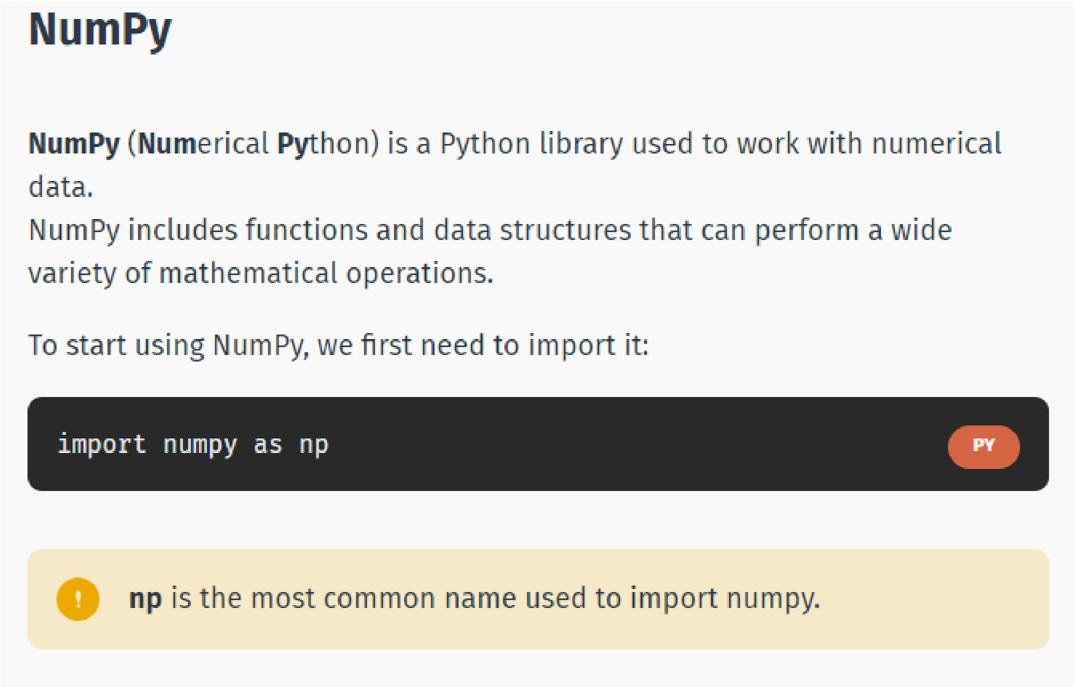


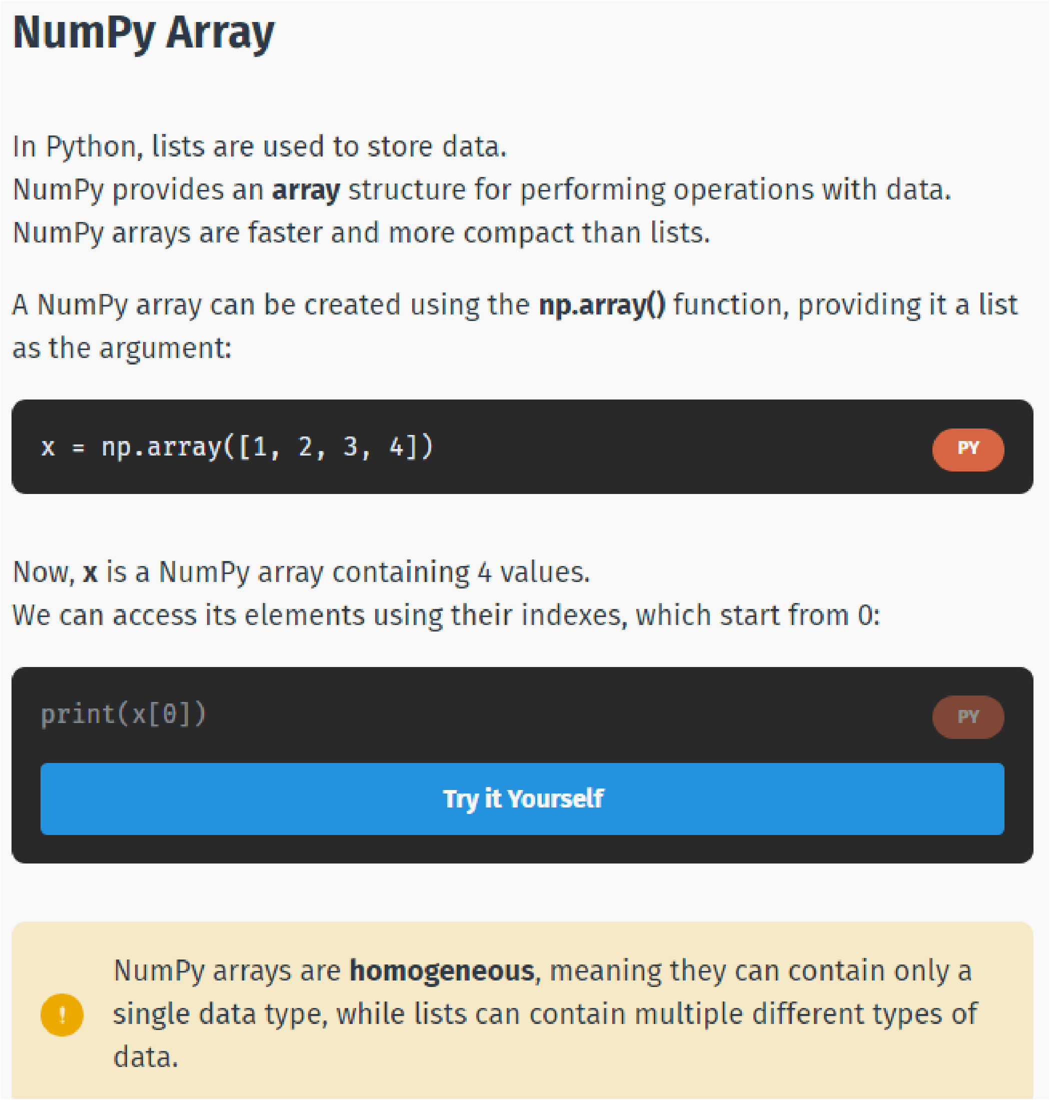
NumPy

NumPy (**Num**erical **Py**thon) is a Python library used to work with numerical data.
NumPy includes functions and data structures that can perform a wide variety of mathematical operations.

To start using NumPy, we first need to import it:

```
import numpy as np
```

np is the most common name used to import numpy.



NumPy Array

In Python, lists are used to store data.
NumPy provides an **array** structure for performing operations with data.
NumPy arrays are faster and more compact than lists.

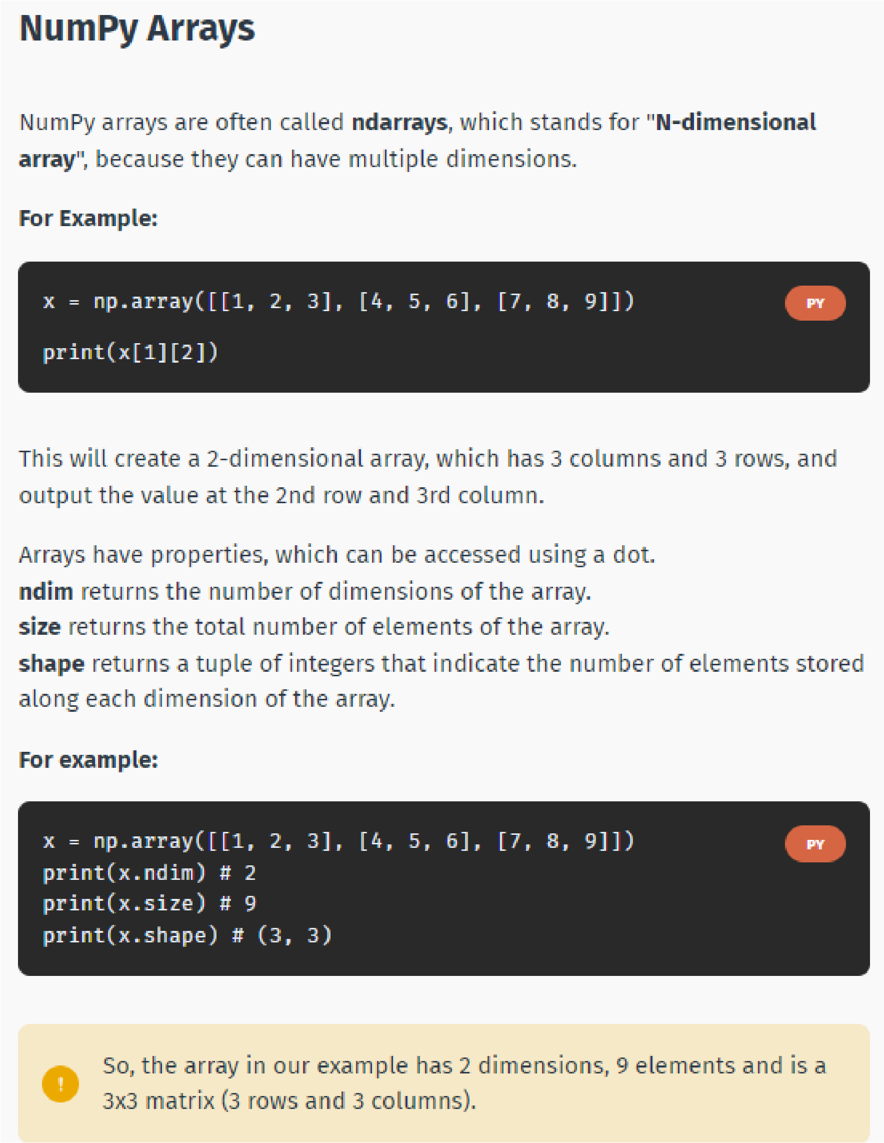
A NumPy array can be created using the **np.array()** function, providing it a list as the argument:

```
x = np.array([1, 2, 3, 4])
```

Now, **x** is a NumPy array containing 4 values.
We can access its elements using their indexes, which start from 0:

```
print(x[0])
```

NumPy arrays are **homogeneous**, meaning they can contain only a single data type, while lists can contain multiple different types of data.



NumPy Arrays

NumPy arrays are often called **ndarrays**, which stands for "**N-dimensional array**", because they can have multiple dimensions.

For Example:

```
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(x[1][2])
```

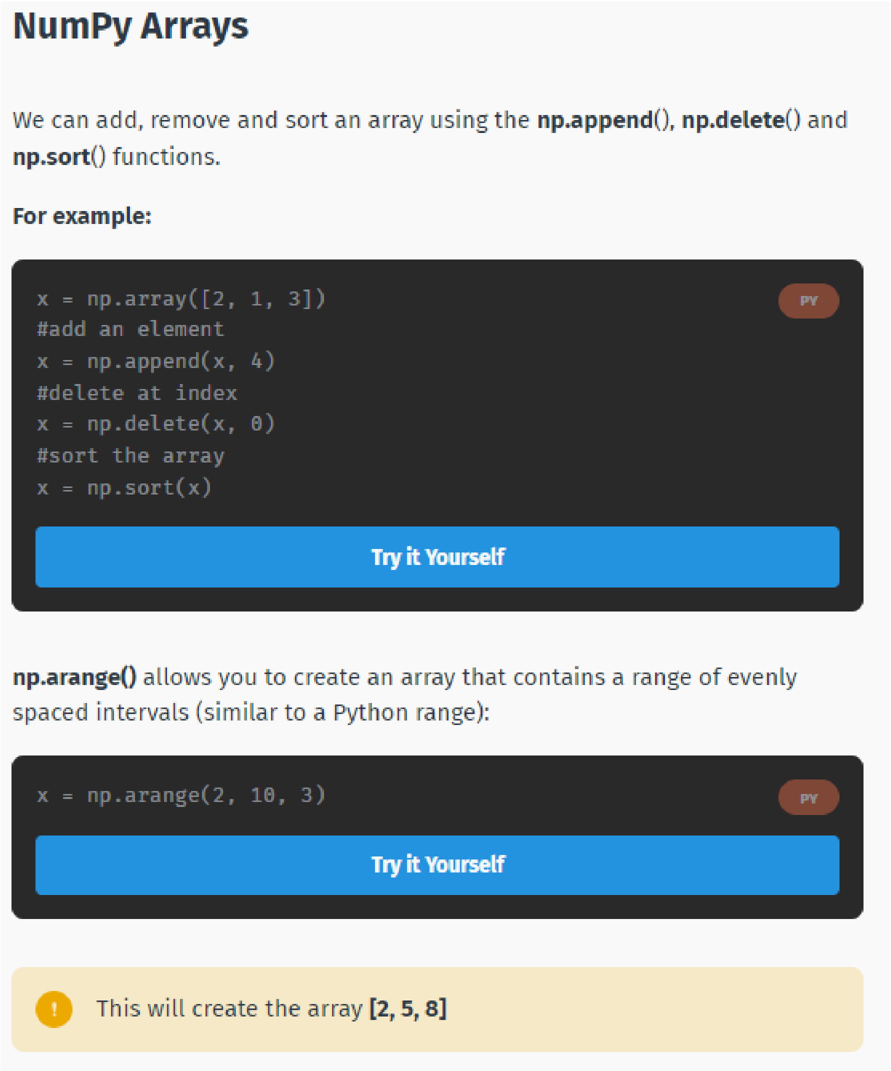
This will create a 2-dimensional array, which has 3 columns and 3 rows, and output the value at the 2nd row and 3rd column.

Arrays have properties, which can be accessed using a dot.
ndim returns the number of dimensions of the array.
size returns the total number of elements of the array.
shape returns a tuple of integers that indicate the number of elements stored along each dimension of the array.

For example:

```
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(x.ndim) # 2
print(x.size) # 9
print(x.shape) # (3, 3)
```

! So, the array in our example has 2 dimensions, 9 elements and is a 3x3 matrix (3 rows and 3 columns).



NumPy Arrays

We can add, remove and sort an array using the **np.append**(), **np.delete**() and **np.sort**() functions.

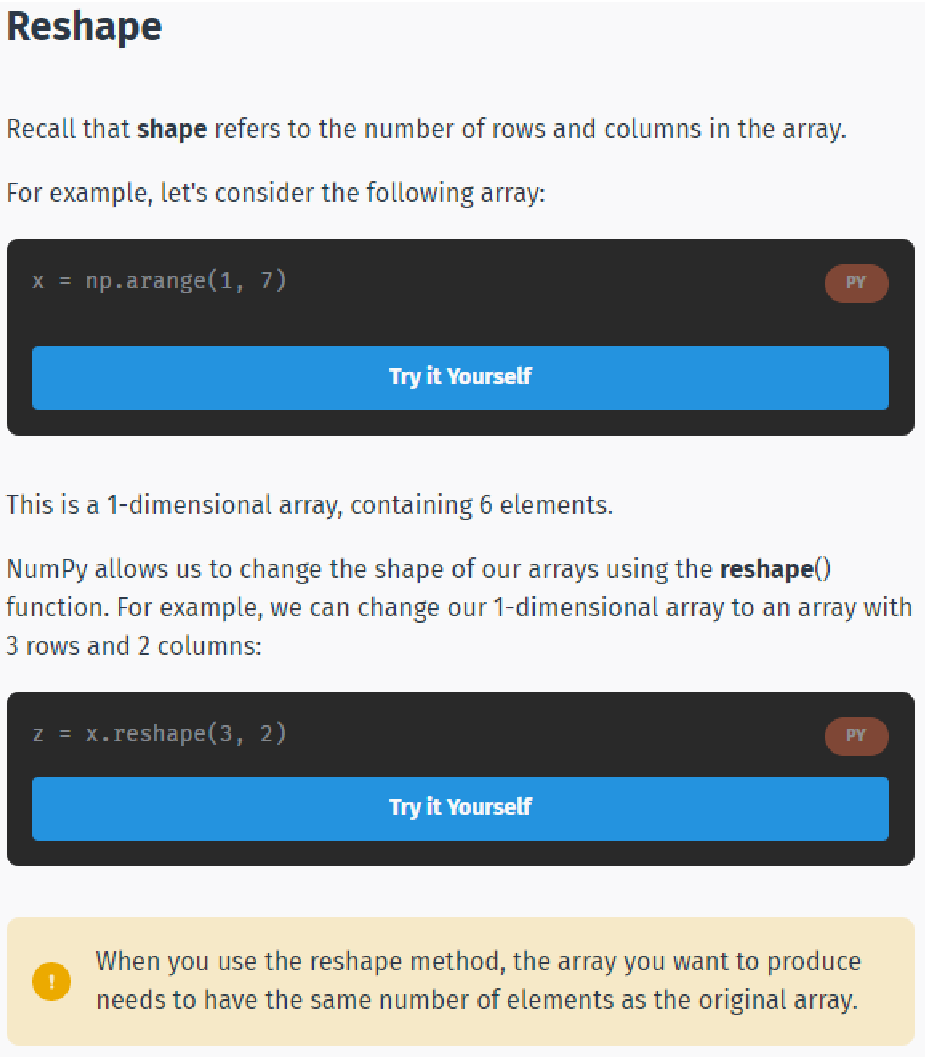
For example:

```
x = np.array([2, 1, 3])
#add an element
x = np.append(x, 4)
#delete at index
x = np.delete(x, 0)
#sort the array
x = np.sort(x)
```

np.arange() allows you to create an array that contains a range of evenly spaced intervals (similar to a Python range):

```
x = np.arange(2, 10, 3)
```

This will create the array **[2, 5, 8]**



Reshape

Recall that **shape** refers to the number of rows and columns in the array.

For example, let's consider the following array:

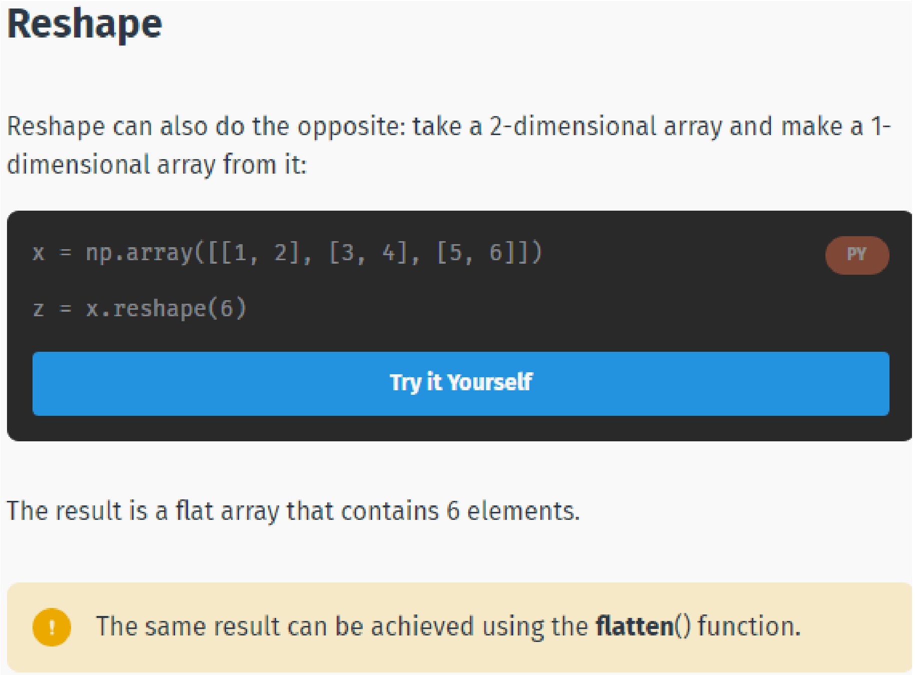
```
x = np.arange(1, 7)
```

This is a 1-dimensional array, containing 6 elements.

NumPy allows us to change the shape of our arrays using the **reshape**() function. For example, we can change our 1-dimensional array to an array with 3 rows and 2 columns:

```
z = x.reshape(3, 2)
```

When you use the reshape method, the array you want to produce needs to have the same number of elements as the original array.



Reshape

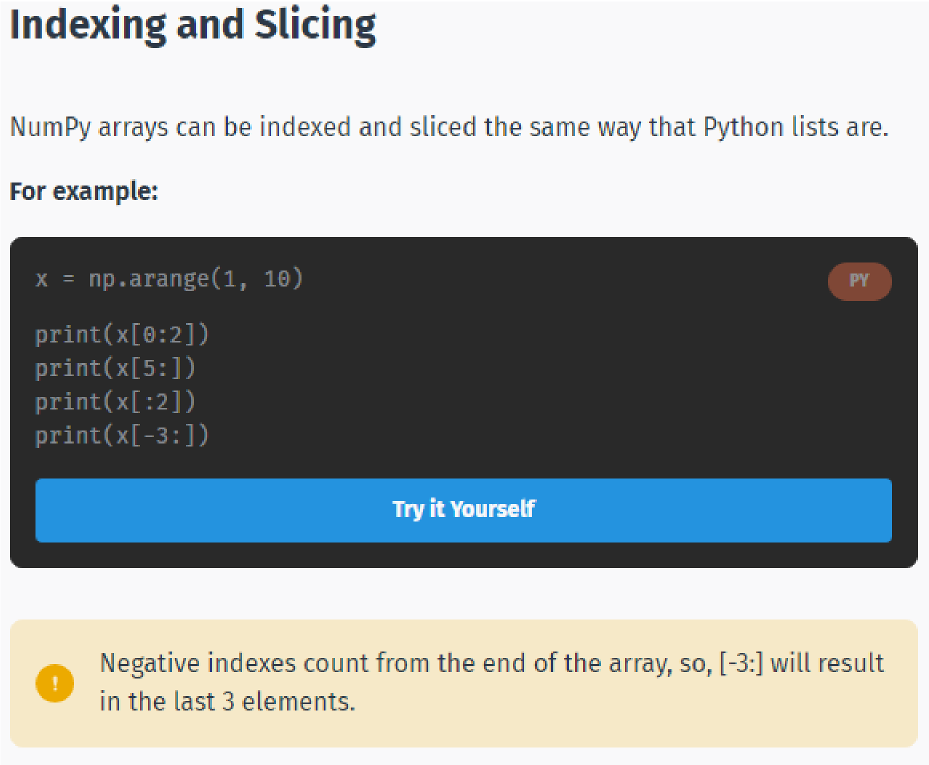
Reshape can also do the opposite: take a 2-dimensional array and make a 1-dimensional array from it:

```
x = np.array([[1, 2], [3, 4], [5, 6]])

z = x.reshape(6)
```

The result is a flat array that contains 6 elements.

The same result can be achieved using the **flatten**() function.



Indexing and Slicing

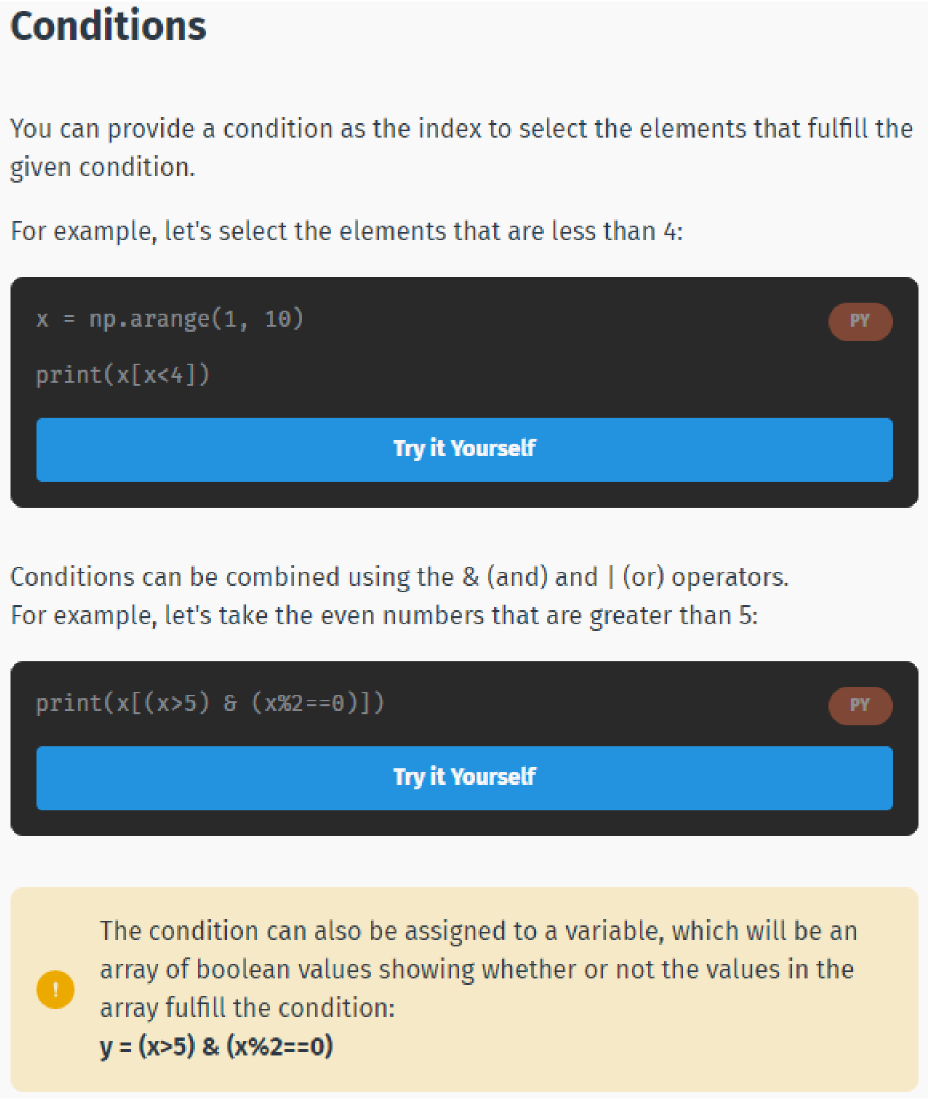
NumPy arrays can be indexed and sliced the same way that Python lists are.

For example:

```
x = np.arange(1, 10)

print(x[0:2])
print(x[5:])
print(x[:2])
print(x[-3:])
```

Negative indexes count from the end of the array, so, [-3:] will result in the last 3 elements.



Conditions

You can provide a condition as the index to select the elements that fulfill the given condition.

For example, let's select the elements that are less than 4:

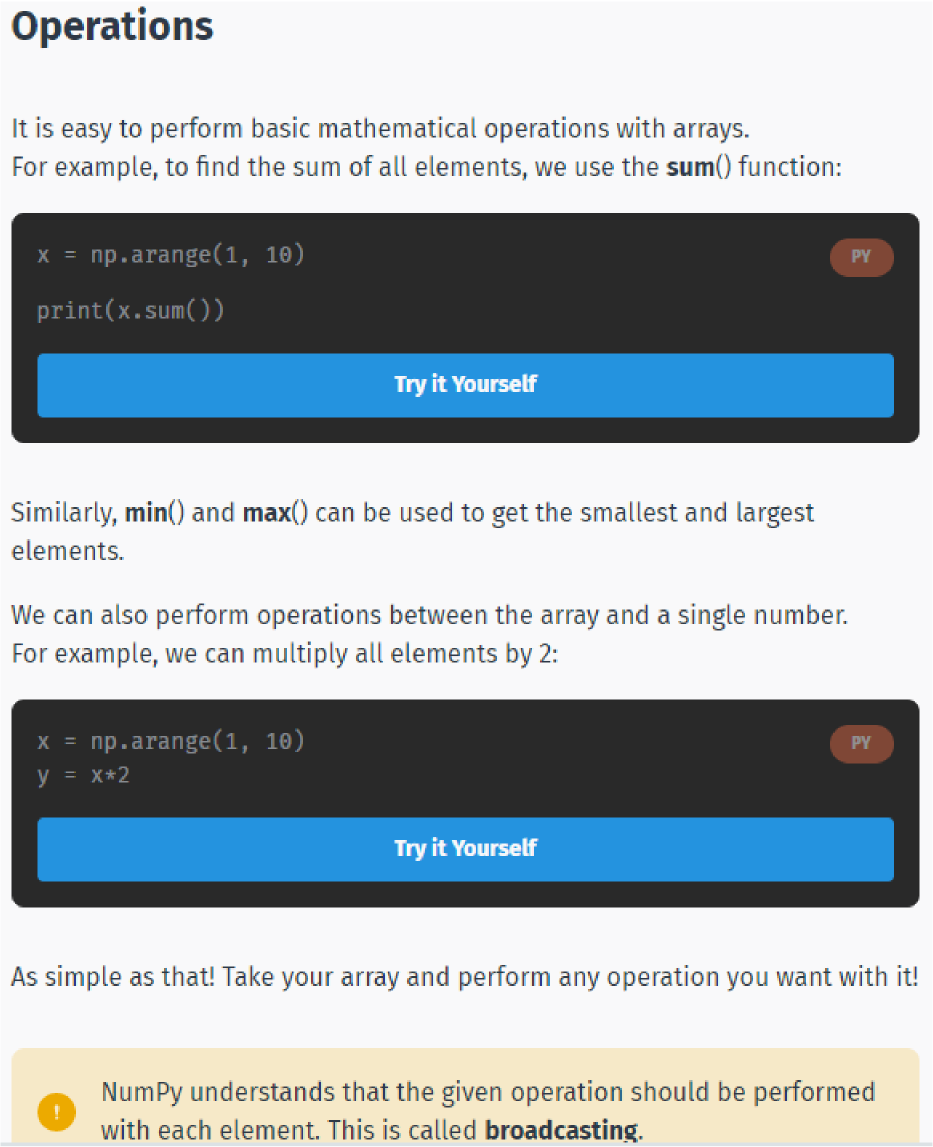
```
x = np.arange(1, 10)

print(x[x<4])
```

Conditions can be combined using the & (and) and | (or) operators.
For example, let's take the even numbers that are greater than 5:

```
print(x[(x>5) & (x%2==0)])
```

The condition can also be assigned to a variable, which will be an array of boolean values showing whether or not the values in the array fulfill the condition:
y = (x>5) & (x%2==0)



Operations

It is easy to perform basic mathematical operations with arrays.
For example, to find the sum of all elements, we use the **sum**() function:

```
x = np.arange(1, 10)

print(x.sum())
```

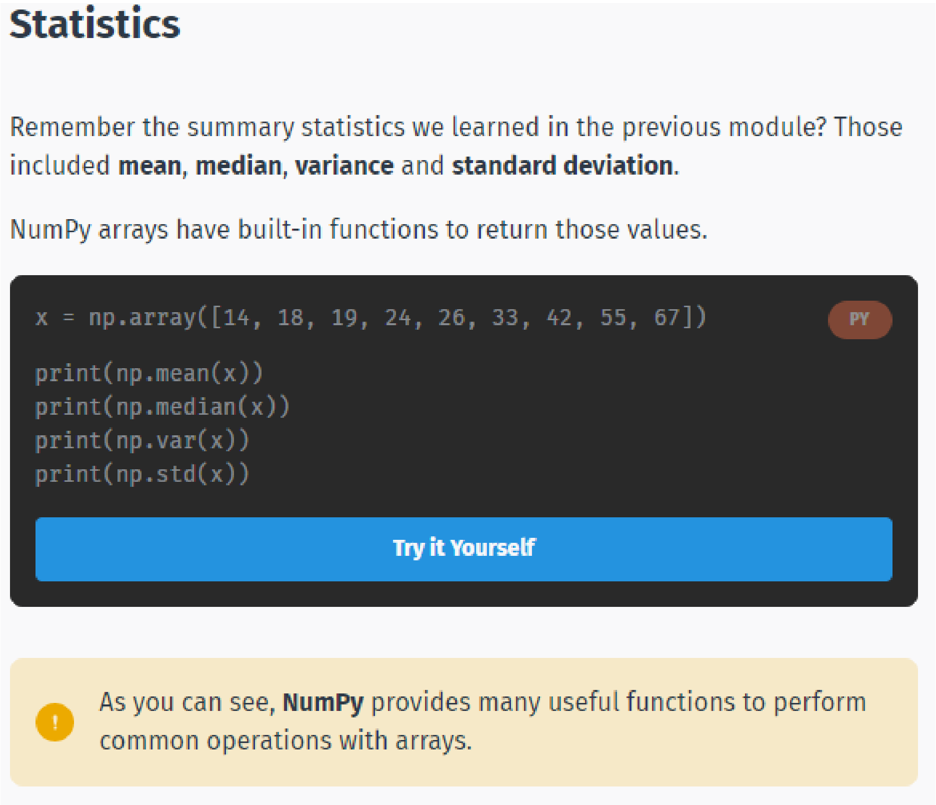
Similarly, **min**() and **max**() can be used to get the smallest and largest elements.

We can also perform operations between the array and a single number.
For example, we can multiply all elements by 2:

```
x = np.arange(1, 10)
y = x*2
```

As simple as that! Take your array and perform any operation you want with it!

! NumPy understands that the given operation should be performed with each element. This is called **broadcasting**.



Statistics

Remember the summary statistics we learned in the previous module? Those included **mean**, **median**, **variance** and **standard deviation**.

NumPy arrays have built-in functions to return those values.

```
x = np.array([14, 18, 19, 24, 26, 33, 42, 55, 67])

print(np.mean(x))
print(np.median(x))
print(np.var(x))
print(np.std(x))
```

As you can see, **NumPy** provides many useful functions to perform common operations with arrays.