Python from scratch

- You can use it as a calculator: `+, -, *, **, /`.
- Printing
- Comments
- Variables:
    - Strings
    - Ints
    - Floats
- Lists, and accessing items:

```
numbers = [1,4,5,2,1,1,1]
numbers[0] # 1
```

- If statements:

```
if x == 0:
        x = 1
else:
        x = 0
```

- For loops (and how it works with any iterator):

```
for i in range(10):
        print("hi")

for number in numbers:
        print(number)
```

- Functions (a black box with an input and output):

```
def average(x,y):
        return (x+y)/2
```

- Indentation is required as part of the syntax for any of these structures.
- Python infers the type that any variable takes. This can be forcibly specified, if required.
- Ask us for help if you need some, but also remember Google is your friend!
- Copying and pasting code examples on Google, and then adjusting *is* the correct approach.
- Error messages (e.g. syntax, zero division etc.) are guaranteed, and wading through them is the most time consuming, but most *important* aspect of programming.

Numpy

Using this package can lead to far more efficient computations. We get the handling of vectors that you would expect, unlike tuples/ lists.

- Addition v.s. concatenation:

```
(1,2) + (3,4) # (1,2,3,4)
np.array((1,2)) + np.array((3,4)) # (4,6)
```

- Scalar multiplication v.s. duplication:

```
(1,2) * 3 # (1,2,1,2,1,2)
np.array((1,2)) * 3 # (3,6)
```

- Computing length:

```
np.linalg.norm((3,4)) # 5
```

Classes

To learn these, it makes the most sense to understand the abstract theory, and then learn about how python implements objects and classes.

We can create an instance of the class, specifying the parameters found in `__init__`.

```
some_vortex = Vortex((0,0), 1)
```

We can access properties, or call functions specific to that class, called methods, like so:

```
some_vortex.color
some_vortex.getField((0,0))
...
```

Any instance, or object acts as a member of the class, with the same property names and functions, but with different values.

Any method must have `self` as their first argument, but when a method is called, this argument is omitted because python passes in the object `self` automatically.

```
class GreetingsBot:
        def __init__(self)

        def greet(self, name):
                print("Nice to meet you, " + name)
                # Insert code for world destruction here.

bot = GreetingsBot()
bot.greet("Dr Cooper") # Nice to meet you, Dr Cooper
```

(We don't write the following:)

```
bot.greet(bot, "Mr Hudson")
```

(Or this:)

```
bot.greet(self, "Mr Jackson")
```