UNIT V Syllabus

Libraries in Python

1. NumPy: Basic Operation, Indexing, slicing and Iterating, multidimensional arrays, NumPy Data types, Reading and writing data on Files.
2. Pandas : Series and Data Frames, Grouping, aggregation, Merge Data Frames, Generate summary tables, Group data into logical pieces, Manipulation of data.
3. SciPy: Introduction to SciPy, Create function, modules of SciPy.
4. Matplotlib: Scatter plot, Bar charts, histogram, Stack charts, Legend title Style, Figures and subplots, Plotting function in pandas, Labelling and arranging figures, Save plots.
5. Seaborn: style function, color palettes, distribution plots, category plot, regression plot.

What is a Library?

1. A library is a collection of pre-combined codes that can be used iteratively to reduce the time required to code.
2. They are particularly useful for accessing the pre-written frequently used codes, instead of writing them from scratch every single time.
3. Similar to the physical libraries, these are a collection of reusable resources, which means every library has a root source.

This is the foundation behind the numerous open-source libraries available in Python.

1. NumPy
2. Pandas
3. Matplotlib
4. Scipy
5. Seaborn

NumPy (Numerical Python)

1. It is an open source python library used in pandas,Scipy,Matplotlib scikit,Tensorflow etc.... packages
2. Numpy library contains multidimensions array and collection of routines for processing of these array.
3. 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 non-negative integers.
4. In NumPy dimensions are called axes.

Reason for using NumPy

1. It provides efficient storage for data values
2. It is fast as compare to builtins data structure list
3. It is easy to learn
4. Provides large collection of function for processing data values.
5. Suitable for matrix data structure.

In [2]:
```
import sys
list1=[1,2]
print(sys.getsizeof(list1))
list2=[3,4]
list3=list1 + list2
print(list3)
```

```
72
[1, 2, 3, 4]
```

In [1]:
```
print(len(dir(str)))
```

```
80
```

Installing NumPy

In [3]:
```
!pip --version
```

```
pip 21.1.3 from c:\users\sachin kumar\appdata\local\programs\python\python39\lib\site
-packages\pip (python 3.9)
```

In [4]:
```
!pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\sachin kumar\appdata\local\programs
\python\python39\lib\site-packages (1.21.0)
```

In [5]:
```
!pip install --upgrade pip
```

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=Non
e)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connection.H
TTPSConnection object at 0x000001D50F973850>: Failed to establish a new connection:
[Errno 11001] getaddrinfo failed')': /simple/pip/
Requirement already satisfied: pip in c:\users\sachin kumar\appdata\local\programs\py
thon\python39\lib\site-packages (21.1.3)
```

```
WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=Non
e)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connection.H
TTPSConnection object at 0x000001D50F9735B0>: Failed to establish a new connection:
[Errno 11001] getaddrinfo failed')': /simple/pip/
WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=Non
e)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connection.H
TTPSConnection object at 0x000001D50F9736A0>: Failed to establish a new connection:
[Errno 11001] getaddrinfo failed')': /simple/pip/
WARNING: Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=Non
e)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connection.H
TTPSConnection object at 0x000001D50F97D1F0>: Failed to establish a new connection:
[Errno 11001] getaddrinfo failed')': /simple/pip/
WARNING: Retrying (Retry(total=0, connect=None, read=None, redirect=None, status=Non
e)) after connection broken by 'NewConnectionError('<pip._vendor.urllib3.connection.H
TTPSConnection object at 0x000001D50F97D0A0>: Failed to establish a new connection:
[Errno 11001] getaddrinfo failed')': /simple/pip/
```

Basic operation in NumPy

In [1]:
```
import numpy as np
```

In [2]:
```
print(dir(np))
```

```
['ALLOW_THREADS', 'AxisError', 'BUFSIZE', 'Bytes0', 'CLIP', 'ComplexWarning', 'DataSo
urce', 'Datetime64', 'ERR_CALL', 'ERR_DEFAULT', 'ERR_IGNORE', 'ERR_LOG', 'ERR_PRINT',
'ERR_RAISE', 'ERR_WARN', 'FLOATING_POINT_SUPPORT', 'FPE_DIVIDEBYZERO', 'FPE_INVALID',
'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'Inf', 'Infinity', 'MAXDIMS', 'MAY_SHARE_B
OUNDS', 'MAY_SHARE_EXACT', 'MachAr', 'ModuleDeprecationWarning', 'NAN', 'NINF', 'NZER
O', 'NaN', 'PINF', 'PZERO', 'RAISE', 'RankWarning', 'SHIFT_DIVIDEBYZERO', 'SHIFT_INVA
LID', 'SHIFT_OVERFLOW', 'SHIFT_UNDERFLOW', 'ScalarType', 'Str0', 'Tester', 'TooHardEr
ror', 'True_', 'UFUNC_BUFSIZE_DEFAULT', 'UFUNC_PYVALS_NAME', 'Uint64', 'VisibleDeprec
ationWarning', 'WRAP', '_NoValue', '_UFUNC_API', '__NUMPY_SETUP__', '__all__', '__bui
ltins__', '__cached__', '__config__', '__deprecated_attrs__', '__dir__', '__doc__',
'__expired_functions__', '__file__', '__getattr__', '__git_version__', '__loader__',
'__mkl_version__', '__name__', '__package__', '__path__', '__spec__', '__version__',
'_add_newdoc_ufunc', '_distributor_init', '_financial_names', '_globals', '_mat', '_p
ytesttester', '_version', 'abs', 'absolute', 'add', 'add_docstring', 'add_newdoc', 'a
dd_newdoc_ufunc', 'alen', 'all', 'allclose', 'alltrue', 'amax', 'amin', 'angle', 'an
y', 'append', 'apply_along_axis', 'apply_over_axes', 'arange', 'arccos', 'arccosh',
'arcsin', 'arcsinh', 'arctan', 'arctan2', 'arctanh', 'argmax', 'argmin', 'argpartitio
n', 'argsort', 'argwhere', 'around', 'array', 'array2string', 'array_equal', 'array_e
quiv', 'array_repr', 'array_split', 'array_str', 'asanyarray', 'asarray', 'asarray_ch
kfinite', 'ascontiguousarray', 'asfarray', 'asfortranarray', 'asmatrix', 'asscalar',
'atleast_1d', 'atleast_2d', 'atleast_3d', 'average', 'bartlett', 'base_repr', 'binary
_repr', 'bincount', 'bitwise_and', 'bitwise_not', 'bitwise_or', 'bitwise_xor', 'black
man', 'block', 'bmat', 'bool8', 'bool_', 'broadcast', 'broadcast_arrays', 'broadcast_
shapes', 'broadcast_to', 'busday_count', 'busday_offset', 'busdaycalendar', 'byte',
'byte_bounds', 'bytes0', 'bytes_', 'c_', 'can_cast', 'cast', 'cbrt', 'cdouble', 'cei
l', 'cfloat', 'char', 'character', 'chararray', 'choose', 'clip', 'clongdouble', 'clo
ngfloat', 'column_stack', 'common_type', 'compare_chararrays', 'compat', 'complex12
8', 'complex64', 'complex_', 'complexfloating', 'compress', 'concatenate', 'conj', 'c
onjugate', 'convolve', 'copy', 'copysign', 'copyto', 'core', 'corrcoef', 'correlate',
'cos', 'cosh', 'count_nonzero', 'cov', 'cross', 'csingle', 'ctypeslib', 'cumprod', 'c
umproduct', 'cumsum', 'datetime64', 'datetime_as_string', 'datetime_data', 'deg2rad',
'degrees', 'delete', 'deprecate', 'deprecate_with_doc', 'diag', 'diag_indices', 'diag
_indices_from', 'diagflat', 'diagonal', 'diff', 'digitize', 'disp', 'divide', 'divmo
d', 'dot', 'double', 'dsplit', 'dstack', 'dtype', 'e', 'ediff1d', 'einsum', 'einsum_p
ath', 'emath', 'empty', 'empty_like', 'equal', 'errstate', 'euler_gamma', 'exp', 'exp
2', 'expand_dims', 'expm1', 'extract', 'eye', 'fabs', 'fastCopyAndTranspose', 'fft',
'fill_diagonal', 'find_common_type', 'finfo', 'fix', 'flatiter', 'flatnonzero', 'flex
ible', 'flip', 'fliplr', 'flipud', 'float16', 'float32', 'float64', 'float_', 'float_
power', 'floating', 'floor', 'floor_divide', 'fmax', 'fmin', 'fmod', 'format_float_po
sitional', 'format_float_scientific', 'format_parser', 'frexp', 'frombuffer', 'fromfi
le', 'fromfunction', 'fromiter', 'frompyfunc', 'fromregex', 'fromstring', 'full', 'fu
ll_like', 'gcd', 'generic', 'genfromtxt', 'geomspace', 'get_array_wrap', 'get_includ
e', 'get_printoptions', 'getbufsize', 'geterr', 'geterrcall', 'geterrobj', 'gradien
t', 'greater', 'greater_equal', 'half', 'hamming', 'hanning', 'heaviside', 'histogra
m', 'histogram2d', 'histogram_bin_edges', 'histogramdd', 'hsplit', 'hstack', 'hypot',
'i0', 'identity', 'iinfo', 'imag', 'in1d', 'index_exp', 'indices', 'inexact', 'inf',
'info', 'infty', 'inner', 'insert', 'int0', 'int16', 'int32', 'int64', 'int8', 'int
_', 'intc', 'integer', 'interp', 'intersect1d', 'intp', 'invert', 'is_busday', 'isclo
se', 'iscomplex', 'iscomplexobj', 'isfinite', 'isfortran', 'isin', 'isinf', 'isnan',
'isnat', 'isneginf', 'isposinf', 'isreal', 'isrealobj', 'isscalar', 'issctype', 'issu
bclass_', 'issubdtype', 'issubsctype', 'iterable', 'ix_', 'kaiser', 'kron', 'lcm', 'l
dexp', 'left_shift', 'less', 'less_equal', 'lexsort', 'lib', 'linalg', 'linspace', 'l
ittle_endian', 'load', 'loads', 'loadtxt', 'log', 'log10', 'log1p', 'log2', 'logaddex
p', 'logaddexp2', 'logical_and', 'logical_not', 'logical_or', 'logical_xor', 'logspac
e', 'longcomplex', 'longdouble', 'longfloat', 'longlong', 'lookfor', 'ma', 'mafromtx
t', 'mask_indices', 'mat', 'math', 'matmul', 'matrix', 'matrixlib', 'max', 'maximum',
'maximum_sctype', 'may_share_memory', 'mean', 'median', 'memmap', 'meshgrid', 'mgri
d', 'min', 'min_scalar_type', 'minimum', 'mintypecode', 'mkl', 'mod', 'modf', 'moveax
is', 'msort', 'multiply', 'nan', 'nan_to_num', 'nanargmax', 'nanargmin', 'nancumpro
d', 'nancumsum', 'nanmax', 'nanmean', 'nanmedian', 'nanmin', 'nanpercentile', 'nanpro
```

```
d', 'nanquantile', 'nanstd', 'nansum', 'nanvar', 'nbytes', 'ndarray', 'ndenumerate',
'ndfromtxt', 'ndim', 'ndindex', 'nditer', 'negative', 'nested_iters', 'newaxis', 'nex
tafter', 'nonzero', 'not_equal', 'numarray', 'number', 'obj2sctype', 'object0', 'obje
ct_', 'ogrid', 'oldnumeric', 'ones', 'ones_like', 'os', 'outer', 'packbits', 'pad',
'partition', 'percentile', 'pi', 'piecewise', 'place', 'poly', 'poly1d', 'polyadd',
'polyder', 'polydiv', 'polyfit', 'polyint', 'polymul', 'polynomial', 'polysub', 'poly
val', 'positive', 'power', 'printoptions', 'prod', 'product', 'promote_types', 'ptp',
'put', 'put_along_axis', 'putmask', 'quantile', 'r_', 'rad2deg', 'radians', 'random',
'ravel', 'ravel_multi_index', 'real', 'real_if_close', 'rec', 'recarray', 'recfromcs
v', 'recfromtxt', 'reciprocal', 'record', 'remainder', 'repeat', 'require', 'reshap
e', 'resize', 'result_type', 'right_shift', 'rint', 'roll', 'rollaxis', 'roots', 'rot
90', 'round', 'round_', 'row_stack', 's_', 'safe_eval', 'save', 'savetxt', 'savez',
'savez_compressed', 'sctype2char', 'sctypeDict', 'sctypes', 'searchsorted', 'select',
'set_numeric_ops', 'set_printoptions', 'set_string_function', 'setbufsize', 'setdiff1
d', 'seterr', 'seterrcall', 'seterrobj', 'setxor1d', 'shape', 'shares_memory', 'shor
t', 'show_config', 'sign', 'signbit', 'signedinteger', 'sin', 'sinc', 'single', 'sing
lecomplex', 'sinh', 'size', 'sometrue', 'sort', 'sort_complex', 'source', 'spacing',
'split', 'sqrt', 'square', 'squeeze', 'stack', 'std', 'str0', 'str_', 'string_', 'sub
tract', 'sum', 'swapaxes', 'sys', 'take', 'take_along_axis', 'tan', 'tanh', 'tensordo
t', 'test', 'testing', 'tile', 'timedelta64', 'trace', 'tracemalloc_domain', 'transpo
se', 'trapz', 'tri', 'tril', 'tril_indices', 'tril_indices_from', 'trim_zeros', 'tri
u', 'triu_indices', 'triu_indices_from', 'true_divide', 'trunc', 'typecodes', 'typena
me', 'ubyte', 'ufunc', 'uint', 'uint0', 'uint16', 'uint32', 'uint64', 'uint8', 'uint
c', 'uintp', 'ulonglong', 'unicode_', 'union1d', 'unique', 'unpackbits', 'unravel_ind
ex', 'unsignedinteger', 'unwrap', 'use_hugepage', 'ushort', 'vander', 'var', 'vdot',
'vectorize', 'version', 'void', 'void0', 'vsplit', 'vstack', 'warnings', 'where', 'wh
o', 'zeros', 'zeros_like']
```

In [3]:
```
print(len(dir(np)))
```

```
608
```

Array

1. Array is collection of homogeneous (similar) data elements stored in contiguous memory location.
2. It is like list data structure in python.
3. It also uses [ ] to represent the data elements.
4. NumPy's array class is called ndarray.

In [4]:
```
# Creation of array
# 1. Using existing sequential data structure
# e.g list,tuple,str,set,dict,range
import numpy as np
import sys
arr1 = np.array([1,2,3,4])
print(arr1)
print(type(arr1))
print(sys.getsizeof(arr1))
list1=[1,232323,3,4,5]
print(list1)
print(type(list1))
print(sys.getsizeof(list1))
```

```
[1 2 3 4]
<class 'numpy.ndarray'>
128
[1, 232323, 3, 4, 5]
<class 'list'>
120
```

In [17]:
```
print(arr1.dtype)  # Data type of each element of the array, 1 byte=8 bit
print(arr1.nbytes)  # Total memory of array in bytes
```

```
int8
4
```

In [11]:
```
import sys
print(sys.getsizeof(list1))
```

```
120
```

In [12]:
```
# 1 D array
import numpy as np
arr1 = np.array([1,4,7,9])
print(arr1)
print(type(arr1))
```

```
[1 4 7 9]
<class 'numpy.ndarray'>
```

Attributes

In [4]:
```
# 1 D array
import numpy as np
arr1 = np.array([1,4,7,9,5,6],np.int32)  #int8,int16,int32,int64
print(arr1)
print(arr1.dtype)   # data type of each element
print(arr1.ndim)    # Number of dimension
print(arr1.itemsize)   # Size in bytes of each element
print(arr1.nbytes)     # size in bytes of whole array
print(arr1.size)       # Number of elements
print(arr1.shape)     # order of array
```

```
[1 4 7 9 5 6]
int32
1
4
24
6
(6,)
```

In [5]:
```
# 2 D array
arr2 = np.array([[1,2,3],[4,5,6]])
print(arr2)
print(arr2.dtype)   # data type of each element
print(arr2.ndim)    # Number of dimension
print(arr2.itemsize)   # Size in bytes of each element
print(arr2.nbytes)     # size in bytes of whole array
print(arr2.size)       # Number of elements
print(arr2.shape)     # order of array
```

```
[[1 2 3]
 [4 5 6]]
int32
2
4
24
6
(2, 3)
```

In [6]:
```
# 3d array
arr3 = np.array([[[1, 2, 3],[4,5,6],[7,8,9]],[[1,0,1],[0,0,1],[1,1,0]]],dtype='int16')
print(arr3)
print(arr3.dtype)   # data type of each element
print(arr3.ndim)    # Number of dimension
print(arr3.itemsize)   # Size in bytes of each element
print(arr3.nbytes)     # size in bytes of whole array
print(arr3.size)       # Number of elements
print(arr3.shape)     # order of array
```

```
[[[1 2 3]
  [4 5 6]
  [7 8 9]]

 [[1 0 1]
  [0 0 1]
  [1 1 0]]]
int16
3
2
36
18
(2, 3, 3)
```

Important attributes of an ndarray object are:

1. ndarray.ndim ---> The number of axes (dimensions) of the array.
2. ndarray.shape ---> The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.
3. ndarray.size ---> The total number of elements of the array. This is equal to the product of the elements of shape.
4. ndarray.dtype ---> An object describing the type of the elements in the array.
5. ndarray.itemsize ---> The size in bytes of each element of the array.
6. ndarray.nbytes ---> The size in bytes of all the elements of the array.

Indexing: Accessing Array elements

In [16]:
```
arr1 = np.array(range(10,21),'int8')    # int8   ----(-128  to 127)
print(arr1)
# Display
```

```
print(arr1[5])
print(arr1[-2])
# Traverse array by index
for i in range(len(arr1)):
    print(arr1[i],end=' ')
print()
# Traverse array by item
for item in arr1:
    print(item,end=' ')
print()
# Sum of all elements of the array
sum = 0
for i in range(len(arr1)):
    sum = sum + arr1[i]
print(sum)
# updation
arr1[4] = 121
print(arr1)
```

```
[10 11 12 13 14 15 16 17 18 19 20]
15
19
10 11 12 13 14 15 16 17 18 19 20
10 11 12 13 14 15 16 17 18 19 20
165
[ 10  11  12  13 121  15  16  17  18  19  20]
```

In [17]:
```
arr2 = np.array([[1,2,3],[4,5,6]],'int8')
print(arr2)
print(arr2[0,1])
```

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

In [18]:
```
# Multiple indices
arr3 = np.array([[[1, 2, 3],[4,5,6],[7,8,9]],[[1,0,1],[0,0,1],[1,1,0]]],dtype='int16')
print(arr3)
print(arr3[1,2,1])    # element in matrix1, row2 and column1
print(arr3[(0,1),(0,1),(0,1)])      # subarray of elements [0,0,0] and [1,1,1]
```

```
[[[1 2 3]
  [4 5 6]
  [7 8 9]]

 [[1 0 1]
  [0 0 1]
  [1 1 0]]]
1
[1 0]
```

slicing

Retrieval of subarray from the original array. slicing by default return view.

In [19]:
```
# Slicing 1D array
import numpy as np
arr1 = np.array([12,34,56,78,3,5,2,7,8,9,11,10,15])
```

```
print(arr1)
print(arr1[3:10])     # start=3, stop=10
print(arr1[3:])       # start=3
print(arr1[:8])       # start=0, stop=8
print(arr1[-9:-2])    # start=-9,stop=-2
print(arr1[2:11:3])    #start=2,stop=11,step=3
print(arr1[10:4:-1])   #start=10,stop=4,step=-1
print(arr1[::-1])     # revrse of array
print(arr1[-4:-8:-2])
```

```
[12 34 56 78  3  5  2  7  8  9 11 10 15]
[78  3  5  2  7  8  9]
[78  3  5  2  7  8  9 11 10 15]
[12 34 56 78  3  5  2  7]
[ 3  5  2  7  8  9 11]
[56  5  8]
[11  9  8  7  2  5]
[15 10 11  9  8  7  2  5  3 78 56 34 12]
[9 7]
```

In [20]:
```
# Slicing 2D array
arr2 = np.array([[4,8,12,1,3],[6,8,11,2,6],[-1,6,7,8,9],[2,7,8,1,7]])
print(arr2)
# 1.matrix of row1 and row2  2.matrix of col2 and col3
print(arr2[1:3,2:4])
print(arr2[-2:-1,-4:-2])
```

```
[[ 4  8 12  1  3]
 [ 6  8 11  2  6]
 [-1  6  7  8  9]
 [ 2  7  8  1  7]]
[[11  2]
 [ 7  8]]
[[6 7]]
```

In [21]:
```
# Slicing in 3D array
arr3 = np.array([[[1,2],[3,4],[5,6]],
                 [[7,8],[9,10],[11,12]],
                 [[1,0],[0,1],[1,1]]])
print(arr3)
print(arr3[1:3,1:3,1:])
```

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

 [[ 7  8]
  [ 9 10]
  [11 12]]

 [[ 1  0]
  [ 0  1]
  [ 1  1]]]
[[[10]
  [12]]

 [[ 1]
  [ 1]]]
```

In [22]:
```
# slice create a view of the original array
arr1 = np.array([12,34,56,78,3,5,2,7,8,9,11,10,15])
```

```
print(arr1)
b = arr1[4:8]
print(b)
b[2]=200
print(b)
print(arr1)
```

```
[12 34 56 78  3  5  2  7  8  9 11 10 15]
[3 5 2 7]
[  3   5 200   7]
[ 12  34  56  78   3   5 200   7   8   9  11  10  15]
```

Creating Arrays in numpy

1.Conversion from other Python structures (i.e. lists and tuples)

In [2]:
```
arr1 = np.array([1,6,3,8],'int8')
print(arr1)
arr2 = np.array([10,26,2])
print(arr2)
arr3 = np.array(range(4,8),'int8')
print(arr3)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [2], in <module>
----> 1 arr1 = np.array([1,6,3,8],'int8')
      2 print(arr1)
      3 arr2 = np.array([10,26,2])

NameError: name 'np' is not defined
```

2. Intrinsic NumPy array creation functions (e.g. arange, ones, zeros, etc.)

In [24]:
```
# Using arange() function
# it creates only 1D array
arr1 = np.arange(7)     # start=0,stop=7
print(arr1)
arr2=np.arange(4,12)    # start=4, stop=12
print(arr2)
arr3 = np.arange(10,30,4)  # start=10,stop=30,step=4
print(arr3)
arr4 = np.arange(10,3,-2)  # start=10,stop=30,step=4
print(arr4)
```

```
[0 1 2 3 4 5 6]
[ 4  5  6  7  8  9 10 11]
[10 14 18 22 26]
[10  8  6  4]
```

In [5]:
```
# Using arange() and reshape() function
import numpy as np
arr1 = np.arange(15)
print(arr1)
arr2 = arr1.reshape((3,5))     #(1,15),(3,5),(5,3),(15,1)
print(arr2)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

In [6]:
```
# Using arange() and reshape() together
arr1 = np.arange(15).reshape((3,5))
print(arr1)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

In [27]:
```
arr1 = np.arange(48).reshape((3,4,4))
print(arr1)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]
  [12 13 14 15]]

 [[16 17 18 19]
  [20 21 22 23]
  [24 25 26 27]
  [28 29 30 31]]

 [[32 33 34 35]
  [36 37 38 39]
  [40 41 42 43]
  [44 45 46 47]]]
```

In [28]:
```
# reshape() of numpy
arr1=np.arange(12)
print(arr1)
arr2 = np.reshape(arr1,(3,4))
print(arr2)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [7]:
```
# Using arange() and resize() function
arr1 = np.arange(12)
print(arr1)
arr1.resize(3,4)
print(arr1)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [8]:
```
# Using arange() and resize() function
arr1 = np.arange(12)
print(arr1)
arr1.resize(3,5)
print(arr1)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11  0  0  0]]
```

In [9]:
```
# Using arange() and resize() function
arr1 = np.arange(12).resize(3,4)
print(arr1)
```

```
None
```

In [10]:
```
# Using arange() and resize() function
arr1 = np.arange(12)
print(arr1)
arr2 = np.resize(arr1,(3,4))
print(arr2)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [33]:
```
# Using arange() and resize() function
arr1 = np.arange(12)
print(arr1)
arr2 = np.resize(arr1,(3,6))
print(arr2)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [ 0  1  2  3  4  5]]
```

In [8]:
```
# Using arange() and resize() function
arr1 = np.arange(12).resize(3,4)
print(arr1)
```

```
None
```

In [35]:
```
# Using zeros() function
# It create array with zero values
a1 = np.zeros((3,5),dtype='int32')
print(a1)
print(a1.dtype)
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
int32
```

In [11]:
```
# Using empty() function
# It creates array with random values
a1 = np.empty((3,5))
print(a1)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

In [17]:
```
# Using ones() function
# It creates array with all values as 1.0
a1 = np.ones((3,5,4))
print(a1)
```

```
[[[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]]
```

In [18]:
```
# Using identity() function
# It creates the identity matrix
a1 = np.identity(5)
print(a1)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

In [21]:
```
# Using eye() function
# It is generalization of identity function, can be applied for any order
a1 = np.eye(3,5)
print(a1)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]]
```

In [22]:
```
# Using linspace() function
a1 = np.linspace(1,5,6)
print(a1)
```

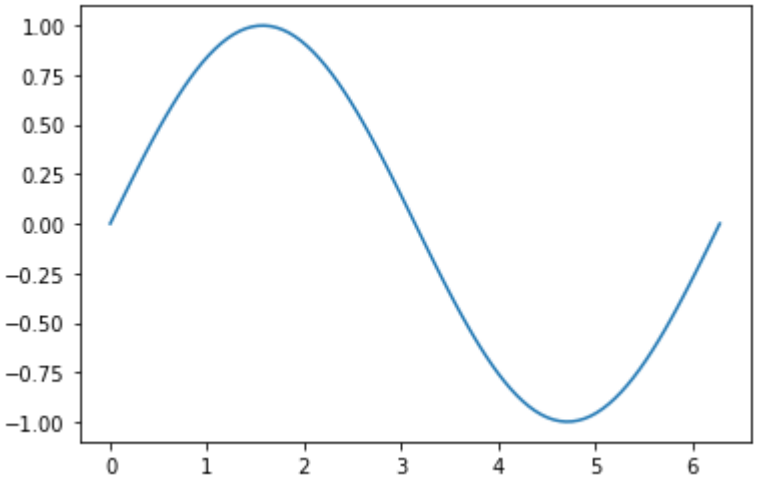
```
[1.  1.8 2.6 3.4 4.2 5. ]
```

In [41]:
```
a = np.linspace(0,2*np.pi,100)
print(a)
b = np.sin(a)
print(b)
```

```
[0.         0.06346652 0.12693304 0.19039955 0.25386607 0.31733259
 0.38079911 0.44426563 0.50773215 0.57119866 0.63466518 0.6981317
 0.76159822 0.82506474 0.88853126 0.95199777 1.01546429 1.07893081
 1.14239733 1.20586385 1.26933037 1.33279688 1.3962634  1.45972992
 1.52319644 1.58666296 1.65012947 1.71359599 1.77706251 1.84052903
 1.90399555 1.96746207 2.03092858 2.0943951  2.15786162 2.22132814
 2.28479466 2.34826118 2.41172769 2.47519421 2.53866073 2.60212725
 2.66559377 2.72906028 2.7925268  2.85599332 2.91945984 2.98292636
 3.04639288 3.10985939 3.17332591 3.23679243 3.30025895 3.36372547
 3.42719199 3.4906585  3.55412502 3.61759154 3.68105806 3.74452458
 3.8079911  3.87145761 3.93492413 3.99839065 4.06185717 4.12532369
 4.1887902  4.25225672 4.31572324 4.37918976 4.44265628 4.5061228
 4.56958931 4.63305583 4.69652235 4.75998887 4.82345539 4.88692191
 4.95038842 5.01385494 5.07732146 5.14078798 5.2042545  5.26772102
 5.33118753 5.39465405 5.45812057 5.52158709 5.58505361 5.64852012
 5.71198664 5.77545316 5.83891968 5.9023862  5.96585272 6.02931923
 6.09278575 6.15625227 6.21971879 6.28318531]
[ 0.00000000e+00  6.34239197e-02  1.26592454e-01  1.89251244e-01
  2.51147987e-01  3.12033446e-01  3.71662456e-01  4.29794912e-01
  4.86196736e-01  5.40640817e-01  5.92907929e-01  6.42787610e-01
  6.90079011e-01  7.34591709e-01  7.76146464e-01  8.14575952e-01
  8.49725430e-01  8.81453363e-01  9.09631995e-01  9.34147860e-01
  9.54902241e-01  9.71811568e-01  9.84807753e-01  9.93838464e-01
  9.98867339e-01  9.99874128e-01  9.96854776e-01  9.89821442e-01
  9.78802446e-01  9.63842159e-01  9.45000819e-01  9.22354294e-01
  8.95993774e-01  8.66025404e-01  8.32569855e-01  7.95761841e-01
  7.55749574e-01  7.12694171e-01  6.66769001e-01  6.18158986e-01
  5.67059864e-01  5.13677392e-01  4.58226522e-01  4.00930535e-01
  3.42020143e-01  2.81732557e-01  2.20310533e-01  1.58001396e-01
  9.50560433e-02  3.17279335e-02 -3.17279335e-02 -9.50560433e-02
 -1.58001396e-01 -2.20310533e-01 -2.81732557e-01 -3.42020143e-01
 -4.00930535e-01 -4.58226522e-01 -5.13677392e-01 -5.67059864e-01
 -6.18158986e-01 -6.66769001e-01 -7.12694171e-01 -7.55749574e-01
 -7.95761841e-01 -8.32569855e-01 -8.66025404e-01 -8.95993774e-01
 -9.22354294e-01 -9.45000819e-01 -9.63842159e-01 -9.78802446e-01
 -9.89821442e-01 -9.96854776e-01 -9.99874128e-01 -9.98867339e-01
 -9.93838464e-01 -9.84807753e-01 -9.71811568e-01 -9.54902241e-01
 -9.34147860e-01 -9.09631995e-01 -8.81453363e-01 -8.49725430e-01
 -8.14575952e-01 -7.76146464e-01 -7.34591709e-01 -6.90079011e-01
 -6.42787610e-01 -5.92907929e-01 -5.40640817e-01 -4.86196736e-01
 -4.29794912e-01 -3.71662456e-01 -3.12033446e-01 -2.51147987e-01
 -1.89251244e-01 -1.26592454e-01 -6.34239197e-02 -2.44929360e-16]
```

In [42]:
```
import matplotlib.pyplot as plt
a = np.linspace(0,2*np.pi,100)
plt.plot(a,np.sin(a))
```

Out[42]:
```
[<matplotlib.lines.Line2D at 0x28edbf8a2e0>]
```



In [10]:
```
# Using empty_like()  function
a1 = np.arange(9)
print(a1)
a2 = np.empty_like(a1)    # a2 is the copy of a1
print(a2)
a2[4]=200
print(a2)
print(a1)
```

```
[0 1 2 3 4 5 6 7 8]
[0 1 2 3 4 5 6 7 8]
[  0   1   2   3 200   5   6   7   8]
[0 1 2 3 4 5 6 7 8]
```

In [44]:
```
# Using ravel() method
# ravel() method creates view of the original array
a1 = np.arange(12).reshape(3,4)
print(a1)
a2 = np.ravel(a1)  # a2 is the view of a1
print(a2)
a2[4]=200
print(a2)
print(a1)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[  0   1   2   3 200   5   6   7   8   9  10  11]
[[  0   1   2   3]
 [200   5   6   7]
 [  8   9  10  11]]
```

In [45]:
```
# Using flatten() method
# flatten() method creates copy of the original array
a1 = np.arange(12).reshape(3,4)
print(a1)
a2= a1.flatten()  # a2 is the copy of a1
print(a2)
a2[4]=200
print(a2)
print(a1)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[  0   1   2   3 200   5   6   7   8   9  10  11]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [46]:
```
# Using flat attribute
a1 = np.arange(12).reshape(3,4)
print(a1)
for item in a1.flat:      # it returns flat iterator
    print(item)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
0
1
2
3
4
5
6
7
8
9
10
11
```

diag()

It can define either a square 2D array with given values along the diagonal or if given a 2D array returns a 1D array that is only the diagonal elements.

In [47]:
```
# using diag() function
a1 = np.diag([4,7,12])
print(a1)
```

```
[[ 4  0  0]
 [ 0  7  0]
 [ 0  0 12]]
```

In [48]:
```
# Passing 2D array inside diag() function
a1 = np.arange(10,19).reshape(3,3)
print(a1)
a2 = np.diag(a1)
print(a2)
a3 = np.diag(a2)
print(a3)
```

```
[[10 11 12]
 [13 14 15]
 [16 17 18]]
[10 14 18]
[[10  0  0]
 [ 0 14  0]
 [ 0  0 18]]
```

In [49]:
```
# Using random() function
rg = np.random.default_rng()
a1 = rg.random((3,4))
print(a1)
a2 = a1*10
print(a2)
```

```
[[0.48004883 0.91095297 0.58959101 0.1832981 ]
 [0.86884071 0.70298133 0.40176307 0.89753044]
 [0.34899557 0.92586963 0.41856204 0.4129483 ]]
[[4.80048828 9.10952968 5.89591006 1.83298105]
 [8.68840705 7.02981333 4.01763073 8.97530439]
 [3.48995568 9.2586963  4.18562044 4.12948298]]
```

In [50]:
```
# Using randint() function
a1 = np.random.randint(2,10,size=(3,5,2))
print(a1)
```

```
[[[7 4]
  [8 7]
  [4 5]
  [8 4]
  [2 8]]

 [[2 5]
  [6 8]
  [6 6]
  [6 9]
  [7 7]]

 [[3 9]
  [6 7]
  [5 8]
  [6 8]
  [3 4]]]
```

3. Replicating, joining existing arrays

In [51]:
```
# Replicating by slicing
# it creates view of the array
a = np.arange(5,17).reshape((3,4))
print(a)
b = a[2:,1:3]      # b is the view of a
print(b)
b[0][1]=200
print(b)
print(a)
```

```
[[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
[[14 15]]
[[ 14 200]]
[[  5   6   7   8]
 [  9  10  11  12]
 [ 13  14 200  16]]
```

In [52]:
```
# Replicating using slicing and copy() method
a = np.arange(5,17).reshape((3,4))
print(a)
b = a[2:,1:3].copy()      # b is the copy of a
print(b)
b[0][1]=200
print(b)
print(a)
```

```
[[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
[[14 15]]
[[ 14 200]]
[[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
```

Joining two arrays

1. np.hstack()
2. np.vstack()
3. np.block()

In [53]:
```
# Using hstack()
a1 = np.arange(6,12).reshape(2,3)
print(a1)
a2 = np.arange(8).reshape(2,4)
print(a2)
a3 = np.hstack([a2,a1])
print(a3)
```

```
[[ 6  7  8]
 [ 9 10 11]]
[[0 1 2 3]
 [4 5 6 7]]
[[ 0  1  2  3  6  7  8]
 [ 4  5  6  7  9 10 11]]
```

In [54]:
```
# Using vstack() function
a1 = np.arange(6,12).reshape(2,3)
print(a1)
a2 = np.arange(12).reshape(4,3)
print(a2)
a3 = np.vstack([a1,a2])
print(a3)
```

```
[[ 6  7  8]
 [ 9 10 11]]
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
[[ 6  7  8]
 [ 9 10 11]
 [ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

In [55]:
```
# Using block() function
a = np.identity(2)
b = np.zeros((2,2))
c = np.diag([7,8])
d = np.eye(2,2)
e = np.block([[a,b],[c,d]])
print(e)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [7. 0. 1. 0.]
 [0. 8. 0. 1.]]
```

In [56]:
```
# split()
a = np.arange(15)
b = np.split(a,5)  # List of subarrays
print(a)
print(b)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8]), array([ 9, 10, 11]), array([1
2, 13, 14])]
```

In [57]:
```
a = np.arange(34,49)
b = np.split(a,[3,5,10,13])  # List of subarrays
print(a)
print(b)
for arr in b:
    print(arr.sum())
```

```
[34 35 36 37 38 39 40 41 42 43 44 45 46 47 48]
[array([34, 35, 36]), array([37, 38]), array([39, 40, 41, 42, 43]), array([44, 45, 4
6]), array([47, 48])]
105
75
205
135
95
```

Iterating array in numpy

In [58]:
```
# Example 1: 2D array iteration
# row-major  or C-style order
a = np.arange(20,35).reshape(3,5)
print(a)
for row in a:
```

```
    for cell in row:
        print(cell,end=' ')
    print()
```

```
[[20 21 22 23 24]
 [25 26 27 28 29]
 [30 31 32 33 34]]
20 21 22 23 24
25 26 27 28 29
30 31 32 33 34
```

In [59]:
```
#Example 2.a: using flatten() function
a = np.arange(20,35).reshape(3,5)
b = a.flatten()
for item in b:
    print(item,end=' ')
```

```
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
```

In [60]:
```
#Example 2.b: using flatten() function
a = np.arange(20,35).reshape(3,5)
for item in a.flatten():     # [20,21,22,...,35]
    print(item,end=' ')
```

```
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
```

In [61]:
```
# Example 3: using flat attribute
a = np.arange(20,35).reshape(3,5)
for item in a.flat:     # <20,21,22,...,35>
    print(item,end=' ')
```

```
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
```

nditer() funtion

It gives the efficient n-dimensional iterator object to iterate over arrays.

In [62]:
```
# Example 4: nditer() function
# display elements in 'C' order i.e. row-major order
a = np.arange(20,35).reshape(3,5)
for item in np.nditer(a,order='C'):
    print(item,end=' ')
```

```
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
```

In [63]:
```
# Example 5: nditer() function
# display elements in 'F'(Fortran style) order i.e. column-major order
print(a)
a = np.arange(20,35).reshape(3,5)
for item in np.nditer(a,order='F'):
    print(item,end=' ')
```

```
[[20 21 22 23 24]
 [25 26 27 28 29]
 [30 31 32 33 34]]
20 25 30 21 26 31 22 27 32 23 28 33 24 29 34
```

external_loop

It causes the values given to be one-dimensional arrays with multiple values instead of zero-dimensional arrays

In [1]:
```
import numpy as np
```

In [4]:
```
# Example 6: nditer() function
# display elements in 'F' order i.e. column-major order
# using option flag as external_loop
a = np.arange(20,35).reshape(3,5)
print(a)
for item in np.nditer(a,order='F',flags=['external_loop']):
    print(item)
```

```
[[20 21 22 23 24]
 [25 26 27 28 29]
 [30 31 32 33 34]]
[20 25 30]
[21 26 31]
[22 27 32]
[23 28 33]
[24 29 34]
```

In [10]:
```
# Example 7: nditer() function
# using op_flags, with option readwrite----allowing to read and write array
# Square each element of the array
a = np.arange(20,35).reshape(3,5)
print(a)
for item in np.nditer(a,order='F',op_flags=['readwrite']):
    item[...] = item*2
print(a)
```

```
[[20 21 22 23 24]
 [25 26 27 28 29]
 [30 31 32 33 34]]
[[40 42 44 46 48]
 [50 52 54 56 58]
 [60 62 64 66 68]]
```

In [9]:
```
a = np.arange(20,35).reshape(3,5)
print(a)
print(a[...,3:])
```

```
[[20 21 22 23 24]
 [25 26 27 28 29]
 [30 31 32 33 34]]
[[23 24]
 [28 29]
 [33 34]]
```

In [14]:
```
# Example 8: Iterating two arrays simultaneously
# Arrays must be broadcastable i.e.
# 1. Both arrays must have same shape or
# 2. One of the array must have dimension==trailing dimension of the other
a1 = np.arange(6).reshape(3,2)
print(a1)
a2 = np.array([100,200])
print(a2)
for x,y in np.nditer([a2,a1]):
    print(x,y)
```

```
[[0 1]
 [2 3]
 [4 5]]
[100 200]
100 0
200 1
100 2
200 3
100 4
200 5
```

Operation in arrays

sum(), max(), min(), sort(), argmax(), argmin(), argsort(), mean(), std(), var()

In [16]:
```
# In 1D array
a1 = np.array([12,56,23,11])
print(a1)
print(a1.sum())
```

```
[12 56 23 11]
102
```

In [17]:
```
a1 = np.array([12,56,23,11])
print(a1)
print(a1.max())
```

```
[12 56 23 11]
56
```

In [18]:
```
a1 = np.array([12,56,23,11])
print(a1)
print(a1.min())
```

```
[12 56 23 11]
11
```

In [19]:
```
a1 = np.array([12,56,23,11])
print(a1)
print(a1.argmax())   # index of maximum value
```

```
[12 56 23 11]
1
```

In [20]:
```
a1 = np.array([12,56,23,11])
print(a1)
print(a1.argmin())
```

```
[12 56 23 11]
3
```

In [21]:
```
a1 = np.array([12,56,23,11])
print(a1)
print(a1.argsort())  #indexing sorting from low to high value
```

```
[12 56 23 11]
[3 0 2 1]
```

In [22]:
```
a1 = np.array([12,56,23,11])
print(a1)
a1.sort()
print(a1)
```

```
[12 56 23 11]
[11 12 23 56]
```

In [24]:
```
a1 = np.array([12,56,23,11])
print(a1)
print(a1.mean())
print(a1.var())
print(a1.std())
```

```
[12 56 23 11]
25.5
332.25
18.227726133558185
```

In [29]:
```
# In 2D array
a1 = np.arange(10,19).reshape(3,3)
print(a1)
print(a1.sum())
print(a1.sum(axis=0))     #sum row
print(a1.sum(axis=1))     #sum column
```

```
[[10 11 12]
 [13 14 15]
 [16 17 18]]
126
[39 42 45]
[33 42 51]
```

In [32]:
```
a1=np.array([[23,56,12,11],[7,3,1,61]])
print(a1)
print(a1.max())
print(a1.max(axis=0))
print(a1.max(axis=1))
```

```
[[23 56 12 11]
 [ 7  3  1 61]]
61
[23 56 12 61]
[56 61]
```

In [33]:
```
a1=np.array([[23,56,12],[7,3,61],[8,4,9]])
print(a1)
print(a1.argmax())
print(a1.argmax(axis=0))
print(a1.argmax(axis=1))
```

```
[[23 56 12]
 [ 7  3 61]
 [ 8  4  9]]
5
[0 0 1]
[1 2 2]
```

In [34]:
```
a1=np.array([[23,56,12],[7,3,61],[8,4,9]])
print(a1)
```

```
a1.sort()
print(a1)
```

```
[[23 56 12]
 [ 7  3 61]
 [ 8  4  9]]
[[12 23 56]
 [ 3  7 61]
 [ 4  8  9]]
```

Arithmetic operation on array

In [69]:
```
# In 1D array
```

In [46]:
```
# In 2D array
a1 = np.array([[1,6,7],[2,5,3],[1,5,8]])
print(a1)
# print(a1 + 5)
# print(a1 - 3)
# print(a1*4)
# print(a1/5)
# print(a1//2)
# print(a1%2)
# print(a1**3)
print(a1**0.5)
```

```
[[1 6 7]
 [2 5 3]
 [1 5 8]]
[[1.         2.44948974 2.64575131]
 [1.41421356 2.23606798 1.73205081]
 [1.         2.23606798 2.82842712]]
```

Matrix operation

add() – add elements of two matrices.

subtract() – subtract elements of two matrices.

divide() – divide elements of two matrices.

multiply() – multiply elements of two matrices.

dot() – It performs matrix multiplication, does not element wise multiplication.

sqrt() – square root of each element of matrix.

sum(x,axis) – add to all the elements in matrix. Second argument is optional, it is used when we want to compute the column sum if axis is 0 and row sum if axis is 1.

"T" – It performs transpose of the specified matrix.

In [48]:
```
# Transpose of the matrix
a1 = np.arange(6).reshape(3,2)
print(a1)
print(a1.T)
```

```
[[0 1]
 [2 3]
 [4 5]]
[[0 2 4]
 [1 3 5]]
```

In [52]:
```
# Addition of two matrices
a1 = np.arange(6).reshape(2,3)
a2 = np.array([[12,6,5],[11,8,23]])
print(a1)
print(a2)
print(a1+a2)       # np.add(a1,a2)
print(np.add(a1,a2))
```

```
[[0 1 2]
 [3 4 5]]
[[12  6  5]
 [11  8 23]]
[[12  7  7]
 [14 12 28]]
[[12  7  7]
 [14 12 28]]
```

In [53]:
```
# Subtraction of two matrices
a1 = np.arange(6).reshape(2,3)
a2 = np.array([[12,6,5],[11,8,23]])
print(a1)
print(a2)
print(a1-a2)
print(np.subtract(a1,a2))
```

```
[[0 1 2]
 [3 4 5]]
[[12  6  5]
 [11  8 23]]
[[-12  -5  -3]
 [ -8  -4 -18]]
[[-12  -5  -3]
 [ -8  -4 -18]]
```

In [54]:
```
# Multiplication of corresponding elements of matrices
a1 = np.arange(6).reshape(2,3)
a2 = np.array([[12,6,5],[11,8,23]])
print(a1)
print(a2)
print(a1*a2)
print(np.multiply(a1,a2))
```

```
[[0 1 2]
 [3 4 5]]
[[12  6  5]
 [11  8 23]]
[[  0   6  10]
 [ 33  32 115]]
[[  0   6  10]
 [ 33  32 115]]
```

In [58]:
```
# Multiplication of two matrices
# Using dot() function
a1 = np.arange(6).reshape(2,3)
a2 = np.array([[1,2],[2,1],[-1,1]])
print(a1)
print(a2)
a3 = np.dot(a1,a2)
print(a3)
```

```
[[0 1 2]
 [3 4 5]]
[[ 1  2]
 [ 2  1]
 [-1  1]]
[[ 0  3]
 [ 6 15]]
```

In [59]:
```
# Multiplication of two matrices
# Using @ operator
a1 = np.arange(6).reshape(2,3)
a2 = np.array([[1,2],[2,1],[-1,1]])
print(a1)
print(a2)
a3 = a1@a2
print(a3)
```

```
[[0 1 2]
 [3 4 5]]
[[ 1  2]
 [ 2  1]
 [-1  1]]
[[ 0  3]
 [ 6 15]]
```

In [62]:
```
# Multiplication of two matrices
# Using matmul() of linalg of linalg
a1 = np.arange(6).reshape(2,3)
a2 = np.array([[1,2],[2,1],[-1,1]])
print(a1)
print(a2)
a3 = np.linalg.linalg.matmul(a1,a2)
print(a3)
```

```
[[0 1 2]
 [3 4 5]]
[[ 1  2]
 [ 2  1]
 [-1  1]]
[[ 0  3]
 [ 6 15]]
```

In [55]:
```
# Division of corresponding elements of the matrices
a1 = np.arange(6).reshape(2,3)
a2 = np.array([[12,6,5],[11,8,23]])
print(a1)
print(a2)
print(a1/a2)
print(np.divide(a1,a2))
```

```
[[0 1 2]
 [3 4 5]]
[[12  6  5]
 [11  8 23]]
[[0.         0.16666667 0.4       ]
 [0.27272727 0.5        0.2173913 ]]
[[0.         0.16666667 0.4       ]
 [0.27272727 0.5        0.2173913 ]]
```

In [56]:
```
# 1/matrix computation
a1 = np.arange(6).reshape(2,3)
print(a1)
print(1/a1)
```

```
[[0 1 2]
 [3 4 5]]
[[       inf 1.         0.5       ]
 [0.33333333 0.25       0.2       ]]
<ipython-input-56-80d77239fe9e>:4: RuntimeWarning: divide by zero encountered in true
_divide
  print(1/a1)
```

In [2]:
```
import numpy as np
```

In [5]:
```
# Determinant of the matrix
a1 = np.array([[2,3],[4,1]])
print(a1)
print(int(np.linalg.det(a1)))
```

```
[[2 3]
 [4 1]]
-10
```

In [7]:
```
# Inverse of the matrix
a1 = np.array([[4,3],[1,1]])
print(a1)
print(np.linalg.inv(a1))
```

```
[[4 3]
 [1 1]]
[[ 1. -3.]
 [-1.  4.]]
```

Some more function

In [9]:
```
# where() function
a1 = np.array([[-2,5,0],[9,0,7],[1,4,2]])
print(a1)
np.where(a1>4)
```

```
[[-2  5  0]
 [ 9  0  7]
 [ 1  4  2]]
```

Out[9]:
```
(array([0, 1, 1], dtype=int64), array([1, 0, 2], dtype=int64))
```

In [10]:
```
# Replacing False with -1 values
a1 = np.array([[-2,5,0],[9,0,7],[1,4,2]])
print(a1)
np.where(a1>4,True,False)
```

```
[[-2  5  0]
 [ 9  0  7]
 [ 1  4  2]]
```

Out[10]:
```
array([[False,  True, False],
       [ True, False,  True],
       [False, False, False]])
```

In [12]:
```
# Replacing False with -1 values
a1 = np.array([[-2,5,0],[9,0,7],[1,4,2]])
print(a1)
b1 = np.where(a1>4,a1,-1)
print(b1)
```

```
[[-2  5  0]
 [ 9  0  7]
 [ 1  4  2]]
[[-1  5 -1]
 [ 9 -1  7]
 [-1 -1 -1]]
```

In [13]:
```
a1 = np.array([[-2,5,0],[9,0,7],[1,4,2]])
print(a1)
b1 = np.where(a1>4,2*a1,a1**3)
print(b1)
```

```
[[-2  5  0]
 [ 9  0  7]
 [ 1  4  2]]
[[-8 10  0]
 [18  0 14]
 [ 1 64  8]]
```

In [14]:
```
a1 = np.array([[-2,5,0],[9,0,7],[1,4,2]])
print(a1)
b1 = np.where(a1>4,a1,np.nan)
print(b1)
```

```
[[-2  5  0]
 [ 9  0  7]
 [ 1  4  2]]
[[nan  5. nan]
 [ 9. nan  7.]
 [nan nan nan]]
```

In [17]:
```
print(a1[a1>2])
```

```
[5 9 7 4]
```

In [18]:
```
# count_nonzero()
a1 = np.array([[-2,5,0],[9,0,7],[1,4,2]])
```

```
print(a1)
np.count_nonzero(a1)
```

```
[[-2  5  0]
 [ 9  0  7]
 [ 1  4  2]]
7
```

Out[18]:

In [19]:
```
# nonzero()
a1 = np.array([[-2,5,0],[9,0,7],[1,4,2]])
print(a1)
np.nonzero(a1)
```

```
[[-2  5  0]
 [ 9  0  7]
 [ 1  4  2]]
(array([0, 0, 1, 1, 2, 2, 2], dtype=int64),
 array([0, 1, 0, 2, 0, 1, 2], dtype=int64))
```

Out[19]:

Data type in numpy

NumPy supports a much greater variety of numerical types than Python does. The following table shows different scalar data types defined in NumPy.

Note :

int8, int16, int32, int64 can be replaced by equivalent string 'i1', 'i2','i4', etc.

Data type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (–128 to 127)
int16	Integer (–32768 to 32767)
int32	Integer (–2147483648 to 2147483647)
int64	Integer (–9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64
float16	Half-precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single-precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double-precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128
complex64	Complex number, represented by two 32-bit floats
complex128	Complex number, represented by two 64-bit floats

In [20]:
```
a = np.array([12,45,2,7],dtype='i2')
a
```

Out[20]:
```
array([12, 45,  2,  7], dtype=int16)
```

Reading and writing data on Files

1. Reading from file: It is the loading of data from the file to numpy array.
2. Writing to file: It is the saving data from numpy array object to file.

In [21]:
```
# Example: Creating 2D array
a1 = np.arange(15,30).reshape(3,5)
print(a1)
```

```
[[15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]]
```

In [22]:
```
# writing in a file
# Using savetxt() function
# It saves an array object to a text file
np.savetxt(r'D:\NIET\EVEN 2020-2021\Advanve Python\PPT\UNIT V\CSE1file.txt',a1)
```

In [27]:
```
# writing in a file
# Using savetxt() function
# It saves an array object to a text file
np.savetxt('CSE1file.txt',a1,fmt='%d',delimiter=',',header="CSE II B",footer='Thank Yo
```

In [36]:
```
# reading the file in numpy format
# loadtxt() function
# It load data from a text file
# Each row in the text file must have the same number of values
b1 = np.loadtxt('CSE1file.txt',delimiter=',',dtype='int32')
print(b1)
```

```
[[15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]]
```

In [34]:
```
b1 = np.loadtxt('CSE1file.txt',delimiter=',',dtype='int32',usecols=(1,2,4))
print(b1)
```

```
[[16 17 19]
 [21 22 24]
 [26 27 29]]
```

In [40]:
```
# tofile() method
# Write array to a file as text or binary(default)
# Data is always written in 'C' order, independent of the order of array.
# The data produced by this method can be recovered using the
# function fromfile()
a = np.array([[1,3,6,8],[7,8,2,1]])
print(a)
a.tofile('CSE2file.txt',sep=',')
```

```
[[1 3 6 8]
 [7 8 2 1]]
```

In [42]:
```
# fromfile() function
# construct an array from data in a text or binary file
b = np.fromfile('CSE2file.txt',sep=',',dtype='int32')
print(b)
```

```
[1 3 6 8 7 8 2 1]
```

In [1]:
```
import numpy as np
```

In [3]:
```
# genfromtxt:for reading the data from file
# Load data from a text file, with missing values handled as specified
a1 = np.array([[13,67,34,12],[56,23,78,90],[11,91,25,62]])
print(a1)
np.savetxt("CSE3file.txt",a1,fmt='%d',delimiter=',')
```

```
[[13 67 34 12]
 [56 23 78 90]
 [11 91 25 62]]
```

In [5]:
```
b = np.genfromtxt('CSE3file.txt',delimiter=',',dtype=np.int32)
print(b)
```

```
[[13 67 34 12]
 [56 23 78 90]
 [11 91 25 62]]
```

In [6]:
```
# Missing value
b = np.genfromtxt('CSE3file.txt',delimiter=',',dtype=np.int32)
print(b)
```

```
[[13 67 34 -1]
 [56 -1 78 90]
 [-1 91 -1 62]]
```

In [7]:
```
# Missing value
b = np.genfromtxt('CSE3file.txt',delimiter=',',dtype=np.int32,filling_values=1000)
print(b)
```

```
[[  13   67   34 1000]
 [  56 1000   78   90]
 [1000   91 1000   62]]
```

In [11]:
```
# Missing value
b = np.genfromtxt('CSE3file.txt',delimiter=',',filling_values=np.inf)
print(b)
```

```
[[13. 67. 34. inf]
 [56. inf 78. 90.]
 [inf 91. inf 62.]]
```

In [92]:
```
# Reading csv file
```

In [12]:
```
c = np.genfromtxt('abc.csv',delimiter=';')
```

In [13]:
```
c
```

Out[13]:
```
array([[ 7.4  ,  0.7  ,  0.   , ...,  0.56 ,  9.4  ,  5.   ],
       [ 7.8  ,  0.88 ,  0.   , ...,  0.68 ,  9.8  ,  5.   ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  0.65 ,  9.8  ,  5.   ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  0.75 , 11.   ,  6.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  0.71 , 10.2  ,  5.   ],
       [ 6.   ,  0.31 ,  0.47 , ...,  0.66 , 11.   ,  6.   ]])
```

In [14]:
```
c.shape
```

Out[14]:
```
(1599, 12)
```

In [ ]:
```
```