Author: Ian Castro

What are arrays?

Arrays are collections of data to which we can apply mathematical operations. They provide a convenient way to perform the same operation on every item in the array. Each item of an array is referred to as an "element" and we can pull individual elements out of an array by using the method `.item(#)`.

How can you create an array?

1) `make_array(value1,value2,...)` will create an array with the specified values

2) `np.arange(start, stop, step)` will create an array with numbers in a range

3) `tbl.column("col_name")` pulls out all of the values in the column of a table.

How does array indexing work?

Arrays use "zero-indexing". Indexes refer to the position of a value/item in a sequence. Python, with one of its quirks as a computer language, counts from 0. In other words, the first item is at index 0, the second item is at index 1, third at index 2, and so forth.

For example:

```
>>> test_array = make_array(5, 6, 7, 8)

>>> test_array.item(1)

6 ## Notice that it is NOT 5
```

How does math on arrays work?

If you want to do the same calculation on every single value of an array, all you need to do is apply the operator to the array. For example:

```
>>> test_array * 5

array([25, 30, 35, 40]) # Because we took test_array, and did
this: (5*5, 6*5, 7*5, and 8*5).
```

This allows us to do many calculations, such as unit conversions, very quickly when given a lot of numbers/data points.

If we want to do math between arrays, such as adding two arrays together, the rule is that both arrays must be the exact same length (i.e. have the same number of elements). In this case, we will be combining the corresponding elements at each index. For example:

```
>>> odd_nums = make_array(1, 3, 5, 7)

>>> even_nums = make_array(2, 4, 6, 8)

>>> odd_nums + even_nums

array([3, 7, 11, 15]) ## In this case, we add (1+2, 3+4, 5+6,
7+8) which gives us this result.
```

Combining two arrays of different lengths produces an error message:

```
>>> odd_nums = make_array(1, 3, 5, 7)

>>> even_nums = make_array(2, 4, 6, 8, 10, 12)

>>> odd_nums * even_nums

Error

# This would give us this: (1*2, 3*4, 5*6, 7*8, ? * 10, ? * 12)

# Python does not know what to do with the missing numbers
multiplied to 10 and 12, so it will output an error.
```

np.arange

`np.arange(start, stop, step)` allows us to create arrays of sequential numbers, based on a start value, an end value, and a "step". We use this to create large arrays, since it's a bit difficult to write an array (for example) from 0 to 10000 out by hand using `make_array(...)`. This function takes a minimum of 1 argument and up to 3 arguments; check the Python reference sheet for more information. By default, step = 1 and start = 0.

The arguments are as follows:

start = the first value to include in the array

stop = the range stops before this number; this number will NOT be included in the final array

step = the difference between each consecutive number

For example:

```
>>> np.arange(5, 55, 5)

array([5, 10, 15, 20, 25, 30, 35, 40, 45, 50]) # Notice that 55 is not included

>>> np.arange(10)

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) ## Notice that the length of this array is 10, starting at 0 and ending at 9
```

Please use the followups to ask any questions you have about arrays so far!

Additional Arrays Functions

In general, we can call various functions on arrays to perform different calculations with sequences or collections of numbers.

Summary functions

As we've seen, we have a few summary functions that will tell us information about the array. Each of these functions can be used by calling the function directly on the array, and will generally return floats or integers. Here are a few; you may find other functions we may use on the Python Reference Sheet.

1) We can find the average of an array using `np.average()` or `np.mean()`
2) We can calculate the sum of all values using `sum()`

3) We can find the largest value using `max()`, and conversely, the smallest value using `min()`
4) Lastly, we can find the number of elements in the array, also known as the length, using `len()`

There are also other functions that allow us to add values to an array. Later on in the course, we'll see just how valuable this functionality can be in data science. One such function is:

np.append

Later in this course, we will use arrays to store values we calculate. We can use `np.append(array, value)` to append or "attach" a value to the end of an array. Generally, if we want to "update" the array in Python so we can use the array containing that new value, we will need to reassign it to the same name. See the example below.

```
>>> values = make_array(9, 18, 27, 36)  #len(values) == 4

>>> values = np.append(values, 45) # Attach 45 to the end of the values array, and then update the values array to include 45

>>> values

array([9, 18, 27, 36, 45]) # len(values) == 5, now that 45 is part of the values array
```