Numpy

Please go through material for these topics. Complete the reading, exercises, and any videos linked. If the instructions ask to turn in any exercises, please do so through slack to your instructor.

Cognixia™

Outline

1. Array Creation
2. Array Shape
3. Sorting
4. Array Math
5. Array Manipulation
6. Open Book Quiz

Instructions

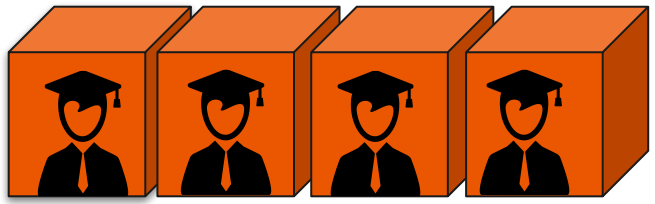
- ➔ Please review all sections on the topics listed in the outline
  - ◆ Read through material, watch accompanying videos, run coding examples, etc.
- ➔ **Examples are optional** and you are **NOT required to turn them in** unless otherwise stated
  - ◆ It is ***recommended you try them*** regardless to help understand the topics better
- ➔ The **open note exercises** at the end is **REQUIRED**
  - ◆ Have it ***turned in by the start of class the next day***
  - ◆ Ask your instructor if you have any questions regarding this
- ➔ Once you have turned in the quiz, feel free to leave for the day

Topic Video

- ➔ Please review the following introduction video on the NumPy exercises: https://youtu.be/NNI0MLxGb0M
- ➔ Similar instructions used in video are attached in the exercise
- ➔ The slides in the next section can be used as additional reference

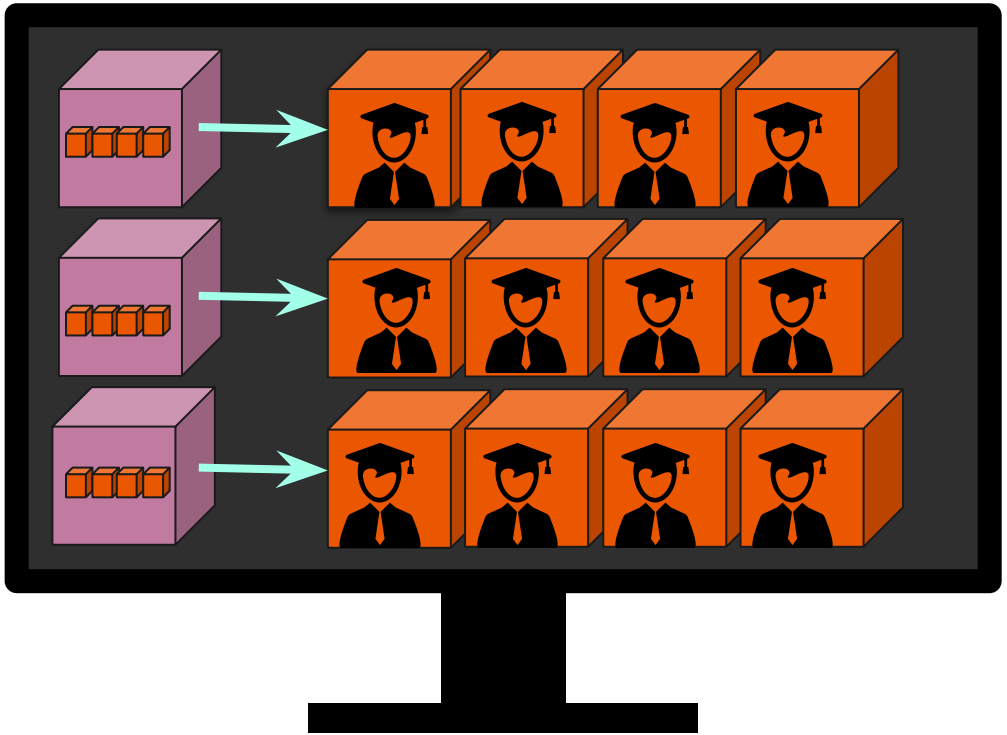
Array

- An **array** is a container object that can hold objects of a single type.
- They are ***faster*** and more ***compact*** than ***Python lists***, as they use less memory
- **Arrays** store data in a grid like structure and the data can be accessed various ways including
  - ***Indexing***
  - ***Slicing***
  - ***Iterating***



Multidimensional Array

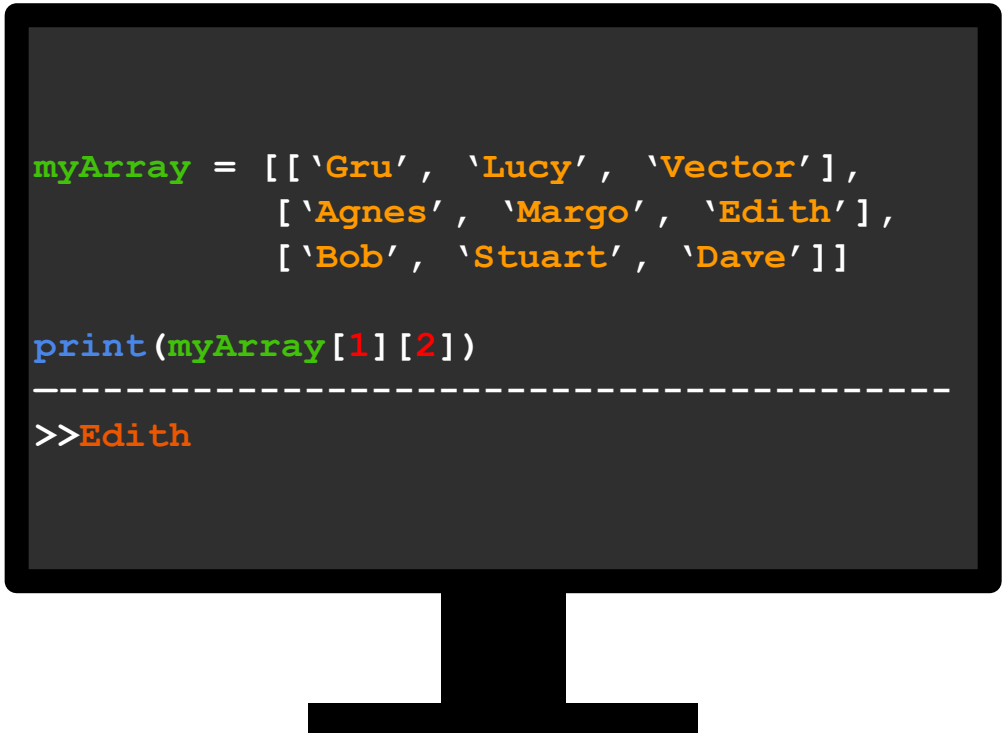
- ***Arrays*** can be populated with other ***arrays*** to create multidimensional grid like objects
- A **2-dimensional array** is an array where each element is another array holding objects



Accessing 2-D Array Elements

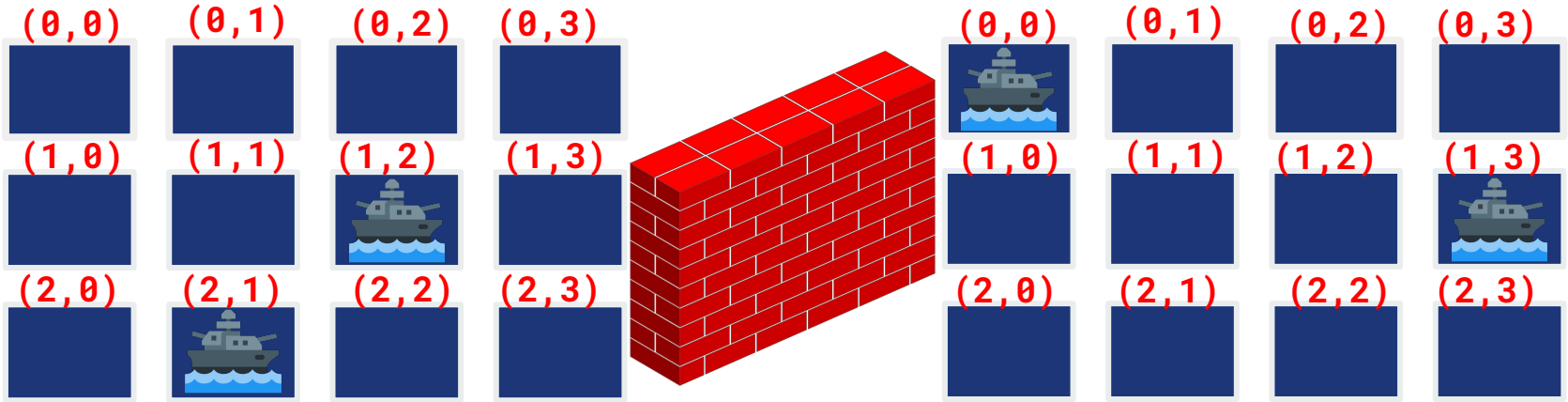
➔ Elements are directly accessed using their ***row*** and ***column*** positions

- *The **index** in the overall array is the **row** index*
- *The **index** in the inner array is the **column** index*



Multidimensional Array

➔ Just like in a game of battleship, to access elements in **multidimensional arrays**, you need to know their exact ***row*** and ***column*** position. Below is the a game of Battleship with their positions labeled



NumPy

- ➔ Python does not have built in methods to create an ***array***. Instead, the **NumPy** package is a great alternative.
- ➔ The NumPy library provides a multidimensional array object ***stored contiguous in memory*** and routines to manipulate those objects
- ➔ NumPy particularly excels in performing ***mathematical operations*** across the entire structure

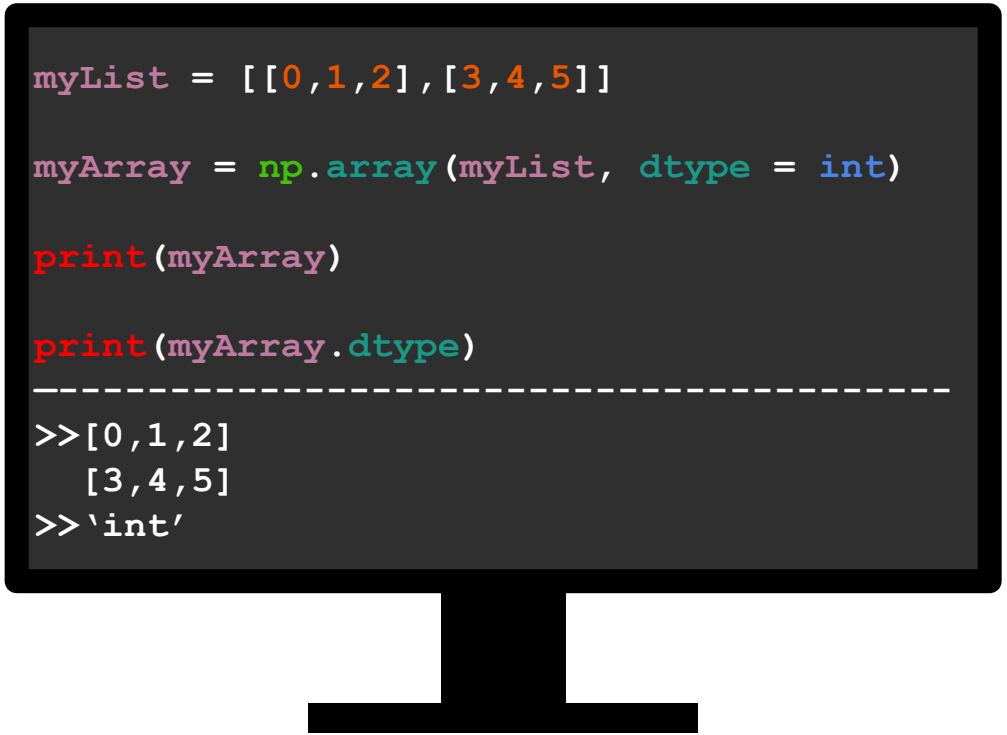
NumPy

- To use the NumPy package you must install it
  - ***pip install numpy***
- To use NumPy in your code, you'll need to import it as well
  - ***import numpy as np***
- Conventionally we shorten ***numpy*** to ***np*** for better readability, and it's recommended you do the same

Array Creation

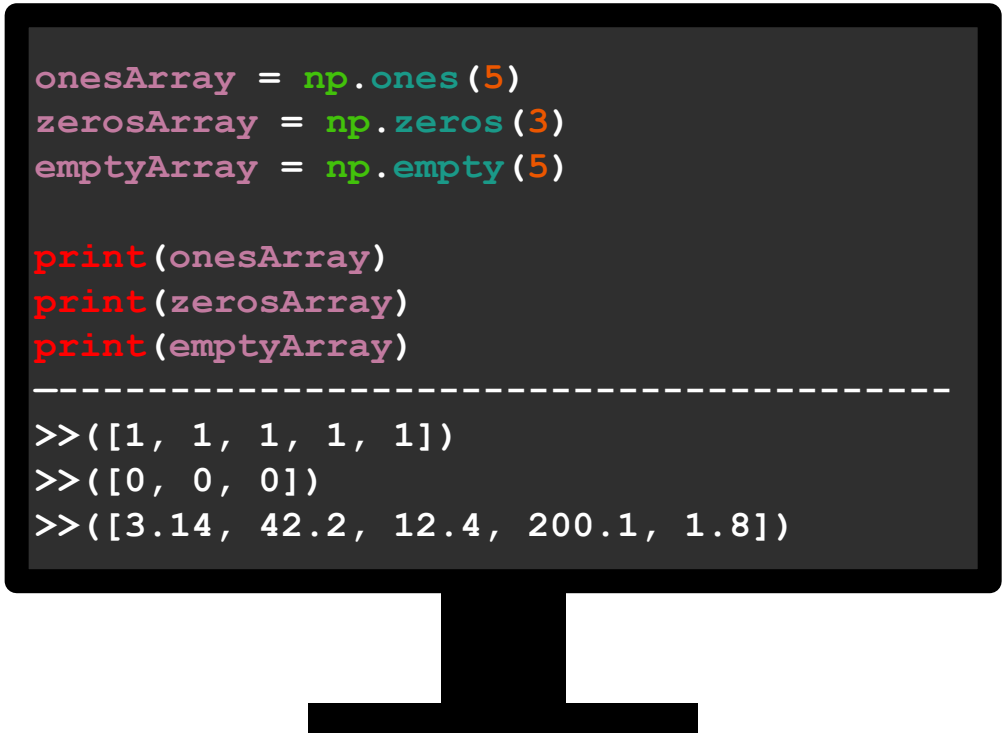
Arrays from lists

- We can create arrays from Python lists by passing the list as a parameter in
  - ***np.array([0,1,2])***
- You can also declare the ***dtype*** when initializing an array
  - *Python will assume the minimum required **type of data** to describe all elements in the array*
  - *See more about data types **here***



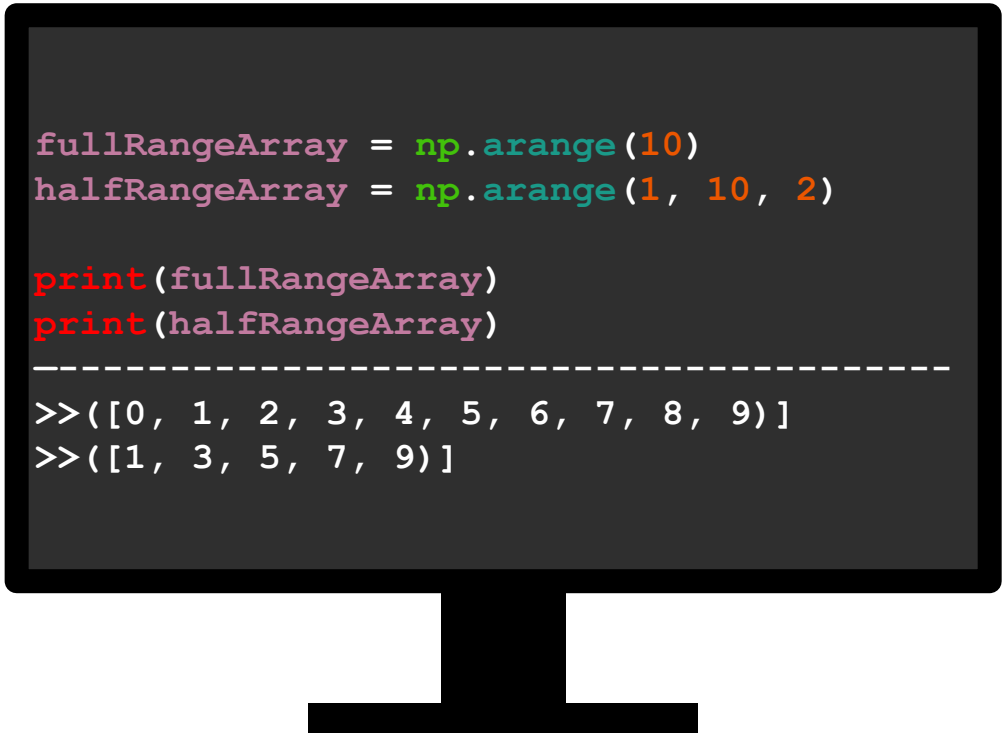
Prepopulated Arrays

- We can create arrays populated with ***n*** number of values
  - ***np.ones(n)*** *creates an array of* ***n*** ***1****s*
  - ***np.zeros(n)*** *creates an array of* ***n*** ***0****s*
  - ***np.empty(n)*** *creates an array of* ***n*** ***random values*** *that are currently stored in memory*

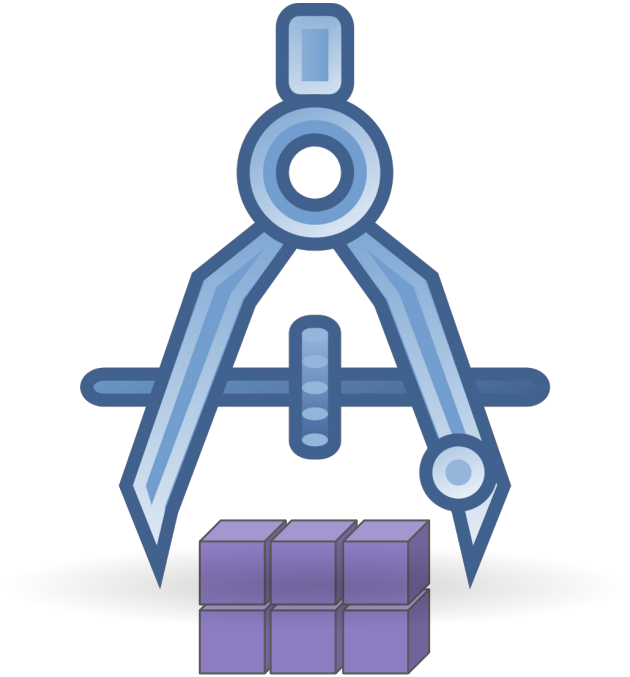


Prepopulated Arrays

- You can also create prepopulated integer arrays using ***ranges***. The function only requires you to declare an ***upper limit*** of your ***range***, but you can also include the ***beginning*** and the ***steps*** between each iteration
  - **np.arange(*start*=0, *stop*, *step*=1)**

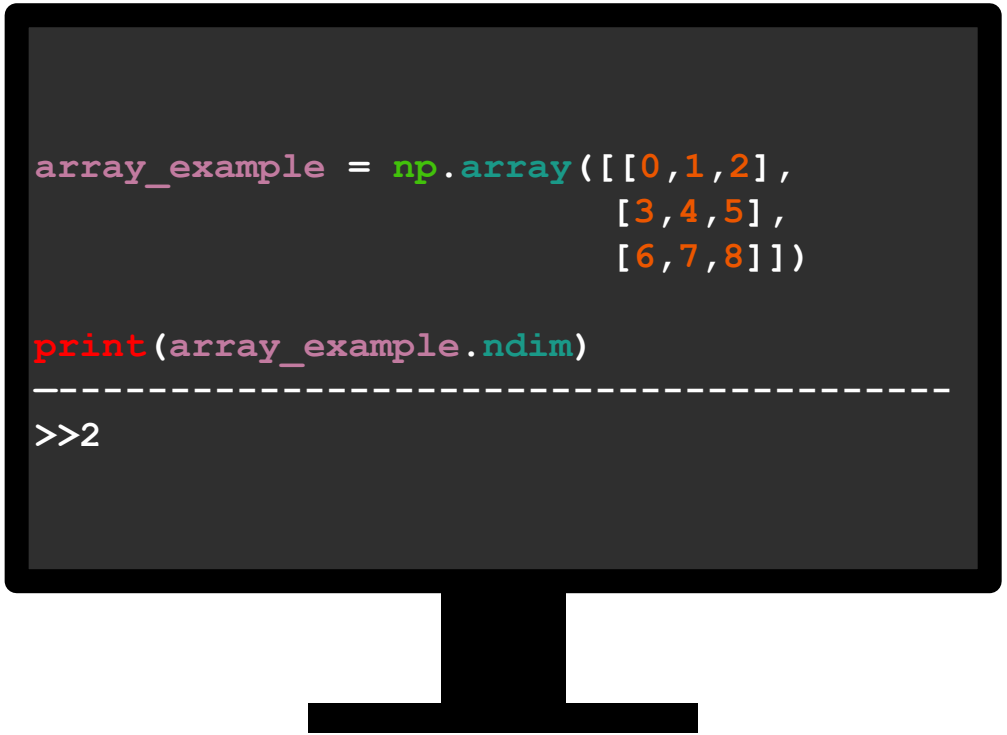


Array Shape



Dimensions of an Array

- The **dimensions** of an array is defined by how many sets of arrays are ***nested*** inside the overall array. ***It is the depth of the array***
- **NumPy** will also tell us the number of dimensions an array has by calling ***ndim*** of the object

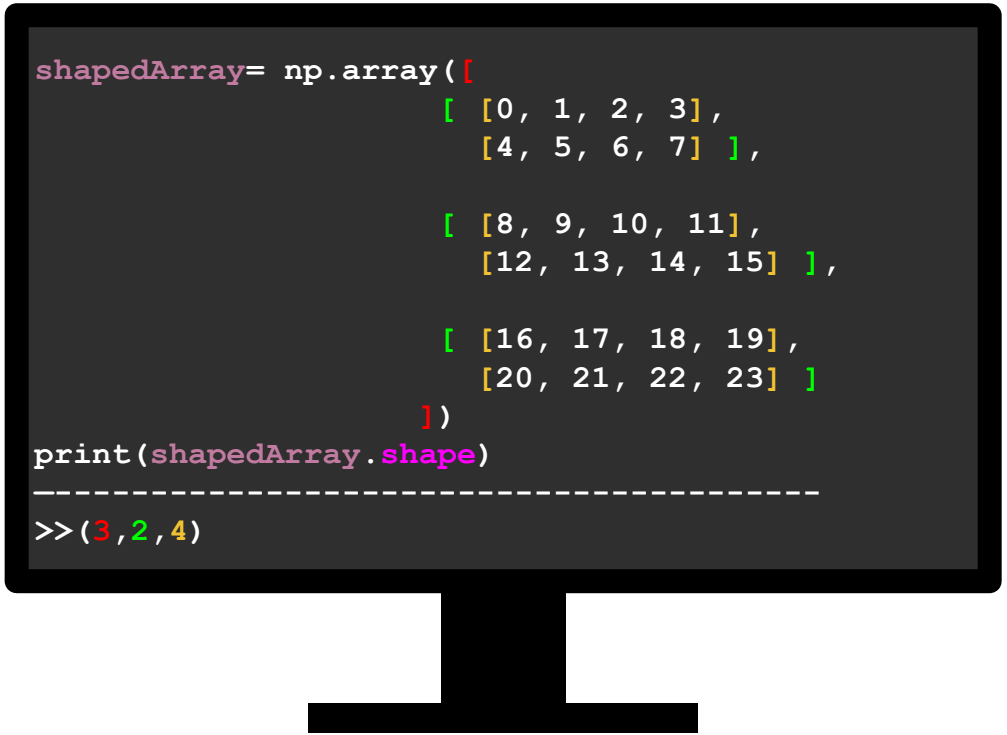


Size of an Array

- The ***size*** of an array is the total count of elements in the array
  - *array_example.size*

Shape of an Array

- The shape of an array is the number of elements in each dimensions
  - ***array_example.shape***
- The results will be listed as
  - *(**dim1**, **dim2**, **dim3**, ...)*
  - *Where each dimension is the size of the **nested list***



Reshaping of an Array

- To change the **shape** of the array we use the command
  - ***array_example.reshape(new_shape)***
  - *The shape we pass to the array must be a **valid shape** for its **size***

```
myArray = np.array(   [[0,1],
                       [2,3],
                       [4,5],
                       [6,7]])
print( myArray.shape )
print( myArray.reshape((2,4)) )
print( myArray.reshape((2,3)) )
-------------------------------------
>>(4,2)
>>([0,1,2,3],
   [4,5,6,7])
>> AttributeError: Incompatible shape for
in-place modification
```

Reshaping of an Array

- We can also quickly change a multidimensional array to a 1-D array using
  - ***array_example.flatten()***
- This will return a copy of the rows flattened into a single list

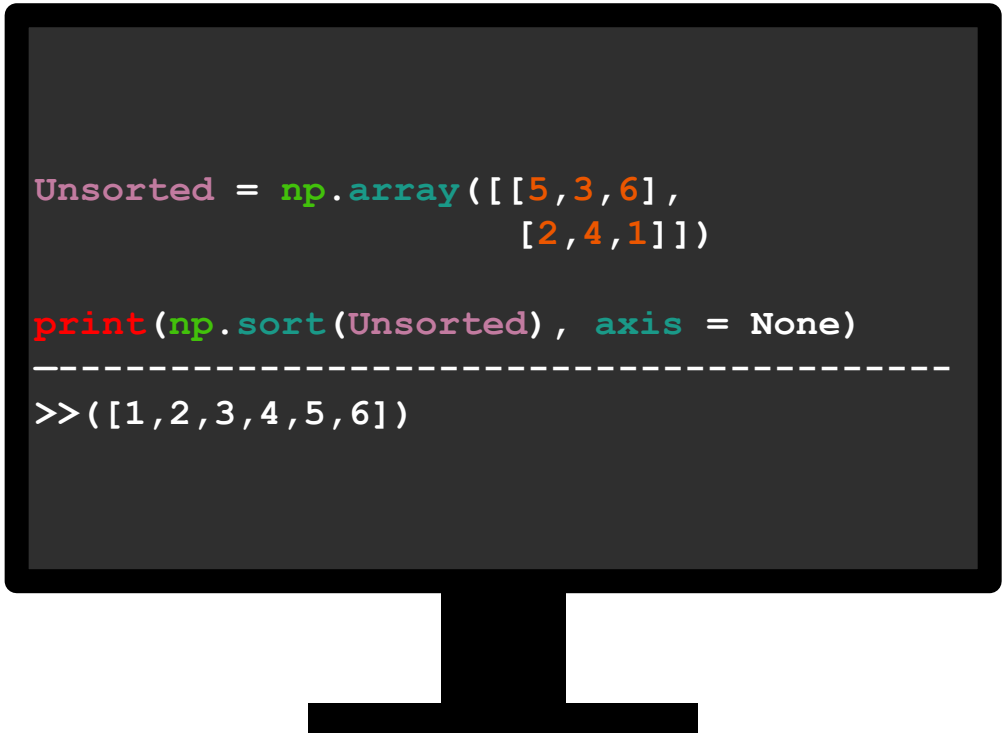
```
shapedArray = np.array([
                        [[0,1],
                         [2,3]]

                        [[3,4],
                         [5,6]]
                         ])
print(shapedArray.flatten())
-------------------------------------
>>([0,1,2,3,4,5,6])
```

Sorting

Simple Sorting

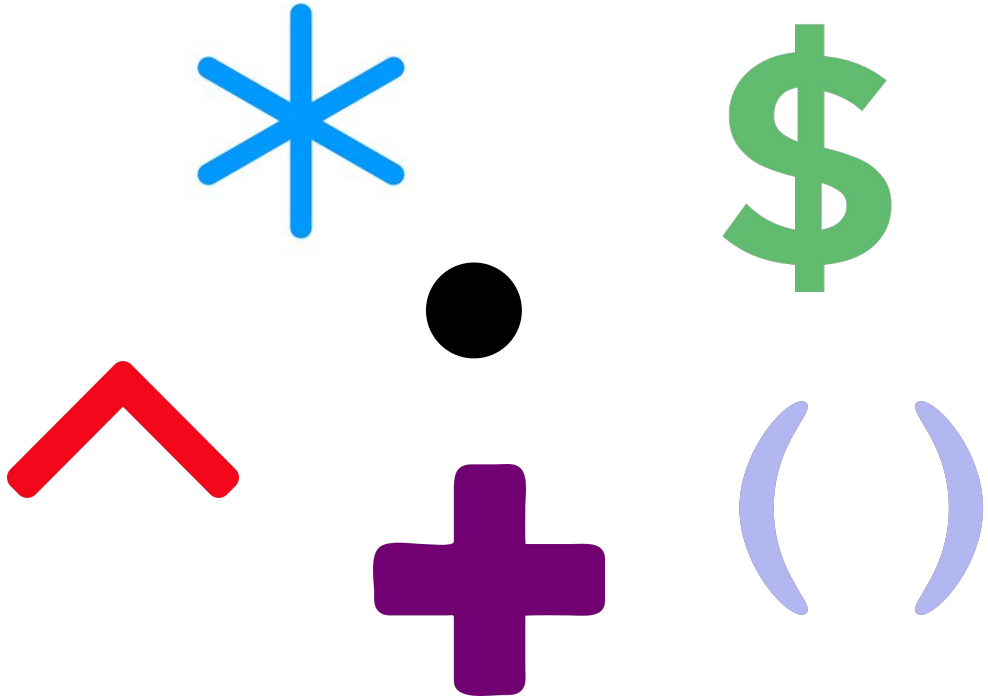
- You can quickly sort an array in ascending order using the command
  - ***np.sort(array_example, axis = -1)***
  - *The default **axis** is the **last axis**, but you can specify your axis as **None**, which will sort the **flattened array***
  - ***Rows** are **axis 0**. **Columns** are **axis 1***



Sorting by positional values

- ➔ You can find the positions of a sorted array using
  - ◆ **np.argsort(arr, axis)**
- ➔ And you can apply those positions to an array using
  - ◆ **np.take_along_axis(arr, indices, axis)**

Array Math



Adding Arrays of 1 shape

- You can add the values of 2 arrays with the same shape by using the **+** ***operator***.
- ***The size of both arrays must be the same***

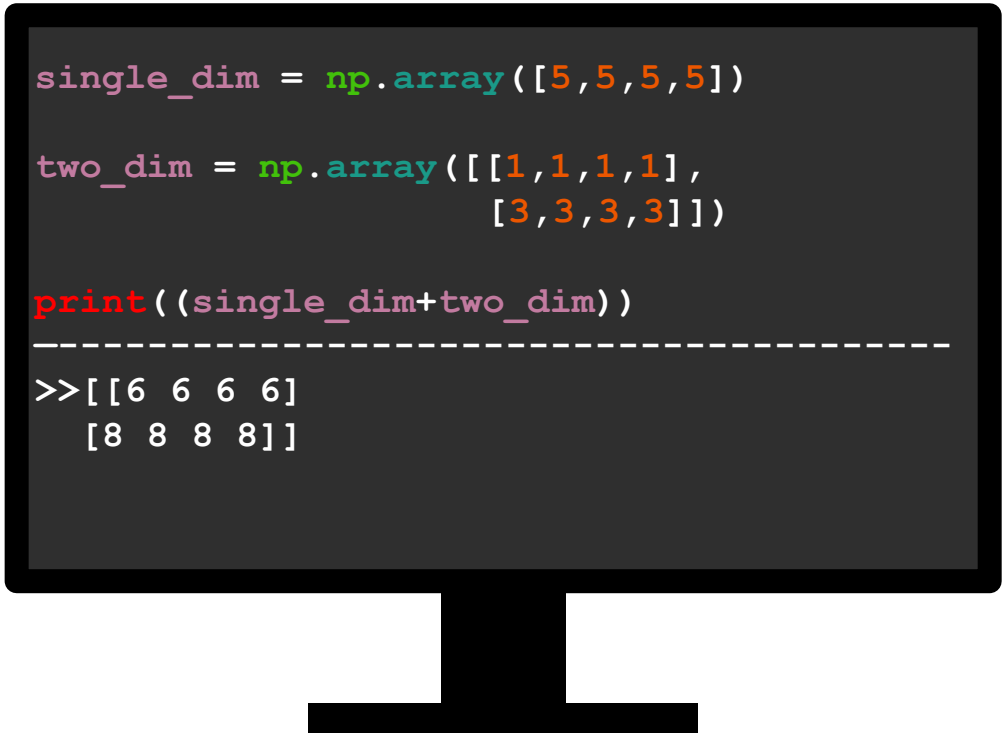
```
arr1 = np.arange(0,10)
arr2 = np.arange(0,100,10)

print("First array: \n", arr1)
print("Second array: \n", arr2)

print("Sum of both arrays: \n",(arr1+arr2))
----------------------------------------
>>First array:
[0 1 2 3 4 5 6 7 8 9]
>>Second array:
[ 0 10 20 30 40 50 60 70 80 90]
>>Sum of both arrays:
[ 0 11 22 33 44 55 66 77 88 99]
```

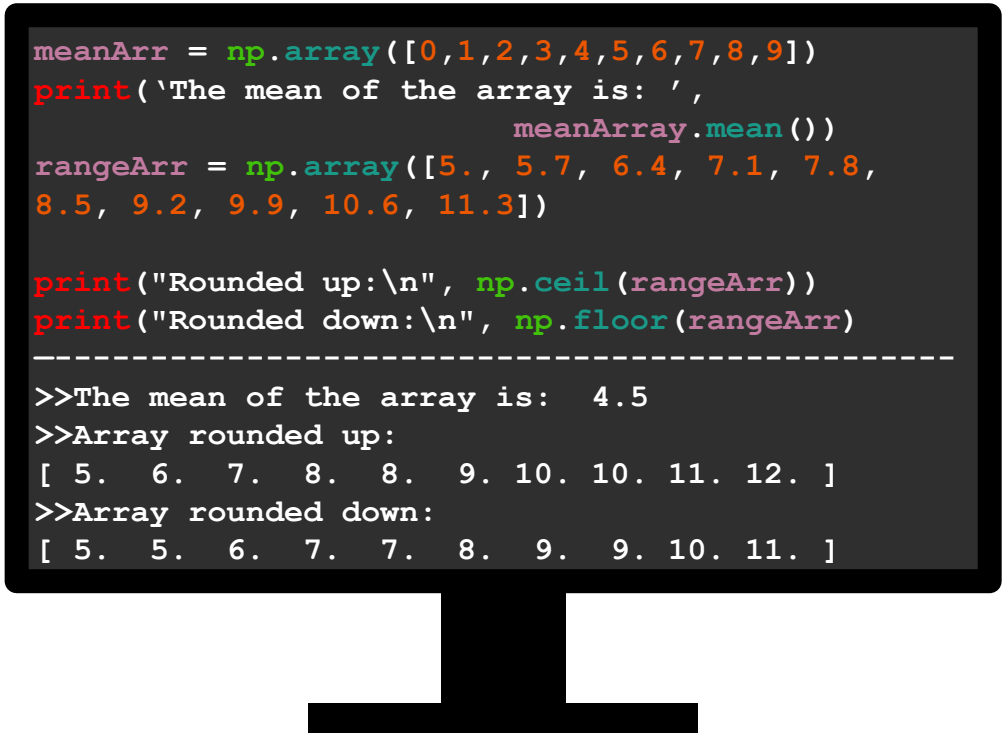
Adding 1-D and 2-D Arrays

- ➔ You can also add arrays with different dimensions using **+**
- ➔ You must have the same number of elements along at least ***one axis***

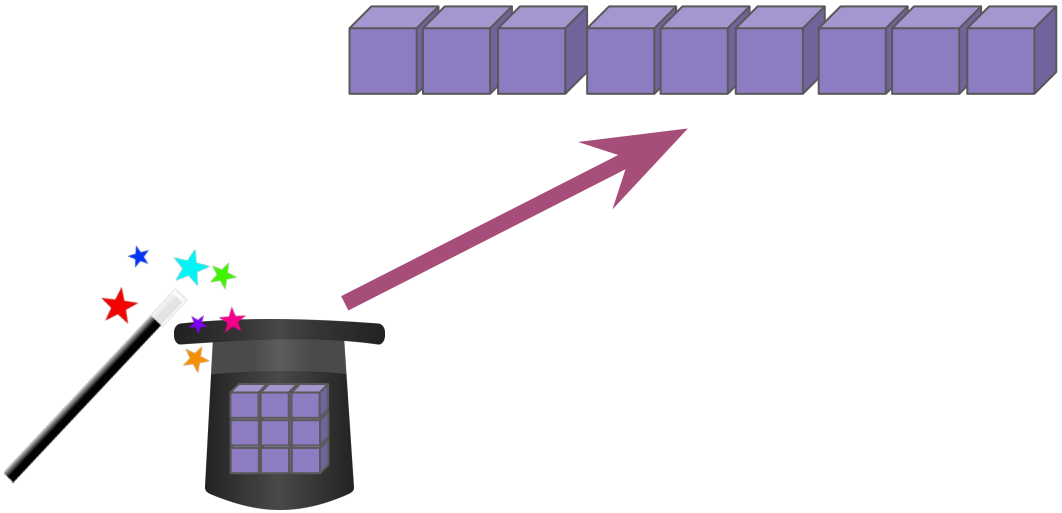


More Operations

- You are also able to perform the following operations on arrays
  - ***array_ex.mean()*** *will return the* ***mean*** *of the array*
  - ***array_ex.sum()*** *will return the* ***sum*** *of the array*
  - ***np.ceil(array_ex)*** *will return a copy of the array with the values* ***rounded up***
  - ***np.floor(array_ex)*** *will return a copy array of the* ***floored*** *values*

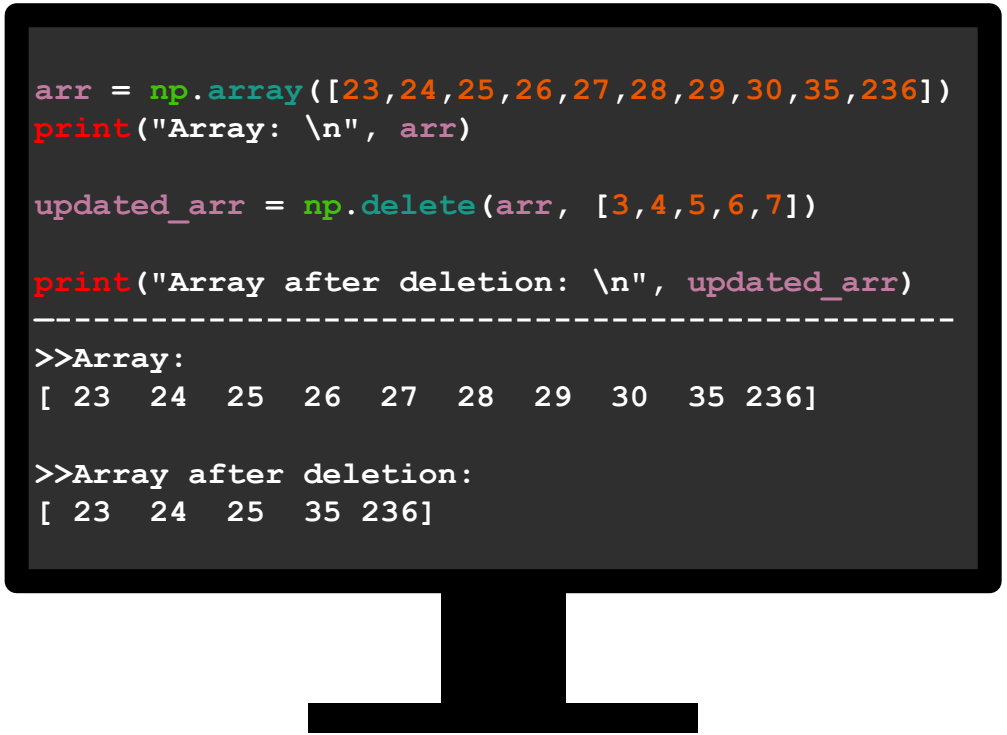


Array Manipulation



Removing Elements

- We can remove elements from the array using the command
  - ***np.delete(array_ex, indices, axis=None)***
- Python will assume we are flattening then removing elements



Combining Arrays

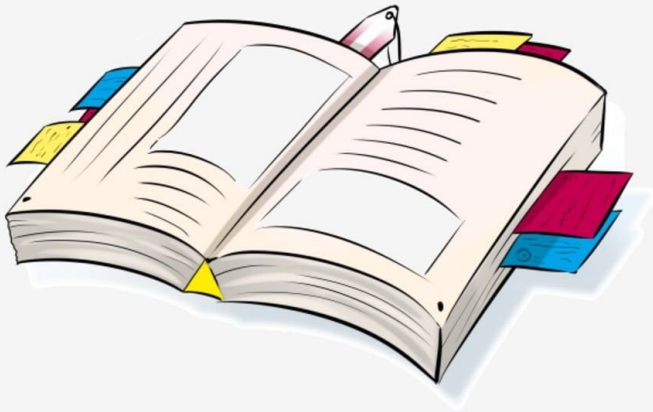
- We can also combine arrays using
  - **np.concatenate((array1, array2), axis=0)**
- This will return a copied array with all of ***array1 listed first***, followed by array2 values

```
arr1 =  np.array([[23,34,54],
                  [12,42,122]])
arr2 =  np.array([[1,2,3],
                  [4,5,6]])
print("Concatenate over first axis: \n",
               np.concatenate((arr1, arr2), axis=0)) )
print("Concatenate over last axis: \n",
               np.concatenate((arr1, arr2), axis=1)) )
```

```
Concatenate over first axis:
 [[ 23  34  54]
 [ 12  42 122]
 [  1   2   3]
 [  4   5   6]]
Concatenate over last axis:
 [[ 23  34  54   1   2   3]
 [ 12  42 122   4   5   6]]
```

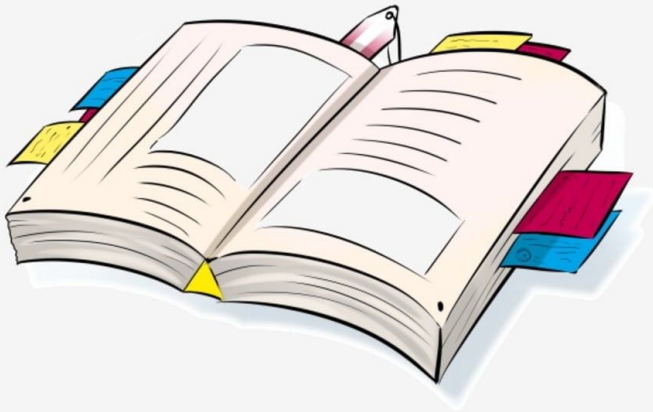
Open Book Exercises on NumPy

- NumPyExercise.ipynb
- This is an **open note** set of exercises. Fill in all code blocks in 'The Challenge' section
- Have it completed by the **start of class tomorrow at 10AM EST**
- If there are ***any questions, ask your instructor during this time or during office hours***, as they may not able available after hours



Open Book Quiz on NumPy

- ➔ https://forms.gle/EP9WbbkTx7koooNh9
- ➔ This is an **open note**, multiple choice quiz
- ➔ Have it completed by the **start of class tomorrow at 10AM EST**
- ➔ If there are ***any questions, ask your instructor during this time or during office hours***, as they may not able available after hours



Cognixia™