NumPy Basics - Array Indexing and Slicing

WE'LL COVER THE FOLLOWING

- 3. Array Indexing: Accessing Single Elements
- 4. Array Slicing

3. Array Indexing: Accessing Single Elements

If we want to get and set the values of individual elements in the array, we need to be able to access single elements, correct? Accessing single elements is called indexing arrays.

Indexing in NumPy is similar to Python's standard list indexing. In a *1D* array, we can access the *ith* value by specifying the index of the element we need in **square brackets**. One important thing to remember here is that indexing in Python starts at zero.

Observe the inputs and outputs *(indicated by "#>" as start marker)* for the examples given below.

> **Note**: *When running the code, if you want to view the output in the console, you can add print statements if they are already not there, like at the end of this first code widget. I have omitted them to remove noise from the code so that you can fully focus on the important bits.*

```
# Input array
x1 = np.array([1, 3, 4, 4, 6, 4])

# Assess the first value of x1
x1[0]
#> 1

# Assess the third value of x1
x1[2]
```

```
#> 4


# To view the output, you can add print statemtents
print(x1[0])
```

We can use **negative indices** to index from the end of the array:

```
# Get the last value of x1
x1[-1]
#> 4

# Get the second last value of x1
x1[-2]
#> 6
```

If we have a multidimensional array, and want to access items based on both column and row, we can pass the row and column indices at the same time using a **comma-separated** tuple as shown in the examples below.

```
# In a multidimensional array, we need to specify row and column index. Given input array x2:
x2 = np.array([[3, 2, 5, 5],[0, 1, 5, 8], [3, 0, 5, 0]])
x2
#> array([[3, 2, 5, 5],
#>        [0, 1, 5, 8],
#>        [3, 0, 5, 0]])

# Value in 3rd row and 4th column of x2
x2[2,3]
#> 0

# 3rd row and last value from the 3rd column of x2
x2[2,-1]
#>0

# Replace value in 1st row and 1st column of x2 with 1
x2[0,0] = 1
#> array([[1,  2,  5,  5],
#>        [ 0,  1,  5,  8],
#>        [ 3,  0,  5,  0]])
```

4. Array Slicing

Slicing array is a way to access subarrays, i.e., accessing multiple or a range of elements from an array instead of individual items. In other words, when you slice arrays you get and set smaller subsets of items within larger arrays.

Again, we need to use square brackets to access individual elements. But this time, we also need the **slice notation**, **":"** to access *a slice* or a range of elements of a given array, *x*:

`x[start:stop:step]`

If we do not specify anything for *start*, *stop*, or *step*, NumPy uses the default values for these parameters: *start=0*, *stop=size of dimension*, and *step=1*.

Carefully go through all of the examples given below, and observe the output values for the different combinations of *slices*. As an exercise **play with the indices and observe the outputs**.

```
x1 = np.arange(10) # Input array
x1
#> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

# Get the first 5 elements of x
x1[:5]
#> array([0, 1, 2, 3, 4])

# Elements after index 4
x1[4:]
#> array([4, 5, 6, 7, 8, 9])

# From 4th to 6th position
x1[4:7]
#> array([4, 5, 6])

# Return elements at even place (every other element)
x1[ : : 2]
#> array([0, 2, 4, 6, 8])

#return elements from 1st position step by 2 (every other element starting at index 1)
x1[1::2]
#> array([1, 3, 5, 7, 9])
```

What do you think would happen if we specify a **negative step value**? In this case, the defaults for start and stop are swapped, a handy way to easily

reverse an array!

```
#reverse the array
x1[::-1]
#> array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

# reverse every other element starting from index 5
x1[5::-2]
#> array([5, 3, 1])
```

We can use this same approach with multi-dimensional slices. We can define multiple slices separated by commas:

```
x2 = np.array([[0,1,2], [3,4,5], [6,7,8]])
#> array([[0, 1, 2],
#>        [3, 4, 5],
#>        [6, 7, 8]])

x2[:2, :2] # Extract the first two rows and two columns
#> array([[ 0,  1],
#>        [ 3,  4]])

x2[:3, ::2]  # all rows, every other column
#>array([[0,  2],
#>       [ 3,  5],
#>       [ 6,  8]])
```

Again, try modifying the values and play with multi-dimensional arrays before moving ahead. We can also perform reverse operations on subarrays:

```
x2[::-1, ] # Reverse only the row positions

#> array([[ 6,  7,  8],
#>        [ 3,  4,  5],
#>        [ 0,  1,  2]])

x2[::-1, ::-1] # Reverse the row and column positions
#> array([[ 8,  7,  6],
#>        [ 5,  4,  3],
#>        [ 2,  1,  0]])
```

> **Note**: Array slices are not copies of the arrays. This means that if we want to do a modification on the array obtained from the slicing operation without changing the original array, we have to use the `copy()` method:

```
x2_subcopy = x2[::-1, ::-1].copy()
```