Another useful attribute is the `dtype`, the data type of the array (which we discussed previously in "Understanding Data Types in Python" on page 34):

```
In[3]: print("dtype:", x3.dtype)
```

```
dtype: int64
```

Other attributes include `itemsize`, which lists the size (in bytes) of each array element, and `nbytes`, which lists the total size (in bytes) of the array:

```
In[4]: print("itemsize:", x3.itemsize, "bytes")
       print("nbytes:", x3.nbytes, "bytes")
```

```
itemsize: 8 bytes
nbytes: 480 bytes
```

In general, we expect that `nbytes` is equal to `itemsize` times `size`.

Array Indexing: Accessing Single Elements

If you are familiar with Python's standard list indexing, indexing in NumPy will feel quite familiar. In a one-dimensional array, you can access the i^{th} value (counting from zero) by specifying the desired index in square brackets, just as with Python lists:

```
In[5]: x1
```

```
Out[5]: array([5, 0, 3, 3, 7, 9])
```

```
In[6]: x1[0]
```

```
Out[6]: 5
```

```
In[7]: x1[4]
```

```
Out[7]: 7
```

To index from the end of the array, you can use negative indices:

```
In[8]: x1[-1]
```

```
Out[8]: 9
```

```
In[9]: x1[-2]
```

```
Out[9]: 7
```

In a multidimensional array, you access items using a comma-separated tuple of indices:

```
In[10]: x2
```

```
Out[10]: array([[3, 5, 2, 4],
                [7, 6, 8, 8],
                [1, 6, 7, 7]])
```

```
In[11]: x2[0, 0]
```

```
Out[11]: 3
```

```
In[12]: x2[2, 0]

Out[12]: 1

In[13]: x2[2, -1]

Out[13]: 7
```

You can also modify values using any of the above index notation:

```
In[14]: x2[0, 0] = 12
        x2

Out[14]: array([[12,  5,  2,  4],
                [ 7,  6,  8,  8],
                [ 1,  6,  7,  7]])
```

Keep in mind that, unlike Python lists, NumPy arrays have a fixed type. This means, for example, that if you attempt to insert a floating-point value to an integer array, the value will be silently truncated. Don't be caught unaware by this behavior!

```
In[15]: x1[0] = 3.14159  # this will be truncated!
        x1

Out[15]: array([3, 0, 3, 3, 7, 9])
```

Array Slicing: Accessing Subarrays

Just as we can use square brackets to access individual array elements, we can also use them to access subarrays with the *slice* notation, marked by the colon (`:`) character. The NumPy slicing syntax follows that of the standard Python list; to access a slice of an array `x`, use this:

```
x[start:stop:step]
```

If any of these are unspecified, they default to the values `start=0`, `stop=`*`size of dimension`*, `step=1`. We'll take a look at accessing subarrays in one dimension and in multiple dimensions.

One-dimensional subarrays

```
In[16]: x = np.arange(10)
        x

Out[16]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In[17]: x[:5]  # first five elements

Out[17]: array([0, 1, 2, 3, 4])

In[18]: x[5:]  # elements after index 5

Out[18]: array([5, 6, 7, 8, 9])

In[19]: x[4:7]  # middle subarray

Out[19]: array([4, 5, 6])
```