NumPy Tutorial with Code and Outputs

NumPy is a fundamental library for scientific computing in Python. It provides support for arrays and matrices, along with a collection of mathematical functions to operate on these data structures.

Creating a 1D NumPy array

```
import numpy as np
arr1 = np.array([1,2,3,4,5])
print(arr1)
print(type(arr1))
print(arr1.shape)
```

Expected Output:

```
Output:
[1 2 3 4 5]
<class 'numpy.ndarray'>
(5,)
```

Creating a 2D NumPy array

```
arr2 = np.array([[1,2,3,4],[2,3,4,5]])
print(arr2)
print(arr2.shape)
```

Expected Output:

```
Output:
[[1 2 3 4]
 [2 3 4 5]]
(2, 4)
```

Creating special arrays

```
# Using arange and reshape
print(np.arange(0,10,2).reshape(5,1))
# Creating an array of ones
print(np.ones((3,4)))
# Creating an identity matrix
print(np.eye(3))
```

Expected Output:

```
Output:
[[0]
 [2]
 [4]
 [6]
 [8]]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

NumPy Vectorized Operations

```
arr1 = np.array([1,2,3,4,5])
arr2 = np.array([10,20,30,40,50])
print(arr1 + arr2)  # Addition
print(arr1 - arr2)  # Subtraction
print(arr1 * arr2)  # Multiplication
print(arr1 / arr2)  # Division
```

Expected Output:

```
Output:
[11 22 33 44 55]
[-9 -18 -27 -36 -45]
[ 10  40  90 160 250]
[0.1 0.1 0.1 0.1 0.1]
```

Statistical Concepts - Normalization

```
data = np.array([1,2,3,4,5])
mean = np.mean(data)
std_dev = np.std(data)
normalized_data = (data - mean) / std_dev
print(normalized_data)
```

Expected Output:

```
Output:
[-1.2649 -0.6324  0.       0.6324  1.2649]
```

Array Slicing and Indexing

```
arr = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print(arr[1:3,1:3])  # Extracting a subarray
print(arr[0,0])      # Accessing a single element
arr[0,0] = 100       # Modifying an element
print(arr)
```

Expected Output:

```
Output:
[[ 6  7]
 [10 11]]
1
[[100   2   3   4]
 [  5   6   7   8]
 [  9  10  11  12]]
```

Logical Operations on Arrays

```
data = np.array([1,2,3,4,5,6,7,8,9,10])
filtered_data = data[(data >= 5) & (data <= 8)]
print(filtered_data)
```

Expected Output:

```
Output:
[5 6 7 8]
```

Array Attributes

```
arr = np.array([[1,2,3], [4,5,6]])
print("Array:", arr)
print("Shape:", arr.shape)
print("Dimensions:", arr.ndim)
print("Size (total number of elements):", arr.size)
print("Data type:", arr.dtype)
print("Item size (in bytes):", arr.itemsize)
```

Expected Output:

```
Output:
Array: [[1 2 3]
 [4 5 6]]
Shape: (2, 3)
Dimensions: 2
Size (total number of elements): 6
Data type: int64  # (or int32, depending on your system)
Item size (in bytes): 8  # (or 4, depending on your system)
```

Universal Functions

```
arr = np.array([2,3,4,5,6])
print("Square Root:", np.sqrt(arr))
print("Exponential:", np.exp(arr))
print("Sine:", np.sin(arr))
print("Natural Log:", np.log(arr))
```

Expected Output:

```
Output:
Square Root: [1.41421356 1.73205081 2.         2.23606798 2.44948974]
Exponential: [  7.3890561   20.08553692  54.59815003 148.4131591  403.42879349]
Sine: [ 0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155 ]
Natural Log: [0.69314718 1.09861229 1.38629436 1.60943791 1.79175947]
```

Reshaping and Transposing Arrays

```
arr = np.array([[1,2,3], [4,5,6]])
reshaped = arr.reshape(3,2)
transposed = arr.T
print("Original Array:", arr)
print("Reshaped Array (3x2):", reshaped)
print("Transposed Array:", transposed)
```

Expected Output:

```
Output:
Original Array: [[1 2 3]
 [4 5 6]]
Reshaped Array (3x2): [[1 2]
 [3 4]
 [5 6]]
Transposed Array: [[1 4]
 [2 5]
 [3 6]]
```

Broadcasting in NumPy

```
arr = np.array([[1,2,3], [4,5,6]])
scalar = 10
```

```
print("Original Array:", arr)
print("After Broadcasting Addition:", arr + scalar)
```

Expected Output:

```
Output:
Original Array: [[1 2 3]
 [4 5 6]]
After Broadcasting Addition: [[11 12 13]
 [14 15 16]]
```