In [1]:
```python
from IPython.display import Image
Image(filename='logo.png', height=340, width=900)
```

Out[1]:

INMOVIDU
TRANSFORM YOURSELF

BY ANKUR KHANNA

Numpy

- The numpy library is one of the **`core packages`** in Python's scientific software stack.
- Many other Python data analysis libraries require **`numpy as a prerequisite`**, because they use its ndarray data structure as a building block.
- The Anaconda Python distribution we installed in part 1 comes with numpy.
- Numpy implements a data structure called the **`N-dimensional array or ndarray`**.
- ndarrays are similar to lists in that they contain a **`collection of items that can be accessed via indexes`**.
- On the other hand, **`ndarrays are homogeneous`**, meaning they can only contain objects of the same type and they can be multi-dimensional, **`making it easy to store 2-dimensional tables or matrices.`**
- To work with ndarrays, we need to **`load the numpy library`**.

- It is standard practice to load numpy with the **`alias "np"`** like so:

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

Numpy Arrays are of 2 types:

1. One-Dimensional Vectors

2. Two-Dimensional Matrices

1. Creation of Arrays

1.1 Creation of One-Dimensional Array

We can create an array using other python objects

```
In [3]: mylist=[1,2,3,4,5]                  #Creating a list with 5 elements
```

```
In [4]: mylist                              #Displaying the output
```

```
Out[4]: [1, 2, 3, 4, 5]
```

```
In [5]: array1 = np.array(mylist)           #Passing the list through np.array to co
        nvert the list into an array
```

```
In [6]: array1                              #Displaying the output
```

```
Out[6]: array([1, 2, 3, 4, 5])
```

- The above is a **`one-dimensional array`**
- The **`no. of SQUARE[] brackets`** in the beginning or are an indication of the dimension, in the above case ONE

- Alternatively, the following can also be used:

Dimension

In [7]:
```
array1.ndim
print("Dimension of array is : ",array1.ndim)
```

```
Dimension of array is :  1
```

Other Attributes of Array

a. Shape

In [8]:
```
array1.shape
print("Shape of array is :", array1.shape)
```

```
Shape of array is : (5,)
```

b. Size

In [9]:
```
array1.size
print("Size of array is: ",array1.size)
```

```
Size of array is:  5
```

c. Dtype

In [10]:
```
array1.dtype
print("Data Type of array is: ",array1.dtype)
```

```
Data Type of array is:  int32
```

1.2 Creation of Two-Dimensional Array

```
In [11]: mylist2=[[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15]]   # Creating a ne
         sted list
```

```
In [12]: array2 = np.array(mylist2)                            #Passing the list through n
         p.array to convert the list into an array
```

```
In [13]: array2
```

```
Out[13]: array([[ 1,  2,  3,  4,  5],
                [ 6,  7,  8,  9, 10],
                [11, 12, 13, 14, 15]])
```

- The above is a **`two-dimensional array`**
- The **`no. of SQUARE[] brackets`** in the beginning or are an indication of the dimension, in the above case TWO
- Alternatively, the following can also be used:

```
In [14]: print("Dimension of array is: ",array2.ndim)
         print("Shape of array is: ",array2.shape)
         print("Size of array is: ",array2.size)
         print("Data Type of array is: ",array2.dtype)
```

```
Dimension of array is:  2
Shape of array is:  (3, 5)
Size of array is:  15
Data Type of array is:  int32
```

1.3. Creation of Arrays using ARANGE

```
In [15]: np.arange(0,10)      # Creates a one-dimensional array starting 0 and en
         ding 9
                              # The syntax (start number, end number, step/increm
         ent)
                              # While the start number is inclusive, the end numb
         er is exclusive
```

```
Out[15]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [16]: np.arange(0,11)
```

```
Out[16]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [17]: # Creating an array of even numbers between 0 to 10
         print("Even number Array", np.arange(0,10,2))  # With Stop Value as 10
         print("Even number Array", np.arange(0,12,2))  # With Stop Value as 12
```

```
Even number Array [0 2 4 6 8]
Even number Array [ 0  2  4  6  8 10]
```

1.4. Creation of Special Arrays

1.4.1 Zeros

```
In [18]: np.zeros(5)                  # Creates a One-Dimensional Array with 5 elem
         ents as "0"
```

```
Out[18]: array([0., 0., 0., 0., 0.])
```

```
In [19]: np.zeros(shape=(4,5))        # Creates a Two-Dimensional Array with 4 Ro
         ws & 5 Columns as 20 elements as "0"
                                      # 4 represents the number of Rows
                                      # 5 represents the number of Columns
```

```
Out[19]: array([[0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.]])
```

1.4.2 Ones

```
In [20]: np.ones(5)                    # Creates a One-Dimensional Array with 5 eleme
         nts as "1"

Out[20]: array([1., 1., 1., 1., 1.])
```

```
In [21]: np.ones(shape=(5,3))           # Creates a Two-Dimensional Array with 5 Ro
         ws & 3 Columns as 15 elements as "1"
                                        # 5 represents the number of Rows
                                        # 3 represents the number of Columns

Out[21]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])
```

1.4.3 Identity Matrix

```
In [22]: np.identity(3)                # np.identity() to create a square 2d array
          with 1's across the diagonal

Out[22]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])
```

1.5 Creation of Arrays using LINSPACE

```
In [23]: np.linspace(0,5,10)            # Creates a one-dimensional array between 0
          and 5
                                        # The third argument - 10, represents the n
         umber of elements in the array
                                        # All elements are evenly spaced between st
         art and stop values

Out[23]: array([0.        , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
                2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.        ])
```

```
In [24]: np.linspace(0,5,100)        # Creates a one-dimensional array between 0
          and 5, look at number of brackets
                                     # The third argument - 100, represents the
          number of elements in the array
                                     # All elements are evenly spaced between st
         art and stop values
```

```
Out[24]: array([0.        , 0.05050505, 0.1010101 , 0.15151515, 0.2020202 ,
                0.25252525, 0.3030303 , 0.35353535, 0.4040404 , 0.45454545,
                0.50505051, 0.55555556, 0.60606061, 0.65656566, 0.70707071,
                0.75757576, 0.80808081, 0.85858586, 0.90909091, 0.95959596,
                1.01010101, 1.06060606, 1.11111111, 1.16161616, 1.21212121,
                1.26262626, 1.31313131, 1.36363636, 1.41414141, 1.46464646,
                1.51515152, 1.56565657, 1.61616162, 1.66666667, 1.71717172,
                1.76767677, 1.81818182, 1.86868687, 1.91919192, 1.96969697,
                2.02020202, 2.07070707, 2.12121212, 2.17171717, 2.22222222,
                2.27272727, 2.32323232, 2.37373737, 2.42424242, 2.47474747,
                2.52525253, 2.57575758, 2.62626263, 2.67676768, 2.72727273,
                2.77777778, 2.82828283, 2.87878788, 2.92929293, 2.97979798,
                3.03030303, 3.08080808, 3.13131313, 3.18181818, 3.23232323,
                3.28282828, 3.33333333, 3.38383838, 3.43434343, 3.48484848,
                3.53535354, 3.58585859, 3.63636364, 3.68686869, 3.73737374,
                3.78787879, 3.83838384, 3.88888889, 3.93939394, 3.98989899,
                4.04040404, 4.09090909, 4.14141414, 4.19191919, 4.24242424,
                4.29292929, 4.34343434, 4.39393939, 4.44444444, 4.49494949,
                4.54545455, 4.5959596 , 4.64646465, 4.6969697 , 4.74747475,
                4.7979798 , 4.84848485, 4.8989899 , 4.94949495, 5.        ])
```

1.6 Creation of Arrays using RANDOM

1.6.1 RANDOM RAND

```
In [25]: np.random.rand(5)                   # Creates a one-dimensional random arra
         y with values between 0 & 1
```

```
Out[25]: array([0.91600622, 0.66138727, 0.96646542, 0.22340354, 0.66486408])
```

```
In [26]: np.random.rand(4,5)         # Creates a two-dimensional random arra
         y with values between 0 & 1
```

```
Out[26]: array([[0.63969948, 0.88608838, 0.60224188, 0.25929146, 0.5323667 ],
                [0.02451886, 0.04806498, 0.65855349, 0.21658371, 0.53991066],
                [0.3067957 , 0.70308708, 0.63025934, 0.14296741, 0.89925478],
                [0.47128262, 0.56711379, 0.51260248, 0.9972896 , 0.79416372]])
```

1.6.2 RANDOM RANDINT

```
In [27]: np.random.randint(low=1, high=10, size=6)     # Creates a one-dimension
         al random array with values between 1 & 10
```

```
Out[27]: array([4, 7, 4, 5, 8, 1])
```

```
In [28]: np.random.randint(low=1, high=10, size=(3,4)) # Creates a two-dimension
         al random (3,4) array with values between 1 & 10
```

```
Out[28]: array([[6, 6, 7, 3],
                [7, 7, 9, 2],
                [9, 6, 5, 7]])
```

1.6.3 RANDOM NORMALLY DISTRIBUTED

```
In [29]: np.random.randn(5)                    # Creates a one-dimensional random a
         rray with values under Standard Normal Distribution
```

```
Out[29]: array([-1.54525249,  0.71829321,  0.27752956,  0.04182163, -1.2402839
         7])
```

```
In [30]: np.random.randn(3,4)                  # Creates a two-dimensional random a
         rray with values under Standard Normal Distribution
```

```
Out[30]: array([[-0.41926712,  1.65392538, -0.40907923, -0.07697937],
                [-0.86649234,  1.50965255,  2.63532372, -0.4012805 ],
                [ 0.2145969 ,  0.75710201, -0.84377477, -2.09070725]])
```

2. Array Operations

2.1 Reshape

In [31]:
```
array3 = np.arange(20)
array3
```

Out[31]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16,
       17, 18, 19])
```

In [32]:
```
array3.reshape(5,4)                # Converted one-dimensional array into
 2d array using reshape
```

Out[32]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])
```

2.2 Max & Min / ArgMax & ArgMin

In [33]:
```
randomarr = np.random.randint(2, 100, 10)
randomarr
```

Out[33]:
```
array([16, 36, 50,  2, 68, 12, 22, 72, 89, 26])
```

In [34]:
```
randomarr.min()         # Method to find the min value in an array
```

Out[34]:
```
2
```

In [35]:
```
randomarr.max()         # Method to find the max value in an array
```

Out[35]:
```
89
```

```
In [36]: randomarr.argmin()       # Method to find the index where the min value
          in an array is placed
```

```
Out[36]: 3
```

```
In [37]: randomarr.argmax()       # Method to find the index where the max value
          in an array is placed
```

```
Out[37]: 8
```

2.3 Mathematical Operations with Arrays

```
In [38]: array_1=np.arange(0,20)
```

```
In [39]: array_1
```

```
Out[39]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
         16,
                17, 18, 19])
```

```
In [40]: array_2=np.arange(100,120)
```

```
In [41]: array_2
```

```
Out[41]: array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
                113, 114, 115, 116, 117, 118, 119])
```

a. Addition

```
In [42]: array_1 + array_2
```

```
Out[42]: array([100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124,
                126, 128, 130, 132, 134, 136, 138])
```

b. Subtraction

```
In [43]: array_1 - array_2
```

```
Out[43]: array([-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
0,
                -100, -100, -100, -100, -100, -100, -100, -100, -100])
```

c. Multiplication

```
In [44]: array_1 * array_2
```

```
Out[44]: array([   0,  101,  204,  309,  416,  525,  636,  749,  864,  981, 110
0,
                1221, 1344, 1469, 1596, 1725, 1856, 1989, 2124, 2261])
```

d. Division

```
In [45]: array_1 / array_2
```

```
Out[45]: array([0.        , 0.00990099, 0.01960784, 0.02912621, 0.03846154,
                0.04761905, 0.05660377, 0.06542056, 0.07407407, 0.08256881,
                0.09090909, 0.0990991 , 0.10714286, 0.11504425, 0.12280702,
                0.13043478, 0.13793103, 0.14529915, 0.15254237, 0.15966387])
```

e. Using Scalar Values

```
In [46]: array_1 + 10
```

```
Out[46]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
26,
                27, 28, 29])
```

```
In [47]: array_1 * 10
```

```
Out[47]: array([  0,  10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120,
       130, 140, 150, 160, 170, 180, 190])
```

2.4 Other Operations

a. Square Root

```
In [48]: np.sqrt(array_1)
```

```
Out[48]: array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
                2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ,
                3.16227766, 3.31662479, 3.46410162, 3.60555128, 3.74165739,
                3.87298335, 4.        , 4.12310563, 4.24264069, 4.35889894])
```

b. logs

```
In [49]: np.log(array_1)
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Run
timeWarning: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```

```
Out[49]: array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
                1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458,
                2.30258509, 2.39789527, 2.48490665, 2.56494936, 2.63905733,
                2.7080502 , 2.77258872, 2.83321334, 2.89037176, 2.94443898])
```

c. Trignometric functions

```
In [50]: np.sin(array_1)
```

```
Out[50]: array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
                -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849,
                -0.54402111, -0.99999021, -0.53657292,  0.42016704,  0.99060736,
                 0.65028784, -0.28790332, -0.96139749, -0.75098725,  0.1498772
         1])
```

```
In [51]: np.cos(array_1)
```

```
Out[51]: array([ 1.        ,  0.54030231, -0.41614684, -0.9899925 , -0.65364362,
                 0.28366219,  0.96017029,  0.75390225, -0.14550003, -0.91113026,
                -0.83907153,  0.0044257 ,  0.84385396,  0.90744678,  0.13673722,
                -0.75968791, -0.95765948, -0.27516334,  0.66031671,  0.9887046
         2])
```

```
In [52]: np.tan(array_1)
```

```
Out[52]: array([ 0.00000000e+00,  1.55740772e+00, -2.18503986e+00, -1.42546543e-
         01,
                 1.15782128e+00, -3.38051501e+00, -2.91006191e-01,  8.71447983e-
         01,
                -6.79971146e+00, -4.52315659e-01,  6.48360827e-01, -2.25950846e+
         02,
                -6.35859929e-01,  4.63021133e-01,  7.24460662e+00, -8.55993401e-
         01,
                 3.00632242e-01,  3.49391565e+00, -1.13731371e+00,  1.51589471e-
         01])
```

d. Statistical Functions

```
In [53]: arr5= np.random.rand(8)
```

```
In [54]: arr5
```

```
Out[54]: array([0.96371839, 0.32988914, 0.97677926, 0.22131449, 0.1199096 ,
                0.31986129, 0.53846315, 0.30176725])
```

```
In [55]: arr5.reshape(2,4)
```

```
Out[55]: array([[0.96371839, 0.32988914, 0.97677926, 0.22131449],
                [0.1199096 , 0.31986129, 0.53846315, 0.30176725]])
```

```
In [56]: np.mean(arr5)
```

```
Out[56]: 0.4714628215086668
```

```
In [57]: np.mean(arr5.reshape(2,4), axis= 0)   #Finding mean() columns wise

Out[57]: array([0.54181399, 0.32487521, 0.75762121, 0.26154087])

In [58]: np.mean(arr5.reshape(2,4), axis= 1)   #Finding mean() rows wise

Out[58]: array([0.62292532, 0.32000032])

In [59]: np.std(arr5)

Out[59]: 0.30824824982854293

In [60]: np.std(arr5.reshape(2,4), axis= 0)    #Finding std() columns wise

Out[60]: array([0.42190439, 0.00501392, 0.21915806, 0.04022638])

In [61]: np.std(arr5.reshape(2,4), axis= 1)    #Finding std() row wise

Out[61]: array([0.34946888, 0.14840381])

In [62]: np.sum(arr5)

Out[62]: 3.7717025720693345

In [63]: np.sum(arr5.reshape(2,4), axis= 0)    #Finding sum() columns wise

Out[63]: array([1.08362799, 0.64975043, 1.51524241, 0.52308174])

In [64]: np.sum(arr5.reshape(2,4), axis= 1)    #Finding sum() row wise

Out[64]: array([2.49170128, 1.28000129])
```

3. Array Indexing

3.1 One-Dimensional Array Indexing & Slicing

```
In [65]: arr1 = np.arange(10,21)
         arr1

Out[65]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20])

In [66]: arr1[7]     # Value at index No. 7 for arr1

Out[66]: 17

In [67]: arr1[1:6]   # Slicing the array starting at index 1 and stoping at inde
         x 6, index 6 being exclusive

Out[67]: array([11, 12, 13, 14, 15])

In [68]: arr1[0:6]

Out[68]: array([10, 11, 12, 13, 14, 15])

In [69]: arr1[:6]  # Everything upto index 6

Out[69]: array([10, 11, 12, 13, 14, 15])
```

NOTE: arr1[0:6] & **arr1[:6]** give the same output

```
In [70]: arr1[6:]  # Everything after index 6

Out[70]: array([16, 17, 18, 19, 20])

In [71]: arr1[:-1]  # Everything except the last value

Out[71]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])

In [72]: arr1[::-1]  # Reversing the index

Out[72]: array([20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10])
```

```
In [73]: arr1[::2]   # everything 2 item in the array starting index 0,
                     # Syntax (Start Index, Stop Index, Step)

Out[73]: array([10, 12, 14, 16, 18, 20])
```

3.2 Two-Dimensional Array Indexing & Slicing

```
In [74]: twod_array = np.array([[5,10,15,20], [6,12,18,24], [7,14,21,28]])

In [75]: twod_array

Out[75]: array([[ 5, 10, 15, 20],
                [ 6, 12, 18, 24],
                [ 7, 14, 21, 28]])

In [76]: twod_array.shape

Out[76]: (3, 4)

In [77]: twod_array[0, 0] # Extracing the element 5

Out[77]: 5

In [78]: twod_array[1, 1] # Extracing the element 12

Out[78]: 12

In [79]: twod_array[0:2,1:3]

Out[79]: array([[10, 15],
                [12, 18]])

In [80]: twod_array[:2, 1:]

Out[80]: array([[10, 15, 20],
                [12, 18, 24]])
```

3.3 Conditional Slicing

```
In [81]: arr2= np.arange(0,15)
         arr2

Out[81]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
In [82]: check_arr = arr2 > 5
         check_arr

Out[82]: array([False, False, False, False, False, False,  True,  True,  True,
                 True,  True,  True,  True,  True,  True])
```

```
In [83]: arr2[check_arr]

Out[83]: array([ 6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
In [84]: arr2[arr2>10]

Out[84]: array([11, 12, 13, 14])
```

```
In [85]: arr2[arr2<5]

Out[85]: array([0, 1, 2, 3, 4])
```

3.4 Practice Indexing

```
In [86]: practicearr = np.arange(0,100,2).reshape(5,10)
         practicearr

Out[86]: array([[ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18],
                [20, 22, 24, 26, 28, 30, 32, 34, 36, 38],
                [40, 42, 44, 46, 48, 50, 52, 54, 56, 58],
                [60, 62, 64, 66, 68, 70, 72, 74, 76, 78],
                [80, 82, 84, 86, 88, 90, 92, 94, 96, 98]])
```

4. Joining Arrays

```
In [87]: array_to_join = np.array([[10,20,30,70],[40,50,60,80],[70,80,90,70]])
```

```
In [88]: array_to_join
```

```
Out[88]: array([[10, 20, 30, 70],
                [40, 50, 60, 80],
                [70, 80, 90, 70]])
```

```
In [89]: twod_array
```

```
Out[89]: array([[ 5, 10, 15, 20],
                [ 6, 12, 18, 24],
                [ 7, 14, 21, 28]])
```

```
In [90]: np.concatenate( (twod_array,array_to_join), axis=1)  #Join at the Colum
         ns
```

```
Out[90]: array([[ 5, 10, 15, 20, 10, 20, 30, 70],
                [ 6, 12, 18, 24, 40, 50, 60, 80],
                [ 7, 14, 21, 28, 70, 80, 90, 70]])
```

```
In [91]: np.concatenate( (twod_array,array_to_join), axis=0)  #Join at the Rows
```

```
Out[91]: array([[ 5, 10, 15, 20],
                [ 6, 12, 18, 24],
                [ 7, 14, 21, 28],
                [10, 20, 30, 70],
                [40, 50, 60, 80],
                [70, 80, 90, 70]])
```