You might have wondered what those

```
import numpy as np
import matplotlib.pyplot as plt
```

mean at the top of each program we write. These `import` statements tell python to read a module, and give a name to its contents. If you omitted the `as np`, then the name of numpy's contents would just be `numpy`.

A python **module** is just some file that has code in it. Most modules just define a bunch of functions, and maybe a few constants, but they could really do anything they want. These modules allow you to use functions that other people have written which is great, because you wouldn't want to have to write everything yourself!

A python **package** is a collection of python modules that someone has published. `numpy` and `matplotlib` are packages that we use. Packages and modules are how python implements what is called a **library**. You don't need to know this name, but as you learn another programming language, you're likely to find things described as "libraries." Even in Python, the modules that are shipped with Python itself are called The Python Standard Library.

In this class, we will restrict ourselves to using `numpy` and `matplotlib`, just so you won't have to learn to use a whole bunch of packages. We also will not be creating our own modules, not because it's not useful, but because it won't be particularly useful for the tasks you'll be working on, and always makes your code a bit harder to read. If you ever find yourself copying huge chunks of code back and forth between two Python files you're working on (outside of this class), you might consider putting that code into a separate module.

You might wonder has how Python knows where to find a module. The short answer is that there are various places where Python will look, and it's not a well-designed system. In particular, you can really screw things up if you create a python file that has the same name as any other module, e.g. having a file named `string.py` is a very bad idea. The problem is that module names are not properly scoped, like local variables are in functions. As a result, if `numpy` tries to import a module named `string`, it might end up loading your file named `string.py` instead of the string module of the standard library. If this happens, the results can be *very*, **very**, *very* confusing.