if statement

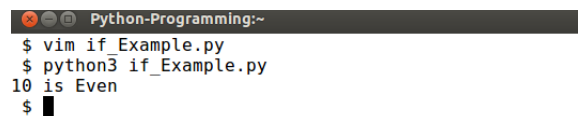
The simplest form of `if` statement is one where a piece of code is executed if a condition exists otherwise nothing is done. For example:

```
if x > 0:
    print("x is positive")
```

The boolean expression after the `if` statement is called the condition. If it is true, then the indented statement gets executed, If not, nothing happens.

Example Program to check whether the number is even or odd.

```
1 n = 10
2 if n % 2 == 0:
3     print(n, " is Even")
4 if n % 2 != 0:
5     print(n, " is Odd")
```

A terminal window titled 'Python-Programming:~' shows the following commands and output: '\$ vim if_Example.py', '\$ python3 if_Example.py', and the output '10 is Even' followed by a prompt '\$ ' and a cursor.

```
Python-Programming:~
$ vim if_Example.py
$ python3 if_Example.py
10 is Even
$
```

Figure 1: If Example

Blocks in Python

Like other compound statements, the `if` statement is made up of a header and a block of statements:

HEADER:

```
FIRST STATEMENT
.....
.....
LAST STATEMENT
```

The header begins on a new line and end with a colon(:). the indented statements that follow are called block. The first unindented statement marks the end of the block. A statement block inside a compound statement is called the body of the statement.

if ... else

This is another form of if statement, in which there are two possibilities and the condition determines which one gets executed. Contrast this to the simple if, where if the condition is not true nothing is done. We look at the program to understand how if ... else works:

```
if x % 2 == 0:
    print(x, "is Even")
else:
    print(x, "is Odd")
```

If the remainder of the condition is "Zero", then we know that x is Even, and the program displays a message. If the condition is false, the second set of statements is executed. Since the condition must be true or false, exactly one of the alternative will be executed. The alternatives are called branches, because they are branches in the flow of execution.

Example Accept marks and check whether the student is passed or failed in a test.

```
1 marks = 76
2 if marks >= 45:
3     print(" Passed")
4     print(" Congratulations!!")
5 else:
6     print(" Failed")
7     print(" Good luck in the retest")
```

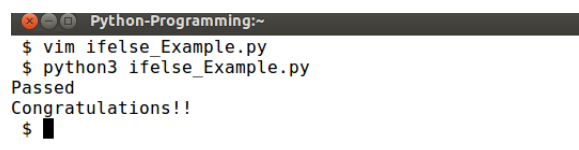


Figure 2: If Example

Nested if

One if statement within another if, such as a conditional statement inside a branch of another conditional statement, is termed as nested if. We look at the program to understand how nested if works:

```
# Check whether the number is even or odd if it is positive.
x = 56
if x >= 0: # Outer if
    if x % 2 == 0: # Inner if
        print(x, "is Even Number")
    else:
        print(x, "is Odd Number")
else:
    print(x, "is Negative")
```

The outer condition is evaluated first, as the condition is true it gets into the block and evaluates an inner condition and it prints "x is Even Number" as the condition is true. If it fails it executes the else block of statement and prints "x is an Odd Number". If the outer condition itself evaluates to false then it executes the else part of it and it prints "x The Number is Negative".

Compound Conditions

Compound conditions are formed by combining multiple conditions in a statement. We use logical operators to form compound conditions.

Example 01 Given three sides of a triangle find out, if it is equilateral.

```
if a == b and b == c:
    print("Equilateral triangle")
else:
    print("Not an equilateral triangle")
```

We used logical operator and in the above example to combine two conditions. If the value of three variables a, b and c are equal then the condition evaluates to true and prints "Equilateral Triangle".

Example 02 Given three angles of a triangle and check whether it is right-angled triangle.

```
if x == 90 or y == 90 or z == 90:
    print("Right Angled Triangle")
else:
    print("Not a Right Angled Triangle")
```

We use logical operator `or` in the above example to combine multiple conditions. If any of the value is equal to 90 then the condition evaluates to true and prints "Right Angled Triangle".

As we know that logical operators often provide a way to simplify nested conditional statements. For Example, we can rewrite the following code using a single condition:

```
1 if 0 < x:
2     if x < 10:
3         print "x is a positive single digit."
```

The `print()` statement is executed only if we make it past both the conditionals, so we can use the `and` operator:

```
1 if 0 < x and x < 10:
2     print "x is a positive single digit."
```

These type of conditions are commonly used, so Python provides an alternative syntax that is similar to mathematical notation:

```
1 if 0 < x < 10:
2     print "x is a positive single digit."
```

This condition is semantically the same as the compound condition and the nested condition.

if ...elif ...else

Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is a series of conditions. In such cases we use `if ...elif ...else`.

`elif` is an abbreviation of "else if". The conditions are checked one after another. The statement associated with the first true condition is executed. There is no limit of the number of `elif` statements, but the last branch has to be an `else` statement:

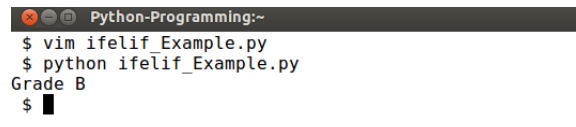
Functions

```
if x < y: # Condition 1
    print(x, "is less than", y)
elif x > y: # Condition 2
    print(x, "is greater than", y)
else:
    print(x, "and", y, "are equal")
```

In the above example, Condition 1 is evaluated first, if it is true it executes the associated block of statements. If it fails Condition 2 is evaluated and if it is true it executes the associated block of statements otherwise it executes the else block of statements.

Example Find the grade of a student based on a marks.

```
1 marks = 75
2 if marks >= 80:
3     print("Grade A")
4 elif marks >= 60:
5     print("Grade B")
6 elif marks >= 40:
7     print("Grade C")
8 else:
9     print("Failed")
```



```
Python-Programming:~
$ vim ifelif_Example.py
$ python ifelif_Example.py
Grade B
$
```

Figure 3: Ifelif Example