

## Table of content

### Contents

Contents.....	1
UNIT-1 .....	3
BASICS OF PYTHON .....	3
Installation process:.....	3
Using the python interpreter, IDLE .....	4
Overview of IDLE .....	5
1.1 Features of Python .....	9
Salient Features of Python .....	10
Enhanced Readability: .....	10
Dynamic Typing: .....	11
Extensible Language .....	12
Standard DB2 API .....	12
GUI Programming .....	12
Embeddable.....	12
1.2 Basic Building blocks (Syntax) of Python.....	13
Identifier:.....	13
Keywords: .....	15
Indents:.....	16
Comment:.....	17
Variables .....	17
1.3 Data Types and Variables: .....	20
Data Types in Python:.....	21
Mutable and Immutable Objects .....	22
1.4 Python Operators.....	28
Arithmetic Operators .....	28
String Data operators: .....	36
Comparison Operators (Relational operators) .....	49
Logical Operators.....	50

Bitwise Operators.....	51
Assignment Operators.....	52
Identity Operators.....	54
Membership Operators .....	54
1.5 Conditional statement .....	55
Introduction to 'If' statement.....	57
If Else Statement .....	59
If...Elif..else Statement.....	59
Nested IF Statement.....	60
1.6 Loopings.....	63
While Loop: .....	64
for loop.....	70
Nested loops.....	76
1.7 Loop Manipulation:.....	77
Continue Statement.....	77
Break Statement:.....	80
Pass Statement: .....	82

## UNIT-1

### BASICS OF PYTHON

Unit	Unit Outcomes (UOs) (in cognitive domain)	Topics and Sub-topics
<b>Unit – I Basic Of Python</b>	1a. Installation of Python 1b. Develop python program using different operators. 1c. Write a python program using conditional statement and looping	1.1 Python Features 1.2 Python building blocks:-Identifier, keywords, Indentation, variables, comments 1.3 Python Data Types:- Number, string, Tuple, List, Sets, Dictionaries 1.4 Python Operators:- Assignment, relational, Logical, Bitwise, Membership, Identity 1.5 Conditional statement:- if, if....else, nested if 1.6 Looping :- while, for, nested loop 1.7 Loop manipulation:- pass, break, continue

### Programming with Python:

Did you know that Dropbox, YouTube, Instagram, reddit, Quora and Bit orient make use of Python? It is so versatile that it can be used for image processing, Graphic design, 3D modelling and scientific data processing. It is awesome programming language which is easy to learn, flexible and open source. It is such a wide spread use that python programmers are in high demand.

### **Installation process:**

#### **Python Installation for windows.**

**Note:** Installation of Python on Windows:

- For 32-bit Windows OS, download the x86 file.
  - For 64-bit Windows OS, download the x86-64 file.
- 1) Visit the website Python.org
  - 2) Click on download of dropdown menu, then windows
  - 3) Select version 3.6.2
  - 4) Click on download Windows X-86 – 64 executable Installer
  - 5) Install Python 3.6.2 [64 bit] dropdown menu opens.
  - 6) Click on checkbox, add Python 3.6 to path
  - 7) Click Customised installation, then click next
  - 8) Click checkbox, install for all users, then click install
  - 9) To check Python interpreter. Click on start and go to command prompt and type python and Crosscheck 3.6.2 is installed, you can see python prompt, which is 3 greater than Signs, we

learnt how to download and install Python on a Windows platform which is Wizard based project

## **Python Installation for Linux and Mac OS**

### **Installation of Python on Linux**

Using the Package Manager

1. If you are using Ubuntu, type the following command in the command prompt:

```
$ sudo apt-get install python3
```

2. If you are using Fedora, type the following command in the command prompt:

```
$ sudo yum install python3
```

3. The most recent version of Python 3 will be downloaded and installed. To verify the installation, type:

```
$ Python
```

The Python prompt (>>>) will appear.

### **Installation of Python on Mac OS X**

A. Using the wizard-based installer

1. Click [here](#) to download the required version for Mac.
2. Run the downloaded file and follow the instructions in the installation wizard.

B. Alternative Installation Method Using Homebrew

Homebrew is a package manager that lets you install, update, and uninstall packages from the command line on the Mac OS.

1. Homebrew depends on Apple's Xcode package, so run the following command to install Xcode first:

```
$ xcode-select --install
```

2. Next, install Homebrew by following the instructions on their website: <https://brew.sh/>
3. After installing Homebrew, from the prompt in the terminal type the following command to install Python:

```
$ brew install python3
```

4. To verify the installation, type:

```
$ Python3
```

The Python prompt (>>>) will appear.

## **Using the python interpreter, IDLE**

Use the python Interpreter IDLE

Write a basic program to display the text "Hello Word"

It is the time to start writing Code

The standard python distribution include the development tool Called IDLE, which allows us to write code and run our code easily

## IDLE -Integrated DeveLopment Environment

When you launch IDLE a new window opens up.



This is the interactive shell these three greater than sign are the **python prompt**, you can type any python instruction such as simple arithmetic expression

for eg-

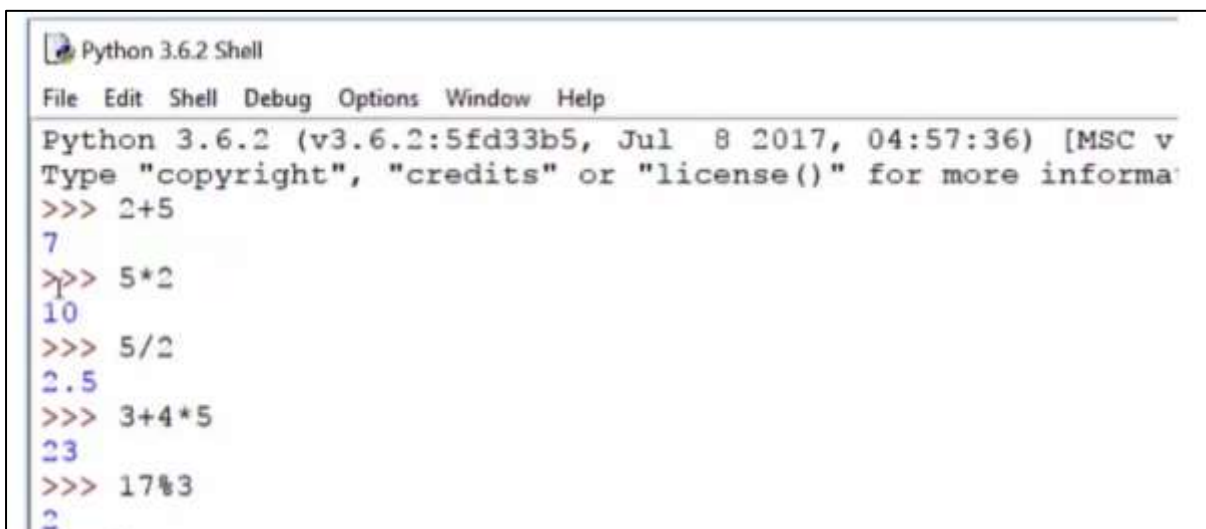
2+5 press enter to see the result or type

5\*2 and press enter here \* ( asterisk) is used as multiplication

For division we use forward slash, you can even use longer expression: [3+4\* 5 ]=23

In Computing terminology the modulo operator % also known as modulus find the remainder after dividing a number by another

for eg: 17 % 3 answer is 2.



## Overview of IDLE

Let us learn more about IDLE and how it come about.

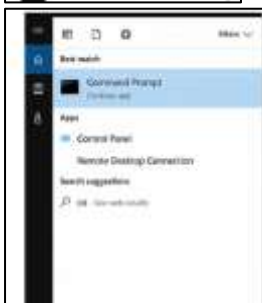
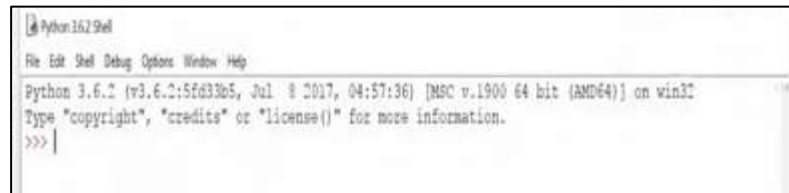
IDLE is written in python and uses **Tkinter graphics library**. The author of Python Guido Van

Rossum says that language is named after British comedian Monty Python. It is believed that the name IDLE was probably also chosen partly to honour Eric Idle one of Monty Python's founding member.

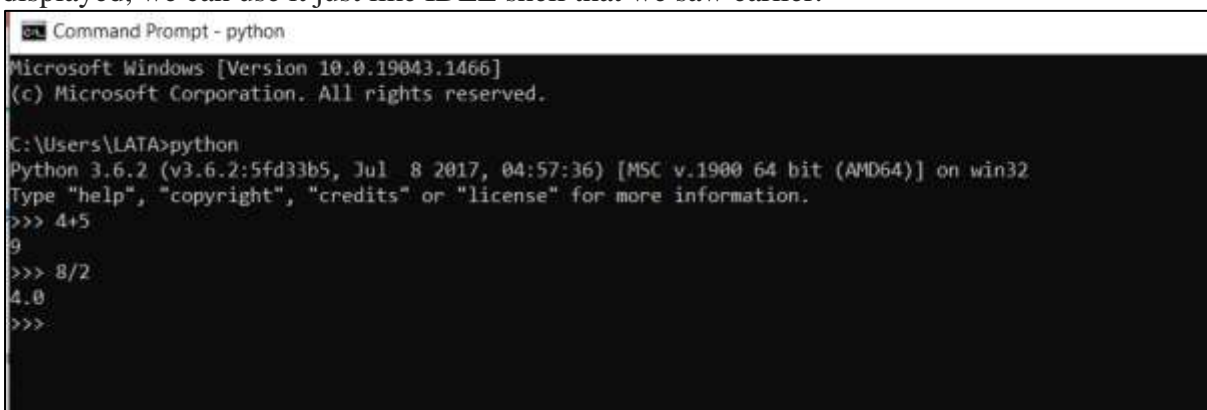
### IDLE has several features

- It has an interactive Python shell
- There is a fully featured text editor to write python code. This editor allows us to highlight syntax auto complete known code elements and adds indents where required.
- There is also a debugger which helps to catch error.

To start IDLE from Windows, simply type IDLE in search box and click on the IDLE python.



You can even launch python shell from command prompt, type python and once python prompt is displayed, we can use it just like IDLE shell that we saw earlier.



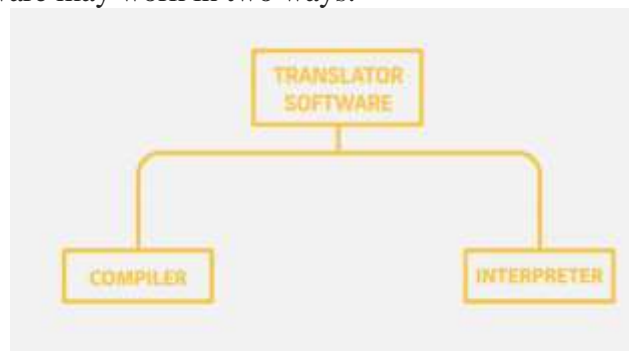
This is known as interpreter mode

## **What is known as interpreter mode?**

Keywords of any programming language need to be converted into machine language so that code can run.



This is done by Translator software may work in two ways.



**Compiler:** It considers all the instructions in a program and converts the source code into equivalent machine code.

Ex: C, C++ use compiler

**Interpreter:** It takes one instruction at a time and executes it in real time. It is an interactive way of executing program.

Ex: Python uses an interpreter.

## Using IDLE:

IDLE can-do lot more than arithmetic expression evaluation.

For ex:

We can use inbuilt function **input** to accept data from keyboard and assign to a variable.

```

>>> colour=input("What is your favourite colour?")
What is your favourite colour? Green
>>> colour
'Green'
>>>
  
```



```
print('Hello World!')
```

Another inbuilt function that comes in handy is the **print** function. This function is used to display the output.

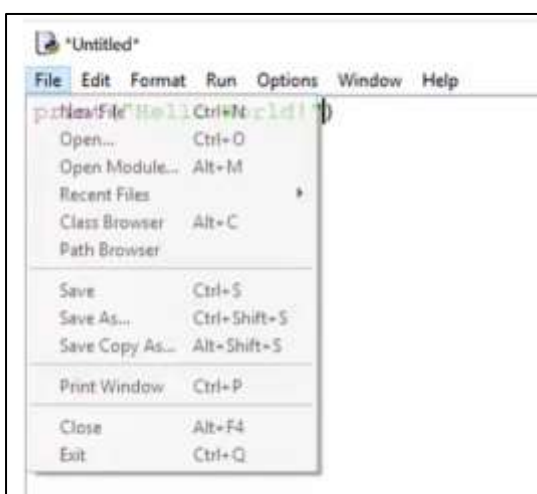
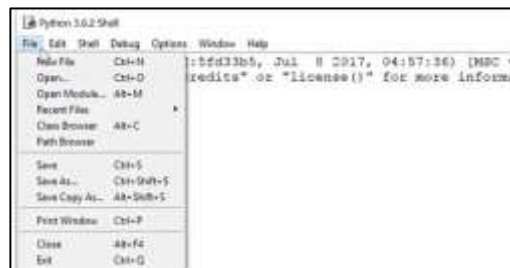
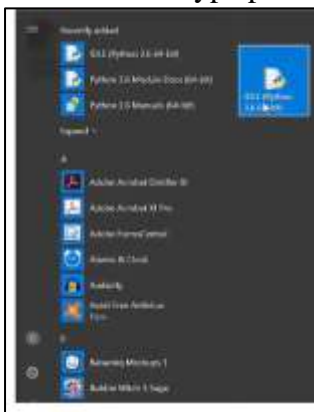
```
>>> print("Python is cool!")
Python is cool!
>>>
```

We can also use IDLE to save our program and run it.

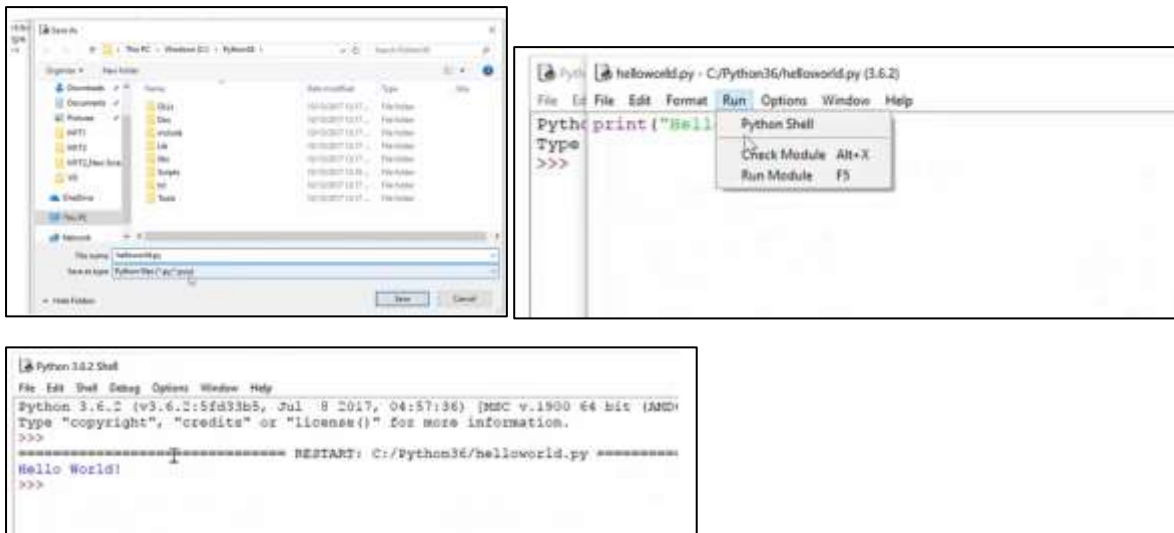
Let us start our first program to display text “Hello World”

Let us open IDLE

- Start text editor by clicking on file and then New
- Here is a new window. This is text editor.
- Let us type print command to display Hello world







## Code Challenge

In the given code, add the text that will display the text, Hello World!

```
>>>print()
```

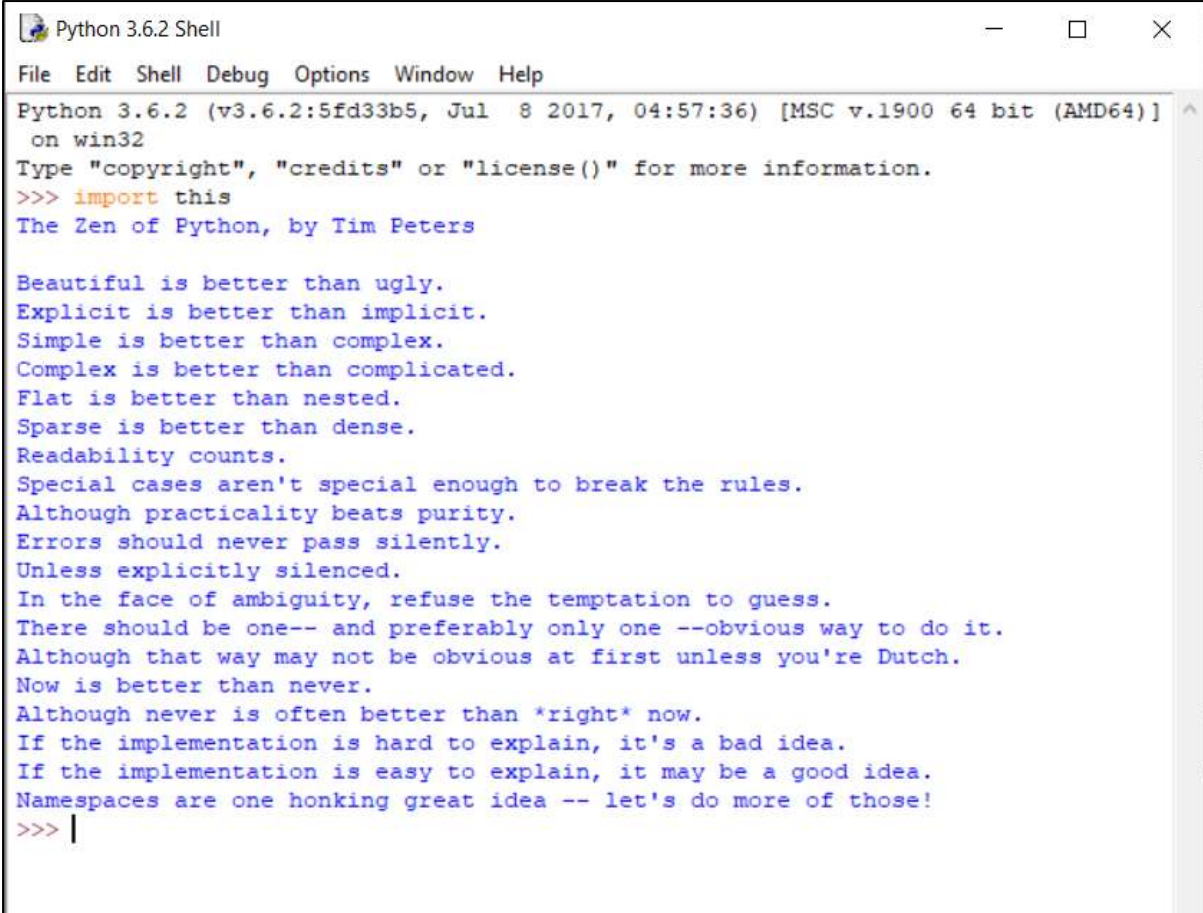
## 1.1 Features of Python

Python is a **High level programming language**. Python is used for web development, scientific computing, Data analytics and so on. Python supports multiple programming paradigms such as

- Imperative
- Procedural
- Object oriented
- Functional programming styles

It is also known as **open-source language**. Design philosophy of python are 20 principles.

Type import in python prompt.



```

Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>> |

```

## **Salient Features of Python**

Let us take a closer look at some of the salient features of Python.

- **Enhanced Readability**
- **Dynamic Typing**
- **Interpreted Language**
- **Extensible Language**
- **Standard DB2 API**
- **GUI Programming**
- **Embeddable**

### **Enhanced Readability:**

In Python, uniform indents are used to delimit blocks of statements instead of curly brackets like in C, C++, and Java. This enhances readability.

**In C, C++, Java**

```

if (X%2==0)
{
if (x%3==0)
System.out.println("divisible by 2 and 3");
else
System.out.println("divisible by 2 not divisible by 3");
}
else
{
if (x%3==0)
System.out.println("divisible by 3 not divisible by 2");
else
System.out.println("not divisible by 3 nor by 2");
}

```

**In Python**

```

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 3 nor by 2")

```

**Dynamic Typing:**

In Python, a variable to be used in a program need not have prior declaration of its name and type. You will learn details about variables in the next module.

**Java is statically typed**

```

String var; // Variable is explicitly declared.
var="Hello";
var=1234; // Not allowed because the variable
has been declared as a String.

```

**Python is dynamically typed**

```

var="Hello" # Variable is NOT explicitly
declared.
var=1234 # Allowed although variable was
initially String type

```

**Interpreted Language**

Python is an interpreter-based language. An interpreter executes one instruction at a time. So if there is an error on line 7 of the code, it will execute instructions till line 6 and then stop. This is unlike a compiler which will not execute any of the instructions even if there is a single error. Thus, an interpreter is useful if you are new to programming because it allows you to see partial output of your code and identify the error location more easily.

<b>Java program (Compiled)</b>	<b>Python program (Interpreted)</b>
//This program will not display any of the phrases as there is an error on line 7.	#This program will display two phrases but not the third phrase as there is an error in line 3.
<pre> 1 public class test 2 { 3   public static void main(String args[]) 4   { 5     System.out.println("Welcome to"); 6     System.out.println("Python training program"); 7     System.out.println(From V.E.S.P); 8   } 9 } </pre>	<pre> print ("welcome to") print ("Python training program from") print (V.E.S.P) </pre>

## Extensible Language

Python extension modules implement new built-in object types and can call C libraries and system calls which Python doesn't have direct access to. Python can be extended to work with other languages like C, C++ and Java.

## Standard DB2 API

A standard data connectivity API facilitates using data sources such as Oracle, MySQL, and SQLite as a backend to a Python program for storage, retrieval, and processing of data. Standard DB2 API is a common interface provider to all the databases. Examples: Oracle, MySQL, DB2, etc.

## GUI Programming

GUI programming is nothing but creating buttons, text boxes, other user interactable elements and attaching function to the events like button click, key press. The standard distribution of Python contains a Tkinter GUI kit, which is an implementation of the popular GUI library called Tcl/Tk. An attractive GUI can be constructed using Tkinter. Many other GUI libraries like Qt, GTK, WxWidgets, etc. are also ported to Python.

## Embeddable

Python can be integrated with other popular programming technologies such as C, C++, Java, ActiveX, and CORBA. Python provides libraries to embed Python code in other languages like Django can be used to embed python code in HTML etc.

## 1.2 Basic Building blocks (Syntax) of Python

The fundamental aspect of any programming language is its syntax.

### What is syntax?

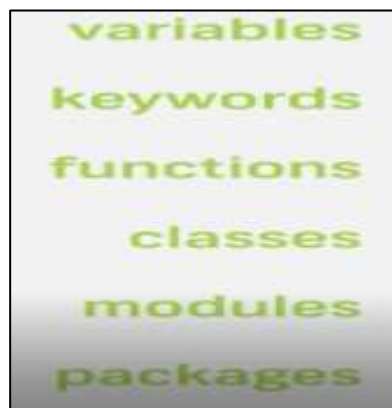
Computers are machines that interpret your instruction only if you type them in exact format that computer expects, this expected format i.e., spelling, formatting and grammar is called syntax.

Basic syntax of python

- **Identifiers**
- **Keywords**
- **Indents**
- **Comments**
- **Variable**

### Identifier:

In a python program, it is common to use programming **elements** such as



These elements have names which are either predefined in the language or can be used by the programmer at the time of writing code. These names are called identifiers.

variables	•————•	<code>_bookname, num1,</code>
keywords	•————•	<code>if, while, yield</code>
functions	•————•	<code>print(), int(), len()</code>
classes	•————•	<code>Complex, Exception</code>
modules	•————•	<code>Calendar, random</code>
packages	•————•	<code>Pillow, tkinter</code>

- Identifier should either start with a letter either lowercase or upper case or with an underscore.

abcdefghijklmnopqrstuvwxyz  
 ABCDEFGHIJKLMNOPQRSTUVWXYZ\_

- No other characters such as

5number  
 \*house  
 \$value  
 )book

Not allowed.

- Rest of identifier may have more than one digit letter, underscore, it doesn't

really matter



book3 ✓  
house\_no ✓

- We cannot use special characters &, %, \$, @, # or symbols in identifier.  
Spot valid and invalid Identifiers

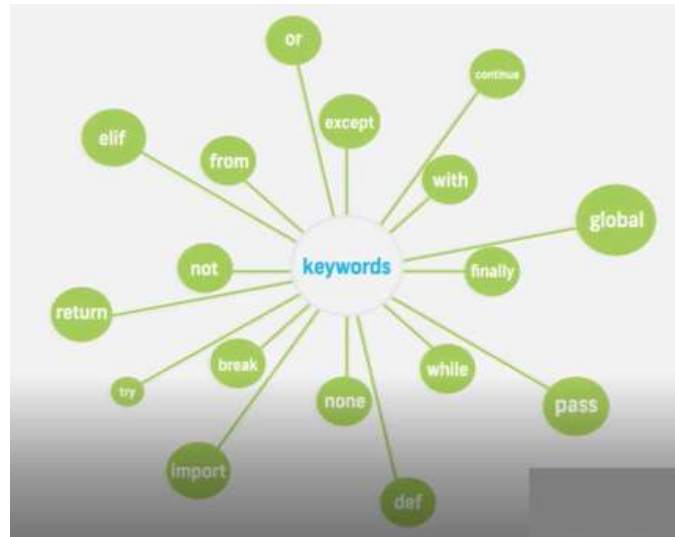


\_house  
BasketNum  
2item  
PENTYPE  
personname  
!return  
\*address\*

Valid	Invalid
_house BasketNum PENTYPE personname	2item !return *address*

### Keywords:

They are always in lowercase. conventionally keywords are also called **reserved words** have predefined meanings and syntax in python. These keywords have to be used to develop programming instructions. They cannot be used as identifiers for other programming elements such as name of variable or functions. Some of reserved keyword in python are



Need not to remember these keywords, If we type import keyword in text editor, we get a list of all keywords.

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', '
def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'retu
rn', 'try', 'while', 'with', 'yield']
>>> |
```

## Indents:

Sometimes more than one statement in the program need to be treated as a block in constructs like class, function, conditions and loops.

When block is to be started, type colon(:) symbol and press enter.

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> num=int(input("Enter number: "))
Enter number: 9
>>> 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")

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



Any python aware editor like IDLE goes to next line leaving additional wide spaces which is indent, subsequent statements in block follows same level of indent. In order to signal the end of the block, the workspace is dedented (This function is **used to remove any common leading whitespace from every line in the input text.**) by pressing the backspace key.

### Comment:

Comment is text that the programmer may add explanation or annotation in the source code. In python symbol” #” indicates of a comment line. Comments are ignored by interpreter while generating machine language code. The purpose of writing comment in source program is only to make easier to other people to better understand the syntax, usage and logic of the algorithm.

# Symbol should be put at starting of a line.

If more than two lines are to be commented, just select all lines and press Alt+3.

A triple coated multiline string is also treated as comment unless it is doc string of a function or class.

```
#Comment 1
#Comment 2

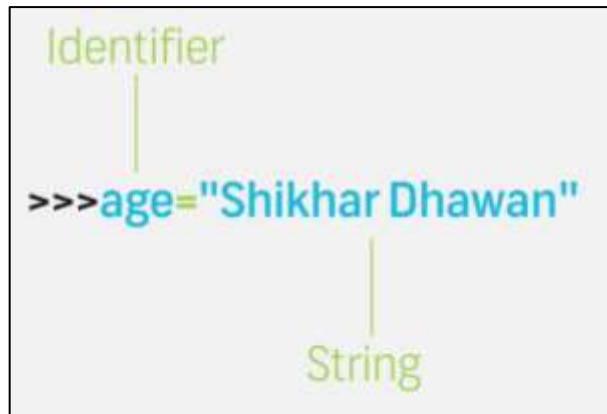
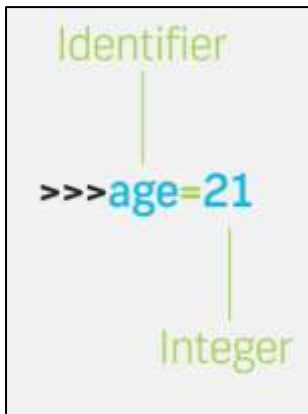
##Comment 3
##Comment 4
##Comment 5

'''This is a multiline
string of
comments.'''
```

### Variables

Data object is stored in randomly chosen location in a computer’s memory. This object is given a suitable identifier or name. Whenever we want to work with stored object, you access it using the name. The name is assigned to an object using the assignment operator ”equal sign”.





In first example, age is an identifier referring to integer 21. However in second example, same identifier is used for different data object i.e, to a string object. So object being referred by the name can change or vary which is the reason, it is called as variable.

Variable is the name given to data object and not to the memory location storing the object.

So Python is called Dynamically Typed Language.

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)] or
Type "copyright", "credits" or "license()" for more information.
>>> name="Shikhar Dhawan"
>>> type(name)
<class 'str'>
>>> name=31
>>> type(name)
<class 'int'>
>>> |
```

```
>>> pokemon="gengar"
>>> type(pokemon)
<class 'str'>
>>> pokemon=2619
>>> type(pokemon)
<class 'int'>
>>> pokemon=34+9j
>>> type(pokemon)
<class 'complex'>
>>> pokemon=(1.5)
>>> type(pokemon)
<class 'float'>
>>> pokemon={"mon":"gengar", "weight": 40.50, "height": 1.50, "MaxCP": 2619}
>>> type(pokemon)
<class 'dict'>
>>>
```

### Rules for naming Variable:

- Name should either start with a letter either lowercase or uppercase or with an underscore.

abcdefghijklmnopqrstuvwxyz  
 ABCDEFGHIJKLMNOPQRSTUVWXYZ\_

- More than one alphanumerical character or underscore may follow

book3

house\_no

- Conventional or starting name with single or double underscore has special meaning in python

Private variable —→ \_aadharum, \_accountnum  
 Strongly private variable —→ \_\_mobilenum, \_\_loginid

Name started with single underscore is treated as private variable and one with double underscore is treated as strongly private variable.

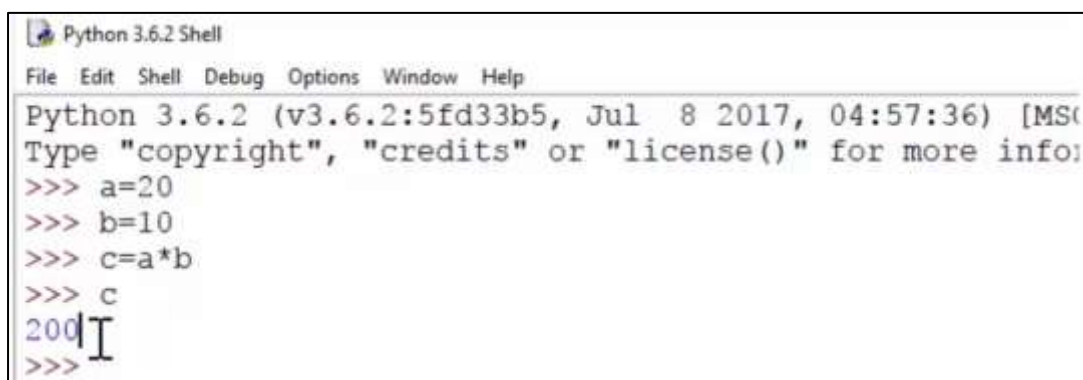
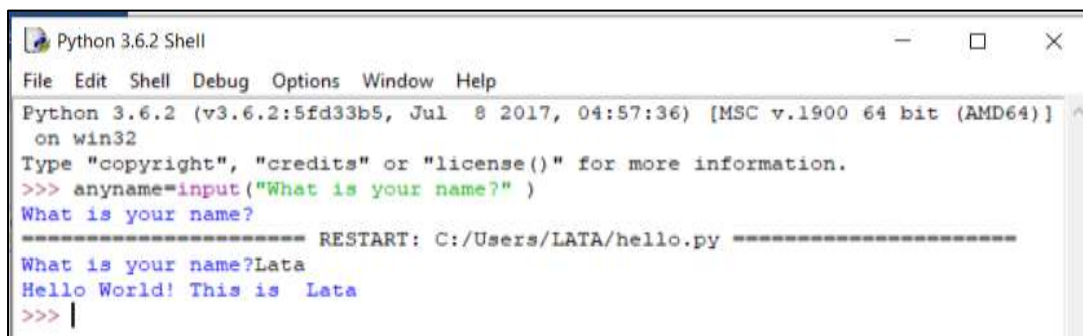
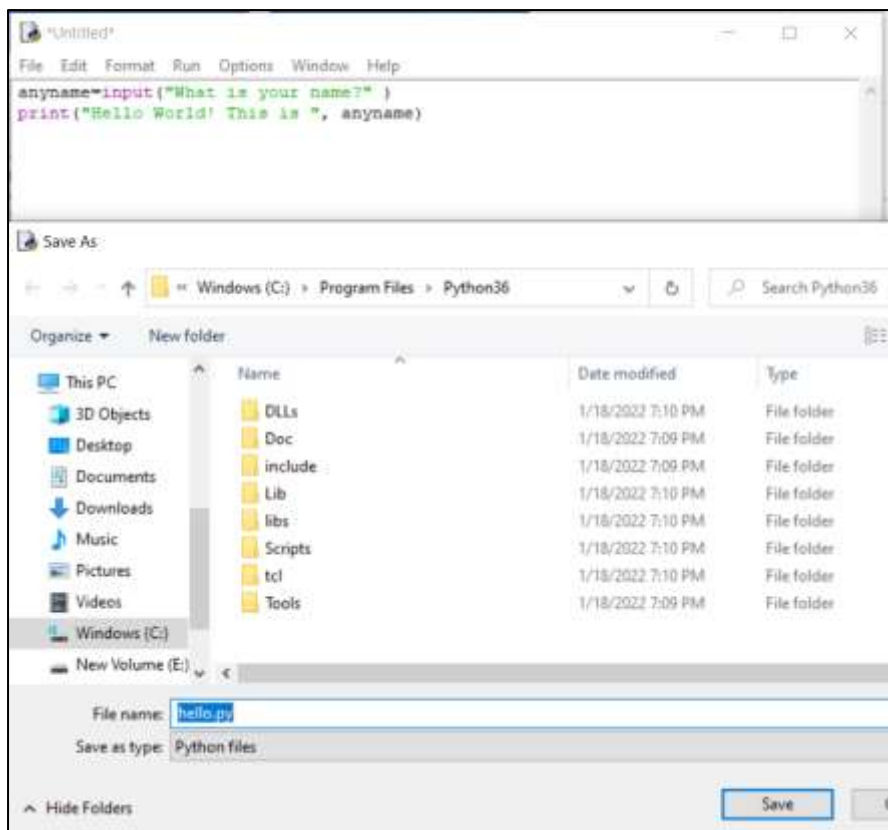
### Introducing Variables:

Let us add some sophistication to our program.

Allow users to add their names to the “Hello World!” message, but to do that we would have to state name some where we can do this by using variables.

```
*helloworld.py - C:\Python36\helloworld.py (3.6.2)*
File Edit Format Run Options Window Help
anyname=input("What is your name?" )
print("Hello World! This is", anyname)
```

More than one comma separated value can appear within brackets of the print function. A non-numeric value is included in inverted commas whereas a variable doesn't have inverted commas.

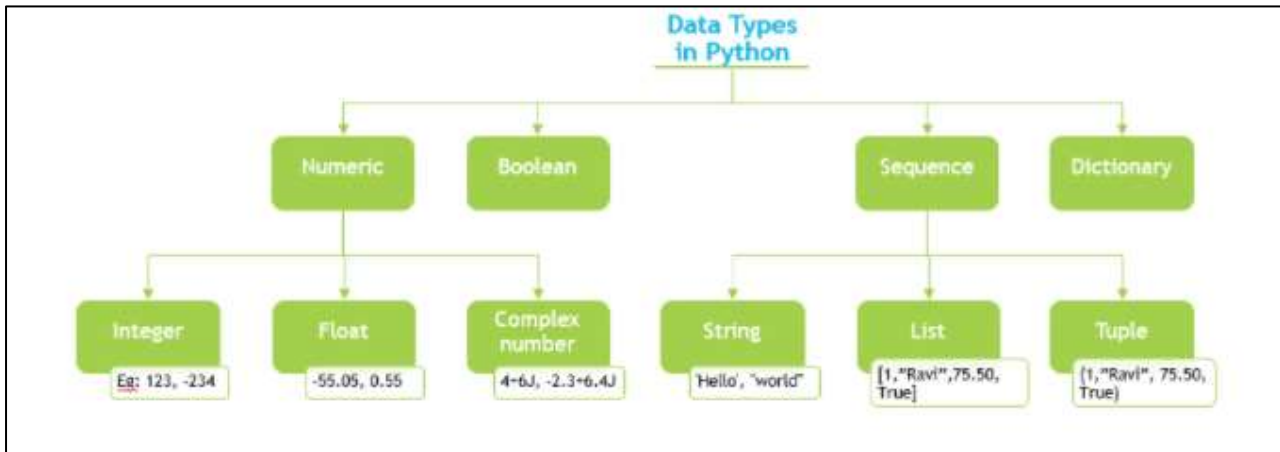


### 1.3 Data Types and Variables:

**Data:** It can take many forms such as numbers, words or sentences.

**Data type:** It is classification of data items.

## Data Types in Python:



<b>Numeric</b>		Any representation of data which has numeric value. Python identifies three types of numbers – integer, float and complex number.
	<b>Integer</b>	<p>Positive and negative whole numbers.</p> <p>Examples: 1234, -234, 0x46 (hexadecimal number), 00123 (octal number))</p> <p>Note: In C and related programming languages such as Python, a hexadecimal number is prefixed with 0x and an octal number is prefixed with 00.</p>
	<b>Float</b>	<p>Real numbers with a floating point representation in which the fractional component is denoted by a decimal or scientific notation</p> <p>Examples: -55.550, 0.005, 1.32E10 (scientific notation))</p>
	<b>Complex number</b>	<p>A number with a real and imaginary component is represented as <math>a + bj</math> in Python where <math>a</math> and <math>b</math> are floats and</p> <p><math>j = \sqrt{-1}</math></p> <p>Examples: <math>4+6j</math>, <math>-2.3+6.4j</math></p> <p>Note: The common mathematical representation of a complex number uses <math>a + bi</math> with <math>i</math> being the imaginary part. But in electronics <math>j</math> is used because <math>i</math> already represent current and the next letter after <math>i</math> is <math>j</math>.</p>

<b>Boolean</b>		Any representation of data which has two values denoted by True and False.
<b>Sequence</b>		An ordered collection of similar or different data types. The built-in Sequence data types in Python are – String, List, and Tuple.
	<b>String</b>	A collection of one or more characters put in single, double or triple quotes.  Examples: 'Hello', "Hello", "Hello", ""Hello""
	<b>List</b>	An ordered collection of one or more data items, not necessarily of the same type, put in square brackets.  Examples: [1,"Ravi",75.50, True]
	<b>Tuple</b>	An ordered collection of one or more data items, not necessarily of the same type put in parentheses. The contents of a tuple cannot be modified – it is immutable - after the tuple is created.  Examples: (1,"Ravi", 75.50, True)  Note: Refer to the Helper Text to learn more about mutability.
<b>Dictionary</b>		An unordered collection of data in key:value pair form. Collection of such pairs is enclosed in curly brackets.  Example: { 1:"Superman", 2:"Wonder Woman", 3:"Thor", 4:"Hulk", 5:"Black Widow" }

## Mutable and Immutable Objects

When a program is run, data objects in the program are stored in the computer's memory for processing. While some of these objects can be modified at that memory location, other data objects can't be modified once they are stored in the memory. *The property of whether or not data objects can be modified in the same memory location where they are stored is called **mutability**.*

We can check the mutability of an object by checking its memory location before and after it is modified. If the memory location remains the same when the data object is modified, it means it is mutable.

To check the memory location of where a data object is stored, we use the function `id()`. Consider the following example (you can try this yourself in IDLE):

1. Assigning values to the list `a`.

```
>>> a=[5, 10, 15]
```

2. Using the function `id()` to get the memory location of `a`.

```
>>> id(a)
```

#### Output

The ID of the memory location where `a` is stored.

```
1906292064
```

3. Replacing the second item in the list, `10` with a new item, `20`.

```
>>> a[1]=20
```

4. Using the `print()` function to verify the new value of `a`.

```
>>> print(a)
```

#### Output

Verified that the value of `a` has changed.

```
[5, 20, 15]
```

5. Using the function `id()` to get the memory location of `a`.

```
>>> id(a)
```

#### Output

The ID of the memory location where `a` is stored.

```
1906292064
```

Notice that the memory location has not changed as the ID remains (`1906292064`) remains the same before and after the variable is modified. This indicates that the list is **mutable**, i.e., it can be modified at the same memory location where it is stored. Now, let us check if a tuple is mutable in Python:

1. Assigning values to the tuple `b`.

```
>>> b=(5, 10, 15)
```

2. Replacing the second item in the list, 10 with a new item, 20.

3. 

```
>>> b[1]=20
```

### Output

Error explaining that a tuple does not support modification in the items – i.e, it is immutable.

Traceback (most recent call last):

```
File "<pyshell#1>", line 1, in <module>
```

```
b[1]=20
```

```
TypeError: 'tuple' object does not support item assignment
```

You can verify the mutability of each of the data types in IDLE.

**Immutable: numeric, string, and tuple**

**Mutable: list, dictionary**

## Code Challenge on Data type:

Python has an in-built function – type() to check the data type.

**Example:** How to use the function type() to check the data type of the number 3368.5:

1. At the Python prompt, enter type (3368.5).
2. Press Enter. The interpreter will return the text, which means 3368.5 is a float object.

### Problem Statement

Now, it's your turn. Verify the data type for each of the objects given below using the type() function:

- 55.50
- 6+4j
- "hello"
- ([1,2,3,4])
- ((1,2,3,4))
- ({1:"one", 2:"two", 3:"three"})



Problem Statement	Ideal Solution
55.50	<pre>&gt;&gt;&gt; type(55.50)</pre> <b>Output</b> <pre>&lt;class 'float'&gt;</pre>
6+4j	<pre>&gt;&gt;&gt; type(6+4j)</pre> <b>Output</b> <pre>&lt;class 'complex'&gt;</pre>
"hello"	<pre>&gt;&gt;&gt; type("hello")</pre> <b>Output</b> <pre>&lt;class 'str'&gt;</pre>
([1,2,3,4])	<pre>&gt;&gt;&gt; type([1,2,3,4])</pre> <b>Output</b> <pre>&lt;class 'list'&gt;</pre>
((1,2,3,4))	<pre>&gt;&gt;&gt; type((1,2,3,4))</pre> <b>Output</b> <pre>&lt;class 'tuple'&gt;</pre>
({1:"one", 2:"two", 3:"three"})	<pre>&gt;&gt;&gt; type({1:"one", 2:"two", 3:"three"})</pre> <b>Output</b> <pre>&lt;class 'dict'&gt;</pre>

### Code Challenge

In the previous topic, you learned how to use the built-in function `type()` to check the data type of a data object. Now, use the same function to check the type of some more data objects, displayed below.

- 8+23j
- 9
- 9.0
- 5.5+3j
- 12+87
- 5.5+4.5
- 3\*\*1/2
- 2+3j+0.5
- .5+.5
- 11

### Numeric data type:

## OBJECTIVES

- Identify the features of the three numeric data types.
- Use basic arithmetic operators with numeric data types.
- Use built-in conversion functions to convert a numeric object of one type into another.

Numeric data type represents what is called Numeric literal, in computer science literal means fixed value in source code.

NumericLiteral

```
x = 10
y = x*2
```

Expression

Python identifies three numeric data type

Integer  
Float  
Complex

Integer data type comprises zero, positive and negative whole numbers.

39 851 0 -92

This data type includes numbers in other basis such as

Binary

```
0110
0011
```

Octal

```
0012
0o30
0o04
```

Hexadecimal

```
0x12
0X21
```

In programming languages such as Python, a hexadecimal number is prefixed with 0x and an octal number is prefixed with 0o.

Float data type represents positive and negative real numbers with fractional part should either with decimal or with the scientific notation.



We use scientific notation to present large numbers with fewer digits so that it is easy to understand large number.

3400000000  
 $3.4 \times 10^8$   
 3.4e8 or 3.4E8

Complex Number: Where x,y are real numbers

$$x+yj$$

$$j = \sqrt{-1}$$

Some examples of complex numbers are

1+2j  
 10-5.5j  
 5.55j  
 2.33j  
 3.11e-6+4j

Problem Statement	Ideal Solution
8+23j	<pre>&gt;&gt;&gt; type(8+23j)</pre> <p><b>Output</b></p> <pre>&lt;class 'complex'&gt;</pre>

9	<pre>&gt;&gt;&gt; type(9)</pre> <b>Output</b> <pre>&lt;class 'int'&gt;</pre>
9.0	<pre>&gt;&gt;&gt; type(9.0)</pre> <b>Output</b> <pre>&lt;class 'float'&gt;</pre>
5.5+3j	<pre>&gt;&gt;&gt; type(5.5+3j)</pre> <b>Output</b> <pre>&lt;class 'complex'&gt;</pre>
12+87	<pre>&gt;&gt;&gt; type(12+87)</pre> <b>Output</b> <pre>&lt;class 'int'&gt;</pre>
5.5+4.5	<pre>&gt;&gt;&gt; type(5.5+4.5)</pre> <b>Output</b> <pre>&lt;class 'float'&gt;</pre>
3**1/2	<pre>&gt;&gt;&gt; type(3**1/2)</pre> <b>Output</b> <pre>&lt;class 'float'&gt;</pre>
2+3j+0.5	<pre>&gt;&gt;&gt; type(2+3j+0.5)</pre> <b>Output</b> <pre>&lt;class 'complex'&gt;</pre>
.5+.5	<pre>&gt;&gt;&gt; type(.5+.5)</pre> <b>Output</b> <pre>&lt;class 'float'&gt;</pre>
11	<pre>&gt;&gt;&gt; type(11)</pre> <b>Output</b> <pre>&lt;class 'int'&gt;</pre>

## 1.4 Python Operators

**Python Operators** in general are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations. In this article, we will look into different types of Python operators.

### Arithmetic Operators

Arithmetic operators are used to performing mathematical operations like addition, subtraction, multiplication, and division.

Operator	Description	Syntax
+	Addition: adds two operands	$x + y$
-	Subtraction: subtracts two operands	$x - y$
*	Multiplication: multiplies two operands	$x * y$
/	Division (float): divides the first operand by the second	$x / y$
//	Division (floor): divides the first operand by the second	$x // y$
%	Modulus: returns the remainder when the first operand is divided by the second	$x \% y$
**	Power: Returns first raised to power second	$x ** y$

### Example: Arithmetic operators in Python

# Examples of Arithmetic Operator

a = 9

b = 4

# Addition of numbers

add = a + b

# Subtraction of numbers

sub = a - b

# Multiplication of number

mul = a \* b

# Division(float) of number

div1 = a / b

# Division(floor) of number

div2 = a // b

# Modulo of both number

mod = a % b

# Power

```
p = a ** b
```

```
# print results
```

```
print(add)
```

```
print(sub)
```

```
print(mul)
```

```
print(div1)
```

```
print(div2)
```

```
print(mod)
```

```
print(p)
```

### Output

```
13
```

```
5
```

```
36
```

```
2.25
```

```
2
```

```
1
```

```
6561
```

### Arithmetic Operators

We have three numeric data types – **integer**, **float**, and **complex**. Let's see how we can use arithmetic operators on these data objects.

Since you are already familiar with arithmetic operations in general, you can quickly learn about how the operators work in Python.

#### Operator for Addition, +

Adds the operands on either side of the operator.

Example 1	Example 2
>>> a=21	>>> a=-21
>>> b=10	>>> b=10
>>> c=a+b	>>> c=a+b

>>> c	>>> c
31	-11

### Operator for Subtraction, -

Subtracts the operand on the right from the operand on the left.

Example 1	Example 2
>>> a=21	>>> a=-21
>>> b=10	>>> b=10
>>> c=a-b	>>> c=a-b
>>> c	>>> c
11	-31

### Operator for Multiplication, \*

Multiplies values on either side of the operator.

Example 1	Example 2
>>> a=21	>>> a= -21
>>> b=10	>>> b=10
>>> c=a*b	>>> c=a*b
>>> c	>>> c
210	-210

### Operator for Division, /

Divides left hand operand by right hand operand.

Example 1	Example 2
>>> a=21	>>> a= -21
>>> b=10	>>> b=10
>>> c=a/b	>>> c=a/b
>>> c	>>> c
2.1	-2.1

Example 1	Example 2
>>> a=21	>>> a= -21
>>> b=10	>>> b=10
>>> c=a/b	>>> c=a/b
>>> c	>>> c
2.1	-2.1

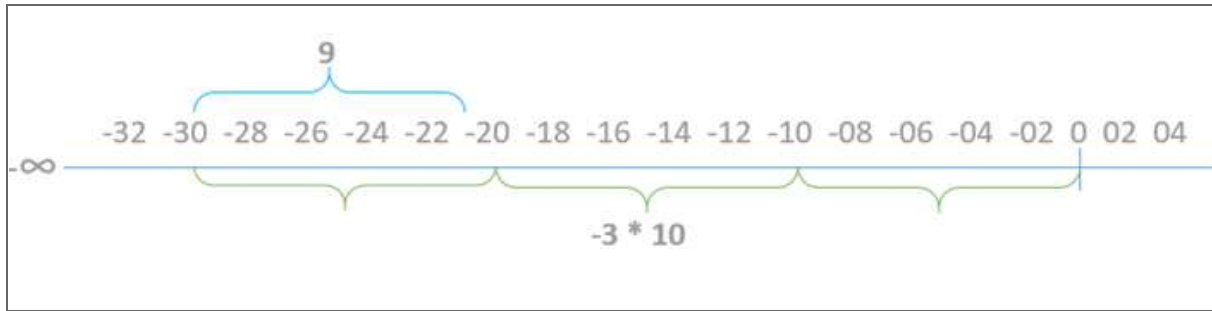
### Operator for Modulus, %

Returns the remainder of division of the operand on the left by the operand on the right of the operator

Example 1	Example 2
>>> a=21	>>> a= -21
>>> b=10	>>> b=10
>>> c=a%b	>>> c=a%b
>>> c	>>> c
1	9

Note: In Python, the modulus is calculated towards negative infinity. Thus, 21%10 is 1 because the closest multiple of 10 towards negative infinity is 20. But -21%10 is 9 because the closest multiple of 10 towards negative infinity is -30 and -30+9 is -21.





### Operator for Exponent, \*\*

Calculates the value of the operand on the left raised to operand on the right of the operator.

#### Example 1

```
>>> a=4
>>> b=3
>>> c=a**b
>>> c
64
```

#### Example 2

```
>>> a=-4
>>> b=3
>>> c=a**b
>>> c
-64
```

### Operator for Floor Division, //

Returns the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity).

#### Example 1

```
>>> a=9
>>> b=5
>>> c=a//b
>>> c
1
```

**Explanation:**  $9/5 = 1.8$ . So the floor division result removes the 8 and returns 1.

#### Example 2

```
>>> a=-9
>>> b=5
>>> c=a//b
>>> c
-2
```

**Explanation:**  $-9/5 = -1.8$ . However, the floor division rounds this away from 0 and makes it -2.

### Arithmetic Operations on Complex Numbers

A complex number – also known as an imaginary number – is defined as the square root of (-1) and is denoted by **J** or **j**. A complex number is represented as  $x+yj$ . Both  $x$  and  $y$  are real numbers.  $y$  multiplied by the imaginary number forms the imaginary part of a complex number.

You can use arithmetic operators on complex numbers in the same way as you would for integer or float data types. The output is governed by rules of mathematics for complex numbers.

Try each of the following examples in IDLE and check the result for yourself.

### Addition and Subtraction

The addition or subtraction of complex numbers involves the addition or subtraction of their corresponding real and imaginary parts.

#### Addition

```
>>> a=6+4j
```

```
>>> b=3+2j
```

```
>>>a+b
```

#### Output

```
(9+6j)
```

#### Subtraction

```
>>> a-b
```

#### Output

```
(3+2j)
```

### Multiplication

The multiplication of two complex numbers is very similar to multiplication of two binomials. Consider the following example:

```
a=6+4j
```

```
b=3+2j
```

```
c=a*b
```

```
c=(6+4j)*(3+2j)
```

$$c=(18+12j+12j+8*-1)$$

$$c=10+24j$$

You can verify this result in IDLE:

```
>>> a=6+4j
```

```
>>> b=3+2j
```

```
>>> a*b
```

### Output

```
(10+24j)
```

### Division

The division of two complex numbers involves multiplying both sides by the conjugate of the denominator, which is a number with the same real part and the opposite imaginary part. Consider the following example:

$$a=6+4j$$

$$b=3+2j$$

$$c=a/b$$

$$c=(6+4j)*(3-2j)/(3+2j)(3-2j)$$

$$c=(18-12j+12j-8*-1)/(9-6j+6j-4*-1)$$

$$c=26/13$$

$$c=2+0j$$

You can verify this result in IDLE:

```
>>> a=6+4j
```

```
>>> b=3+2j
```

```
>>> a/b
```

## Output

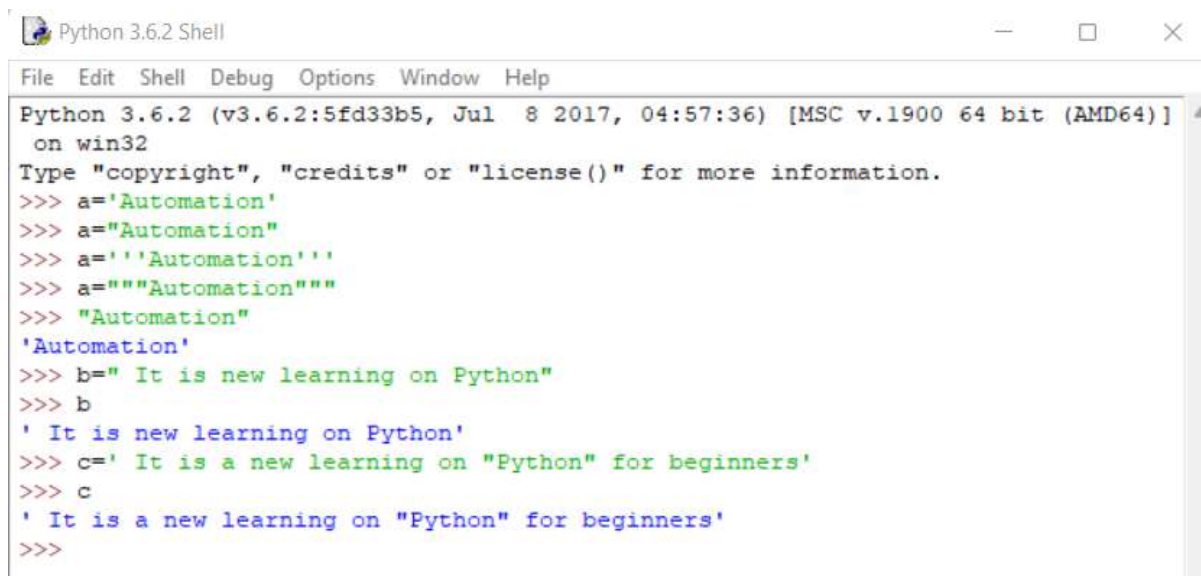
(2+0j)

## String Data operators:

### OBJECTIVES

- Create strings in Python using single, double and triple quotes.
- Access characters in a Python string.
- Use String operators in Python.
- Format strings.
- Use string functions to process strings.

String object is one of the sequence data types. It is an immutable sequence of characters. You can easily create a string. Simply enclose a character within quotes. You may use single quotes or double quotes or triple quotes to enclose string. **Single and double quotes can be used as interchangeably. You can choose between the two depending on the whether your string itself has any quotes within it.** Let us say you have a string with apostrophe, then you would use double quotes to enclose it. If your string has double quotes, you would enclose it in single quotes.



```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> a='Automation'
>>> a="Automation"
>>> a='''Automation'''
>>> a="""Automation"""
>>> "Automation"
'Automation'
>>> b=" It is new learning on Python"
>>> b
' It is new learning on Python'
>>> c=' It is a new learning on "Python" for beginners'
>>> c
' It is a new learning on "Python" for beginners'
>>>
```

Triple quotes are useful when you need to use a multiline string. Let us say you want to store lyrics of a song.

```
>>> intheend="""One thing, I don't know why
It doesn't even matter how hard you try
Keep that in mind
I designed this rhyme to explain in due time
All I know
Time is a valuable thing
Watch it fly by as the pendulum swings
Watch it count down to the end of the day
The clock ticks life away
It's so unreal"""
>>> intheend
"One thing, I don't know why\nIt doesn't even matter how hard you try
xplain in due time\nAll I know\nTime is a valuable thing\nWatch it fl
n to the end of the day\nThe clock ticks life away\nIt's so unreal"
>>>
```

We can print multiline string using a print function.

```
>>> print(intheend)
One thing, I don't know why
It doesn't even matter how hard you try
Keep that in mind
I designed this rhyme to explain in due time
All I know
Time is a valuable thing
Watch it fly by as the pendulum swings
Watch it count down to the end of the day
The clock ticks life away
It's so unreal
>>>
```

**Accessing characters in a string:**

a	u	t	o	m	a	t	i	o	n	!
0	1	2	3	4	5	6	7	8	9	10

A string is an ordered set of characters. It means each of the character has an index within the first character having an index of zero.

```

Create a string
>>>a='Automation'

Access Characters
>>>a =[4]

>>>m

```

```

>>> orgname = 'Automation '
>>> orgname [3]
'o '
>>>

```

## Formatting Strings using Escape Sequences

You can use two or more specially designated characters within a string to format a string or perform a command. These characters are called **escape sequences**. An escape sequence in Python starts with a backslash (\). For example, \n is an escape sequence in which the common meaning of the letter n is literally escaped and given an alternative meaning - a new line.

Displayed here are a few common escape sequences available in Python. You can try these out in IDLE or the Python prompt from the windows command prompt.

Escape sequence	Description	Example	Result
\n	Breaks the string into a new line	>>> print('I designed this rhyme to explain in due time\nAll I know')	I designed this rhyme to explain in due time All I know
\t	Adds a horizontal tab	>>> print('Time is a \tvaluable thing')	Time is a valuable thing
\\	Prints a backslash	>>> print('Watch it fly by\\ as the pendulum swings')	Watch it fly by\ as the pendulum swings
\'	Prints a single quote	>>> print('It doesn\'t even matter how hard you try')	It doesn't even matter how hard you try
\"	Prints a double quote	>>> print('It is so \"unreal\" ')	It is so "unreal"
\a	makes a sound like a bell	>>> print('\a')	A sound effect is heard. <b>Note:</b> /a may not produce a sound as it depends on the system settings.

## String Operators

You can use certain operators for string processing.

- **+ Concatenation**
- **\* Repetition**
- **[] Slice**
- **[:] Range Slice**
- **in Membership**
- **not in Membership**

#### **+Concatenation**

**[+] Appends the second string to the first.**

#### **Example**

```
>>> a="Hello, ' #String a
>>> b='Nick",' #String b
>>> print(a+b+ ' said Harry to Nearly Headless Nick.') #The + here appends the strings a and b to the
string in quotes.
"Hello, Nick", said Harry to Nearly Headless Nick.
>>>
```

#### **\* Repetition**

**[\*] Concatenates multiple copies of the same string.**

#### **Example**

```
>>> a='I must not tell lies...' # String a
>>> print('Again and again Harry wrote the words on the parchment in his own blood – '+a*5) # Here,
the * joins or concatenates the string a repeatedly for 5 times.
Again and again Harry wrote the words on the parchment in his own blood – I must not tell lies...I
must not tell lies...I must not tell lies...I must not tell lies...I must not tell lies...
>>>
```

#### **[] Slice**

**[] Gives the character at given index.**

#### **Example**

T	H	e		B	u	r	r	o	w
0	1	2	3	4	5	6	7	8	9

```
>>> a='The Burrow' #String a
```

```
>>> a[8] #Here the index number in square brackets [ ] slices out the character at that index, which is
o in this example.
```

```
'o'
```

```
>>>
```

### [:] Range Slice

**[ : ] Fetches characters in the range specified by two index operands separated by a colon.**

**If the first operand is omitted, the range starts at zero index.**

**If the second operand is omitted, the range goes up to the end of the string.**

Note: The slice starts at the first index. The slice ends one index before the second index, that is at the value of the index - 1.

### Example

T	H	e		B	u	r	r	o	w
0	1	2	3	4	5	6	7	8	9

```
>>> a='The Burrow' #String a
```

```
>>> a[2:7] #Starting index = 2 = e, Ending index = 7-1 = r
```

```
'e Bur'
```

```
>>> a[:6] #Starting index = 0 = T, Ending index = 6-1 = u
```

```
'The Bu'
```

```
>>> a[5:] #Starting index = 5 = u, Ending index = end of string = w
```

```
'urrow'
```



```
>>>
```

### in Membership

**[in]** Returns true if a character exists in the given string.

#### Example

```
>>> a='Harry watched Dumbledore striding up and down in front of him, and thought. He thought of
his mother, his father and Sirius. He thought of Cedric Diggory.' #String a

>>> 'v' in a #Checks if the character 'v' is present in the string, a
False

>>> 'dig' in a #Checks if the characters 'dig' are present in the string,
a
False

>>> 'Dig' in a #Note that this is case-sensitive. As 'Dig' is present in
'Diggory', this returns True.
True

>>>
```

### not in Membership

**[not in]** Returns true if a character does not exist in the given string.

#### Example

```
>>> a='' "For HIM?" shouted Snape. "Expecto Patronum!"

From the tip of his wand burst the silver doe: She landed on the office
floor, bounded once across the office, and soared out of the window.
Dumbledore watched her fly away, and as her silvery glow faded he turned
back to Snape, and his eyes were full of tears.

"After all this time?"
```

```

"Always," said Snape. ''' #Multi-line string, a

>>> 'v' not in a #Checks that the character 'v' is not present in the
string, a

False

>>> 'Red' not in a #Checks that the characters 'Red' iare not present in
the string, a

True

>>> 'red' notin a #This is case sensitive. Since 'red' is present in
'soared', this returns False.

False

>>>

```

## 1. Format Specification Symbols

Please go through the format symbols and their specifications below:

Format Symbol	Conversion	Examples
%c	character	<pre>&gt;&gt;&gt; print("The letter that comes after d is %c" % ('e',))</pre> <p><b>Output</b></p> <p>The letter that comes after d is e</p>
%s	string conversion via str() prior to formatting	<p><b>Example 1</b></p> <pre>&gt;&gt;&gt; balltype='basketball' &gt;&gt;&gt; result='hit' &gt;&gt;&gt; print('I wondered why the %s was getting bigger. Then it %s me.' % (balltype, result))</pre> <p><b>Output</b></p> <p>I wondered why the basketball was getting bigger. Then it hit me.</p>

		<p><b>Example 2</b></p> <pre>&gt;&gt;&gt; print("%20s" % ('Robotics', ))</pre> <p><b>Output</b></p> <p>Robotics</p> <p><b>Example 3</b></p> <pre>&gt;&gt;&gt; print("%-20s" % ('Robotics', ))</pre> <p><b>Output</b></p> <p>Robotics</p> <p><b>Example 4</b></p> <pre>&gt;&gt;&gt; print("%.5s" % ('Robotics', ))</pre> <p><b>Output</b></p> <p>Robot</p>
%i	signed decimal integer	%i is identical to %d
%d	signed decimal integer	<pre>&gt;&gt;&gt; match=12553 &gt;&gt;&gt; site='eBay' &gt;&gt;&gt; print("%s is so useless. I tried to look up lighters and all they had was %d matches." % (site, match))</pre> <p><b>Output</b></p> <p>eBay is so useless. I tried to look up lighters and all they had was 12553 matches.</p>
%u	unsigned decimal integer	%u it is an obsolete type and is identical to %d

%o	octal integer	<pre>&gt;&gt;&gt; print("83 is represented in octal as %o" % (0o123,))</pre> <p><b>Output</b></p> <p>83 is represented in octal as 123</p>
%x	hexadecimal integer	<pre>&gt;&gt;&gt; print("83 is represented in hexadecimal as %x" % (0x53,))</pre> <p><b>Output</b></p> <p>83 is represented in hexadecimal as 53</p>

### Formatting Strings Using the format() Method

With Python 3.0, the format() method has been introduced for handling complex string formatting more efficiently. This method of the built-in string class provides functionality for complex variable substitutions and value formatting. This new formatting technique is regarded as more elegant.

The general syntax of the format() method is: **string.format(var1, var2,...)**

The string itself contains placeholders { } in which values of variables are successively inserted.

```
>>> name="Malhar"
>>> age=23
>>> percentage=55.5
>>> "my name is { } and my age is { } years".format(name, age)
'my name is Malhar and my age is 23 years'
>>>
```

You can also specify formatting symbols. Only change is using colon (:) instead of %. For example, instead of %s use { :s } and instead of %d use { :d }

```
>>> "my name is { :s } and my age is { :d } years".format(name, age)
'my name is Malhar and my age is 23 years'
>>>
```

Precision formatting of numbers can be accordingly done.

```
>>> "my name is {:s}, age {:d} and I have scored {:.3f} percent
marks".format(name, age, percentage)

'my name is Malhar, age 23 and I have scored 55.500 percent marks'

>>>
```

## **Methods for String Processing**

To learn about other methods that can be used to process strings,

### **capitalize()**

**capitalize():** This method converts the first character of a string to uppercase letter.

```
>>> var='robotics'

>>> var.capitalize()

'Robotics'

>>>
```

### **upper()**

**upper():** This method returns a string with lowercase characters replaced by corresponding uppercase characters.

```
>>> var='robotics'

>>> var.upper()

'ROBOTICS'

>>>
```

### **lower()**

**lower():** This method results in a string with uppercase characters replaced by corresponding lowercase characters.

```
>>> var='ROBOTICS'

>>> var.lower()

'robotics'

>>>
```

**title()**

**title():** This method results in a string with the first character of each word converted to uppercase.

```
>>> var='python training from v.e.s.p'
>>> var.title()
'Python Training From V.e.s.p'
>>>
```

**find()**

**find():** This method finds the first occurrence of a substring in another string. If not found, the method returns -1.

```
>>> var='python training from v.e.s.p'
>>> var.find('in')
10
>>> var.find('on')
4
>>> var.find('run')
-1
>>>
```

The substring 'in' first occurs at the 10<sup>th</sup> position (count starts from 0), 'on' is found at the 4<sup>th</sup> position, but 'run' is not found hence returns -1

**index()**

**index():** This method is similar to find() but throws a ValueError if the substring is not found.

```
>>> var='python training from v.e.s.p'
>>> var.index('in')
10
```

```
>>> var.index('run')
```

Traceback (most recent call last):

File "<pyshell#19>", line 1, in <module>

```
    var.index('run')
```

ValueError: substring not found

```
>>>
```

### **count()**

**count():** This method returns the number of occurrences of a substring in given string.

```
>>> var='python training from v.e.s.p'
```

```
>>> var.count('in')
```

```
2
```

```
>>>
```

### **isalpha()**

**isalpha():** This method returns true if all the characters in a string are alphabetic (a-z or A-Z), otherwise returns false.

```
>>> var='Robotics'
```

```
>>> var.isalpha()
```

```
True
```

```
>>> var='Robo tics'
```

```
>>> var.isalpha()
```

```
False
```

```
>>>
```

### **isdigit()**

**isdigit():** This method returns true if all characters in a string are digits( 0-9), if not returns false.

```
>>> var='2000'
```

```
>>> var.isdigit()

True

>>> var='2,000'

>>> var.isdigit()

False

>>>
```

### **islower()**

**islower():** This method returns true if all characters in a string are lowercase characters else returns false.

```
>>> var='robotics'

>>> var.islower()

True

>>> var='Robotics'

>>> var.islower()

False

>>> var='robo tics'

>>> var.islower()

True

>>>
```

### **isupper()**

**isupper():** This method returns true if all characters in a string are uppercase characters else returns false.

```
>>> var='ROBO TICS'

>>> var.isupper()

True

>>> var='ROBOtics'
```



```
>>> var.isupper()

False

>>> var='ROBO+TICS'

>>> var.isupper()

True

>>> var='1234'

>>> var.isupper()

False

>>>
```

### Comparison Operators (Relational operators)

Comparison of Relational operators compares the values. It either returns **True** or **False** according to the condition.

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	$x > y$
<	Less than: True if the 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 the left operand is greater than or equal to the right	$x >= y$
<=	Less than or equal to True if the left operand is less than or equal to the right	$x <= y$

Example: Comparison Operators in Python

```
# Examples of Relational Operators
```

```
a = 13
```

```
b = 33
```

```
# a > b is False
```

```
print(a > b)
```

```
# a < b is True
```

```
print(a < b)
```

```
# a == b is False  
print(a == b)
```

```
# a != b is True  
print(a != b)
```

```
# a >= b is False  
print(a >= b)
```

```
# a <= b is True  
print(a <= b)
```

## Output

False

True

False

True

False

True

## Logical Operators

Logical operators perform **Logical AND**, **Logical OR**, and **Logical NOT** operations. It is used to combine conditional statements.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

## Example: Logical Operators in Python

```
# Examples of Logical Operator
```

```
a = True
```

```
b = False
```

```
# Print a and b is False
```

```
print(a and b)
```

```
# Print a or b is True
```

```
print(a or b)
```

```
# Print not a is False
```

```
print(not a)
```

### Output

False

True

False

### Bitwise Operators

Bitwise operators act on bits and perform the bit-by-bit operations. These are used to operate on binary numbers.

Operator	Description	Syntax
&	Bitwise AND	x & y
	Bitwise OR	x   y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

### Example: Bitwise Operators in Python

```
# Examples of Bitwise operators
```

```
a = 10
```

```
b = 4
```

```
# Print bitwise AND operation
```

```
print(a & b)
```

```
# Print bitwise OR operation
```

```
print(a | b)
```

# Print bitwise NOT operation

```
print(~a)
```

# print bitwise XOR operation

```
print(a ^ b)
```

# print bitwise right shift operation

```
print(a >> 2)
```

# print bitwise left shift operation

```
print(a << 2)
```

### Output

0

14

-11

14

2

40

### Assignment Operators

Assignment operators are used to assigning values to the variables.

Operator	Description	Syntax
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right-side operand with left side operand and then assign to left operand	a+=b    a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b    a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b    a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b    a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign the result to left operand	a%=b    a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b    a=a//b

<code>**=</code>	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	<code>a**=b</code> <code>a=a**b</code>
<code>&amp;=</code>	Performs Bitwise AND on operands and assign value to left operand	<code>a&amp;=b</code> <code>a=a&amp;b</code>
<code> =</code>	Performs Bitwise OR on operands and assign value to left operand	<code>a =b</code> <code>a=a b</code>
<code>^=</code>	Performs Bitwise xOR on operands and assign value to left operand	<code>a^=b</code> <code>a=a^b</code>
<code>&gt;&gt;=</code>	Performs Bitwise right shift on operands and assign value to left operand	<code>a&gt;&gt;=b</code> <code>a=a&gt;&gt;b</code>
<code>&lt;&lt;=</code>	Performs Bitwise left shift on operands and assign value to left operand	<code>a &lt;&lt;= b</code> <code>a= a &lt;&lt; b</code>

### Example: Assignment Operators in Python

# Examples of Assignment Operators

```
a = 10
```

# Assign value

```
b = a
```

```
print(b)
```

# Add and assign value

```
b += a
```

```
print(b)
```

# Subtract and assign value

```
b -= a
```

```
print(b)
```

# multiply and assign

```
b *= a
```

```
print(b)
```

# bitwise lishift operator

```
b <<= a
```

```
print(b)
```

**Output**

```
10
20
10
100
102400
```

**Identity Operators**

**is** and **is not** are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

**is**            True if the operands are identical

**is not**      True if the operands are not identical

**Example: Identity Operator**

```
a = 10
```

```
b = 20
```

```
c = a
```

```
print(a is not b)
```

```
print(a is c)
```

**Output**

```
True
```

```
True
```

**Membership Operators**

**in** and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

**in**            True if value is found in the sequence

**not in**      True if value is not found in the sequence

**Example: Membership Operator**

```
# Python program to illustrate
```

```
# not 'in' operator
```

```
x = 24
```

```
y = 20
```

```
list = [10, 20, 30, 40, 50]
```

```
if (x not in list):
```

```
    print("x is NOT present in given list")
```

```
else:
```

```
    print("x is present in given list")
```

```
if (y in list):
```

```
    print("y is present in given list")
```

```
else:
```

```
    print("y is NOT present in given list")
```

### **Output**

x is NOT present in given list

y is present in given list

## **1.5 Conditional statement**

### **Program:**

Program is set of instructions given to computer for performing a certain task.

```

# This function adds two numbers
def add(x, y):
    return x + y

# This function subtracts two numbers
def subtract(x, y):
    return x - y

# This function multiplies two numbers
def multiply(x, y):
    return x * y

# This function divides two numbers
def divide(x, y):
    return x / y

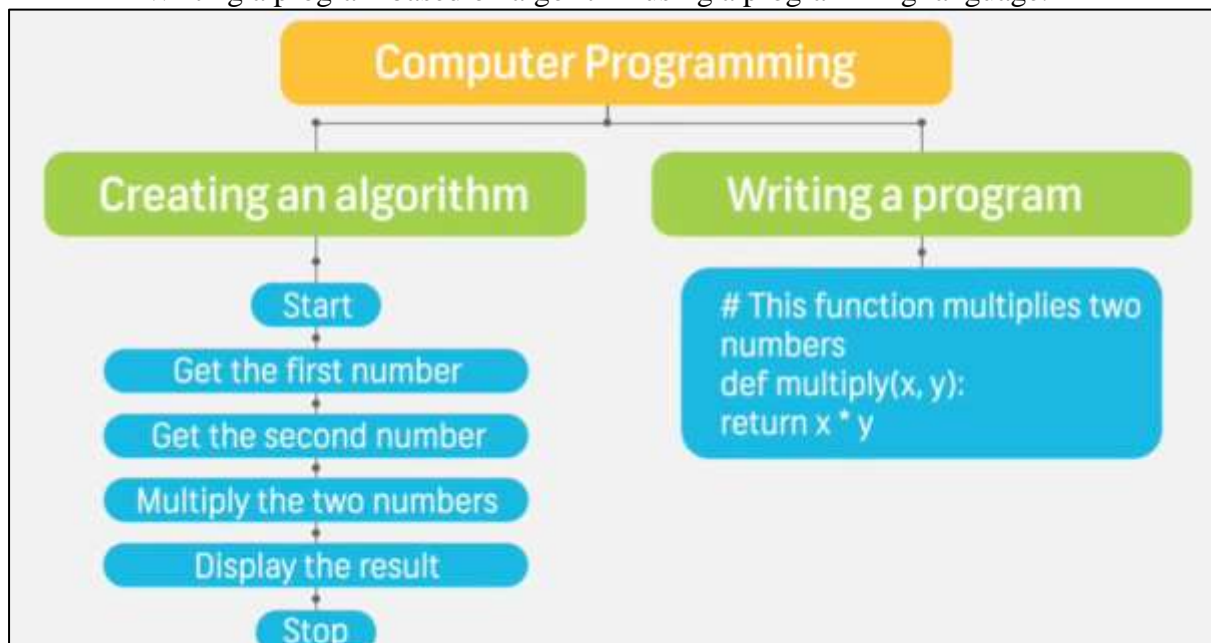
```

COMPUTER PROGRAM

## Computer programming:

Computer programming involves two basic steps.

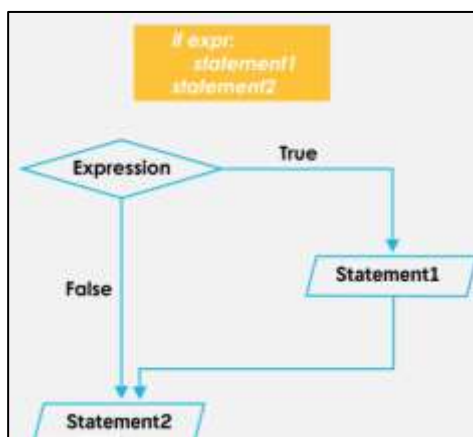
- Planning an algorithm for a task.
- Writing a program based on algorithm using a programming language.







## Introduction to 'If' statement



The If statement is the most fundamental decision-making statement, in which the code is executed based on whether it meets the specified condition. It has a code body that only executes if the condition in the if statement is true. The statement can be a single line or a block of code.

The if statement in Python has the subsequent syntax:

```
if expression
Statement
```

#If the condition is true, the statement will be executed.

Examples for better understanding:

Example – 1

```
num = 5
if num > 0:
    print(num, "is a positive number.")
print("This statement is true.")
```

#When we run the program, the output will be:

5 is a positive number.

This statement is true.

Example – 2

```
a = 25
```

```
b = 170
```

```
if b > a:
```

```
    print("b is greater than a")
```

```
output : b is greater than a
```

### Sample Program:

```
Marks = 33
```

```
if marks >=30:
```

```
    print ("Passed")
```

### Output:

#### Passed

Here, the marks are 33 which justify our condition saying, if marks are greater than or equal to 30, it would show 'passed'.

In case, the 'If' condition is not true, it won't execute the program and error would pop-up. For example: In the case of numbers less than 30, it won't execute.

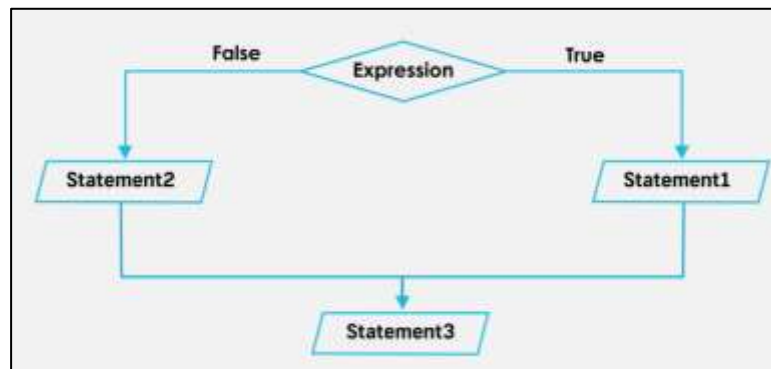
### Key Points:

1. The colon ( : ) at the end of each condition is required indicating that the code block is following directly afterward and thus help to read the program easily.
2. The statements must be indented and would execute only if the condition is TRUE else not.
3. The condition can be used with bracket '( ' )' as well.

### Example:

```
If (marks > = 30):
```

## If Else Statement



This statement is used when both the true and false parts of a given condition are specified to be executed. When the condition is true, the statement inside the if block is executed; if the condition is false, the statement outside the if block is executed.

The if...Else statement in Python has the following syntax:

```

if condition :
    #Will executes this block if the condition is true
else :
    #Will executes this block if the condition is false
  
```

Example for better understanding:

```

num = 5
if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
output : Positive or Zero
  
```

## If...Elif..else Statement

In this case, the If condition is evaluated first. If it is false, the Elif statement will be executed; if it also comes false, the Else statement will be executed.

The If...Elif..else statement in Python has the subsequent syntax:

```
if condition :  
    Body of if  
elif condition :  
    Body of elif  
else:  
    Body of else
```

Example for better understanding:

We will check if the number is positive, negative, or zero.

```
num = 7  
if num > 0:  
    print("Positive number")  
elif num == 0:  
    print("Zero")  
else:  
    print("Negative number")  
output: Positive number
```

## **Nested IF Statement**

A Nested IF statement is one in which an If statement is nestled inside another If statement. This is used when a variable must be processed more than once. If, If-else, and If...elif...else statements can be used in the program. In Nested If statements, the indentation (whitespace at the beginning) to determine the scope of each statement should take precedence.

The Nested if statement in Python has the following syntax:

```
if (condition1):  
    #Executes if condition 1 is true  
if (condition 2):  
    #Executes if condition 2 is true  
    #Condition 2 ends here  
#Condition 1 ends here
```

Examples for better understanding:

#### Example-1

```
num = 8
if num >= 0:
    if num == 0:
        print("zero")
    else:
        print("Positive number")
else:
    print("Negative number")
output: Positive number
```

#### Example-2

```
price=100
quantity=10
amount = price*quantity
if amount > 200:
    if amount >1000:
        print("The amount is greater than 1000")
    else:
        if amount > 800:
            print("The amount is between 800 and 1000")
        elif amount > 600:
            print("The amount is between 600 and 1000")
        else:
            print("The amount is between 400 and 1000")
elif amount == 200:
    print("Amount is 200")
else:
    print("Amount is less than 200")
```

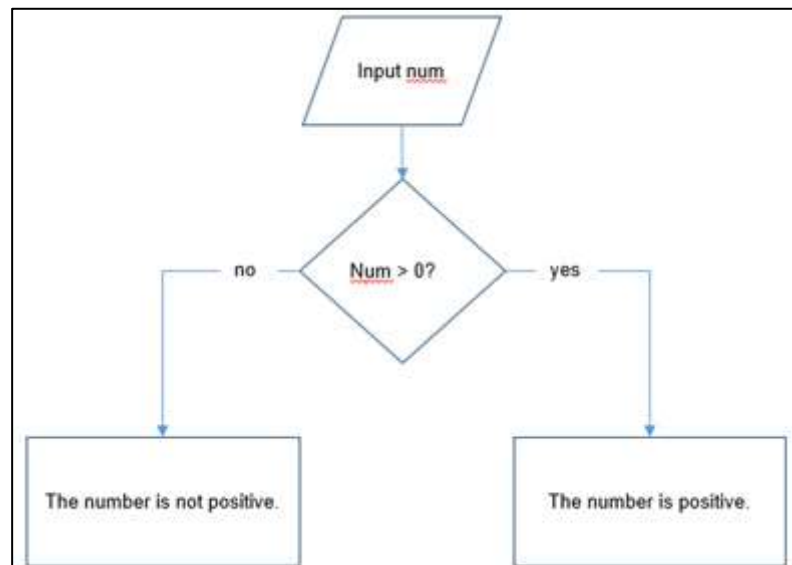
The output : "The amount is between 400 and 1000."

## Exercise

### Practice programs using if, if-else, and elif

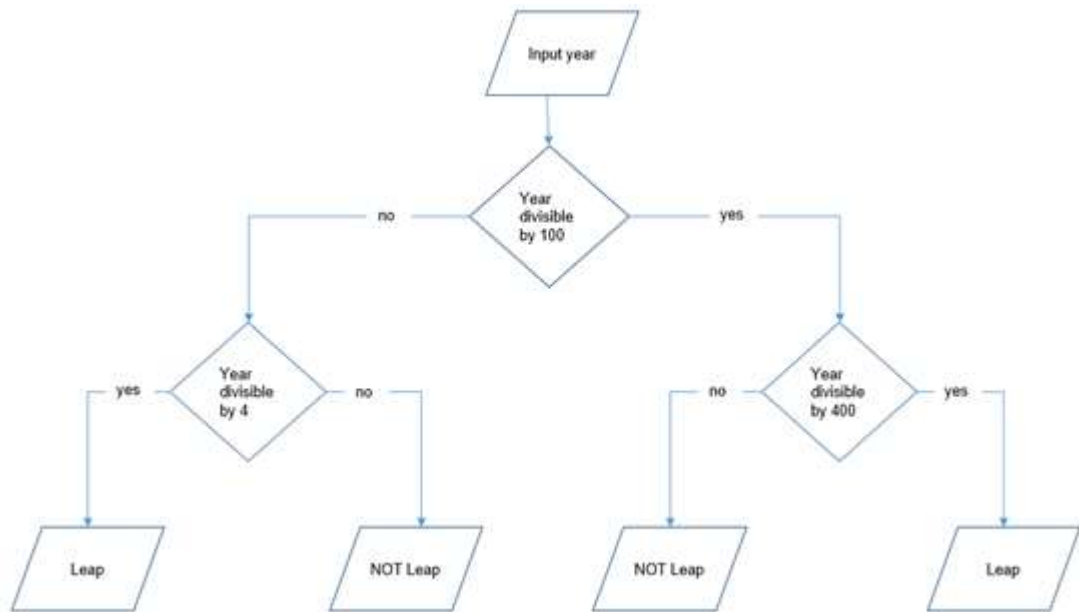
1. Write a program to check if a number is positive and print the message, "This is a positive number." if it is a positive number.

**Hint:**



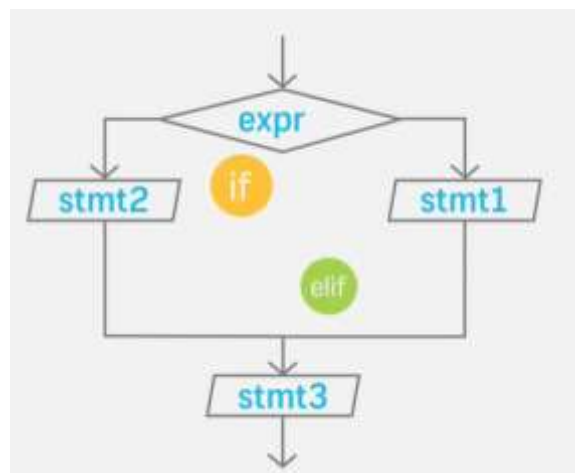
2. Check if a year is a leap year and print the message, "This is a Leap Year." if it is a leap year.

**Hint:** A leap year is a leap year if it is evenly divisible by 4 but not evenly divisible by 100 unless it is also evenly divisible by 400.

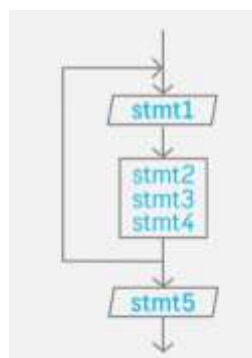


## 1.6 Loopings

Loop is a popular phrase, program flow may skip from statement with the use of statements such as if and elif.



If the program flow is redirected towards any of earlier statements in the program, it is known as loop



In the diagram, you would have expected statement 5 to be executed after statement 4, however the controller of the program is redirected back to the previous statement. So statements 1 to 4 will be

repeated in a loop. Unless we specify some condition to stop loop, it will continue loop infinitely. We do this in python using while and for keywords. These keywords constitute a conditional loop specifically a block of statements until a Boolean expression is true or not true

The three types of loops in Python programming are:

1. while loop
2. for loop
3. nested loops

## **While Loop:**

It continually executes the statements(code) as long as the given condition is TRUE. It first checks the condition and then jumps into the instructions. In python, while loop is used to execute a block of statements repeatedly until a given a condition is satisfied. And when the condition becomes false, the line immediately after the loop in program is executed.

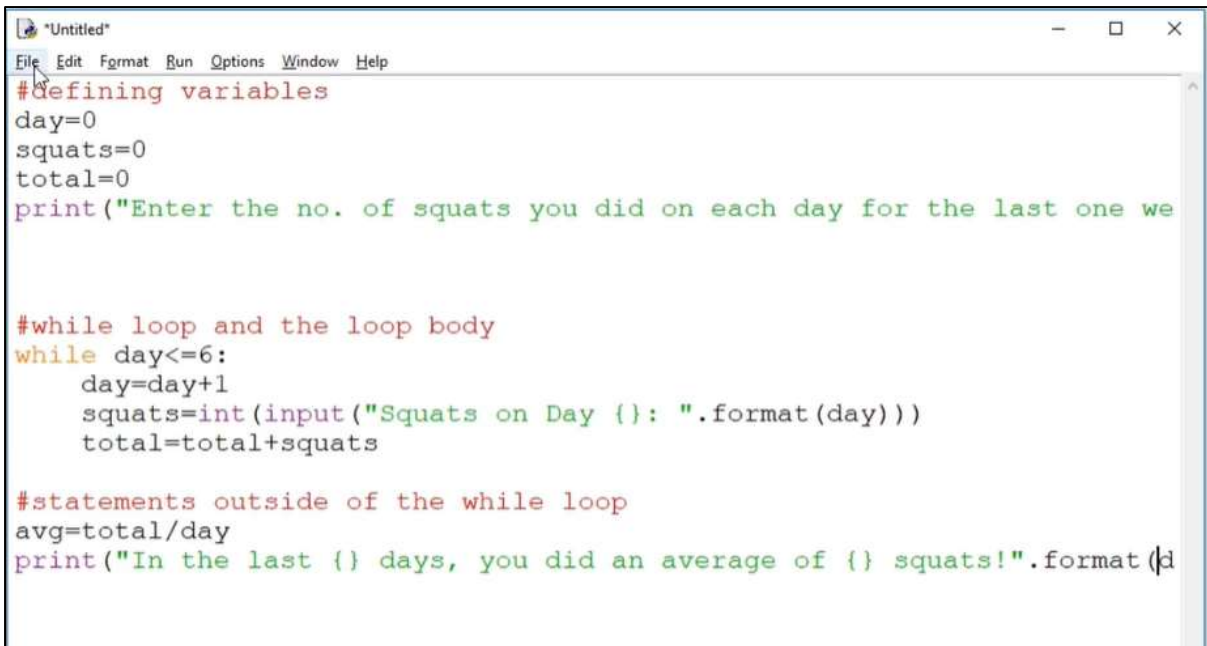
All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

**Syntax:** While keyword followed by Boolean expression and a colon. The colon starts a new indented block. This block has statements which are executed repeatedly. This is usually known as body of the loop. The body is executed till the Boolean expression remains true. When the expression evaluates to false, the program flow comes out of the loop and remaining statements are executed.

```
while expr==True:
    stmt1
    stmt2
    stmt3
stmt4
```

## **Example**





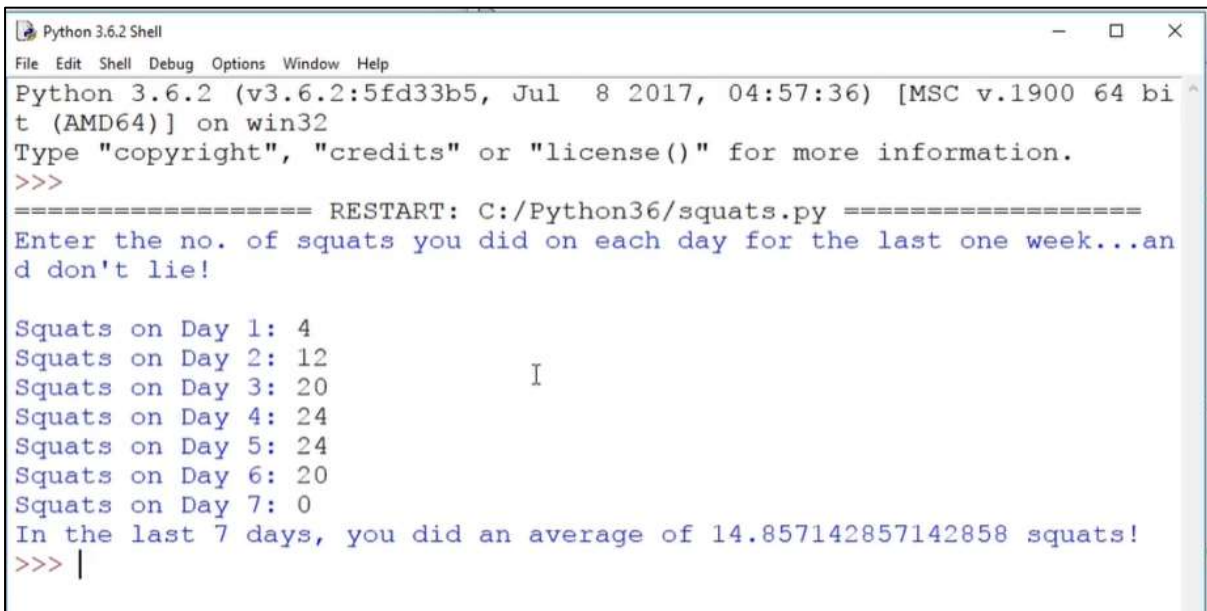
```

File Edit Format Run Options Window Help
#Defining variables
day=0
squats=0
total=0
print("Enter the no. of squats you did on each day for the last one we

#while loop and the loop body
while day<=6:
    day=day+1
    squats=int(input("Squats on Day {}: ".format(day)))
    total=total+squats

#statements outside of the while loop
avg=total/day
print("In the last {} days, you did an average of {} squats!".format(d

```



```

Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bi
t (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python36/squats.py =====
Enter the no. of squats you did on each day for the last one week...an
d don't lie!

Squats on Day 1: 4
Squats on Day 2: 12
Squats on Day 3: 20
Squats on Day 4: 24
Squats on Day 5: 24
Squats on Day 6: 20
Squats on Day 7: 0
In the last 7 days, you did an average of 14.857142857142858 squats!
>>> |

```

Inside the while loop, we can have any number of statements. The condition may be anything as per our requirement. The loop stops running when the condition fails (become false), and the execution will move to the next line of code.

### Flow Diagram of while loop

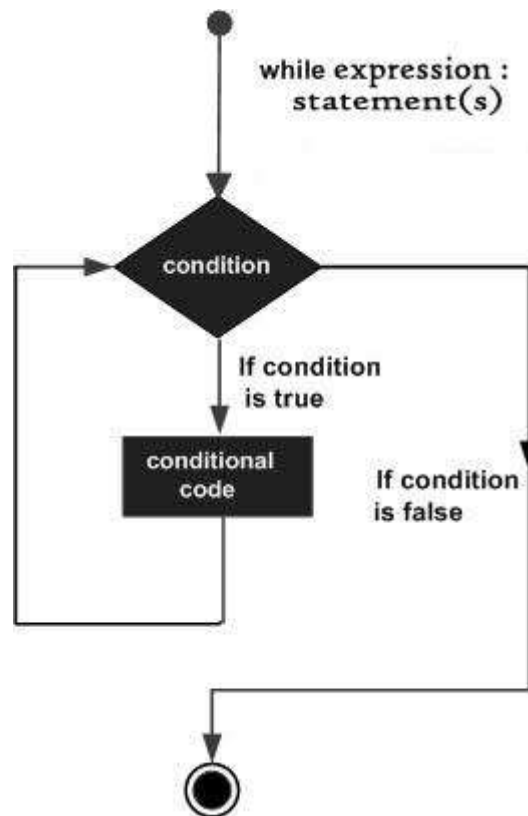


Image 1

It first checks the condition, executes the conditional code if the condition is TRUE, and checks the condition again. Program control exits the loop if the condition is FALSE.

Example:

```
# Python program to illustrate
# while loop
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
```

**Output:**

Hello Geek

Hello Geek

Hello Geek

**Using else statement with while loops:** As discussed above, while loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed.

The else clause is only executed when your while condition becomes false. If you break out of the

loop, or if an exception is raised, it won't be executed.

### **If else like this**

```
if condition:
    # execute these statements
else:
    # execute these statements
```

### **and while loop like this are similar**

```
while condition:
    # execute these statements
else:
    # execute these statements
```

```
#Python program to illustrate
# combining else with while
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
else:
    print("In Else Block")
```

### **Output:**

Hello Geek

Hello Geek

Hello Geek

In Else Block

**Example :** Print the text “Hello, World!” 5 times.

```
num_of_times = 1
while num_of_times <= 5:
    print("Hello, World!")
    num_of_times += 1
```

**Explanation:** The loop runs as long as the num\_of\_times variable is less than or equal to 5.

num\_of\_times increments by 1 after each iteration.

### **Output:**

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

**Example :** Create a list of all the even numbers between 1 and 10

```
i=1
while(i<=10):
    if(i%2==0):
        print(i,end="")
    i=i+1
Output
246810
>>>
```

**Explanation:** The loop runs as long as the num variable is less than or equal to 10. If the condition is TRUE, the program control enters the loop and appends the num to the even\_numbers list if the number is cleanly divisible by 2.

**Output:**


```
Even Numbers list: [2, 4, 6, 8, 10]
```

**Example :** Creating an infinite loop

A loop runs infinite times when the condition never fails.

```
i = True
while i:
    print("Condition satisfied")
```

**Output:**



```

Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied
Condition satisfied

```

### **Example :** use else with a while loop

When an else statement is used along with a while loop, the control goes to the else statement when the condition is False.

```

x=0
while x<5:
    x=x+1
    print ("iteration no {} in while loop".format(x))
else:
    print ("else block in loop")
print ("Out of loop")

```

### **Output:**

```

iteration no 1 in while loop
iteration no 2 in while loop
iteration no 3 in while loop
iteration no 4 in while loop
iteration no 5 in while loop
else block in loop
Out of loop

```

```
>>>
```

Write a program to print the multiplication table of 16 upto 16 \* 12 using the 'while' loop.

```

i=1
while i<=12 :

```

```
num=i*16
print(num)
i=i+1
```

**Output:**

```
16
32
48
64
80
96
112
128
144
160
176
192
```

**for loop**

Any for loop begins with the for statement.

**for- symbol -in “VESP”:**

**keyword-variable-keyword proceeding sequence-sequence-colon-end of the statement**

**Write a program to type each letter in VESP**

```
for symbol in "VESP":
    print(symbol)
```

```
V
E
S
P
```

A for loop is used to iterate over a sequence like lists, type, dictionaries, sets, or even strings.

Loop statements will be executed for each item of the sequence.

**Syntax of for loop:**

```
for x in sequence:
    stmt1
    stmt2
    stmt3
    stmt4
```

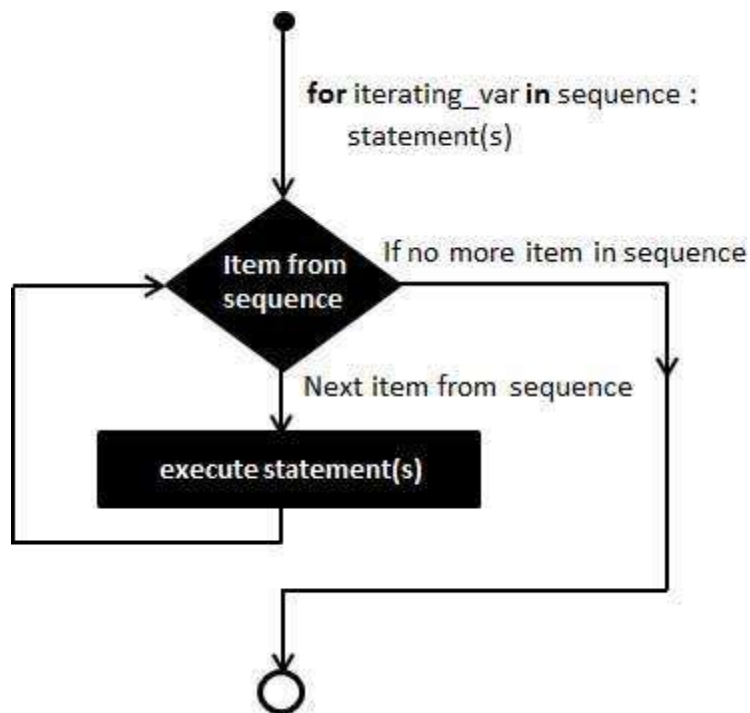
**Flow diagram of for loop:**

Image 2

Takes the first item of the iterable, executes the statement, and moves the pointer to the next item until it reaches the last item of the sequence.

**EXAMPLES:**

```
for num in range(1,5):
```

```
    print(num)
```

**OUTPUT:**

```
1
```

```
2
```

```
3
```

```
4
```

**EXAMPLE:**

```
# Python program to illustrate
```

```
# Iterating over range 0 to n-1
```

```
n = 4
```

```
for i in range(0, n):
    print(i)
```

**Output :**

```
0
1
2
3
```

# Python program to illustrate

# Iterating over a list

```
print("List Iteration")
```

```
l = ["geeks", "for", "geeks"]
```

```
for i in l:
```

```
    print(i)
```

EXAMPLE:

```
for num in range(1, 11):
    print (num)
print ("End of sequence")
```

```
1
2
3
4
5
6
7
8
9
10
End of sequence
>>>
```

EXAMPLE:



```
#Calculate factorial

num=int(input("Enter a number: "))
fact=1
for x in range(1,num+1):
    fact=fact*x
print ("The factorial of {} is {}".format(num, fact))
```

OUTPUT:

```
Enter a number: 6
The factorial of 6 is 720
```

EXAMPLE:

```
#String iteration

for char in "Push-ups":
    print (char)
print ("Push-ups done!")
```

P  
u  
s  
h  
-  
u  
p  
s  
Push-ups done!  
>>>

# Iterating over a tuple (immutable)

```
print("\nTuple Iteration")
```

```
t = ("geeks", "for", "geeks")
```

```
for i in t:
```

```
    print(i)
```

# Iterating over a String

```
print("\nString Iteration")
```

```
s = "Geeks"
```

```
for i in s:
```

```
print(i)
```

```
# Iterating over dictionary
```

```
print("\nDictionary Iteration")
```

```
d = dict()
```

```
d['xyz'] = 123
```

```
d['abc'] = 345
```

```
for i in d:
```

```
    print("%s %d" %(i, d[i]))
```

### Output:

List Iteration

geeks

for

geeks

Tuple Iteration

geeks

for

geeks

String Iteration

G

e

e

k

s

Dictionary Iteration

xyz 123

abc 345

**Iterating by index of sequences:** We can also use the index of elements in the sequence to iterate. The key idea is to first calculate the length of the list and then iterate over the sequence within the range of this length.

See the below example:

```
# Python program to illustrate
```

```
# Iterating by index
```

```
list = ["geeks", "for", "geeks"]
```

```
for index in range(len(list)):
```

```
    print list[index]
```

**Output:**

geeks

for

geeks

EXAMPLE:

```
#Average of numbers in a list
total=0
numbers=[10,20,30,40,50]
for num in numbers:
    total= total+num
avg= total /len(numbers)
print ('The average of the given numbers is:', avg)
```

OUTPUT:

```
| The average of the given numbers is: 30.0
```

**Using else statement with for loops:** We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate so the else block will be executed immediately after for block finishes execution.

Below example explains how to do this:

```
# Python program to illustrate
```

```
# combining else with for
```

```
list = ["geeks", "for", "geeks"]
```

```
for index in range(len(list)):
```

```
print list[index]
```

```
else:
```

```
    print "Inside Else Block"
```

### Output:

```
geeks
```

```
for
```

```
geeks
```

```
Inside Else Block
```

### Nested loops

Nested loops mean using a loop inside another loop. We can use any type of loop inside any type of loop. We can use a while loop inside for loop, a for loop inside a while loop, a while loop inside a while loop, or a for loop inside a for loop.

Nested loops	
while condition: statements	while condition: statements
for item in iterator: statements	while condition: statements
for item in iterator: statements	for item in iterator: statements
while condition: statements	for item in iterator: statements

**Nested Loops:** Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax:

```
for iterator_var in sequence:
    for iterator_var in sequence:
```

```
statements(s)
statements(s)
```

The syntax for a nested while loop statement in Python programming language is as follows:

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

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

```
# Python program to illustrate
# nested for loops in Python
from __future__ import print_function
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

**Output:**

```
1
2 2
3 3 3
4 4 4 4
```

## 1.7 Loop Manipulation:

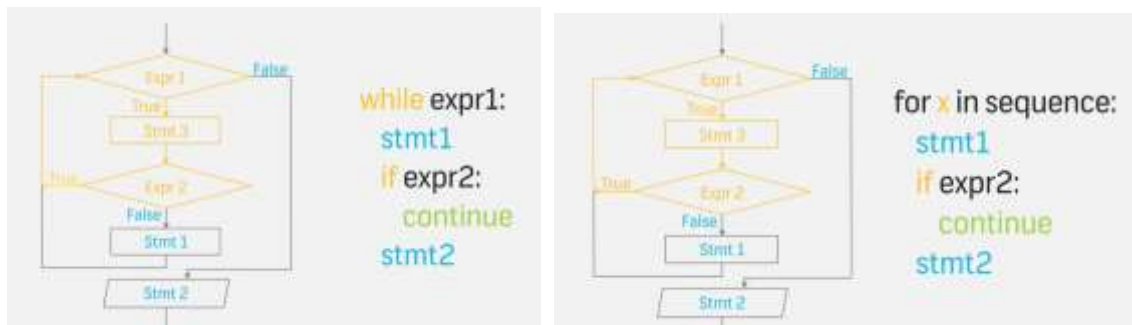
Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

### Continue Statement.

The **continue** statement in Python returns the control to the beginning of the while loop. The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

The **continue** statement can be used in both *while* and *for* loops.

Generic view of continue with While and for statement are displayed below.



### Example:

1. Accept input from user (n)	-	<code>num=int(input("enter a number"))</code>
2. Set divisor (d) to 2	-	<code>d = 2</code>
3. Perform steps 4 and 5 till $n > 1$	-	<code>while num &gt; 1:</code>
4. Check if given number (n) is divisible by divisor (d).		
5. If $n \% d == 0$	-	<code>if num % d == 0:</code>
6. Print d as a factor	-	<code>print (d)</code>
7. Set new value of n as $n/d$	-	<code>num=num/d</code>
8. Repeat from 4	-	<code>continue</code>
9. Else		
10. Increment d by 1	-	<code>d=d+1</code>
11. Repeat from 3		

prime.py - C:\Python36\prime.py (3.6.2)

File Edit Format Run Options Window Help

```
num=int(input("Enter a number: "))
d=2
while num>1:
    if num%d==0:
        print (d)
        num=num/d
        continue
    d=d+1
```

### Output:

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit
 (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Python36\prime.py =====
Enter a number: 60
2
2
3
5
>>>
===== RESTART: C:\Python36\prime.py =====
Enter a number: 105
3
5
7
>>>
```

### Example

```
# Prints all letters except 'e' and 's'
for letter in 'geeksforgeeks':
    if letter == 'e' or letter == 's':
        continue
    print ('Current Letter :', letter)
var = 10
```

**Output:**

```
Current Letter : g
Current Letter : k
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter : g
Current Letter : k
```

**Example**

```
#!/usr/bin/python

for letter in 'Python':      # First Example
    if letter == 'h':
        continue
    print ('Current Letter :', letter)

var = 10                      # Second Example
while var > 0:
    var = var -1
    if var == 5:
        continue
    print ('Current variable value :', var)
print ("Good bye!")
```

**Output:**

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

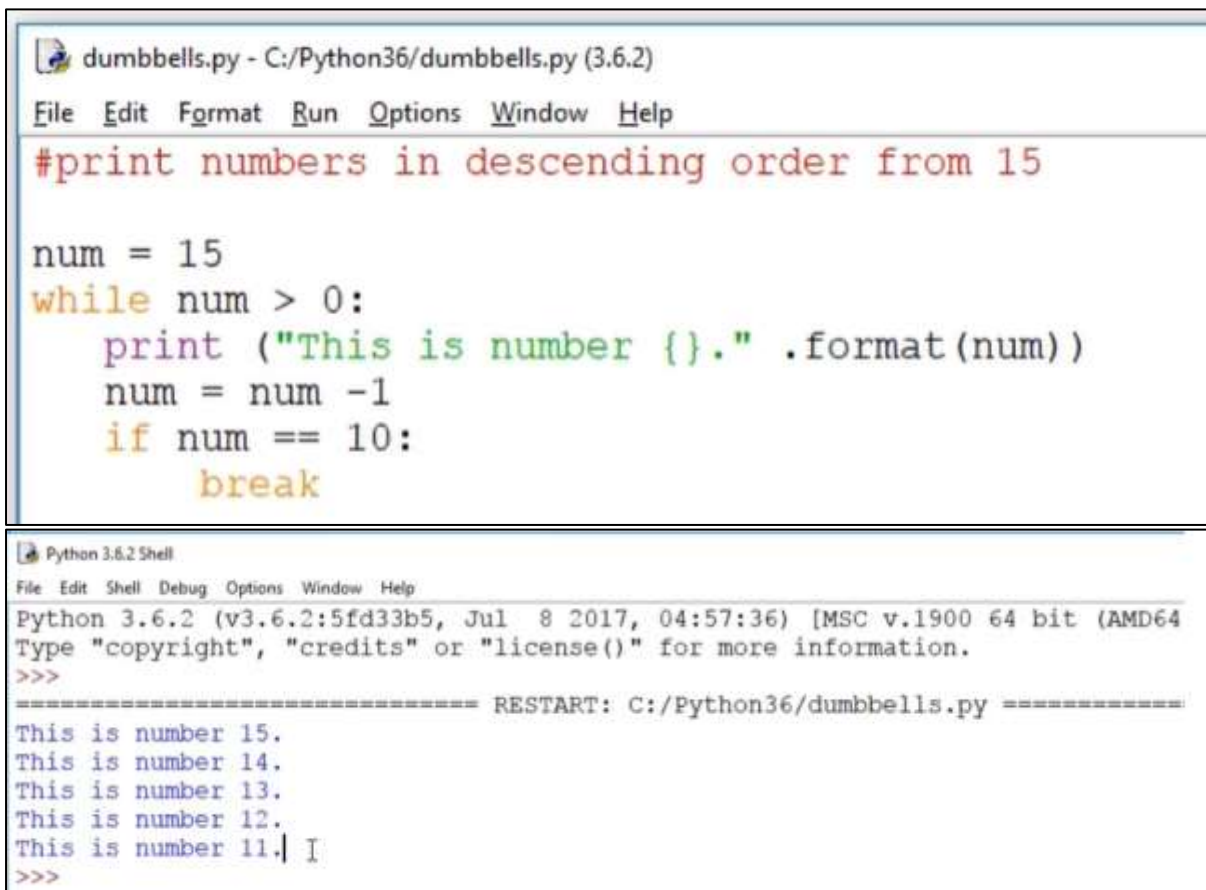
## Break Statement:

It brings control out of the loop. The **break** statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops.

```
while expr1:
    stmt1
    if expr2:
        break
    stmt2
    stmt3
```

### Example using While loop



The screenshot shows a Python IDE window titled 'dumbbells.py - C:/Python36/dumbbells.py (3.6.2)'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor is as follows:

```
#print numbers in descending order from 15

num = 15
while num > 0:
    print ("This is number {}".format(num))
    num = num -1
    if num == 10:
        break
```

Below the editor is a 'Python 3.6.2 Shell' window. Its menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell displays the following output:

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python36/dumbbells.py =====
This is number 15.
This is number 14.
This is number 13.
This is number 12.
This is number 11. | I
>>>
```

### Example using for loop



```
*prime.py - C:/Python36/prime.py (3.6.2)*
File Edit Format Run Options Window Help
#Check for a prime number

num=int(input("Enter a number: "))
for x in range(2,num):
    if num%x==0:
        print("{} is NOT a prime number".format(num))
        break
    else:
        print("{} is a prime number." .format(num))
```

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bi
t (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python36/prime.py =====
Enter a number: 23579
23579 is NOT a prime number
>>>
===== RESTART: C:/Python36/prime.py =====
Enter a number: 999983
999983 is a prime number.
>>>
```

### Example

for letter in 'fdsgeeksforggeeks':

```
# break the loop as soon it sees 'e'
# or 's'
if letter == 'e' or letter == 's':
    break
```

print ('Current Letter :', letter)

### Output:

Current Letter : s

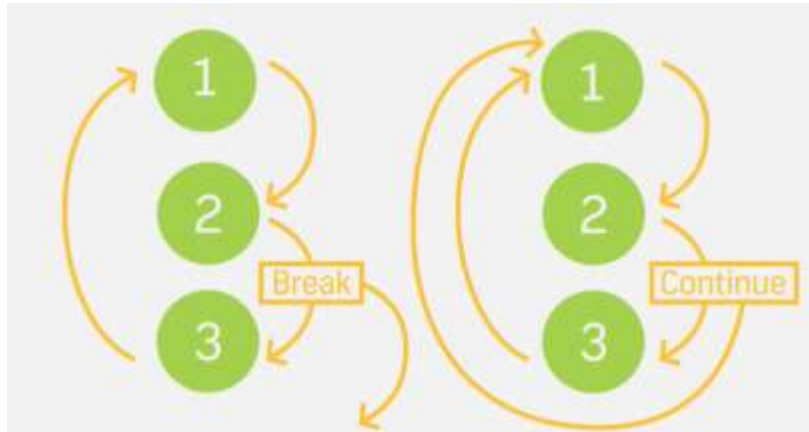
### Example:

```
#!/usr/bin/python
```

```
for num in range(10,20): #to iterate between 10 to 20
    for i in range(2,num): #to iterate on the factors of the number
        if num%i == 0:    #to determine the first factor
            j=num/i #to calculate the second factor
            print('%d equals %d * %d' % (num,i,j))
            break #to move to the next number, the #first FOR
    else:    # else part of the loop
        print (num, 'is a prime number')
```

**Output:**

10 equals 2 \* 5  
 11 is a prime number  
 12 equals 2 \* 6  
 13 is a prime number  
 14 equals 2 \* 7  
 15 equals 3 \* 5  
 16 equals 2 \* 8  
 17 is a prime number  
 18 equals 2 \* 9  
 19 is a prime number

**Pass Statement:**

We use pass statement to write empty loops. Pass is also used for empty control statement, function and classes. The **pass** statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

**Example**

# An empty loop

```

for letter in 'geeksforgeeks':
    pass
print 'Last Letter :', letter
  
```

**Output:**

Last Letter : s

**Example**

```

for letter in 'Python':
    if letter == 'h':
        pass
    print ('This is pass block')
    print ('Current Letter :', letter)
  
```

```
print ("Good bye!")
```

Output:

Current Letter : P  
Current Letter : y  
Current Letter : t  
This is pass block  
Current Letter : h  
Current Letter : o  
Current Letter : n  
Good bye!