**Python Tutorial**

* Python Programming Language is a high-level and interpreted programming language
* Which was created by Guido Van Rossum in 1989? It was first released in 1991,
* Which results in a great general purpose language capable of creating anything from desktop software to web applications and frameworks?

**Why we Learn Python Programming:**

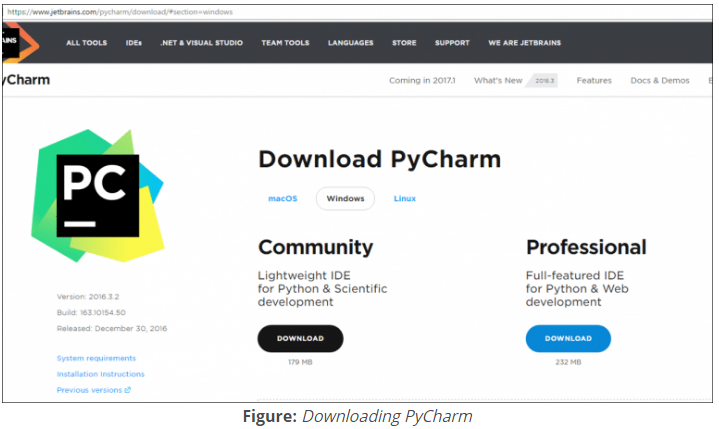
Python is a high-level dynamic programming language. It is quite easy to learn and provides powerful typing. Python code has a very ‘natural’ style to it, in that it is easy to read and understand.

* Highly readable language
* Clean visual layout
* Less syntactic exceptions
* Superior string manipulation
* Elegant and dynamic typing
* Interpreted nature
* Ideal for scripting and rapid application
* Fit for many platforms

Python can do more it is a very popular language in multiple domains like automation, big data, AI etc.

**Python Installation**: Go to the the link: <https://www.python.org/downloads/> and install the latest version on your machines.

Download and install PyCharm IDE:



PyCharm is an Integrated Development Environment (IDE) used in computer programming, specifically for the Python programming language. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django.

**Variables in Python:**

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc. Variables can be declared by any name or even alphabets like a, aa, abc, etc.

* How to declare and use a variable
* Re-declare a variable
* Concatenate variable
* Local & Global variable
* Delete a variable

**How to declare and use a variable:**

Example

A=100

Print(A)

**Re-declare a variable**

a=1000  
print(a)  
a=**'this is re declaring a variable'**print(a)

**Concatenate variable**

f=**'hi this is uma'**print(str(1000)+**' '**+ f)

**Local & Global variable**

In Python when you want to use the same variable for rest of your program or module you declare it a global variable, while if you want to use the variable in a specific function or method, you use a local variable.

Let's understand this difference between local and global variable with the below program.

1. Variable "f" is **global** in scope and is assigned value 101 which is printed in output
2. Variable f is again declared in function and assumes **local** scope. It is assigned value "I am learning Python." which is printed out as an output. This variable is different from the global variable "f" define earlier
3. Once the function call is over, the local variable f is destroyed. At line 12, when we again, print the value of "f" is it displays the value of global variable f=101

**Example:**

f=100  
print(f)  
**def** somefunction():  
 f=**'hi this is uma'** print(f)  
somefunction()  
print(f)

**Delete a variable**

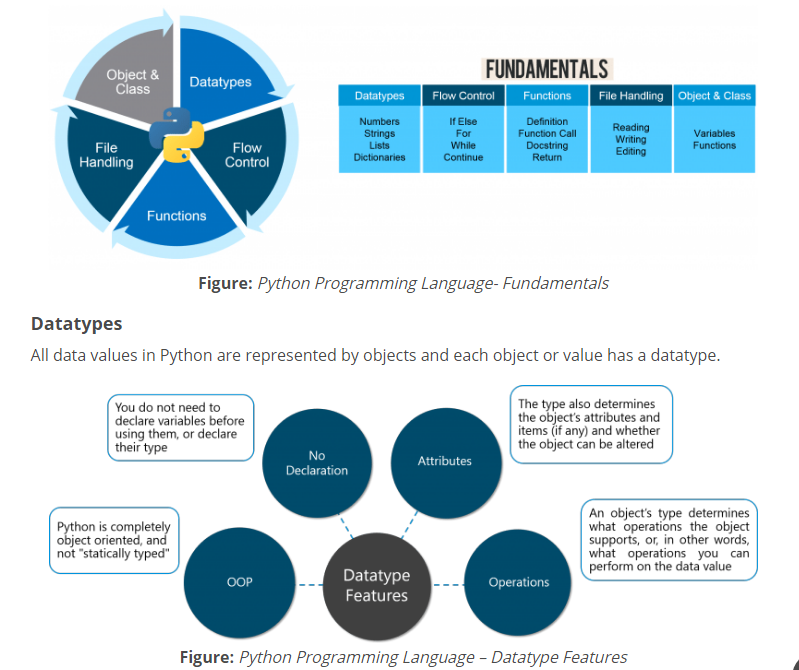
We can also delete a variable by using the command del “variable name”

Example:

f=11  
print(f)  
  
**del** f  
print(f)

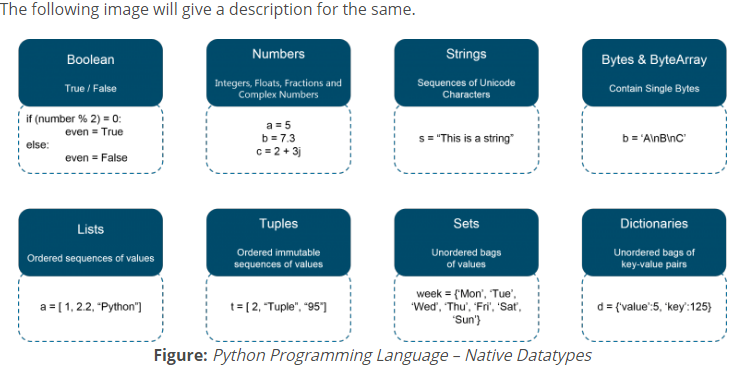
**Python Fundamentals:**

The following are the five fundamentals required to master Python:

1. Datatypes
2. Flow Control
3. Functions
4. File Handling
5. Object & Class****

**There are eight native datatypes in Python:**

1. Boolean
2. Numbers
3. Strings
4. Bytes & Byte Arrays
5. Lists
6. Tuples
7. Sets
8. Dictionaries



Examples of Data Types:

**Boolean:**

>>> number=[1,2,3,4,5]

>>> boolean=3 in number

>>> print(boolean)

True

**Strings:**

In python strings can be created as enclosing characters in double quotes.

Var =’this is uma’

* Accessing values in string
* Various string operators
* Some more examples
* Python string replace() method
* Changing upper and lower case strings
* Using “join” function for the string
* Reversing string
* Split string

## Accessing Values in Strings:

**Python does not support a character type**, these are treated as strings of length one, also considered as substring.

We use square brackets for slicing along with the index or indices to obtain a substring.

Example:

var1 = **"hi this is uma!"**var2 = **"Software Testing"**print (**"var1[0]:"**,var1[8:])  
print (**"var2[1:7]:"**,var2[1:6])

**Various String Operators:**

There is various string operators can be used in strings like

[] ---- Slice –it gives the letter from the given index

[:]---- Range slice –it gives the characters from the given range

In----Membership –returns true if a letter exist in the given string

Not in----Not exist in the string

r/R----Raw string suppresses actual meaning of escape characters.

%-Used for string format---- %r- it inserts the canonical string representation of the object

%s-it inserts the presentation string representation of the object

%d- it will format a number for display.

+ ---- it concatenates for 2 strings

\*---- Repeat (print the character twice)

**Python String replace() method:**

The method replace() returns the string the values old values replaces with new values.

Example:

old\_string=**'here i am writing c program'**new\_string=old\_string.replace(**'c'**,**'python'**)  
print(new\_string)

**Changing Upper and Lower case Strings:**

Strings can be changes either lower or upper case also.

Example:

old\_string=**'here i am writing c program'**print(old\_string.lower())

old\_string=**'here i am writing c program'**print(old\_string.upper())

**Using join function in the string:**

The join function is more flexible way to concatenate strings with the join condition we can join the characters from the strings.

Example:

f=**'i am uma'**print(**'@ '**.join(f))

Every character in the string is join with the @.

**Reverse String:**

By using the reverse function, you can reverse the string. For example, if we have string "12345" and then if you apply the code for the reverse function

Example:

string=**'my favorite star is gujun pyo'**print(**' '**.join(reversed(string)))

**o/p:** o y p n u j u g s i r a t s e t i r o v a f y m

**Split Strings:**

Split strings are another function that can be used by variable. Split to get the result.

Example:

string=**'my favorate star is gujun pyo'**print(string.split(**' '**))

>>> str1='Hello world welcome to python tutorial'

>>> str1

'Hello world welcome to python tutorial'

>>> str1[-2]

'a'

>>> str1[0:]

'Hello world welcome to python tutorial'

>>> str1[3:]

'lo world welcome to python tutorial'

>>> str1[:5]

'Hello'

>>> str1[5:2]

''

>>> str1[5:3]

''

>>> str1[-5:]

'orial'

>>> str1[:-5]

'Hello world welcome to python tut'

>>> str1[0:10]

'Hello worl'

**Lists :**

Is a collection which is ordered and changeable Allows duplicate members.

List Contents:

* [Quick example](https://thomas-cokelaer.info/tutorials/python/lists.html#quick-example)
* [Difference between append() and extend()](https://thomas-cokelaer.info/tutorials/python/lists.html#difference-between-append-and-extend)
* [Other list methods](https://thomas-cokelaer.info/tutorials/python/lists.html#other-list-methods)
* [Operators](https://thomas-cokelaer.info/tutorials/python/lists.html#operators)
* [Slicing](https://thomas-cokelaer.info/tutorials/python/lists.html#slicing)
* [List comprehension](https://thomas-cokelaer.info/tutorials/python/lists.html#list-comprehension)
* [Filtering Lists](https://thomas-cokelaer.info/tutorials/python/lists.html#filtering-lists)
* [Lists as Stacks](https://thomas-cokelaer.info/tutorials/python/lists.html#lists-as-stacks)
* [Lists as Queues](https://thomas-cokelaer.info/tutorials/python/lists.html#lists-as-queues)
* [How to copy a list](https://thomas-cokelaer.info/tutorials/python/lists.html#how-to-copy-a-list)

**Quick Example:**

a=[1,2,3,4,5,6]  
print(a[0])  
print(a[3])  
print(a[5])

[**Difference between append() and extend()**](https://thomas-cokelaer.info/tutorials/python/lists.html#difference-between-append-and-extend)**:**

Lists have several methods amongst which the **append()** and **extend()** methods. The former appends an object to the end of the list (e.g., another list) while the later appends each element of the iterable object (e.g., anothee list) to the end of the list.

**Example:**

string=[**'uma'**,**'devi'**,**'ram'**]  
string.append(**'shan'**)  
print(string)

string=[**'uma'**,**'devi'**,**'ram'**]  
string.extend(**'shan'**)  
print(string)

**Other List Methods:**

**Index():**

The **index()** methods searches for an element in a list.

string=[**'uma'**,**'devi'**,**'ram'**]  
print(string.index(**'devi'**))

**Insert():**

You can remove element but also insert element wherever you want in a list:

string=[**'uma'**,**'devi'**,**'ram'**]  
string.insert(3,**'shan'**)  
print(string)

**Remove():**

Similarly remove the first occurrence of an element from a list.

string=[**'uma'**,**'devi'**,**'ram'**]  
string.remove(**'devi'**)  
print(string)

### pop():

Remove the last element of a list by using:

string=[**'uma'**,**'devi'**,**'siva'**,**'ram'**,**'krishna'**,**'balu'**]  
string.pop()  
print(string)

o/p: ['uma', 'devi', 'siva', 'ram', 'krishna']

**Count():** count the no.of elements from a list

**Sort:** here is a sort() method that performs an in-place sorting:

string=[**'uma'**,**'devi'**,**'siva'**,**'ram'**,**'krishna'**,**'balu'**]  
string.sort()  
print(string)

**Reverse():** reverse the elements in the list

string=[**'uma'**,**'devi'**,**'siva'**,**'ram'**,**'krishna'**,**'balu'**]  
string.reverse()  
print(string)

Aliasing and cloning of list objects:

The process of giving another reference variable to the existing list is called aliasing.

Example:

*#aliasing*x=[10,20,30,50,60]  
y=x  
print(x)

Print(y)

The problem in this reference variable is for changing of any element in a list then those changes will reflect on the other reference variable.

To overcome this problem we are going to cloning process.

*#Cloning*x=[10,20,30,50,60]  
y=x  
y[2]=200  
print(y)

*# By using slice operator*x=[10,20,40,50]  
y=x[:]  
y[1]=1000  
print(x)  
print(y)

*#By using copy of function*x=[10,20,40,50]  
y=x.copy()  
y[1]=1000  
print(x)  
print(y)

Operator meant for aliasing

Copying meant for cloning.

>>> Countries=['India','Australia','Japan','North KOrea']

>>> Countries

['India', 'Australia', 'Japan', 'North KOrea']

>>>

>>> len(countries)

4

>>> print(len(countries))

4

>>> countries.append('Brazil')

>>> countries

['India', 'Australia', 'Japan', 'North KOrea', 'Brazil']

>>> countries.extend('Chain')

>>> countries

['India', 'Australia', 'Japan', 'North KOrea', 'Brazil', 'C', 'h', 'a', 'i', 'n']

SyntaxError: unexpected indent

>>> countries

['India', 'Australia', 'Japan', 'North KOrea', 'Brazil', 'C', 'h', 'a', 'i', 'n']

>>> countries.insert(2,'USA')

>>> countries

['India', 'Australia', 'USA', 'Japan', 'North KOrea', 'Brazil', 'C', 'h', 'a', 'i', 'n']

>>>

>>> countries[7]='Kochhi'

>>> countries

['India', 'Australia', 'USA', 'Japan', 'North KOrea', 'Brazil', 'C', 'Kochhi', 'a', 'i', 'n']

>>>

**#Tuples:**

A Tuple is a collection of Python objects separated by commas. In some ways a tuple is similar to a list in terms of indexing, nested objects and repetition but a tuple is immutable unlike lists which are mutable.

* [Packing and Unpacking](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#1)
* [Comparing tuples](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#2)
* [Using tuples as keys in dictionaries](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#3)
* [Deleting Tuples](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#4)
* [Slicing of Tuple](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#5)
* [Built-in functions with Tuple](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#6)
* [Advantages of tuple over list](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#7)

**Tuple assignment:** Tuple assignment has assigned more than one objects at a time.

Example:

tup1 = (**'Robert'**, **'Carlos'**,**'1965'**,**'Terminator 1995'**, **'Actor'**,**'Florida'**);  
tup2 = (1,2,3,4,5,6,7);  
print(tup1[0])  
print(tup1[1:4])  
print(tup2[1:5])

**Tuple Packing and Unpacking:**

In packing, we place value into new tuple while unpacking, we extract those values into revert back into variables.

**Example:**

x=(**'uma'**,**'devi'**)  
(name,sur\_name) = x  
print(name,sur\_name)

**Comparing Operators:**

A comparison operator in Python can work with tuples.

The comparison starts with a first element of each tuple. If they do not compare to =, < or > then it proceed to the second element and so on.

Example:

a=(2,4)  
b=(3,7)  
**if** (a>b): print(**'a is greater'**)  
**else**:  
 print(**'b is greater'**)

**Using tuples as keys in dictionaries:**

Since tuples are hashable, and list is not, we must use tuple as the key if we need to create a composite key to use in a dictionary.

**Example**: We would come across a composite key if we need to create a telephone directory that maps, first-name, last-name, pairs of telephone numbers, etc. Assuming that we have declared the variables as last and first number, we could write a dictionary assignment

a = {**'x'**:100, **'y'**:200}  
 b = list(a.items())  
 print(b)

**Deleting Tuple:**

Tuples are immutable cannot be deleted you cannot delete or remove items from a tuple. But deleting tuple entirely is possible by using the keyword

Del.

**Slicing of Tuple:**

To fetch specific sets of sub-elements from tuple or list, we use this unique function called slicing. Slicing is not only applicable to tuple but also in array and list.

x=(**'uma'**,**'devi'**,**'ram'**)  
print(x[2])

>>> tu=['uma',123,45,0.78,'cricbuzz']

>>> tu

['uma', 123, 45, 0.78, 'cricbuzz']

SyntaxError: invalid syntax

>>> tu

['uma', 123, 45, 0.78, 'cricbuzz']

>>> tu.append('sports')

>>> tu

['uma', 123, 45, 0.78, 'cricbuzz', 'sports']

>>> tu[1]='crl'

>>> tu

['uma', 'crl', 45, 0.78, 'cricbuzz', 'sports']

>>> tu[123]='has'

Traceback (most recent call last):

File "<pyshell#148>", line 1, in <module>

tu[123]='has'

IndexError: list assignment index out of range

**#Dictionaries:**

Dictionary is an unordered collection of **key-value pairs**. Dictionaries are used to handle large amount of data.

Python Dictionary are defined into two elements Keys and Values.

* Keys will be a single element
* Values can be a list or list within a list, numbers,

Python dictionary is an implementation of a hash table and is a **key-value** store. It is not ordered and it requires that the keys are hash table. Also, **it is fast for lookups by key.**

**Ex: phonebook, index.**

* [Python Dictionary Methods](https://www.guru99.com/python-dictionary-beginners-tutorial.html#1)
* [Copying dictionary](https://www.guru99.com/python-dictionary-beginners-tutorial.html#2)
* [Updating Dictionary](https://www.guru99.com/python-dictionary-beginners-tutorial.html#3)
* [Delete Keys from the dictionary](https://www.guru99.com/python-dictionary-beginners-tutorial.html#4)
* [Dictionary items() Method](https://www.guru99.com/python-dictionary-beginners-tutorial.html#5)
* [Sorting the Dictionary](https://www.guru99.com/python-dictionary-beginners-tutorial.html#6)
* [Python Dictionary in-built Functions](https://www.guru99.com/python-dictionary-beginners-tutorial.html#7)
* [Dictionary len() Method](https://www.guru99.com/python-dictionary-beginners-tutorial.html#8)
* [Variable Types](https://www.guru99.com/python-dictionary-beginners-tutorial.html#9)
* [Python List cmp() Method](https://www.guru99.com/python-dictionary-beginners-tutorial.html#10)
* [Dictionary Str(dict)](https://www.guru99.com/python-dictionary-beginners-tutorial.html#11)

Syntax:

dict={1:**'uma'**,2:**'devi'**,3:**'ram'**}  
print(dict)  
print(dict.items())  
print(dict.values())  
print(dict.keys())

Dictionaries are declared as curly braces and inside these keys and values are declared.

**Properties of Dictionary Keys**

There are two important points while using dictionary keys

* More than one entry per key is not allowed ( no duplicate key is allowed)
* The values in the dictionary can be of any type while the keys must be immutable like numbers, tuples or strings.
* Dictionary keys are case sensitive- Same key name but with the different case are treated as different keys in Python dictionaries.

Copying dictionary:

We can also copying the entire dictionary to new dictionary.

Dict = {**'Tim'**: 18,**'Charlie'**:12,**'Tiffany'**:22,**'Robert'**:25}  
Boys = {**'Tim'**: 18,**'Charlie'**:12,**'Robert'**:25}  
Girls = {**'Tiffany'**:22}  
studentX=Boys.copy()  
studentY=Girls.copy()  
print (studentX)  
print (studentY)

Updating Dictionary:

We can also update a dictionary by adding a new entry or key value pair.

Example:

Dict = {**'Tim'**: 18,**'Charlie'**:12,**'Tiffany'**:22,**'Robert'**:25}  
dict.update({**'uma'**:32})  
print(dict)

**Delete keys from the dictionaries:**

Python dictionaries give you the liberty to delete any element from the dictionary list.

Dict = {**'Tim'**: 18,**'Charlie'**:12,**'Tiffany'**:22,**'Robert'**:25}  
**del** dict[**'Charlie'**]  
print(dict)

**Dictionary Methods:**

Copy()—copy the entire dictionary to new dictionary

Update()--- update the dictionary by adding a new elements from the dictionary

Items()—returns a list of tuple pairs (key,value) in the dictionary

Sort()—sort the elements

Len()—number of pairs in the dictionary

Cmp()—compare values and keys.

Str()---make a dictionary into printable string format.

#dictionaries taking input from student details print on screen:

rec={}  
n=int(input(**'enter number of students:'**))  
i=1  
**while** i<=n:  
 name=input(**'enter student name:'**)  
 marks=input(**'enter % of marks:'**)  
 rec[name]=marks  
 i=i+1  
print(**'name of student'**,**'\t'**,**'% of marks'**)  
**for** x **in** rec:  
 print(**'\t'**,x,**'\t\t'**,rec[x])

dict={1:'uma',2:'shri',5:'ram'}

>>> dict

{1: 'uma', 2: 'shri', 5: 'ram'}

>>> del dict {1}

SyntaxError: invalid syntax

>>> del dict[1]

>>> dict

{2: 'shri', 5: 'ram'}

>>>

>>> Govt=['YSRCP','Congress','BJP','Janasena']

>>> govt

Traceback (most recent call last):

File "<pyshell#154>", line 1, in <module>

govt

NameError: name 'govt' is not defined

>>> Govt

['YSRCP', 'Congress', 'BJP', 'Janasena']

>>>

>>> print('the Indian government has',Govt)

the Indian government has ['YSRCP', 'Congress', 'BJP', 'Janasena']

>>> Govt['Congrass']='Prajarajyam'

Traceback (most recent call last):

File "<pyshell#158>", line 1, in <module>

Govt['Congrass']='Prajarajyam'

TypeError: list indices must be integers or slices, not str

>>> Govt['Congress']='Prajarajyam'

Traceback (most recent call last):

File "<pyshell#159>", line 1, in <module>

Govt['Congress']='Prajarajyam'

TypeError: list indices must be integers or slices, not str

>>>

**SET:**

A Python set is a slightly different concept from a list or a tuple. A set, in Python, is just like the mathematical set. It does not hold duplicate values and is unordered. However, it is not immutable, unlike a tuple.

Sets can user methods like pop(),discard(),clear(),remove(),add()

And Functions like len(),max()..

*#sets  
#Union*a=set([2,3,4,6,8])  
b=set([3,4,7,8,9])  
print(a|b)

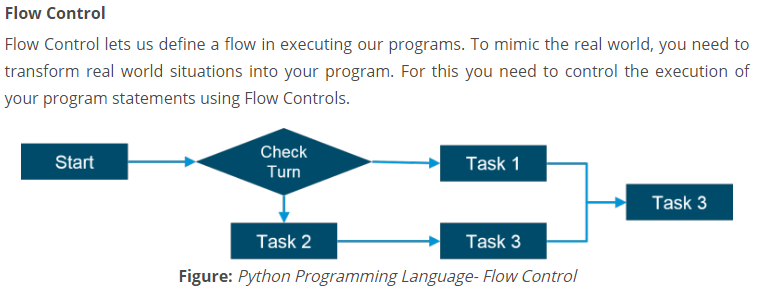
*#Intersection*a=set([2,3,4,6,8])  
b=set([3,4,7,8,9])  
print(a&b)

*#subset*a=set([2,3,4,6,8])  
b=set([3,4,7,8,9])  
print(a<b)

*#Difference*a=set([2,3,4,6,8])  
b=set([3,4,7,8,9])  
print(a-b)

Example:

set=[12,3,4,55,66,**'uma'**]  
12 **in** set  
print(**'yes i am in set'**)



**FLOW CONTROLS:**

There are six flow controls in python

* If
* For
* While
* Break
* Continue
* Pass

**If Statements:**

**Syn**: if expressions:

elif statements;

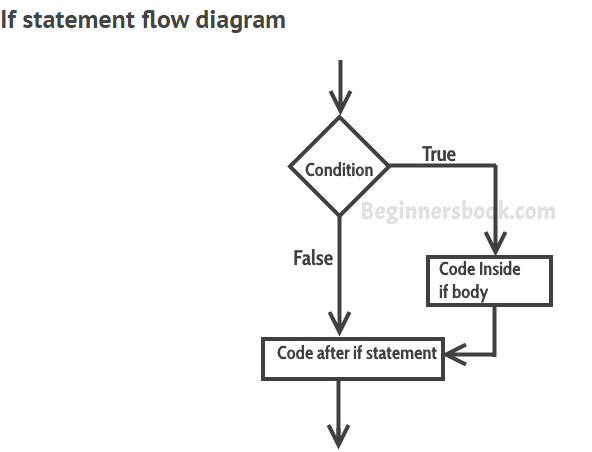
if expression:

elif statements;

if expressions:

elif statements;

In python IF Statement is used for decision making it will run the code only the statement is true.



**Example of If:**

flag =**True  
if** flag == **True**:  
 print(**'this is python if statement'**)  
 print(**'this is initial program for python code'**)

**IF..ELSE Statement:**

**If..else statements** are like extension of ‘if’ statements, with the help of if..else we can execute certain statements if condition is true and a different set of statements if condition is false. For example, you want to print ‘even number’ if the number is even and ‘odd number’ if the number is not even, we can accomplish this with the help of if..else statement.

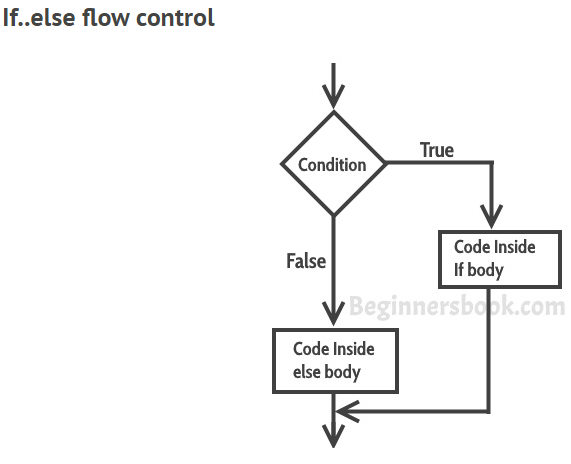
**Syntax:**

If condition:

Block of statements

Else

Block of statements



**Example:**

num = 10  
**if** num%2 == 1:  
 print(**'even number'**)  
**else**:  
 print(**'this is not even number'**)

**IF..ELIF..ELSE Statement:**

The if..elif..else statement is used when we need to check multiple conditions.

**Syntax:**

If condition1:

Block of statemnets

Elif condition2:

Block of statements

Elif condition3:

Block of statements

.

.

.

.

Else:

Block of statements

1. There can be multiple ‘elif’ blocks, however there is only ‘else’ block is allowed.

2. Out of all these blocks only one block\_of\_code gets executed. If the condition is true then the code inside ‘if’ gets executed, if condition is false then the next condition(associated with elif) is evaluated and so on. If none of the conditions is true then the code inside ‘else’ gets executed.

**Example:**

num = 1122  
**if** 9 < num < 99:  
 print(**"Two digit number"**)  
**elif** 99 < num < 999:  
 print(**"Three digit number"**)  
**elif** 999 < num < 9999:  
 print(**"Four digit number"**)  
**else**:  
 print(**"number is <= 9 or >= 9999"**)

**Nested if..else statements:**

When there is an if statement (or if..else or if..elif..else) is present inside another if statement (or if..else or if..elif..else) then this is calling the **nesting of control statements**.

## Nested if..else statement example

Here we have a if statement inside another if..else statement block. Nesting control statements makes us to check multiple conditions.

**Example:**

num=-90  
**if** num >0:  
 print(**'positive number'**)  
**else**:  
 print(**'the number is negative'**)  
 **if** -99 < num:  
 print(**'the two digit negative number'**)

**FOR Loop:**

A loop is a used for iterating over a set of statements repeatedly. In Python we have three types of loops **for**, **while** and **do-while**.

**Syntax:**

For <variable> in <sequence>

Body of loop that of statements

Loop repeatedly

Example: #Pattern Matching

For loop with range()

n=10  
**for** i **in** range(n):  
 **for** j **in** range(i):  
 print(**'\*'**,end=**" "**)  
 print(**' '**)  
**for** i **in** range(n,0,-1):  
 **for** j **in** range(i):  
 print(**'\*'**,end=**" "**)  
 print(**' '**)

For loop with else block:

**for** val **in** range(10):  
 print(val)  
**else**:  
 print(**"The loop has completed execution"**)

For loop with Nested Block:

**for** num1 **in** range(4):  
 **for** num2 **in** range(10, 15):  
 print(num1, **","**, num2)

**While Loop:**

**While loop** is used to iterate over a block of code repeatedly until a given condition returns false.

Syntax:

While condition:

Body of while

The body\_of\_while is set of Python statements which require repeated execution. These set of statements execute repeatedly until the given condition returns false.

## Flow of while loop:

1. First the given condition is checked, if the condition returns false, the loop is terminated and the control jumps to the next statement in the program after the loop.  
2. If the condition returns true, the set of statements inside loop are executed and then the control jumps to the beginning of the loop for next iteration.

Example:

#Lines in Upper case

line=[]  
**while True**:  
 l=input()  
 **if** l:  
 line.append(l.upper())  
 **else**:  
 **break**;  
**for** l **in** line:  
 print(l)

## Infinite while loop:

**Example 1:**

This will print the word ‘hello’ indefinitely because the condition will always be true.

Example:

**while True**:  
 print(**"hello"**)

num = 1  
**while** num<5:  
 print(num)

This will print ‘1’ indefinitely because inside loop we are not updating the value of num, so the value of num will always remain 1 and the condition num < 5 will always return true.

## Nested while loop in Python

When a while loop is present inside another while loop then it is called nested while loop. Lets take an example to understand this concept.

Example:

i = 1  
j = 5  
**while** i < 4:  
 **while** j < 8:  
 print(i, **","**, j)  
 j = j + 1  
 i = i + 1

## Python – while loop with else block

We can have a ‘else’ block associated with while loop. The ‘else’ block is optional. It executes only after the loop finished execution.

Example:

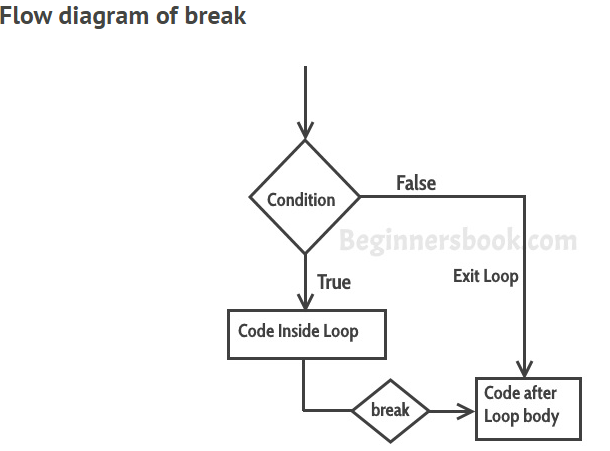
num = 10  
**while** num > 6:  
 print(num)  
 num = num-1  
**else**:  
 print(**"loop is finished"**)

**Break Statement:**

The **break statement** is used to terminate the loop when a certain condition is met.

For example,

Let’s say we are searching an element in a list, so for that we are running a loop starting from the first element of the list to the last element of the list. Using break statement, we can terminate the loop as soon as the element is found because why run the loop unnecessary till the end of list when our element is found. We can achieve this with the help of break statement (we will see this example programmatically in the example section below).

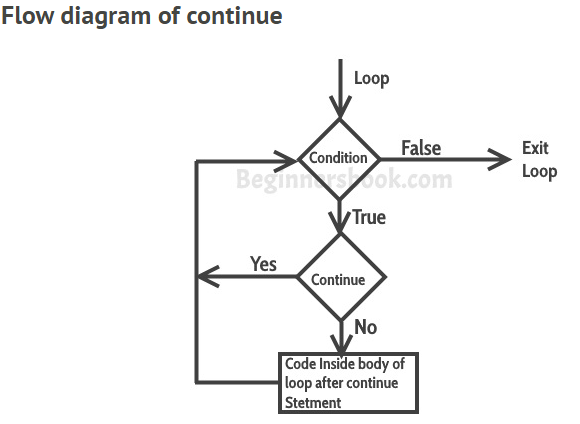


Example:

**for** num **in** [11, 9, 88, 10, 90, 3, 19]:  
 print(num)  
 **if**(num==88):  
 print(**"The number 88 is found"**)  
 print(**"Terminating the loop"**)  
 **break**

**Countinue Statement:**

The **continue statement** is used inside a loop to skip the rest of the statements in the body of loop for the current iteration and jump to the beginning of the loop for next iteration. The [break](https://beginnersbook.com/2018/01/python-break-statement/) and continue statements are used to alter the flow of loop, break terminates the loop when a condition is met and continue skip the current iteration.



Example:

**for** num **in** [22,34,54,6,8,9,89]:  
 **if** num %2 == 0:  
 **continue** print (num)

**Pass Statement:**

The **pass statement** acts as a **placeholder** and usually used when there is no need of code but a statement is still required to make a code syntactically correct.

Example:

**for** num **in** [20, 11, 9, 66, 4, 89, 44]:  
 **if** num%2 == 1:  
 **pass  
 else**:  
 print(num)

**Functions in Python:**

Functions in Python are used to utilize the code in more than one place in a program, sometimes also called method or procedures. Python provides you many inbuilt functions like print(), but it also gives freedom to create your own functions.

This means function performs the same task when called, which avoids the need of rewriting the same code again and again.

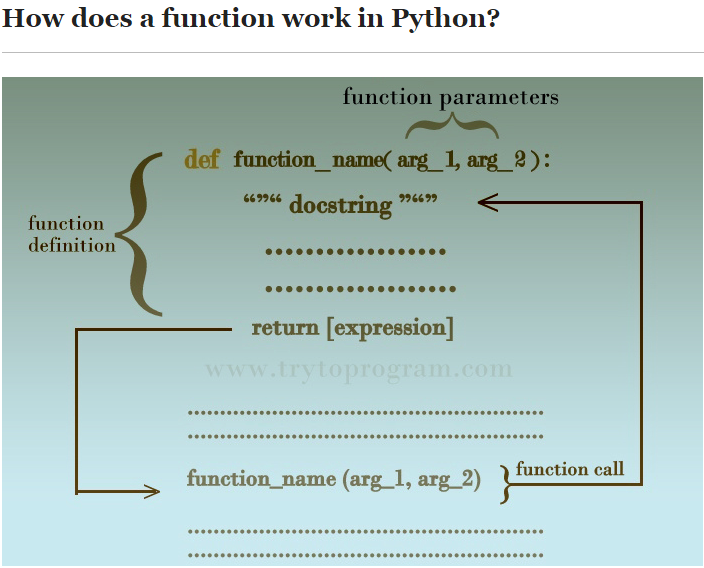
Functions help in the simplification of complex programs by breaking it into modules.

Remember that statements inside the function are not executed until the function is called. Function calls are like a detour in the flow of execution. Instead of going to the next statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.

**Types of Functions in python:**

There are two types of functions in python

1. Built-in functions: These are pre-defined functionalities of python.
2. User-defined functions: Users manually taking the functions are known as user-defined functions.



A Python function consists of **function definition** where the functionality of a function is defined. Function definition as seen above consists of a **function name,** **function arguments, docstring, code statements, and the return statement.**

Once a function is defined, we need to call the function in the main program to execute the function. For that, there is function call statement.

Syntax:

def function\_name (arg 1, arg2,...):

"""docstring"""

statement(s)

return [expression]

* A function definition in Python starts with the keyword def followed by the function name.
* Depending on the operational requirement or user’s choice, a function may or may not have parameters inside the parentheses just after function name and it is followed by a colon (:).
* The information into the functions can be passed as the parameters. The parameters are specified in the parenthesis. We can give any number of parameters or arguments in parenthesis those are separated by comma (,).
* After that is an optional statement – the documentation string of function or docstring followed by the indented block of codes or statements designated for the operation of the function.
* The return statement is to exit the function and return the computed value, making the program flow to jump wherever the function is called.

## ****Function call:****

Defining a function means giving the function a name, specifying the parameters and defining the operation of function with code statements.

Once a function is defined, we need to call the function wherever and whenever we need to execute it. To call a function, we simply need to write the function and supply corresponding arguments for the parameters we included in the function definition.

Example:

Function\_name(arg1,arg2…argn)

*# function define and function call***def** printme(str):  
 *"hi"* print(str)  
 **return**;  
printme(**'this is function calling'**)  
printme(**'this second time function calling'**)

**Docstring:**

The first statement of the body function is a documentation string is called as docstring. This is the header of the function.

Syntax:

Function\_name.\_\_docstring\_\_

**Note:**  The doc attribute has two underscores (\_\_) before and after. Using single underscore (\_) will raise an error.

**def** Hello():

*""" Hello World """* print (**'Hi'**)  
print (**"The doctsring of the function Hello is: "**+ Hello.\_\_doc\_\_)

**Global Variables & Local Variable:**

Python supports two types of variables:

* + Global variables
  + Local variables.

Variable declares outside of the function is known as global variable

x=100 *# Global Variable***def** fun1():  
 print(x)  
**def** func2():  
 print(x)  
fun1()  
func2()

the variable declares inside of a function is known as Local variable.

*# Local variable***def** fun1():  
 a=100  
 print(a)  
**def** fun2():  
 b=200  
 print(b)  
fun1()  
fun2()

**Recursive functions in a python:**

Recursion is the process of defining something in terms of itself.

A function can call another function this type of construct is called as recursive function.

*# Factorial of a number***def** factorial(n):  
 **if** n==1:  
 **return** 1  
 **else**:  
 **return** n\* factorial(n-1)  
num=4  
print(**'the factorial value of number is:'**,factorial(num))

calc\_factorial(4) # 1st call with 4

4 \* calc\_factorial(3) # 2nd call with 3

4 \* 3 \* calc\_factorial(2) # 3rd call with 2

4 \* 3 \* 2 \* calc\_factorial(1) # 4th call with 1

4 \* 3 \* 2 \* 1 # return from 4th call as number=1

4 \* 3 \* 2 # return from 3rd call

4 \* 6 # return from 2nd call

24 # return from 1st call

**Advantages:**

1. Recursive functions looks code clean and elegant.
2. Complex tasks can be broken down into simpler
3. Sequence generation is easier with recursion than using some nested iteration.

**Dis-advantages:**

1. Sometimes the logic behind the recursion is hard to follow through.
2. Recursive calls are very expensive means they take up lot of memory and time.
3. Recursive functions are hard to debug.

🡪Python function with arguments and return values:

This method allows passing the arguments from the function while calling the function. This type of functions in python returns the values when we call the function.

Example:

*# Python Function with arguments and Return value Example***def** function(a,b):  
 sum=a+b  
 **return** sum  
print(**'after calling the outside function:'**,function(20,30))

### 🡪Python Function with No argument and No Return value:

This type of function in python, while defining, declaring, or calling the function and we won’t pass any parameters /any arguments to the function. This type of function wont return any value if we call the function.

Example:

*#Python Function with No argument and No Return value Example***def** add():  
 a=20  
 b=30  
 sum=a+b  
 print(**'the sum of two numbers:'**,sum)  
add()

🡪Python function with no arguments and with a return value:

This type of function in python, we won’t pass any arguments while declaring, defining and calling the function. When we call the function it will return some value.

Example:

*#Python Function with no argument and with a Return value***def** multi():  
 a=20  
 b=21  
 multiplication=a\*b  
 **return** multiplication  
print(**'after return value call the function:'**,multi())

🡪Python function with arguments and no return value:

This type of function in python, we are passing the parameters or arguments while declaring, defining and calling the function

*#Python Function with argument and No Return value***def** multi(x,y):  
 multi=x\*y  
 print(**'the multiplication is:'**,multi)  
multi(2,4)

*# Passing parameters and return result from the function.*

**def** multi(x,y):  
 multi=x\*y  
 print(**'the multiplication is:'**,multi)  
multi(2,4)  
**def** add(x,y):  
 add=x+y  
 print(**'the addition is:'**,add)  
add(20,2)  
**def** sub(a,b):  
 sub=a-b  
 print(**'the subtraction of two values:'**,sub)  
sub(20,3)  
**def** name(firstname,lastname):  
 separator=**','** result=firstname+separator+lastname  
 **return** result  
print(**'the result string is:'**,name(**'uma'**,**'devi'**))  
print(**'the result string is:'**,name(**'rama'**,**'krishna'**))

Python Math functions:

Fabs(x)- it returns the absolute value of x.

Ceil()- it returns the smallest integer value greater than or equals to x

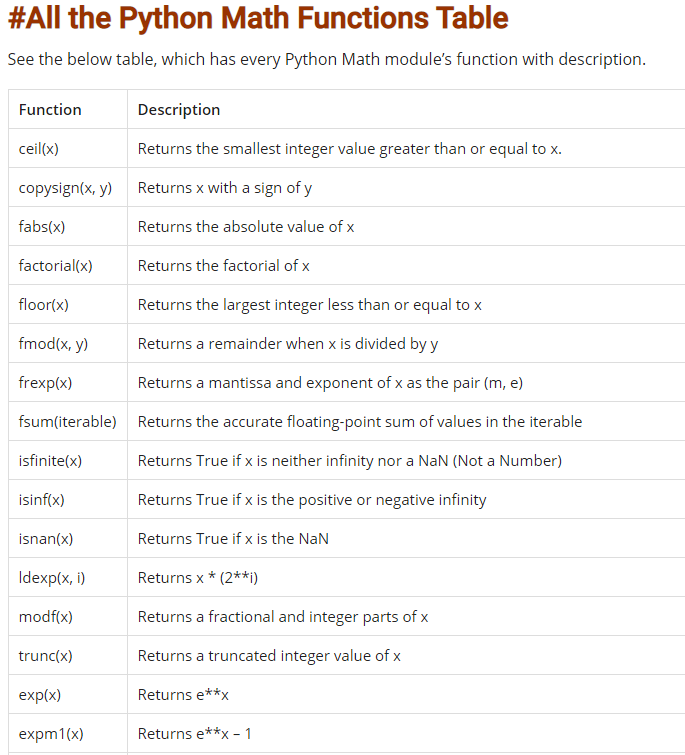
Floor()- it returns the largest integer value less than or equals to x

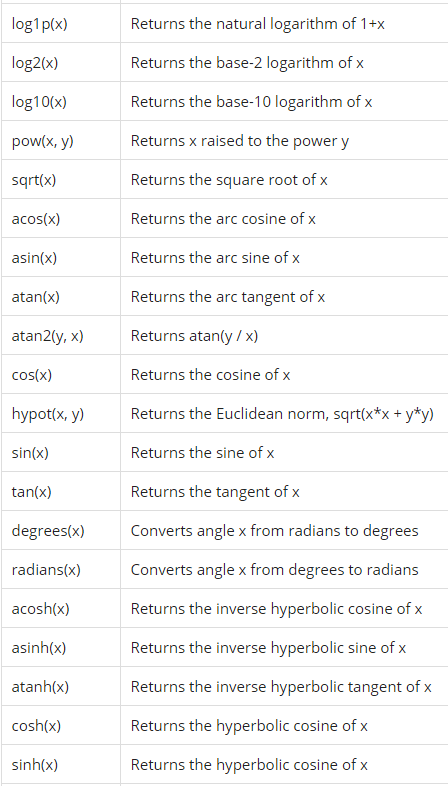
Factorial()- it returns the factorial of x.

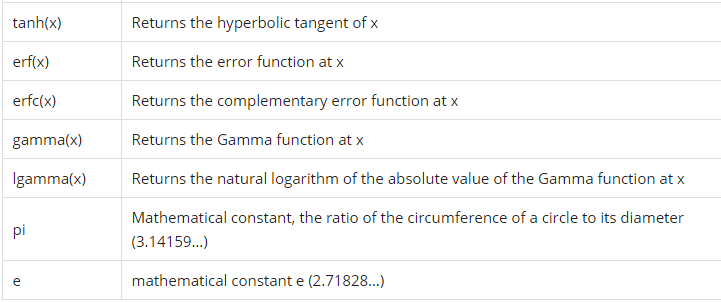
Gcd()- this function is used to compute the greatest divisor of two numbers from the mentioned arguments. This function works only python 3.5 and above

Examples:

*#Math functions  
#Floor()***'''import math  
data =21.6  
print('the floor value of 21.6 is:',math.floor(data))  
'''***#ceil***'''import math  
data =21.6  
print('the ceil value of 21.6 is:',math.ceil(data))  
'''***#pi value***'''import math  
print('the pi value is:',math.pi)  
'''  
  
import** math  
data=24.6  
print(**'the floor value:'**,math.floor(data))  
print(**'the ceil value:'**,math.ceil(data))  
print(**'the fabs value:'**,math.fabs(data))







**Function call by value and call by reference:**

These are passing parameter techniques; python behaves like a call by value.

--passed a parameter in to the function, it is called as call by reference.

The Function which calls another Function is called **Calling Function** and function which is called by another Function is call **Called Function**.

Example:

*#passed by reference***'''student={'uma':100,'rama':101,'shan':102,'nandu':103}  
def test(student):  
 new={'sai':104,'bhagya':105}  
 student.update(new)  
 # print('inside the function',student)  
 return  
test(student)  
print('outside the function:',student)  
'''***#calling function multiple times*list={12,3,4,5,6,7,8,9,0,2,44}  
**def** test(list):  
 new={50,45,11,1,2}  
 list.update(new)  
 print(**'inside function is:'**,list)  
 **return**test(list)  
print(**'the outside function is:'**,list)  
new={30,31,32,33,34,43}  
list.update(new)  
print(**'updated list:'**,list)  
test(list)  
print(**'here the values are updated :'**,list)  
prev={101,102,103,2,3,4,5,6,7,8,90}  
list.update(prev)  
print(**'update the list:'**,list)

**Python Built-in Functions:**

1. **\_\_import\_\_( ):**  
   This is an advanced function which is invoked when we use the import statement.
2. **abs( ):**  
   Returns the absolute value of a number. If the argument is a complex number, its magnitude is returned.
3. **all( ):**  
   Returns True if all elements of the iterable are true or if the iterable is empty.
4. **any(iterable):**  
   Returns True if any element of the iterable is true.
5. **ascii( ):**  
   It returns printable version of string ‘str’.
6. **bin( ):**  
   Converts an integer number to a binary string.
7. **bool( ):**  
   Converts a value to a Boolean, using the standard truth testing procedure.
8. **bytearray( ):**  
   Returns an array of given byte size. The bytearray type is a mutable sequence of integers in the range 0 <= x < 256.
9. **bytes( ):**  
   Return a new bytes object, which is an immutable sequence of integers in the range 0 <= x < 256. It is an immutable version of bytearray.
10. **callable( ):**  
    Checks if the object is Callable. Returns True if the object argument appears callable, False if not.
11. **chr(i):**  
    Returns the string of one character whose Unicode codepoint is the integer i. For example, chr(98) returns the string ‘b’.
12. **classmethod( ):**  
    Returns a class method for a function. A class method receives the class as implicit first argument, just like an instance method receives the instance.
13. **compile():**  
    Returns a Python code object. Compiles the source into a code or AST (Abstract Syntax Tree) object.
14. **complex():**  
    Creates a complex number.
15. **delattr( ):**  
    Deletes attribute from the object. It deletes the named attribute, provided the object allows it.
16. **dict():**  
    Creates a dictionary.
17. **dir( ):**  
    It tries to return attributes of the object. Without arguments, returns the list of names in the current local scope. With an argument, attempts to return a list of valid attributes for that object.
18. **divmod(a, b):**  
    Returns a tuple of quotient and remainder.
19. **enumerate ( ):**  
    Returns an enumerate object for an iterable.
20. **eval():**  
    The eval function lets a python program run python code within itself.
21. **exec():**  
    Executes dynamically created program. It supports dynamic execution of a Python code.
22. **filter( ):**  
    It constructs iterator from elements of an iterable for which function returns true.
23. **float( ):**  
    It converts a string or a number to floating point.
24. **format( ):**  
    Converts a value to a formatted representation i.e. it returns formatted representation of a value.
25. **frozenset( [iterable] ):**  
    Return a frozenset object, optionally with elements taken from iterable.
26. **getattr( ):**  
    It returns the value of the named attribute of an object.
27. **globals( ):**  
    It returns a dictionary of the current global symbol table.
28. **hasattr( ):**  
    Returns whether the object has named attribute or not.
29. **hash(object):**  
    Returns hash value of the object.
30. **help([object]):**  
    It invokes the built-in help system. If no argument is given, the interactive help system starts on the interpreter console else a help page on the object is generated.
31. **hex( ):**  
    Converts an integer to the corresponding hexadecimal number.
32. **id(object):**  
    It returns the identity of the object.
33. **input( ):**  
    It reads and returns a line of string.
34. **int( ):**  
    Converts a number or string to an integer.
35. **isinstance( ):**  
    Checks if an object is an instance of the class. Return true if the object is an instance of the class.
36. **issubclass(class, classinfo):**  
    Checks if a class is a subclass of another. Returns true if class is a subclass of the class classinfo.
37. **itr( ):**  
    It returns an iterator for an object.
38. **len( ):**  
    It returns length of the object.
39. **list( ):**  
    Used to create a list in Python.
40. **locals( ):**  
    Updates and returns a dictionary representing the current local symbol table.
41. **map( ):**  
    Returns an iterator that applies the function to every item of iterable, yielding the results.
42. **max( ):**  
    Returns the largest item in the iterable.
43. **memoryview( ):**  
    returns memory view of an object supplied as an argument.
44. **min( ):**  
    Returns the smallest item in the iterable.
45. **next( ):**  
    Retrieves the next item from the iterator by calling its **\_\_next\_\_()** method.
46. **object( ):**  
    Creates a new featureless object.
47. **oct( ):**  
    Converts an integer to an octal string.
48. **open( ):**  
    Opens a file and return the corresponding stream.
49. **ord( ):**  
    Returns an integer representing the Unicode code point of the character.
50. **pow( x, y):**  
    Returns x to the power y.
51. **print( ):**  
    Used to print the given object.
52. **property( ):**  
    Returns the property attribute.
53. **range( ):**  
    Returns a sequence of integers between given range.
54. **repr( ):**  
    It returns a string containing a printable representation of an object.
55. **reversed( ):**  
    Returns an iterable in reversed order.
56. **round(x[, n]):**  
    Returns the floating point value x rounded to n digits after the decimal point.
57. **set( ):**  
    Returns a new Python set.
58. **setattr( ):**  
    Sets the value of an attribute of the object.
59. **slice( ):**  
    Returns a slice object representing the set of indices specified by range(start, stop, step).
60. **sorted( ):**  
    Returns sorted list from a given iterable.
61. **staticmethod( ):**  
    It creates static method from a function.
62. **str( ):**  
    Returns a string version of an object.
63. **sum( ):**  
    Adds items of an iterable from left to right and returns the total.
64. **super( ):**  
    Returns a proxy object that delegates method calls to a parent or sibling class.
65. **tuple( ):**  
    Creates a new tuple.
66. **vars( ):**  
    Returns \_\_dict\_\_ attribute of a class.
67. **zip( ):**  
    Returns an iterator of tuples

**Python Nested functions:**

Python supports the concept of “Nested functions” or “Inner Functions” which is simply a function defined inside of another function.

**Define an Inner Function:**

To define an inner function in python, we simply create function inside of another function using “def” keyword.

Example:

**def** fun1():------Outer Function  
 print(**'this is outer function'**)  
 **def** fun2():------Inner Function  
 print(**'this is inner function'**)  
 fun2()----call the inner function  
fun1()-----call the outer function

Here function2 () has been defined inside of function1 () making it an inner function.

To call function2 (), we must first call function1 (). The function1 () will then a go ahead and call the function2 ().

Here the outer function has to be called in order for the inner function to execute. If the outer function is not called, the inner function will never execute.

**Python Lambda Functions:**

Anonymous functions are a function without a name. As we know that “def” is the keyword for all regular functions. & “lambda” is the keyword for to create anonymous functions.

Lambda functions only passing one expressions.

Syntax: lambda arguments: expression

Lambda functions can have many numbers of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever objects are required.

Example:

**f=lambda x:x+x  
print(f(5))**

Every anonymous function you define in Python will have 3 essential parts:

* The lambda keyword.
* The parameters (or bound variables), and
* The function body.

A lambda function can have any number of parameters, but the function body can only contain **one** expression.

Uses of Lambda function:

When we require a nameless function for a short period of time.

Lambda functions used for built in functions like filter (), map () ..etc

**Filter ():**

The filter () function in python takes in a function and list as arguments.

The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.

Example:  
*# Even\_numbers list::*

my\_list=[2,4,33,22,44,55,66,2,3,1,2,65,39,4]  
result=list(filter(**lambda** x:(x % 2==0),my\_list))  
print(result)

**Map ():**

The map () in python takes in a function and a list.

The function is called with all items in a list and a new list is returned which contains items returned by that function for each item.

Example:

sequences = [10,2,8,7,5,4,3,11,0, 1]  
filtered\_result = map (**lambda** x: x\*x, sequences)  
print(list(filtered\_result))

**Reduce ():**

The reduce () function is defined the funtools module. Like map (), filter () functions same reduce () function also receives two arguments and a function and an iterable.

Example:

import functools

def mult(x,y):

print("x=",x," y=",y)

return x\*y

fact=functools.reduce(mult, range(1, 4))

print ('Factorial of 3: ', fact)

**Python Modules:**

Modules in Python are simply Python files with a .py extension. The name of the module will be the name of the file. A Python module can have a set of functions, classes or variables defined and implemented.

We can use any Python source file as a module by executing an import statement in some other Python source file.

Example:

**from** math **import** sqrt,factorial  
print(sqrt(16))  
print(factorial(5))

A python from statement lets you import specific attributes from a module.

**# built in modules in python:**

Dir (): The dir () built-in function returns a sorted list of strings containing the names defined by a module. The list contains the names of all the modules, variables and functions that are defined in a module.

**import** math  
print(dir(math))

Example:

*# Built-in module functions::  
  
# importing built-in module math***import** math  
  
*# using square root(sqrt) function contained in math module*print(math.sqrt (25))  
print (math.pi)  
  
*# 2 radians = 114.59 degreees*print (math.degrees (2))  
  
*# 60 degrees = 1.04 radians*print (math.radians (60))  
  
*# Sine of 2 radians*print (math.sin (2))  
  
*# Cosine of 0.5 radians*print (math.cos (0.5))  
  
*# Tangent of 0.23 radians*print (math.tan (0.23))  
  
*# 1 \* 2 \* 3 \* 4 = 24*print (math.factorial (4))  
  
*# importing built in module random***import** random  
  
*# printing random integer between 0 and 5*print (random.randint (0, 5))  
  
*# print random floating point number between 0 and 1*print (random.random ())  
  
*# random number between 0 and 100*print (random.random () \* 100)  
  
List = [1, 4, **True**, 800, **"python"**, 27, **"hello"**]  
  
*# using choice function in random module for choosing  
# a random element from a set such as a list*print (random.choice (List))  
  
*# importing built in module datetime***import** datetime  
**from** datetime **import** date  
**import** time  
  
*# Returns the number of seconds since the  
# Unix Epoch, January 1st 1970*print (time.time ())  
  
*# Converts a number of seconds to a date object*print (date.fromtimestamp (454554))

Aliasing import statement::

**import** math **as** m  
print(**'tha math value of pi is :'**,m.pi)

Import all name:

Import \*

**from** math **import** \*  
print(**'the squareroot value is:'**,sqrt(25))  
print(**'the pi value is:'**,pi)  
print(**'the sin value is:'**,sin(100))  
print(**'the cos value is:'**,cos(200))  
print(**'the factorial of value is:'**,factorial(5))

Python module search path:

While importing a module, Python looks at several places. Interpreter first looks for a built-in module then (if not found) into a list of directories defined in sys.path. The search is in this order.

* The current directory.
* PYTHONPATH (an environment variable with a list of directory).
* The installation-dependent default directory.

Example:

Import sys

Sys.path

**Reloading a module:**

The python interpreter imports a module only once during a session. This makes things more efficient.

Example: here we see the multiple imports, here our code import runs only one time.

Python provides a neat way of doing this. We can use the reload () function inside the imp module to reload a module. This is how it’s done.

**import** math  
*#this is got executed---1. only once during a session***import** math  
**import** math

import imp

>>> import my\_module

This code got executed

>>> import my\_module

>>> imp.reload(my\_module)

This code got executed

<module 'my\_module' from '.\\my\_module.py'>

Reload (module\_name)

**Python Packages:**

We organize a large number of files in different folders and subfolders based on some criteria, so that we can find and manage them easily. In the same way, a package in Python takes the concept of the modular approach to next logical level. As you know, a [module](https://www.tutorialsteacher.com/python/python-module) can contain multiple objects, such as classes, functions, etc. A package can contain one or more relevant modules. Physically, a package is actually a folder containing one or more module files.

Let's create a package named mypackage, using the following steps:

* Create a new folder named D:\MyApp.
* Inside MyApp, create a subfolder with the name 'mypackage'.
* Create an empty \_\_init\_\_.py file in the mypackage folder.
* Using a Python-aware editor like IDLE, create modules greet.py and functions.py with following code:

MyApp

(Directory name)

MyPackage

(Package name)

\_\_int\_\_.py

Functions.py (Module)

greet.py (Module)

Package folder structure

**NumPy:**

NumPy stands for ‘Numerical Python’ or ‘Numeric Python’. It is an open source module of Python which provides fast mathematical computation on arrays and matrices.

NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation. Numpy can be imported into the notebook using

>>> Import numpy as np

NumPy’s main object is the homogeneous multidimensional array. It is a table with same type elements, i.e, integers or string or characters (homogeneous), usually integers. In NumPy, dimensions are called axes. The number of axes is called the rank.

There are several ways to create an array in NumPy like np.array, np.zeros, no.ones, etc. Each of them provides some flexibility.

Some of the elements of numpy:

np.array

np.ones

np.full

np.arange

np.linspace

np.random.rand(2,3)

np.empty((2,3))

some of the important attributes of a NumPy object are:

1. **Ndim:** displays the dimension of the array
2. **Shape:** returns a tuple of integers indicating the size of the array
3. **Size:** returns the total number of elements in the NumPy array
4. **Dtype**: returns the type of elements in the array, i.e., int64, character
5. **Itemsize:** returns the size in bytes of each item
6. **Reshape**: Reshapes the NumPy array

NumPy array elements can be accessed using indexing. Below are some of the useful examples:

* A[2:5] will print items 2 to 4. Index in NumPy arrays starts from 0
* A[2::2] will print items 2 to end skipping 2 items
* A[::-1] will print the array in the reverse order
* A[1:] will print from row 1 to end

The session covers these and some important attributes of the NumPy array object in detail.

**Pandas:**

Python is increasingly being used as a scientific language. Matrix and vector manipulations are extremely important for scientific computations. Both NumPy and Pandas have emerged to be essential libraries for any scientific computation, including machine learning, in python due to their intuitive syntax and high-performance matrix computation capabilities.

In this post, we will provide an overview of the common functionalities of NumPy and Pandas. We will realize the similarity of these libraries with existing toolboxes in R and MATLAB. This similarity and added flexibility have resulted in wide acceptance of python in the scientific community lately. Topic covered in the blog are:

1. Overview of NumPy
2. Overview of Pandas
3. Using Matplotlib

This post is an excerpt from a live hands-on training conducted by CloudxLab on 25th Nov 2017. It was attended by more than 100 learners around the globe. The participants were from countries namely; United States, Canada, Australia, Indonesia, India, Thailand, Philippines, Malaysia, Macao, Japan, Hong Kong, Singapore, United Kingdom, Saudi Arabia, Nepal, & New Zealand.

**What is NumPy?**

NumPy stands for ‘Numerical Python’ or ‘Numeric Python’. It is an open source module of Python which provides fast mathematical computation on arrays and matrices. Since, arrays and matrices are an essential part of the Machine Learning ecosystem, NumPy along with Machine Learning modules like Scikit-learn, Pandas, Matplotlib, TensorFlow, etc. complete the Python Machine Learning Ecosystem.

NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation. Numpy can be imported into the notebook using

|  |  |
| --- | --- |
| 1 | >>> import numpy as np |

NumPy’s main object is the homogeneous multidimensional array. It is a table with same type elements, i.e, integers or string or characters (homogeneous), usually integers. In NumPy, dimensions are called axes. The number of axes is called the rank.

There are several ways to create an array in NumPy like np.array, np.zeros, no.ones, etc. Each of them provides some flexibility.

|  |  |
| --- | --- |
| Command to create an array | Example |
| np.array | |  |  | | --- | --- | | 1  2  3  4  5  6  7 | >>> a = np.array([1, 2, 3])  >>> type(a)  <type 'numpy.ndarray'>    >>> b = np.array((3, 4, 5))  >>> type(b)  <type 'numpy.ndarray'> | |
| np.ones | |  |  | | --- | --- | | 1  2  3  4 | >>> np.ones( (3,4), dtype=np.int16 )  array([[ 1,  1,  1,  1],        [ 1,  1,  1,  1],        [ 1,  1,  1,  1]]) | |
| np.full | |  |  | | --- | --- | | 1  2  3  4 | >>> np.full( (3,4), 0.11 )  array([[ 0.11,  0.11,  0.11,  0.11],    [ 0.11,  0.11,  0.11,  0.11],    [ 0.11,  0.11,  0.11,  0.11]]) | |
| np.arange | |  |  | | --- | --- | | 1  2  3  4  5  6 | >>> np.arange( 10, 30, 5 )  array([10, 15, 20, 25])    >>> np.arange( 0, 2, 0.3 )  # it accepts float arguments  array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8]) | |
| np.linspace | |  |  | | --- | --- | | 1  2 | >>> np.linspace(0, 5/3, 6)  array([0. , 0.33333333 , 0.66666667 , 1. , 1.33333333  1.66666667]) | |
| np.random.rand(2,3) | |  |  | | --- | --- | | 1  2  3 | >>> np.random.rand(2,3)  array([[ 0.55365951,  0.60150511,  0.36113117],        [ 0.5388662 ,  0.06929014,  0.07908068]]) | |
| np.empty((2,3)) | |  |  | | --- | --- | | 1  2  3 | >>> np.empty((2,3))  array([[ 0.21288689,  0.20662218,  0.78018623],        [ 0.35294004,  0.07347101,  0.54552084]]) | |

Some of the important attributes of a NumPy object are:

1. **Ndim:** displays the dimension of the array
2. **Shape:** returns a tuple of integers indicating the size of the array
3. **Size:** returns the total number of elements in the NumPy array
4. **Dtype**: returns the type of elements in the array, i.e., int64, character
5. **Itemsize:** returns the size in bytes of each item
6. **Reshape**: Reshapes the NumPy array

NumPy array elements can be accessed using indexing. Below are some of the useful examples:

* A[2:5] will print items 2 to 4. Index in NumPy arrays starts from 0
* A[2::2] will print items 2 to end skipping 2 items
* A[::-1] will print the array in the reverse order
* A[1:] will print from row 1 to end

The session covers these and some important attributes of the NumPy array object in detail.

**Why NumPy and Pandas over regular Python arrays?**

In python, a vector can be represented in many ways, the simplest being a regular python list of numbers. Since Machine Learning requires lots of scientific calculations, it is much better to use NumPy’s ndarray, which provides a lot of convenient and optimized implementations of essential mathematical operations on vectors.

Vectorized operations perform faster than matrix manipulation operations performed using loops in python. For example, to carry out a 100 \* 100 matrix multiplication, vector operations using NumPy are two orders of magnitude faster than performing it using loops.

Some ways in which NumPy arrays are different from normal Python arrays are:

1. If you assign a single value to a ndarray slice, it is copied across the whole slice

|  |  |
| --- | --- |
| NumPy Array | Regular Python array |
| |  |  | | --- | --- | | 1  2  3  4 | >>> a = np.array([1, 2, 5, 7, 8])  >>> a[1:3] = -1  >>> a  array([ 1, -1, -1,  7,  8]) | | |  |  | | --- | --- | | 1  2  3 | >>> b = [1, 2, 5, 7, 8]  >>> b[1:3] = -1  TypeError: can only assign an iterable | |

So, it is easier to assign values to a slice of an array in a NumPy array as compared to a normal array wherein it may have to be done using loops.

1. ndarray slices are actually views on the same data buffer. If you modify it, it is going to modify the original ndarray as well.

|  |  |
| --- | --- |
| NumPy array slice | Regular python array slice |
| |  |  | | --- | --- | | 1  2  3  4  5  6 | >>> a = np.array([1, 2, 5, 7, 8])  >>> a\_slice = a[1:5]  >>> a\_slice[1] = 1000  >>> a  array([   1,    2, 1000, 7,    8])  # Original array was modified | | |  |  | | --- | --- | | 1  2  3  4  5  6  7 | >>> a=[1,2,5,7,8]  >>> b=a[1:5]  >>> b[1]=3  >>> print(a)  >>> print(b)  [1, 2, 5, 7, 8]  [2, 3, 7, 8] | |

If we need a copy of the NumPy array, we need to use the copy method as another\_slice = another\_slice = a[2:6].copy(). If we modify another\_slice, a remains same

1. The way multidimensional arrays are accessed using NumPy is different from how they are accessed in normal python arrays. The generic format in NumPy multi-dimensional arrays is:

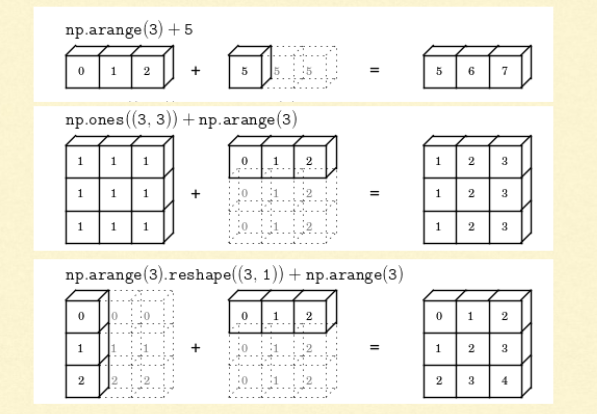
Array[row\_start\_index:row\_end\_index, column\_start\_index: column\_end\_index]

NumPy arrays can also be accessed using boolean indexing. For example,

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | >>> a = np.arange(12).reshape(3, 4)  array([[ 0, 1, 2, 3], [ 4, 5, 6, 7], [ 8, 9, 10, 11]])  >>> rows\_on = np.array([True, False, True])  >>> a[rows\_on , : ]      # Rows 0 and 2, all columns  array([[ 0,  1,  2,  3],        [ 8,  9, 10, 11]]) |

NumPy arrays are capable of performing all basic operations such as addition, subtraction, element-wise product, matrix dot product, element-wise division, element-wise modulo, element-wise exponents and conditional operations.

An important feature with NumPy arrays is broadcasting.



In general, when NumPy expects arrays of the same shape but finds that this is not the case, it applies the so-called broadcasting rules.

Basically, there are 2 rules of Broadcasting to remember:

1. For the arrays that do not have the same rank, then a 1 will be prepended to the smaller ranking arrays until their ranks match. For example, when adding arrays A and B of sizes (3,3) and (,3) [rank 2 and rank 1], 1 will be prepended to the dimension of array B to make it (1,3) [rank=2]. The two sets are compatible when their dimensions are equal or either one of the dimension is 1.
2. When either of the dimensions compared is one, the other is used. In other words, dimensions with size 1 are stretched or “copied” to match the other. For example, upon adding a 2D array A of shape (3,3) to a 2D ndarray B of shape (1, 3). NumPy will apply the above rule of broadcasting. It shall stretch the array B and replicate the first row 3 times to make array B of dimensions (3,3) and perform the operation.

NumPy provides basic mathematical and statistical functions like mean, min, max, sum, prod, std, var, summation across different axes, transposing of a matrix, etc.

A particular NumPy feature of interest is solving a system of linear equations. NumPy has a function to solve linear equations. For example,

|  |  |
| --- | --- |
| 1  2 | 2x + 6y = 6  5x + 3y = -9 |

Can be solved in NumPy using

|  |  |
| --- | --- |
| 1  2  3  4  5 | >>> coeffs  = np.array([[2, 6], [5, 3]])  >>> depvars = np.array([6, -9])  >>> solution = linalg.solve(coeffs, depvars)  >>> solution  array([-3.,  2.]) |

**Pandas:**

**What is Pandas?**

Similar to NumPy, Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools. Unlike NumPy library which provides objects for multi-dimensional arrays, Pandas provides in-memory 2d table object called Dataframe. It is like a spreadsheet with column names and row labels.

Hence, with 2d tables, pandas is capable of providing many additional functionalities like creating pivot tables, computing columns based on other columns and plotting graphs. Pandas can be imported into Python using:

>>> import pandas as pd

Some commonly used data structures in pandas are:

1. **Series objects**: 1D array, similar to a column in a spreadsheet
2. **DataFrame objects:** 2D table, similar to a spreadsheet
3. **Panel objects:** Dictionary of DataFrames, similar to sheet in MS Excel

Pandas Series object is created using pd.Series function. Each row is provided with an index and by defaults is assigned numerical value starting from 0. Like NumPy, Pandas also provide the basic mathematical functionalities like addition, subtraction and conditional operations and broadcasting.

Pandas dataframe object represents a spreadsheet with cell values, column names, and row index labels. Dataframe can be visualized as dictionaries of Series. Dataframe rows and columns are simple and intuitive to access. Pandas also provide SQL-like functionality to filter, sort rows based on conditions. For example,

1. **head():** returns the top 5 rows in the dataframe object
2. **tail():** returns the bottom 5 rows in the dataframe
3. **info():** prints the summary of the dataframe
4. **describe():** gives a nice overview of the main aggregated values over each column

**------------------------------------------------------------------------------------------------**

**Strip Punctuation Python**

Punctuations are symbols or sign which is used to indicates the structure of syntax. It is also known as separators. following are the examples of the punctuation used in programming:

* **( ) -** to represents arguments for a method.
* **[ ] -**to represents array indices.
* **{ } -** to represents block of statements.
* **, -** It is used to separate items in sets/lists.
* **; -** It is used to terminates statements and declarations of fields.

**Using str.translate() Method**

This is a fast method to remove all punctuation from a string.

In the following example, we will use the translate() function from the built-in string library to remove all punctuation from the string.

**Ex:**

import string

My\_string = “Hello!!!, This is ##STechies$$."

print(my\_string.translate(str.maketrans('', '', string.punctuation)))

BY Using Regular Expression:

**import** re

*# String with punctuation's*

string = "Hello!!!, $#@!\*()&,.This is ##STechies$$."

final\_string = re.sub(r'[^\w\s]','',string)

*# Print final String*

print('String with Punctuation: ', string)

print('String without Punctuation: ', final\_string)