



SILVER OAK UNIVERSITY

EDUCATION TO INNOVATION



Silver Oak Group of Institute
Silver Oak College of Computer Application

Gajendrasingh Ranawat

Python

Python, we all know this is a buzzword nowadays. Is it a breed of snake? going to learn about any reptile? What is it? Is it a software? Or is it a hardware or a programming language?



Are we

2.1.1 Introduction of Python

Python is a simple, clear, high-level, interpretive, general-purpose, and modular programming language.

Python became the most popular programming language all over the world since 2019.

Python is an open-source programming language, so anyone can contribute to its development.



Python was introduced in the late 1980s and its implementation was started in December 1989 by Dutch programmer Guido van Rossum at CWI (Centrum Wiskunde & Informatica) Research lab in the Netherlands.

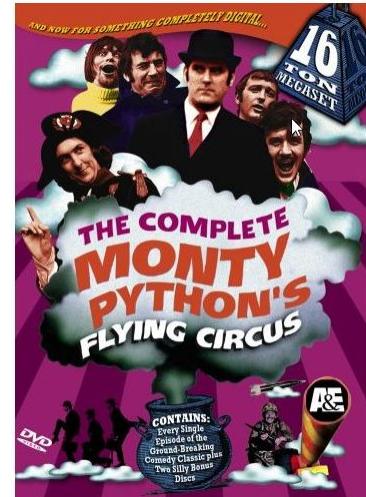


Why he gave his programming language the name “Python”?

In the 1970s, there was a popular BBC comedy TV show called "Monty Python's Flying Circus" and Guido Van Rossum happened to be a fan of that show.

At the time when he began implementing Python, Guido Rossum was also reading the published scripts from "Monty Python's Flying Circus." but the name is too high then he decided on the short, unique, and slightly mysterious name, so finally he gave the name "Python."

Logo of Python



big
van
and

2.1.2 Versions of Python

The main motto behind developing a new programming language is, programmers can express concepts in fewer lines of code with a more user-friendly, simple scripting language and computer programming for everybody.

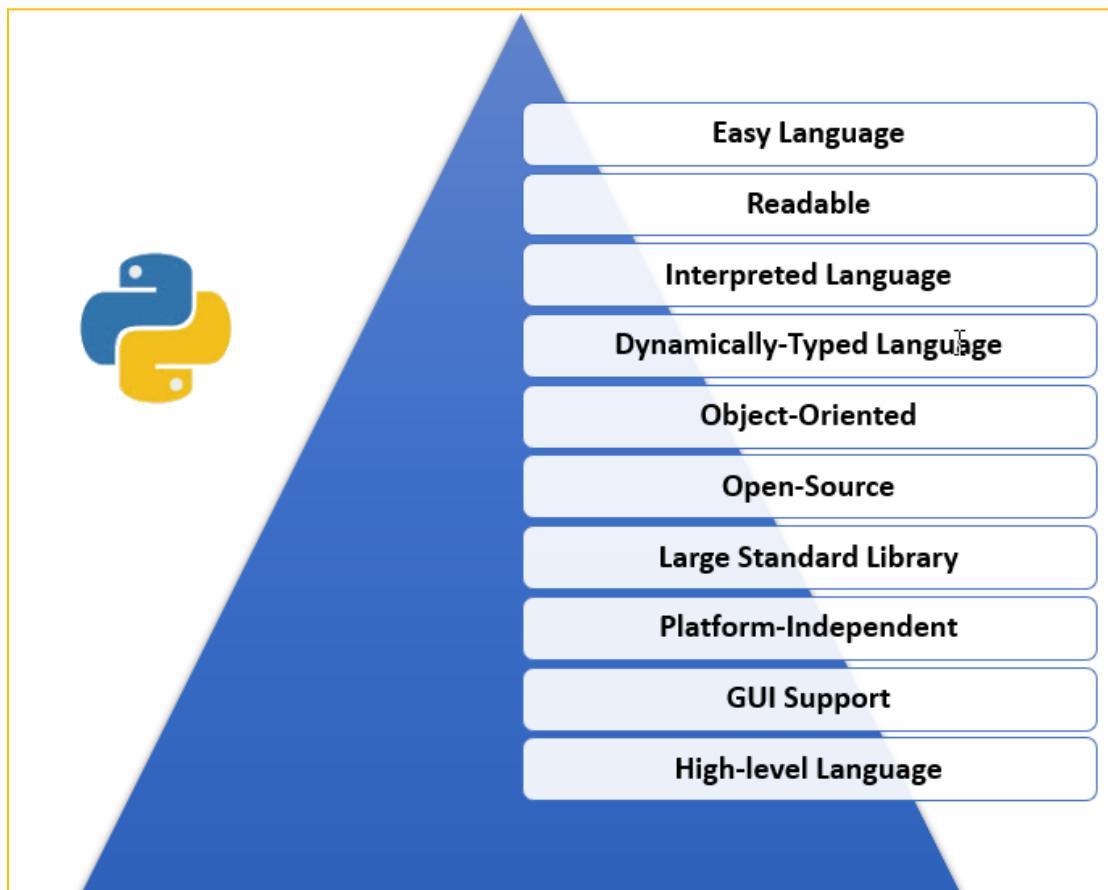
There are more than 40 different versions of Python, major releases are as given below:



Version	Release Date
Python 1.4	25 October 1996.
Python 2.0	16 October 2000.
Python 2.1	15 April 2001.
Python 2.2	21 December 2001
Python 2.3	29 July 2003.
Python 2.4	30 November 2004.
Python 2.5	19 September 2006.
Python 2.6	1 October 2008.
Python 2.7	4 July 2010.
Python 3.0	3 December 2008.
Python 3.1	27 June 2009.
Python 3.2	20 February 2011.
Python 3.3.0	29 September 2012.
Python 3.4.0	16 March 2014.
Python 3.5.0	13 September 2015.
Python 3.6.0	23 December 2016.
Python 3.7.0	27 June 2018.
Python 3.8.0	14 October 2019.
Python 3.9.0	5 October 2020.

2.1.3 Features of Python

Python is everywhere in IT. According to one survey, the most popular language of 2020 and 2021 is Python. Features of Python are always added according to its version. Let's see its features



1. Easy Language

Python has a simple syntax.

It is also easy to read, write, learn and understand.



For example, # If we want to print Hello World just write `print("Hello World")`.

2. Readable

The Python language is like reading an English sentence.



3. Interpreted Language

Python is an interpreted language. Python code is executed line by line at a time.

The source code of python is converted into an immediate form called  bytecode.

0000
0010

4. Dynamically-Typed Language

There is no need to declare data type while defining a variable. The interpreter determines this at runtime based on the types of the parts of the expression.

Example#

```
rollno=5  
name="ram"
```



5. Object-Oriented

Everything in Python is an object. Python strongly supports OOPs(Object Oriented Programming) concepts.



6. Open-Source

Python is open-source. It is free and its source code is freely available to the public.



We can download it from its official website www.python.org.

7. Large Standard Library

For any programming language, the inbuilt library is the most important function.



There are approx. 1,37,000.

Even if we want to install more packages we can install using, pip command.

8. Platform-Independent

Python is platform-independent.

If we write a program, it will run on different platforms like Windows, Mac, and Linux.



There is no need to write programs separately for each platform.

9. GUI Support

We can use Python to create GUI (Graphical User Interfaces).



We can use Tkinter, PyQt, wxPython, or Pyside for this.

10. High-level Language

Python is a high-level language.

When we write programs in python, we do not need to remember the architecture, nor do we need to manage the memory.



system

11. Automatic Garbage Collection

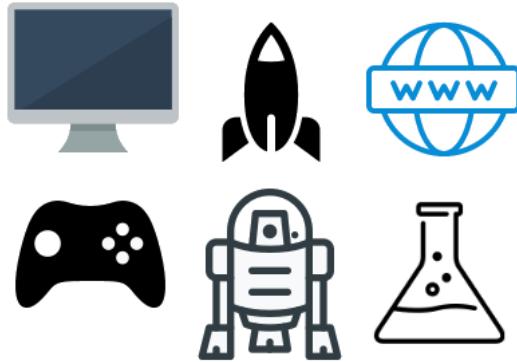
Python initiates automatic garbage collection for memory and performance management. Due to Garbage collection memory can be utilized to its maximum, and application becomes more robust.



2.1.4 Application of Python

As python has reached several libraries to develop any kind of software, we can use Python programming language in every field of Computer Science.

1. Desktop Applications
2. Web Applications
3. Mobile Apps
4. Scientific and Numeric Computing
5. ERP - Business Applications
6. Software Testing
7. Data Science
8. Artificial Intelligence
9. Machine Learning
10. Robotics
11. Internet of Things (IoT)
12. Gaming
13. Data Analysis and Preprocessing



2.15 Popular tools or Framework of Python are:

Web Development

Django, Flask, Bottle, Tornado, Pyramid, web2py, CherryPy, CubicWeb, Dash

GUI Development

Tkinter, Libavg, PyGObject, PySimpleGUI, PyQt, PySide, Kivy, wxPython, PyForms

Scientific and Numeric

SciPy, Numpy, Pandas, IPython, TensorFlow, Seaborn, Matplotlib

2.1.6 What is an Interpreter?

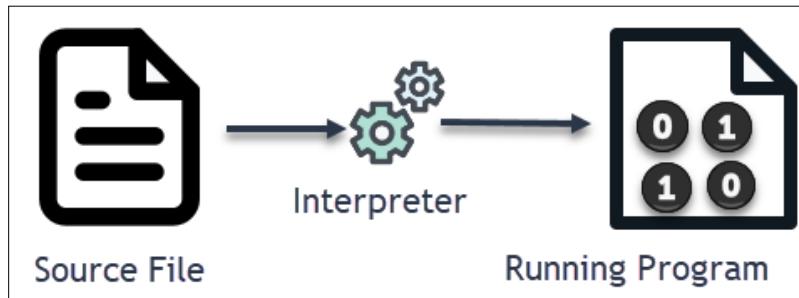
Python uses an Interpreter. An interpreter is a simple software that executes the source code line by line.

An interpreter is a program/software that converts a programming language into Machine language that a computer can understand and execute. While interpreting a code, an interpreter will report any error it finds in the code. If it does not find any error it will translate the code into Machine language.



Python is an interpreted language, which means the source code of a Python program is converted into bytecode that is then executed by the Python virtual machine.

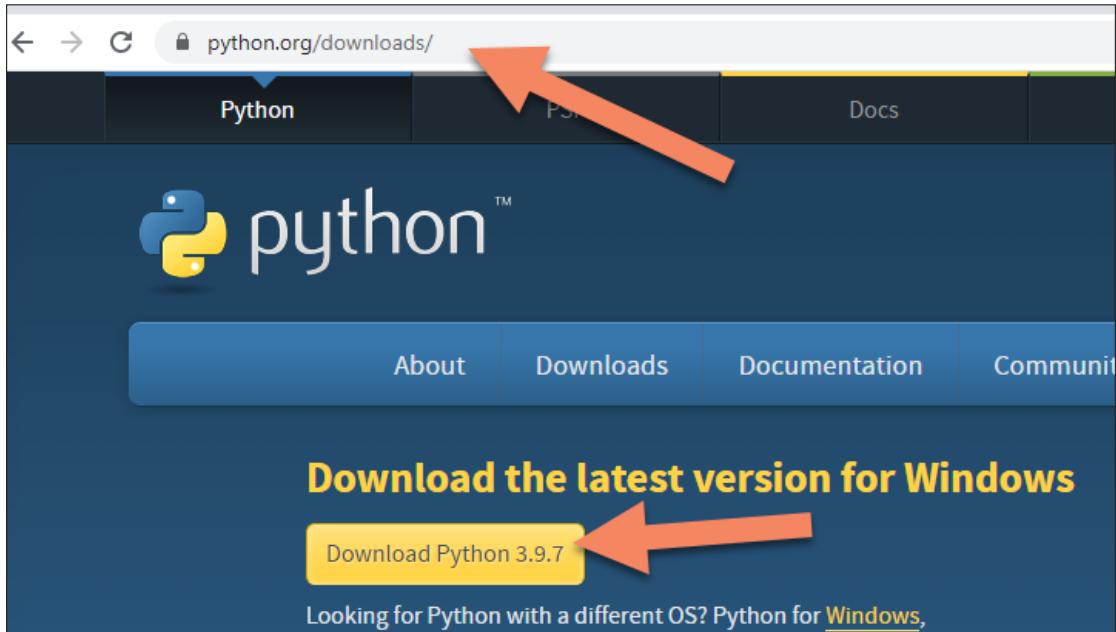
The famous Python interpreter name is CPython. It is written in C Language.



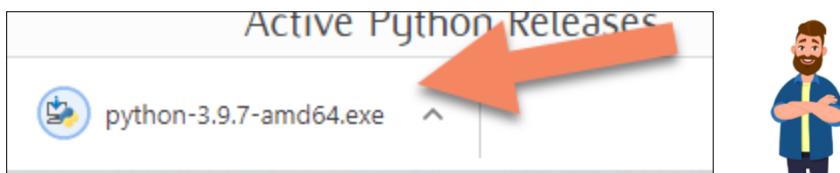
2.2.1 Python Installation

Let's download Python and it's Editor IDLE

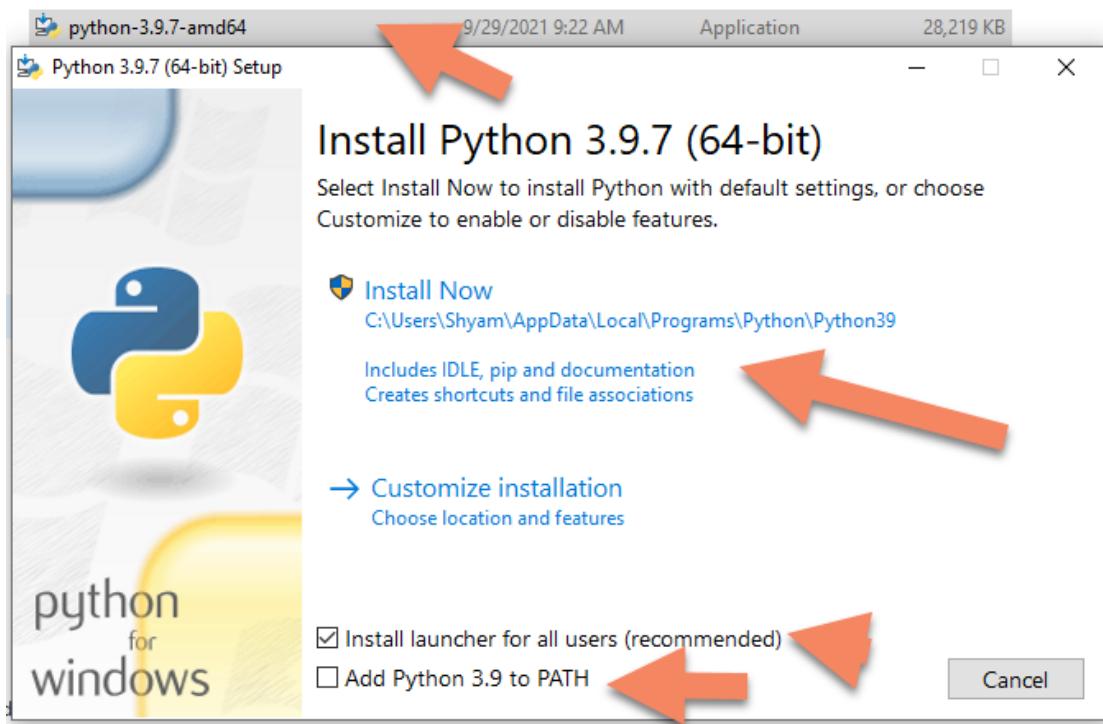
Step 1: Open <https://www.python.org/downloads/> and click on Download



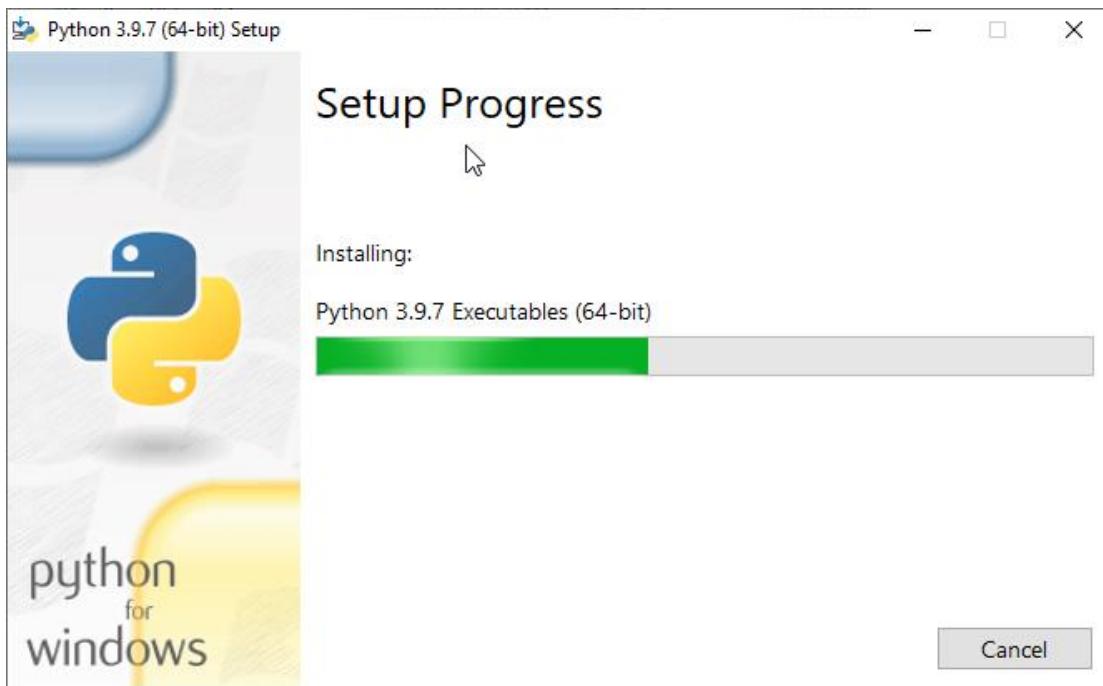
Step 2: After Downloading Install



Step 3: Install



Step 4: Just wait for a minute



Step 5: Final Step , Close it

Setup was successful

New to Python? Start with the [online tutorial](#) and [documentation](#). At your terminal, type "py" to launch Python, or search for Python in your Start menu.

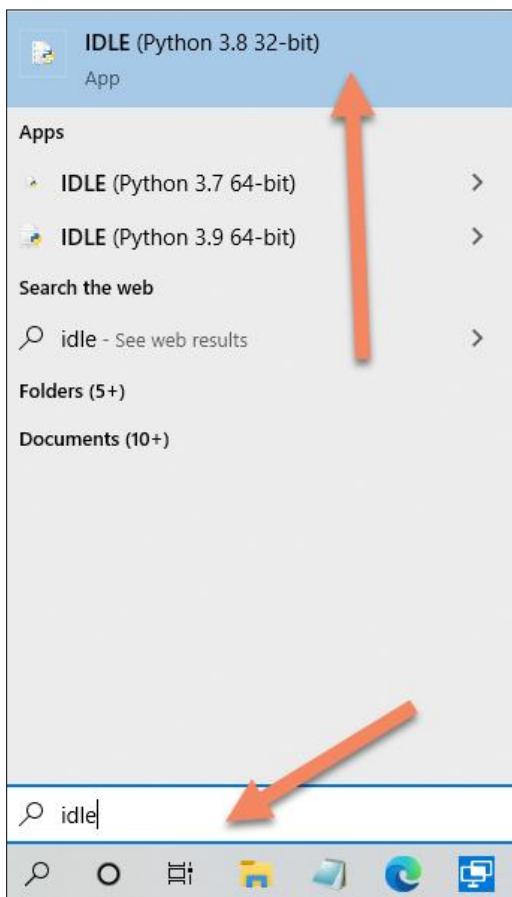
See [what's new](#) in this release, or find more info about [using Python on Windows](#).

Disable path length limit

Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

Close

Now Let's Open



First Screen of IDLE editor of Python, the full form of IDLE is Integrated Development and Learning Environment. IDLE is a simple editor for Python programming. The image of IDLE is given below where we can write and execute the Python scripts.



Python 3.8.5 Shell

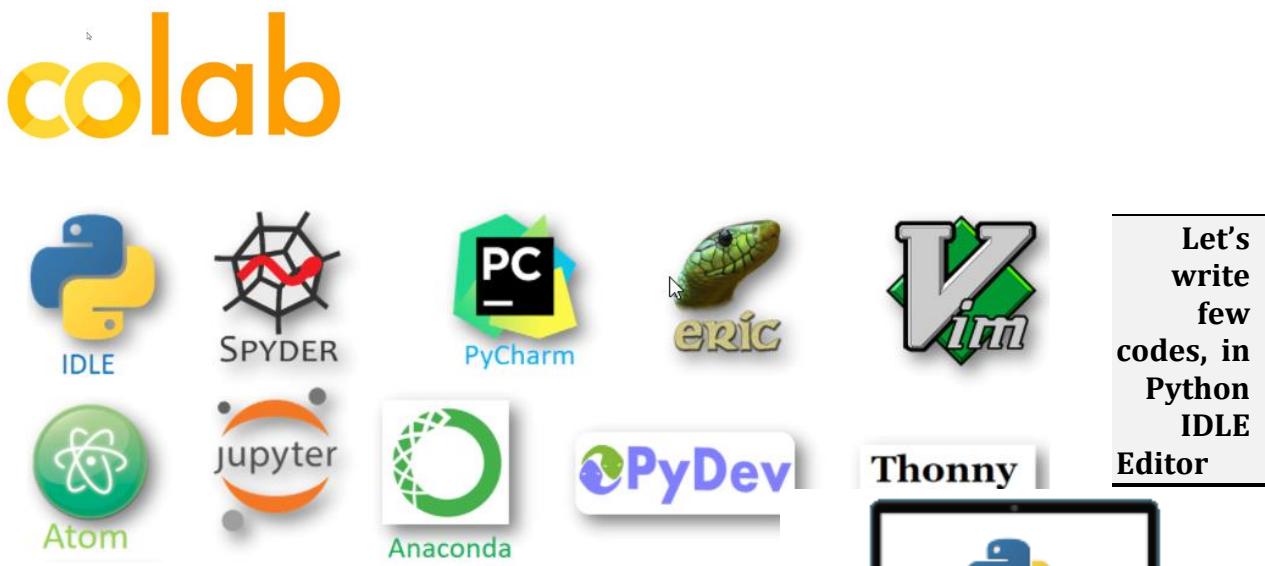
File Edit Shell Debug Options Window Help

Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020)

Type "help", "copyright", "credits" or "license"

>>> |

Other Popular Editors of Python



We can use any editor to write Python Programming Language.

In this book we are using most easiest Editor, IDLE.

Let's write,



To print anything in python we use, print() function

print("") or print('') or print(expression)

```
printf("any text")
```

Syntax#

To print any text

```
print(" any text ")  
Text  print(' any text ')
```

Example#

```
print("Hello")
```

```
print('Hello')
```

There is no difference in between
" "double quotes or Single '' quotes

To print any expression

```
print(expression)
```

```
print(22+10)
```

To print text and expression together

```
print(text,expression)  print("Add = ",22+10)
```

We can use the "," comma separator to separate the "Text", expression as used above

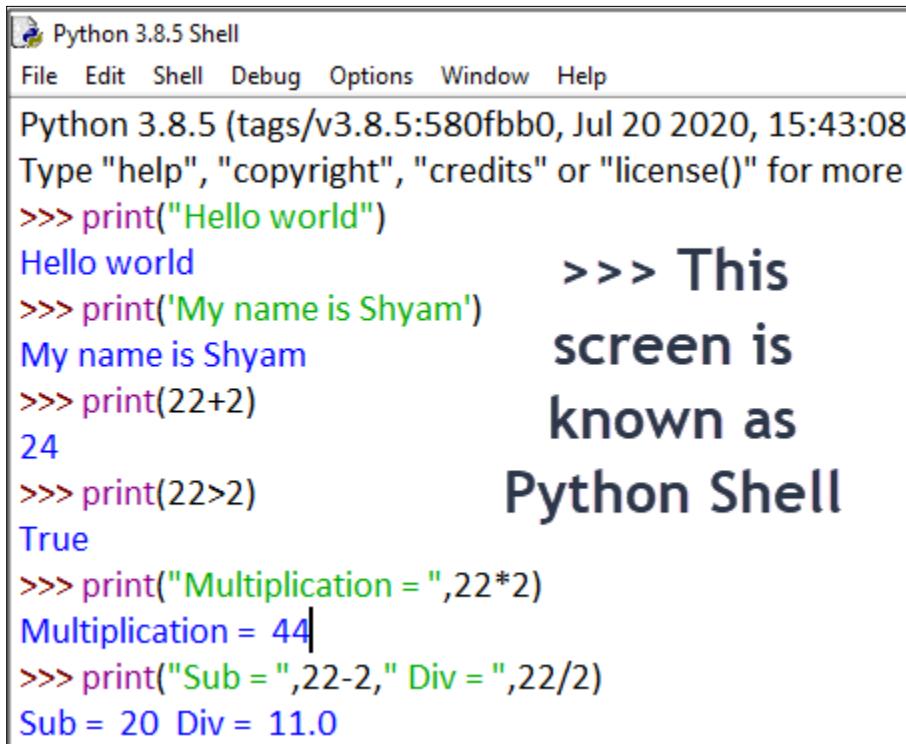
Example# In Python IDLE, Let's print Hello World, We can execute Python statements same as in Python Shell as shown below.

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15  
Type "help", "copyright", "credits" or "license()" for  
>>> print("Hello world")  ←  
Hello world
```

Press Enter




Example# Examples of print() function with different expression



Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08)
Type "help", "copyright", "credits" or "license()" for more

```
>>> print("Hello world")
Hello world
>>> print('My name is Shyam')
My name is Shyam
>>> print(22+2)
24
>>> print(22>2)
True
>>> print("Multiplication =",22*2)
Multiplication = 44
>>> print("Sub =",22-2," Div =",22/2)
Sub = 20 Div = 11.0
```

>>> This
screen is
known as
Python Shell

When we close Python Shell, our code will be deleted, so let's save our program.

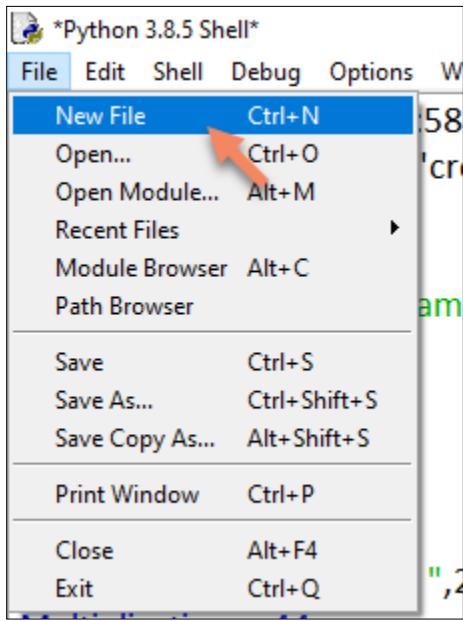
For example, Let's create our first program, Type address, and store our codes in different files. Why do we need to create different files? It's for our convince, each program is stored in a separate file so that it allows the user to access that file with ease.

For instance, we save many phone numbers of different people in our phone's memory so that we can contact the person whenever we want just by searching the name on our phone.



The above image represents 4 different files for 4 different types of programs.

Step1: Create New File in IDLE, To execute a Python script, create a new file by selecting File -> New File from the menu.



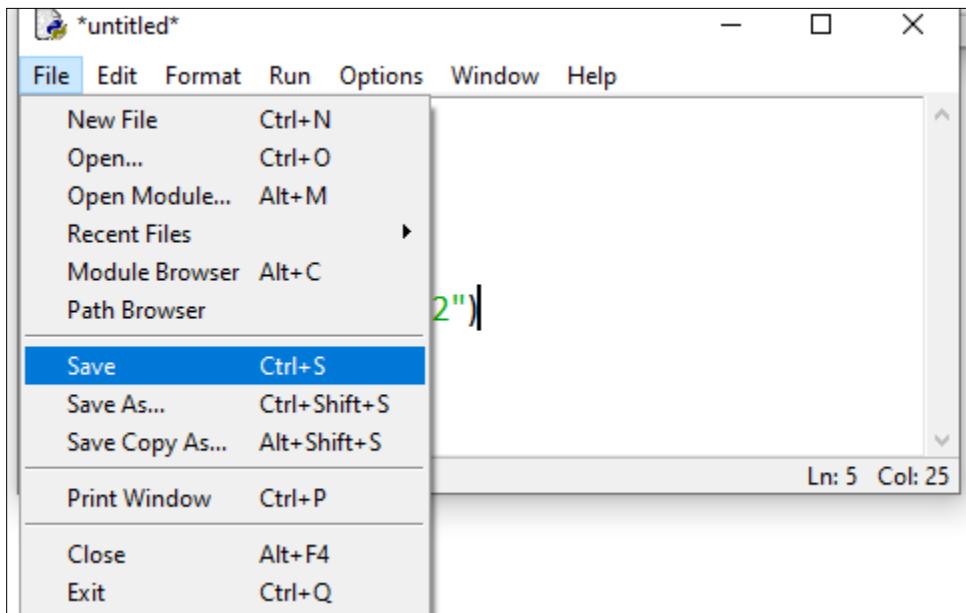
Step 2: Write down the code in New File

A screenshot of the Python 3.8.5 Shell. A new file named 'untitled*' is open in the editor. The code inside the file is:

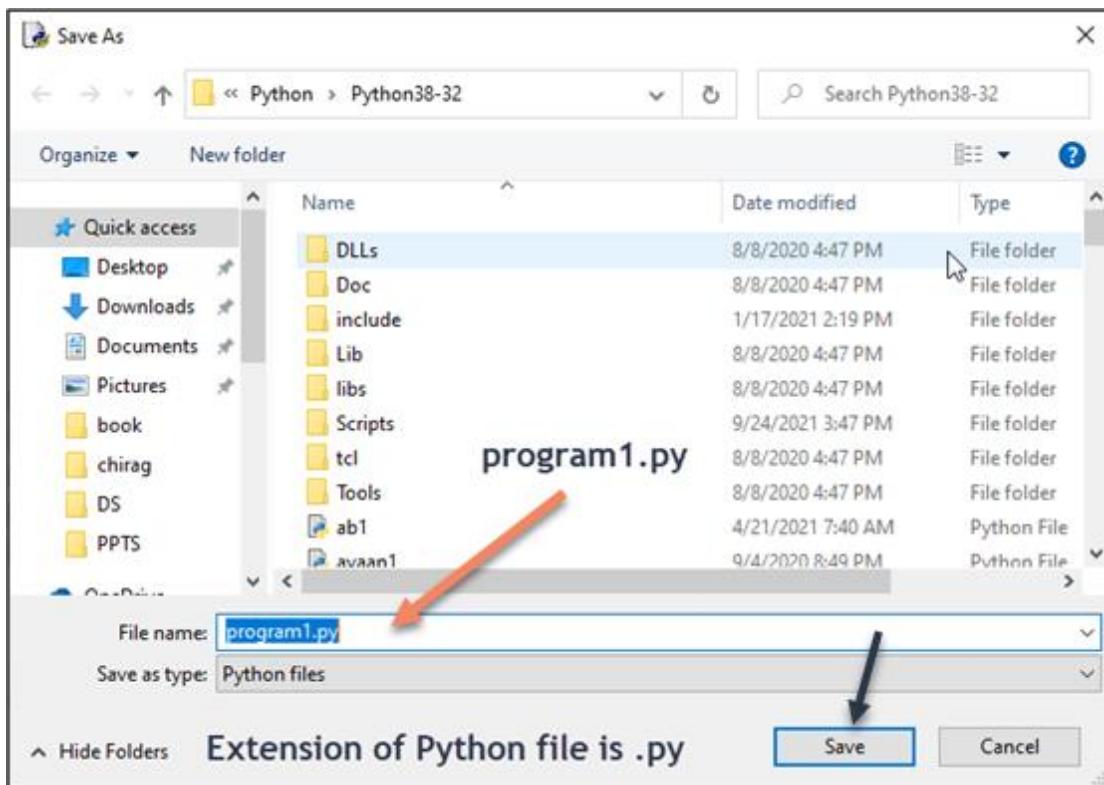
```
print("Address")
print("3 Raju Complex")
print("Gurukul Road")
print("Memnagar")
print("Ahmedabad-380052")
```

The code is written in purple, and the output from the interpreter is shown on the left side of the shell window.

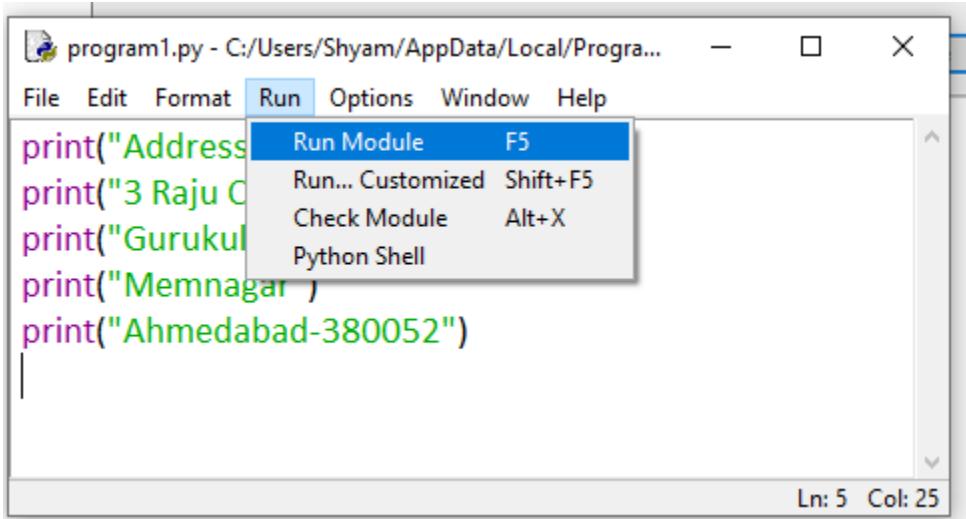
Step 3: Save the file with extension .py using File -> Save.



Step 4: Give it the proper name along with its extension (.py) and Save it.



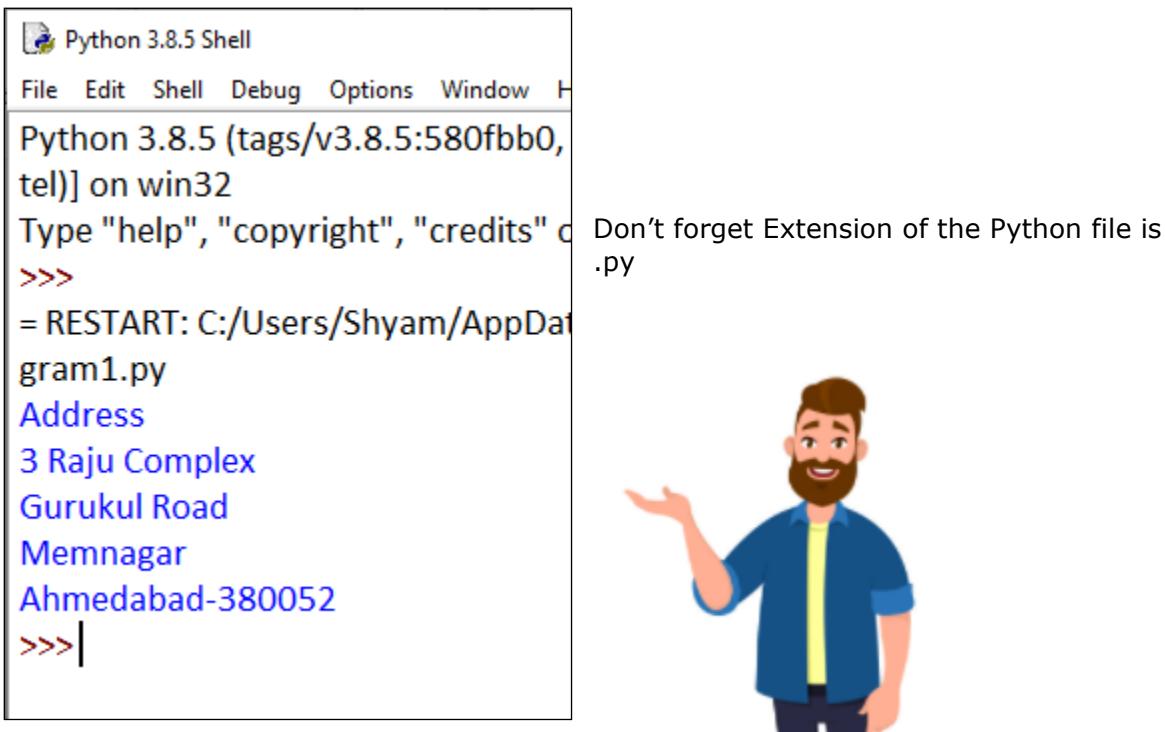
Step 5: Run your program, using the Run menu or shortcut F5



```
program1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38/idle.pyw
File Edit Format Run Options Window Help
print("Address")
print("3 Raju C")
print("Gurukul")
print("Memnagar")
print("Ahmedabad-380052")
```

Ln: 5 Col: 25

Step 6: Check the output, The IDLE shell will show the output.



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, tel) on win32
Type "help", "copyright", "credits" or "license" for more information
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38/idle.pyw
gram1.py
Address
3 Raju Complex
Gurukul Road
Memnagar
Ahmedabad-380052
>>>|
```

Don't forget Extension of the Python file is .py



2.3.1 Basic structure of Python program

Every programming language has a basic structure to write a particular program. Python has 6 different sections.

1. Documentation section

The documentation section consists of a set of comment lines giving the name of the program, the author, and other details, which the programmer would like to use later.

2. imports

The imports section provides instructions to the interpreter to link functions from the module such as using the import statement.

3. Global variables section

Some variables are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions.

4. Class definitions

Here, we can declare different classes. Classes are a collection of data members and data functions.

5. User-defined functions

This section also declares all the user-defined functions.

6. Executable part

There is at least one statement in the executable part.



Example# Basic Structure

"""\Basic Program : Developed by Shyam""""

```
import math # Adding math module

x=22 # Global variable

class student: # Class
    def __init__(self):
        self.no=5
        self.name="Shyam"

def add(): #User defined function
    a=20
    b=2
    print("Add = ",a+b)

print("Let's start Python Programming") #Executable Part
add()
s1=student()
```

2.3.2 Keywords

Keywords are the specific words of the Python language. In Python, they have a special meaning. Keywords are also reserved words. For example, if a keyword is used to make decision. Keywords are known to the interpreter. We cannot change the meaning of the keywords.

Since keywords are referred to as names for interpreters, they can't be used as the variable name.



In called the

they

There are 35 keywords in Python Language.

Following are examples of Python keywords:

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

The above keywords may differ as per versions of Python. Sometimes they add, delete or modify them in the newer versions.

If we want to check keywords of the current version of Python, just type the below code in the IDLE editor:

```
>>> import keyword  
>>> print(keyword.kwlist)  
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',  
'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',  
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',  
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while',  
'with', 'yield']
```



Use the `help()` command to know more about each keyword. The following will display information on the "while" keyword.

```
>>> help("while") ←
```

The "while" statement

The "while" statement is used for repeated execution as long as an expression is true:

2.3.3 Identifiers

An identifier is a name given to an entity. An identifier is a user-defined name to represent the basic building blocks of Python.

Identifiers are used to give a name to the variables, functions, class, etc. The name of the identifier should be meaningful.

Following are the rules when declaring identifiers.

- Identifier contains letters (a-z), digits (0-9) and underscores characters (_) as a character sets.

Example# a12_b is valid, but a12?b is not valid.

- Reserved words are not allowed as an identifier.

Example# while=55 is not valid

- The first character must be a letter (A-Z or a-z) or underscore (_).

Example# _Rollno7 is valid, but 7Rollno is invalid.

- Blank space or white space character is not allowed.

Example# a b=55 is not valid

- An identifier can be of any length.

Example# abcdefgh=123

- Python language is case-sensitive. Thus, names TOTAL and total are considered as different identifiers.

Example# Total=22, total=33, TOTAL=44 both are different identifiers.



Example#

Valid

_abc
A1, a1
Abc_12

**Invalid**

1hour
-sn
if



2.3.4 Variables

Variables are used to store data. A variable is a container (storage area) to hold data. We store data in computer memory through the variables.

We can think of a variable like a bucket and values are like water in a bucket which can easily change.



strings, lists, etc. It is similar to a box or a container.

A Variable is an entity where the user can store digits,

Example# Let's create two variables `a` and `name`, store some values, when we create two variables in memory two boxes are created and these boxes have memory address and its value.



2.3.4.1 How does the variable store the value?

The Data is normally stored in a computer's memory which is organized as cells.

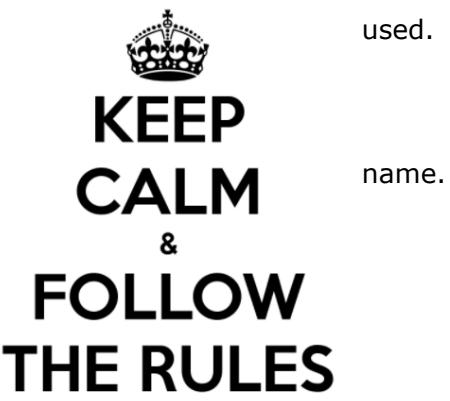
These cells are called the location or address that stores the data. Variables are used to give a name to these locations.

2.3.4.2 Variable names

A name given to the memory location is known as a variable name. Some rules must be followed before giving a variable name in C. They are mentioned below.

- In Python Programming, a Variable name must start with an alphabet or underscore (_) and the following characters can be the letter, underscores, and digits only.

- Both uppercase and lowercase letters can be used.
- Spaces are not allowed in the variable name.
- Certain keywords (these are Python language reserved words) cannot be used in a variable
- The variable name can be any length.
- The variable names should be meaningful.



2.3.4.3 Declaration of the variables

Python has no command for declaring a variable like another programming language.

There is only one rule to declare a variable, When we need, just declare it and assign a value to it.

Syntax#

VariableName = Value

Example#

rollno = 51

rs_of_apple=100.30

name="Shyam"

city='Srinagar'

complex1=35+9j

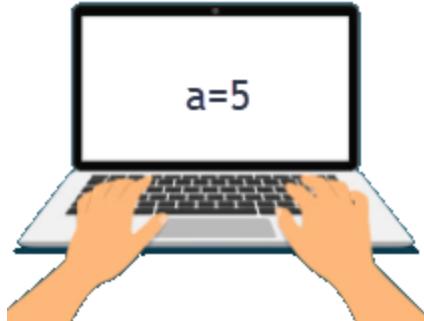
movie=True

list1=[11,22,33,44,55]

set1={11,22,33,44,55}

tuple1=(11,22,33,44,55)

dictionary1={11:"Vedika",12:"Karna",13:"Yash",14:"Pankti"}



Python is **Dynamically-Typed**, the interpreter decides runtime, the type of any variable by variable's value. Even one variable can hold different types of objects.

There are many ways to assign variables.

1. Declare a single variable and assign a value

```
>>> fruitname="Apple"  
>>> price=25  
>>> print("Fruit name is ",fruitname,"and it's Price is ",price)  
Fruit name is Apple and it's Price is 25
```

2. Assigning Multiple Variables

We can assign different values to multiple variables in a statement.

```
>>> marks_english,marks_hindi,marks_ss=45,45,47  
>>> print("English marks = ",marks_english)  
English marks = 45  
>>> print("Hindi marks = ",marks_hindi)  
Hindi marks = 45  
>>> print("SS marks =",marks_ss)  
SS marks = 47
```

3. Assigning Single Value

We can also assign a single value to many variables.

```
>>> a=b=20  
>>> print(a)  
20  
>>> print(b)  
20
```

3. Reassigning the Variables

We can hold different types of values at any time.

```
>>> a=10  
>>> a=55.60  
>>> a="Mac Book"  
>>> print(a)  
Mac Book
```

4. Deleting Variables

We can delete a variable in Python using the `del` keyword. It's used to save memory.

```
>>> a=55  
>>> b=20  
>>> print(a,b)  
55 20  
>>> del a  
>>> print(a,b)  
Traceback (most recent call last):  
  File "<pyshell#20>", line 1, in <module>  
    print(a,b)  
NameError: name 'a' is not defined
```



5. Take value from the user, `input()` function

We can take value from the user and store it into a variable using the `input()` function. Whatever we entered in the `input` function is considered as a Text or say string.

Syntax#

```
VariableName=input("Any Text")
```

Example#

```
Fruitname=input("Enter your favorite fruit name")
```

Example# Take the value of name and age from the user

Type "help", "copyright", "credits" or "license" for more information
=====
RESTART: C:/Users/Shyam/AppData/Local/Temp/m23.py
Enter name =>Shyam
Enter age =>39
Your name = Shyam Age = 39
=====
output

The code window shows the script content:

```
name=input("Enter name =>")  
age=input("Enter age =>")  
print("Your name = ",name," Age = ",age)
```

The status bar at the bottom right indicates "Ln: 3 Col: 0".

Example# Take any number from the user and display its square

Python 2.0.5 Shell
File
Python (Int)
Type
=====
2/m23.py ===
Enter no =>5
Traceback (most recent call last):
 File "C:/Users/Shyam/AppData/Local/Programs/Python/Python", line 2, in <module>
 print("Square = ",a*a)
TypeError: can't multiply sequence by non-int of type 'str'

The code window shows the script content:

```
a=input("Enter no =>")  
print("Square = ",a*a)
```

Two red X marks are present: one over the line "a=input..." and another over the line "TypeError: can't multiply sequence by non-int of type 'str'".

In the above code, we entered numeric value, and in the next line we multiply it so the issue is that the default value of any value is considered as a string, so here the value of a is "5" not 5, so we can't multiply it.

So solution is use, casting function int(),float() for specific data type.

Syntax#

```
VariableInteger=int(input("Any Text"))
```

```
VariableFloat=float(input("Any Text"))
```

Example#

```
Rollno=int(input("Enter your rollno =>"))
```

```
Fruitprice=int(input("Enter fruitprice =>"))
```



Example# Take 2 values from the user and display + - * /

```
>>>
== RESTART: C:/User
Enter no1 =>100
Enter no2 =>20
Add = 120
Sub = 80
Mul = 2000
Add = 5.0
>>>
                                output                                code
a=int(input("Enter no1 =>"))
b=int(input("Enter no2 =>"))

print("Add =",a+b)
print("Sub =",a-b)
print("Mul =",a*b)
print("Add =",a/b)

Ln: 5  Col: 19
```

2.4 type() function

Using the type() function we can check the data type of the variables.

In Python, each variable is treated as an object, and data type is referred to as a class. We can use the type() function to get the class name of an object, in other words, we can say the type() function returns the type of data type of a particular variable.

Example#

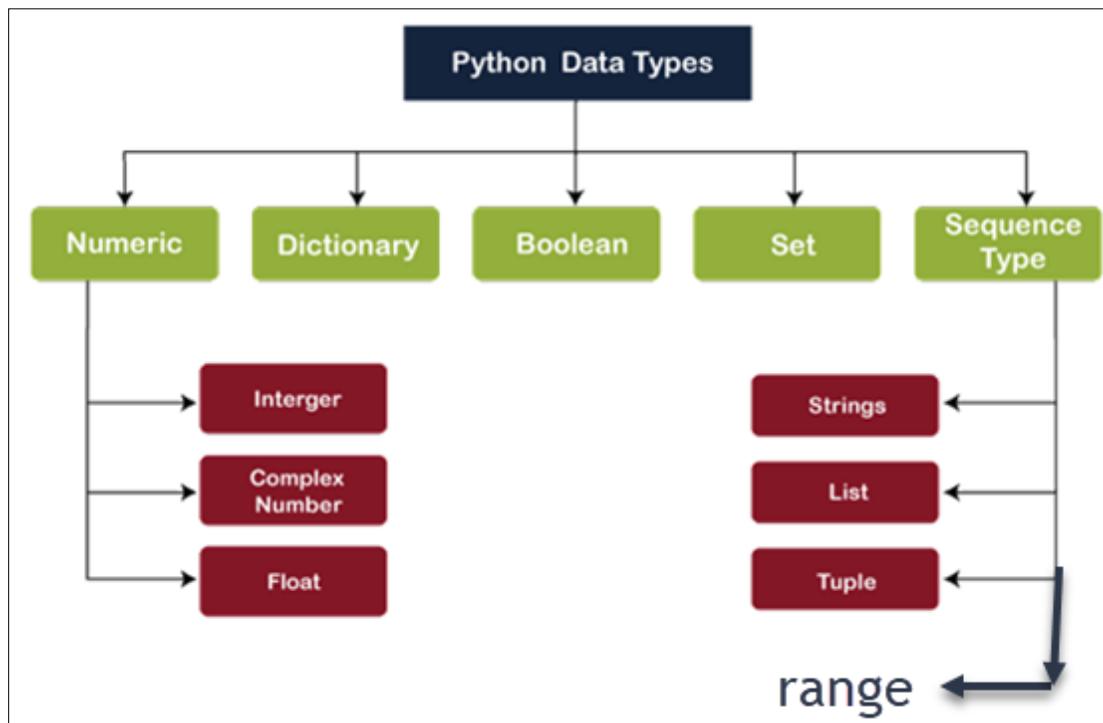
```
>>> type(50)
<class 'int'>
```

The type of 50 is int. In other words, an object of int class contains an integer literal 50.

2.5 Data types in Python

Data types represent the type of data to be stored in a variable. There is a fixed amount of memory assigned to each data type.

Python has the following data types:



Python supports the following data types.

2.5.1 Numeric

There are three numeric Python data types.

1. int

int data type stores integer, it can be of any length.

Example# a=20 , b=300, c=99999999999

2. float

float data Type stores floating-point real values. The decimal limit of float data type is up to 15 decimal points in Python.

Example# a=55.66, b=888888888.99 , c=10.3000

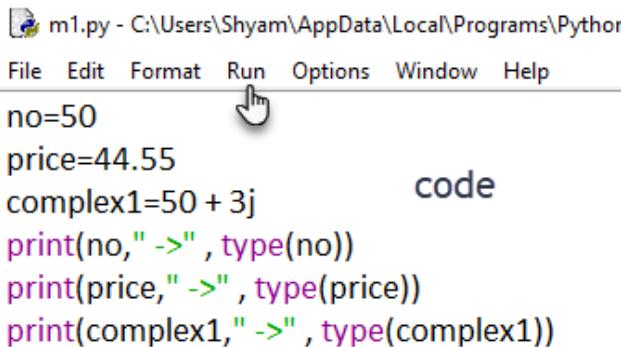
3. complex

complex data Type stores a complex number. A complex number has two parts, real and imaginary.

- 1 int
- 2 float
- 3 complex

Example# a=33 + 5j

Example# All numeric data types

>>> ==== RESTART: C:\Users\Shyam\A 50 -> <class 'int'> 44.55 -> <class 'float'> (50+3j) -> <class 'complex'> >>> output	
---	---

2.5.2 Sequence

There are three sequence Python data types.

1. string

A sequence of one or more characters. We can store string values using single, double, or triple quotation marks.

Example#

```
name="shyam"  
school='St Xavier'  
address="  
3 Raju Complex  
Opp Sterling Hopsital"
```

- 1 string
- 2 list
- 3 tuple
- 4 range

2. list

When we want to store multiple values in a single variable that time we can use List.

We can store unlimited data. In the list we can store values inside square brackets [], separated by commas. Here the position of the list always starts with 0.



Examples#

```
listRollno=[11,23,45,67]
friendsName=["Aalok","Ripal","Pooja","Mayur"]
mixList=[11,"Rahul","Apple",34.55]
```

3. tuple

When a user wants to store multiple values in a single variable, that time we can use Tuple, but values of tuple can't be modified after it is created. The list is mutable while the tuple is immutable.

We can store unlimited data. To store values inside tuple we can use rounded brackets (), separated by commas.

Examples#

```
tupleRollno=(11,23,45,67)
friendsName=("Aalok","Ripal","Pooja","Mayur")
mixList=(11,"Rahul","Apple",34.55)
```

4. range

range datatype worked as a function, it returns a fixed finite number of values. Usually, we used it for "For...loop". Without loop

Examples#

```
a=range(1,200)
```

Example# All sequence data types

```
name="Shyam"
city='Srinagar'
address="""3 Raju Complex
Opp Sterling Hospital
Near by Gurukul Temple
Memnagar"""
listfriends=["Aalok","Mayur","Pooja","Ripal"]
tuple1=(11,22,33,44,55)
a=range(5)
```

```
b=range(1,20)

c=range(1,20,2)

print(name," ->" , type(name))

print(city," ->" , type(city))

print(address," ->" , type(address))

print(listfriends," ->" , type(listfriends))

print(tuple1," ->" , type(tuple1))

print(a,"-->",type(a))

print(b,"-->",type(b))

print(c,"-->",type(c))
```

Output#

```
Shyam -> <class 'str'>

Srinagar -> <class 'str'>

3 Raju Complex
Opp Sterling Hospital
Near by Gurukul Temple

Memnagar -> <class 'str'>

['Aalok', 'Mayur', 'Pooja', 'Ripal'] -> <class 'list'>

(11, 22, 33, 44, 55) -> <class 'tuple'>

range(0, 5) --> <class 'range'>

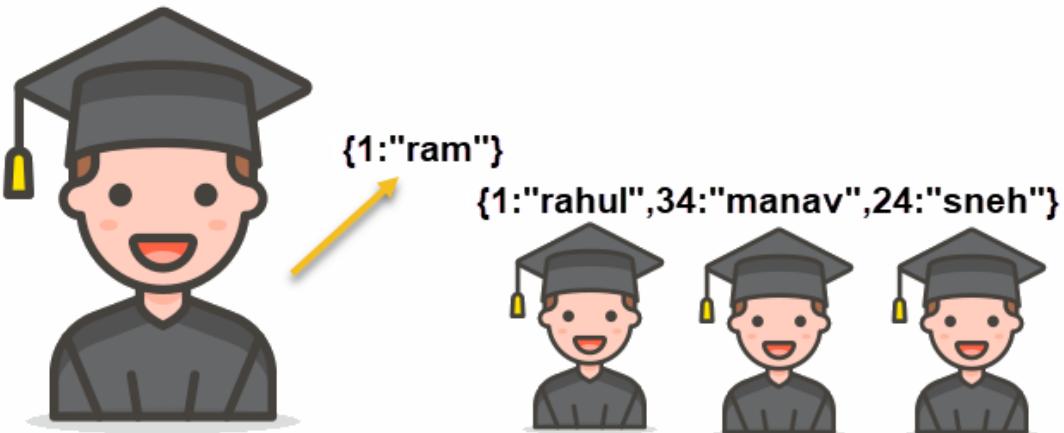
range(1, 20) --> <class 'range'>

range(1, 20, 2) --> <class 'range'>
```

2.5.3 Dictionary

When we want to store data like roll number and name in python it can be stored as a key and value.

The dictionary data type in Python is a set of key-value pairs of items. Dictionary is an unordered collection of data values.



We can store unlimited data. To store values inside the dictionary we can use curly brackets {}, separated by commas. Key and Value are separated by : (colon), In a dictionary, keys can store only primitive data-types (int, float, string, etc), and value can store anything. In a dictionary, all keys consist of unique but values can be repeated.

Example# Dictionary

```
dic1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/dic1... ━ X
File Edit Format Run Options Window Help
dicStudents= {1:"Ram",2:"Shyam",3:"Manav",4:"Mansi",5:"Vedika"}
dicMarks= {"Ram":35,"Shyam":40,"Manav":50,"Mansi":33,"Vedika":39}

print(dicStudents, " -->" , type(dicStudents))
print(dicMarks, " -->" , type(dicMarks))
code
Ln: 6 Col: 0
== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/dic1.py ==
{1: 'Ram', 2: 'Shyam', 3: 'Manav', 4: 'Mansi', 5: 'Vedika'} --> <class 'dict'>          output
{'Ram': 35, 'Shyam': 40, 'Manav': 50, 'Mansi': 33, 'Vedika': 39} --> <class 'dict'>
```

2.5.4 boolean

In Boolean data type, There are just two inbuilt values, True and False.

Example# Boolean Data type

```
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/test/booleantype.py - C:/Users/Shyam/AppD
True --> <class 'bool'>
False --> <class 'bool'>
>>> output
```

File Edit Format Run Options Window

ans1=True
ans2=False

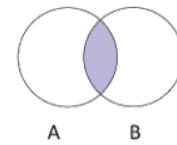
code

```
print(ans1," --> ",type(ans1))
print(ans2," --> ",type(ans2))
```

2.5.5 set

A set is an unordered collection of items. Unordered means there is no surety about the sequence of the elements in the set while printing the set in output. Every set element is unique and must be immutable. The set is a Python implementation of the set Theory in Mathematics.

We can perform mathematical set operations like union, intersection, difference, minus, subset, superset, etc.



We can store values inside the set using curly brackets {}, separated by commas.

We can create an Empty set using the set() function.

Example# Set Data type

```
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/test/setdemo.py - C:/Users/Shyam/AppD
set() --> <class 'set'>
{1, 2, 3, 4, 5, 6} --> <class 'set'>
>>> output
```

File Edit Format Run Options Window Help

set1=set()
set2={1,2,3,4,5,6}

code

```
print(set1," --> ",type(set1))
print(set2," --> ",type(set2))
```

2.8 Operators & Expression

Operators are the symbols that provide instructions to perform some mathematical or logical operations. Operators operate on certain data types called operands, and they form a part of the mathematical or logical expressions. More complex expressions use operators.

Example# + - X / and or % > < ==

The expression means whose evaluation yields numeric value. An expression means any statement with operators. An expression contains constant, variable, and operators. In Python, every expression evaluates a value.

Example# a+b-c

2.8.1 Operators

Operators are special symbols that perform some operations on operands and return the result. There are 7 types of operators in Python.

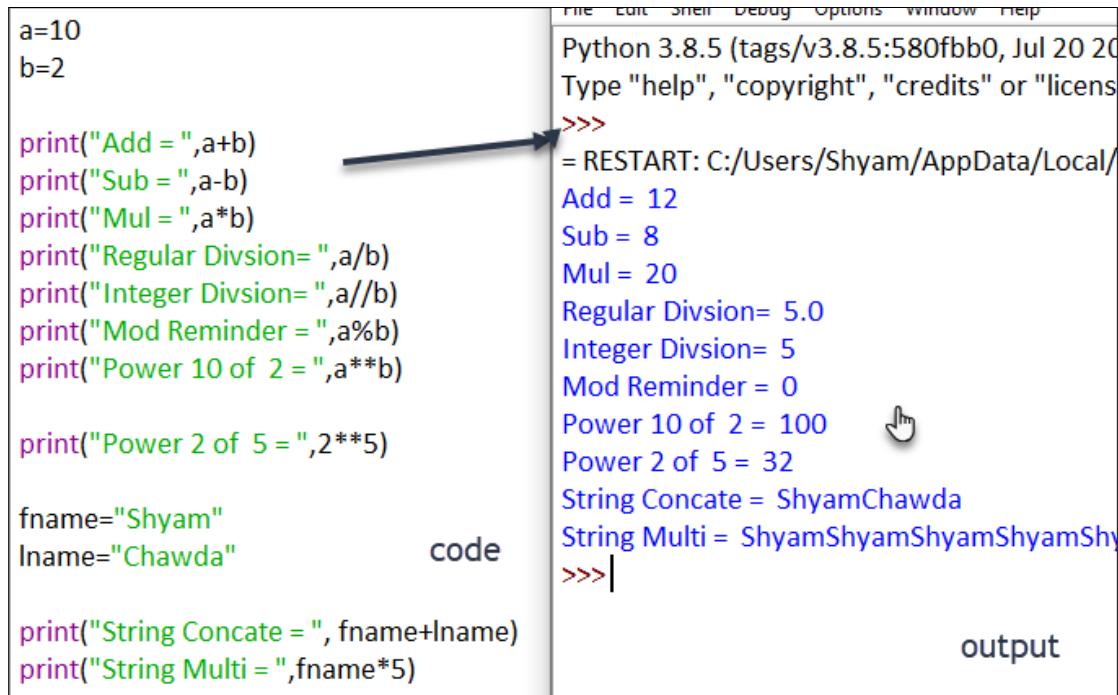


2.8.1.1 Arithmetic operators

These operators are used to perform some basic operations like addition, subtraction, division, multiplication, power, and modulus. These require two operands to perform their operation hence they are also called binary operators. They are also known as mathematical operators. + and * operators can work with string values too.

Operator	Meaning	Example
+	To perform addition operation.	a+b
-	To perform subtraction operation.	a-b
*	To perform multiplication operation.	a*b
/	To perform division operation.	a/b
//	To perform integer division operation. Floor division.	a//b
%	Modulus: To gives a remainder of the division operation.	a%b
**	To perform power operation. Exponentiation	a**b

Example# Arithmetic Operators



The screenshot shows a Python 3.8.5 IDLE window. On the left, there is a code editor with the following Python script:

```

a=10
b=2

print("Add = ",a+b)
print("Sub = ",a-b)
print("Mul = ",a*b)
print("Regular Divsion= ",a/b)
print("Integer Divsion= ",a//b)
print("Mod Reminder = ",a%b)
print("Power 10 of 2 = ",a**b)

print("Power 2 of 5 = ",2**5)

fname="Shyam"
lname="Chawda"

print("String Concate = ", fname+lname)
print("String Multi = ", fname*5)
    
```

The right side of the window shows the output of the script. An arrow points from the line "print("Add = ",a+b)" to the output "Add = 12". A hand cursor is over the output "Power 2 of 5 = 32". The word "code" is written vertically next to the code editor, and "output" is written vertically next to the output window.

2.8.1.2 Assignment operators

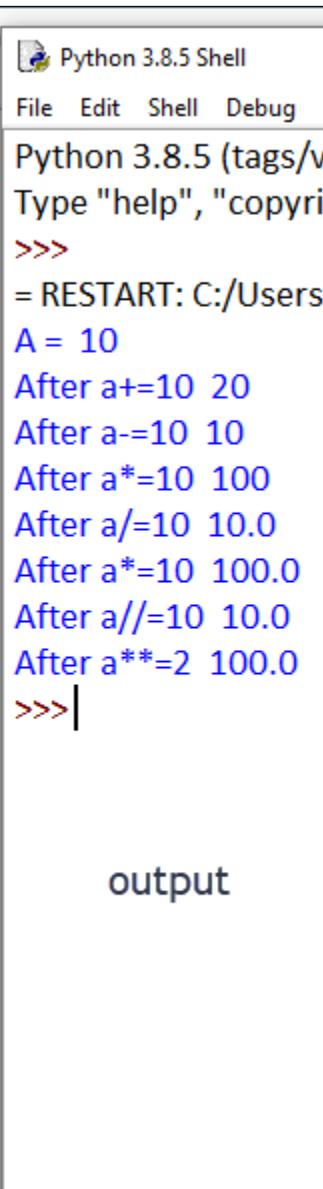
Assignment operators are used for assigning value to any variable. It is denoted by the '=' sign. An assignment operator is a binary operator i.e. it operates on two values. Both a+=5 and a+=5 provides the same answer but " +=" is known as a short end operator.

It has the lowest precedence

Operator	Meaning	Example	Equivalent to
=	Assigns a value to an operand		a=5;

<code>+ =</code>	Add given value to operand value	<code>a+=5;</code>	<code>a=a+5;</code>
<code>- =</code>	Subtract the given value from operand value	<code>a-=5;</code>	<code>a=a-5;</code>
<code>* =</code>	Multiplies operand's value to the given value	<code>a*=5;</code>	<code>a=a*5;</code>
<code>/ =</code>	Divides the operand's value by the given value	<code>a/=5;</code>	<code>a=a/5;</code>
<code>// =</code>	Divides the oprand's value by the given value and return floor(integer) value	<code>a//=5;</code>	<code>a=a/5;</code>
<code>% =</code>	Assign remainder of operator's value by the given value	<code>a%=5;</code>	<code>a=a%5;</code>
<code>** =</code>	Provides the exponential value of the given value as per the operator's value.	<code>a**=5</code>	<code>a=a**5</code>

Example# Assignment Operators



The screenshot shows two windows side-by-side. The left window is a code editor titled 'assignop.py - C:/Users/Shyam/AppData/Local' with the following Python code:

```

a=10
print("A = ",a)

a+=10
print("After a+=10 " , a)

a-=10
print("After a-=10 " , a)

a*=10
print("After a*=10 " , a)

a/=10
print("After a/=10 " , a)

a*=10
print("After a*=10 " , a)

a//=10
print("After a//=10 " , a)

a**=2
print("After a**=2 " , a)

```

The right window is the 'Python 3.8.5 Shell' with the following output:

```

Python 3.8.5 (tags/v3.8.5:5800a8d, Jan 29 2021, 15:37:35) [MSC v.1916 64 bit (AMD64)]
Type "help", "copyright" or "credits" for more information.
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Temp/assignop.py
A = 10
After a+=10 20
After a-=10 10
After a*=10 100
After a/=10 10.0
After a*=10 100.0
After a//=10 10.0
After a**=2 100.0
>>> |

```

The word 'output' is written below the shell window.

2.8.1.2 Relational operators

Relational operators return Boolean values only.

Relational operators are used for comparing two operands and evaluate them as True or False. If the condition is true then it returns True otherwise it returns False.

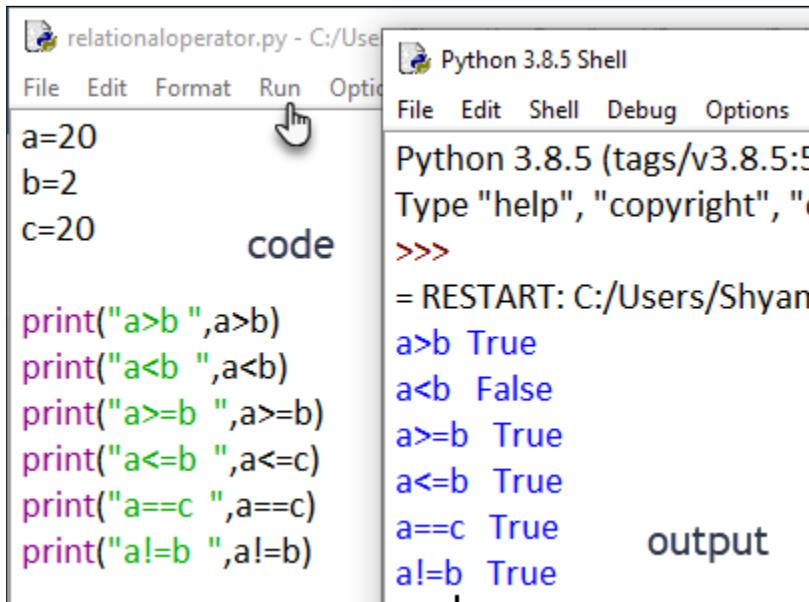
They are used in conjunction with logical operators and conditional & looping statements.

To compare the numeric value we can directly use: `a==5`, for string comparison `a==" ram"`.

Operator	Meaning	Example
<	Less than	<code>a<b</code>
>	Greater than	<code>a>b</code>

>=	Greater than or equal to.	a>=b
<=	Less than or equal to.	a<=b
!=	Not equal to	a!=b
==	Equal to.	a==b

Example# Relational Operators



The screenshot shows a Python IDE interface. On the left, there is a code editor window titled "relationaloperator.py - C:/User". It contains the following Python code:

```

a=20
b=2
c=20
print("a>b ",a>b)
print("a<b ",a<b)
print("a>=b ",a>=b)
print("a<=b ",a<=c)
print("a==c ",a==c)
print("a!=b ",a!=b)
    
```

A hand cursor is pointing at the first line of code. On the right, there is a terminal window titled "Python 3.8.5 Shell" with the following output:

```

Python 3.8.5 (tags/v3.8.5:5800a8d, Jan 29 2021, 15:37:35)
[Clang 12.0.0 (clang-1200.0.32.29)] on darwin
Type "help", "copyright", "c"
>>>
= RESTART: C:/Users/Shyam
a>b True
a<b False
a>=b True
a<=b True
a==c True
a!=b True
    
```

The word "code" is written below the code editor, and the word "output" is written below the terminal window.

2.8.1.3 Logical operators

They are also known as binary operators. Logical operators return a Boolean value. Usually, we use logical operators with comparison operators.

Operator	Meaning	Example
or	To perform logical OR operation.	a==5 or a==7
and	To perform logical AND operation	a>20 and a<40
not	To perform logical NOT operation	a not 5

Example# Logical Operators

 logical.py - C:/Users/Shyam/AppData/Local/Programs File Edit Format Run Options Window Help <pre>a=30 b=40 print("a>20 and b>20 ",a>20 and b>20) print("a>20 or b<20 ",a>20 or b<20) print("not(a<20),not (a<20)) print("not(a>20),not (a>20))</pre> <p style="text-align: center;">code</p>	 Python 3.8.5 Shell File Edit Shell Debug Options Win <pre>>>> = RESTART: C:/Users/Shyam/ a>20 and b>20 True a>20 or b<20 True not(a<20) True not(a>20) False output >>></pre>
---	---

2.8.1.4 Membership operators

Membership operators are used to check whether a value exists in List, Tuple, Set, etc.

Operator	Meaning	Example
in	Returns True if the first value is in second.	22 in [11,22,33,44]
not in	Returns True if the first value is not in second.	22 not in [11,22,33,44]

Example# Membership Operators

 membershipop.py - C:/Users/Shyam/AppData/Local/Programs File Edit Format Run Options Wind <pre>a=22 list1=[11,22,33,44] print(a in list1) print(45 in [55,66,77]) print(45 not in list1)</pre> <p style="text-align: center;">code</p>	 Python 3.8.5 Shell File Edit Shell Debug Options <pre>Python 3.8.5 (tags/v3.8.5:5800, Jul 29 2020, 15:53:45) Type "help", "copyright", "credits" or "license" for more information >>> = RESTART: C:/Users/Shyam/ True False True >>></pre> <p style="text-align: center;">output</p>
--	---

2.8.1.5 Identity Operators

Identity operators compare the memory locations of the two objects.

The identity operators check whether the two objects have the same id.

Operator	Meaning	Example
is	Returns True if the variables on either side of the operator point to the same object	x is y, here it returns 1 if id(x) is equal to id(y).
not is	Returns False if the first value is in second.	x is not y, here it does not return 1 if id(x) is not equal to id(y).

Example# Identity Operators

Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

```
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Temp/identityop.py
51553152 -- 1540664000 -- 1540664000
a is b False
a is c False
b is c True
a not is b True
>>>
```

output

identityop.py - C:/Users/Shyam/A... — X

a = 500
b = 50
c = 50

print(id(a), " -- ", id(b), " -- ", id(c))

print(" a is b ", a is b)
print(" a is c ", a is c)
print(" b is c ", b is c)
print(" a not is b ", a is not b)|

Ln: 10 Col: 32

code

2.8.1.6 Bitwise operators

We have performed various operations using various operators on entire numbers so far. But suppose we want to operate on bits rather than entire byte, Python supports bitwise operator to operate on bits of integral operands.

Bitwise operators work on binary levels.

Syntax#

Operand1 symbol(bitwise) **operand2**

Here operand1, operand2 are the variable or constant.

Operator	Meaning	Example
&	Bitwise AND. If both the bits of both operands are 1 then the result in the bit is 2. If any one of the bits is 0 then the result in bit is 0. (referred AND true table)	a & b

	Bitwise OR. If any one of the bits of both operands is 1 then the result in bit is 2. If both bits are 0 then the result in bit is 0. (referred OR true table)	a b
\sim	Bitwise Complement. Set 1 to 0 and 0 to 1 bit.	$\sim a$
\wedge	Bitwise Exclusive OR. If both bits of both operands are 1 then resulted in bit is 1 otherwise resulted bit is 0.	$a \wedge b$
$>>$	Bitwise shift right. It moves the bit of the first operand to the right by the number of bits specified by the second operand. It discards the far right bit and fill from the right with 0 bits.	$a >> 2$
$<<$	Bitwise shift left. It moves the bit of the first operand to the left by the number of bits specified by the second operand. It discards the far left bit and fills from the right with 0 bits.	$a << 2$

Example# Bitwise Operators

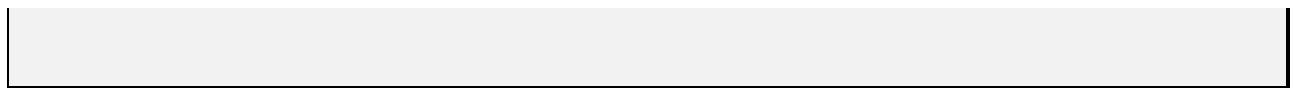
```
x=5
y=10
print("x & y",x & y)
```



```
print("x | y",x | y)
print("x ^ y",x ^ y)
print("~x",~x)
print("x<<2",x<<2)
print("x>>2",x>>2)
```

Output:

```
x & y 0
x | y 15
x ^ y 15
~x -6
x<<2 20
x>>2 1
```



2.8.2 Precedence and Associativity of operator

Precedence

This represents the evaluation of expression starts from the "what" operator.

Operator precedence determines which operator is performed first in an expression with more than one operator with different precedence.

Example# $5 + 10 * 20$ is calculated as $5 + (10 * 20)$ and not as $(5 + 10) * 20$.

Associativity

Associativity is only used when there are two or more operators of the same precedence.

Operators	Meaning
<code>()</code>	Parentheses
<code>**</code>	Exponent
<code>+x, -x, ~x</code>	Unary plus, Unary minus, Bitwise NOT
<code>*, /, //, %</code>	Multiplication, Division, Floor division, Modulus
<code>+, -</code>	Addition, Subtraction
<code><<, >></code>	Bitwise shift operators
<code>&</code>	Bitwise AND
<code>^</code>	Bitwise XOR
<code> </code>	Bitwise OR
<code>==, !=, >, >=, <, <=, is, is not, in, not in, Not, And, Or</code>	Comparisons, Identity, Membership operators Logical NOT Logical AND Logical OR

Example#

$$10 + 4 - 6 \times 10 / 5$$

2
14
2
2
-8
2
2
10
4
6
10
5
10
6
2
4
2
10
5
10
6
2
4
2
14

$$10 - 2 + 5 \times 10$$

58
48
50
10
2
58

$$10 / 5 \times 2 + 10$$

14
4
2
2
10
5
10
2
4
14

2.9 Type Conversion

Converting an expression of a given type into another type is known as type-casting.

Data will be truncated when the higher data type is converted to lower. For example, if the float is converted to int, data that is present after the decimal point will be lost.

There are 7 functions – int(), float(), str(), bool(), set(), list(), tuple()

7

int() - To convert value into integer	float() - To convert value into float
Example# a=int(False) b=int(22.3) c=int("89") d=int(44.89) print("a = ", a) print("b = ", b) print("c = ", c) print("d = ", d) a = 0 b = 22 c = 89 d = 44	Example# a=float(7) b=float(55.66) c=float("99") d=float("55.66") print("a = ",a) print("b = ",b) print("c = ",c) print("d = ",d) a = 7.0 b = 55.66 c = 99.0 d = 55.66
str() - To convert value into string	bool() - To convert value into Boolean
Example# a=str(7) b=str(100.44) c=str(True) print("a = ",a) print("b = ",b) print("c = ",c) a = 7 b = 100.44 c = True	Example# a=bool(22) b=bool(0) c=bool("maurya") d=bool(0.5) print("a = ",a) print("b = ",b) print("c = ",c) print("d = ",d) a = True b = False c = True d = True
set() - To convert value into set	list() - To convert value into list
Example#	Example#

<pre>a=set([1,2,2,3]) b=set({1:22,2:33}) print("a = ",a) print("b = ",b) a = {1, 2, 3} b = {1, 2}</pre>	<pre>a=list((11,22,33)) b=list({1:22,2:33}) c=list({11,22,33,44}) print("a = ",a) print("b = ",b) print("c = ",c) a = [11, 22, 33] b = [1, 2] c = [33, 11, 44, 22]</pre>
---	--

tuple() - To convert value into tuple	
--	--

Example#	
-----------------	--

<pre>a=tuple([11,22,33]) b=tuple({1:22,2:33}) c=tuple({11,22,33,44}) print("a = ",a) print("b = ",b) print("c = ",c) a = (11, 22, 33) b = (1, 2) c = (33, 11, 44, 22)</pre>	
---	--

2.10 Comments

In Python Programming Language, you can place comments in your source code that are not executed as a part of the program.

Comments provide clarity to the Python source code allowing others to understand what the code was intended to accomplish and greatly helping in debugging the code.

Comments are especially important in large projects containing hundreds or thousands of lines of source code or in projects in which many contributors are working on the source code.

There are two types of comments:

Single Line

1

"""

""" Multi Line

2

A comment starts with a slash asterisk # and ends with a """ and can be anywhere in your program.

Comments can span several lines within your Python program. """ comments are also known as docstrings.

Adding source code comments to our Python source code is a highly recommended practice. In general, it is always better to write comments in Python source code.

Single Line Comment	Multi-Line Comment
Syntax# # comment goes here	Syntax# """ comment goes here """
Example# Comment in Single Line # Author: Shyam	Example# """ Date: 9-30-2021 Day: Thursday Place: Ahmedabad """

Example#

File Edit Format Run Options Window Help

#Let's learn Python Programming

```
a=50
```

```
b=20
```

```
"""
```

```
Now add  
sub  
mul  
div  
them """
```

```
print(a+b)
```



2.11 end keyword

By default python's print() function ends with a newline. If we don't want to print in a new line then we can use "end="" " at the end of the print() function (as shown in the fig.).

```
>>> File Edit Format Run Options  
= RESTART:  
HiHello  
>>> print("Hi",end="")  
      print("Hello",end="")  
output | code
```

```
>>> File Edit Format Run Options Window H  
= RESTART: C  
Hi Hello  
>>> print("Hi",end=" ") space  
      print("Hello",end=" ")  
output | code
```



3.1.1 Introduction of Flow Control

We can control the order of execution of the program with the help of a control statement, based on logic and values.

For example : we can control flow of water in a river with the dam .

In the sequential flow of control, statements are executed line by line but is based

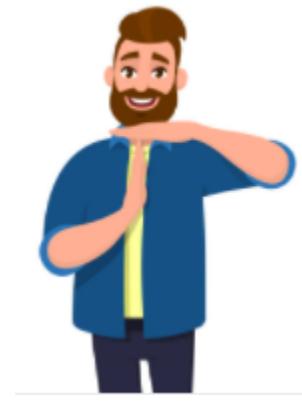


help of a

by line but
is based

Conditional statements causes variable flow of execution of the same program based on certain conditions to be true or false, whenever a program is executed.

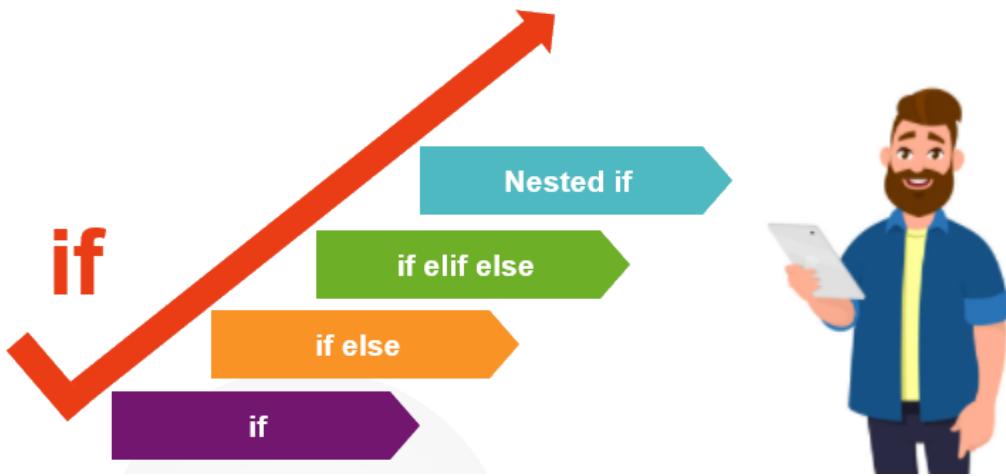
The control statements can be classified into three groups:



3.2.1 Decision-making statement / Selection Statement

The decision-making statement is also known as the Selection statement.

- In some situations, we would like to execute some logic when a condition is TRUE, and another logic when the condition is FALSE and this can be done by using the if-else statement.
- The condition of the result consists of a comparison of two values.
- Condition is an expression that is formed by relational operators ($>$, $<$, $>=$, $<=$, $==$, etc.) and operand and logical operators (and, or) if required, and it returns true or false.
- There are 4 ways of writing "if statements" as shown in the given image.

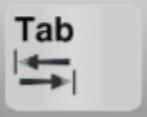


3.2.1.1 If

- ➊ In some situations, we would like to execute a certain logic when a condition is True. An "if statement" checks whether a condition is equal to true or false.
- ➋ The condition usually results from the comparison of two values. Comparison can be greater than, less than, equal to, or for the difference.
- ➌ If the condition is true then the control will execute its logic and if the condition is false, the output will be null.
- ➍ Don't forget to press the TAB key for alignment of the logic (in the syntax# given below) and a colon(:) (in the syntax# given below) must always be there after the "if condition statement".

Syntax#

```
if condition:  
    Logic here
```



Example#

```
a=int(input("Enter how many Rupees you have? =>"))

if a>100:
    print("Let's have a party")
```



If you forget to indent your logic and ":" operator after the "if" condition, then it will generate an error.

```
a=int(input("Enter how many Rupees you have? =>"))

if a>100:
    print("Let's have a party")
```



Example# To check whether the number is 5

File Edit Format Run Options Window Help	File Edit Shell Debug Options V
	Python 3.8.5 (tags/v3.8.5:5897e88, May 18 2020, 16:00:00) [MSC v.1916 64 bit (AMD64)]
a=int(input("Enter any number =>")) if a==5: print("Yes it's 5")	Type "help", "copyright", "credits" or "license" for more information. >>> = RESTART: C:/Users/Shyam/ Enter any number =>5 Yes it's 5 >>>
code	output

3.2.1.2 If else

- In some situations, we would like to execute some logic when a condition is TRUE, and some other logic when the condition is FALSE.
- If the condition is True then the control goes to the Logic between 'if' and If the condition is False then the control goes to the else block.
- We have to press the TAB key to maintain the indentation of the logic and a colon (:) to mark the end of the if-else statement.

Syntax#

if condition:



Logic here

else:



Logic here

Example# To check whether you are eligible to vote or not

```
age=int(input("Enter age =>"))
if age>18:
    print("Eligible for voting")
else:
    print("Not eligible for voting")
```

Example# To check whether a number is positive or negative

<pre>a=int(input("Enter any number =>")) if a>0: print(a,"is positive") else: print(a,"is negative")</pre>	<p>Python 3.8.5 Shell</p> <pre>File Edit Shell Debug Options Window Python 3.8.5 (tags/v3.8.5:580f1cd, May 19 2020, 15:53:45) Type "help", "copyright", "credits" or "license" for more information >>> = RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38/ifPosneg.py Enter any number =>-20 -20 is negative >>></pre>
code	output

Example# To check whether a given character is vowel or not (Using or Condition)

```

ifvowel.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/ifvowel.py (3.8.5)
File Edit Format Run Options Window Help
x=input("Enter any character =>")

if x=="a" or x=="e" or x=="i" or x=="o" or x=="u" or x=="A" or x=="E" or x=="I" or x=="O" or x=="U":
    print("Entered character is vowel")
else:
    print("Entered character is not vowel")
```

code

= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/ifvowel.py =

Enter any character =>a ✓

Entered character is vowel ✓

>>>

= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/ifvowel.py =

Enter any character =>m ✓

Entered character is not vowel ✓

Ln: 7

Example# To check whether a given character is vowel or not (Using in Operator , List)

```

incheck.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/incheck.py (3.8.5)
File Edit Format Run Options Window Help
list1=['a','A','e','E','i','I','o','O','u','U']
value=input("Enter any character =>")

if value in list1:
    print("Entered Character is Vowel")
else:
    print("Entered Character is not Vowel")
```

code

= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/incheck.py =

Enter any character =>a ✓

Entered Character is Vowel ✓

>>>

Enter any character =>m ✓

Entered Character is not Vowel ✓

3.2.1.3 If elif else

- ☒ There are cases when we would like to execute some more conditions when a condition is TRUE, and some other condition logic when the condition is FALSE.
- ☒ We can write one or multiple elif conditions with if statement.
- ☒ Don't forget to press the TAB key and a colon (:) to mark the end of the if-elif-else statement.

Syntax#

Example# To print the name of the month associated with the number entered by user

```

if condition:
    Logic here
elif condition:
    Logic here
elif condition:
    Logic here
elif .....
    Logic here
else:
    Logic here

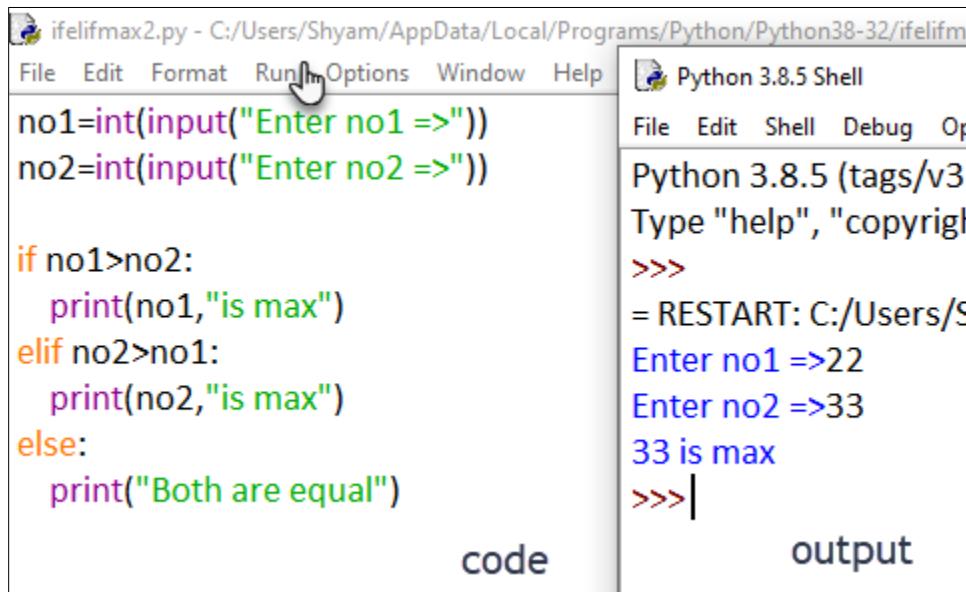
```

```

a=int(input("Enter month number =>"))
if a==1:
    print("January")
elif a==2:
    print("Feb")
elif a==3:
    print("March")
..... upto 12 ... months

```

Example# Enter two values and compare them



The screenshot shows a Python 3.8.5 Shell window. On the left, under 'code', is the script:

```

ifelimax2.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/ifelimax2.py
File Edit Format Run Options Window Help
no1=int(input("Enter no1 =>"))
no2=int(input("Enter no2 =>"))

if no1>no2:
    print(no1,"is max")
elif no2>no1:
    print(no2,"is max")
else:
    print("Both are equal")

```

On the right, under 'output', is the terminal session:

```

Python 3.8.5 Shell
File Edit Shell Debug Op
Python 3.8.5 (tags/v3.8.5:40e94d9, May 26 2021, 16:53:48)
[PyQt5: PySide6] Type "help", "copyright" or "credits" for more information.
>>>
= RESTART: C:/Users/S
Enter no1 =>22
Enter no2 =>33
33 is max
>>> |

```

Example# Take three subject marks, display it's total and grade. If the total is between 0-50 display C grade, 51-100 B grade, >100 display A grade

<pre> ifelife.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38/python.exe ifelife.py File Edit Format Run Options Window Help maths=int(input("Enter marks of maths =>")) english=int(input("Enter marks of english =>")) hindi=int(input("Enter marks of hindi =>")) total=maths+english+hindi print("Total marks = ",total) if total>=0 and total<=50: print("C grade") elif total>50 and total<=100: print("B grade") else: print("A grade") </pre>	code	<pre> Python 3.8.5 Shell File Edit Shell Debug Options Window Help Python 3.8.5 (tags/v3.8.5:580f8d6, May 19 2020, 15:53:45) Type "help", "copyright", "credits" or "license" for more information >>> = RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38/python.exe ifelife.py Enter marks of maths =>20 Enter marks of english =>30 Enter marks of hindi =>40 Total marks = 90 B grade >>> output </pre>
--	-------------	--

3.2.1.4 Nested If

- ☒ An 'if' inside an 'if', is known as a Nested if...else.
- ☒ An if statement that contains another if statement either in its if block or else block is called a "Nested if". In some cases, we would like to execute some more conditions when a condition is TRUE, and some other conditions when a condition is FALSE. In some situations, you have to place an "if statement" inside another statement.
- ☒ This works by cascading several comparisons.
- ☒ As soon as one of these gives a true result, the following statement or block is executed, and no further comparisons are performed.

Syntax#

if condition:

if



Logic here

else:



Logic here

else:



Logic here



Example# Input Gender and Age, to check whether you are eligible to get married or not.

<pre>File Edit Format Run Options Window Help gender=input("Enter Your Gender (M/F) =>") if gender=="m" or gender=="M": age=int(input("Your Age =>")) if age>21: print("You are eligible for Marriage") else: print("You are not eligible for Marriage") elif gender=="f" or gender=="F": age=int(input("Your Age =>")) if age>18: print("You are eligible for Marriage") else: print("You are not eligible for Marriage") else: print("Wrong option")</pre>	<p>Python 3.8.5 Shell</p> <pre>File Edit Shell Debug Options Window Python 3.8.5 (tags/v3.8.5:580fbb0, 2020-09-28 13:47:39 UTC) Type "help", "copyright", "credits" => = RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/ Enter Your Gender (M/F) =>M Your Age =>22 You are eligible for Marriage => = RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/ Enter Your Gender (M/F) =>M Your Age =>15 You are not eligible for Marriage =></pre>
code	output

Example# Input three integer numbers and find the largest amongst them using nested if-else

The screenshot shows a Python 3.8.5 Shell window. On the left, there is a code editor with the file name 'ifnestedif1.py' and the following code:

```
no1=int(input("Enter no1 =>"))
no2=int(input("Enter no2 =>"))
no3=int(input("Enter no3 =>"))

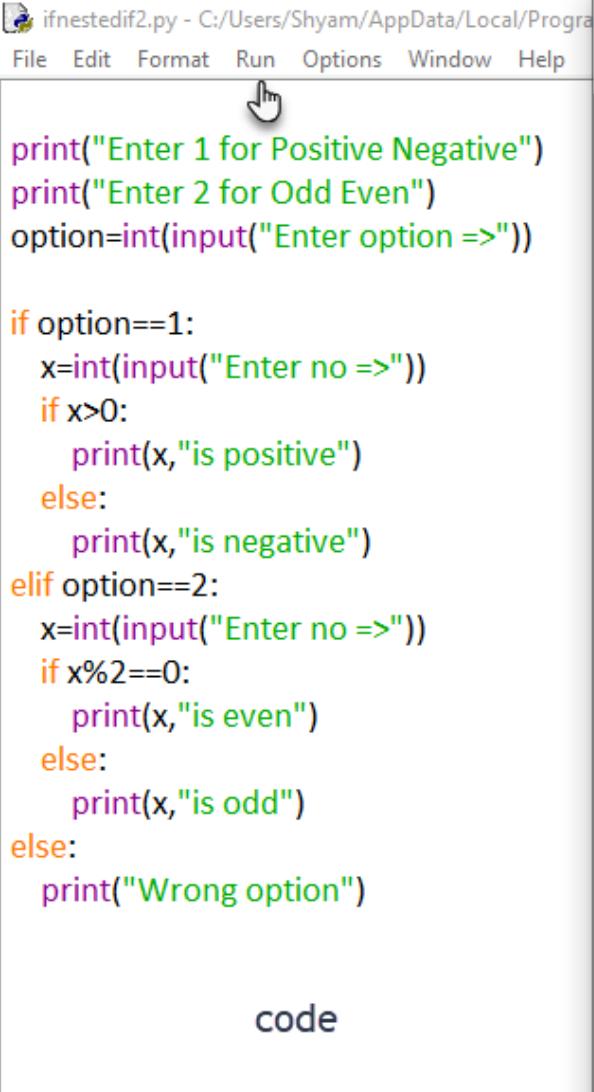
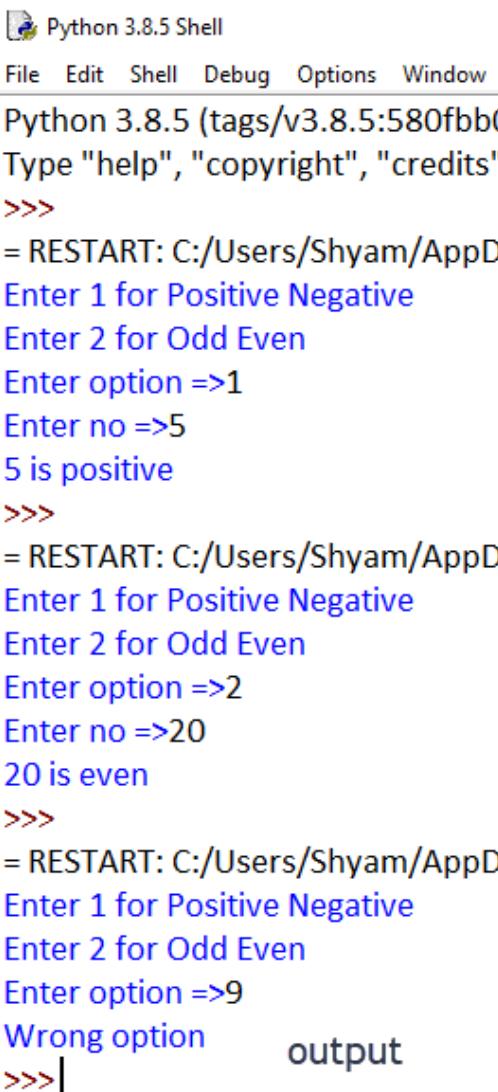
if no1>no2:
    if no1>no3:
        print(no1,"is max")
    else:
        print(no3,"is max")
else:
    if no2>no3:
        print(no2,"is max")
    else:
        print(no3,"is max")
```

The word 'code' is written below the code editor. On the right, the Python shell window shows the execution of the program:

```
>>>
= RESTART: C:/Users/S
Enter no1 =>10
Enter no2 =>33
Enter no3 =>25
33 is max
>>>
```

The word 'output' is written below the shell window.

Example# Create a menu-driven program, If the user presses 1 then input an integer value and find whether the given number is positive or negative if the user presses 2 then input an integer value and find out whether the given number is odd or even.

 <pre> print("Enter 1 for Positive Negative") print("Enter 2 for Odd Even") option=int(input("Enter option =>")) if option==1: x=int(input("Enter no =>")) if x>0: print(x,"is positive") else: print(x,"is negative") elif option==2: x=int(input("Enter no =>")) if x%2==0: print(x,"is even") else: print(x,"is odd") else: print("Wrong option") </pre> <p style="text-align: center;">code</p>	 <pre> Python 3.8.5 (tags/v3.8.5:580fbb0, May 18 2020, 15:53:45) Type "help", "copyright", "credits" or "license" for more information >>> = RESTART: C:/Users/Shyam/AppD Enter 1 for Positive Negative Enter 2 for Odd Even Enter option =>1 Enter no =>5 5 is positive >>> = RESTART: C:/Users/Shyam/AppD Enter 1 for Positive Negative Enter 2 for Odd Even Enter option =>2 Enter no =>20 20 is even >>> = RESTART: C:/Users/Shyam/AppD Enter 1 for Positive Negative Enter 2 for Odd Even Enter option =>9 Wrong option >>> </pre> <p style="text-align: center;">output</p>
---	--

Properties of if..else Statement

- It's not compulsory to write else block.
- We cannot write multiple if or else statements.
- We may or may not have elif block.
- We can have multiple elif conditions in one if-else block.
- Don't forget alignment.
- We cannot write conditions with else. ("else>15" is invalid.)
- Don't forget to write colon(:) after if, elif, else :
- We can use logical and relational operators in a expression. if a>b: , if a>b and a>c:



invalid.)
conditional

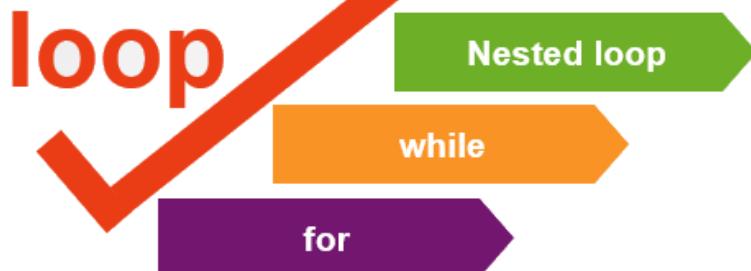
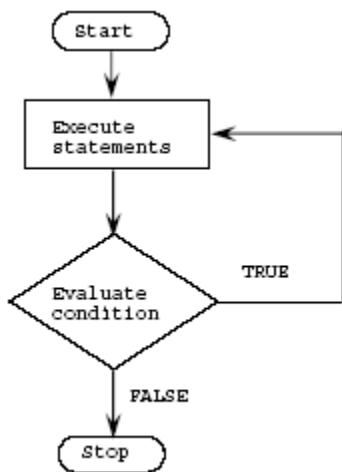
3.3.1 Loop/Repetition Control Structure

Loops are one of the most basic entities in a given programming language.

With loop/repetition statements, we can execute a given piece of a certain condition is fulfilled. When we want to perform the same again and again, then that time we can use the loop concept.

Iteration statements are also known as Looping or Repetition statements. Looping statements are used when a group of statements is to be executed repeatedly, till a condition is True or condition is False.

The following illustration shows a loop structure that runs a set of statements until a condition becomes true.



3.3.1.1 For loop

It is the most common and most popular loop used in any programming language.

A "For loop" is mainly used for

1. Iterate over a collection of items
2. Iterate over a sequence of numbers using the "range()" function.

3.3.1.1.1 For loop for collection

With the "for" loop, we can execute a set of statements, once for each item in a list, tuple, set, etc.

Using the "for" loop we can iterate over a collection/sequence like Tuple, List, Set, Dictionary, String. We can access each element from the given collection/sequence and execute a block of code once for each iteration.

It separates the item from the sequence.

Syntax# For Sequence

for variableName in sequence



Logic here

Example# Print all the values from List

The screenshot shows a Python 3.8.5 terminal window. On the left, under 'output', it says 'Python 3.8.5 (tags/...'. Below that, it says 'Type "help", "copy' and then '>>>'. Under 'RESTART: C:/User...', it lists four names: 'Yash', 'Karna', 'Vedika', and 'Pankti'. On the right, under 'code', there is a code block:

```
for x in ["Yash", "Karna", "Vedika", "Pankti"]:
    print(x)
```

Example# The following code will count all the even values in a list

 Python 3.8.5 Shell File Edit Shell Debug Options Python 3.8.5 (tags/v3.8.5) Type "help", "copyright", "credits" or "license" for more information. >>> = RESTART: C:/Users/Shyam/Desktop/forloop1.py Even values are 20 40 Total even values are 2 >>> output	 forloop1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/forloop1.py File Edit Format Run Options Window Help listnumbers=[11,20,35,40,63] count=0 print("Even values are") for x in listnumbers: if x%2==0: print(x) count=count+1 code print("Total even values are ",count)
---	---

Example# Print each mobile company's name in a Mobile Company list

 Python 3.8.5 (tags/v3.8.5) Type "help", "copyright", "credits" or "license" for more information. >>> = RESTART: C:/Users/Shyam/Desktop/forloop2.py Apple OnePlus Samsung Oppo Vivo MI output	 *forloop2.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/forloop2.py (3.8) File Edit Format Run Options Window Help mobilecomp = ["Apple", "OnePlus", "Samsung", "Oppo", "Vivo", "MI"] for m in mobilecomp: print(m) code
---	---

Example# Iterate Each Character of a given text/string

Python 3.8
Type "help", "copyright" or "credits" for more information.
>>>
= RESTART: C:/Users/k/Desktop/forloop3.py
k
h
d
a
k
a output

```
for x in song:  
    print(x)
```

code

Example# Count total numbers of K in a given String

Python 3.8.5 (tag)
Type "help", "copyright" or "credits" for more information.
>>>
= RESTART: C:/Users/k/Desktop/forloop3.py
Total k are = 5
=> output

```
song="khdak sing ke khdak ne se"  
count=0  
for x in song:  
    if x=="k":  
        count=count+1  
print("Total k are =",count)
```

code

3.3.1.1.2 For loop with range() function for several times

When using "for" loops in Python, the range() function is used to specify the number of times the loop is executed.

Syntax# with Range

`for variableName in range(starting, ending, step)`



Logic here

`range([start], stop,[step])`

datatype/function has three arguments.

- The first argument

is the starting value. It's an optional, default value or it is 0.

- The second argument is the ending value of the range.
- The third argument is the number of steps (Increment/Decrement) to take after each iteration. It's optional, the default value or it is 1. We can have a positive or negative step.

Example# range(1,20,1) , range(30) , range(22,30,1) , range(100,10,-10)

Example# for loop demo

The image shows two side-by-side Python code editors. The left editor has the following code:

```
= RESTART: C:/Users/Shyam/AppData/Local/Temp/forloop4.py
0
1
2 for i in range(10):
3     print(i)
4
5
6
7
8
9 output
```

The right editor has the following code:

```
= RESTART: C:/Users/Shyam/AppData/Local/Temp/forloop4.py
1
2
3 for i in range(1,10):
4     print(i)
5
6
7
8 output
9
```

A large downward-pointing arrow is positioned between the two editors, indicating a comparison between the two different range function usages.

In the above-left program, the loop starts with 0 and goes up to 9 (10-1), and right program it starts with 1 and goes up to 9 (10-1).

This is a limitation of the range function, it always provides 1 value less than the upper limit, so if we want to get 15 values or run the loop 15 times, we have to set 16 as the upper limit.

Example# Take ending value from the user and display numbers

The image shows a Python code editor with the following code:

```
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:348dfcc, Jul 1 2020, 14:39:37) [MSC v.1916 64 bit (AMD64)]
Type "help", "copyright" for more information.
>>>
= RESTART: C:/Users/SHYAM/PycharmProjects/Python_Lesson/frmloop5.py
Enter end value =>5
1
2
3
4 output
```

The code uses `input()` to prompt the user for an ending value, which is then used as the upper limit for a `range()` loop.

Example# Take ending value from the user and display numbers upto that

```
Python 3.8.5 (tags/v3.8.5:5800a8d, Jan 29 2021, 15:37:35) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright" or "credits" for more information.
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe
Enter end value =>5
1
2
3
4
5
```

code

output

In the above code, I have used range(1,n+1), so it will go up to number 1 to 5

Example# Take starting and ending values from the user and display numbers up to them

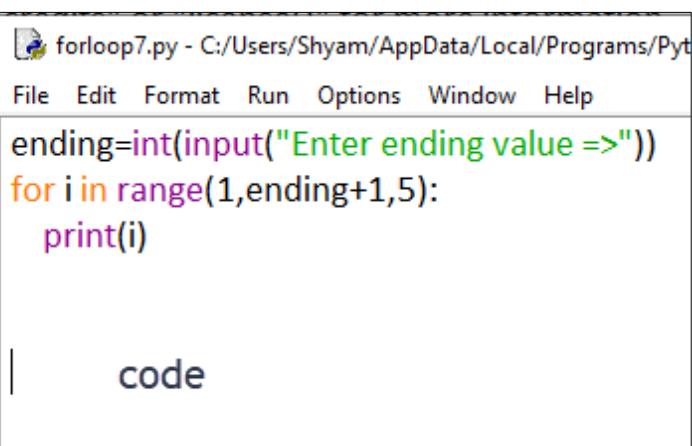
```
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe
Enter starting value =>22
Enter ending value =>27
22
23
24
25
26
27
```

code

output

Example# Take ending value from the user and display numbers up to that in a gap of 5

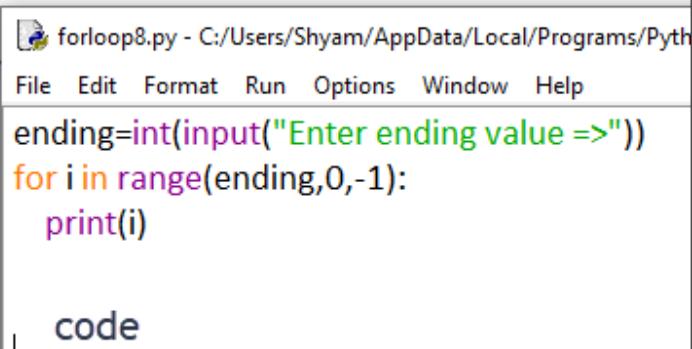
```
type 'help', 'copyright',  
>>>  
= RESTART: C:/Users/Shyam/  
Enter ending value =>40  
1  
6  
11  
16  
21  
26  
31  
36
```



```
forloop7.py - C:/Users/Shyam/AppData/Local/Programs/Python  
File Edit Format Run Options Window Help  
ending=int(input("Enter ending value =>"))  
for i in range(1,ending+1,5):  
    print(i)  
  
code
```

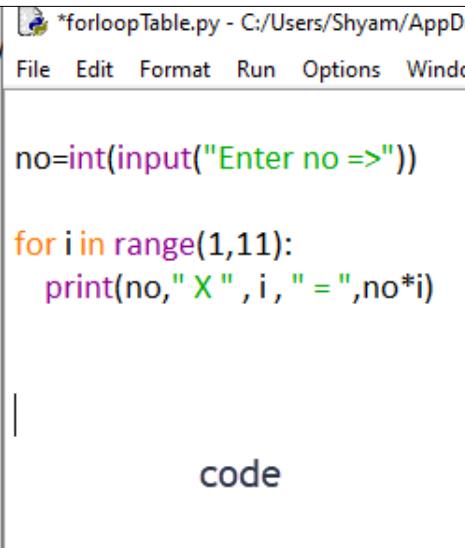
Example# Take ending value from the user and display numbers in reverse order up to that

```
>>>  
= RESTART: C:/Users/Shyam/  
Enter ending value =>5  
5  
4  
3  
2  
1
```

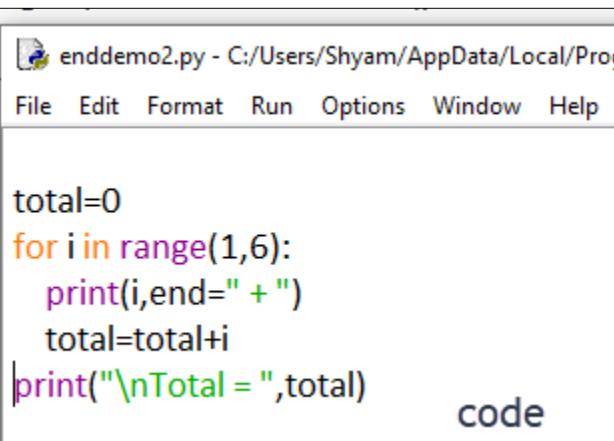


```
forloop8.py - C:/Users/Shyam/AppData/Local/Programs/Python  
File Edit Format Run Options Window Help  
ending=int(input("Enter ending value =>"))  
for i in range(ending,0,-1):  
    print(i)  
  
code
```

Example# Enter a number and print its multiplication table.

<pre>= RESTART: C:/Users/Shyam/AppData/Local/Temp/forloopTable.py - C:/Users/Shyam/AppData/Local/Temp/forloopTable.py Enter no =>5 5 X 1 = 5 5 X 2 = 10 5 X 3 = 15 5 X 4 = 20 5 X 5 = 25 5 X 6 = 30 5 X 7 = 35 5 X 8 = 40 5 X 9 = 45 5 X 10 = 50</pre>	output	 <pre>no=int(input("Enter no =>")) for i in range(1,11): print(no, " X ", i, " = ", no*i)</pre>	code
---	---------------	--	-------------

Example# Addition of first five natural numbers.

<pre>>>> = RESTART: C:/Users/Shyam/AppData/Local/Temp/enddemo2.py - C:/Users/Shyam/AppData/Local/Temp/enddemo2.py 1 + 2 + 3 + 4 + 5 + Total = 15 >>></pre>	output	 <pre>total=0 for i in range(1,6): print(i,end=" + ") total=total+i print("\nTotal = ",total)</pre>	code
--	---------------	--	-------------

Some good examples of for loop

<pre>for i in range(2,n/2): Logic..</pre>	<pre>for i in range(2,n+1,2): Logic..</pre>
<pre>for i in range(n,0,-2): Logic..</pre>	<pre>for i in range(1,n+1): print(i,end=" X ")</pre>

3.3.1.1 while loop

- ⌚ Same as For loop the condition is checked before entering into the loop. So, if the condition is false for the first time, the statements inside the while loop may not be executed at all.

- While loop keeps executing till the condition remains true.
- In the While loop, the programmer has to maintain or keep track of increment or decrement of value.

Syntax#

while condition:



Logic here



Example#

```
>>>
= RESTART: C:/U
1
2
3
4
5      output          code
>>>
```

forloop0.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32
File Edit Format Run Options

```
i=1
while i<=5:
    print(i)
    i=i+1
```

The while loop evaluates the conditional expression first. If the test expression is False, the while loop is terminated at that time. But if the test expression is True, logic inside the body of the while loop is executed and the expression is updated.

When the test expression is false, the while loop is terminated. The loop iterates while the condition is true.

Example# Print 1 to N

Python 3.8.5 (ta) Type "help", "co

```
>>>
= RESTART: C:/U Enter no =>5
1
2
3
4
5   output
>>>
```

no=int(input("Enter no =>"))
i=1
while i<=no:
 print(i)
 i=i+1

I code

Example# Print factorial of a number

Type "help", "copyr

```
>>>
= RESTART: C:/User Enter no =>5
5 X4 X3 X2 X1 X
Factorial = 120
>>>
```

no=int(input("Enter no =>"))
i=no
f=1
while i>=1:
 print(i,"X",end="")
 f=f*i
 i=i-1
print("\nFactorial = ",f)

output

I code

Example# Enter a number and print it in reverse order.

 Python 3.8.5 Shell	 whileRev.py - C:/Users/Shyam/AppData/Local
File Edit Shell Debug Options Window	File Edit Format Run Options Window
Python 3.8.5 (tags/v3.8.5:fd9871f, Jul 8 2021, 06:37:32) [MSC v.1916 64 bit (AMD64)]	no=int(input("Enter no =>"))
Type "help", "copyright", "credits" or "license" for more information	rem=0
>>>	rev=0
= RESTART: C:/Users/Shyam/AppData/Local/Temp/whileRev.py	while no>0:
Enter no =>843	rem=no%10
Reverse No = 348	rev=rev*10+rem
>>>	no=no//10
output	code
	print("Reverse No =",rev)

Example# Menu driven program to display a menu with three options square, cube, and exit. it should run until the user presses 3 to exit from the loop.

```
= RESTART: C:/Users/Shyam//  
Enter 1 for square  
Enter 2 for cube  
Enter 3 for exit  
Enter =>2  
Enter no =>10  
Cube = 1000  
Enter 1 for square  
Enter 2 for cube  
Enter 3 for exit  
Enter =>1  
Enter no =>5  
Square = 25  
Enter 1 for square  
Enter 2 for cube  
Enter 3 for exit  
Enter =>5  
Wrong opt  
Enter 1 for square  
Enter 2 for cube  
Enter 3 for exit  
Enter =>3  
Bye
```

output

whileMenu.py - C:/Users/Shyam/AppData/Local/Programs/
File Edit Format Run Options Window Help

```
while True:  
    print("Enter 1 for square")  
    print("Enter 2 for cube")  
    print("Enter 3 for exit")  
    option=int(input("Enter =>"))  
  
    if option==1:  
        x=int(input("Enter no =>"))  
        print("Square = ",x*x)  
    elif option==2:  
        x=int(input("Enter no =>"))  
        print("Cube = ",x*x*x)  
    elif option==3:  
        print("Bye")  
        break  
    else:  
        print("Wrong opt")
```

code

Difference between While and For Loop

While	For...loop
A while loop will "do" something as long as or until a condition is met.	A for loop will "do" something to everything which you wish to iterate through.
A while loop is an indefinite iteration that is used when a loop repeats an unknown number of times and ends when some condition is met.	A for loops is used when you have a definite iteration (the number of iterations is known).
Every element is fetched through the iterator/generator.	Every element is explicitly incremented or decremented by the user
while loop is slower than for loop.	for loop is faster than while loop.
While loop uses <code>i=i+1</code> or <code>i=i-1</code> expression for increment/decrement value of variable	For loop with <code>range()</code> function for increment/decrement value of variable
Syntax# while condition: Logic...	Syntax# for variable Name in sequence: Logic... for variable in range(starting, ending): Logic....
Example# while True: print "Hello" break	Example# for i in [11,55,66]: print(i)

3.3.1.3 Nested Loop

A loop within the loop is known as a nested loop. The depth of the nested loop depends on the complexity of a problem. We can have any number of nested loops as required. We can write a for loop within a for loop or a while loop within a while loop or a while within a for loop.

Syntax#

```
for variableName in range(starting,ending,step):
```



```
    for variableName in range(starting,ending,step)
```



```
        Logic here
```

```
while True:
```



```
    while True:
```



```
        Logic here
```

Example# Nested Loop – Square Pattern

The screenshot shows a Python terminal window. On the left, the output area displays the following text:

```
>>>
= RESTART: C:/Us
Enter limit =>5
*****
*****
*****
*****
*****
>>>
          output
```

On the right, the code area shows the following Python script:

```
Loopnested1.py - C:/Users/Shyam/AppData/Local
File Edit Format Run Options Window Help
no=int(input("Enter limit =>"))

for i in range(1,no+1):
    for j in range(1,no+1):
        print("*",end="")
    print("")
```

The word "code" is centered below the script. A hand cursor is hovering over the "Run" menu item in the top menu bar.

Example# Nested Loop – Number Pattern

```

>>>
= RESTART: C:/U
Enter limit =>5
1
12
123
1234
12345
>>> output

```

code

```

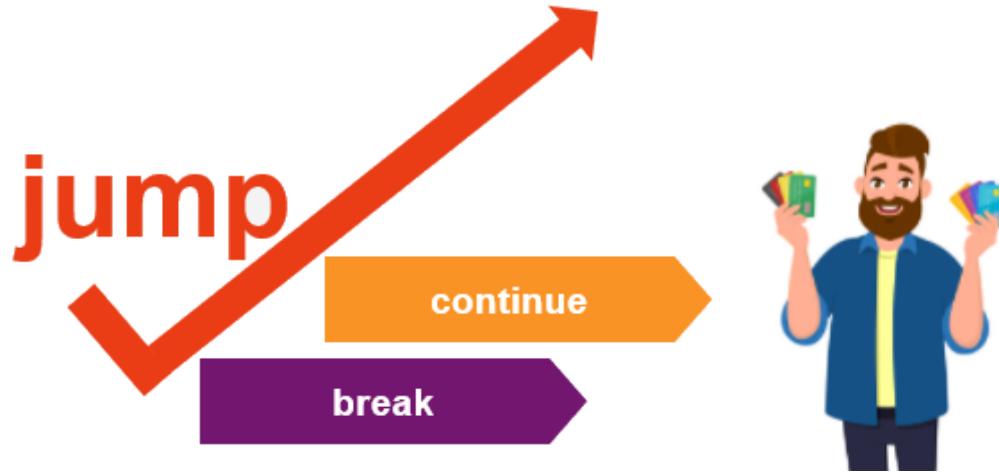
Loopnested2.py - C:/Users/Shyam/AppD
File Edit Format Run Options Wind
no=int(input("Enter limit =>"))

for i in range(1,no+1):
    for j in range(1,i+1):
        print(j,end="")
    print()

```

3.3.3 Branching Statement

Branching statements are also known as jumping statements. Jumping statements, jump from one statement to another, conditionally or unconditionally.



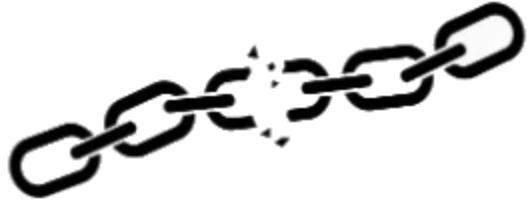
3.3.3.1 break

- Many a times in our daily life, we say that let's take a break from some work or let's make breakfast which translates to let's stop our routine work and now it's time for a break, let's jump out of it!
- The break statement is used to stop the current execution of the code and helps to exit the loop.

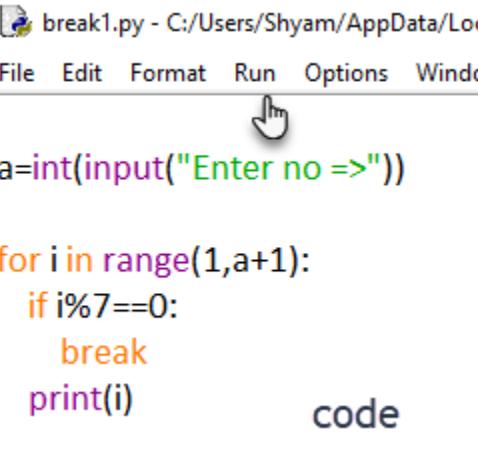
- In many situations, we need to get out of the loop before the loop execution is normally complete. We can use break keyword, in a For loop and a While loop.

Syntax#

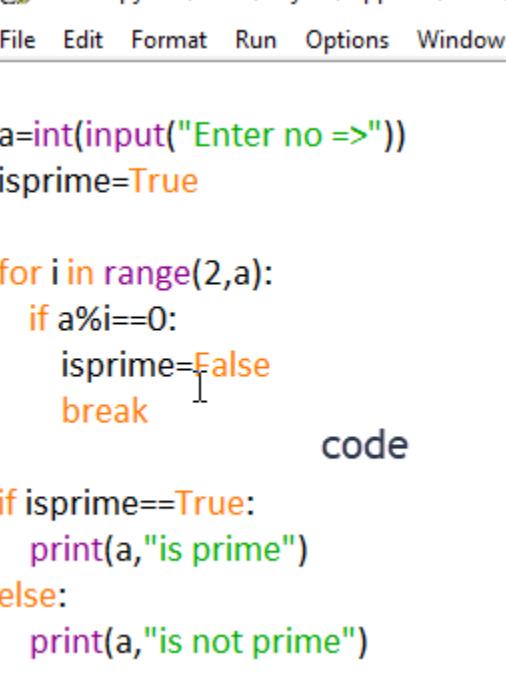
Break



Example# break the loop if the given number is divisible by 7

<pre>>>> == RESTART: C:/Users/Shyam/AppData/Local/Temp/Python27/pythonw.exe Enter no =>20 1 2 3 4 5 6 output >>></pre>	<p></p> <pre>a=int(input("Enter no =>")) for i in range(1,a+1): if i%7==0: break print(i)</pre>
--	--

Example# To check whether a number is prime or not.

<pre>>>> == RESTART: C:/Users/Shyam/AppData/Local/Temp/Python27/pythonw.exe Enter no =>22 22 is not prime >>></pre>	<p></p> <pre>a=int(input("Enter no =>")) isprime=True for i in range(2,a): if a%i==0: isprime=False break if isprime==True: print(a, "is prime") else: print(a, "is not prime")</pre>
--	--

3.3.3.2 Continue

- In Garba or any dance form, if we forget steps, we don't worry! We simply just continue after certain steps.
- When the program reaches the continue statement, the program skips the logic after continue and the flow reaches the next iteration of the loop.
- Continue is similar to break, but when a break statement is executed, it breaks the loop whereas continuing skips the rest of the statement and proceeds for the next iteration.
- We can use the Continue statement with For Loop and While Loop

Syntax#

continue

Example# Print 1 to N except 3 and 5

The screenshot shows a Python code editor window with a script titled 'code.py'. The code is as follows:

```
Type "help", "copyright" or "credits" for more information.
>>>
= RESTART: C:/Users/.../code.py
Enter limit =>10
1
2
3
4
5
6
7
8
9
10    output
>>>
```

The code uses a for loop to iterate from 1 to 10. It includes an if statement that checks if the current value of i is 3 or 5. If either condition is true, the continue statement is executed, which skips the remaining code in the loop body for that iteration. The output shows that the numbers 3 and 5 are omitted from the printed sequence.

4.1 Function

A function is a part of a program that performs a specific task and can be invoked from any part of the program.

A function is a set of statements that take inputs, do some specific computation, and produce output.

A function is a block of code that has a name and we can call it one or more times.

The basic philosophy of function is divide and conquer, by which a complicated task is successively divided into simpler and more manageable tasks which can be easily handled.

There are some situations when we need to write a particular block of code more than once in our program.

A program can be divided into smaller sub-programs that can be developed and tested successfully.

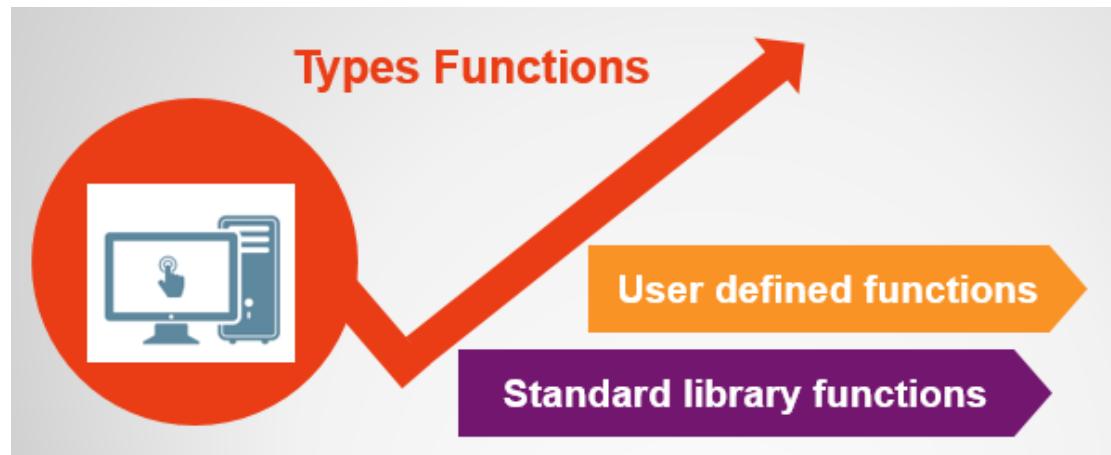
A function is a complete and independent program that is used (or invoked) by the main program or other subprograms.

A subprogram receives values called arguments from a calling program, performs calculations, and returns the results to the calling program.

A function is a self-contained block of statements that performs a task of some kind.

The function allows us to use the Top-down approach, functions are executed from top to bottom.

There are 2 types of functions



Advantages of Functions

- It provides modularity to the program.
- Program development is made easy due to modularity.
- Modular programming makes the Python program more readable.
- Avoid enabling of codes, due to function we can reuse code again and again hence it supports code reusability. We just have to call the function by its name to use it.
- It makes the program easier to design and understand
- Program testing becomes easy
- In the case of large programs with thousands of lines of code, debugging and editing becomes easier due to the usage of functions.

Characteristics of a function.

- There is no limit on the number of functions.
- Each function in a program is called in the sequence specified by the function.
- It facilitates top-down modular programming.
- A function can be called more than once.
- The length of a source program can be reduced by using functions.
- One function can call any other function.
- A function can also call itself.
- A function can return only one value or multiple values.
- We can specify as many parameters as are specified in the function definition.



4.4.1 Library Function or Inbuilt Function

These are the readymade functions available for use and they reside in their respective package or module. You do not need to write the code for it.

Example#

`abs()`, `set()`, `dict()`, `help()`, `min()`, `input()`, `eval()`, `int()`, `pow()`,
`print()`, `len()`, `max()`, `round()`



`float()`,

4.4.2 User Defined Function (UDF)

We can divide our code into separate functions.

User-defined functions are those functions that are defined by the user.

It is also known as a subprogram.

Functions are made for code reusability and for saving time and memory.

Advantages of UDF

- We can avoid writing redundant program code of some instructions again and again.
- Reducing the complexity of a program.
- Programs, by using functions become compact & easy to understand.

- Testing and correcting errors is easy because errors are localized and hence can be corrected easily.
- It provides top to down model which is easy to understand.
- Code Reusability: once a user-defined function is implemented it can be called or used as many times as required which reduces code repeatability and increase code reusability.

Disadvantages of UDF

- Execution is slow

4.4.2.1 Difference between UDF and Library function

Library function	UDF
It is provided by Python Programming Language.	The user creates it for reusability.
Sometimes we have to add a package/module to it.	There is no need to add any extra package/module file.
There is no error because they are pre-compiled.	Possibility of errors.
Library Functions are part of Python which is called runtime.	UDF is part of the program which compiles runtime.
Example# max() , min()	Example# <pre>def add(): a=20 b=2 print(a+b)</pre>

Syntax#

```
def fun(parameters):
    code
    return
```

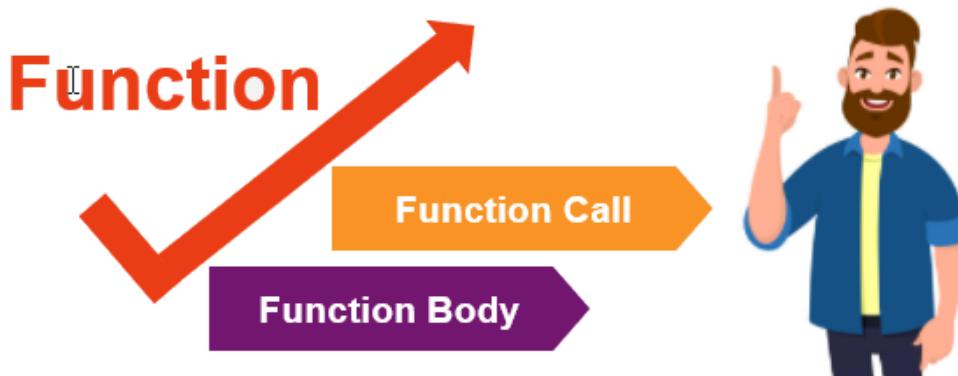
To define a function, we use the def keyword then the name of the function and we can pass some parameters if we want. Our logic/coding block of code will be executed when we call the function or the last line is an optional return statement.

Example#

<pre>def my_first_function(): print("Hello from a function")</pre>	<pre>def Square(): a=int(input("Enter any number =>")) print("Square = ",a*a)</pre>
--	--

Function Body	Function Call
<pre>def Add(): Logic... I def LeapYear(): Logic... def DollarToRupee(): Logic... def Maxbetween3(): Logic...</pre>	<pre>DollarToRupee() Add() LeapYear() Add() Maxbetween3() LeapYear()</pre>

4.4.2.2 Elements of User-defined function



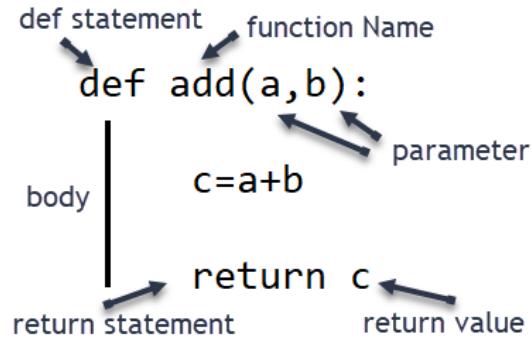
4.4.2.2.1 Function body

Function body/logic is a statement that informs the interpreter about

1. def statement

2. Name of the function
3. Number of parameters
4. Body of function, Logic/Code of the function
5. Return statement
6. Return value

The Function body consists of a block of statements that specify what task is to be performed. The function body consists of actual code - programmer's logic.



To use the function, the user needs to call it.

A function can be called by using the function name followed by a list of actual parameters. The arguments that we pass from the calling function are called actual arguments. The arguments declared at the time of definition of the program are called formal arguments. The variable name of actual arguments and formal arguments doesn't need to be the same.

Example # Function Body

```
def power():
    a=int(input("Enter base =>"))
    b=int(input("Enter power =>"))
    print("Answer =",a**b)
```



4.4.2.2.2 Function Call

Without calling the statement, there is no meaning of creating any function. To call a function, use the function name followed by parenthesis.

Example# power()

Example: Function to print the power of a number

Type "help", "copyright", >>> = RESTART: C:/Users/Shya Enter base =>2 Enter power =>5 Answer = 32 >>>	*fun2Power.py - C:/Users/Shyam/AppData/Local/Pro File Edit Format Run Options Window Help  def power(): a=int(input("Enter base =>")) b=int(input("Enter power =>")) print("Answer =",a**b)	
output	power()	code

Example: Function for max between two values, area of triangle, Fibonacci series

```

def max2():
    a=int(input("Enter no1 =>"))
    b=int(input("Enter no2 =>"))
    if a>b:
        print(a,"is greater")
    else:
        print(b,"is greater")

def areaofTriangle():
    height=int(input("Enter Height =>"))
    base=int(input("Enter Base =>"))
    print("Area of Triangle = ",height*base*0.5)

def fibonacciSeries():
    nterms = int(input("How many terms? "))
    n1, n2 = 0, 1
    count=0
    print("Fibonacci sequence:")
    while count < nterms:
        print(n1,end=" ")
        nth = n1 + n2
        n1 = n2
        n2 = nth
        count=count+1

max2()
areaofTriangle()
fibonacciSeries()

```



Enter no1 =>22	output
Enter no2 =>33	
33 is greater	
Enter Height =>100	
Enter Base =>200	
Area of Triangle = 10000.0	
How many terms? 5	
Fibonacci sequence:	
0 1 1 2 3	

Up till here, we have discussed functions without arguments, So now let's discuss another type of functions known as a function with argument.

4.1.1.2 What is an argument or parameter?

- A function may or may not have arguments.
- When a function is defined, its parameters are listed in parentheses next to its name.
- If a function does not accept any parameters, leave the parentheses empty ()
- If we want to pass values at the time of calling in user define function then we need to create a function with arguments.
- Usually, we pass the argument when we want to pass the same values to the number of functions.
- In the below image, the function has 2 arguments a and b.

```
def add(a,b):  
    print("Add = ", a+b)  
  
add(5,2)  
  
add(10,30)
```

Parameter and Argument

A parameter sometimes is also known as the formal argument in the variable listed inside the parentheses in the function declaration/body.

Example# def areaofTriangle(height,base):

The arguments that are sent or passed in a function call are called
or actual arguments.



Example#

```
def areaofTriangle(height,base): #height and base are parameters  
    print("Area = ",height*base*0.5)
```

areaofTriangle(100,200) # arguments – 100,200

Examples#

Function Body	Function Call
---------------	---------------

<code>def Add(a,b):</code>	<code>Add(22,3)</code>
<code> Logic..</code>	<code>LeapYear(2022)</code>
<code>def LeapYear(year):</code>	<code>DollartoRs(1000)</code>
<code> Logic..</code>	<code>MaxBetween3(11,22,33)</code>
<code>def DollartoRs(amt):</code>	<code>SimpleInterest(20000,2,2)</code>
<code> Logic..</code>	
<code>def MaxBetween3(a,b,c):</code>	
<code> Logic..</code>	
<code>def SimpleInterest(p,r,t):</code>	
<code> Logic..</code>	

4.1.1.2.0 Types of Function Arguments

Required/Compulsory arguments

Default arguments

Keyword arguments

Variable-length arguments



4.1.1.2.1

Required/Compulsory arguments

Compulsory argument means when a user calls the function, we have to pass the value. For instance, in the example shown below, function "Add5Values" having 5 arguments, we have to call this function with 5 values separated by comma (,), which means we have to pass 5 values exactly.

Syntax#

```
def functionName(variableName1,variableName2,...N):  
    Logic...  
    I  
functionName(value1,value2,value....N)
```

Example#

```
def Add5Values(a,b,c,d,e):  
    print("Add =",a+b+c+d+e)  
  
Add5Values(50,20,30,10,40)
```

Example: Create four functions add(),sub(),mul(),div() with two arguments and pass the value while calling the function.

<pre> File Edit Shell Debug Python 3.8.5 (tags/v3.8.5:40e98d7, May 27 2021, 16:50:31) Type "help", "copyright" or "license" for more information >>> = RESTART: C:/Users/DELL/PycharmProjects/untitled1.py Enter no1 =>22 Enter no2 =>2 Add = 24 Sub = 20 Mul = 44 Div = 11.0 >>> </pre> <p style="text-align: center;">output</p>	<pre> File Edit Format Run Options Window def add(a,b): print("Add =",a+b) def sub(a,b): print("Sub =",a-b) def mul(a,b): print("Mul =",a*b) def div(a,b): print("Div =",a/b) a=int(input("Enter no1 =>")) b=int(input("Enter no2 =>")) add(a,b) sub(a,b) mul(a,b) div(a,b) </pre> <p style="text-align: right;">code</p>
---	---

4.1.1.2.2 Variable name of Parameter and Actual Arguments

The variable name of the parameters and actual arguments doesn't need to be the same, even though one can pass the value directly without storing values in variables.

Example: Create a function for Simple Interest which accepts three values Principal value, Rate, Time, and display the Interest, Try to call it two times, First time with user-defined values (taking values from the user) and the second time with fixed values.

The screenshot shows a Python IDLE window. On the left, the 'output' pane displays the following session:

```

Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 29 2019, 19:19:28)
Type "help", "copyright", "credits" or
>>>
= RESTART: C:/Users/Shyam/AppData
Enter Principal value =>30000
Enter Rate of Interest =>2.5
Enter No of Year =>5
Interest = 3750.0
Interest = 400.0
>>>

```

On the right, the 'code' pane contains the following Python code:

```

def SimpleInterest(p,r,t):
    print("\nInterest =", (p*r*t)/100)

a=int(input("Enter Principal value =>"))
b=float(input("Enter Rate of Interest =>"))
c=float(input("Enter No of Year =>"))

SimpleInterest(a,b,c) # p=a r=b t=c

SimpleInterest(10000,2,2) # p=10000 r=2 t=2

```

4.1.1.2.3 Default Argument

- Usually, we have to pass all the arguments to the function when we are trying to call it, but it is possible to call a function without specifying all of its arguments. This won't work on just any function: The function declaration must provide default values for those arguments that are not specified. If no value is passed for an argument when the function is called then the default value specified above will be passed.
- We can specify a default value for arguments. If the user calls the function, they can skip providing a value for that argument. The default value is used. We can set any value as a default argument.
- Functions can have any number of default arguments. default value is assigned by using the assignment (=) operator.
- Example# The first one required argument and the required default.

Example# def add(int a, int b=0):

- This one is a wrong def add(int a=0, int b): The rule defining default value is When a default is provided in the function argument; all the subsequent arguments must have a default value. The correct one would be def add(int a=1,int b=0,int c=0). It can also be stated as default arguments are assigned from right to left



The

other

of

the

Syntax#

def functionName(varName1,varName2=defaultValue,varName3=defaultValue):
 Logic...

functionName(value1)
functionName(value1,value2)
functionName(value1,value2,value3)

Example#

- def Addition(a,b,c,d=20): # a b c are required arguments and d is default argument
- def Addition(a,b,c=10,d=20): # a and b are required arguments and c and d are default arguments
- def Addition(a=10,b=10,c=10,d=20): # Here all four arguments are default argument
- def Addition(a=10,b=10,c,d=20): # Invalid , we can have (a,b=10,c=10,d=20)
- def Addition(a=10,b=10,c,d): # Invalid , we can have (a,b,c=10,d=10)

Example# A multiplication function with five arguments, where two arguments are compulsory and rest of three contains default value 1.

```
Type "help", "copyright", "credits" or "license" for more information
>>>
= RESTART: C:/Users/Shyam/AppData/Local/... - 
File Edit Format Run Options Window Help
def multi(no1,no2,no3=1,no4=1,no5=1):
    print("Multi = ",no1*no2*no3*no4*no5)

Multi = 200
Multi = 6000
Multi = 240000
Multi = 12000000
>>> output
multi(10,20)
multi(10,20,30)
multi(10,20,30,40)      code
multi(10,20,30,40,50)
```

Example# Create a print function, which accepts a maximum of two arguments if we pass only one argument then it displays square of that number else displays power that raise a number m to power n.

```
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/fun7def.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/fun7def.py =  
def printdata(string="m",no=10):  
    print(string*no)  
  
>>>  
printdata()  
printdata("t")  
printdata("k",20)
```

If we use any smart Editor like PyCharm or Jupiter then it will display,

```
def printdata(string="m", no=10):  
    print(string*no)
```

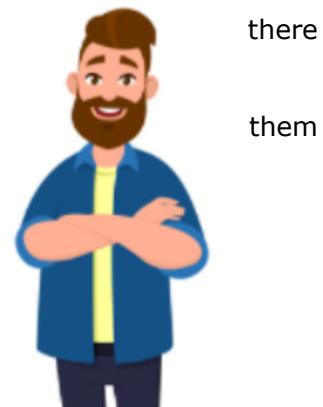
4.1.1.2.4 Keyword/Named Argument

Keyword arguments are related to the function calls.

We can also pass values in the arguments of function with the key = value syntax. A keyword argument is preceded by a parameter and the assignment operator, =

If we pass value according to keyword arguments in a function, then there is no need to remember the order of the parameters.

Keyword arguments allow us to skip sequence arguments or place them out of order because the Python interpreter identifies the keywords provided to match the values with parameters.



Syntax#

```
def functionName(variableName1,variableName2):  
    Logic...
```

Example#

```
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python  
No = 12 Name = Shyam Std = 5 Div = B  
No = 22 Name = Radha Std = 7 Div = A
```

output

 *funarb1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/funarb

File Edit Format Run Options Window Help

```
def printStudentData(no,name,std,div):  
    print("No =",no,"Name =",name,"Std =",std,"Div =",div)  
    code  
printStudentData(name="Shyam",std=5,div="B",no=12)  
printStudentData(std=7,div="A",no=22,name="Radha")
```

4.1.1.2.5 Variable Length/Arbitrary Argument

When we are not sure about how many arguments our function will get at runtime, we can use the arbitrary arguments `*args`(Non-keyword Arguments) and `**kwargs`(keyword Arguments).

Syntax#

```
def functionName(*args,**kargs):  
    Logic...
```

`*args` is a variable number of arguments, `*args` contains all the values which user passed at the time of call type of `*args`, `*args` data type is a tuple, and `**kwargs` is a variable number of keyword arguments, which contains argument names and values of those arguments passed at the time of call and data type of is a dictionary.

```
>>>  
= RESTART: C:/U  
Programs/Python  
<class 'tuple'>  
<class 'dict'>  
>>>  
output      def add(*a,**krgs):  
                  print(type(a))  
                  print(type(krgs))  
                  code  
add(11,22,c=30,d=40)
```



Both arguments are optional, we can use either `*args` or `**krgs` in the function, it is not compulsory to use them together.

Example# Let's print the value of * and **

```
= RESTART: C:/Users/Shyam/AppData/Local...  

File Edit Format Run Options Window Help  

def add(*a,**krgs):  

    print(a)  

    for key, value in krgs.items():  

        print(key, value)  

    add(11,22,33,44)  

    a=5  

    b=7  

    add(a,b,c=30,d=40)
```

We can change the variable name of these two arguments, instead of *args we can write *a or anything, and same way instead of **kwargs we can write **b or anything but just keep in mind the difference between * and **.

Example# Let's print the value of * addition of all the passed value

```
= RESTART: C:/Users/Shyam/AppData/Local...  

File Edit Format Run Options Window  

Values = (10, 20, 30)  

Total of all = 60  

Values = (1, 2, 3, 4, 5)  

Total of all = 15  

>>>  

    add(10,20,30)          code  

    add(1,2,3,4,5)
```

3.5 What is return value?

- The python 'return value', exists as a function and instructs python to continue executing the main program.
- A function doesn't need to return a value but it can.
- A function can return one value or more values.

If a function returns a value, then the function calling statement will change as given below:

Syntax#

variableName=functionName(arguments)

Example# Create 2 functions, one with Max between 2 numbers, it should return the maximum number from 2 values, while another function is for values which are divisible by 7 and return the count of numbers which are divisible by 7.

```
>>>
= RESTART: C:/Users/S
7 14 21 28
Total Count = 4

Max value is = 33
>>>

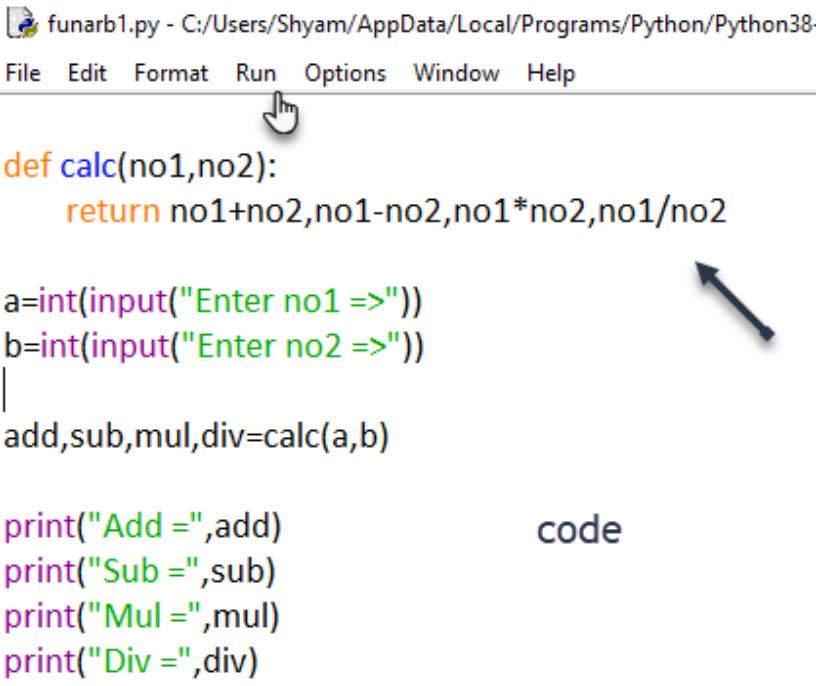
output                                     code
def count7Divisable(no):
    cnt=0
    for i in range(1,no+1):
        if i%7==0:
            print(i,end=" ")
            cnt=cnt+1
    return cnt ←

def maxbetweentwo(no1,no2):
    if no1>no2:
        return no1 ←
    else:
        return no2 ←

totalcount=count7Divisable(30)
print("\nTotal Count = ",totalcount)

maxvalue=maxbetweentwo(22,33)
print("\nMax value is = ",maxvalue)
```

Example# Function with multiple return statement

<pre>[= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/funarb1.py] Enter no1 =>35 Enter no2 =>5 Add = 40 Sub = 30 Mul = 175 Div = 7.0 >>></pre>	<p></p> <pre>def calc(no1,no2): return no1+no2,no1-no2,no1*no2,no1/no2 a=int(input("Enter no1 =>")) b=int(input("Enter no2 =>")) add,sub,mul,div=calc(a,b) print("Add =",add) print("Sub =",sub) print("Mul =",mul) print("Div =",div)</pre>
output	code

Some examples of function returning value.

Power function <pre>def power(a,b): return a**b c=power(2,7) print("Power of 2 ** 7 is",c)</pre>	Square function <pre>def square(a): return a*a print(square(5))</pre>
CheckCase function <pre>def checkcase(string): if string.isupper()==True: return "upper" elif string.islower()==True: return "lower" else: return "mix" print(checkcase("ram"))</pre>	Check Palindrome function <pre>def checkPalindrom(no): c=no x=0 rev=0 while no>0: x=no%10 rev=rev*10+x no=no//10 if c==rev: return "Number is Palindrome" else: return "Number is not Palindrome" print(checkPalindrom(121))</pre>

--	--

4.2.1 Scope and visibility of variables in functions

4.2.1.1 Scope

The scope determines that over which parts of the program the variable is available and can be accessed.

Variable scope is a region of a program in which a variable is available for use.

There are two types of Scope:



Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

Global Scope	Local Scope
Global variables can be accessed anywhere in the program	Local variables can be accessed only inside the function in which they are declared
When we run a program, the global variables are brought into scope	When we call a function, the variables declared inside it are brought into scope
We can change the global variable's value from the function	We cannot change the local variable's value outside of the function
To access variables inside the function we have to use global keywords.	There is no local keyword.
Example#	Example#

a=50	Type "help", "copyright", "credits" or "license" for more information	<code>def printdata():</code>	= RESTART: C:/Users
<code>def printdata():</code>	<code>>>></code>	<code>a=50</code>	Local a= 50
print("Global a= ",a)	<code>>>> </code>	<code>print("Local a= ",a)</code>	<code>>>> </code>

Example# Global and Local Scope of Variable, here variable a is Global and we are just printing value of Global variable inside the function

<code>def printdata():</code>	= RESTART: C:/Users
a=50	Local a= 50
print("Local a= ",a)	<code>>>> </code>
<code>printdata()</code>	output

Example# Global and Local Scope of Variable, here variable a is Global and we changed the value of 'a' Locally

FILE EDIT OPEN DEBUG OPTIONS	File Edit Format Run Options Window Help
Python 3.8.5 (tags/v3.8.5:58	a=50
Type "help", "copyright", "cre	
>>>	<code>def CheckScope():</code>
= RESTART: C:/Users/Shyam/	a=100
Before function Call = 50	print("Inside the function = ",a)
Inside the function = 100	
After function Call = 50	print("Before function Call = ",a)
>>>	
	CheckScope()
	code
	print("After function Call = ",a)
	output

In the above code, we print the program before the call of global value 'a'. When we call the function () check scope () it will create a new

variable inside the local variable with the same name 'a' and initialize 'I' with 100 and then when we create print function again after the function call it will f=print 50 global value because it creates local variable inside the function and not storing the value 100 in the global one.

Example# Global and Local Scope of Variable, here variable 'a' is Global and we changed the value of 'a' Locally

If we want to access global variable inside the function we have to use the "global" keyword

<pre>Python 3.8.5 (tags/v3.8.5:58004a2, Jul 29 2020, 15:53:45) [Cl] globalvar2.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe File Edit Shell Debug Options Window Help Type "help", "copyright", "credits" or "license" for more information. >>> = RESTART: C:/Users/Shyam/PycharmProjects/PythonBasics/globalvar2.py Before function Call = 50 Inside the function = 100 After function Call = 100 >>></pre>	<pre>a=50 def CheckScope(): global a ← a=100 print("Inside the function =",a) print("Before function Call =",a) CheckScope() print("After function Call =",a)</pre>
output	code

4.3.1 Python Standard Library

There are more than 137,000 libraries that exist in Python and updating according to a new version. Each library contains several functions. Let's see some functions

4.3.1.1 For Input

input()

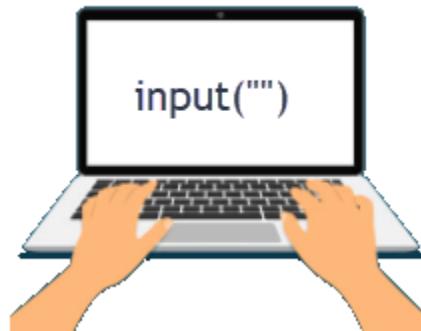
The input () function takes input from the user and returns it. Input functions have only one argument and that is also optional. The default return type of input () function is a string or say text. Whatever value we entered with the help of input () function is always string even though if we entered 5.5, it converts into "5.5" so We have to use the casting function to convert the entered value.

Syntax#

Variable Name=input([prompt])

Example#

```
x = input ()  
name = input ("Enter your name =>")  
age=int (input ("Enter your age =>"))  
  
print ('You Entered ', x)  
print ('Your name is', name)  
print ("Your age is", age)
```



4.3.1.2 For Print

print()

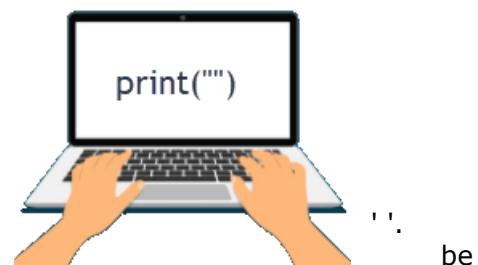
The print () function in Python is used to print a specified message on the screen or to the file.

Syntax#

```
print(object(s), sep=separator, end=end, file=file,  
flush=flush)
```

objects (optional) - object or objects to be printed. sep (optional)- objects are separated by sep. Default value is end='end' (optional): it determines which object should be printed at last. Default value is '\n'.

file(optional). An object with a write method. Default is sys. stdout default prints objects on the screen.



flush (optional)- If True, the stream is forcibly flushed. Default value: False

Example#

```
print ()  
  
print ("Hello My name is Shyam")  
  
print (5 * "Ram")  
  
lightbill = 8250  
print ("Your current light bill is", lightbill)  
  
name = 'Yash'  
print ("My name is",name,sep='---',end = "\n\n")  
print ("I am living in Canada")  
  
a=20  
b=10  
c=100  
print ("a = ",a,"b = ",b)  
print ("a =", a, sep='00000', end='\n\n')  
print (a, b, c, sep=':')  
  
listx = ["apple", "banana", "cherry","mango"]  
print(listx)  
  
print(a,end="+")  
  
print(b)
```



Output#

```
= RESTART: C:/Users/Shyam/AppData/Local/Progra
```

```
Hello My name is Shyam  
RamRamRamRamRam  
Your current light bill is 8250  
My name is---Yash
```

```
I am living in Canada  
a = 20 b = 10  
a =0000020
```

```
20:10:100  
['apple', 'banana', 'cherry', 'mango']  
20+10
```



4.3.2 Some Math Functions

abs()

Returns the absolute x, which means only positive value.

Syntax#

```
variableName = abs(x)
```

Example#

The screenshot shows a Python code editor window titled "absdemo.py". The code defines variables a, b, c, and d with their respective negative values, then prints their absolute values using a for loop. The output window shows the results: a=5.6, b=6.7, c=-10, d=9.

```
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/python.exe
a = -5.6 abs(a) = 5.6
b = -5.6 abs(b) = 6.7
c = -10 abs(c) = 10
d = 9 abs(d) = 9
>>>

output
print("a =",a," abs(a) =",abs(a))
print("b =",a," abs(b) =",abs(b))
print("c =",c," abs(c) =",abs(c))
print("d =",d," abs(d) =",abs(d))
```

sum()

Returns the sum of the value stored in the List/Tuple/Set.

Syntax#

```
variableName = sum(iterable)
```

Example#

The screenshot shows a Python code editor window titled "sumdemo.py". The code defines two lists, list1 and tuple1, and prints their sum using the sum() function. The output window shows the results: sum of list1 = 29 and sum of tuple1 = 10.

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/python.exe
sum of list1 = 29
sum of tuple1 = 10
>>>

output
list1=[1,1,5,6,7]
tuple1=(1,2,3,4)
print("sum of list1 =",sum(list1))
print("sum of tuple1 =",sum(tuple1))
```

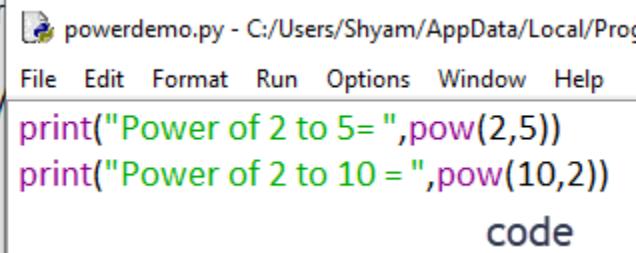
pow(x, y)

Returns the value of x to the power of y

Syntax#

```
variableName = pow(x, y)
```

Example#

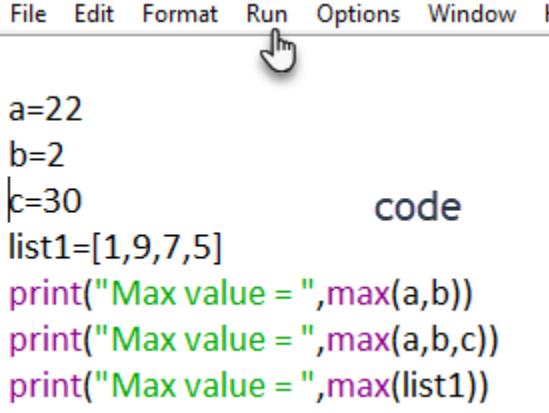
max (x, y,...)	Type "help", "copyright", "cr >>> = RESTART: C:/Users/Shyam/ Power of 2 to 5= 32 Power of 2 to 10 = 100 >>> output	 powerdemo.py - C:/Users/Shyam/AppData/Local/Prog File Edit Format Run Options Window Help print("Power of 2 to 5 =",pow(2,5)) print("Power of 2 to 10 =",pow(10,2)) code
---------------------------	---	--

maximum value from 2 or more values.

Syntax#

```
variableName = max(x,y,...)
```

Example#

Type "help", "COPYT >>> = RESTART: C:/Users/ Max value = 22 Max value = 30 Max value = 9 >>> output	 File Edit Format Run Options Window Help a=22 b=2 c=30 list1=[1,9,7,5] print("Max value =",max(a,b)) print("Max value =",max(a,b,c)) print("Max value =",max(list1)) code
---	--

min(x, y,...)

Returns the minimum value from 2 or more values.

Syntax#

```
variableName = min(x,y,...)
```

Example#

```
Type "help", "copyright", "credits" or "license()" for more information
>>>
= RESTART: C:/Us a=22
Min value = 2 b=2
Min value = 2 c=30          code
Min value = 1 list1=[1,9,7,5]
>>>           print("Min value =",min(a,b))
                  print("Min value =",min(a,b,c))
                  print("Min value =",min(list1))
output
```

divmod (x, y)

Takes two numbers and returns a pair of numbers consisting of their quotient and remainder.

Syntax#

```
variableName1,variableName2 = divmod(x, y)
```

Example#

```
Type "help", "copyright", "credits" or "license()" for more information
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Pro
divmod(22,2) Quotient = 11 Remainder = 0
divmod(22,3) Quotient = 7 Remainder = 1
divmod(22,5) Quotient = 4 Remainder = 2
>>>           output
                           q1,r1=divmod(22,2)          code
                           q2,r2=divmod(22,3)
                           q3,r3=divmod(22,5)
                           print("divmod(22,2) Quotient =",q1," Remainder =",r1)
                           print("divmod(22,3) Quotient =",q2," Remainder =",r2)
                           print("divmod(22,5) Quotient =",q3," Remainder =",r3)
```

4.3.3 Module

Python has several inbuilt functions. These functions are grouped into a module.

For Example # Whatsapp, NetFlix, Amazon, Uber are modules which we use for a specific purpose, So we know when we want to chat we use WhatsApp module, when we want to watch web series we use NetFlix, when we want to buy something we use Amazon and when we want to book a cab we use Uber.



The module is nothing but a Python file containing a set of functions and variables to be used in an application. For example, the Math module contains various functions and variables related to mathematics.



If we want a checklist of in-built modules then we can use the help () function.

```
>>> help('modules') code
```

Please wait a moment while I gather a list of all available modules...

```
pygame 2.0.1 (SDL 2.0.14, Python 3.8.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
Arithmet icop     builtins      logging      pyparse
Loopn ested1     bz2         logical      pyparsing      .....
PIL          cProfile    logici l     ppvri nd
```

To get information about a particular module we can write help('module name').

```
>>> help('math')
Help on built-in module math:

```

code

NAME

math

DESCRIPTION

This module provides access to the mathematical functions defined by the C standard.

FUNCTIONS

acos(x, /)
Return the arc cosine (measured in radians) of x.

acosh(x, /)
Return the inverse hyperbolic cosine of x.

4.3.4 math Module

This module provides access to the mathematical functions.

Pi

Returns constant PI's value 3.141592

Syntax#

variableName = pi



Example#

```
>>>
= RESTART: C:/Users/Shyam/
PI = 3.141592653589793
>>>
```

output

```
mathmodul1.py - C:/Users/Shyam/
File Edit Format Run Options
import math
print ("PI =",math.pi)
```

code

ceil()

Returns the smallest integer greater than or equal to x.

Syntax#

```
variableName = ceil(x)
```

Example#

The screenshot shows a Python IDLE window with the following code and output:

```
PI = 3.141592653589793
>>>
= RESTART: C:/Users/Shyam/PycharmProjects/PythonProject/venv/Scripts/python.exe
5.5 and ceil= 6
-5.5 and ceil= -5
5.9 and ceil= 6
-5.9 and ceil= -5
5.1 and ceil= 6
-5.1 and ceil= -5
>>>
                                code
output
                                print ("5.5 and ceil= ",math.ceil(a))
print ("-5.5 and ceil= ",math.ceil(b))
print ("5.9 and ceil= ",math.ceil(c))
print ("-5.9 and ceil= ",math.ceil(d))
print ("5.1 and ceil= ",math.ceil(e))
print ("-5.1 and ceil= ",math.ceil(f))
```

floor()

Returns the largest integer greater than or equal to x.

Syntax#

```
variableName = floor(x)
```

Example#

>>>	File Edit Format Run Options Window Help
= RESTART: C:/Users/Sh	import math
5.5 and floor= 5	a=5.5
-5.5 and floor= -6	b=-5.5
5.9 and floor= 5	c=5.9
-5.9 and floor= -6	d=-5.9
5.1 and floor= 5	e=5.1
-5.1 and floor= -6	f=-5.1
>>>	code
output	print ("5.5 and floor= ",math.floor(a)) print ("-5.5 and floor= ",math.floor(b)) print ("5.9 and floor= ",math.floor(c)) print ("-5.9 and floor= ",math.floor(d)) print ("5.1 and floor= ",math.floor(e)) print ("-5.1 and floor= ",math.floor(f))

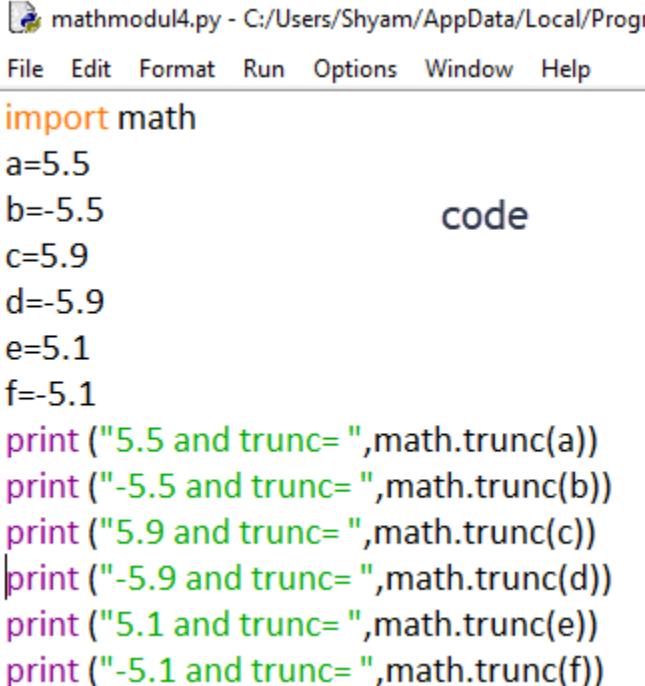
trunc()

Returns the Real value x truncated to an int

Syntax#

variableName = trunc(x)

Example#

<pre>>>> = RESTART: C:/Users/Shyam/PycharmProjects/PythonPrograms/mathmodul4.py 5.5 and trunc= 5 -5.5 and trunc= -5 5.9 and trunc= 5 -5.9 and trunc= -5 5.1 and trunc= 5 -5.1 and trunc= -5 >>></pre> <p style="text-align: center;">output</p>	 <p>File Edit Format Run Options Window Help</p> <pre>import math a=5.5 b=-5.5 c=5.9 d=-5.9 e=5.1 f=-5.1 print ("5.5 and trunc= ",math.trunc(a)) print ("-5.5 and trunc= ",math.trunc(b)) print ("5.9 and trunc= ",math.trunc(c)) print ("-5.9 and trunc= ",math.trunc(d)) print ("5.1 and trunc= ",math.trunc(e)) print ("-5.1 and trunc= ",math.trunc(f))</pre> <p>code</p>
---	--

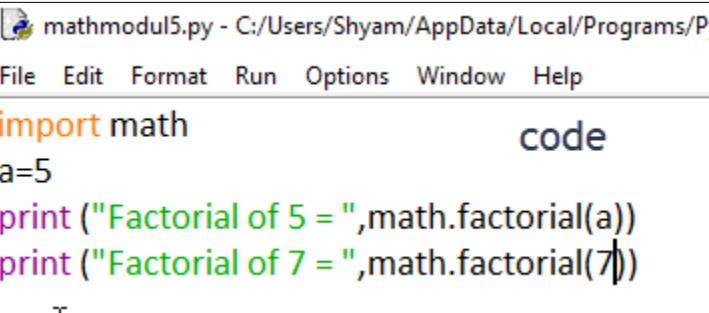
factorial()

Returns the factorial of x.

Syntax#

variableName = factorial(x)

Example#

<pre>type help , copyright >>> = RESTART: C:/Users/Shyam/PycharmProjects/PythonPrograms/mathmodul5.py Factorial of 5 = 120 Factorial of 7 = 5040 >>></pre> <p style="text-align: center;">output</p>	 <p>File Edit Format Run Options Window Help</p> <pre>import math a=5 print ("Factorial of 5 = ",math.factorial(a)) print ("Factorial of 7 = ",math.factorial(7))</pre> <p>code</p>
--	---

remainder

Returns remainder after dividing x by y.

Syntax#

variableName = remainder (x, y)

QUOTIENT 87

8 | 703
DIVISOR - 64

63
- 56
REMAINDER 7

Example#

```
>>>  
= RESTART: C:/User  
20%2 = 0.0  
20%3 = -1.0  
>>> output
```

mathmodule.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe
File Edit Format Run Options Window Help code

```
import math  
print ("20%2 =",math.remainder(20,2))  
print ("20%3 =",math.remainder(20,3))
```

sqrt

Returns the square root of x

Syntax#

variableName = sqrt(x)

Example#

```
= RESTART: C:/Users/Shyam/  
Square Root of 25 = 5.0  
Square Root of 10 = 10.0  
>>> output
```

mathmodul6.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe
File Edit Format Run Options Window Help code

```
import math  
print ("Square Root of 25 =",math.sqrt(25))  
print ("Square Root of 10 =",math.sqrt(100))
```

sin(x)

Returns the sine of x radians.

Syntax#

variableName = sin(x)

Example#

<pre>>>> = RESTART: C:/Users sin(x) is : 1.0 >>> output</pre>	<p>File Edit Format Run Options Window Help</p> <pre>import math angleInRadian = math.radians(90) print('sin(x) is :', math.sin(angleInRadian))</pre>
---	---

cos()

Returns the cosine of x radians.

Syntax#

```
variableName = cos(x)
```

Example#

<pre>>>> = RESTART: C:/Users/Shyam/AppData cos(x) is : 6.123233995736766e-17 >>> output</pre>	<p>File Edit Format Run Options Window Help</p> <pre>import math angleInRadian = math.radians(90) print('cos(x) is :', math.cos(angleInRadian))</pre>
---	---

tan(x)

Returns the tangent of x radians.

Syntax#

```
variableName = tan(x)
```

Example#

<pre>cos(x) is : 6.123233995736766e-17 >>> = RESTART: C:/Users/Shyam/AppData tan(x) is : 1.633123935319537e+16 >>> output</pre>	<p>mathmodul7.py - C:/Users/Shyam/AppData/Local/Programs/Py</p> <p>File Edit Format Run Options Window Help</p> <pre>import math angleInRadian = math.radians(90) print('tan(x) is :', math.tan(angleInRadian))</pre>
---	---

4.3.5 random Module

Python has a built-in module that we can use to generate random numbers. Random numbers mean any number in a particular range. We can use random numbers to make puzzles or store values in the sequence like List, Tuple.

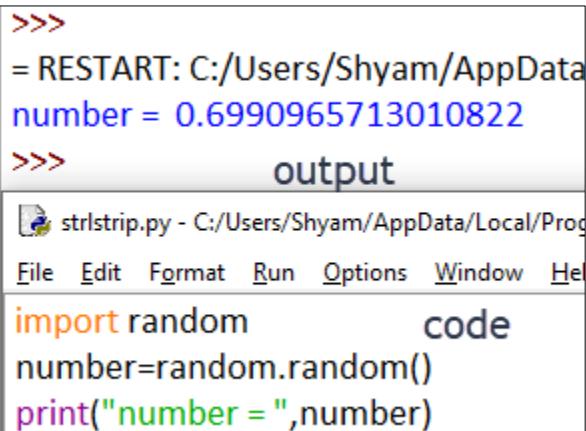
Generate Random Floats

Returns a random float number between 0.0 to 1.0

Syntax#

random. random()

Example#

```
>>>
= RESTART: C:/Users/Shyam/AppData
number = 0.6990965713010822
>>>      output


```

import random code
number=random.random()
print("number = ",number)

```


```



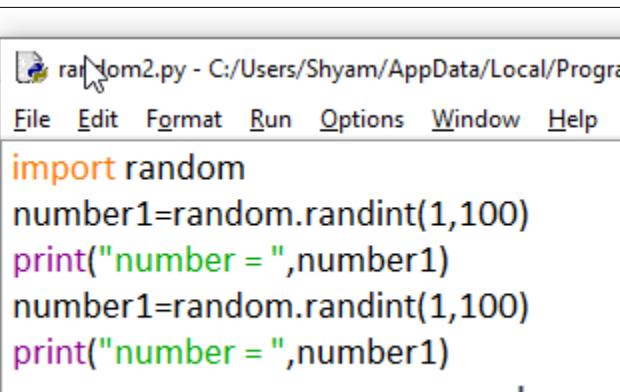
Generate Random Integers

Returns a random integer number between the starting and ending value.

Syntax#

random. Randint (starting, ending)

Example#

```
>>>
= RESTART: C:/U
number = 3      File Edit Format Run Options Window Help
number = 5
>>>      output
import random
number1=random.randint(1,100)
print("number = ",number1)
number1=random.randint(1,100)
print("number = ",number1)
code

```



Example# Let's create a puzzle of Addition

random3.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/random3.py (3.8.5)

File Edit Format Run Options Window Help

```
import random
number1=random.randint(1,20)
number2=random.randint(1,20)
add=number1+number2
print("Please Enter Addition between ",number1," and ",number2)
ans=int(input("Enter =>"))

if ans==add:
    print("Correct Answer")          code
else:
    print("Wrong Answer")
```

= RESTART: C:/Users/Shyam/AppData/Local/Programs/P

Please Enter Addition between 18 and 16

Enter =>24

Wrong Answer

output

>>>

= RESTART: C:/Users/Shyam/AppData/Local/Programs/P

Please Enter Addition between 3 and 20

Enter =>23

Correct Answer

Example# Guess my number

```
File Edit Format Run Options Window Help
import random
number=random.randint(1,20)

youtry=0
guessno=0

while guessno!=number:
    youtry=youtry+1
    guessno=int(input("Enter your guess no =>"))
    if guessno!=number:
        print("Please try ones again")

print("Total Try = ",youtry)
```

code

```
= RESTART: C:/Users/Shyam/AppData/Local/I  
Enter your guess no =>1  
Please try ones again  
Enter your guess no =>2      output 1  
Total Try = 2  
>>>  
= RESTART: C:/Users/Shyam/AppData/Local/I  
Enter your guess no =>2  
Please try ones again  
Enter your guess no =>4  
Please try ones again  
Enter your guess no =>5  
Please try ones again  
Enter your guess no =>6      output 2  
Please try ones again  
Enter your guess no =>5  
Please try ones again  
Enter your guess no =>8  
Please try ones again  
Enter your guess no =>9  
Please try ones again  
Enter your guess no =>1  
Total Try = 8  
>>>
```

Generate Random Numbers within Range

Returns a randomly selected element from the range specified in the function start, stop and step arguments. The default Range function is 0 and the step is 1.

Syntax#

random. randrange (starting, ending, step/gap)

Example#

Type "help", "copyright", "credits" or "license()" for more information

>>>

= RESTART: C:/Users/

Number1 = 2

Number2 = 10

Number3 = 1

>>>

output

random5.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8

File Edit Format Run Options Window Help

```
import random
number1=random.randrange(1, 20)
number2=random.randrange(1, 20,3)
number3=random.randrange(1, 20,5)
print("Number1 =",number1)
print("Number2 =",number2)
print("Number3 =",number3)
```

code

Select Random Elements

Returns a randomly selected element from a given sequence.

Syntax#

random.choice(sequence)

Example#

>>>

= RESTART: C:/Users/

Number1 = 12

Number2 = 11

Number3 = 14

>>>

output

>>>

random6.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8

File Edit Format Run Options Window Help

```
import random
number1=random.choice([11,12,13,14,15])
number2=random.choice([11,12,13,14,15])
number3=random.choice([11,12,13,14,15])
print("Number1 =",number1)
print("Number2 =",number2)
print("Number3 =",number3)
```

code

4.3.6 statistics Module

The statistics module is inbuilt. It contains various functions related to the mathematical statistics of numeric data.

Mean

Returns the mean of the given data

Syntax#

```
variableName = mean(sequence)
```

Example#

```
>>>
= RESTART: C:/Users/Shyam/AppData/Local/P
File Edit Format Run Options Window
import statistics
x=statistics.mean([10,20,30,40])
print("Mean = ",x)      code
```

Median

Returns the middle value of numeric data in a list.

Syntax#

```
variableName = median(sequence)
```

Example#

```
>>>
= RESTART: C:/User
File Edit Format Run Options Window Help
import statistics
Median = 30
>>>
x=statistics.median([10,20,30,40,50])
print("Median = ",x)      code
```

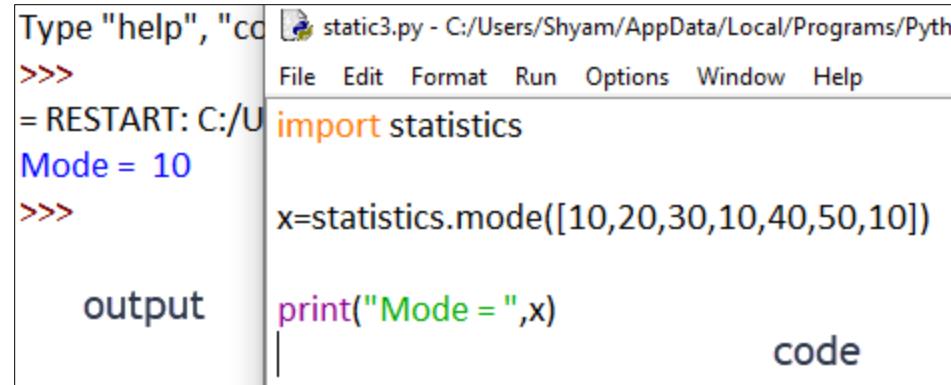
Mode

Returns the most common data point in the list.

Syntax#

```
variableName = mode(sequence)
```

Example#



Type "help", "cc
File Edit Format Run Options Window Help
>>> = RESTART: C:/U import statistics
Mode = 10
>>> x=statistics.mode([10,20,30,10,40,50,10])
output print("Mode = ",x) code

The screenshot shows a Python IDLE window. On the left, there is a text area labeled 'output' containing the command 'print("Mode = ",x)'. To the right of this is another text area labeled 'code' containing the code 'x=statistics.mode([10,20,30,10,40,50,10])'. At the top of the window, there is a menu bar with options: File, Edit, Format, Run, Options, Window, and Help. The status bar at the bottom shows the path 'static3.py - C:/Users/Shyam/AppData/Local/Programs/Python'.

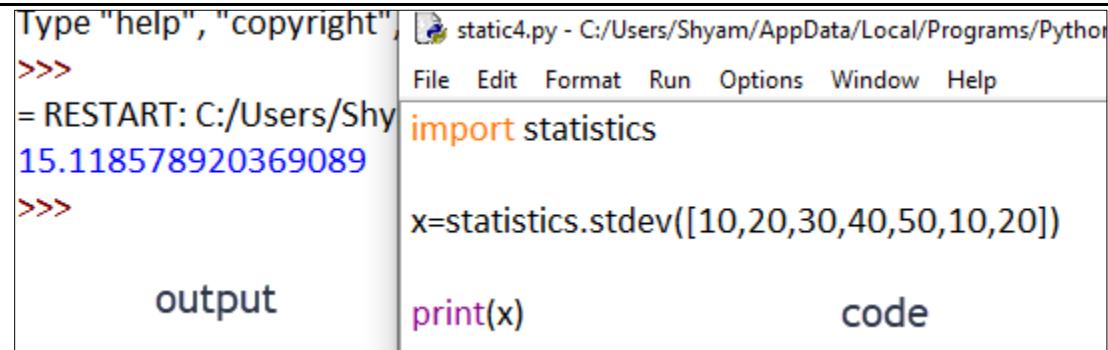
Standard Deviation

Returns standard deviation on a given value in the list.

Syntax#

```
variableName = stdev(sequence)
```

Example#



Type "help", "copyright", File Edit Format Run Options Window Help
>>> = RESTART: C:/Users/Shy 15.118578920369089
>>> import statistics
output x=statistics.stdev([10,20,30,40,50,10,20]) print(x) code

The screenshot shows a Python IDLE window. On the left, there is a text area labeled 'output' containing the command 'print(x)'. To the right of this is another text area labeled 'code' containing the code 'x=statistics.stdev([10,20,30,40,50,10,20])'. At the top of the window, there is a menu bar with options: File, Edit, Format, Run, Options, Window, and Help. The status bar at the bottom shows the path 'static4.py - C:/Users/Shyam/AppData/Local/Programs/Python'.

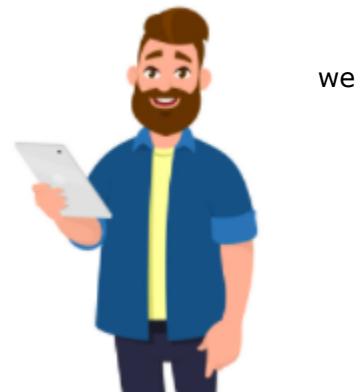
5.1.1 Introduction to Strings, String Operations, Traversing a String

Sequences of characters are referred to as strings. A string is nothing but a combination of characters, numbers, or any special symbols.

A string can be any length. In Python, a string is an immutable sequence data type. Strings are "immutable" which means, once we create a string, we cannot change its value. So whenever we are doing any operation on String, it creates a new string variable in memory.

Strings can be created by enclosing characters inside a single quote or double quotes. We can use triple quotes for multiline strings. Python treats single quotes the same as double-quotes.

There is a built-in class 'str' for handling Python string.



5.1.1.1 Assign String to a Variable

Syntax#

```
variableName="any text"
```

```
variableName='any text'
```

```
variableName="""multiline line1  
line2  
line3"""
```



Example#

<pre> File Edit Format Run Options Window str1 = 'Shyam' print(str1) str2 = "Yash" print(str2) str3 = """Karna""" print(str3) myaddress = """12,Hazar Dastan, Dal Lake, Srinagar""" print(myaddress) code print(type(myaddress)) </pre>	 Python 3.8.5 Shell File Edit Shell Debug Option Python 3.8.5 (tags/v3.8.5:580f8d6, Jul 20 2020, 15:47:08) [MSC v.1920 32 bit (I64)] Type "help", "copyright", "credits" or "license()" for more information. >>> = RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38/python.exe Shyam Yash Karna 12,Hazar Dastan, Dal Lake, Srinagar <class 'str'> >>> output
--	---

5.1.1.2 Input string from a user and assign to a Variable

Syntax#

```
variableName=input("Enter any text")
```

Example#

 Python 3.8.5 (tags/v3.8.5:580f8d6, Jul 20 2020, 15:47:08) [MSC v.1920 32 bit (I64)] Type "help", "copyright", "credits" or "license()" for more information. >>> = RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38/python.exe Enter your first name =>Dhaval Enter your last name =>Shah Welcome Dhaval Shah >>> code	 strdemo3.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38/python.exe File Edit Format Run Options Window Help fname = input("Enter your first name =>") lname = input("Enter your last name =>") print("Welcome",fname,lname) output
---	--

5.1.1.3 Escape Sequence

An escape sequence contains a backslash (\) symbol followed by one of the escape sequence characters. We can use escape sequences with String.

Escape Sequence	Meaning
-----------------	---------

\\	Backslash
\'	Single quotes
\"	Double quotes
\n	Linefeed
\t	Horizontal tab

If we want to use quotes inside Python String then we can use \' or \". For the the-new line, we can use \n , for the tab (5 spaces) \t, and for displaying \\ backslash we need to write \\\.

Example#

```
strdemo2.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/strdemo2.py (3.8.5)
File Edit Format Run Options Window Help
str1 = 'Shyam\\Sir'
print(str1)

str2 = "He\'s good boy"
print(str2)

str3 = 'Karna is living in \"Canada\"'
print(str3)                                     code

myaddress = "My Address is \n12,Hazar Dastan Dal Lake\nSrinagar Kashmir"
print(myaddress)

str4="Vedika\tdo you want to eat IceCream?"
print(str4)
```

```
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 12:10:00) [MSC v.1916 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license" for more information.
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/python.exe
Shyam\Sir
He's good boy
Karna is living in "Canada"
My Address is
12,Hazar Dastan Dal Lake
Srinagar Kashmir
Vedika    do you want to eat IceCream?
>>>
```

output

5.1.1.4 String Length

We can get the length of a string, use the `len()` function.

Example#

```
Type "l" to start a new session
>>>
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/python.exe
25
>>> a = "khidak sing ke khidak ne se"
>>> print(len(a))
25
>>> output
```

code

5.1.2 How to Access the Python String?

There are five ways to access the Python String

1. Simply Store and Print or Assign a value to another variable

```
str="Ram is good"
```

```
print(str)
```

```
str2=str
```

```
print(str2)
```



2. Indexing, We can access individual characters

Square brackets can be used to access elements of the string. We can access individual characters of a string using a numeric index or say position.

Example#

```
a = "Python is Easy"
```

```
print(a[1]) #prints y
```

From the left side first character of a string is at index 0, the second character at index 1, and so on. From right to left, we can access each character starting with -1 then -2, and so on. Positive indexes from left to right and Negative indexes from right to left.

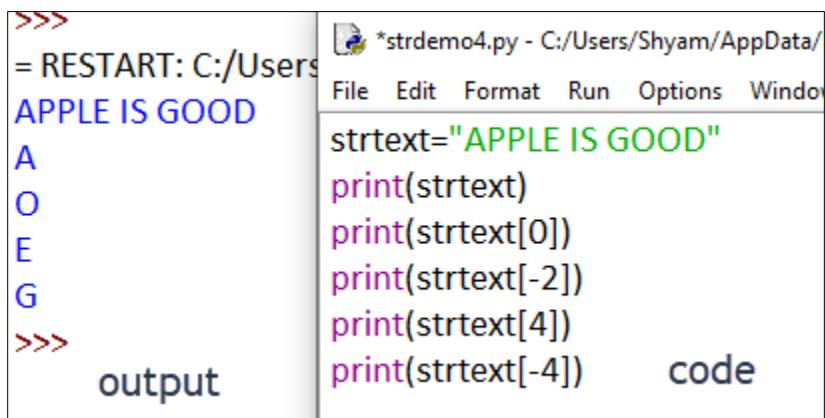
Example# text="APPLE IS GOOD"

A	P	P	L	E		I	S		G	O	O	D
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

We have a total of 14 characters. If we perform indexing left to right in the above Python String, we have the indices from left to right 0 to 12 and from the right side -1 to -13. Even the space and special characters are counted in characters.

Example#

```
>>>
= RESTART: C:/Users
APPLE IS GOOD
A
O
E
G
>>> output
```

A screenshot of a Python code editor window. On the left, the code is shown in a light gray background:
```python  
strtext="APPLE IS GOOD"  
print(strtext)  
print(strtext[0])  
print(strtext[-2])  
print(strtext[4])  
print(strtext[-4])  
```  
On the right, the output is shown in a white background:
```text  
APPLE IS GOOD  
A  
O  
E  
G  
```

3. Slicing, We can fetch Sub String from String

One more step from indexing. Slicing means sub-string from the given string. There are three parts to it.



Syntax#

variableName=String[start,stop,step]

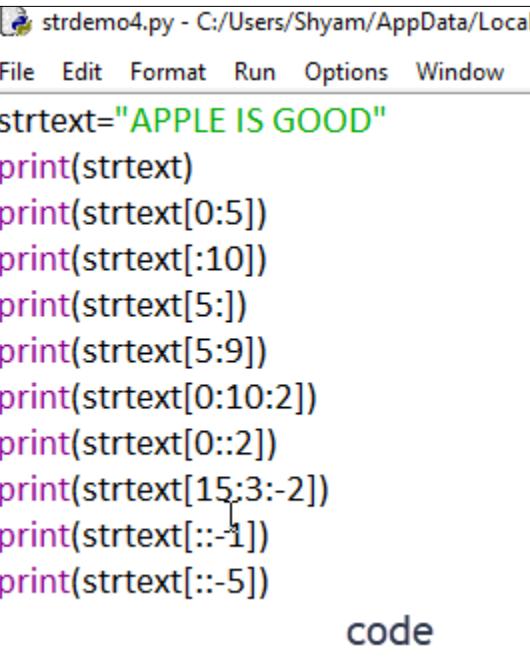
All the arguments are optional.

1. start: Starting index where slice starts. The default value is 0.
2. stop: Ending index where slice stops. The character at this index is not included in the slice. The default value is the length of the string.
3. step: Number of jumps/gaps to take while going from starting to end. It takes the default value of 1.

Let's take the example of: APPLE IS GOOD

A	P	P	L	E		I	S		G	O	O	D
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Example#

<pre>>>> = RESTART: C:/Users APPLE IS GOOD APPLE APPLE IS G IS GOOD IS APEI APEI OD DO IE DOOG SI ELPPA DSP >>></pre>	 <p>File Edit Format Run Options Window</p> <pre>strtext="APPLE IS GOOD" print(strtext) print(strtext[0:5]) print(strtext[:10]) print(strtext[5:]) print(strtext[5:9]) print(strtext[0:10:2]) print(strtext[0::2]) print(strtext[15:3:-2]) print(strtext[::-1]) print(strtext[::-5])</pre>
output	code

4. Looping Through a String/Traversing a String each character

We can access each element from String step by step using a loop.

There are four ways to access using a loop:

Using For in	Using for in
for x in "Mumbai": print(x)	city ="Surat" for x in city: print(x)
Using For Range function	Using For Range function
movie="Kal ho na ho" for i in range(0,len(movie)): print(movie[i])	movie="Kal ho na ho" i=0 while i<len(movie): print(movie[i]) i=i+1

Example# Let's count the number of characters in a String

<pre>>>> = RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python36-32/straemob.py Enter song =>khdak sing ke khdak Enter character to find =>k Cnt = 5 >>> = RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python36-32/straemob.py Enter song =>khdak sing ke khdak Enter character to find =>m Cnt = 0</pre>	<pre>File Edit Format Run Options Window Help song=input("Enter song =>") find=input("Enter character to find =>") cnt=0 for x in song: if x==find: cnt=cnt+1 print("Cnt = ",cnt)</pre> <p style="text-align: right;">code</p>
--	--

Example# Let's count the number of characters in a String

<pre>File Edit Format Run Options Window Help song=input("Enter song =>") find=input("Enter character to find =>") replace=input("Enter character to find =>") for x in song: if x==find: print(replace,end="") else: print(x,end="")</pre>	<p style="text-align: right;">code</p>
---	--

<pre>= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python36-32/straemob.py Enter song =>khdak sing ke khdak ne se Enter character to find =>k Enter character to find =>m mhdam sing me mhdam ne se >>></pre>	<p style="text-align: right;">output</p>
---	--

Example# Let's count the number of character in a String

```
song=input("Enter song =>")  
upper=0  
lower=0  
other=0  
for x in song:  
    if x.isupper():  
        upper+=1  
    elif x.islower():  
        lower+=1  
    else:  
        other+=1  
  
print("Total Upper cases =",upper)  
print("Total Lower cases =",lower)  
print("Total Other cases=",other)|
```

```
= RESTART: C:/Users/Shyam/AppData/Local/Pro  
Enter song =>Zindagi EK Safar HAI Suhana  
Total Upper cases = 8  
Total Lower cases = 15  
Total Other cases= 4  
|>>> |
```

Output

5.1.3 How to modify String?

To Modify Existing String

A string is an immutable data type, we cannot change its value.

Example#

```
>>> title="kuch happy ho jaye"
>>> title[0]='m'
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    title[0]='m'
TypeError: 'str' object does not support item assignment
```



As above, we can't modify any particular character from a given string.

So, what to do for modifying? We have replaced the function.

Syntax#

```
String.replace("OldWord/OldCharacter","NewWord/NewCharacter")
```

Example#

```
>> title="kuch happy ho jaye"
>> title[0]='m'
>> title.replace('k','m')
>> 'much happy ho jaye'
```

To Update String

To update the string we can use + operator.

Example#

```
>>> str1="Rahul"
>>> str2="Shah"
>>> str1=str1+str2
>>> print(str1)
RahulShah
```

Operator with String

In/Not In Operator

In and Not In are known as membership operator.

Returns True or False, In operator check if a certain word or character is present in a string,

Syntax#

"Word" in StringVariable

Example# In Operator

```
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python  
Song is Zindagi Ek Safar Hai Suhana Yahan Kal Kya Hoga.....  
True  
Enter word or character for search =>Life  
Sorry does not exist  
>>>  
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python  
Song is Zindagi Ek Safar Hai Suhana Yahan Kal Kya Hoga.....  
True  
Enter word or character for search =>H  
Yes exist  
>>>|
```

output

```
song="Zindagi Ek Safar Hai Suhana Yahan Kal Kya Hoga....."  
print("Song is ",song)  
print("Ek" in song)  
  
search= input("Enter word or character for search =>")  
  
if search in song:  
    print("Yes exist")  
else:  
    print("Sorry does not exist")
```

code

Returns True or False, Not In operator check if a certain word or character is Not present in a string, does the exact reverse of in operator.

Syntax#

"Word" not in StringVariable

Example# Not In Operator

File Edit Format Run Options Window Help

```
song="Zindagi Ek Safar Hai Suhana Yahan Kal Kya Hoga....."
print("Song is ",song)
print("Ek" not in song)

search= input("Enter word or character for search =>")

if search not in song:
    print("Yes does not exist")
else:
    print("Sorry exist")
```

code

```
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/P
Song is Zindagi Ek Safar Hai Suhana Yahan Kal Kya Hoga.....[1]
False
Enter word or character for search =>m
Yes does not exist
```

output

+ operator – String Concatenation Operator

Returns a concatenated string. + Operator takes two arguments - a string and any value (String or Integer)

Syntax#

String1 + String2

Example# Concatenation of String

```
= RESTART: C:/Users/Shyam/AI
Enter any thing =>Dil Se
Enter any thing =>Kashmir
Dil SeKashmir
Dil Se Kashmir
>>>
```

output

```
strdemo9.py - C:/Users/Shyam/AppData/Local/Program
File Edit Format Run Options Window Help
str1=input("Enter any thing =>")
str2=input("Enter any thing =>")
print(str1+str2)
print(str1+ " " + str2)code
```

* operator – String Replication Operator

Returns a string which is a number of repetitions of the input string like Loop.* * operator takes two operands - a string and an integer.

Syntax#

String1 * IntegerValue (No of times)

Example# Multiplication of String

A screenshot of the Python IDLE interface. On the left, under 'output', the text is:
= RESTART: C:/Users/Shyam/
Enter any thing =>hello
Enter no of time =>5
hellohellohellohellohello
>>>

On the right, under 'code', the code is:
str1=input("Enter any thing =>")
a=int(input("Enter no of time =>"))
print(str1*a)

5.2.1 Strings Methods and Built-in Functions

capitalize()

Returns string with the first character converted to uppercase and all other characters converted to lowercase.

Syntax#

String.capitalize()

Example#

A screenshot of the Python IDLE interface. On the left, under 'output', the text is:
= RESTART: C:/Users/Shyam/AppData/Local/Temp/strdemo10.py - C:/Users/Shyam/AppData/Local/Temp/
Enter any string =>abcd efg hij
Text = abcd efg hij
Text = Abcd efg hij
>>>

On the right, under 'code', the code is:
text=input("Enter any string =>")
print("Text =", text)
print("Text =", text.capitalize())

lower()

Returns all the characters of a string in lowercase. It converts all alphabetic characters to lowercase.

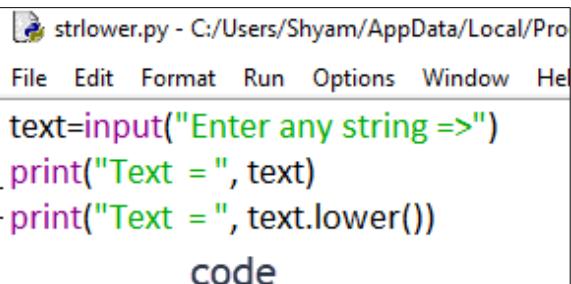
Syntax#

String.lower()

Example#

```
= RESTART: C:/Users/Shyam/AppD  
Enter any string =>AbCD EfG HIj  
Text = AbCD EfG HIj  
Text = abcd efg hij  
>>>
```

output



strlower.py - C:/Users/Shyam/AppData/Local/Pro
File Edit Format Run Options Window Help
text=input("Enter any string =>")
print("Text = ", text)
print("Text = ", text.lower())

code

upper()

Returns all the characters of a string in lowercase. Basically, it converts all alphabetic characters to lowercase.

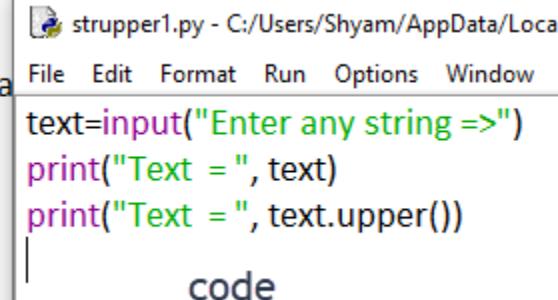
Syntax#

String.lower()

Example#

```
>>>  
= RESTART: C:/Users/Shyam/AppDa  
Enter any string =>AbCD EfG HIj  
Text = AbCD EfG HIj  
Text = ABCD EFG HIJ  
>>>
```

output



strupper1.py - C:/Users/Shyam/AppData/Local/Pro
File Edit Format Run Options Window
text=input("Enter any string =>")
print("Text = ", text)
print("Text = ", text.upper())

code

swapcase()

Swap cases of alphabetic characters.

Returns a string with uppercase alphabetic characters converted to lowercase and lowercase to uppercase.

Syntax#

String.swapcase()

Example#

```
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.9/python.exe strswapcase.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.9/python.exe
Enter any string =>AbC DEF Ghi
Text = AbC DEF Ghi
Text = aBc def gHI
>>>                                         output
                                                code
```

title()

Returns a string in which the first letter of each word is converted to uppercase and the remaining letters are lowercase.

Syntax#

String.title()

Example#

```
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.9/python.exe strtitle.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.9/python.exe
Enter any string =>abcd efG HIJ
Text = abcd efG HIJ
Text = Abcd Efg Hij
>>>                                         output
                                                code
```

count()

Returns the number of times word/character appears in the String. If found then returns several times that word/character occurs else returns 0.

Syntax#

String.count(Word/Character[, [start][, [end]])

1. Word/Character: First Argument which finds.
2. Start: Optional Argument, from where to find.
3. End: Optional Argument, up to find.

Example#

```
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/
Enter any string =>raj is from mumbai
Text = raj is from mumbai
Count of a = 2
Count of d = 0
Count of raj = 1
Count of ('m',5) = 3
Count of ('a',5,10) = 0      output
>>>
DESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/
```

```
strcount.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/
File Edit Format Run Options Window Help
text=input("Enter any string =>")
print("Text = ", text)
print("Count of a = ", text.count("a"))
print("Count of d = ", text.count("d"))
print("Count of raj = ", text.count("raj"))
print("Count of ('m',5) = ", text.count("m",5))
print("Count of ('a',5,10) = ", text.count("a",5,10))
```

find()

Returns the index of the first occurrence of Word/Character in String. Returns -1 if the Word/Character is not present in the string.

Syntax#

String.count(Word/Character[, [start][, [end]])

1. Word/Character: First Argument which finds.
2. Start: Optional argument, from where to find.
3. End: Optional argument, up to find.

Example#

```
= RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/
Enter any string =>raj is from mumbai
Text = raj is from mumbai
find of a = 1
find of d = -1
find of mumbai = 12
find of ('m',5) = 10
find of ('a',5,10) = -1
>>>
output
```

```
strfind.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/
File Edit Format Run Options Window Help
text=input("Enter any string =>")
print("Text = ", text)
print("find of a = ", text.find("a"))
print("find of d = ", text.find("d"))
print("find of mumbai = ", text.find("mumbai"))
print("find of ('m',5) = ", text.find("m",5))
print("find of ('a',5,10) = ", text.find("a",5,10))
```

isalnum()

Returns True/False, True if all the characters in the string are alphabets or numbers only else False even if space or symbols.

Syntax#

String.isalnum()

Example#

```
>>>  
= RESTART: C:/Users/Shyam  
text1 = 1234 abcd ABCD  
IsAlnum = False  
  
text2 = 1234  
IsAlnum = True  
  
text3 = abcd  
IsAlnum = True  
  
text4 = ABCD  
IsAlnum = True  
  
Yes all Alpha num
```

```
>>>
```

output

```
stralnum.py - C:/Users/Shyam/AppData/Local/Programs  
File Edit Format Run Options Window Help  
text1="1234 abcd ABCD"  
text2="1234"  
text3="abcd"  
text4="ABCD"  
print("text1 =", text1)  
print("IsAlnum =", text1.isalnum())  
print("\ntext2 =", text2)  
print("IsAlnum =", text2.isalnum())  
print("\ntext3 =", text3)  
print("IsAlnum =", text3.isalnum())  
print("\ntext4 =", text4)  
print("IsAlnum =", text4.isalnum())  
  
if text4.isalnum(): code  
    print("\nYes all Alpha num")  
else:  
    print("\nSorry not Alphanum")
```

isalpha()

Returns True/False, True if all the characters in the string are alphabets else False.

Syntax#

```
String.isalpha()
```

Example#

```
>>>
= RESTART: C:/Users/Shyam
text1 = 1234 abcd ABCD
IsAlpha = False

text2 = 1234
IsAlpha = False

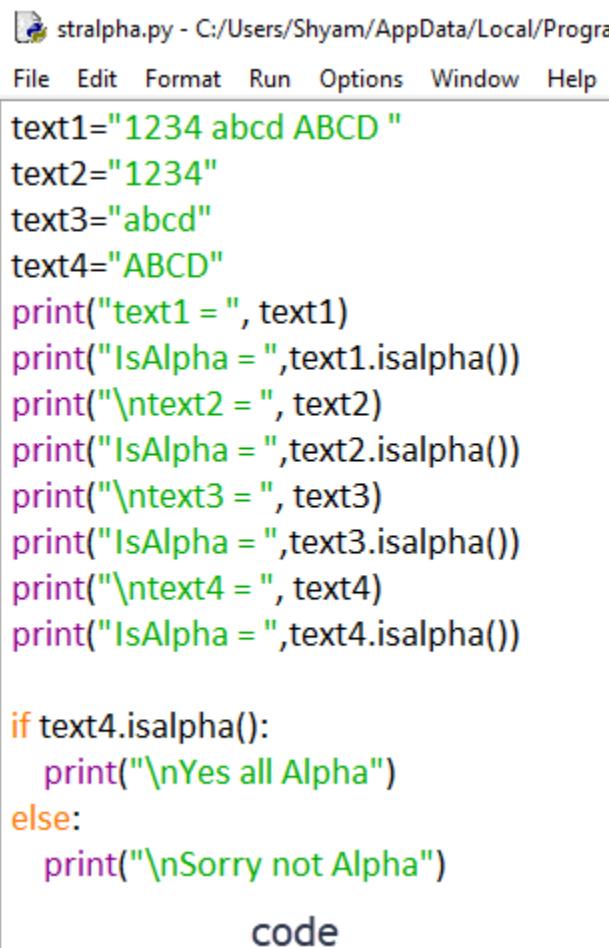
text3 = abcd
IsAlpha = True

text4 = ABCD
IsAlpha = True

Yes all Alpha
```

```
>>>
```

output



```
stralpha.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32
File Edit Format Run Options Window Help
text1="1234 abcd ABCD"
text2="1234"
text3="abcd"
text4="ABCD"
print("text1 = ", text1)
print("IsAlpha = ",text1.isalpha())
print("\ntext2 = ", text2)
print("IsAlpha = ",text2.isalpha())
print("\ntext3 = ", text3)
print("IsAlpha = ",text3.isalpha())
print("\ntext4 = ", text4)
print("IsAlpha = ",text4.isalpha())

if text4.isalpha():
    print("\nYes all Alpha")
else:
    print("\nSorry not Alpha")
```

code

isdigit()

Returns True/False, True if all the characters in the string are numbers else False.

Syntax#

String.isdigit()

Example#

```

Type help, copyright, credits or license for more information.
>>>
= RESTART: C:/Users/Shyam/
text1 = 1234 abcd ABCD
sdigit = False

text2 = 1234
sdigit = True

text3 = abcd
sdigit = False

text4 = ABCD
sdigit = False

Sorry not all Digits
>>>

```

output

```

strisdigit.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32
File Edit Format Run Options Window Help
text1="1234 abcd ABCD"
text2="1234"
text3="abcd"
text4="ABCD"
print("text1 =", text1)
print("isdigit =",text1.isdigit())
print("\ntext2 =", text2)
print("isdigit =",text2.isdigit())
print("\ntext3 =", text3)
print("isdigit =",text3.isdigit())
print("\ntext4 =", text4)
print("isdigit =",text4.isdigit())

if text4.isdigit():
    print("\nYes all Digits")      code I
else:
    print("\nSorry not all Digits")

```

islower()

Returns True/False, True if all the characters in the string are alphabets in lower case else False.

Syntax#

String.islower()

Example#

```
= RESTART: C:/Users/Shya
text1 = 1234 abcd ABCD
islower = False

text2 = 1234
islower = False

text3 = abcd
islower = True

text4 = ABCD
islower = False

Sorry not all Lower
>>>
alpha
```

```
File Edit Format Run Options Window Help
text1="1234 abcd ABCD "
text2="1234"
text3="abcd"
text4="ABCD"
print("text1 = ", text1)
print("islower = ",text1.islower())
print("\ntext2 = ", text2)
print("islower = ",text2.islower())
print("\ntext3 = ", text3)
print("islower = ",text3.islower())
print("\ntext4 = ", text4)
print("islower = ",text4.islower())

if text4.islower():
    print("\nYes all Lower")      code
else:
    print("\nSorry not all Lower")
```

isupper()

Returns True/False, True if all the characters in the string are alphabets in upper case else False.

Syntax#

String.isupper()

Example#

```
>>>  
= RESTART: C:/Users/Shyam/  
text1 = 1234 abcd ABCD  
isupper = False  
  
text2 = 1234  
isupper = False  
  
text3 = abcd  
isupper = False  
  
text4 = ABCD  
isupper = True
```

Yes all Upper

```
>>>
```

output

```
strisupper.py - C:/Users/Shyam/AppData/Local/Program  
File Edit Format Run Options Window Help  
text1="1234 abcd ABCD "  
text2="1234"  
text3="abcd"  
text4="ABCD"  
print("text1 =", text1)  
print("isupper =", text1.isupper())  
print("\ntext2 =", text2)  
print("isupper =", text2.isupper())  
print("\ntext3 =", text3)  
print("isupper =", text3.isupper())  
print("\ntext4 =", text4)  
print("isupper =", text4.isupper())  
  
if text4.isupper():  
    print("\nYes all Upper")      code  
else:  
    print("\nSorry not all Upper")
```

lstrip()

Removes first characters from a string. Default it removes first whitespaces if we don't provide any argument.

Syntax#

```
lstrip([chars])
```

Example#

```
Type "help", "copyright", "d  
>>>  
= RESTART: C:/Users/Shyam  
text1 =      ABCD  
Length = 14  
text1 = ABCD  
Length = 4  
*****  
text2 = $$$$ABCD  
Length = 8  
text2 = ABCD  
Length = 4  
>>>  
          output
```

```
strlstrip.py - C:/Users/Shyam/AppData/Local  
File Edit Format Run Options Window  
text1="      ABCD"  
print("text1 =", text1)  
print("Length =", len(text1))  
text1=text1.lstrip()  
print("text1 =", text1)  
print("Length =", len(text1))  
print("*"*20)           code  
text2="$$$$ABCD"  
print("text2 =", text2)  
print("Length =", len(text2))  
text2=text2.lstrip("$")  
print("text2 =", text2)  
print("Length =", len(text2))
```

rstrip()

Removes last characters from a string. Default it removes last whitespaces if we don't provide any argument.

Syntax#

```
rstrip([chars])
```

Example# Let's remove spaces and \$\$\$ sign

```
Type "help", "copyright"  
>>>  
= RESTART: C:/Users/Shyam/  
text1 = ABCD  
Length = 14  
text1 = ABCD  
Length = 4  
*****  
text2 = ABCD$$$$  
Length = 8  
text2 = ABCD  
Length = 4  
>>>
```

output

```
strstrip.py - C:/Users/Shyam/AppData/Local/Program  
File Edit Format Run Options Window Help  
text1="ABCD      "  
print("text1 = ", text1)  
print("Length = ",len(text1))  
text1=text1.rstrip()  
print("text1 = ", text1)  
print("Length = ",len(text1))  
print("*"*20)  
text2="ABCD$$$$"  
print("text2 = ", text2)  
print("Length = ",len(text2))  
text2=text2.rstrip("$")  
print("text2 = ", text2)      code  
print("Length = ",len(text2))
```

strip()

Removes starting characters from a string. Removes starting and ending whitespaces if we don't provide any argument.

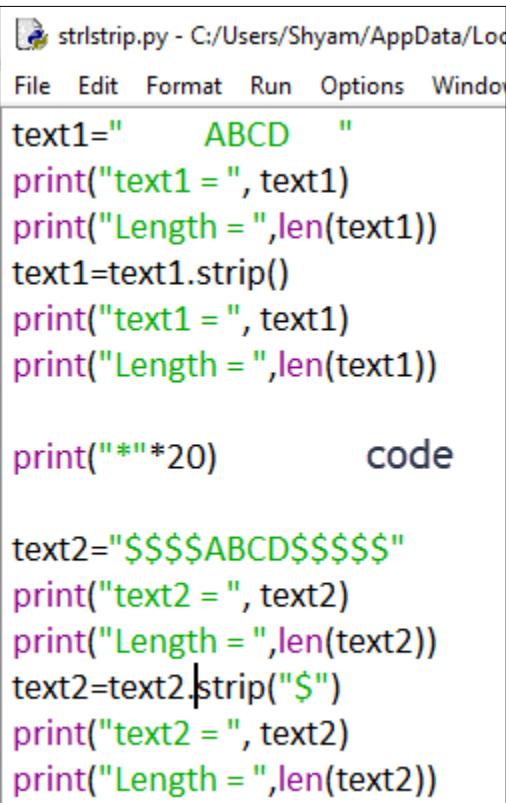
Syntax#

```
strip([<chars>])
```

Example#

```
>>>
= RESTART: C:/Users/Shyam/
text1 =      ABCD
Length = 20
text1 = ABCD
Length = 4
*****
text2 = $$$$ABCD$$$$
Length = 13
text2 = ABCD
Length = 4
>>>
```

output



strlstrip.py - C:/Users/Shyam/AppData/Local/Temp/strlstrip.py

```
File Edit Format Run Options Windows
text1="      ABCD      "
print("text1 =", text1)
print("Length =", len(text1))
text1=text1.strip()
print("text1 =", text1)
print("Length =", len(text1))

print("*"*20)      code

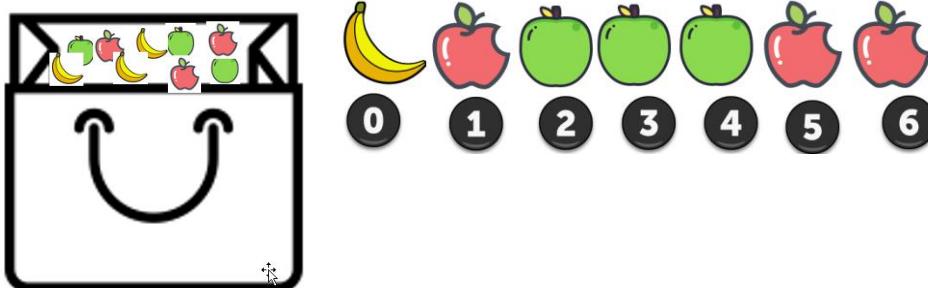
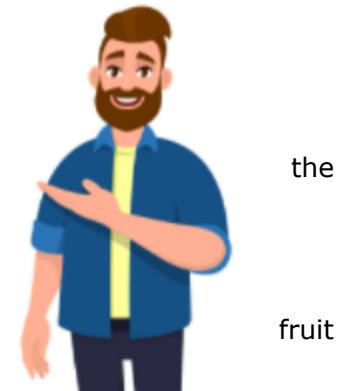
text2="$$$$ABCD$$$$"
print("text2 =", text2)
print("Length =", len(text2))
text2=text2.strip("$")
print("text2 =", text2)
print("Length =", len(text2))
```

5.3.1 Introduction to List and its Operations

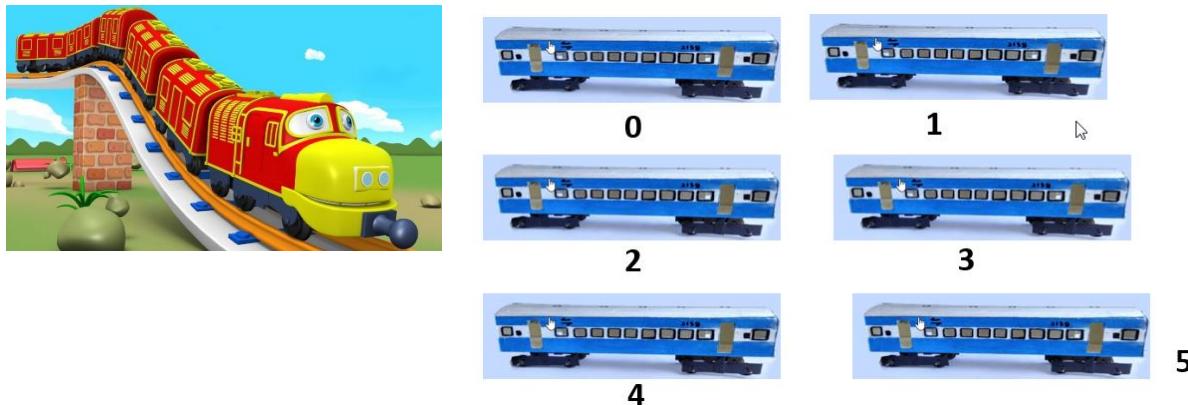
Up to this, we need to store two, three values, so we declared two three variables and stored them, but when we want to store ten-fifteen values then? we need to declare ten-fifteen variables like a,b,c,d,e,f,g,h,i.....etc. or name1,name2,name3,.....etc. But it is difficult to remember all the variables' names, even storing and getting is difficult, code becomes lengthy and confusing. So here the list comes.

Lists are used to store multiple items in a single variable.

It's like a bag where all fruits can be stored together where each has a position.



Similarly, it's like a train where each compartment has a unique number to identify it.



The list is a mutable sequence type. We can store one or more items of different data types. The items in the list are separated with the comma (,) and enclosed with the square brackets []. A list is an ordered collection of items.

Syntax#

```
list1 = []
list2 = [value1,value2,value3,.....]
```

Example#

```
File Edit Format Run Options Window Help
list1=[]

list2=[11,22,33,44,55]
    |
friends=["Ram","Rahul","Anjali","Naiya","Vedika"]

listData=["Ram",25,"Anjali",35,"Vedika"]

print(list1)

print(list2)

print(friends)

print(listData)
|
```

code



```
== RESTART: C:/Users/Shyam/AppData/Loc
[]                                output
[11, 22, 33, 44, 55]
['Ram', 'Rahul', 'Anjali', 'Naiya', 'Vedika']
['Ram', 25, 'Anjali', 35, 'Vedika']
>>> |
```

Don't forget

- Lists are ordered.
- Lists are heterogeneous we can store different types of elements.
- Lists are mutable. We can change values in them.
- We can access each element by its index.
- We can add, delete, search, modify, etc operations on List.
- The list has many inbuilt functions.

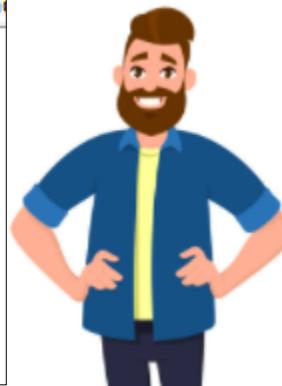


5.3.2 Operations On List

5.3.2.1 Generating List by Range Function

```
File Edit Format Run Options Window
list1=list(range(0,5))
list2=list(range(1, 10))
list3=list(range(1, 20,2))
print(list1)
print(list2)
print(list3)
```

code



[0, 1, 2, 3, 4] output
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

5.3.2.2 Accessing Elements of List

There are many ways to access a list:

Accessing by its index

We can access each element by its position/index. Similar to String. Indexing left to right starts at 0. And Right to left, starts at -1.

Syntax#

ListVariableName[position]

Example#

List1 =[11,"Rahul",45,"Om","Parth"]

11	Rahul	45	Om	Parth
0	1	2	3	4
-5	-4	-3	-2	-1

Example#

```
listData=[11,55,78,90,23,45]

print(listData)

print("0 = ",listData[0])
print("2 = ",listData[2])
print("5 = ",listData[5])

print("-1 = ",listData[-1])
print("-3 = ",listData[-3])
print("-5 = ",listData[-5])
```



Accessing using For loop

We can access each element using for loop.

Syntax#

```
ListVariableName=[Element1,Element2,...]
```

```
for x in ListVariableName:  
    print(x)
```

Example#

A screenshot of a Python terminal window. The title bar says "list2.py - C:/Users/Shyam/AppData/Local/Prog". The code in the terminal is:

```
== RESTART: C:\Users\Shyam\AppData\Local\Prog\list2.py  
11  
55  
78  
90  
23  
45  
output  
>>>
```

The output of the code is:

```
listData=[11,55,78,90,23,45]  
for x in listData:  
    print(x)
```

The word "code" is highlighted in red in the terminal window.

Example#

```
>>> for x in [1,20,45,67,20]:  
    print(x)  
code  
  
1  
20  
45  
67  
67  
20
```

Accessing using For loop with index

We can access each element using for loop with its index

Syntax#

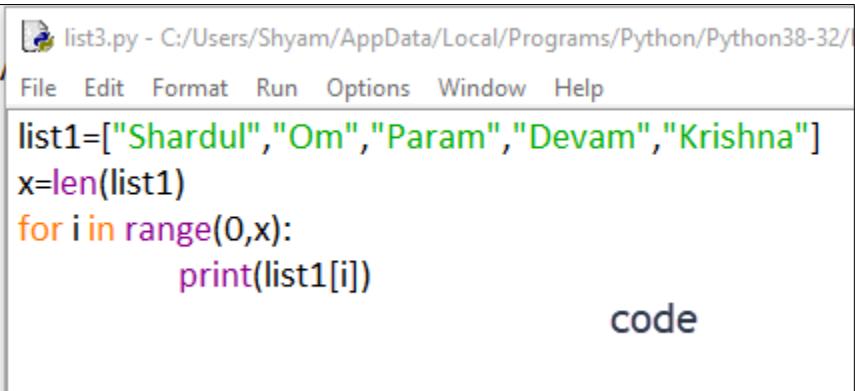
```
ListVariableName=[Element1,Element2,...]
```

```
X=len(ListVariableName)
```

```
for index in range(0,X):  
    ListVariableName[i]
```

Example#

```
>>>  
== RESTART: C:/list3.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python38-32/  
Shardul  
Om  
Param  
Devam  
Krishna  
>>> output
```



```
list1=["Shardul","Om","Param","Devam","Krishna"]  
x=len(list1)  
for i in range(0,x):  
    print(list1[i])
```

code

Accessing using While with index

We can access each element using while with its index.

Syntax#

```
ListVariableName=[Element1,Element2,...]
```

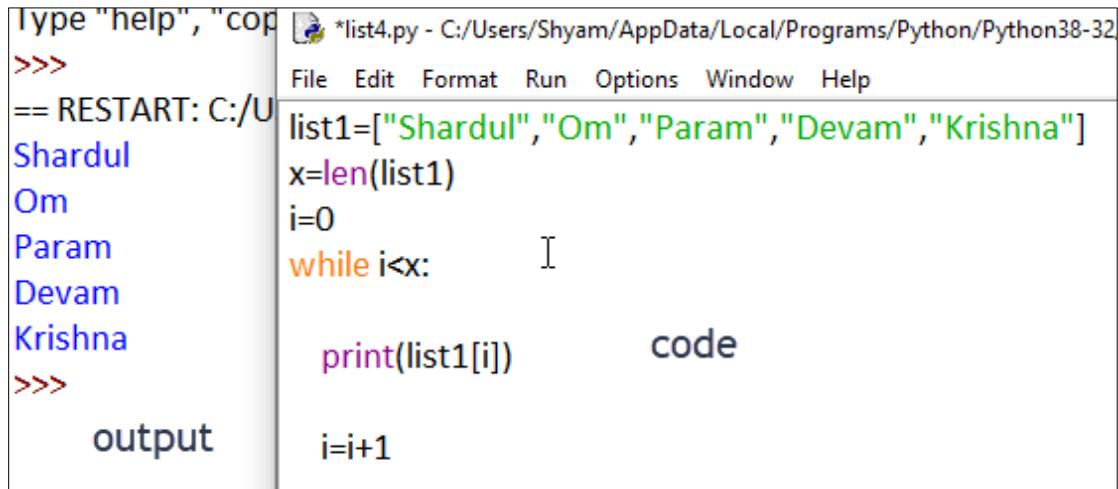
```

X=len(ListVariableName)

index=0
While index<x:
    ListVariableName[index]
    index=index+1

```

Example#



```

Type "help", "cop
>>>
== RESTART: C:/U
Shardul
Om
Param
Devam
Krishna
>>>
      output

```

```

list1=["Shardul","Om","Param","Devam","Krishna"]
x=len(list1)
i=0
while i<x:
    print(list1[i])
    i=i+1

```

code

5.3.2.3 Slicing, We can fetch Sub List from List

One more step from indexing. Slicing means sub-list from the given list. There are three parts to it. (Same as String)

Syntax#

VariableName=List[start,stop,step]

All the arguments are optional.

1. start: Starting index slice starts. The default value is 0.
2. stop Ending index where slice stops. The character at this index is not included in the slice. The default value is the length of the string.
3. step: Number of jumps/gaps to take while going from starting to end. It takes the default value of 1.

Let's take the example of : ['A','P','P','L','E','','I','S','','G','','O','O','D']

A	P	P	L	E		I	S		G	O	O	D
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Example#

```
list1=['A','P','P','L','E',' ','I','S',' ','G','O','O','D']
print(list1)
print(list1[0:5])
print(list1[0:7:2])
print(list1[ : :2])
print(list1[ :10:2])
print(list1[5: :2])                                code
print(list1[-12:-5:2])
print(list1[-12:-1:2])
print(list1[ : :-1])
print(list1[12:5:-2])
print(list1[::])
```

```
['A', 'P', 'P', 'L', 'E', ' ', 'I', 'S', ' ', 'G', 'O', 'O', 'D']
['A', 'P', 'P', 'L', 'E']
['A', 'P', 'E', 'I']
['A', 'P', 'E', 'I', ' ', 'O', 'D']
['A', 'P', 'E', 'I', ' ']
[' ', 'S', 'G', 'O']                               output
['P', 'L', ' ', 'S']
['P', 'L', ' ', 'S', 'G', 'O']
['D', 'O', 'O', 'G', ' ', 'S', 'I', ' ', 'E', 'L', 'P', 'P', 'A']
['D', 'O', 'O', 'I']
['A', 'P', 'P', 'L', 'E', ' ', 'I', 'S', ' ', 'G', 'O', 'O', 'D']
```

5.3.2.4 Concatenating, Repetition, and Membership

+ Operator

It joins one or more lists.

Syntax#

ListVariable1+ListVariable2+ListVariable3

Example#

<pre>>>> == RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/test.py [11, 22, 5, 6, 100, 200] >>></pre>	<p>File Edit Format Run Options Window</p> <pre>list1=[11,22] list2=[5,6] list3=[100,200]</pre>
output	list4=[] code <pre>list4=list1+list2+list3 print(list4)</pre>

* operator

* operator is used to repeating the list a particular number of times.

Syntax#

ListVariableName*IntegerValue

Example#

<pre>>>> == RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/test.py [11, 22, 5, 6, 100, 200] >>></pre>	<p>File Edit Format Run Options</p> <pre>list1=[11,22,33] list2=[] list2=list1*3 print(list1) print("") print(list2)</pre>
output	

In Operator

In is a membership operator. We can check whether the list contains that particular element or not.

Example#

```
list1=["ram","rahul","anjali","manav"]
print("ram" in list1)

print("aa" in list1)
```

Modifying Lists in Python

Lists are mutable. We can change its values.

Syntax#

ListName[index]=value

Example#

The screenshot shows a Python IDLE interface. On the left, under 'output', there is a command-line history:

```
>>>
== RESTART: C:/Users/Shivam/PycharmProjects/PythonProject/venv/test.py
[11, 22, 33, 44]
[11, 22, 'Rahul', 44]
>>>
```

On the right, under 'code', is the corresponding Python code:

```
list1=[11,22,33,44]
print(list1)

list1[2]="Rahul"
print(list1)
```

5.3.2.5 Adding value to the list

There are two ways to add value in the list, one way is manually as above, and the other way is we can use append() function, insert() function, and extend() function.

append()

Using the append function we can add a new value at the end of the list.

Syntax#

ListVariable.append(value)

Example#

```
listData=[11,55,6,77,8,90]
```

```
listData.append(500)  
listData.append(200)
```

```
print(listData)      code
```

```
[11, 55, 6, 77, 8, 90, 500, 200]
```

output

Example#

```
>>>  
== RESTART: C:/Users/Shyam/  
Enter value1 =>10  
Enter value2 =>20  
[11, 55, 6, 77, 8, 90, 10, 20]
```

```
>>>
```

output

```
listData=[11,55,6,77,8,90]
```

```
X=int(input("Enter value1 =>"))  
Y=int(input("Enter value2 =>"))
```

```
listData.append(X)  
listData.append(Y)
```

```
print(listData)
```

code

I

Example#

```
>>>  
== RESTART: C:/Users/Shyam/App  
[19, 17, 15, 4, 17, 18, 8, 18, 7]
```

```
>>>      I
```

output

```
File Edit Format Run Options Window Help  
import random
```

```
list1=[]  
for i in range(1,10):  
    y=random.randint(1,20)  
    list1.append(y)
```

```
print(list1)
```

code

Example#

```
== RESTART: C:/Users/SI
Enter limit =>5
Enter value =>11
Enter value =>20
Enter value =>33
Enter value =>44
Enter value =>500
[11, 20, 33, 44, 500]
>>>      output          code
```

insert()

We can insert values in a list on a specified position. There are two arguments for it, value and position.

Syntax#

```
ListVariableName.insert(position,value)
```

Example#

File Edit Format Run Options Window Help

```
import random
```

```
list1=["rahul","anjali","dev","paro","anurag","prerna"]
```

```
print(list1)
```

```
list1.insert(0,200)
```

```
print(list1)          code
```

```
value=input("Enter value =>")
```

```
list1.insert(3,value)
```

```
print(list1)
```

```
[rahul', 'anjali', 'dev', 'paro', 'anurag', 'prerna']
```

```
[200, 'rahul', 'anjali', 'dev', 'paro', 'anurag', 'prerna']
```

```
Enter value =>600
```

output

[

```
[200, 'rahul', 'anjali', '600', 'dev', 'paro', 'anurag', 'prerna']
```

```
>>>
```

extend()

extend() can add multiple items to a list. We can merge two or more list.

Syntax#

```
ListVariableName.extend(sequence)
```

Example#

```
== RESTART: C:/Users/Shya
Before extend
['rahul', 'anjali']
[11, 22]

After extend
['rahul', 'anjali', 11, 22]
[11, 22]
>>>

output
```

```
list12.py - C:/Users/Shyam/AppData/Local/Prog
File Edit Format Run Options Window Help
import random

list1=["rahul","anjali"]
list2=[11,22]

print("Before extend\n")
print(list1)
print(list2)

print("\nAfter extend\n")
list1.extend(list2)

print(list1)           code
print(list2)
```

5.3.2.6 Deleting Elements

There are three functions for deleting elements in the list.

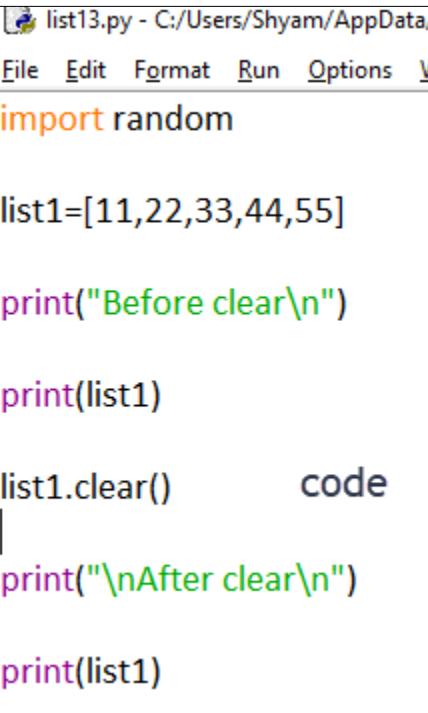
clear()

clear() function removes all the elements of the list.

Syntax#

ListVariableName.clear()

Example#

<pre> == RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/list13.py Before clear [11, 22, 33, 44, 55] After clear [] >>> </pre> <p style="text-align: center;">↔</p> <p style="text-align: center;">output</p>	 <pre> list1=[11,22,33,44,55] print("Before clear\n") print(list1) list1.clear() code print("\nAfter clear\n") print(list1) </pre>
--	---

remove()

remove() function removes the first occurrence of a particular value from the list.

Syntax#

ListVariableName.remove(value)

Example#

```
== RESTART: C:/Users/Shya
Before remove
[11, 22, 33, 44, 55, 11]
After remove
[22, 33, 44, 55, 11]
>>>
output

```

```
File Edit Format Run Options Window Help
import random

list1=[11,22,33,44,55,11]

print("Before remove\n")
print(list1)

list1.remove(11)      code

print("\nAfter remove\n")
print(list1)
```

pop()

In pop() function position is optional argument, Default pop() function removes the last element from the list, and when we pass position it removes an element from that list.

Syntax#

ListVariableName.pop(position/index)

Example#

<pre>>>> == RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/list1.py Before remove [11, 22, 33, 44, 55, 11] After remove Last Element [11, 22, 33, 44, 55] After remove Specific position [11, 22, 33, 55] >>></pre> <p style="text-align: center;">output</p>	<p>File Edit Format Run Options Window Help</p> <pre>import random list1=[11,22,33,44,55,11] print("Before remove\n") print(list1) list1.pop() print("\nAfter remove Last Element\n") print(list1) list1.pop(3) print("\nAfter remove Specific position\n") print(list1)</pre> <p style="text-align: right;">code</p>
---	--

sort()

If we want to sort() list elements then use sort() function.

Syntax#

ListVariableName.sort()

Example#

<pre>>>> == RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/list16.py Before Sort [11, 23, 45, 66, 77, 88, 29, 30] After Sort [11, 23, 29, 30, 45, 66, 77, 88] >>></pre> <p style="text-align: center;">output</p>	<p>File Edit Format Run Options Window Help</p> <pre>list1=[11,23,45,66,77,88,29,30] print("Before Sort ",list1) list1.sort() print("\nAfter Sort ",list1)</pre> <p style="text-align: right;">code</p>
---	--

reverse()

If we want to reverse() all the elements from the list then use reverse() function.

Syntax#

ListVariableName.reverse()

Example#

```
list1=[11,23,45,66,77,88,29,30]

print("Before reverse ",list1)

list1.reverse()           code

print("\nAfter reverse ",list1)

== RESTART: C:/Users/Shyam/AppData/Local/Pro
Before reverse [11, 23, 45, 66, 77, 88, 29, 30]

After reverse [30, 29, 88, 77, 66, 45, 23, 11]
>>> |                                output
```

Example# Descending order of List

```
>>>
== RESTART: C:/Users/Shyam/AP
Before sort and reverse
[11, 23, 45, 66, 77, 88, 29, 30]

After sort and reverse
[88, 77, 66, 45, 30, 29, 23, 11]
>>>
output
```

	<pre>list1=[11,23,45,66,77,88,29,30] print("Before sort and reverse \n",list1) list1.sort() list1.reverse() code print("\nAfter sort and reverse \n",list1)</pre>
--	--

index()

`index()` function is used to find index of any value. It returns first position of particular value, if not found then return error.

Syntax#

```
variableName=listName.index(value)
```

Example#

Type "help", "copyright", "credits" or "license()" for more

```
>>> == RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8/list17.py
At Position = 2
>>>
          list1=[11,23,45,66,77,45,88,29,30]
          x=list1.index(45)           code
output
          print("At Position = ",x)
```

count()

`count()` function is used to count number of occurrence of particular value. It returns number of times of particular value, if not found then return 0.

Example#

```
>>> == RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8/list18.py
Count of 45 is 3
>>>
          list1=[11,23,45,66,45,88,45,30]
          x=list1.count(45)
          print("Count of 45 is ",x)
```

copy()

`copy()` function creates a copy of a list.

Syntax#

```
ListName2=ListName1.copy()
```

Example#

```
type help, copyright, credits or license() for more information.

>>>
== RESTART: C:/Users/Shyam/AppDa
listfruit
['apple', 'banana', 'mango', 'cherry']

list2
['apple', 'banana', 'mango', 'cherry']
>>>

output      I

listfruit=["apple","banana","mango","cherry"]

print("listfruit")
print(listfruit)
list2=listfruit.copy()

print("\nlist2")
print(list2)
```

5.4.1.2 Nested List

A list within a list is known as a nested list

Syntax#

ListName=[elements,[list],[list]]

Example#

```
nums=[11,250,30,33,[40,[45,62,72],81,94,['Mango','Apple']]]

print(nums)      code

[11, 250, 30, 33, [40, [45, 62, 72], 81, 94, ['Mango', 'Apple']]]
```
output
```

### 5.4.1.3 List as Arguments to Function

We can create a user-defined function and pass list as an argument as given below:

```
[11, 22, 33, 44, 55]
>>>
== RESTART: C:/Python34/test/list21.py
Total = 165
>>>
output
def AddFunction(list1):
 total=0
 for x in list1:
 total=total+x
 print("Total = ",total)
list1=[11,22,33,44,55]

AddFunction(list1)
```

code

## 1.1.1.2 Introduction to Lists

Lists are used to store multiple items in a single variable. Similarly, it's like a train where each compartment has a unique number to identify it.

The list is a mutable sequence type. We can store one or more items of different data types. The items in the list are separated with the comma (,) and enclosed with the square brackets []. A list is an ordered collection of items.

### Create a list

#### Syntax#

```
list1 = []
list2 = [value1,value2,value3,.....]
```

#### Example#

```
friends=["Ram", "Anjali", "Rahul", "Om"]
print(friends)
```



### Access the items of List

#### Accessing by its index

We can access each element by its position/index. Similar to String. Indexing left to right starts at 0. And Right to left, starts at -1.

#### Syntax#

ListVariableName[position]

#### Example#

```
List1 =[11,"Rahul",45,"Om","Parth"]
```

|    |       |    |    |       |
|----|-------|----|----|-------|
| 11 | Rahul | 45 | Om | Parth |
| 0  | 1     | 2  | 3  | 4     |
| -5 | -4    | -3 | -2 | -1    |



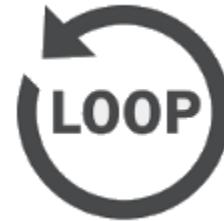
### Accessing using For loop

We can access each element using for loop.

### Syntax#

```
ListVariableName=[Element1,Element2,...]
```

```
for x in ListVariableName:
 print(x)
```



### Accessing using For loop with index

We can access each element using for loop with its index

### Syntax#

```
ListVariableName=[Element1,Element2,...]
X=len(ListVariableName)
```

```
for index in range(0,X):
 ListVariableName[index]
```

### Accessing using While with index

We can access each element using while with its index.

### Syntax#

```
ListVariableName=[Element1,Element2,...]
```

```
x=len(ListVariableName)
```

```
index=0
while index<x:
 ListVariableName[index]
 index=index+1
```



### List Operations

| Operations    | Description                                                                                                                                     | Syntax                                               |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| Concatenation | Join two or more lists                                                                                                                          | List1 + List2                                        |
| Repetition    | repeat the given list to n number of time                                                                                                       | List1 * IntegerValue (No of times)                   |
| Membership    | <b>in</b> operator check if a certain item is present in a list.<br><br><b>not in</b> operator check if a certain item is not present in a list | item in ListVariable<br><br>item not in ListVariable |

|         |                                             |                                     |
|---------|---------------------------------------------|-------------------------------------|
| Slicing | Slicing means sub-list from the given list. | VariableName=List[start ,stop,step] |
|---------|---------------------------------------------|-------------------------------------|

## List Methods and Built-in Functions

| Method  | Description                                                                                                                                             | Syntax                                                 |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------|
| append  | Using the append function, we can add a new value at the end of the list.                                                                               | ListVariable.append(value)                             |
| insert  | We can insert values in a list on a specified position. There are two arguments for it, value and position.                                             | ListVariable.insert(position,value)                    |
| extend  | extend() can add multiple items to a list. We can merge two or more list.                                                                               | ListVariable.extend(sequence)                          |
| clear   | clear() function removes all the elements of the list.                                                                                                  | ListVariable.clear()                                   |
| remove  | remove() function removes the first occurrence of a particular value from the list.                                                                     | ListVariable.remove(value)                             |
| pop     | Default pop() function removes the last element from the list, and when we pass position it removes an element from that list.                          | ListVariable.pop()<br>ListVariable.pop(position/index) |
| sort    | If we want to sort() list elements then use sort() function. (in-place sort)                                                                            | ListVariable.sort()                                    |
| reverse | If we want to reverse() all the elements from the list then use reverse() function.                                                                     | ListVariable.reverse()                                 |
| index   | index() function is used to find index of any value. It returns first position of particular value, if not found then return error.                     | variableName=listName.index(value)                     |
| count   | count() function is used to count number of occurrence of particular value. It returns number of times of particular value, if not found then return 0. | variableName=listName.count(value)                     |
| copy    | copy() function creates a copy of a list.                                                                                                               | ListName2=ListName1.copy()                             |
| min     | Returns minimum or smallest element of the list                                                                                                         | min(list1)                                             |
| max     | Returns maximum or largest element of the list                                                                                                          | max(list1)                                             |
| sum     | Returns sum of the elements of the list                                                                                                                 | sum(list1)                                             |
| sorted  | It takes a list as parameter and creates a new list consisting of the same elements arranged in sorted order                                            | list2 = sorted(list1)                                  |

## Nested List

A list within a list is known as a nested list

### Syntax#

```
ListName=[elements,[list],[list]]
```

### Example#

```
Nums=[11,200,255,[40,50],["Mango","Orange"]]
```

## 1.2.1 Set

Python set has similar concept of mathematical set. A set is an unordered collection of items. Each item in the set must be unique, immutable, and the sets remove the duplicate items.

Sets are mutable which means we can modify it after its creation.

Unlike other collections in Python, there is no index attached to elements of the set, i.e., we cannot directly access any element of the set by the index.

There is a built-in class 'set' for handling Python set.



the  
of

### 1.2.1.1 Creating a set

The set can be created by enclosing the comma-separated immutable items with the curly braces { }.

### Syntax#

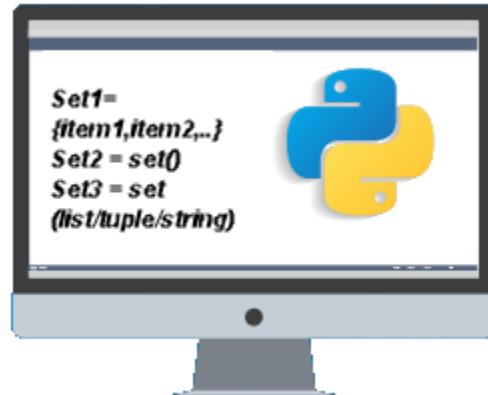
```
Set1={item1,item2,..}
```

Creating an empty set is a bit different because empty curly { } braces are also create a dictionary. So python provides method without an argument to create an empty set.

```
Set2 = set()
```

Creating set using list, tuple, or string as argument.

```
Set3 = set(list/tuple/string)
```



used to  
**set()**  
empty

### Example#

```

set1={10, 20, 70, 50, 50}
set2={9, 10, 20, "yes", "no"}

print("Set1-", set1) code
print("Set2-", set2)

emptySet=set()
print(emptySet)
print(type(emptySet)) creates empty
emptyDictionary={} dict not set
print(type(emptyDictionary))

```

```

= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python39-32/python.exe
set1.py
[Set1- {10, 20, 50, 70}
 Set2- {9, 10, 'no', 20, 'yes'}
 set() output
 <class 'set'>
 <class 'dict'>

```

### Example#

```

studentsList=["Ram", "Om", "Rahul", "Anjali"]

studentsTuple=("Ram", "Om", "Rahul", "Anjali")

studentName="Harsh" code
|_
| print("Using List:", set(studentsList))
| print("Using Tuple:", set(studentsTuple))
| print("Using String:", set(studentName))

```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/3.9/python.exe set2.py
 output
Using List: {'Rahul', 'Om', 'Ram', 'Anjali'}
Using Tuple: {'Rahul', 'Om', 'Ram', 'Anjali'}
Using String: {'s', 'h', 'r', 'a', 'H'}
```

### Example#

Set can contain any type of element such as integer, float, tuple etc. But mutable elements (list, dictionary, set) can't be a member of the set.

```
Creating a set which have immutable elements
set1 = {1,2,3,"Python", 3.14, 143}
print(type(set1)) code

#Creating a set which have mutable element
set2 = {1,2,3,["Python",1]}
print(type(set2)) list is mutable
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/3.9/python.exe set3.py
 output
<class 'set'>
Traceback (most recent call last):
 File "C:/Users/Pradip/AppData/Local/Programs/Python/3.9/python.exe", line 6, in <module>
 set2 = {1,2,3,["Python",1]}
TypeError: unhashable type: 'list'
```

### 1.2.1.2 Accessing Set

Sets in python are unindexed, so we can not access the specific item from it. We can only access all the items together.

Since the sets are unindexed, we cannot use slicing or indexing on sets, it will give us a TypeError error when using indexing.

### Example#

```
students={"Ram", "Om", "Rahul", "Anjali", "Rahul"}
print(students) indexing is not supported code
print(students[0])
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/
set4.py
{'Anjali', 'Ram', 'Om', 'Rahul'} output
Traceback (most recent call last):
 File "C:/Users/Pradip/AppData/Local/Programs/Py
4.py", line 5, in <module>
 print(studentsList[0])
TypeError: 'set' object is not subscriptable
```

### Accessing using For loop

We can access each item using for loop.

#### Syntax#

```
SetVariableName={Element1,Element2,...}
```

```
for x in SetVariableName:
 print(x)
```



#### Example#

```
students={"Ram", "Om", "Rahul", "Anjali", "Rahul"}

print("Set removes duplicate items") code
print(students)
for s in students:
 print(s)
```

```
= RESTART: C:/Users/Pradip/AppData/
set5.py
Set removes duplicate items
{'Anjali', 'Ram', 'Om', 'Rahul'}
Anjali
Ram output
Om
Rahul
```

### 1.2.1.3 Delete from sets

There are four functions for deleting items in the sets.

## discard()

discard() function deletes the specified item from the set. While trying to remove already removed item, discard does not report an error.

### Syntax#

```
set.discard(item)
```

## Example#

```
students={"Ram", "Om", "Rahul", "Anjali"}

students.discard("Rahul") code
print(students)
```

```
= RESTART: C:/Users/Pradip/A
set6.py output
{'Anjali', 'Ram', 'Om'}
```

## remove()

remove() function is similar to discard function but remove function throws an error when the specified item to delete is not found in the set.

### Syntax#

```
set.remove(item)
```

## Example#

```
students={"Ram", "Om", "Rahul", "Anjali"}

students.remove("Rahul") code
print(students)

students.remove("Harsh") Harsh is not in set
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/3.9/Scripts/set7.py
{'Ram', 'Om', 'Anjali'} output
Traceback (most recent call last):
 File "C:/Users/Pradip/AppData/Local/Programs/Python/3.9/Scripts/set7.py", line 6, in <module>
 students.remove("Harsh")
KeyError: 'Harsh'← error
```

## pop()

pop() function doesn't take any argument and it removes an any item from the set and returns the deleted item.

### Syntax#

```
set.pop()
```

## Example#

```
students={"Ram", "Om", "Rahul", "Anjali"}

print("Original set:", students) code
students.pop()
print("Poped set:", students)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/3.9/Scripts/set8.py
{'Ram', 'Om', 'Rahul', 'Anjali'} output
Original set: {'Ram', 'Om', 'Rahul', 'Anjali'}
Poped set: {'Om', 'Rahul', 'Anjali'}
```

## clear()

clear() function removes all the items of the set.

### Syntax#

```
set.clear()
```

## Example#

```
students={"Ram", "Om", "Rahul", "Anjali"}

print("Original set:", students) code
students.clear()
print("Cleared set:", students)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/
thon/Python38/books/set/set9.py output
Original set: {'Anjali', 'Rahul', 'Om', 'Ram'}
Cleared set: set() ← empty set
```

#### 1.2.1.4 Update sets

##### add()

add() method is used to add a single item in a set

###### Syntax#

```
set.add(item)
```

##### Example#

```
fruits={"Apple", "Banana", "Mango"}

fruits.add("Orange") code
fruits.add("Avocado")

print("Updated set:")
print(fruits)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/
thon/Python38/books/set/set10.py output
Updated set:
{'Mango', 'Orange', 'Avocado', 'Apple', 'Banana'}
```

##### update()

update() method is used to add multiple items to the set.

###### Syntax#

```
set.update(items)
```

### Example#

```
fruits={"Apple", "Banana", "Mango"}

fruits.update(["Orange", "Avocado"])

print("Updated set:") code
print(fruits)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python38/books/set/set11.py P
thon/Python38/books/set/set11.py output
Updated set:
{'Apple', 'Avocado', 'Mango', 'Banana', 'Orange'}
```

### 1.2.1.5 Frozen sets

Frozen sets are similar to sets but the only difference is that frozen sets are immutable.

There is a built-in class 'frozenset' for handling Python frozenset.

#### Creating a frozenset

frozenset() inbuilt function is used to create a frozen set. We can pass any iterable like list, set, tuple, etc and it will return an immutable set of class type frozenset.

#### Syntax#

frozenset(iterable)

### Example#

```
immutable_set=frozenset((2,4,2,1,2,3))
print("Frozen set:", end=' ') code
print(immutable_set)
```

```
= RESTART: C:/Users/Pradip/AppData/Python/Python38/books/set/set12.py
Frozen set:frozenset({1, 2, 3, 4})
>>> output
```

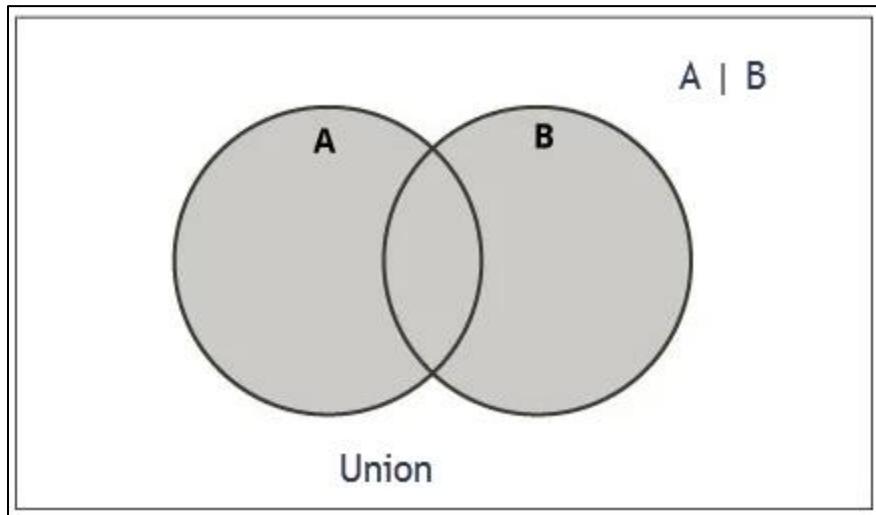
## 1.2.1.6 Python Set Operations

### union()

union() of the two sets which contains all the items that are present in both the sets.

#### Syntax#

```
set1.union(set2)
set1 | set2
```



#### Example#

```
day1=set(["Mon", "Tue", "Wed"])
day2=set(["Wed", "Thu", "Fri", "Sat", "Sun"])

allDays1=day1.union(day2)
print("Using union operation-") code
print(allDays1)

allDays2=day1|day2
print("Using union (|)-")
print(allDays2)
```

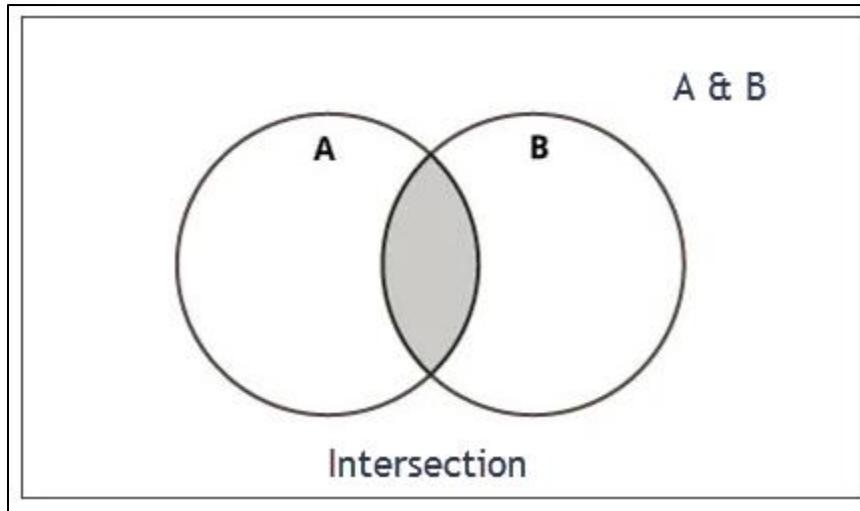
```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/
thon/Python38/books/set/set13.py output
Using union operation-
{'Tue', 'Sun', 'Fri', 'Wed', 'Sat', 'Thu', 'Mon'}
Using union (|)-
{'Tue', 'Sun', 'Fri', 'Wed', 'Sat', 'Thu', 'Mon'}
```

## intersection()

Intersection operation (&) will create a new set with items common to both the sets.

### Syntax#

```
set1.intersection(set2)
set1 & set2
```



## Example#

```
day1=set(["Mon", "Tue", "Wed"])
day2=set(["Wed", "Thu", "Fri", "Sat", "Sun"])

allDays1=day1.intersection(day2)
print("Using intersection operation-")
print(allDays1)

allDays2=day1 & day2
print("Using intersection (&) ")
print(allDays2)
```

code

```
= RESTART: C:/Users/Pradip/AppData/
thon/Python38/books/set/set14.py
Using intersection operation-
{'Wed'}
Using intersection (&) output
{'Wed'}
```

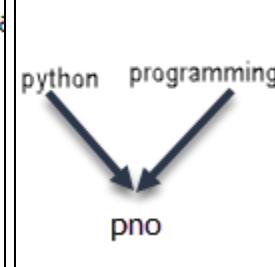
### Example# To Check Common Letters in two input Strings using set

```
s1=input("Enter first string:")
s2=input("Enter second string:")

letters=list(set(s1)&set(s2))

print("The common letters are:")
for i in letters:
 print(i)
```

```
= RESTART: C:/Users/Pradip/AppDa
set19.py
Enter first string:python
Enter second string:programming
The common letters are:
p
n
o
```

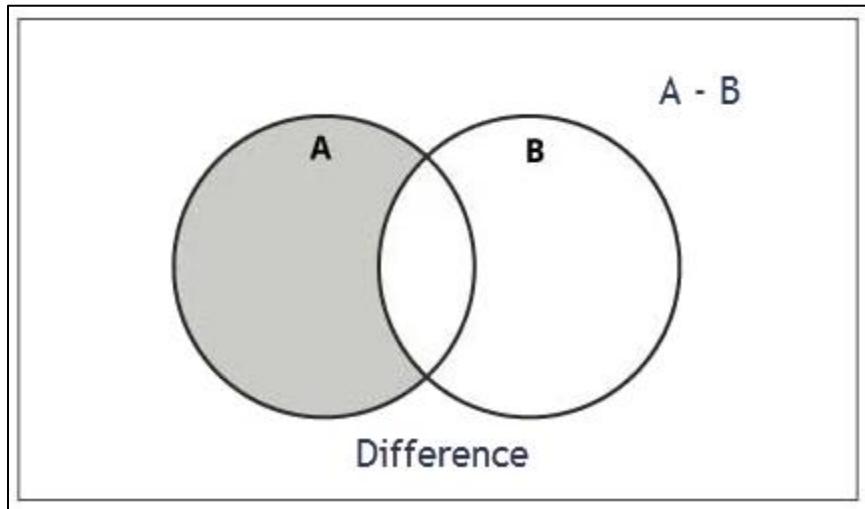


### difference()

Difference operation (-) returns a set that contains the items that only exist in set 1, and not in set 2.

#### Syntax#

```
set1.difference(set2)
set1 - set2
```



### Example#

```

day1=set(["Mon", "Tue", "Wed"])
day2=set(["Wed", "Thu", "Fri", "Sat", "Sun"])

allDays1=day1.difference(day2)
print("Using difference operation-")
print(allDays1) code

allDays2=day1 - day2
print("Using difference (-) ")
print(allDays2)

```

```

= RESTART: C:/Users/Pradip/AppData
thon/Python38/books/set/set15.py
Using difference operation-
{'Tue', 'Mon'}
Using difference (-) output
{'Tue', 'Mon'}

```

### issubset()

The `issubset()` method returns True if all elements of a set 1 are present in another set 2 (passed as an argument). If not, it returns False.

#### Syntax#

```
set1.issubset(set2)
```

## Example#

```
set1={1, 2, 3}
set2={1, 2, 3, 4, 5} code
set3={1, 2, 4, 5}

print(set1.issubset(set2))
print(set2.issubset(set1))
print(set1.issubset(set3))
print(set3.issubset(set2))
```

= RESTART: C:/  
thon/Python38/
True output
False
False
True

## issuperset()

The issuperset() method returns True if a set1 has every elements of another set 2 (passed as an argument). If not, it returns False.

### Syntax#

```
set1.issuperset(set2)
```

## Example#

```
set1={1, 2, 3, 4, 5}
set2={1, 2, 3} code
set3={1, 2, 3}

print(set1.issuperset(set2))
print(set2.issuperset(set1))
print(set3.issuperset(set2))
```

= RESTART: C:/Users/E
thon/Python38/books/s
True output
False
True

## isdisjoint()

Two sets are said to be disjoint sets if they have no common elements.

The isdisjoint() method returns True if two sets are disjoint sets. If not, it returns False.

### Syntax#

```
set1.isdisjoint(set2)
```

## Example#

```

set1={1, 2, 3, 4}
set2={5, 6, 7} code
set3={4, 5, 6}

print('are set1 and set2 disjoint?')
print(set1.isdisjoint(set2))

print('are set1 and set3 disjoint?')
print(set1.isdisjoint(set3))

```

```

= RESTART: C:/Users/Pradip/Ap
thon/Python38/books/set/set18
are set1 and set2 disjoint?
True output
are set1 and set3 disjoint?
False

```

### 1.3.1 Tuple

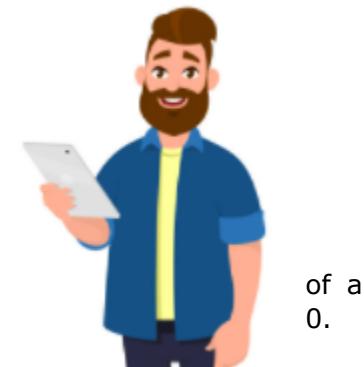
Similar to List, Tuple is used to store multiple items in a single variable. A tuple is an ordered sequence of items of different data types e.g int, float, string, list, or even tuple.

**Tuple is a immutable sequence type.** Tuples are "immutable" which means, once we create a tuple, we cannot change its value.

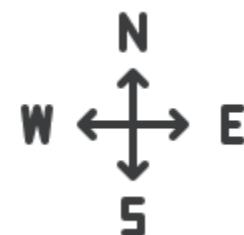
The items in the Tuple are separated with the comma (,) and enclosed with the parenthesis (). Similar to List and String, items tuple can be accessed using index values which are starting from

We can also think of a tuple is like a **constant list** which means once data is assigned to a tuple, it cannot be changed. For example latitude and longitude of home address, directions (North, South, East & West).

There is a built-in class 'tuple' for handling Python Tuple.



of a  
0.



#### 1.3.1.1 Creating Tuples

## Syntax#

Tuple1=()

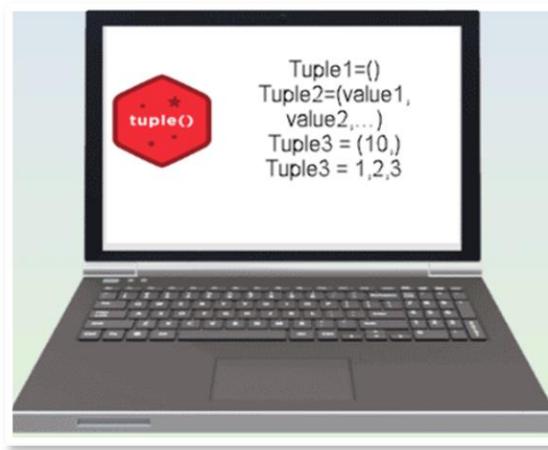
Tuple2=(value1, value2,...)

Creating a tuple with a single item is slightly different. We will need to put a comma after the item to declare the tuple.

Tuple3 = (10,)

A sequence without parenthesis is by default treated as a tuple.

Tuple3 = 1,2,3



## Example#

```
tuple1=()
print(tuple1)
print(type(tuple1)) #type of data

directions = ("North", "South", "East", "West")
print(directions)

tuple_int=(1,2,3) # integer type tuple
print(tuple_int)

tuple_mix=(1,20,"Python",3.8,"Program")
print(tuple_mix)

tuple2=(100,200,300,(400,500))
print(tuple2)

tuple3=(10,20,30,[400,500])
print(tuple3)
```

```
= RESTART: C:\Users\Pradip\AppData
e1.py
()
<class 'tuple'>
('North', 'South', 'East', 'West')
(1, 2, 3)
(1, 20, 'Python', 3.8, 'Program')
(100, 200, 300, (400, 500))
(10, 20, 30, [400, 500])
```

## Example#

```
tuple3=(10)
print(tuple3) code
print(type(tuple3))

tuple4 = (10,)
print(tuple4)
print(type(tuple4))

tuple5 = 1,2,3
print(tuple5)
print(type(tuple5))
```

```
= RESTART: C:\Us
e2.py
10 output
<class 'int'>
(10,)
<class 'tuple'>
(1, 2, 3)
<class 'tuple'>
```

### 1.3.1.2 Accessing Tuple

There are many ways to access a tuple:

#### Accessing by its index

We can access each item by its position/index. Similar to String and List. Indexing left to right starts at 0. And Right to left, starts at -1.

#### Syntax#

`TupleVariableName[position]`

## Example#

`Tuple1 =(11,"Ram",45,"Shyam","Mohan")`

|                           |    |     |    |       |       |
|---------------------------|----|-----|----|-------|-------|
|                           | 11 | Ram | 45 | Shyam | Mohan |
| <b>Positive<br/>Index</b> | 0  | 1   | 2  | 3     | 4     |

|                       |    |    |    |    |    |
|-----------------------|----|----|----|----|----|
| <b>Negative Index</b> | -5 | -4 | -3 | -2 | -1 |
|-----------------------|----|----|----|----|----|

### Example#

```
tuple1=(1, 3, 5, 7, 9, 11) = RESTART: C:\e3.py
print(tuple1[0]) code 1
print(tuple1[4]) 9 output
print(tuple1[1+2])
print(tuple1[-1])
```

### Accessing using For loop

We can access each item using for loop.

#### Syntax#

```
TupleVariableName=(Element1,Element2,...)
for x in TupleVariableName:
 print(x)
```



### Example#

```
tuple=(0, 2, 4, 6, 8, 10) = RESTART: C:\tuple4.py
for index in tuple:
 print(index) code 0
 output
 2
 4
 6
 8
 10
```

### Accessing using For loop with index

We can access each item using for loop with its index

### Syntax#

```
TupleVariableName =(Element1,Element2,...)
```

```
X=len(TupleVariableName)
```

```
for index in range(0,X):
 TupleVariableName[index]
```



### Example#

|                                                                                                                                                   |                                                                                  |
|---------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| <pre>tuple1= ("Om", "Harsh", "Anushka",\<br/>        , "Deval")<br/>x=len(tuple1)<br/>for index in range(0,x):<br/>    print(tuple1[index])</pre> | <pre>= RESTART: C:\Us<br/>tuple6.py<br/>Om<br/>Harsh<br/>Anushka<br/>Deval</pre> |
|---------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|

### Accessing using While with index

We can access each item using while with its index.

### Syntax#

```
TupleVariableName=(Element1,Element2,...)
```

```
x=len(TupleVariableName)
```

```
index=0
while index<x:
 TupleVariableName[index]
 index=index+1
```



### Example#

|                                                                                                                                                                                   |                                                                             |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| <pre>tuple1= ("Om", "Harsh", "Anushka",\<br/>        , "Deval")<br/>x=len(tuple1)<br/><br/>index=0<br/>while index&lt;x:<br/>    print(tuple1[index])<br/>    index=index+1</pre> | <pre>= RESTART:<br/>\tuple7.py<br/>Om<br/>Harsh<br/>Anushka<br/>Deval</pre> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|

### 1.3.1.3 Slicing, We can fetch Sub Tuple from Tuple

One more step from indexing. Slicing means to take some part from the tuple. We use the slicing operator([]). There are three parts to it. (Same as String and List)

#### Syntax#

TupleVariableName[start:stop:step]

All the arguments are optional.



1. start: Starting index slice starts. The default value is 0.
2. stop: Ending index where slice stops. The character at this index is not included in the slice. The default value is the length of the string.
3. step: Number of jumps/gaps to take while going from starting to end. It takes the default value of 1.

**Let's take the example of : ('A','P','P','L','E',' ','I','S',' ','G','O','O','D')**

|                       |     |     |     |     |    |    |    |    |    |    |    |    |    |
|-----------------------|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|
|                       | A   | P   | P   | L   | E  |    | I  | S  |    | G  | O  | O  | D  |
| <b>Positive Index</b> | 0   | 1   | 2   | 3   | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 |
| <b>Negative Index</b> | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

#### Example#

```
tuple1=('A','P','P','L','E',' ','I','S',' ','G','O','O','D')
print(tuple1)

print(tuple1[0:5])
print(tuple1[0:7:2])
print(tuple1[::-2])
print(tuple1[:10:2])
print(tuple1[5::-2])
print(tuple1[-12:-5:2])
print(tuple1[-12:-1:2])
print(tuple1[::-1])
print(tuple1[12:5:-2])
print(tuple1[:])
```

code

```
= RESTART: C:\Users\Pradip\AppData\Local\Programs\Python\Python38
\tuple8.py
('A', 'P', 'P', 'L', 'E', ' ', 'I', 'S', ' ', 'G', 'O', 'O', 'D')
('A', 'P', 'P', 'L', 'E')
('A', 'P', 'E', 'I')
('A', 'P', 'E', 'I', ' ', 'O', 'D') output
('A', 'P', 'E', 'I', ' ')
(' ', 'S', 'G', 'O')
('P', 'L', ' ', 'S')
('P', 'L', ' ', 'S', 'G', 'O')
('D', 'O', 'O', 'G', ' ', 'S', 'I', ' ', 'E', 'L', 'P', 'P', 'A')
('D', 'O', ' ', 'I')
('A', 'P', 'P', 'L', 'E', ' ', 'I', 'S', ' ', 'G', 'O', 'O', 'D')
```

### 1.3.1.4 Tuples are Immutable

Tuples are immutable which means items of a tuple can not be changed once it has created. If we want to change the values, we have to create a new tuple.

Let's try changing a tuple value and see what error we can see.

#### Example#

```
>>> tuple1=(10,20,30,40,50)
>>> tuple1[4]=5
Traceback (most recent call last):
 File "<pyshell#1>", line 1, in <module>
 tuple1[4]=5
TypeError: 'tuple' object does not support item assignment
```

As we see in the above example, a tuple1 is declared with some items. Now if we are trying to change the value at 4<sup>th</sup> index "50" to "5" using the assignment operator, the interpreter generates an error.

Since tuples are immutable but there is a special case when a tuple is containing a list inside it. Lists are mutable so we can change the list value inside the tuples.

#### Example#

```
tup=('a','e',[1, 2, 3],'i','o','u')
print(tup)
tup[2][0]=10 code
print(tup)

= RESTART: C:/Users/Pradip/AppData/Lo
yton/Python38/books/tuple22.py output
('a', 'e', [1, 2, 3], 'i', 'o', 'u')
('a', 'e', [10, 2, 3], 'i', 'o', 'u')
```

### 1.3.1.5 Python Tuple Operations

#### Concatenation (+)

The concatenation operator works the same as the list. This operator joins the tuples. This is done by the + operator in Python.

```
tuple1=(0,2,4,6,8,10)
tuple2=(1,3,5,7,9) code

tuple3=tuple1+tuple2
print(tuple3)

= RESTART: C:/Users/Pradip/AppData/I
e10.py
(0, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9) output
...
```

#### Example#

Write a program to input n numbers from user and store it in tuple. Print the odd number from these tuple.

```

numbers = tuple()
n=int(input("How many numbers you want to enter?: "))

for i in range(0,n):
 num = int(input())
 numbers = numbers +(num,)

print('The numbers in the tuple are:')
print(numbers)

oddList = list()
for i in numbers:
 if i % 2 != 0:
 oddList.append(i)

print('The Odd numbers in the tuple are:')
print(tuple(oddList))

```

```

= RESTART: C:/Users/Pradip/AppData/Local
e24.py
How many numbers you want to enter?: 5
10
52
4
37
85
The numbers in the tuple are:
(10, 52, 4, 37, 85)
The Odd numbers in the tuple are:
(37, 85)

```

## Repetition (\*)

The repetition (\*) operator is used to repeat the tuple a particular number of times.

### Syntax#

TupleVariableName \* IntegerValue

### Example#

```

tuple1= ('Hello', 'World!')
print(tuple1)
tuple2=()
code

tuple2=tuple1*2
print(tuple2)

= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python37-32/test/e11.py
('Hello', 'World!') output
('Hello', 'World!', 'Hello', 'World!')

```

### Membership (in, not in)

The in operator checks if the item in the tuple is present or not. It gives a Boolean output, that is TRUE or FALSE. If the given item exists in the tuple, it gives the TRUE as output, otherwise FALSE

The not in operator returns True if the item is not present in the tuple, else it returns False.

```

tuple1 = ('Olive', 'Pink', 'Yellow')
print('Pink' in tuple1) = RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python37-32/test/e12.py
print('Blue' in tuple1) True
| False
print('Blue' not in tuple1) output
print('Yellow' not in tuple1) True
code False

```

### 1.3.1.6 Tuple Methods and Built-in functions

Since tuples are immutable, we don't have methods to add or remove items.

#### count()

count() method is used to count a number of occurrences of a particular value. It returns number of times of particular value, if not found then return 0.

#### Syntax#

variableName=tupleName.count(item)

## Example#

```
tuple1=(10,20,30,40,50,30)

total=tuple1.count(30)
print(total)
|
total1=tuple1.count(70)
print(total1)
```

```
=|RESTART: C:/Users/ellie/Desktop/e13.py
2
0
```

## output

**index()**

`index()` function is used to find index of any value. It returns the first position of a particular value, if not found then returns an error.

## Syntax#

```
variableName=tupleName.index(value)
```

## Example#

```
tuple1=(10,20,30,40,50,30)

index=tuple1.index(30)
print("Item found at index")
= RESTART: C:/Users/Pra
e14.py
Item found at index: 2
```

code

## output

len0

`len()` function is used to find the total number of items available in tuple.

## Syntax#

variableName=len(tuple)

## Example#

```
tuple1=(1,2,3,4,5)
print("Length of tuple", len(tuple1)) code
= RESTART: C:/Users/
e15.py output
Length of tuple 5
```

## tuple()

tuple() function is used to create an empty tuple if no argument is passed. If we pass sequence to argument of tuple, it converts into an immutable tuple.

### Syntax#

```
variableName=tuple()
```

```
variableName=tuple(sequence)
```

## Example#

```
tuple1=tuple()
print(tuple1) code

tuple1=tuple('Python')#string to tuple
print(tuple1)

tuple2=tuple([1,2,3])#list to tuple
print(tuple2)

tuple3=tuple(range(5))
print(tuple3)
= RESTART: C:/Users/Pradip/AppD
ython/Python38/books/tuple16.py
() output
('P', 'y', 't', 'h', 'o', 'n')
(1, 2, 3)
(0, 1, 2, 3, 4)
```

## min() / max() / sum()

min() function is used to find smallest item from the tuple.

`max()` function is used to find largest item from the tuple.  
`sum()` function is used to find sum of the items of the tuple.

#### Syntax#

```
min(tuple)
max(tuple)
sum(tuple)
```

#### Example#

```
tuple1=(10,12,46,38,91,87,24)

print("Minimum item", min(tuple1)) code
print("Maximum item", max(tuple1))
print("Sum of items", sum(tuple1))

= RESTART: C:/Users/Pradip/AppDε
ython/Python38/books/tuple17.py
Minimum item 10
Maximum item 91 output
Sum of items 308
```

#### sorted()

`sorted()` function is used to sort the items in ascending order. It takes a tuple as an argument and returns a list of sorted items. To sort items in descending order, pass `reverse=True` to the `sorted()` function.

#### Syntax#

```
sorted(tuple)

sorted(tuple, reverse=True)
```

#### Example#

```
tuple1=(2,5,8,1,91,13,77)
colors=("Pink","Yellow","Purple","Red","Green")

print(sorted(tuple1)) code
print(sorted(colors))
|
|= RESTART: C:/Users/Pradip/AppData/Local/Prog
ython/Python38/books/tuple18.py output
[1, 2, 5, 8, 13, 77, 91]
['Green', 'Pink', 'Purple', 'Red', 'Yellow']
...
```

### 1.3.1.7 Tuple unpacking

Unpacking a tuple means splitting the tuple's items into individual variables.

#### Example#

```
student=("Harsh","IT","LECP morbi")

(name,branch,college_name)=student
|
print(name) code |= RESTART: C:/Users/P
print(branch) = python/Python38/books/
print(college_name) Harsh output
 IT
 LEC morbi
```

#### Example#

Swap two numbers without using a temporary variable.

```
num1=int(input('Enter the first number: '))
num2=int(input('Enter the second number: '))
print("\nNumbers before swapping:")
print("First Number:",num1)
print("Second Number:",num2) code
(num1,num2)=(num2,num1)
print("\nNumbers after swapping:")
print("First Number:",num1)
print("Second Number:",num2)

Enter the first number: 25
Enter the second number: 52 output
```

Numbers before swapping:

First Number: 25  
Second Number: 52

Numbers after swapping:

First Number: 52  
Second Number: 25

### Example#

Compute the area and circumference of a circle using a function.

```
def circle(radius):
 circumference=2 * 3.14 * radius code
 area=3.14 * radius * radius
 return (circumference,area)

radius=int(input('Enter radius of circle: '))
circumference,area = circle(radius)
print('Circumference of circle is:',circumference)
print('Area of circle is:',area)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/e23.py
Enter radius of circle: 3 output
Circumference of circle is: 18.84
Area of circle is: 28.259999999999998
```

### 1.3.1.8 Tuple deleting

Using the del keyword, we can delete the whole tuple.

#### Syntax#

```
TupleName = (item1, item2,...n)
del TupleName
```

#### Example#

```
tup = ('physics', 'chemistry', 'maths')
print(tup)
del tup
print("After deleting tuple : ")
print(tup)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/tuple_delete.py
('physics', 'chemistry', 'maths')
After deleting tuple : output
Traceback (most recent call last):
 File "C:/Users/Pradip/AppData/Local/Pr
le_delete.py", line 7, in <module>
 print(tup)
NameError: name 'tup' is not defined
```

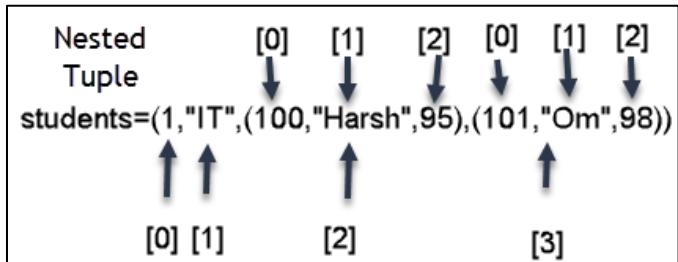
### 1.3.1.9 Nested Tuple

A tuple inside another tuple is called a nested tuple.

#### Syntax#

```
TupleName=(items,(tuple))
```

We can access the nested tuple using TupleName[index1][index2], where index1 is index of outer tuple and index2 is index of nested tuple.



### Example#

```
students=(1, "IT", (100, "Harsh", 95), (101, "Om", 98))

print(students) code
print("Student Name:", students[2][1])
|
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python38/books/tuple20.py
(1, 'IT', (100, 'Harsh', 95), (101, 'Om', 98))
Student Name: Harsh output
```

### 1.3.1.10 List vs Tuple

| List                                                                  | Tuple                                                                    |
|-----------------------------------------------------------------------|--------------------------------------------------------------------------|
| Lists are mutable                                                     | Tuples are immutable                                                     |
| Lists consume more memory                                             | Tuple consume less memory as compared to the list                        |
| Square bracket[] is used to represent list                            | Parenthesis() is used to represent tuple                                 |
| List are slower than Tuple                                            | Tuples are faster than lists because they have a constant set of values. |
| Example: list_num = [10, 20, 30, 40]                                  | Example: tup_num = (10, 20, 30, 40)                                      |
| If the data is not fixed and keep on changing then we should use List | If the data is fixed and never changes then we should use tuple          |

### 1.4.1 Dictionary

Like a real-life dictionary has words and meanings, Python dictionaries have keys and

The Python dictionary is an unordered collection items. List and tuple data types can hold only a value as an item but the dictionary holds **key:value** pair.

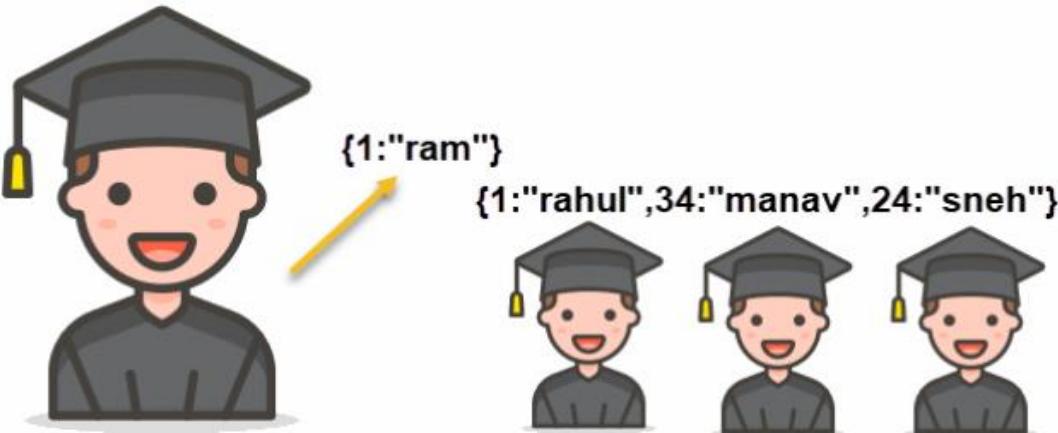


values.

of single

Dictionaries are written with curly brackets {} and have keys and values. Key and its value are separated by colons(:). Key-value pairs are separated by commas(,). Keys in dictionaries are unique and immutable data types(number, string, tuple).

There is a built-in class 'dict' for handling Python Dictionary.



#### 1.4.1.1 Creating Dictionaries

##### Syntax#

###### Empty dictionary:

```
dict1={}
```

**Empty dictionary created using built-in function dic()**

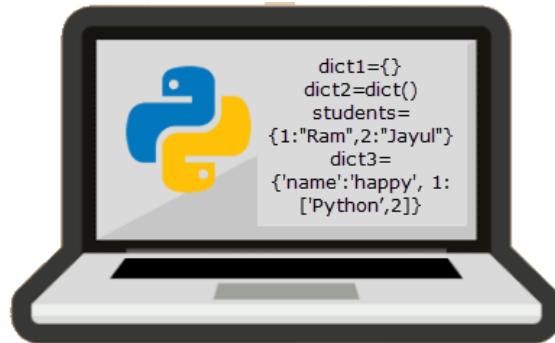
```
dict2=dict()
```

###### Dictionary with integer keys

```
students={1:"Ram", 2:"Jayul", 3:"Rahul",4:"Anjali",5:"Riya"}
```

###### Dictionary with mixed keys

```
dict3={'name':'happy', 1:['Python',2]}
```



##### Example#

```

dict1={}
print(dict1)

dict2=dict()
print(dict2)

students={1:"Ram", 2:"Jayul", 3:"Rahul", 4:"Anjali"}
print(students)
|
dict3={'name':'happy', 1:['Python',2]}
print(dict3)

print(type(dict3))

```

```

= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python38/books/dic/dic1.py
{} output
{}
{1: 'Ram', 2: 'Jayul', 3: 'Rahul', 4: 'Anjali'}
{'name': 'happy', 1: ['Python', 2]}
<class 'dict'>

```

### Example#

```

students={}
students[1]="Ram"
students[22]="Jayul"
students[3]="Rahul"
students[44]="Anjali"
print(students) code

```

```

= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python38/books/dic/dic2.py
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali'} I output
>>> Ln: 22 Co

```

### 1.4.1.2 Accessing Items in Python Dictionaries

There are many ways two to access the dictionary once it is created.

#### Accessing by its key

We can access each item by its key. In sequence data types, items can be accessed using indices and that is the only type of integer but in the dictionary, we can access value using key enclosed in square brackets.

### Syntax#

DictVariableName[key]

### Example#

Accessing value using key in square bracket.

```
>>> students={1:"Ram", 22:"Jayul", 3:"Rahul", 44:"Anjali"}
>>> print(students[22])
Jayul
```

When the key is not found in the dictionary, it returns KeyError

```
>>> students={1:"Ram", 22:"Jayul", 3:"Rahul", 44:"Anjali"}
>>> print(students[55])
[Traceback (most recent call last):
 File "<pyshell#8>", line 1, in <module>
 print(students[55])
KeyError: 55
```

### Accessing by get() method

get() method takes an argument as a key in order to access the value. Get method will return None if the key does not exist in the dictionary.

### Syntax#

DictVariableName.get(key)

### Example#

```
>>> students={1:"Ram", 22:"Jayul", 3:"Rahul", 44:"Anjali"}
>>> print(students.get(44))
Anjali
>>> print(students.get(55))
None
```

## 1.4.1.3 Operations in Dictionary

### Traversing

Like string and list, we can traverse a dictionary using for loop.

### Example#

```
students={1:"Ram", 22:"Jayul", 3:"Rahul", 44:"Anjali"}
|
| for key in students:
| print(key, ":", students[key])
|
|
```

```
= RESTART: C:/Users/Pradip/AppDa
dic4.py
1 : Ram
22 : Jayul
3 : Rahul
44 : Anjali
```

### Example#

Write a program to count the number of times a character appears in a given string using the dictionary.

```
st=input("Enter a string: ")
dic={}

for ch in st:
 if ch in dic:
 dic[ch] += 1
 else:
 dic[ch] = 1

for key in dic:
 print(key, ':', dic[key])
```



```
= RESTART: C:/Users/Pradip/App
dic5.py
Enter a string: good morning
g : 2
o : 3 code
d : 1
 : 1
m : 1
r : 1
n : 2
i : 1
```

### Example#

```
students={1:"Ram", 22:"Jayul", 3:"Rahul",44:"Anjali"}

for key,value in students.items(): code
 print(key,':',value)
```

```
= RESTART: C:/Users/Pradip/App
dic5.py
1 : Ram output
22 : Jayul
3 : Rahul
44 : Anjali
```

### Membership

The membership operator **in** checks if the key is present in the dictionary and returns True, else it returns False.

The **not in** operator returns True if the key is not present in the dictionary, else it returns False.

```

students={1:"Ram", 22:"Jayul", 3:"Rahul",44:"Anjali"}

membership in operator
print(1 in students)
print(5 in students) code

membership not in operator
print(5 not in students)
print(1 not in students)| I

```

= RESTART: C:/t  
dic/dic6.py  
True  
False output  
True  
False

#### 1.4.1.4 Properties of Dictionary Keys

Two important points to be kept in mind about keys are as below:

1. Duplicate keys are not allowed in the dictionary and they must be unique. Whenever a new value is assigned with duplicate keys in a dictionary, the latest value is considered for that key and previous value will be lost.

##### #Example

```

marks={1:22,2:33,3:16,4:39,5:45,4:45}

print(marks)

```

= RESTART: C:/Users/Pradip/AppData/  
dic/dic7.py  
{1: 22, 2: 33, 3: 16, 4: 45, 5: 45}

Duplicate key '4'

2. Keys are immutable which means we can use string, integers, or tuples for dictionary keys, but something like list - ['key'] is not valid.

##### #Example

```

list data-type as key
>>> students={[5]:"Anjali",1:"Ram", 22:"Jayul"}
Traceback (most recent call last):
File "<pyshell#0>", line 1, in <module>
 students={[5]:"Anjali",1:"Ram", 22:"Jayul"}
TypeError: unhashable type: 'list'

```

#### 1.4.1.5 Built-In Dictionary Methods and Functions

##### Adding a value to the Dictionary

We can add a new item to the dictionary.

```
students={1:"Ram",22:"Jayul",3:"Rahul",44:"Anjali"}
print(students)

students[33]="Hiral"
print(students)
```

code

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python
dic8.py
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali'} output
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali', 33: 'Hiral'}
```

## setdefault()

setdefault() method is like get() but adds a key-value pair to a dictionary if it is not in the dictionary.

### Syntax#

```
dict.setdefault(key,value)
```

## Example#

```
students={1:"Ram",22:"Jayul",3:"Rahul",44:"Anjali"}
print(students)

students.setdefault(101, "Tina")
print(students)

students.setdefault(102, "")
students[102]="Aman"

print(students)
```

code

```
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali'}
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali', 101: 'Tina'} output
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali', 101: 'Tina', 102: 'Aman'}
```

## Modifying value to the Dictionary

Dictionaries are mutable. We can change its values.

### Syntax#

```
dict[key]=value
```

If the key is present in the dictionary, then the associated value with that key is updated or changed, otherwise a new key:value pair is added.

### Example#

```
student={'name':'Rahul', 'age':27}

student['age']=29
print(student) code

student['enrollNo']=1
print(student)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Pro
dic10.py output
{'name': 'Rahul', 'age': 29}
{'name': 'Rahul', 'age': 29, 'enrollNo': 1}
```

### Deleting items from Dictionaries

Dictionaries are mutable and so we can remove items from them.

#### pop()

pop() method removes the item from the dictionary for which the key is given. It also returns the value of the popped item.

If the key is not present in the dictionary, it raises a KeyError.

### Syntax#

```
dict.pop(key)
```

### Example#

```
>>> students={1:"Ram", 22:"Jayul", 3:"Rahul",44:"Anjali"}
>>> students.pop(22)
'Jayul'
>>> print(students)
{1: 'Ram', 3: 'Rahul', 44: 'Anjali'}
>>> students.pop(33) ← key is not present in dict
Traceback (most recent call last):
 File "<pyshell#13>", line 1, in <module>
 students.pop(33)
KeyError: 33
```

## popitem()

popitem() method removes the last inserted item from dictionary. If the key is not present in the dictionary, it raises a KeyError.

### Syntax#

```
dict.popitem()
```

## Example#

```
students={1:"Ram", 22:"Jayul", 3:"Rahul",44:"Anjali"}
```

```
print(students) code
print(students.popitem())
print(students)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs.
dic11.py
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali'}
(44, 'Anjali')
{1: 'Ram', 22: 'Jayul', 3: 'Rahul'} output
```

## clear()

clear() method removes all the items from a dictionary at once. When this operation is performed, the dictionary becomes an empty dictionary - {}.

### Syntax#

```
dict.clear()
```

## Example#

```
students={1:"Ram", 22:"Jayul", 3:"Rahul",44:"Anjali"}
print(students)

print("Cleared dictionary") code
students.clear()

print(students)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs
dic12.py
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali'}
Cleared dictionary
{} output
```

## del keyword

del keyword is used to delete the entire dictionary or item.

### Syntax#

```
del dict
del dict[key]
```

## Example#

```
students={1:"Ram", 22:"Jayul", 3:"Rahul",44:"Anjali"}
print(students)

print("Delete Anjali") code
del students[44]

print(students)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs,
dic13.py
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali'}
Delete Anjali
{1: 'Ram', 22: 'Jayul', 3: 'Rahul'} output
```

```
>>> students={1:"Ram", 22:"Jayul", 3:"Rahul",44:"Anjali"}
>>> del students
>>> print(students)
Traceback (most recent call last):
 File "<pyshell#18>", line 1, in <module>
 print(students)
NameError: name 'students' is not defined
```



## len()

len() function returns the length or number of **key:value** pairs of the dictionary passed as the argument.

### Syntax#

```
len(dict)
```

## Example#

```
students={1:"Ram", 22:"Jayul", 3:"Rahul",44:"Anjali"}
size=len(students) code
|
| print("Length:", size)
|= RESTART: C:/Users/F
dic14.py
Length: 4 output
```

## dict()

dict() creates a dictionary from a sequence of key-value.

### Syntax#

```
dict(sequence)
```

## Example#

```
Dict=dict({1:'Hello', 2: 'World!'})
print("Create Dictionary by using dict(): ")
print(Dict) code

Dict=dict([(1, 'Python'), (2, 'Programming')])
print("Dictionary with each item as a pair: ")
print(Dict)
```

```
= RESTART: C:/Users/Pradip/AppData/Local
/dic/dic15.py output
Create Dictionary by using dict():
{1: 'Hello', 2: 'World!'}
Dictionary with each item as a pair:
{1: 'Python', 2: 'Programming'}
```

## keys()

keys() function returns a list of keys in the dictionary.

### Syntax#

```
dict.keys()
```

## Example#

```
students={1:"Ram", 22:"Jayul", 3:"Rahul", 44:"Anjali"}
|
keys=students.keys() code
print("Student dictionaries Keys:")
print(keys)
```

```
= RESTART: C:/Users/Pradip/A
/dic/dic16.py output
Student dictionaries Keys:
dict_keys([1, 22, 3, 44])
```

## Example#

```
product={'name':'computer', 'brand':'dell', 'price':60000}

Iterating using key and value code
for p in product.keys():
 if p == 'price' and product[p] > 40000:
 print("computer price is too high",)
```

```
= RESTART: C:/Users/Pradip/App
/dic/dic17.py output
computer price is too high
```

## values()

values() function returns a list of values in the dictionary.

### Syntax#

```
dict.values()
```

## Example#

```
product={'name':'computer', 'brand':'dell', 'price':60000}
|
| print("Product Values:") code
| print(product.values())
```

```
= RESTART: C:/Users/Pradip/AppData/Local/
/dic/dic19.py output
Product Values:
dict_values(['computer', 'dell', 60000])
```

### Example#

```
bookInventory={'Java': 200, 'Python':150, 'Ruby':1000}
|
| bookInventory=bookInventory.values()| code
| print(" Stock available:\n",bookInventory)
```

```
= RESTART: C:/Users/Pradip/AppData/
/dic/dic18.py output
Stock available:
dict_values([200, 150, 1000])
```

### items()

items() returns a list of tuples(key–value) pair.

#### Syntax#

dict.items()

### Example#

```
students={1:"Ram", 22:"Jayul", 3:"Rahul", 44:"Anjali"}
|
| print("Students Details:") code
| for key,value in students.items():
| print(key,value)
```

```
= RESTART: C:/Users/Pra
/dic/dic20.py
Students Details:
1 Ram
22 Jayul
3 Rahul
44 Anjali
```

## update()

update() method merges the dictionaries

### Syntax#

```
dict.update(dict2)
```

It adds the items from dict2 to dict

## Example#

```
students={1:"Ram", 22:"Jayul"}

students1={3:"Rahul", 44:"Anjali"}
I
students.update(students1) code

print("Updated students dict.")
print(students)

print("Original students1 dict.")
print(students1)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/
dic21.py
Updated students dict.
{1: 'Ram', 22: 'Jayul', 3: 'Rahul', 44: 'Anjali'}
Original students1 dict.
{3: 'Rahul', 44: 'Anjali'}
```

## fromkeys()

fromkeys() function creates a new dictionary with specified keys and value.

**Syntax#**

```
dict.fromkeys(keys,value)
```

**Example#**

```
keys={ 'a', 'e', 'i', 'o', 'u' }
value='vowel' code
|
vowels=dict.fromkeys(keys, value)
print("Use of fromkey method")
print(vowels)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python38/books
dic22.py output
Use of fromkey method
{'i': 'vowel', 'a': 'vowel', 'e': 'vowel', 'u': 'vowel', 'o': 'vowel'}
```

**copy()**

copy() function returns a copy of the dictionary.

**Syntax#**

```
dict.copy()
```

**Example#**

```
original_marks={'Python':70,'Physics':60}
|
copied_marks = original_marks.copy() code
|
print('Original Marks:', original_marks)
print('Copied Marks:', copied_marks)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Prog
dic23.py output
Original Marks: {'Python': 70, 'Physics': 60}
Copied Marks: {'Python': 70, 'Physics': 60}
```

**sorted()**

sorted() function returns the sorted list of keys.

#### Syntax#

```
sorted(dict)
```

#### Example#

```
students={44:"Anjali",1:"Ram",22:"Jayul",3:"Rahul"}

print("Students dictionary key's in sorted order")
print(sorted(students))
```

```
= RESTART: C:/Users/Pradip/AppData/Local/P
dic24.py
output
Students dictionary key's in sorted order
[1, 3, 22, 44]
```

#### 1.4.1.6 Nested Dictionaries

A dictionary inside a dictionary is called nested dictionary

#### Syntax#

```
nestedDict={ 'dictA': {'key1': 'value11'}, 'dictB': {'key2': 'value2'} }
```

#### Example#

```
emp={ 'emp1': { 'name': 'Bob', 'job': 'Mgr' },
 'emp2': { 'name': 'Alice', 'job': 'Dev' },
 'emp3': { 'name': 'Trudy', 'job': 'Dev' } }
print(emp)
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Pytho
n38/books/dic/dic26.py
{'emp1': {'name': 'Bob', 'job': 'Mgr'}, 'emp2': {'name': 'Alic
e', 'job': 'Dev'}, 'emp3': {'name': 'Trudy', 'job': 'Dev'}}
```

#### : EXERCISE-1 :

1. What do you mean by "Strings are immutable"?
2. Explain String Operations.
3. Explain String built-in functions and methods.
4. Explain List Operations.

5. Explain List built-in functions and methods.
  6. Define Set and how is it created in python?
  7. Explain accessing items in set.
  8. List functions to delete items in sets.
  9. Explain Frozen sets.
  10. Explain Set operations in python.
  11. Define Tuple and how is it created in Python?
  12. What do you mean by "Tuples are immutable"?
  13. Explain Tuple Operations.
  14. Explain Tuple Built-in functions and methods.
  15. Differentiate between tuple and list.
  16. Explain Tuple unpacking.
  17. Explain Nested Tuple.
  18. What is Dictionary? Explain with example.
  19. Explain different operations performed on the dictionary.
  20. Explain properties of dictionary keys.
  21. Explain Dictionary Built-in functions and methods.

### **Do the following (MCQs)**

**1. Tuples are** \_\_\_\_\_



**2. Tuples can contain elements of any data type.(T/F)**



**3. In tuples values are enclosed in \_\_\_\_\_**

- (A) Square brackets                          (B) Curly brackets  
(C) Parenthesis                                (D) None of the above

**4. Write the output of the following.**

```
A = tuple("Tuple")
print(A)
```

- (A) (tuple)  
(B) ('T', 'u', 'p', 'l', 'e')  
(C) ("Tuple")  
(D) None of the above

**5. len( ) function returns the number of elements in tuple.(T/F)**



**Write the**  
**A = list(**

- (A) ('P', 'y', 't', 'h', 'o', 'n')      (B) ['P', 'y', 't', 'h', 'o', 'n']  
(C) ~~["P", "y", "t", "h", "o", "n"]~~

© 2010 The Authors. Journal compilation © 2010 Association for Child and Adolescent Mental Health.

```
a=(32,29,56,30,11)
print(a[0]+a.index(30))
```

- (A) 35                    (B) 32                    (C) 36                    (D) 62

**8. Which of the following is not a function of tuple?**



**9. Write the output of the following:**

```
a=(11,9,38,16)
s=0
for i in a:
 if i%2==0:
 s=s+a[i]
print(s)
```



**10. Write the output of the following:**

```
a=(9, 2, 2, 3, 11, 6, 1)
print(min(a) + max(a) + a.count(3))
```



**11. Which of the following is/are features of tuple?**

- (A) Tuple is immutable
  - (B) Tuple is a sequence data type.
  - (C) In tuple, elements are enclosed in Parenthesis.
  - (D) All of the above

**12. Which of the following is not a tuple?**



**13. Which of the following statement will create an empty tuple?**

- (A) A = ()  
(B) B = tuple()  
(C) Both a and b  
(D) None of the above

**14. Which of the following is a tuple with single element?**

- (A)  $A = (1,)$       (B)  $A = 1,$   
 (C) Both a and b      (D) None of the above

**15. Write the output of the following:**

```
a = (2)
type(a)
```

**16. What is the length of the given tuple? t1=(1,2,(8,9,10))**



**17. Write the output of the following:**

```
a1 = (1,2,3,4,5)
a2 = a1
print(a2)
```



**18. Write the output of the following:**

```
a1= ('Ram', 'Shyam')
a2 = (1,2,3)
a2+a1
```

- (A) ('Ram', 'Shyam', 1, 2, 3)      (B) (1, 2, 3, 'Ram', 'Shyam')  
(C) (1, 'Ram', 2, 'shyam', 3)      (D) None of the above

**19. Which of the following statement will return an error. A1 is a tuple.**

- (A) A1 + (12)  
 (B) A1 + [2]  
 (C) Both a and b  
 (D) None of the above

**20. Which mathematical operator is used to replicate a tuple?**



**21. Write the output of the following:**

```
A1 = (12, 56, 88, 43, 9, 41)
print(A1[2:3])
```

- (A) (88)  
(B) (88,)  
(C) (88,43)  
(D) None of the above

**22. Write the output of the following:**

```
a1 = ('1', '2', '3', '4', '5')
print(a1 + tuple("Code"))
```

- (A) ('1', '2', '3', '4', '5', 'C', 'o', 'd', 'e')      (B) ('1', '2', '3', '4', '5', 'Code')  
(C) Error      (D) None of the above

**23. Which function returns the length of tuple?**

**24. Write the output of the following:**

```
a1 = (2,3)
a2 = (3,2)
a1 == a2
```

- (A) True      (B) False      (C) Error      (D) None of above

**25. del a1 statement will delete the tuple named 'a1'.(T/F)**

- (A) True      (B) False

**26. Write the output of the following.**

```
a1=((1,3),(1,2),(2,))
print(min(a1))
```

- (A) 1,3      (B) (1,2)      (C) (1,3)      (D) Error

**27. min(t1) will return an error if the tuple t1 contains value of mixed data type.**

- (A) True      (B) False

**28. Which of the following function return the frequency of particular element in tuple?**

- (A) index( )      (B) max( )
(C) count( )      (D) None of the above

**29. What is the output of the line of code shown below, if s1= {1, 2, 3}?**

```
s1.issubset(s1)
```

- (A) True      (B) Error
(C) No output      (D) False

**30. What will be the output of the following Code?**

```
s1 = {1,2,3,4,5,6}
s2 = {7,8,9,1,2,3,4}
```

```
print(s1.issubset(s2))
print(s2.issuperset(s1))
```

- (A) False      (B) False
False      True
(C) True      (D) True
False      True

**31. What will be the output of the Following code ?**

```
s1 = {"1", "bat", "21"}
print(s1[1])
```

- (A) 1      (B) bat
(C) Syntax Error      (D) None of the Above

**32. What will be the output of the Following Code ?**

```
Set = {"Cake", "Cold Drink", 1, "Orange"}
Set.add("Cold Drink")
Set.add("Orange")
print(Set)
```

- (A) {"Cake", "Cold Drink", 1, "Orange"}
- (B) {"Cake", "Cold Drink", 1, "Orange", "Cold Drink", "Orange" }
- (C) Error
- (D) None of the Above

**33. Which one of the following will be used to delete 'Jay' from Set1 ?**

```
Set1 = {"Jay", "Ray", "Black"}
```

- (A) Set1.pop("Jay")
- (B) Set1.discard("Jay")
- (C) del Set1 ["Jay"]
- (D) None of the Above

**34. What will be the output of the following code ?**

```
set1 = {1, 2, 3, 4}
set2 = {51, 22, "11", 6}
set3 = set1.union(set2)
print(set3)
```

- (A) {1, 2, 3, 4, 6, 51, 22}
- (B) {1, 2, 3, 4, 6, '11', 51, 22}
- (C) Error
- (D) {1, 2, 3, 4, 6, 11, 51, 22}

**35. set1 = {"Cricket", "BaseBall", "Burger"}**

```
set2 = {"Burger", "Laptop", "Phone"}
set3 = set2.difference(set1)
print(set3)
```

- (A) {"Laptop", "Phone"}
- (B) {"Cricket", "BaseBall", "Laptop", "Phone"}
- (C) Error
- (D) None of the Above

**36. What will be the output of the following code ?**

```
Set = {"", "Orange", "Black"}
Set.update(["Cake", "Burger", "Ball"])
print(Set)
```

- (A) {"", 'Cake', 'Black', 'Ball', 'Orange', 'Burger'}
- (B) {'Cake', 'Black', 'Ball', 'Orange', 'Burger'}
- (C) Error
- (D) None of the Above

**37. Which of the following statements is used to create an empty set?**

- (A) {}
- (B) set()
- (C) []
- (D) ()

**38. Keys of dictionary must be \_\_\_\_\_**

- (A)antique
- (B)unique
- (C)mutable
- (D)integers

**39. Dictionaries in python are \_\_\_\_\_**

- (A) Mutable data type                    (B) Non-Mutable data type  
(C) Mapping data type                    (D) Both a and c

**40. Which of the following is used to delete an element from Dictionary?**

- (A) pop()                                (B) delete                            (C) remove                            (D) None of the above

**41. Which statement is used to create an empty dictionary?**

- (A) A = [ ]                                (B) A = ( )  
(C) A = dict{ }                            (D) A = { }

**42. In dictionary Keys and values are separated by ?**

- (A) Colon (:)                                (B) Comma( ,)                            (C) Semicolon(;)                            (D) dot(.)

**43. All elements in dictionary are separated by ?**

- (A) Colon (:)                                (B) Comma( ,)                            (C) Semicolon(;)                            (D) dot(.)

**44. Both Keys and Values have to be unique in a dictionary.**

- (A) True                                        (B) False

**45. 1,2,3 are the \_\_\_\_\_ in the following dictionary.**

**Dict = {1 : "One", 2 : "Two", 3 : "Three"}**

- (A) Keys                                        (B) Values                                    (C) Items                                    (D) None of the above

**46. Keys in dictionary are \_\_\_\_\_ .**

- (A) Mutable                                    (B) Immutable                                (C) antique                                    (D) integers

**47. What type of error is returned by the following code :**

**Dict1 = {'H' : "Hat", 'B' : "Banana", 2 : "Vehicle"}  
print(Dict1[1])**

- (A) KeyError (B) KeyNotFoundError (C) NotFoundError (D) Syntax Error

**48. Traversing a dictionary can be done using \_\_\_\_\_**

- (A) if statement (B) loop                            (C) jump statement                            (D) None of above

**49. What will be the output of the following Code ?**

**A = {1 : "One", 2 : "Two"}  
B = {'A' : "Apple", 'B' : "Bat"}  
print(B.get(5,'Enter The Key First'))**

- (A) {'A' : "Apple", 'B' : "Bat", 5 : "Enter The Key First"}  
(B) enter the key first  
(C) Enter The Key First  
(D) Error

**50. pop( ) function delete and \_\_\_\_\_ the element of dictionary.**

- (A) display      (B) return      (C) not return      (D) add

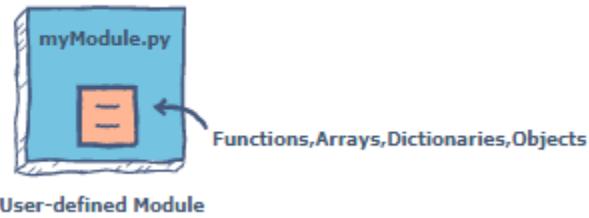
## Introduction to module

A module is a file containing Python definitions and statements. A module is nothing but a python file containing a set of functions, variables, and classes.

In another word, we can say that our python code file saved with the extension (.py) is treated as the module. For example, test.py is called a module and its name is a test.

Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized. For example, we can group the code of arithmetic operation in a module and can be reused in another file.

We can define our most used functions in a module and import them, instead of copying their definitions into different programs.



### Advantages of modules

**Code Reusability:** Working with modules makes the code reusable. Once the module is created, we can import it to another file.

**Simplicity:** The module focuses on a small proportion of the problem, rather than focusing on the entire problem.

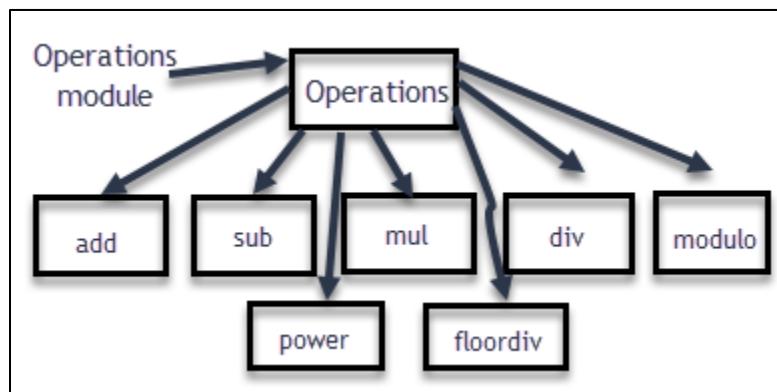
**Maintainability** of the function will be increased.

### 2.1.2 Creating user defined module

To create a module, just put the code(functions, classes, variables, etc) inside a .py file.

#### Example#

Let's create a module named Operations.py. The Operations module contains functions for different mathematical operations like addition, subtraction, multiplication, division, modulo, floor division, and power.



```

def add(x, y):
 return (x+y)

def subtract(x, y):
 return (x-y)

def multiplication(x, y):
 return (x*y)

def division(x, y): Operations.py module
 return (x/y)

def floorDivision(x, y):
 return (x//y)

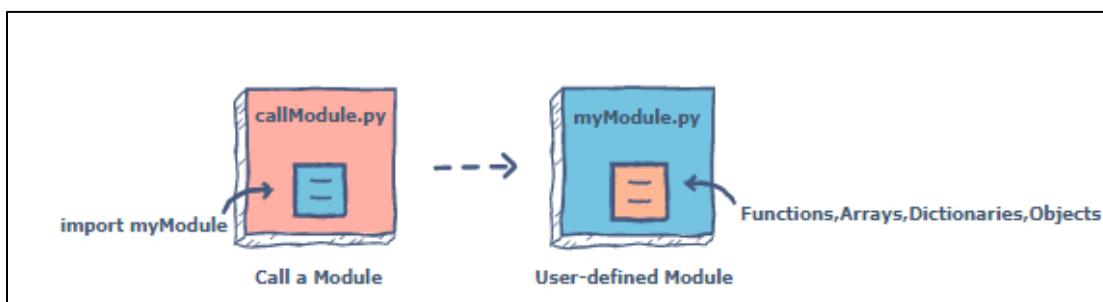
def modulo(x, y):
 return (x%y)

def power(x, y):
 return (x**y)

```

### 2.1.3 Importing a module in python

We can import the functions, classes defined in a module to another module using the import statement in some other Python source file.



#### Normal import

The import statement is used to import all the functionality of one module into another program. If we want to use the functionality of any python file then just import those files as modules in another python file.

### Syntax#

```
import module1
import module1,module2,..... module n
```

After importing the module, each function (or variable) must be called by the name of the module followed by dot (.) and name of the function/variablename.

### Example#

```
import module1
module1.variable_name
module1.function_name()
```

A module is loaded only once, regardless of the number of times it is imported.

### Example#

Now, import the functionality of the **Operations** module in the Main module.

```
import Operations

no1=int(input("Enter the first number: "))
no2=int(input("Enter the second number: "))

print("Sum = ", Operations.add(no1,no2))
print("Mul = ", Operations.multiplication(no1,no2))
print("modulo = ", Operations.modulo(no1,no2))
```

```
= RESTART: C:/Users/Pradip/Ap
le/Main.py
Enter the first number: 45
Enter the second number: 15
Sum = 60
Mul = 675
modulo = 0
```

output

### From import

We can import a specific function, class, or variable from a module rather than importing the entire module.

### Syntax#

```
from module1 import name1, name2..,n
```

Using this method, we can directly access the members without a module name.

### Example#

```
from module1 import variable_name, function_name
variable_name
function_name()
```

### Example#

Import some of the functionality of the Operations module into another module.

```
from Operations import add, multiplication

no1=int(input("Enter the first number: "))
no2=int(input("Enter the second number: "))

print("Sum = ", add(no1,no2))
print("Mul = ", multiplication(no1,no2))
```

```
= RESTART: C:/Users/Pradip/AppD
le/mod1.py
Enter the first number: 45
Enter the second number: 15
Sum = 60
Mul = 675
```

output

### Example#

Now, if we try to access the method which is not imported in the import section then an interpreter will generate an error.

```
from Operations import add

no1=int(input("Enter the first number: "))
no2=int(input("Enter the second number: "))

print("Sum = ", add(no1,no2))
print("Mod = ", modulo(no1,no2))
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/3.8/mod2.py
Enter the first number: 45 output
Enter the second number: 15
Sum = 60
Traceback (most recent call last):
 File "C:/Users/Pradip/AppData/Local/Programs/Python/3.8/mod2.py", line 7, in <module>
 print("Mod = ", modulo(n01,no2))
NameError: name 'modulo' is not defined
```

## From import with \*

If we need to import everything from a module and we don't want to use the dot operator with module name then we use from import with \*

### Syntax#

```
from module1 import *
```

Similar to from import, we can directly access the members without a module name.

### Example#

```
from module1 import *
variable_name
function_name()
```

Note: If you know exactly what you will be needing from the module, it is not recommended to use \*, else can use it.

## Example#

Now, import all functionality of the Operations module in another module.

```
from Operations import *

no1=int(input("Enter the first number: "))
no2=int(input("Enter the second number: "))

print("Sum = ", add(no1,no2)) code
print("Mul = ", multiplication(no1,no2))
print("Mod = ", modulo(no1,no2))
print("Power = ", power(no1,no2))
```

```
= RESTART: C:/Users/Pradip/AppData
le/mod3.py
Enter the first number: 4 output
Enter the second number: 2
Sum = 6
Mul = 8
Mod = 0
Power = 16
```

## 2.1.4 Renaming a module

We can rename the module while importing it using the **as** keyword. This allows you to give a shorter name to a module while using it in your program. We can also rename the function name in the import section.

### Syntax#

```
import module1 as m1
m1.function_name()
```

Now, we will rename the module Operations as **ops** and use it in the program.

### Example#

```
import Operations as ops

no1=int(input("Enter the first number: "))
no2=int(input("Enter the second number: "))

print("Sum = ", ops.add(no1,no2)) code
print("Mul = ", ops.multiplication(no1,no2))
print("modulo = ", ops.modulo(no1,no2))
```

```
= RESTART: C:/Users/Pradip/A
le/mod4.py
Enter the first number: 45
Enter the second number: 15
Sum = 60
Mul = 675 output
modulo = 0
```

Once we define the alias name or rename it, we cannot use the original name of the module, if try to access any function by using the original name, python will throw a name error

### Example#

```
import Operations as ops

no1=int(input("Enter the first number: "))
no2=int(input("Enter the second number: "))

print("Sum = ", ops.add(no1,no2)) code
print("Sum = ", Operations.add(no1,no2)) throw error
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Pro
le/mod5.py
Enter the first number: 45
Enter the second number: 15
Sum = 60
Traceback (most recent call last):
 File "C:/Users/Pradip/AppData/Local/Progra
mod5.py", line 8, in <module>
 print("Sum = ", Operations.add(no1,no2))
NameError: name 'Operations' is not defined
```

## 2.1.5 dir() function

The `dir()` function will return the names of all the properties and methods present in a module. This includes all the sub-modules, variables, and functions defined in this module.

### Syntax#

```
dir(moduleName)
```

### Example#

```
>>> import Operations
>>> dir(Operations)
['__builtins__', '__cached__', '__doc__', '__
file__', '__loader__', '__name__', '__package__
__', '__spec__', 'add', 'division', 'floorDiv
ision', 'modulo', 'multiplication', 'power',
'subtract']
```

## 2.1.6 Reloading modules

For efficiency reasons, each module is only imported once per interpreter session. Therefore, if you change your modules, you must restart the interpreter or, if it's just one module you want to test interactively, use `reload` function.

To reload the module in the program, first import **importlib** module and then use **reload** function.

### Syntax#

```
import importlib
importlib.reload(module-name)
```

### Example#

```
import importlib
```

```
importlib.reload(Operations)
```

## 2.1.7 Executing modules as scripts

When you run a Python module, the code in the module will be executed, just as if you imported it, but with the `__name__` set to "`__main__`". Within a module, the module's name (as a string) is available as the value of the global variable `__name__`.

The `__name__` variable describes whether the program is executing directly or indirectly, that means an individual program or as a module.

Every module has a `__name__` variable which has two possible values:

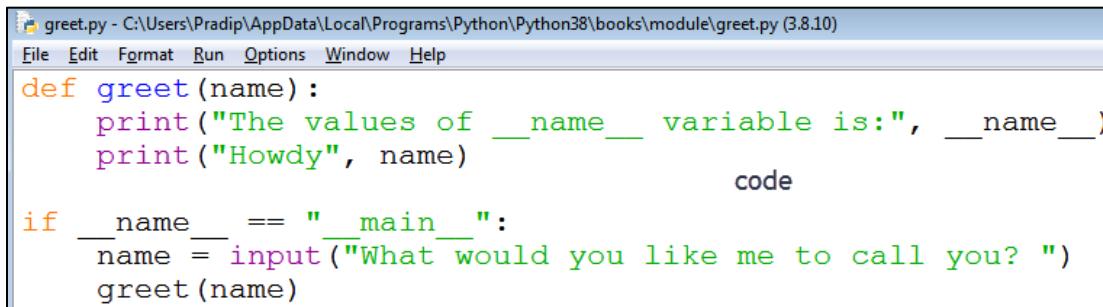
- When our module is imported by another module, `__name__` will be set to the **name of our module**
- When our module is being run from the command line, `__name__` will be set to the string '`__main__`'

That means by adding the below code at the end of your module:

```
if __name__ == "__main__":
```

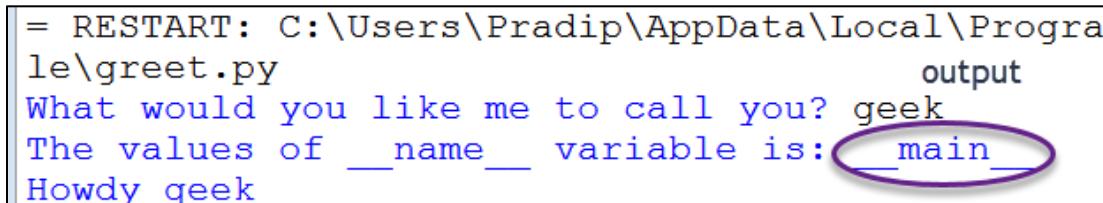
you can make the file usable as a script as well as an importable module.

### Example#



```
greet.py - C:\Users\Pradip\AppData\Local\Programs\Python\Python38\books\module\greet.py (3.8.10)
File Edit Format Run Options Window Help
def greet(name):
 print("The values of __name__ variable is:", __name__)
 print("Howdy", name)
 code
if __name__ == "__main__":
 name = input("What would you like me to call you? ")
 greet(name)
```

Here, a module named `greet.py` is created. If this module is executed, the output will display as below.



```
= RESTART: C:\Users\Pradip\AppData\Local\Program
le\greet.py
What would you like me to call you? geek
The values of __name__ variable is: __main__
Howdy geek
```

But when the `greet` module is imported, the code inside `if __name__ == "__main__":` will not be executed as shown below.

```

>>> import greet
>>> greet.greet("geek") output module name
The values of __name__ variable is: greet
Howdy geek

```

## 2.1.8 Module search path

While importing a module, Python looks at several places. The interpreter first looks for a built-in module. If not found then it looks for a list of directories defined in **sys.path**.

The python interpreter searches for modules in the following order.

- The current directory.
- PYTHONPATH (an environment variable with a list of directories).
- If all fails, python checks the installation-dependent default directory configured at the time of installation.

### Example#

List of directories defined in sys.path.

```

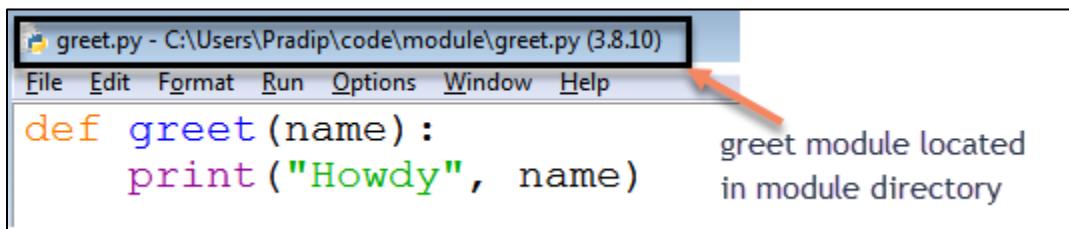
>>> import sys directories list
>>> sys.path
['', 'C:\\\\Users\\\\Pradip\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python38\\\\Lib\\\\idlelib',
 'C:\\\\Users\\\\Pradip\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python38\\\\python38.zip',
 'C:\\\\Users\\\\Pradip\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python38\\\\DLLs',
 'C:\\\\Users\\\\Pradip\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python38\\\\lib',
 'C:\\\\Users\\\\Pradip\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python38', 'C:\\\\Users\\\\Pradip\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python38\\\\lib\\\\site-packages']

```

Till now, we have created a module and imported a module in another program that is in the same directory. But when we have a module in one directory and imported in python file which is in another directory. In this case, we will not be able to import the module.

### Example#

For example, the greet module is located in the **module** directory.



Now, import the greet module in a demo python file which located in the **code** directory.

```

demo.py - C:\Users\Pradip\code\demo.py (3.8.10)
File Edit Format Run Options Window Help
import greet
greet.greet ("geek")

```

demo file is located in code directory

When we run the demo.py file, it will throw an error as the greet.py module is located at a different location than demo.py.

```

=====
RESTART: C:\Users\Pradip\code\demo.py =
Traceback (most recent call last):
 File "C:\Users\Pradip\code\demo.py", line 1, in <module>
 import greet
ModuleNotFoundError: No module named 'greet'

```

A module path can be added to python's module search path by appending the sys.path variable. This can be done by using the **append()** function of sys.path. The directory in which your module is located should be appended in sys.path.

```

import sys
sys.path.append("C:\\\\Users\\\\Pradip\\\\code\\\\module")

import greet
greet.greet ("geek")

```

greet module location

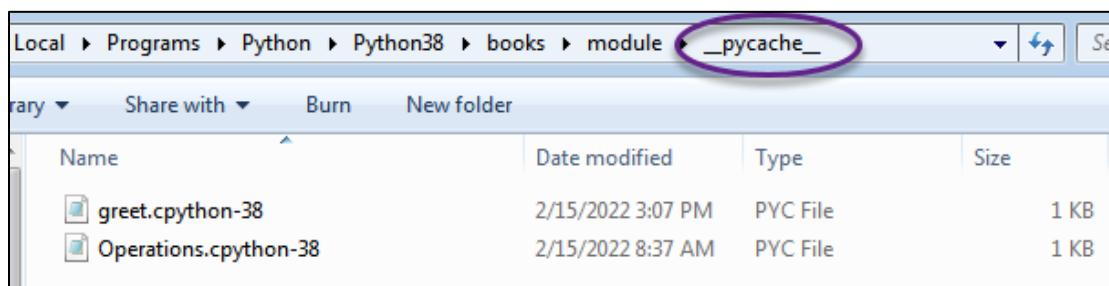
```

>>>
=====
RESTART: C:\Users\Pradip\code\demo.py =
Howdy geek output

```

## 2.1.9 Compiled version of the module

Whenever we are using any module in python for other programs then a separate compiler file will be created with the extension as **.pyc**. The compiler file will be stored in a separate folder called **\_\_pycache\_\_**. This will speed up the loading module and improve the performance.



## Introduction to Packages

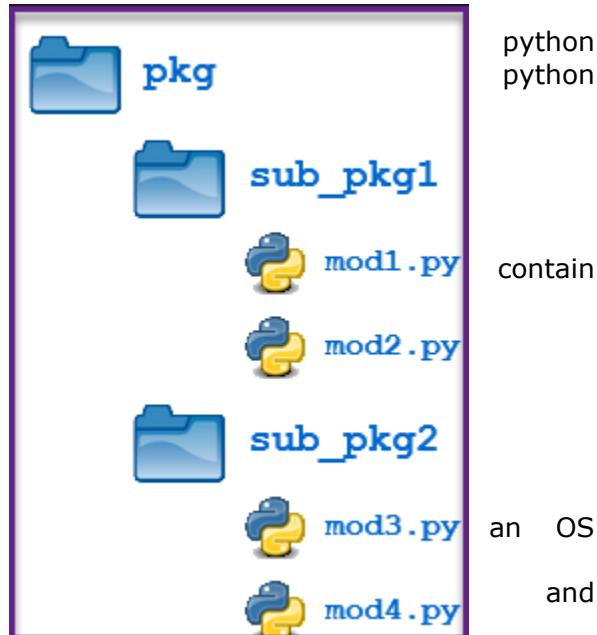
A package is a directory that contains multiple modules. It is used to group multiple related modules together.

The packages are used to categorize the application-level code efficiently.

A python package in addition to modules should have a file called `__init__.py`. This file maybe empty or contains data like other modules of package or initialization of package-level data.

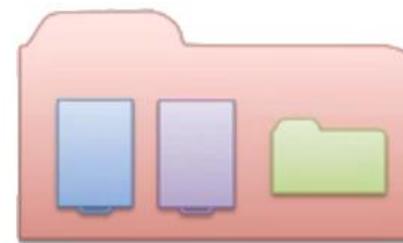
In python, a package may also contain sub-packages.

Just like there are different drives and folders in to help us store files and manage, similarly packages help us in storing other sub-packages modules, so that it can be used by application efficiently.



## Advantages of packages

- Naming conflicts can be solved easily.
- Improves the **Modularity** of the application.
- The **readability** of the application will be improved.
- **Maintainability** of the application will be improved.



### 2.2.2 Creating user defined package

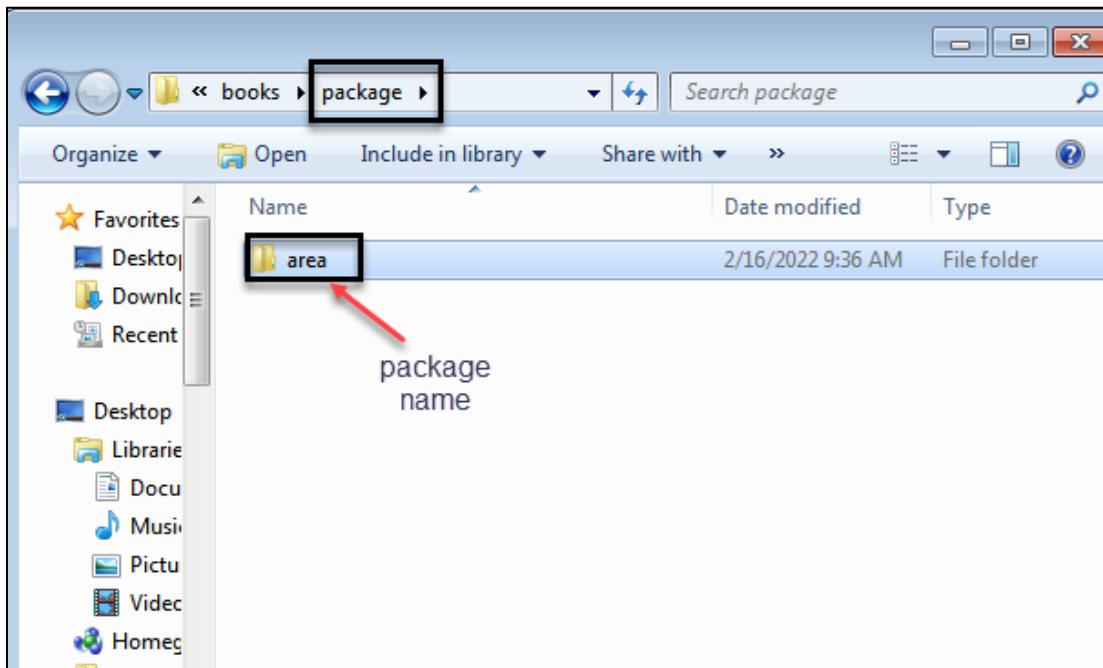
Steps to create a user defined package:

1. First, create a directory and give it a package name, the package name should be related to the operation.
2. Put the required python files aka modules in it.
3. Finally, create an `__init__.py` file inside the directory.

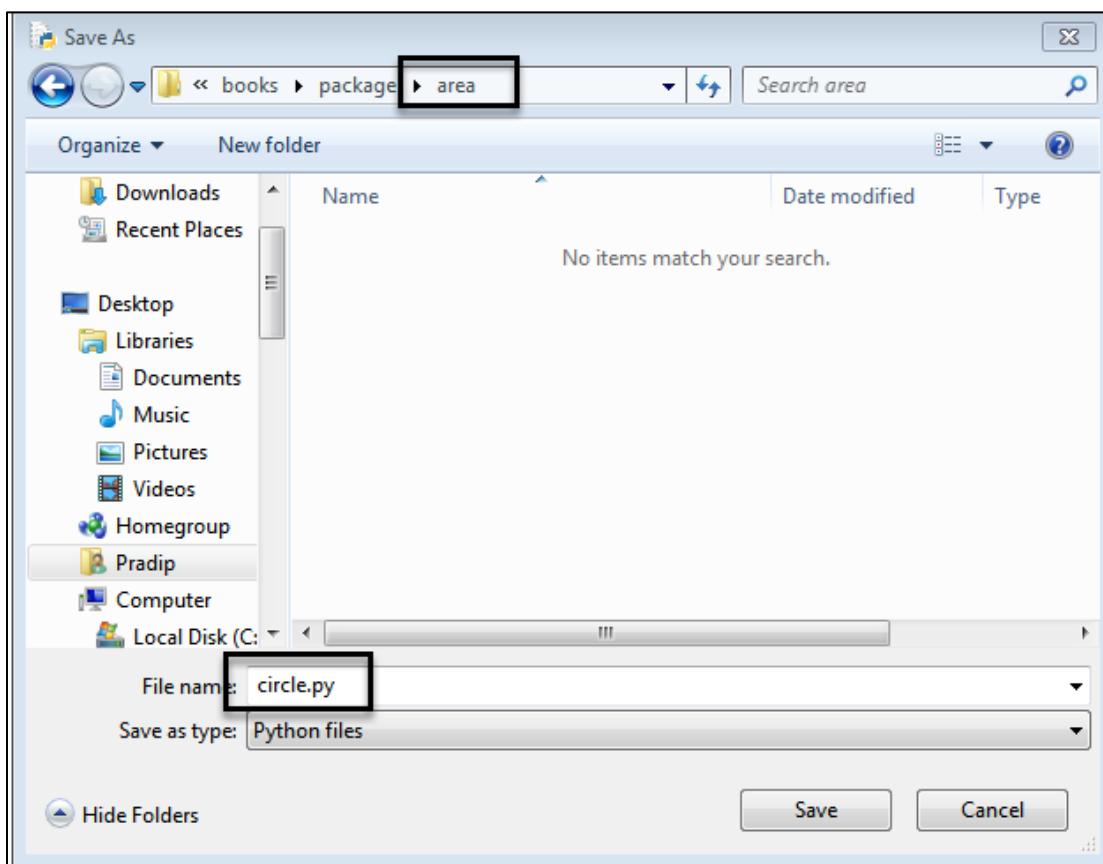
#### Example#

Create a package named area and create 3 modules in it named –square, circle, and rectangle each having a function to calculate the area of square, circle, and rectangle respectively.

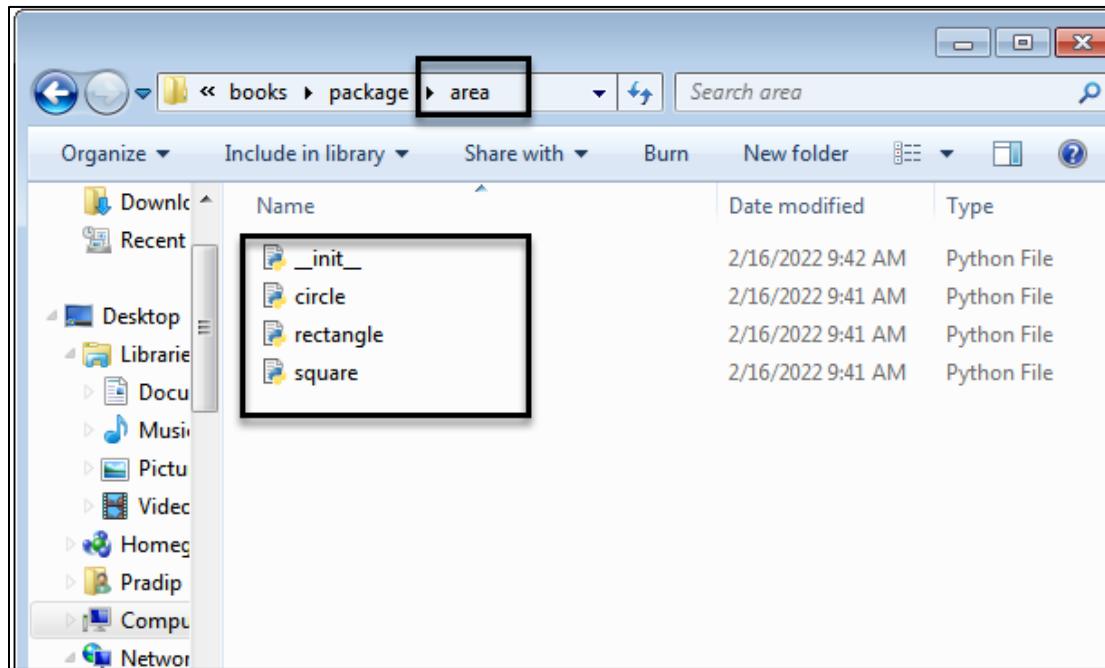
**Step 1:** Create an **area** folder in my current working directory called package.



**Step 2:** Create a circle module under the area package.



Similarly, create a python file named square, rectangle, and \_\_init\_\_.py in the area package.



**Step 3:** Now, add the module code as per their functionality. Here, the \_\_init\_\_.py file created is empty.

```
def circle(radius):
 import math
 return math.pi*radius**2
```

```
def square(side):
 return side*side
```

```
def rectangle(l, b):
 return l*b
```



### 2.2.3 Importing a package in python

Packages can be imported in the same way as we import modules. We can import modules from the package using the dot (.) operator.

#### Normal import

The **import** statement is used to import modules from packages.

### Syntax#

```
import packageName.moduleName
import packageName.subPackageName.moduleName
```

After importing the module from the package, each function (or variable) must be called by the name of the package followed by a dot(.) followed by module name followed by a dot(.) and name of the function/variablename.

### Example#

```
import packageName.moduleName

packageName.moduleName.functionName()
packageName.moduleName.variableName
```

### Example#

Now, create the test.py module and import the functionality of the circle, square, and rectangle modules.

```
import area.circle
import area.square
import area.rectangle

print("Area of Circle:", area.circle.circle(3))
print("Area of Square:", area.square.square(3))
print("Area of Rectangle:", area.rectangle.rectangle(3, 4))
```

```
= RESTART: C:\Users\Pradip\AppData\Local\test.py
Area of Circle: 28.274333882308138
Area of Square: 9
Area of Rectangle: 12
```

### From import

We can import a specific function, class, or variable from a module or entire module.

**Approach 1:** Import all functions from a specific module

### Syntax#

```
from packageName import moduleName
from packageName.subPackageName import moduleName
```

### Example#

```
from packageName import moduleName
```

```
moduleName.functionName()
moduleName.variableName
```

### Example#

```
from area import circle
from area import square
from area import rectangle

print("Area of Circle:", circle.circle(3))
print("Area of Square:", square.square(3))
print("Area of Rectangle:", rectangle.rectangle(3, 4))
```

```
= RESTART: C:/Users/Pradip/AppData/L
age/test1.py
Area of Circle: 28.274333882308138
Area of Square: 9
Area of Rectangle: 12
```

**Approach 2:** Import specific functions from the module

### Syntax#

```
from packageName.moduleName import functionName
from packageName.subPackageName.moduleName import functionName
```

### Example#

```
from packageName.moduleName import functionName

functionName()
```

Using Approach 2, we don't have to use the module name to call the function name.

### Example#

```
from area.circle import circle
from area.square import square
from area.rectangle import rectangle

print("Area of Circle:", circle(3)) code
print("Area of Square:", square(3))
print("Area of Rectangle:", rectangle(3, 4))
```

```
= RESTART: C:/Users/Pradip/AppData/
age/test2.py
Area of Circle: 28.274333882308138
Area of Square: 9
Area of Rectangle: 12
```

### From import with \*

We have used \* to import all the functions from a module in the module section. This method can also be used here. But **by default** importing package modules using \* will show an error.

```
from area import *
print("Area of Circle:", circle.circle(3))
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/
age/test3.py
Traceback (most recent call last):
 File "C:/Users/Pradip/AppData/Local/Programs/
/test3.py", line 3, in <module>
 print("Area of Circle:", circle.circle(3))
NameError: name 'circle' is not defined
```

The solution is to put \_\_all\_\_ variable in \_\_init\_\_.py file. The \_\_all\_\_ variable takes a list of module names that will be imported when from package import \* is encountered.

```
__all__ = ["circle", "square", "rectangle"]
```

The above statement makes module circle.py, square.py, and rectangle.py accessible using from import \* statement.

The screenshot shows a Python code editor window with the title bar "C:\Users\Pradip\AppData\Local\Programs\Python\Python38\books\package\area\\_\_init\_\_.py (3.8.10)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code area contains the following code:

```
__all__ = ["circle", "square", "rectangle"]
```

Now we can use **from import \*** method.

#### Syntax#

```
from packageName import *
```

#### Example#

```
from packageName import *
moduleName.functionName()
```

#### Example#

```
from area import *
print("Area of Circle:", circle.circle(3)) code
print("Area of Square:", square.square(3))
print("Area of Rectangle:", rectangle.rectangle(3, 4))
```

```
= RESTART: C:/Users/Pradip/AppData/Local/Programs/Python/Python38/test3.py
Area of Circle: 28.274333882308138
Area of Square: 9
Area of Rectangle: 12
```

### 2.2.4 Intra-package References

When packages are structured into subpackages, you can use absolute or relative imports to refer to submodules of siblings packages.

For example, the pkg package contains two sub-package named sub-pkg1 and sub-pkg2. Now each subpackage contains one module.

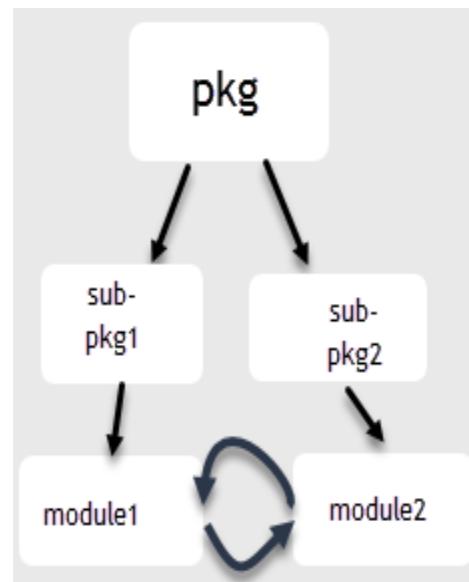
Now, module2 wants to use functions from sub-pkg1's module1, then it can be possible using absolute import or relative import.

#### Absolute import example:

```
from pkg.sub-pkg1 import module1
```

#### Relative import example:

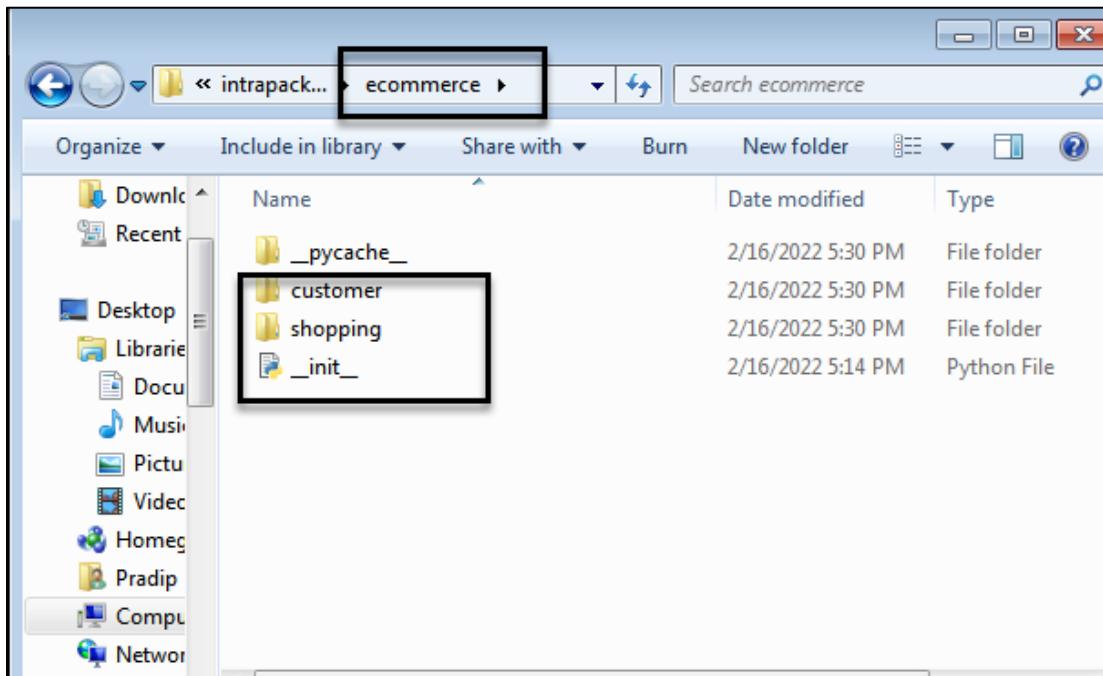
```
from ..sub-pkg1 import module1
```



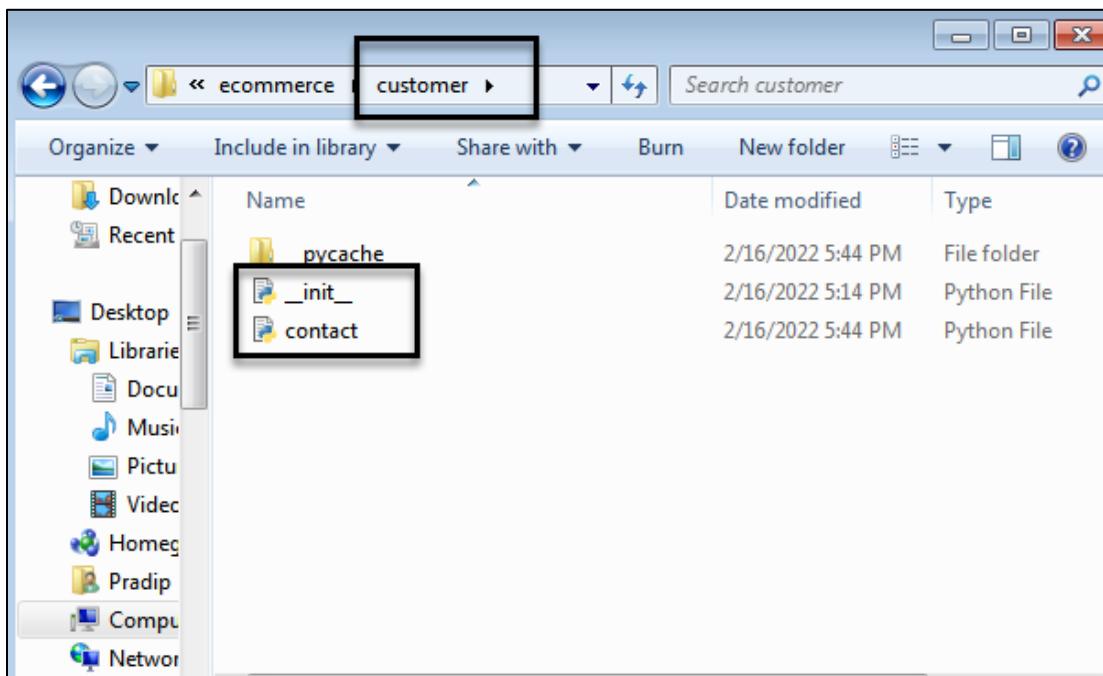
#### Example#

Create a package named ecommerce. ecommerce package has two sub-packages named customer and shopping. Create a module in customer and shopping sub-package named contact and sales respectively. Now, use the contact module in the sales module.

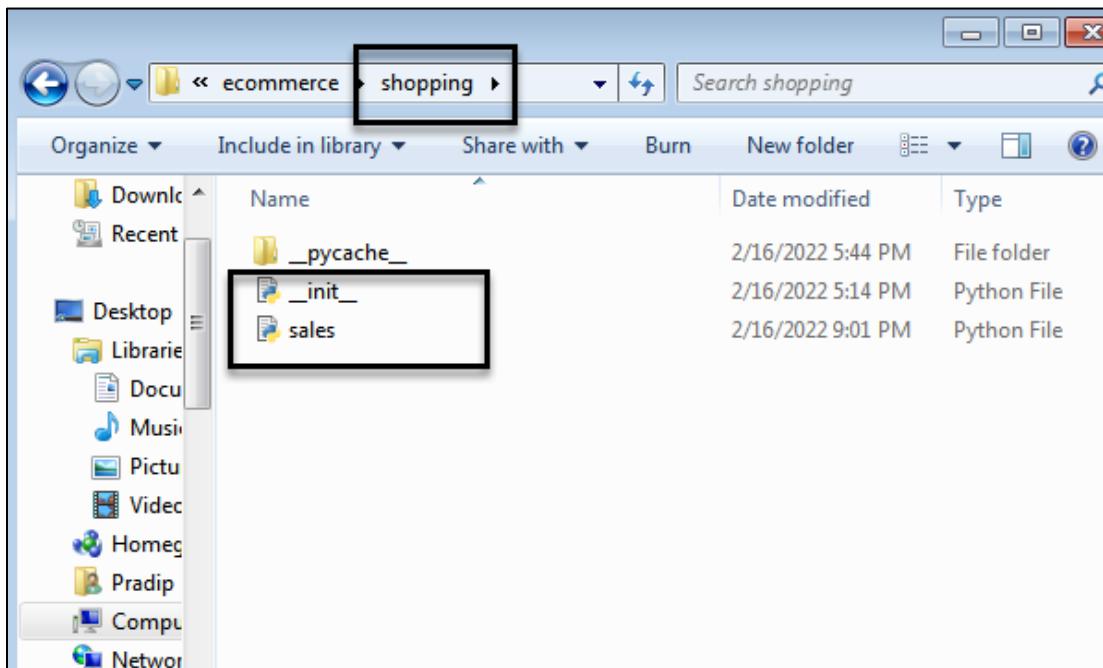
**Step 1: Create a ecommerce folder in the current working directory called intrapackage. Here ecommerce folder is the package.**



**Step 2: Create a customer sub-package in ecommerce package.**



### Step 3: Create a shopping sub-package in ecommerce package.



### Step 4: Create a contact module in customer sub-package.

A screenshot of a Python code editor showing a file named 'contact.py'. The code defines a function 'contactCustomer' that returns the string 'User Ram'. The file path 'C:/Users/Pradip/AppData/Local/Programs/Python/Python38...' is visible in the title bar.

```
def contactCustomer():
 return "User Ram"
```

### Step 5: Create a sales module in the shopping sub-package.

A screenshot of a Python code editor showing a file named 'sales.py'. It contains code for calculating shipping and tax. Two import statements are shown: an absolute import 'from ecommerce.customer import contact' and a relative import '#from ..customer import contact'. Arrows point from the text labels 'absolute import' and 'relative import' to their respective code lines.

```
from ecommerce.customer import contact
#from ..customer import contact
def calcShipping():
 return (contact.contactCustomer() + " shipping charge is " \
 + str(129))

def calcTax():
 pass
```

## **Step 6: Create a test module to execute the functionality.**

```
test.py - C:/Users/Pradip/AppData/Local/Programs/Python/Python38/books/intrapacka
File Edit Format Run Options Window Help
from ecommerce.shopping import sales
print(sales.calcShipping()) code

= RESTART: C:/Users/Pradip/AppDat
apackage/test.py output
User Ram shipping charge is 129
```

## **: EXERCISE-2 :**

1. Define module and package.
  2. Explain the advantages of the module.
  3. Explain the advantages of the package.
  4. List different ways to import a module.
  5. Explain different ways of importing modules.
  6. How can you alias a module in a Python program?
  7. What is the `dir()` function in Python?
  8. Explain the module search path concept.
  9. List out the steps to create user-defined packages?
  10. Explain different ways of importing packages.
  11. Explain the Intra-package reference concept.
  12. How do you install and uninstall Python packages?

## **Do the following (MCQs)**

- 1. Program code making use of a given module is called a \_\_\_\_\_ of the module.**

- (A) Client
- (B) Docstring
- (C) Interface
- (D) Modularity

- ## **2. What will be the output of the following Python code?**

```
from math import factorial
print(math.factorial(5))
```

- (A) 120
- (B) Nothing is printed
- (C) Error, method factorial doesn't exist in math module
- (D) Error, the statement should be: print(factorial(5))

**3. A module can be used in another module by importing it using \_\_\_\_\_ statement.**

- (A) modules
- (B) import
- (C) call
- (D) define

**4. Following is not an advantage of using modules**

- (A) Facilitates reuse of the program
- (B) Makes it convenient to test individual modules.
- (C) Reduces the program size
- (D) Facilitates modularity of the project.

**5. The Python module file should have a \_\_\_\_\_ extension**

- (A) py
- (B) python
- (C) pym
- (D) module

**6. Given a module "business" contains profit() function. Following are valid way to import and use the profit() function**

- |                                        |                                |
|----------------------------------------|--------------------------------|
| <b>(1) from business import profit</b> | <b>(2) import business</b>     |
| <b>res = profit()</b>                  | <b>res = business.profit()</b> |
| <b>(3) from business import profit</b> | <b>(4) import business</b>     |
| <b>res = business.profit()</b>         | <b>res = profit()</b>          |

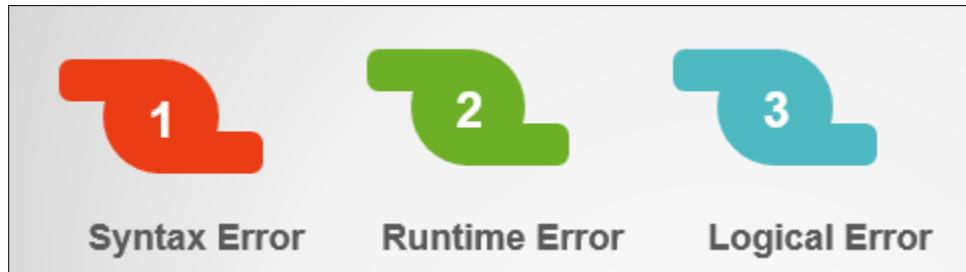
- (A) 1 and 2
- (B) 1 and 4
- (C) 3 and 4
- (D) 2 and 4

## Introduction to Exception

### 3.1.1.1 Error

- Error is something that is unexpected. Error will stop our program from executing.
- Errors are drawbacks of the program.

**There are three types of error.**



#### 3.1.1.1.1 Syntax Error

- Syntax errors are those that appear while we write our code.
- It is also known as Design time error.
- It is the most common type of error.
- Syntax errors are easy to track down in Python, because we get a red line pointing them out and we can fix them easily at coding time.



#### Example#

A screenshot of a Python code editor window titled 'er1.py - C:/Users/Shyam/AppData/Local/Programs'. The code in the editor is:

```
a=5
b=77

if a>b
 print(c)
else:
 print(b)
```

The line 'if a>b' is highlighted with a red background, indicating a syntax error. A small floating window titled 'SyntaxError' appears, showing a red 'X' icon and the message 'expected ':''. There is an 'OK' button at the bottom of the error window.

### 3.1.1.1.2 Runtime Error (Exception)

- As their name suggests, these errors occur when the program is executed.
- Runtime errors usually cause your program to crash.
- Python considers Runtime error as an Exception.
- Python provides try..except block to handle runtime error.



#### Example#

- Trying to open a file that doesn't exist.
- Trying to check email with an incorrect username or password.
- Dividing a number by 0.
- Users entering character data where a number is expected.

```
==== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe er1.py
Enter no1 =>11
Enter no2 =>abc
Traceback (most recent call last):
 File "C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe", line 1, in <module>
 a=int(input("Enter no1 =>"))
 File "C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe", line 1, in <module>
 b=int(input("Enter no2 =>"))
ValueError: invalid literal for int() with base 10: 'abc'

er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe
File Edit Format Run Options Window Help
a=int(input("Enter no1 =>"))
b=int(input("Enter no2 =>"))
if a>b:
 print(a)
else:
 print(b)
```

**When exceptions occur, Python Interpreter prints a four-line message on the screen.**

Line1: Traces the Python exception back to its source.

Line2: Displays the line number

Line3: Displays which line that caused the Python exception.

Line4: Displays the type of Python exception that has occurred.



### 3.1.1.1.3 Logical Error

- Logical errors are those that appear at run time. They happen when your code doesn't behave quite the way you thought it would.
- It is a programmer's logical mistake.
- Logical errors are generally difficult to find.

#### Example#

Instead of doing Multiplication we write the logic of division then.

ans=5 \* 10; #instead of the symbol of addition +, programmer write X that is symbol of multiplication.



### 3.2.1 Types of Exception

There are two types of exceptions:

Built In

User defined



#### 3.2.1.1 Library Exception Classes/Built in Exception

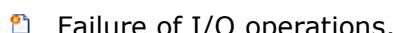
Runtime error is also known as an exception.

Exception is an unexpected situation which occurs at any time and we can handle that situation at run time. An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. When an Exception occurs, interpreter stops the current process and passes it to the calling process until it is handled. If it is not handled, the program will crash.

Exceptions can arise due to a number of situations like

- Trying to access the element from a specified position of a List but the position does not exist in that List.
- When a number is divided by zero.
- Trying to access a file which is not present.





## Some built-in exceptions:

| Exception                | Meaning                                                                                    |
|--------------------------|--------------------------------------------------------------------------------------------|
| IOError                  | Raised when an input/output operation fails.                                               |
| Arithmetic Error         | Raised when numeric calculations fail.                                                     |
| Floating-point Error     | Raised when a floating-point calculation fails.                                            |
| Zero Division Error      | Raised when the second operand of division or modulo operation is zero.                    |
| Assertion Error          | Raised when the assert statement fails.                                                    |
| Overflow Error           | Raised when the result of an arithmetic operation is too large to be represented.          |
| Import Error             | Raised when the imported module is not found.                                              |
| Index Error              | Raised when the index of a sequence is out of range.                                       |
| Keyboard Interrupt Error | Raised when the user interrupts program execution, usually by pressing (Ctrl+C or Delete). |
| Indentation Error        | Raised when there is an incorrect indentation.                                             |
| Syntax Error             | Raised by the parser when a syntax error is encountered.                                   |
| Key Error                | Raised when the specified key is not found in the dictionary.                              |
| Name Error               | Raised when an identifier is not found in the local or global namespace.                   |
| Type Error               | Raised when a function or operation is applied to an object of an incorrect type.          |
| TabError                 | Raised when indentation consists of inconsistent tabs and spaces.                          |
| Value Error              | Raised when a function gets an argument of correct type but improper value.                |
| Runtime Error            | Raised when a generated error does not fall into any category                              |

### Examples of Built in Exceptions

#### Example# Here is an example of Value Error when we enter a wrong data type.

```
===== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe
Enter no1 =>11
Enter no2 =>abc
Traceback (most recent call last):
 File "C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe", line 3, in <module>
 b=int(input("Enter no2 =>"))
 ^
ValueError: invalid literal for int() with base 10: 'abc'

er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8.5/python.exe
File Edit Format Run Options Windows
a=int(input("Enter no1 =>"))
b=int(input("Enter no2 =>"))
print(a/b)
```

#### Example# ZeroDivision Error example when we divide a number by 0.

```
===== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/
er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/
Enter no1 =>11
Enter no2 =>0
Traceback (most recent call last):
 File "C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/
 print(a/b)
ZeroDivisionError: division by zero
```

#### Example# Index Error Illustration when we try to call a position that does not exist.

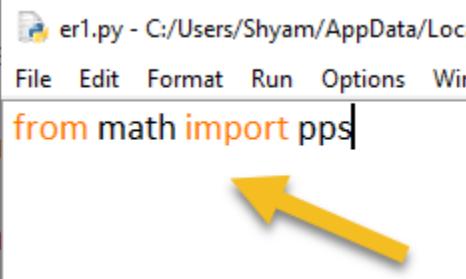
```
===== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/
er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/
Traceback (most recent call last):
 File "C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/
 print(friendList[4])
IndexError: list index out of range
```

#### Example# When we are trying to add a number with a string or if we enter wrong input then we will get TypeError.

```
===== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/
er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/
Traceback (most recent call last):
 File "C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/
 print(a+b)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

#### Example# When we are trying to import wrong module then we will get ImportError.

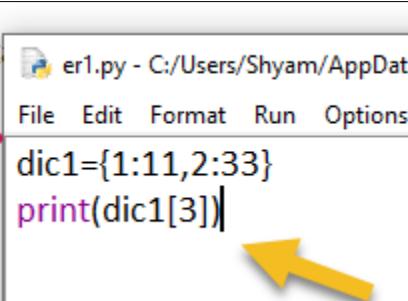
```
==== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe
Traceback (most recent call last):
 File "C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe", line 1, in <module>
 from math import pps
ImportError: cannot import name 'pps' from 'math' (which is not a package)
```



```
er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe
File Edit Format Run Options Window
from math import pps
```

**Example# Here is an example of KeyError. It is same as IndexError but KeyError is for dictionaries.**

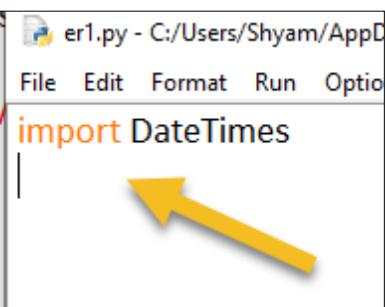
```
==== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe
Traceback (most recent call last):
 File "C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe", line 1, in <module>
 print(dic1[3])
KeyError: 3
```



```
er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe
File Edit Format Run Options Window
dic1={1:11,2:33}
print(dic1[3])
```

**Example# Here is an example of ModuleNotFoundError. It happens when Module is not installed in your environment.**

```
==== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe
Traceback (most recent call last):
 File "C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe", line 1, in <module>
 import DateTimes
ModuleNotFoundError: No module named 'DateTimes'
```



```
er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8/python.exe
File Edit Format Run Options Window
import DateTimes
```

### 3.2.1.2 User defined Exception

On first look we feel that why do we need it? So, let's see some examples where we can raise our own exception, which Python Interpreter is unable to think.

- 1) If entered Password and Retyped password does not match.
  - 2) If entered name's length is less than 2.
  - 3) If entered subject mark is less than 0.
  - 4) If entered email address is not valid.
  - 5) If entered age is less than 18.
- etc.

When we feel this should be an exception and Python Interpreter does not, in that situation we need to create our own custom exceptions which will be the derived class of an Exception class. In other way we can say it is a customized exception as per the users need.



| Syntax#                                          | Example#                                             |
|--------------------------------------------------|------------------------------------------------------|
| class <b>classErrorName</b> (Exception):<br>pass | <b>class InvalidMarks(Exception):</b><br><b>pass</b> |
| Calling Syntax#                                  | Calling Syntax#                                      |
| raise classErrorName                             | raise InvalidMarks                                   |

Will see example of it on [Page Number](#)

### Handling Exceptions

In Python, if we want to handle runtime exception, then we can use below block:

```
try:
 logic
except:
 Error Message
else:
 No Error
finally:
 I will always run
```

Now Let's see each block Step by Step

## Example# Demo of try...except...else....finally

```
ModuleNotFoundError: No module named 'DateTime'

===== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.10/python.exe er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.10/python.exe
Enter no1 =>22
Enter no2 =>2
11.0
in else block , all good
in finally block

===== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.10/python.exe
Enter no1 =>22
Enter no2 =>abc
Error
in finally block

try:
 a=int(input("Enter no1 =>"))
 b=int(input("Enter no2 =>"))
 ans=a/b
 print(ans)
except:
 print("Error")
else:
 print("in else block , all good")
finally:
 print("in finally block")
```

In above program, we tried to perform Division of two numbers, according to argument we need to pass number1 and number2. First time we entered number1 and number2 with two proper inputs and output displayed but in second time by mistake we entered "abc", so it throws an exception.

try

- The 'try' block contains a logic of the program within where an exception might occur.
  - A 'try' block must be followed by either an 'except' block or a 'finally' block.
  - If any exception occurs in 'try' block then control of the program shifts from there and searches for proper 'except' block for throwing an exception.
  - Try block can be nested as in one try block can contain another try block.



|                                                                                                                                        |                                                                |                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| try:<br>logic...<br>                                                                                                                   | try:<br>logic..<br>except:<br>print("Error")<br>               | try:<br>logic....<br>except ZeroDivisionError:<br>print("Dont enter zero")<br>except ArithmeticError:<br>print("ArithmetricError")<br>except:<br>print("Error")<br> |
| try:<br>logic....<br>except ZeroDivisionError:<br>print("Dont enter zero")<br>except ArithmeticError:<br>print("ArithmetricError")<br> | try:<br>logic....<br>else:<br>print("No Error")<br>            | try:<br>logic....<br>else:<br>print("No Error")<br>                                                                                                                 |
| try:<br>a=5<br>except:<br>print("Error")<br>else:<br>print("in else block , all good")<br>                                             | try:<br>logic....<br>finally:<br>print("in finally block")<br> | try:<br>Logic...<br>except:<br>print("Error")<br>else:<br>print("in else block , all good")<br>finally:<br>print("in finally block")<br>                            |
| except                                                                                                                                 |                                                                |                                                                                                                                                                     |

- 'except' block will execute exception that occurs in try block.
- It's an optional block.

- If an exception occurs, Python Interpreter examines the 'except' statements. If it finds an 'except' statement that handles the generated exception, it executes the corresponding statement block.
- When we want different reaction in different runtime error, we can use multiple 'except' statement in single try block.
- We can write any number of 'except' statements or none. We can handle specific exception.
- At a time only one 'except' block will execute out of multiple 'except' blocks.
- An 'Exception' class is base of all other exceptions. Whenever we are writing multiple catch we should write 'except' of Exception class in the last.
- The 'except' block is required with a 'try' block, even if it contains only the pass statement.
- We can write multiple except block in one line also.



```
try:
 Logic...
except:
 print("Error")
```

✓

```
try:
 Logic...
except:
 print("Error")
except ZeroDivisionError:
 print("Error")
except ValueError:
 print("Error")
```

✗

```
try:
 logic..
except ZeroDivisionError:
 print("Error")
except ValueError:
 print("Error")
except:
 print("Error")
```

✓

```
try:
 logic...
except(ArithmeticError, ValueError):
 print("Exception")
except:
 print("Other Error")
```

✓

```
try:
 logic...
except(ArithmeticError, ValueError):
 print("Exception")
```

✓

## else

- Code in the 'else' block is only executed if no exceptions were raised in the try block.
- It is an optional block



## finally

- 'finally' block is an optional last part of try...except block.
- The 'finally' block provides the assurance of execution of some important code that must be executed whether exception is handled or not.
- We can not write more than one 'finally' block.
- 'finally' block is a better place for clean-up actions.
- For example, closing file handles, closing connection, stream and freeing up other resources.



|                                                                                                             |                                                                                                        |                                                                                                                                                       |                                                                                                         |                                                                                                                                                                        |                                                                                                        |
|-------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <pre>try:<br/>    logic..<br/>except:<br/>    print("Error")<br/>finally:<br/>    print("in finally")</pre> | A green checkmark icon is positioned to the right of the first code snippet, indicating it is correct. | <pre>try:<br/>    logic...<br/>except:<br/>    print("Error")<br/>finally:<br/>    print("in finally")<br/>finally:<br/>    print("in finally")</pre> | A large red 'X' icon is positioned to the right of the middle code snippet, indicating it is incorrect. | <pre>try:<br/>    Logic...<br/>except:<br/>    print("Error")<br/>else:<br/>    print("in else block , all good")<br/>finally:<br/>    print("in finally block")</pre> | A green checkmark icon is positioned to the right of the third code snippet, indicating it is correct. |
|-------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|

## Let's see Some Example of Try...except...else...finally block

Example 1 # Enter two numbers and display division of it, also use proper exception handling as required.

```
===== RESTART: C:/Users/Shya/PycharmProjects/Python/er1.py =====
Enter no1 =>22
Enter no2 =>0
Why You Entered Zero
in finally block
=====
===== RESTART: C:/Users/Shya/PycharmProjects/Python/er1.py =====
Enter no1 =>33 output
Enter no2 =>abc
Value Error
in finally block
=====
===== RESTART: C:/Users/Shya/PycharmProjects/Python/er1.py =====
Enter no1 =>22
Enter no2 =>2
11.0
in else block , all good
in finally block
```

```
try:
 a=int(input("Enter no1 =>"))
 b=int(input("Enter no2 =>"))
 ans=a/b
 print(ans)
except ZeroDivisionError:
 print("Why You Entered Zero")
except ValueError:
 print("Value Error")
except:
 print("Other Error") code
else:
 print("in else block , all good")
finally:
 print("in finally block")
```

Example 2# Find Area of Triangle. Enter height, base and display its area, also use proper exception handling as required.

|                                                                                                             |                                                                                                     |
|-------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| <pre>==== RESTART: C:/Users/Shya Enter Height =&gt;100 Enter Base =&gt;200 Area of Triangle = 10000.0</pre> | <pre>er1.py - C:/Users/Shya/AppData/Local/Programs/P File Edit Format Run Options Window Help</pre> |
| <pre>&gt;==== RESTART: C:/Users/Shya Enter Height =&gt;100 Enter Base =&gt;abc Error</pre>                  | <p>output</p>                                                                                       |
|                                                                                                             | <p>code</p>                                                                                         |

Example 3# Create a function for addition() with two arguments , Pass two values and display it's addition , also use proper exception handling as required.

|                                                       |                                                                                                     |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| <pre>==== RESTART: C:/Users/Shya Add = 24 Error</pre> | <pre>er1.py - C:/Users/Shya/AppData/Local/Programs/P File Edit Format Run Options Window Help</pre> |
| <p>output</p>                                         | <pre>def add(a,b):     try:         print("Add =",a+b)     except:         print("Error")</pre>     |
|                                                       | <p>code</p>                                                                                         |

Example 4# Create a function for subtraction() with two arguments, pass two values and display its subtraction also use proper exception handling as required.

```
===== RESTART: C:/Users/
Sub = 20

```

Other Error

output

```
*er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Py
File Edit Format Run Options Window Help

def subtraction(a,b):
 try:
 c=a-b
 print("Sub =",c)
 except(ArithmetricError, ValueError):
 print("Exception")
 except:
 print("Other Error")
 else:
 print("*"*20) code

subtraction(22,2)
subtraction(22,"abc")
```

### 3.4.1 Raising Exceptions

Using raise keyword, we can raise an exception from our code. The raise statement allows the programmer to force a specific exception to occur.

#### Syntax#

```
Raise ErrorName("Argument")
```

**Example 1#** Create a function for division of two numbers, if the value of any argument is non-integer, then raise the error or if second argument is 0 then raise the error.

|                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| =====<br>==== RESTART: C:/Users/Shyam/AppData/L<br>Div = 11.0<br>*****<br>Please Enter Integer Only in Second value<br>*****<br>Please Enter Integer Only in First value<br>*****<br>We can't divide by 0.<br><br>output | <pre>File Edit Format Run Options Window Help<br/><br/>def div2numbers(x, y):<br/>    try:<br/>        if type(x) is not int:<br/>            raise TypeError("Please Enter Integer Only in First value")<br/>        elif type(y) is not int:<br/>            raise TypeError("Please Enter Integer Only in Second value")<br/>        elif y==0:<br/>            raise ValueError("We can't divide by 0.")<br/>        else:<br/>            print("Div =", x / y)<br/>    except TypeError as e:<br/>        print(e)<br/>    except ValueError as e:<br/>        print(e)<br/>    except:<br/>        print("Error")<br/><br/>div2numbers(22, 2)<br/>print("*"*20)<br/>div2numbers(22, "abc")<br/>print("*"*20)<br/>div2numbers("abc", 2)<br/>print("*"*20)<br/>div2numbers(22, 0)</pre> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Example 2#** Create a function for counting Result which contains two arguments English and Hindi , if the value of any argument is less than 0 then raise the error

```
==== RESTART: C:/Users/Shyam/AppData
Total Marks = 70

Please Enter Positive Marks in Hindi

Please Enter Positive Marks in English

```

output

er1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python310/er1.py (3.10.2)

```
File Edit Format Run Options Window Help
def MarksResult(EngMarks, HindiMarks):
 try:
 if EngMarks<0:
 raise ValueError("Please Enter Positive Marks in English")
 elif HindiMarks<0:
 raise ValueError("Please Enter Positive Marks in Hindi")
 else:
 print("Total Marks = ", EngMarks+HindiMarks)
 except ValueError as e:
 print(e)
 except:
 print("Error")
```

code

```
MarksResult(30,40)
print("*"*20)
MarksResult(30,-2)
print("*"*20)
MarksResult(-2,40)
print("*"*20)
```

Example 3# Enter an address and if the length of the address is less than 10 characters then display an error.

```
try:
 address=input("Enter Your Address :")

 if len(address)<10:
 raise Exception
 else:
 print("Your Address ",address)|

except:
 print("Please enter valid address")
```

code

```
Enter Your Address : 12 Hazar Dastan Near Post Office Dal Lake Srinagar
Your Address 12 Hazar Dastan Near Post Office Dal Lake Srinagar
```

```
Enter Your Address : 32
Please enter valid address
```

output



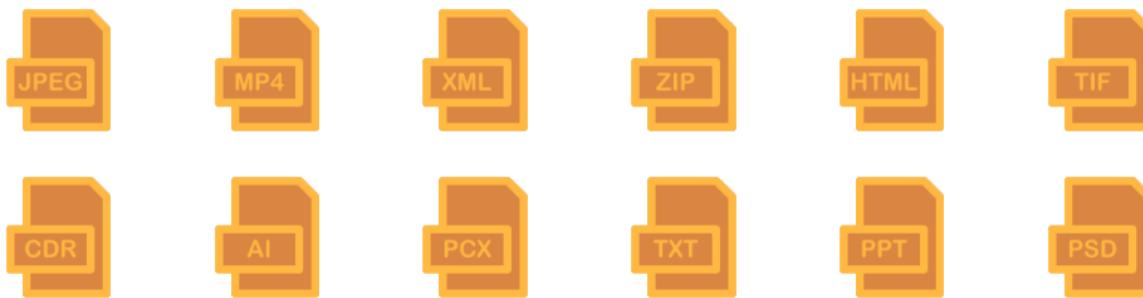
## Introduction to files and their types.

Python provides a way to read or write the data from the secondary storage devices in the form of a file.

A file is a collection of bytes stored on a secondary storage device, which is generally a disk.

The collection of bytes may be Characters, words, lines, paragraphs.

The file is created for the permanent storage of data. It is a readymade structure. There are different types of extensions available for the file.



To work with files, Python provides many built-in library functions.

There are two types of files:

- Binary files
- Text files

Binary and Text files contains data stored as a series of 1s and 0s. Let's see the difference between them.



### 4.1.1.1 Difference between Text and Binary File

| Text File                                    | Binary File                                                                            |
|----------------------------------------------|----------------------------------------------------------------------------------------|
| The bits in text files represent characters. | The bits in binary files represent custom data.                                        |
| Text files contain alphabets, numbers, etc.  | Binary files typically contain a sequence of bytes or ordered groupings of eight bits. |

|                                                        |                                                                                                       |
|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| In-text file it's less prone to get corrupt as changes | Binary files it is easily get corrupted                                                               |
| Can store only plain text in a file                    | Can store text, image, audio, etc.                                                                    |
| A text file can open in any basic editor.              | Binary files need a specific application. For example, to play any music file we need a music player. |
| .txt or .rtf are used as extensions to text files.     | Many different extensions. For example .jpg,.zip,.mp4, etc.                                           |
| Occupies less memory                                   | Occupies more memory                                                                                  |
| It's stores data in a human-readable format.           | It's not a human-readable format.                                                                     |

#### 4.1.1.2 Operation on File

There are various operations we can do with the file.

1. Creating a file
2. Opening a file
3. Reading from a file
4. Writing data into a file
5. Closing a file



To Work with File, we need to create a reference as given below:

| Syntax#                      | Example#                     |
|------------------------------|------------------------------|
| f1=open (FilePath, FileMode) | f1=open("f:\\abcde.txt",'r') |

#### 4.1.1.3 File Mode

Python provides some modes related to File operation.

| Mode | Description                                                |
|------|------------------------------------------------------------|
| r    | The default mode. It opens the file to read-only mode.     |
| rb   | Opens the file to read-only in binary format.              |
| r+   | Opens the file to read and write both.                     |
| rb+  | It opens the file to read and write both in binary format. |

|     |                                                                                                                                                                                                |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| w   | Opens the file to write only. It overwrites the file if any file previously exists with the same name or it creates a new file if the file does not exist.                                     |
| wb  | Opens the file to write only in binary format. It overwrites the file if it exists with the same name previously or it creates a new file if the file does not exist.                          |
| w+  | Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, it creates a new file for reading and writing.                         |
| wb+ | Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, it creates a new file for reading and writing.        |
| a   | Opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any.                                                                            |
| ab  | Opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name. |
| a+  | Opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.                            |
| ab+ | Opens a file to append and read both in binary format. The file pointer remains at the end of the file.                                                                                        |

## 4.2.1 Opening and Closing Text File

### 4.2.1.1 File Open

The open() function has three arguments filename, mode, and encoding. The only argument that is compulsory is the first one that is the filename, the second argument is file mode that one is optional, the default file mode is 'r' read.

#### Syntax#

```
FileVariable = open("FileName", "Mode")
```

#### Example#

```
F1 = open("abc.txt")
F1 = open("data.txt",'w')
```



### 4.2.1.1.2 File Close

To close a file we can use close() method.

#### Syntax#

Filename.close()

**Example#**

F1.close()



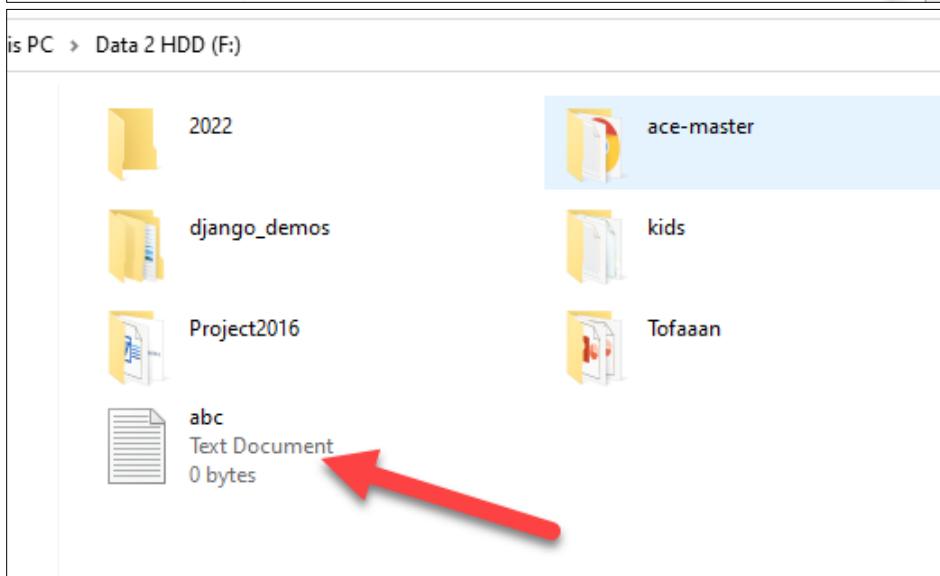
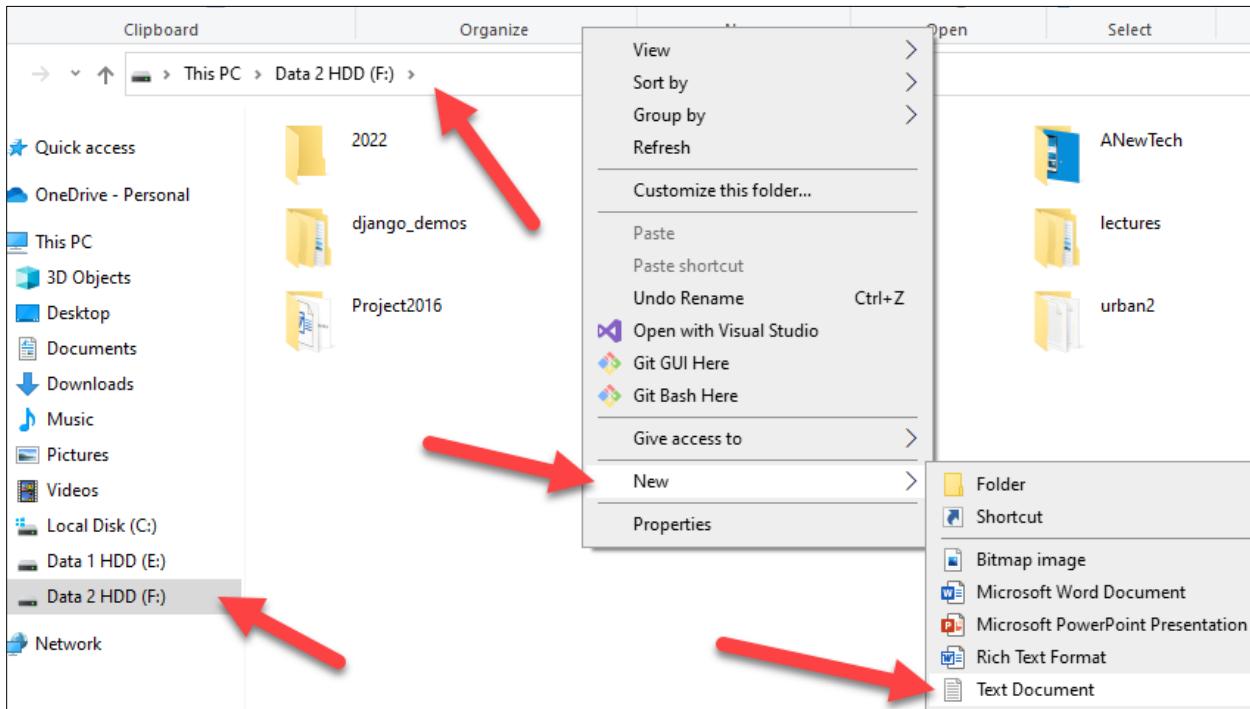
## 4.2.2 Reading and Writing Files

### 4.2.2.1 Reading File

There are four methods to read the contents of the file.

| Function Name  | Details                                                                                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| read()         | reads all the characters from the beginning of the file.                                                                                                              |
| read(position) | reads the characters from the beginning till the position specified in the parameter.<br><br>When you want to work with each character at a time this option is good. |
| readline()     | reads the characters starting from the newline character.                                                                                                             |
| readlines()    | reads all lines from the starting returns a List                                                                                                                      |

**Let's create a text file named abc.txt and add some content to it. So just right-click and create a text file.**



**Open the file and add some content to it.**



abc - Notepad

- □ ×

File Edit Format View Help

A computer is a digital electronic machine that can be programmed to carry out sequences of arithmetic or logical operations automatically. Modern computers can perform generic sets of operations known as programs. These programs enable computers to perform a wide range of tasks. A computer system is a "complete" computer that includes the hardware, operating system, and peripheral equipment needed and used for "full" operation. This term may also refer to a group of computers that are linked and function together, such as a computer network or computer cluster.|

Ln 1, Col 569

100%

Windows (CRLF)

UTF-8

Example# Read all the characters from the file using read() and display its content on the screen.

```
==> RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.8/file1.py
A computer is a digital electronic machine that can be programmed to perform logical operations automatically. Modern computers can store large amounts of data and execute programs. These programs enable computers to perform a "complete" computer that includes the hardware, operating system and used for "full" operation. This term may also refer to a group of computers that are connected together, such as a computer network or computer cluster.
output
```

```
*file1.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8/file1.py
File Edit Format Run Options Window Help
f1=open("f:\\abc.txt","r")
data=f1.read()
print(data)
f1.close()
```

Example# Read all the characters from the file using the read(1) method and display its content on the screen.

```
C:/Users/Shyam/AppData/Local/Programs/Python/3.8/file1.py
A computer is a digital electronic machine that can be programmed to perform logical operations automatically. Modern computers can store large amounts of data and execute programs. These programs enable computers to perform a "complete" computer that includes the hardware, operating system and used for "full" operation. This term may also refer to a group of computers that are connected together, such as a computer network or computer cluster.
output
```

```
*file1.py - C:/Users/Shyam/AppData/Local/Programs/Python/3.8/file1.py
File Edit Format Run Options Window Help
f1=open("f:\\abc.txt","r")
while True:
 c=f1.read(1)
 if not c:
 break
 print(c,end="")
f1.close()
```

Example# Read all the characters from the file using the read(1) method and display how many vowels are there in the file.

```
== RESTART: C:/Users/Shyam/
Total Vowels are: 181
```

output

```
*file1.py - C:/Users/Shyam/AppData/Local/Programs... — □
File Edit Format Run Options Window Help
f1=open("f:\\\\abc.txt","r")

vowelcount=0

vowelList=['a','e','i','o','u','A','E','I','O','U']

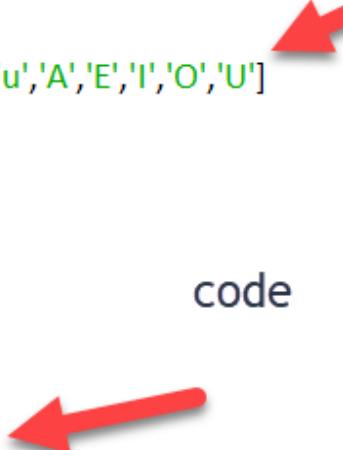
while True:
 c=f1.read(1)
 if not c:
 break

 if c in vowelList:
 vowelcount+=1

f1.close()

print("Total Vowels are: ",vowelcount)
```

code



Example# Read all the characters from the file using the read(1) method and display the count of how many uppercase and lowercase characters are there in the file.

|        |                                                                                                                                                                                                                                                                                                              |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | File Edit Format Run Options Window Help<br>=== RESTART: C:/Users/Shya<br>Total Upper are: 5<br>Total Lower are: 463                                                                                                                                                                                         |
| output | <pre> f1=open("f:\\abc.txt","r") uppercount=0 lowercount=0  while True:     c=f1.read(1)      if not c:         break      if c.isupper():         uppercount+=1      elif c.islower():         lowercount+=1  f1.close() print("Total Upper are: ",uppercount) print("Total Lower are: ",lowercount) </pre> |

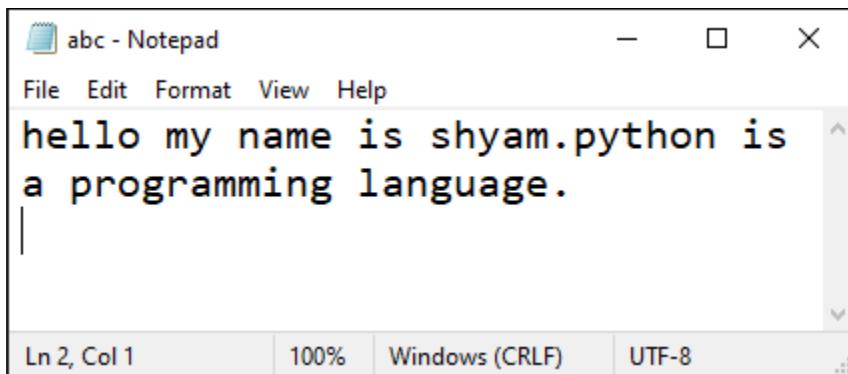
Example# Example of Readline() function which reads only one line from the File.

|        |                                                                                                                                                                                 |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | A computer is a digital electronic machine that can be programmed to carry out a sequence of operations that can be repeated, often with branching and loops.                   |
| output | <pre> file1.py - C:/Users/Shyam/AppData/Local/Programs/Py File Edit Format Run Options Window Help<br/> f1=open("f:\\abc.txt","r")  c=f1.readline() print(c)  f1.close() </pre> |

Example# Using Readlines() function read all the lines from the Text file. It returns data in a List.

```
file1.py - C:/Users/Shyam/AppData/Local/Program
==== RESTART: C:/Users/Shyam/AppData/Local/Program
['A computer is a digital electronic machine th
c or logical operations automatically. Modern
as programs. These programs enable comput
a "complete" computer that includes the har
and used for "full" operation. This term may
ion together, such as a computer network or
output
f1=open("f:\\abc.txt","r")
c=f1.readlines()
print(c)
f1.close()
code
```

Example# Using Readlines() function read all the lines from the Text file. It returns a List and displays each word in ascending order from the file.



```
f1=open("f:\\abc.txt",'r')
```

```
c=f1.readlines()
```

```
f1.close()
```

```
print("Contents of File")
```

```
print(c)
```

```
list1=c[0].split()
```

```
print("After Split List1")
```

```
print(list1)
```

code

```
list1.sort()
```

←

```
print("After Sort List1")
```

```
print(list1)
```

Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)]  
Type "help", "copyright", "credits" or "license()" for more information.

```
>>> === RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python310/
```

**Contents of File**

```
['hello my name is shyam.python is a programming language.\n']
```

**After Split List1**

```
['hello', 'my', 'name', 'is', 'shyam.python', 'is', 'a', 'programming', 'language.']
```

**After Sort List1**

```
['a', 'hello', 'is', 'is', 'language.', 'my', 'name', 'programming', 'shyam.python']
```

output

#### 4.2.2.2 Writing File

There are two methods to read the content of the file.

| Function Name | Details                      |
|---------------|------------------------------|
| write()       | Writes data in the file      |
| writeline()   | Writes List data in the file |

Example# Let's create a file and write some content in it.

File1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python310/file1.py

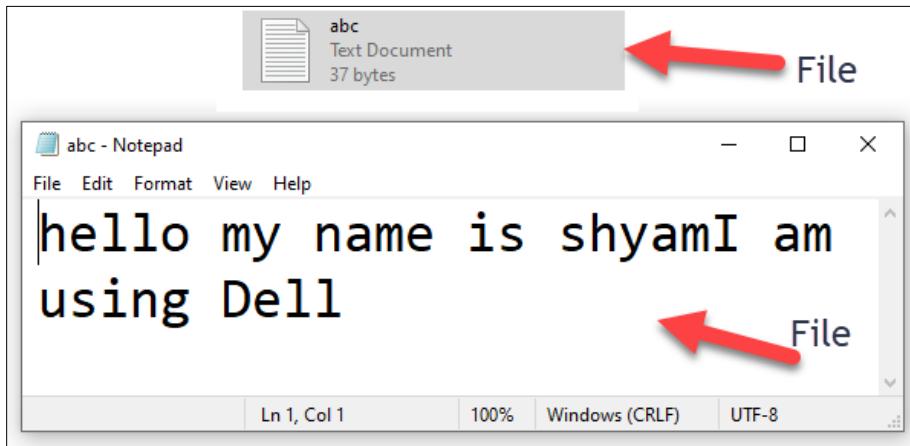
File Edit Format Run Options Window Help

```
== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python310/file1.py
File Created
```

output

```
f1=open("f:\\abc.txt","w")
f1.write("hello my name is shyam")
f1.write("I am using Dell")
f1.close()
print("File Created")
```

code

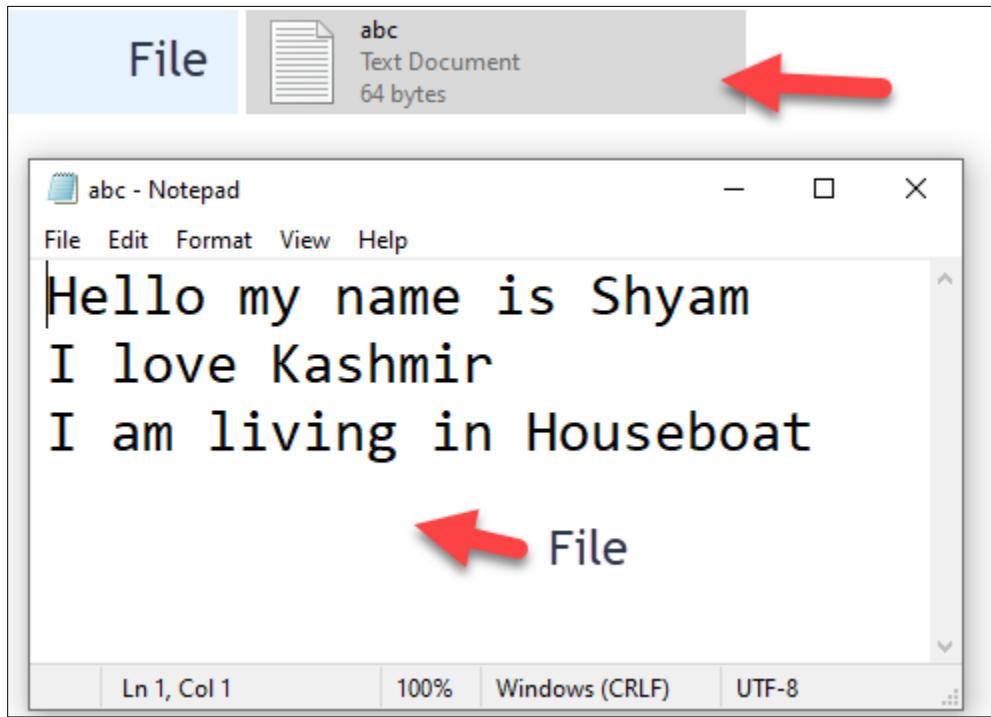


Example# Let's create a file and write some content with \n (newline).

A screenshot of a terminal window. On the left, the output section shows "==== RESTART: C:" followed by "File Created" in blue text, with a red arrow pointing to it. On the right, the code section shows a Python script named "file1.py" with the following content:

```
file1.py - C:/Users/Shyam/AppData/Local/Programs/Python
File Edit Format Run Options Window Help
f1=open("f:\\abc.txt","w")
f1.write("Hello my name is Shyam")
f1.write("\nI love Kashmir")
f1.write("\nI am living in Houseboat")
f1.close()
print("File Created")
```

The word "code" is written next to the script content.



Example# Let's copy content from one file to another file using the write() function.

File Created  
==== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/3.9/python.exe file1.py

```
f1=open("f:\\abc.txt",'r')
f2=open("f:\\pqr.txt",'w')
while True:
 c=f1.read(1)
 if not c:
 break
 f2.write(c)

f1.close()
f2.close()
print("Copied")
```

output

code



Example# Write a program to read the content of a file character by character and write it to another file except for vowels.

```
== RESTART: C:/Users/Sh
Copied without Vowels
```

output

```
File Edit Format Run Options Window Help
f1=open("f:\\abc.txt",'r')
f2=open("f:\\vowel.txt","w")

listvowel=['a','e','i','o','u','A','E','I','O','U']

while True:
 c=f1.read(1)
 if not c:
 break
 if c not in listvowel:
 f2.write(c)

f1.close()
f2.close()
print("Copied without Vowels")
```

code



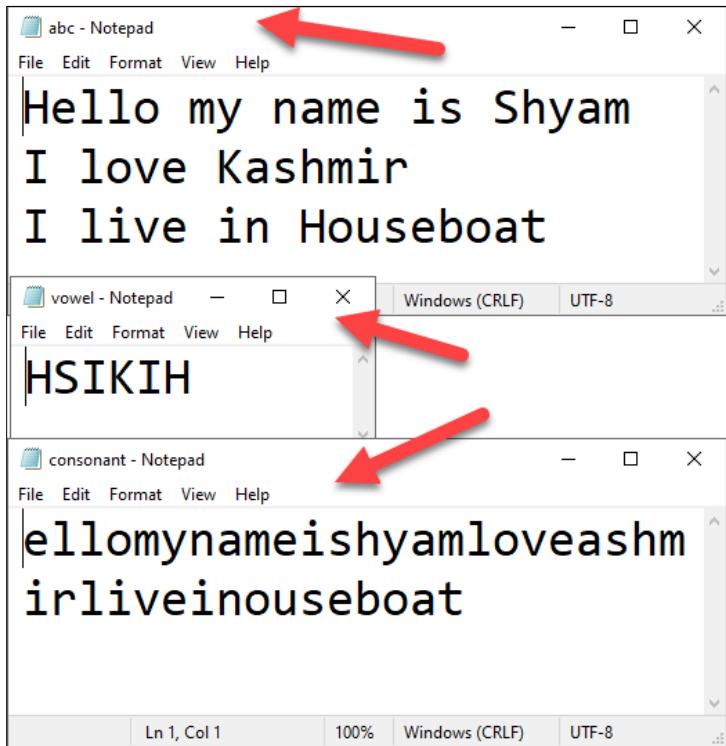
Two windows are shown side-by-side. The left window is titled 'abc - No...' and contains the text 'hello', 'myname is', and 'shyam'. The right window is titled 'vowel - ...' and contains the text 'hll', 'mynm s', and 'shym'. A red arrow points from the word 'shyam' in the 'abc' window to the word 'shym' in the 'vowel' window.

|                            |                            |
|----------------------------|----------------------------|
| abc - No...                | vowel - ...                |
| File Edit Format View Help | File Edit Format View Help |
| hello                      | hll                        |
| myname is                  | mynm s                     |
| shyam                      | shym                       |
| Windows (CRLF) UTF-8       | Windows (CRLF) UTF-8       |

Example# Read the data from a file and separate the uppercase character and lowercase character into two separate files.

```
file1.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/file1.py
File Edit Format Run Options Window Help
== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/file1.py ==
Copied
output
code
f1=open("f:\\abc.txt",'r')
f2=open("f:\\vowel.txt",'w')
f3=open("f:\\consonant.txt",'w')
while True:
 c=f1.read(1)
 if not c:
 break
 if c.isupper():
 f2.write(c)
 elif c.islower():
 f3.write(c)
 f1.close()
 f2.close()
 f3.close()
print("Copied")
```

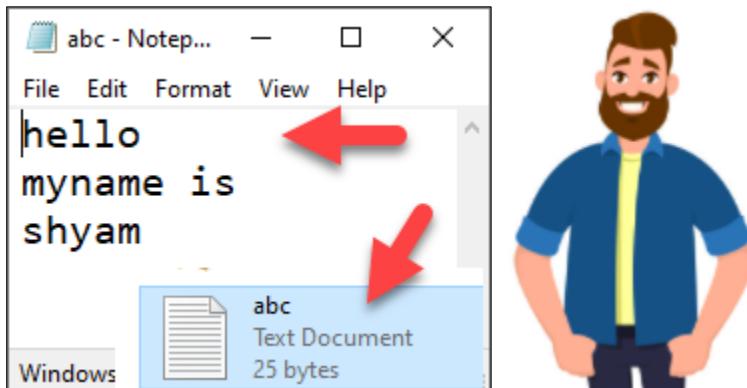




Example# Let's write data using writelines() method.

```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15)
Type "help", "copyright" or "credits" for more information.
Type "exit" or "quit" to quit.
==== RESTART: C:\Users\Shyam\AppData\Local\Programs\Python\Python310\file1.py ====
Copied
output
code
print("Copied")
```

The code above shows a Python script named file1.py. It prints the string "Copied" to the console. The output is labeled "output" and the code is labeled "code".



#### 4.2.2.3 Append mode of File

- In the above examples, we have seen the use of reading and write modes of a file, the read mode only opens an existing file, and the write mode creates a new file if the file doesn't exist. But if the file is found, it overwrites the context of the file.
- If we want to add data to an existing file, and if you don't want to overwrite the content of the file, we can open the file in append mode. The file can be opened in append mode using 'a' or 'a+' as a parameter inside the open() method.
- Here, the 'a' suggests opening the file in appending mode only that is open the file for writing. The file is created if it does not exist.
- By using the 'a+' we can open the file for reading and writing. The file is created if it does not exist.
- The data will be inserted at the end, after the existing data inside the file.

```
f1=open("f:\\abc.txt","a") ←
f1.write("\nhello my name is shyam")
f1.close()
```

#### 4.3.1 Setting Offsets in File

The file is associated with the file pointer. The file pointer returns the current position of the file. That current position is known as an offset.

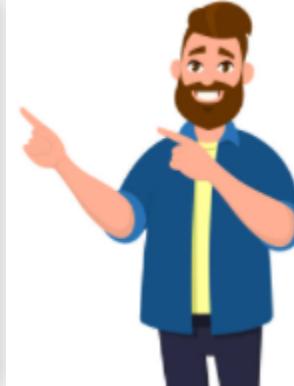
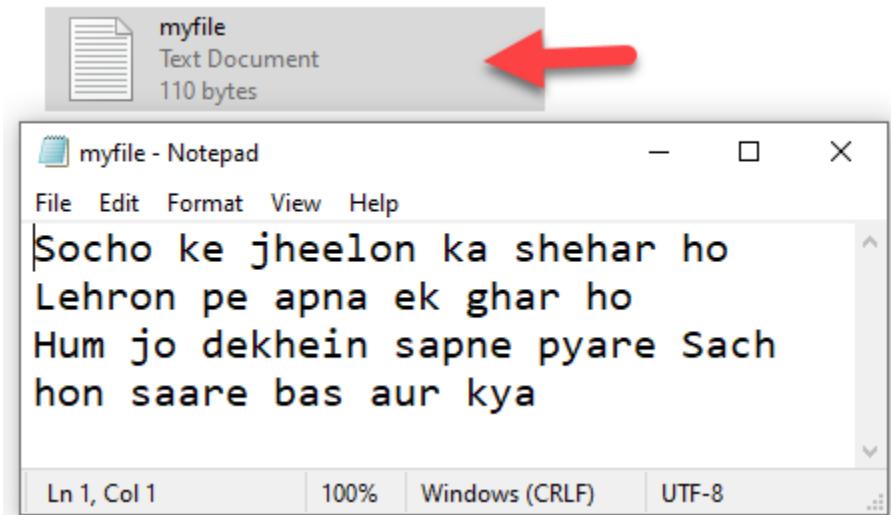
Offset refers to the new position of the file pointer within the file.

There are two methods associated with offsets.

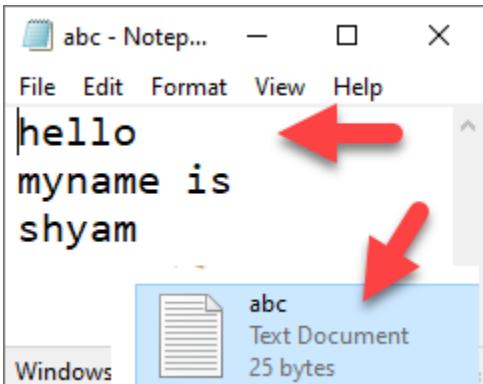
| Method             | Description                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tell()             | Returns an integer value which is the file position in the file stream.<br>Syntax#<br>file1.tell()                                                                                                                                                                                                                                     |
| seek(offset, from) | Enables us to modify the current file pointer position in the file stream. There are two arguments of seek() offset and from.<br>Syntax#<br>Pos=file1.seek(offset, from)<br>Offset: Number of positions to move forward.<br>from: It indicates the reference position from where the bytes are to be moved. It's an optional argument. |

Example# Read data from abc.txt, before reading the content, display the position of the file pointer, and after reading the data display the position of the file pointer.

|                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                   |      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| ==== RESTART: C:/Users/Shyam/AppData/From the starting position file1.tell(): 0After line1 position file1.tell(): 55After line2 position file1.tell(): 110<br><br>output | file1 = open("f:\\myfile.txt", "w")<br><br>print("From the starting position file1.tell(): ", file1.tell())<br><br>file1.write("Socho ke jheelon ka shehar ho Lehron pe apna ek ghar ho")<br><br>print("After line1 position file1.tell(): ", file1.tell())<br><br>file1.write("\nHum jo dekhein sapne pyare Sach hon saare bas aur kya")<br> <br>print("After line2 position file1.tell(): ", file1.tell())<br><br>file1.close() | code |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|



Example# Read data from abc.txt, Before reading the content, display the position of the file pointer, and after reading the data display the position of the file pointer.



```
The position of file pointer is : 0
hello
myname is
shyam
```

output

After reading all the contents, the filepointer is at : 25

```
file1.py - C:\Users\Shyam\AppData\Local\Programs\Python\Python310\file1.py (3.10.2)
File Edit Format Run Options Window Help
f1 = open("f:\\abc.txt","r")
|
print("The position of file pointer is :",f1.tell())

data = f1.read()

print(data)

print("After reading all the contents, the filepointer is at : ",f1.tell())
```

Example# Read data from NewFile.txt, Before reading the content, display the position of the file pointer, and after reading the data display position of the file pointer.



```
NewFile - N... — X
File Edit Format View Help
Srinagar
Pahalgam
Doodhpatri
Gulmarg
Window NewFile
Text Document
41 bytes
```

| code                                          | output                                                                                                                                  |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| file content from 8th position...             | print("file content from 8th position...")<br>file1.seek(8)<br>print(file1.read())                                                      |
| file content from 0th position...             | print("file content from 0th position...")<br>file1.seek(0)<br>print(file1.read())                                                      |
| file content from 20th position...            | print("file content from 20th position...")<br>file1.seek(20)<br>print(file1.read())                                                    |
| Srinagar<br>Pahalgam<br>Doodhpatri<br>Gulmarg | ==== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32/f3.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python37-32 |
|                                               | file1 = open("f:\\NewFile.txt", "r")<br>print("File Data")<br>print(file1.read())                                                       |
| Pahalgam<br>Doodhpatri<br>Gulmarg             |                                                                                                                                         |
|                                               |                                                                                                                                         |

Example# Write content in a file then read up to 17 characters and write again from the starting and print that. Use the seek() method.

```
file2.py - C:/Users/Shyam/AppData/Local/Programs/Python/Python3
File Edit Format Run Options Window Help
f1 = open('f:\\NewFile.txt', 'w')

f1.write('Srinagar\\n')
f1.write('Pahalgam\\n')
f1.write('Doodhpatri\\n')
f1.write('Gulmarg\\n')
f1.close()

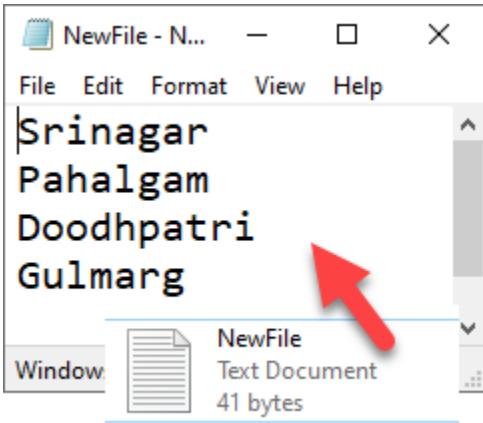
code

f1 = open('f:\\NewFile.txt', 'r')
data = f1.read(17);
print('Data upto 17 ==>', data)

currentOffSet = f1.tell();
print('Current offset Position : ', currentOffSet)

print("Now Let's go again Starting")
currentOffset = f1.seek(0, 0);
Data = f1.read(17);
print('Again from the Starting upto 17 : ', Data)
f1.close()
```

```
IDLE Shell 3.10.2
File Edit Shell Debug Options Window Help
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2023, 16:37:40) [MSC v.1932 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license()" for more information.
==== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Python310/file2.py ===
Data upto 17 ==> Srinagar
Pahalgam
Current offset Position : 18
Now Let's go again Starting
Again from the Starting upto 17 : Srinagar
Pahalgam
output
```

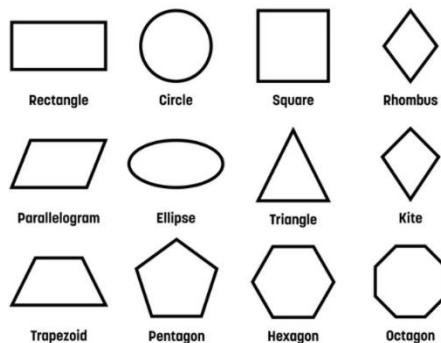


# Introduction to turtle graphics



Turtle graphics were part of the original Logo of the programming language developed by Wally Feurzeig, Seymour Papert, and Cynthia Solomon in 1967.

Using the Python turtle library, we can draw and create various types of shapes, images, mini-games, and animations.



## Steps to work with Turtle library.

Step 1: import turtle

Step 2: Create an object of turtle.

Step 3: Draw or Write around using the turtle methods.

Step 4: turtle.done()



## Example#

```

import turtle
t1 = turtle.Turtle()
t1.screen.bgcolor("aqua")
t1.color('red')
t1.write("hello")
t1.screen.setup(300,300)
t1.screen.title("First")
turtle.done()

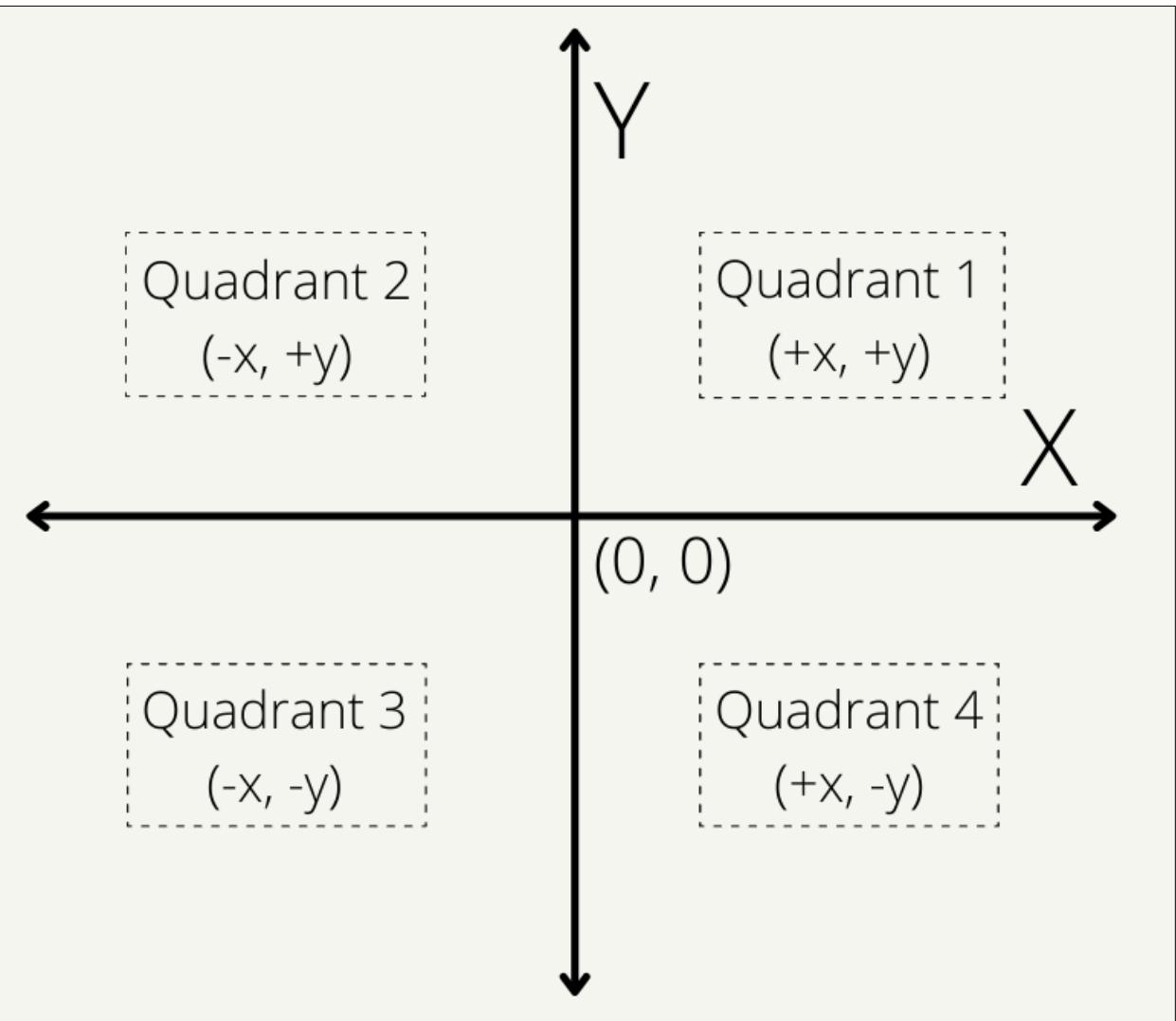
```

Let's understand above code

|                           |                                                                                                                                              |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Import turtle             | It brings the module turtle into our code.                                                                                                   |
| t1=turtle.Turtle()        | It returns a turtle, and that returned turtle is assigned to the variable t1.                                                                |
| t1.screen.bgcolor("aqua") | It takes function 'bgcolor' from module 'screen' from our turtle. It is used to change the background color our screen.                      |
| t1.color('red')           | It changes the color of our turtle.                                                                                                          |
| t1.write("hello")         | It writes the text entered by the user.                                                                                                      |
| t1.screen.setup(300,300)  | It takes the function 'setup' from module 'screen' from our turtle and it is used to set the screen size which currently is 300x300.         |
| t1.screen.title("First")  | It take the function 'title' from module 'screen' from our turtle and it is used to change the title of our screen as per users requirement. |
| turtle.done()             | This will start the execution of our program.                                                                                                |

### Turtle's Screen

The screen is made up of x and y coordinates.



## 5.2.1 Turtle methods

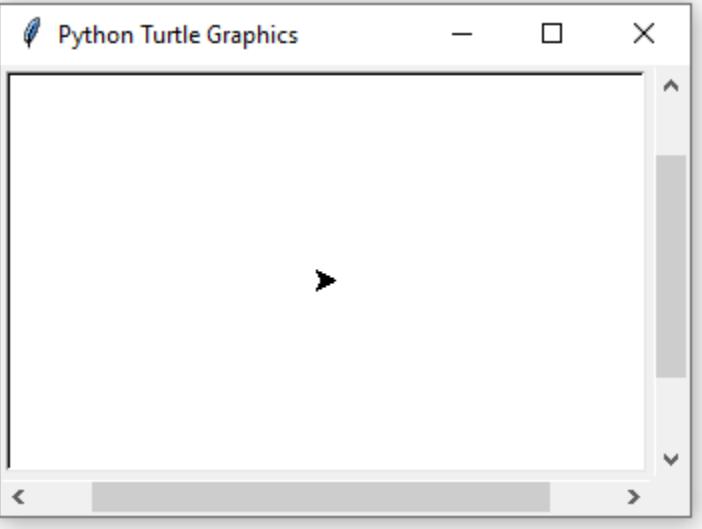
### Turtle()

It creates and returns new turtle objects.

#### Syntax#

```
TurtleVariableName= turtle.Turtle()
```

#### Example#



```
import turtle

t1 = turtle.Turtle()

turtle.done()
```

### done()

It is used to complete the entire code. It should be the last statement in a program.

#### Syntax#

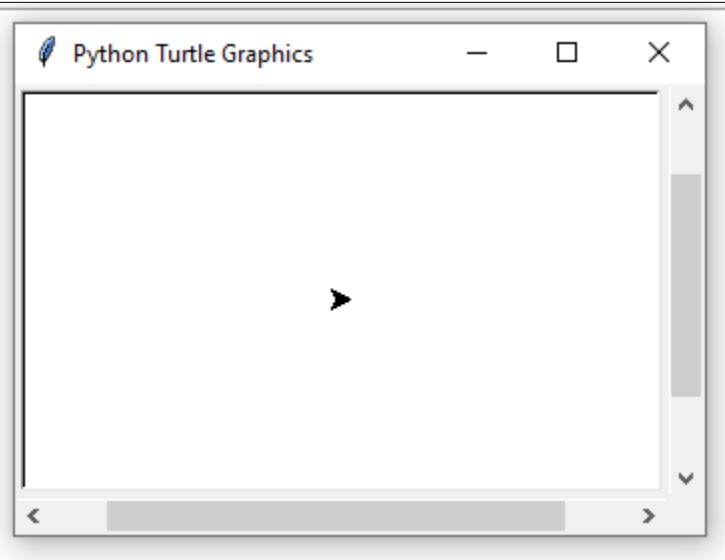
```
turtle.done()
```

#### Example#

```
import turtle

t1 = turtle.Turtle()

turtle.done()
```



## shape()

The default shape of Turtle is an Arrow. We can change the shape of the Turtle using the `shape()` method. The `shape()` method has one argument that is the shapes that are available in the library.

### Available shapes are:

|        |                                                  |
|--------|--------------------------------------------------|
| Arrow  | A black arrow pointing upwards and to the right. |
| Turtle | A black turtle head with a tail.                 |
| Circle | A black circle with a tail.                      |
| Square | A black square with a tail.                      |

|          |                                                                                   |
|----------|-----------------------------------------------------------------------------------|
| Triangle |   |
| Classic  |  |

### Syntax#

TurtleVariableName.shape(ShapeName)

### Example# Square shape

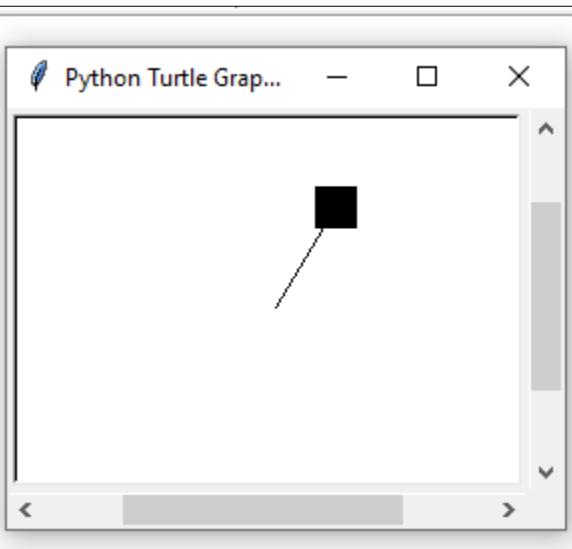
```
import turtle

t1 = turtle.Turtle()

t1.shape("square")

t1.goto(30,50)

turtle.done()
```



### Example# Turtle Shape

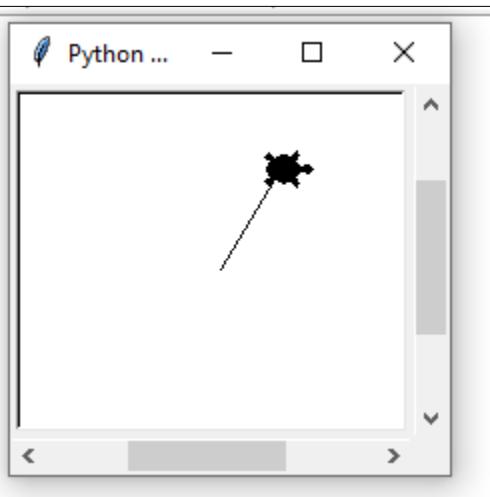
```
import turtle
```

```
t1 = turtle.Turtle()
```

```
t1.shape("turtle")
```

```
t1.goto(30,50)
```

```
turtle.done()
```



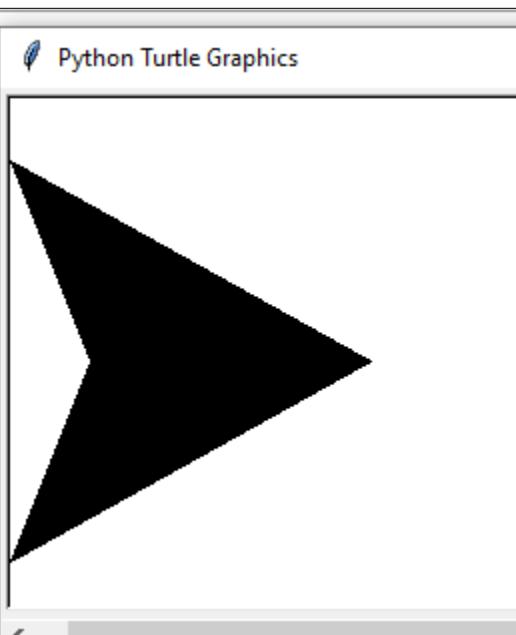
## shapesize()

It is used to change the size of the turtle. The shapesize() method has two arguments which are (x, y) coordinates.

### Syntax#

```
TurtleVariableName.shapesize(x,y)
```

### Example# Turtle ShapeSize



```
import turtle

t1 = turtle.Turtle()

t1.shapesize(20,20)

turtle.done()
```

A screenshot of the Python Turtle Graphics window. The title bar says "Python Turtle Graphics". Inside the window, there is a large, solid black right-angled triangle pointing towards the bottom-left. The triangle's hypotenuse is oriented diagonally upwards and to the left. The rest of the window is white.

## screensize()

To change screen size we can use screensize() method. The screensize() method has two arguments - (canvas height, canvas width) coordinates.

### Syntax#

```
turtle.screensize(x,y)
```

### Example# Turtle Screen Size

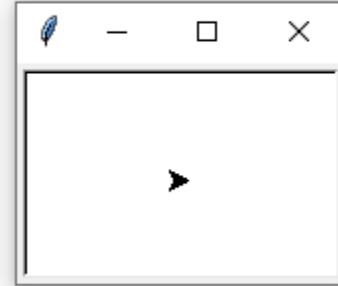
```
import turtle

t1 = turtle.Turtle()

print(turtle.screensize())

turtle.screensize(canvwidth=50, canvheight=50)

turtle.done()
```



## title()

The title() method is used to set the Title on the title bar of the window.

### Syntax#

TurtleVariableName.title(Title Name)

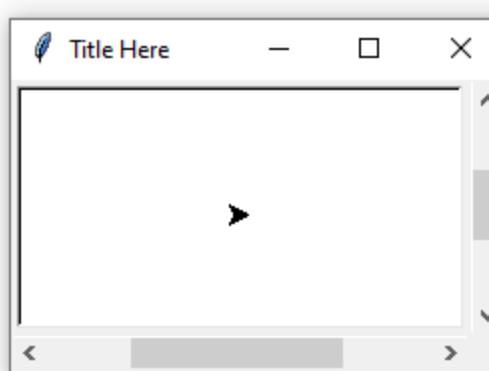
#### Example# Turtle window title

```
import turtle

t1=turtle.Turtle()

turtle.title("Title Here")

turtle.done()
```



## bgcolor()

We can modify the background color using the bgcolor() method. The bgcolor() method has one argument that is the color name.

### Syntax#

TurtleVariableName.bgcolor(colorName)

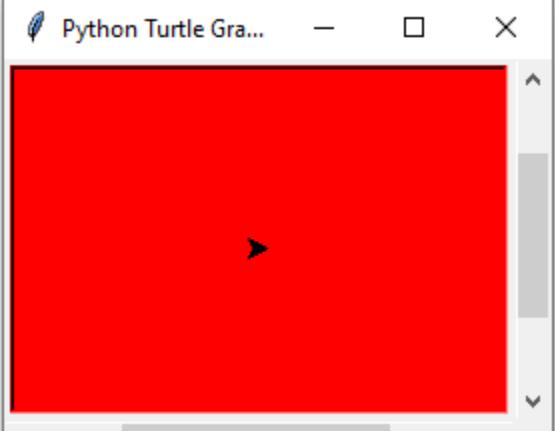
## Example# Turtle background-color

```
import turtle

t = turtle.Turtle()

turtle.bgcolor("red")

turtle.done()
```

A screenshot of a Python Turtle Graphics window titled "Python Turtle Gra...". The window shows a solid red background with a small black arrow cursor in the center. The window has standard operating system window controls (minimize, maximize, close) at the top right.

## color()

We can modify the background color of the turtle initially. The `color()` method has one argument that is the color name.

### Syntax#

`TurtleVariableName.color(colourName)`

## Example# Turtle's colour

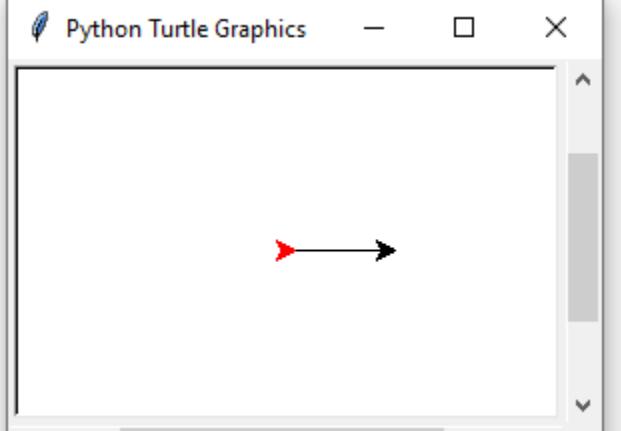
```
import turtle

t1 = turtle.Turtle()

turtle.color("red")

t1.forward(50)

turtle.done()
```

A screenshot of a Python Turtle Graphics window titled "Python Turtle Graphics". The window shows a white background with a red arrow cursor pointing to the right. A small red segment of the arrow indicates the current pen color. The window has standard operating system window controls at the top right.

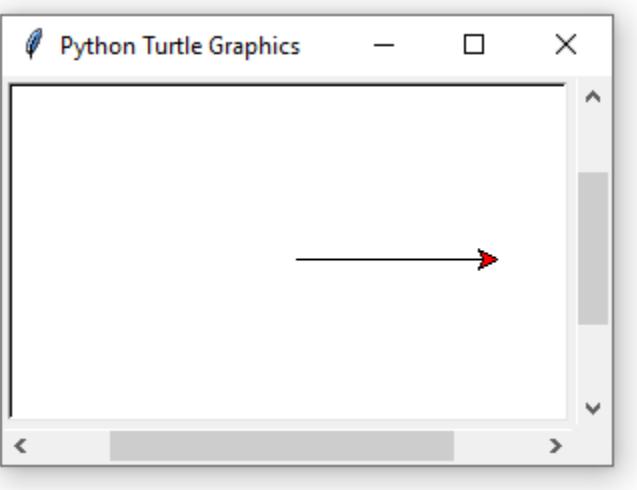
## fillcolor()

The fillcolor() method is used to change the color of the turtle. The fillcolor() method has one argument that is the color name.

### Syntax#

TurtleVariableName.fillcolor(colourName)

#### Example# Turtle's fillcolor



A screenshot of a Python Turtle Graphics window titled "Python Turtle Graphics". The window shows a single red arrow pointing to the right, indicating the direction of the turtle's movement. The code on the left side of the window is:

```
import turtle
t1 = turtle.Turtle()

t1.fillcolor("red")

t1.forward(100)

turtle.done()
```

#### pencolor()

It is used to change the color of the pen's outline. The pencolor() method has one argument that is the color name.

### Syntax#

TurtleVariableName.pencolor(colourName)

#### Example# Turtle's pencolor

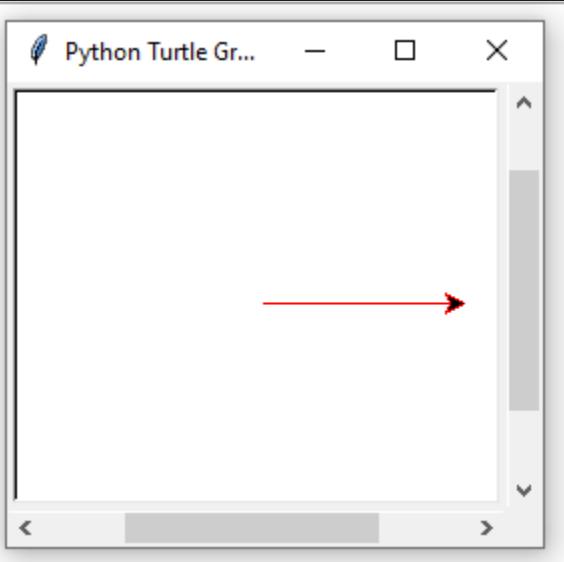
```
import turtle

t1 = turtle.Turtle()

t1.pencolor("red")

t1.forward(100)

turtle.done()
```



## pensize()

We can set the width of the pen using pensize() method. The pensize() method has one argument that is a positive number.

### Syntax#

TurtleVariableName.pensize(integerValue)

## Example# Turtle's pensize

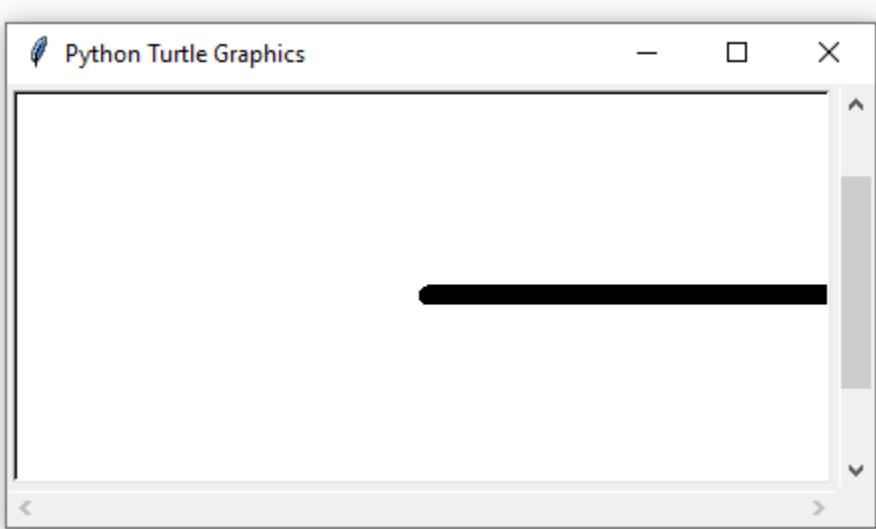
```
import turtle

t1 = turtle.Turtle()

t1.pensize(10)

t1.forward(300)

turtle.done()
```



## speed()

We can control the speed of pen using the speed() method. The speed() method has one argument that is a positive number.

### Syntax#

TurtleVariableName.speed(integerValue)

#### Example# Turtle's pen speed

```
import turtle

t1 = turtle.Turtle()

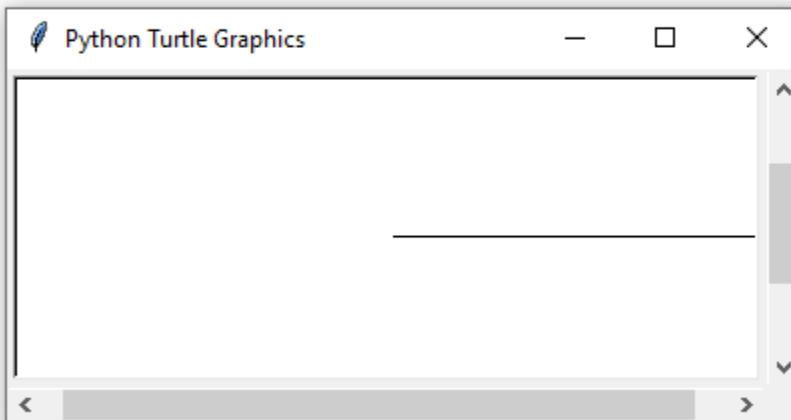
t1.speed(2)

t1.forward(100)

t1.speed(5)

t1.forward(100)

turtle.done()
```



#### bgpic()

The bgpic() method is used to set a background image. The bgpic() method has one argument imagename.

### Syntax#

TurtleVariableName.bgpic(imagename)

#### Example# Turtle's background

```
import turtle

t1 = turtle.Screen()

t1.bgpic('f:\shyam.png')

turtle.done()
```



### 5.2.2 TurtleScreen/Screen Methods

home()

It is used to move the turtle pointer on the home position.

**Syntax#**

TurtleVariableName.home()

Example# Turtle's home()

```
import turtle

t1=turtle.Turtle()

print(t1.position())

t1.forward(100)

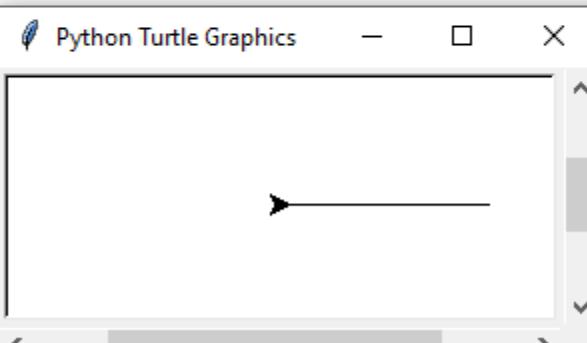
print(t1.position())

t1.home() ←

print(t1.position())

turtle.done()
```

>>>  
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan  
Type "help", "copyright", "credits" or "licen  
===== RESTART: C:/Users/Shyam/AppData/L  
(0.00,0.00) ←  
(100.00,0.00)  
(0.00,0.00) ←



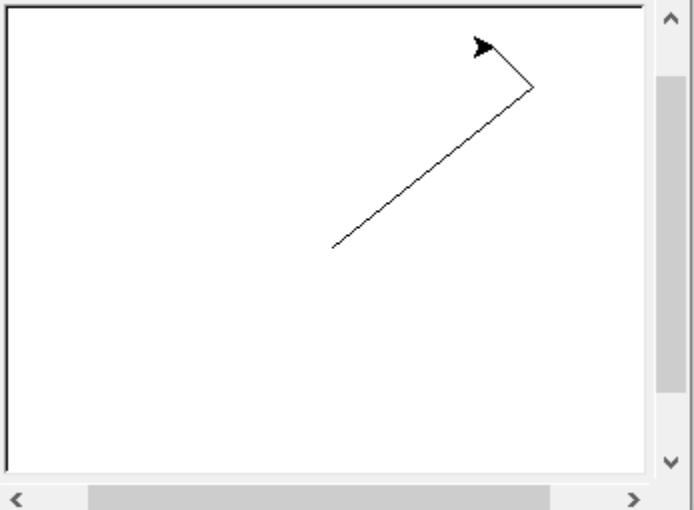
## goto()

It is used to move the turtle pointer to a specific position. The goto() method has two arguments (x, y) coordinates.

### Syntax#

TurtleVariableName.goto(x,y)

### Example#



```
import turtle

t1 = turtle.Turtle()

t1.goto(100, 80)

t1.goto(80,100)

turtle.done()
```

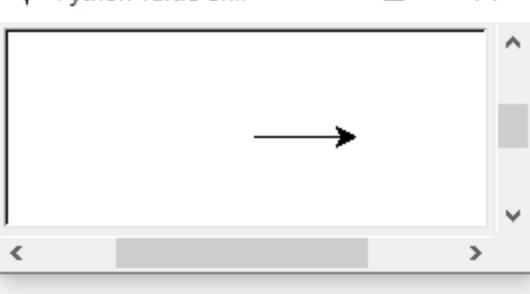
## Forward()

It is used to move the turtle pointer forward. Shortcut of this function fd(). There is only one argument of forward() function that is distance.

### Syntax#

```
TurtleVariableName.forward(distance)
```

#### Example# Forward() method



```
import turtle

t1 = turtle.Turtle()

t1.forward(50)

turtle.done()
```

## backward()

It is used to move the turtle pointer backward. Shortcut of this function bk(). There is only one argument of backward() function that is distance.

### Syntax#

```
TurtleVariableName.backward(distance)
```

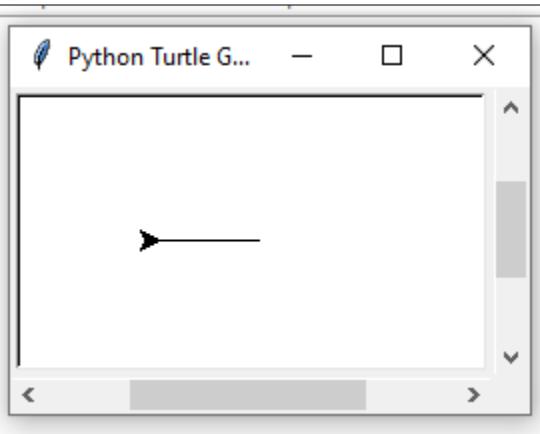
#### Example# Backward() method

```
import turtle
```

```
t1 = turtle.Turtle()
```

```
t1.backward(50)
```

```
turtle.done()
```



#### left()

It is used to rotate the turtle pointer left. Shortcut of this function lt(). There is only one argument of left() function that is the angle.

#### Syntax#

```
TurtleVariableName.left(angle)
```

#### Example# left() method

```
import turtle
```

```
t1 = turtle.Turtle()
```

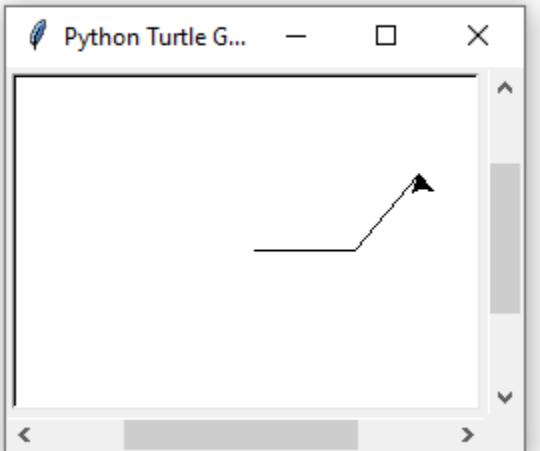
```
t1.forward(50)
```

```
t1.left(50)
```

```
t1.forward(50)
```

```
t1.left(50)
```

```
turtle.done()
```



## right()

It is used to rotate the turtle pointer right. Shortcut of this function `rt()`. There is only one argument of `right()` function that is the angle.

### Syntax#

`TurtleVariableName.right(angle)`

### Example# `right()` method

```
import turtle
```

```
t1 = turtle.Turtle()
```

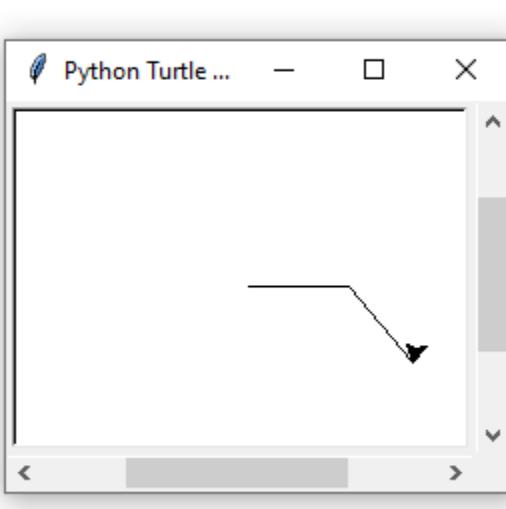
```
t1.forward(50)
```

```
t1.right(50)
```

```
t1.forward(50)
```

```
t1.right(50)
```

```
turtle.done()
```



### Example# Let's draw a rectangle using turtle

```
import turtle
```

```
t1=turtle.Turtle()
```

```
t1.forward(200)
```

```
t1.left(90)
```

```
t1.forward(100)
```

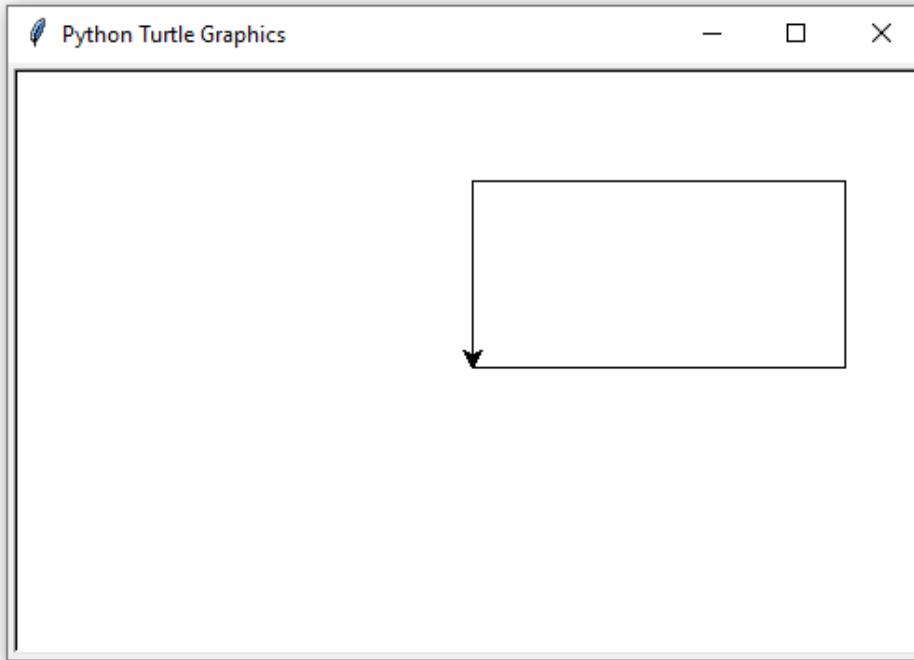
```
t1.left(90)
```

```
t1.forward(200)
```

```
t1.left(90)
```

```
t1.forward(100)
```

```
turtle.done()
```



### Example# Let's draw a square using turtle

```
import turtle
```

```
shyam=turtle.Turtle()
```

```
shyam.forward(100)
```

```
shyam.left(90)
```

```
shyam.forward(100)
```

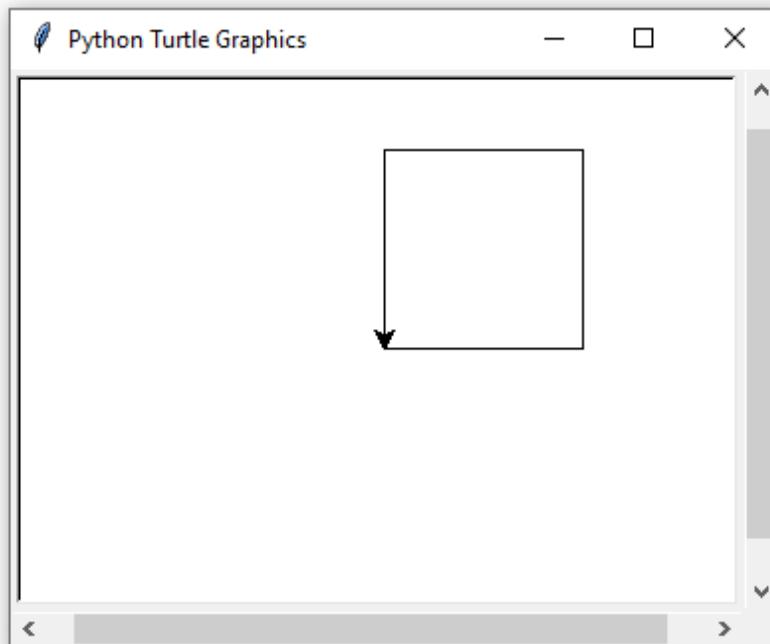
```
shyam.left(90)
```

```
shyam.forward(100)
```

```
shyam.left(90)
```

```
shyam.forward(100)
```

```
turtle.done()
```



**Example# Let's draw a Triangle using turtle and also change its color**

```
import turtle
```

```
t1=turtle.Turtle()
```

```
t1.color("blue")
```

```
t1.forward(100)
```

```
t1.left(135)
```

```
t1.forward(100)
```

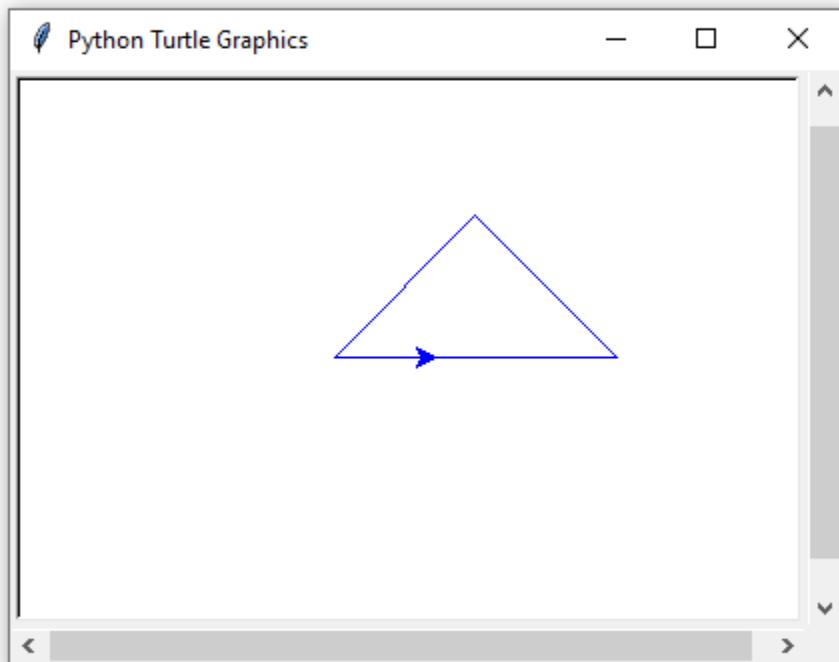
```
t1.left(90)
```

```
t1.forward(100)
```

```
t1.left(135)
```

```
t1.forward(50)
```

```
turtle.done()
```



## Picking the Pen Up and Down

We may want to move our turtle to another point on the screen without drawing anything on the screen itself.



## penup() or pu()

The penup() or the pu() method picks up the turtle's Pen and we can move the turtle without drawing.

### Syntax#

TurtleVariableName.penup()

## pendown() or pd()

The pendown() or pd() method puts down the turtle's Pen. We can bring the turtle down and draw.

### Syntax#

TurtleVariableName.pendown()

```
import turtle
```

```
t1=turtle.Turtle()
```

```
t1.fd(100)
```

```
t1.rt(90)
```

```
t1.penup()
```

```
t1.fd(100)
```

```
t1.rt(90)
```

```
t1.pendown()
```

```
t1.fd(100)
```

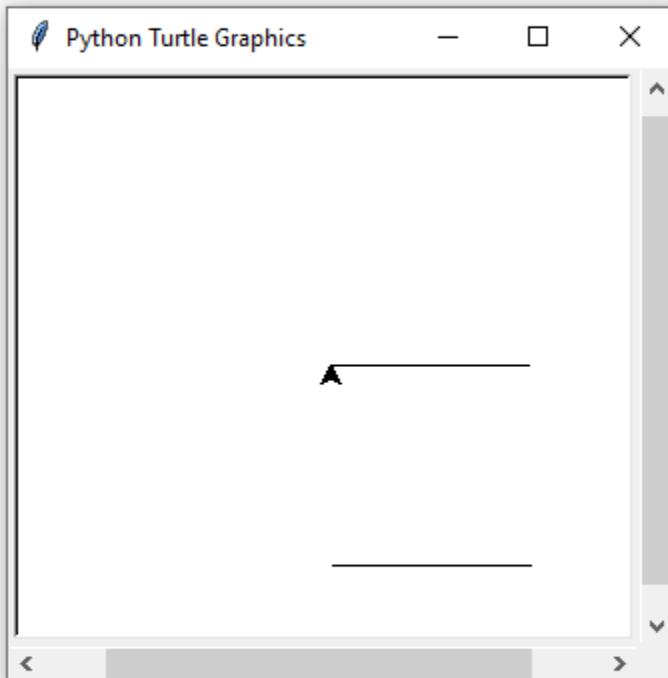
```
t1.rt(90)
```

```
t1.penup()
```

```
t1.fd(100)
```

```
t1.pendown()
```

```
turtle.done()
```



## Circle

It is used to draw the circle to the screen. There are three arguments.

### Syntax#

TurtleVariableName.circle(radius,extent,steps)

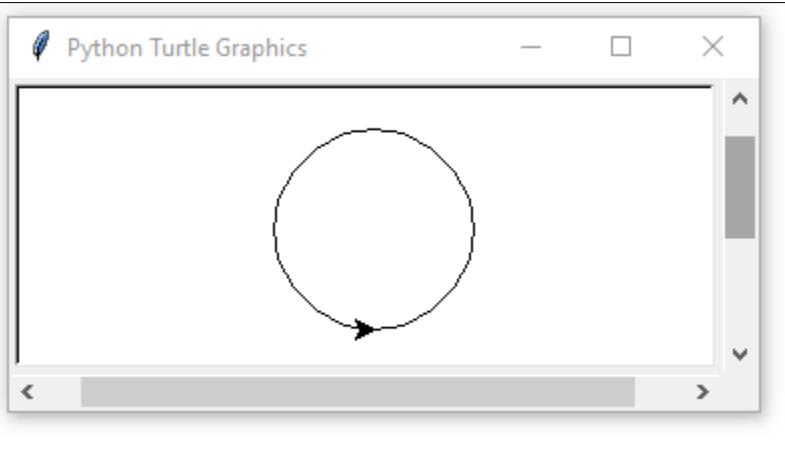
### Example# Let's draw a circle

```
import turtle

t1 = turtle.Turtle()

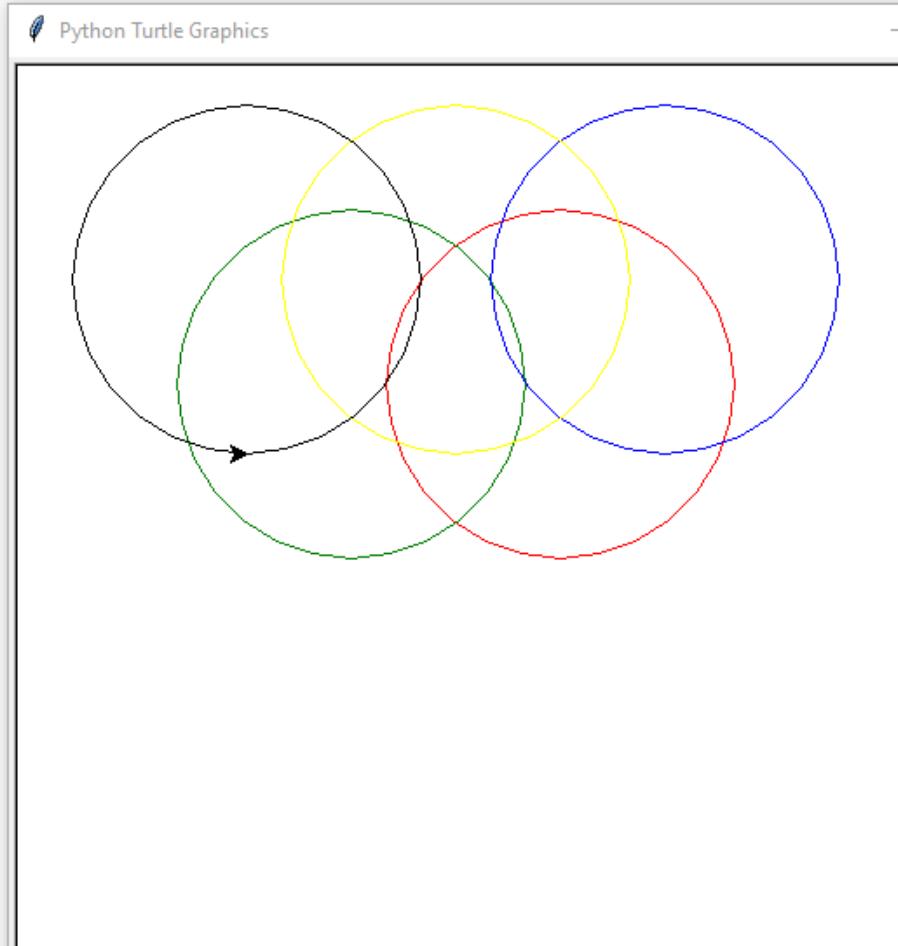
t1.circle(50)

turtle.done()
```



### Example# Let's draw Olympic symbol

```
import turtle
t1 = turtle.Turtle()
t1.color("red")
t1.circle(100)
t1.penup()
t1.color("green")
t1.setposition(-120, 0)
t1.pendown()
t1.circle(100)
t1.penup()
t1.color("blue")
t1.setposition(60,60)
t1.pendown()
t1.circle(100)
t1.penup()
t1.color("yellow")
t1.setposition(-60, 60)
t1.pendown()
t1.circle(100)
t1.penup()
t1.color("black")
t1.setposition(-180, 60)
t1.pendown()
t1.circle(100)
turtle.done()
```



**Example# Let's draw a circle with the border color set to red.**

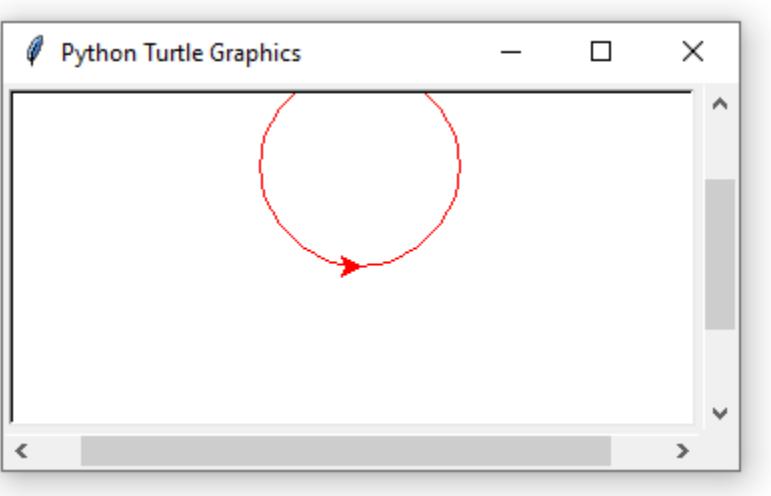
```
import turtle

t1=turtle.Turtle()

t1.color("red")

t1.circle(50)

turtle.done()
```



## **begin\_fill() and end\_fill()**

It is used to fill shape using the current fill color.

For example, if the current pen color is Red, then any shape we draw in between begin\_fill() and end\_fill() will be filled with the color Red. If the color of the pen is currently Blue, then any shape drawn will be filled with Blue.

### **Syntax#**

TurtleVariableName.begin\_fill()

Other Logic...

TurtleVariableName.end\_fill()

## **Example# Let's draw a circle with full red color**

```
import turtle

t1=turtle.Turtle()

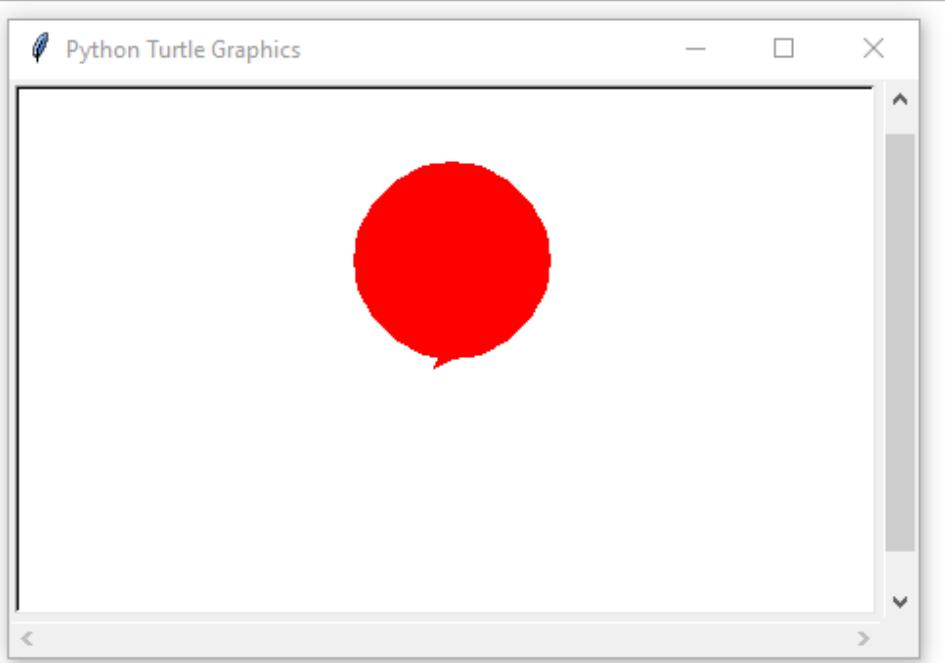
t1.begin_fill()

t1.color("red")

t1.circle(50)

t1.end_fill()

turtle.done()
```



## dot()

The dot() is used to draw a filled circle.

### Syntax#

TurtleVariableName.dot(radius)

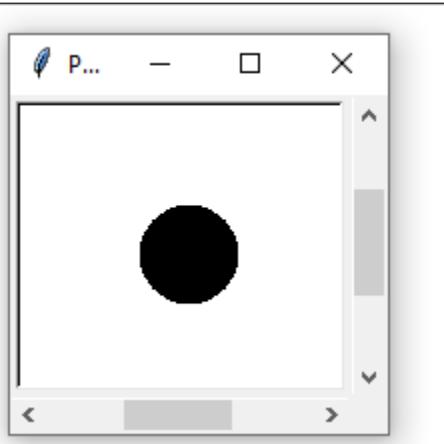
### Example# Let's draw a circle using dot

```
import turtle

t1 = turtle.Turtle()

t1.dot(50)

turtle.done()
```



### Example# Let's draw a blue-colored circle with dot.

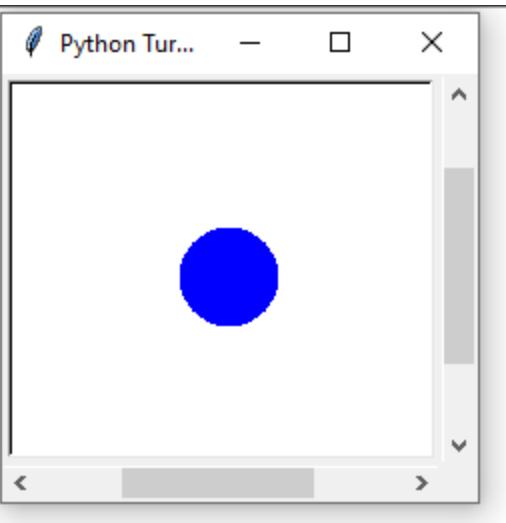
```
import turtle

t1 = turtle.Turtle()

t1.color("Blue")

t1.dot(50)

turtle.done()
```



## clear()

The clear() method is used to clean up our screen so that we can continue drawing.

### Syntax#

TurtleVariableName.clear()

#### Example# Let's draw two circles but in between let's use clear()

```
import turtle

t1=turtle.Turtle()

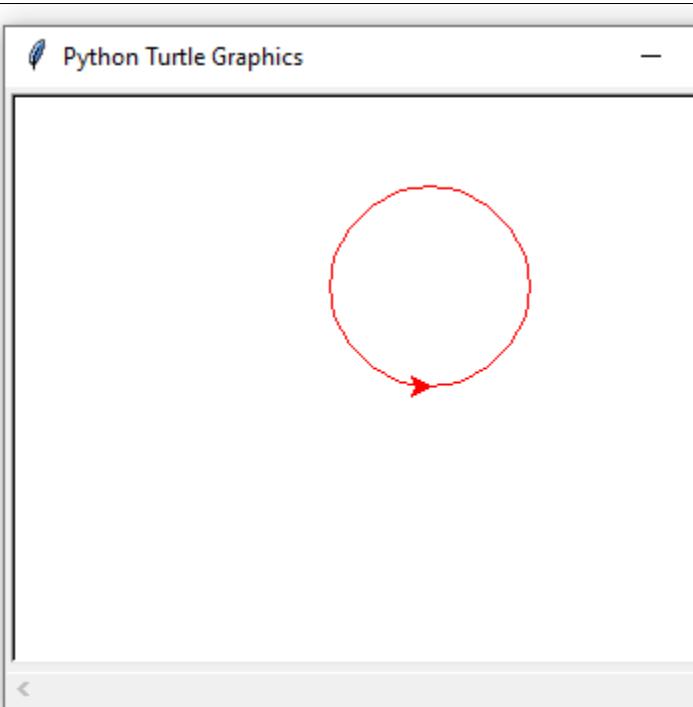
t1.color("red")

t1.circle(50)

t1.clear()

t1.circle(50)

turtle.done()
```



#### 5.3.1 Turtle programming - loops and conditional statements.

We can use loops and conditional statements with the turtle module.

#### Example# Ask the user to enter the steps to move forward or backward

```
import turtle

t1 = turtle.Turtle()

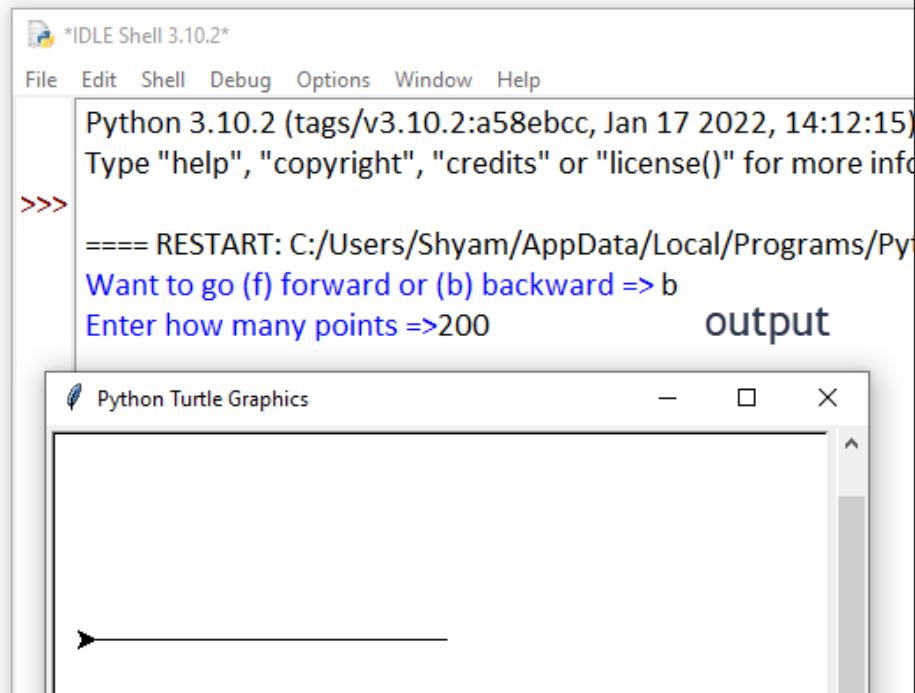
direction=input("Want to go (f) forward or (b) backward => ").lower()
point=int(input("Enter how many points =>"))
```

```
if direction=="f":
 t1.forward(point)

elif direction=="b":
 t1.backward(point)

turtle.done()
```

code



**Example# Let's give a choice to user to color the line**

File Edit Format Run Options Window Help

```
import turtle
```

```
t1 = turtle.Turtle()
```

```
colorname=input("Red or Blue color dot? ").lower()
```

```
if colorname=="red":
```

```
 t1.color("red")
```

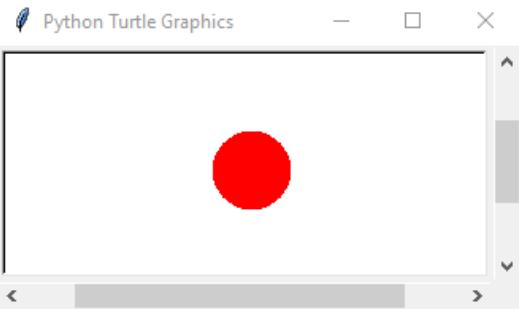
```
elif colorname=="blue":
```

```
 t1.color("Blue")
```

```
t1.dot(50)
```

```
turtle.done()
```

code



\*IDLE Shell 3.10.2\*

File Edit Shell Debug Options Window Help

Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.

Type "help", "copyright", "credits" or "license()" for more information

>>>

===== RESTART: C:/Users/Shyam/AppData/Local/Programs/Python/Pyt  
Red or Blue color dot? red

output

### Example# Draw a Square using loop

```
import turtle
```

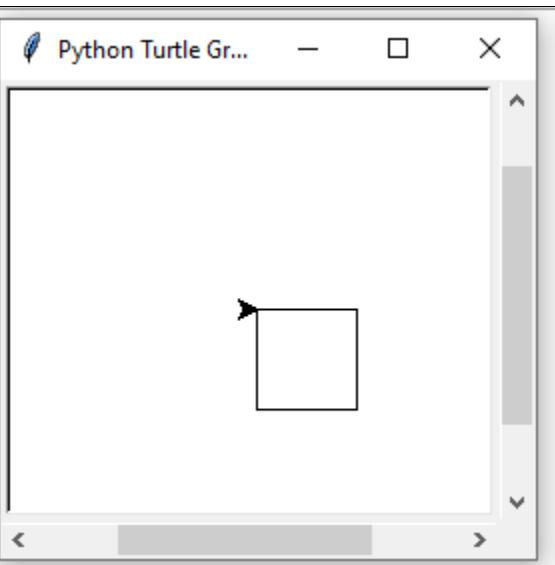
```
t1 = turtle.Turtle()
```

```
for i in range(4):
```

```
 t1.forward(50)
```

```
 t1.right(90)
```

```
turtle.done()
```



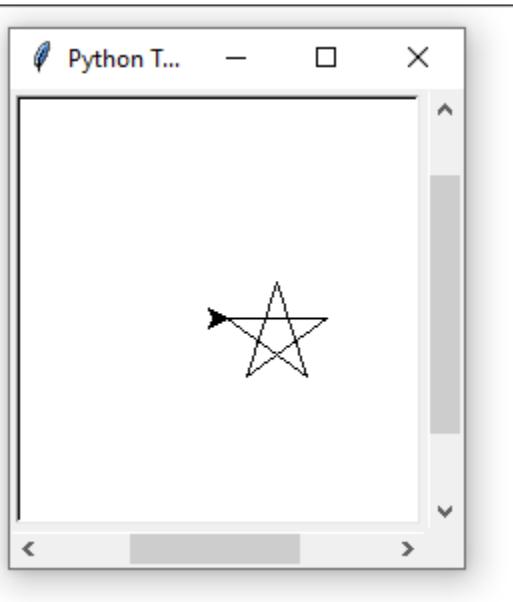
### Example# Draw a Star using loop

```
import turtle

t1 = turtle.Turtle()

for i in range(5):
 t1.forward(50)
 t1.right(144)

turtle.done()
```



### Example# Draw a Triangle using loop and also fill the triangle.

```
import turtle

t1=turtle.Turtle()

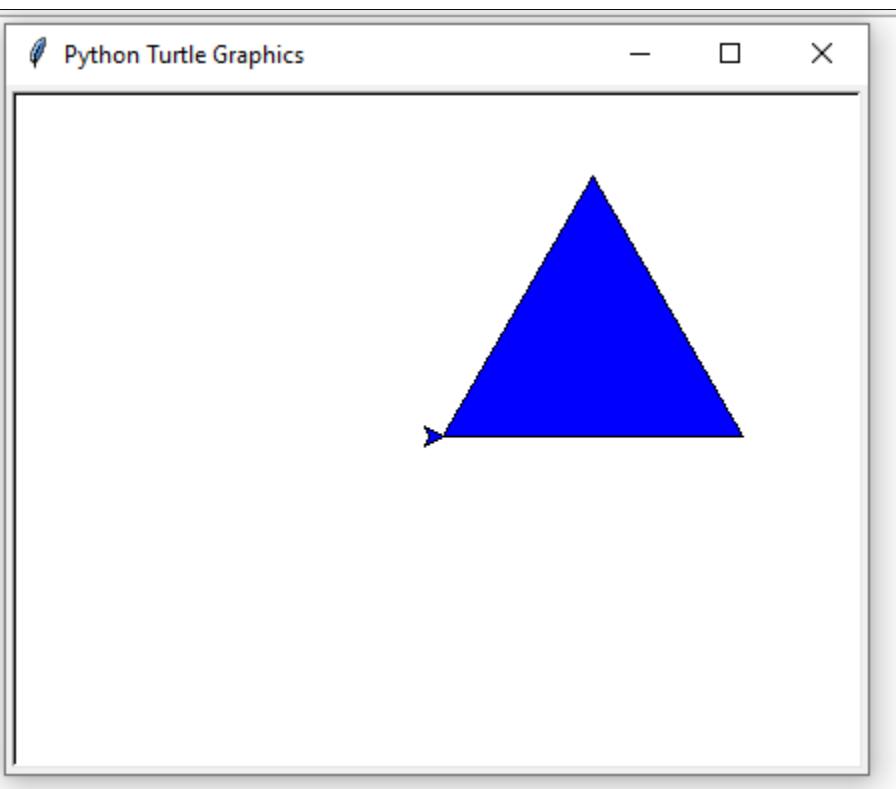
t1.fillcolor("blue")

t1.begin_fill()

for i in range(3):
 t1.forward(150)
 t1.left(120)

t1.end_fill()

turtle.end()
```



**Example# Draw an Octagon using loop and also fill it.**

```
import turtle

t1 = turtle.Turtle()

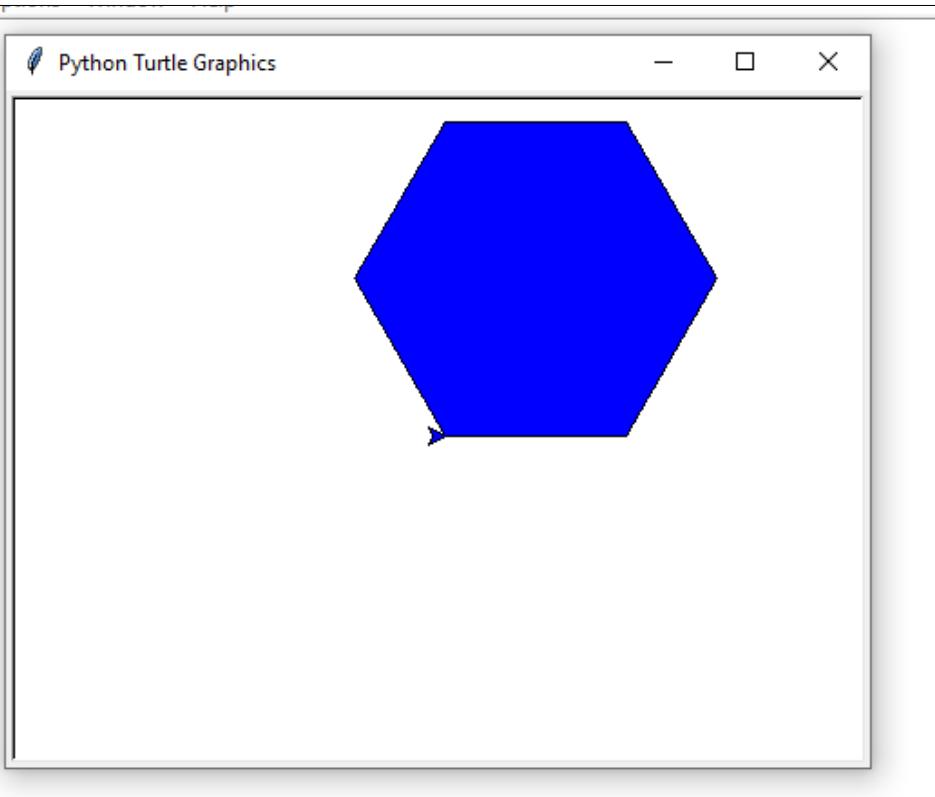
t1.fillcolor("Blue")

t1.begin_fill()

for _ in range(6):
 t1.forward(100)
 t1.right(-60)

t1.end_fill()

turtle.end()
```



**Example# Draw a SpinoGraph using loop and also fill it.**

```
import turtle

t1=turtle.Turtle()
turtle.bgcolor("black")
t1.pensize(2)
t1.speed(0)

for i in range (6):
 for colors in ["red", "magenta", "blue", "cyan", "green", "yellow", "white"]:
 t1.color(colors)
 t1.circle(100)
 t1.left(10)

turtle.done()
```



Python Turtle Graphics

- □ X

