# NumPy - Array From Existing Data

In this chapter, we will discuss how to create an array from existing data.

## numpy.asarray

This function is similar to numpy.array except for the fact that it has fewer parameters. This routine is useful for converting Python sequence into ndarray.

```
numpy.asarray(a, dtype = None, order = None)
```

The constructor takes the following parameters.

| Sr.No. | Parameter & Description |
|---|---|
| 1 | **a**<br>Input data in any form such as list, list of tuples, tuples, tuple of tuples or tuple of lists |
| 2 | **dtype**<br>By default, the data type of input data is applied to the resultant ndarray |
| 3 | **order**<br>C (row major) or F (column major). C is default |

The following examples show how you can use the **asarray** function.

### Example 1

Live Demo

```
# convert list to ndarray

import numpy as np


x = [1,2,3]
```

```
a = np.asarray(x)

print a
```

Its output would be as follows −

```
[1  2  3]
```

## Example 2

Live Demo

```
# dtype is set

import numpy as np



x = [1,2,3]

a = np.asarray(x, dtype = float)

print a
```

Now, the output would be as follows −

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

## Example 3

Live Demo

```
# ndarray from tuple

import numpy as np



x = (1,2,3)

a = np.asarray(x)

print a
```

Its output would be −

```
[1  2  3]
```

## Example 4

Live Demo

```
# ndarray from list of tuples

import numpy as np


x = [(1,2,3),(4,5)]

a = np.asarray(x)

print a
```

Here, the output would be as follows –

```
[(1, 2, 3) (4, 5)]
```

# numpy.frombuffer

This function interprets a buffer as one-dimensional array. Any object that exposes the buffer interface is used as parameter to return an **ndarray**.

```
numpy.frombuffer(buffer, dtype = float, count = -1, offset = 0)
```

The constructor takes the following parameters.

| Sr.No. | Parameter & Description |
|---|---|
| 1 | **buffer**<br>Any object that exposes buffer interface |
| 2 | **dtype**<br>Data type of returned ndarray. Defaults to float |
| 3 | **count**<br>The number of items to read, default -1 means all data |
| 4 | **offset**<br>The starting position to read from. Default is 0 |

## Example

The following examples demonstrate the use of **frombuffer** function.

Live Demo

```
import numpy as np

s = 'Hello World'

a = np.frombuffer(s, dtype = 'S1')

print a
```

Here is its output −

```
['H'  'e'  'l'  'l'  'o'  ' '  'W'  'o'  'r'  'l'  'd']
```

# numpy.fromiter

This function builds an **ndarray** object from any iterable object. A new one-dimensional array is returned by this function.

```
numpy.fromiter(iterable, dtype, count = -1)
```

Here, the constructor takes the following parameters.

| Sr.No. | Parameter & Description |
|---|---|
| 1 | **iterable**<br>Any iterable object |
| 2 | **dtype**<br>Data type of resultant array |
| 3 | **count**<br>The number of items to be read from iterator. Default is -1 which means all data to be read |

The following examples show how to use the built-in **range()** function to return a list object. An iterator of this list is used to form an **ndarray** object.

## Example 1

Live Demo

```
# create list object using range function

import numpy as np

list = range(5)

print list
```

Its output is as follows −

```
[0,  1,  2,  3,  4]
```

## Example 2

Live Demo

```
# obtain iterator object from list

import numpy as np

list = range(5)

it = iter(list)


# use iterator to create ndarray

x = np.fromiter(it, dtype = float)

print x
```

Now, the output would be as follows −

```
[0.   1.   2.   3.   4.]
```