Numpy Crash Course

In [5]:
```
import numpy as np
```

In [6]:
```
np.__version__
```

Out[6]:
```
'1.26.4'
```

In [7]:
```
import sys
sys.version
```

Out[7]:
```
'3.12.7 | packaged by Anaconda, Inc. | (main, Oct  4 2024, 13:17:27) [MSC v.192
9 64 bit (AMD64)]'
```

Creating arrays

In [9]:
```
my_list = [0, 1, 2, 3, 4, 5]  #List data structure is defined with square bracke
my_list
```

Out[9]:
```
[0, 1, 2, 3, 4, 5]
```

In [10]:
```
type(my_list)  #Type of list is list
```

Out[10]:
```
list
```

Converting List to Array

In [12]:
```
#! pip install numpy -> To install any library we use pip install numpy/panda et
```

In [13]:
```
arr = np.array(my_list)
arr
```

Out[13]:
```
array([0, 1, 2, 3, 4, 5])
```

In [14]:
```
np.   #np. click tab - all functions of numpy are displayed
```

```
  Cell In[14], line 1
    np.   #np. click tab - all functions of numpy are displayed
          ^
SyntaxError: invalid syntax
```

In [ ]:
```
type(arr)  #Type of array is ndarray
```

In [73]:
```
np.arange(15) #Press shift+tab by clicking on arange -> Docstring is displayed
```

Out[73]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

March 12, 2025

In [76]:
```
np.arange(10) #Print index from 0 to n-1. Press Shift+tab to read docstring
```

Out[76]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [78]:
```
np.arange(3.0) # 1 float type argument passed, thus we get float output
```

Out[78]:
```
array([0., 1., 2.])
```

In [80]:
```
np.arange(9)
```

Out[80]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [82]:
```
np.arange(0,5) #Here 0 is the start index and 5 is the stop index
```

Out[82]:
```
array([0, 1, 2, 3, 4])
```

In [84]:
```
np.arange(10,20) #Here 10 is the start index and 20 is the stop index
```

Out[84]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [86]:
```
np.arange(20,10) # We get empty array as output. 1st argument should always be <
```

Out[86]:
```
array([], dtype=int32)
```

In [88]:
```
np.arange(-20, 10)
```

Out[88]:
```
array([-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 [90]:
```
np.arange() # We get error, because it miss stop argument
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[90], line 1
----> 1 np.arange()

TypeError: arange() requires stop to be specified.
```

In [92]:
```
np.arange(10,30,5)  #Here 10 = Starting point, 30 = Stop Point, 5 = Step count
```

Out[92]:
```
array([10, 15, 20, 25])
```

In [94]:
```
np.arange(0,10,3)
```

Out[94]:
```
array([0, 3, 6, 9])
```

In [96]:
```
np.arange(10,30,5,8) #We get error as it takes only 3 argument and not 4 - Start
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[96], line 1
----> 1 np.arange(10,30,5,8)

TypeError: Cannot interpret '8' as a data type
```

In [98]:
```
np.zeros(3)
```

Out[98]:
```
array([0., 0., 0.])
```

In [100…
```
np.zeros(10) # Zeros is a funtion printing 0, by default the output is in float
```

Out[100…
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

To get integer output, add another parameter

In [103…
```
np.zeros(10, dtype=int) #This is called Hyperparameter Tuning
```

Out[103…
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

array([0,0,0,0,0,0,0,0]) -> This is a 1-D array

For 2-D Array - np.zeros((2,2), dtype=int)

In [107…
```
np.zeros((2,2),dtype=int)   # We get 2-D array
```

Out[107…
```
array([[0, 0],
       [0, 0]])
```

np.zeros((2,10)) #If we write only 1 bracket then it gives error

Above is 2 rows and 10 columns

In [110…
```
np.zeros((5,10))   #We get 5 rows and 10 columns
```

Out[110…
```
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.]])
```

In [111…
```
np.zeros((5,10), dtype=int)   #We get 5 rows and 10 columns of int type. Here dt
```

Out[111…
```
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]])
```

In [112…
```
np.ones(3)
```

Out[112…
```
array([1., 1., 1.])
```

In [113…
```
np.ones(3,dtype=int)  # Output is 1-d array
```

Out[113…
```
array([1, 1, 1])
```

In [114…
```
np.ones((2,3))  #This is a 2-d array
```

Out[114…
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

In [115…
```
np.ones((3,3))  #This is a 3-d array
```

Out[115…
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [116…
```
np.ones(4, dtype=int)  #This is a 1-d array
```

Out[116…
```
array([1, 1, 1, 1])
```

In [117…
```
np.ones((5,4), dtype=int)  # 5 rows and 4 columns
```

Out[117…
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

In [118…
```
np.twos((2,3))  #twos is not a function of np. Check by clicking on .tab
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[118], line 1
----> 1 np.twos((2,3))

File ~\anaconda3\Lib\site-packages\numpy\__init__.py:333, in __getattr__(attr)
    330     "Removed in NumPy 1.25.0"
    331     raise RuntimeError("Tester was removed in NumPy 1.25.")
--> 333 raise AttributeError("module {!r} has no attribute "
    334                      "{!r}".format(__name__, attr))

AttributeError: module 'numpy' has no attribute 'twos'
```

In [ ]:
```
np.three((2,3))  #threes is not a function of np. Check by clicking on .tab
```

In [ ]:
```
rand(3,2) #Gives error as rand is not a function
```

In [ ]:
```
random.rand(3,2) #Gives error as rand is not a function
```

In [ ]:
```
np.random.rand(5)  # This generates random 5 numbers which vary everytime
```

In above -

1. np - Package
2. random - module
3. rand - function
4. (5) - argument

In [120…
```
np.rand(4) # This gives error as module is missing
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[120], line 1
----> 1 np.rand(4)

File ~\anaconda3\Lib\site-packages\numpy\__init__.py:333, in __getattr__(attr)
    330     "Removed in NumPy 1.25.0"
    331     raise RuntimeError("Tester was removed in NumPy 1.25.")
--> 333 raise AttributeError("module {!r} has no attribute "
    334                      "{!r}".format(__name__, attr))

AttributeError: module 'numpy' has no attribute 'rand'
```

In [ ]:
```
np.random.rand(5) #Now it works as module is present
```

In [ ]:
```
np.random.rand(3,5) # Means 3 rows and 5 columns. These values shuffle everytime
```

In [121…
```
np.random.randint(2, 4)  #shift+tab for details - 2 is inclusive and 4 is exclus
```

Out[121…
```
3
```

Everytime we will get 2 or 3 but not 4 as 4 is exclusive

In [123…
```
np.random.randint(4, 6)
```

Out[123…
```
4
```

In [124…
```
np.random.randint(2, 20) # 2nd argument is exclusive. It prints random numbers.
```

Out[124…
```
14
```

In [125…
```
np.random.randint(0, 1) # 2nd argument is exclusive. It prints random numbers.
```

Out[125…
```
0
```

In [126…
```
np.random.randint(10, 20, 5) # 2nd argument is exclusive. 3rd argument is the co
```

Out[126…
```
array([13, 18, 10, 19, 10])
```

In [127…
```
np.random.randint(1, 6, 4) # 2nd argument is exclusive. 3rd argument is the coun
```

Out[127…
```
array([1, 3, 1, 1])
```

In [128…
```
np.random.rand(3)
```

Out[128…
```
array([0.44702123, 0.77033116, 0.18656642])
```

In [129…
```
np.arange(1, 13)
```

Out[129…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [130…
```
np.arange(1, 13).reshape(3, 4)  # 3 rows and 4 columns
```

Out[130…
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [131…
```
np.arange(1, 13).reshape(5, 4)  # error as we need 20 elements for printing in r
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[131], line 1
----> 1 np.arange(1, 13).reshape(5, 4)

ValueError: cannot reshape array of size 12 into shape (5,4)
```

In [ ]:
```
np.arange(1, 13).reshape(6, 2)
```

In [132…
```
np.arange(1, 13).reshape(6, 1) #error as reshape needs 6 elemnts not 12
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[132], line 1
----> 1 np.arange(1, 13).reshape(6, 1)

ValueError: cannot reshape array of size 12 into shape (6,1)
```

In [ ]:
```
np.arange(1, 13).reshape(12, 1) #error as reshape needs 6 elemnts not 12
```

March 13' 2025

In [164…
```
np.random.randint(10,40,(10,10))  #random is a module and randint is a function.
```

Out[164…
```
array([[19, 19, 17, 24, 29, 23, 25, 23, 11, 21],
       [33, 23, 14, 19, 27, 32, 12, 37, 10, 33],
       [17, 12, 27, 12, 28, 17, 29, 11, 33, 27],
       [26, 35, 12, 30, 11, 35, 11, 15, 23, 27],
       [25, 20, 13, 14, 20, 35, 38, 27, 36, 18],
       [26, 14, 20, 21, 23, 15, 14, 29, 28, 10],
       [20, 14, 25, 33, 24, 25, 14, 28, 24, 34],
       [27, 38, 28, 37, 36, 27, 39, 31, 31, 36],
       [23, 35, 29, 13, 26, 10, 20, 27, 30, 33],
       [14, 35, 32, 36, 11, 39, 19, 36, 15, 36]])
```

In [166…
```
np.arange(1,13)
```

Out[166…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [168…
```
np.arange(1,13).reshape(3,4)
```

Out[168…
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [170…
```
np.arange(1,13).reshape(5,4) #Invalid bec 5*4=20, but we want to arrange only 12
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[170], line 1
----> 1 np.arange(1,13).reshape(5,4)

ValueError: cannot reshape array of size 12 into shape (5,4)
```

Matrix Slicing

In [173…
```
b=np.random.randint(10,20,(5,4))
b
```

Out[173…
```
array([[18, 15, 11, 11],
       [18, 19, 17, 11],
       [19, 13, 18, 12],
       [17, 14, 13, 12],
       [15, 11, 12, 19]])
```

In [175…
```
type(b)
```

Out[175…
```
numpy.ndarray
```

ndarray means 'n' number of rows and 'n' number of columns

Calling b directly

In [179…
```
b  # Array begins with 0 i.e Rows - 0, 1, 2, 3
```

Out[179…
```
array([[18, 15, 11, 11],
       [18, 19, 17, 11],
       [19, 13, 18, 12],
       [17, 14, 13, 12],
       [15, 11, 12, 19]])
```

In [181…
```
b  # b slicing - Prints entire matrix
```

Out[181…
```
array([[18, 15, 11, 11],
       [18, 19, 17, 11],
       [19, 13, 18, 12],
       [17, 14, 13, 12],
       [15, 11, 12, 19]])
```

In [183…
```
b[1:3] #slicing Indexing formula is applicable for matrix also i.e 1 to n-1(2nd
```

Out[183…
```
array([[18, 19, 17, 11],
       [19, 13, 18, 12]])
```

here 1:3 means 1th row to (3-1)th row

In [186…
```
b[1,2] # Indexing starts with 0 only in matrix for rows and columns
```

Out[186…
```
17
```

[:] - Slicing prints entire row

[,] - prints specific elements not entire row

In [189…
```
b[1,3]
```

Out[189…
```
11
```

In [195…
```
b[1,-1]  # -1 means backward slicing (1th row and -1th column element)
```

Out[195…
```
11
```

In [193…
```
b
```

Out[193…
```
array([[18, 15, 11, 11],
       [18, 19, 17, 11],
       [19, 13, 18, 12],
       [17, 14, 13, 12],
       [15, 11, 12, 19]])
```

In [197…
```
b[2:3]
```

Out[197…
```
array([[19, 13, 18, 12]])
```

In [199…
```
b[2:2]  # Gives blank array -> bec 2nd row and (2-1)th row from bottom
```

Out[199…
```
array([], shape=(0, 4), dtype=int32)
```

- The slice b[2:2] starts and ends at index 2.
- Since there's no room between the start and end index, the result is an empty list.

In [202…
```
b[0:-2]  # 2nd index formula -> n-1 thus -2-1 = -3. So it prints until -3 backwa
```

Out[202…
```
array([[18, 15, 11, 11],
       [18, 19, 17, 11],
       [19, 13, 18, 12]])
```

In [204…
```
b[-4:2]
```

Out[204…
```
array([[18, 19, 17, 11]])
```

- b[-4:2] is the same as b[1:2]

1. Step 1: Understand the indices

- -4 means start from the 2nd element (indexing starts at 0, so -4 is equivalent to index 1)
- 2 means stop before index 2

2. Step 2: Determine the direction

- In Python slicing, slicing always goes left to right (forward) unless a negative step is specified. If the start index is greater than the stop index and no negative step is given, it returns an empty list.
- In this case:

Start at index 1 (value: 20)

End before index 2 (value: 30)

So the slice includes just index 1.

Operations

In [211…
```
a = np.random.randint(10, 20, 10)  # This generates 10 numbers between 10 to 20,
a
```

Out[211…
```
array([19, 12, 16, 10, 11, 15, 11, 10, 18, 11])
```

In [213…
```
id(a)  #id gives memory location i.e address of 'a'
```

Out[213…
```
1412780254736
```

In [217…
```
arr2 = np.random.randint(0,100,(10,10))  # This generates 10*10 matrix of number
arr2
```

Out[217…
```
array([[95, 94, 61, 68, 29, 62, 89,  4, 68, 32],
       [18, 52, 69, 71,  9, 91, 73, 48, 44, 44],
       [ 8, 34, 10, 70, 25, 91, 12, 68, 56, 23],
       [ 9,  1, 90,  8,  0,  9, 76, 89, 57, 20],
       [80, 88, 90, 88, 53, 95, 31, 67, 59, 30],
       [94, 35, 80, 80, 91, 51, 99, 79, 81, 57],
       [95,  9, 63, 22, 64, 28, 84, 12,  2, 10],
       [67, 92, 68, 14, 48, 76,  4, 13,  0, 91],
       [47, 91, 92, 42, 95, 61, 48, 32, 62, 82],
       [84, 53, 90, 60, 68, 74, 93, 96, 38, 50]])
```

In [219…
```
arr2[::-1]  # This reverse the matrix  (Last row becomes 1st and 1st becomes las
```

Out[219…
```
array([[84, 53, 90, 60, 68, 74, 93, 96, 38, 50],
       [47, 91, 92, 42, 95, 61, 48, 32, 62, 82],
       [67, 92, 68, 14, 48, 76,  4, 13,  0, 91],
       [95,  9, 63, 22, 64, 28, 84, 12,  2, 10],
       [94, 35, 80, 80, 91, 51, 99, 79, 81, 57],
       [80, 88, 90, 88, 53, 95, 31, 67, 59, 30],
       [ 9,  1, 90,  8,  0,  9, 76, 89, 57, 20],
       [ 8, 34, 10, 70, 25, 91, 12, 68, 56, 23],
       [18, 52, 69, 71,  9, 91, 73, 48, 44, 44],
       [95, 94, 61, 68, 29, 62, 89,  4, 68, 32]])
```

In [221…
```
arr2[::-2]  #-2 is the step count.
```

Out[221…
```
array([[84, 53, 90, 60, 68, 74, 93, 96, 38, 50],
       [67, 92, 68, 14, 48, 76,  4, 13,  0, 91],
       [94, 35, 80, 80, 91, 51, 99, 79, 81, 57],
       [ 9,  1, 90,  8,  0,  9, 76, 89, 57, 20],
       [18, 52, 69, 71,  9, 91, 73, 48, 44, 44]])
```

In [225…
```
arr2[::-3]  #-3 is the step count
```

Out[225…
```
array([[84, 53, 90, 60, 68, 74, 93, 96, 38, 50],
       [95,  9, 63, 22, 64, 28, 84, 12,  2, 10],
       [ 9,  1, 90,  8,  0,  9, 76, 89, 57, 20],
       [95, 94, 61, 68, 29, 62, 89,  4, 68, 32]])
```

In [227…
```
arr2[:-3]  # Slice -3 means n-1 = -3-1 = -4 so, from -4 uptill 1st row everythin
```

Out[227…
```
array([[95, 94, 61, 68, 29, 62, 89,  4, 68, 32],
       [18, 52, 69, 71,  9, 91, 73, 48, 44, 44],
       [ 8, 34, 10, 70, 25, 91, 12, 68, 56, 23],
       [ 9,  1, 90,  8,  0,  9, 76, 89, 57, 20],
       [80, 88, 90, 88, 53, 95, 31, 67, 59, 30],
       [94, 35, 80, 80, 91, 51, 99, 79, 81, 57],
       [95,  9, 63, 22, 64, 28, 84, 12,  2, 10]])
```

In [229…
```
arr2[3:]  # 3 slice means print all rows starting from 3rd row untill last
```

Out[229…
```
array([[ 9,  1, 90,  8,  0,  9, 76, 89, 57, 20],
       [80, 88, 90, 88, 53, 95, 31, 67, 59, 30],
       [94, 35, 80, 80, 91, 51, 99, 79, 81, 57],
       [95,  9, 63, 22, 64, 28, 84, 12,  2, 10],
       [67, 92, 68, 14, 48, 76,  4, 13,  0, 91],
       [47, 91, 92, 42, 95, 61, 48, 32, 62, 82],
       [84, 53, 90, 60, 68, 74, 93, 96, 38, 50]])
```

In [231…
```
arr
```

Out[231…
```
array([0, 1, 2, 3, 4, 5])
```

In [233…
```
arr.max()
```

Out[233…
```
5
```

In [235…
```
arr.min()
```

Out[235…
```
0
```

In [237…
```
arr.mean()
```

Out[237…
```
2.5
```

In [239…
```
arr.median()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[239], line 1
----> 1 arr.median()

AttributeError: 'numpy.ndarray' object has no attribute 'median'
```

In [241…
```
from numpy import *
a = array([1,2,3,4,9])
median(a)
```

Out[241…
```
3.0
```

To find the median of a dataset, first arrange the numbers in ascending order. If there's an odd number of values, the median is the middle value. If there's an even number of values, the median is the average of the two middle values.

Indexing

In [245…
```
mat = np.arange(0,100).reshape(10,10)
```

In [247…
```
mat
```

Out[247…
```
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]])
```

In [249…
```
row = 4
col = 5
```

In [251…
```
row
```

Out[251…
```
4
```

In [253…
```
col
```

Out[253…
```
5
```

In [255…
```
mat[row,col]  #This captures the value of 4th row and 5th column
```

Out[255…
```
45
```

In [257…
```
mat[4,5]
```

Out[257…
```
45
```

In [259…
```
mat[:] #Empty slice gives entire matrix
```

Out[259…
```
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]])
```

In [261…
```
mat[6] #displays 6th row
```

Out[261…
```
array([60, 61, 62, 63, 64, 65, 66, 67, 68, 69])
```

To Print specific column

In [264…
```
mat[:,5]
```

Out[264…
```
array([ 5, 15, 25, 35, 45, 55, 65, 75, 85, 95])
```

In [267…
```
mat[0:10]
```

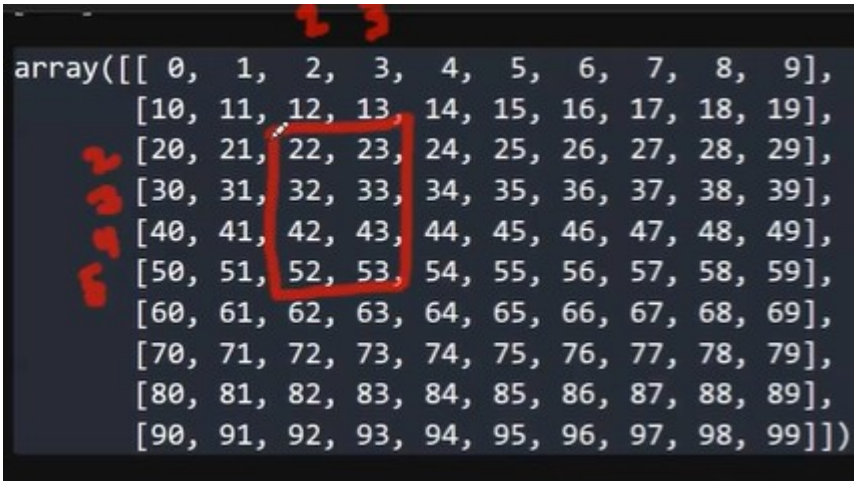
Out[267…
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 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]])
```

In [269…
```
mat[0:10:3] # 3 indicates steps
```

Out[269…
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

In [271…
```
mat[2:6,2:4]   #2 slice 6 --> Only row part, starting from 2nd row until 5th row
```

Out[271…
```
array([[22, 23],
       [32, 33],
       [42, 43],
       [52, 53]])
```



In [273…
```
mat[1:2,2:4]
```

Out[273…
```
array([[12, 13]])
```

Numpy Masking or Numpy Filters

In [276…
```
mat < 50 #It prints bool matrix. Upto 49 we get true. Filters value
```

Out[276…
```
array([[ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False]])
```

In [278…
```
mat[mat < 50] #It prints numbers. Upto 49 we get the output
```

Out[278…
```
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])
```

In [280…
```
mat[mat <= 50] #To print 50 also
```

Out[280…
```
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])
```

In [282…
```
mat[mat != 50] #50 is not printed
```

Out[282…
```
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, 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 [284…
```
mat[mat == 50] #50 is printed
```

Out[284…
```
array([50])
```

In [286…
```
mat[mat > 50] #It prints numbers more than 50
```

Out[286…
```
array([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 [ ]:

In [ ]: