**UNIT-I INTRODUCTION TO PYTHON**

**PART-I**

**Introduction to Python, Installing Python. How a Program Works, Using Python, Program Development Cycle, Input and Output, Comments, Variables, Data types, Reading Input from the Keyboard, Displaying Output with the Print Function, Performing Calculations, Operators, Type conversions, Expressions.**

**Introduction to Python:** Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

* **Python is Interpreted**: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* **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 of Python**

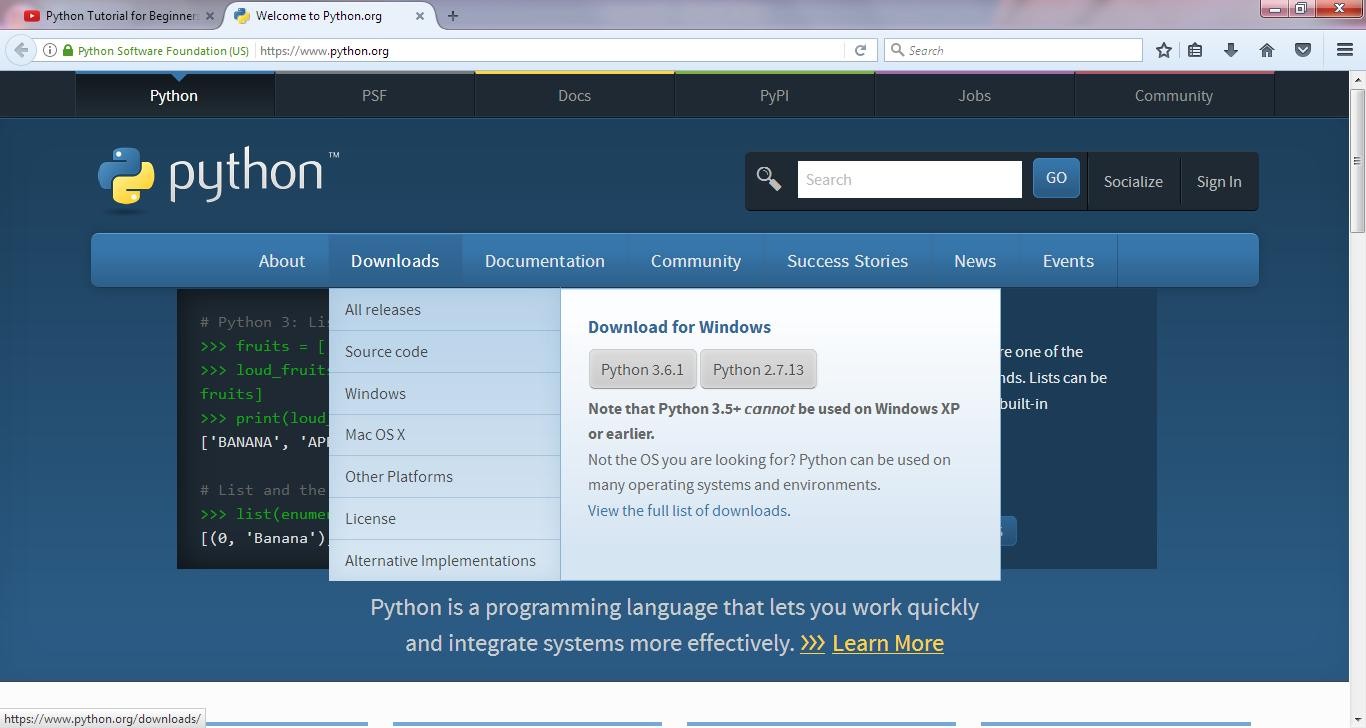
* Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
* Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, and other scripting languages.
* At the time when he began implementing Python, Guido van Rossum was also reading the published scripts from "Monty Python's Flying Circus" (a BBC comedy series from the seventies, in the unlikely case you didn't know). It occurred to him that he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.
* Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.
* Python 1.0 was released on 20 February, 1991.
* Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed.
* Python 3.0 (which early in its development was commonly referred to as Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008 after a long period of testing. Many of its major features have been back ported to the backwards-compatible Python 2.6.x and 2.7.x version series.
* In January 2017 Google announced work on a Python 2.7 to go transcompiler, which The Register speculated was in response to Python 2.7's planned end-of-life.

**Python Features**: Python's features include:

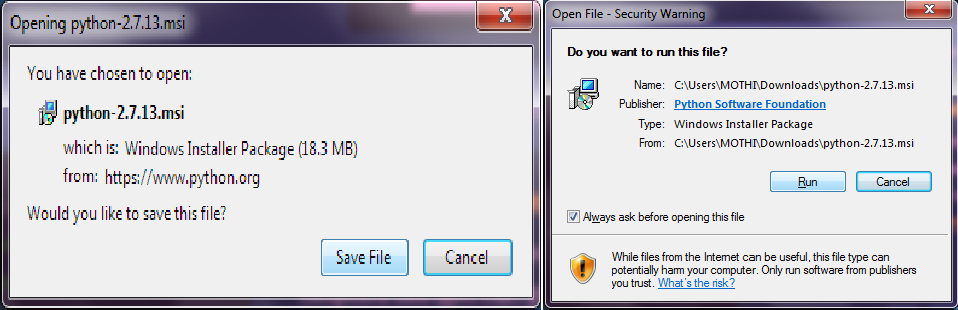
* Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
* Easy-to-read: Python code is more clearly defined and visible to the eyes.
* Easy-to-maintain: Python's source code is fairly easy-to-maintain.
* A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
* Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
* Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
* Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
* Databases: Python provides interfaces to all major commercial databases.
* GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of UNIX.
* Scalable: Python provides a better structure and support for large programs than shell scripting.

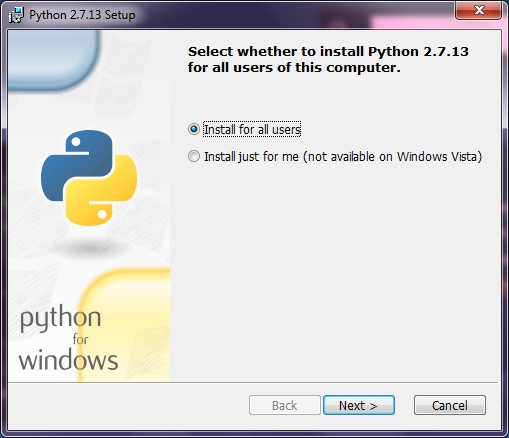
**Installing Python:**

**Step 1:** Go to website [www.python.org](http://www.python.org/) and click downloads select version which you want.



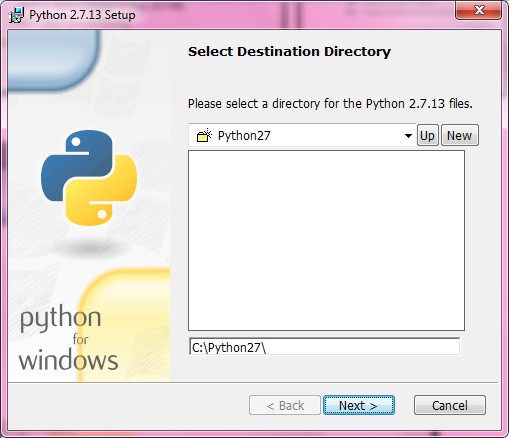
**Step 2:** Click on **Python 2.7.13** and download. After download open the file.



**Step 3:** Click on **Next** to continue.

**Step 4:** After installation location will be displayed. The Default location is **C:\Python27.**

Click on next to continue.



**Step 5:** After the python interpreter and libraries are displayed for installation. Click on Next to continue.



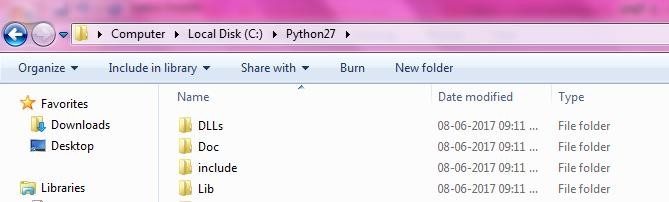
**Step 6:** The installation has been processed.

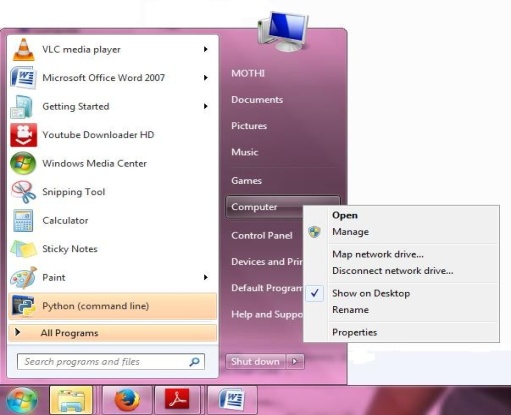
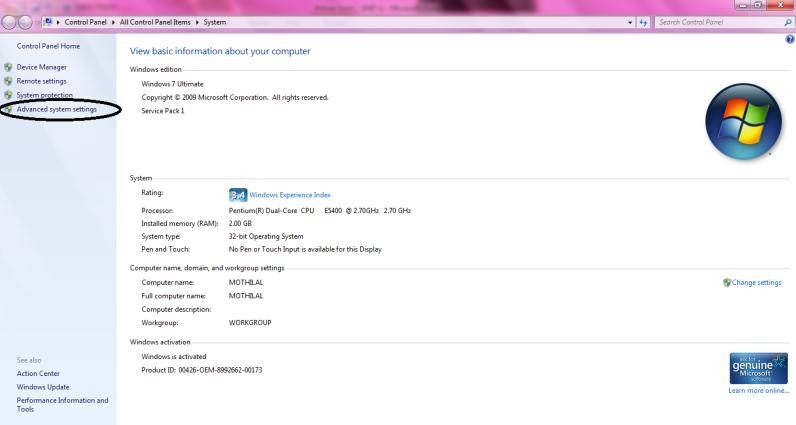


**Step 7:** Click the **Finish** to complete the installation.



# Setting up PATH to python:

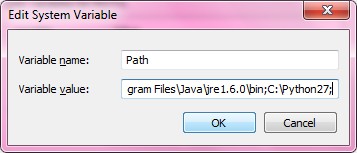
* Programs and other executable files can be in many directories, so operating systems provide a search path that lists the directories that the OS searches for executables.
* The path is stored in an environment variable, which is a named string maintained by the operating system. This variable contains information available to the command shell and other programs.
* Copy the Python installation location C:\Python27
* Right-click the My Computer icon on your desktop and choose **Properties**. And then select **Advanced System properties.**

* Goto **Environment Variables** and go to **System Variables** select **Path** and click on

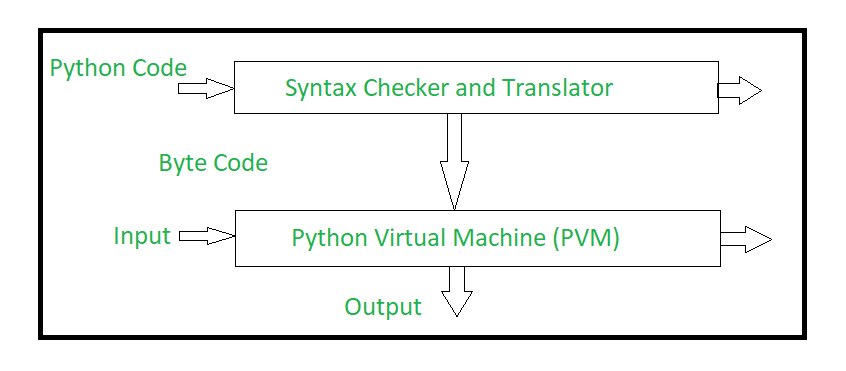
## Edit.

* Copy the location **C:\Python27 &** give semicolon (;) & click OK.



**How a Program Works:** Python is an object oriented programming language like Java. Python is called an interpreted language. Python uses code modules that are interchangeable instead of a single long list of instructions that was standard for functional programming languages. The standard implementation of python is called “cpython”. It is the default and widely used implementation of the Python.

Python doesn’t convert its code into machine code, something that hardware can understand. It actually converts it into something called byte code. So within python, compilation happens, but it’s just not into a machine language. It is into byte code and this byte code can’t be understood by CPU. So we need actually an interpreter called the python virtual machine. The python virtual machine executes the byte codes.

****

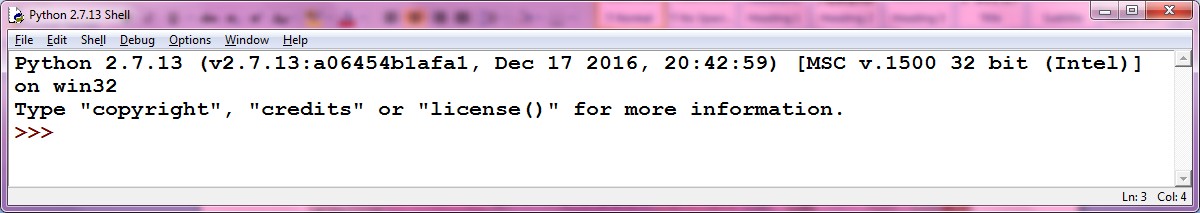
The Python interpreter performs following tasks to execute a Python program :

* Step 1 : The interpreter reads a python code or instruction. Then it verifies that the instruction is well formatted, i.e. it checks the syntax of each line.If it encounters any error, it immediately halts the translation and shows an error message.
* Step 2 : If there is no error, i.e. if the python instruction or code is well formatted then the interpreter translates it into its equivalent form in intermediate language called “Byte code”.Thus, after successful execution of Python script or code, it is completely translated into Byte code.
* Step 3 : Byte code is sent to the Python Virtual Machine(PVM).Here again the byte code is executed on PVM.If an error occurs during this execution then the execution is halted with an error message.

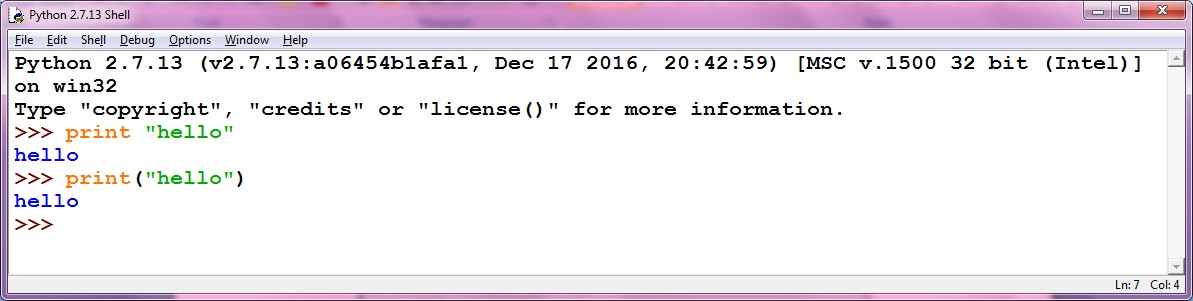
**Using Python:**

**Running Python Interpreter:**

Python comes with an interactive interpreter. When you type python in your shell or command prompt, the python interpreter becomes active with a >>> prompt and waits for your commands.



Now you can type any valid python expression at the prompt. Python reads the typed expression, evaluates it and prints the result.



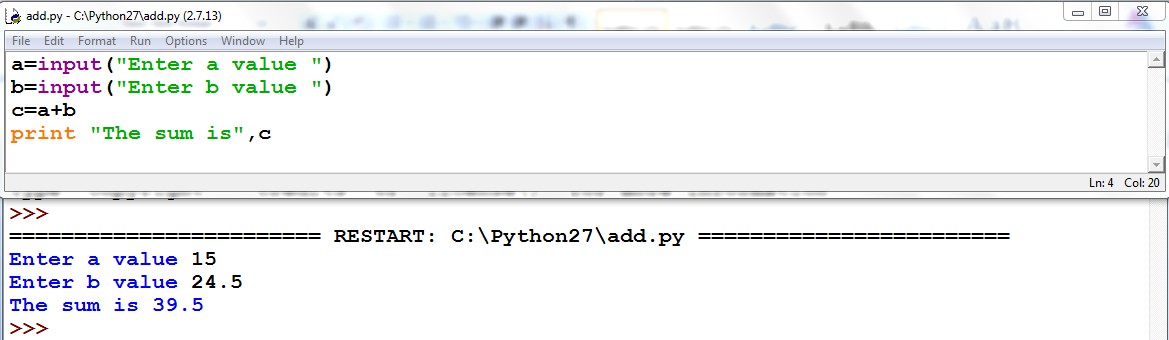
**Running Python Scripts in IDLE:**

* + Goto **File** menu click on New File (CTRL+N) and write the code and save add.py a=input("Enter a value ")

b=input("Enter b value ") c=a+b

print "The sum is",c

* + And run the program by pressing F5 or RunRun Module.

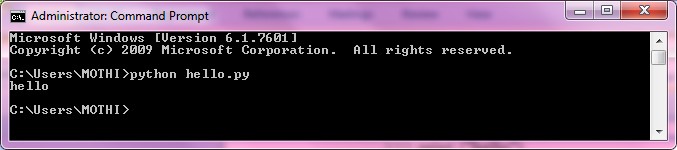


**Running python scripts in Command Prompt:**

* + Before going to run we have to check the PATH in environment variables.
  + Open your text editor, type the following text and save it as hello.py.

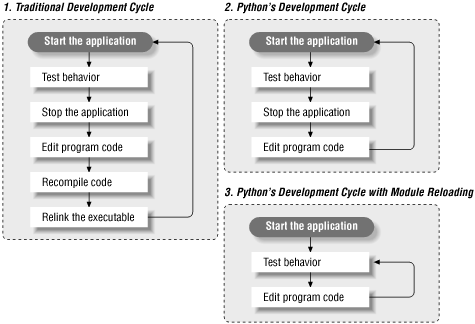
**print "hello"**

* + And run this program by calling python hello.py. Make sure you change to the directory where you saved the file before doing it.

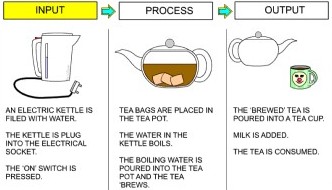


**Program Development Cycle:**

**Analyze->Design->Code->Test/Debug->Document**



**Input, Processing, and Output:** A simple example to demonstrate all three.



**Displaying Output with the Print Function:**

**Python print() function**prints the message to the screen or any other standard output device.

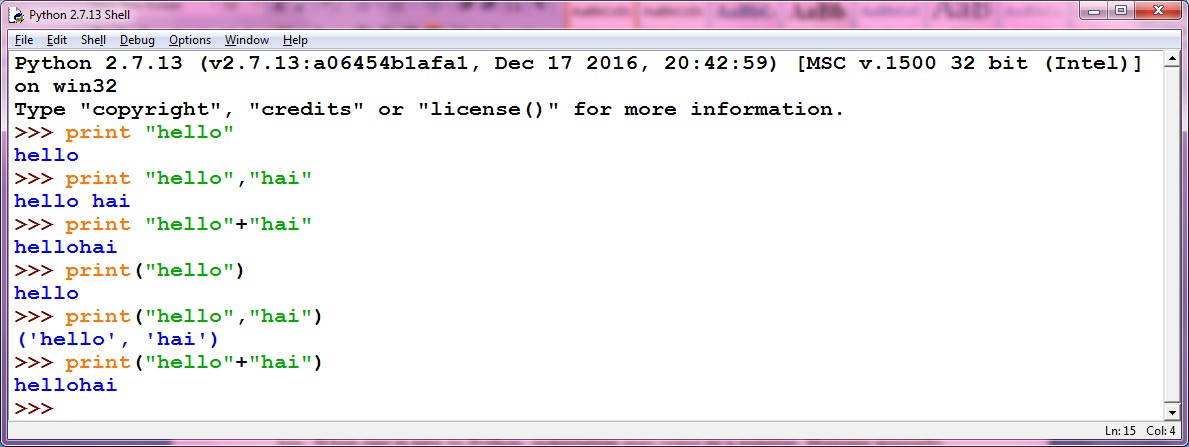
**Syntax:**print(value(s), sep= ‘ ‘, end = ‘\n’, file=file, flush=flush)

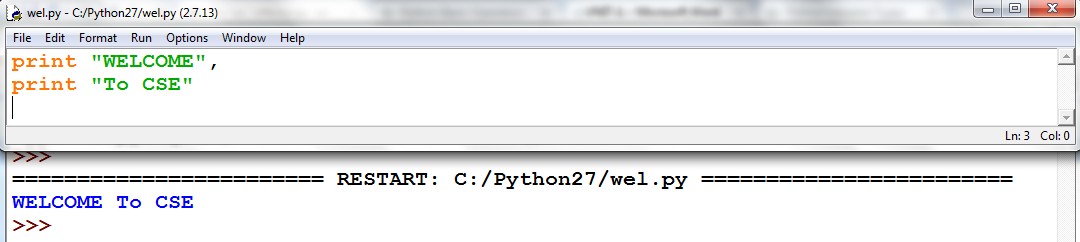
**Parameters:**

* **value(s) :**Any value, and as many as you like. Will be converted to string before printed
* **sep=’separator’ :**(Optional) Specify how to separate the objects, if there is more than one.Default :’ ‘
* **end=’end’:**(Optional) Specify what to print at the end.Default : ‘\n’
* **file :**(Optional) An object with a write method. Default :sys.stdout
* **flush :**(Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

**Returns:**It returns output to the screen.

Though it is not necessary to pass arguments in the print() function, it requires an empty parenthesis at the end that tells python to execute the function rather calling it by name. Now, let’s explore the optional arguments that can be used with the print() function.

****

****

**Example program**

print "Welcome to Guru99"

print("USA")

print (8 \* "\n")

print ("Welcome to", end = ' ')

print ("Guru99", end = '!')

print('can do this',5)

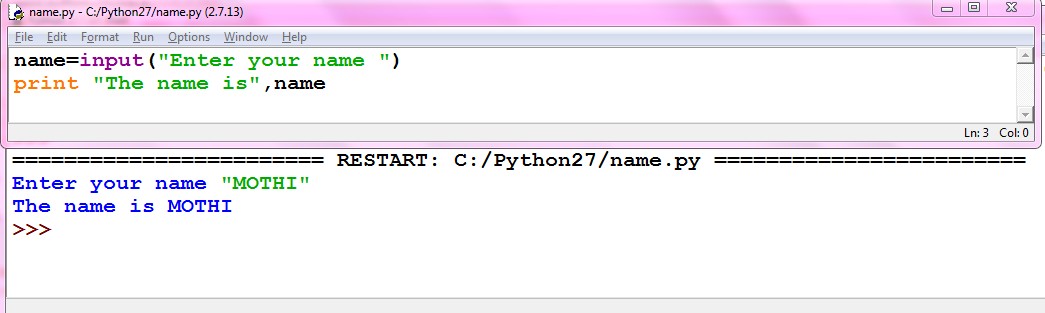
print('cannot do this:'+5)

**Note:** You cannot use the "+" to join strings with ints or floats, you must use the ","

**Reading Input from the Keyboard:**

To get input from the user you can use the input function. When the input function is called the program stops running the program, prompts the user to enter something at the keyboard by printing a string called the prompt to the screen, and then waits for the user to press the Enter key. The user types a string of characters and presses enter. Then the input function returns that string and Python continues running the program by executing the next statement after the input statement.

Python provides the function input(). input has an optional parameter, which is the prompt string. For example



**Comments:** Comments in Python start with the hash character, # , and extend to the end of the physical line. Acomment may appear at the start of a line or following whitespace or code, but not within a string literal. A hash character within a string literal is just a hash character.

#This is a comment

#print out Hello

print('Hello')

**Multi-line comments:**

"""This is also a

perfect example of

multi-line comments"""

**Variables:** Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

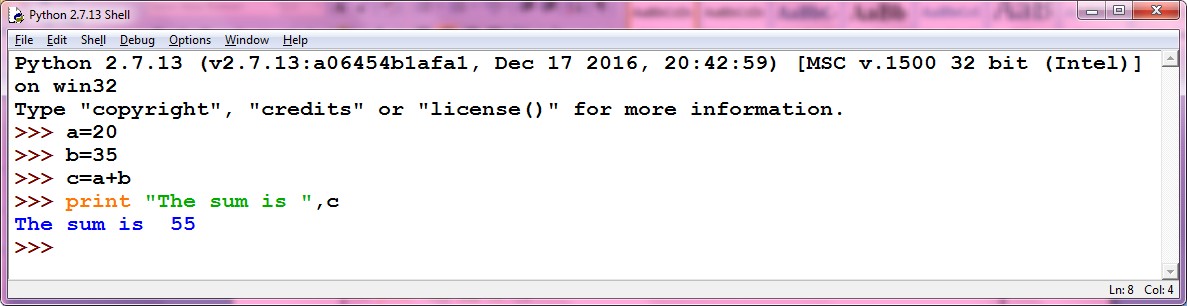
**Assigning Values to Variables**

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume). Rules for Python variables:

* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive (age, Age and AGE are three different variables)

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example –



**Multiple Assignments to variables:**

Python allows you to assign a single value to several variables simultaneously.

For example –

**a = b = c = 1**

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables.

For example –

**a, b, c = 1, 2.5, ”mothi”**

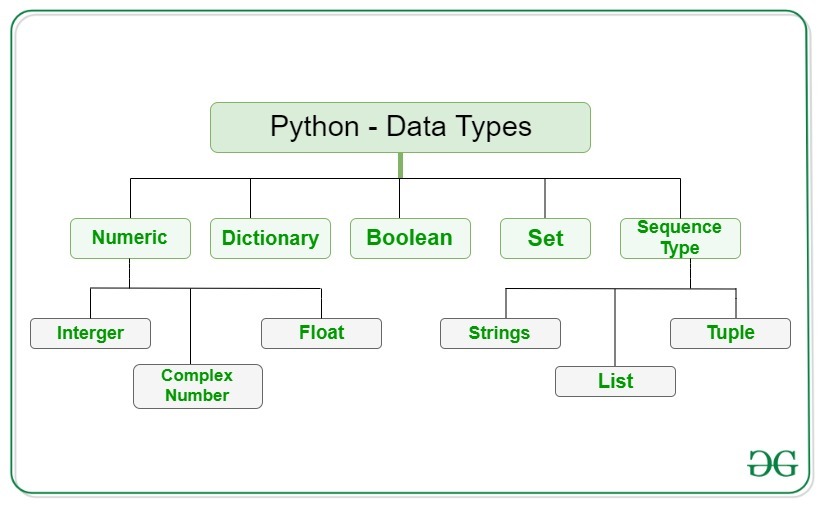
Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

**Data types:**

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

Following are the standard or built-in data type of Python:

* [Numeric](https://www.geeksforgeeks.org/python-data-types/#numeric)
* [Sequence Type](https://www.geeksforgeeks.org/python-data-types/#Sequence)
* [Boolean](https://www.geeksforgeeks.org/python-data-types/#boolean)
* [Set](https://www.geeksforgeeks.org/python-data-types/#set)
* [Dictionary](https://www.geeksforgeeks.org/python-data-types/#dictionary)



**Numeric**

In Python, numeric data type represent the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

* **Integers**– This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
* **Float**– This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
* **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

**Note** – type() function is used to determine the type of data type.

**Example:**

#Python program to demonstrate numeric value

a = 5

print("Type of a: ", type(a))

b = 5.0

print("\nType of b: ", type(b))

c = 2 + 4j

print("\nType of c: ", type(c))

**Output:**

Type of a: <class 'int'>

Type of b: <class 'float'>

Type of c: <class 'complex'>

**Sequence Type**

In Python, sequence is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

* [String](https://www.geeksforgeeks.org/python-data-types/#string)
* [List](https://www.geeksforgeeks.org/python-data-types/#list)
* [Tuple](https://www.geeksforgeeks.org/python-data-types/#tuple)

**1) String**

In Python, [Strings](https://www.geeksforgeeks.org/python-strings/) are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

**Creating String**

Strings in Python can be created using single quotes or double quotes or even triple quotes.

Example Program

# Creating a String with single Quotes

String1 = 'Welcome to the CMRIT'

print("String with the use of Single Quotes: ")

print(String1)

# Creating a String with double Quotes

String1 = "I'm a CMRITIAN"

print("\nString with the use of Double Quotes: ")

print(String1)

print(type(String1))

# Creating a String with triple Quotes

String1 = '''I'm a CMRITIAN and I live in a world of "CMR"'''

print("\nString with the use of Triple Quotes: ")

print(String1)

print(type(String1))

# Creating String with triple Quotes allows multiple lines

String1 = '''hi

            HELLO

            WELCOME'''

print("\nCreating a multiline String: ")

print(String1)

**Output:**

String with the use of Single Quotes:

Welcome to the CMRIT

String with the use of Double Quotes:

I'm a CMRITIAN

<class 'str'>

String with the use of Triple Quotes:

I'm a CMRITIAN and I live in a world of "CMR"

<class 'str'>

Creating a multiline String:

hi

HELLO

WELCOME

**Accessing elements of String**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| **H** | **E** | **L** | **L** | **O** |
| -5 | -4 | -3 | -2 | -1 |

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.

**Example Program**

#python Program to Access characters of String

String1 = "SWAPNA"

print("Initial String: ")

print(String1)

# Printing First character

print("\nFirst character of String is: ")

print(String1[0])

# Printing Last character

print("\nLast character of String is: ")

print(String1[-1])

**Output:**

Initial String:

SWAPNA

First character of String is:

S

Last character of String is:

A

**2) List**

[Lists](https://www.geeksforgeeks.org/python-list/) are just like the arrays, declared in other languages which is a ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

**Creating List**

Lists in Python can be created by just placing the sequence inside the square brackets [].

# Python program to demonstrate Creation of List

# Creating a List

List = []

print("Intial blank List: ")

print(List)

# Creating a List with the use of a String

List = ['cmrit']

print("\nList with the use of String: ")

print(List)

# Creating a List with the use of multiple values

List = ["Ram", "Laxman", "Bharath"]

print("\nList containing multiple values: ")

print(List[0])

print(List[2])

# Creating a Multi-Dimensional List (By Nesting a list inside a List)

List = [['swapna', 'rani'], ['ganaji']]

print("\nMulti-Dimensional List: ")

print(List)

**Output:**

Intial blank List:

[]

List with the use of String:

['cmrit']

List containing multiple values:

Ram

Bharath

Multi-Dimensional List:

[['swapna', 'rani'], ['ganaji']]

**Accessing elements of List**

In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

**Example Program**

# Python program to demonstrate accessing of element from list

# Creating a List with the use of multiple values

List = ["AIML", "CSE", "DS"]

# accessing a element from the list using index number

print("Accessing element from the list")

print(List[0])

print(List[2])

# accessing a element using negative indexing

print("Accessing element using negative indexing")

# print the last element of list

print(List[-1])

# print the third last element of list

print(List[-3])

**Output:**

Accessing element from the list

AIML

DS

Accessing element using negative indexing

AIML

DS

**3) Tuple**

Just like list, [tuple](https://www.geeksforgeeks.org/python-tuples/) is also an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by  tuple class.

**Creating Tuple**

In Python, [tuples](https://www.geeksforgeeks.org/python-tuples/) are created by placing a sequence of values separated by ‘comma’ with or without the use of parentheses for grouping of the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, list, etc.).

**Note:** Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing ‘comma’ to make it a tuple.

**Example:**

# Python program to demonstrate creation of Set

# Creating an empty tuple

Tuple1 = ()

print("Initial empty Tuple: ")

print (Tuple1)

# Creating a Tuple with the use of Strings

Tuple1 = ('AIML', 'DS')

print("\nTuple with the use of String: ")

print(Tuple1)

# Creating a Tuple with the use of list

list1 = [1, 2, 4, 5, 6]

print("\nTuple using List: ")

print(tuple(list1))

# Creating a Tuple with the use of built-in function

Tuple1 = tuple('swapna')

print("\nTuple with the use of function: ")

print(Tuple1)

# Creating a Tuple  with nested tuples

Tuple1 = (0, 1, 2, 3)

Tuple2 = ('swapna', 'rani')

Tuple3 = (Tuple1, Tuple2)

print("\nTuple with nested tuples: ")

print(Tuple3)

**Output:**

Initial empty Tuple:

()

Tuple with the use of String:

('AIML', 'DS')

Tuple using List:

(1, 2, 4, 5, 6)

Tuple with the use of function:

('s', 'w', 'a', 'p', 'n', ‘a’)

Tuple with nested tuples:

((0, 1, 2, 3), ('swapna', 'rani'))

**Note**– Creation of Python tuple without the use of parentheses is known as Tuple Packing.

**Accessing elements of Tuple**

In order to access the tuple items refer to the index number. Use the index operator [ ] to access an item in a tuple. The index must be an integer. Nested tuples are accessed using nested indexing.

**Example program:**

# Python program to demonstrate accessing tuple

tuple1 = tuple([1, 2, 3, 4, 5])

# Accessing element using indexing

print("First element of tuple")

print(tuple1[0])

# Accessing element from last

# negative indexing

print("\nLast element of tuple")

print(tuple1[-1])

print("\nThird last element of tuple")

print(tuple1[-3])

**Output:**

First element of tuple

1

Last element of tuple

5

Third last element of tuple

3

**Boolean**

Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool.

**Note –** True and False with capital ‘T’ and ‘F’ are valid booleans otherwise python will throw an error.

**Example Program:**

# Python program to demonstrate boolean type

print(type(True))

print(type(False))

print(type(true))

**Output:**

<class 'bool'>

<class 'bool'>

Traceback (most recent call last):

File "/home/7e8862763fb66153d70824099d4f5fb7.py", line 8, in

print(type(true))

NameError: name 'true' is not defined

**Set**

In Python, [Set](https://www.geeksforgeeks.org/python-sets/) is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

**Creating Sets**

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by ‘comma’. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

# Python program to demonstrate Creation of Set in Python

# Creating a Set

set1 = set()

print("Intial blank Set: ")

print(set1)

# Creating a Set with

# the use of a String

set1 = set("CMRIT")

print("\nSet with the use of String: ")

print(set1)

# Creating a Set with the use of a List

set1 = set(["CSE", "AIML", "DS", “CSE”])

print("\nSet with the use of List: ")

print(set1)

# Creating a Set with a mixed type of values (Having numbers and strings)

set1 = set([1, 2, 'AIML', 4, ‘DS', 6, 'AIML'])

print("\nSet with the use of Mixed Values")

print(set1)

**Output:**

Intial blank Set:

set()

Set with the use of String:

{'C', 'M', 'R', 'I', 'T'}

Set with the use of List:

{'CSE', 'AIML', ‘DS’}

Set with the use of Mixed Values

{1, 2, 4, 6, 'AIML', 'DS'}

**Accessing elements of Sets**

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

**Example Program**

#Python program to demonstrate Accessing of elements in a set

# Creating a set

set1 = set(["CSE", "AIML", "CSE"])

print("\nInitial set")

print(set1)

# Accessing element using  for loop

print("\nElements of set: ")

for i in set1:

    print(i, end =" ")

# Checking the element using in keyword

print("CSE" in set1)

**Output**

Initial set:

{'CSE', 'DS'}

Elements of set:

CSE DS

True

**Adding Elements to a Set**

Elements can be added to the Set by using built-in **add()** function. Only one element at a time can be added to the set by using add() method, loops are used to add multiple elements at a time with the use of add() method.

**Using update() method**

For addition of two or more elements Update() method is used. The update() method accepts lists, strings, tuples as well as other sets as its arguments. In all of these cases, duplicate elements are avoided.

**Accessing a Set**

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

**Removing elements from the Set**

Using remove() method or discard() method

Elements can be removed from the Set by using built-in remove() function but a KeyError arises if element doesn’t exist in the set. To remove elements from a set without KeyError, use discard(), if the element doesn’t exist in the set, it remains unchanged.

**Example Program**

set1 = set()

print("Initial blank Set: ")

print(set1)

# Adding element and tuple to the Set

set1.add(8)

set1.add(9)

set1.add((6,7))

print("\nSet after Addition of Three elements: ")

print(set1)

# Adding elements to the Set using Iterator

for i in range(1, 6):

    set1.add(i)

print("\nSet after Addition of elements from 1-5: ")

print(set1)

# Addition of elements to the Set using Update function

set1.update([10, 11])

print("\nSet after Addition of elements using Update: ")

print(set1)

# Accessing element using for loop

print("\nElements of set: ")

for i in set1:

    print(i, end=" ")

set1 = set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

# Removing elements from Set using Remove() method

set1.remove(8)

set1.remove(9)

print("\nSet after Removal of two elements: ")

print(set1)

# Removing elements from Set using Discard() method

set1.discard(10)

set1.discard(13)

print("\nSet after Discarding two elements: ")

print(set1)

# Removing elements from Set using iterator method

for i in range(1, 5):

    set1.remove(i)

print("\nSet after Removing a range of elements: ")

print(set1)

output:

Initial blank Set:

set()

Set after Addition of Three elements:

{8, 9, (6, 7)}

Set after Addition of elements from 1-5:

{1, 2, 3, (6, 7), 4, 5, 8, 9}

Set after Addition of elements using Update:

{1, 2, 3, (6, 7), 4, 5, 8, 9, 10, 11}

Elements of set:

1 2 3 (6, 7) 4 5 8 9 10 11

Set after Removal of two elements:

{1, 2, 3, 4, 5, 6, 7, 10, 11, 12}

Set after Discarding two elements:

{1, 2, 3, 4, 5, 6, 7, 11, 12}

Set after Removing a range of elements:

{5, 6, 7, 11, 12}

**Dictionary**

[Dictionary](https://www.geeksforgeeks.org/python-dictionary/) in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a ‘comma’.

**Creating Dictionary**

In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by ‘comma’. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable. Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it to curly braces{}.

**Note**– Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

**Example program**

# Creating an empty Dictionary

Dict = {}

print("Empty Dictionary: ")

print(Dict)

# Creating a Dictionary with Integer Keys

Dict = {1: 'CSE', 2: 'DS', 3: 'AIML'}

print("\nDictionary with the use of Integer Keys: ")

print(Dict)

# Creating a Dictionary with Mixed keys

Dict = {'Name': 'CMRIT', 1: [1, 2, 3, 4]}

print("\nDictionary with the use of Mixed Keys: ")

print(Dict)

# Creating a Dictionary with dict() method

Dict = dict({1: 'AIML', 2: 'CSE', 3:'DS'})

print("\nDictionary with the use of dict(): ")

print(Dict)

# Creating a Dictionary with each item as a Pair

Dict = dict([(1, 'KIRAN'), (2, 'BHAVYESH')])

print("\nDictionary with each item as a pair: ")

print(Dict)

**Output:**

Empty Dictionary:

{}

Dictionary with the use of Integer Keys:

{1: 'CSE', 2: 'DS', 3: 'AIML'}

Dictionary with the use of Mixed Keys:

{'Name': 'CMRIT', 1: [1, 2, 3, 4]}

Dictionary with the use of dict():

{1: 'AIML', 2: 'CSE', 3: 'DS'}

Dictionary with each item as a pair:

{1: 'KIRAN', 2: 'BHAVYESH'}

**Accessing elements of Dictionary**

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets. There is also a method called get() that will also help in accessing the element from a dictionary.

# Python program to demonstrate accessing a element from a Dictionary

# Creating a Dictionary

Dict = {1: '526', 'name': 'swapna', ‘address’: 'hyderabad'}

# accessing a element using key

print("Accessing a element using key:")

print(Dict['name'])

# accessing a element using get() method

print("Accessing a element using get:")

print(Dict.get(1))

**Output:**

Accessing a element using key:

swapna

Accessing a element using get:

526

**Operators:** Operators are used to perform operations on variables and values.Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

**Python Arithmetic Operators:** Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |
| --- | --- | --- |
| Operator | Name | Example |
| + | Addition | x + y |
| - | Subtraction | x - y |
| \* | Multiplication | x \* y |
| / | Division | x / y |
| % | Modulus | x % y |
| \*\* | Exponentiation | x \*\* y |
| // | Floor division | x // y |

**Python Assignment Operators:** Assignment operators are used to assign values to variables

|  |  |  |
| --- | --- | --- |
| Operator | Example | Same As |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| //= | x //= 3 | x = x // 3 |
| \*\*= | x \*\*= 3 | x = x \*\* 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

**Python Comparison Operators:** Comparison operators are used to compare two values:

|  |  |  |
| --- | --- | --- |
| Operator | Name | Example |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

**Python Logical Operators**: Logical operators are used to combine conditional statements:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| and | Returns True if both statements are true | x < 5 and  x < 10 |
| or | Returns True if one of the statements is true | x < 5 or x < 4 |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) |

**Python Identity Operators**: Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| is | Returns true if both variables are the same object | x is y |
| is not | Returns true if both variables are not the same object | x is not y |

**Python Membership Operators**: Membership operators are used to test if a sequence is presented in an object:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| in | Returns True if a sequence with the specified value is present in the object | x in y |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y |

**Python Bitwise Operators**: Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| Operator | Name | Description |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

**Type conversions:** Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function. For example, it is not possible to perform “2”+4 since one operand is integer and the other is string type. To perform this we have convert string to integer i.e., **int(“2”) + 4 = 6**.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

|  |  |
| --- | --- |
| **Function** | **Description** |
| int(x [,base]) | Converts x to an integer. |
| long(x [,base] ) | Converts x to a long integer. |
| float(x) | Converts x to a floating-point number. |
| complex(real [,imag]) | Creates a complex number. |
| str(x) | Converts object x to a string representation. |
| repr(x) | Converts object x to an expression string. |
| eval(str) | Evaluates a string and returns an object. |
| tuple(s) | Converts s to a tuple. |
| list(s) | Converts s to a list. |
| set(s) | Converts s to a set. |
| dict(d) | Creates a dictionary, d must be a sequence of (key, value) tuples. |
| frozenset(s) | Converts s to a frozen set. |
| chr(x) | Converts an integer to a character. |
| unichr(x) | Converts an integer to a Unicode character. |
| ord(x) | Converts a single character to its integer value. |
| hex(x) | Converts an integer to a hexadecimal string. |
| oct(x) | Converts an integer to an octal string. |

**Expressions:** An expression is a combination of variables constants and operators written according to the syntax of Python language. In Python every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable. Some examples of Python expressions are shown in the table given below.

|  |  |
| --- | --- |
| **Algebraic Expression** | **Python Expression** |
| a x b – c | a \* b – c |
| (m + n) (x + y) | (m + n) \* (x + y) |
| (ab / c) | a \* b / c |
| 3x2 +2x + 1 | 3\*x\*x+2\*x+1 |
| (x / y) + c | x / y + c |

**UNIT I**

**PART-II**

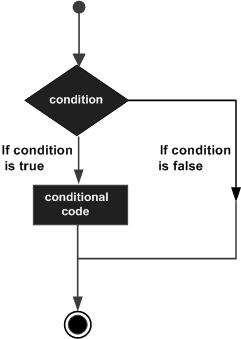
**Control Flow Statements**: Decision Structures and Boolean Logic: if, if-else, if-elif-else Statements, Nested Decision Structures, Repetition Structures: Introduction, while loop, for loop, Input Validation Loops, Nested Loops, control statements-break, continue, pass.

**Decision Structures and Boolean Logic:**

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce True or False as outcome. You need to determine which action to take and which statements to execute if outcome is True or False otherwise.

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

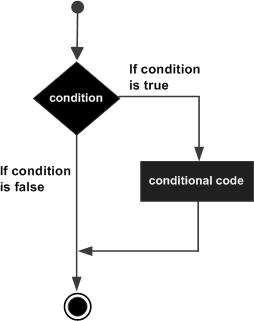


Python programming language assumes any non-zero and non-null values as True, and if it is either zero or null, then it is assumed as False value.

|  |  |
| --- | --- |
| **Statement** | **Description** |
| if statements | **if statement** consists of a boolean expression followed by one or more  statements. |
| if...else statements | **if statement** can be followed by an optional **else statement**, which  executes when the boolean expression is FALSE. |
| nested if statements | You can use one **if** or **else if** statement inside another **if** or **else if**  statement(s). |

**The *if* Statement**

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



**Syntax:**

if condition: statements

First, the condition is tested. If the condition is True, then the statements given after colon (:) are executed. We can write one or more statements after colon (:).

**Example:**

a=10 b=15

if a < b:

print “B is big” print “B value is”,b

**Output:**

B is big

B value is 15

**The *if ... else* statement**

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

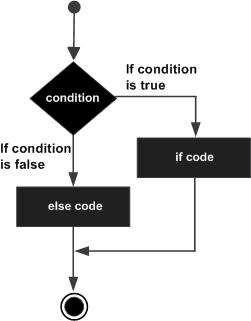
The *else* statement is an optional statement and there could be at most only one **else**

statement following **if**. **Syntax:**

if condition: statement(s)

else:

statement(s)



**Example:**

a=48 b=34

if a < b:

print “B is big” print “B value is”, b

else:

print “A is big” print “A value is”, a

print “END”

**Output:**

A is big

A value is 48 END

**Q) Write a program for checking whether the given number is even or not. Program:**

a=input("Enter a value: ") if a%2==0:

print "a is EVEN number" else:

print "a is NOT EVEN Number"

**Output-1: Output-2:**

Enter a value: 56 Enter a value: 27

a is EVEN Number a is NOT EVEN Number

**The *elif* Statement**

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

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

if condition1: statement(s)

elif condition2:

statement(s) else:

statement(s)

**Syntax:**

**Example:**

a=20 b=10 c=30

if a >= b and a >= c: print "a is big"

elif b >= a and b >= c: print "b is big"

else:

print "c is big"

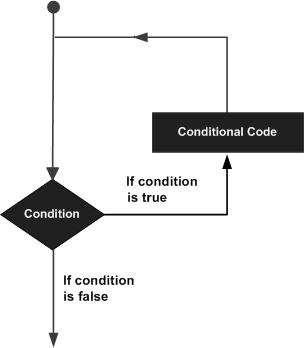
**Output:**c is big

**Repetition Structures:**

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement:



Python programming language provides following types of loops to handle looping requirements.

|  |  |
| --- | --- |
| **Loop Type** | **Description** |
| while loop | Repeats a statement or group of statements while a given condition is  TRUE. It tests the condition before executing the loop body. |
| for loop | Executes a sequence of statements multiple times and abbreviates the  code that manages the loop variable. |
| nested loops | You can use one or more loop inside any another while, for loop. |

**The *while* Loop**

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

**Syntax**

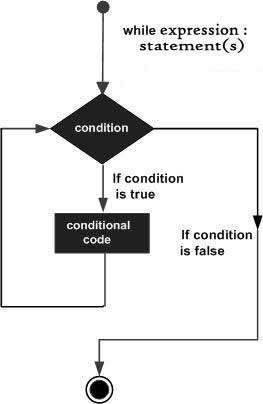
The syntax of a **while** loop in Python programming language is: while expression:

statement(s)

Here, **statement(s)** may be a single statement or a block of statements.

The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

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



**Example program for while loop:**

# Python 3 program to compute sum of digits in number.

# Function to get sum of digits

def getSum(n):

sum = 0

while (n != 0):

sum = sum + int(n % 10)

n = int(n/10)

return sum

# Driver code

n = 687

print(getSum(n))

**Example 2:**

count = 0

while (count < 9):

print 'The count is:', count

count = count +

**Using else with while loop**

Python allows us to use the else statement with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed, and the statement present after else block will be executed. The else statement is optional to use with the while loop. Consider the following example.

**Example 1**

i=1

**while**(i<=5):

    print(i)

    i=i+1

**else**:

    print("The while loop exhausted")

**Example 2**

i=1

**while**(i<=5):

    print(i)

    i=i+1

**if**(i==3):

**break**

**else**:

    print("The while loop exhausted")

**Output:**

1

2

In the above code, when the break statement encountered, then while loop stopped its execution and skipped the else statement.

**Example-3 Program to print Fibonacci numbers to given limit**

terms = **int**(input("Enter the terms "))

# first two intial terms

a = 0

b = 1

count = 0

# check **if** the number of terms is Zero or negative

**if** (terms <= 0):

   print("Please enter a valid integer")

elif (terms == 1):

   print("Fibonacci sequence upto",limit,":")

   print(a)

**else**:

   print("Fibonacci sequence:")

**while** (count < terms) :

       print(a, end = ' ')

       c = a + b

       # updateing values

       a = b

       b = c

       count += 1

**Output:**

Enter the terms 10

Fibonacci sequence:

0 1 1 2 3 5 8 13 21 34

**The *for* loop**:

The *for* loop is useful to iterate over the elements of a sequence. It means, the *for* loop can be used to execute a group of statements repeatedly depending upon the number of elements in the sequence. The *for* loop can work with sequence like string, list, tuple, range etc.

The syntax of the for loop is given below:

for var in sequence:

statement (s)

The first element of the sequence is assigned to the variable written after „for‟ and then the statements are executed. Next, the second element of the sequence is assigned to the variable and then the statements are executed second time. In this way, for each element of the sequence, the statements are executed once. So, the *for* loop is executed as many times as there are number of elements in the sequence.

**Example 1:**

fruits = ["apple", "banana", "cherry"]

for x in fruits:

print(x)

**Example-2: Program to print the sum of the given list.**

list = [10,30,23,43,65,12]

sum = 0

for i in list:

sum = sum+i

print("The sum is:",sum)

**For loop Using range() function**

**The range() function**

The **range()** function is used to generate the sequence of the numbers. If we pass the range(10), it will generate the numbers from 0 to 9. The syntax of the range() function is given below.

**Syntax:**

1. range(start,stop,step size)

* The start represents the beginning of the iteration.
* The stop represents that the loop will iterate till stop-1. The **range(1,5)** will generate numbers 1 to 4 iterations. It is optional.
* The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

***Example-1: Program to print numbers in sequence.***

for i in range(10):

print(i,end = ' ')

**Output:**

0 1 2 3 4 5 6 7 8 9

**Example - 2: Program to print table of given number.**

n = int(input("Enter the number "))

for i in range(1,11):

c = n\*i

print(n,"\*",i,"=",c)

**Example-3: Program to print even number using step size in range().**

n = int(input("Enter the number "))

for i in range(2,n,2):

print(i)

Output:

Enter the number 10

2

4

6

8

We can also use the **range()** function with sequence of numbers. The **len()** function is combined with range() function which iterate through a sequence using indexing. Consider the following example.

list = ['Peter','Joseph','Ricky','Devansh']

for i in range(len(list)):

    print("Hello",list[i])

**Output:**

Hello Peter

Hello Joseph

Hello Ricky

Hello Devansh

**Nested for loop in python**

Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax is given below.

**Syntax**

for**iterating\_var1**in**sequence:  #outer loop**

for**iterating\_var2**in**sequence:  #inner loop**

**#block of statements**

**#Other statements**

**Example- 1: Nested for loop**

**# User input for number of rows**

**rows = int(input("Enter the rows:"))**

**# Outer loop will print number of rows**

for**i**in**range(0,rows+1):**

**# Inner loop will print number of Astrisk**

for**j**in**range(i):**

print**("\*",end = '')**

print**()**

**Output:**

Enter the rows:5

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

**Example-2: Program to number pyramid.**

**rows = int(input("Enter the rows"))**

for**i**in**range(0,rows+1):**

for**j**in**range(i):**

print**(i,end = '')**

print**()**

**Output:**

1

22

333

4444

55555

**Using else statement with for loop**

Unlike other languages like C, C++, or Java, Python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

## **Example 1**

**for i in range(0,5):**

**print(i)**

**else:**

**print("for loop** completely exhausted, since there is no break.")

**Output:**

0

1

2

3

4

for loop completely exhausted, since there is no break.

The for loop completely exhausted, since there is no break.

## **Example 2**

**for** i **in** range(0,5):

**print**(i)

**break**;

**else**:**print**("for loop is exhausted");

**print**("The loop is broken due to break statement...came out of the loop")

In the above example, the loop is broken due to the break statement; therefore, the else statement will not be executed. The statement present immediate next to else block will be executed.

**Output:**

**0**

**Input Validation Loops**

 Input validation is the process of inspecting data that has been input to a program, to make sure it is valid before it is used in a computation. Input validation is commonly done with a loop that iterates as long as an input variable references bad data.

**Example:**

def calcAreaCircle(rad):

area = 3.14159 \* (rad \*\* 2)

print("Area of circle is %.2f" % area)

while True:

rad = int(input("Enter a radius: "))

calcAreaCircle(rad)

againYN = input("Perform another calculation (y/n? ")

while againYN not in ["y","n","Y","N","Yes","No"]:

againYN = input("Invalid choice. Please enter y or n: ")

if againYN in ["n","N","No"]:

break

**output:**

Enter a radius: 2  
  Area of circle is 12.57  
  Perform another calculation (y/n? pp  
  Invalid choice. Please enter y or n: y  
  Enter a radius: 3  
  Area of circle is 28.27  
  Perform another calculation (y/n? n

**Example: While True loop with function and lists**

all\_list = ["y","n","Y","N","Yes","No"]

no\_list = ["n","N","No"]

def calcAreaCircle(rad):

area = 3.14159 \* (rad \*\* 2)

print("Area of circle is %.2f" % area)

while True:

rad = int(input("Enter a radius: "))

calcAreaCircle(rad)

againYN = input("Perform another calculation (y/n? ")

while againYN not in all\_list:

againYN = input("Invalid choice. Please enter y or n: ")

if againYN in no\_list:

break

Output:

  Enter a radius: 2  
  Area of circle is 12.57  
  Perform another calculation (y/n? rr  
  Invalid choice. Please enter y or n: 2  
  Invalid choice. Please enter y or n: y  
  Enter a radius: 3  
  Area of circle is 28.27  
  Perform another calculation (y/n? n

**Python Loop Control Statements**

In Python, loops statements gives you a way execute the block of code repeatedly. But sometimes, you may want to exit a loop completely or skip specific part of a loop when it meets a specified condition. It can be done using loop control mechanism.

In Python, you have loop control statements that can be used to alter or control the flow of loop execution based on specified conditions. In Python we have following loop control statements –

* Break Statement
* Continue Statement
* Pass Statement

**Python Break Statement**

In Python, break statement inside any loop gives you way to break or terminate the execution of loop containing it, and transfers the execution to the next statement following the loop.

**Syntax:-**

**break**

**Example:-**

count = 0

while count < 10:

    count += 1

    if count == 5:

         break

    print("inside loop", count)

print("out of while loop")

In this above program, the variable **count**is initialized as **0**. Then a **while** loop is executed as long as the variable **count** is less than 10.

Inside the while loop, the **count** variable is incremented by **1** with each iteration

**count = count + 1**.

Next, we have an **if** statement that checks the variable **count** is equal to **5,** if it return **TRUE** causes loop to break or terminate.

Within the loop there is a **print()** statement that will execute with each iteration of the **while** loop until the loop breaks.

Then, there is a final **print()** statement outside of the **while** loop.

When we run this code, our output will be as follows –

**Output:-**

inside loop 1

inside loop 2

inside loop 3

inside loop 4

out of while loop

**Python Continue Statement**

The continue statement gives you way to skip over the current iteration of any loop. When a continue statement is encountered in the loop, the python interpreter ignores rest of statements in the loop body for current iteration and returns the program execution to the very first statement in th loop body. It does not terminates the loop rather continues with the next iteration.

**Syntax:-**

Continue

**Example:-**

ctr = 0

for ctr in range(10):

ctr = ctr + 1

if ctr == 5:

print("5 is skipped")

continue # continue here

print ("This won't be printed too.")

print('Number is ' + str(ctr))

print('Out of loop')

**Output:-**

Number is 1

Number is 2

Number is 3

Number is 4

5 is skipped

Number is 6

Number is 7

Number is 8

Number is 9

Number is 10

Out of loop

**Python pass statement**

In Python , the pass statement is considered as no operation statement, means it consumes the execution cycle like a valid python statement but nothing happens actually when pass is executed.The pass statement is much like a comment, but the python interpreter executes the pass statements like a valid python statements, while it ignores the comment statement completely. It is generally used to indicate “null” or unimplemented functions and loops body.

**Syntax:-**

**pass**

**Example:-**

numbers = [ 1, 2, 4, 3, 6, 5, 7, 10, 9 ]

#loop through the numbers

for number in numbers:

        #check if the number is even

if number % 2 == 0:

                #if even, then pass ( No operation )

                pass

else:

                #print the odd numbers

print 'Current number is:', number

**Output:-**

Current number is: 1

Current number is: 3

Current number is: 5

Current number is: 7

Current number is: 9