

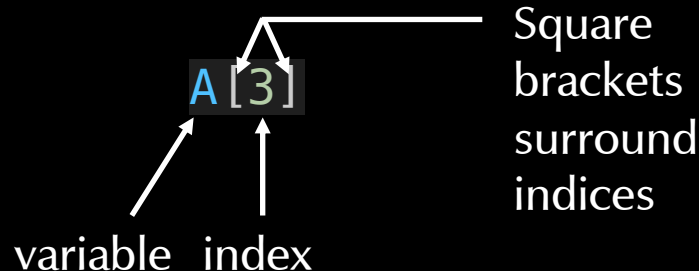
MR2020: Coding for METOC

Module 5: Indexing and Slicing in Python

Many objects can contain multiple elements (e.g., lists, tuples, dictionaries, NumPy arrays). In these cases, each data element has its own address in memory. We use indexing to access each individual element. Consider the list A below:

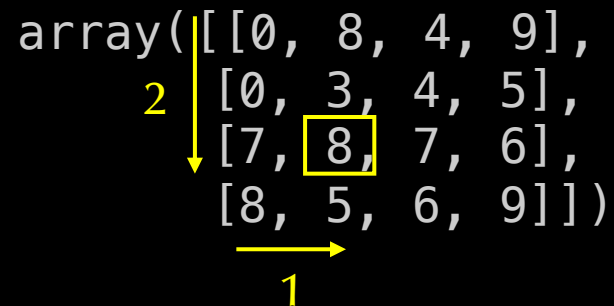
	<code>A = [1, 2, 4, 8, 16, 32, 64]</code>						
	↓	↓	↓	↓	↓	↓	↓
Python index	0	1	2	3	4	5	6
MATLAB index	1	2	3	4	5	6	7

Suppose I want to pull out the 8 from the list A. I would use the following code:



```
A = np.random.randint(10, size=(4,4))
```

```
array([[0, 8, 4, 9],  
       [0, 3, 4, 5],  
       [7, 8, 7, 6],  
       [8, 5, 6, 9]])
```



A[2,1] is 8.

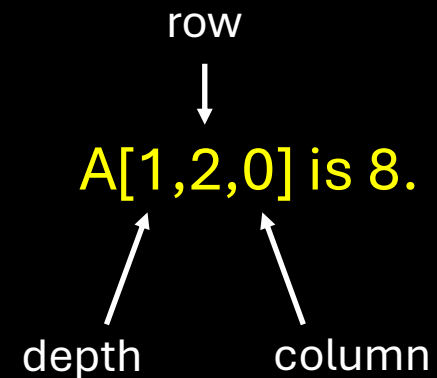
row

column

In METOC datasets, the indexing often corresponds to the y-axis (rows) and the x-axis (columns).

```
A = np.random.randint(10, size=(3,3,2))
```

```
array([[[2, 9],  
        [6, 2],  
        [6, 0]],  
       [[3, 5],  
        [0, 9],  
        [8, 8]],  
       [[9, 0],  
        [6, 5],  
        [7, 2]]])
```



In METOC datasets, the indexing often corresponds to the z-axis (depth), y-axis (rows) and the x-axis (columns). This often corresponds to height (or depth), north-south direction, and east-west direction. One way of thinking about this is that 1D arrays in Python have several columns in a single row. As we add additional dimensions, they appear sequentially to the left in the order of indices. So, we could add a fourth dimension (like time), and its index would appear first in a 4D array.

Boolean Indexing

Rather than explicitly listing indices to access elements in a list, array, or other multi-elemental data structure, we can select data based on whether it meets some condition. This is called **Boolean indexing**. Suppose we have two arrays:

A	B
<pre>array([[0, 8, 4, 9], [0, 3, 4, 5], [7, 8, 7, 6], [8, 5, 6, 9]])</pre>	<pre>array([[0, 1, 1, 0], [0, 0, 0, 1], [1, 0, 1, 0], [0, 1, 1, 1]])</pre>

If I run the following code:

```
A[B==1]
```

then I will get the following 1D array, which contains only the elements of A where the statement `B == 1` is True.

```
array([8, 4, 5, 7, 7, 5, 6, 9])
```

For more complicated Boolean statements, we could also define the condition as a Boolean variable first:

```
cond = (B==1)  
A[cond]
```

Sometimes, we need to grab the last few elements of a list, array, or string. In these cases, negative indices are useful.

	<code>A = [1, 2, 4, 8, 16, 32, 64]</code>						
	↓	↓	↓	↓	↓	↓	↓
Regular index	0	1	2	3	4	5	6
Negative index	-7	-6	-5	-4	-3	-2	-1

Or suppose I had a string that was a filename including its extension:

```
fname = 'testfile_20240712_132713.csv'
```

If I just wanted to grab the file extension (`' .csv '`), I could just run the following:

```
fname[-4:]
```

But what does `-4:` mean?

Slicing

We can also extract sequential subsets of data from a list, array, etc. using **slicing**.

```
A = [1, 2, 4, 8, 16, 32, 64]
```

Access all data up to the 4th element:

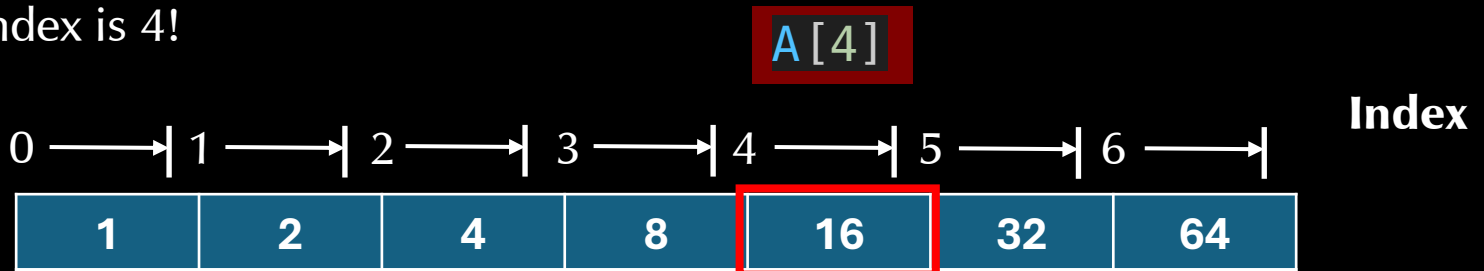
```
A[:4]  
# Returns [1, 2, 4, 8]
```



Why does this not include 16? Its index is 4!

Access data from 4th element onward:

```
A[4:]  
# Returns [16, 32, 64]
```



Slicing

We can also extract sequential subsets of data from a list, array, etc. using **slicing**.

```
A = [1, 2, 4, 8, 16, 32, 64]
```

Access all data up to the 4th element:

```
A[:4]  
# Returns [1, 2, 4, 8]
```



Why does this not include 16? Its index is 4!

Access data from 4th element onward:

```
A[4:]  
# Returns [16, 32, 64]
```



Slicing

We can also extract sequential subsets of data from a list, array, etc. using **slicing**.

```
A = [1, 2, 4, 8, 16, 32, 64]
```

Access all data up to the 4th element:

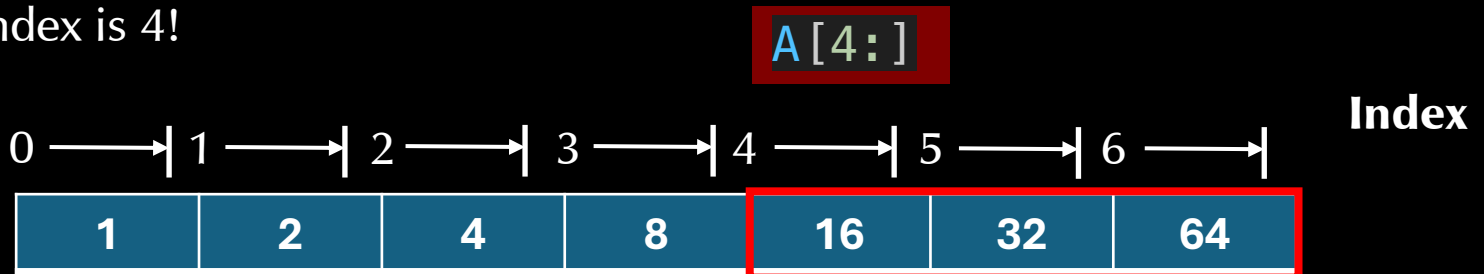
```
A[:4]  
# Returns [1, 2, 4, 8]
```



Why does this not include 16? Its index is 4!

Access data from 4th element onward:

```
A[4:]  
# Returns [16, 32, 64]
```



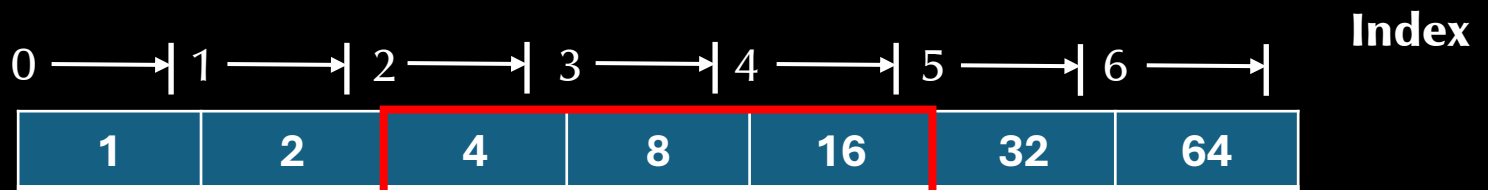
Slicing

We can also extract sequential subsets of data from a list, array, etc. using **slicing**.

```
A = [1,2,4,8,16,32,64]
```

Access elements 2 through 4:

```
A[2:5] # start:stop  
# Returns [4,8,16]
```



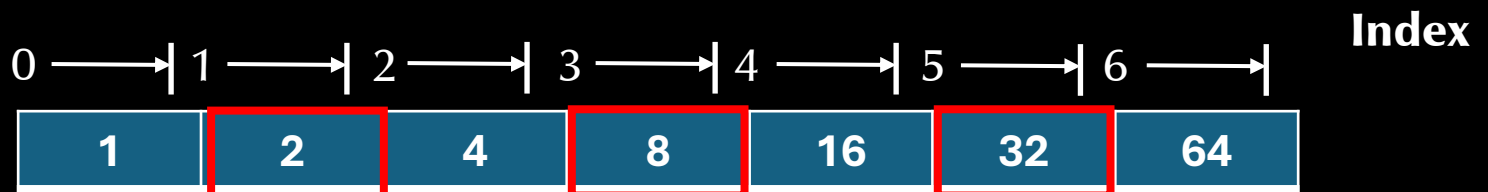
Slicing

We can also extract sequential subsets of data from a list, array, etc. using **slicing**.

```
A = [1,2,4,8,16,32,64]
```

Access every other element from 1 through 5:

```
A[1:6:2] # start:stop:step  
# Returns [2,8,32]
```



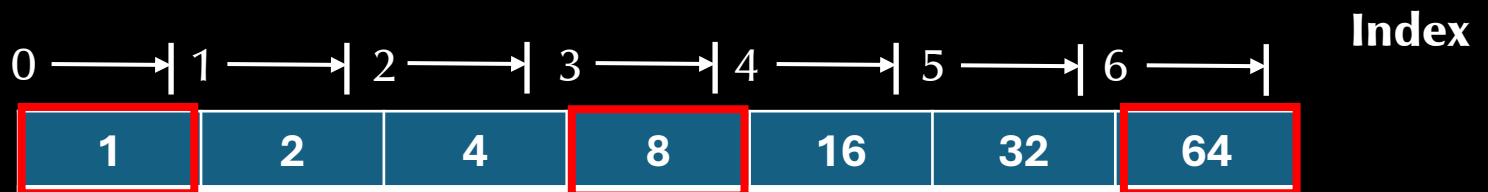
Slicing

We can also extract sequential subsets of data from a list, array, etc. using **slicing**.

```
A = [1, 2, 4, 8, 16, 32, 64]
```

Access every third element from the start:

```
A[::3] # Two colons mean start  
and stop at ends of array.  
# Returns [64]
```



Combining Indexing and Slicing

```
A = np.random.randint(8, size=(3,3))
```

```
array([[2, 0, 4],  
       [7, 1, 6],  
       [5, 3, 4]])
```

What would `A[1, :2]` return?

Combining Indexing and Slicing

```
A = np.random.randint(8, size=(3,3))
```

```
array([[2, 0, 4],  
       [7, 1, 6],  
       [5, 3, 4]])
```

What would `A[1, :2]` return?

```
array([7, 1])
```

This is the second row (index 1), and all elements up to the third element (slice :2).

Indexing Dictionaries

Dictionaries also contain multiple key-value pairs that we may want to access, but their indices are not the same as that for a sequential data type or NumPy array.

```
storms = {  
    'name': ['Alberto', 'Beryl', 'Chris', 'Debby', 'Ernesto', 'Francine'],  
    'category': ['TS', '5', 'TS', '1', '2', 'TS'],  
    'minpres': np.array([1000, 938, 1002, 989, 973, 999]),  
    'maxwind': np.array([45, 145, 35, 65, 85, 45]),  
    'landfall': [False, True, True, False, False, False]  
}
```

What if I want to access the minimum pressure from the above dictionary?

```
storms['minpres']
```

—————> Code the name of the key (as a string) inside square brackets.

I can also access a specific element within the data structure contained in the returned value. The first square brackets grab the value at the specified key (like the code above) and the second square brackets access the position within the object stored in that key:

```
storms['minpres'][3] # Returns 989
```