NumPy

NumPy (Numerical Python) is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices.

Creating NumPy Arrays

```
import numpy as np

# Create a simple 1D array
arr = np.array([1, 2, 3])
print("1D Array:", arr)

# Create a 2D array
matrix = np.array([[1, 2], [3, 4]])
print("2D Array:\n", matrix)
```

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

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array. These are often used to represent a 3rd order tensor.

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
print("The array dimension is:", arr.ndim)
```

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

 [[1 2 3]
  [4 5 6]]]
The array dimension is: 3
```

Other Functions for creating an array

```
# arange(start, stop, step)
arr1 = np.arange(0, 10, 2)
print("arange:", arr1)

# linspace(start, stop, num)
arr2 = np.linspace(0, 1, 5) #It divides the interval [0, 1] into 4 equal segments.
print("linspace:", arr2) # starting from 0 to 1 (inclusive).
```

```
arange: [0 2 4 6 8]
linspace: [0.   0.25 0.5  0.75 1.  ]
```

- `arange()` is useful when you know the **step size**.
- `linspace()` is useful when you know how many **evenly spaced values** you want between a range.

```
print(np.zeros((2, 3)))       # 2x3 array of zeros
print(np.ones((2, 3)))        # 2x3 array of ones
print(np.full((2, 3), 7))     # 2x3 array filled with 7
print(np.eye(3))              # Identity matrix (3x3)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]]
[[7 7 7]
 [7 7 7]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Function	Purpose
np.zeros((m, n))	m×n array of **0s**
np.ones((m, n))	m×n array of **1s**
np.full((m, n), val)	m×n array of any **value**
np.eye(n)	n×n **identity matrix**

Array Indexing

Array indexing is the same as accessing an array element. You can access an array element by referring to its index number.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
print(arr[2])
```

```
1
3
```

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])
```

```
7
```

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr)
print('5th element on 2nd row: ', arr[1, 4])
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
5th element on 2nd row:  10
```

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr)
print(arr[0, 1, 2])
```

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

 [[ 7  8  9]
  [10 11 12]]]
6
```

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr)
print('Last element from 2nd dim: ', arr[1, -1])
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
Last element from 2nd dim:  10
```

Slicing Array

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step].

If we don't pass start, it’s considered 0

If we don't pass end, its considered length of array in that dimension

If we don't pass step, it’s considered 1

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5],"\n")
print(arr[4:])
print("\n")
print(arr[:4])
```

```
[2 3 4 5]

[5 6 7]


[1 2 3 4]
```

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

```
[5 6]
```

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

```
[2 4]
```

Slicing in 2D array

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

```
[7 8 9]
```

```
import numpy as np
matrix = np.array([[1, 2, 3], [3, 4, 5], [5, 6, 7]])
print("2D Array:\n", matrix)
matrix[0:3, 2 ]
```

```
2D Array:
 [[1 2 3]
 [3 4 5]
 [5 6 7]]
array([3, 5, 7])
```

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
```

```
[[2 3 4]
 [7 8 9]]
```

Data Types in NumPy

NumPy has some extra data types, and refer to data types with one character, like `i` for integers, `u` for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

NumPy dtype	Description	Example
`int8`	8-bit integer	-128 to 127
`int16`	16-bit integer	
`int32`	32-bit integer	
`int64`	64-bit integer	
`uint8`	8-bit unsigned int	0 to 255
`uint16`	16-bit unsigned int	
`float16`	Half precision float	
`float32`	Single precision float	
`float64`	Double precision float	(default)
`complex64`	Complex number (2×32)	
`complex128`	Complex (2×64)	

```
import numpy as np

arr1 = np.array([1, 2, 3, 4])
arr2 = np.array(['apple', 'banana', 'cherry'])
print(arr1.dtype)
print(arr2.dtype)
```

```
int64
<U6
```

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
```

```
[b'1' b'2' b'3' b'4']
|S1
```

b’1’: byte strings
U6: Unicode string

```python
import numpy as np

# Integer
a = np.array([1, 2, 3], dtype=np.int32)

# Float
b = np.array([1.1, 2.2], dtype=np.float64)

# Boolean
c = np.array([True, False, True], dtype=np.bool_)

# String
d = np.array(['apple', 'banana'], dtype='S')

# Unicode
e = np.array(['apple', 'banana'], dtype='U')

# Object
f = np.array([{'a': 1}, [1, 2, 3]], dtype=object)

# DateTime
g = np.array(['2023-01-01', '2023-05-01'], dtype='datetime64')

# Print dtypes
print("a:", a.dtype)
print("b:", b.dtype)
print("c:", c.dtype)
print("d:", d.dtype)
print("e:", e.dtype)
print("f:", f.dtype)
print("g:", g.dtype)
```

```
a: int32
b: float64
c: bool
d: |S6
e: <U6
f: object
```

`S6`: refers to a **byte string (ASCII encoded)** of **fixed length 6 byte.**

The Difference Between Copy and View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The copy *owns* the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The view *does not own* the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 31

print(arr)
print(x)
```

```
[31  2  3  4  5]
[31  2  3  4  5]
```

Get the Shape of an Array

NumPy arrays have an attribute called `shape` that returns a tuple with each index having the number of corresponding elements.

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```

```
(2, 4)
```

We can also reshape an array and go through it (iterating).

Joining NumPy Arrays

Joining means putting contents of two or more arrays in a single array. In SQL we join tables based on a key, whereas in NumPy we join arrays by axes.

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))

print(arr)
```

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

Splitting NumPy Arrays

Splitting is reverse operation of Joining.

Joining merges multiple arrays into one and Splitting breaks one array into multiple.

We use `array_split()` for splitting arrays, we pass it the array we want to split and the number of splits.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 4)
print(newarr)
```

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

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
```

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

Searching Arrays

You can search an array for a certain value, and return the indexes that get a match.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)

print(x)
```

```
(array([3, 5, 6]),)
```

Sorting Arrays

Sorting means putting elements in an *ordered sequence*.

```
import numpy as np
arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

```
[[2 3 4]
 [0 1 5]]
```

Arithmetic operations on numpy arrays

```
# Define two arrays
a = np.array([10, 20, 30])
b = np.array([1, 2, 3])

# Element-wise addition
print("Addition:", a + b)     # [11 22 33]

# Element-wise subtraction
print("Subtraction:", a - b)  # [9 18 27]

# Element-wise multiplication
print("Multiplication:", a * b)  # [10 40 90]

# Element-wise division
print("Division:", a / b)     # [10. 10. 10.]

# Integer division
print("Integer Division:", a // b)  # [10 10 10]

# Modulus (remainder)
print("Modulus:", a % b)      # [0 0 0]

# Exponentiation (a^b)
print("Power:", a ** b)       # [10^1 20^2 30^3]
```

```
Addition: [11 22 33]
Subtraction: [ 9 18 27]
Multiplication: [10 40 90]
Division: [10. 10. 10.]
Integer Division: [10 10 10]
Modulus: [0 0 0]
Power: [   10   400 27000]
```

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

print("Add 10:", arr + 10)      # [11 12 13 14]
print("Multiply by 2:", arr * 2) # [2 4 6 8]
print("Square:", arr ** 2)       # [1 4 9 16]
```

```
Add 10: [11 12 13 14]
Multiply by 2: [2 4 6 8]
Square: [ 1  4  9 16]
```

Mathematical operations in numpy

```
radii = np.array([1, 2, 3])
areas = np.pi * radii ** 2
print("Areas of circles:", areas)
```

```
Areas of circles: [ 3.14159265 12.56637061 28.27433388]
```

```
import numpy as np
arr = np.array([2, 4, 6, 8])
normalized = (arr - np.min(arr)) / (np.max(arr) - np.min(arr))
print("Normalized:", normalized)
```

```
Normalized: [0.         0.33333333 0.66666667 1.        ]
```

```python
import numpy as np  # Import NumPy for efficient array operations

# Step 1: Electricity usage (in kWh) for 6 households
usage_kwh = np.array([80, 150, 220, 95, 180, 300])  # Units consumed by
each household

# Step 2: Create an array to store total bill for each household
bills = np.zeros_like(usage_kwh, dtype=float)

# Step 3: Calculate bill based on tiered rates
# Loop over each household's usage
for i in range(len(usage_kwh)):
    units = usage_kwh[i]

    if units <= 100:
        # ₹5 per unit for first 100 units
        bills[i] = units * 5
    elif units <= 200:
        # ₹5 for first 100 + ₹7 for units between 101-200
        bills[i] = (100 * 5) + (units - 100) * 7
    else:
        # ₹5 for first 100 + ₹7 for next 100 + ₹10 for rest
        bills[i] = (100 * 5) + (100 * 7) + (units - 200) * 10

# Step 4: Print results
print("Electricity usage (kWh):", usage_kwh)
print("Electricity bills (INR):", bills)

# Step 5: Find household(s) with highest usage
max_usage = np.max(usage_kwh)
max_users = np.where(usage_kwh == max_usage)[0]
print(f"\nHousehold(s) with highest usage ({max_usage} kWh):",
max_users + 1)

# Step 6: Calculate average bill
average_bill = np.mean(bills)
print("Average bill amount (INR):", average_bill)
```

```
Electricity usage (kWh): [ 80 150 220  95 180 300]
Electricity bills (INR): [ 400.  850. 1400.  475. 1060. 2200.]

Household(s) with highest usage (300 kWh): [6]
Average bill amount (INR): 1064.1666666666667
```