# *Assignment 8*

# NUMPY

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

In [2]:
```
np.__version__
```

Out[2]:
```
'1.26.4'
```

In [4]:
```
import sys
sys.version
```

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

In [9]:
```
l=[12,23,445,4,53,445,34,53]
l
```

Out[9]:
```
[12, 23, 445, 4, 53, 445, 34, 53]
```

In [10]:
```
type(l)
```

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

# array creation

In [12]:
```
arr=np.array(l)
arr
```

Out[12]:
```
array([ 12,  23, 445,   4,  53, 445,  34,  53])
```

In [13]:
```
type(arr)
```

Out[13]:
```
numpy.ndarray
```

In [14]:
```
type(l)
```

Out[14]:
```
list
```

# arange()

- The arange() function in NumPy returns an array with regularly spaced values within a given interval.

`numpy.arange([start,] stop[, step], dtype=None)`

start (optional): Start of the interval. Default is 0.

stop: End of the interval (not included).

step (optional): Step size. Default is 1.

dtype (optional): Desired data type.

In [15]:
```
np.arange(10)
```

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

In [16]:
```
np.arange(28)
```

Out[16]:
```
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])
```

In [17]:
```
np.arange(45)
```

Out[17]:
```
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])
```

In [18]:

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

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

#np.arange(20, 10) creates an array starting at 20 and ending before 10, with th

#Since it's counting up from 20 to 10, and 20 is already greater than 10, it ret
```

Out[20]:
```
array([], dtype=int32)
```

In [21]:
```
np.arange(10,35,5)
```

Out[21]:
```
array([10, 15, 20, 25, 30])
```

In [26]:
```
np.zeros(10)

#p.zeros(10) creates a NumPy array of length 10 filled with zeros.

#The default data type is float64.
#All elements are 0.0 (not just 0), unless you specify dtype=int
```

Out[26]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [27]:
```
np.zeros(10,dtype=int)
```

Out[27]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [25]:
```
np.zeros((2,2),dtype=int)

# np.zeros((2, 2), dtype=int) creates a 2x2 matrix (2 rows × 2 columns).
```

```
# All elements are zeros.
# The data type is set to int instead of the default float
```

Out[25]:
```
array([[0, 0],
       [0, 0]])
```

In [28]:
```
np.zeros((3,3),dtype=int)
```

Out[28]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

In [33]:
```
zero =np.zeros([2,2])
print(zero)

print(type(zero))


# np.zeros([2, 2]) creates a 2x2 NumPy array filled with 0.0 (default float type

# The shape [2, 2] is the same as (2, 2) — both are valid for defining dimension
```

```
[[0. 0.]
 [0. 0.]]
<class 'numpy.ndarray'>
```

In [34]:
```
np.zeros((10,10),dtype=int)
```

Out[34]:
```
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, 0]])
```

In [45]:
```
np.ones((2,2))

#np.ones((2, 2)) creates a 2x2 NumPy array filled with 1.0 (default type is floa
```

Out[45]:
```
array([[1., 1.],
       [1., 1.]])
```

In [46]:
```
np.ones((2,2),dtype=int)
```

Out[46]:
```
array([[1, 1],
       [1, 1]])
```

# np.random.rand in NumPy

np.random.rand is used to generate random numbers from a uniform distribution over [0, 1).

- 📌 Syntax:
  `np.random.rand(d0, d1, ..., dn)`
- Takes one or more dimensions as arguments.
- Returns an array of the given shape filled with random floats between 0 (inclusive) and 1 (exclusive).

In [47]:
```
np.random.rand(4,8)
```

Out[47]:
```
array([[0.73644451, 0.85938973, 0.66828797, 0.78533014, 0.75179144,
        0.20880871, 0.32016755, 0.56654268],
       [0.0230776 , 0.24601273, 0.70368906, 0.31287716, 0.2854384 ,
        0.53035797, 0.60813786, 0.84997838],
       [0.50777322, 0.63088279, 0.81404012, 0.03867555, 0.07122858,
        0.08208778, 0.55742217, 0.38555534],
       [0.77901097, 0.22152709, 0.72234473, 0.22316319, 0.25802061,
        0.97438492, 0.93304012, 0.31025579]])
```

In [48]:
```
np.random.rand(3,2)
```

Out[48]:
```
array([[0.48991506, 0.43647921],
       [0.40139803, 0.92057623],
       [0.91115884, 0.80240708]])
```

In [54]:
```
np.random.rand(3,12)
```

Out[54]:
```
array([[0.98886752, 0.18085799, 0.61357887, 0.56291474, 0.37902103,
        0.24319659, 0.29788721, 0.92476745, 0.00699442, 0.89619944,
        0.57453475, 0.92923496],
       [0.46875188, 0.65152641, 0.56136911, 0.29603014, 0.83253098,
        0.2624841 , 0.33388804, 0.46591759, 0.50576041, 0.61560582,
        0.8478839 , 0.78701028],
       [0.1391883 , 0.57817293, 0.9068151 , 0.68353711, 0.40895466,
        0.40568396, 0.73516271, 0.20685632, 0.07628909, 0.52183515,
        0.80018193, 0.95172695]])
```

# np.random.randint() in NumPy

np.random.randint() generates random integers from a specified range.

📌 Syntax:

`np.random.randint(low, high=None, size=None, dtype=int)`

---

|Parameter |Description

---

|low -----> |Lowest (inclusive) integer to be drawn.

---

|high -----> |Highest (exclusive) integer to be drawn. If None, values are drawn from [0, low).

---

|size -----> |Output shape (e.g., 5 or (2,3)).

---

|dtype -----> |Output type (default: int).

---

In [55]:
```
np.random.randint(3)
```

Out[55]:
```
0
```

In [59]:
```
np.random.randint(1,10,2)
```

Out[59]:
```
array([3, 9])
```

In [57]:
```
np.random.randint(1,12,10)
```

Out[57]:
```
array([ 6,  3,  7, 10,  1,  2,  5, 11,  8, 10])
```

In [61]:
```
np.random.randint(1,12,((10,10)))
#        🔍 General Form:
#                        np.random.randint(low, high, size)
# Parameter      Meaning
# low        Minimum value (inclusive) — numbers will be ≥ low
# high      Maximum value (exclusive) — numbers will be < high
# size      Shape of the output array (e.g., (rows, columns))
```

Out[61]:
```
array([[ 5,  3, 10, 10,  2, 10,  5, 11,  6,  9],
       [ 6,  3,  9, 10,  1,  1, 10,  5,  9, 10],
       [ 4,  9, 10,  4,  5,  7,  3,  6,  4,  9],
       [ 5,  7, 11,  2,  7,  9,  6,  6,  5,  2],
       [ 7, 11,  9,  3,  5,  2,  8,  5,  3,  4],
       [ 9,  9,  2,  6,  1,  1,  4,  9,  9,  8],
       [ 5,  9,  1,  1,  7,  9,  3,  4,  1,  5],
       [ 5,  9,  6,  9,  7,  8,  7, 11,  3,  1],
       [10,  7, 11,  3,  3,  7, 11,  6,  1,  5],
       [ 4, 10,  5, 10,  3,  5,  6,  6,  7, 11]])
```

In [63]:
```
np.arange(1,12)
```

Out[63]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

# reshape()

The reshape() function in NumPy is used to change the shape of an existing array without changing its data.Syntax:

📌 Syntax:

`numpy.reshape(a, newshape)`

or, using array object:

`a.reshape(newshape)`

- **a**: The array to reshape.

- **newshape**: Tuple defining the new shape. Must have the same total number of elements as the original array.

In [67]:
```
np.arange(1,13).reshape(3,4)

# ⚠️ Important Rule:
# Total elements must match:

# np.arange(1, 13) → 12 elements

# reshape(3, 4) → 3 × 4 = 12 elements ✅

# If you try reshape(4, 4) with this, it will raise a ValueError.

# Let me know if you want to reshape it into other formats like 4x3, 2x6,
```

Out[67]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [68]:
```
np.arange(1,13).reshape(4,3)
```

Out[68]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [69]:

Out[69]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

In [70]:
```
np.arange(1,13).reshape(6,2)
```

Out[70]:
```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12]])
```

In [71]:
```
np.arange(1,13).reshape(12,1)
```

Out[71]:
```
array([[ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12]])
```

In [75]:
```
np.arange(1,13).reshape(1,12) # reshape argu must be such that the product of th
```

Out[75]:
```
array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]])
```

# slicing

In [77]:
```
b=np.random.randint(100,200,(4,5))
b
```

Out[77]:
```
array([[150, 196, 118, 135, 142],
       [177, 140, 101, 198, 122],
       [164, 151, 126, 136, 114],
       [119, 131, 161, 129, 128]])
```

In [78]:
```
b
```

Out[78]:
```
array([[150, 196, 118, 135, 142],
       [177, 140, 101, 198, 122],
       [164, 151, 126, 136, 114],
       [119, 131, 161, 129, 128]])
```

In [79]:
```
b[:]
```

Out[79]:
```
array([[150, 196, 118, 135, 142],
       [177, 140, 101, 198, 122],
       [164, 151, 126, 136, 114],
       [119, 131, 161, 129, 128]])
```

In [82]:
```
b[1:3]
#   🔍 Meaning of b[1:3]:
#  b[1:3] slices rows from index 1 to 2 (not including 3).

# It returns the 2nd and 3rd rows of the 4×5 matrix.
```

Out[82]:
```
array([[177, 140, 101, 198, 122],
       [164, 151, 126, 136, 114]])
```

In [83]:
```
b[1,3]
#Explanation:
#     If b is a 2D array (like shape (4, 5)), then:


#    b[row_index, column_index]

#    b[1, 3]

#    means:
#    Row index = 1 → 2nd row
#    Column index = 3 → 4th column
#    Returns the element at row 2, column 4
```

Out[83]:
```
198
```

In [85]:
```
b[1:-1]
#   Means:

#   from row index 1 (2nd row)

#   Go up to but not including the last row (-1)

#   So it returns all rows from 1 to second-last row.
```

Out[85]:
```
array([[177, 140, 101, 198, 122],
       [164, 151, 126, 136, 114]])
```

In [86]:
```
b[2:3]
```

Out[86]:
```
array([[164, 151, 126, 136, 114]])
```

In [87]:
```
b[0:-2]
```

Out[87]:
```
array([[150, 196, 118, 135, 142],
       [177, 140, 101, 198, 122]])
```

In [88]:
```
b
```

Out[88]:
```
array([[150, 196, 118, 135, 142],
       [177, 140, 101, 198, 122],
       [164, 151, 126, 136, 114],
       [119, 131, 161, 129, 128]])
```

# operations

In [90]:
```
a2=np.random.randint(1,12,(10,10))
a2
```

Out[90]:
```
array([[11,  8,  4,  9,  3,  3,  1,  7,  4,  9],
       [ 4,  8,  8,  9,  4,  7,  8,  4,  7,  4],
       [ 4,  2,  6,  4,  5, 10, 10,  9,  1,  1],
       [ 8,  3,  4,  7,  9,  2,  1,  5,  3,  4],
       [ 8, 10,  3,  7, 10,  9,  4,  9,  9,  9],
       [ 1,  9,  5,  4,  8,  1,  4,  7,  1,  8],
       [ 1,  3,  7,  7,  1,  6,  7,  1,  6,  5],
       [11,  8,  3,  7,  8,  9,  9,  6,  8, 10],
       [ 6, 10,  6,  4,  9,  2,  2,  8,  3, 11],
       [ 5,  9,  1, 10, 10,  8,  2,  5, 11,  3]])
```

In [93]:
```
a2[::1]
# 🔍 Explanation: a2[::]

#This is a slice of the rows in the array a2.

#a2[::] is equivalent to:

#              a2[0:10:1]

#Which means:
#         Start from row index 0
#         Go to the end (10)
#         Step by 1

# ✅ So a2[::] returns all rows of a2 — it's effectively the same as just writi
```

Out[93]:
```
array([[11,  8,  4,  9,  3,  3,  1,  7,  4,  9],
       [ 4,  8,  8,  9,  4,  7,  8,  4,  7,  4],
       [ 4,  2,  6,  4,  5, 10, 10,  9,  1,  1],
       [ 8,  3,  4,  7,  9,  2,  1,  5,  3,  4],
       [ 8, 10,  3,  7, 10,  9,  4,  9,  9,  9],
       [ 1,  9,  5,  4,  8,  1,  4,  7,  1,  8],
       [ 1,  3,  7,  7,  1,  6,  7,  1,  6,  5],
       [11,  8,  3,  7,  8,  9,  9,  6,  8, 10],
       [ 6, 10,  6,  4,  9,  2,  2,  8,  3, 11],
       [ 5,  9,  1, 10, 10,  8,  2,  5, 11,  3]])
```

- Tip: You can use variations like:

a2[::2] → Every 2nd row

a2[::-1] → Reverse the rows

In [94]:
```
a2[::2]
```

Out[94]:
```
array([[11,  8,  4,  9,  3,  3,  1,  7,  4,  9],
       [ 4,  2,  6,  4,  5, 10, 10,  9,  1,  1],
       [ 8, 10,  3,  7, 10,  9,  4,  9,  9,  9],
       [ 1,  3,  7,  7,  1,  6,  7,  1,  6,  5],
       [ 6, 10,  6,  4,  9,  2,  2,  8,  3, 11]])
```

In [95]:
```
a2[::-2]
```

Out[95]:
```
array([[ 5,  9,  1, 10, 10,  8,  2,  5, 11,  3],
       [11,  8,  3,  7,  8,  9,  9,  6,  8, 10],
       [ 1,  9,  5,  4,  8,  1,  4,  7,  1,  8],
       [ 8,  3,  4,  7,  9,  2,  1,  5,  3,  4],
       [ 4,  8,  8,  9,  4,  7,  8,  4,  7,  4]])
```

In [96]:
```
a2[::-3]
```

Out[96]:
```
array([[ 5,  9,  1, 10, 10,  8,  2,  5, 11,  3],
       [ 1,  3,  7,  7,  1,  6,  7,  1,  6,  5],
       [ 8,  3,  4,  7,  9,  2,  1,  5,  3,  4],
       [11,  8,  4,  9,  3,  3,  1,  7,  4,  9]])
```

In [98]:
```
a2[0:10:3]
```

Out[98]:
```
array([[11,  8,  4,  9,  3,  3,  1,  7,  4,  9],
       [ 8,  3,  4,  7,  9,  2,  1,  5,  3,  4],
       [ 1,  3,  7,  7,  1,  6,  7,  1,  6,  5],
       [ 5,  9,  1, 10, 10,  8,  2,  5, 11,  3]])
```

In [101…
```
a2.max()  # return the maximum vale in array
```

Out[101…
```
11
```

In [102…
```
a2.min() # return the min vale in array
```

Out[102…
```
1
```

In [103…
```
a2.mean()
```

Out[103…
```
5.94
```

# indexing

In [109…
```
math=np.arange(0,100).reshape(0b1010,0b1010)
math
```

Out[109…
```
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 [110…
```
row=4
col=5
```

In [112…
```
col
```

Out[112…
```
5
```

In [114…
```
row
```

Out[114…
```
4
```

In [115…
```
math
```

Out[115…
```
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 [118…
```
math[row,col]  # return values
```

Out[118…
```
45
```

In [119…
```
math[:]
```

Out[119…
```
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 [121…

```
math[:,5]
#This is NumPy slicing, using the form:

#       array[rows, columns]
#                 : → All rows (from row 0 to row 9)

#                 5 → Only the 6th column (since indexing starts from 0)

#                 So it returns the values in column index 5 from every row
```

Out[121…

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

## 🔁 Summary:

| Row | Column 5 Value |
|---|---|
| 0 | 5 |
| 1 | 15 |
| 2 | 25 |
| … | … |
| 9 | 95 |

In [123…

```
math[row,:]

#            🔍 What happens:
#             math is a 10×10 matrix.

#           row = 2 means you access the 3rd row (since indexing starts at 0).

#           : selects all columns in that row.
```

Out[123…

```
array([40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

In [124…

```
math[:col]
```

Out[124…

```
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 [126…

```
math[row:]
```

Out[126…
```
array([[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 [127…
```
math[:,8]
```

Out[127…
```
array([ 8, 18, 28, 38, 48, 58, 68, 78, 88, 98])
```

In [130…
```
math[:,-1] # negative index
```

Out[130…
```
array([ 9, 19, 29, 39, 49, 59, 69, 79, 89, 99])
```

## ✅ Summary:

- `:` → all rows
- `-1` → last column
- So this gives the **last column values** from top to bottom

---

Let me know if you want:

- The **last row** → `math[-1, :]`
- All but last column → `math[:, :-1]`
- Submatrix or conditional slicing!

In [131…
```
math[:]
```

Out[131…
```
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 [132…
```
math[3:-3]
```

Out[132…
```
array([[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]])
```

In [133…
```
math[0]
```

Out[133…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [134…
```
math[5:7]
```

Out[134…
```
array([[50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69]])
```

In [135…
```
math[:10:3]
```

Out[135…
```
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 [138…
```
math[::-1]  # reverse
```

Out[138…
```
array([[90, 91, 92, 93, 94, 95, 96, 97, 98, 99],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9]])
```

In [139…
```
math[2:6]
```

Out[139…
```
array([[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]])
```

In [140…
```
math
```

Out[140…
```
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 [142…
```
math[2:6,2:5]
#  Part     Meaning
#  2:6      ( rows)     Rows from index 2 up to (not including) 6 → rows 2, 3, 4
#  2:5      (columns)   Columns from index 2 up to (not including) 5 → columns 2
```

Out[142…

```
array([[22, 23, 24],
       [32, 33, 34],
       [42, 43, 44],
       [52, 53, 54]])
```

Matrix `math` (10×10) looks like:

less Copy Edit

```less
Row 2: [20 21 22 23 24 25 26 27 28 29]
Row 3: [30 31 32 33 34 35 36 37 38 39]
Row 4: [40 41 42 43 44 45 46 47 48 49]
Row 5: [50 51 52 53 54 55 56 57 58 59]
```

### 📃 Output: values at rows 2–5 and columns 2–4

lua Copy Edit

```lua
[[22 23 24]
 [32 33 34]
 [42 43 44]
 [52 53 54]]
```

# Masking

**Masking in NumPy means selecting elements of an array using a Boolean condition. It's super useful for filtering data**.

## 📌 Basic Idea:

You create a **Boolean mask**:

python Copy Edit

```python
mask = (array > value)
```

Then use it to filter the array:

python Copy Edit

```python
filtered = array[mask]
```

## 🧠 Common Use Cases:

| Condition | Usage Example |
|---|---|
| Equal to | `a[a == 5]` |
| Not equal | `a[a != 0]` |
| Greater than | `a[a > 10]` |
| In a range | `a[(a > 10) & (a < 20)]` |
| Replace values | `a[a < 5] = 0` |

In [143…

```python
math
```

Out[143…

```
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 [144…

```python
id(math)
```

Out[144…

```
2067705247184
```

In [145…

```python
math<50
```

Out[145…

```
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 [146…

```
math>50
```

Out[146…

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

In [147…

```
math==50
```

Out[147…

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

In [149…

```
math[math==50]
#    🔍 Explanation:
#    This is an example of masking (boolean indexing) in NumPy.

#    math == 50 creates a Boolean array of the same shape as math:

#    True where the element is 50

#    False everywhere else

#     math[math == 50] returns only the elements equal to 50
```

Out[149…

```
array([50])
```

In [151…

```
math[math!=50]

#Returns all elements not equal to 50
#✅ Output: all values except 50
```

Out[151…

```
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 [152…

```
math[math>=50]
# Returns all elements greater than or equal to 50

# ✅ Output:
```

Out[152…

```
array([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 [156…

```
math[math<=50]
#Returns all elements less than or equal to 50
```

```
# ✅ Output:
```

Out[156…

```
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 [160…

```
math[math>50]
#Returns all elements strictly greater than 50

# ✅ Output:
```

Out[160…

```
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 [ ]: