Serial Communication

＞ A common method of communication between microcontrollers and PCs.

FOR TWO DEVICES TO COMMUNICATE SERIALLY

＞ Digital Pulses are sent back and forth between devices.
＞ They must agree on a rate of communication and synch to that rate. Why? Each device has a different clock. Duh.

>>> 'Sender' sends pulses that represent data, at the agreed on rate.
>>> 'Receiver' listens for pulses at the same rate.
Three connections needed for Serial Comm.

› A common ground -- so both devices have a common reference point to measure voltage by...
› A wire to send data.
› A wire to receive data.

A bit about data rates (get it?)

› Example: Both devices agree to transmit data at a speed of 9600 bits per second (9600 “baud”)
› Receiver will continually read the voltage sent by the Sender.
› Every 1/9600th of a second, the Receiver interprets this voltage as a new BIT of data.
› Arduino: +5V is a bit of 1, 0V is a bit of 0 (HIGH or LOW).
› This way, in one second, 1200 BYTES of data can be sent/second.

Bits’n’Bytes

› a BIT is 0 or 1.
› There are 8 BITS in a BYTE.
› Each alphanumerical symbol takes up a byte.
› The ASCII code assigns a number from 0-255 to each byte.
Example:

- The byte 90 ("Z") is 01011010 in binary. It can be represented by a series of HIGH and LOW pulses, like this, sent at 9600 baud.

- There are different protocols for data transmission. Arduino uses something called "RS-232" to communicate with your PC. Arduino uses an onboard USB-to-Serial converter. Different micros use different protocols.
Notes:

- Your computer has multiple serial ports, but only one program can control a serial port at a time! This can sometimes lead to confusion.

- If you’re only sending one number at a time, and that # is <255, then you can send it in one byte using Serial.write() !!

- If you’re sending a larger number, or multiple values at a time, the receiving end needs to know when the data begins and ends so that ABC,ABC,ABC isn’t read as BCA,BCA,BCA ...

Once your Arduino is programmed and hooked up to your PC, the first thing you do is open a Serial Monitor to see what the values being sent are.
1. Hook up a potentiometer to your Arduino

2. Divide the output by 4 (to get it between 0-255)

3. Use Serial.write() to print the values
Try looking at it in a few different formats:

```cpp
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int val = analogRead(A0) / 4;
  Serial.write(val);
  Serial.print(" ");
  Serial.println(val);
  delay(10);
}
```

Use Arduino’s Serial Monitor, or a program such as ZTerm, CoolTerm, or even the Terminal to view the results!

Using the ASCII encoded DEC format, a val of 64 would be 4 bytes!
5. Now, read the outgoing byte in Processing.
Sending Multiple Sensors...

To send multiple sensors, the Receiving end must do a bit more work & unpack the bytes you’re sending in the correct order & format.

THE PUNCTUATION METHOD

Let’s say you’re sending three readings to Processing
Punctuate them, coming out of Arduino, so that it looks like this:

```
void loop() {
    // read the sensor:
    sensorValue = analogRead(analogOne);
    // print the results:
    Serial.print(sensorValue, DEC);
    Serial.print(“,，“);

    // read the sensor:
    sensorValue = analogRead(analogTwo);
    // print the results:
    Serial.print(sensorValue, DEC);
    Serial.print(“,，“);

    // read the sensor:
    sensorValue = digitalRead(digitalOne);
    // print the last sensor value with a println() so that
    // each set of four readings prints on a line by itself:
    Serial.println(sensorValue, DEC);
}
```

Now the receiving end can take a \n character to mean: “Okay, that was just the Third sensor. Now, onto the new reading of the First Sensor.”

THE HANDSHAKE METHOD -- Read more about this on the class site!