

NumPy Introduction

What is NumPy?

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

Installation of NumPy

```
C:\Users\Your Name>pip install numpy
```

Import NumPy

Once NumPy is installed, import it in your applications by adding the `import` keyword:

```
import numpy
```

Example

```
import numpy

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

print(arr)
```

NumPy as np

NumPy is usually imported under the `np` alias.

Create an alias with the `as` keyword while importing:

```
import numpy as np

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

print(arr)
```

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called `ndarray`.

We can create a NumPy `ndarray` object by using the `array()` function.

Example

```
import numpy as np

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

print(arr)

print(type(arr))
```

type(): This built-in Python function tells us the type of the object passed to it. Like in above code it shows that `arr` is `numpy.ndarray` type.

To create an `ndarray`, we can pass a list, tuple or any array-like object into the `array()` method, and it will be converted into an `ndarray`:

Example

Use a tuple to create a NumPy array:

```
import numpy as np

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

print(arr)
```

Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

nested array: are arrays that have arrays as their elements.

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example

Create a 0-D array with value 42

```
import numpy as np

arr = np.array(42)

print(arr)
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

Example

Create a 1-D array containing the values 1,2,3,4,5:

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

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

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

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.

Example

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

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

print(arr)
```

Check Number of Dimensions?

NumPy Arrays provides the `ndim` attribute that returns an integer that tells us how many dimensions the array have.

Example

Check how many dimensions the arrays have:

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Access Array Elements

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[1])
```

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
```

Slicing arrays

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 its considered 0

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

If we don't pass step its considered 1

```
import numpy as np

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

print(arr[1:5])
```

Negative Slicing

Use the minus operator to refer to an index from the end:

Example

Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np

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

print(arr[-3:-1])
```

Slicing 2-D Arrays

Example

From the second element, slice elements from index 1 to index 4 (not included):

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

Shape of an Array

The shape of an array is the number of elements in each dimension.

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.

Example

Print the shape of a 2-D array:

```
import numpy as np

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

print(arr.shape)
```

Reshaping arrays

Reshaping means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

Reshape From 1-D to 2-D

Example

Convert the following 1-D array with 12 elements into a 2-D array.

The outermost dimension will have 4 arrays, each with 3 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

Can We Reshape Into any Shape?

Yes, as long as the elements required for reshaping are equal in both shapes.

We can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3 elements 3 rows 2D array as that would require $3 \times 3 = 9$ elements.