NumPy N-D Array Creation

NumPy is not restricted to 1-D arrays, it can have arrays of multiple dimensions, also known as N-dimensional arrays or ndarrays.

An N-dimensional array refers to the number of dimensions in which the array is organized.

An array can have any number of dimensions and each dimension can have any number of elements.

For example, a 2D array represents a table with rows and columns, while a 3D array represents a cube with width, height, and depth.

There are multiple techniques to create N-d arrays in NumPy, and we will explore each of them below.

N-D Array Creation From List of Lists

To create an N-dimensional NumPy array from a Python List, we can use the np.array() function and pass the list as an argument.

Create a 2-D NumPy Array

Let's create a 2D NumPy array with **2** rows and **4** columns using lists.

```
import numpy as np


# create a 2D array with 2 rows and 4 columns

array1 = np.array([[1, 2, 3, 4],

                   [5, 6, 7, 8]])


print(array1)
```

Output

```
[[1 2 3 4]

 [5 6 7 8]]
```

In the above example, we first created a 2D list (list of lists) [[1, 2, 3, 4], [5, 6, 7, 8]] with **2** rows and **4** columns. We then passed the list to the np.array() function to create a 2D array.

Create a 3-D NumPy Array

Let's say we want to create a 3-D NumPy array consisting of two **"slices"** where each slice has **3** rows and **4** columns.

Here's how we create our desired 3-D array,

```
import numpy as np


# create a 3D array with 2 "slices", each of 3 rows and 4 columns

array1 = np.array([[[1, 2, 3, 4],

               [5, 6, 7, 8],

               [9, 10, 11, 12]],


               [[13, 14, 15, 16],

                [17, 18, 19, 20],

                [21, 22, 23, 24]]])


print(array1)
```

Output

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

 [[13 14 15 16]
  [17 18 19 20]
  [21 22 23 24]]]
```

Here, we created a 3D list [list of lists of lists] and passed it to the np.array() function. This creates the 3-D array named array1.

In the 3D list,

- The outermost list contains two elements, which are lists representing the two "slices" of the array. Each slice is a 2-D array with **3** rows and **4** columns.
- The innermost lists represent the individual rows of the 2-D arrays.

Note: In the context of an N-D array, a slice is like a subset of the array that we can take out by selecting a specific range of rows, columns.

Creating N-d Arrays From Scratch

We saw how to create N-d NumPy arrays from Python lists. Now we'll see how we can create them from scratch.

To create multidimensional arrays from scratch we use functions such as

- np.zeros()
- np.arange()
- np.random.rand()

Create N-D Arrays using np.zeros()

The np.zeros() function allows us to create N-D arrays filled with all zeros. For example,

```
import numpy as np


# create 2D array with 2 rows and 3 columns filled with zeros

array1 = np.zeros((2, 3))




print("2-D Array: ")

print(array1)


# create 3D array with dimensions 2x3x4 filled with zeros

array2 = np.zeros((2, 3, 4))




print("\n3-D Array: ")

print(array2)
```

Output

```
2-D Array:

[[0. 0. 0.]
```

```
 [0. 0. 0.]]


3-D Array:
[[[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]

 [[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]]
```

In the above example, we have used the np.zeros() function to create a 2-D array and 3-D array filled with zeros respectively.

- np.zeros((2, 3)) - returns a zero filled 2-D array with **2** rows and **3** columns
- np.zeros((2, 3, 4)) - returns a zero filled 3-D array with 2 slices, each slice having **3** rows and **4** columns.

Note: Similarly we can use np.ones() to create an array filled with values **1.**

Create N-D Array with a Specified Value

In NumPy, we can use the np.full() function to create a multidimensional array with a specified value.

For example, to create a 2-D array with the value **5**, we can do the following:

```
import numpy as np


# Create a 2-D array with elements initialized to 5

numpy_array = np.full((2, 2), 5)


print("Array:", numpy_array)
```

Output

```
[[5 5]
[5 5]]
```

Here, we have used the np.full() function to create a 2-D array where all elements are initialized to **5**.

Creating Arrays With np.random.rand()

The np.random.rand() function is used to create an array of random numbers.

Let's see an example to create an array of **5** random numbers,

```
import numpy as np


# create a 2D array of 2 rows and 2 columns of random numbers

array1 = np.random.rand(2, 2)



print("2-D Array: ")

print(array1)


# create a 3D array of shape (2, 2, 2) of random numbers

array2 = np.random.rand(2, 2, 2)



print("\n3-D Array: ")

print(array2)
```

Run Code

Output

```
2-D Array:

[[0.13198621 0.54730421]

 [0.36570987 0.16233836]]


3-D Array:

[[[0.15666007 0.4580507 ]
```

```
 [0.84769856 0.76699589]]


[[0.45395202 0.39944328]
 [0.62999479 0.39629496]]]
```

Here,

- np.random.rand(2, 2) - creates a 2D array of **2** rows and **2** columns of random numbers.
- np.random.rand(2, 2, 2) - creates a 3D array with 2 slices, each slice having **2** rows and **2** columns of random numbers.

Create Empty N-D NumPy Array

To create an empty N-D NumPy array, we use the np.empty() function. For example,

```
import numpy as np


# create an empty 2D array with 2 rows and 2 columns
array1 = np.empty((2, 2))



print("2-D Array: ")
print(array1)


# create an empty 3D array of shape (2, 2, 2)
array2 = np.empty((2, 2, 2))



print("\n3-D Array: ")
print(array2)
```

Output

```
2-D Array:
```

```
[[8.86495615e-317 0.00000000e+000]
 [2.21149159e-316 1.76125651e-312]]


3-D Array:
[[[1.0749539e-316 0.0000000e+000]
  [0.0000000e+000 0.0000000e+000]]


 [[0.0000000e+000 0.0000000e+000]
  [0.0000000e+000 0.0000000e+000]]]
```

In the above example, we used the np.empty() function to create an empty 2-D array and a 3-D array respectively.

If we look into the output of the code, we can see the empty array is actually not empty, it has some values in it.

It is because although we are creating an empty array, NumPy will try to add some value to it. The values stored in the array are arbitrary and have no significance value.