

Welcome to PYTHON TUTORIAL

By
MLTUT
www.mltut.com

OBJECTIVE

PYTHON TUTORIAL

This Tutorial aims to teach everyone from scratch. This Tutorial is for everyone. The one who is thinking to start a career in Python.

And for those who know Python, but just to brushup their knowledge. This Tutorial is made with very simple language and covers every important topic related to Python. At the end of the Tutorial, you will have a good knowledge of Python. And you don't need to learn from other sources. This Tutorial is enough for you.

•
•
•

Let's Get Started

Table of Content

1. Python Installation
2. Numbers in Python
3. Strings in Python
4. List in Python
5. Standard Input, Output in Python
6. If...Else Statements in Python
7. Loop in Python
8. Range() Function in Python
9. Function in Python
10. Variable Scope in Python
11. Dictionary in Python
12. Sets in Python
13. Classes in Python
14. Methods & Attributes in Python
15. Modules & Packages in Python
16. List Comprehension in Python
17. Map, Filter, and Lambda in Python
18. Decorators in Python
19. File Handling in Python

Chapter 1- Introduction and Python

Installation

What is Python?

A python is an object-oriented, interpreted, and high-level programming language. Python is easy to understand language. Its syntax is easily readable. Even beginners can easily understand its syntax without any complications. Unlike in other programming languages, which requires complex code to just print “Hello World”, in python you can simply print it by just typing-

```
print("Hello World")  
OUTPUT- Hello World
```

That’s all you need to write for printing “Hello World”. It is as simple as the English language. Python is open-source, which means it is free to use. Python is portable, which means you can write code once, and run it on different platforms.

Why learn Python?

After getting to know What is Python? one more question is striving into your mind that Why Python? There are various reasons to use Python. Some of the reasons are given below-

- The main reason for most of us, to use python is that it is a straightforward language. The syntax is beginner-friendly.

- Python requires fewer lines of code as compared to other programming languages.
- You can run python smoothly to any platforms like Windows, Mac, Linux, etc)
- Python has a powerful standard library.
- In Python, there are various career opportunities. There is a high demand for python developers as Data Scientist after the rise of Big data.

What Python can do?

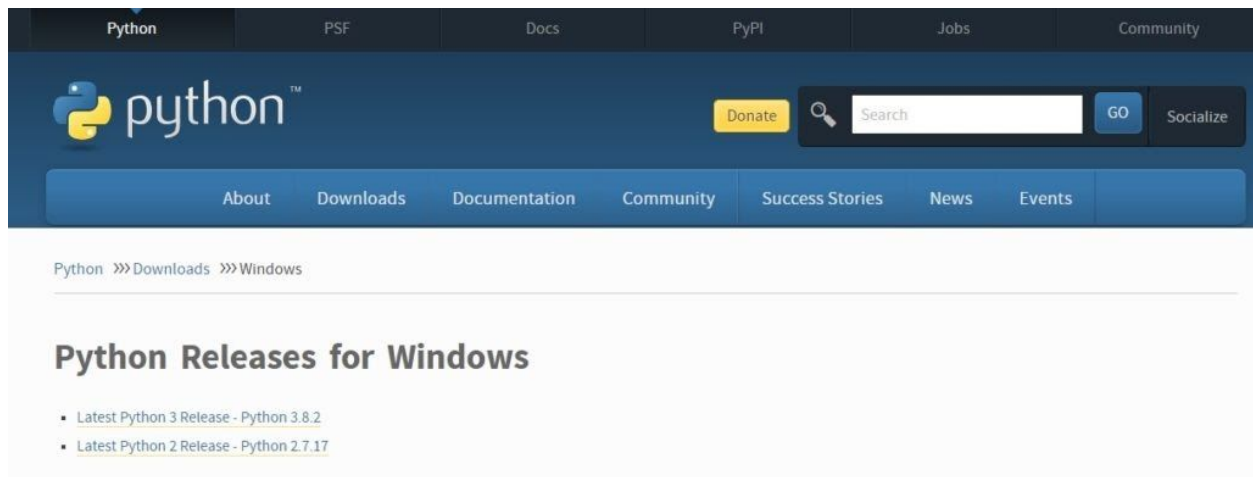
By using python, you can do the following tasks-

- You can use python to do Big Data related works, it easily performs complex mathematics.
- Python can create a web application.
- You can use python in database-related jobs, it can read and modify files.
- By using python, you can create games. There is one library, known as **Pygame**. This library helps you to create any type of game.
- For, Machine Learning python is the best choice. You can solve machine learning tasks very easily by using python.

How to Install Python?

Python Installation steps for **Window 10**—

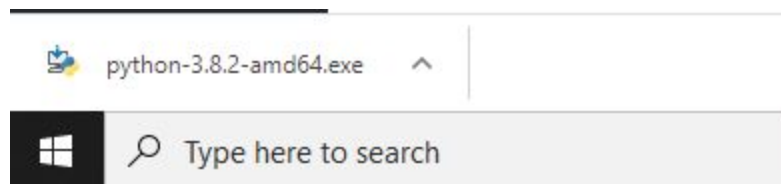
- You can install Python from here- <https://www.python.org/>
- The latest version of Python in March 2020 is Python 3.8.2.
- After you open the python home page, just go in the download section and choose windows.



- Click on Python 3.8.2, a new window will open, If you have a 64-bit processor, then choose red one-

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		f9f3768f757e34b342dbc06b41cbc844	24007411	SIG
XZ compressed source tarball	Source release		e9d6ebc92183a177b8e8a58cad5b8d67	17869888	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	f12203128b5c639dc08e5a43a2812cc7	30023420	SIG
Windows help file	Windows		7506675dccb9a1569b54e600ae66c9fb	8507261	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	1a98565285491c0ea65450e78afe6f8d	8017771	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	b5df1cbb2bc152cd70c3da9151cb510b	27586384	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	2586cdad1a363d1a8abb5fc102b2d418	1363760	SIG
Windows x86 embeddable zip file	Windows		1b1f0f0c5ee8601f160cfad5b560e3a7	7147713	SIG
Windows x86 executable installer	Windows		6f0ba59c7dbeba7bb0ee21682fe39748	26481424	SIG
Windows x86 web-based installer	Windows		04d97979534f4bd33752c183fc4ce680	1325416	SIG

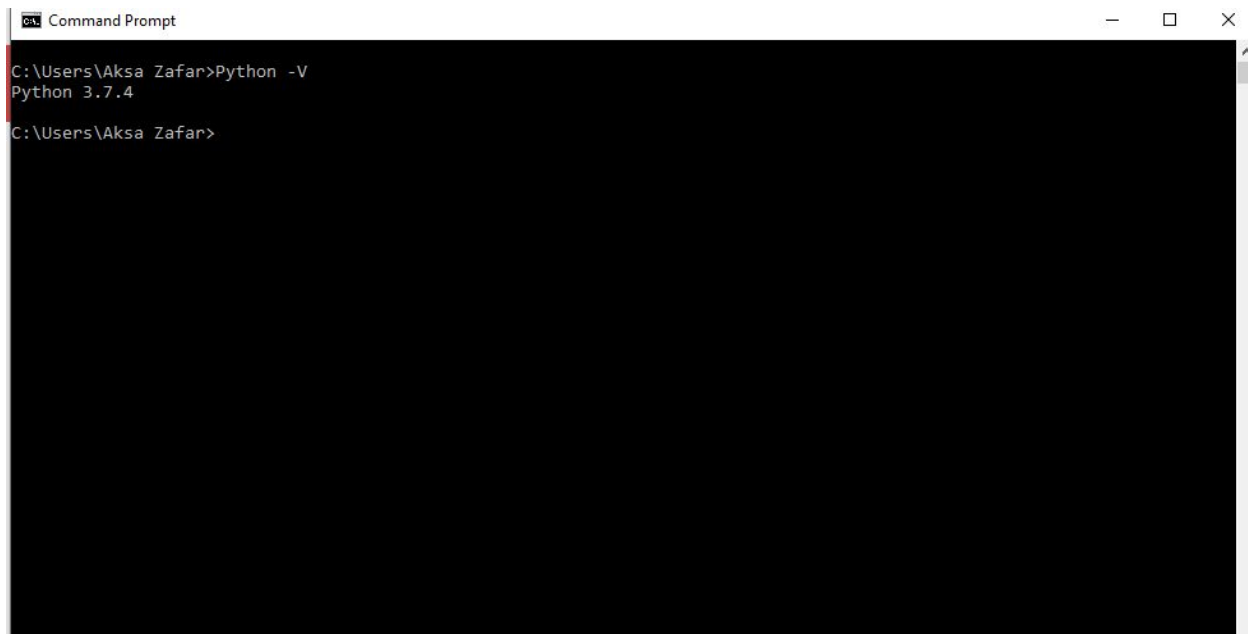
- As you click on that, it starts downloading. Once it is downloaded, click on that-



- Once you click on python-3.8.2-amd64.exe, a new window will come to ask permission. Click on RUN.
- After clicking on RUN, this new window will come-

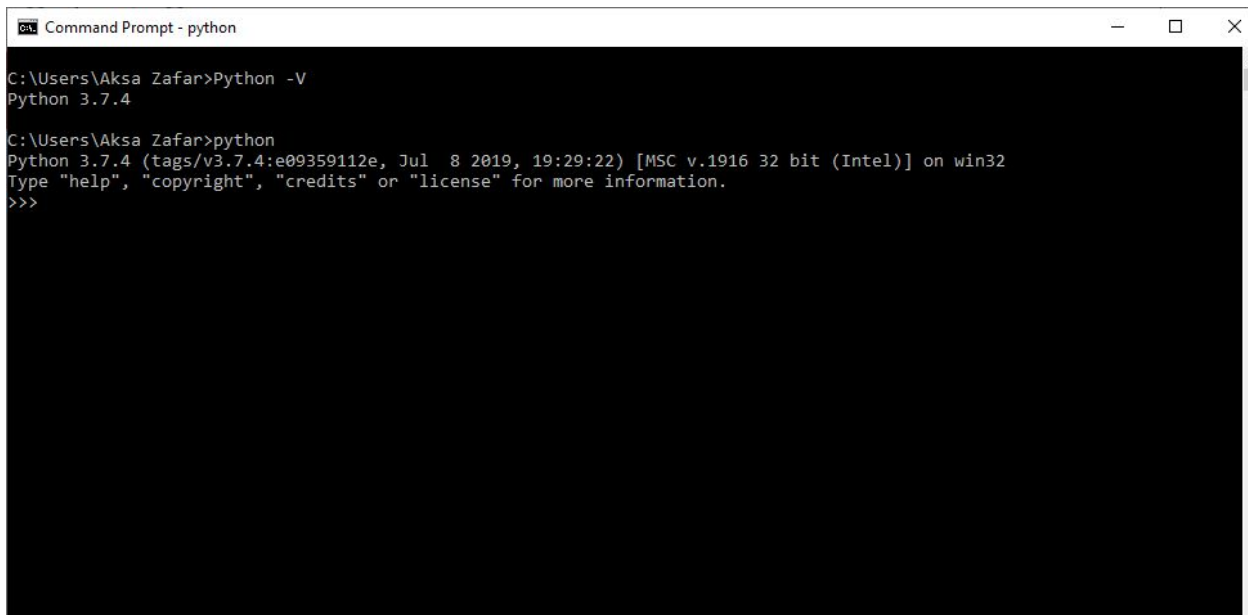


- Click on Install Now, to install Python on your Windows 10.
- Make sure that you check the box of **Add Python 3.8 to PATH**. If you don't check this box, you will face issues while running Python programs.
- After installing the python Successfully installed message will come.
- Close this window, and open command line, just by typing cmd in your search panel.
- To check the version of python installed on your system- just type... **Python -V**




```
Command Prompt
C:\Users\Aksa Zafar>Python -V
Python 3.7.4
C:\Users\Aksa Zafar>
```

- To go under the python environment, just type **python**.



```
Command Prompt - python
C:\Users\Aksa Zafar>Python -V
Python 3.7.4

C:\Users\Aksa Zafar>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



And here we go, congratulations! You successfully installed python on your windows 10 system.

In the next chapter, we will start learning **Python Numbers**.

Chapter 2 - Numbers in Python

Types of Numbers-

In Python, there are three types of numbers-

Integer Type- The Integer type is a whole number. It may be positive or negative. It doesn't contain a decimal. For Example-

```
a=5 #int  
b= 1079876509 #int  
c= -673409 #int
```

Float Type- It contains a decimal, and it may be positive or negative. For Example-

```
a= 5.20 #float  
b=5.0 #float  
c= -67.83 #float
```

Complex Type- This type of numbers are in the form of $a+bj$, where j is the square root.

```
a= 9+8j # complex
b= 9j #complex
c= -4j #complex
```

To check the type of any number, you just need to type-

```
print(type(a))
```

and it will return the type of number. For example, if you want to check the type of number of $a=5$ so, you can check it like that-

```
a=5
print(type(a))

OUTPUT-> <class 'int'>
```

Type Conversion-

If you want to convert your number from one type to another, for example, int to float, float to complex, etc. you can simply do it.

Have a look How you can convert a type of number-

```
a= 6 #int
b=7.9 #float
c= 7+5j #complex

#from int to float-
x= float(a)

#from float to int-
y= int(b)

#from float to complex-
z= complex(b)
```

See, it is as simple as that, just by writing one line of code, you can convert from one type to another.

Note- You can't convert complex numbers to any other type.

Basic Calculations using Python-

After getting proper knowledge of types of numbers and conversion, its time to practice some basic calculations by python.

So, are you ready? Let's get started by simple maths-

- For simple **addition**, **subtraction**, **multiplication**, and **division**, you just need to type as you type in the calculator. Very easy...right? Let's see, how to do it-

Write this code in your cmd, to practice with me-

```
>>> 5+5    #addition
10

>>>5-5     #substraction
0

>>>5*5     #multiplication
25

>>>5/5     #division
1.0 # float type
```

-> As you see in the division it is returning the result in a float as **1.0**. To get the result in int, you need to type it as-

```
>>>5//5    #division  
1    #int type
```

-> For the **square root** of number-

```
>>>5**5  
3125
```

-> For **remainder/modulus**—

```
>>>10%3  
1
```

-> **BIDMAS Rule**-To perform longer calculation, you need to take care of BIDMAS Rule. Let's see an example, how it affects the whole calculation.

```
>>>5+5*3  
20
```

- Here it is returning 20 as a result, just because * has more priority than + according to BIDMAS Rule.
- If you want to do addition first, put it into a bracket, because in BIDMAS Rule a bracket has more priority than *
- Let's see in this example, how the result is changed just by using ().

```
>>>(5+5)*3
30
```

Store number in variables-

-> You can store numbers on variables and perform a calculation on it.
Let's see how you can do it-

```
>>>age=25    #store number in variable "age"
>>>age       #print "age"
25           #OUTPUT

#perform addition on "age"
>>>age+5
30
```


->"age+5", simply add 5 to your age variable whose value is 25, but it doesn't alter the variable "age" value. If you enter "age", it returns 25 as a result. I'll show you here-

```
>>>age
5          #print 25 again
```

->If you want to update the value of "age", then you can simply do it by writing-

```
>>>age= age+5    #add 5 in variable "age", not direct.
>>>age          #print age
30              #OUTPUT
```

-> There is **one more way to update the value of variable "age"**. Have a look at it too-

```
>>>age+=5        #add 5 in variable "age", not direct.
>>>age          #print age
30              #OUTPUT
```

-> Just like addition, you can perform other calculations on a variable, by just writing a few lines of code. Let's have a look-

```
>>>age=25

>>>age-=5  #subtraction

>>>age

20

>>>age/=2   #division

>>>age

15.0
```

I hope, now you have a clear understanding of numbers in python. Now it's time to perform some high-level calculations.

Let's see one example-

```
>>>wages=1000
>>>bills=200
>>>rent=500
>>>food=200
>>>savings= wages-bills-rent-food    #formula to find saving
>>>savings
100
```

->In this example, we are calculating the total saving of month, and look it is how much simple in python, just put all values in variables, apply a formula, and get the results.

And here we go, congratulations! You successfully learned numbers in python.

In the next chapter, we will start learning **Python Strings**.

Chapter 3 - Strings in Python

How to represent Strings?

String in Python is represented by single quotation marks (' ') or double quotation marks (" ").

For Example-

```
print("hello")  
print('hello')
```

OUTPUT -

```
hello  
hello
```

-> But there is one problem, suppose if you have to write- **'He's a good man'**, this sentence gives a syntax error, because **'He'** ends here.

->Therefore for apostrophe words use double quotation marks (" "). Let's have a look-

```
print("He's is a good man")
```

OUTPUT-

```
He's is a good man
```

There is one method to use single quotation marks (' ') for apostrophe words, just put \ before apostrophe, like that- 'he\'s a good man'.

->Let's see in this example-

```
print('He\'s a good man')
```

OUTPUT-

```
He's is a good man
```

Store String to Variable-

We can store string to a variable in the same way we store numbers to a variable. Let's have a look, how you can store string to a variable-

```
>>>greet = 'Hello'
```

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

OUTPUT-

```
Hello
```

String in Array Form-

In Python, Strings are an array of bytes, which represent Unicode characters. However, in Python, there is no Character Data Type, an individual character in python, is a string with a length of 1.

Like in an array, in strings, we use a square bracket [] to access elements.

-> If you want to get any specific element in a string, so you can it easily by just writing following line-

```
>>>greet[0]
```

OUTPUT-

```
'H'
```

->Here, we are taking greet= 'Hello'.

->In this example, we are getting 'H', because we are giving index number 0.

Note- Index starts from 0 to so on.

By Using Negative Indexing-

-> You can perform the same task from reverse order like 'Hello' <-, just using a negative index like -1.

Confused?... Let's see in the example, to clear your confusion.

```
>>>greet[-1]
```

OUTPUT -

```
'o'
```

-> Bypassing -1 as an index, it is returning 'o', because, in negative indexing-

-1 represents 'o'

-2 represents 'l'

-3 represents 'l'

-4 represents 'e'

-5 represents 'h'

NOTE- Backward indexing starts from -1 and forward indexing start from 0. For a more clear understanding of indexing, Let's have a look in the following example-

```
-5  -4  -3  -2  -1    #Backward Indexing
H   e   l   l   o
0    1   2   3   4    #Forward Indexing
```

String Slicing-

Slicing means, cut some parts from the whole piece. The same thing, we can do with string in python.

If we want to get only some parts from the word, for that purpose we do slicing.

So, are you ready to know how to perform slicing? Let's get started.

->Let's consider the same example of greet= 'Hello', and if you want to cut this 'Hello' word, so you can do it by the following code-


```
>>> greet [0:3]
```

OUTPUT-

```
'Hel'
```

-> greet [0:3] means, it starts from index 0 to 2, it doesn't take 3rd element. Let's see in detail-

```
H e l l o
```

```
0 1 2 3 4
```

```
----          #slice till index 2
```

-> You can perform slicing with negative indexing, just have a look, how you can do it-

```
>>>greet [2:-1]
```

OUTPUT

```
'll'
```

-> You are wondering, how it return, 'll' as a result. Don't worry, I will explain to you, how, 'll' come in output.

```

-5  -4  -3  -2  -1
H    e   l   l   o
0    1   2   3   4
-----

```

- `greet[2 : 1]`, starts from Index 2, that is 'l' to Index before -1, that is Index -2, and at Index -2, the word is 'l', that's why it is returning as, 'll'.

-> By doing all these operations, the value of `greet` is still 'Hello'.

-> If you want to permanently change the value of **greet** variable, then you need to reassign it. Let's have a look at how you can reassign.

```

>>>greet = greet[0:3]

>>>greet

```

OUTPUT -

```
'Hel'
```

Most Used String Functions–

In String, you can perform various operations, here we discuss some most used and important functions. Let's have a look at these functions.

String Concatenation-

->If you want to concatenate or add two strings, then you can do it simply by using '+' sign. Let's see in this example-

```
>>>greet = 'Hello'
>>>str = 'John'
>>>greet + str
```

OUTPUT-

```
'HelloJohn'
```

There is no space between Hello and John, so if you want to give space, just give ' '. See in the example below-

```
>>>greet+' '+str
```

OUTPUT-

```
'Hello John'
```

Repetition of Strings-

If you want that one word repeats multiple times, so use * sign with the number you want to repeat that word. Let's see in the example below-

```
>>>greet*3
```

OUTPUT -

```
'HelloHelloHello'
```

Note- To use any string method, just use the (.) operator.

String UPPER CASE-

->If you want to convert a string from lower case to upper, use the upper() method. Let's see in the example.

```
>>>greet.upper()
```

OUTPUT -

```
'HELLO'
```

For converting the upper case to lower case, just use a lower() method.

String Splitting-

->If you want to split a different string, you can do it by writing a split(',') method. Let's see in the example below-

```
>>>Names= "John, mike, alex"  
>>>Names.split(',')  
  
OUTPUT -  
['John', 'mike', 'alex']
```

String Length-

-> To find the length of string, write len(string) function. For example-

```
>>>len(greet)  
  
OUTPUT -  
5
```

And here we go, congratulations! You successfully learned strings in python.

In the next chapter, we will start learning **Python Lists**.

Chapter 4 - List in Python

How to create a List?

In python, a list is created by using the Square bracket [] and put all item inside that bracket and separate them by comma (,).

The powerful feature of the list is that there is no size limit, and you can put different data type items in one list.

->You can also create an empty list.

For Example-

```
#empty list
List = []

#list with integer values
List = [1,2,3]

#List with different Data type
List = [5, 7.9, "John"]
```

Split a string into List?

We have covered this part in the previous tutorial of [string](#). Let's have a look in the example-

```
>>>List = "I am good man"  
>>>List.split(' ')
```

OUTPUT-

```
['I', 'am', 'good', 'man']
```

How to access an item in a List?

-> Like in [string](#), in List element index start from 0 to so on. If we want to access any particular item from the list, then we do the same job as done in the string.

```
>>>List = [1,2,3,4,5,6]  
>>>List[2]
```

OUTPUT-

```
3
```

```
>>>List[5]
```

OUTPUT-

```
6
```

Access element via Negative Indexing-

In list, backward direction starts from -1,-2, and so on, same as in string. Let's have a look-

```
>>>List = [1,2,3,4,5,6]
```

```
>>>List[-1]
```

OUTPUT -

```
6
```

Slicing in List?

Slicing means, cut some parts from the whole piece. The same thing, we can do with List in python.

If we want to get only a few items from a list, then we can perform slicing. The procedure is the same as we did in the string.

So, are you ready to know how to perform slicing? Let's get started.

```
>>>List = [1,2,3,4,5,6,7,8]
```

```
>>>List[0:4]
```

OUTPUT -

```
[1,2,3,4]
```


-> It starts taking from index 0 to index 3, It doesn't take element at index 4.

-> I'll show you how index numbers are assigned to each item.

```
>>>List = [1, 2, 3, 4, 5, 6, 7, 8]
        index= 0  1  2  3  4  5  6  7
```

-> I hope, now you have a clear understanding of list indexing.

How to concatenate 2 List?

If you wanna add or concatenate two lists and make it one, so you can do it simply by using '+' operator. It is as simple as that. Let's have a look at an example.

```
>>>List1 = [1,2,3,4,5]
>>>List2 = [6,7,8,9,10]
>>>List3 = List1 + List2
>>>List3

OUTPUT-
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Update items on the List?

If you make a list and suddenly you have to change some items from the list, so you can do it very easily, by just passing an index value, on which index location you want to update with a new item.

Let's see in the example-

```
>>>List=[1,2,3,4,5,6]
#change index 0 value with new value
>>>List[0]=9
>>>List
```

OUTPUT-

```
[9,2,3,4,5,6]
```

How to append in List?

If you wanna add a new item in your current list, you can do it by using the `append()` method. Just pass the value in the append method, you wanna append in your list.

Let's see in the example-

```
>>>List=[1,2,3,4,5,6]

#append element 7 in List[]

>>>List.append(7)

>>>List
```

OUTPUT-

```
[1,2,3,4,5,6,7]
```

How to delete in List?

If you don't want to use some items in your list, then you can delete this item from the list, by using the **pop()** method. In the **pop()** method, you don't have to need to pass the value, it deletes the last element from your list by default.

Let's see in the example-

```
>>>List=[1,2,3,4,5,6]

#use pop() method to delete the last element in your list that is 6 in this List

>>>List.pop()

>>>List
```

OUTPUT-

```
[1,2,3,4,5]
```

- But what?... If you wanna delete some specific items from your list that may be in between your list, don't worry! For that task, python has one more method that is- **remove()**
- By using remove() method, you can delete any item from your list, no matter its location in the list.

Let's see in the example-

```
>>>List= [1,2,3,4,5,6,7]

#remove item 4 from your List

>>>List.remove(4)

>>>List

OUTPUT-

[1, 2, 3, 5, 6, 7]
```

- There is one more problem, Guys!, If there are 2 same numbers in your list, and you wanna remove both numbers, then it fails.
- This remove() method deletes only one item.

Let's see in the example if there are 2 same numbers present in the list.

```
>>>List=[1,2,3,4,5,6,4]
```

```
>>>List.remove(4)
```

```
>>> List
```

OUTPUT -

```
[1, 2, 3, 5, 6, 4]
```

- It deletes an item 4 only once, which comes first from index 0.

->Guys!... Don't worry, there is one more trick to delete the item which you wanna delete, no matter how much time it repeats in a list.

->Suppose in List=[1,2,3,4,5,6,4,8,9], you don't want to delete 4 at index 3, and you want to delete 4 which is at index 6, so you can use- **del()** method, which deletes an element of your choice.

Let's see in the example-

```
>>>List=[1,2,3,4,5,6,4,8,9]
```

```
>>>del(List[6])
```

```
>>>List
```

OUTPUT -

```
[1,2,3,4,5,6,8,9]
```

->Syntax of del() method is- **del(name of List[index position of item])**

Creating List inside List.

If you wanna create a list inside another list, then you can do it by passing the list name to the new list, in which you wanna add.

Confused?... Let's see in the example to make it more clear.

```
#List 1
>>>Colors=['red','green','yellow']

#List 2
>>>Fruits=['apple','banana','cherry']

#List 3 with List 1 and List 2
>>>num= [Colors,Fruits,[1,2,3,4,5]]
>>>num

OUTPUT-
[['red', 'green', 'yellow'], ['apple', 'banana', 'cherry'], [1, 2, 3, 4, 5]]
```

- Here Colors is a list and Fruits is also a list, and we make one more list as num. In the num list, we pass both lists-

Colors and Fruits and we pass num list own items. And all are added in one list as a num list.

- As you can see in that example, in one list different data types of items are present. This the main feature of the list, that we can store different data type items in one list.

-> We can get a particular list from num list by writing the following lines-

```
>>>num= [Colors,Fruits,[1,2,3,4,5]]
```

```
>>>num[0]
```

OUTPUT-

```
['red', 'green', 'yellow']    #which is List Colors at index 0
```

->We can get any particular element inside the list by writing following line-

```
>>>num= [Colors,Fruits,[1,2,3,4,5]]
```

```
>>>num[2][1]
```

OUTPUT-

```
2
```

* In `num[2][1]`-

- `[2]` represents the index location of the List.
- `[1]` represents the index location of the element inside the list.

That's all for List in Python, I hope now you have a better understanding of List.

congratulations! You successfully learned List in python.

In the next chapter, we will start learning **Standard Input, Output in Python**

Chapter 5 - Standard Input, Output in Python

Python Output- print() function.

In python, for printing something on the console or the standard output screen, we use a simple print() function.

For example, if you wanna print some messages on the console, you can print it by using a print() function.

Syntax of print() function is- print(message)

Let's have a look in the example-

```
>>>print("hello! My name is John")
```

OUTPUT-

```
hello! My name is John
```

you can also print any value by the print() function, along with the message you wanna print. Let's see in the example-

```
>>>x=56
```

```
>>>print("Your age is",x)
```

OUTPUT-

```
Your age is 56.
```

If you wanna print more than one value in one print statement, then you can do it by concatenation. Let's see in the example.

```
>>>name=input("Tell your name")
```

```
Tell your name John
```

```
>>> age=input("Tell your age:")
```

```
Tell your age:25
```

```
#Use ' ' to concatenate.
```

```
>>> print(name,' you are',age)
```

OUTPUT-

```
John you are 25
```

Python Input- input() function.

Till now, we are providing value to the program, but if you want that user will give input value, then there is a function – input().

Input() function allows users to provide the value of their choice.

Let's take a look in the example-

```
>>>age= input("Tell your Age:")
```

```
Tell your Age: 25
```

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

OUTPUT-

25

- But, here the value entered by the user as an age “25” is a string, it is not a number. For converting this into a number, `int()` or `float()` functions are there.

Let's take a look in the example-

```
>>>int('25')
```

OUTPUT-

25

```
>>>float('25')
```

OUTPUT-

25.0

Program for calculating the Area of Circle-

The formula for the area of circle = πr^2

- Where r is a radius,
- π value is 3.142

```
>>>radius=input('Enter radius of circle(m):')
Enter radius of circle(m):5
>>> area=3.142*int(radius)**2
>>> print('The area of circle is',area)
```

OUTPUT-

```
The area of circle is 78.55
```

Output Formatting – format() function.

In a normal way, we write our code in a print() statement, but we can use format() method to make our output more attractive and structured.

Let's see in the example, how to use format() method.

```
>>> num1=5
>>> num2=7
>>> print('value of num1 is{0} and num2 is {1}'.format(num1,num2))
```

OUTPUT-

```
value of num1 is 5 and num2 is 7
```

- Here {0} represents index value 0, and {1} represents index value 1.

If you have float number and you wanna restrict the decimal number in some limit, like 3.14254678 to only 3.14 so, you can do it.

Let's see how you can do it-

```
>>> a=3.14254678
>>> b=10.2903348
>>> print('value of a is {0:.3} and b is {1:.3}'.format(a,b))
```

OUTPUT-

```
value of a is 3.14 and b is 10.3
```

- Here, in {0:.3} the .3 decides the decimal limit to- .3
- It prints only 3.14 which is .3 decimal place.

- If you want that it will consider 3.14 as float, then you can write it as- **{0:.3f}**

That's all for Standard Input, Output in Python, I hope now you have a better understanding of Standard Input, Output.

congratulations! You successfully learned Standard Input, Output in python.

In the next chapter, we will start learning **If-else Statements**.

Chapter 6 - if ... else Statements

If statement and its syntax

These conditional statements in python are used, when you want to run your code with certain conditions.

If statement is used when you test some condition.

Syntax of if statement is-

```
if test condition:  
    action(a)
```

Here, first, it checks the condition, if the condition is true then the action executes otherwise, no statement executes.

Let's have a look in the example-

```
age=5  
if age<10:  
    print('You are young')
```

OUTPUT-

You are young

Here, age=5 and it's less than 10, which means the condition is true, that's why it prints a message.

Indentation–

Indentation in python is compulsory because if we don't do indentation, code doesn't work.

For example, if you write code without indentation, then it will give you an error.

```
age=5
if age<10:
print('You are young')    #wrong indentation, gives error.
```

elif statement and its syntax.

you can use elif statement when you wanna use more than one condition to check. When if condition is false, then it switches to elif condition and execute the elif statement.

Syntax of elif statement is-

```
if test condition:
    action(a)
elif test condition:
    action(b)
```


Here, first, it checks the if condition, if it is true so the action is executed, but if the condition is not true then elif condition is checked and if it's true, then its action is executed.

Let's take a look in the example-

```
a=5
b=5
if a<b:
    print('a is less than b')
elif a==b:
    print('a and b are equal')
```

OUTPUT-

```
a and b are equal
```

Here, the first condition is not true, which is `a<b`, but second condition `a==b` is true, that's why elif statement is executed.

if-elif-else statement and its syntax.

if-elif-else is used when **if** and **elif** statement gets false, so **else** will execute. But if anyone in **if** or **elif** is true, so **else** will not execute.

Syntax of if-elif-else-

```
if test condition:  
    action(a)  
elif test condition:  
    action(b)  
else test condition:  
    action(c)
```

Here, first, it checks **if** and **elif** conditions, if both are false then **else** will execute.

Let's see in the example-

```
a=50  
b=10  
if a<b:  
    print('a is less than b')  
elif a==b:  
    print('a and b are equal')  
else a>b  
    print('a is greater than b')
```

OUTPUT-

```
a is greater than b
```

Nested if-else and its syntax.

Nesting means use if-else statements inside another if-else statement. Indentation is important to perform nesting.

Syntax of Nested if-else statement-

```
if test condition:
    if test condition:
        action(a)
    else:
        action(b)
else
    action(c)
```

Here, there are two cases-

1. **When If is true-** It checks the first if condition, if it is true then it will go inside another if and test condition if it is true, the action is executed. If it is not true, then else statement is executed.
2. **If is not true-** If the starting **if** condition is not true, then without going inside another **if**, directly the last **else** statement is executed.

Example of nested if-else-

```
age = 30

if age <= 19:
    if age == 2:
        print("Infant")
    else:
        print("Teenager")
else:
    print('Adult')
```

OUTPUT -

Adult

Here, age=30, which is not less than 19, that's why, the whole if statement is truncated, and control goes to the last else statement, and it is executed.

Logical operators.

There are basically two logical operators-

1. AND
2. OR

AND Operator-

AND operator is used to combining two or more conditional statements.

Lets' see in the example-

```
John=35
Alex=30
Max= 40
if John> Alex and Max>John:
    print("Max is elder than John and Alex")
```

OUTPUT-

```
Max is elder than John and Alex
```

- AND operator executes only when both the conditions are true. If anyone of them is false, then it doesn't execute.

OR Operator-

You can use the OR operator to combine two or more than two conditional statements. It executes if any one of the statements is true.

Suppose you have two conditional statements and one is true another one is false, still OR operator executes.

Let's see in the example-

```
John=35
Alex=30
Max= 40

if John> Alex or Max==John:
    print("Max is elder than John and Alex")
```

OUTPUT-

```
Max is elder than John and Alex
```

Logical Conditions-

There are following Logical conditions-

- **< (less than)**
 - For eg. $x < y$
- **> (greater than)**
 - For eg. $x > y$
- **== (Equal)**
 - For eg. $x == y$
- **!= (not equal to)**
 - For eg. $x != y$
- **=> (equal to or greater than)**
 - For eg. $x >= y$
- **=< (equal to or less than)** For eg. $x <= y$



That's all for Python if ... else Statements, I hope now you have a better understanding of Python if ... else Statements.

congratulations! You successfully learned Python if ... else Statements.

In the next chapter, we will start learning **Loop in python**.

Chapter 7 - Loop in Python

For Loop-

For loop is used to perform some operation repeatedly until the condition gets false.

It is used in traversing a list, strings, and arrays.

Suppose we create a list called colors-

- colors=['red','green','yellow','black']

And we want to traverse the whole list, then we can for loop.

Syntax of for loop is-

```
for iteration_variable_name in List_Name:  
    statements(S)
```

Let's take an example of colors list, and perform traversing-


```
colors=['red','green','yellow','black']  
for c in colors:  
    print(c)
```

OUTPUT-

```
red  
green  
yellow  
black
```

- You can use any name for iteration_variable_name. Here we use 'c'.

Slicing via For Loop-

You can perform slicing as we have done in [List](#) and [string](#). If you wanna get some particular element from the whole list, then you can perform this task via for loop.

Let's see in the example below-

```
colors=['red','green','yellow','black']  
for c in colors[1:3]:  
    print(c)
```

OUTPUT-

```
green  
yellow
```

- You are wondering, How it is printing green and yellow as output?
- Don't worry! ... I'll tell you how?
- Here, we are getting only 'green' and 'yellow', because 'green' index position is 1 and 'yellow' index position is 2. And we give colors[1:3]. It starts at index 1(green) to index 2(yellow).

While Loop-

While loop runs until the certain condition is true, but as the condition becomes false, it terminates.

Syntax of while loop-

```
while condition:  
    statement(S)
```

Let's have a look at while loop example-

```
age=25
num=0
while num<age:
    print(num)
    num+=1
```

OUTPUT -

```
1
2
3
.
.
.
24
```

- It will print num till 24 times because at 25 it gets terminated, as the condition becomes false.

Nested Loop–

You can perform nesting in loops as in **if-else**, which means you can use a loop inside another loop.

You can use nesting in for loop as well as while loop.

Syntax of Nested For loop-

```
for iteration_variable_name in List_Name:
    for iteration_variable_name in List_Name:
        statements(S)
        statements(S)
```

Syntax of Nested While loop-

```
while condition:
    while condition:
        statement(S)
        statement(S)
```

- **Note-** You can perform nesting of a loop by using any loop inside another type of loop. For example, you can use while loop inside for loop and similarly for loop inside while loop.

Do-While loop-

Do you know? Python doesn't provide a feature of a Do-While loop, But if you wanna use it in python, then you can create a program using a Do-While loop

The Do-While loop works similarly as a while loop but with one difference.

The Do-While loop first executes and then check the condition, which means it executes once, no matter the condition is true or false.

Syntax of Do-While loop-

```
do{  
    statements  
} while(condition):
```

Break Statement-

If you wanna break a code, or want to stop the loop before traversing through the whole list, you can use Break Statement. In other words, the Break statement aborts the execution of the current loop.

Let's have a look at an example-

```
colors=['red','green','yellow','black']  
for c in colors:  
    print(c)  
    if c=="green":  
        break
```

OUTPUT-

red

green

- Here, The Break statement, break the loop as c="green".

Continue Statement-

If you use Continue Statement in the loop, then it stops the current iteration of the loop, and continue with the next. It sounds a bit confusing, but don't worry!... Let's see in the example.

```
colors=['red','green','yellow','black']  
for c in colors:  
    if c=="green":  
        continue  
    print(c)
```

OUTPUT-

red

yellow

black

- Look in the output, it prints only three colors- red, yellow and black. Not green. Because we use Continue Statement, which stops the current iteration as



```
if c=="green": continue
```

- And continue with next. That's why it prints red, yellow, and black.
- If you are still confused then in simple words continue statement skips the 'green' color because we give a condition if c='green', then skip the green color and print the rest of the colors.
- I hope now you understand Continue Statment.

That's all for Loop in Python, I hope now you have a better understanding of Loop in Python.

congratulations! You successfully learned Loop in Python.

In the next chapter, we will start learning **Ranges in Python**

Chapter 8 - Range Function in Python

What is the Range() Function?

Range() Function basically generates a list of numbers for us, which we can use to iterate over with for loop.

Let's understand the range() function more deeply with the help of an example.

```
print("Show numbers from the range of 0 to 8")  
  
for n in range(8):  
    print(n, end=',')
```

OUTPUT -

```
Show numbers from the range of 0 to 8  
  
0,1,2,3,4,5,6,7
```

- Here, in the example, it creates a range from 0 to 7. It doesn't include the last number, that's why 8 is not printed.

Syntax of Range() Function.

Syntax of Range() Function –


```
for variable_name in range(start_point,end_point,step_no.)
```

Here, variable_name is the name of the variable. It may be anything. Inside the range(), there are three arguments, in which two are optional. end_point is a compulsory argument, rest two are optional.

Range() Function arguments.

As you see in the syntax of range() function, there are three arguments, two are optional- start_point and step_no.

Let's discussed these arguments in detail.

1. **Start Point-** it is the start point of the range, from where the range starts. If you didn't define the start-no, then it will start from 0. It's by default value is 0.
2. **End Point-** This is also known as the upper limit of the range. This is a compulsory argument in the range. It prints the number up to this upper limit, but it doesn't print the endpoint number. For example, if you define endpoint as 8 so range end at 7 (endpoint-1)
3. **Step Number-** The step number is the difference or gap between two numbers. If you don't define step number, then it assumes it as 1. The default value of the step number is 1. For example, if you want to print the Table of 2, then you need to pass the step number as 2, so it will print the number in the gap of 2.

Some Range() Function Examples.

Example 1: Only define End Point-

If you want to define only the endpoint and don't define the start point and step number, then you can do it by the following example-

```
for n in range(7)
    print(n, end=',')
```

OUTPUT -

0,1,2,3,4,5,6

- Here, start point and step number is not defined, that's why it considers as start point=0 and step number=1

Example 2: Only define Start Point and End Point-

If you wanna define the start point and endpoint of the range, then you can do it by the following example-

```
for n in range(3,10):
    print(n, end=',')
```

OUTPUT -

3,4,5,6,7,8,9

- Here, you define the start point as 3 and endpoint as 10, that's why it starts from 3 up to 9 {(endpoint-1)->(10-1) =9}

Example 3: Define Start Point, End Point, and Step Number–

If you wanna define all three arguments in the range, then you can do it by the following example–

```
for n in range(0,20,4):  
    print(n, end=',')
```

OUTPUT–

0,4,8,12,16

- Here, 0 represents the start point, 20 represents the endpoint and 4 represents the step number. As step number is 4, that's why you can see numbers are in the gap of 4 like- $0+4=4$, $4+4=8$, and so on.

Range() Function with loop.

As we learn in [for\(\) loop tutorial](#), for() loop is used to perform some operation repeatedly until the condition gets false. We can use loop function in the range.

By using for() loop, the range() function doesn't produce all numbers in one go. The range() function produces the next number only when for loop iteration gave an order to produce.

Let's understand with the help of an example-

```
burgers= ['cheesy','chicken','veg','supreme','double']  
for n in range(len(burgers)):  
    print(n,burgers[n])
```

OUTPUT -

```
0    cheesy  
1    chicken  
2    veg  
3    supreme  
4    double
```

- Here, by using len(burgers), we can get total elements of a list.
- In print(n,burgers[n]), n represents the index position and burgers[n] shows the item in every specific index position.
- The variable n is not getting the all values 0,1,2,3,4 at one go. First, n starts at 0, then n=1, and so on.

Python Range Reverse.

If you want to reverse the range numbers, which means start it from the endpoint to start point, then you can do it by using negative numbers.

Let's consider the previous example of Burgers if you want to reverse the list, like-

```
4 double
3 supreme
2 veg
1 chicken
0 cheesy
```

So, you can do it by providing -1 as the start point, endpoint, and step number. Let's understand with the help of an example-

```
burgers= ['cheesy','chicken','veg','supreme','double']  
for n in range(len(burgers)-1,-1,-1):  
    print(n,burgers[n])
```

OUTPUT -

```
4  double  
3  supreme  
2  veg  
1  chicken  
0  cheesy
```

- Here, (burgers-1) represents the start point as 4(reverse order), next -1 shows the endpoint as 0, and the third -1 represents the step number.

Note-

- Range() function arguments must be only integers. You can't pass float or string in start point, endpoint, and step numbers.
- You can't provide 0 in the step number, it gives you an error.

That's all for Range Function in Python, I hope now you have a better understanding of Range Function.



congratulations! You successfully learned Range() Function in Python.

In the next chapter, we will start learning **Functions in Python**

Chapter 9 - Function in Python

What is Function() in Python?

A Function is consists of a number of statements that take some input to perform some calculation and generate output. The Function makes your code more manageable.

You can create a function only one time and use it multiple times. It saves your time.

Suppose you are writing code and you created a function for some task, but in the future, you need to perform the same task again, so instead of writing the full code again, you can simply call this function.

A Function makes code reusable and manageable.

Syntax of Function()–

Syntax of Function() –

```
def function_name(parameters):  
    statement
```

- For defining any function in python, there is a keyword 'def', which means 'define'.

- After using the 'def' keyword we use, function name. Inside the function name, we pass parameters.

Let's have a look at the example of a function()-

```
def greet():  
    print("Hello World")  
greet()
```

- In function, indentation is very important for the function body.
- A colon(:) is compulsory at the end of the function.

How to call a Function?

After defining a function you can call the function anywhere in the program by just writing the function name with the parameters.

Let's have a look at the example-

```
>>>greet('Alex')
```

How to pass parameters in Function()?

You can pass multiple parameters in the function. First, have a look at one example-

```
def greet(name,time):  
    print(f'Good{time}{name},hope you are good')  
greet('Alex','morning')
```

OUTPUT-

Good morning Alex,hope you are good.

- Here, in greet(name, time), name and time are variable or parameters we are passing through function, which we can access it in function later.

You can use user input, which means a user can enter their name and time. Let's see in the example.

```
def greet(name,time)                #function define
    print(f'Good{time}{name},hope you are good')  #Function Body
name=input('enter your name:')      #user input
time=input('enter greet time:')     #user input
greet(name,time)                    #Function calling
```

OUTPUT -

```
enter your name: Alex
enter greet time: Morning
Good Morning Alex, hope you are good.
```

- Here, at the time of function calling `greet(name, time)`, we are passing name and time as a parameter, but if we don't pass anything here, means we left it blank like `greet()`, so error comes. Therefore, we must pass the variable at the time of function call.
- To avoid this error, there is one alternate method, you can pass a value in the variable at the time of function definition.
- Let's understand it with the help of an example.

```
def greet(name='Alex',time='Morning')           #function definition.  
    print(f'Good{time}{name},hope you are good') #Function Body  
greet()                                         #Function calling
```

OUTPUT-

Good Morning Alex, hope you are good.

- Here, in the example, as you see, we pass values at the time of a function definition, and then we call a function with 0 parameters. By doing this, you will not get an error.
- One thing should be kept in mind that if you are using user input as we did in the previous example, then you have to pass values at the time of a function call.

Parameter Overriding-

Overriding means put a new value on the previous one. This case happens in python too. This happened when you pass a value at the time of **function definition** and again you pass another value at the time of **function call**. Then the new value passed at the time of **function call** override the previous value of **function definition**.

Let's understand it with the help of an example-

```
def greet(name='Alex',time='Morning')           #function definition.  
    print(f'Good{time}{name},hope you are good') #Function Body  
greet(name='John')                             #Function calling
```

OUTPUT-

Good Morning John, hope you are good.

- Here, in the example, as you see Output is 'Good Morning John, hope you are good', here name is 'John' not an 'Alex', because 'John' overrides the previous value 'Alex'.

Return in Python.

Return statements allow the function to return a value. After the return statements, no other statement is executed. Return function is used only inside the function.

Syntax of Return Statement-

```
def function_name():  
    statements  
    return [value]
```

Let's understand Return statement with the help of an example-

```
def multiplication(i):  
    return 2*i  
  
print(multiplication(2))  
print(multiplication(3))  
print(multiplication(4))
```

OUTPUT -

```
4  
  
6  
  
8
```

- Here, first value of i is 2 and it is multiplied by 2 that is 4 and the return statement is returning a value 4.

Some Examples using Function().

Here, we will discuss some basic programs, which are made using a function. So, you get a clear idea of function.

Let's start with our first program that is a program to calculate the area of Circle.

Example 1- Program to Calculate an Area of Circle-

```
def area(radius):  
    print(3.142*radius*radius)  
    radius=int(input('enter a radius of circle-'))  
    area(radius)
```

OUTPUT-

```
enter a radius of circle- 5
```

```
78.55
```

Here, first, we passed 'radius' as a parameter in a function definition, then we have written the formula of area of a circle in print. After that, we asked the user to enter the radius of the circle by using input. Then we called the function 'area' and inside 'area' function we passed the 'radius' parameter.

Example 2- Program to Calculate Volume of Cylinder-

The formula of cylinder volume= area of circle * length of a cylinder

Here, we need to calculate the area of a circle, then use the area of a circle to find the volume of a cylinder.

```
def area(radius):
    return 3.142*radius*radius    #we are returning value of area, and store it
def vol(area,length):
    print(area*length)
radius=int(input('enter a radius of circle-'))
length=int(input('enter a length of cylinder-'))
area_calc=area(radius)           #we are storing the return value of area funct
vol(area_calc,length)
```

OUTPUT-

enter a radius of circle- 5

enter a length of cylinder-6

30

- Here, we are storing the return value of area function in the variable named area_calc. But, if you don't want to store the return value in any variable, then you can directly pass the **function inside the function**. Like that-

vol(area(radius), length)

- Here, **area(radius)** is also a function which we are passing inside another function **vol**.



That's all for Function() in Python, I hope now you have a better understanding of Function() in Python.

congratulations! You successfully learned Function() in Python.

In the next chapter, we will start learning **Variable Scope in Python**

Chapter 10 - Variable Scope in Python

What is Variable Scope?

Variable Scope in Python shows the visibility of the variable in the code. The visibility of variables declares on the basis of the variable definition location. Where the variable is defined, its scope decides at the same time.

You will understand the whole concept in a few minutes.

Types of Variable Scope–

There are basically two types of Variable Scope–

1. Local Scope
2. Global Scope

Python Global Scope–

Global Scope is when you define a variable outside the function. The accessibility of the global variable is inside the function as well as outside the function.

Let's have a look at the example of Global Variable–

```
my_name='Alex'          #global variable

def print_name():
    print('name inside the function',my_name)

print_name()

print('name outside the function',my_name)
```

OUTPUT-

```
name inside the function-Alex
name outside the function-Alex
```

Here, we declare variable **my_name** as a global variable. This means we can use **my_name** variable inside the function as well as outside the function. The scope of the variable is global because we define **my_name** outside the function.

After looking at the example, I hope you have a clear idea about the Global Variable definition. Let's revise it once- **The variable which is declared outside the function is known as a global variable.**

Python Local Scope–

Local Scope in python is when you define a variable inside the function.

If you define the variable inside the function so it is a local variable. Which means it is only accessible inside the function, and not outside the function.

Let's understand with the help of an example-

```
my_name='Alex'

def print_name():
    my_name='John'
    print('name inside the function',my_name)

print_name()

print('name outside the function',my_name)
```

OUTPUT-

```
name inside the function-John
name outside the function-Alex
```

Here, **my_name='Alex'** is a Global variable, but **my_name='John'** is a local variable. Which overrides the global variable value inside the function, but not outside the function. Therefore, as you see in the output 'John' print as the name inside a function. And 'Alex' print as a name outside the function.

You get an error when you don't define the variable as global (outside the function). But try to access a local variable in a global scope.

Let's understand with the help of an example-

```
def print_name():  
    my_name='John'  
  
print_name()  
  
print('name outside the function',my_name)
```

OUTPUT-

```
NameError: name 'my_name' is not defined.
```

You will get an error because, in the example, we define **my_name** inside the function. Therefore its scope is local and we can not access it globally.

Global Keyword-

As we saw in the global keyword example, **my_name='John'** override the value of **my_name='Alex'** inside the function. But if you want that it doesn't override the value inside the function, then you can do it by using **Global Keyword**.

Let's see how Global Keyword works-

```
my_name='Alex'

def print_name():
    global my_name      #define local variable as global
    my_name='John'
    print('name inside the function',my_name)

print_name()

print('name outside the function',my_name)
```

OUTPUT-

```
name inside the function-Alex
name outside the function-Alex
```

As you can see, by using **Global Keyword**, the value of the local variable is not printed. The only global variable value is printed two times.

Due to **Global Keyword**, the value gets changes globally inside and outside the function.

That's all for the Variable Scope in Python. I hope now you have a better understanding of the Variable Scope in Python.

Congratulations! You successfully learned the Variable Scope in Python

In the next chapter, we will start learning **Dictionaries in Python**.

Chapter 11 - Dictionary in Python

What is Dictionary in Python?

Dictionary is a new data type and its a mapping type, which means it is a set of key, value pairs. Dictionary is an unordered collection of data values. A Dictionary is changeable and indexed.

How to Create a Dictionary in Python?

For creating a dictionary, a curly bracket {} is used. Let's see how to create a dictionary in python-

```
ninja_belts= {"crystal":"red","ryu":"black"}  
print(ninja_belts)
```

OUTPUT -

```
{'crystal' : 'red', 'ryu' : 'black'}
```

Here, “crystal” is a **Key** and “red” is **value**. Key and value are put into ” ” because values are by default strings.

Values can be of any data type and it can repeat, but a key must be unique and fixed type.

You can create a dictionary by using a built-in function `dict()`, like that-

```
dict= {}
```

Accessing Elements from Dictionary.

If you want to access value from a dictionary, you can do that by **Key**, whatever the key is assigned to the value. This key can be used inside a square bracket `[]` or with the `get()` method.

Let's have a look at the example of a **square bracket `[]`** method-

```
ninja_belts= {"crystal":"red","ryu":"black"}  
print(ninja_belts['ryu'])
```

OUTPUT-

red

Let's have a look at an example of **`get()`** method-

```
ninja_belts= {"crystal":"red","ryu":"black"}  
print(ninja_belts.get('ryu'))
```

OUTPUT-

red

How to check the availability of a Key in the Dictionary?

If you want to check that that certain key is present in the dictionary or not, then you need to use an **'in'** keyword in python.

The syntax for checking a key in a dictionary-

```
key in dictionary
```

> Here, a **key** may be any particular key name, which we are looking for.

-> **in** is a keyword in python.

-> **Dictionary** is a particular dictionary in which we are checking.

Let's understand with the help of an example-

```
>>>ninja_belts= {"crystal":"red","ryu":"black"}  
>>>'yoshi' in ninja_belts
```

OUTPUT -

False

Here, we are getting **False** as an output because **'Yoshi'** is not present in the dictionary **ninja_belts**.

Another Method-

There is one more method for checking a key in a dictionary. Let's see in the example below-

```
>>>ninja_belts= {"crystal":"red","ryu":"black"}  
>>> ninja_belts.keys()
```

OUTPUT-

```
dict_keys(['crystal', 'ryu'])
```

Here, we are getting the full list of **keys** in the '**ninja_belts**' dictionary. Therefore, you can check the key for which you are looking for,

Typecasting of Keys and Values into List.

If you want to typecast keys of a dictionary into a list, so you can do it by using a list keyword.

Let's see in the example-

```
>>>ninja_belts= {"crystal":"red","ryu":"black"}  
>>> list(ninja_belts.keys())
```

OUTPUT-

```
['crystal', 'ryu']
```

Here, we are getting all the keys in the form of a list in output.

You can also typecast the **values of a dictionary** into a list. First, we get all the values of the dictionary and then store them in a variable in the form of a list.

Let's see in the example-

```
>>>ninja_belts= {"crystal":"red","ryu":"black"}  
>>> ninja_belts.values()  
  
output-  
dict_values(['red', 'black'])
```

Here, we are getting all the values of the dictionary, now we store values in the variable in the form of a list.

Let's see in the example below-

```
>>> val=list(ninja_belts.values())  
>>> val  
  
OUTPUT-  
['red', 'black']
```

Here, '**val**', is the variable in which we are storing our values in the form of a list.

By doing this, you can typecast keys and values into the list.

Now, you can work on that list, like you can count how many instances in values, which means you can check how many times one value is used.

Let's see in the example-

```
>>> vals.count('black')
```

OUTPUT -

```
1
```

The output is 1 because 'black' is used only 1 time in the dictionary.

Python Dictionary Append.

You can append a new key into the dictionary very easily. Without wasting your time, Let's see in the example-

```
>>> ninja_belts['Yoshi']='red'
```

```
>>> ninja_belts
```

OUTPUT-

```
{'crystal': 'red', 'ryu': 'black', 'Yoshi': 'red'}
```

Here, we have added a new **key** as 'Yoshi' and **value** as 'red'.

Removing Elements from Dictionary.

You can remove elements from the dictionary in various ways. We will discuss approx all the ways to remove an item from the dictionary.

Using pop() method-

The pop() method is used to remove the item with a particular key name. Let's see in the example-

```
>>>ninja_belts= {"crystal":"red","ryu":"black","Yoshi":"red"}
```

```
>>>ninja_belts.pop("Yoshi")
```

```
>>>ninja_belts
```

OUTPUT-

```
{'crystal': 'red', 'ryu': 'black'}
```

Using popitem() method-

The popitem() method item removes only the last item from the dictionary. Let's see in the example-

```
>>>ninja_belts= {"crystal":"red","ryu":"black","Yoshi":"red"}
>>>ninja_belts.popitem()
>>>ninja_belts
```

OUTPUT-

```
{'crystal': 'red', 'ryu': 'black'}
```

Using del Keyword-

The 'del' keyword deletes the element with the particular key name. Let's see in the example-

```
>>>ninja_belts= {"crystal":"red","ryu":"black","Yoshi":"red"}
>>>del ninja_belts["Yoshi"]
>>>ninja_belts
```

OUTPUT-

```
{'crystal': 'red', 'ryu': 'black'}
```

The "del" Keyword is also used for deleting the whole dictionary. Let's see in the example-

```
>>> del ninja_belts
```

```
>>> ninja_belts
```

OUTPUT-

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'ninja_belts' is not defined

Using clear() method-

By using a clear() method, you can clear your dictionary, which means it makes your dictionary empty. Let's see in the example-

```
>>>ninja_belts= {"crystal":"red","ryu":"black","Yoshi":"red"}
```

```
>>>ninja_belts.clear()
```

```
>>>ninja_belts
```

OUTPUT-

```
{}
```

Python Nested Dictionary.

You can create a nested dictionary in Python. A nested dictionary means, dictionary inside another dictionary.

Let's see how to create a nested dictionary in python-

```
>>>my_dict={1:"hello",2:"I",3:{'A':"am","B':"Alex","C':"John"}}
>>>my_dict
```

OUTPUT-

```
{1: 'hello', 2: 'I', 3: {'A': 'am', 'B': 'Alex', 'C': 'John'}}
```

Here, we first created a **my_dict** dictionary inside **my_dict**, we have created another dictionary as **3:{'A':"am","B':"Alex","C':"John"}**.

Update value in Dictionary.

You can change or update the value of the dictionary by pointing to its key name. Let's see in the example how to update the value in the dictionary.

```
>>>ninja_belts= {"crystal":"red","ryu":"black","Yoshi":"red"}
>>>ninja_belts["Yoshi"]="green"
>>> ninja_belts
```

OUTPUT-

```
{'crystal': 'red', 'ryu': 'black', 'Yoshi': 'green'}
```


Copy a Dictionary.

If you want to make a copy of your dictionary, then you can do it by a `copy()` method. Let's have a look in the example-

```
>>>ninja_belts= {"crystal":"red","ryu":"black","Yoshi":"red"}  
>>>new_ninja_belts=ninja_belts.copy()  
>>>new_ninja_belts
```

OUTPUT-

```
{'crystal': 'red', 'ryu': 'black', 'Yoshi': 'red'}
```

Here, we are copying our “**ninja_belts**” dictionary into the “**new_ninja_belts**” dictionary by just using a **copy()** method.

One more method to copy-

You can copy your dictionary by one more method that is a built-in method in python **dict()**. By using **dict()** method you can copy your dictionary. Let's see in the example-

```
>>>ninja_belts= {"crystal":"red","ryu":"black","Yoshi":"red"}  
  
>>>new_ninja_belts=dict(ninja_belts)  
  
>>>new_ninja_belts
```

OUTPUT-

```
{'crystal': 'red', 'ryu': 'black', 'Yoshi': 'red'}
```

That's all for the **Dictionary in Python**. I hope now you have a better understanding of the **Dictionary in Python**.

Congratulations! You successfully learned the **Dictionary in Python**.

In the next chapter, we will start learning **Sets in Python**.

Chapter 12 - Set in Python

What is Set in Python?

A set is a collection of unordered and unindexed items. It is similar to list but Sets doesn't allow duplicate items. A set is defined by curly braces {}. Sets perform different mathematical operations like union, intersection, and differences.

Let's see how to create a set in python.

```
my_set={"orange","red","green","yellow"}  
print(my_set)
```

OUTPUT-

```
{'yellow', 'red', 'green', 'orange'}
```

In the output, items are not in the same order as we put in the set. This shows that sets are an unordered collection of an item.

How to Access an Item in Sets?

As we discuss, the set is an unindexed collection of items, that's why you can not access items in sets. But you can check that specific item is present in the set or not by using 'in' keyword. You can loop through by for loop.

Let's understand with the help of an example by using **for loop**–

```
my_set={"orange","red","green","yellow"}  
for i in my_set:  
    print(i)
```

OUTPUT -

orange

green

red

yellow

Using **'in'** Keyword-

```
my_set={"orange","red","green","yellow"}  
print("red" in my_set)
```

OUTPUT -

True

Here, we are checking that “red” is present in the set or not, and we are getting output as True.

Change an Item in Sets.

After creating a set, you can't change or modify its element. You can add an item in sets but not update any item.

Add an item in Sets.

You can add an item to the sets. If you want to add only one item, then you can add it by using **add()** method. But if you want to add more than one item, then use the **update()** method.

Let's have a look at **add()** method example-

```
my_set={"orange","red","green","yellow"}  
my_set.add("Black")  
print(my_set)  
  
OUTPUT-  
{'yellow', 'red', 'Black', 'orange', 'green'}
```

Let's have a look at **update()** method-

```
my_set={"orange","red","green","yellow"}  
my_set.update(["Black","White","Pink"])  
  
OUTPUT-  
{'Pink', 'yellow', 'red', 'Black', 'orange', 'green', 'White'}
```

Here, by using **the update()** method, we are adding more than one item in the set.

Remove an Item from Sets.

If you want to remove any item from the set, then there are multiple ways in python to remove an item. We will discuss all the methods one by one.

Using remove() method-

You can delete any item from the set by using the remove() method.

Let's see in the example-

```
my_set={"orange","red","green","yellow"}  
my_set.remove("red")  
  
OUTPUT-  
{'yellow', 'green', 'orange'}
```

Using discard() method-

Discard() method works mostly the same as remove() method. You can remove the item from the set by using the discard() method.

Let's see in the example-

```
my_set={"orange","red","green","yellow"}  
my_set.discard("red")
```

output-

```
{'yellow', 'green', 'orange'}
```

Using pop() method-

The pop() method removes the last element of the set. But, as we know that set is unordered and you don't know which item is the last item in the set.

Let's see in the example below-

```
my_set={"orange","red","green","yellow"}  
i=my_set.pop()  
print(i)          #deleted item  
print(my_set)
```

OUTPUT-

```
'yellow'  
{'red', 'green', 'orange'}
```

Here, “yellow” is deleted from the list. we are storing a deleted value in “i” and printing it to check which item is deleted.

Using clear() method-

The clear() method makes a set empty, which means no item will be there after using the clear() method.

Let's see in the example-

```
my_set={"orange","red","green","yellow"}  
my_set.clear()  
print(my_set)
```

OUTPUT-

```
set()
```


In the output, we get an empty set.

Using del Keyword-

If you wanna delete your set completely then you should use **del** keyword.

Let's understand with the help of an example-

```
my_set={"orange","red","green","yellow"}  
del my_set  
print(my_set)
```

OUTPUT-

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'my_set' is not defined
```

Concatenation of two Sets.

If you wanna join two sets into one, then you can do it by using an **update()** method and by **union()** method.

Let's have a look at **update()** method-

```
my_set1={"orange","red","green","yellow"}  
my_set2={"Black","White","Pink"}  
my_set1.update(my_set2)  
print(my_set1)
```

OUTPUT-

```
{'Pink', 'yellow', 'red', 'Black', 'orange', 'green', 'White'}
```

Here, we have joined two sets into one by using the `update()` method.

Let's have a look at `union()` method example-

```
my_set1={"orange","red","green","yellow"}  
my_set2={"Black","White","Pink"}  
my_set3=my_set1.union(my_set2)  
print(my_set3)
```

OUTPUT-

```
{'Pink', 'yellow', 'red', 'Black', 'orange', 'green', 'White'}
```

Here, both method **`update()`** and **`union()`** doesn't allow duplicate items in a set.

Python Set Operation.

Set is used to perform different set operations like union, intersection, and symmetric difference.

Union of Set

Union of set means joins or concatenate two sets into one. It combines all the items of both sets into one.

You can perform a union in python by using `|` symbol, or you can use `union()` method.

Let's understand with the help of an example-

```
X={1,2,3,4,5,6}
```

```
Y={6,7,8,9,10}
```

```
print(X|Y)
```

OUTPUT -

```
{1,2,3,4,5,6,7,8,9,10}
```

The intersection of Set

The intersection of set means, select only common items from both the sets. It doesn't take all the items from the set, only take common items.

You can perform an intersection by using & operator or by using an intersection() method.

```
X={1,2,3,4,5,6}
```

```
Y={6,7,8,9,10}
```

```
print(X & Y)
```

OUTPUT-

```
{6}
```

Set Difference

A difference of two sets means $X-Y$ (means items that are in X but not in Y), similarly $Y-X$ (means items that are in Y not in X).

You can perform a difference by using $-$ operator or by using the difference() method.


```
X={1,2,3,4,5,6}
```

```
Y={5,6,7,8,9,10}
```

```
print(X-Y)
```

OUTPUT-

```
{1,2,3,4}
```



That's all for the **Sets in Python**. I hope now you have a better understanding of the Sets in Python.

Congratulations! You successfully learned the **Sets in Python**.

In the next chapter, we will start learning **Classes in Python**.

Chapter 13 - Classes in Python

What are Classes and Objects in Python?

Python is an Object-Oriented Programming Language, which is based on objects. An object is basically a collection of functions and attributes. And Class shows the blueprint of an object.

You can take a class as a prototype or sketch of a car, which includes all the information about its machines, body, and design. Based on this information you can build a car. Here, the car is an object. Because you can make many cars from this prototype. Similarly, you can create multiple objects with one Class.

How to Create a Class?

For creating a Class in python, the 'class' keyword is used. After 'class' keyword use class name and at the end, use colon (:).

Syntax of Class declaration-

```
class class_name:
```

Let's have a look in the example, to create a class in Python-

```
class Car:  
    //class functions and attributes
```

Methods and attributes in Class.

After declaring a class, you can define functions or methods inside the class. You can create multiple methods inside the class.

You can define attributes inside the class as many as you want.

Let's understand with the help of an example-

```
Class Car:  
    car_name="BMW"  
    def change_name(self,new_name):  
        self.name=new_name
```

Here, we have created a class '**car**', inside the class 'car', we have created an attribute name as **car_name** and one method **change_name**. Similarly, you can create any method and attributes inside the class.

You can also create a constructor function, like the `__init__()` function. We will discuss the `__init__()` function later in detail, but here I am gonna tell you how you can create an `__init__()` function.

Let's have a look at the example-

```
class Planet:
    def __init__(self):
        self.name="Hoth"
        self.radius=200000
        self.gravity=5.5
        self.system="Hoth system"
```

Here, an `__init__()` function is **constructor function**. `self` is an instance. We will discuss this later in detail.

How to create an Object in Class?

After class declaration and function declaration, next, we need to create an object of a class. Object creation is very easy in Python.

You can create an object by class name and object name. Let's have a look at the syntax of object creation.

```
object_name=class_name()
```

Look 😊 it is as simple as that.

Let's see in the example, how we can create an object of our class **Planet**.


```
class Planet:
    def __init__(self):
        self.name="Hoth"
        self.radius=200000
        self.gravity=5.5
        self.system="Hoth system"
hoth=Planet()
```

Here, 'hoth' is an object of **Planet** class. **One important thing** you should keep in mind that an object of a class is created from outside of the class, which means start writing from the corner, without any indentation. It is considered as outside of the class.

How to Access attributes or Methods of Class?

For accessing any attribute or method of a class, you need to use an object name and dot operator(.). Let's see its syntax-

```
object_name.attribute_name    #for attribute
object_name.function_name()    #for function or method
```

After looking at syntax, now let's understand with our Planet class example, how you can call a function or attributes.

```
class Planet:
    def __init__(self):
        self.name="Hoth"
        self.radius=200000
        self.gravity=5.5
        self.system="Hoth system"
    hoth=Planet()
    print(f'Name is:{hoth.name}')    #attribute calling
```

Here, in 'hoth.name' 'hoth' is an object of a class and name is an attribute of a class.

You can also access the method or functions of a class. First, we will create one method then we will call this method outside of the class.

Let's see in the example of our **Planet** class.

```
class Planet:

    def __init__(self):

        self.name="Hoth"

        self.radius=200000

        self.gravity=5.5

        self.system="Hoth system"

    def orbit(self):

        return f'{self.name} is orbitting in {self.system}'

hoth=Planet()

print(f'Name is:{hoth.name}')

print(hoth.orbit())          # method calling
```

Here, we are calling the **orbit** method by object name **hoth**.

Now, I hope you have learned, how to call the method and attribute of a class.

The Self Parameter.

Self is a pointer to the class instance, which means the Self shows an object of a class. The self parameter is used to access attributes which belong to the class.

Whenever you create a function inside a class, you must write Self first, because it represents the class.

For example-

```
class person
    def name(self, first_name, last_name):
        self.first_name=first_name
        self.last_name=last_name
```

To call any attribute or variable of a class, always use Self, because it represents that it is a member of a class.

Let's understand with the help of an example-

```
def print_name(self):
    print(self.first_name,"hi",self.last_name)
```

Here, we are calling attribute **first_name** and **last_name** with the help of the Self parameter.

Always pass the Self in every member function or attribute of a class, then pass other arguments. Suppose if member function doesn't take any arguments, which means function with no arguments, then also pass the Self.

For Example-

```
def print_name(self):  
    print(self.first_name,"hi",self.last_name)
```

Here, function `print_name` doesn't have any argument, but we have passed the `Self`.

The `__init__()` Function in Class.

A function with start with a double underscore (`__`) is known as special functions. An `__init__()` function is built-in function. All classes have an `__init__()` function, which executes when the class initiates.

By using an `__init__()` function, you can create a class with different values for arguments. An `__init__()` function is known as special functions.

You can create multiple instances with the help of an `__init__()` function.

Let's understand with the help of an example-

```
class Planet:
    def __init__(self,name,age):
        self.name=name
        self.age=age

#1st Object
planet1=Planet('Hoth',22)
print(f'Name:{planet1.name}')
print(f'Age:{planet1.age}')

#2nd Object
planet2=Planet('Naboo',25)
print(f'Name:{planet2.name}')
print(f'Age:{planet2.age}')
```

Here, we have created multiple objects of a class by `__init__()` function, and we have provided different values to different objects.

That's all for the **Classes in Python**. I hope now you have a better understanding of the **Classes in Python**.

Congratulations! You successfully learned the **Classes in Python**.

In the next chapter, we will start learning **Methods & Attributes in Python**.

Chapter 14 - Methods & Attributes in Python

Instance Methods and Attributes in Python.

The methods and attributes which we create in the class are known as the Instance method and Instance attributes.

Instance methods and Instance attributes are used only with the instance of a class or object of a class.

Let's see in the example what is instance method and attributes in python-

```
Class Car:  
  
    def change_name(self,new_name):  
  
        self.name=new_name
```

Here “**change_name**” is an instance method and “**new_name**” is an instance attribute.

Class level Methods and Class level Attributes in Python.

Class level Attributes-

When we create any attribute inside the class, not inside any function of a class, it is known as class-level attributes.

Confused?...

Don't worry! I will explain to you with the help of an example-

```
class Planet
    shape="round"
```

Here, the “shape” attribute is a class-level attribute, because it is created inside the class not under any function of a class.

In the previous example “new_name” is not a class-level attribute because it is inside the function of a class.

I hope now you understand class-level attributes 😊.

Class level Methods-

Class level Methods are different from normal methods. In order to create class level method, you need to write **decorator @classmethod** and inside the method, you need to write “cls” unlike in normal method in python, we write self.

Let's understand with the help of an example-


```
class Planet  
    shape="round"  
    @classmethod      #decorator  
    def common(cls)  
        //method body
```

Here, “common” is a class-level method, and inside a class-level method, you need to pass “cls”.

Difference between Instance Attributes and Class level Attributes.

Class Level attributes are common to the class, and it is accessible by class name as well as with instance or objects.

But,

Instance attributes are not accessible with a class name.

Let's understand with the help of an example-

```
Class Planet
    shape="round"
print(Planet.shape)    #Planet is a class name
```

OUTPUT -

round

Here, we call the class-level attribute “shape” in a print statement with a class name. It didn’t give any error, because class-level attributes are accessible with a class name.

Similarly, a class-level attribute is accessible with instances or objects of a class.

Let’s have a look at the example-

```
Class Planet
    shape="round"
naboo=Planet()
print(naboo.shape)    #naboo is object of a class
```

OUTPUT -

round

Here, “naboo” is an object of a class and we are accessing class-level attributes with the object.

But, if you want to access any **instance attribute** with the class name, it will give an error.

Let's see in the example-

```
Class Car:
    def change_name(self,new_name):
        self.name=new_name
print(Car.new_name)      #Car is a class name
```

OUTPUT -

Error

Here, “new_name” is an instance attribute and we are calling it with a class name, which we can't. That's why an error is coming.

In Short- The main difference between class-level attributes and instance attribute is that you can call a class-level attribute with a class name but you can't call instance attributes with a class name.

Difference between Instance Methods and Class level Methods.

As you learned in the difference of class-level attributes and instance attributes, the same with methods.

Instance methods are the methods that are created for instance of a class. It shows individuals property and it is accessible only with an instance of a class. which means you can call Instance methods only with the object of a class.

But,

Class level methods are common for all and it is accessible with both a class name and object of a class.

Let's see how you can access class-level methods-

```
class Planet
    shape="round"
    @classmethod      #decorator
    def common(cls)
        return f'All planets are {cls.shape}'
```

So, here in **class level method**, you don't need to pass "self", you can use "cls". With the help of "cls", you can call class attributes like- **cls.shape**.

You can access this class-level method with the class name as well as to object to the class.

Let's see in the example-

```
class Planet
    shape="round"
    @classmethod      #decorator
    def common(cls)
        return f'All planets are {cls.shape}'
print(Planet.common())
```

OUTPUT -

All planets are round.

Here, we are calling the **class-level method** “common” with a class name **Planet**.

You can also call class-level methods with the object of the class. Let’s see in the example-

```
class Planet
    shape="round"
    @classmethod      #decorator
    def common(cls)
        return f'All planets are {cls.shape}'
naboo=Planet()
print(naboo.common())    #naboo is a object of a class
```

OUTPUT-

All planets are round.

Here, naboo is an object of a class and we are calling a class-level method with an object.

But, you can't access the instance method with the class name.

Static Method.

The static method has access to only its own attributes. It doesn't have access to class-level attributes or methods not to instance attributes or methods. Which means Static methods can't use **Class-level(methods & attributes)** and **Instance(methods & attributes)**.

A static method can only use its own attributes or variable.

Let's see how to create a static method-

```
@staticmethod  
def spin(speed='2000 mileperhour')  
    return f'Planet spins at {speed}'
```

Here, “speed” is a static method attribute, therefore static method can only use this attribute.

NOTE- The Static method is accessible with a class name as well as with instance or object. This means you can call a static method with the class name and with the object name too.

The static method neither takes “self” nor “cls”.

For example-

```
class Planet
    shape="round"
    @classmethod      #decorator
    def common(cls)
        return f'All planets are {cls.shape}'
    @staticmethod
    def spin(speed='2000 mileperhour')
        return f'Planet spins at {speed}'
print(Planet.spin())    #Planet is a class name
```

OUTPUT-

Planet spins at 2000 mileperhour

Here, we are calling a static method “spin” with a class name. Similarly, you can call a static method with the object of a class.

Let's see-


```
class Planet
    shape="round"
    @classmethod      #decorator
    def common(cls)
        return f'All planets are {cls.shape}'
    @staticmethod
    def spin(speed='2000 mileperhour')
        return f'Planet spins at {speed}'
naboo=Planet()
print(naboo.spin())    #naboo is a object of a class
```

OUTPUT-

Planet spins at 2000 mileperhour

Here, we are calling a static method with the object of a class “naboo”.

That’s all for the **Method and Attributes in Python**. I hope now you have a better understanding of the **Method and Attributes in Python**.

Congratulations! You successfully learned the **Method and Attributes** in Python.

In the next chapter, we will start learning **Modules and Packages in Python**.

Chapter 15 - Modules & Packages in Python

What are the Modules in Python?

A Module is a section or segment in which different functions are present and these functions perform some different operations.

A module helps you to logically organize your code. Suppose you have to perform some specific operation in your code and your code is too long and messy, so instead of writing more code, you can simply import the module. After importing the module in your code you can use it in your program.

By doing so, you save your time and your code will not become messy. Because writing the full code again in your program makes it messy and unable to read.

By importing modules in your program, you can simply access its functionality in your code.

Packages in Python.

A Package is nothing but a collection of modules. It is also imported into programs.

In Package, several modules are present, which you can import in your code.

Suppose, you are working on any project in python, and which require a lot of code with different classes and functions. So instead of writing all code in one file, you need to divide it into different files. Which you can import it later whenever you have need of this code.

For example-

You are working on one file named **Classes.py** and there are lots of code including different functions and attributes. And you want to do more work, but a file with lots of code looks messy. So instead of writing all code in one file, you can create a new file with the name **Test.py**. In **Test.py** you can use all the functions and attributes of **Classes.py**.

In the next section, we will see how you can import a module into your new file **Test.py**.

How to Import Modules?

As we are discussing, to import functions and attributes of **Classes.py** into **Test.py** file. So, for importing any module in a file you need to write-

```
from Classes import Planet
```

Here, **Classes** is the name of a class from where you are importing modules. And **Planet** is the name of the module, which means what you want to import.

So, after writing this line of code, you can use the **Planet class** inside your **Test.py** file.

This means you can write this code in your **Test.py** file and it works perfectly fine because you have imported Planet class here.

```
from Classes import Planet

Class Test

    shape="round"

naboo=Planet("naboo",300,8,'Naboo')

print(f'Name:{naboo:name}')
```

Here, we are using **Planet class** and its attribute inside our **Test class**.

How to create and Import a Package?

A Package is a collection of modules. To create a package, you need to create a folder, suppose you create a folder name as **Space**.

To tell python that **Space** is a **package**, just create an **__init__.py** file. Which is enough to tell python that **Space** is a package.

Don't write anything in **__init__.py** file, just left it blank.

then,

Inside **Space**, package creates a new file named **Planet.py** file. This **Planet.py** acts as a module for the **Space** package.

Then, in **Planet.py** file just paste the code from the previous **planet.py** file.

NOTE- Name of files may be anything it is up to you, these names are just for reference

So, after pasting your code in **Planet.py** file, create one more file inside the **Space** package name as **calc.py**, where different functions are performed.

After creating **calc.py**, create one file outside from the **Space Package** just name as **Classes.py**, and in that file import packages and its modules.

Are you Lost with too many file names?

Don't worry! we will revise all the files which we have created.

- **One Package – Space**
 - **__init__.py**
 - **Planet.py (act as a Module of a Package)– Paste your code from the previous file**
 - **calc.py (Different functions are performed here)**

- **Classes.py (Outside of the Package)**

I hope now you understand.

Import a Package-

Now, let's import the package and its modules in the **Classes.py** file. To import package **Space** and its module in **Classes.py** file, you need to write following line of code-

```
from Space.Planet import Earth
```

Here, as you know **Space** is a package name. **Planet** is the module name. And **Earth** is the class name inside the **Planet** module.


You can import multiple functions and classes from the same module just by writing comma (,).

Let's see in the following example-

```
from Space.calc import earth_mass, earth_vol
```

Here, **Space** is a package name, **calc** is module name and **earth_mass** and **earth_vol** are the function name inside the **calc** module.

NOTE- By importing packages and modules in another file. And when you run this file the output of all modules which you have imported will be shown on your file.



That's all for the **Modules and Packages in Python**. I hope now you have a better understanding of the **Modules and Packages in Python**.

Congratulations! You successfully learned the **Modules and Packages in Python**.

In the next chapter, we will start learning **List Comprehensions in Python**.

Chapter 16. List Comprehension in Python

How to Create a List using For Loop.

You can create a list using For Loop. Here, in this tutorial, I will teach you with the help of an example. Suppose, there is a list-

```
prizes=[5,10,50,100,1000]
```

This is the list of employees prize, but boss announces a double bonanza offer, so to update the list with double bonanza, you don't need to override the list, you can do it with for loop.

Let's see how you can do it by using for loop-


```
prizes=[5,10,50,100,1000]
dbl_prize=[]
for prize in prizes;
    dbl_prize.append(prize*2)
print(dbl_prize)
```

OUTPUT-

```
[10,20,100,200,2000]
```

Here, we have created an empty list `dbl_prize`, then we use for loop. Inside for loop, we use the `append` method to append a new list in the multiple of 2. Therefore, we are getting output as `[10,20,100,200,2000]`.

Using List Comprehension.

By using List Comprehension, you can create a list in python in one line only. Unlike in for loop we have written two to three lines of code, but in the list comprehension, we can perform the same task in one line.

First, let's see the syntax of List Comprehension-

```
list_name=[expression for item in list]
```

Here, an **expression** may be anything, which you wanna give. Like in for Loop, we give expression as “`prize*2`”. So you can pass any expressions like that. The expression must return a value.

Item is the object or value in the list. It may be anything. Like in for loop example, the item is a “prize”. You can give it as ‘i’ or anything.

The **list** is the name of **list** that can return its elements only one at a time. In for loop example, list is “prizes”.

Now, if you have to perform the same task as we did by for loop, you can do it by **List Comprehension**.

Let’s see how you update the prize list with double bonanza with the help of list comprehension-

```
prizes=[5,10,50,100,1000]
dbl_prizes=[prize*2 for prize in prizes]
print(dbl_prizes)
```

OUTPUT-

```
[10,20,100,200,2000]
```

Here, we have written in one line as “**dbl_prizes=[prize*2 for prize in prizes]**”. Unlike in for loop, we have to write two or three lines of code.

Here, prize*2 is an expression.

prize is item

and prizes is the name of the list.

One More Example-

Let's take one more example to understand List Comprehension. Suppose you have a list of numbers like-

```
nums=[1,2,3,4,5,6,7,8,9,10]
```

and you have to convert this list into squared even numbers. So you can do this with two methods.

The first method is by using For Loop. And the second method is List Comprehension.

I will discuss both methods here. Let's start with the For Loop method.

```
nums=[1,2,3,4,5,6,7,8,9,10]
squared_even_nums=[]
for num in nums:
    if(num**2)%2==0:
        squared_even_nums.append(num**2)
print(squared_even_nums)
```

OUTPUT -

```
[4,16,36,64,100]
```

Here, first, we create an empty list as “**squared_even_nums**”. Then we use For Loop method, inside **For Loop** we use **if condition** to check

that number is even or odd. And if the number is even then we perform **multiplication with 2**. Then print the list.

By List Comprehension-

The same task is performed in less line of code with the List Comprehension method. Let's see how to perform in List Comprehension-

```
nums=[1,2,3,4,5,6,7,8,9,10]
squared_even_nums=[num*2 for num in nums if(num**2)%2==0]
print(squared_even_nums)
```

OUTPUT-

```
[4,16,36,64,100]
```

Here, the same task is performed with one line of code. That's why List Comprehension is easy as compare to For Loop.

That's all for the **List Comprehension in Python**. I hope now you have a better understanding of the **List Comprehension in Python**.

Congratulations! You successfully learned the **List Comprehension in Python**.

In the next chapter, we will start learning **Map, Filter, and Lambda in Python**.

Chapter 17- Map, Filter, and Lambda in Python

Map in Python.

A map() function is a way to make a list, apply some kind of function on each data item within a list. And to change the item and create a new list without updating items within it.

Syntax of a map() function-

```
map(function, data)
```

Here, a **function** performs some operation. And **data** is a list that we want to update.

Let's understand with the help of an example-

Here, in the example, we have to jumble or shuffle the words.

```
from random import shuffle

def jumble(word):
    anagram=list(word)
    shuffle(anagram)
    return "".join(anagram)

words=["Beetroot","Carrots","Potatoes"]

anagram=[]

print(list(map(jumble,words)))
```

Here, in “**from random import shuffle**” shuffle is a by-default package.

In “**print(list(map(jumble,words)))**”, a list typecasts into a list. A **jumble** is a function. And “**words**” is data or list.

Filter in Python.

A Filter() method is an easy way to take a collection or list and filter some values based on function result and this gives a new filter collection.

Syntax of Filter() method-

```
filter(function,list)
```

Suppose there is a grade list, consist of different grades-

```
grades=['A','B','F','C','F','A']
```

but inside this list, we don't want to show less grade, so to remove less grade we need to filter the list.

For example from grades=['A', 'B', 'F', 'C', 'F', 'A'] we want to remove 'F' grade, so for this task, we need to create a function which removes 'F'.

```
grades=['A','B','F','C','F','A']

def remove_fails(grade):
    return grade != 'F'

print(list(filter(remove_fails,grades)))
```

OUTPUT-

```
['A','B','C','A']
```

Here, in “**print(list(filter(remove_fails, grades)))**” a list typecasts into the list. “**remove_fails**” is a function which removes 'F' and **grades** is a list in which we wanna apply filter.

Lambda in Python–

A lambda in python is an anonymous function, which means it doesn't require any names. It is used when there is a need of function which is used only once, so at that time, you can use lambda. A Lambda is used once.

Syntax of Lambda function-

```
lambda arguments: statements
```

A lambda function can take more than one number of arguments.

Suppose, you want to square numbers in a list, then you can use a lambda function for this task.

Let's understand with the help of an example-

```
nums=[1,2,3,4,5,6]
print(list(map(lambda n:n*n,nums)))
```

OUTPUT-

```
[1,4,9,16,25,36]
```

Here, a lambda function is used, which performs a square operation.

That's all for the **Map, Filter, and Lambda in Python**. I hope now you have a better understanding of the **Map, Filter, and Lambda in Python**.

Congratulations! You successfully learned the **Map, Filter, and Lambda in Python**.

In the next chapter, we will start learning **Decorators in Python**.

Chapter 18- Decorators in Python

What is a Decorator in Python?

A Decorator extends the behavior of the function. A decorator allows implementing new features to an existing function without modifying the function itself. You can call the decorator before the function in which you wanna decorate.

For example-

```
@class_method  
def commons(cls):
```

Here, `@class_method` is a decorator. It extends the behavior of the function and it describes the function behavior.

How to Create a Decorator in Python?

A Decorator is itself a function. You can create your own decorator, so for creating own decorator, let's see in the example below-

Before defining any decorator, use the “@” operator.

Syntax of Decorator-

```
@decorator_name  
  
def function_name  
  
    #function body
```

Call decorator before any function, in which you wanna use a decorator.

Example-

```
def cough_dec(func):  
    def func_wrapper():  
        print("cough")          #code before function  
        func()  
        print("cough")          #code after function  
    return func_wrapper  
  
@cough_dec  
def question():  
    print("can you give discount?")  
  
  
@cough_dec  
def answer():  
    print("it's only 50")  
  
question()  
answer()
```

OUTPUT -

cough

can you give discount?

cough

cough

it's only 50

cough

Here, “**def func_wrapper()**” is a wrapper function, which is defining the decorator. “**return func_wrapper**” is returning a wrapper function. “**@cough_dec**” is a decorator.

To create your own decorator, you can create it as a function like in the example “**def cough_dec(func)**“. And inside that function, create a wrapper function like “**def func_wrapper()**“. Where you can define the behavior of a function.

After that, outside from wrapper function “**def func_wrapper()**“, return it as “**return func_wrapper**“. So, by doing this you can create your own decorator and then you can use this decorator many times, whenever you want.

Use of Decorator in Python–

Decorators are used in –

- Logging,
- Run time check,

- Synchronization,
- Type checking,
- Debugging,
- In web framework and many more.

That's all for the **Decorator in Python**. I hope now you have a better understanding of the **Decorator in Python**.

Congratulations! You successfully learned the **Decorator in Python**.

In the next chapter, we will start learning **Reading & Writing files in Python**.

Chapter 19- File Handling in Python

In this chapter, you will learn following-

1. How to Create or Open a Text File?
2. Close File in Python.
3. Modes of File in Python.
4. Writing Files in Python.
5. Append Data to a File.
6. Reading File in Python.

How to Create or Open a Text File?

In Python, you can create a .text file by using the **open()** method. Inside the **open()** method you need to pass the file name along with the path of the file.

Let's see in the example-

```
data_file=open('files/data.txt')
```

Here, we are storing the “**data.txt**” file in the “**data_file**” reference variable. “**files/data.txt**” is the path of the file.

Close File in Python.

You can close any file by using the `close()` function in Python. You can use a `close()` function when you don't have any need for that file. A `close()` function leaves the memory space taken by that file.

Let's see in the example, how to close any file-

```
data_file=open('files/data.txt')  
  
data_file.close()
```

Here, we have closed the “**data_file**”, which we opened.

Modes of File in Python–

There are different modes available in Python to perform Read and Write operations. We will discuss all modes here-

Modes	Definition
“r”	You can use it to open a file for reading text.
“w”	You can use it for writing something in the text file. If the file doesn't exist, so it can create a new file.
“a”	If you wanna add something in your file, then use this append “a” to append something in a file.

"x"	It is used to create a new file, if the file exists already, then it fails.
"r+"	It means read and write . It opens a file to read and write in the file.
"w+"	It means write and read . It opens the file for writing and reading.
"a+"	It is append and read . If the file doesn't exist, then it will create a new file. It writes data at the end of the existing data.

Writing Files in Python–

In order to write something in a file, first, you need to create a blank text file. Then inside this **.py** python file just write this lines of code–

```
with open('files/write.txt', 'w') as write_file:  
    write_file.write('Hey I am John, Python is awesome')
```

After writing this code, these strings are written in the **write.txt** file.

Here, '**w**' is a mode to write something in the file. We have created a "**write.txt**" file and inside that file, we have written the text.

Append Data to a File-

Append means adding something in the previous text. If you wanna add something in your text file then you can do it by append mode.

If you write again this code in a file-

```
with open('files/write.txt','w') as write_file:  
    write_file.write('Hey I am John, Python is awesome')
```

So it will override the first text, so to avoid it, you need to use append mode as “a”, so it will write the text at the end of the first text.

Let's see in the example-

```
with open('files/write.txt','a') as write_file:  
    write_file.write('\n I love it so much, I dream python')
```

Here, this text is written after the first text like that-

OUTPUT-

Hey I am John, Python is awesome.

I love it so much, I dream python.

To write text from a new line use '\n'.

If you wanna write something from any list, then you need to write it like that-

```
quotes=[
    '\n Hello! its John'.
    '\n I love Python'.
    '\n It is very easy'.
]

with open('files/write.txt','a') as write_file:
    write_file.writelines(quotes)
```

Here, “**writelines**” is a function to print lines. And “**quotes**” is the name of a list.

Reading File in Python-

You can read a text file in Python inside another file. Here, we will discuss some methods to read a file in python.

So to read any file, first, you need to open the file. So to open a file you need to write this code-

```
data_file=open('files/data.txt')
```

The first method to read the file is using for Loop-

Method 1-

By using for loop you can read a text file in python. It will print the whole file. Let's see how to do it-

```
data_file=open('files/data.txt')  
for line in data_file:  
    print(line.rstrip())
```

Here, “rstrip()” is a function to avoid gaps in the lines.

Method 2-

Here, the cursor locates at the start, and this method read the lines and store it into the list.

Let's see in the example-

```
data_file=open('files/data.txt')  
data_file.seek(0)           # it will locate the cursor at the start  
lines=data_file.readlines()  
print(lines)
```

Here, “seek(0)” locates the cursor at the start and “readlines()” read the line and store it into a list.

Method 3-

If you wanna read something fixed in a file, which means you wanna read-only from 50th character to 100th character. Then you can do it by this method.

Let's see in the example-

```
data_file=open('files/data.txt')  
data_file.seek(50)  
file_text=data_file.read(100)  
print(file_text)  
data_file.close()    # to close the file.
```

Here, “**seek(50)**” start reading from the 50th character. And “**data_file.read(100)**” stop reading at 100th character.

Method 4-

If you don't wanna close the file after reading, then you can use this method. Here, as you come out from the indentation block, it will automatically close the file.

Let's see in the example-

```
def sequence_filter(line):  
    return '>' not in line  
  
with open('files/dna.txt') as dna_file:  
    lines= dna_file.readlines()  
    print(list(filter(sequence_filter,lines)))
```

Here, “**dna_file**” is a variable name, where we store our file. This method closes the file automatically.

That’s all for the **File Handling in Python 3**. I hope now you have a better understanding of the **File Handling in Python 3**.

Congratulations! You successfully learned **File Handling in Python 3**.

**And Here we go, You Successfully Completed
This Tutorial**

CONGRATULATIONS TO YOU!

For more interesting articles, keep reading my blogs@
www.mltut.com



All The Best