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.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

Dr. Uma

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.

The array object in NumPy is called `ndarray`, it provides a lot of supporting functions that make working with `ndarray` very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

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.

This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Which Language is NumPy written in?

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Where is the NumPy Codebase?

The source code for NumPy is located at this github repository https://github.com/numpy/numpy

github: enables many people to work on the same codebase.

NumPy Getting Started

Installation of NumPy

If you have Python and PIP already installed on a system, then installation of NumPy is very easy.

Install it using this command:

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

If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

Import NumPy

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

```
import numpy
```

Now NumPy is imported and ready to use.

Dr. Uma

Example

```
import numpy

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

print(arr)
```

OUTPUT:

[1 2 3 4 5]

NumPy as np

NumPy is usually imported under the `np` alias.

alias: In Python alias are an alternate name for referring to the same thing.

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

```
import numpy as np
```

Now the NumPy package can be referred to as `np` instead of `numpy`.

Example

```
import numpy as np

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

print(arr)
```

OUTPUT:

[1 2 3 4 5]

Dr. Uma

Checking NumPy Version → 1.23.0

The version string is stored under `__version__` attribute.

June 22, 2022

Example

```
import numpy as np

print(np.__version__)
```

OUTPUT:

↓

1.23.5 (releasenotes)

↓

19th Nov, 22

1.16.3

NumPy Creating Arrays

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))
```

OUTPUT:

[1 2 3 4 5]

<class 'numpy.ndarray'>

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)
```

[1 2 3 4 5]

Dimensions in Arrays → Nested Array

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

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

Dr.Uma

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)
```

42

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)
```

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

Dr. Uma

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.

NumPy has a whole sub module dedicated towards matrix operations called `numpy.mat`

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)
```

```
[[1 2 3]
```

```
[4 5 6]]
```

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)
```

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

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

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)
```

Dr. Uma

NumPy Array Indexing

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.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example

Get the first element from the following array:

```
import numpy as np

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

print(arr[0])
```

OUTPUT:

```
1
```

Example

Get the second element from the following array.

```
import numpy as np

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

print(arr[1])
```

OUTPUT:

2

Example

Get third and fourth elements from the following array and add them.

```
import numpy as np

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

print(arr[2] + arr[3])
```

OUTPUT:

7

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.

Think of 2-D arrays like a table with rows and columns, where the row represents the dimension and the index represents the column.

Example

Access the element on the first row, second column:

```
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])
```

2nd element on 1st dim: 2

Example

Access the element on the 2nd row, 5th column:

```
import numpy as np

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

print('5th element on 2nd row: ', arr[1, 4])
```

6

Dr.Uma

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.

Example

Access the third element of the second array of the first array:

```
import numpy as np

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

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

`arr[0, 1, 2]` prints the value 6.

And this is why:

The first number represents the first dimension, which contains two arrays:
[[1, 2, 3], [4, 5, 6]]

and:
[[7, 8, 9], [10, 11, 12]]
Since we selected `0`, we are left with the first array:
[[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays:
[1, 2, 3]
and:
[4, 5, 6]
Since we selected `1`, we are left with the second array:
[4, 5, 6]

The third number represents the third dimension, which contains three values:
4
5
6
Since we selected `2`, we end up with the third value:
6

Dr.Uma

Negative Indexing

Use negative indexing to access an array from the end.

Example

Print the last element from the 2nd dim:

```
import numpy as np

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

print('Last element from 2nd dim: ', arr[1, -1])
```

Last element from 2nd dim: 10

NumPy Array Slicing

Dr. Uma

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

Example

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np

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

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

0 1 2 3 4 5 6

[1 5]

OUTPUT:

```
[2 3 4 5]
```

Note: The result *includes* the start index, but *excludes* the end index.

Example

Slice elements from index 4 to the end of the array:

```
import numpy as np

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

print(arr[4:])
```

OUTPUT:

```
[5 6 7]
```

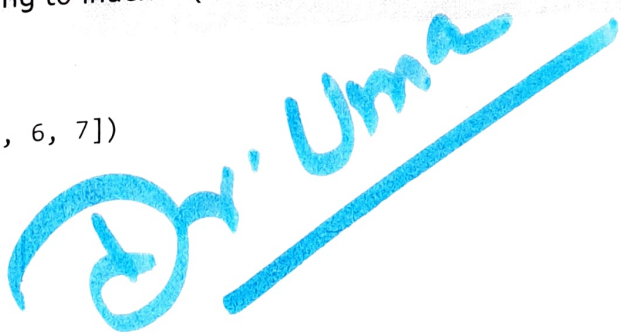
Example

Slice elements from the beginning to index 4 (not included):

```
import numpy as np

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

print(arr[:4])
```



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])
```

OUTPUT:

[5 6]

STEP

Use the `step` value to determine the step of the slicing:

Example

Return every other element from index 1 to index 5:

```
import numpy as np

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

print(arr[1:5:2])
```

OUTPUT:

[2 4]