**Numpy Arrays**

**What they are** – Numpy arrays store data as a grid or a matrix. They are defined in Python as data structures, and all arrays have the prefix np. in their data type. Numpy arrays have a grid structure, allowing them to store data in multiple dimensions. This makes arrays useful for storing large amounts of data, assuming they are all the same base datatype. This usefulness leads to Numpy arrays serving as the building blocks for more advanced modules and operations, such as Pandas dataframes.

**How they differ from lists** – All elements in Numpy arrays must consist of the same datatype (string, int, float, etc.) while lists can mix datatypes. This requirement, however, gives arrays much more versatility when it comes to elementwise mathematical operations, which cannot be performed on lists. While lists only operate in a single dimension, Numpy arrays, as previously mentioned, can store data along multiple dimensions. Additionally, Numpy arrays are not edited when altered or operated on, the array is instead deleted and recreated according to the operation.

**How to make them** – To create an array, first import the numpy module (using np as shorthand), and then use the command np.array, followed by the values you want within it. For example, typing test=np.array([8 2 9 7]) would create a 1-dimensional array called test with integer values of 8, 2, 9, and 7 (adding a .0 to each number would be necessary if float values were desired). An example of a 2d array is twod=np.array([8 2 9 7],

[6 2 0 1])

In other words, it is necessary to create a separate set of brackets on the next line when adding an additional row to an array, with a comma just before the line gap, while saving the closing parentheses until the end when creating a 2d array. In this example, the array twod is a 2x4 Numpy integer array. Array creation can also be combined with other functions or input data to avoid typing out every single element. For example, the line rand=np.random.randint(10, size = (6,3)) gives a 2-dimensional array with 6 rows an 3 columns, each element containing a random integer between 1 and 9.

**Indexing and Slicing** – To index a Numpy array (indexing can only retrieve a single value at a time), type in the row and column number of the value you wish to retrieve, being sure to start with 0 for the first row/column. From the previous examples, test[3] returns a value of 7 while twod[1, 1] returns a value of 2. To return multiple values at once, slice the array by taking a range of indices. From the previous examples, test[0:2] returns [8 2] while twod[0:1,1:3] returns [2 9]. Note that while the starting index is inclusive, the ending index is not, and that, unlike indexing, colons are used to specify the value range. To include the second row of twod in the previous slice, twod[:,1:3] would have to be ran. In this case, the : without any numbers tells Python to return all rows of the array, giving an output of [2 9

[2 0]

**Array methods** – Arrays have a variety of methods associated with them. Some of the most useful (at least for working with streamflow data) are listed below.

**ndarray.mean()** – Returns the average of array elements along the axis defined in the parentheses.

**ndarray.min()** – Returns the lowest value along the axis defined in the parentheses.

**ndarray.max()** – Returns the highest value along the axis defined in the parentheses.

**ndarray.sort()** – Sorts elements based on value, either ascending or descending (defined in parentheses, along with the axis).

**ndarray.round()** – Rounds each array element to a specified number of decimals, defined within the parentheses, or the nearest integer if no decimal placing is defined (blank parentheses).

**ndarray.item()** – Returns a copy of an array element in the form of a standard Python scalar. Similar to indexing.

For each of the first three methods, leaving the parentheses blank tells Python to perform the operation over all axes/elements. For **ndarray.sort**, an axis must be defined or else the code will return an error.

**Array attributes** – Arrays also have a set of common attributes. Some key ones include **ndim**, the number of dimensions an array has, **shape**, the size of each array dimension, **size**, the total number of elements within the array, and **dtype**, the data type contained within the array. For example, test would have ndim=1, shape=(1,4), size=4, and dtype=’numpy.int16’, while twod would have ndim=2, shape=(2,4), size=8, and dtype=’numpy.int16’.

**Useful Numpy functions** – Numpy also includes a variety of useful functions, which can be used to operate on arrays. Many of these functions overlap with array methods. Some of the most useful for me during streamflow calculation were **np.mean**, **np.min**, **np.max**, **np.median**, **np.sort** and **np.round**, most of which overlap with an array method and are therefore described in the array methods section, though using ndarray rather than np as the prefix. The function **np.median** does not have a corresponding ndarray method; this function computes the median of an array, defined in np.median(array\_name). Note that, unlike with methods, the array name must be defined within the parentheses, this is true for all the functions mentioned. An axis can also be defined as an argument.