Python Classes - Week 7

1 Modules and Packages

We don't want to reinvent the wheel - If someone has implemented something for us, then we can just use that.

- Modules are collections of functions which somebody else has built
- Packages are nicely wrapped up collections of modules

1.1 Import

The way we load a module into our code is with the **import** statement, e.g.:

```
In [5]: # We snuck this in a while ago
        # "time" is a package which comes with python

        import time   # We import the time module
        time.sleep(3) # sleep function is part of time module
```

1.2 From

Sometimes we don't want to import everything from a module - we can pick using **from**

- E.g. if we have a function sleep(), we probably don't want to import the sleep() function from time

Additionally, if we use from, we don't need to explicitly call the function via the module

```
In [10]: # All I want to do is sleep
         from time import sleep
         sleep(3)  # Don't need to call via time module
```

1.3 As

If we want to refer to the imported functions using a different name, then we can import them **as** something else

```
In [12]: # Zen
         from time import sleep as meditate
         meditate(3)
```

This can be useful if you have a function with the same name, or if the name of the module/function is annoyingly long

```
In [ ]: import time as t # Can change module name
        t.sleep(3)
```

2 PIP

You'll often find that the package you want to use is not installed alongside python by default. But if it's popular, then chances are you'll be able to add it using python's package manager, **pip**

pip can be accessed from the **terminal** (linux/mac) or **command prompt** (windows) if python is properly installed.

Installing the numpy package on Windows:

```
C:\WINDOWS\System32>pip install numpy
```

and on Linux:

```
alex@Alex-Blade ~ pip install numpy
```

3 Coding in the real world

Most new code is written to solve new problems. It is rare that these problems don't build on previous work

In reality, much of your time will probably be spent figuring out how to repurpose code written by strangers, or colleagues, in order to solve the task in front of you.

If you're lucky, there will be thorough **documentation** for the package / modules in question (and *good comments!*)

4 Turtle

In the exercises you'll be playing around with numpy and turtle. You will make use of the documentation! :O

Turtle is a very basic animation module that comes with python:

- You tell a turtle to move, and it draws a line behind it
- The turtle can also fill in regions using **begin_fill()** and **end_fill()** [see example]

```
In [ ]: from turtle import * # This way we can use all functions without turtle.Blabla()

        color('blue', 'green') # Line color, fill color
        begin_fill()
        # Move the turtle
        forward(100) # Move forward 100 coordinate points
        left(90) # Rotate left by 90 degrees
        forward(100)
        left(90)
        forward(100)
        left(90)
        forward(100)
        end_fill()
```

Drawing made by the simple turtle script

5 Additional Stuff

5.1 Using the terminal

This makes you look like a pro-hacker

Most professional programming is done on linux or mac, both of which are based on UNIX. Windows users can install the unix subsystem with fairly little effort (I recommend looking at https://www.windowscentral.com/how-install-bash-shell-command-line-windows-10).

"pip install" is an example of a terminal command, but there are many more. Amongst these, the most commonly used are those which allow you to move around inside your PCs directories:

- **ls** lists the files and folders in your current directory, and **ls -a** includes additional information
  - Here -a is an example of an additional parameter. Many commands have these, and you can typically find out about the possible parameters/usage of a command by using the --help tag. For example, "ls --help"
- **cd** short for change directory allows you to move between folders
  - e.g. "cd my_folder* moves you into the folder my_folder, if it exists in the current directory
- **pwd** prints the working directory, so you know where you are in the filesystem
- **mkdir** creates a directory. E.g. "mkdir fish" creates a folder named "fish" in your current directory
- **less** allows you to view the contents of a file without opening it. You can exit this using the "q" key.
  - e.g. if there's a "test.txt" file, you can use "less.txt" to directly view its contents in the terminal window
- **rm** removes stuff - **be careful with this**. The **-r** parameter removes recursively, allowing you to delete folders and their contents
  - e.g. "rm -r my_folder" deletes the folder "my_folder"
  - "sudo rm -rf /*" deletes everything on your pc! :D (the -f parameter forces deletion of protected files)