

**Python?**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

In contrast to other popularlanguages such as C, C++, Java, and C#, Python strives to provide a simple but powerful syntax. Python is used for software development at companies and organizations such as Google, Yahoo,CERN, Industrial Light and Magic, and NASA.

**What is use of Python?**

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.
* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
* Python has a simple syntax similar to the English language.
* The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
* The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

## **Getting Python**

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python <https://www.python.org/downloads/>

## **First Python Program**

>>> print (“hello! This is my first python program”)

# **Python - Variable Types**

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

## **Assigning Values to Variables**

>>>a=1

>>>a=b=c=1

**Python Variables Example**

x = "awesome"  
print("Python is " + x)

x = "awesome"  
y = "Python is " + x  
print(y)

x = 5  
y = 10  
print(x + y)

x = 1    # int  
y = 2.8 # float  
z = 1j   # complex

Example:

x = 1

y = 2.8

z = 1j

print(type(x))

print(type(y))

print(type(z))

**Result:**

<class 'int'>  
<class 'float'>  
<class 'complex'>

## **Deleting a Variables**

>>>Del variable name

# **Python Casting**

x = float(1)   # x will be 1.0  
y = int(2.8) # y will be 2  
z = int("3") # z will be 3

x = str("s1") # x will be 's1'  
y = str(2)    # y will be '2'  
z = str(3.0)  # z will be '3.0'

## **Types of Operator**

Python language supports the following types of operators.

* Arithmetic Operators
* Comparison (Relational) Operators
* Assignment Operators
* Logical Operators
* Membership Operators
* Identity Operators

|  |  |  |
| --- | --- | --- |
| + Addition | Adds values on either side of the operator. | a + b |
| - Subtraction | Subtracts right hand operand from left hand operand. | a – b |
| \* Multiplication | Multiplies values on either side of the operator | a \* b |
| / Division | Divides left hand operand by right hand operand | b / a |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | b % a |
| \*\* Exponent | Performs exponential (power) calculation on operators | a\*\*b to the power |
| // | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) − |  |

Example:

>>>a=6

>>>b=2

>>>c=a + b or >>>c=a % b or >>>c=a\*\*b or >>>c=a//b

>>>print(c) >>>print(c) >>>print(c) >>>print(c)

Ans: 8 Ans: 0 Ans:36 And:3 (quotient)

## **Python Comparison Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | If the values of two operands are equal, then the condition becomes true. | (a == b) is not true. |
| != | If values of two operands are not equal, then condition becomes true. | (a != b) is true. |
| <> | If values of two operands are not equal, then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a > b) is not true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. | (a < b) is true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a >= b) is not true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | (a <= b) is true. |

## **Python Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Assigns values from right side operands to left side operand | c = a + b assigns value of a + b into c |
| += Add AND | It adds right operand to the left operand and assign the result to left operand | c += a is equivalent to c = c + a |
| -= Subtract AND | It subtracts right operand from the left operand and assign the result to left operand | c -= a is equivalent to c = c - a |
| \*= Multiply AND | It multiplies right operand with the left operand and assign the result to left operand | c \*= a is equivalent to c = c \* a |
| /= Divide AND | It divides left operand with the right operand and assign the result to left operand | c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a |
| %= Modulus AND | It takes modulus using two operands and assign the result to left operand | c %= a is equivalent to c = c % a |
| \*\*= Exponent AND | Performs exponential (power) calculation on operators and assign value to the left operand | c \*\*= a is equivalent to c = c \*\* a |
| //= Floor Division | It performs floor division on operators and assign value to the left operand | c //= a is equivalent to c = c // a |

## **Python Logical Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

## **Python Membership Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| In | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a 1 if x is a member of sequence y. |
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |

Example:

a =10

b =20

list=[1,2,3,4,5];

if( a in list ):

print"Line 1 - a is available in the given list"

else:

print"Line 1 - a is not available in the given list"

if( b notin list ):

print"Line 2 - b is not available in the given list"

else:

print"Line 2 - b is available in the given list"

a =2

if( a in list ):

print"Line 3 - a is available in the given list"

else:

print"Line 3 - a is not available in the given list"

Line 1 - a is not available in the given list

Line 2 - b is not available in the given list

Line 3 - a is available in the given list

**How to Input Value from Keyboard**

>>>variable = input (“Your Message”)

>>>a = input (“Enter a value”)

Note: It will store value in string format so if you want to convert string into integer then have to use this

>>>a=int (input (“Enter a value”)

**Save & Run Python file:**

Step: 1 Open Python IDE

Step: 2 File ---- New File

Step: 3 then type your program

Step: 4 then save your file (note: python extinction is .py)

Step: 5 How to run (Run --- Run Module (F5))

x=int(input("Enter 1st no."))

y=int(input("Enter 2sd no."))

z=int(input("Enter 3rd no."))

print (max(x,y,z))

input("press enter to exit")

phy=26

che=50

math=66

eng=75

hindi=44

total=phy+che+math+eng+hindi

print("total is ",total)

per=total/5

print("% is ",per)

input("Press any to exit !")

# **Python Strings**

a = "hello"  
print(a[1])

The len() method returns the length of a string:

a = "Hello, World!"  
print(len(a))

The lower() method returns the string in lower case:

a = "Hello, World!"  
print(a.lower())

a = "Hello, World!"  
print(a.upper())

a = "Hello, World!"  
print(a.replace("H", "J"))

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

Some other string example:

print("Enter your name:")  
x = input()  
print("Hello, " + x)

# **Python Lists**

## **Python Collections (Arrays)**

There are four collection data types in the Python programming language:

* **List** is an collection which is ordered and changeable. Allows duplicate members.
* **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
* **Set** is a collection which is unordered and unindexed. No duplicate members.
* **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

## **List**

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Ex: 1

thislist = ["apple", "banana", "cherry"]  
print(thislist)

Ex: 2

thislist = ["Ram", "Mohan", "Sonu"]  
thislist[1] = "Sumit"  
print(thislist)

Ex: 3

thislist = list(("kota", "jaipur", "bundi"))

thislist.append("ajmer")

print(thislist)

Ex: 3

thislist = list(("kota", "ajmer", "bundi"))

thislist.remove("ajmer")

print(thislist)

Ex: 4

thislist = list(("apple", "banana", "cherry"))  
print(len(thislist))

# **Python Tuples**

thistuple = ("computer", "keyboard", "mouse")  
thistuple[1] = "monitor" # test changeability  
print(thistuple)

Note: value will not change

# **Python Sets**

thisset = set(("ram", "mohan", "sonu"))  
thisset.add("sumit")  
print(thisset)

# **Python Dictionaries**

thisdict ={  
  "apple": "green",  
  "banana": "yellow",  
  "cherry": "red"  
}  
thisdict["apple"] = "red"  
print(thisdict)

# **Python Conditions**

## **Python Conditions and If statements**

a = 33  
b = 200  
if b > a:

print("b is greater than a")

a = 33  
b = 200  
if b > a:  
  print("b is greater than a")

a = 33  
b = 200  
if b > a:  
print("b is greater than a") # you will get an error

## **Elif**

The elif keyword is pythons way of saying "if the previous conditions were not true, then do this condition".

a = 33  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")

## **Else**

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")  
else:  
  print("a is greater than b")

## **Python Loops**

Python has two primitive loop commands:

* while loops
* for loops

## **The while Loop**

i = 1  
while i< 6:  
  print(i)  
  i += 1

## **The break Statement**

i = 1  
while i< 6:  
  print(i)  
  if i == 3:  
    break  
  i += 1

## **The continue Statement**

i = 0  
while i< 6:  
  i += 1   
  if i == 3:  
    continue  
  print(i)

# **Python for Loops**

books = ["HTML", "CSS", "PHP"]  
for x in books:  
  print(x)

## **The range() Function**

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

for x in range(6):  
  print(x)

for x in range(2, 6):  
  print(x)

for x in range(2, 20, 2):

print(x)

# **Python Functions**

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

## **Creating a Function**

In Python a function is defined using the **def** keyword:

def my\_function():  
  print("Hello from a function")

## **Calling a Function**

def my\_function():  
  print("Hello from a function")  
  
**my\_function()**

## **Parameters**

defmy\_function(fname):

print("Your :" + fname)

my\_function("Emil")

my\_function("Mobile")

my\_function("City")

## **Return Values**

def my\_function(x):  
  return 5 \* x  
  
print(my\_function(3))  
print(my\_function(5))  
print(my\_function(9))

## **Python Classes/Objects**

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

## **Creating a class**

class Demo:

def sum(self):

a=2

b=3

print(a+b)

## **Create Object**

p1 = Demo()  
p1.sum()

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
p1 = Person("Anil", 36)  
  
print(p1.name)  
print(p1.age)

**Note:** The \_\_init\_\_() function is called automatically every time the class is being used to create a new object.

class Person:

def \_\_init\_\_(a, name, age):

a.name = name

a.age = age

defmyfunc(a):

print("Hello my name is " + a.name)

print("Hello my age is " , a.age)

p1 = Person("anil", 36)

p1.myfunc()

**Note:** The self parameter is a reference to the class itself, and is used to access variables that belongs to the class.

# **Python Program to Create a Class which Performs Basic Calculator Operations**

classcal():

def \_\_init\_\_(self,a,b):

self.a=a

self.b=b

def add(self):

returnself.a+self.b

defmul(self):

returnself.a\*self.b

def div(self):

returnself.a/self.b

def sub(self):

returnself.a-self.b

a=int(input("Enter first number: "))

b=int(input("Enter second number: "))

obj=cal(a,b)

choice=1

while choice!=0:

print("0. Exit")

print("1. Add")

print("2. Subtraction")

print("3. Multiplication")

print("4. Division")

choice=int(input("Enter choice: "))

if choice==1:

print("Result: ",obj.add())

elif choice==2:

print("Result: ",obj.sub())

elif choice==3:

print("Result: ",obj.mul())

elif choice==4:

print("Result: ",round(obj.div(),2))

elif choice==0:

print("Exiting!")

else:

print("Invalid choice!!")

print()

## **Class Inheritance**

Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.

The child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.

### Syntax

classSubClassName(ParentClass1[,ParentClass2,...]):

class Parent: # define parent class

parentAttr = 100

def \_\_init\_\_(self):

print ("Calling parent constructor")

defparentMethod(self):

print ('Calling parent method')

defsetAttr(self, attr):

Parent.parentAttr = attr

defgetAttr(self):

print ("Parent attribute :", Parent.parentAttr)

class Child(Parent): # define child class

def \_\_init\_\_(self):

print ("Calling child constructor")

defchildMethod(self):

print ("Calling child method")

c = Child() # instance of child

c.childMethod() # child calls its method

c.parentMethod() # calls parent's method

c.setAttr(200) # again call parent's method

c.getAttr() # again call parent's method

## **Overloading Operators**

class Vector:

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

def \_\_str\_\_(self):

return 'Vector (%d, %d)' % (self.a, self.b)

def \_\_add\_\_(self,other): # add is operator overloading

return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)

v2 = Vector(5,-2)

print (v1 + v2)

When the above code is executed, it produces the following result −

Vector(7,8)

## **What is a Module?**

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

## **Create a Module**

Save this code in a file named mymodule.py

def greeting(name):  
  print("Hello, " + name)

## **Use a Module**

Now we can use the module we just created, by using the import statement:

import mymodule  
  
mymodule.greeting("Jonathan")

**Note:** When using a function from a module, use the syntax: module\_name.function\_name.

## **Using Variables in Module**

Save this code in the file mymodule.py

person1 = {  
  "name": "John",  
  "age": 36,  
  "country": "Norway"  
}

import mymodule  
  
a = mymodule.person1["age"]  
print(a)

## **Re-naming a Module**

You can create an alias when you import a module, by using the as keyword:

Create an alias for mymodule called mx:

import mymodule as mx  
  
a = mx.person1["age"]  
print(a)

## **Import from Module**

You can choose to import only parts from a module, by using the from keyword.

The module named mymodule has one function and one dictionary:

def greeting(name):  
  print("Hello, " + name)  
  
person1 = {  
  "name": "John",  
  "age": 36,  
  "country": "Norway"  
}

Import only the person1 dictionary from the module:

from mymodule import person1  
  
print (person1["age"])

**Note:** When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1.age, **not** ~~mymodule.person1.age~~

# **Python Datetime**

Import the datetime module and display the current date:

import datetime  
  
x = datetime.datetime.now()  
print(x)

## **Date Output**

import datetime  
  
x = datetime.datetime.now()  
  
print(x.year)  
print(x.strftime("%A"))

## **The strftime() Method**

|  |  |  |  |
| --- | --- | --- | --- |
| **Directive** | **Description** | **Example** | **Try it** |
| %a | Weekday, short version | Wed | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_a) |
| %A | Weekday, full version | Wednesday | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_a2) |
| %w | Weekday as a number 0-6, 0 is Sunday | 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_w) |
| %d | Day of month 01-31 | 31 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_d) |
| %b | Month name, short version | Dec | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_b) |
| %B | Month name, full version | December | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_b2) |
| %m | Month as a number 01-12 | 12 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_m) |
| %y | Year, short version, without century | 18 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_y) |
| %Y | Year, full version | 2018 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_y2) |
| %H | Hour 00-23 | 17 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_h2) |
| %I | Hour 00-12 | 05 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_i2) |
| %p | AM/PM | PM | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_p) |
| %M | Minute 00-59 | 41 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_m2) |
| %S | Second 00-59 | 08 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_s2) |
| %f | Microsecond 000000-999999 | 548513 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_f) |
| %j | Day number of year 001-366 | 365 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_j) |
| %U | Week number of year, Sunday as the first day of week, 00-53 | 52 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_u2) |
| %W | Week number of year, Monday as the first day of week, 00-53 | 52 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_w2) |
| %c | Local version of date and time | Mon Dec 31 17:41:00 2018 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_c) |
| %x | Local version of date | 12/31/18 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_x) |
| %X | Local version of time | 17:41:00 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_datetime_strftime_x2) |

## **File Handling**

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exist

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

## **Open a File on the Server**

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

f = open("demofile.txt", "r")  
print(f.read())

Note: Python file and text file must be saved on same location

## **Read Only Parts of the File**

By default the read() method returns the whole text, but you can also specify how many character you want to return:

f = open("demofile.txt", "r")  
print(f.read(**10**))

## **Read Lines**

You can return one line by using the readline() method:

f = open("demofile.txt", "r")  
print(f.readline())

By calling readline() two times, you can read the two first lines:

By looping through the lines of the file, you can read the whole file, line by line:

Loop through the file line by line:

f = open("demofile.txt", "r")  
for x in f:  
  print(x)

# **Python File Write**

## **Write to an Existing File**

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

f = open("python1.txt", "w")

f.write("Hii I am ")

f.close()

f = open("python1.txt", "a")

f.write("Abhishek ")

f.close()

## **Delete a File**

To delete a file, you must import the OS module, and run its os.remove() function:

import os  
os.remove("python1.txt")

import os  
if os.path.exists("demofile.txt"):  
  os.remove("demofile.txt")  
else:  
  print("The file does not exists")

# **Python Exception Handling**

Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.

Whenever an exception occurs the program halts the execution and thus further code is not executed. Thus exception is that error which python script is unable to tackle with.

Exception in a code can also be handled. In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

### Syntax

Here is simple syntax of *try....except...else* blocks −

try:

You do your operations here;

......................

except*ExceptionI*:

If there is ExceptionI, then execute this block.

except*ExceptionII*:

If there is ExceptionII, then execute this block.

......................

else:

If there is no exception then execute this block.

**Common Exceptions**

**ArithmeticError**

Base class for all errors that occur for numeric calculation.

**TypeError**

Raised when an operation or function is attempted that is invalid for the specified data type.

**ValueError**

Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.

# In which we checking ArithmeticError by passing second argument as 0(zero)

try:

a=int(input("Enter Ist no"))

b=int(input("Enter IInd no"))

c=a/b

print (c)

exceptArithmeticError:

print ("Divide by zero is not allowed")

else:

print ("Welcome")

# In which we are checking data type error

try:

a=4

b="a"

c=a/b

print (c)

exceptTypeError:

print ("Plz input integer no.")

else:

print ("Welcome")

# In which we are checking ArithmeticError and ValueError

try:

a=int(input("Enter Ist no"))

b=int(input("Enter IInd no"))

c=a/b

print (c)

exceptArithmeticError:

print ("Divide by zero is not allowed ")

exceptValueError:

print ("Plz Input Number Only")

else:

print ("Welcome")

Note: In this program if we use “w” file mode then data will be stored but if we use “r” file mode then error will be come because we handled exception

try:

fh = open("testfile", "r")

fh.write("This is my test file for exception handling!!")

exceptIOError:

print("Error: can\'t find file or read data")

else:

print("Written content in the file successfully")

# Python GUI – tkinter

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter outputs the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.  
**To create a tkinter:**

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

Importing tkinter is same as importing any other module in the python code. Note that the name of the module in Python 2.x is ‘Tkinter’ and in Python 3.x is ‘tkinter’.

import tkinter

There are two main methods used you the user need to remember while creating the Python application with GUI.

1. **Tk(screenName=None,  baseName=None,  className=’Tk’,  useTk=1):** To create a main window, tkinter offers a method ‘Tk(screenName=None,  baseName=None,  className=’Tk’,  useTk=1)’. To change the name of the window, you can change the className to the desired one. The basic code used to create the main window of the application is:

m=tkinter.Tk() where m is the name of the main window object

1. **mainloop():** There is a method known by the name mainloop() is used when you are ready for the application to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event till the window is not closed.

m.mainloop()

*filter\_none*

*brightness\_4*

|  |
| --- |
| import tkinter  m = tkinter.Tk()  '''  widgets are added here  '''  m.mainloop() |

tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows. There are mainly three geometry manager classes class.

1. **pack() method:**It organizes the widgets in blocks before placing in the parent widget.
2. **grid() method:**It organizes the widgets in grid (table-like structure) before placing in the parent widget.
3. **place() method:**It organizes the widgets by placing them on specific positions directed by the programmer.

There are a number of widgets which you can put in your tkinter application. Some of the major widgets are explained below:

1. **Button**:To add a button in your application, this widget is used.  
   The general syntax is:

w=Button(master, option=value)

master is the parameter used to represent the parent window.  
There are number of options which are used to change the format of the Buttons. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **activebackground**: to set the background color when button is under the cursor.
  + **activeforeground**: to set the foreground color when button is under the cursor.
  + **bg**: to set he normal background color.
  + **command**: to call a function.
  + **font**: to set the font on the button label.
  + **image**: to set the image on the button.
  + **width**: to set the width of the button.
  + **height**: to set the height of the button.

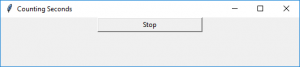
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| import tkinter as tk  r = tk.Tk()  r.title('Counting Seconds')  button = tk.Button(r, text='Stop', width=25, command=r.destroy)  button.pack()  r.mainloop() |

Output:  


1. **Canvas:** It is used to draw pictures and other complex layout like graphics, text and widgets.  
   The general syntax is:
2. w = Canvas(master, option=value)

master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **bd**: to set the border width in pixels.
  + **bg**: to set the normal background color.
  + **cursor**: to set the cursor used in the canvas.
  + **highlightcolor**: to set the color shown in the focus highlight.
  + **width**: to set the width of the widget.
  + **height**: to set the height of the widget.

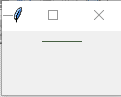
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  master = Tk()  w = Canvas(master, width=40, height=60)  w.pack()  canvas\_height=20  canvas\_width=200  y = int(canvas\_height / 2)  w.create\_line(0, y, canvas\_width, y )  mainloop() |

Output:  


1. **CheckButton:** To select any number of options by displaying a number of options to a user as toggle buttons. The general syntax is:

w = CheckButton(master, option=value)

There are number of options which are used to change the format of this widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **Title**: To set the title of the widget.
  + **activebackground**: to set the background color when widget is under the cursor.
  + **activeforeground**: to set the foreground color when widget is under the cursor.
  + **bg**: to set he normal backgrouSteganography

Break

Secret Code:

Attach a File:nd color.

* + **command**: to call a function.
  + **font**: to set the font on the button label.
  + **image**: to set the image on the widget.

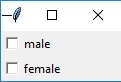
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  master = Tk()  var1 = IntVar()  Checkbutton(master, text='male', variable=var1).grid(row=0, sticky=W)  var2 = IntVar()  Checkbutton(master, text='female', variable=var2).grid(row=1, sticky=W)  mainloop() |

Output:  


1. **Entry:**It is used to input the single line text entry from the user.. For multi-line text input, Text widget is used.  
   The general syntax is:
2. w=Entry(master, option=value)

master is the parameter used to represent the parent window.  
There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **bd**: to set the border width in pixels.
  + **bg**: to set the normal background color.
  + **cursor**: to set the cursor used.
  + **command**: to call a function.
  + **highlightcolor**: to set the color shown in the focus highlight.
  + **width**: to set the width of the button.
  + **height**: to set the height of the button.

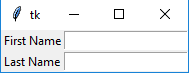
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  master = Tk()  Label(master, text='First Name').grid(row=0)  Label(master, text='Last Name').grid(row=1)  e1 = Entry(master)  e2 = Entry(master)  e1.grid(row=0, column=1)  e2.grid(row=1, column=1)  mainloop() |

Output:  


1. **Frame:** It acts as a container to hold the widgets. It is used for grouping and organizing the widgets. The general syntax is:
2. w = Frame(master, option=value)

master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **highlightcolor**: To set the color of the focus highlight when widget has to be focused.
  + **bd**: to set the border width in pixels.
  + **bg**: to set the normal background color.
  + **cursor**: to set the cursor used.
  + **width**: to set the width of the widget.
  + **height**: to set the height of the widget.

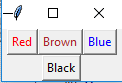
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*    root = Tk()  frame = Frame(root)  frame.pack()  bottomframe = Frame(root)  bottomframe.pack( side = BOTTOM )  redbutton = Button(frame, text = 'Red', fg ='red')  redbutton.pack( side = LEFT)  greenbutton = Button(frame, text = 'Brown', fg='brown')  greenbutton.pack( side = LEFT )  bluebutton = Button(frame, text ='Blue', fg ='blue')  bluebutton.pack( side = LEFT )  blackbutton = Button(bottomframe, text ='Black', fg ='black')  blackbutton.pack( side = BOTTOM)  root.mainloop() |

Output:  


1. **Label**: It refers to the display box where you can put any text or image which can be updated any time as per the code.  
   The general syntax is:
2. w=Label(master, option=value)

master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **bg**: to set he normal background color.
  + **bg** to set he normal background color.
  + **command**: to call a function.
  + **font**: to set the font on the button label.
  + **image**: to set the image on the button.
  + **width**: to set the width of the button.
  + **height**” to set the height of the button.

*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  root = Tk()  w = Label(root, text='GeeksForGeeks.org!')  w.pack()  root.mainloop() |

Output:  
http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Screenshot-68-5.png

1. **Listbox**: It offers a list to the user from which the user can accept any number of options.  
   The general syntax is:
2. w = Listbox(master, option=value)

master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **highlightcolor**: To set the color of the focus highlight when widget has to be focused.
  + **bg**: to set he normal background color.
  + **bd**: to set the border width in pixels.
  + **font**: to set the font on the button label.
  + **image**: to set the image on the widget.
  + **width**: to set the width of the widget.
  + **height**: to set the height of the widget.

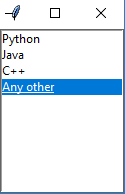
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*    top = Tk()  Lb = Listbox(top)  Lb.insert(1, 'Python')  Lb.insert(2, 'Java')  Lb.insert(3, 'C++')  Lb.insert(4, 'Any other')  Lb.pack()  top.mainloop() |

Output:  


1. **MenuButton**: It is a part of top-down menu which stays on the window all the time. Every menubutton has its own functionality. The general syntax is:
2. w = MenuButton(master, option=value)

master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **activebackground**: To set the background when mouse is over the widget.
  + **activeforeground**: To set the foreground when mouse is over the widget.
  + **bg**: to set he normal background color.
  + **bd**: to set the size of border around the indicator.
  + **cursor**: To appear the cursor when the mouse over the menubutton.
  + **image**: to set the image on the widget.
  + **width**: to set the width of the widget.
  + **height**: to set the height of the widget.
  + **highlightcolor**: To set the color of the focus highlight when widget has to be focused.

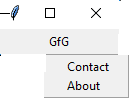
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*    top = Tk()  mb =  Menubutton ( top, text = &quot;GfG&quot;)  mb.grid()  mb.menu  =  Menu ( mb, tearoff = 0 )  mb[&quot;menu&quot;]  =  mb.menu  cVar  = IntVar()  aVar = IntVar()  mb.menu.add\_checkbutton ( label ='Contact', variable = cVar )  mb.menu.add\_checkbutton ( label = 'About', variable = aVar )  mb.pack()  top.mainloop() |

Output:  


1. **Menu**: It is used to create all kinds of menus used by the application.  
   The general syntax is:
2. w = Menu(master, option=value)

master is the parameter used to represent the parent window.

There are number of options which are used to change the format of this widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **title**: To set the title of the widget.
  + **activebackground**: to set the background color when widget is under the cursor.
  + **activeforeground**: to set the foreground color when widget is under the cursor.
  + **bg**: to set he normal background color.
  + **command**: to call a function.
  + **font**: to set the font on the button label.
  + **image**: to set the image on the widget.

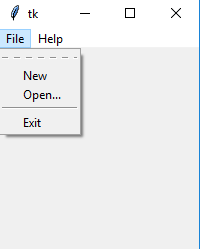
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*    root = Tk()  menu = Menu(root)  root.config(menu=menu)  filemenu = Menu(menu)  menu.add\_cascade(label='File', menu=filemenu)  filemenu.add\_command(label='New')  filemenu.add\_command(label='Open...')  filemenu.add\_separator()  filemenu.add\_command(label='Exit', command=root.quit)  helpmenu = Menu(menu)  menu.add\_cascade(label='Help', menu=helpmenu)  helpmenu.add\_command(label='About')  mainloop() |

Output:  


1. **Message**: It refers to the multi-line and non-editable text. It works same as that of Label.  
   The general syntax is:
2. w = Message(master, option=value)
3. master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **bd**: to set the border around the indicator.
  + **bg**: to set he normal background color.
  + **font**: to set the font on the button label.
  + **image**: to set the image on the widget.
  + **width**: to set the width of the widget.
  + **height**: to set the height of the widget.

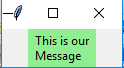
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  main = Tk()  ourMessage ='This is our Message'  messageVar = Message(main, text = ourMessage)  messageVar.config(bg='lightgreen')  messageVar.pack( )  main.mainloop( ) |

Output:  


1. **RadioButton:** It is used to offer multi-choice option to the user. It offers several options to the user and the user has to choose one option.  
   The general syntax is:
2. w = RadioButton(master, option=value)

There are number of options which are used to change the format of this widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **activebackground**: to set the background color when widget is under the cursor.
  + **activeforeground**: to set the foreground color when widget is under the cursor.
  + **bg**: to set he normal background color.
  + **command**: to call a function.
  + **font**: to set the font on the button label.
  + **image**: to set the image on the widget.
  + **width**: to set the width of the label in characters.
  + **height**: to set the height of the label in characters.

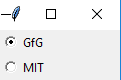
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  root = Tk()  v = IntVar()  Radiobutton(root, text='GfG', variable=v, value=1).pack(anchor=W)  Radiobutton(root, text='MIT', variable=v, value=2).pack(anchor=W)  mainloop() |

Output:  


1. **Scale:** It is used to provide a graphical slider that allows to select any value from that scale. The general syntax is:
2. w = Scale(master, option=value)
3. master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **cursor**: To change the cursor pattern when the mouse is over the widget.
  + **activebackground**: To set the background of the widget when mouse is over the widget.
  + **bg**: to set he normal background color.
  + **orient**: Set it to HORIZONTAL or VERTICAL according to the requirement.
  + **from\_**: To set the value of one end of the scale range.
  + **to**: To set the value of the other end of the scale range.
  + **image**: to set the image on the widget.
  + **width**: to set the width of the widget.

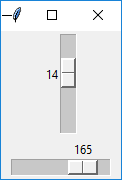
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  master = Tk()  w = Scale(master, from\_=0, to=42)  w.pack()  w = Scale(master, from\_=0, to=200, orient=HORIZONTAL)  w.pack()  mainloop() |

Output:  


1. **Scrollbar**: It refers to the slide controller which will be used to implement listed widgets.  
   The general syntax is:
2. w = Scrollbar(master, option=value)
3. master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **width**: to set the width of the widget.
  + **activebackground**: To set the background when mouse is over the widget.
  + **bg**: to set he normal background color.
  + **bd**: to set the size of border around the indicator.
  + **cursor**: To appear the cursor when the mouse over the menubutton.

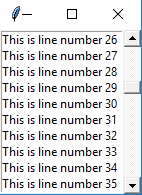
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  root = Tk()  scrollbar = Scrollbar(root)  scrollbar.pack( side = RIGHT, fill = Y )  mylist = Listbox(root, yscrollcommand = scrollbar.set )  for line in range(100):     mylist.insert(END, 'This is line number' + str(line))  mylist.pack( side = LEFT, fill = BOTH )  scrollbar.config( command = mylist.yview )  mainloop() |

Output:  


1. **Text:** To edit a multi-line text and format the way it has to be displayed.  
   The general syntax is:
2. w =Text(master, option=value)

There are number of options which are used to change the format of the text. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **highlightcolor**: To set the color of the focus highlight when widget has to be focused.
  + **insertbackground**: To set the background of the widget.
  + **bg**: to set he normal background color.
  + **font**: to set the font on the button label.
  + **image**: to set the image on the widget.
  + **width**: to set the width of the widget.
  + **height**: to set the height of the widget.

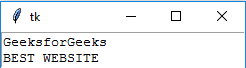
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  root = Tk()  T = Text(root, height=2, width=30)  T.pack()  T.insert(END, 'GeeksforGeeks\nBEST WEBSITE\n')  mainloop() |

Output:  


1. **TopLevel:** This widget is directly controlled by the window manager. It don’t need any parent window to work on.The general syntax is:
2. w = TopLevel(master, option=value)

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **bg**: to set he normal background color.
  + **bd**: to set the size of border around the indicator.
  + **cursor**: To appear the cursor when the mouse over the menubutton.
  + **width**: to set the width of the widget.
  + **height**: to set the height of the widget.

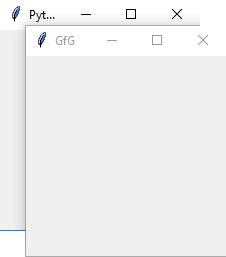
*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  root = Tk()  root.title('GfG')  top = Toplevel()  top.title('Python')  top.mainloop() |

Output:  


1. **SpinBox:** It is an entry of ‘Entry’ widget. Here, value can be input by selecting a fixed value of numbers.The general syntax is:
2. w = SpinBox(master, option=value)

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **bg**: to set he normal background color.
  + **bd**: to set the size of border around the indicator.
  + **cursor**: To appear the cursor when the mouse over the menubutton.
  + **command**: To call a function.
  + **width**: to set the width of the widget.
  + **activebackground**: To set the background when mouse is over the widget.
  + **disabledbackground**: To disable the background when mouse is over the widget.
  + **from\_**: To set the value of one end of the range.
  + **to**: To set the value of the other end of the range.

*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  master = Tk()  w = Spinbox(master, from\_ = 0, to = 10)  w.pack()  mainloop() |

Output:  
http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Screenshot-68-16.png

1. **PannedWindow**It is a container widget which is used to handle number of panes arranged in it. The general syntax is:
2. w = PannedWindow(master, option=value)

master is the parameter used to represent the parent window.  
There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

* + **bg**: to set he normal background color.
  + **bd**: to set the size of border around the indicator.
  + **cursor**: To appear the cursor when the mouse over the menubutton.
  + **width**: to set the width of the widget.
  + **height**: to set the height of the widget.

*filter\_none*

*edit*

*play\_arrow*

*brightness\_4*

|  |
| --- |
| from tkinter import \*  m1 = PanedWindow()  m1.pack(fill = BOTH, expand = 1)  left = Entry(m1, bd = 5)  m1.add(left)  m2 = PanedWindow(m1, orient = VERTICAL)  m1.add(m2)  top = Scale( m2, orient = HORIZONTAL)  m2.add(top)  mainloop() |

Output:  


Creating simple form from tkinter in python

from tkinter import \*  
  
root = Tk()  
  
def getvals():  
 print("It works!")  
root.geometry("644x344")  
#Heading  
Label(root, text="Simple Form", font="comicsansms 13 bold", pady=15).grid(row=0, column=3)  
  
#Text for our form  
name = Label(root, text="Name")  
phone = Label(root, text="Phone")  
gender = Label(root, text="Gender")  
emergency = Label(root, text="Emergency Contact")  
paymentmode = Label(root, text="Payment Mode")  
  
#Pack text for our form  
name.grid(row=1, column=2)  
phone.grid(row=2, column=2)  
gender.grid(row=3, column=2)  
emergency.grid(row=4, column=2)  
paymentmode.grid(row=5, column=2)  
  
# Tkinter variable for storing entries  
namevalue = StringVar()  
phonevalue = StringVar()  
gendervalue = StringVar()  
emergencyvalue = StringVar()  
paymentmodevalue = StringVar()  
foodservicevalue = IntVar()  
  
  
#Entries for our form  
nameentry = Entry(root, textvariable=namevalue)  
phoneentry = Entry(root, textvariable=phonevalue)  
genderentry = Entry(root, textvariable=gendervalue)  
emergencyentry = Entry(root, textvariable=emergencyvalue)  
paymentmodeentry = Entry(root, textvariable=paymentmodevalue)  
  
# Packing the Entries  
nameentry.grid(row=1, column=3)  
phoneentry.grid(row=2, column=3)  
genderentry.grid(row=3, column=3)  
emergencyentry.grid(row=4, column=3)  
paymentmodeentry.grid(row=5, column=3)  
  
#Checkbox & Packing it  
foodservice = Checkbutton(text="Want to prebook your meals?", variable = foodservicevalue)  
foodservice.grid(row=6, column=3)  
  
#Button & packing it and assigning it a command  
Button(text="Submit", command=getvals).grid(row=7, column=3)  
  
  
  
root.mainloop()

## Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.

Here are naming conventions for Python identifiers −

* Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
* Starting an identifier with a single leading underscore indicates that the identifier is private.
* Starting an identifier with two leading underscores indicates a strongly private identifier.
* If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

## Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example −

if True:

print "True"

else:

print "False"

However, the following block generates an error −

if True:

print "Answer"

print "True"

else:

print "Answer"

print "False"

# Python Numpy

**Numpy** is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.  
Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

Installation of numpy

n Windows, first open command prompt.

Then change directory to \***Scripts*** by typing the correct path.

Then type ***pip list*** *.* A list appears to show what has been installed.

Almost there!

Then type ***pip install numpy*** *.* numpy will be installed.

#### Arrays in Numpy

Array in Numpy is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy, number of dimensions of the array is called rank of the array.A tuple of integers giving the size of the array along each dimension is known as shape of the array. An array class in Numpy is called as **ndarray**. Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists.

**Creating a Numpy Array**  
Arrays in Numpy can be created by multiple ways, with various number of Ranks, defining the size of the Array. Arrays can also be created with the use of various data types such as lists, tuples, etc. The type of the resultant array is deduced from the type of the elements in the sequences.  
**Note:** Type of array can be explicitly defined while creating the array.

|  |
| --- |
| # Python program for  # Creation of Arrays  import numpy as np    # Creating a rank 1 Array  arr = np.array([1, 2, 3])  print("Array with Rank 1: \n",arr)    # Creating a rank 2 Array  arr = np.array([[1, 2, 3],                  [4, 5, 6]])  print("Array with Rank 2: \n", arr)    # Creating an array from tuple  arr = np.array((1, 3, 2))  print("\nArray created using "        "passed tuple:\n", arr) |

**Output:**

Array with Rank 1:

[1 2 3]

Array with Rank 2:

[[1 2 3]

[4 5 6]]

Array created using passed tuple:

[1 3 2]

**Accessing the array Index**  
In a numpy array, indexing or accessing the array index can be done in multiple ways. To print a range of an array, slicing is done. Slicing of an array is defining a range in a new array which is used to print a range of elements from the original array. Since, sliced array holds a range of elements of the original array, modifying content with the help of sliced array modifies the original array content.

|  |
| --- |
| # Python program to demonstrate  # indexing in numpy array  import numpy as np    # Initial Array  arr = np.array([[-1, 2, 0, 4],                  [4, -0.5, 6, 0],                  [2.6, 0, 7, 8],                  [3, -7, 4, 2.0]])  print("Initial Array: ")  print(arr)    # Printing a range of Array  # with the use of slicing method  sliced\_arr = arr[:2, ::2]  print ("Array with first 2 rows and"      " alternate columns(0 and 2):\n", sliced\_arr)    # Printing elements at  # specific Indices  Index\_arr = arr[[1, 1, 0, 3],                  [3, 2, 1, 0]]  print ("\nElements at indices (1, 3), "      "(1, 2), (0, 1), (3, 0):\n", Index\_arr) |

**Output:**

Initial Array:

[[-1. 2. 0. 4. ]

[ 4. -0.5 6. 0. ]

[ 2.6 0. 7. 8. ]

[ 3. -7. 4. 2. ]]

Array with first 2 rows and alternate columns(0 and 2):

[[-1. 0.]

[ 4. 6.]]

Elements at indices (1, 3), (1, 2), (0, 1), (3, 0):

[ 0. 54. 2. 3.]

**Basic Array Operations**  
In numpy, arrays allow a wide range of operations which can be performed on a particular array or a combination of Arrays. These operation include some basic Mathematical operation as well as Unary and Binary operations.

|  |
| --- |
| # Python program to demonstrate  # basic operations on single array  import numpy as np    # Defining Array 1  a = np.array([[1, 2],                [3, 4]])    # Defining Array 2  b = np.array([[4, 3],                [2, 1]])    # Adding 1 to every element  print ("Adding 1 to every element:", a + 1)    # Subtracting 2 from each element  print ("\nSubtracting 2 from each element:", b - 2)    # sum of array elements  # Performing Unary operations  print ("\nSum of all array "         "elements: ", a.sum())    # Adding two arrays  # Performing Binary operations  print ("\nArray sum:\n", a + b) |

**Output:**

Adding 1 to every element:

[[2 3]

[4 5]]

Subtracting 2 from each element:

[[ 2 1]

[ 0 -1]]

Sum of all array elements: 10

Array sum:

[[5 5]

[5 5]]

#### Data Types in Numpy

Every Numpy array is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. Every ndarray has an associated data type (dtype) object. This data type object (dtype) provides information about the layout of the array. The values of an ndarray are stored in a buffer which can be thought of as a contiguous block of memory bytes which can be interpreted by the dtype object. Numpy provides a large set of numeric datatypes that can be used to construct arrays. At the time of Array creation, Numpy tries to guess a datatype, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype.

**Constructing a Datatype Object**  
In Numpy, datatypes of Arrays need not to be defined unless a specific datatype is required. Numpy tries to guess the datatype for Arrays which are not predefined in the constructor function.

|  |
| --- |
| # Python Program to create  # a data type object  import numpy as np    # Integer datatype  # guessed by Numpy  x = np.array([1, 2])  print("Integer Datatype: ")  print(x.dtype)    # Float datatype  # guessed by Numpy  x = np.array([1.0, 2.0])  print("\nFloat Datatype: ")  print(x.dtype)    # Forced Datatype  x = np.array([1, 2], dtype = np.int64)  print("\nForcing a Datatype: ")  print(x.dtype) |

**Output:**

Integer Datatype:

int64

Float Datatype:

float64

Forcing a Datatype:

int64