```
# NUMPY DETAILS
```

Importing numpy

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

Creating array

In [5]:
```
arr=np.array([1,2,3,4])
```

Zeros and ones array:

In [13]:
```
import numpy as np
```

In [14]:
```
zeros_arr=np.zeros(5)
zeros_arr
```

Out[14]:
```
array([0., 0., 0., 0., 0.])
```

In [15]:
```
zeros_arr=np.ones(5)
zeros_arr
```

Out[15]:
```
array([1., 1., 1., 1., 1.])
```

In [ ]:
```

```

In [26]:
```
range_arr=np.arange(2,5)
range_arr
```

Out[26]:
```
array([2, 3, 4])
```

In [17]:
```
range_arr
```

Out[17]:
```
array([ 1,  6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81,
       86, 91, 96])
```

In [34]:
```
a=np.arange(10,40).reshape(10,5)
print(a)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[34], line 1
----> 1 a=np.arange(10,40).reshape(10,5)
      2 print(a)

ValueError: cannot reshape array of size 30 into shape (10,5)
```

In [35]:
```
b=np.random.rand(5,5)
b
```

Out[35]:
```
array([[0.18253452, 0.82180475, 0.30950094, 0.78756264, 0.12034803],
       [0.00327529, 0.80055672, 0.43246036, 0.81745446, 0.72230395],
       [0.03089845, 0.98628993, 0.23352284, 0.07935794, 0.53436447],
       [0.28157702, 0.61329011, 0.0778637 , 0.69405657, 0.24697999],
       [0.94476274, 0.00168918, 0.45710465, 0.62683509, 0.98986317]])
```

In [40]:
```
import numpy as np
c=np.random.randint(5,40,(4,3))
c
```

Out[40]:
```
array([[11, 29, 18],
       [37,  6, 26],
       [26, 18,  5],
       [16, 12, 11]], dtype=int32)
```

Array Operations

Basic arithmetic operations:

In [43]:
```
arr1=np.arange(1,10)
arr1
```

Out[43]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [46]:
```
arr2=np.arange(11,20)
arr2
```

Out[46]:
```
array([11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [48]:
```
Result=arr1+arr2
Result
```

Out[48]:
```
array([12, 14, 16, 18, 20, 22, 24, 26, 28])
```

In [49]:
```
result=arr1-arr2
result
```

Out[49]:
```
array([-10, -10, -10, -10, -10, -10, -10, -10, -10])
```

In [50]:
```
result=arr1*arr2
result
```

Out[50]:
```
array([ 11,  24,  39,  56,  75,  96, 119, 144, 171])
```

In [51]:
```
result=arr1/arr2
result
```

Out[51]:
```
array([0.09090909, 0.16666667, 0.23076923, 0.28571429, 0.33333333,
       0.375     , 0.41176471, 0.44444444, 0.47368421])
```

In [52]:
```
result=arr1%arr2
result
```

Out[52]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [53]:
```
result=arr1//arr2
result
```

Out[53]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Element-wise operations:

In [55]:
```
result=np.square(arr1)
result
```

Out[55]:
```
array([ 1,  4,  9, 16, 25, 36, 49, 64, 81])
```

In [56]:
```
res=np.sqrt(arr1)
res
```

Out[56]:
```
array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974, 2.64575131, 2.82842712, 3.        ])
```

In [57]:
```
res=np.exp(arr1)
res
```

Out[57]:
```
array([2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01,
       1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
       8.10308393e+03])
```

Dot product:

In [58]:
```
dp=np.dot(arr1,arr2)
dp
```

Out[58]:
```
np.int64(735)
```

In [ ]:
```
#the dot product is used to multiply two arrays in a way that combines correspon
```

In [ ]:
```

```

Array Manipulation

In [59]:
```
#Reshape:
```

In [69]:
```
ar=np.arange(10,100)
ar
```

Out[69]:
```
array([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 [73]:
```
arrs=ar.reshape(9,10)
```

```
arrs
```

Out[73]:
```
array([[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 [74]:
```
# Transpose:
function is used to flip a matrix over its diagonal, which means:

* Rows become columns

* Columns become rows
```

In [76]:
```
trans=ar.T
trans
```

Out[76]:
```
array([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 [82]:
```
array=np.zeros(5)
array
```

Out[82]:
```
array([0., 0., 0., 0., 0.])
```

In [83]:
```
array1=np.ones(5)
array
```

Out[83]:
```
array([0., 0., 0., 0., 0.])
```

In [84]:
```
tarns=array1.T
trans
```

Out[84]:
```
array([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 [90]:
```
x=np.arange(10,20,2)
x
```

Out[90]:
```
array([10, 12, 14, 16, 18])
```

In [86]:
```
transpose=x.T
transpose
```

Out[86]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [91]:
```
import numpy as np

mat = np.array([[1, 2, 3],
                [4, 5, 6]])

print("Original Matrix:\n", mat)
print("Transposed Matrix:\n", mat.T)
```

```
Original Matrix:
 [[1 2 3]
 [4 5 6]]
Transposed Matrix:
 [[1 4]
 [2 5]
 [3 6]]
```

In [92]:
```
# flatten:
```

In [94]:
```
flatened_array=arr1.flatten()
flatened_array
```

Out[94]:
```
array([1., 1., 1., 1., 1.])
```

In [97]:
```
flatn_arr=ar.flatten()
flatn_arr
```

Out[97]:
```
array([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 [98]:
```
x=np.random.randint(10,100,(10,10))
x
```

Out[98]:
```
array([[81, 72, 18, 90, 58, 64, 55, 13, 95, 31],
       [33, 22, 71, 45, 91, 88, 93, 49, 36, 83],
       [18, 16, 25, 54, 38, 20, 97, 30, 34, 44],
       [66, 30, 78, 67, 21, 13, 68, 88, 84, 24],
       [43, 78, 64, 87, 68, 71, 20, 74, 20, 91],
       [70, 79, 69, 50, 84, 49, 94, 22, 24, 31],
       [37, 91, 66, 50, 71, 80, 68, 85, 66, 19],
       [30, 87, 15, 39, 47, 52, 77, 29, 73, 26],
       [15, 57, 34, 17, 56, 22, 87, 27, 25, 60],
       [98, 70, 19, 21, 45, 16, 39, 64, 93, 43]], dtype=int32)
```

In [99]:
```
flat=x.flatten()
flat
```

Out[99]:
```
array([81, 72, 18, 90, 58, 64, 55, 13, 95, 31, 33, 22, 71, 45, 91, 88, 93,
       49, 36, 83, 18, 16, 25, 54, 38, 20, 97, 30, 34, 44, 66, 30, 78, 67,
       21, 13, 68, 88, 84, 24, 43, 78, 64, 87, 68, 71, 20, 74, 20, 91, 70,
       79, 69, 50, 84, 49, 94, 22, 24, 31, 37, 91, 66, 50, 71, 80, 68, 85,
       66, 19, 30, 87, 15, 39, 47, 52, 77, 29, 73, 26, 15, 57, 34, 17, 56,
       22, 87, 27, 25, 60, 98, 70, 19, 21, 45, 16, 39, 64, 93, 43],
      dtype=int32)
```

In [103…
```
y=np.ones(20,dtype=int)
y
```

Out[103…
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [109…
```
import numpy as np

mat = np.array([[1, 2, 3],
                [4, 5, 6]])

print("Original Matrix:\n", mat)
```

```
Original Matrix:
 [[1 2 3]
 [4 5 6]]
```

In [108…
```
flat2=mat.flatten()
flat2
```

Out[108…
```
array([1, 2, 3, 4, 5, 6])
```

In [122…
```
# indexing,slicing
```

In [128…
```
a1=np.array([2,4,54,3,26,5,6])
a1
```

Out[128…
```
array([ 2,  4, 54,  3, 26,  5,  6])
```

In [129…
```
a1[2]
```

Out[129…
```
np.int64(54)
```

In [130…
```
a1[::2]
```

Out[130…
```
array([ 2, 54, 26,  6])
```

In [131…
```
a1[:5]
```

Out[131…
```
array([ 2,  4, 54,  3, 26])
```

In [105…
```
# Statistical Operations
```

In [106…
```
#mean,median,standard deviation:
```

In [107…
```
import numpy as np

mat = np.array([[1, 2, 3],
                [4, 5, 6]])

print("Original Matrix:\n", mat)
```

```
Original Matrix:
 [[1 2 3]
 [4 5 6]]
```

In [121…
```
import numpy as np
x=np.array([10,20,30,50])
```

```
y=np.mean(x)
q=np.median(x)
w=np.std(x)
print(y)
print(w)
print(q)
```

```
27.5
14.79019945774904
25.0
```

In [ ]:

In [111…
```
mean_val=np.mean(mat)
mean_val
```

Out[111…
```
np.float64(3.5)
```

In [112…
```
median=np.median(mat)
median
```

Out[112…
```
np.float64(3.5)
```

In [113…
```
std_dev=np.std(mat)
std_dev
```

Out[113…
```
np.float64(1.707825127659933)
```

Sum,min,max:

In [3]:
```
import numpy as np
```

In [5]:
```
arr=[23,1,2,4,5,8]
```

In [6]:
```
total_sum=np.sum(arr)
total_sum
```

Out[6]:
```
np.int64(43)
```

In [7]:
```
min=np.min(arr)
min
```

Out[7]:
```
np.int64(1)
```

In [8]:
```
max=np.max(arr)
max
```

Out[8]:
```
np.int64(23)
```

INDEXING AND SLICING

indexing:

In [13]:
```
arr[3]
```

Out[13]:
```
4
```

In [23]:
```
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
```

In [24]:
```
matrix
```

Out[24]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [25]:
```
print(matrix[0, 0])   # First row, first column → 1
print(matrix[2, 1])
```

```
1
8
```

Slicing:

In [26]:
```
matrix[0:2,1:3]
```

Out[26]:
```
array([[2, 3],
       [5, 6]])
```

In [27]:
```
matrix[:1]
```

Out[27]:
```
array([[1, 2, 3]])
```

In [28]:
```
matrix[1:]
```

Out[28]:
```
array([[4, 5, 6],
       [7, 8, 9]])
```

In [29]:
```
matrix[:]
```

Out[29]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [30]:
```
matrix[0:-1]

Row slicing – take rows starting at index 0 (first row)

Stop before index -1 (which means the last row).

So it gives all rows except the last one.
```

Out[30]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

Logical operator

In [31]:
```
arr=[23,1,2,4,5,8]
```

In [33]:
```
arr[2]>3
```

Out[33]:
```
False
```

In [34]:
```
import numpy as np

a = np.array([True, False, True])
b = np.array([True, True, False])

print(np.logical_and(a, b))  # [ True False False ]
print(np.logical_or(a, b))   # [ True  True  True ]
print(np.logical_not(a))     # [False  True False]
print(np.logical_xor(a, b))  # [False  True  True]
```

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

In [35]:
```
arr = np.array([10, 20, 30, 40, 50])

# Numbers greater than 20 and less than 50
print((arr > 20) & (arr < 50))


# Numbers less than 15 or greater than 40
print((arr < 15) | (arr > 40))
```

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

In [36]:
```
arr=[23,1,2,4,5,8]
```

In [38]:
```
arr[0]>30
```

Out[38]:
```
False
```

In [39]:
```
arr[0]==23
```

Out[39]:
```
True
```

BROADCASTING

In [46]:
```
import numpy as np

a = np.array([1, 2, 3])
b = 5

print(a + b)
```

```
[6 7 8]
```

Here: a has shape (3,) b has shape () (scalar) NumPy broadcasts b so it acts like [5, 5, 5] without actually creating it.

In [ ]:
```
# Broadcasting with Different Shapes:
```

In [48]:
```
A = np.array([[1, 2, 3],
              [4, 5, 6]])   # Shape (2, 3)

B = np.array([10, 20, 30])  # Shape (3,)

print(A + B)
```

```
[[11 22 33]
 [14 25 36]]
```

here: Row vector B is broadcasted to match each row of A.

broadcasting rules:

NumPy compares dimensions from right to left: If dimensions are equal, they're compatible. If one of them is 1, it is stretched to match the other. If they're different and neither is 1, broadcasting fails.

In [ ]:
```
Example of incompatible shapes:
```

In [53]:
```
X = np.ones((3, 2))
Y = np.ones((3, 3))
res=X+Y
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[53], line 3
      1 X = np.ones((3, 2))
      2 Y = np.ones((3, 3))
----> 3 res=X+Y

ValueError: operands could not be broadcast together with shapes (3,2) (3,3)
```

In [ ]:
```
Higher-Dimensional Example
```

In [54]:
```
A = np.array([[[1], [2], [3]]])  # Shape (1, 3, 1)
B = np.array([[10, 20]])         # Shape (1, 2)

result = A + B
print(result.shape)  # (1, 3, 2)
print(result)
```

```
(1, 3, 2)
[[[11 21]
  [12 22]
  [13 23]]]
```

CONCATENATING:

concatenation means joining two or more arrays into a single array. It works for 1D, 2D, and higher-dimensional arrays as long as their shapes are compatible along the chosen axis.

In [64]:
```
import numpy as np

a=np.array([20,50,60,30])
b=np.array([40,90,50,10])
```

```
concatenate=np.concatenate((a,b))
print(concatenate)
```

```
[20 50 60 30 40 90 50 10]
```

In [ ]:

In [65]:
```
A = np.array([[1, 2],
              [3, 4]])

B = np.array([[5, 6]])

# Concatenate vertically (axis=0 → rows)
print(np.concatenate((A, B), axis=0))
# Output:
# [[1 2]
#  [3 4]
#  [5 6]]

# Concatenate horizontally (axis=1 → columns)
C = np.array([[7],
              [8]])
print(np.concatenate((A, C), axis=1))
```

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

Shortcuts for Concatenation:

STACKING

- np.vstack() → vertical stack (rows)
- np.hstack() → horizontal stack (columns)
- np.column_stack() → stack 1D arrays as columns
- np.row_stack() → stack 1D arrays as rows

In [66]:
```
x = np.array([1, 2])
y = np.array([3, 4])

print(np.vstack((x, y)))
# [[1 2]
#  [3 4]]

print(np.hstack((x, y)))
```

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

Important Rules

Arrays must have the same shape except along the axis you're concatenating.

For 2D arrays:

axis=0 → join rows (same number of columns required)

axis=1 → join columns (same number of rows required)

In [ ]:

LINEAR ALGEBRA

In NumPy, linear algebra operations are handled mainly by the numpy.linalg module (and a few functions outside it). It allows you to work with vectors and matrices just like in mathematics — things like dot products, determinants, inverses, eigenvalues, and solving systems of equations.

Function Purpose np.dot(a, b) / a @ b Matrix multiplication / dot product np.transpose(a) / a.T Transpose a matrix np.linalg.inv(a) Inverse of a square matrix np.linalg.det(a) Determinant of a square matrix np.linalg.matrix_rank(a) Rank of a matrix np.linalg.eig(a) Eigenvalues & eigenvectors np.linalg.norm(a) Vector or matrix norm np.linalg.solve(A, b) Solve a system of equations Ax=b

Determinant:

In [68]:
```
det = np.linalg.det(A)
print(det)
```

```
-2.0000000000000004
```

Inverse:

In [70]:
```
inv_A = np.linalg.inv(A)
print(inv_A)
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Eigenvalues & Eigenvectors:

In [71]:
```
vals, vecs = np.linalg.eig(A)
print("Eigenvalues:", vals)
print("Eigenvectors:\n", vecs)
```

```
Eigenvalues: [-0.37228132  5.37228132]
Eigenvectors:
 [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

Solve System of Equations

Example: Solve

A = np.array([[2, 1], [3, 4]]) b = np.array([8, 18])

x = np.linalg.solve(A, b) print(x)

In [ ]:

```

```

RANDOM SAMPLING

Random sampling is handled mainly by the numpy.random module. It allows you to generate random numbers, pick random samples, shuffle arrays, and simulate probability distributions.Function Purpose np.random.rand() Uniform floats [0, 1) np.random.randint() Random integers np.random.choice() Random sample from array np.random.shuffle() Shuffle array in place np.random.normal() Gaussian distribution np.random.seed() Fix randomness

Random Numbers:

Uniform Distribution (default: between 0 and 1)

In [73]:

```python
import numpy as np

print(np.random.rand(3))      # 1D array of 3 random floats [0,1)
print(np.random.rand(2, 3))   # 2D array 2x3
```

```
[0.02516752 0.16834262 0.50841861]
[[0.55737079 0.33228072 0.85247444]
 [0.79478035 0.24081971 0.74242464]]
```

Random Integers:

In [75]:

```python
print(np.random.randint(1, 10, size=5))  # 5 integers from 1 to 9
print(np.random.randint(0, 2, size=(3, 3)))  # Random 0 or 1 in 3x3
```

```
[8 7 2 1 4]
[[0 1 1]
 [0 0 1]
 [0 0 0]]
```

Random Sampling from Existing Data:

In [77]:

```python
arr = np.array([10, 20, 30, 40, 50])

# Choose 3 elements with replacement
print(np.random.choice(arr, size=3))

# Choose 3 elements without replacement
print(np.random.choice(arr, size=3, replace=False))
```

```
[40 40 20]
[10 40 30]
```

Shuffle Data:

In [79]:
```
arr = np.array([1, 2, 3, 4, 5])
np.random.shuffle(arr)
print(arr)  # Randomly rearranged in place
```

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

Normal (Gaussian) Distribution:

In [81]:
```
# mean=0, std=1, 1D array of 5 numbers
print(np.random.normal(0, 1, 5))

# 2D array from normal distribution
print(np.random.normal(5, 2, (2, 3)))  # mean=5, std=2
```

```
[-1.21106855 -0.45801328 -0.30242568 -1.56815099  1.82337417]
[[5.48159172 4.22025441 3.51701805]
 [7.12378063 4.62892134 2.50293566]]
```

Seed for Reproducibility:

In [83]:
```
np.random.seed(42)
print(np.random.randint(1, 10, size=5))
```

```
[7 4 8 5 7]
```

tips:

Use np.random.choice() when working with NumPy arrays, large datasets, or need weighted sampling.

Use random.sample() for small, simple lists in pure Python.

In [ ]:

In [84]:
```
# COPY
```

In [86]:
```
arr1=[3,4,5,2,1,5,6,9]
new_arr=arr1.copy()
new_arr
```

Out[86]:
```
[3, 4, 5, 2, 1, 5, 6, 9]
```

In [ ]:

In [87]:
```
# handling Nan
```

In [88]:
```
has_nan=np.isnan(arr1).any() # having no nan value in the array
has_nan
```

Out[88]:
```
np.False_
```

In [89]:
```
has_nan=np.isnan(arr1)
has_nan
```

Out[89]:
```
array([False, False, False, False, False, False, False, False])
```

In [ ]:

In [90]:
```
# VECTORIZED OPERATIONS:
```

In [91]:
```
vec=np.sin(arr1)
vec
```

Out[91]:
```
array([ 0.14112001, -0.7568025 , -0.95892427,  0.90929743,  0.84147098,
       -0.95892427, -0.2794155 ,  0.41211849])
```

In [92]:
```
vec1=np.cos(arr1)
vec1
```

Out[92]:
```
array([-0.9899925 , -0.65364362,  0.28366219, -0.41614684,  0.54030231,
        0.28366219,  0.96017029, -0.91113026])
```

In [ ]:

save and load:

Saves in NumPy's native binary format → fast, keeps dtype and shape.

Only one array per .npy file.

In [95]:
```
np.save("my_array.npy",arr1)
```

In [101…
```
loaded_arr1=np.load('my_array.npy')
loaded_arr1
```

Out[101…
```
array([3, 4, 5, 2, 1, 5, 6, 9])
```

Save Multiple Arrays in One File (.npz):

In [103…
```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# Save multiple arrays
np.savez('my_arrays.npz', first=a, second=b)

# Load
data = np.load('my_arrays.npz')
```

```
print(data['first'])   # [1 2 3]
print(data['second'])  # [4 5 6]
```

```
[1 2 3]
[4 5 6]
```

Save as Text (.txt or .csv):

In [105…

```
np.savetxt('my_array.txt', arr)

# Load
loaded_txt = np.loadtxt('my_array.txt')
print(loaded_txt)
```

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

MEMORY USAGE

In [108…

```
new_array=np.array([2,3,4,1,2,9])
size_in_bytes=new_array.nbytes
size_in_bytes
```

Out[108… 48

In [ ]: