In [ ]:

```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\dilip\anaconda3\lib\site-packages (1.19.
2)
Note: you may need to restart the kernel to use updated packages.
```

Creating array

Creating a NumPy ndarray Object

NumPy is used to work work arrays.

The array object in NumPy is called ndarray.

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

In [ ]:

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

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

In [ ]:

```
print(np.__version__)
```

```
1.19.2
```

Dimensions in Arrays

Creating 0 Dimension

In [ ]:

```
import numpy as np

arr = np.array(42)

print(arr)
```

```
42
```

1-D Arrays

In [ ]:

```
import numpy as np

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

print(arr)
```

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

2-D Array

In [ ]:

```
import numpy as np

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

print(arr)
```

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

3-D Array

In [ ]:

```
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]]]
```

Checking No.of Dimensions

In [ ]:

```
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)
```

```
0
1
2
3
```

NumPy Array Indexing

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.

In [ ]:

```
import numpy as np

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

print(arr[0])
```

```
1
```

In [ ]:

```
import numpy as np

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

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

```
3 4
```

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 dimension represents the row and the index represents the column.

In [ ]:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr)
print('2nd element on 1st row: ', arr[1, 4])
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
2nd element on 1st row:  10
```

In [ ]:

```
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])
```

```
5th element on 2nd row:  10
```

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 elemen

In [ ]:

```
import numpy as np

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

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

```
11
```

Negative Indexing

Use negative indexing to access an array from the end.

In [ ]:

```
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

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

In [ ]:

```
import numpy as np

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

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

```
[2 3]
```

Negative Slicing

In [ ]:

```
import numpy as np

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

print(arr[-3:-2])
```

```
[5]
```

STEP

- **Use the step value to determine the step of the slicing**

In [ ]:

```
import numpy as np

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

```
print(arr[1:6:2])
```

```
[2 4 6]
```

Return every element of list in step wise

In [ ]:

```
import numpy as np

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

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

```
[1 3 5]
```

Slicing 2-D Arrays

In [ ]:

```
import numpy as np

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

print(arr[0, 1:4])
```

```
[2 3 4]
```

In [ ]:

```
import numpy as np

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

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

```
[6 7]
```

In [ ]:

```
import numpy as np

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

print(arr[0:2,3])
```

```
[4 9]
```

In [ ]:

```
import numpy as np

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

print(arr[0:2, 1:4])
```

```
[[2 3 4]
 [7 8 9]]
```

NumPy Array Shape

shape of an array

- **NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.**

In [ ]:

```
import numpy as np

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

print(arr.shape)
```

```
(2, 4)
```

Reshaping arrays

Reshape From 1-D to 2-D

In [ ]:

```
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)
```

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

Reshape From 1-D to 3-D

In [ ]:

```
import numpy as np

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

newarr = arr.reshape(3,4)

print(newarr)
```

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

In [ ]:

```

```