😊 NumPy Indexing and Selection

1. In this lab, we will discuss how to select elements or groups of elements from an array.
2. First, we import the NumPy library as np, then we create a simple array that contains digits from 0 to 10.

```
[1]: import numpy as np
```

```
[2]: #Creating sample array
     arr = np.arange(0,11)
```

```
[3]: #Show
     arr
```

```
[3]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

3. The Bracket Indexing and Selection is the most straightforward way to pick one or some elements of an array looks very similar to Python lists.

```
[4]: #Get a value at an index
     arr[8]
```

```
[4]: 8
```

```
[5]: #Get values in a range
     arr[1:5]
```

```
[5]: array([1, 2, 3, 4])
```

```
[6]: #Get values in a range
     arr[0:5]
```

```
[6]: array([0, 1, 2, 3, 4])
```

4. NumPy arrays differ from normal Python lists because of their ability to broadcast.
5. With lists, you can only reassign parts of a list with new parts of the same size and shape.
6. That is, if you wanted to replace the first 5 elements in a list with a new value, you would have to pass in a new 5 element list.
7. With NumPy arrays, you can broadcast a single value across a larger set of values.

[7]:
```
#Setting a value with index range (Broadcasting)
arr[0:5]=100

#Show
arr
```

[7]:
```
array([100, 100, 100, 100, 100,   5,   6,   7,   8,   9,  10])
```

[8]:
```
# Reset array, we'll see why I had to reset in  a moment
arr = np.arange(0,11)

#Show
arr
```

[8]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

8. Slicing a NumPy array creates a view, not a copy, so changes to the slice affect the original array.
9. To prevent this, use the copy() method to create an independent copy that can be modified without altering the original data.

[9]:
```
#Important notes on Slices
slice_of_arr = arr[0:6]

#Show slice
slice_of_arr
```

[9]:
```
array([0, 1, 2, 3, 4, 5])
```

[10]:
```
#Change Slice
slice_of_arr[:]=99

#Show Slice again
slice_of_arr
```

[10]:
```
array([99, 99, 99, 99, 99, 99])
```

Now note the changes also occur in our original array!

[11]:
```
arr
```

[11]:
```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

Data is not copied, it's a view of the original array! This avoids memory problems!

[12]:
```
#To get a copy, need to be explicit
arr_copy = arr.copy()

arr_copy
```

[12]:
```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

10. This code demonstrates basic indexing and slicing in a 2D NumPy array. You can access entire rows using a single index or individual elements using either double brackets or a comma-separated format. Slicing allows extraction of subarrays based on row and column ranges, offering flexibility in manipulating matrix-like data structures.

[13]:
```
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))

#Show
arr_2d
```

[13]:
```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

[14]:
```
#Indexing row
arr_2d[1]
```

[14]:
```
array([20, 25, 30])
```

[15]:
```
# Format is arr_2d[row][col] or arr_2d[row,col]

# Getting individual element value
arr_2d[1][0]
```

[15]:
```
20
```

[16]:
```
# Getting individual element value
arr_2d[1,0]
```

[16]:
```
20
```

[17]:
```
# 2D array slicing

#Shape (2,2) from top right corner
arr_2d[:2,1:]
```

[17]:
```
array([[10, 15],
       [25, 30]])
```

[18]:
```
#Shape bottom row
arr_2d[2]
```

[18]:
```
array([35, 40, 45])
```

[19]:
```
#Shape bottom row
arr_2d[2,:]
```

[19]:
```
array([35, 40, 45])
```

11. This code demonstrates the use of boolean indexing in NumPy. First, a 1D array from 1 to 10 is created. Then, a boolean condition (arr > 4) is applied, resulting in a boolean array.
12. This array is used to filter and retrieve only the values in arr that satisfy the condition. You can also dynamically compare with a variable (e.g., x = 2) to filter elements greater than that value. This is a powerful way to perform conditional filtering on arrays.

[20]:
```
arr = np.arange(1,11)
arr
```

[20]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[21]:
```
arr > 4
```

[21]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,
       True])
```

[22]:
```
bool_arr = arr>4
```

•••

[23]:
```
bool_arr
```

[23]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,
       True])
```

[24]:
```
arr[bool_arr]
```

[24]:
```
array([ 5,  6,  7,  8,  9, 10])
```

[25]:
```
arr[arr>2]
```

[25]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

[26]:
```
x = 2
arr[arr>x]
```

[26]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```