Integer Mask

Let's select all elements with even indices in the array.

```
# create 10 random integers between 1 and 20
my_array = np.random.randint(1, 20, 10)
my_array
'Output': array([ 1, 18,  8, 12, 10,  2, 17,  4, 17, 17])
my_array[np.arange(1,10,2)]
'Output': array([18, 12,  2,  4, 17])
```

Remember that array indices are indexed from 0. So the second element, 18, is in index 1.

```
np.arange(1,10,2)
'Output': array([1, 3, 5, 7, 9])
```

Slicing a 1-D Array

Slicing a NumPy array is also similar to slicing a Python list.

```
my_array = np.array([14,  9,  3, 19, 16,  1, 16,  5, 13,  3])
my_array
'Output': array([14,  9,  3, 19, 16,  1, 16,  5, 13,  3])
# slice the first 2 elements
my_array[:2]
'Output': array([14,  9])
# slice the last 3 elements
my_array[-3:]
'Output': array([ 5, 13,  3])
```

Basic Math Operations on Arrays: Universal Functions

The core power of NumPy is in its highly optimized vectorized functions for various mathematical, arithmetic, and string operations. In NumPy these functions are called universal functions. We'll explore a couple of basic arithmetic with NumPy 1-D arrays.

```
# create an array of even numbers between 2 and 10
my_array = np.arange(2,11,2)
'Output': array([ 2,  4,  6,  8, 10])
# sum of array elements
np.sum(my_array) # or my_array.sum()
'Output': 30
# square root
np.sqrt(my_array)
'Output': array([ 1.41421356,  2.        ,  2.44948974,  2.82842712,
                  3.16227766])
# log
np.log(my_array)
'Output': array([ 0.69314718,  1.38629436,  1.79175947,  2.07944154,
                  2.30258509])
# exponent
np.exp(my_array)
'Output': array([  7.38905610e+00,   5.45981500e+01,   4.03428793e+02,
                   2.98095799e+03,   2.20264658e+04])
```

Higher-Dimensional Arrays

As we've seen earlier, the strength of NumPy is its ability to construct and manipulate n-dimensional arrays with highly optimized (i.e., vectorized) operations. Previously, we covered the creation of 1-D arrays (or vectors) in NumPy to get a feel of how NumPy works.

This section will now consider working with 2-D and 3-D arrays. 2-D arrays are ideal for storing data for analysis. Structured data is usually represented in a grid of rows and columns. And even when data is not necessarily represented in this format, it is often transformed into a tabular form before doing any data analytics or machine learning. Each column represents a feature or attribute and each row an observation.

Also, other data forms like images are adequately represented using 3-D arrays. A colored image is composed of $n \times n$ pixel intensity values with a color depth of three for the red, green, and blue (RGB) color profiles.