# NumPy - Copies & Views

While executing the functions, some of them return a copy of the input array, while some return the view. When the contents are physically stored in another location, it is called **Copy**. If on the other hand, a different view of the same memory content is provided, we call it as **View**.

## No Copy

Simple assignments do not make the copy of array object. Instead, it uses the same id() of the original array to access it. The **id()** returns a universal identifier of Python object, similar to the pointer in C.

Furthermore, any changes in either gets reflected in the other. For example, the changing shape of one will change the shape of the other too.

### Example

Live Demo

```
import numpy as np

a = np.arange(6)


print 'Our array is:'

print a


print 'Applying id() function:'

print id(a)


print 'a is assigned to b:'

b = a

print b
```

```
print 'b has same id():'

print id(b)



print 'Change shape of b:'

b.shape = 3,2

print b



print 'Shape of a also gets changed:'

print a
```

It will produce the following output –

```
Our array is:
[0 1 2 3 4 5]

Applying id() function:
139747815479536

a is assigned to b:
[0 1 2 3 4 5]
b has same id():
139747815479536

Change shape of b:
[[0 1]
 [2 3]
 [4 5]]

Shape of a also gets changed:
[[0 1]
 [2 3]
 [4 5]]
```

# View or Shallow Copy

NumPy has **ndarray.view()** method which is a new array object that looks at the same data of the original array. Unlike the earlier case, change in dimensions of the new array doesn't change dimensions of the original.

## Example

Live Demo

```
import numpy as np

# To begin with, a is 3X2 array
```

```
a = np.arange(6).reshape(3,2)


print 'Array a:'

print a


print 'Create view of a:'

b = a.view()

print b


print 'id() for both the arrays are different:'

print 'id() of a:'

print id(a)

print 'id() of b:'

print id(b)


# Change the shape of b. It does not change the shape of a

b.shape = 2,3


print 'Shape of b:'

print b


print 'Shape of a:'

print a
```

It will produce the following output –

```
Array a:
[[0 1]
 [2 3]
 [4 5]]

Create view of a:
[[0 1]
```

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

id() for both the arrays are different:
id() of a:
140424307227264
id() of b:
140424151696288

Shape of b:
[[0 1 2]
 [3 4 5]]

Shape of a:
[[0 1]
 [2 3]
 [4 5]]
```

Slice of an array creates a view.

## Example

Live Demo

```
import numpy as np

a = np.array([[10,10], [2,3], [4,5]])



print 'Our array is:'

print a



print 'Create a slice:'

s = a[:, :2]

print s
```

It will produce the following output –

```
Our array is:
[[10 10]
 [ 2 3]
 [ 4 5]]

Create a slice:
[[10 10]
 [ 2 3]
 [ 4 5]]
```

# Deep Copy

The **ndarray.copy()** function creates a deep copy. It is a complete copy of the array and its data, and doesn't share with the original array.

## Example

Live Demo

```
import numpy as np

a = np.array([[10,10], [2,3], [4,5]])


print 'Array a is:'

print a


print 'Create a deep copy of a:'

b = a.copy()

print 'Array b is:'

print b


#b does not share any memory of a

print 'Can we write b is a'

print b is a


print 'Change the contents of b:'

b[0,0] = 100


print 'Modified array b:'

print b


print 'a remains unchanged:'

print a
```

It will produce the following output –

```
Array a is:
[[10 10]
 [ 2 3]
 [ 4 5]]

Create a deep copy of a:
Array b is:
[[10 10]
 [ 2 3]
 [ 4 5]]
Can we write b is a
False

Change the contents of b:
Modified array b:
[[100 10]
 [ 2 3]
 [ 4 5]]

a remains unchanged:
[[10 10]
 [ 2 3]
 [ 4 5]]
```