In this video we will be covering numpy in 1D, in particular ND arrays. Numpy is a library for scientific computing. It has many useful functions. There are many other advantages like speed and memory. Numpy is also the basis for pandas. So check out our pandas video.

In this video we will be covering the

* basics and array creation,
* indexing and slicing,
* basic operations,
* universal functions.

Let's go over how to create a numpy array.

A Python list is a container that allows you to store and access data. Each element is associated with an index.

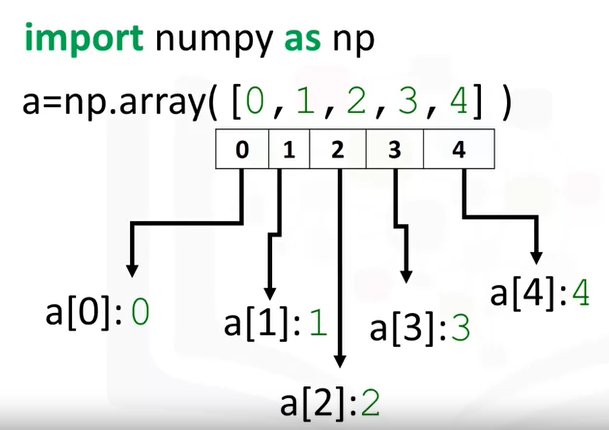
We can access each element using a square bracket as follows.

A numpy array or ND array is similar to a list.

It's usually fixed in size and each element is of the same type, in this case integers.

We can cast a list to a numpy array by first importing numpy.

We then cast the list as follows;

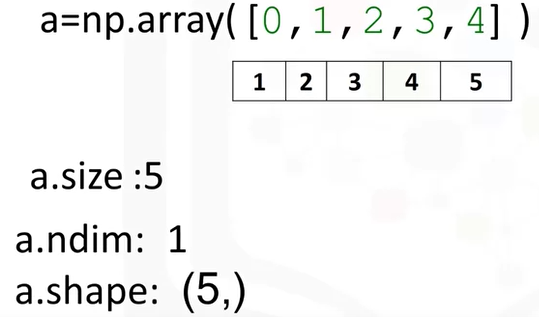


We can access the data via an index. As with the list, we can access each element with an integer and a square bracket. The value of a is stored as follows.

If we check the type of the array we get, numpy.ndarray. As numpy arrays contain data of the same type, we can use the attribute dtype to obtain the data type of the array's elements. In this case a 64-bit integer.

**BASIC ARRAY**

Let's review some basic array attributes using the array a. The attribute size is the number of elements in the array.



As there are five elements the result is five. The next two attributes will make more sense when we get to higher dimensions, but let's review them.

The **attribute ndim** represents the number of array dimensions or the rank of the array, in this case one.

The **attribute shape** is a tuple of integers indicating the size of the array in each dimension.

We can create a numpy array with real numbers.

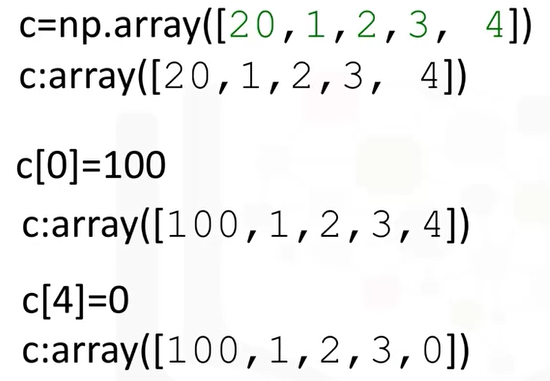
When we check the type of the array, we get **numpy.ndarray**.

If we examine the attribute D type, we see float64 as the elements are not integers. There were many other attributes, check out numpy.org.

**INDEXING METHODS**

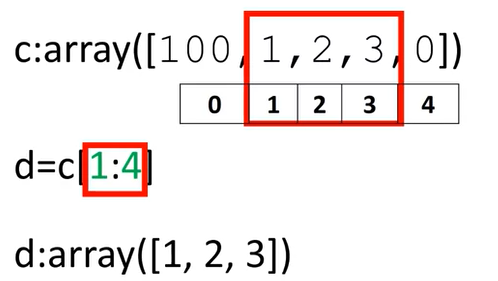
Let's review some indexing and slicing methods.

We can change the first element of the array to 100 as follows. The array's first value is now 100. We can change the fifth element of the array as follows. The fifth element is now zero.

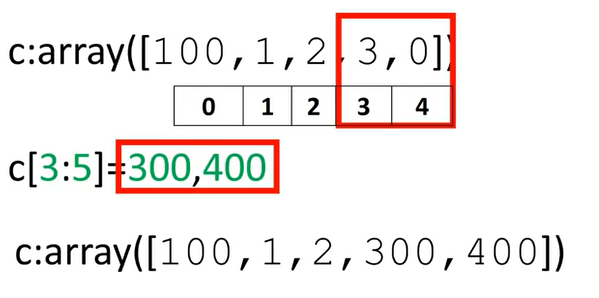


**SLICING METHODS**

Like lists and tuples we can slice a NumPy array. The elements of the array correspond to the following index. We can select the elements from one to three and assign it to a new numpy array d as follows. The elements in d correspond to the index.

****

**Like lists, we do not count the element corresponding to the last index.** We can assign the corresponding indices to new values as follows. The array c now has new values.



See the labs or numpy.org for more examples of what you can do with numpy.

c=np.array([20,1,2,3,4])

c[0]=100

c[0]=2

= 2

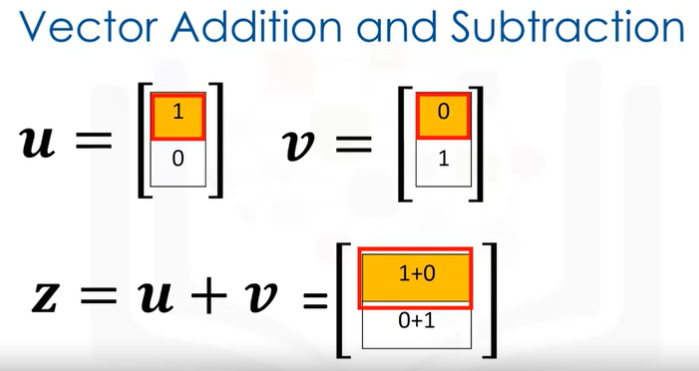
**BASIC OPERATIONS**

Numpy makes it easier to do many operations that are commonly performed in data science. The same operations are usually computationally faster and require less memory in numpy compared to regular Python.

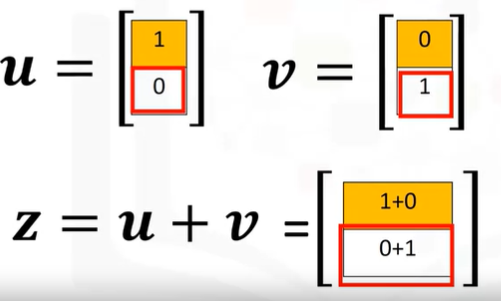
**Euclidian vectors**

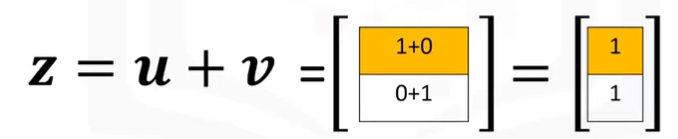
Let's review some of these operations on one-dimensional arrays. We will look at many of the operations in the context of Euclidian vectors to make things more interesting. Vector addition is a widely used operation in data science.

Consider the vector u with two elements, the elements are distinguished by the different colors. Similarly, consider the vector v with two components. In vector addition, we create a new vector in this case z. The first component of z is the addition of the first component of vectors u and v.



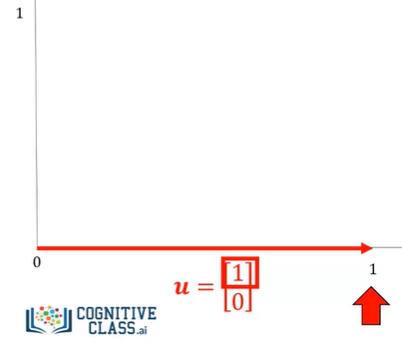
Similarly, the second component is the sum of the second components of u and v. This new vector z is now a linear combination of the vector u and v.

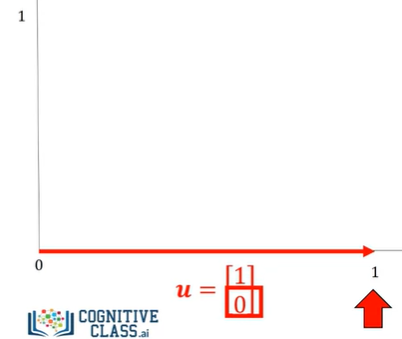




Representing vector addition with line segment or arrows is helpful. The first vector is represented in red. The vector will point in the direction of the two components.

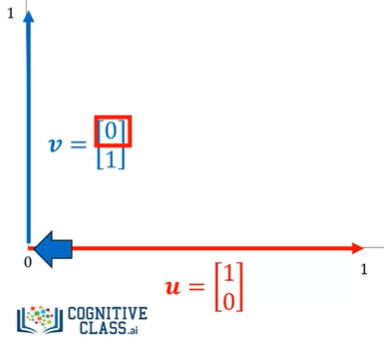
The first component of the vector is one. As a result the arrow is offset one unit from the origin in the horizontal direction.



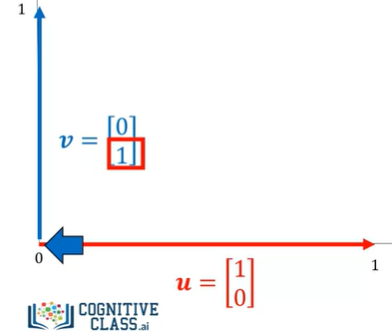


The second component is zero, we represent this component in the vertical direction. As this component is zero, the vector does not point in the horizontal direction.

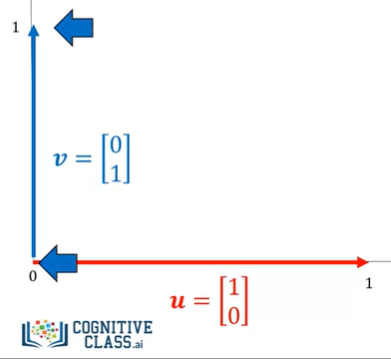
We represent the second vector in blue. The first component is zero, therefore the arrow does not point to the horizontal direction.



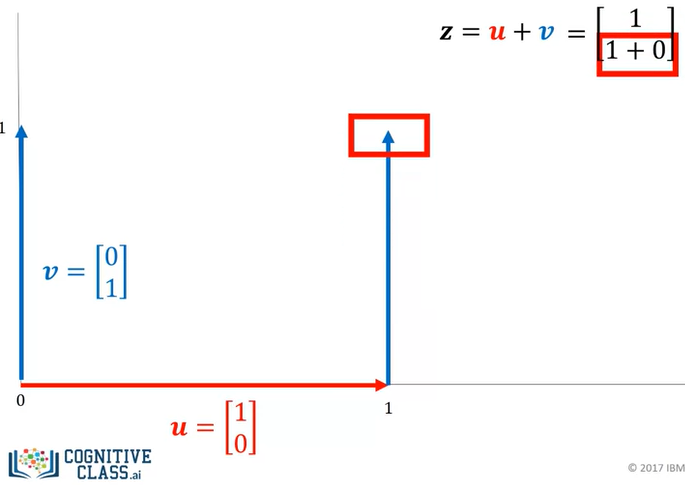
The second component is one.



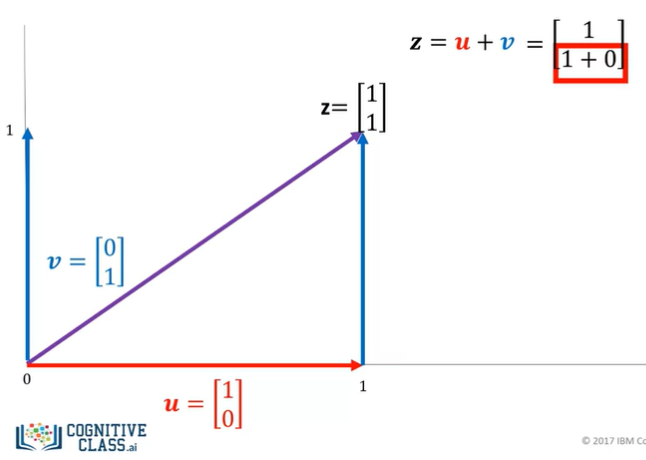
As a result the vector points in the vertical direction one unit.



When we add the vector u and v, we get the new vector z. We add the first component, this corresponds to the horizontal direction. We also add the second component.

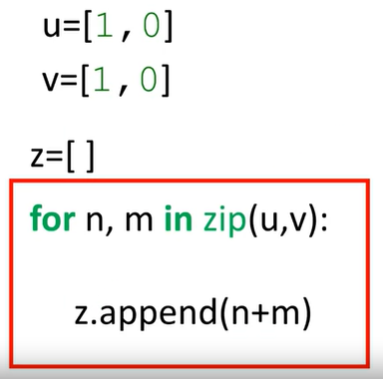


It's helpful to use the tip to tail method when adding vectors, placing the tail of the vector v on the tip of vector u.

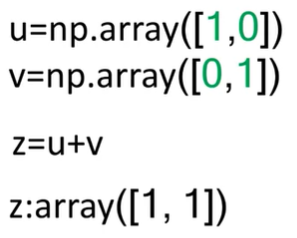


The new vector z is constructed by connecting the base of the first vector u with the tail of the second v.

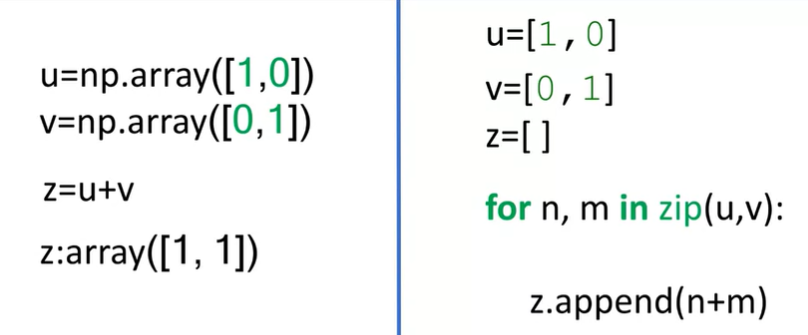
The following three lines of code we'll add the two lists and place the result in the list z.



We can also perform vector addition with **one line of NumPy code**.

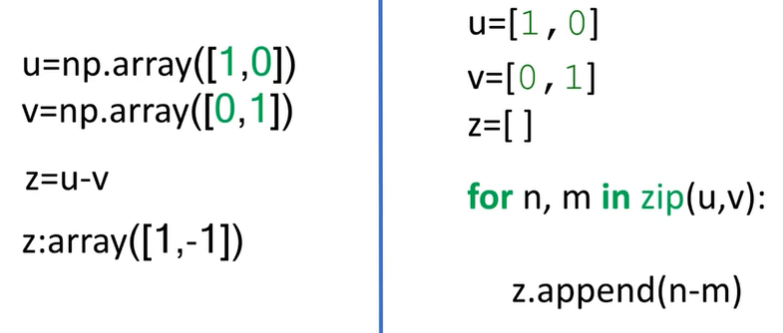


It would require multiple lines to perform vector subtraction on two lists as shown on the right side of the screen.



In addition, the numpy code will run much faster. This is important if you have lots of data.

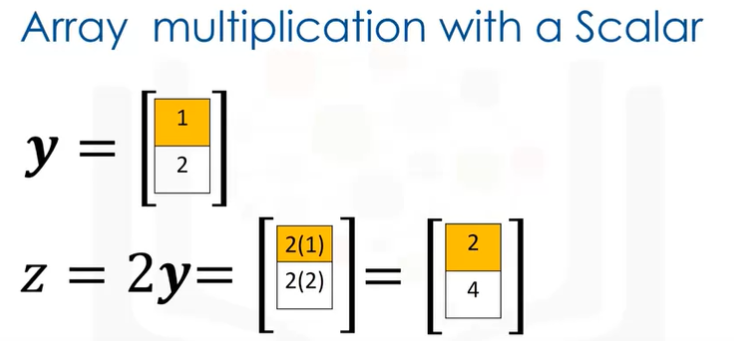
We can also perform vector subtraction by changing the addition sign to a subtraction sign.



It would require multiple lines perform vector subtraction on two lists as shown on the right side of the screen.

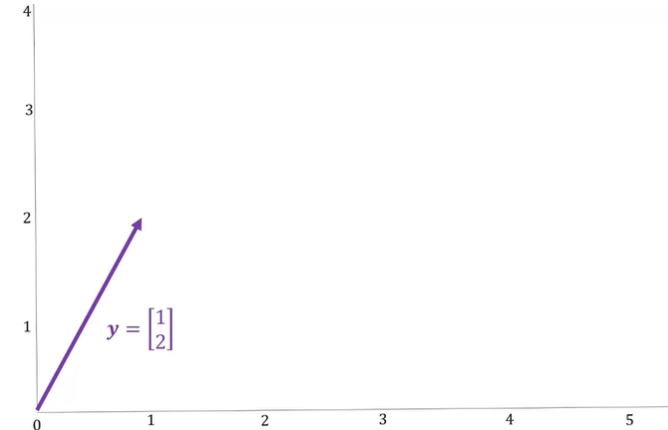
Vector multiplication with a scalar is another commonly performed operation.

Consider the vector y, each component is specified by a different color. We simply multiply the vector by a scalar value in this case two. Each component of the vector is multiplied by two, in this case each component is doubled.

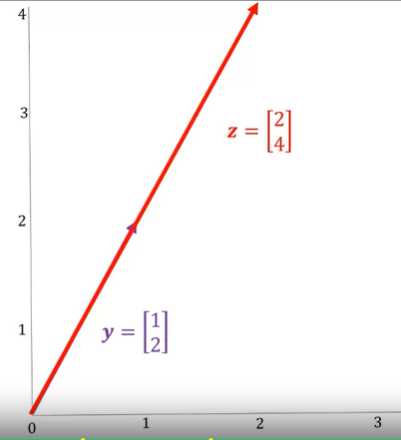


We can use the line segment or arrows to visualize what's going on.

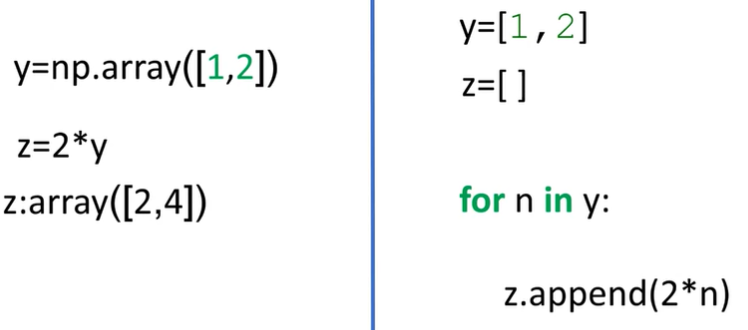
The original vector y is in purple.



After multiplying it by a scalar value of two, the vector is stretched out by two units as shown in red. The new vector is twice as long in each direction.



Vector multiplication with a scalar only requires one line of code using numpy. It would require multiple lines to perform the same task as shown with Python lists as shown on the right side of the screen.

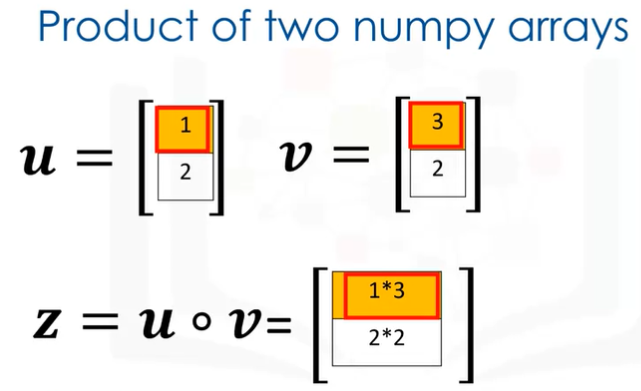


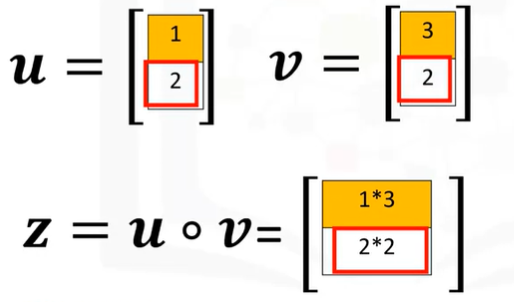
In addition, the operation would also be much slower.

**Hadamard product**

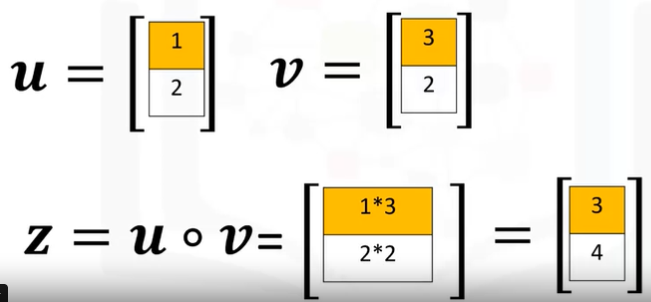
Hadamard product is another widely used operation in data science.

Consider the following two vectors, u and v. The Hadamard product of u and v is a new vector z. The first component of z is the product of the first element of u and v.

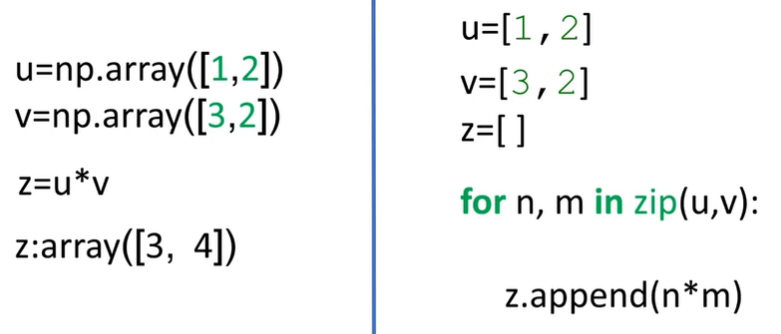


Similarly, the second component is the product of the second element of u and v. 

The resultant vector consists of the entry wise product of u and v.



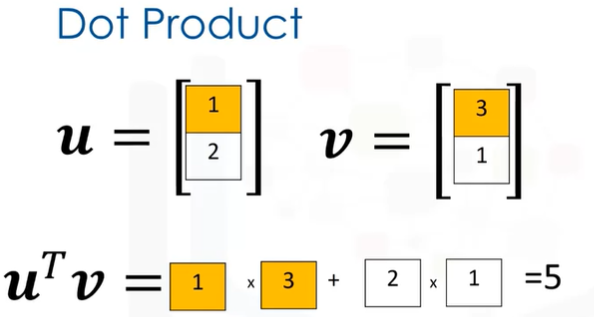
We can also perform hadamard product with one line of code in numpy. It would require multiple lines to perform hadamard product on two lists as shown on the right side of the screen.

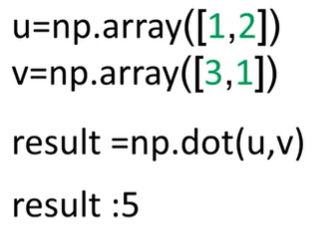


**dot product**

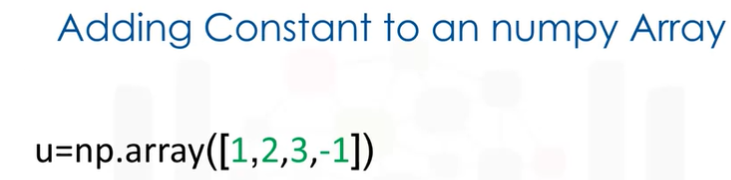
The dot product is another widely used operation in data science. Consider the vector u and v, the dot product is a single number given by the following term and represents how similar two vectors are.

We multiply the first component from v and u, we then multiply the second component and add the result together. The result is a number that represents how similar the two vectors are.



We can also perform dot product using the numpy function dot 

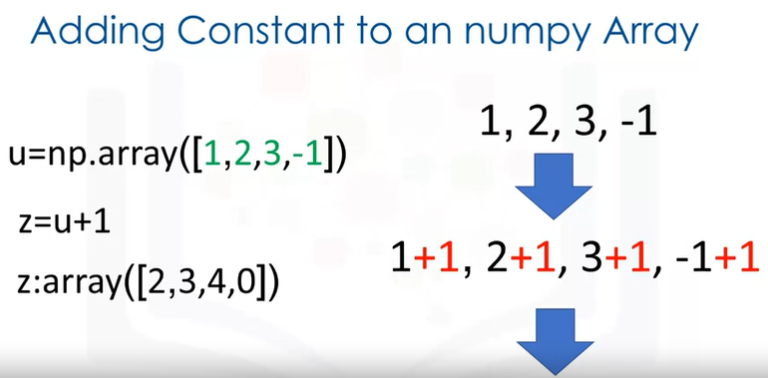
and assign it with the variable result as follows.



How would you perform the dot product between the numpy arrays **u** and **v**

**= np.dot(u,v)**

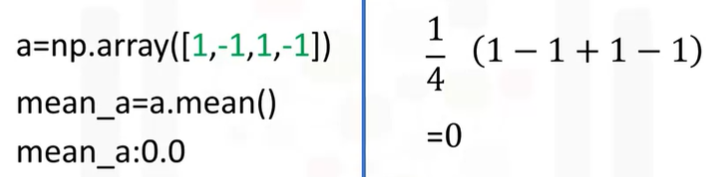
Consider the array u, the array contains the following elements. If we add a scalar value to the array, numpy will add that value to each element. This **property is known as broadcasting.**



**universal function**

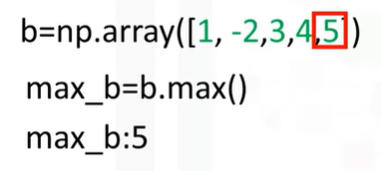
A universal function is a function that operates on ND arrays. We can apply a universal function to a numpy array.

Consider the arrays a, we can calculate the mean or average value of all the elements in a using the method mean. This corresponds to the average of all the elements. In this case the result is zero.



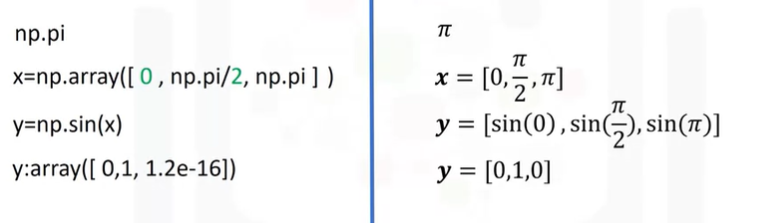
**Max**

There are many other functions. For example, consider the numpy arrays b. We can find the maximum value using the method five. We see the largest value is five, therefore the method max returns a five.



**Map numpy arrays to new numpy arrays**

We can use numpy to create functions that map numpy arrays to new numpy arrays. Let's implement some code on the left side of the screen and use the right side of the screen to demonstrate what's going on.



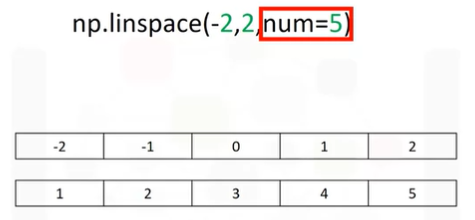
We can access the value of pie in numpy as follows. We can create the following numpy array in radians. This array corresponds to the following vector.

**SIN function**

We can apply the function sin to the array x and assign the values to the array y. This applies the sin function to each element in the array, this corresponds to applying the sine function to each component of the vector. The result is a new array y, where each value corresponds to a sine function being applied to each element in the array x.

**Line space**

A useful function for plotting mathematical functions is line space. Line space returns evenly spaced numbers over specified interval. We specify the starting point of the sequence, the ending point of the sequence. The parameter num indicates the number of samples to generate, in this case five.



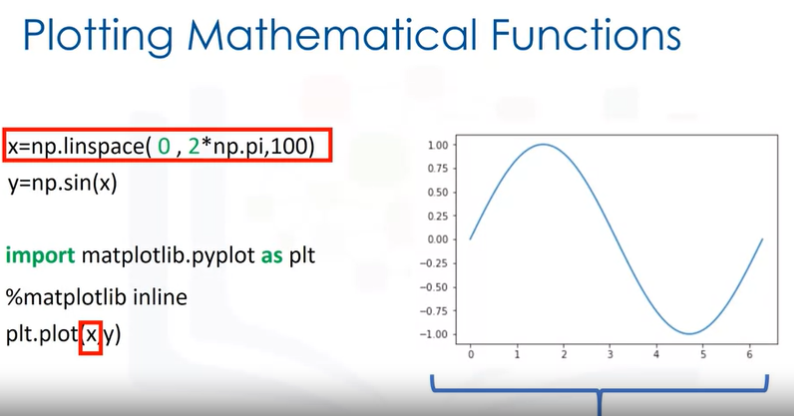
The space between samples is one. If we change the parameter num to nine, we get nine evenly spaced numbers over the integral from negative two to two. The result is the difference between subsequent samples is 0.5 as opposed to one as before.

We can use the function line space to generate 100 evenly spaced samples from the interval zero to two pie. We can use the numpy function sin to map the array x to a new array y.

**pyplot**

We can import the library pyplot as plt to help us plot the function. As we are using a Jupiter notebook, we use the command **matplotlib inline** to display the plot.

The following command plots a graph.



The first input corresponds to the values for the horizontal or x-axis. The second input corresponds to the values for the vertical or y-axis. There's a lot more you can do with numpy.

Check out the labs and numpy.org for more. Thanks for watching this video.