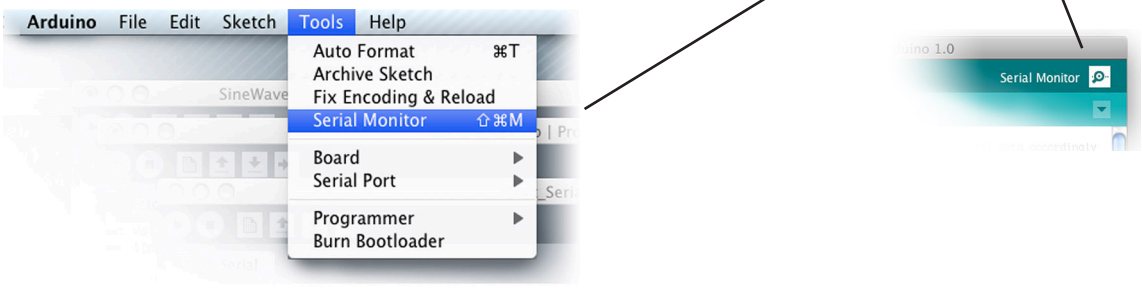
Reading Serial data from the Arduino (Atmega 168/328)

The people who designed the Arduino had ease-of-use in mind. Consider this example:

```
//-----------  Simplest serial send from Arduino  ----------------
void setup() {                    // Only do once at startup
  Serial.begin(9600);             // start serial port at 9600 bps:
}
void loop()  {                    // repeat forever
   Serial.println("A");           // Send message to serial (USB)
   delay(500);                    // delay a half second
   Serial.println("1");           // Send message to serial (USB)
   delay(500);                    // delay a half second
}
```

In `setup()` we establish serial communications at 9600 bits-per-second, and then in `loop()`, we repeatedly send an "A" and a "1" with a half second delay between each character sent to the serial port.
There is even a convenient tool to view the traffic of the serial port built into the Arduino interface: the *Serial Monitor*. It can be called up by either the Tools dropdown menu or an icon in the upper right corner of the Arduino interface window.



Launching the Serial Monitor while running the above program on the Arduino, we can confirm the output of an "A" character followed a half second later on a new line by a "1" character. Unfortunately, (or fortunately maybe) there's other stuff going on here with serial data.
Specifically, the `Serial.println()` command doesn't send the letters or numbers, it sends the *ASCII codes* for those alphanumeric symbols. Not only that, it follows each character with a "carriage return" character and a "newline" character (ASCII codes 13 and 10, respectively.)
If you watch the Arduino's serial data with an external program (like Max or pd), instead of an "A" followed by a "1", what you'll in fact see is the sequence: " 65, 13, 10, 49, 13, 10..."
It turns out that numbers and letters are handled differently in computer memory than the human-readable *symbols* for those things. ASCII codes are references to the human-readable symbols. The built-in Serial Monitor in the Arduino interface assumes that when it receives a byte of data, that it is ASCII code so instead of printing the number 65, it displays the letter "A". And similarly, instead of displaying the numbers 13 and 10, it terminates the character and goes to a *new line*.
When passing sensor data from the Arduino to the serial port, there is another command you can use instead of `Serial.println()`. It is the `Serial.write()` command. It sends raw binary data to the serial port which is parsed as the actual numbers that are being sent from the sensors -instead of the ASCII values for those numbers.
(There is a handy command in Processing for turning ASCII values for numbers back into the numbers themselves: the `int()` function. (*example usage:* `sensorData = int(serialString);`)

To recap:

-- When using Arduino serial data with Processing: `Serial.println()`
(Remember to use the `int()` function to translate back to the actual numbers being sent.)

-- When used with Max, puredata, Python, etc., -best to use `Serial.write()`