Introduction:

Multidimensional arrays are a fundamental data structure in numerical computing, enabling efficient storage and manipulation of multi-dimensional data. In this article, we'll explore various aspects of creating and manipulating multidimensional arrays using NumPy.

Creating Multi-dimensional Arrays:

- Creating a 2D Array from a List of Lists:
    - NumPy's np.array() function can create a 2D array from a list of lists, where each sublist represents a row in the array.

```
import numpy as np

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

print(arr)
```

Output

```
[[1 2 3]

 [4 5 6]]
```

- Creating a 3D Array from a List of Lists:
    - Similarly, a 3D array can be created from a nested list structure, with each sublist representing a 2D slice of the array.

```
import numpy as np

arr = np.array([[[1,2,3],[4,5,6],[7,8,9]],

        [[1,2,3],[4,5,6],[7,8,9]],

        [[1,2,3],[4,5,6],[7,8,9]]])

print(arr)
```

Output

```
[[[1 2 3]

  [4 5 6]

  [7 8 9]]


 [[1 2 3]

  [4 5 6]

  [7 8 9]]
```

[[1 2 3]

[4 5 6]

[7 8 9]]]

- Creating a 2D Array from a List of Tuples:
    - NumPy arrays can also be created from a list of tuples, with each tuple representing a row in the array.

```
import numpy as np

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

print(arr)
```

Output

[[1 2 3]

[4 5 6]]

- Creating a 2D Array from a List of Arrays:
    - Arrays within a list can be used to construct a 2D array, with each array representing a row in the resulting array.

```
import numpy as np

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

print(arr)
```

Output

[[1 2 3]

[4 5 6]]

- Creating a 2D Array from a List of Strings:
    - Even a list of strings can be converted into a 2D array, where each string is treated as an element in the array.

```
import numpy as np

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

```
print(arr)
```

Output

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

- Creating a 2D Array from a List of Dictionaries:
    - Although less common, it's possible to create a 2D array from a list of dictionaries, with each dictionary representing a row where keys are column names.

```
import numpy as np
arr = np.array([{'a' : 1, 'b' : 2, 'c' : 3},{'d' : 4, 'e' : 5, 'f' : 6}])
print(arr)
```

Output

```
[{'a': 1, 'b': 2, 'c': 3}{'d': 4, 'e': 5, 'f': 6}]
```

Finding Details of Arrays:

- Printing the Shape of the Array:
    - The shape attribute of a NumPy array provides information about its dimensions, indicating the number of rows and columns.

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

Output

```
(3, 2)
```

- Printing the Number of Dimensions of the Array:
    - The ndim attribute of an array gives the number of dimensions or axes of the array.

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

Output

2

- Printing the Data Type of the Array:
    - The dtype attribute of an array specifies the data type of its elements, such as integer, float, or string.

```
import numpy as np

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

print(arr.dtype)
```

Output

int64

- Printing the Size of the Array:
    - The size attribute of an array returns the total number of elements in the array.

```
import numpy as np

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

print(arr.size)
```

Output

8

- Printing the Item Size:
    - The itemsize attribute of an array specifies the size of each element in bytes.

```
import numpy as np

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

print(arr.itemsize)
```

Output

8

- Printing the Type of the Array:
    - The type() function provides the type of the array, which is typically <class 'numpy.ndarray'>.

```
import numpy as np

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

print(type(arr))
```

Output

```
<class 'numpy.ndarray'>
```

Conclusion:

Understanding how to create and manipulate multidimensional arrays is essential for efficient data processing and analysis using NumPy. By exploring the creation methods and attributes of arrays, users can effectively work with multi-dimensional data structures in their projects.