1. Array Creation Functions

```
import numpy as np
```

```
# Create an array from a list
a = np.array([1, 2, 3])
print("Array a:", a)
```

```
Array a: [1 2 3]
```

linspace() generates linearly spaced values, while arange() generates values in steps.

```
# Create an array with evenly spaced values
b = np.arange(0, 10, 2)  # Values from 0 to 10 with step 2
print("Array b:", b)
```

```
Array b: [0 2 4 6 8]
```

Archana Kapu
12:46 PM Today

linspace() generates linearly spaced values, while arange() generates values in steps.

```
# Create an array with linearly spaced values
c = np.linspace(0, 1, 5)  # 5 values evenly spaced between 0 and 1
print("Array c:", c)
```

```
Array c: [0.   0.25 0.5  0.75 1.  ]
```

Archana Kapu
12:46 PM Today

linspace() generates linearly spaced values, while arange() generates values in steps.

```
# Create an array filled with zeros
d = np.zeros((2, 3))  # 2x3 array of zeros
print("Array d:\n", d)
```

```
Array d:
 [[0. 0. 0.]
 [0. 0. 0.]]
```

```
t=np.zeros((3,3))
print(t)
```

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

```
# Create an array filled with ones
e = np.ones((3, 2))  # 3x2 array of ones
print("Array e:\n", e)
```

```
Array e:
 [[1. 1.]
 [1. 1.]
 [1. 1.]]
```

```
# Create an identity matrix
f = np.eye(4)  # 4x4 identity matrix
print("Identity matrix f:\n", f)
```

```
Identity matrix f:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Archana Kapu
1:00 PM Today

all values in matrix are zero's expect kth value and its diagonal values with k value

```
r=np.eye(4,k=1)
print("identy matrix with 4x4 matrix with diagonal value 1st pos:\n",r)
```

```
identy matrix with 4x4 matrix with diagonal value 1stpos:
 [[0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]]
```

Archana Kapu
12:59 PM Today

Returns
I : ndarray of shape (N,M)
An array where all elements are equal to zero, except for the k-th diagonal, whose values are equal to one.

See Also
identity : (almost) equivalent function
diag : diagonal 2-D array from a 1-D array specified by the user.

Examples

```
#(practise)
r=np.eye(5,k=2,dtype=float)
print("identity matrix with 5x5 matrix with diagonal value 2nd pos:\n",r)
```

```
identity matrix with 5x5 matrix with diagonal value 2nd pos:
 [[0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

>>> np.eye(2, dtype=int)
array([[1, 0],
[0, 1]])
>>> np.eye(3, k=1)
array([[0., 1., 0.],
[0., 0., 1.],

Archana Kapu
1:02 PM Today

here in the position of kth row and its diagonal values will be 1,with datatype values as float

2. Array Manipulation Functions

```
# Reshape an array
a1 = np.array([1, 2, 3])
reshaped = np.reshape(a1, (1, 3))  # Reshape to 1x3
print("Reshaped array:", reshaped)
```

```
Reshaped array: [[1 2 3]]
```

```
#reshape of an arary--practise1
a2=np.array([1,2,3,4,5,6,7,8,9,10])
resha2=np.reshape(a2,(5,2))#reshape to 5X2 matrix
print("reshape into 5x2 matrix:\n",resha2)
```

```
reshape into 5x2 matrix:
 [[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]]
```

```
# Flatten an array
f1 = np.array([[1, 2], [3, 4]])
flattened = np.ravel(f1)  # Flatten to 1D array
print("Flattened array:", flattened)
```

```
Flattened array: [1 2 3 4]
```

```
#prints 1d array,(practise)
f2=[[1,2],[3,4],[5,6],[7,8],[9,10]]
flatenedf2=np.ravel(f2)
print("flattened array:\n",flatenedf2)
```

Archana Kapu
1:36 PM Today
(edited 1:37 PM Today)

ravel() is used for flattended the array,ravel method prints 1d-array

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

```
# Transpose an array
e1 = np.array([[1, 2], [3, 4]])
transposed = np.transpose(e1)  # Transpose the array
print("Transposed array:\n", transposed)
```

Archana Kapu
3:38 PM Today

moving the rows data to the column and columns data to the rows

```
Transposed array:
 [[1 3]
 [2 4]]
```

```
#tarnspose array ex2(practise)
e2=np.array([[5,6],[7,8]])
e2transposed=np.transpose(e2)
print("transposed values as output:\n",e2transposed)
```

```
transposed values as output:
 [[5 7]
 [6 8]]
```

```
# Stack arrays vertically
a2 = np.array([1, 2])
b2 = np.array([3, 4])
stacked = np.vstack([a2, b2])  # Stack a and b vertically
print("Stacked arrays:\n", stacked)
```

```
Stacked arrays:
 [[1 2]
 [3 4]]
```

```
#stacked array ex2(practise)
q1=np.array([5,6])
q2=np.array([7,8])
stacked=np.vstack([q1])
print("stacked array:\n",stacked)
```

Archana Kapu
5:47 PM Today

by passing only 1 argument

```
stacked array:
 [[5 6]]
```

```
#stacked array ex2(practise)
q1=np.array([5,6])
q2=np.array([7,8])
stacked=np.vstack([q1,q2])
print("stacked array:\n",stacked)
```

```
stacked array:
 [[5 6]
 [7 8]]
```

3. Mathematical Functions

```
# Add two arrays
g = np.array([1, 2, 3, 4])
added = np.add(g, 2)  # Add 2 to each element
print("Added 2 to g:", added)
```

```
Added 2 to g: [3 4 5 6]
```

```
# Add two arrays ex2(practise)
g1 = np.array([5, 6, 7, 8])
added = np.add(g1, 10)  # Add 2 to each element
print("Added 10 to g1:", added)
```

```
Added 10 to g1: [15 16 17 18]
```

```
g
```

```
array([1, 2, 3, 4])
```

```
# Square each element
squared = np.power(g, 2)  # Square each element
print("Squared g:", squared)
```

```
Squared g: [ 1  4  9 16]
```

```
# Square each element--ex2(practise)
practicepow=np.power(g,3)
print("squared values:\n",practicepow)
```

```
squared values:
 [ 1  8 27 64]
```

```
g
```

```
array([1, 2, 3, 4])
```

```
# Square root of each element
sqrt_val = np.sqrt(g)  # Square root of each element
print("Square root of g:", sqrt_val)
```

```
Square root of g: [1.         1.41421356 1.73205081 2.        ]
```

```
#sqrt ex practise2
y=np.array([4,9,16,25])
sqr_rt=np.sqrt(y)
print("squarroot of y values:\n",sqr_rt)
```

```
squarroot of y values:
 [2. 3. 4. 5.]
```

```
print(a1)
print(g)
```

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

```
# Dot product of two arrays
a2 = np.array([1, 2, 3])
dot_product = np.dot(a2, g)  # Dot product of a and g
print("Dot product of a and g:", dot_product)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-43-fce3ab80c017> in <cell line: 0>()
      1 # Dot product of two arrays
      2 a2 = np.array([1, 2, 3])
----> 3 dot_product = np.dot(a2, g)  # Dot product of a and g
      4 print("Dot product of a and g:", dot_product)

ValueError: shapes (3,) and (4,) not aligned: 3 (dim 0) != 4 (dim 0)
```

Next steps: Explain error

```
print(a)
print(a1)
```

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

The dot product is a mathematical operation that multiplies two vectors to produce a single number. It's also known as the scalar product or inner product

```
[46] print(a)
     print(a1)
```

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

$1\times1+2\times2+3\times3$
$=1+4+9$
$=14$

Archana Kapu
4:05 PM Today
The dot product is a mathematical operation that multiplies two vectors to produce a single number. It's also known as the scalar product or inner product

```
a3 = np.array([1, 2, 3])
dot_product = np.dot(a1, a)  # Dot product of a and g
print("Dot product of a1 and a:", dot_product)
```

```
Dot product of a1 and a: 14
```

4. Statistical Functions

Archana Kapu
4:10 PM Today
1+2+3+4/4=2.5

```
s = np.array([1, 2, 3, 4])
mean = np.mean(s)
print("Mean of s:", mean)
```

```
Mean of s: 2.5
```

```
#mean caluculation
s1q = np.array([5, 6, 7, 8])
mean = np.mean(s1q)
print("Mean of s1q:", mean)
```

```
Mean of s1q: 6.5
```

```
s
```

```
array([1, 2, 3, 4])
```

Archana Kapu
4:24 PM Today
need clarification

```
# Standard deviation of an array
std_dev = np.std(s)
print("Standard deviation of s:", std_dev)
```

```
Standard deviation of s: 1.118033988749895
```

```
# Minimum element of an array
minimum = np.min(s)
print("Min of s:", minimum)
```

```
Min of s: 1
```

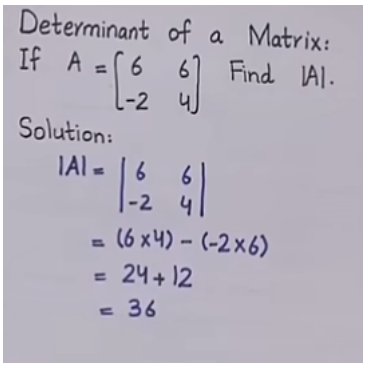
```
# Maximum element of an array
maximum = np.max(s)
print("Max of s:", maximum)
```

```
Max of s: 4
```

5. Linear Algebra Functions

```
# Create a matrix
matrix = np.array([[1, 2], [3, 4]])
```

how to find determinant of matrix



Archana Kapu
4:39 PM Today
(edited 4:40 PM Today)
|A|=1x4-6=-2

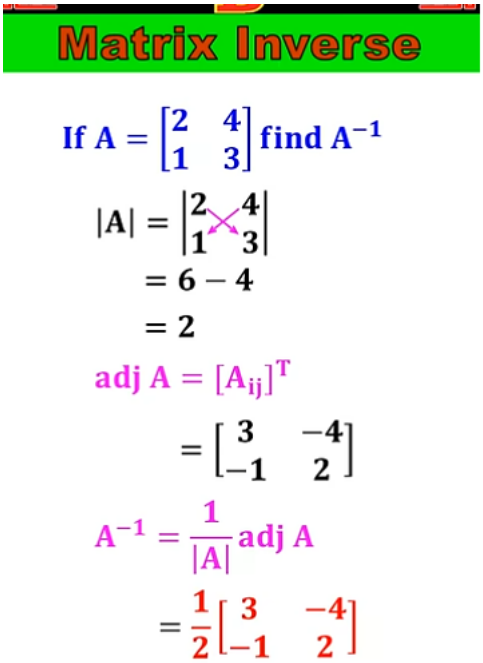
Archana Kapu
5:01 PM Today
ad-bc

```
# Determinant of a matrix
determinant = np.linalg.det(matrix)
print("Determinant of matrix:", determinant)
```

```
Determinant of matrix: -2.0000000000000004
```

Inverse matrix

The inverse of a matrix is obtained by dividing the adjugate(also called adjoint) of the given matrix by the determinant of the given matrix.

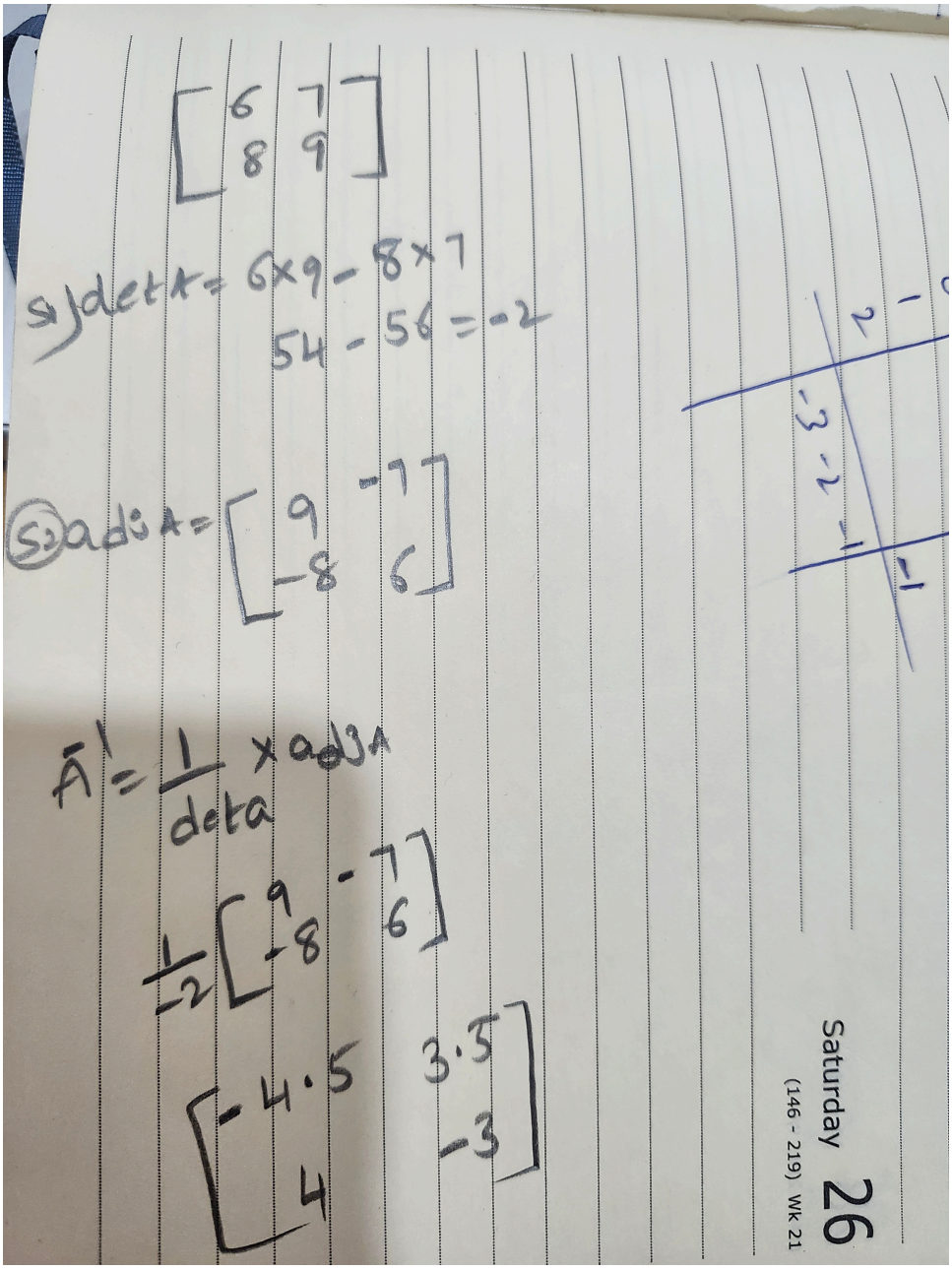


Archana Kapu
4:41 PM Today
1/|A|*adj(A)

```
# Inverse of a matrix
inverse = np.linalg.inv(matrix)
print("Inverse of matrix:\n", inverse)
```

```
Inverse of matrix:
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

```
#(practise)
s_arr=np.array([(6,7),(8,9)])
```




```
s_arr
```

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

```
#(practise)
determinant_a=np.linalg.det(s_arr)
print("determinant of matrix:\n",determinant_a)
```

```
determinant of matrix:
 -1.9999999999999998
```

```
#(practise)
invers_a=np.linalg.inv(s_arr)
print("inverse of an array:\n",invers_a)
```

```
inverse of an array:
 [[-4.5  3.5]
 [ 4.  -3. ]]
```

6. Random Sampling Functions

```
# Generate random values between 0 and 1
random_vals = np.random.rand(3)  # Array of 3 random values between 0 and 1
print("Random values:", random_vals)
```

```
Random values: [0.13716897 0.69904694 0.10186839]
```

```
# Set seed for reproducibility
np.random.seed(0)

# Generate random values between 0 and 1
random_vals = np.random.rand(3)  # Array of 3 random values between 0 and 1
print("Random values:", random_vals)
```

```
Random values: [0.5488135  0.71518937 0.60276338]
```

```
# Generate random integers
rand_ints = np.random.randint(0, 10, size=5)  # Random integers between 0 and 10
print("Random integers:", rand_ints)
```

```
Random integers: [8 1 6 7 7]
```

```
# Set seed for reproducibility
np.random.seed(0)

# Generate random integers
rand_ints = np.random.randint(0, 10, size=5)  # Random integers between 0 and 10
print("Random integers:", rand_ints)
```

```
Random integers: [5 0 3 3 7]
```

7. Boolean & Logical Functions

```
# Check if all elements are True
# all
logical_test = np.array([True, False, True])
all_true = np.all(logical_test)  # Check if all are True
print("All elements True:", all_true)
```

```
All elements True: False
```

```
# Check if all elements are True
logical_test = np.array([False, False, False])
all_true = np.all(logical_test)  # Check if all are True
print("All elements True:", all_true)
```

```
All elements True: False
```

```
# Check if all elements are True(practise)
# all
logical_prac = np.array([True, False, True])
all_true = np.all(logical_prac)  # Check if all are True
print("All elements True:", all_true)
```

```
All elements True: False
```

```
# Check if any elements are True
# any
any_true = np.any(logical_test)  # Check if any are True
print("Any elements True:", any_true)
```

```
Any elements True: True
```

```
# Check if any elements are True(practise)
# any
any_true = np.any(logical_prac)  # Check if any are True
print("Any elements True:", any_true)
```

```
Any elements True: True
```

8. Set Operations

```
# Intersection of two arrays
set_a = np.array([1, 2, 3, 4])
set_b = np.array([3, 4, 5, 6])
intersection = np.intersect1d(set_a, set_b)
print("Intersection of a and b:", intersection)
```

```
Intersection of a and b: [3 4]
```

```
# Union of two arrays
union = np.union1d(set_a, set_b)
print("Union of a and b:", union)
```

```
Union of a and b: [1 2 3 4 5 6]
```

```
# Intersection of two arrays(practise)
set_a1 = np.array([1, 2, 3, 4])
set_b1 = np.array([1, 4, 2, 6])
intersection = np.intersect1d(set_a1, set_b1)
print("Intersection of a1 and b1:", intersection)
```

```
Intersection of a1 and b1: [1 2 4]
```

```
# Union of two arrays(practise)
union = np.union1d(set_a1, set_b1)
print("Union of a1 and b1:", union)
```

```
Union of a1 and b1: [1 2 3 4 6]
```

9. Array Attribute Functions

```
# Array attributes
a = np.array([1, 2, 3])
shape = a.shape  # Shape of the array
size = a.size    # Number of elements
dimensions = a.ndim  # Number of dimensions
dtype = a.dtype   # Data type of the array

print("Shape of a:", shape)
print("Size of a:", size)
print("Number of dimensions of a:", dimensions)
print("Data type of a:", dtype)
```

```
Shape of a: (3,)
Size of a: 3
Number of dimensions of a: 1
Data type of a: int64
```

```
# Array attributes(practise)
s_arr1=np.array([(6,7),(8,9)])
```

```
shape_a=s_arr1.shape
size_a=s_arr1.size
dimen_a=s_arr1.ndim
dtype_a=s_arr1.dtype
print(" shape of s_arr1:\n",shape_a)
print("size of s_arr1:\n",size_a)
print("dimensions  of s_arr1:\n",dimen_a)
print("datatype of s_arr1:\n",dtype_a)
```

```
 shape of s_arr1:
 (2, 2)
size of s_arr1:
 4
dimensions  of s_arr1:
 2
datatype of s_arr1:
 int64
```

10. Other Functions

```
# Create a copy of an array
a = np.array([1, 2, 3])
copied_array = np.copy(a)  # Create a copy of array a
print("Copied array:", copied_array)
```

```
Copied array: [1 2 3]
```

```
# Create a copy of an array(practise)
b=np.array([14,5,6])
copiedarray1=np.copy(b)
print("copied array:",copiedarray1)
```

```
copied array: [14  5  6]
```

```
# Size in bytes of an array
array_size_in_bytes = a.nbytes  # Size in bytes
print("Size of a in bytes:", array_size_in_bytes)
```

```
Size of a in bytes: 24
```

```
# Size in bytes of an array(practse)
sizeofanarray_in_bytes=b.nbytes
print("size of an array in bytes:",sizeofanarray_in_bytes)
```

```
size of an array in bytes: 24
```

```
# Check if two arrays share memory
shared = np.shares_memory(a, copied_array)  # Check if arrays share memory
print("Do a and copied_array share memory?", shared)
```

```
Do a and copied_array share memory? False
```

```
b
```

```
array([14,  5,  6])
```

```
copiedarray1
```

```
array([14,  5,  6])
```

```
# Check if two arrays share memory(practise)
sharedornot = np.shares_memory(b,copiedarray1)
print("does b and copiedarray1 shared memory?",sharedornot)
```

```
does b and copiedarray1 shared memory? False
```

```
Start coding or generate with AI.
```