NumPy Case Study

1. Import NumPy Import the NumPy library, which is essential for numerical computations in Python.

In [1]:
```
# Import the NumPy library for numerical operations
import numpy as np
```

2. Array Creation

- Create 1D, 2D, and 3D arrays
- Use `arange`, `linspace`, and `random` functions

In [2]:
```
# Create a 1D array
arr1 = np.array([1, 2, 3, 4, 5])
print("1D Array:", arr1)

# Create a 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print("2D Array:\n", arr2)

# Create a 3D array
arr3 = np.array([[[1,2],[3,4]], [[5,6],[7,8]]])
print("3D Array:\n", arr3)

# Use arange to create a range of values
arange_arr = np.arange(0, 10, 2)
print("arange:", arange_arr)

# Use linspace to create evenly spaced values
linspace_arr = np.linspace(0,15, 5)
print("linspace:", linspace_arr)

# Generate random numbers with np.random.rand
rand_arr = np.random.rand(1,3)
print("Random Array:\n", rand_arr)
```

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

 [[5 6]
  [7 8]]]
arange: [0 2 4 6 8]
linspace: [ 0.    3.75  7.5  11.25 15.  ]
Random Array:
 [[0.54824363 0.56481618 0.32285927]]
Random Array:
 [[0.54824363 0.56481618 0.32285927]]
```

3. Array Attributes and Reshaping

- Shape, size, dtype
- Reshape, flatten, ravel

In [3]:
```
# Get the shape, size, and data type of the array
print("Shape:", arr2.shape)
print("Size:", arr2.size)
print("Data type:", arr2.dtype)

# Reshape the array to a new shape
reshaped = arr2.reshape((3,2))
print("Reshaped :\n", reshaped)

# Flatten the array to 1D (returns a copy)
flattened = arr2.flatten()
print("Flattened 2D to 1D:", flattened)

# Ravel the array to 1D (returns a view if possible)
raveled = arr2.ravel()
print("Raveled 2D to 1D:", raveled)
```

```
Shape: (2, 3)
Size: 6
Data type: int64
Reshaped :
 [[1 2]
 [3 4]
 [5 6]]
Flattened 2D to 1D: [1 2 3 4 5 6]
Raveled 2D to 1D: [1 2 3 4 5 6]
```

4. Indexing and Slicing

- Basic and advanced indexing
- Boolean indexing

In [4]:
```
# Access elements and slices in arrays
print("Element at (1,2):", arr2[1,2])
print("First row:", arr2[0])
print("First column:", arr2[:,0])

# Boolean indexing to filter elements
bool_idx = arr2 > 2
print("Elements > 2:", arr2[bool_idx])
```

```
Element at (1,2): 6
First row: [1 2 3]
First column: [1 4]
Elements > 2: [3 4 5 6]
```

In [ ]:
```

```

5. Mathematical Operations

- Element-wise operations

- Aggregate functions
- Broadcasting

In [5]:
```
# Perform element-wise addition and multiplication
arr_a = np.array([1,2,3])
arr_b = np.array([4,5,6])
print("Addition:", arr_a + arr_b)
print("Multiplication:", arr_a * arr_b)
# Aggregate functions: mean and sum
print("Mean:", arr_a.mean())
print("Sum:", arr_a.sum())
# Broadcasting: add a scalar to all elements
print("Broadcasting:", arr_a + 10)
```

```
Addition: [5 7 9]
Multiplication: [ 4 10 18]
Mean: 2.0
Sum: 6
Broadcasting: [11 12 13]
```

6. Universal Functions (ufuncs)

- `np.sqrt` , `np.exp` , `np.log` , `np.sin` , etc.

In [6]:
```
# Apply universal functions to arrays
print("Square root:", np.sqrt(arr_a))
print("Exponential:", np.exp(arr_a))
print("Logarithm:", np.log(arr_a))
print("Sine:", np.sin(arr_a))
```

```
Square root: [1.         1.41421356 1.73205081]
Exponential: [ 2.71828183  7.3890561  20.08553692]
Logarithm: [0.         0.69314718 1.09861229]
Sine: [0.84147098 0.90929743 0.14112001]
```

7. Random Module

- `rand` , `randn` , `randint` , `seed`

In [7]:
```
# Set the random seed for reproducibility
np.random.seed(42)
# Generate random numbers from different distributions
print("Random float array:", np.random.rand(3))
print("Random normal array:", np.random.randn(3))
print("Random integers:", np.random.randint(0, 10, 5))
```

```
Random float array: [0.37454012 0.95071431 0.73199394]
Random normal array: [-1.11188012  0.31890218  0.27904129]
Random integers: [7 2 5 4 1]
```

8. Advanced Array Manipulation

- Stacking, splitting, repeating, tiling

In [8]:
```
# Stack arrays vertically and horizontally
a = np.array([[1,2],[3,4]])
```

```
b = np.array([[5,6]])
print("Vertical Stack:\n", np.vstack([a,b]))
print("Horizontal Stack:\n", np.hstack([a,b.T]))

# Split an array at specified indices
split_arr = np.split(arr1, [2,4])
print("Split array:", split_arr)

# Repeat elements of an array
repeated = np.repeat(arr1, 2)
print("Repeated:", repeated)

# Tile an array (repeat the whole array)
tiled = np.tile(arr1, 3)
print("Tiled:", tiled)
```

```
Vertical Stack:
 [[1 2]
 [3 4]
 [5 6]]
Horizontal Stack:
 [[1 2 5]
 [3 4 6]]
Split array: [array([1, 2]), array([3, 4]), array([5])]
Repeated: [1 1 2 2 3 3 4 4 5 5]
Tiled: [1 2 3 4 5 1 2 3 4 5 1 2 3 4 5]
```

9. Linear Algebra

- Dot product, matrix multiplication, transpose, inverse

In [ ]:
```
# mat: 2x2 matrix, vec: 1D vector
mat = np.array([[1,2],[3,4]])
vec = np.array([5,6])
# np.dot() computes the dot product of matrix and vector
print("Dot product:", np.dot(mat, vec))
# np.matmul() performs matrix multiplication
print("Matrix multiplication:", np.matmul(mat, mat))
# .T gives the transpose of the matrix
print("Transpose:\n", mat.T)
# np.linalg.inv() computes the inverse of the matrix
print("Inverse:\n", np.linalg.inv(mat))
```

```
Dot product: [17 39]
Matrix multiplication: [[ 7 10]
 [15 22]]
Transpose:
 [[1 3]
 [2 4]]
Inverse:
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

10. Masking and Conditional Logic

- `np.where`, `np.select`, masking arrays

In [10]:
```
# Create a boolean mask and use np.where for conditional selection
arr = np.array([1,2,3,4,5])
mask = arr % 2 == 0
print("Even numbers:", arr[mask])

labels = np.where(arr > 3, 'High', 'Low')
print("Labels:", labels)
```

```
Even numbers: [2 4]
Labels: ['Low' 'Low' 'Low' 'High' 'High']
```

11. Advanced Functions

- `np.apply_along_axis`, `np.vectorize`, `np.fromfunction`

In [11]:
```
# Define a custom function and apply it using vectorize and apply_along_axis
def custom_func(x):
    return x**2 + 1

arr = np.arange(5)
# Apply np.sum along axis 0 (trivial for 1D)
print("Apply along axis:", np.apply_along_axis(np.sum, 0, arr))
# Vectorize the custom function
vec_func = np.vectorize(custom_func)
print("Vectorized function:", vec_func(arr))

# Create an array using a function of indices
f = np.fromfunction(lambda i, j: i + j, (3, 3), dtype=int)
print("Fromfunction array:\n", f)
```

```
Apply along axis: 10
Vectorized function: [ 1  2  5 10 17]
Fromfunction array:
 [[0 1 2]
 [1 2 3]
 [2 3 4]]
```

12. Saving and Loading Data

- `np.save`, `np.load`, `np.savetxt`, `np.loadtxt`

In [12]:
```
# Save and load arrays in binary and text formats
arr = np.arange(10)
np.save('my_array.npy', arr)
loaded = np.load('my_array.npy')
print("Loaded array:", loaded)

np.savetxt('my_array.txt', arr)
loaded_txt = np.loadtxt('my_array.txt')
print("Loaded from txt:", loaded_txt)
```

```
Loaded array: [0 1 2 3 4 5 6 7 8 9]
Loaded from txt: [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

13. Broadcasting Rules and Examples

- Demonstrate how broadcasting works with different shapes

In [13]:
```
# Demonstrate broadcasting with arrays of different shapes
a = np.array([[1],[2],[3]])
b = np.array([10,20,30])
print("Broadcasted sum:\n", a + b)
```

```
Broadcasted sum:
 [[11 21 31]
 [12 22 32]
 [13 23 33]]
```