python

# UNIT – I

**Introduction to Python**: History

- Features
- Setting up path
- Working with Python Basic Syntax
- Variable and Data Types
- Operators
- Conditional Statements(If ,If- else ,Nested if-else)
- Looping( for, While Nested loops)
- Control Statements(Break , Continue ,Pass)

**Input-Output:** Printing on screen, Reading data from keyboard, Opening and closing file

## Introduction to Python: History

Guido Van Rossum

## Introduction to Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. It is designed to be highly readable.

- ***Python is Interpreted:*** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it .
- ***Python is Interactive:*** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- ***Python is Object-Oriented:*** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- ***Python is a Beginner's Language:*** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

## History

➢ Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

➢ Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, and other scripting languages.

➢ Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

➢ python first version is 1.0 and latest version is 3.7

➢ The logo of python shows two intertwined snakes

➢ Major implementations of Python are Cpython, IronPython, Jpython,MicroPython,PyPy

➢Top software companies like Google , Microsoft ,IBM , Yahoo using python

## Features

- Simple & Easy to Learn
- Free and Open Source
- High Level Language
- Platform Independent Language
- Dynamically Typed Language
- Procedure oriented and Object oriented
- Interpreted
- Portable, Extendable
- Embedded
- Extensive Standard Library
- Databases and GUI Programming
- Automatic Garbage Collection

## *A simple language which is easier to learn*

Python has a very simple and elegant syntax. It's much easier to read and write when compared to other languages like: C++, Java, C#.

## *Free and open-source*

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software's written in it, you can even make changes to the Python's source code.

## *High level language*

High level languages use English words to develop programs. These are easy to learn and use. Like  C, Java and PHP, Python  also uses English words in its program

## *Platform Independent*

It is platform independent programming language, its code easily run on any platform such as Windows, Linux, Unix , Macintosh etc. A Python program written on a Macintosh computer will run on a Linux system and vice versa

## *Dynamically typed language*

In python there is no need to declare the type of a variable.Whenever we assign a value to the variable, based on the value the type will be allocated automatically.

## *Object-oriented*

A programming language that can model the real world is said to be object-oriented. Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem very easily With OOP, you are able to divide these complex problems into smaller sets by creating objects.

## *A high-level, interpreted language*

interpreter executes the code line by line at a time. Unlike C/C++, you don't have to worry about difficult tasks like memory management, garbage collection and so on. Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

## *Portability*

You can move Python programs from one platform to another, and run it without any changes. It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

## *Extensible*

Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code. This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

## *Embeddable*

We can embed the python code into other languages such as C, C++, Java, Etc..To provide the scripting capabilities to other languages we can use the python code in it.

## *Extensive Standard Library*

Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself. Developers can use the built-in libraries for their applications. By making use of these built-in libraries, development will become faster.

## *Databases and GUI Programming*

Python provides interfaces to connect its programs to all major databases like Oracle, Sybase or Mysql. We can develop GUI based applications and Web applications using Python Language.

## Applications of Python

- Automation Applications
- Data Analytics & Visualization
- Scientific Applications
- Web Applications
- Web Scrapping
- Administration Script
- Networking with IOT Applications
- Test Cases
- GUI Applications
- Gaming Applications
- Animation Applications

# Python is simple

```
print "Hello World!"
```

Python

```
#include <iostream.h>
int main()
{
cout << "Hello World!";
}
```

C++

```
public class helloWorld
{
      public static void main(String [] args)
      {
      System.out.println("Hello World!");
      }
}
```

Java

## Installation and Setting up path

To install Python on a Windows machine, follow these steps:

**1.** Open a web browser and go to http://www.python.org.

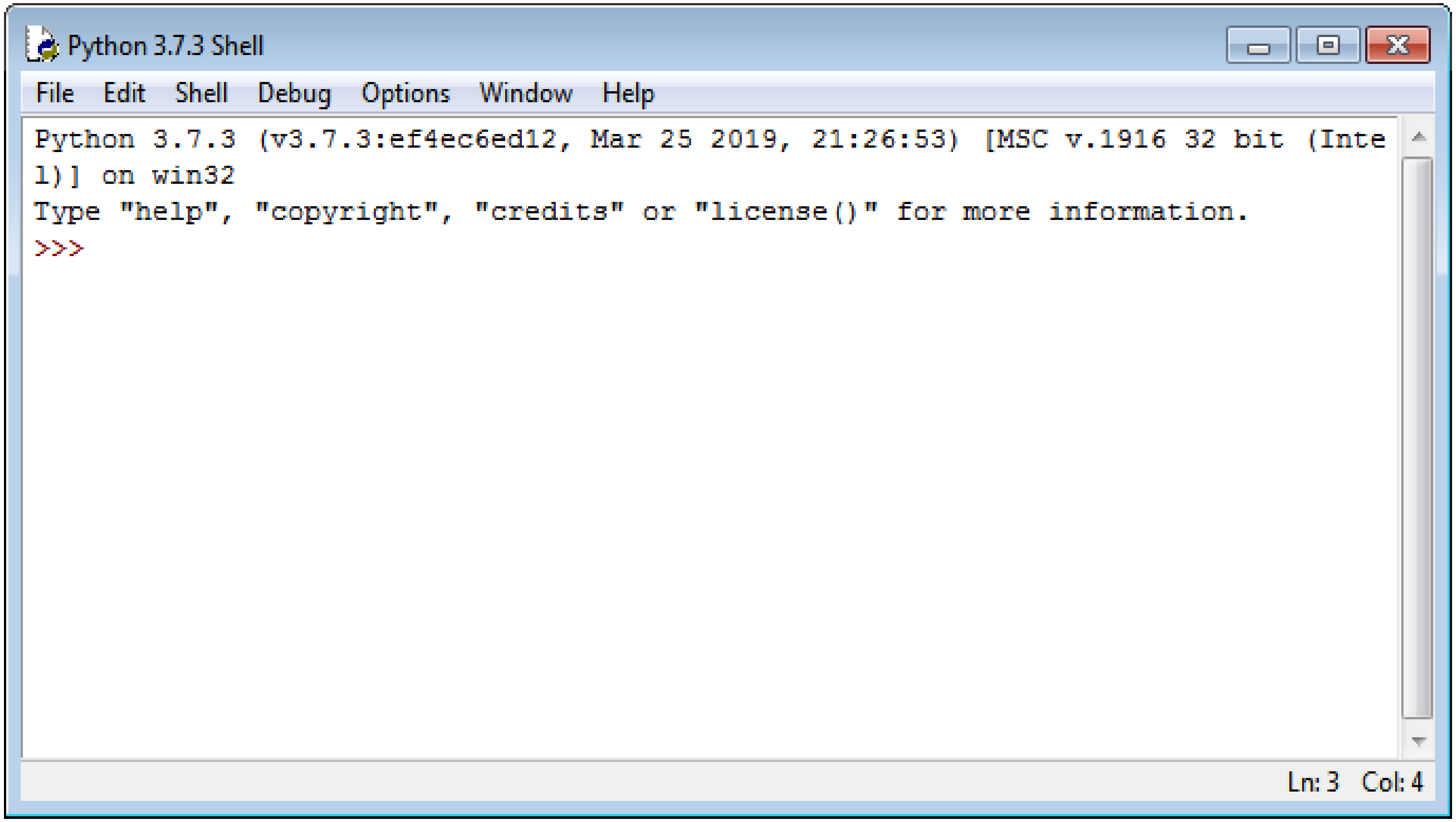
**2.** Click the Downloads.

**3.** Click the python 3.7.3 link to download the installer file.

**4.** Store the Windows Installer file somewhere on your computer, such as C:\download\ python 3.7.3.exe (Just create a directory where you can find it later.)

**5.** Run the downloaded file by double-clicking it in Windows Explorer. Just accept the default settings, wait until the installation is finished, and you’re ready to roll!

▪ Run the Python Integrated Development Environment (IDLE) by selecting Start ➤Programs ➤ Python3 ➤ IDLE (Python GUI).



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Inte
l)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

This menu option will probably include your version number, as in Python 3.7.3

## Path Setting

Right click on my computer → click on advanced system settings →click on Environmental variables → go to System variables click on path →click on edit →type the installation path of python software in the beginning of the variable ( C:\python 35-32;)

## Working with Python Basic Syntax

There are two ways to use the Python interpreter:

***1. Interactive Mode / shell mode (line by line execution)***

In shell mode, you type Python expressions into the Python shell, and the interpreter immediately shows the result.

```
>>> 2 + 3
5
 >>>
```

The >>> is called the **Python prompt.**

***2. Batch Mode (entire program execution at a time)***

The concept of writing **group** of python statements into a file, saving that file with the extension ***'.py'*** and submitting the entire file to the python interpreter at a time is known as Batch mode.

- In order to develop the python files we use either editors or IDE's.
- Different editors are notepad, notepad++, edit plus, vi, nano, gedit----
- Different IDE's are pycharm, eric, netbeans, eclipse
- Open the notepad and write the following statements.

```
X=100
Y=20
print(x+y)
print(x-y)
print(x*y)
print(x/y)
```

Save the above file with demo.py in C:\foldername

Open the command prompt and execute the following commands.

```
C:\foldername >python demo.py
120
80
2000
5.0
```

## Keywords in Python

Keywords are the reserved words in Python. We can not use a keyword as variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive. There are 33 keywords in Python. All the keywords except True, False and None are in lower case and they must be written as it is. The list of all the keywords are given below.

Keywords in Python programming language

| | | | | |
|---|---|---|---|---|
| False | class | finally | is | return |
| None | continue | for | lambda | try |
| True | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass | |
| break | except | in | raise | |

## Python Identifiers

- ➢ Identifier is the name given to entities like class, functions, variables etc. in Python.
- ➢ It helps differentiating one entity from another.

Rules for writing Identifiers:

- ➢ Identifiers can be a combination of letters in lowercase(atoz) or uppercase(AtoZ) or digits(0to9) or anunderscore(_).
- ➢An identifier cannot start with a digit. First character must be alphabetic character or underscore.
- ➢No special character is allowed except underscore
- ➢ Keywords cannot be used as identifiers.
- ➢Identifier can be of any length.
- ➢Case matters (that is, upper and lowercase letters). Thus, the names **count** and **Count** refer to two different identifiers

```
>>> global = 1
File "<interactive input>", line 1
global = 1
^
Syntax Error: invalid syntax
```

## Python Statement

- Instructions that a Python interpreter can execute are called statements.

For example, a = 1 is an assignment statement.

**Multi-line statement**

In Python, end of a statement is marked by a newline character. Statements in Python can be extended to one or more lines using parentheses (), braces {}, square brackets [], semi-colon (;), continuation character slash (\).

Example:

Declared using Continuation Character (\)

```
s = 1 + 2 + 3 + \
4 + 5 + 6 + \
7 + 8 + 9
```

Declared using parentheses ()

```
a = (1 + 2 + 3 +
4 + 5 + 6 +
7 + 8 + 9)
```

Declared using semicolons(;)

```
f = 2; r = 3; p = 4
```

Declared using square brackets []

```
footballer = ['MESSI',
             'NEYMAR',
             'SUAREZ']
```

## Python Comments

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. In Python, we use the hash(#) symbol to start writing a comment.

It extends up to the new line character. Comments are for programmers for better understanding of a program. Python Interpreter ignores  comment.

**Example**

```
#This is a comment
#print out Hello
print('Hello')
```

**Multi-line comments**

If we have comments that extend multiple lines, We can use triple quotes, either '''or """. These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well.

**Example**

```
"""This is also a
Perfect example of
multi-line comments"""
```

## Python Indentation

- Most of the programming languages like C, C++, Java use braces { } to define a block of code .
- But Python provides no braces{} to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation
- **Whitespace** is used for indentation in Python
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.
- A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block. If a block has to be more deeply nested, it is simply indented further to the right.

```
for i in range(1,11):
    print(i)
    if i == 5:
        break
```

## Python Variables

A variable is a location in memory used to store some data(value). The rules for writing a variable name is same as the rules for writing identifiers in Python. We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist.

We don't even have to declare the type of the variable. This is handled internally according to the type of value we assign to the variable.

### Variable assignment

We use the assignment operator (=) to assign values to a variable. Any type of value can be assigned to any valid variable.

Ex:

```
a = 5                          a=b=c=1
b = 3.2
c = "Hello“
```

### Changing the value of a variable

Ex:

```
a=4
print(a)
a=“hello”
print(a)
```

### Assigning multiple values to multiple variables

Ex:

```
name , rno , avg  = “Hello”, 15, 95.3
```

***Note***: Every Variable in Python is a Object

## Built-in function : type( )

It is used to identity the type of the object

***Ex:***

```
a=10
print(“Type of  a:”,type(a))
b=“Hello”
print(“Type of  b:”, type(b))
c=[ ]
print(“Type of c:”,type(c))
```

***Output:***

```
Type of a:<class ‘int'>
Type of b:<class ‘str’>
Type of c:<class ‘list’>
```

## Built-in function : id( )

id (object)

- It returns identity of an object.
- It is the address of object in memory
- It will be unique and constant throughout the lifetime of an object

**Ex:**
```
a=20
b=a
print (“value of a and b before incrementation”)
print(“id of a:”,id(a))
print(“id of b:”,id(b))
b=a+1
print(“Value of a and b after incrementation”)
print(“id of b: “ , id(b))
```

## Data types

Every value in Python has a data type. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. There are various data types in Python. Some of the important types are listed below.

- None :Nothing or no value associated
- Numbers
- List
- Tuple
- String
- Set
- Dictionary

**Numbers:**

Number stores numeric values. Python creates Number objects when a number is assigned to a variable

Ex: a,b=4,5    # a  and b  are objects

There are 3 sub types

- ➢int  : The int data type represents an integer number( 200)
- ➢float: The float data type represents floating point number(3.14)
- ➢complex: A complex number is a number  that is written in the form  of a+bj   or a+bJ(3.14j,  2-3.5J)

**bool:**   It is used to represent Boolean values(True and False)

**LISTS:**

Lists  in python are similar to arrays in C. However  the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].We can use slice [:] operators to access the data of the list.

```
 l  = [1, "hi", "python", 2]
print (l[2:])
print (l[0:2])
```

## Tuple:

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The elements in the tuple are separated with a comma (,) and enclosed in parentheses ().Whereas the list elements can be modified, it is not possible to modify the tuple elements

```
t  = (10,5.3,"hi", "python", 2)
print (t[1:]);
print (t[0:1]);
 t[3]=77                    #Error
```

## String :

A string is represented by group of characters represented in quotation marks. In python, strings are enclosed in single or double quotes. Multi-line strings can be denoted using triple quotes, ''' or """. slicing operator [ :] can be used to retrieve parts of string

```
s=‘hello’
 str2=“welcome “
str3 =“ “ “ python ” ” ”
print(s[-4])
print(s[2:])
```

## Dictionary:

Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key: value. Key and value can be of any type.

```
d = {1:'value','key':2}
print(d[1])
print(d[‘key’]
```

## Range:

The range data type represents a sequence of numbers. The numbers in the range are not modifiable. Generally range is used for repeating a for loop for specific number of times. To create a range of numbers, we can simply write

```
r=range(10)                r=range(2,9)
r=range(1,10,2)
```

## Set:

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered. it means the elements may not appear in the same order as they are entered into the sets

```
a = {5,2,3,1,4}
print(a)
```

**Type conversion:**

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. It is done by using different type conversion functions like int(), float(), complex and str() etc.

**int():**This function converts any data type to integer except complex type

```
>>>int(23.45)           #23
>>>int(“44”)            #44
>>>int(True)            #1
```

**float():**This function is used to convert any data type to a floating point number except complex type

```
>>>float(5)             #5.0
>>>float(“10.5”)        #10.5
>>>float(“10”)          #10.0
>>>float(True)          #1
```

**Complex():** This function convert other types to complex type

```
>>>complex(10)              #10+0j
>>>complex(False)           #0j
>>>complex(“10.5”)          #10.5+0j
```

**str():** This function convert other type values to str type

```
>>> str(10)                 #’10’
>>>str(True)                #’True’
```

## Operators

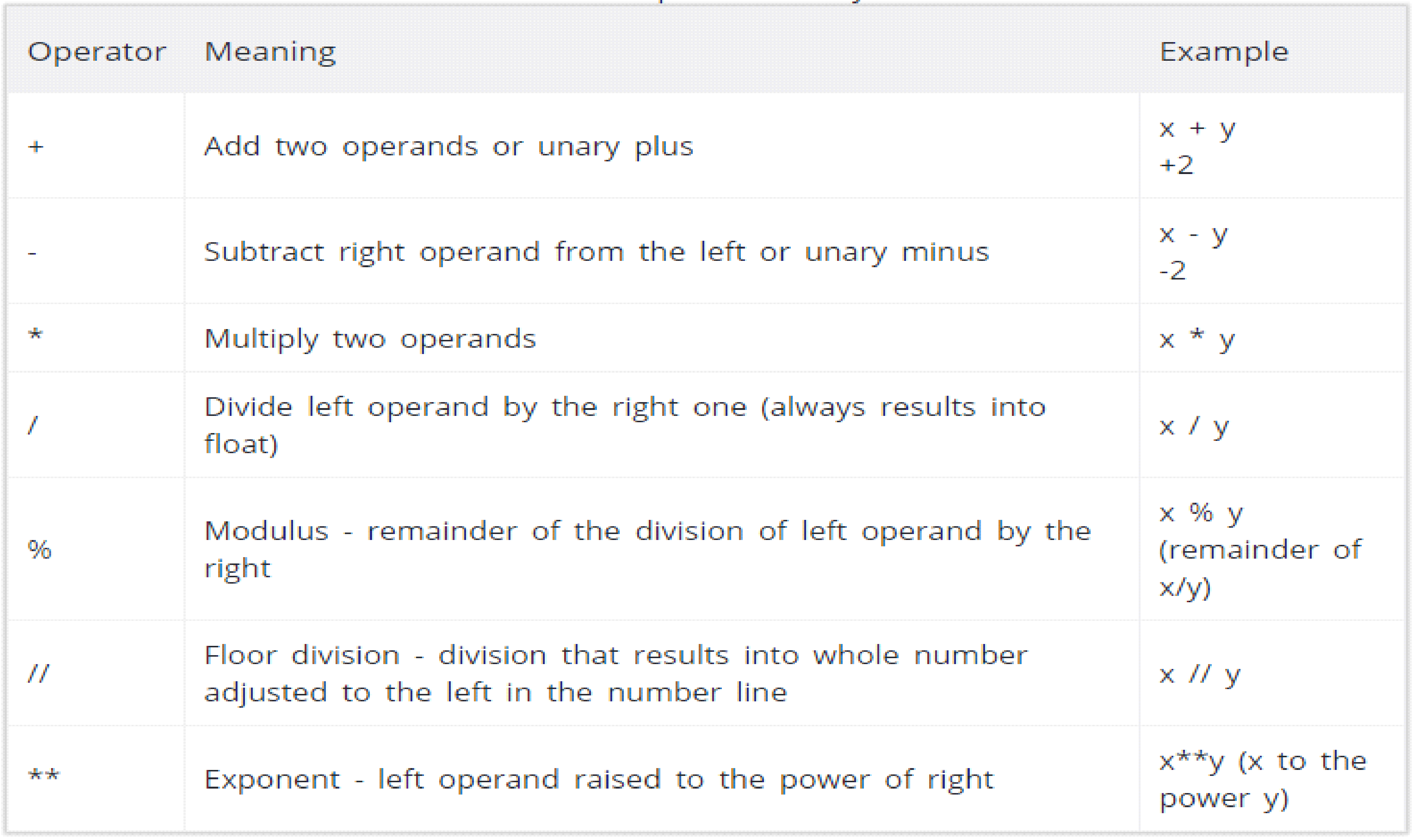
Operators are special symbols in Python that carryout arithmetic or logical computation.
The value that the operator operates on is called the operand.
Operators are used to perform specific operations on one more operands(variables) and provides a result

- **Arithmetic operators**
- **Relational Operators**
- **Logical Operators**
- **Assignment Operators**
- **Bitwise Operators**
- **Membership Operators**
- **Identity Operators**

## Arithmetic operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Arithmetic operators in Python



| Operator | Meaning | Example |
|---|---|---|
| + | Add two operands or unary plus | x + y<br>+2 |
| - | Subtract right operand from the left or unary minus | x - y<br>-2 |
| * | Multiply two operands | x * y |
| / | Divide left operand by the right one (always results into float) | x / y |
| % | Modulus - remainder of the division of left operand by the right | x % y (remainder of x/y) |
| // | Floor division - division that results into whole number adjusted to the left in the number line | x // y |
| ** | Exponent - left operand raised to the power of right | x**y (x to the power y) |

**Ex:**

```
x=15
Y=4
Print('x+y=',x+y)
Print('x-y=',x-y)
Print('x*y=',x*y)
Print('x/y='x/y)
Print('x||y=',x||y)
Print('x**y=',x**y)
```

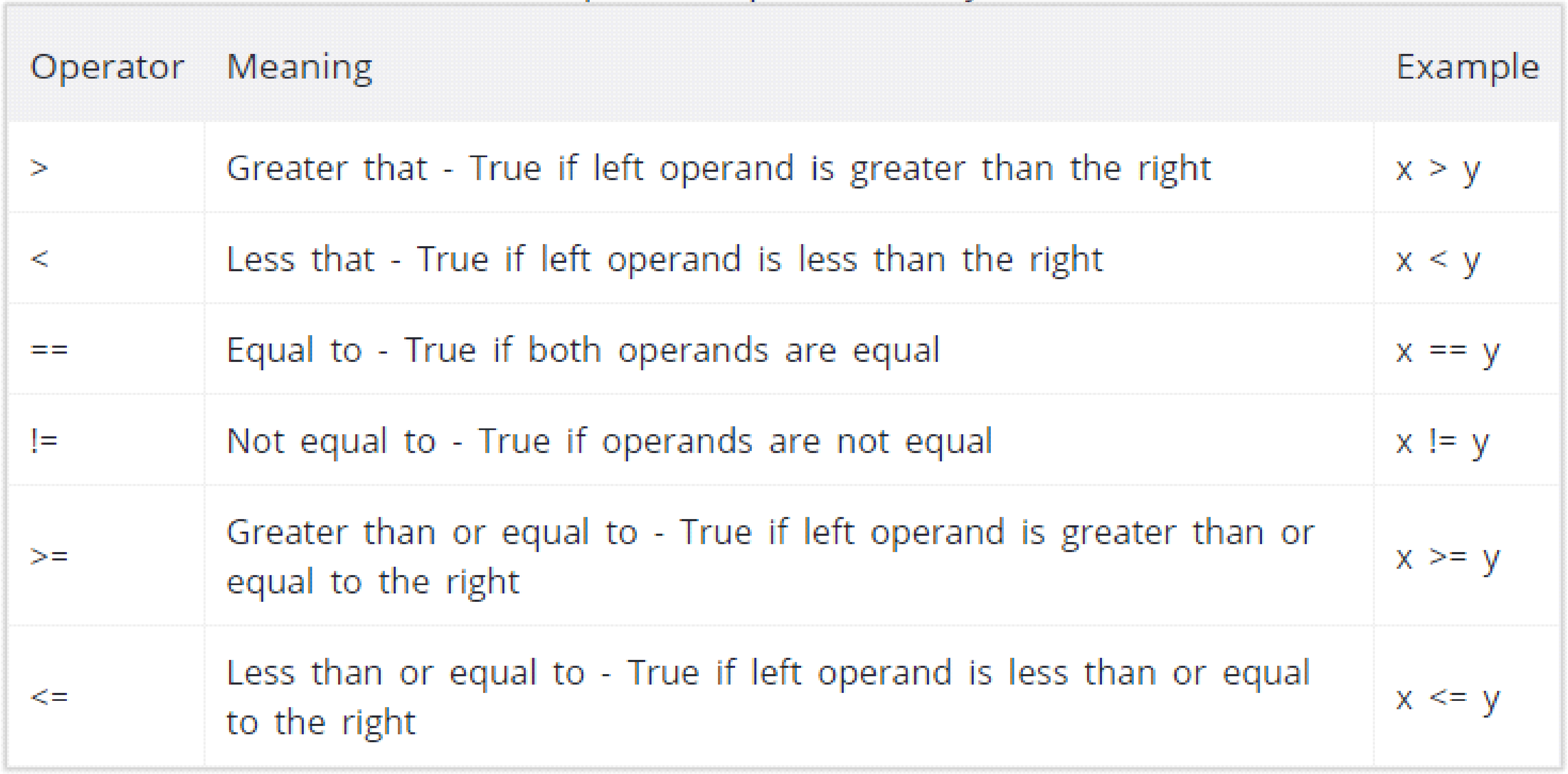
**Output**

```
x+y=19
x-y=11
x*y=60
x/y=3.75
x||y=3
x**y=50625
```

## Relational / Comparison operators

Comparison operators are used to compare values. It either returns True or False according to the condition. Used in conditional Statements.

Comparision operators in Python



| Operator | Meaning | Example |
|---|---|---|
| > | Greater that - True if left operand is greater than the right | x > y |
| < | Less that - True if left operand is less than the right | x < y |
| == | Equal to - True if both operands are equal | x == y |
| != | Not equal to - True if operands are not equal | x != y |
| >= | Greater than or equal to - True if left operand is greater than or equal to the right | x >= y |
| <= | Less than or equal to - True if left operand is less than or equal to the right | x <= y |

**Eg:**

```
x=10
Y=12
Print(‘x>y is’,x>y)
Print(‘x<y is’,x<y)
Print(‘x==y is’,x==y)
Print(‘x!=y is’,x!=y)
Print(‘x>=y is’,x>=y)
Print(‘x<=y is’,x<=y)
10<20<30<40
10>20<30<40
10<20>30<40
```

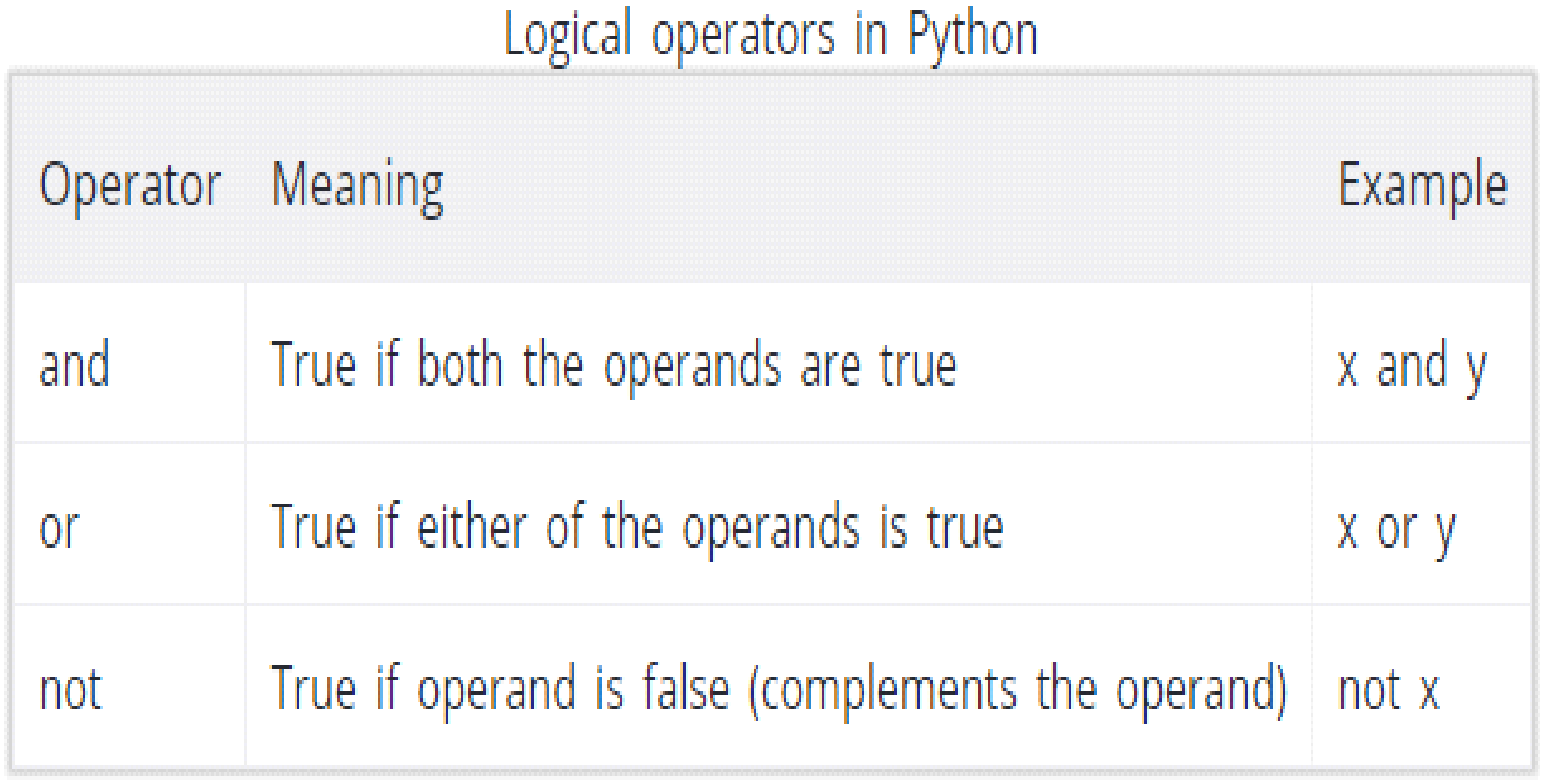
**Output**

```
x>y is FALSE
x<y is TRUE
x==y is FALSE
 x!=y is TRUE
x>=y is FALSE
 x<=y is TRUE

True
False
False
```

## Logical operators

Logical operators are the and, or, not operators. Based on Boolean Algebra returns results as either True or False



Logical operators in Python

| Operator | Meaning | Example |
|---|---|---|
| and | True if both the operands are true | x and y |
| or | True if either of the operands is true | x or y |
| not | True if operand is false (complements the operand) | not x |

**Eg**

```
x=True
Y=False
Print('x and y is',x and y)
Print('x or y is',x or y)
Print('not x is ',not x)
```

**Output**

```
x and y is false
x or y is true
not x is false
```

| | |
|---|---|
| 40 and 50 | 50 |
| 0 and 50 | 0 |
| 40 or 50 | 40 |
| 0 or 50 | 0 |

## Assignment operators

- Assignment operators are used in Python to assign values to variables. a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
- Multiple Assignments: Same value can be assigned to more than one variable

| Operator | Example | Equivatent to |
|---|---|---|
| = | x = 5 | x = 5 |
| += | x += 5 | x = x + 5 |
| -= | x -= 5 | x = x - 5 |
| *= | x *= 5 | x = x * 5 |
| /= | x /= 5 | x = x / 5 |
| %= | x %= 5 | x = x % 5 |
| //= | x //= 5 | x = x // 5 |
| **= | x **= 5 | x = x ** 5 |
| &= | x &= 5 | x = x & 5 |
| \|= | x \|= 5 | x = x \| 5 |
| ^= | x ^= 5 | x = x ^ 5 |
| >>= | x >>= 5 | x = x >> 5 |
| <<= | x <<= 5 | x = x << 5 |

Bitwise Operators
PYTHON
&
0
1
^
1
0
|
1
0
~
1
0

## Bitwise operators

Bitwise operators converts the operands data in the form of binary format performs operations on the binary values and gives the result in the form of decimal format.

| **Operator** | **Meaning** |
|---|---|
| & | bitwise AND |
| \| | Bitwise OR |
| ~ | Bitwise NOT |
| ^ | Bitwise XOR |
| >> | Bitwise right shift |
| << | Bitwise left shift |

## Implementation of And Operation on Binary Digits

| Operation | Result |
|---|---|
| 0 & 0 | 0 |
| 1 & 0 | 0 |
| 0 & 1 | 0 |
| 1 & 1 | 1 |

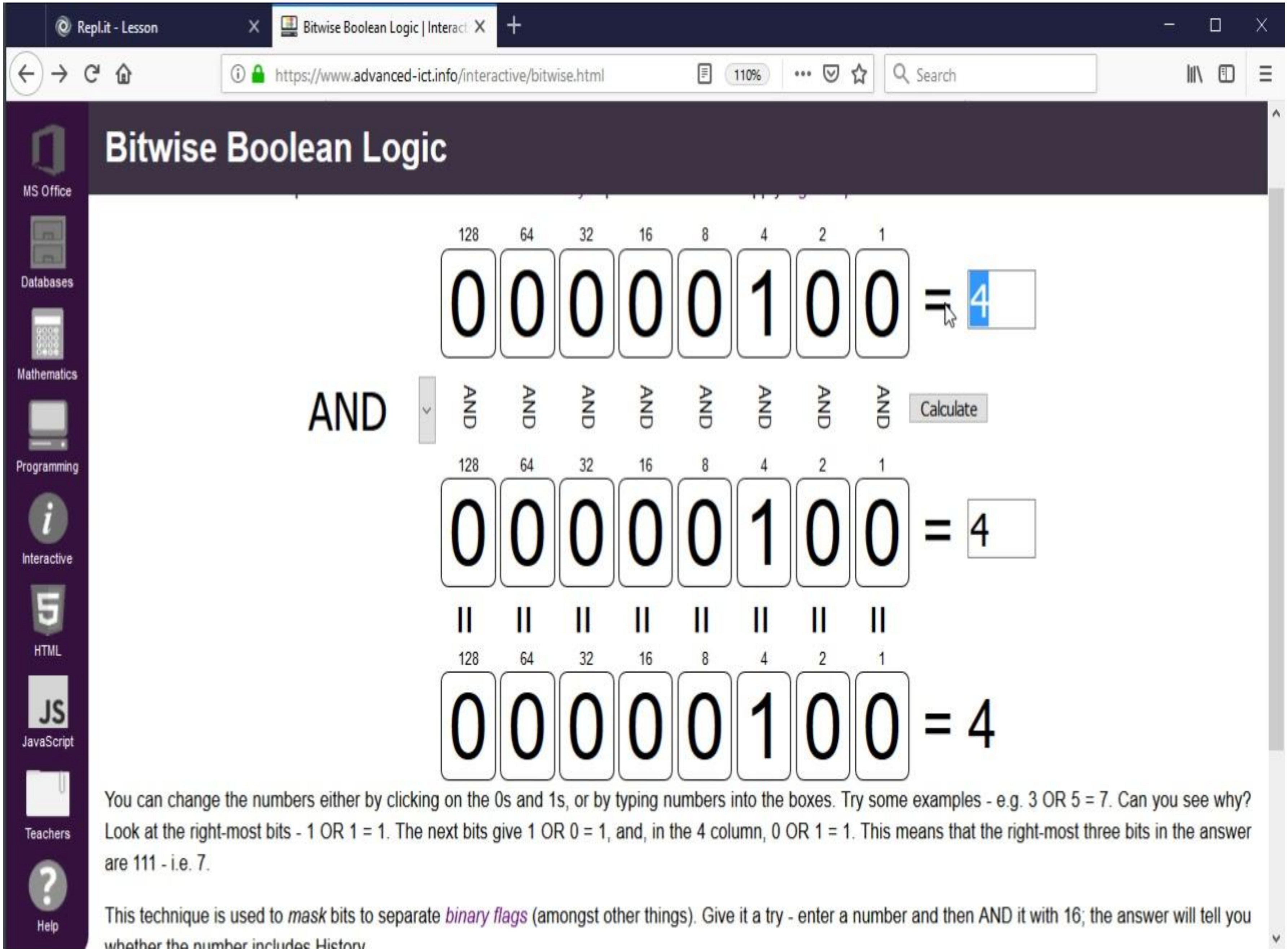
## Bitwise Operator

✓ Applied to integer type – long, int, short, byte and char.

| A | B | A \| B | A & B | A^ B | ~A |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

| X | Y | X&Y | X\|Y | X^Y | ~(X) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

| Operator | Description |
|---|---|
| ** | Exponentiation (raise to the power) |
| ~ + - | Complement, unary plus and minus (method names for the last two are +@ and -@) |
| * / % // | Multiply, divide, modulo and floor division |
| + - | Addition and subtraction |
| >> << | Right and left bitwise shift |
| & | Bitwise 'AND'td> |
| ^ \| | Bitwise exclusive `OR' and regular `OR' |
| <= < > >= | Comparison operators |
| <> == != | Equality operators |
| = %= /= //= -= += *= **= | Assignment operators |
| is is not | Identity operators |
| in not in | Membership operators |
| not or and | Logical operators |



# Bitwise Boolean Logic

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | = 4 |
| AND | AND | AND | AND | AND | AND | AND | AND | |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | = 4 |
| = | = | = | = | = | = | = | = | |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | = 4 |

You can change the numbers either by clicking on the 0s and 1s, or by typing numbers into the boxes. Try some examples - e.g. 3 OR 5 = 7. Can you see why? Look at the right-most bits - 1 OR 1 = 1. The next bits give 1 OR 0 = 1, and, in the 4 column, 0 OR 1 = 1. This means that the right-most three bits in the answer are 111 - i.e. 7.

This technique is used to *mask* bits to separate *binary flags* (amongst other things). Give it a try - enter a number and then AND it with 16; the answer will tell you whether the number includes History

## Bitwise operators in Python

| Operator | Meaning | Example |
|---|---|---|
| & | Bitwise AND | x& y = 0 (`0000 0000`) |
| \| | Bitwise OR | x \| y = 14 (`0000 1110`) |
| ~ | Bitwise NOT | ~x = -11 (`1111 0101`) |
| ^ | Bitwise XOR | x ^ y = 14 (`0000 1110`) |
| >> | Bitwise right shift | x>> 2 = 2 (`0000 0010`) |
| << | Bitwise left shift | x<< 2 = 40 (`0010 1000`) |

| | | |
|---|---|---|
| **Ex:** | x=10 | **Output** |
| | Y=4 | 0 |
| | Print(x&y) | 4 |
| | Print(x\|y) | -11 |
| | Print(x~y) | 14 |
| | Print(x^y) | 2 |
| | Print(x>>y) | 40 |
| | Print(x<<y) | |

## Special operators

Python language offers some special type of operators like the identity operator or the membership operator.

### Membership operators

- **in** and **not in** are the membership operators in Python.
- They are used to test whether a value or variable is found in a sequence(string, list, tuple, set and dictionary).

| **Operators** | **Description** |
|---|---|
| **in** | Returns to true if it finds a variable in given sequence else false |

**Ex**

```
x="hello "
print( 'h' in x)                              #True
print('hel' in x)
```

| **Operators** | **Description** |
|---|---|
| **not in** | Returns to true if it does not find a variable in given sequence else false |

Ex

```
a=[1,2,3,4]
print(5 not in a)
  print(5 in a)
```

## Identity Operators

- "**is** " and "**is not** " are the identity operators in Python.
- They are used to check if two values(or variables) are located on the same part of the memory. [used to compare memory locations of 2 objects]
- Two variables that are equal does not imply that they are identical.

| **Operator** | **Meaning** |
|---|---|
| **is** | Returns to true if variable on either side of operator are referring to same object else false |
| **Ex :** | a=10<br>b=10<br>x= a is b        # True |
| **is not** | Returns to false if variable on either side of operator are referring to same |
| **Ex**: | a=10<br>b=10<br>x= a is not b        #False |

## Conditional operator:

syntax: value1 if(condition) else value2

```
x=40 if(5<6) else 60
print(x)
```

## Expressions

An expression is a combination of values, variables, and operators. A value all by itself is considered an expression, and so is a variable. To evaluate these type of expressions there is a rule of precedence in Python.

**Precedence:** is used to determine the order in which different operators in a complex expression are evaluated

**Associativity** : Associativity is the order in which an expression is evaluated that has multiple operator of the same precedence. Almost all the operators have left-to-right associativity except exponent(**) operator

➢Some operators like assignment operators and comparison operators do not have associativity in Python

**EX:**

```
>>> 1 + 1
>>> x=3
>>> x+4
>>> 3/2*4+3+(10/4)**3-2
```

**EX:**

```
y=10
x=y+=12    #error
```

**x<y<z is equivalent to x<y and y<z**

## Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

| Operator | Description |
|---|---|
| () | parentheses |
| ** | Exponentiation (raise to the power) |
| ~ + - | Complement, unary plus and minus |
| */ % // | Multiply, divide, modulo and floor division |
| + - | Addition and subtraction |
| >> << | Right and left bitwise shift |
| & | Bitwise 'AND' |
| ^ \| | Bitwise exclusive `OR' and regular `OR' |
| <= < > >= | Comparison operators |
| <> == != | Equality operators |
| = %= /= //= -= += *= **= | Assignment operators |
| is   is not | Identity operators |
| in  not in | Membership operators |
| not or and | Logical operators |

## Escape Characters

➢ An *escape character is a special character that is preceded with a backslash (\), appearing inside a string literal.*

➢ When a string literal that contains escape characters is printed, the escape characters are treated as special commands that are embedded in the string.

➢Some of Python‟s escape characters

| **Escape Character** | **Effect** |
|---|---|
| \n | New line. |
| \t | Horizontal tab |
| \v | Vertical tab |
| \b | back space |
| \f | form feed |
| \’ | Single quotes |
| \“ | Double quotes |
| \\ | backslash symbol |

## Python Input [Reading Input from the Keyboard ]

➢In Python, **input( ) –**input function is used to take input from the user
➢This  function takes a value from the  keyboard  in the form of string

The syntax for input() is :

```
        variable = input(prompt)
```

where prompt is the string we wish to display on the screen. It is optional.

***Example:***

```
        >>>name = input()
        Apple
        >>> print(name)                          #Apple
        >>>name=input("enter your name")
```

We can use int () function before the input() to accept an  integer from the keyboard

```
        >>>x=int(input("enter your age"))
```

We can use float () function before the input() to accept a  float from the keyboard

```
        >>>x=float(input("enter your age"))
```

Reading multiple values from Keyboard

```
a, b=[int(x) for x in input("enter number").split()]
x,y,z =[float(i) for i in input("enter number").split()]
```

## Command Line arguments

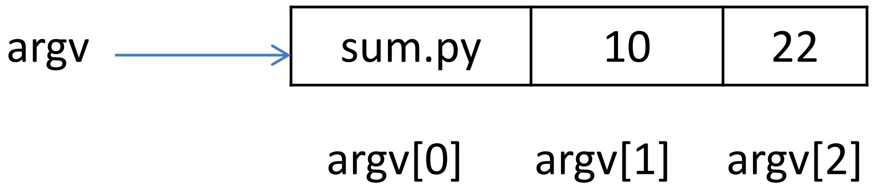
Command line arguments are arguments supplied to the program when it is invoked

Command line arguments are passed to the program from outside the program.

All the arguments should be entered from the keyboard separating them by space

These arguments are stored by default in the form of strings in list with name **argv** which is available in **sys** module

argv is a list that contains all the values passed to the program, argv[0] represents the name of the program,argv[1] represents the first value, argv[2] represents the second value and so on



Len(sys.argv) : number of command line arguments

## Python Output Using print( ) function

We use the print( ) function to output data to the standard output device (screen).

The actual syntax of the print() function is:

**print(*objects, sep=' ', end='\n', file= sys. stdout, flush=False)**

- **objects** is the value(s) to be printed
- **sep-** separator is used between the values. It defaults into a space character.
- **end**-After all values are printed, end is printed. It defaults into a new line.
- **File**-The file is the object where the values are printed and its default value

  is sys.stdout (screen)
- **flush:** If True, the stream is forcibly flushed. **Default value**: False

**Example**

```
>>>print()                              #
>>>print('hello')                       # hello
>>>a = 5
>>>print('The value of a is', a)        # The value of a is  5
>>>print("hello","world")               # hello  world
>>>print("hello"+"world")               # helloworld
```

```
>>>print(3*"hi")                          #hihihi
>>>print("hello \t world")                          #hello                world
>>>print(1,2,3,4)                                   #1 2 3 4
>>>print(1,2,3,4,sep='@')                           #1 @2 @3@ 4
>>> print(1,2,3,4, sep="->")                        #1->2->3->4
>>>a=5
>>>print("a=", a, sep='#####', end='\n\n\n')
```

print() with end attribute

```
        print("hello", end=' ')
        print("ITF4" , end=' ')                     # hello  ITF4
```

pass object(like list, tuple , set)  to print ()

```
        l=[10,20,30,40]
        print(l)                                    #[10,20,30,40]
```

print(formatted string)

```
        print("formatted string"%(variable list))
         print("a=%d \n b=%d" %(a, b))
```

print with replacement operator{}

```
        name, rno, avg="IT",15,99.2
        print("Hello {0} your rno is{1} and your avg is{3}".format(name , rno, avg))

        print("hello {0}  {1}". format("IT", "good morning")

        print("hello {name},{greet}".format(greet="good  morning" , name="SNIST"))
```

## Displaying Multiple Items with the + Operator

+ operator is used with two strings, it performs *string concatenation. This means that it appends one string to another.*

*>>> print('This is ' + 'one string.')*

***O/P :*** *This is one string.*

*>>> print('Enter the amount of ' + \*

'sales for each day and ' + \\

'press Enter.')

**O/p :** Enter the amount of sales for each day and press Enter.

## Conditional Statements (if ,if- else ,Nested if-else)

- ➢ Generally Python Program execution starts from the first line, executes each and every statement only once and stops the execution of the program when control reached to the last line.
- ➢ These Statements are used to disturb the normal flow of execution of the program.

### Block:

- ➢ The set of statements which are following some space indentation is known as a Block.
- ➢ Block begins when the indentation increases.
- ➢ Blocks can contain other Blocks.
- ➢ Blocks end when the indentation decreases to zero or to a containing blocks indentation.

Python supports 3 types of conditional statements. They are

- If
- else
- elif

**1. if:**

Condition returns true then it executes the if block otherwise if block execution is skipped.

***Syntax:*** **if condition:** statement (one statement)

**Or**

```
if condition :
        Statement 1
        Statement 2              (multiple statements)
        --------------
```

**EX:**

```
Print (“begin”)
x = input (“enter a positive number”)
i= int(x)
if i<10:
        print (“given number is 1 digit number”)
print (“end”)
```

**Output 1:** begin

```
enter a positive number5
Given number is 1 digit number
end
```

**Output2:** begin

```
enter a positive number120
end
```

## 2. else:

else block should be preceded by any one of the following blocks.

1.if 2.elif 3.while 4.for

else block preceding block condition returns false then only it will execute the following else block.

***Syntax:***

```
if condition:
          statement1
          statement2
          --------------
else:
          statement1
          statement2
          -------------
```

**Ex:**

```
print (“begin”)
x=input (“enter a positive number”)
i = int(x)
if x>10:
        print (“Given number is 1 digit number”)
else:
        print (“Given number is >=2 digit number”)
print (“end”)
```

**output 1:**

```
enter a positive number5
Given number is 1 digit number
end
```

**output 2:**

```
enter a positive number5
Given number is >=2 digit number
end
```

## 3.elif:

- ➢ elif block should be preceded by either if block or another elif block.
- ➢ elif block preceding block returns false then only control will go to the elif block.
- ➢ After reaching the control block to the elif block, if condition returns true then only it will execute elif block.

Syntax:

```
if(con1):
        stmt
elif(con2):
        stmt
elif(con3)
        stmt
else
        stmt
```

**Ex:**

```
print (“begin”)
x = input(“enter a positive number”)
i = int(x)
if i<10:
        print (“given number is 1 digit number”)
elif i<100:
        print (“given number is 2 digit number”)
elif i<1000:
        print (“given number is 3 digit number”)
else :
        print(“given number is  >=4 digit number”)
print (“end”)
```

**Output1:**

```
begin
enter a positive number5
given number is 1 digit number
end
```

**Output2:**

```
begin
enter a positive number15
given number is 2 digit number
end
```

# Nested If else:

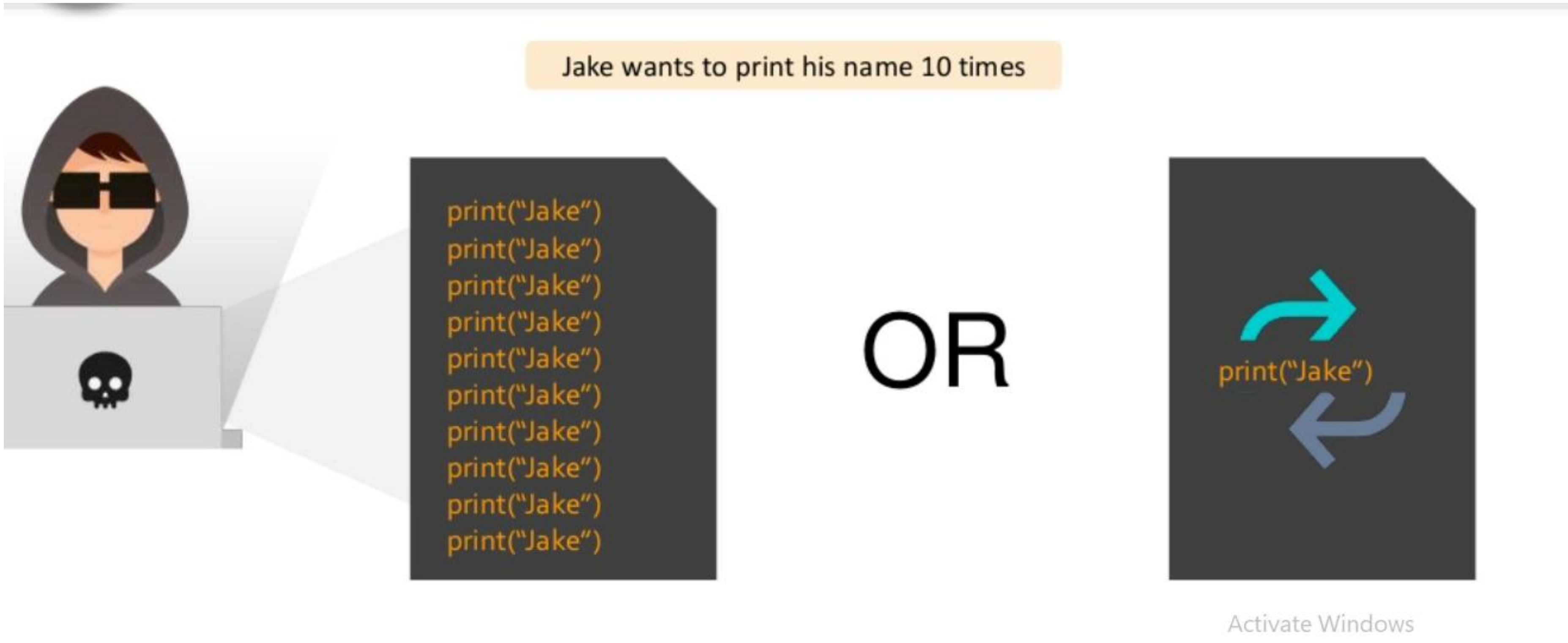
```
if(cond1):
        if(cond2):
                stmt
        else:
                stmt
else:
        if(cond3):
                stmt
        else:
                stmt
```

```
if(cond1):
        if(cond2):
                stmt
        else:
                stmt
else:
        stmt
```

```
if(cond1):
        if(cond2):
                stmt
        else:
                stmt
```

```
if(cond1):
        stmt
else:
        if(cond2):
                stmt
        else:
                stmt
```

## LOOPS



Jake wants to print his name 10 times

```
print("Jake")
print("Jake")
print("Jake")
print("Jake")
print("Jake")
print("Jake")
print("Jake")
print("Jake")
print("Jake")
print("Jake")
```

OR

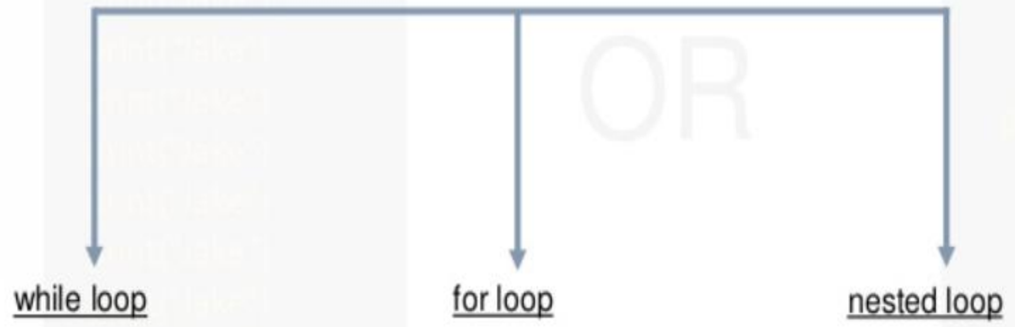
```
print("Jake")
```

**Iteration** means executing the same block of code over and over, potentially many times. A programming structure that implements iteration is called a **loop**.

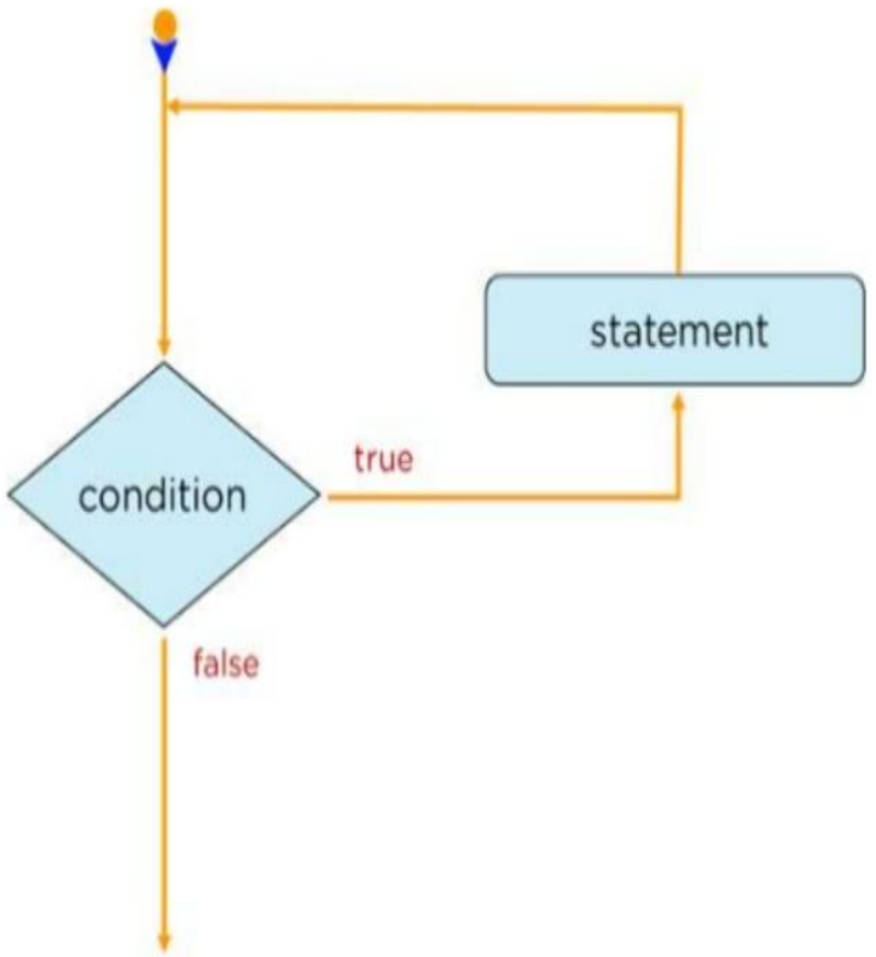
## What are loops?

Jake wants to print his name 10 times

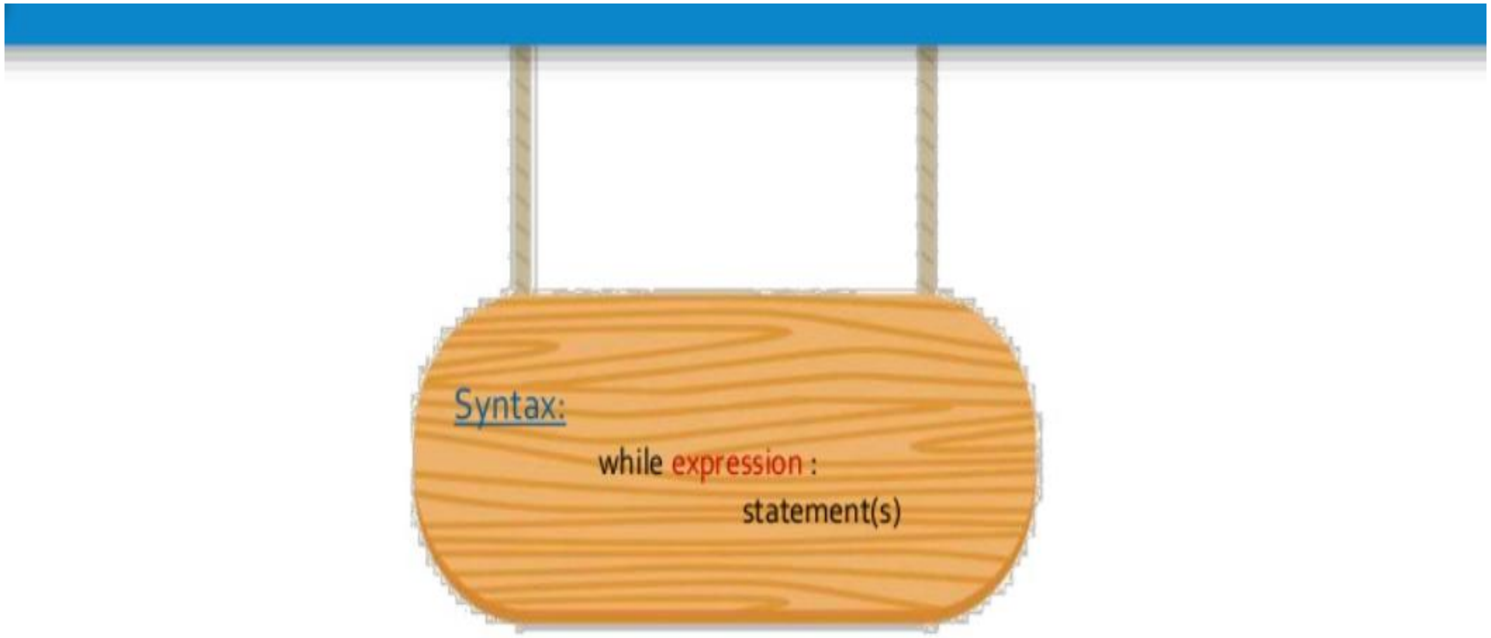
A loop is an instruction that repeats multiple times as long as some condition is met



# LOOPS



# FOR LOOP



Syntax:

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

The while loop is used to repeat a section of code an unknown number of times until a specific condition is met

```
for val in sequence:
        Body of for
```

## The `while` Loop

Let's see how Python's `while` statement is used to construct loops. We'll start simple and embellish as we go.

The format of a rudimentary `while` loop is shown below:

Python

```python
while <expr>:
    <statement(s)>
```

`<statement(s)>` represents the block to be repeatedly executed, often referred to as the body of the loop. This is denoted with indentation, just as in an `if` statement.

Consider this loop:

Python >>>

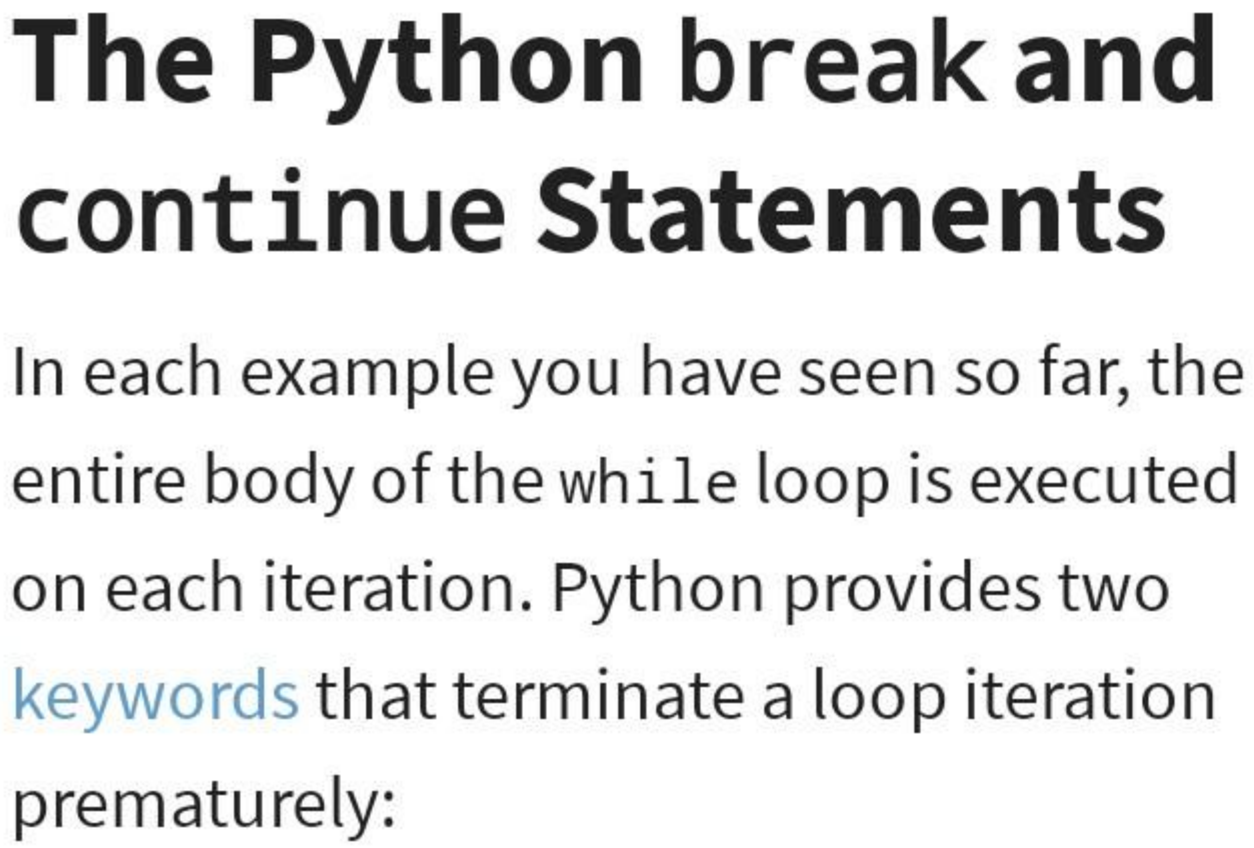
```
>>> n = 5
>>> while n > 0:
...     n -= 1
...     print(n)
...
4
3
2
1
0
```

```
Python                                                    >>>

>>> n = 0
>>> while n > 0:
...     n -= 1
...     print(n)
...
```

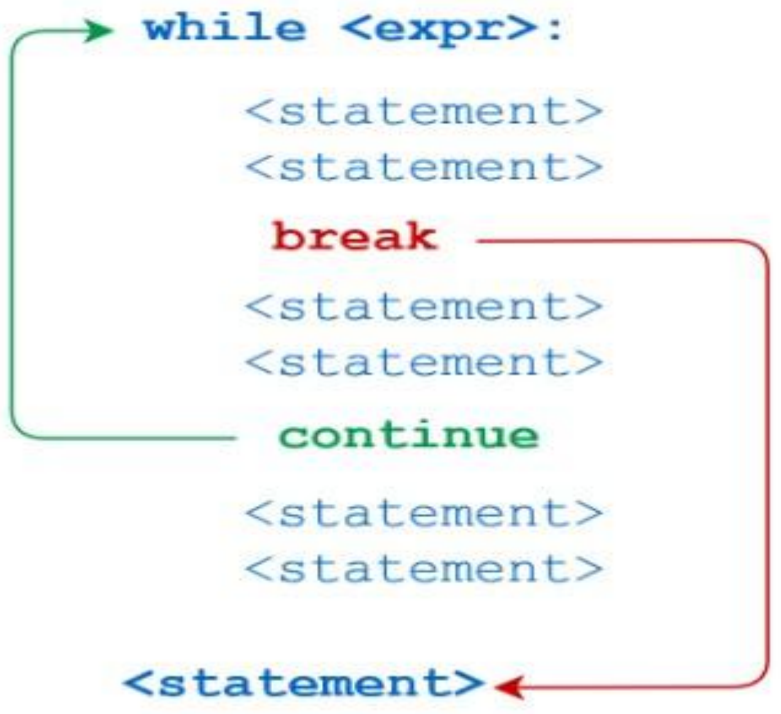
In the example above, when the loop is encountered, `n` is `0`. The controlling expression `n > 0` is already false, so the loop body never executes.

# Break and Continue



## The Python `break` and `continue` Statements

In each example you have seen so far, the entire body of the `while` loop is executed on each iteration. Python provides two keywords that terminate a loop iteration prematurely:



break and continue

Here’s a script file called `break.py` that demonstrates the `break` statement:

- The Python **`break`** statement immediately terminates a loop entirely. Program execution proceeds to the first statement following the loop body.

- The Python **`continue`** statement immediately terminates the current loop iteration. Execution jumps to the top of the loop, and the controlling expression is re-evaluated to determine whether the loop will execute again or terminate.

Python

```python
n = 5
while n > 0:
    n -= 1
    if n == 2:
        break
    print(n)
print('Loop ended.')
```

Running `break.py` from a command-line interpreter produces the following output:

Windows Console

```
C:\Users\john\Documents>python break.
4
3
```

## Looping( for, while nested loops)

### While

- In python, while loop is used to execute a block of statements repeatedly until a given a condition is satisfied.
- when the condition becomes false, the line immediately after the loop in program is executed.

Syntax:

```
while (condition):
        stmts
```

Ex:

```
i =1
sum =0
while i<=100:
        sum = sum+i
        i = i+1
print (“sum”)
```

## while loop with else

A while loop can have an optional else block as well. 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 case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

```
print (“begin”)
i =1
while i<=5:
        print (“welcome”)
        i = i + 1
else:
        print (“in else”)
print (“end”)
```

**Output:**

```
begin
welcome
welcome
welcome
welcome
welcome
in else
end
```

## for

for loop executes the given logic with respect to every element of the iterable object

***Syntax:*** for val in sequence:

```
                ----------------------------
                ----------------------------
```

Where sequence can be string or any collection. val is the variable that takes the value of the item inside the sequence on each iteration.

**Ex**

```
num = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in num:
        print (i)

for i in range(10,20,2):
        print(i)

s="snist"
for i in s:
     print(i)
```

## 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. Break statement can be used to stop a for loop. In such case, the else part is ignored. Hence, a for loop's else part runs if no break occurs.

Ex:

```
digits = [0, 1, 5]
for i in digits:
        print(i)
else:
        print("No items left.")
```

**Nested Loops:** loop inside another loop.

**syntax:**

```
for var in sequence:
        for var in sequence:
                statements(s)
        statements(s)

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

## Control Statements(break , continue ,pass)

**break**

To interrupt the loop, to start a new iteration (one “round” of executing the block), or to simply end the loop we use break.

**Ex:**

```
print("begin")
i=1
for i in "python":
    if i=='h':
        break
    else:
        print(i)
print("end")
```

**output**

```
begin
p
y
t
end
```

**Ex:**

```
print (“begin”)
i =1
while True:
        print (“welcome”)
                i =i+1
        if i==5:
                break
else:
        print (“if else”)
print (“end”)
```

```
Output
begin
welcome
welcome
welcome
welcome
end
```

## continue

- It causes the current iteration to end, and to “jump” to the beginning of the next.
- It basically means “skip the rest of the loop body, but don’t end the loop.”

**Ex:**

```
print("begin")
i=1
for i in "python":
    if i=='h':
        continue
    else:
        print(i)
print("end")
```

**output**

```
begin
p
y
t
o
n
end
```

## Pass

In Python programming, pass is a null statement. The difference between a **comment** and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored.

However, nothing happens when pass is executed. It results into no operation (NOP).

**Ex**

```
sequence = {'p', 'a', 's', 's'}
for  val  in  sequence:
        pass
```

**File:** File is a named location on Hard disk to store related information.

- ➢File is a chunk of logically related data or information which can be used by computer programs.
- ➢File is used to permanently store data in a non-volatile memory(e. g. hard disk)
- ➢Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.
- ➢When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

**Types of Files**

Text file : stores the data in the form of characters

Binary file: stores data in the form of bytes. These files can be used to store text, images ,audio and video

In python, a file operation takes place in the following order

1.open a file
2.process file(read or write)
3.close file

## Opening a file:

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.
we use **open()** function along with two arguments, that accepts file name and the mode

Syntax: **fileObject=open(file_name,access_mode)**

**file_name**: It is the name of the file which you want to access.
**access_mode**: It specifies the mode in which File is to be opened.

```
Ex:     f = open("python.txt",'w')            # open file in current directory
        f = open("/home/rgukt/Desktop/python/hello.txt")
                                              # specifying full path
```

There are four different methods (modes) for opening a file:

| | |
|---|---|
| **r** | Default value. Opens a file for reading, error if the file does not exist |
| **a** | Opens a file for appending, creates the file if it does not exist |
| **w** | Opens a file for writing, creates the file if it does not exist |
| **X** | Creates the specified file, returns an error if the file exists |

you can specify if the file should be handled as binary or text mode
"t" - Text - Default value. Text mode
"b" - Binary - Binary mode (e.g. images)

| | |
|---|---|
| **r** | Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode. |
| **w** | Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| **a** | Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| **r+** | Opens a file for both reading and writing. The file pointer placed at the beginning of the file. |
| **w+** | Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| a+ | Opens file in reading and appending mode. If file already exists, then append the data at the end of existing file, else create a new file. |

|  |  |
|---|---|
| **rb** | Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. |
| **wb** | Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| **ab** | Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| **rb+** | Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file. |
| **wb+** | Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| **ab+** | Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |

## Closing a file:

When we are done with operations to the file, we need to properly close the file.
Closing a file will free up the resources that were tied with the file and is done using Python close() method.
Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

```
file object. close( )
```