Often in statistics, we tend to represent a set of data by a representative value which would approximately define the entire collection. This representative value is called the measure of central tendency, and the name suggests that it is a value around which the data is centred. These central tendencies are mean, median and mode.

```
import statistics
data = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9]
# Mean
mean = statistics.mean(data)
print("Mean:", mean)
# Median
median = statistics.median(data)
print("Median:", median)
# Mode
mode = statistics.mode(data)
print("Mode:", mode)
# Variance
variance = statistics.variance(data)
print("Variance:", variance)
# Standard Deviation
std_deviation = statistics.stdev(data)
print("Standard Deviation:", std_deviation)
```

```
def variance(data):
...     # Number of observations
...     n = len(data)
...     # Mean of the data
...     mean = sum(data) / n
...     # Square deviations
...     deviations = [(x - mean) ** 2 for x in data]
...     # Variance
...     variance = sum(deviations) / n
...     return variance
...

>>> variance([4, 8, 6, 5, 3, 2, 8, 9, 2, 5])
5.76
```

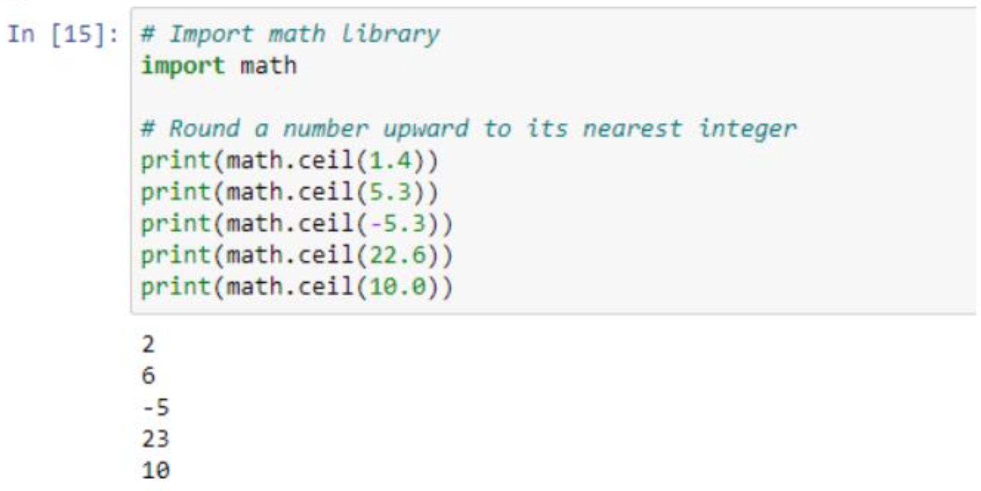
```
import math

>>> # We rely on our previous implementation for the variance
>>> def variance(data, ddof=0):
...     n = len(data)
...     mean = sum(data) / n
...     return sum((x - mean) ** 2 for x in data) / (n - ddof)
...

>>> def stdev(data):
...     var = variance(data)
...     std_dev = math.sqrt(var)
...     return std_dev

>>> stdev([4, 8, 6, 5, 3, 2, 8, 9, 2, 5])
2.4
```

Python Math Library

The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math. It gives access tothe underlying C library functions. This module does not support complex datatypes. The cmath module is the complex counter part.

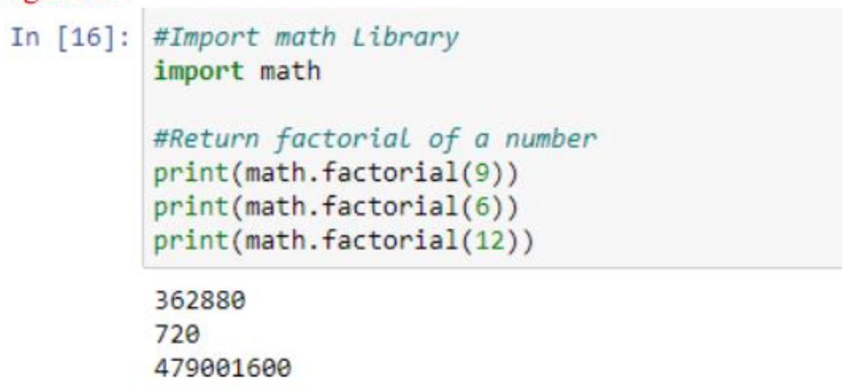
List of Functions in Python Math Module	
Function	**Description**
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
frexp(x)	Returns the mantissa and exponent of x as the pair (m, e)
fsum(iterable)	Returns an accurate floating point sum of values in the iterable
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
isinf(x)	Returns True if x is a positive or negative infinity
isnan(x)	Returns True if x is a NaN
ldexp(x, i)	Returns x * (2**i)
modf(x)	Returns the fractional and integer parts of x
trunc(x)	Returns the truncated integer value of x
exp(x)	Returns e**x
expm1(x)	Returns e**x - 1

rogram-1



```
In [15]: # Import math Library
         import math

         # Round a number upward to its nearest integer
         print(math.ceil(1.4))
         print(math.ceil(5.3))
         print(math.ceil(-5.3))
         print(math.ceil(22.6))
         print(math.ceil(10.0))

         2
         6
         -5
         23
         10
```

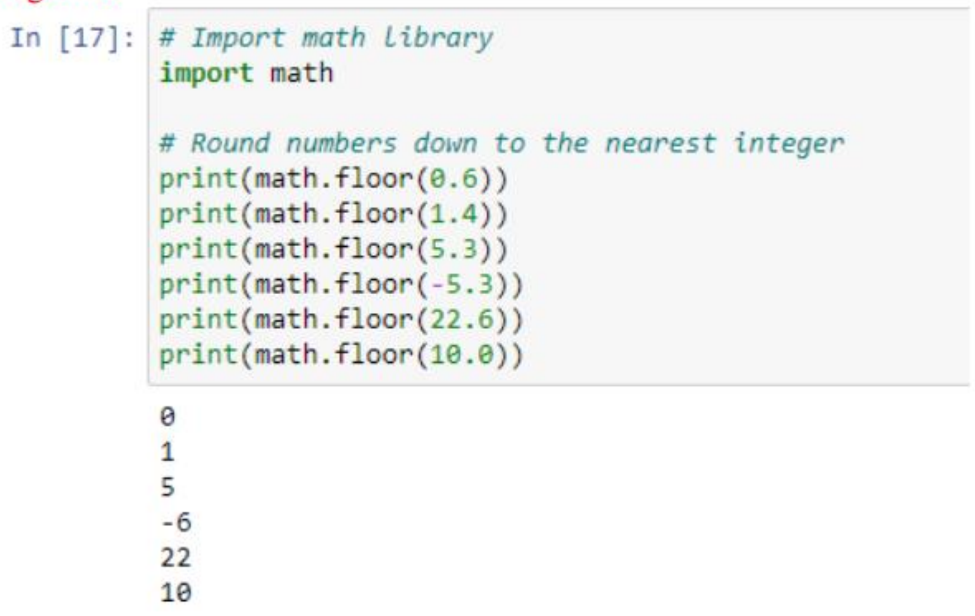
Program-2



```
In [16]: #Import math Library
         import math

         #Return factorial of a number
         print(math.factorial(9))
         print(math.factorial(6))
         print(math.factorial(12))
```

```
362880
720
479001600
```

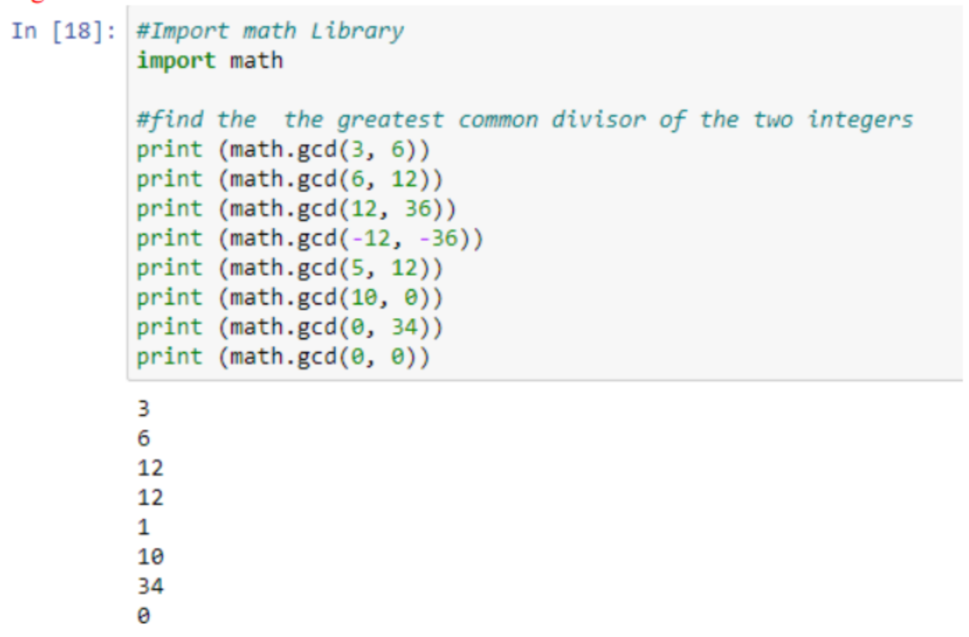
rogram-3



```
In [17]: # Import math Library
         import math

         # Round numbers down to the nearest integer
         print(math.floor(0.6))
         print(math.floor(1.4))
         print(math.floor(5.3))
         print(math.floor(-5.3))
         print(math.floor(22.6))
         print(math.floor(10.0))
```

```
0
1
5
-6
22
10
```

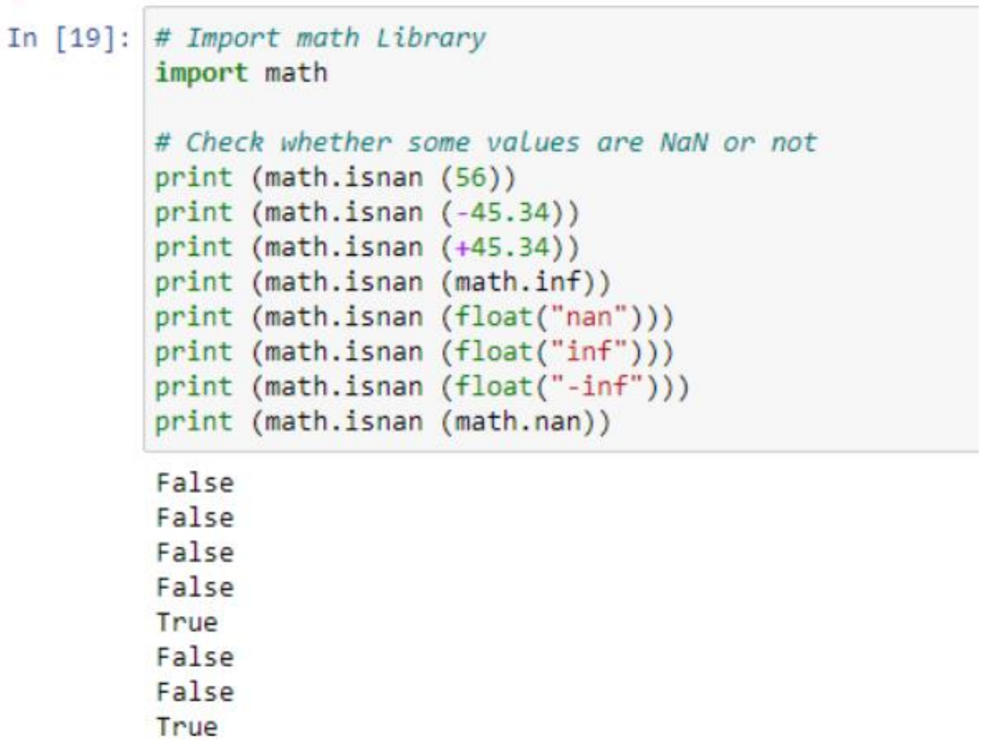
Program-4



```
In [18]: #Import math Library
         import math

         #find the  the greatest common divisor of the two integers
         print (math.gcd(3, 6))
         print (math.gcd(6, 12))
         print (math.gcd(12, 36))
         print (math.gcd(-12, -36))
         print (math.gcd(5, 12))
         print (math.gcd(10, 0))
         print (math.gcd(0, 34))
         print (math.gcd(0, 0))
```

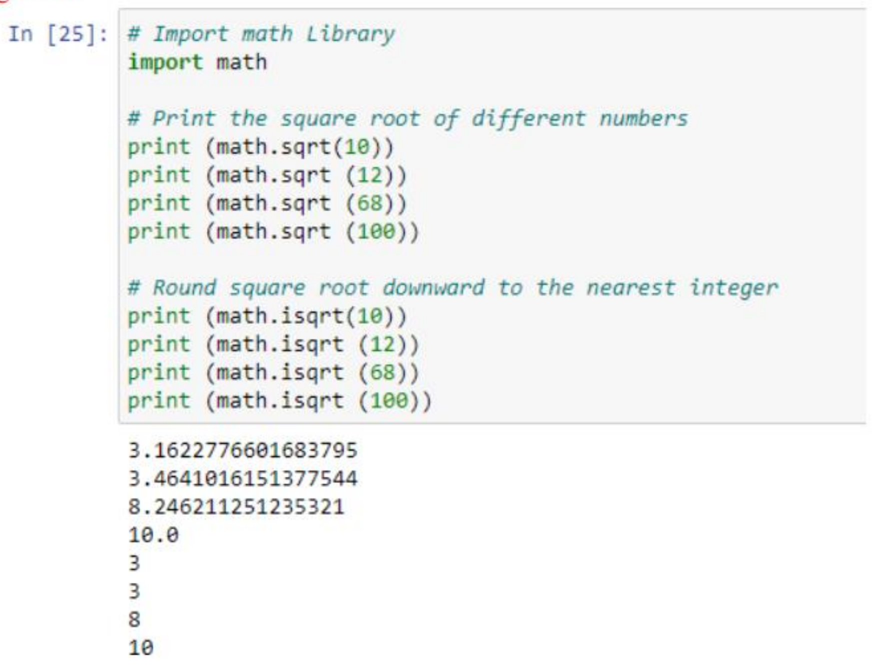
```
3
6
12
12
1
10
34
0
```

Program-5



```
In [19]: # Import math Library
         import math

         # Check whether some values are NaN or not
         print (math.isnan (56))
         print (math.isnan (-45.34))
         print (math.isnan (+45.34))
         print (math.isnan (math.inf))
         print (math.isnan (float("nan")))
         print (math.isnan (float("inf")))
         print (math.isnan (float("-inf")))
         print (math.isnan (math.nan))

         False
         False
         False
         False
         True
         False
         False
         True
```

Program-6



```
In [25]: # Import math Library
         import math

         # Print the square root of different numbers
         print (math.sqrt(10))
         print (math.sqrt (12))
         print (math.sqrt (68))
         print (math.sqrt (100))

         # Round square root downward to the nearest integer
         print (math.isqrt(10))
         print (math.isqrt (12))
         print (math.isqrt (68))
         print (math.isqrt (100))

         3.1622776601683795
         3.4641016151377544
         8.246211251235321
         10.0
         3
         3
         8
         10
```

Python Numpy Library

NumPy is an open source library available in Python that aids in mathematical, scientific, engineering, and data science programming. NumPy is an incredible library to perform mathematical and statisticaloperations. It works perfectly well for multi-dimensional arrays and matrices multiplication

- **Arrays in NumPy:** NumPy's main object is the homogeneous multidimensional array.

  ⍰ It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positiveintegers.
  ⍰ In NumPy dimensions are called axes. The number of axes is rank.
  ⍰ NumPy's array class is called **ndarray**. It is also known by the alias **array**.

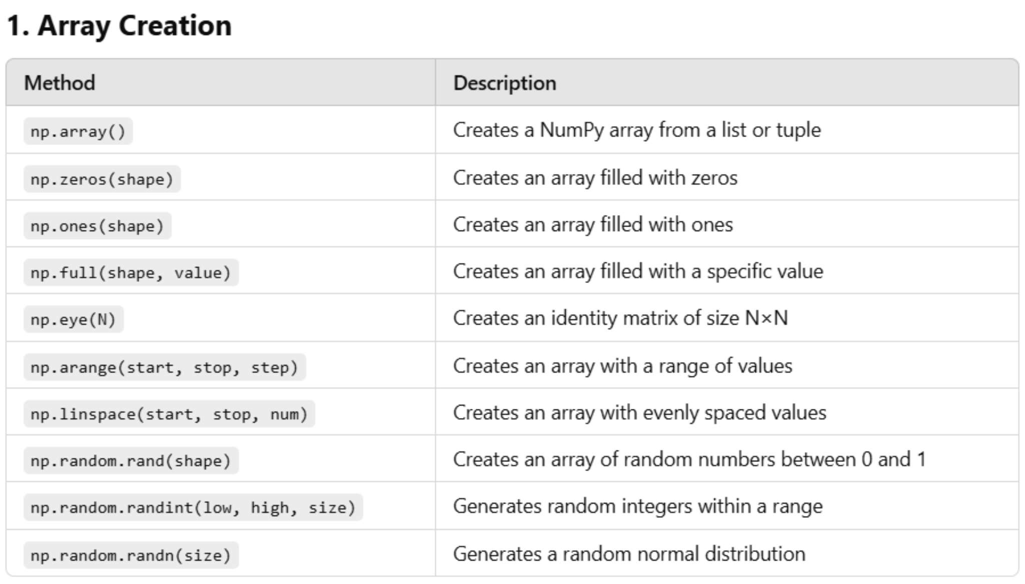
  We use python numpy array instead of a list because of the below three reasons:
  1. Less Memory
  2. Fast
  3. Convenient

- **Numpy Functions**

  Numpy arrays carry attributes around with them. The most important
  ones are:ndim: The number of axes or rank of the array
  shape: A tuple containing the length in each
  dimensionsize: The total number of elements

NumPy provides a vast collection of methods for various operations, including array creation, mathematical computations, statistical analysis, linear algebra, and more. Below is a categorized list of some commonly used NumPy methods:

1. Array Creation



1. Array Creation

Method	Description
`np.array()`	Creates a NumPy array from a list or tuple
`np.zeros(shape)`	Creates an array filled with zeros
`np.ones(shape)`	Creates an array filled with ones
`np.full(shape, value)`	Creates an array filled with a specific value
`np.eye(N)`	Creates an identity matrix of size N×N
`np.arange(start, stop, step)`	Creates an array with a range of values
`np.linspace(start, stop, num)`	Creates an array with evenly spaced values
`np.random.rand(shape)`	Creates an array of random numbers between 0 and 1
`np.random.randint(low, high, size)`	Generates random integers within a range
`np.random.randn(size)`	Generates a random normal distribution

2. Array Manipulation

Method	Description
`np.reshape(array, new_shape)`	Reshapes an array without changing its data
`np.ravel(array)`	Flattens an array into a 1D array
`np.transpose(array)`	Transposes a matrix (rows ↔ columns)
`np.concatenate((arr1, arr2), axis=0)`	Joins two or more arrays along an axis
`np.hstack((arr1, arr2))`	Horizontally stacks arrays
`np.vstack((arr1, arr2))`	Vertically stacks arrays
`np.split(array, sections)`	Splits an array into multiple sub-arrays

3. Mathematical Operations

Method	Description
`np.add(arr1, arr2)`	Element-wise addition
`np.subtract(arr1, arr2)`	Element-wise subtraction
`np.multiply(arr1, arr2)`	Element-wise multiplication
`np.divide(arr1, arr2)`	Element-wise division
`np.exp(array)`	Computes the exponential of each element
`np.log(array)`	Computes the natural logarithm (ln)
`np.sqrt(array)`	Computes the square root of each element
`np.power(array, n)`	Raises elements to the power of `n`
`np.abs(array)`	Computes the absolute values
`np.mod(arr1, arr2)`	Computes element-wise remainder
`np.dot(arr1, arr2)`	Computes dot product
`np.cross(arr1, arr2)`	Computes cross product

4. Statistical & Aggregation Methods

Method	Description
`np.mean(array)`	Computes the mean (average)
`np.median(array)`	Computes the median
`np.std(array)`	Computes the standard deviation
`np.var(array)`	Computes the variance
`np.min(array)`	Finds the minimum value
`np.max(array)`	Finds the maximum value
`np.percentile(array, q)`	Computes the `q`-th percentile
`np.histogram(array, bins)`	Computes histogram bins
`np.corrcoef(array1, array2)`	Computes Pearson correlation coefficient

5. Linear Algebra Functions (`numpy.linalg`)

Method	Description
`np.linalg.inv(matrix)`	Computes the inverse of a matrix
`np.linalg.det(matrix)`	Computes the determinant of a matrix
`np.linalg.eig(matrix)`	Computes eigenvalues and eigenvectors
`np.linalg.svd(matrix)`	Computes Singular Value Decomposition
`np.linalg.norm(array)`	Computes the norm of a vector
`np.linalg.solve(A, B)`	Solves a system of linear equations

6. Sorting & Searching

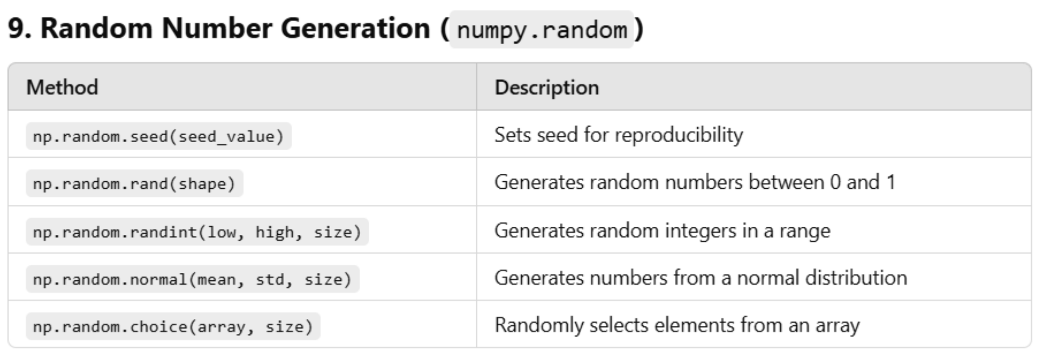
Method	Description
`np.sort(array)`	Sorts elements in ascending order
`np.argsort(array)`	Returns indices that would sort an array
`np.argmin(array)`	Returns index of the minimum element
`np.argmax(array)`	Returns index of the maximum element
`np.where(condition)`	Returns indices where condition is met
`np.unique(array)`	Returns unique elements of an array

7. Logical Operations

Method	Description
`np.all(condition)`	Checks if all elements meet a condition
`np.any(condition)`	Checks if any element meets a condition
`np.logical_and(arr1, arr2)`	Element-wise logical AND
`np.logical_or(arr1, arr2)`	Element-wise logical OR
`np.logical_not(arr)`	Element-wise logical NOT

8. Broadcasting & Masking

Method	Description
`np.clip(array, min, max)`	Limits values within a given range
`np.isnan(array)`	Checks for NaN (Not a Number) values
`np.isinf(array)`	Checks for infinity values
`np.ma.masked_where(condition, array)`	Masks elements that meet a condition



9. Random Number Generation (`numpy.random`)

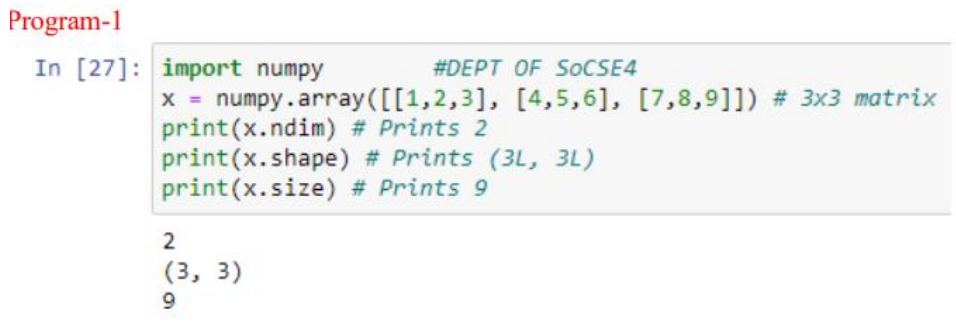
Method	Description
`np.random.seed(seed_value)`	Sets seed for reproducibility
`np.random.rand(shape)`	Generates random numbers between 0 and 1
`np.random.randint(low, high, size)`	Generates random integers in a range
`np.random.normal(mean, std, size)`	Generates numbers from a normal distribution
`np.random.choice(array, size)`	Randomly selects elements from an array

```
import statistics as stats
import math
import numpy as np
from scipy import stats as scipy_stats

# Sample data
data = [12, 15, 14, 10, 18, 20, 22, 24, 19, 17]

# Using statistics module
mean_value = stats.mean(data)
median_value = stats.median(data)
mode_value = stats.mode(data)
stdev_value = stats.stdev(data)

print("Statistics Module:")
print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Mode: {mode_value}")
print(f"Standard Deviation: {stdev_value}\n")

# Using math module
num = 16
sqrt_value = math.sqrt(num)
factorial_value = math.factorial(5)
log_value = math.log(100, 10)

print("Math Module:")
print(f"Square Root of {num}: {sqrt_value}")
print(f"Factorial of 5: {factorial_value}")
print(f"Log base 10 of 100: {log_value}\n")

# Using numpy
arr = np.array(data)
arr_mean = np.mean(arr)
```

```
arr_variance = np.var(arr)

print("NumPy Module:")
print(f"Mean using NumPy: {arr_mean}")
print(f"Variance using NumPy: {arr_variance}\n")

# Using scipy
z_score = scipy_stats.zscore(arr)
t_stat, p_val = scipy_stats.ttest_1samp(arr, 15)

print("SciPy Module:")
print(f"Z-Scores: {z_score}")
print(f"T-Test (compared to mean=15): T-stat={t_stat}, P-value={p_val}")
```

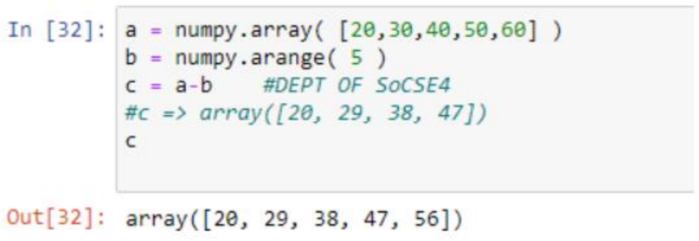
Numpy arrays carry attributes around with them.

The most important ones are:ndim: The number of axes or rank of the array shape: A tuple containing the length in each dimensionsize: The total number of elements


Program-1

```
In [27]: import numpy        #DEPT OF SoCSE4
         x = numpy.array([[1,2,3], [4,5,6], [7,8,9]]) # 3x3 matrix
         print(x.ndim) # Prints 2
         print(x.shape) # Prints (3L, 3L)
         print(x.size) # Prints 9

         2
         (3, 3)
         9
```

Can be used just like Python lists
x[1] will access the second element
x[-1] will access the last element

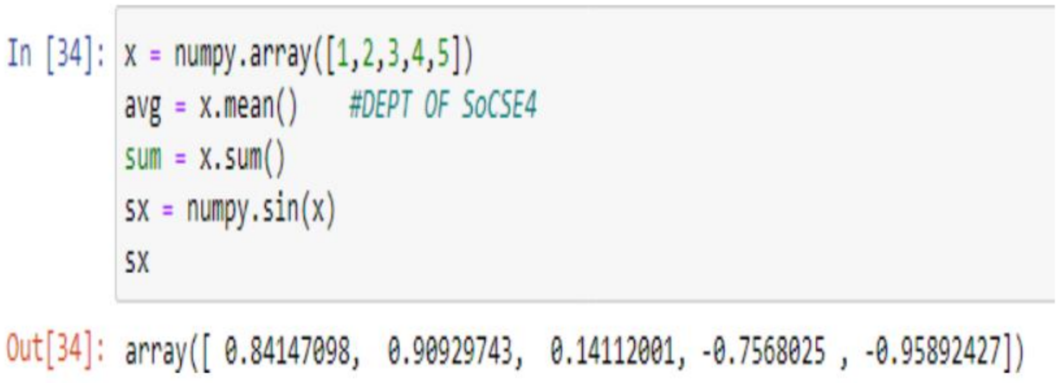


```
In [32]: a = numpy.array( [20,30,40,50,60] )
         b = numpy.arange( 5 )
         c = a-b    #DEPT OF SoCSE4
         #c => array([20, 29, 38, 47])
         c

Out[32]: array([20, 29, 38, 47, 56])
```

- **Built-in Methods**

Many standard numerical functions are available as methods out of the box:

Program-3



```
In [34]: x = numpy.array([1,2,3,4,5])
         avg = x.mean()    #DEPT OF SoCSE4
         sum = x.sum()
         sx = numpy.sin(x)
         sx

Out[34]: array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.95892427])
```

- **Python Scipy Library**

SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing. SciPy also pronounced as "Sigh Pi."

- SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
- SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
- Easy to use and understand as well as fast computational power.
- It can operate on an array of NumPy library.

Numpy VS SciPyNumpy:

1. Numpy is written in C and use for mathematical or numeric calculation.
2. It is faster than other Python Libraries
3. Numpy is the most useful library for Data Science to perform basic calculations.
4. Numpy contains nothing but array data type which performs the most basic operation like sorting,shaping, indexing, etc.

SciPy:

1. SciPy is built in top of the NumPy
2. SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
3. Most new Data Science features are available in Scipy rather than Numpy.

Linear Algebra with SciPy

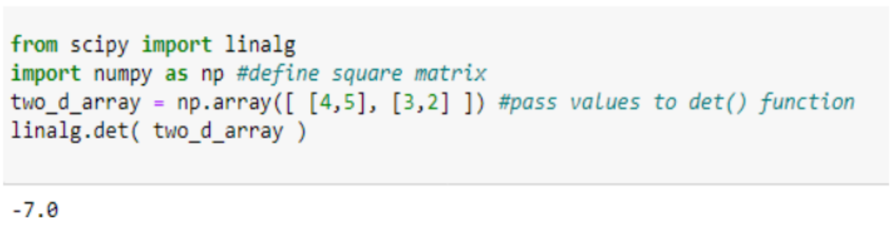
1. Linear Algebra of SciPy is an implementation of BLAS and ATLAS LAPACK libraries.
2. Performance of Linear Algebra is very fast compared to BLAS and LAPACK.

Linear algebra routine accepts two-dimensional array object and output is also a two-dimensional array.

Now let's do some test with **scipy.linalg,**

Calculating **determinant** of a two-dimensional matrix,

Program-1



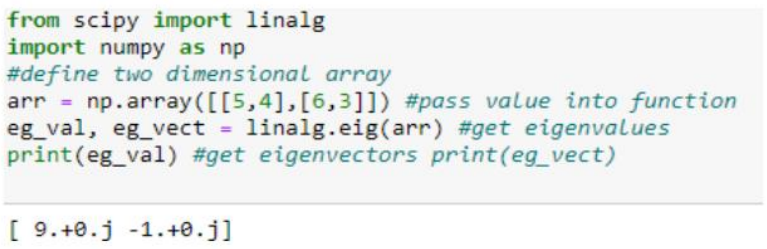
```
from scipy import linalg
import numpy as np #define square matrix
two_d_array = np.array([ [4,5], [3,2] ]) #pass values to det() function
linalg.det( two_d_array )
```

```
-7.0
```

Eigenvalues and Eigenvector – scipy.linalg.eig()

- The most common problem in linear algebra is eigenvalues and eigenvector which can beeasily solved using **eig()** function.
- Now lets we find the Eigenvalue of (**X**) and correspond eigenvector of a two-dimensionalsquare matrix.

Program-2



```
from scipy import linalg
import numpy as np
#define two dimensional array
arr = np.array([[5,4],[6,3]]) #pass value into function
eg_val, eg_vect = linalg.eig(arr) #get eigenvalues
print(eg_val) #get eigenvectors print(eg_vect)
```

```
[ 9.+0.j -1.+0.j]
```