NUMPY

What is NumPy?

- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
- NumPy stands for Numerical Python.

NUMPY

NUMPY

There are 6 general mechanisms for creating arrays:

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

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

3. Replicating, joining, or mutating existing arrays

4. Reading arrays from disk, either from standard or custom formats

5. Creating arrays from raw bytes through the use of strings or buffers

6. Use of special library functions (e.g., random)

In [1]:

```
## Version of numpy
import numpy as np
print("Numpy version=",np.__version__)
```

```
C:\Users\This Pc\anaconda3\lib\site-packages\numpy\_distributor_init.py:30: UserWarning:
loaded more than 1 DLL from .libs:
C:\Users\This Pc\anaconda3\lib\site-packages\numpy\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3
OXXGRN2NRFM2.gfortran-win_amd64.dll
C:\Users\This Pc\anaconda3\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKGDG
Q3XBKLKTF43H.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

```
Numpy version= 1.23.5
```

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

In [3]:

```
import numpy as np
l1=[1,2,3]
tup=(4,5,6)
A1=np.array(l1)
A2=np.array(tup)
```

```
print("1st array (A1) = ",A1)
print("2nd array (A2) = ",A2)
```

```
1st array (A1) =  [1 2 3]
2nd array (A2) =  [4 5 6]
```

List Multiplication

In [4]:

```
l1=[1,2,3]
l2=[4,5,6]
print(l1*l2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_23872\3164686912.py in <module>
      1 l1=[1,2,3]
      2 l2=[4,5,6]
----> 3 print(l1*l2)

TypeError: can't multiply sequence by non-int of type 'list'
```

Task 1

In [5]:

```
import numpy as np
l1=[1,2,3]
l2=[4,5,6]
A1=np.array([l1])
A2=np.array([l2])
print("1st array (A1) = ",A1)
print("2nd array (A2) = ",A2)
print("Multiplying both (A1 X A2) = ",A1*A2)
#[[]]multi idimensional because list was passed within a list
```

```
1st array (A1) =  [[1 2 3]]
2nd array (A2) =  [[4 5 6]]
Multiplying both (A1 X A2) =  [[ 4 10 18]]
```

Task 2

In [6]:

```
import numpy as np
l1=[1,2,3]
l2=[4,5,6]
A1=np.array(l1)
A2=np.array(l2)
print("A1 = ",A1)
print("A2 = ",A2)
A=A1*A2
print("A1 x A2 = ",A)
print("> The type of array using type : ",type(A))
print("> The type of array using dtype : ",A.dtype) #no() with dtype bcz it is attribute not a function
print("> The dimension of array : ",A.shape)
```

```
A1 =  [1 2 3]
A2 =  [4 5 6]
A1 x A2 =  [ 4 10 18]
> The type of array using type :  <class 'numpy.ndarray'>
> The type of array using dtype :  int32
> The dimension of array :  (3,)
```

Task 3 (Elements Size Reduction)

In [7]:

```
import numpy as np
t1=(1,2,3)
t2=(4,5,6)
A1=np.array(t1,np.int8)
A2=np.array(t2,np.int8)     #reducing size from 32 to 8
print("A1 = ",A1)
print("A2 = ",A2)
A=A1*A2
print("A1 x A2 = ",A)
print("> The type of array using type : ",type(A))
print("> The type of array using dtype : ",A.dtype)
print("> The size of array using dtype : ",A.size)
print("> The dimension of array : ",A.shape) #dimension of arrays
```

```
A1 =  [1 2 3]
A2 =  [4 5 6]
A1 x A2 =  [ 4 10 18]
> The type of array using type :  <class 'numpy.ndarray'>
> The type of array using dtype :  int8
> The size of array using dtype :  3
> The dimension of array :  (3,)
```

1-D Array or Vector

In [8]:

```
import numpy as np
A=np.array([4,5,6])
print("> The type of array using dtype : ",A.dtype)
print("> The dimension of array : ",A.shape)
print("> The size of array using dtype : ",A.size)
```

```
> The type of array using dtype :  int32
> The dimension of array :  (3,)
> The size of array using dtype :  3
```

In [9]:

```
arr=np.array([1,4,5,6])
print("Original array = ",arr)
arr[1]=10
print("After Replacing",arr)
```

```
Original array =  [1 4 5 6]
After Replacing [ 1 10  5  6]
```

2-D array using lists

In [10]:

```
import numpy as np
l1=[1,2,3]
l2=[4,5,6]
A=np.array((l1,l2))
print("> The type of array using dtype : ",A.dtype)
print("> The dimension of array : ",A.shape)
```

```
> The type of array using dtype :  int32
> The dimension of array :  (2, 3)
```

N-Dimentional Arrays or Martix

In [11]:

```
import numpy as np
r1=[1,2,3,4]
r2=[4,5,6,7]
r3=[1,3,5,7]
r4=[2,4,6,8]
A=np.array((r1,r2,r3,r4))
print("\t4 x 4 Matrix \n",A)
print("> The type of array using dtype : ",A.dtype)
print("> The dimension of array : ",A.shape)
print("> The size of array using dtype : ",A.size)
```

```
 4 x 4 Matrix
 [[1 2 3 4]
 [4 5 6 7]
 [1 3 5 7]
 [2 4 6 8]]
> The type of array using dtype :  int32
> The dimension of array :  (4, 4)
> The size of array using dtype :  16
```

Data Insertion

In [12]:

```
arr=np.array([[1,4,5,6]])
print("Original array = ",arr)
arr[0,1]=10
print("After Replacing",arr)
#rows and columns because its 2d
```

```
Original array =  [[1 4 5 6]]
After Replacing [[ 1 10  5  6]]
```

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

Zeros Function Vector

In [14]:

```
z=np.zeros(100)
print(" zeros Vector \n",z)
print("> The type of array using dtype : ",z.dtype)
print("> The dimension of array : ",z.shape)
print("> The size of array using dtype : ",z.size)
```

```
 zeros Vector
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0.]
> The type of array using dtype :  float64
> The dimension of array :  (100,)
> The size of array using dtype :  100
```

Zeros Matrix

In [15]:

```
z=np.zeros([5,5])
```

```
print("5 x 5 Matrix of zeros \n",z)
print("> The type of array using dtype : ",z.dtype)
print("> The dimension of array : ",z.shape)
print("> The size of array using dtype : ",z.size)
```

```
5 x 5 Matrix of zeros
 [[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
> The type of array using dtype :  float64
> The dimension of array :  (5, 5)
> The size of array using dtype :  25
```

Ones Matrix

In [16]:

```
z=np.ones([5,5])
print("5 x 5 Matrix of ones \n",z)
print("> The type of array using dtype : ",z.dtype)
print("> The dimension of array : ",z.shape)
print("> The size of array using dtype : ",z.size)
```

```
5 x 5 Matrix of ones
 [[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.]]
> The type of array using dtype :  float64
> The dimension of array :  (5, 5)
> The size of array using dtype :  25
```

Arange Function

In [17]:

```
arr=np.arange(1,100)
print("Printing range = \n",arr)
print("> The type of array using dtype : ",arr.dtype)
print("> The dimension of array : ",arr.shape)
print("> The size of array using dtype : ",arr.size)
```

```
Printing range =
 [ 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 48
 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96
 97 98 99]
> The type of array using dtype :  int32
> The dimension of array :  (99,)
> The size of array using dtype :  99
```

Arrange Function with step size

In [18]:

```
arr=np.arange(1,100,10)
print("Printing range = \n",arr)
print("> The type of array using dtype : ",arr.dtype)
print("> The dimension of array : ",arr.shape)
print("> The size of array using dtype : ",arr.size)
```

```
Printing range =
 [ 1 11 21 31 41 51 61 71 81 91]
```

```
> The type of array using dtype :  int32
> The dimension of array :  (10,)
> The size of array using dtype :  10
```

Linespace Function

In [20]:

```
arr=np.linspace(2,3,1000)
print("Printing 100 values between 2 and 3 = \n",arr)
print("> The type of array using dtype : ",arr.dtype)
print("> The dimension of array : ",arr.shape)
print("> The size of array using dtype : ",arr.size)
```

```
Printing 100 values between 2 and 3 =
 [2.         2.001001   2.002002   2.003003   2.004004   2.00500501
 2.00600601 2.00700701 2.00800801 2.00900901 2.01001001 2.01101101
 2.01201201 2.01301301 2.01401401 2.01501502 2.01601602 2.01701702
 2.01801802 2.01901902 2.02002002 2.02102102 2.02202202 2.02302302
 2.02402402 2.02502503 2.02602603 2.02702703 2.02802803 2.02902903
 2.03003003 2.03103103 2.03203203 2.03303303 2.03403403 2.03503504
 2.03603604 2.03703704 2.03803804 2.03903904 2.04004004 2.04104104
 2.04204204 2.04304304 2.04404404 2.04504505 2.04604605 2.04704705
 2.04804805 2.04904905 2.05005005 2.05105105 2.05205205 2.05305305
 2.05405405 2.05505506 2.05605606 2.05705706 2.05805806 2.05905906
 2.06006006 2.06106106 2.06206206 2.06306306 2.06406406 2.06506507
 2.06606607 2.06706707 2.06806807 2.06906907 2.07007007 2.07107107
 2.07207207 2.07307307 2.07407407 2.07507508 2.07607608 2.07707708
 2.07807808 2.07907908 2.08008008 2.08108108 2.08208208 2.08308308
 2.08408408 2.08508509 2.08608609 2.08708709 2.08808809 2.08908909
 2.09009009 2.09109109 2.09209209 2.09309309 2.09409409 2.0950951
 2.0960961  2.0970971  2.0980981  2.0990991  2.1001001  2.1011011
 2.1021021  2.1031031  2.1041041  2.10510511 2.10610611 2.10710711
 2.10810811 2.10910911 2.11011011 2.11111111 2.11211211 2.11311311
 2.11411411 2.11511512 2.11611612 2.11711712 2.11811812 2.11911912
 2.12012012 2.12112112 2.12212212 2.12312312 2.12412412 2.12512513
 2.12612613 2.12712713 2.12812813 2.12912913 2.13013013 2.13113113
 2.13213213 2.13313313 2.13413413 2.13513514 2.13613614 2.13713714
 2.13813814 2.13913914 2.14014014 2.14114114 2.14214214 2.14314314
 2.14414414 2.14514515 2.14614615 2.14714715 2.14814815 2.14914915
 2.15015015 2.15115115 2.15215215 2.15315315 2.15415415 2.15515516
 2.15615616 2.15715716 2.15815816 2.15915916 2.16016016 2.16116116
 2.16216216 2.16316316 2.16416416 2.16516517 2.16616617 2.16716717
 2.16816817 2.16916917 2.17017017 2.17117117 2.17217217 2.17317317
 2.17417417 2.17517518 2.17617618 2.17717718 2.17817818 2.17917918
 2.18018018 2.18118118 2.18218218 2.18318318 2.18418418 2.18518519
 2.18618619 2.18718719 2.18818819 2.18918919 2.19019019 2.19119119
 2.19219219 2.19319319 2.19419419 2.1951952  2.1961962  2.1971972
 2.1981982  2.1991992  2.2002002  2.2012012  2.2022022  2.2032032
 2.2042042  2.20520521 2.20620621 2.20720721 2.20820821 2.20920921
 2.21021021 2.21121121 2.21221221 2.21321321 2.21421421 2.21521522
 2.21621622 2.21721722 2.21821822 2.21921922 2.22022022 2.22122122
 2.22222222 2.22322322 2.22422422 2.22522523 2.22622623 2.22722723
 2.22822823 2.22922923 2.23023023 2.23123123 2.23223223 2.23323323
 2.23423423 2.23523524 2.23623624 2.23723724 2.23823824 2.23923924
 2.24024024 2.24124124 2.24224224 2.24324324 2.24424424 2.24524525
 2.24624625 2.24724725 2.24824825 2.24924925 2.25025025 2.25125125
 2.25225225 2.25325325 2.25425425 2.25525526 2.25625626 2.25725726
 2.25825826 2.25925926 2.26026026 2.26126126 2.26226226 2.26326326
 2.26426426 2.26526527 2.26626627 2.26726727 2.26826827 2.26926927
 2.27027027 2.27127127 2.27227227 2.27327327 2.27427427 2.27527528
 2.27627628 2.27727728 2.27827828 2.27927928 2.28028028 2.28128128
 2.28228228 2.28328328 2.28428428 2.28528529 2.28628629 2.28728729
 2.28828829 2.28928929 2.29029029 2.29129129 2.29229229 2.29329329
 2.29429429 2.2952953  2.2962963  2.2972973  2.2982983  2.2992993
 2.3003003  2.3013013  2.3023023  2.3033033  2.3043043  2.30530531
 2.30630631 2.30730731 2.30830831 2.30930931 2.31031031 2.31131131
 2.31231231 2.31331331 2.31431431 2.31531532 2.31631632 2.31731732
 2.31831832 2.31931932 2.32032032 2.32132132 2.32232232 2.32332332
 2.32432432 2.32532533 2.32632633 2.32732733 2.32832833 2.32932933
```

2.33033033 2.33133133 2.33233233 2.33333333 2.33433433 2.33533534
2.33633634 2.33733734 2.33833834 2.33933934 2.34034034 2.34134134
2.34234234 2.34334334 2.34434434 2.34534535 2.34634635 2.34734735
2.34834835 2.34934935 2.35035035 2.35135135 2.35235235 2.35335335
2.35435435 2.35535536 2.35635636 2.35735736 2.35835836 2.35935936
2.36036036 2.36136136 2.36236236 2.36336336 2.36436436 2.36536537
2.36636637 2.36736737 2.36836837 2.36936937 2.37037037 2.37137137
2.37237237 2.37337337 2.37437437 2.37537538 2.37637638 2.37737738
2.37837838 2.37937938 2.38038038 2.38138138 2.38238238 2.38338338
2.38438438 2.38538539 2.38638639 2.38738739 2.38838839 2.38938939
2.39039039 2.39139139 2.39239239 2.39339339 2.39439439 2.3953954
2.3963964 2.3973974 2.3983984 2.3993994 2.4004004 2.4014014
2.4024024 2.4034034 2.4044044 2.40540541 2.40640641 2.40740741
2.40840841 2.40940941 2.41041041 2.41141141 2.41241241 2.41341341
2.41441441 2.41541542 2.41641642 2.41741742 2.41841842 2.41941942
2.42042042 2.42142142 2.42242242 2.42342342 2.42442442 2.42542543
2.42642643 2.42742743 2.42842843 2.42942943 2.43043043 2.43143143
2.43243243 2.43343343 2.43443443 2.43543544 2.43643644 2.43743744
2.43843844 2.43943944 2.44044044 2.44144144 2.44244244 2.44344344
2.44444444 2.44544545 2.44644645 2.44744745 2.44844845 2.44944945
2.45045045 2.45145145 2.45245245 2.45345345 2.45445445 2.45545546
2.45645646 2.45745746 2.45845846 2.45945946 2.46046046 2.46146146
2.46246246 2.46346346 2.46446446 2.46546547 2.46646647 2.46746747
2.46846847 2.46946947 2.47047047 2.47147147 2.47247247 2.47347347
2.47447447 2.47547548 2.47647648 2.47747748 2.47847848 2.47947948
2.48048048 2.48148148 2.48248248 2.48348348 2.48448448 2.48548549
2.48648649 2.48748749 2.48848849 2.48948949 2.49049049 2.49149149
2.49249249 2.49349349 2.49449449 2.4954955 2.4964965 2.4974975
2.4984985 2.4994995 2.5005005 2.5015015 2.5025025 2.5035035
2.5045045 2.50550551 2.50650651 2.50750751 2.50850851 2.50950951
2.51051051 2.51151151 2.51251251 2.51351351 2.51451451 2.51551552
2.51651652 2.51751752 2.51851852 2.51951952 2.52052052 2.52152152
2.52252252 2.52352352 2.52452452 2.52552553 2.52652653 2.52752753
2.52852853 2.52952953 2.53053053 2.53153153 2.53253253 2.53353353
2.53453453 2.53553554 2.53653654 2.53753754 2.53853854 2.53953954
2.54054054 2.54154154 2.54254254 2.54354354 2.54454454 2.54554555
2.54654655 2.54754755 2.54854855 2.54954955 2.55055055 2.55155155
2.55255255 2.55355355 2.55455455 2.55555556 2.55655656 2.55755756
2.55855856 2.55955956 2.56056056 2.56156156 2.56256256 2.56356356
2.56456456 2.56556557 2.56656657 2.56756757 2.56856857 2.56956957
2.57057057 2.57157157 2.57257257 2.57357357 2.57457457 2.57557558
2.57657658 2.57757758 2.57857858 2.57957958 2.58058058 2.58158158
2.58258258 2.58358358 2.58458458 2.58558559 2.58658659 2.58758759
2.58858859 2.58958959 2.59059059 2.59159159 2.59259259 2.59359359
2.59459459 2.5955956 2.5965966 2.5975976 2.5985986 2.5995996
2.6006006 2.6016016 2.6026026 2.6036036 2.6046046 2.60560561
2.60660661 2.60760761 2.60860861 2.60960961 2.61061061 2.61161161
2.61261261 2.61361361 2.61461461 2.61561562 2.61661662 2.61761762
2.61861862 2.61961962 2.62062062 2.62162162 2.62262262 2.62362362
2.62462462 2.62562563 2.62662663 2.62762763 2.62862863 2.62962963
2.63063063 2.63163163 2.63263263 2.63363363 2.63463463 2.63563564
2.63663664 2.63763764 2.63863864 2.63963964 2.64064064 2.64164164
2.64264264 2.64364364 2.64464464 2.64564565 2.64664665 2.64764765
2.64864865 2.64964965 2.65065065 2.65165165 2.65265265 2.65365365
2.65465465 2.65565566 2.65665666 2.65765766 2.65865866 2.65965966
2.66066066 2.66166166 2.66266266 2.66366366 2.66466466 2.66566567
2.66666667 2.66766767 2.66866867 2.66966967 2.67067067 2.67167167
2.67267267 2.67367367 2.67467467 2.67567568 2.67667668 2.67767768
2.67867868 2.67967968 2.68068068 2.68168168 2.68268268 2.68368368
2.68468468 2.68568569 2.68668669 2.68768769 2.68868869 2.68968969
2.69069069 2.69169169 2.69269269 2.69369369 2.69469469 2.6956957
2.6966967 2.6976977 2.6986987 2.6996997 2.7007007 2.7017017
2.7027027 2.7037037 2.7047047 2.70570571 2.70670671 2.70770771
2.70870871 2.70970971 2.71071071 2.71171171 2.71271271 2.71371371
2.71471471 2.71571572 2.71671672 2.71771772 2.71871872 2.71971972
2.72072072 2.72172172 2.72272272 2.72372372 2.72472472 2.72572573
2.72672673 2.72772773 2.72872873 2.72972973 2.73073073 2.73173173
2.73273273 2.73373373 2.73473473 2.73573574 2.73673674 2.73773774
2.73873874 2.73973974 2.74074074 2.74174174 2.74274274 2.74374374
2.74474474 2.74574575 2.74674675 2.74774775 2.74874875 2.74974975
2.75075075 2.75175175 2.75275275 2.75375375 2.75475475 2.75575576
2.75675676 2.75775776 2.75875876 2.75975976 2.76076076 2.76176176

```
 2.76276276 2.76376376 2.76476476 2.76576577 2.76676677 2.76776777
 2.76876877 2.76976977 2.77077077 2.77177177 2.77277277 2.77377377
 2.77477477 2.77577578 2.77677678 2.77777778 2.77877878 2.77977978
 2.78078078 2.78178178 2.78278278 2.78378378 2.78478478 2.78578579
 2.78678679 2.78778779 2.78878879 2.78978979 2.79079079 2.79179179
 2.79279279 2.79379379 2.79479479 2.7957958  2.7967968  2.7977978
 2.7987988  2.7997998  2.8008008  2.8018018  2.8028028  2.8038038
 2.8048048  2.80580581 2.80680681 2.80780781 2.80880881 2.80980981
 2.81081081 2.81181181 2.81281281 2.81381381 2.81481481 2.81581582
 2.81681682 2.81781782 2.81881882 2.81981982 2.82082082 2.82182182
 2.82282282 2.82382382 2.82482482 2.82582583 2.82682683 2.82782783
 2.82882883 2.82982983 2.83083083 2.83183183 2.83283283 2.83383383
 2.83483483 2.83583584 2.83683684 2.83783784 2.83883884 2.83983984
 2.84084084 2.84184184 2.84284284 2.84384384 2.84484484 2.84584585
 2.84684685 2.84784785 2.84884885 2.84984985 2.85085085 2.85185185
 2.85285285 2.85385385 2.85485485 2.85585586 2.85685686 2.85785786
 2.85885886 2.85985986 2.86086086 2.86186186 2.86286286 2.86386386
 2.86486486 2.86586587 2.86686687 2.86786787 2.86886887 2.86986987
 2.87087087 2.87187187 2.87287287 2.87387387 2.87487487 2.87587588
 2.87687688 2.87787788 2.87887888 2.87987988 2.88088088 2.88188188
 2.88288288 2.88388388 2.88488488 2.88588589 2.88688689 2.88788789
 2.88888889 2.88988989 2.89089089 2.89189189 2.89289289 2.89389389
 2.89489489 2.8958959  2.8968969  2.8978979  2.8988989  2.8998999
 2.9009009  2.9019019  2.9029029  2.9039039  2.9049049  2.90590591
 2.90690691 2.90790791 2.90890891 2.90990991 2.91091091 2.91191191
 2.91291291 2.91391391 2.91491491 2.91591592 2.91691692 2.91791792
 2.91891892 2.91991992 2.92092092 2.92192192 2.92292292 2.92392392
 2.92492492 2.92592593 2.92692693 2.92792793 2.92892893 2.92992993
 2.93093093 2.93193193 2.93293293 2.93393393 2.93493493 2.93593594
 2.93693694 2.93793794 2.93893894 2.93993994 2.94094094 2.94194194
 2.94294294 2.94394394 2.94494494 2.94594595 2.94694695 2.94794795
 2.94894895 2.94994995 2.95095095 2.95195195 2.95295295 2.95395395
 2.95495495 2.95595596 2.95695696 2.95795796 2.95895896 2.95995996
 2.96096096 2.96196196 2.96296296 2.96396396 2.96496496 2.96596597
 2.96696697 2.96796797 2.96896897 2.96996997 2.97097097 2.97197197
 2.97297297 2.97397397 2.97497497 2.97597598 2.97697698 2.97797798
 2.97897898 2.97997998 2.98098098 2.98198198 2.98298298 2.98398398
 2.98498498 2.98598599 2.98698699 2.98798799 2.98898899 2.98998999
 2.99099099 2.99199199 2.99299299 2.99399399 2.99499499 2.995996
 2.996997   2.997998   2.998999   3.        ]
> The type of array using dtype :  float64
> The dimension of array :  (1000,)
> The size of array using dtype :  1000
```

Identity Function

In [21]:

```
i=np.identity(5)
print("Identity Matric =\n",i)
print("> The type of array using dtype : ",i.dtype)
print("> The dimension of array : ",i.shape)
print("> The size of array using dtype : ",i.size)
```

```
Identity Matric =
 [[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.]]
> The type of array using dtype :  float64
> The dimension of array :  (5, 5)
> The size of array using dtype :  25
```

with sum function

In [22]:

```
print("> The sum of identity matrix is  : ",np.sum(i))
print("> The sum of identity matrix is  : ",i.sum())
```

```
> The sum of identity matrix is  :  5.0
> The sum of identity matrix is  :  5.0
```

Row and Column wise sum

In [24]:

```
r1=[4,1,3,2]
r2=[1,0,0,1]
r3=[2,4,3,0]
r4=[4,1,1,1]
arr=np.array([r1,r2,r3,r4])
print(arr)
#print("sum of row ",np.sum(arr[1,:]))
print("sum of row ",arr.sum(axis=0))                                    #if axis=0
column wise sum
print("> The type of array using dtype : ",arr.dtype)
print("> The dimension of array : ",arr.shape)
print("> The size of array using dtype : ",arr.size)
```

```
[[4 1 3 2]
 [1 0 0 1]
 [2 4 3 0]
 [4 1 1 1]]
sum of row  [11  6  7  4]
> The type of array using dtype :  int32
> The dimension of array :  (4, 4)
> The size of array using dtype :  16
```

Min ,Max (Row wise, Column wise)

In [25]:

```
print("Maximum value in row = ",arr.max(axis=1))
print("Maximum value in column = ",arr.max(axis=0))
```

```
Maximum value in row =  [4 1 4 4]
Maximum value in column =  [4 4 3 2]
```

Transpose

In [26]:

```
Atrans=arr.T
print("Transpose of the given matrix is = \n",Atrans)
```

```
Transpose of the given matrix is =
 [[4 1 2 4]
 [1 0 4 1]
 [3 0 3 1]
 [2 1 0 1]]
```

Matix Flatten

In [27]:

```
Aflat=Atrans.flat
print("Flat matrix = ",Aflat)
```

```
Flat matrix =  <numpy.flatiter object at 0x0000024F5CDFB430>
```

Matrix Reshaping

In [30]:

```
print("Actual Matrix=",arr)
print("Reshaping to 8 x 2 = \n",arr.reshape(16,1)) #when reshaping number of elements sh
ould be equal
```

```
Actual Matrix= [[4 1 3 2]
 [1 0 0 1]
 [2 4 3 0]
 [4 1 1 1]]
Reshaping to 8 x 2 =
 [[4]
 [1]
 [3]
 [2]
 [1]
 [0]
 [0]
 [1]
 [2]
 [4]
 [3]
 [0]
 [4]
 [1]
 [1]
 [1]]
```

Ravel

In [31]:

```
a=arr.reshape(8,2)
b=a.ravel()     #convert in single vector
print("Ravel = ",b)
```

```
Ravel =  [4 1 3 2 1 0 0 1 2 4 3 0 4 1 1 1]
```

Argmax, Argmin, Argsort

In [32]:

```
a=[1,16,31,4]
arr=np.array(a)
print(arr)
print("Index of maximum value = ",arr.argmax())  #arg provide indexes
print("Index of mainimum value = ",arr.argmin())
print("Sorted indexes = ",arr.argsort())
```

```
[ 1 16 31  4]
Index of maximum value =  2
Index of mainimum value =  0
Sorted indexes =  [0 3 1 2]
```

Data Stacking

In [ ]:

```
f1=np.full((2,2),5)
print("\nf1 = \n",f1)
f2=np.full((2,2),[[2,91],[4,86]])
print("\nf2 = \n",f2)
a=np.vstack([f1,f2])
```

```
print("\nvstack = \n",a)
b=np.hstack([f1,f2])
print("\nhstack = \n",b)
c=np.column_stack([f1,f2])
print("\ncolumnstack = \n",c)
```

Dot Multiplication with Full function

In [ ]:

```
import numpy as np
f1=np.full((2,2),5)
print("\nf1 = \n",f1)
f2=np.full((2,2),[[2,91],[4,86]])
print("\nf2 = \n",f2)
print("point to point multiplication = ",f1*f2)
print("point to point multiplication = ",np.dot(f1,f2))
```

Reading arrays from disk, either from standard or custom formats

In [ ]:

```
np.save('untitled.npy',a)
```

In [ ]:

```
np.load('untitled.npy')
```

Loading CSV Data Files

In [ ]:

```
csv=np.loadtxt("C:\\Users\\This Pc\\Downloads\\data_csv.csv", delimiter = ',', skiprows =
1)
csv
```

In [ ]:

```
from PIL import Image
import numpy as np

# Open the image
img = Image.open("C:\\Users\\This Pc\\Downloads\\images\\person1.jpg")

# Convert the image to a numpy array
img_array = np.array(img)

# Print the shape of the array
print(img_array.shape)

# Show the array
print(img_array)
```

Array from Raw bytes

In [ ]:

```
import numpy as np

# Create a raw bytes object
raw_bytes = b'\x01\x02\x03\x04\x05'
```

```
# Convert the raw bytes to a NumPy array
arr = np.frombuffer(raw_bytes, dtype=np.uint8)

# Print the array
print(arr)
```

Memory Allocation

In [ ]:

```
a=10
b=10
c=5
print("a=",a)
print("b=",b)
print("c=",c)
print("Memory Location of a=",id(a))
print("Memory Location of b=",id(b))
print("Memory Location of c=",id(c))
```

In [ ]:

```

```

In [ ]:

```

```