# Good Morning Everyone

## Welcome to Day 9 Session of Python Programming

## General Programming Concepts

- Data Types, Type Conversion, Operators
- Conditional Statements and Loops
- Lists, Tuples, Sets, Dictionaries
- Functions
- OOPs in Python
- Functional Programming i.e. List, Set and Dictionary Comprehensions

## If we are going to work on Data Fields like

- Data Analytics
- Machine Learning
- AI
- Deep Learning
- Field in Data

### Important Packages for those fields

- Numpy --> Mathematical Analysis
- Pandas --> Data Analysis, Cleaning, Visualizations
- Matplotlib --> Powerful Visualiation Package

In [1]:

```
1 import numpy as np
```

In [2]:

```
1 pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\jesus\anaconda3\lib\site-pa
ckages (1.16.2)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:

```
1 np.__version__
```

Out[3]:

```
'1.16.2'
```

In [4]:

```
li = [1,2,3,4,5,7]
type(li)
```

Out[4]:

```
list
```

In [5]:

```
ar1 = np.array([1,2,3,4,5])
type(ar1)
```

Out[5]:

```
numpy.ndarray
```

In [6]:

```
print(ar1.dtype)
```

```
int32
```

In [13]:

```
ar2 = np.array([1,2,3,4,'apssdc'])
print(type(ar2))
```

```
<class 'numpy.ndarray'>
```

In [14]:

```
print(ar2.dtype)
```

```
<U11
```

- 'b' − boolean
- 'i' − (signed) integer
- 'u' − unsigned integer
- 'f' − floating-point
- 'c' − complex-floating point
- 'm' − timedelta
- 'M' − datetime
- 'O' − (Python) objects
- 'S', 'a' − (byte-)string
- 'U' − Unicode
- 'V' − raw data (void)

In [19]:

```
date = np.array('28-05-2020')
```

In [20]:

```
date.dtype
```

Out[20]:

```
dtype('<U10')
```

In [22]:

```
complex1 = np.array(3+5j)
complex1.dtype
```

Out[22]:

```
dtype('complex128')
```

In [25]:

```
ar2 = np.arange(10)
print(ar2)
```

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

arange(initilization,condition,increment/decrement)

In [26]:

```
even = np.arange(0,1001,2)
print(even)
```

. . .

In [29]:

```
even[0:10]
```

Out[29]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [30]:

```
even.ndim
```

Out[30]:

```
1
```

In [31]:

```
arr = np.arange(0,10)
print(arr)
```

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

In [32]:

```
1 arr.ndim
```

Out[32]:

```
1
```

In [33]:

```
1 arr.shape
```

Out[33]:

```
(10,)
```

In [36]:

```
1 arr.reshape(5,2)
```

Out[36]:

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

In [38]:

```
1 arr.reshape(2,5)
```

Out[38]:

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

In [39]:

```
1 arr.reshape(10,1)
```

...

In [37]:

```
1 arr
```

Out[37]:

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

In [40]:

```
d2 = arr.reshape(2,5)
print(d2)
```

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

In [41]:

```
d2.shape
```

Out[41]:

```
(2, 5)
```

In [42]:

```
d2.ndim
```

Out[42]:

```
2
```

In [45]:

```
d3 = np.arange(0,20).reshape(2,2,5)
print(d3)
```

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

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

In [46]:

```
d3.shape
```

Out[46]:

```
(2, 2, 5)
```

In [47]:

```
d3.ndim
```

Out[47]:

```
3
```

In [49]:

```
ar2 = np.arange(1,10)
```

In [50]:

```
print(ar2)
```

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

In [51]:

```
ar2.reshape(3,-1)
```

Out[51]:

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

In [53]:

```
ar2.reshape(-1,3)
```

Out[53]:

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

In [56]:

```
ar = np.arange(1000)
```

In [57]:

```
print(ar)
```

...

In [59]:

```
ar2 = ar.reshape(-1,5)
print(ar2)
```

...

In [60]:

```
ar2.shape
```

Out[60]:

```
(200, 5)
```

In [61]:

```
ar3 = ar.reshape(5,-1)
```

In [62]:

```
1 print(ar3)
```

...

In [63]:

```
1 ar.reshape(500,2)
```

...

## Array Slicing

In [65]:

```
1 print(ar3.shape)
```

```
(5, 200)
```

In [68]:

```
1 ar3
```

Out[68]:

```
array([[  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,  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, 100, 101, 102, 103,
        104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
        117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
        130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
        143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
        156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
        169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
        182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
        195, 196, 197, 198, 199],
       [200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212,
        213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225,
        226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238,
        239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251,
        252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264,
        265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277,
        278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290,
        291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303,
        304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316,
        317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329,
        330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342,
        343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355,
        356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368,
        369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381,
        382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394,
        395, 396, 397, 398, 399],
       [400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412,
        413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425,
        426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438,
        439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451,
        452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464,
        465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477,
        478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490,
        491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503,
        504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516,
        517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529,
        530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542,
        543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555,
        556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568,
        569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581,
        582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594,
        595, 596, 597, 598, 599],
       [600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612,
        613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625,
        626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638,
        639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651,
        652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664,
        665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677,
```

```
        678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690,
        691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703,
        704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716,
        717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729,
        730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742,
        743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755,
        756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768,
        769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781,
        782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794,
        795, 796, 797, 798, 799],
       [800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812,
        813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825,
        826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838,
        839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851,
        852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864,
        865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877,
        878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890,
        891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903,
        904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916,
        917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929,
        930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942,
        943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955,
        956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968,
        969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981,
        982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994,
        995, 996, 997, 998, 999]])
```

In [71]:

```
1 ar3[4][100]
```

Out[71]:

```
900
```

In [72]:

```
1 ar3[4,100]
```

Out[72]:

```
900
```

In [73]:

```
1 ar3[4,100:150]
```

Out[73]:

```
array([900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912,
       913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925,
       926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938,
       939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949])
```

In [74]:

```
li = [1,2,3,4,5,6,7,8,9,10]

li2 = [3,4,5,6,7,8,9,10,11,12]
```

In [76]:

```
li2 = []
for i in range(0,len(li)):
    result = li[i] + 2
    li2.append(result)
print(li2)
```

```
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

In [88]:

```
ar = np.arange(1,11)
print(ar)
```

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

In [91]:

```
sumar = ar + 2
print(sumar)
```

```
[ 3  4  5  6  7  8  9 10 11 12]
```

In [92]:

```
print(ar)
```

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

In [85]:

```
mul2 = ar * 2
```

In [86]:

```
sub = ar - 0.5
```

In [82]:

```
ar
```

Out[82]:

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

In [83]:

```
ar
```

Out[83]:

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

In [95]:

```
bol = ar>5
print(bol)
```

```
[False False False False False  True  True  True  True  True]
```

In [96]:

```
ar[bol]
```

Out[96]:

```
array([ 6,  7,  8,  9, 10])
```

In [98]:

```
ar[ar % 2 == 0]
```

Out[98]:

```
array([ 2,  4,  6,  8, 10])
```

In [99]:

```
ar[ar % 3 ==0]
```

Out[99]:

```
array([3, 6, 9])
```

In [100]:

```
ar
```

Out[100]:

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

In [103]:

```
ar1 = np.arange(0,50)
ar2 = np.arange(25,75)
print(ar1.shape,ar2.shape)
```

```
(50,) (50,)
```

- np.logical_and()

- np.logical_or()
- np.locical_not()

In [109]:

```
years = np.arange(1900,2021)
print(years)
```

```
[1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913
 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927
 1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941
 1942 1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955
 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969
 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983
 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997
 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011
 2012 2013 2014 2015 2016 2017 2018 2019 2020]
```

- divisible by 4 and 400
- not divisible 100

In [111]:

```

```

```
[1904 1908 1912 1916 1920 1924 1928 1932 1936 1940 1944 1948 1952 1956
 1960 1964 1968 1972 1976 1980 1984 1988 1992 1996 2000 2004 2008 2012
 2016 2020]
```

# 11:00 - 11:10AM Break

In [114]:

```
zero = np.zeros(10)
print(zero)
```

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

In [115]:

```
zeroint = np.zeros(10, dtype = int)
print(zeroint)
```

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

In [117]:

```
zerostr = np.zeros(10, dtype = object)
print(zerostr)
```

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

In [118]:

```
1 zerostr.dtype
```

Out[118]:

```
dtype('O')
```

In [121]:

```
1 ones = np.ones(10, dtype = int)
2 print(ones)
```

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

In [124]:

```
1 identity = np.eye(5,5, dtype = int)
2 print(identity)
```

```
[[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 [125]:

```
1 numbers = np.linspace(1,10,50)
2 print(numbers)
```

```
[ 1.         1.18367347  1.36734694  1.55102041  1.73469388  1.91836735
  2.10204082  2.28571429  2.46938776  2.65306122  2.83673469  3.02040816
  3.20408163  3.3877551   3.57142857  3.75510204  3.93877551  4.12244898
  4.30612245  4.48979592  4.67346939  4.85714286  5.04081633  5.2244898
  5.40816327  5.59183673  5.7755102   5.95918367  6.14285714  6.32653061
  6.51020408  6.69387755  6.87755102  7.06122449  7.24489796  7.42857143
  7.6122449   7.79591837  7.97959184  8.16326531  8.34693878  8.53061224
  8.71428571  8.89795918  9.08163265  9.26530612  9.44897959  9.63265306
  9.81632653 10.        ]
```

In [127]:

```
1 num = np.linspace(1,2,5)
2 print(num)
```

```
[1.   1.25 1.5  1.75 2.  ]
```

# Trignometric Methods

- sin(radians)
- cos(radians)

- tan(radians)

In [129]:

```
deg = np.arange(0,91,15)
print(deg)
```

```
[ 0 15 30 45 60 75 90]
```

In [130]:

```
rad = np.radians(deg)
print(rad)
```

```
[0.         0.26179939 0.52359878 0.78539816 1.04719755 1.30899694
 1.57079633]
```

In [131]:

```
sin = np.sin(rad)
print(sin)
```

```
[0.         0.25881905 0.5        0.70710678 0.8660254  0.96592583
 1.        ]
```

In [132]:

```
cos = np.cos(rad)
print(cos)
```

```
[1.00000000e+00 9.65925826e-01 8.66025404e-01 7.07106781e-01
 5.00000000e-01 2.58819045e-01 6.12323400e-17]
```

In [133]:

```
np.sinh(rad)
```

Out[133]:

```
array([0.        , 0.26480023, 0.54785347, 0.86867096, 1.24936705,
       1.71618361, 2.3012989 ])
```

In [136]:

```
np.rad2deg(rad)
```

Out[136]:

```
array([ 0., 15., 30., 45., 60., 75., 90.])
```

# Statistical Methods

In [139]:

```
numbers = np.arange(100)
print(numbers)
```

```
[ 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
 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]
```

In [138]:

```
print(numbers.max())
```

```
99
```

In [141]:

```
print(numbers.min())
print(numbers.mean())
```

```
0
49.5
```

In [143]:

```
np.median(numbers)
```

Out[143]:

```
49.5
```

In [146]:

```
np.cumsum(numbers)
```

Out[146]:

```
array([   0,    1,    3,    6,   10,   15,   21,   28,   36,   45,   55,
         66,   78,   91,  105,  120,  136,  153,  171,  190,  210,  231,
        253,  276,  300,  325,  351,  378,  406,  435,  465,  496,  528,
        561,  595,  630,  666,  703,  741,  780,  820,  861,  903,  946,
        990, 1035, 1081, 1128, 1176, 1225, 1275, 1326, 1378, 1431, 1485,
       1540, 1596, 1653, 1711, 1770, 1830, 1891, 1953, 2016, 2080, 2145,
       2211, 2278, 2346, 2415, 2485, 2556, 2628, 2701, 2775, 2850, 2926,
       3003, 3081, 3160, 3240, 3321, 3403, 3486, 3570, 3655, 3741, 3828,
       3916, 4005, 4095, 4186, 4278, 4371, 4465, 4560, 4656, 4753, 4851,
       4950], dtype=int32)
```

In [149]:

```
1 np.cumprod(numbers)
```

Out[149]:

```
array([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, 0, 0, 0], dtype=int32)
```

In [150]:

```
1 num = np.arange(1,10)
2 print(num)
3 print(np.cumprod(num))
```

```
[1 2 3 4 5 6 7 8 9]
[     1      2      6     24    120    720   5040  40320 362880]
```

In [151]:

```
1 a1 = np.array([1,2,3])
2 a2 = np.array([3,2,1])
3 a3 = np.array([1,2,3])
```

In [152]:

```
1 np.array_equal(a1,a3)
```

Out[152]:

```
True
```

In [153]:

```
1 np.equal(a1,a2)
```

Out[153]:

```
array([False,  True, False])
```

2x^2 + 5sinx

In [154]:

```
1 x = np.array([1,2,3,4,5])
2 print(x)
```

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

In [155]:

```
2 * (x ** 2) + 5 * np.sin(np.radians(x))
```

Out[155]:

```
array([ 2.08726203,  8.17449748, 18.26167978, 32.34878237, 50.43577871])
```

In [162]:

```
np.not_equal(a1,a2)
```

Out[162]:

```
array([ True,  True,  True])
```

In [158]:

```
a1 != a3
```

Out[158]:

```
array([False, False, False])
```

^ --> bitwise xor operation

** --> exponential power

## Random Numbers

In [168]:

```
np.random.randint(1,100)
```

Out[168]:

```
73
```

In [170]:

```
1 np.random.random(100)
```

Out[170]:

```
array([0.99187952, 0.73116514, 0.90103482, 0.04025764, 0.95197634,
       0.24855552, 0.60721659, 0.13659625, 0.47547249, 0.99201665,
       0.66942856, 0.704177  , 0.56944934, 0.93138814, 0.62693366,
       0.68173532, 0.08306797, 0.61835636, 0.44638904, 0.68523825,
       0.84791384, 0.36965244, 0.75527206, 0.18824151, 0.43626094,
       0.41196182, 0.3030146 , 0.00424554, 0.87736995, 0.39197552,
       0.81585793, 0.75347431, 0.46340412, 0.26195866, 0.9610449 ,
       0.96904765, 0.17700463, 0.49079018, 0.88979116, 0.23987191,
       0.84659338, 0.62089698, 0.95046459, 0.12743962, 0.52441723,
       0.01961013, 0.95945248, 0.09187789, 0.57709395, 0.26654954,
       0.3777221 , 0.13265546, 0.68571107, 0.6497305 , 0.68265522,
       0.20812561, 0.97895933, 0.5535884 , 0.0883006 , 0.30776899,
       0.95622865, 0.99799084, 0.19322488, 0.22332927, 0.35864917,
       0.21941715, 0.56045444, 0.66160281, 0.6653938 , 0.30035802,
       0.97305044, 0.55230935, 0.84460598, 0.3714271 , 0.87886489,
       0.16022065, 0.14511631, 0.9117501 , 0.01369222, 0.86133289,
       0.10968715, 0.91502441, 0.23035742, 0.68275414, 0.608539  ,
       0.84644824, 0.99103147, 0.60321993, 0.54803084, 0.29872957,
       0.68192761, 0.72576303, 0.46851616, 0.76970849, 0.7800821 ,
       0.19493687, 0.33139021, 0.25122968, 0.97727863, 0.65326277])
```

In [172]:

```
1 np.random.randn(2,5)
```

Out[172]:

```
array([[-1.270119  , -1.3511539 ,  0.6994072 , -0.65504949, -0.1548724 ],
       [-0.94652132, -0.65390392,  0.19858826,  0.54376094,  1.31853664]])
```

In [173]:

```
1 np.random.random_integers(1,5)
```

```
C:\Users\Jesus\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Deprecat
ionWarning: This function is deprecated. Please call randint(1, 5 + 1) inste
ad
  """Entry point for launching an IPython kernel.
```

Out[173]:

```
2
```

# Brodcasting of Arrays

In [174]:

```
ar1 = np.arange(50).reshape(10,5)
ar2 = np.arange(50,0,-1).reshape(10,5)
print(ar1.shape,ar2.shape)
```

```
(10, 5) (10, 5)
```

In [178]:

```
print(ar1)
print('----------')
print(ar2)
```

```
[[ 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 48 49]]
----------
[[50 49 48 47 46]
 [45 44 43 42 41]
 [40 39 38 37 36]
 [35 34 33 32 31]
 [30 29 28 27 26]
 [25 24 23 22 21]
 [20 19 18 17 16]
 [15 14 13 12 11]
 [10  9  8  7  6]
 [ 5  4  3  2  1]]
```

In [175]:

```
np.hstack((ar1,ar2))
```

. . .

In [239]:

```
1 np.vstack((ar1,ar2))
```

Out[239]:

```
array([[ 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, 48, 49],
       [50, 49, 48, 47, 46],
       [45, 44, 43, 42, 41],
       [40, 39, 38, 37, 36],
       [35, 34, 33, 32, 31],
       [30, 29, 28, 27, 26],
       [25, 24, 23, 22, 21],
       [20, 19, 18, 17, 16],
       [15, 14, 13, 12, 11],
       [10,  9,  8,  7,  6],
       [ 5,  4,  3,  2,  1]])
```

In [190]:

```
1 ar3 = np.arange(0,10).reshape(2,5)
2 print(ar3, ar3.shape)
```

...

In [193]:

```
1 np.vstack((ar2,ar3))
```

Out[193]:

```
array([[50, 49, 48, 47, 46],
       [45, 44, 43, 42, 41],
       [40, 39, 38, 37, 36],
       [35, 34, 33, 32, 31],
       [30, 29, 28, 27, 26],
       [25, 24, 23, 22, 21],
       [20, 19, 18, 17, 16],
       [15, 14, 13, 12, 11],
       [10,  9,  8,  7,  6],
       [ 5,  4,  3,  2,  1],
       [ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9]])
```

In [194]:

```
1 ar3 = np.arange(0,20).reshape(10,2)
2 print(ar3, ar3.shape)
```

...

In [196]:

```
1 np.hstack((ar3,ar2))
```

. . .

In [197]:

```
1 print(ar1,ar2)
```

```
[[ 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 48 49]] [[50 49 48 47 46]
 [45 44 43 42 41]
 [40 39 38 37 36]
 [35 34 33 32 31]
 [30 29 28 27 26]
 [25 24 23 22 21]
 [20 19 18 17 16]
 [15 14 13 12 11]
 [10  9  8  7  6]
 [ 5  4  3  2  1]]
```

In [200]:

```
1 a1 = np.arange(0,9).reshape(3,3)
2 print(a1.shape)
```

```
(3, 3)
```

In [201]:

```
1 a2 = np.arange(9,18).reshape(3,3)
```

In [203]:

```
1 np.dot(a1,a2)
```

Out[203]:

```
array([[ 42,  45,  48],
       [150, 162, 174],
       [258, 279, 300]])
```

In [204]:

```
1 a1 * a2
```

Out[204]:

```
array([[  0,  10,  22],
       [ 36,  52,  70],
       [ 90, 112, 136]])
```

In [205]:

```
1 a1.T
```

Out[205]:

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

In [206]:

```
1 print(a1)
2 print(a2)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[ 9 10 11]
 [12 13 14]
 [15 16 17]]
```

In [209]:

```
1 print(a1)
```

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

In [232]:

```
1 np.random.shuffle(a1)
2 print(a1)
```

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

In [234]:

```
1 np.sort(a1, axis = 0)
```

Out[234]:

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

In [ ]:

```
1
```

In [227]:

```
1 x
```

Out[227]:

```
array(['9'], dtype='<U32')
```

In [235]:

```
1 np.diagonal(a1)
```

Out[235]:

```
array([6, 1, 5])
```

In [237]:

```
1 from numpy import diagonal as d
```

In [238]:

```
1 d(a1)
```

Out[238]:

```
array([6, 1, 5])
```