Creating Arrays

You can create arrays in NumPy using lists, built-in functions, or random data generators.

In [98]:
```
import numpy as np

# Create an array from a list
arr = np.array([1, 2, 3, 4, 5])
print("Array from list:", arr)
```

```
Array from list: [1 2 3 4 5]
```

In [8]:
```
# Create an array with zeros
zeros = np.zeros((2, 3))
print("Array of zeros:\n", zeros)
```

```
Array of zeros:
 [[0. 0. 0.]
 [0. 0. 0.]]
```

In [10]:
```
# Create an array with random values
random_arr = np.random.rand(3, 2)
print("Random Array:\n", random_arr)
```

```
Random Array:
 [[0.32650214 0.11341953]
 [0.29198711 0.55963978]
 [0.88383681 0.89218538]]
```

Array Indexing

Access specific elements using their indices.

In [12]:
```
arr = np.array([10, 20, 30, 40, 50])

# Access first element
print("First Element:", arr[0])
```

```
First Element: 10
```

In [14]:
```
# Access first element
print("First Element:", arr[0])
```

```
First Element: 10
```

In [16]:
```
# Access last element
print("Last Element:", arr[-1])
```

```
Last Element: 50
```

In [18]:
```
# Access 2D array element
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
print("Element at (1, 2):", arr_2d[1, 2])
```

```
Element at (1, 2): 6
```

Array Slicing

Retrieve subsets of an array using slicing.

In [21]:
```
arr = np.array([10, 20, 30, 40, 50])

# Slice from index 1 to 3
print("Sliced Array:", arr[1:4])
```

```
Sliced Array: [20 30 40]
```

In [23]:
```
# Slice from index 1 to 3
print("Sliced Array:", arr[1:4])
```

```
Sliced Array: [20 30 40]
```

In [25]:
```
# Slice with step
print("Every Other Element:", arr[::2])
```

```
Every Other Element: [10 30 50]
```

In [27]:
```
# 2D array slicing
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
print("First Row:", arr_2d[0, :])
print("First Column:", arr_2d[:, 0])
```

```
First Row: [1 2 3]
First Column: [1 4]
```

Data Types

Each NumPy array has a specific data type (dtype).

In [30]:
```
arr = np.array([1, 2, 3])
print("Data Type:", arr.dtype)
```

```
Data Type: int32
```

In [32]:
```
# Specify data type
float_arr = np.array([1, 2, 3], dtype='float')
print("Array with float type:", float_arr)
```

```
Array with float type: [1. 2. 3.]
```

Copy vs. View

Understand how copying or viewing an array impacts memory usage.

In [37]:
```
arr = np.array([10, 20, 30])

# View (linked to the original array)
view = arr.view()
```

```
view[0] = 99
print("Original Array after View Modification:", arr)
```

```
Original Array after View Modification: [99 20 30]
```

In [39]:
```
# Copy (independent of the original array)
copy = arr.copy()
copy[1] = 88
print("Original Array after Copy Modification:", arr)
```

```
Original Array after Copy Modification: [99 20 30]
```

Array Shape

The shape of an array indicates its dimensions.

In [42]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Shape of the array:", arr.shape)
```

```
Shape of the array: (2, 3)
```

In [44]:
```
# Create a 3D array
arr_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("Shape of 3D array:", arr_3d.shape)
```

```
Shape of 3D array: (2, 2, 2)
```

Reshaping Arrays

Change the shape of an array without altering its data.

In [47]:
```
arr = np.array([1, 2, 3, 4, 5, 6])

# Reshape to 2x3
reshaped = arr.reshape(2, 3)
print("Reshaped Array:", reshaped)
```

```
Reshaped Array: [[1 2 3]
 [4 5 6]]
```

Iterating Arrays

Iterate over each element, row, or column of an array.

In [50]:
```
arr = np.array([[1, 2], [3, 4]])

# Iterate over rows
for row in arr:
    print("Row:", row)
```

```
Row: [1 2]
Row: [3 4]
```

In [52]:
```
# Iterate over elements
for element in arr.flatten():
```

```
    print("Element:", element)
```

```
Element: 1
Element: 2
Element: 3
Element: 4
```

stacking Arrays

Combine multiple arrays into one.

In [55]:
```
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])

# Join horizontally
joined = np.hstack((arr1, arr2))
print("Horizontally Joined Array:", joined)
```

```
Horizontally Joined Array: [1 2 3 4]
```

In [57]:
```
# Join vertically
joined = np.vstack((arr1, arr2))
print("Vertically Joined Array:\n", joined)
```

```
Vertically Joined Array:
 [[1 2]
 [3 4]]
```

Splitting Arrays

Split an array into smaller subarrays.

In [60]:
```
arr = np.array([10, 20, 30, 40, 50, 60])

# Split into 3 parts
splitted = np.array_split(arr, 3)
print("Splitted Arrays:", splitted)
```

```
Splitted Arrays: [array([10, 20]), array([30, 40]), array([50, 60])]
```

Searching in Arrays

Find specific elements in an array

In [82]:
```
arr = np.array([10, 20, 30, 40])

# Find the index of 30
index = np.where(arr == 30)
print("Index of 30:", index)
```

```
Index of 30: (array([2], dtype=int64),)
```

In [76]:
```
# Find even numbers
even_indices = np.where(arr % 2 == 0)
```

```
print("Indices of Even Numbers:", even_indices)
```

```
Indices of Even Numbers: (array([0, 1, 2, 3], dtype=int64),)
```

Sorting Arrays

Sort elements in ascending order.

In [87]:
```
arr = np.array([30, 10, 20, 50])

# Sort array
sorted_arr = np.sort(arr)
print("Sorted Array:", sorted_arr)
```

```
Sorted Array: [10 20 30 50]
```

In [89]:
```
# Sort a 2D array
arr_2d = np.array([[3, 1], [2, 4]])
print("Sorted 2D Array:\n", np.sort(arr_2d, axis=1))
```

```
Sorted 2D Array:
 [[1 3]
 [2 4]]
```

Filtering Arrays

Extract elements that satisfy a condition.

In [92]:
```
arr = np.array([10, 20, 30, 40])

# Filter elements greater than 20
filtered = arr[arr > 20]
print("Filtered Array:", filtered)
```

```
Filtered Array: [30 40]
```

Measure of Central Tendency

Central tendency in statics are the numerical values that are used to represent mid value or central value a large collection of numerical data

In [102…
```
#return a random number with the step value
salary = np.random.randint(1000,10000,40)
salary
```

Out[102…
```
array([7097, 3584, 2738, 1719, 4317, 3700, 3659, 8926, 2442, 3099, 9091,
       4586, 1532, 4672, 6026, 5723, 2861, 9774, 4634, 4559, 9157, 2312,
       3905, 6647, 9083, 2285, 5277, 3494, 1526, 2365, 2087, 1403, 9131,
       8957, 4257, 2863, 9399, 8614, 2764, 9922])
```

In [104…
```
np.mean(salary) #return the average of the array elements
```

Out[104…
```
5004.675
```

In [106…
```
np.median(salary) #return the median of the array elements
```

Out[106…
```
4287.0
```

Dispersion

In [111…
```
#range
np.max(salary)-np.min(salary) #return the maximum and minimum of the array elements
```

Out[111…
```
8519
```

In [115…
```
#variance
np.var(salary) #return the variance of the array elements
```

Out[115…
```
7621033.019374999
```

In [117…
```
#standard deviation
np.std(salary) #compare the standard deviation along the specified axis
```

Out[117…
```
2760.621853745094
```

IQR

In [122…
```
Q1 = np.quantile(salary,0.25) #compute the q-th quantile of the data along the spec
Q1
```

Out[122…
```
2757.5
```

In [126…
```
Q2 = np.quantile(salary,0.50) #compute the q-th quantile of the data along the spec
Q2
```

Out[126…
```
4287.0
```

In [124…
```
Q3 = np.quantile(salary,0.75) #compute the q-th quantile of the data along the spec
Q3
```

Out[124…
```
7476.25
```

In [128…
```
IQR = Q3 - Q1
IQR
```

Out[128…
```
4718.75
```

PERCENTILES

In [133…
```
np.percentile(salary,25) #return the qth percentile(s) of the array elements
```

Out[133…
```
2757.5
```

In [135…
```
np.percentile(salary,50) #return the qth percentile(s) of the array elements
```

Out[135…
```
4287.0
```

In [137…
```
np.percentile(salary,75) #return the qth percentile(s) of the array elements
```

Out[137…
```
7476.25
```

relantionships

In [140…
```
import numpy as np
```

In [156…
```
exp = np.array([1,2,3,4,5])
sal = np.array([2000,5000,8000,10000,13000])
```

In [158…
```
np.cov(exp,sal)  #covariance indicates the level to which two variable vary togethe
```

Out[158…
```
array([[2.50e+00, 6.75e+03],
       [6.75e+03, 1.83e+07]])
```

In [160…
```
np.corrcoef(exp,sal)
```

Out[160…
```
array([[1.        , 0.99794872],
       [0.99794872, 1.        ]])
```

Finding outlier

Outliers are defined elements more than 1.5 interquartile ranges above the upper quartile 75% are below the lower quarter 25% these methods usefull when the data is A is not normally distributed

lower = max(min(data),q1-1.5*iqr)<*outlier upper = min(max(data),q2+1.5*iqr)>outlier

In [165…
```
import numpy as np
```

In [187…
```
def outlier(data):
    IQR = np.quantile(data,0.75) - np.quantile(data,0.25)
    lower = max(np.max(data),np.quantile(data,0.25)-1.5*IQR)
    upper = min(np.min(data),np.quantile(data,0.75)-1.5*IQR)
    return data[(data > upper) | (data < lower)]
```

In [185…
```
arr  = np.array([1,2,3,4,5,-100,500,32,566])
```

In [183…
```
outlier(arr)
```

Out[183…
```
array([   1,    2,    3,    4,    5, -100,  500,   32,  566])
```

N-DIMENSION ARRAY

Before creating

In [193…
```
arr  = np.array([1,3,6,79,21,5])
```

In [195…
```
arr.ndim
```

Out[195…
```
1
```

In [201…
```
arr  = np.array([1,3,6,79,21,5],ndmin = 4)
arr.ndim
```

Out[201…
```
4
```

In [203…
```
arr
```

Out[203…
```
array([[[[ 1,  3,  6, 79, 21,  5]]]])
```

after creating

In [210…
```
arr = np.arange(0,15)
arr
```

Out[210…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [212…
```
arr.ndim
```

Out[212…
```
1
```

In [214…
```
arr2 = arr.reshape((3,5))
```

In [216…
```
arr2.ndim
```

Out[216…
```
2
```

In [218…
```
arr = np.arange(0,12)
arr
```

Out[218…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

mathematical functions

In [221…
```
np.pi #convert a string or number to a flaoting point number
```

Out[221…
```
3.141592653589793
```

In [223…
```
np.e #convert a string or number to a floating - point number
```

Out[223…
```
2.718281828459045
```

In [225…
```
np.inf
```

Out[225…
```
inf
```

In [227…
```
np.nan #missing value
```

Out[227…
```
nan
```

In [229…
```
age = np.array([23,56,np.nan,56])
```

In [231…
```
np.mean(age) #return the average of the array element
```

Out[231…
```
nan
```

In [233…
```
np.nanmean(age) #return average of an array
```

Out[233…
```
45.0
```

In [235…
```
np.nanmin(age) #return minimum of an array
```

Out[235…
```
23.0
```

Linear Algebra

linear algebra of numpy offers various method to apply linear algebra on any numpy array

In [241…
```
a = np.array([[1,2],[2,-3]])
b = np.array([[5],[6]])
b
```

Out[241…
```
array([[5],
       [6]])
```

In [243…
```
np.linalg.solve(a,b)
```

Out[243…
```
array([[3.85714286],
       [0.57142857]])
```

2x+3y-10z = 23 10x+4y-9z = 1 3x+10y-z =100

In [246…
```
a = np.array([[2,3,10,],[10,4,-9],[3,10,-1]])
b = np.array([[23],[1],[100]])
```

In [248…
```
np.linalg.solve(a,b)
```

Out[248…
```
array([[-4.61438561],
       [11.36563437],
       [-0.18681319]])
```

In [250…
```
a = np.array([[1,2],[3,4]])
```

In [252…
```
np.linalg.det(a)
```

Out[252…
```
-2.0000000000000004
```

In [258…
```
x = np.array([[1,2],[3,5]])
y = np.array([[3],[6]])
np.dot(x,y)
```

Out[258…
```
array([[15],
       [39]])
```

trignometry

In [270…
```
angle = np.pi
```

In [272…
```
np.sin(angle)  #print sines of an array
```

Out[272…
```
1.2246467991473532e-16
```

In [274…
```
np.cos(angle)
```

Out[274…
```
-1.0
```

In [276…
```
np.tan(angle)
```

Out[276…
```
-1.2246467991473532e-16
```

In [278…
```
np.sinh(angle)
```

Out[278…
```
11.548739357257746
```

In [280…
```
a = np.array([1,2,3,4,5])
b = np.array([3,6,7,8,9])
```

In [282…
```
np.union1d(a,b) #combination of all unique element
```

Out[282…
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [286…
```
np.intersect1d(a,b)  #common element
```

Out[286…
```
array([3])
```

In [288…
```
np.setdiff1d(a,b)
```

Out[288…
```
array([1, 2, 4, 5])
```

In [ ]:
```

```

Array boardcasting

The term broadcasting refers to the ability of Numpy to treat Array with different dimensions during arithmetic operation

In [294…
```
import numpy as np
```

In [314…
```
a = np.arange(0,8).reshape(4,2)
a.shape
```

Out[314…
```
(4, 2)
```

In [310…
```
b= np.array([[2,3]])
b.shape
```

Out[310…
```
(1, 2)
```

In [316…
```
a + b
```

Out[316…
```
array([[ 2,  4],
       [ 4,  6],
       [ 6,  8],
       [ 8, 10]])
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: