Notes on Floats

In most arduinos, a float is 4 bytes.

If we print a float over a serial port:

```
float pi = 3.14
Serial.print(pi);
```

We will see the ASCII representation of 3.14.

The micro controller is sending the ASCII values to the UART. So for 3.14, it sends 4 bytes. The byte for '3', '.', '1', '4'.

Consider a float like 314.00. If we Serial.print() this float, we will get 6 bytes! The ASCII for '3', '1', '4', '.', '0', '0'.

But if we want to know exactly how many bytes to expect, this is bad. We know every float can be ~~accessed~~ represented by 4 bytes regardless of how many bytes the ASCII representation takes.

To access these bytes, we need to go to the memory address of our float and treat it like it is the start of an array.

```
byte * b = (byte *)&pi;
```

Create a pointer called 'b'

That starts at the memory address of Pi

* - Create a pointer

& - memory address

With this new pointer, we can see the bytes that make up pi!

b[0] → 0xC3 Least Significant Byte
b[1] → 0xF5
b[2] → 0x48
b[3] → 0x40 Most Significant Byte

For arduino, we are Little endian. This means the smaller byte is stored in the smaller address.

Arduino is also using the IEEE 754 Single-precision standard for floating point numbers.

In our example:

~~0x~~ MSB LSB
0x40 0x48 0xF5 0xC3

0 | 1000000 0 | 1001000 | 1111 0101 | 1100 0011

sign exponent fraction

$(-1)^{sign} \times 2^{exponent-128} \times 1.(fraction)$

$(-1)^{0} \times 2^{128-127} \times (1. + 2^{-1} + 2^{-4} + 2^{-8} + 2^{-9} + 2^{-10} + 2^{-11} + 2^{-13} + 2^{-15} + 2^{-16} + 2^{-19} + 2^{-22} + 2^{-23})$

$2 \times 1.5700000525 = 3.1400000105$

good up to 6 decimal places!