# Week Two

Arrays, packages, and writing programs

## Review

- UNIX is the OS/environment in which we work
- We store files in directories, and we can use commands in the terminal to navigate around, make and delete directories, look at their contents, copy/move/delete files, etc.
- We can use the ipython interpreter to do basic math, and to set variables equal to ints, floats, lists, etc.

## Overview for Today

- Numpy, Matplotlib, astropy, and why we need them
- Focus on numpy arrays, how they differ from lists, and why they are useful
- Organizing commands into a coherent program which can be saved and run in python.

# Python Libraries

- Python can do some basic math: +,-,x, / , x^y
- To do anything more (sin(x), sqrt(x), plot y vs. x, etc.) we need to import some libraries.
- ***Libraries*** are collections of functions which increase the usability of python dramatically.
- There are libraries for almost every purpose- but a few are essential to almost any code

## Numpy

- "Numerical Python"— A library of a huge number of functions related to what you might need to do mathematically in python.
- Also contains the datatype "array," which is essential to scientific work
- Part of the scipy (scientific python) family; sometimes you will be importing specific functions not in numpy from scipy.

# Matplotlib

- A Matlab style plotting library
- Anytime you are going to need to plot your data, you will need this library imported into your code
- Also has a huge, daunting set of features. It takes time to learn them, but you gain intuition as you need specific features for your plot and you look up how to implement them in the documentation

# Astropy

- A bunch of astronomers got fed up with not having a good library of astronomy geared functions, so they made one
- We will be using it primarily in the second half of the course when we start looking at images from telescopes.

# Importing Libraries

- These libraries have to be ***imported*** into our code, or into ipython, if we want to access the functions in them. To import we simply type
    - >>>[IN]: import numpy
- To use a function within the library, we use dot notation and call the library first, then the name of the function (since sometimes you will have multiple libraries imported containing functions of the same name).
- For example, to make a sine out of a predefined variable “x” we could use numpy.sin(x).

## Importing Libraries

- For matplotlib we say: import matplotlib.pyplot
    - This is because matplotlib is huge, and we only need the functions in the pyplot sub-library.
- We can name the libraries we import whatever we want within our code (normally we import numpy as np, and matplotlib.pyplot as plt)

## Return to Data-types

- We have discussed that there are datatypes in python- that an integer has different rules as a float, for example
- Today we will be focusing on strings and arrays, and how to index/slice through them (hint: you actually did this in your last tutorial).

# Strings

- Strings are defined by putting quotes (single or double) around them when you name a variable, e.g.,
    - >>>[IN]: cat_name = ‘bozo’
- Words like this *have* to be stored as strings, since they clearly can’t be integers or floats, etc.
- You can have any type of character within a string, including numbers

## String Concatenation

- We can combine strings using the mathematical operator “+”
- ‘String1’ + ‘string2’ becomes ‘String1string2’
- ‘One’ + ‘ ‘ + ‘Two’ becomes ‘One Two’ (the space could also be tacked on to the end of the one or the beginning of the two)
- This syntax becomes helpful for file importing later on

## Arrays

- Arrays are part of numpy. Unlike lists, there can only be one data-type within an array (if you enter an array with floats and ints, it converts them all to floats).

- Arrays are extremely useful. For example, math can be performed on them— if you divide an array by a number, every single element in the array is divided by that number, etc.

- To define a numpy array we would use np.array([1,2,3,…])

- You can also take any list that has only numbers in it, and turn it into an array by typing np.array(list_name)

## Arrays

- We can initialize arrays with 0's by using np.zeros(num) where num is how many 0's you want in the array
- We can create an ascending list similar to range( ) by using np.arange(start, stop) (0 is the default start)
- We can also define multidimensional arrays and matrices (more on this a bit later).

# Indexing

- The index-able datatypes (strings, lists, and arrays) are made up of units called elements- these are straightforward. Each character in a string, or each comma separated entry in a list or array, constitutes one element. These elements have an index number as follows
  - list1 = [ 1 , 2 , ‘cat’ , 7. , ‘p’ ]
  - index: 0 1 2 3 4
- I.e., 1 is the 0th element of the list

## Indexing

- As you saw in the tutorial, we can “pull” the i’th  element in a string, list, or array by typing something like string1[i]
- In the example before, typing list1[0] would return 1
- Typing list1[0:2] would return 1, 2, ‘cat’  (the syntax is [start:stop])
- We can use negative indexes to start from the end and work left—typing list1[-1] would return ‘p’.

# Writing a Program

- So far we have been putting commands one by one into the python interpreter, but this is won’t work for when you have tasks with many lines of code.
- We can write a text file containing all the commands we would have used in python, and have python run the list of commands in order, as if we had typed them in one by one to the interpreter.
- You can use vim, emacs, sublime, notepad++, enthought, etc. to write code—just DON’T use something like microsoft word, because these programs add extra junk to text you write for formatting purposes.
- Even the notepad app on a pc can be used- but the fancy programs like sublime and enthought will color code your text for special python words to make it easier to read and edit.

## Writing a Program

- Commenting: ALWAYS comment your code! It seems dumb at first, and take up time you'd rather be spending moving forward in your code, but trust us, commenting will save time in the long run, because you won't remember the subtle structure of your code a year later when you need to use it again. And people you share it with will have no clue how to use it.

- Once you have written up a file with the commands (each separate command on a new line), save it with the extension ".py"

- To run it, open ipython (in the same directory as the file), and type run file_name.py where file_name is whatever you saved it as.

## Tarring/zipping

- For the tutorials from now on, you may need to turn in multiple files. In order to do so, you need to compress them into a single file to be uploaded.

- In terminal on a Mac, you can type tar -cvf file_one file_two (etc)

- Alternatively you can find the files in your finder and right click->compress to make a zip.

- On windows you can use the windows zipping utility or a program like winrar. (Right click —> send to… —>compressed folder

# Review

- For any code we write, we basically always need to import numpy and matplotlib (and possibly astropy) at the top of our code, usually in shorthand. We call functions from these libraries using the dot notation

- Arrays and lists can be indexed by element number, starting from zero (or in reverse, starting from -1)

- We can arrange the inputs we would originally have typed into python as a program, which can be run all at once, and easily edited, saved, and shared.