Chapter 7

ARDUINO PROGRAMMING

Assumptions and Goals

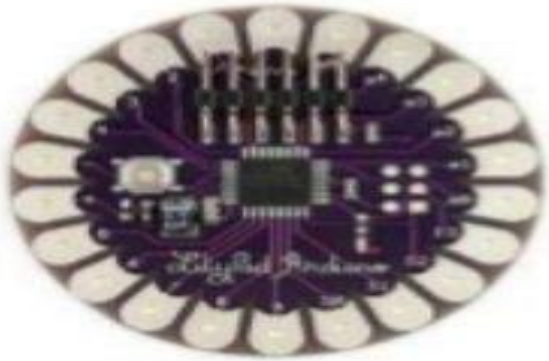
- Assumptions
  - You have taken at least one programming course prior
- Goal: Be able to create applications on arduino using the following methods
  - Adding Shields (Accessories; Attachments)
  - Adding Libraries
  - Using control flows (If-Else, Do While, While loop)
  - Using functions (Methods: e.g. getVariable, set Variable, change Variables)
  - Reading connection diagrams
  - Debugging applications

Arduino Models

- Arduino's are microcontrollers: mini-computers with their own memory, disk, and processor

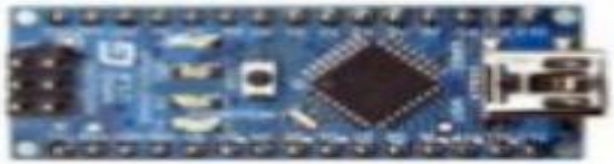
UNO

Mega



LilyPad

Arduino BT

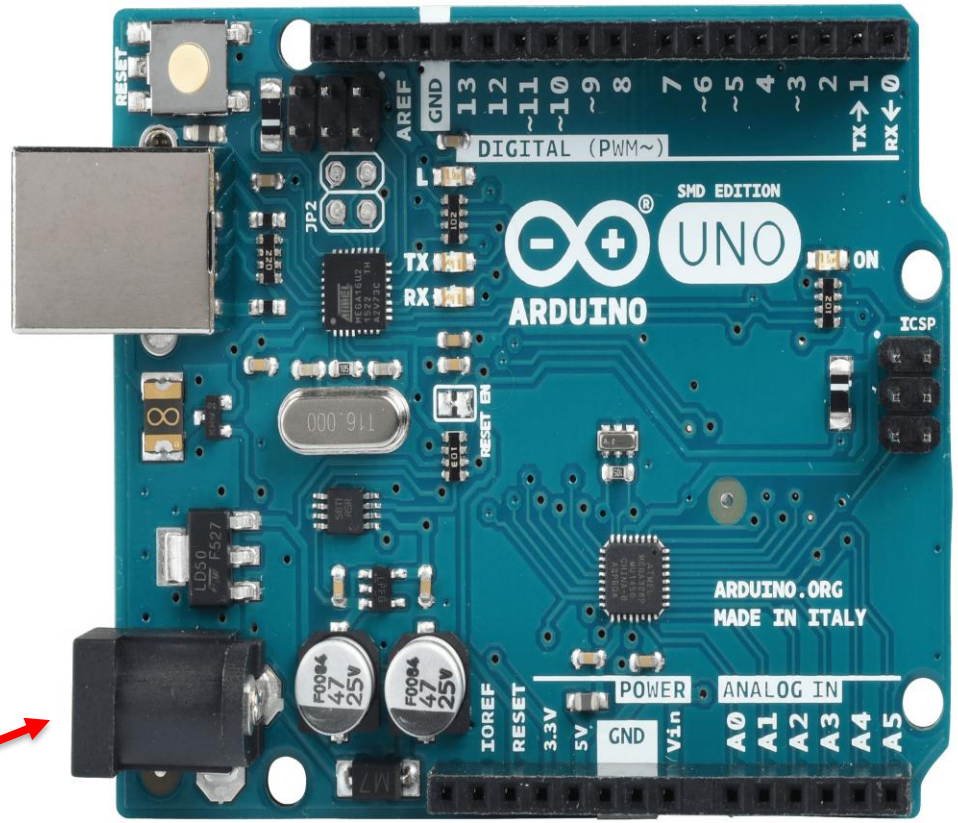


Arduino Nano

Arduino Mini

Lights on an Arduino (Arduino UNO)

- 4 Lights on an Arduino
  - 1.ON Light - shows that it is powered on
  - 2. TX Light - shows that data is being transmitted
  - 3. RX Light - shows that data is being received
  - 4. L Light - an LED you are able to control in your program
  - Serial Port (USB)
  - External Power Source (AA, AAA Batteries)



Topic 1: Meet Arduino Uno

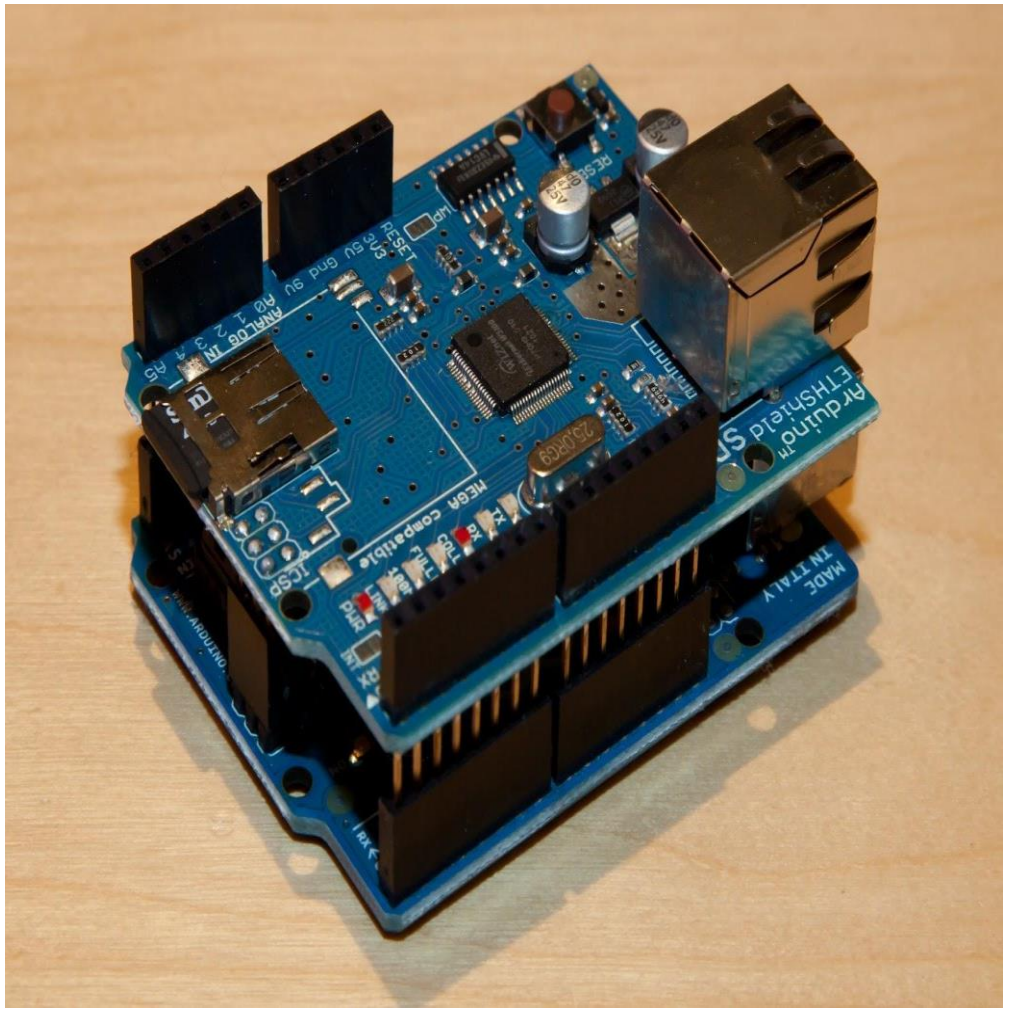
Arduino Uno

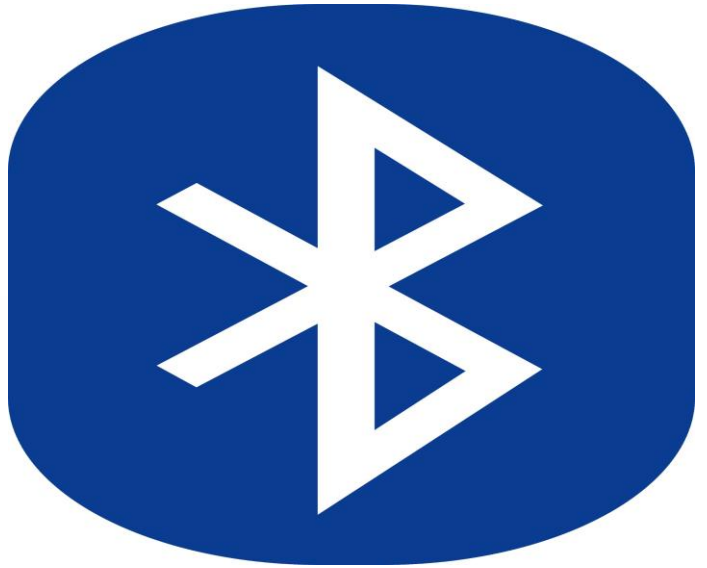
1) Power In (Barrel Jack): Can be used with either a 9V or 12V wall-wart or battery
2) Power In (USB Port): Provides power and communicates with your board when plugged into your computer via USB.
3) LED (RX: Receiving): Lights up when the Arduino is receiving data (such as when being programmed).
4) LED (TX: Transmitting): Lights up when your Arduino is transmitting data (such as when running a program).
5) LED (Pin 13: Troubleshooting): LED is incorporated into your sketch to show if your program is running properly.
6) Pins (ARef, Ground, Digital, Rx, Tx): Used for inputs, outputs, power, and ground.
7) LED (Indicates Arduino is ON): Simple power indicator LED.
8) Reset button: Used to manually reset Arduino, which makes code restart.
9) ICSP Pins: For “in-circuit serial programming,” used if you want to bypass the bootloader.
10) Pins (Analog In, Power In, Ground, Power Out, Reset): Used for inputs, outputs, power, and ground.

Arduino Accessories & Shields

- Accessories include USB A-B cable, external power source, breadboard, resistors, variable resistors (potentiometers), switches, wires, sensors, motors
- Shields" are add ons or accessories that extend the functionality of Arduino. The code is already written for them
  - Ethernet Shields
  - LCD Shields
  - Motor Shields extends the number of motors you can use on an arduino from 6
  - Prototype Shield - for circuit development rather than soldering
    - Use breadboard as an alternative to this shield
  - There are many more shields including bluetooth, wireless, etc.
    - Arduino models allow you to connect a battery or AC/DC converter to run without being connected to a computer

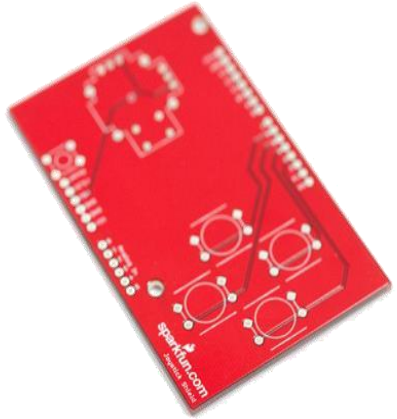
Arduino Accessories & Shields



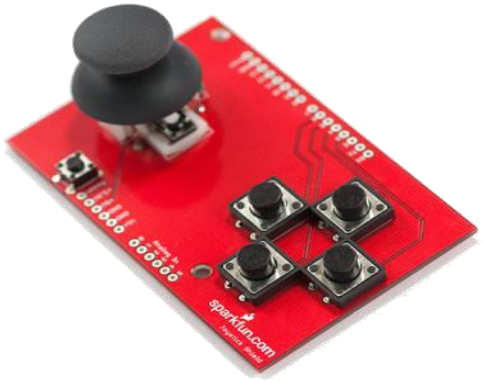


Arduino Shields

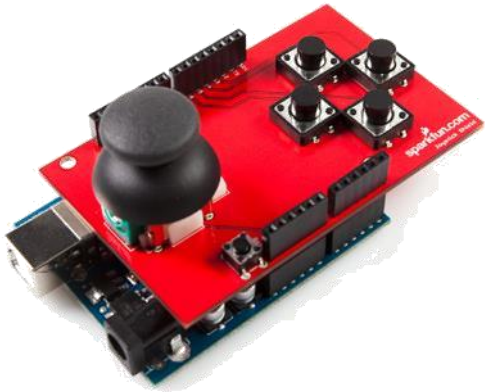
PCB



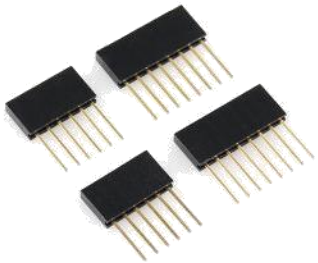
Built Shield



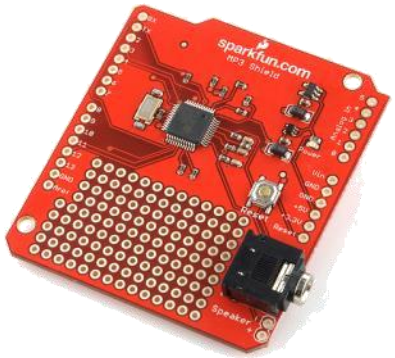
Inserted Shield



Micro SD

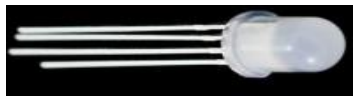


MP3 Trigger



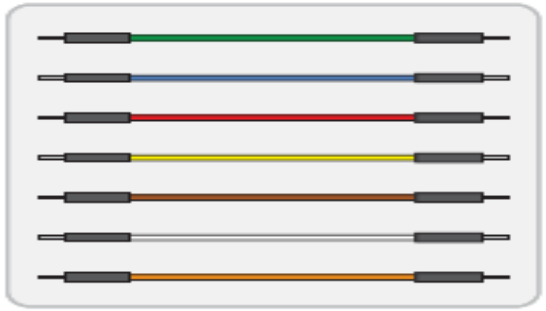
LCD

SIK Components

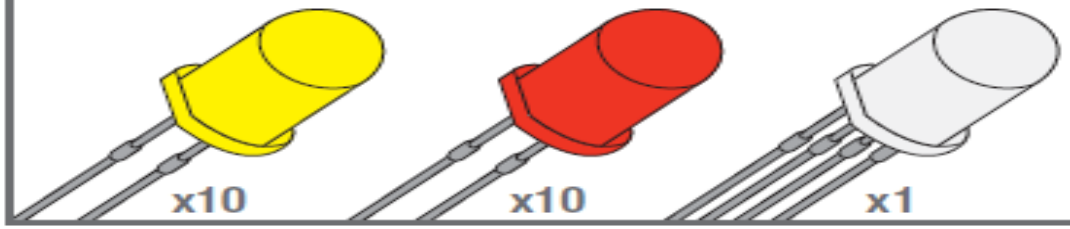
Name	Image	Type	Function	Notes
Push Button		Digital Input	Switch - Closes or opens circuit	Polarized, needs resistor
Trim potentiometer		Analog Input	Variable resistor	Also called a Trimpot.
Photoresistor		Analog Input	Light Dependent Resistor (LDR)	Resistance varies with light.
Relay		Digital Output	Switch driven by a small signal	Used to control larger voltages
Temp Sensor		Analog Input	Temp Dependent Resistor	
Flex Sensor		Analog Input	Variable resistor	
Soft Trimpot		Analog Input	Variable resistor	Careful of shorts
RGB LED		Dig & Analog Output	16,777,216 different colors	Ooh... So pretty.

SIK Components

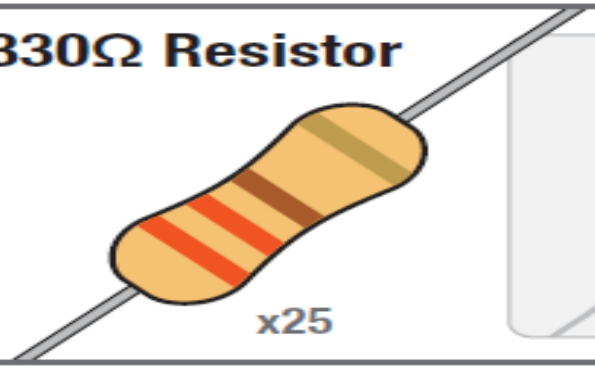
Jumper Wire
Various Colors



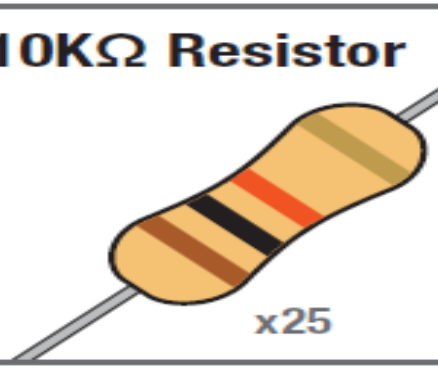
LED (5mm)
(Light Emitting Diode)



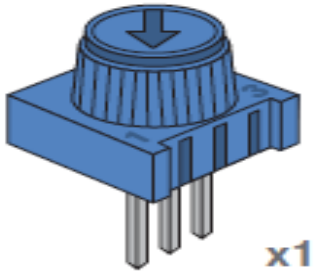
330Ω Resistor

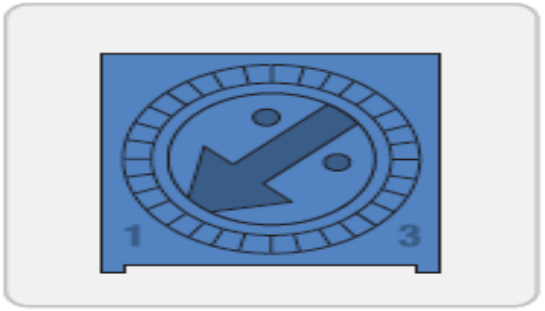


10KΩ Resistor

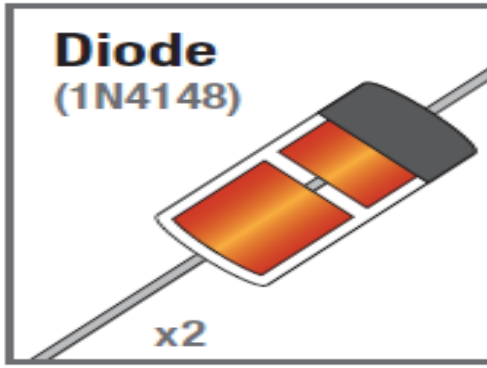


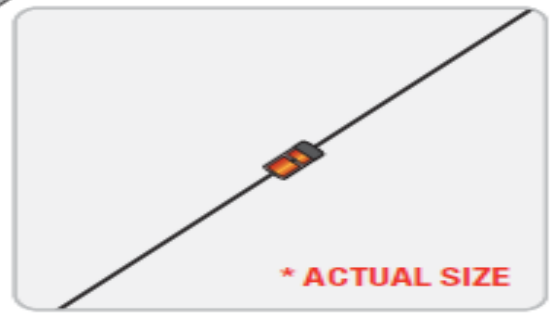
Potentiometer





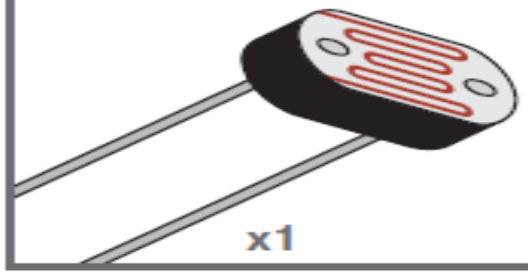
Diode
(1N4148)



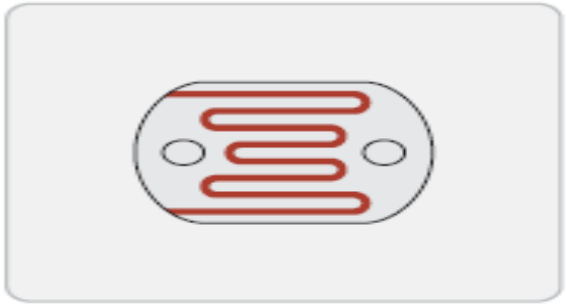


SIK Components

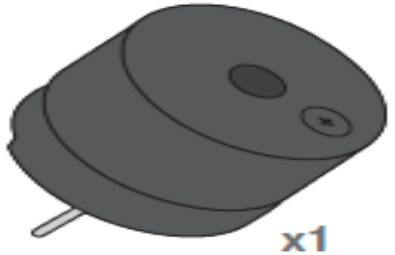
Photo Resistor



x1

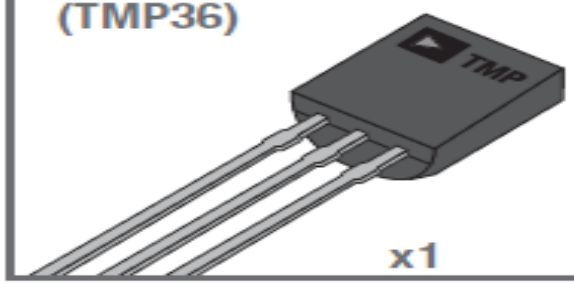


Piezo Element

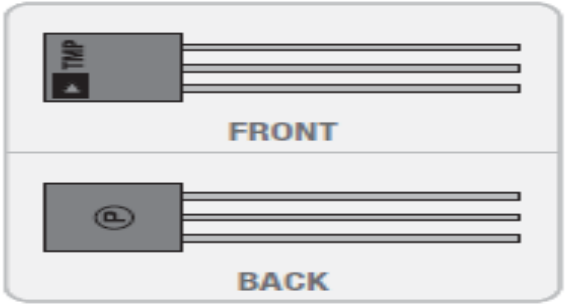


x1

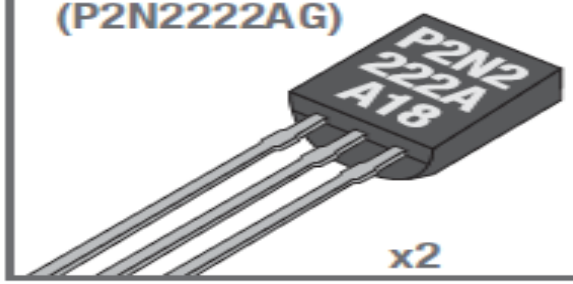
Temp. Sensor
(TMP36)



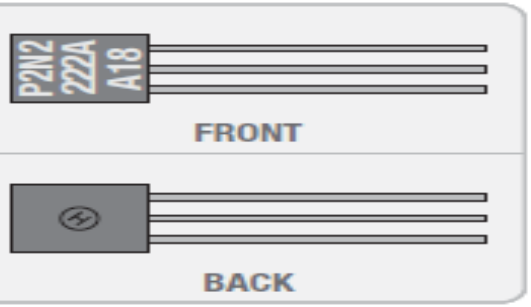
x1



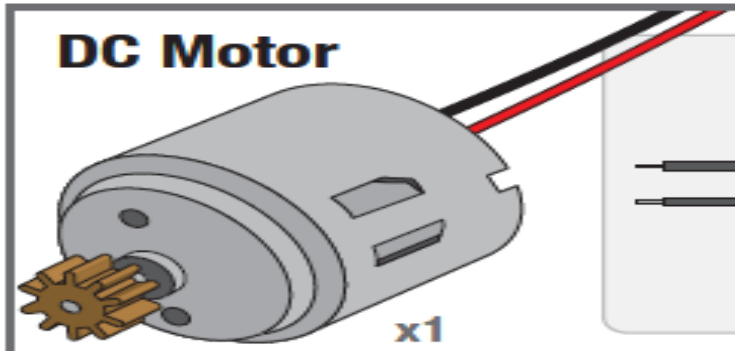
Transistor
(P2N2222AG)



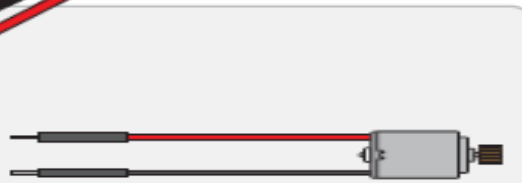
x2



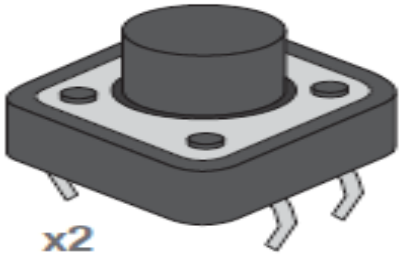
DC Motor



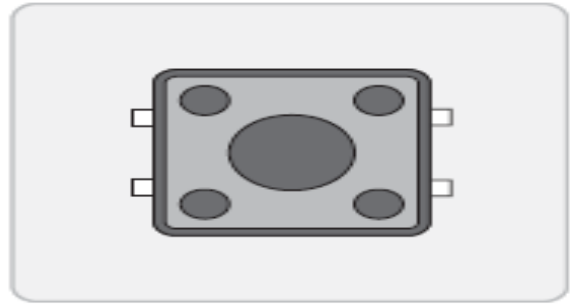
x1



Push Button

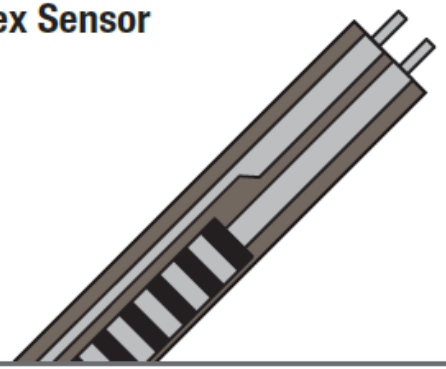


x2



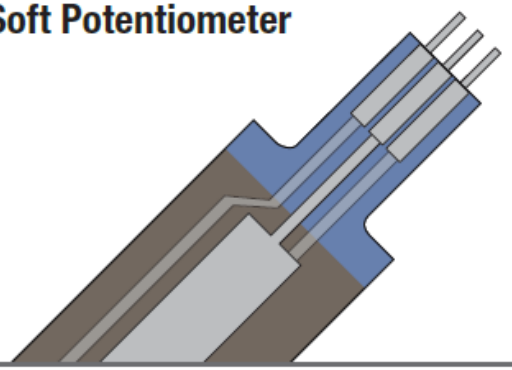
SIK Components

Flex Sensor



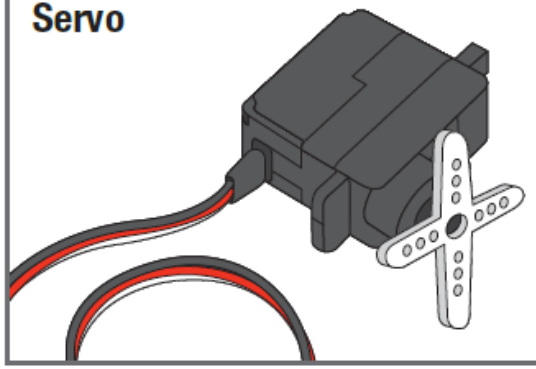
x1

Soft Potentiometer



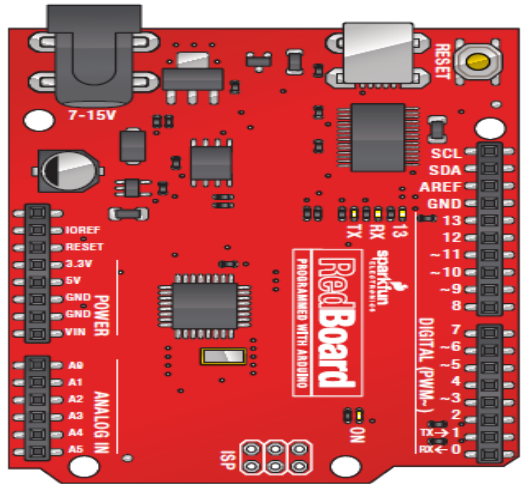
x1

Servo



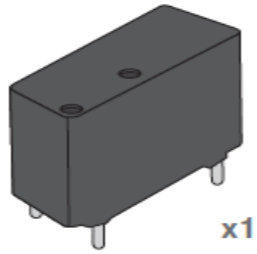
x1

SparkFun RedBoard



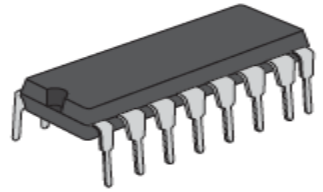
x1

Relay



x1

Integrated Circuit (IC)



x1

LCD

x1

Breadboard

Standard Solderless (Color may vary)

x1

Basic components of the Arduino Microcontroller

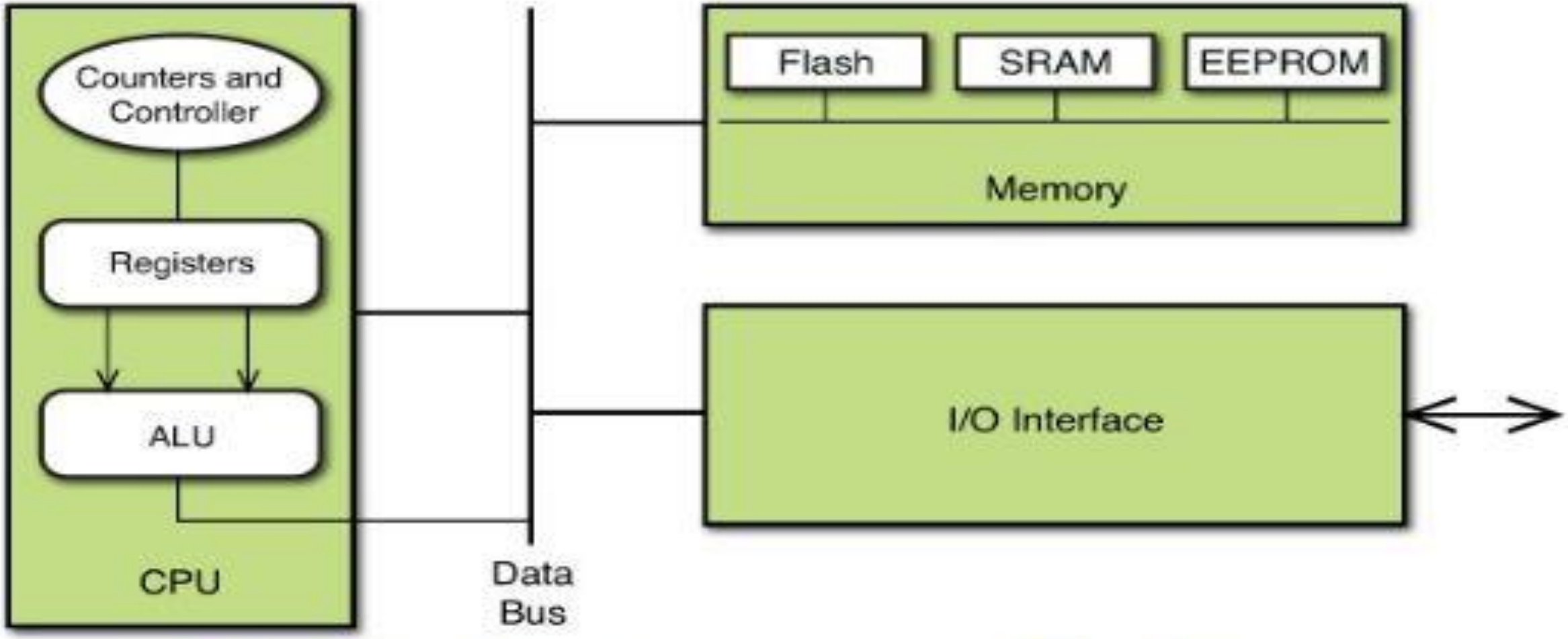


FIGURE 2.1 The ATmega microcontroller block layout.

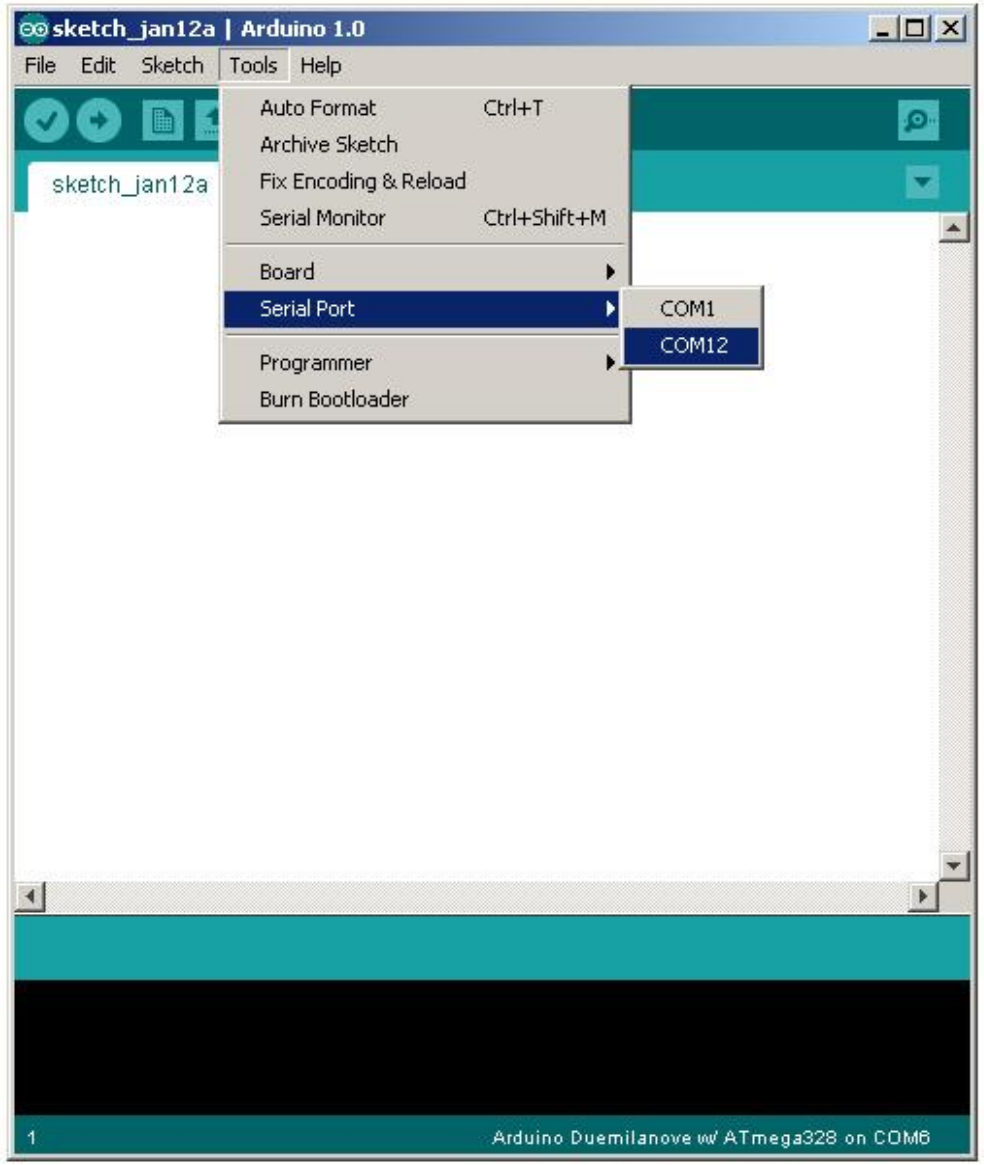
Getting Started

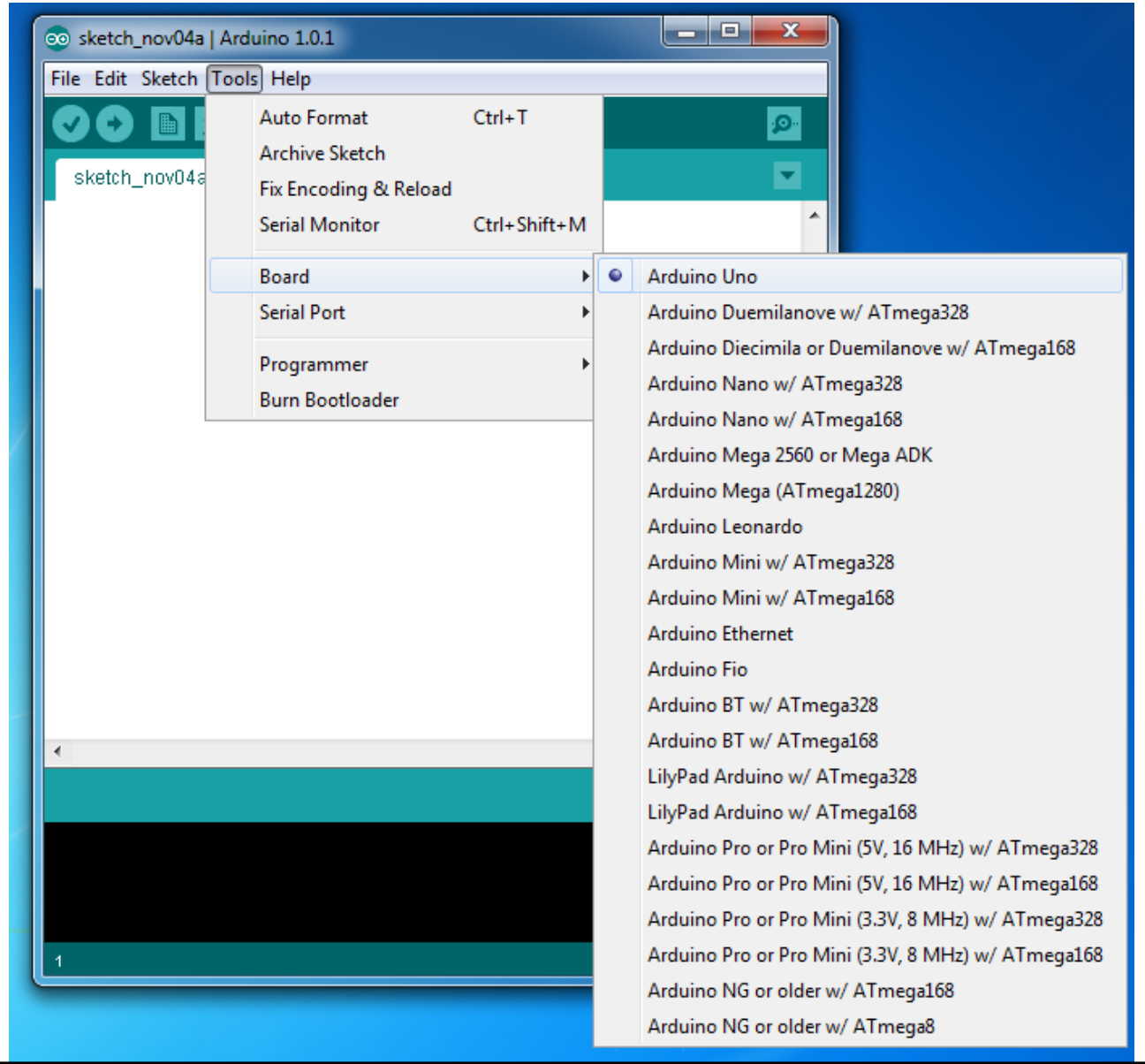
- Check out: http://arduino.cc/en/Guide/HomePage
  1. Download & install the Arduino environment (IDE)**(not needed in lab)**
  2. Connect the board to your computer via the USB cable
  3. If needed, install the drivers **(not needed in lab)**
  4. Launch the Arduino IDE
  5. Select your board
  6. Select your serial port
  7. Open the blink example
  8. Upload the program

Arduino IDE

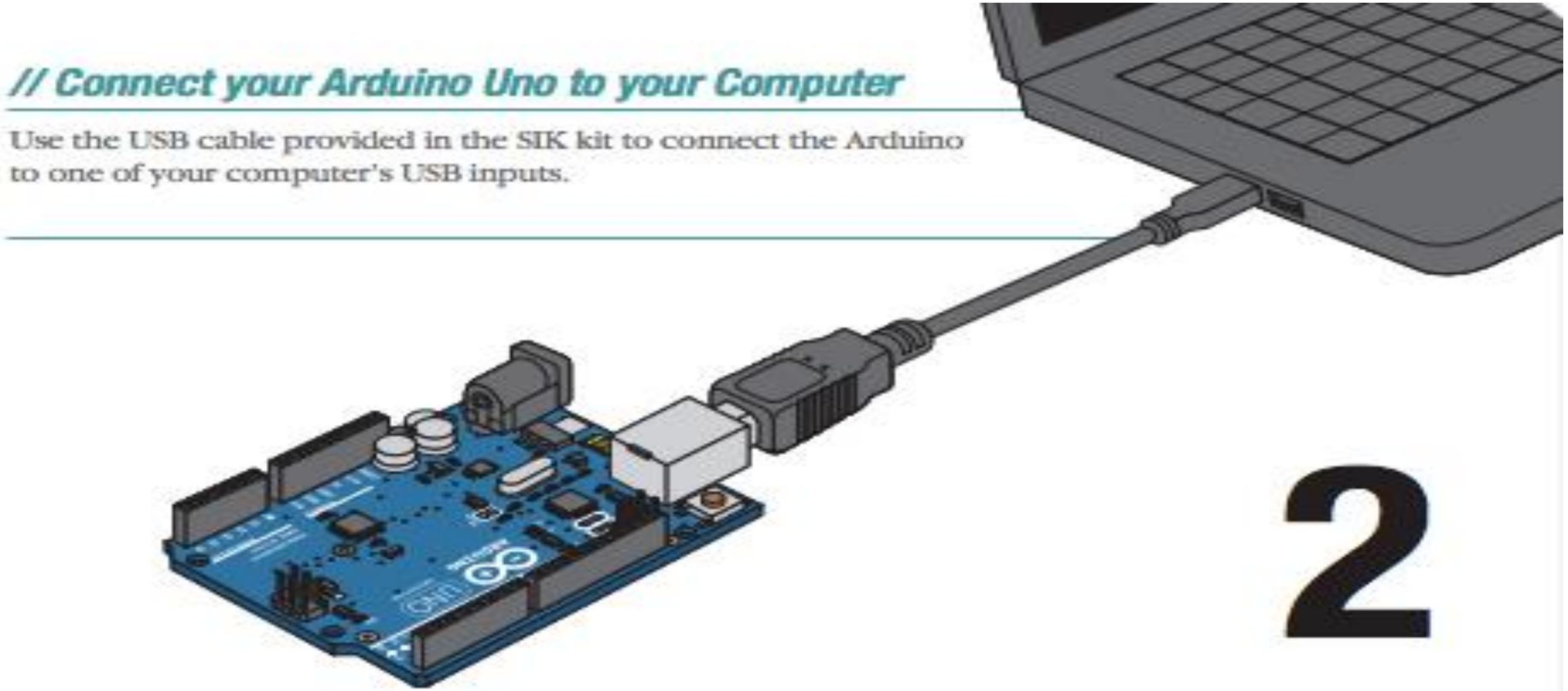
See: http://arduino.cc/en/Guide/Environment for more information

Select Serial Port and Board





Connect RedBoard to your Computer



Install Arduino Drivers

3

// Install Drivers

Depending on your computer's operating system, you will need to follow specific instructions. Please consult the URLs below for specific instructions on how to install the drivers onto your Arduino Uno.

* **You will need to scroll to the section labeled "Install the drivers".**

Windows Installation Process

Go to the web address below to access the instructions for installations on a Windows-based computer.

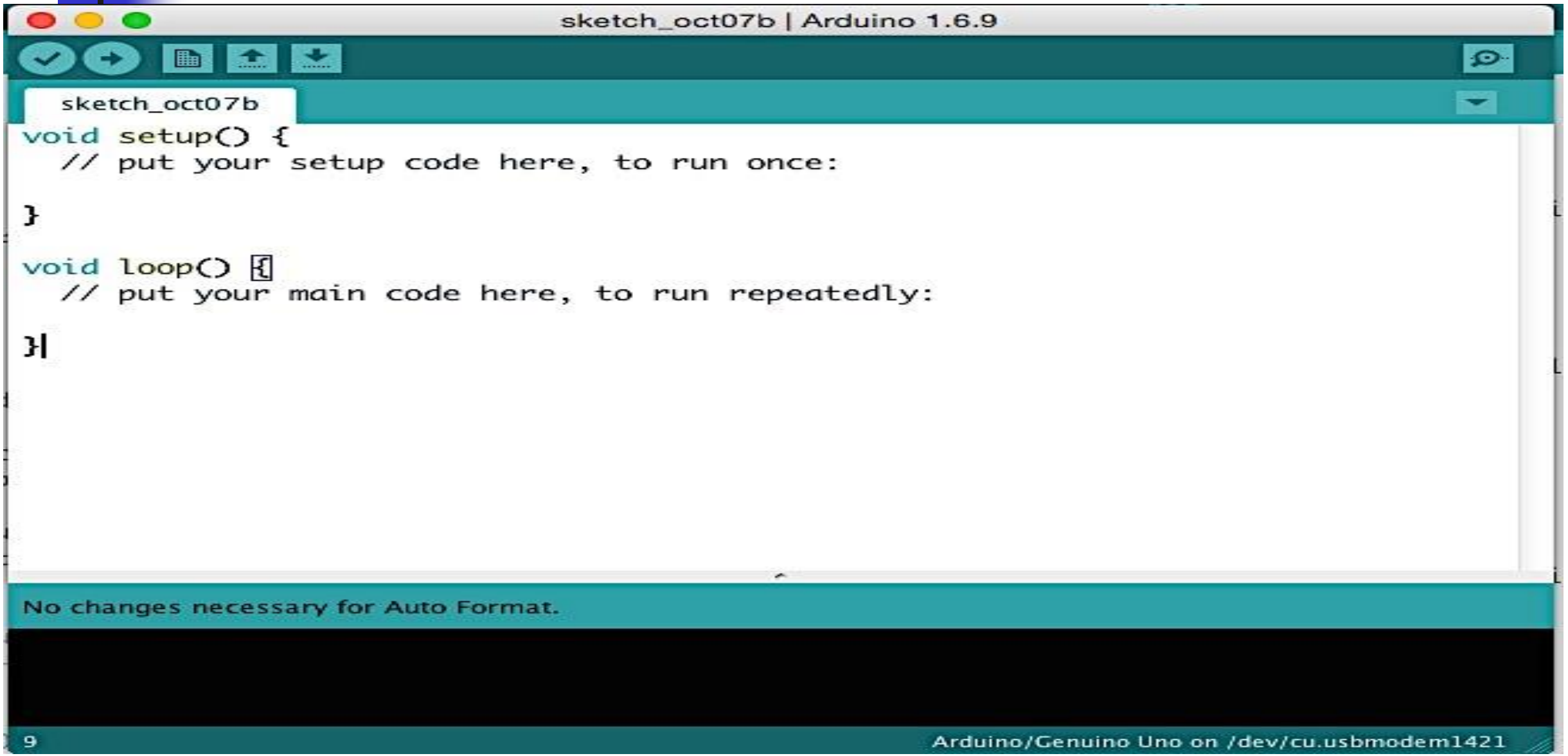
http://arduino.cc/en/Guide/Windows

Macintosh OS X Installation Process

Macs do not require you to install drivers. Enter the following URL if you have questions. Otherwise proceed to next page.

http://arduino.cc/en/Guide/MacOSX

OPEN arduino ide



Arduino Programs – “Sketches”

- Sketch = Program
- Unit of code uploaded to and run on an Arduino board
- 5 parts:
  1. A descriptive header comment block
  2. Definition of global variables
  3. **Setup**() function
  4. **Loop**() function
  5. User-defined functions
- Arduino Language Reference: https://www.arduino.cc/en/Reference/HomePage

Arduino Language

- Roughly based on C
- All statements must end with a ;
- Variables are storage compartments for numbers
- Variables must be defined before use
- // Comments are preceded by two forward slashes

What's the best way to learn the language?

Study the examples and other people's code!!!

Exercise – The “Blink” Sketch

- Plug your Arduino into your computer via the USB cable
- Open the Arduino IDE software
- From the **Tools** drop down menu select:

  Board → Arduino/Genuino UNO

  Port → <whichever port is labeled Arduino/Genuino Uno>
- From the **File** drop down menu select:

  Examples → 01. Basics → Blink

Example Sketch

The header block describes the sketch

Import libraries or declare Global variables here

The **setup**() function

The **loop**() function

User-defined functions here

```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Exercise – The “Blink” Sketch

- Plug your Arduino into your computer via the USB cable
- Open the Arduino IDE software
- From the **Tools** drop down menu select:

  Board → Arduino/Genuino UNO

  Port → <whichever port is labeled Arduino/Genuino Uno>
- From the **File** drop down menu select:

  Examples → 01. Basics → Blink
- On the toolbar, click the “verify” button
- On the toolbar, click the “upload” button

Closer Look – the setup() function

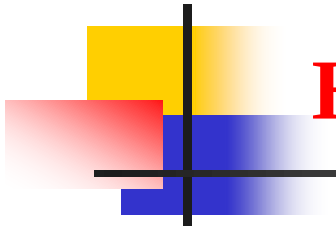
- Run once when you power the Arduino or when you send a new program
- Put everything in the **setup**() function that you want to happen before the main program loop starts

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
```

Closer look – the loop() function

- Runs continuously as long as the Arduino is powered
- Yep – its an endless loop!
- Put everything in the **loop**() function that you want to execute continuously
  - Measurements
  - Control logic
  - Data output
  - Etc.

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

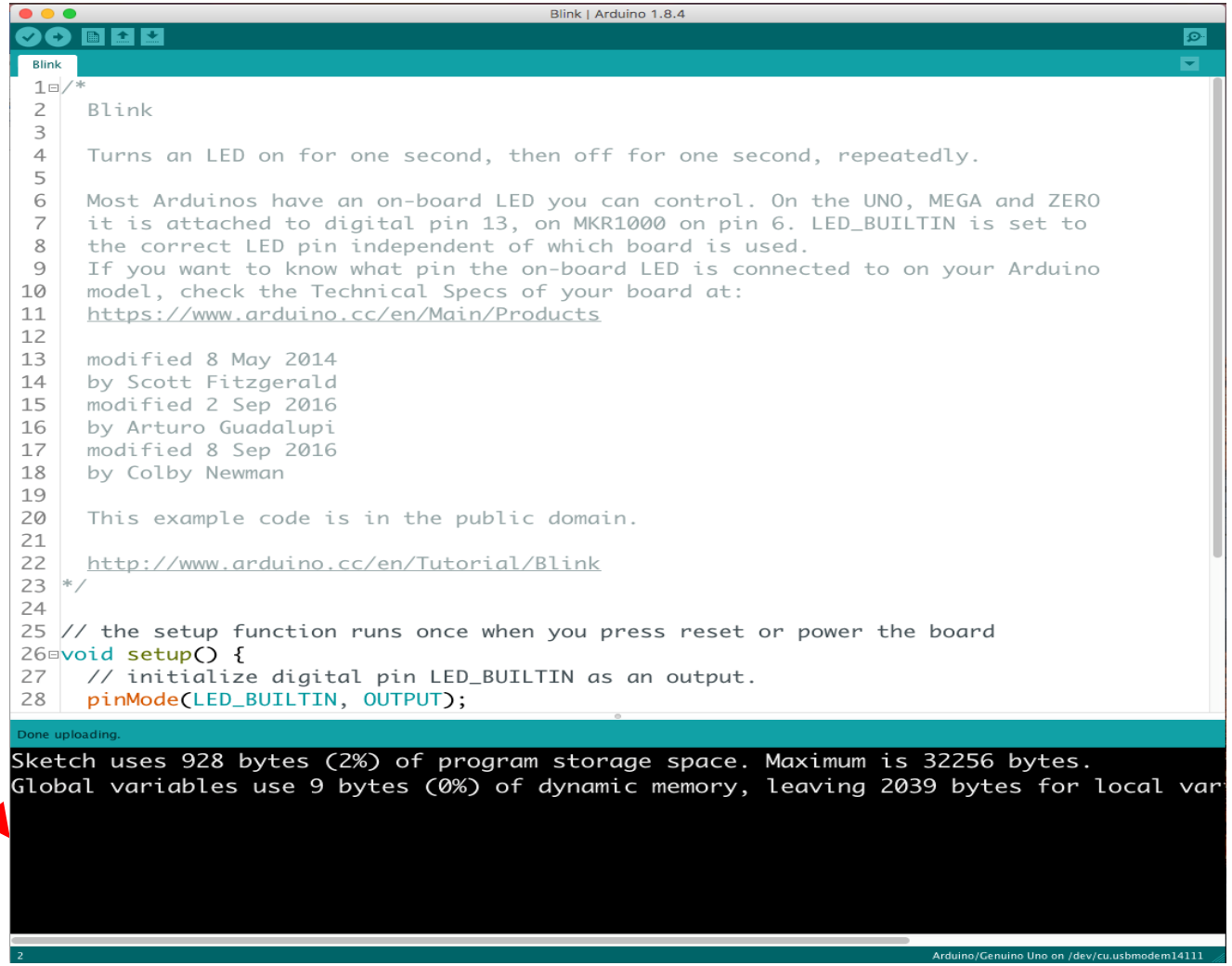


How do we turn an Arduino into a datalogger?

- Some things we have to figure out:
  - Debugging
  - Timing
  - Interfacing with sensors and making measurements
  - Recording data to a file

Debugging Using the Output Pane

- When you compile your code, errors will show up here



Exercise – "Blink_Example2"

- Send the program, then open the serial monitor by clicking the button on the toolbar  (top right)

Start a serial port and print a line of text to it.

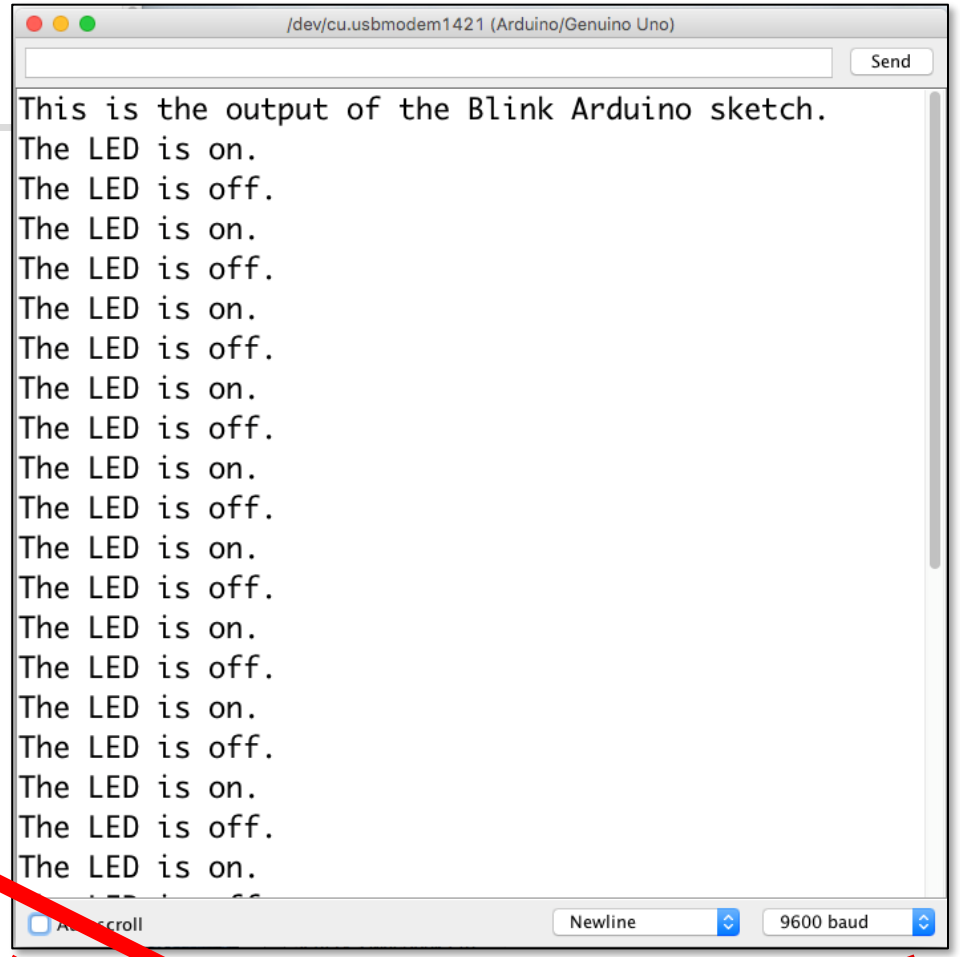
Print a line of text with LED status

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);

  // initialize a serial port
  Serial.begin(9600);
  // print a header line to the serial port
  Serial.println("This is the output of the Blink Arduino sketch.");
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  Serial.println("The LED is on."); // print a message to the serial port
  delay(1000);                       // wait for a second

  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  Serial.println("The LED is off.");
  delay(1000);                       // wait for a second
}
```

Debugging with Serial Output

- Useful when you want to see what's going on as the program executes
- Print values and messages to a serial monitor
- The Arduino IDE has it's own serial monitor
- Super helpful for debugging

Make sure the baud rate matches your Serial.begin() statement in your sketch



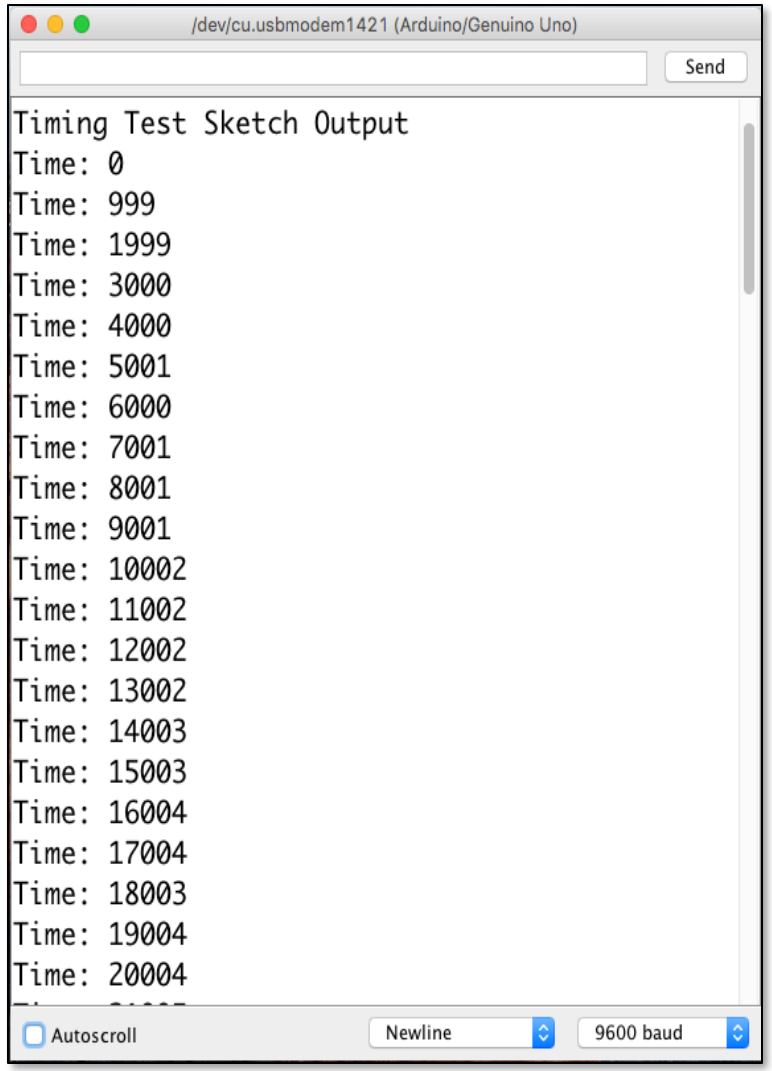
Timing

- The **loop**() function runs indefinitely as fast as it can
- The loop itself takes a couple of clock cycles, but total time is dependent on what is in the loop
- Arduino UNO has no realtime clock (bummer!)
- But, it has some useful timing functions:
  - **millis**() – number of milliseconds since the Arduino board began running the current program
  - **micros**() – same as millis, but for microseconds
  - **delay**() – pauses the program for an amount of time (in milliseconds)
  - **delayMicroseconds**() – same as delay but for microseconds

NOTE: **millis**() and **micros**() will overflow and go back to zero after a certain period of time

Exercise – "Timing_Example1"

```
/*
Test sketch to demonstrate how the timing functions work
Created By:  Jeff Horsburgh
Creation Date: 8/12/2016
*/

// Create a long integer variable to store the number of milliseconds
unsigned long time;

void setup(){
  // Create a serial port and print a header line
  Serial.begin(9600);
  Serial.println("Timing Test Sketch Output");
}

void loop(){
  Serial.print("Time: ");

  // Set the "time" variable to the number of milliseconds since the
  // program started by calling the millis() function
  time = millis();

  // Prints the time since program started to the serial monitor
  Serial.println(time);

  // wait a second so as not to send massive amounts of data
  delay(1000);
}
```



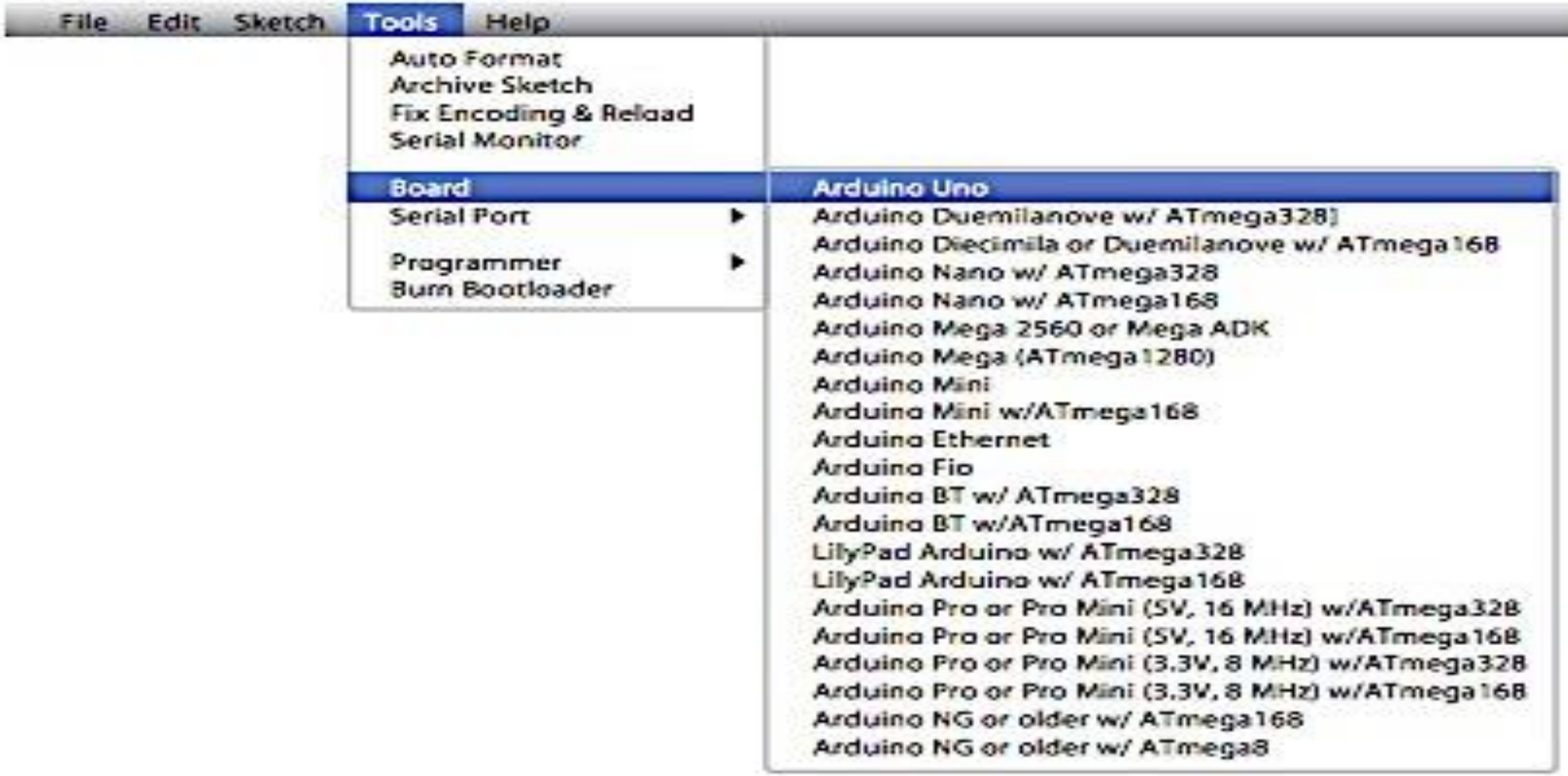
Timing

- **millis**() and **delay**() are great, but not exact
- Plus, **delay**() pauses the program and you can't do anything else at the same time
- What if I want the Arduino to do something (like make a measurement) on a more precise, set time interval?
- What if I want to do multiple things at the same time?

Select your Board

4

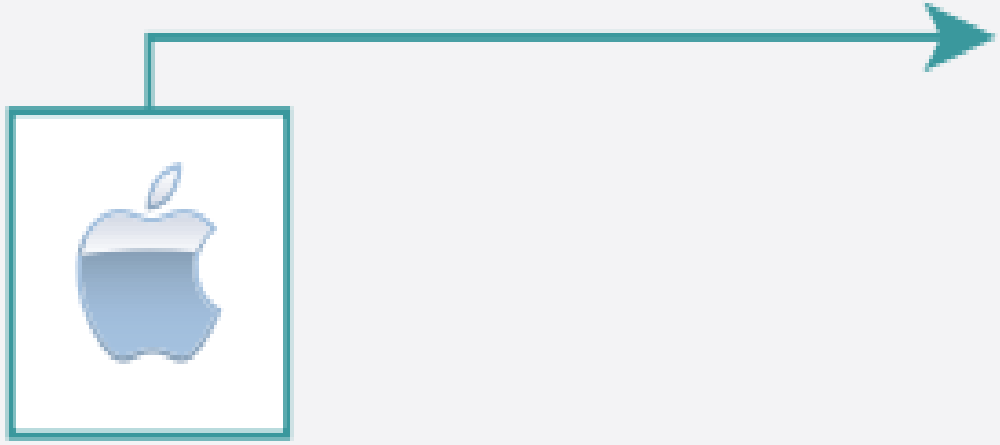
// Select your board: Arduino Uno



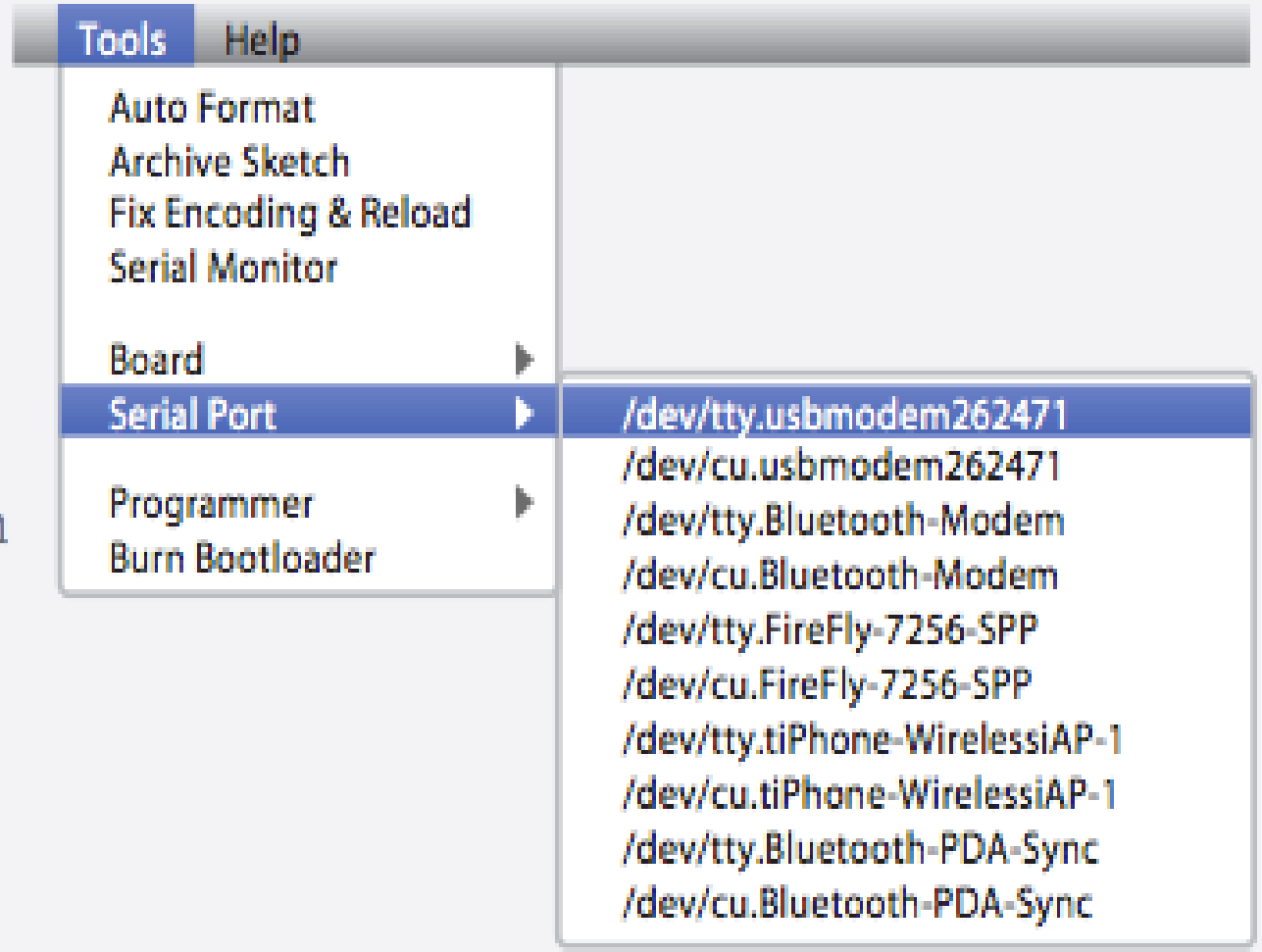
Select Serial Device (Windows)

Select the serial device of the Arduino board from the Tools | Serial Port menu. This is likely to be **com3 or higher** (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port.

Select Serial Device (Mac)

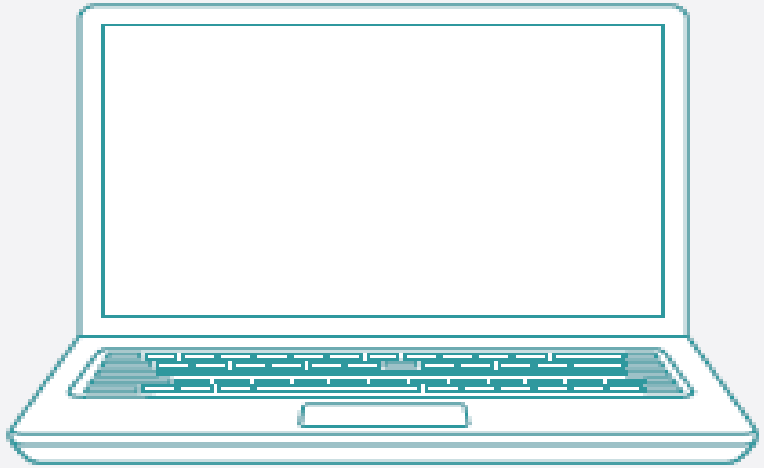


Select the serial device of the Arduino board from the Tools > Serial Port menu. On the Mac, this should be something with **/dev/tty.usbmodem** (for the Uno or Mega 2560) or **/dev/tty.usbserial** (for older boards) in it.



DOWNLOAD Code

5



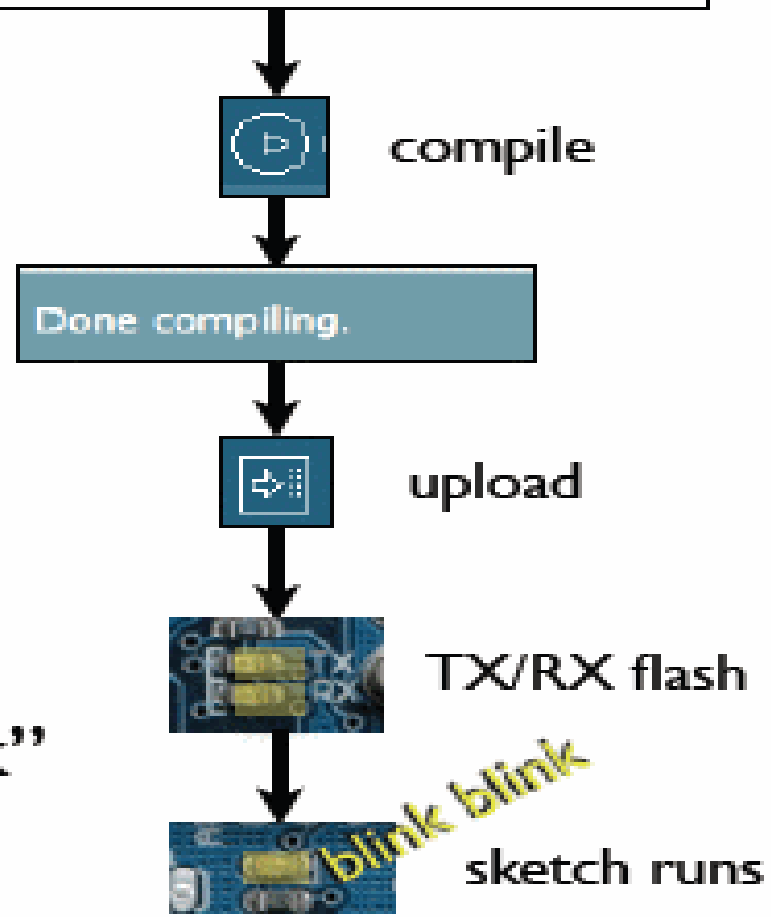
Type in the following URL to download the code:

sparkfun.com/sikcode

Using Arduino

- Write your sketch
- Press Compile button
  (to check for errors)
- Press Upload button to program
  Arduino board with your sketch

```
void setup() {
  pinMode(ledPin, OUTPUT);      // sets t
}
void loop() {
  digitalWrite(ledPin, HIGH);   // sets t
  delay(1000);                  // waits
  digitalWrite(ledPin, LOW);    // sets t
  delay(1000);                  // waits
}
```

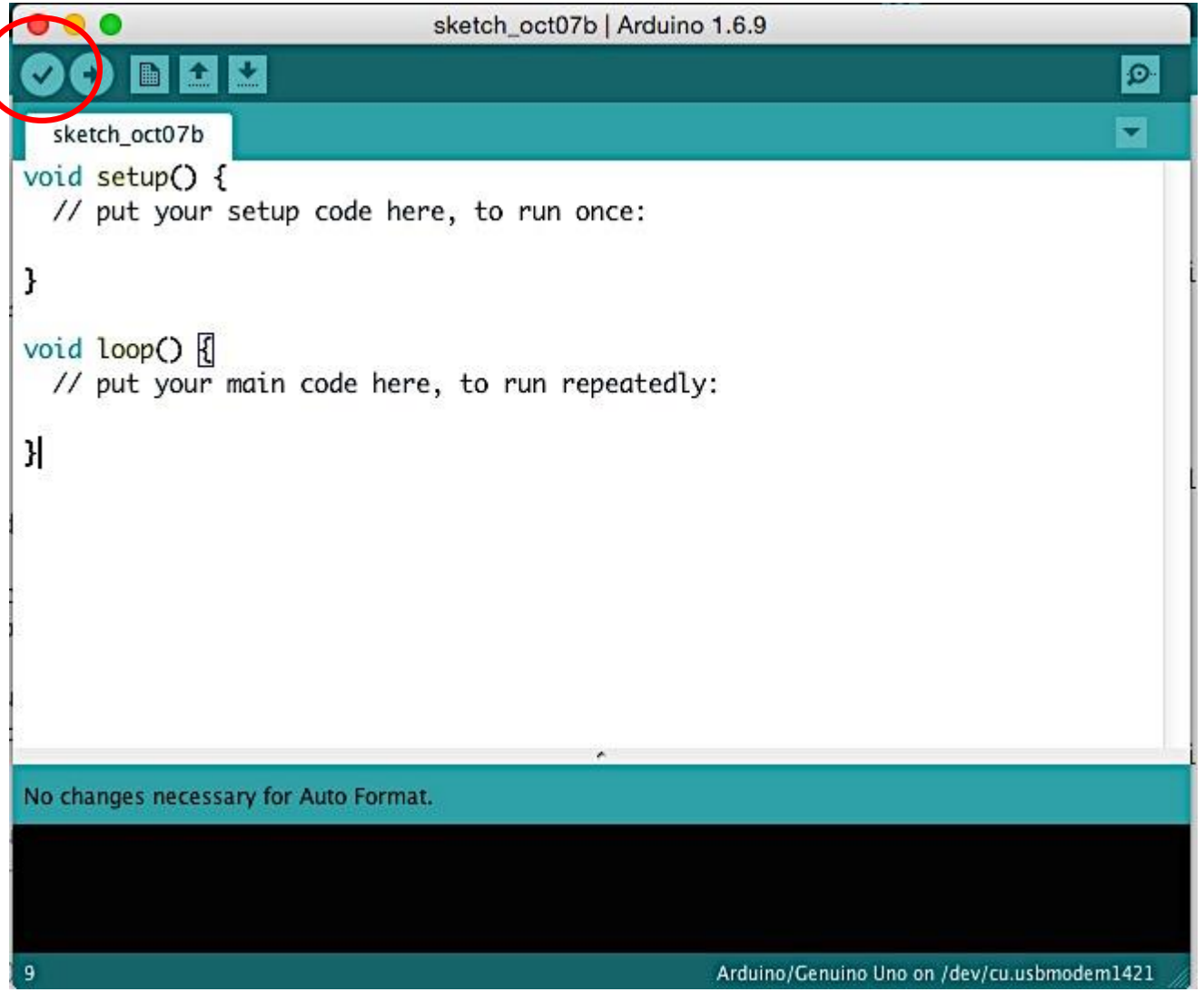


Try it out with the "Blink" sketch!

Load "File/Sketchbook/Examples/Digital/Blink"

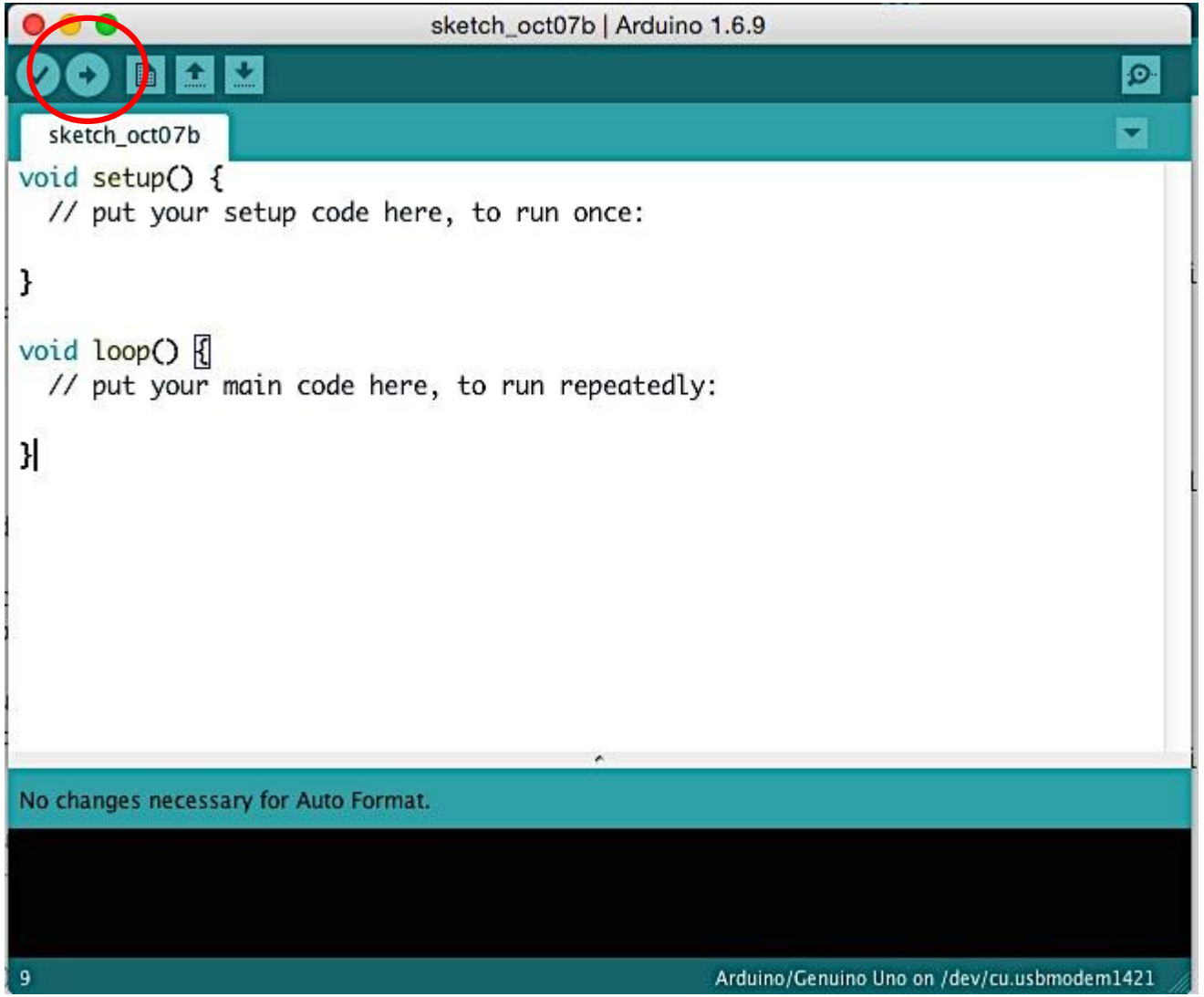
KNOW the Arduino GUI

Verify: Compiles and approves your code. Will catch errors in syntax.



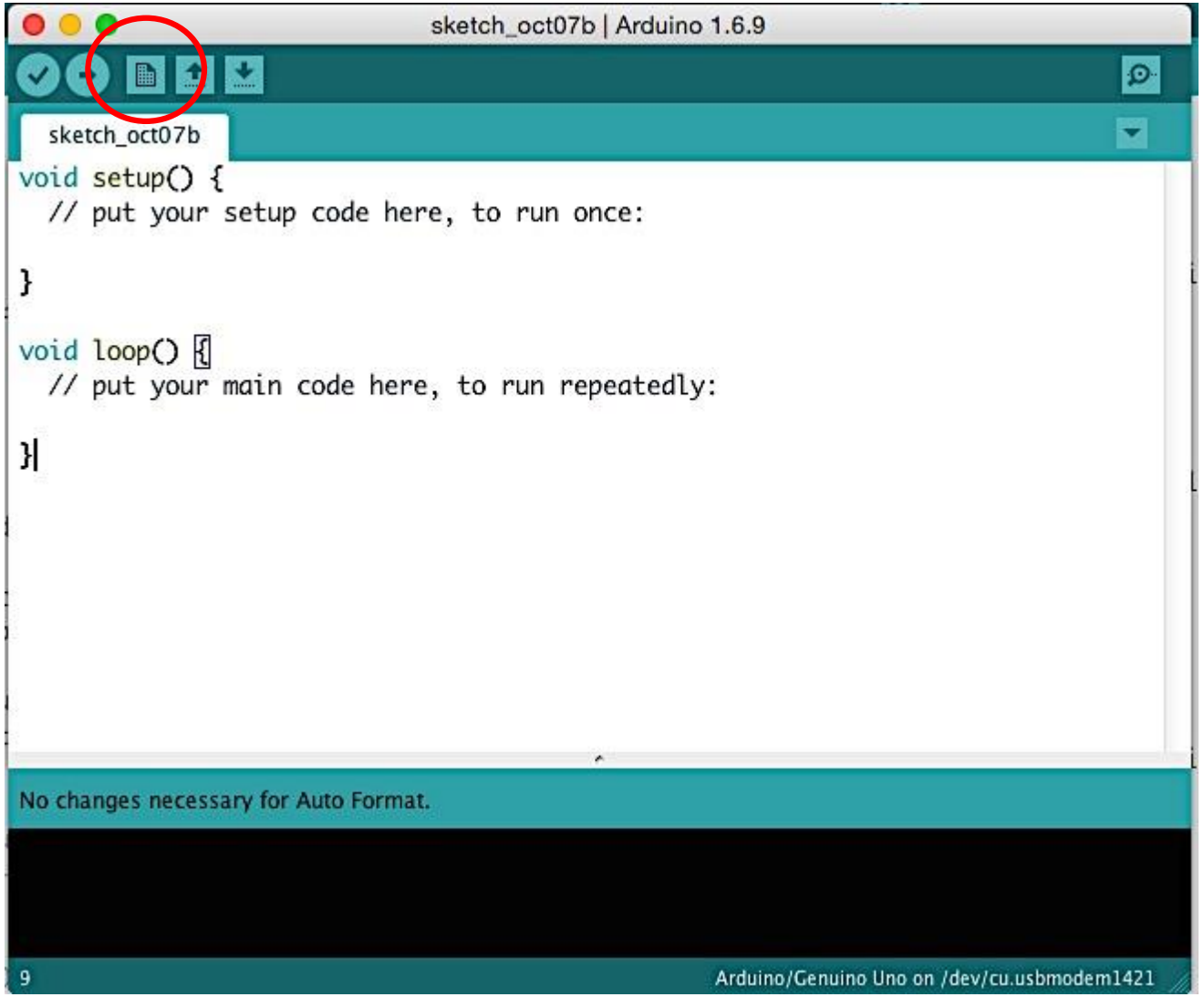
KNOW the Arduino GUI

Upload: Sends your code to the RedBoard. When you click it you should see lights on your board blink rapidly.



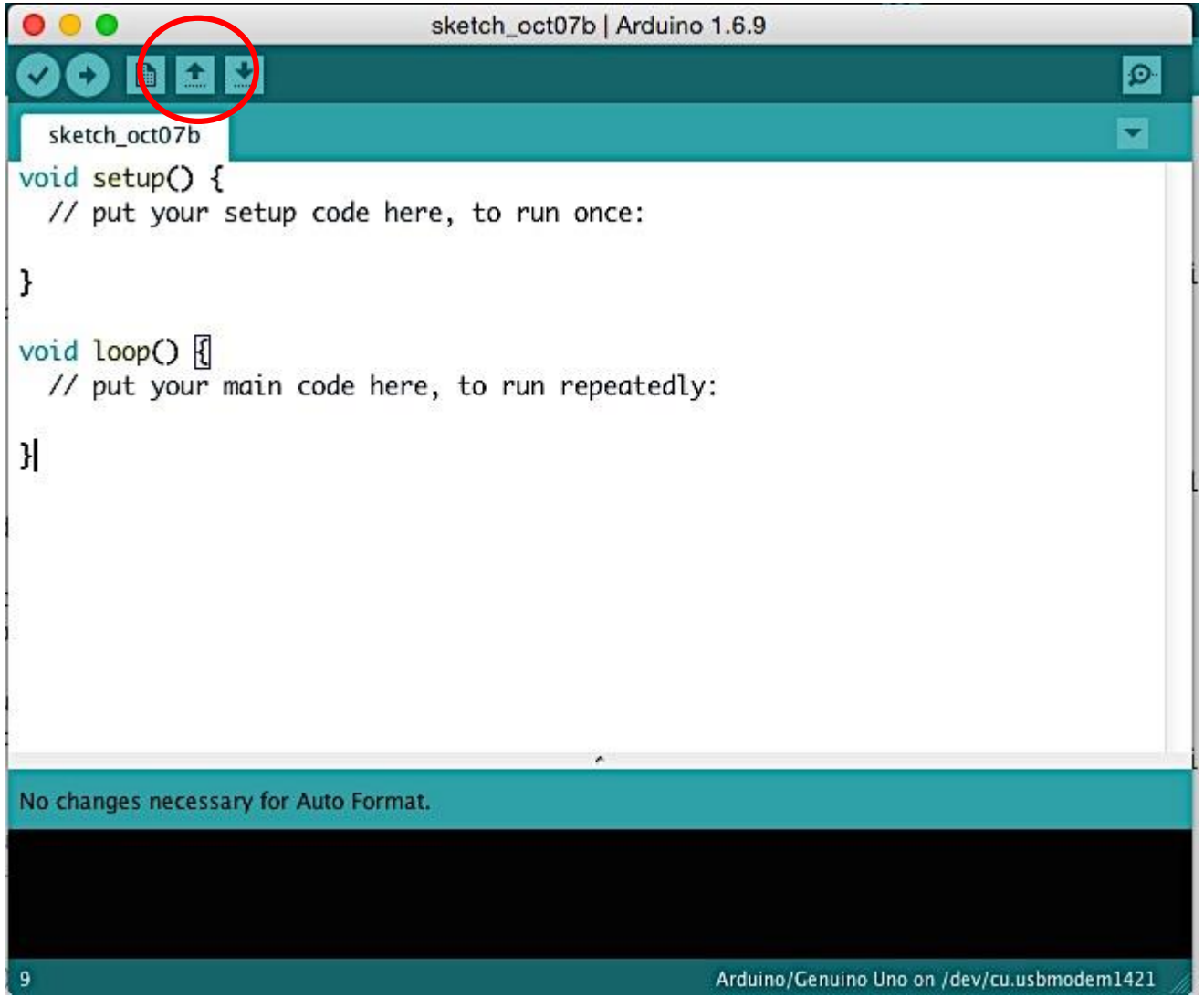
KNOW the Arduino GUI

New: Opens up a new code window tab.



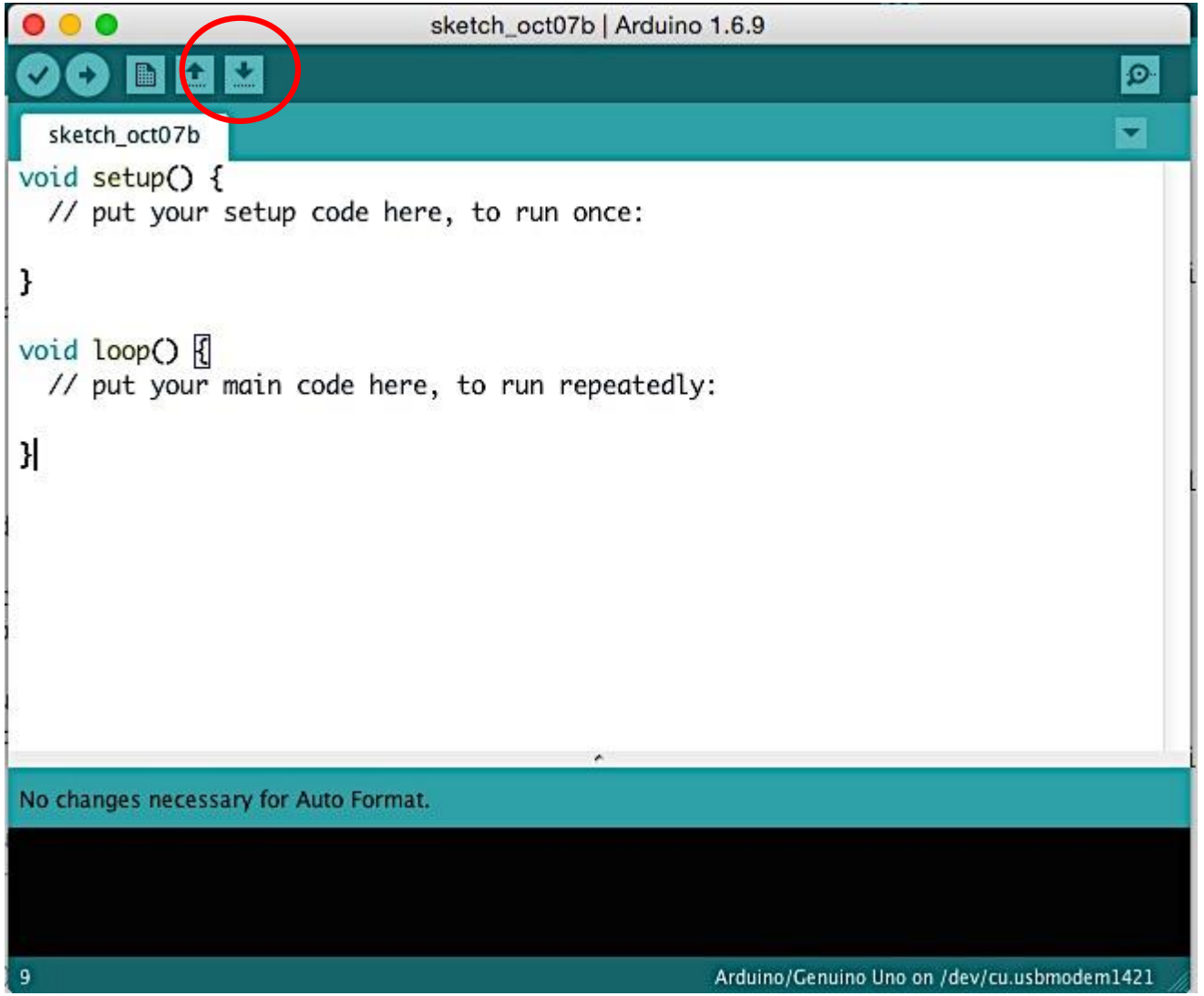
KNOW the Arduino GUI

Open: Open an existing sketch, which is where you write your code.



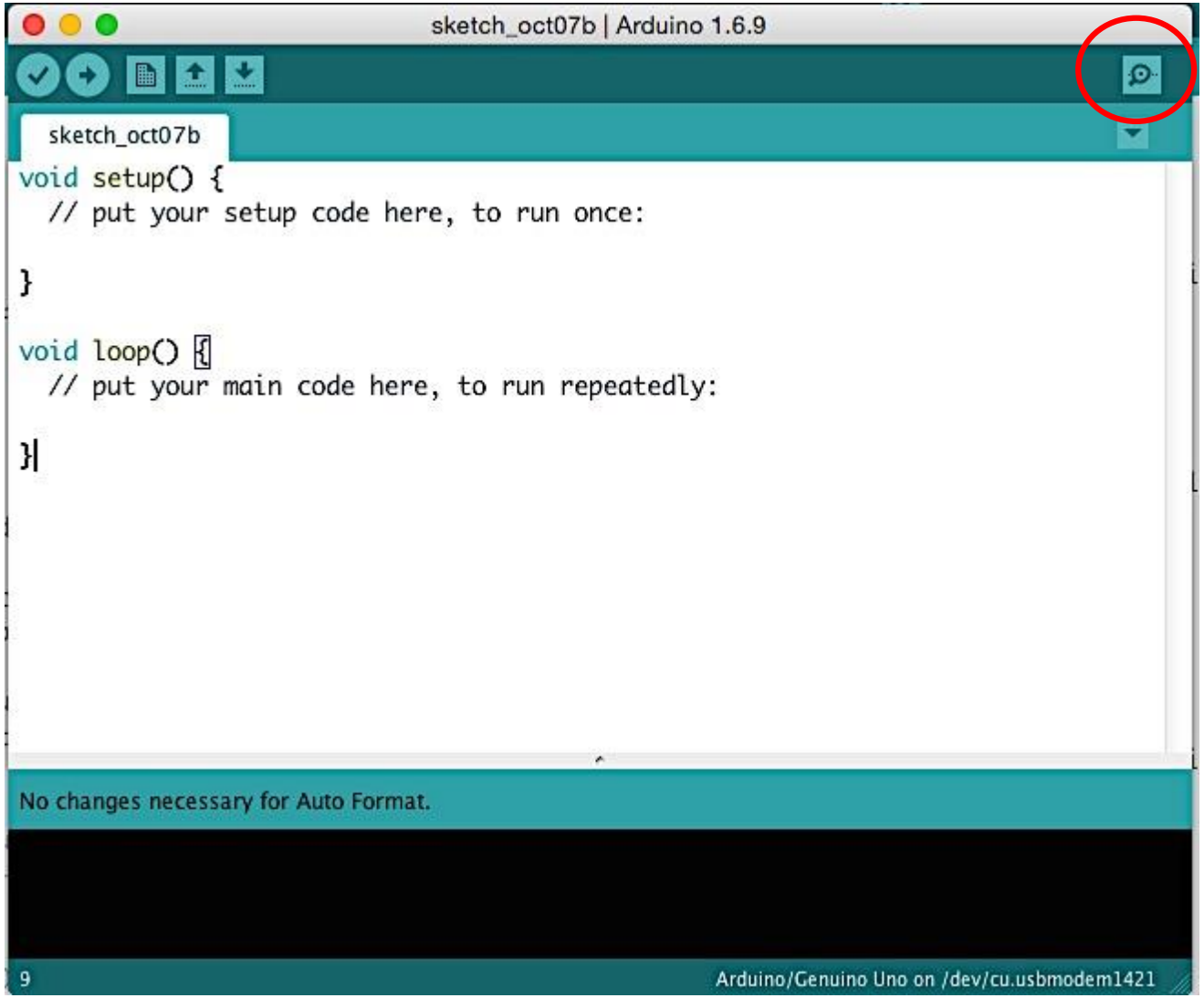
KNOW the Arduino GUI

Save: Saves the currently open sketch.



KNOW the Arduino GUI

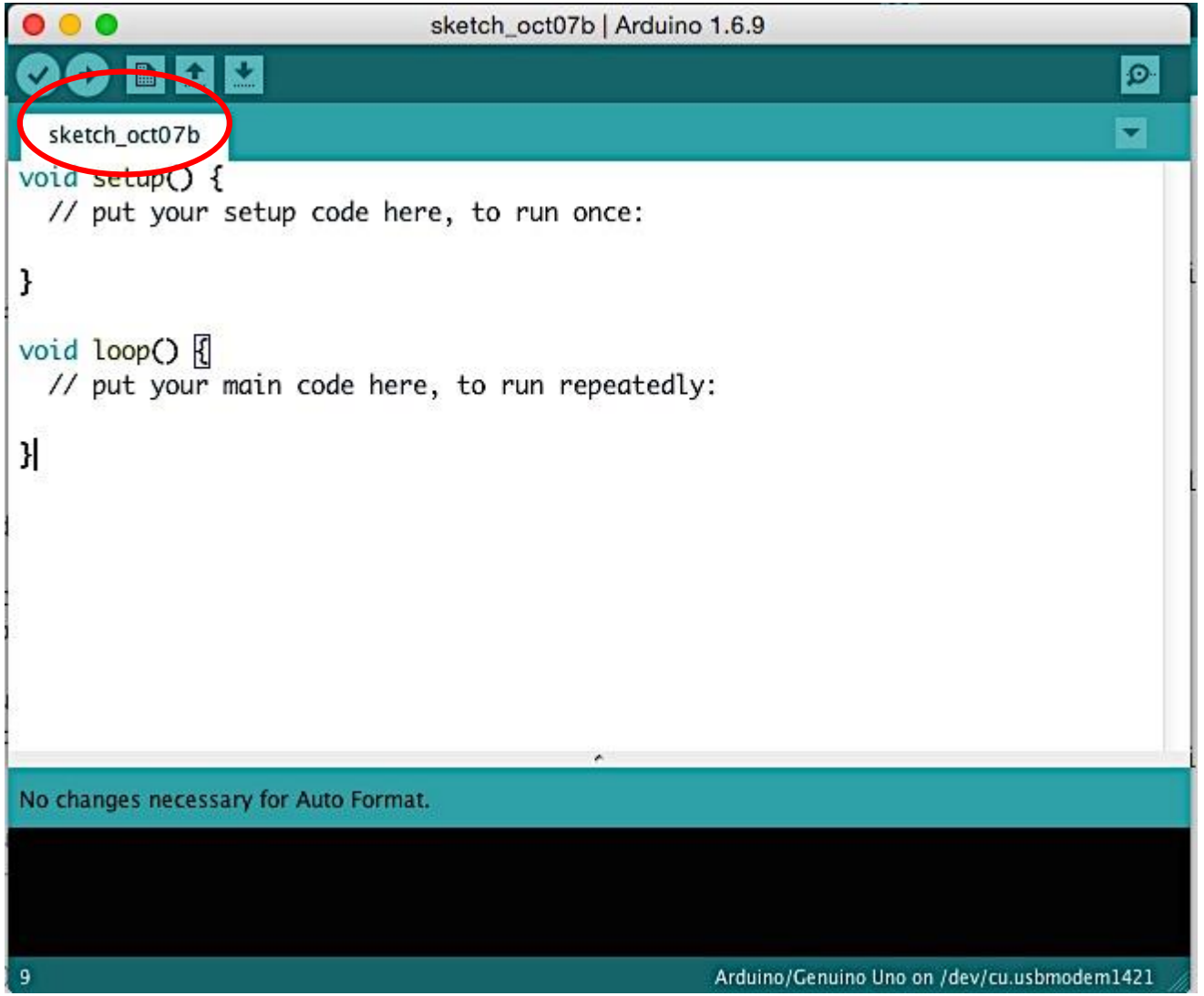
Serial Monitor: Opens a window that displays any serial info the RedBoard is transmitting. Very useful for debugging.



What is Serial?
Process of sending data one bit (0 or 1) at a time.

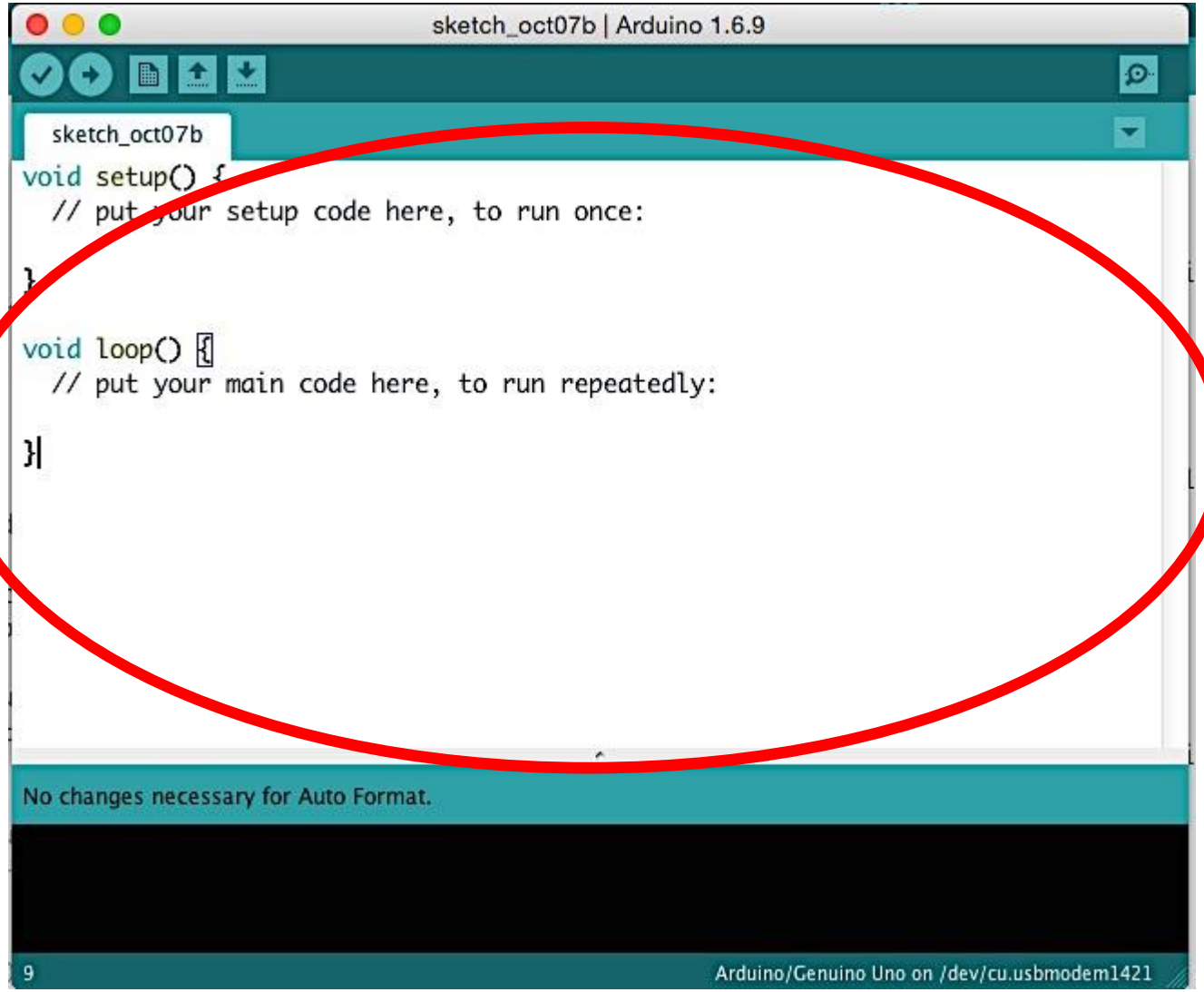
KNOW the Arduino GUI

Sketch Name: Name of the sketch you are currently working on.



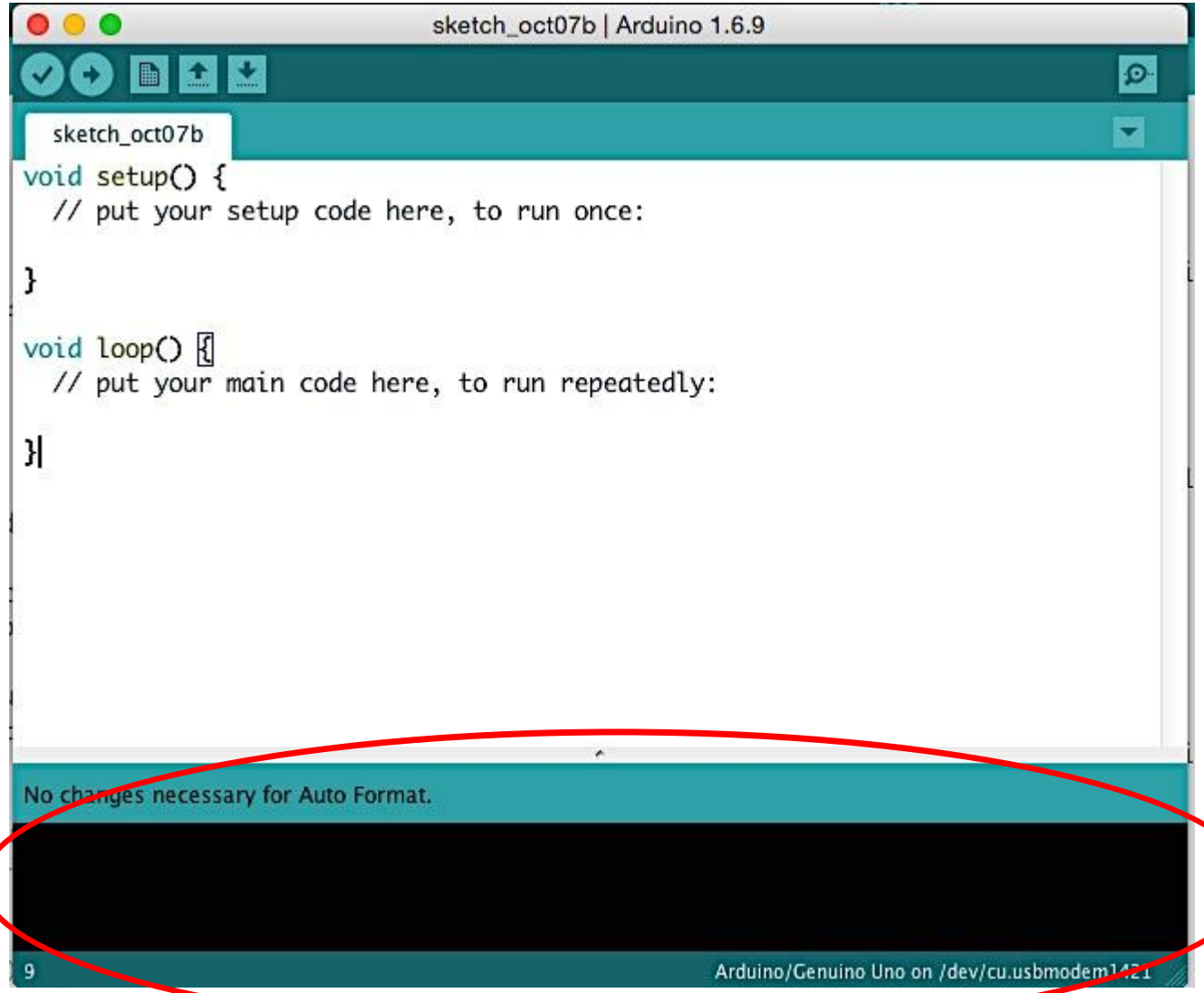
KNOW the Arduino GUI

Code Area: Area where you compose the code for your sketch.



KNOW the Arduino GUI

Message Area: Where the IDE tells you if there were any errors in your code.

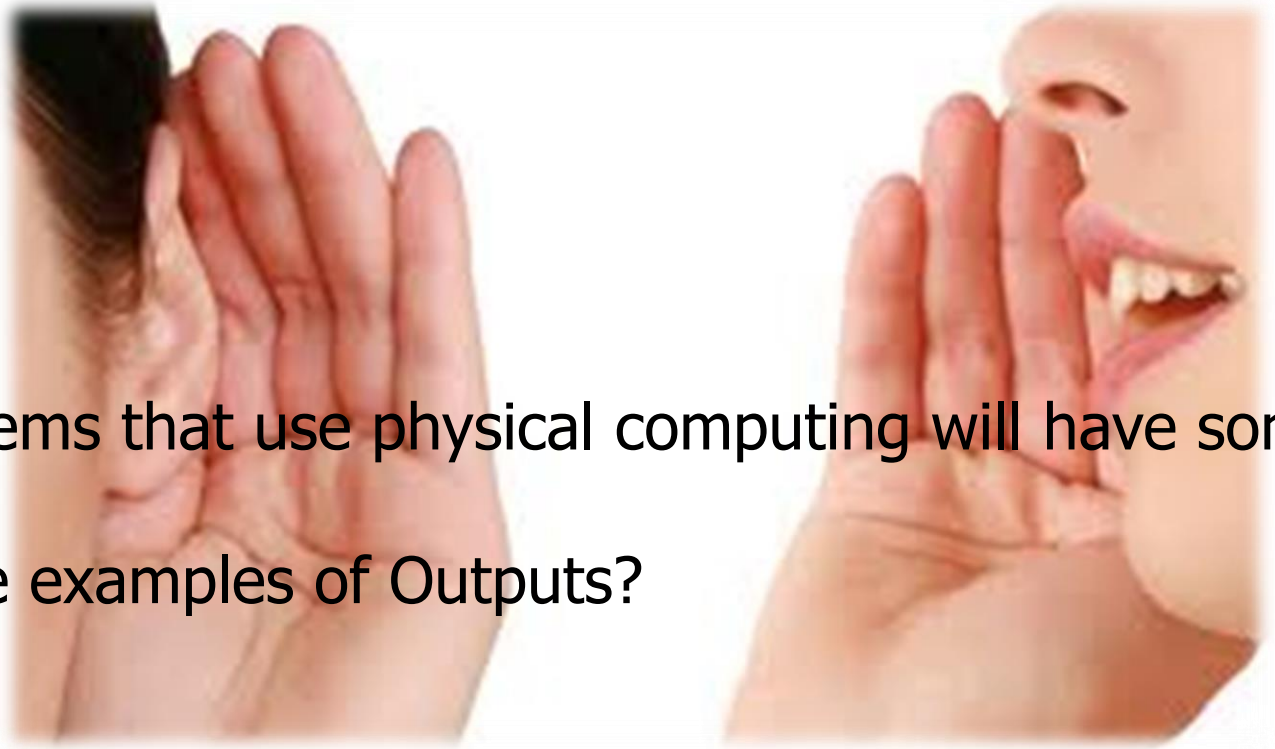


Concepts: INPUT vs. OUTPUT

Referenced from the perspective of the microcontroller (electrical board).

Inputs is a signal / information going into the board.

Output is any signal exiting the board.



Almost all systems that use physical computing will have some form of output

What are some examples of Outputs?

Concepts: INPUT vs. OUTPUT

Referenced from the perspective of the microcontroller (electrical board).

Inputs is a signal / information going into the board.

Output is any signal exiting the board.

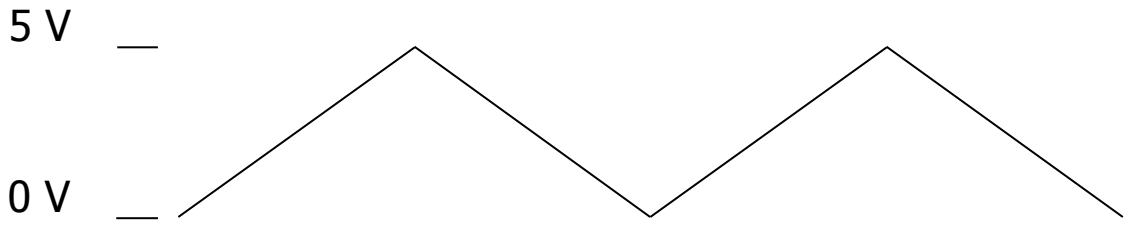
Examples: Buttons Switches, Light Sensors, Flex Sensors, Humidity Sensors, Temperature Sensors…	Examples: LEDs, DC motor, servo motor, a piezo buzzer, relay, an RGB LED

Concepts: Analog vs. Digital

- Microcontrollers are **digital** devices – ON or OFF. Also called – discrete.

- **Analog** signals are anything that can be a full range of values. What are some examples? More on this later…

Digital Signal



Analog Signal

Open up Arduino

- **Hints:**

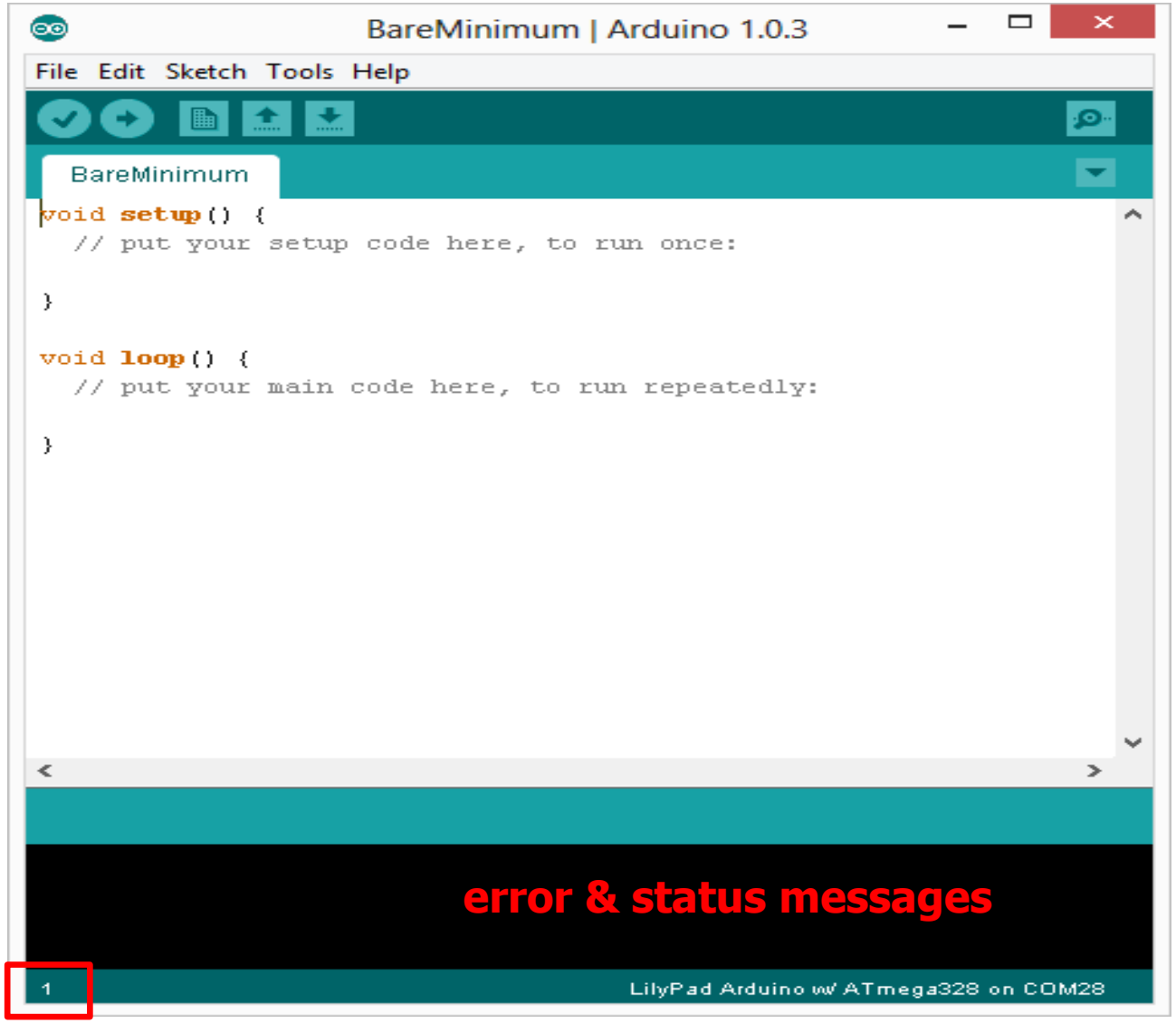
For PC Users →

1. Let the installer copy and move the files to the appropriate locations, or
2. Create a folder under C:\Program Files (x86) called Arduino. Move the entire Arduino program folder here.

For Mac Users →

1. Move the Arduino executable to the dock for ease of access.
2. Resist the temptation to run these from your desktop.

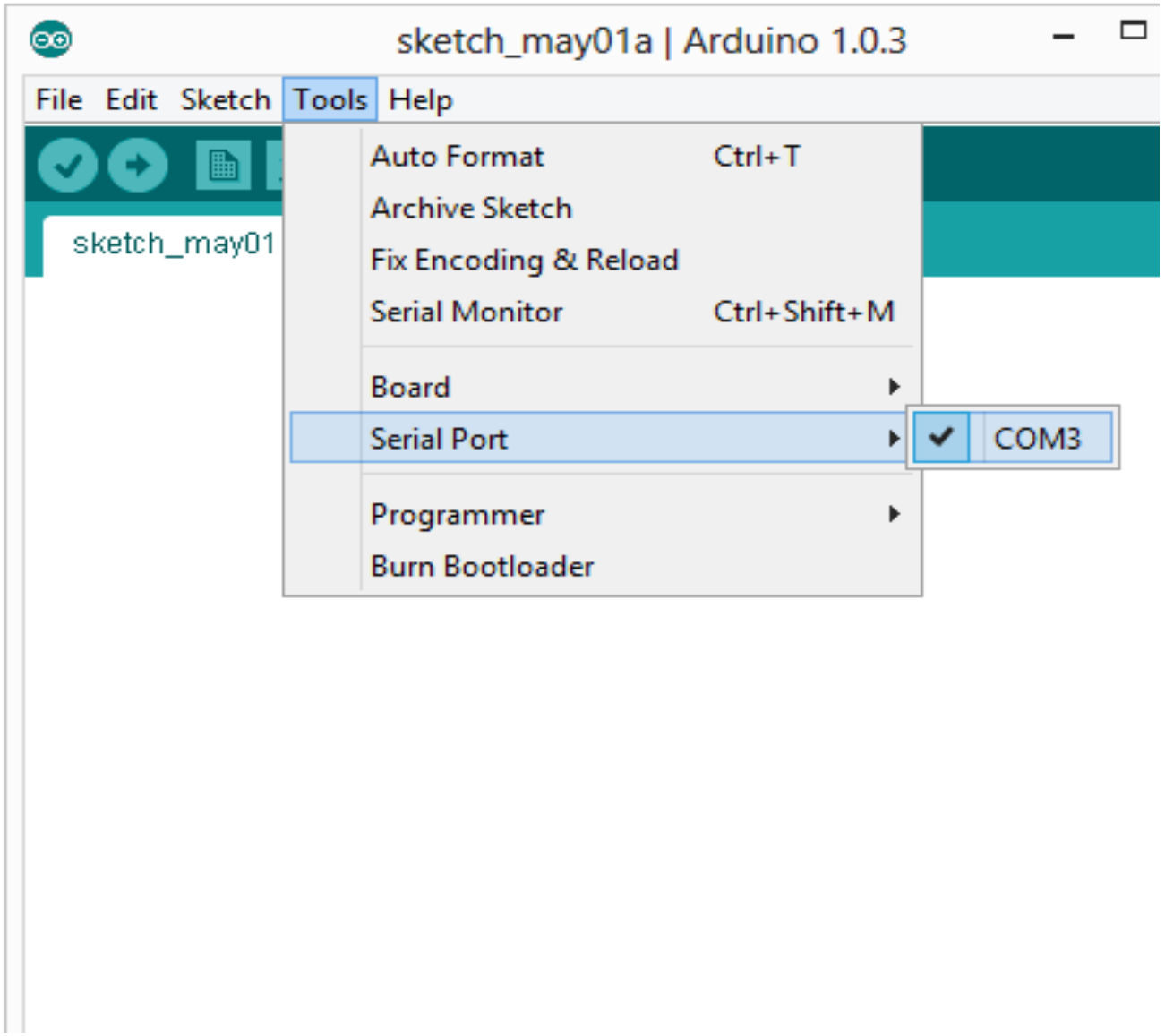
Arduino Integrated Development Environment (IDE)



Two required functions / methods /routines:

```
void setup()
{
        // runs once
}

void loop()
{
        // repeats
}
```

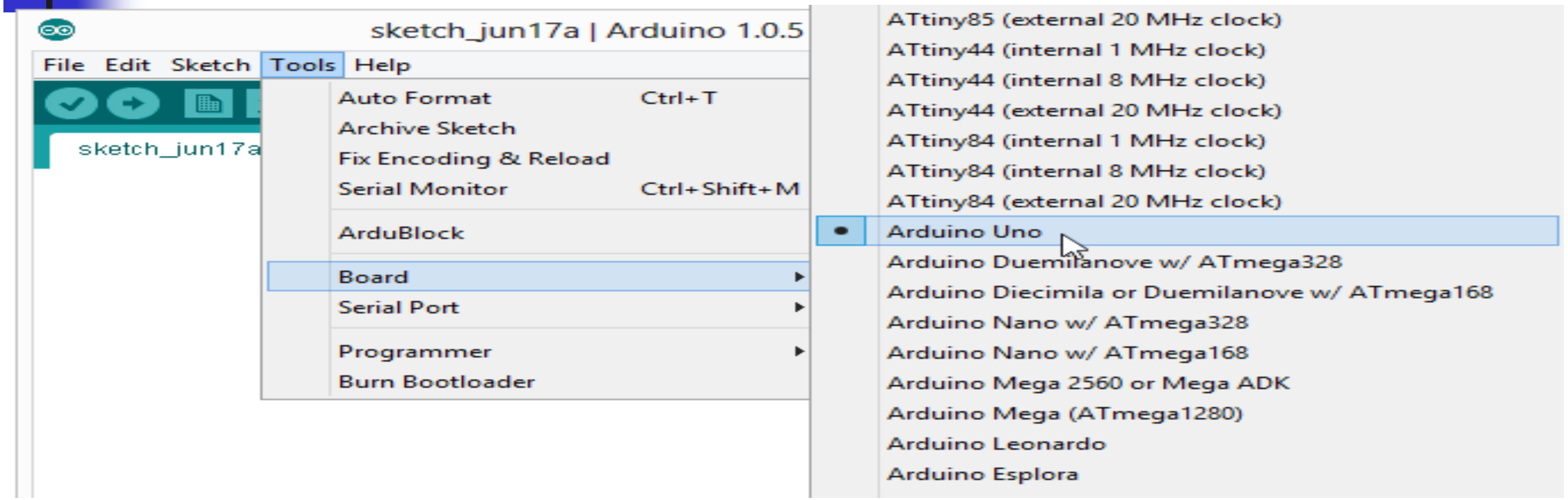
Settings: Tools → Serial Port



- Your computer communicates to the Arduino microcontroller via a serial port → through a USB-Serial adapter.

- Check to make sure that the drivers are properly installed.

Settings: Tools → Board



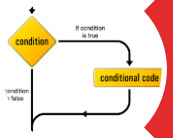
- Next, double-check that the proper board is selected under the Tools →Board menu.

BIG 6 CONCEPTS

digitalWrite()

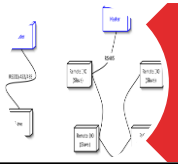
analogWrite()

digitalRead()



if() statements / Boolean

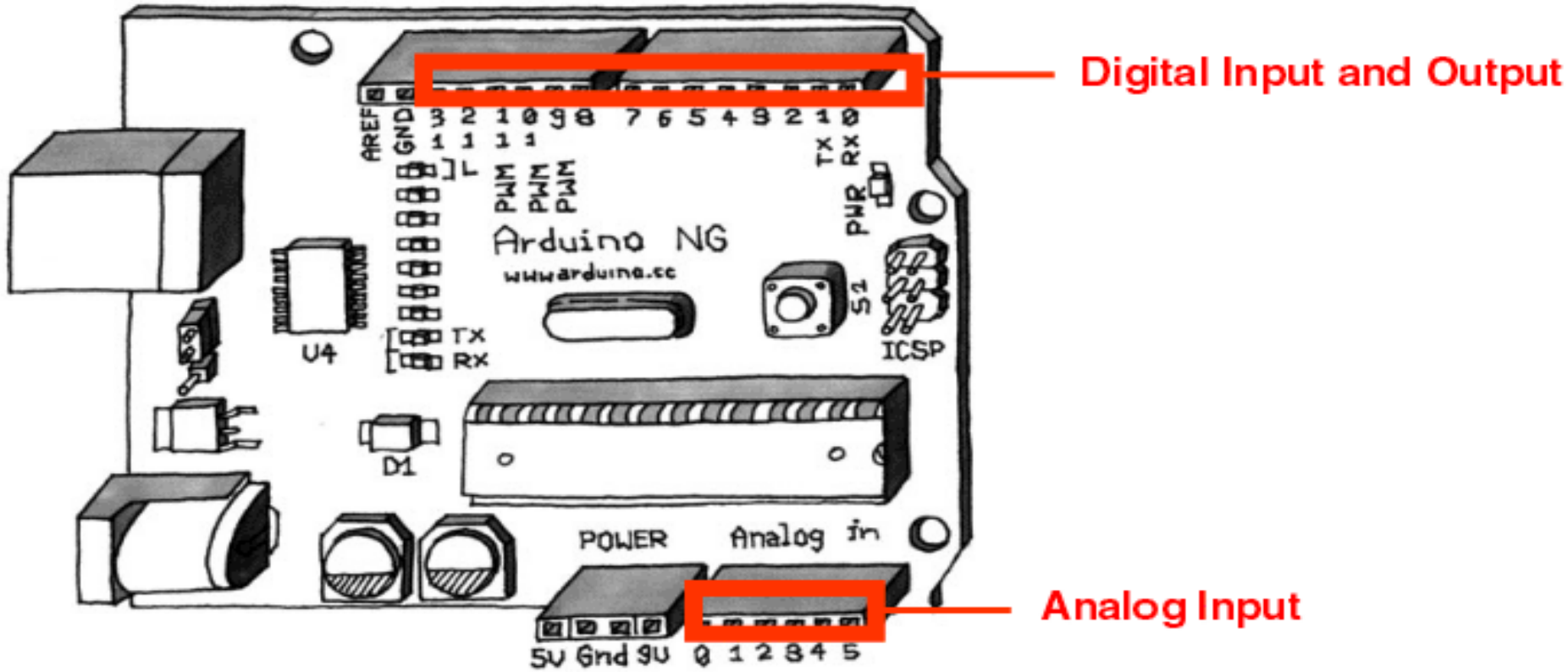
analogRead()



Serial communication

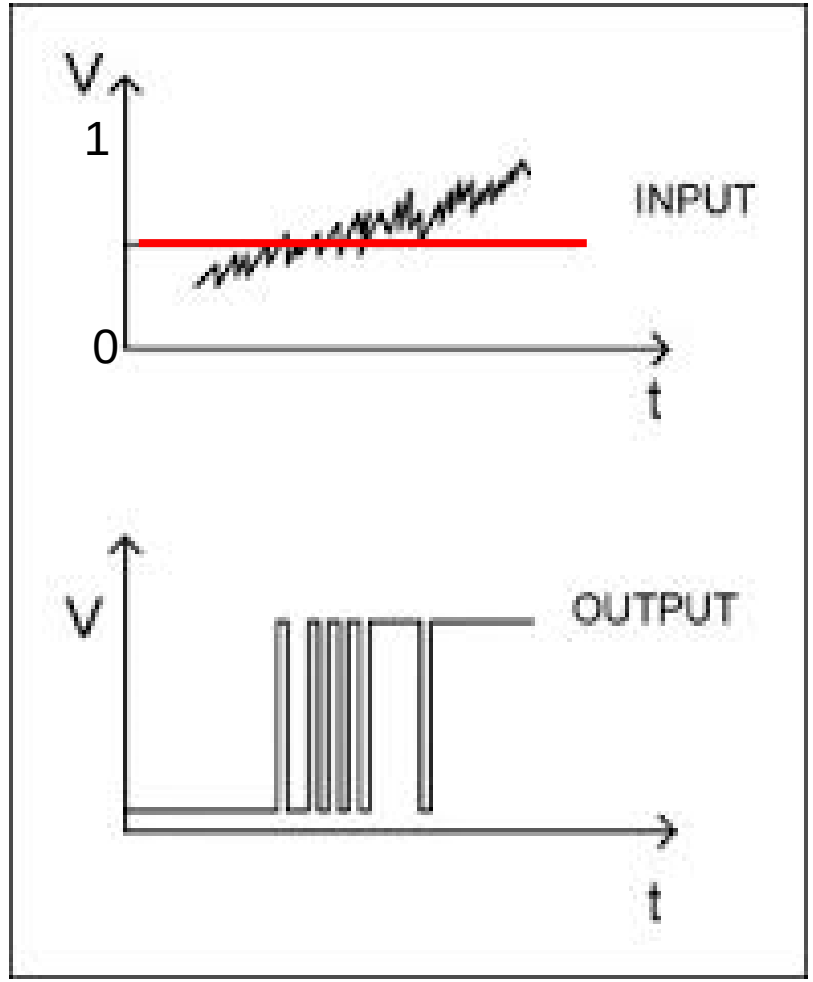
IO Pins

Two states (binary signal) vs. multiple states (continuous signal)



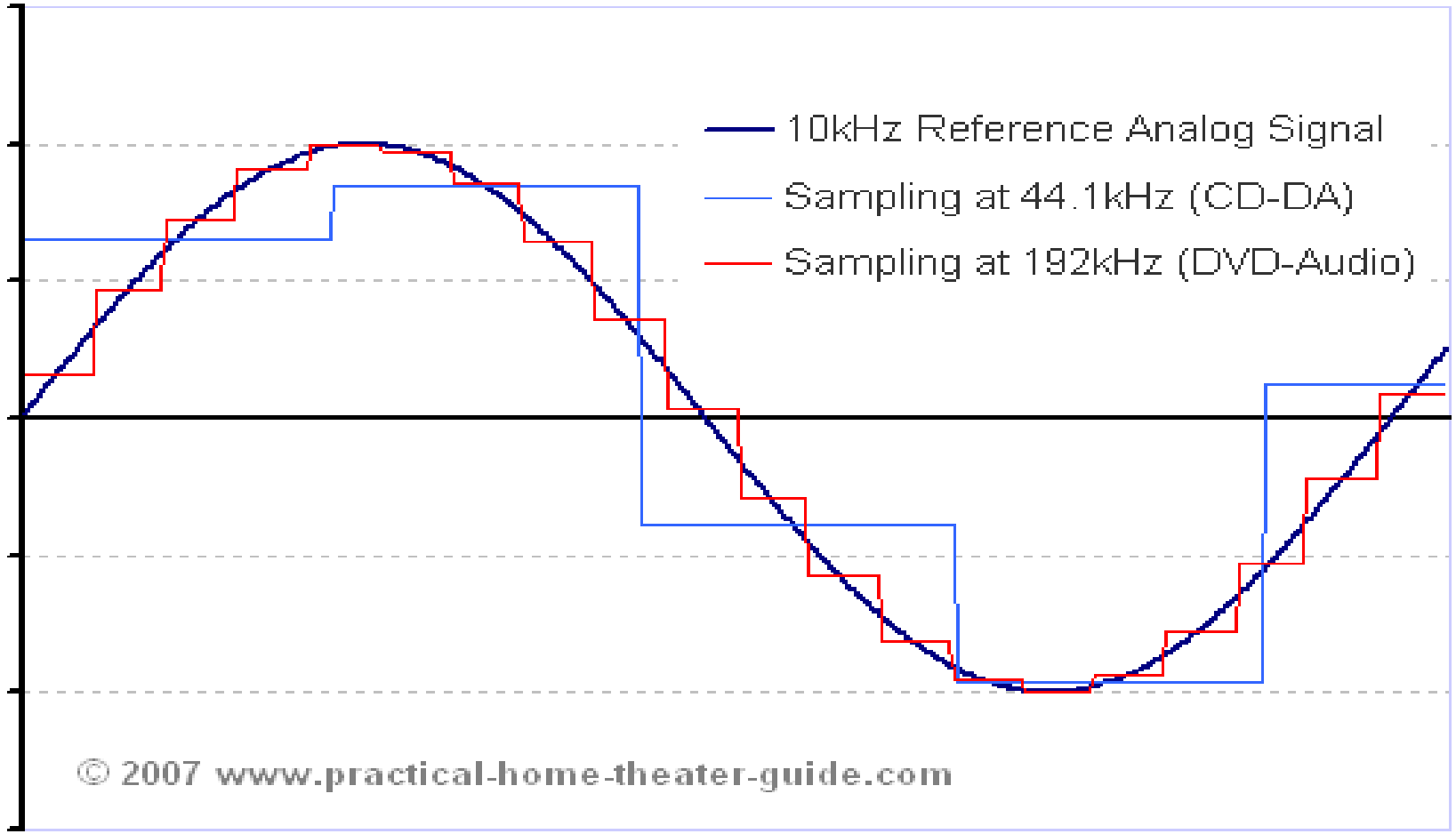
Topic 2: Digital Input/Output

- Digital IO is binary valued—it's either *on* or *off*, 1 or 0
- Internally, all microprocessors are digital, **why**?



Topic 3: Analog Input

- Think about music stored on a CD---an analog signal captured on digital media
  - Sample rate
  - Word length



Arduino Analog Input

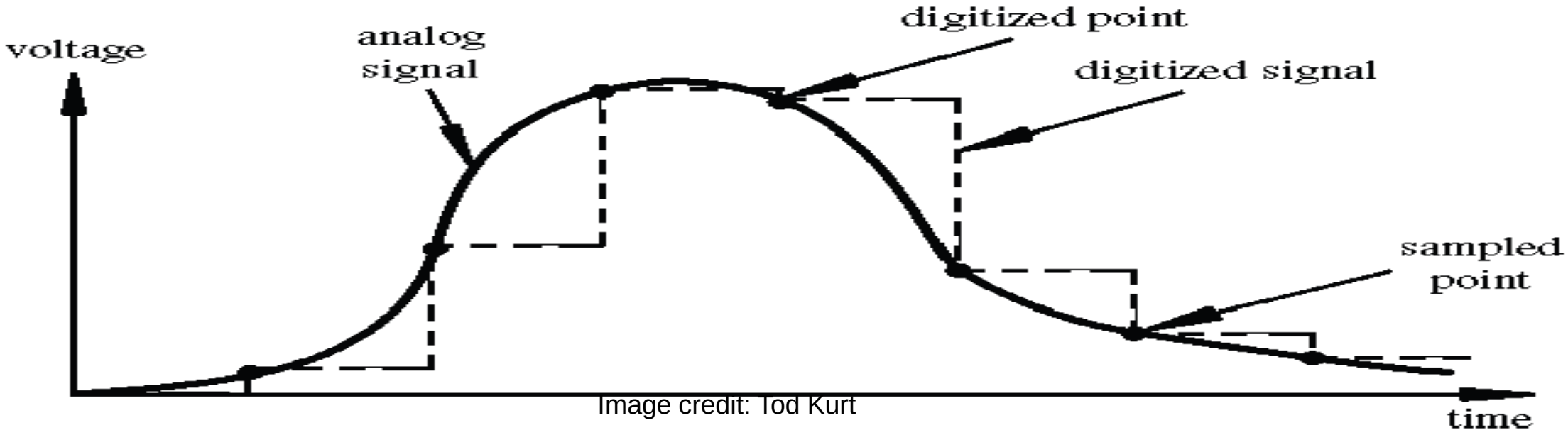
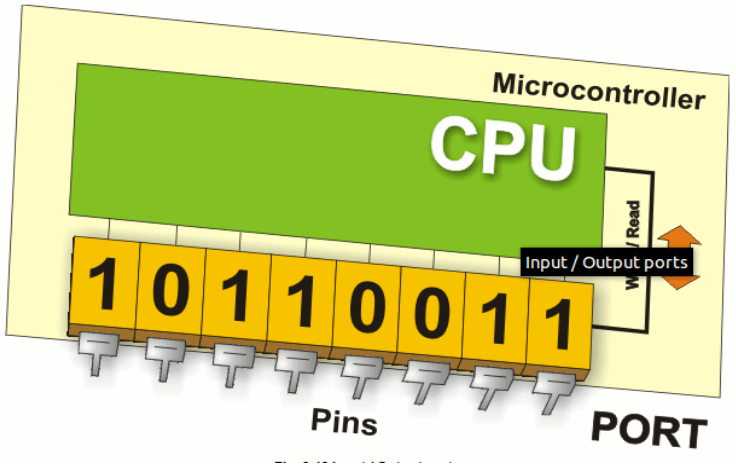


Image credit: Tod Kurt

- *Resolution*: the number of different voltage levels (i.e., *states*) used to discretize an input signal
- Resolution values range from 256 states (8 bits) to 4,294,967,296 states (32 bits)
- The Arduino uses 1024 states (10 bits)
- Smallest measurable voltage change is 5V/1024 or 4.8 mV
- Maximum sample rate is 10,000 times a second

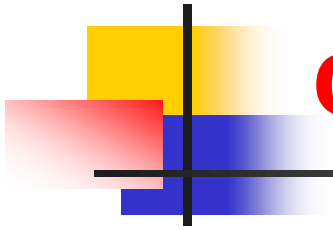
Arduino Digital I/0



- `pinMode(`*`pin, mode`*`)`
  - Sets pin to either `INPUT` or `OUTPUT`
- `digitalRead(`*`pin`*`)`
  - Reads `HIGH` or `LOW` from a pin
- `digitalWrite(`*`pin, value`*`)`
  - Writes `HIGH` or `LOW` to a pin
- Electronic stuff
  - Output pins can provide 40 mA of current
  - Writing `HIGH` to an input pin installs a 20KΩ pullup

Let's get to coding…

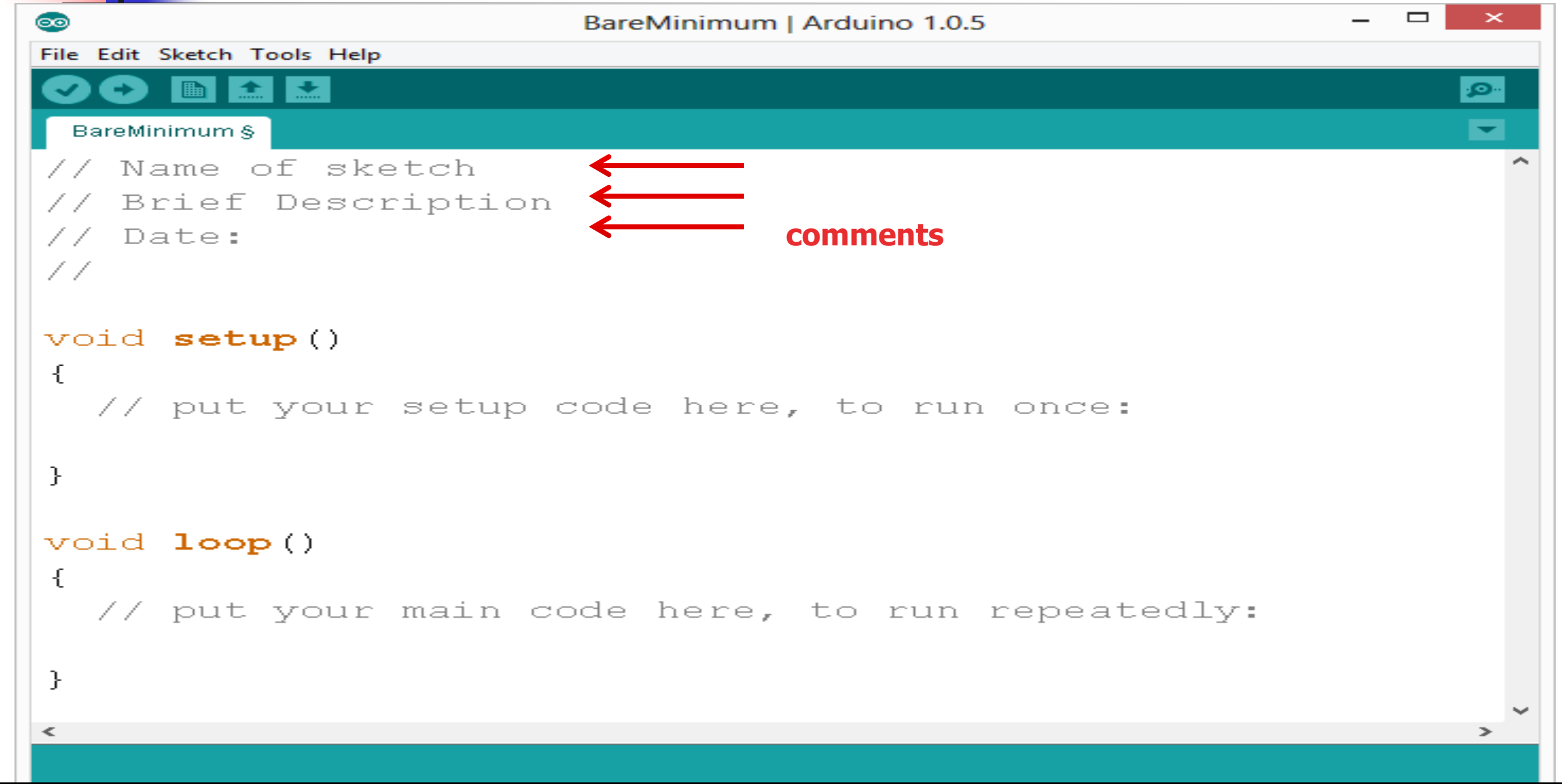
- Project #1 – Blink
  - "Hello World" of Physical Computing
- *Psuedo-code – how should this work?*



Comments, Comments, Comments

- Comments are for you – the programmer and your friends…or anyone else human that might read your code.

```
// this is for single line comments
// it's good to put a description at the top and
before anything 'tricky'
/* this is for multi-line comments
   Like this…
   And this….
*/
```


BareMinimum | Arduino 1.0.5
File Edit Sketch Tools Help
BareMinimum §
// Name of sketch
// Brief Description
// Date:
//
void setup()
{
// put your setup code here, to run once:
}
void loop()
{
// put your main code here, to run repeatedly:
}
comments

Three commands to know…

- **pinMode**(pin, INPUT/OUTPUT);
- ex: **pinMode**(13, OUTPUT);

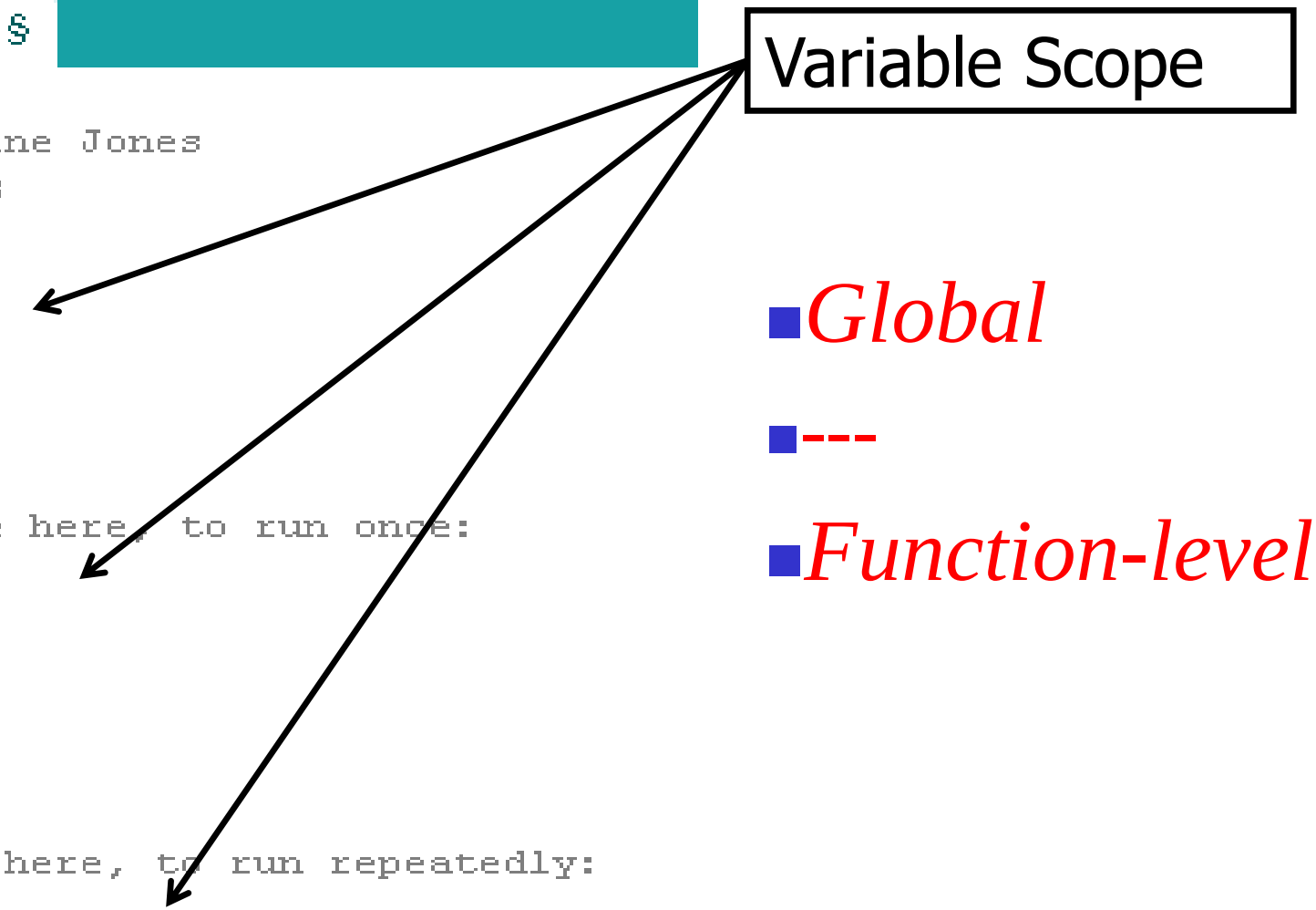
- **digitalWrite**(pin, HIGH/LOW);
- ex: digitalWrite(13, HIGH);
- **delay**(time_ms);
- ex: delay(2500); // delay of 2.5 sec.

- **// NOTE: -> commands are CASE-sensitive**

Programming Concepts: Variables

ProtosnapProMiniExample2 §

```
// Comments go here
// Written by:  Joesephine Jones
// Date:  April 12, 2013

int sensorValue;
int ledPin;

void setup()
{
  // put your setup code here, to run once:
  int setupVariable;

}

void loop()
{
  // put your main code here, to run repeatedly:
  int loopScopeVariable
}
```



- *Global*
- ---
- *Function-level*

Scope of Variables

- Variable scope determines where the variable can be used in the sketch. There are two variable scopes
  - Local Variables
    - Can only be used inside a function, and only appear inside that function block.
    - You won't see a local variable from the setup function in the loop function
  - Global Variables (Static Variables)
    - Can be used in ALL functions
    - Common to see them at the start of the sketch before the setup function

Declaring and Instantiating Variables

Data Type	Size (Bytes)	Value Range
boolean	1	Logic true or false
char	1	–128 to +127
byte	1	0 to 255
int	2	–32,768 to 32,767
word	2	0 to 65,535
long	4	–2,147,483,648 to 2,147,483,647
float	4	–3.4028235E+38 to 3.4028235E+38
double	4	–3.4028235E+38 to 3.4028235E+38

Declaring and Instantiating Variables

- **Declaring a Variable**
  dataType variableName;
- Example:
  int year;
- **Instantiating a Variable**
  Add equals sign
- Example:
  int year;
  year = 2017;

- **Declaring and Instantiating Simultaneously**
- Example:
  int year = 2017;
- **For Constants**
  Add ‘const’ before datatype
- Example:
  const float pi = 3.14;

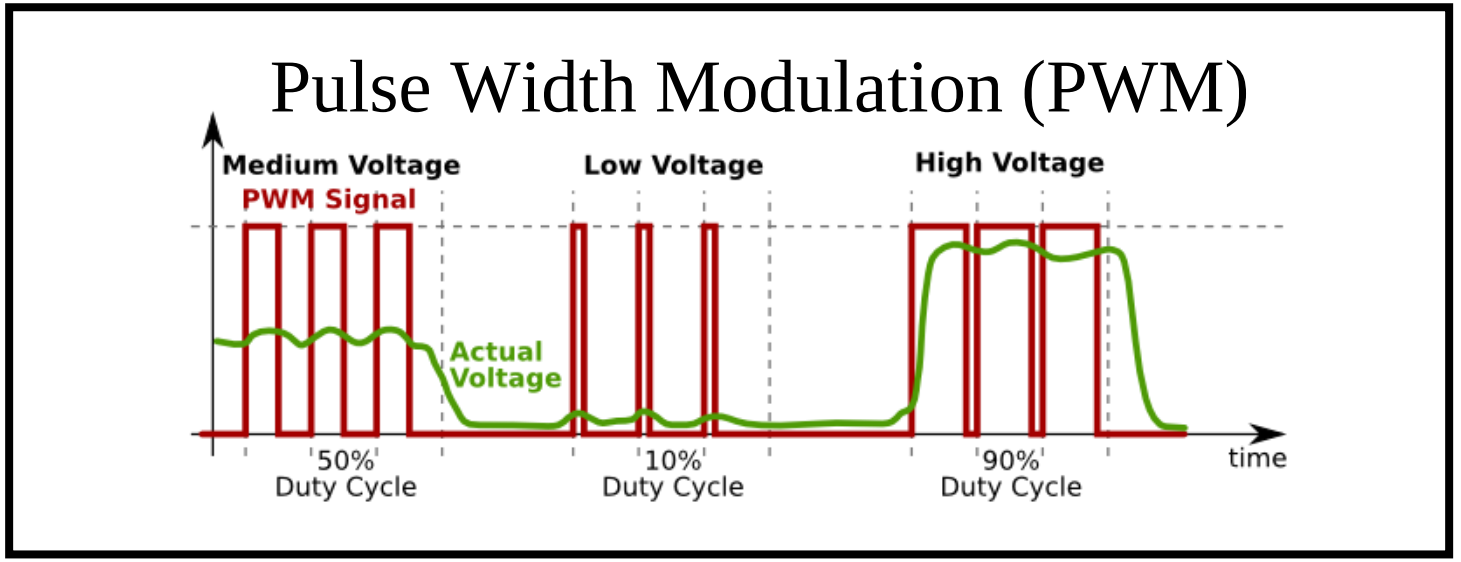
Math Operators

- Standard Operators are built-in for use and can be used with variables. Two examples below:
- int x;
  float y;
  int z;
  x = 5;
  y = 2.7;
  z = x+y;
  What is z equal to above?
- int x = 5;
  float y = 2.7;
  float z = x+y;
  What is z equal to above?

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement
!	Logical NOT
&&	Logical AND
||	Logical OR
&	Bitwise AND
|	Bitwise OR
<<	Left shift
>>	Right shift

Fading in and Fading Out (Analog or Digital?)

- A few pins on the Arduino allow for us to modify the output to mimic an analog signal.
- This is done by a technique called:
- Pulse Width Modulation (PWM)
- To create an analog signal, the microcontroller uses a technique called PWM. By varying the duty cycle, we can mimic an "average" analog voltage

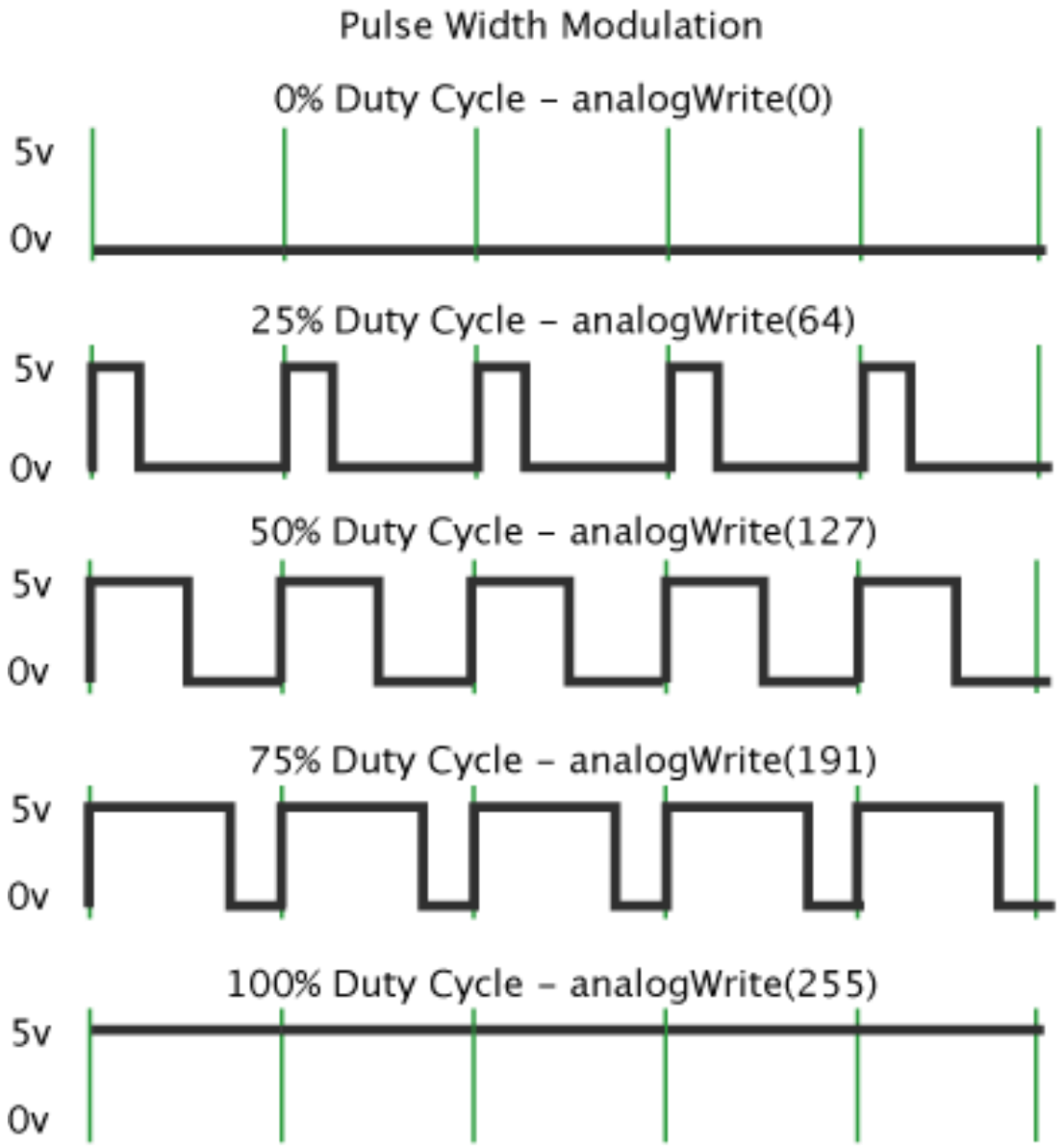


Project #2 – Fading: Introducing a new command…

- **`analogWrite`**`(pin, val);`

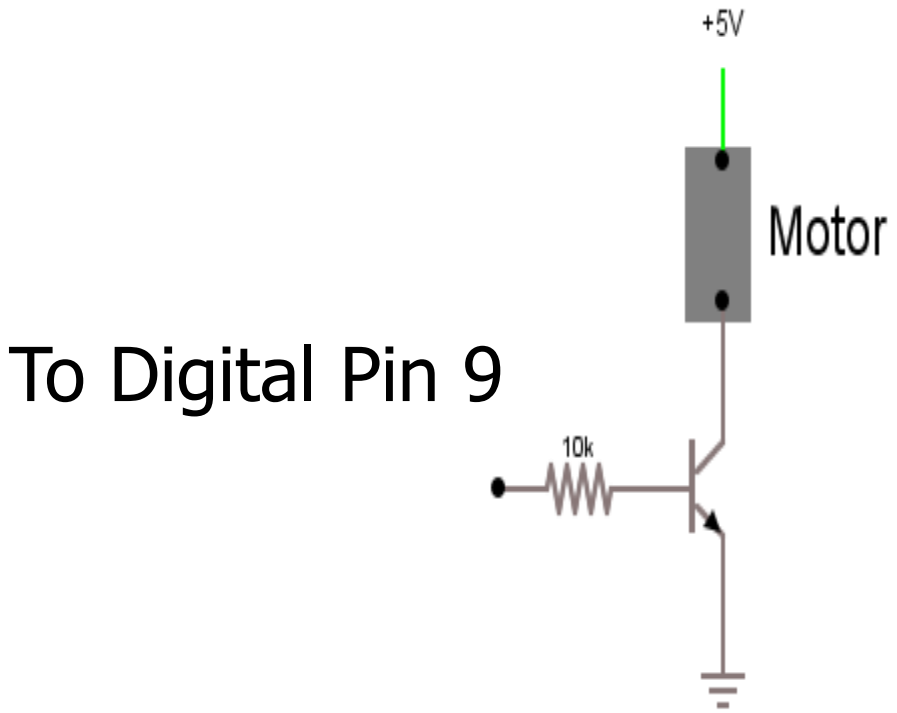
- **`pin`** – refers to the OUTPUT pin (limited to pins 3, 5, 6, 9, 10, 11.) – denoted by a ~ symbol

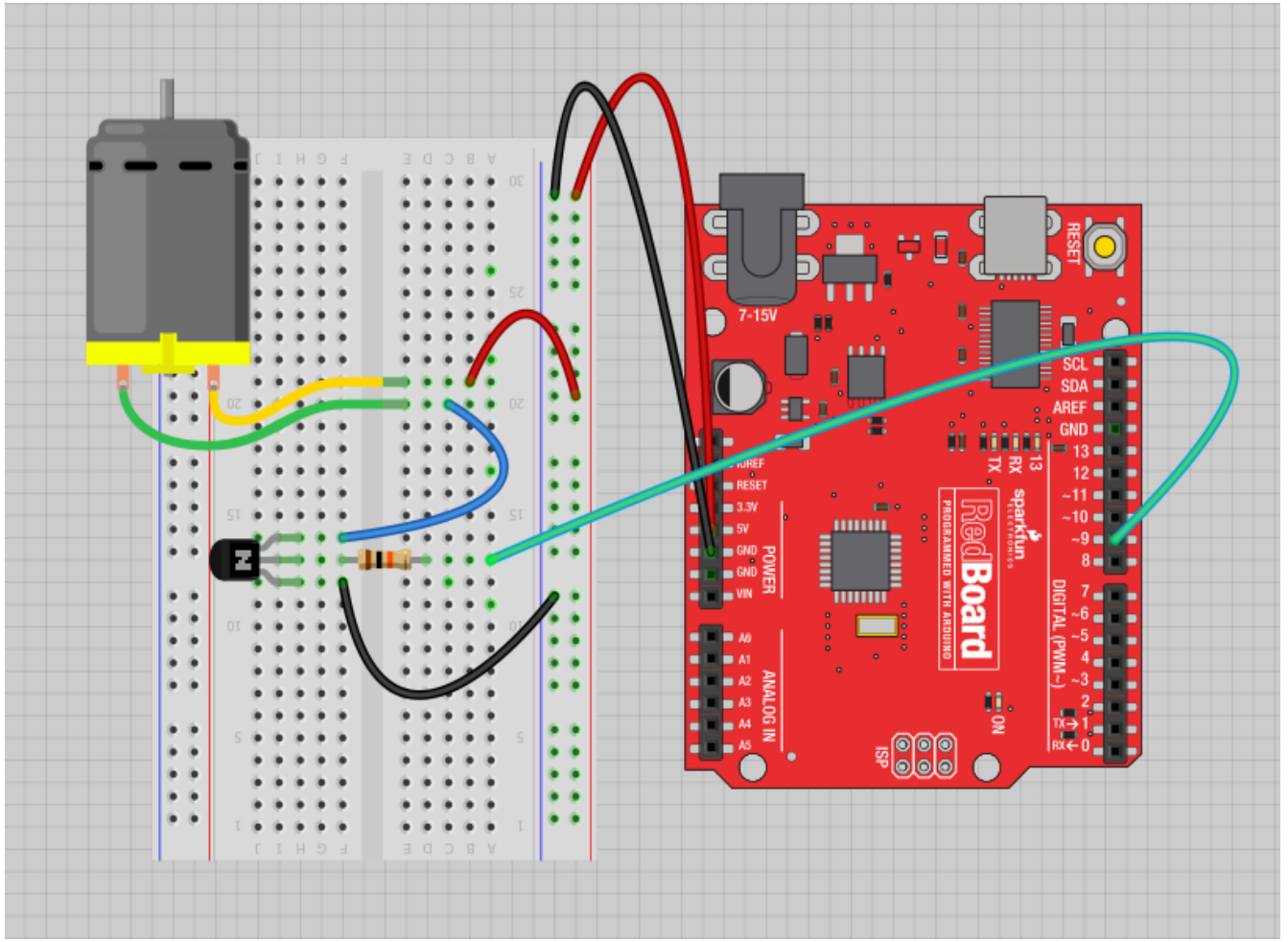
- **`val`** – 8 bit value (0 – 255).
- 0 => 0V | 255 => 5V



Driving Motors or other High Current Loads

- **NPN Transistor (Common Emitter "Amplifier" Circuit)**





Input

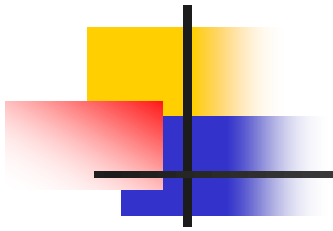
- Input is any signal entering an electrical system
- Both digital and analog sensors are forms of input
- Input can also take many other forms: Keyboards, a mouse, infrared sensors, biometric sensors, or just plain voltage from a circuit

Digital Input

- **Connect digital input to your Arduino using Pins # 0 – 13 (Although pins # 0 & 1 are also used for programming)**
- **Digital Input needs a pinMode command:**
  - **pinMode (pinNumber, INPUT);**
- *Make sure to use ALL CAPS for **INPUT***
- To get a digital reading:
  - **int buttonState = digitalRead (pinNumber);**
- *Digital Input values are only HIGH (On) or LOW (Off)*

Digital Sensors

- Digital sensors are more straight forward than Analog
- No matter what the sensor there are only two settings: On and Off
- Signal is always either HIGH (On) or LOW (Off)
- Voltage signal for HIGH will be a little less than 5V on your Uno
- Voltage signal for LOW will be 0V on most systems



We set it equal to the function digitalRead(pushButton)

The function digitalRead() will return the value 1 or 0, depending on whether the button is being pressed or not being pressed.

We declare a variable as an integer.

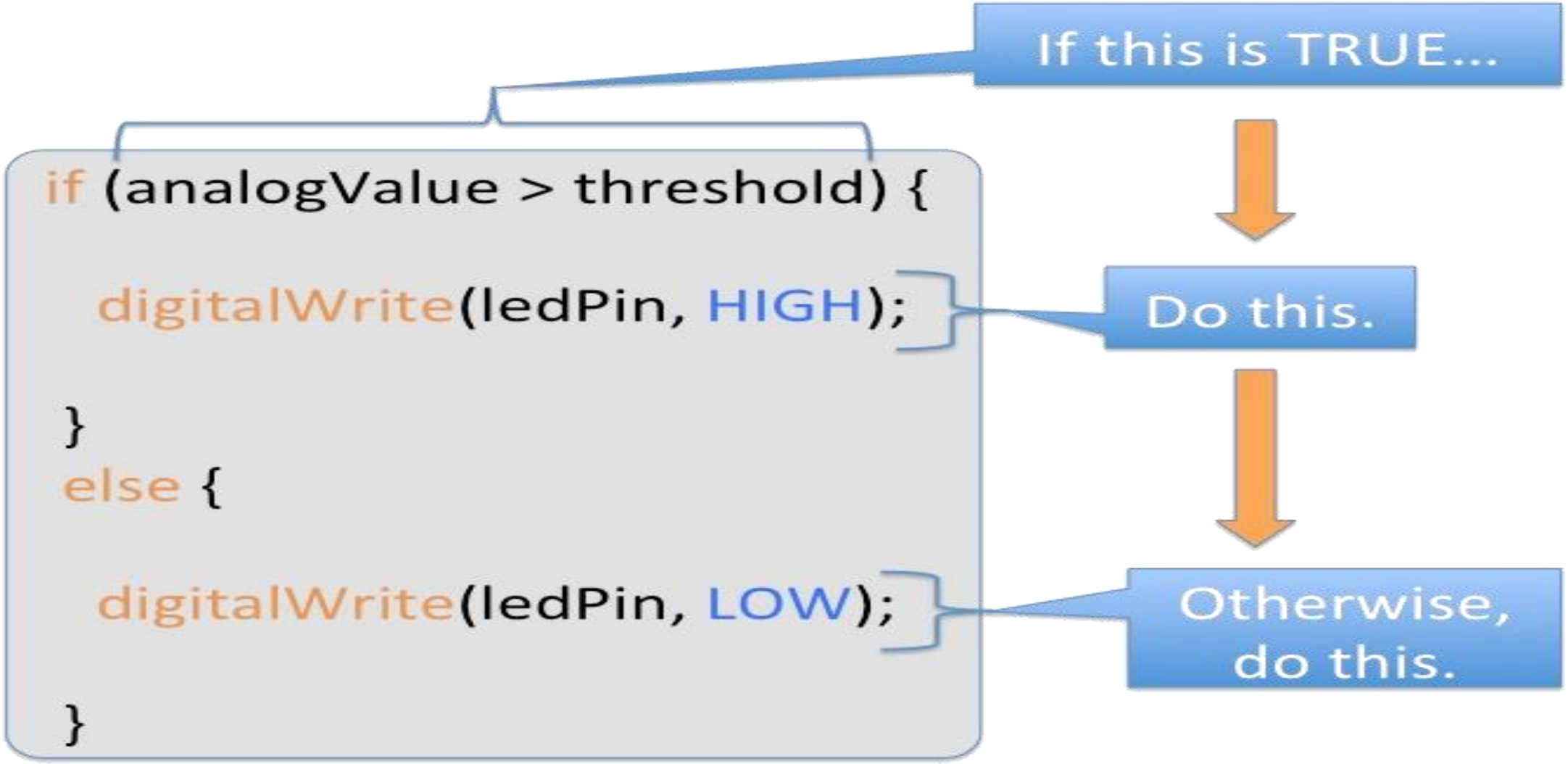
```
int buttonState = digitalRead(pushButton);
```

We name it buttonState

The value 1 or 0 will be saved in the variable buttonState.

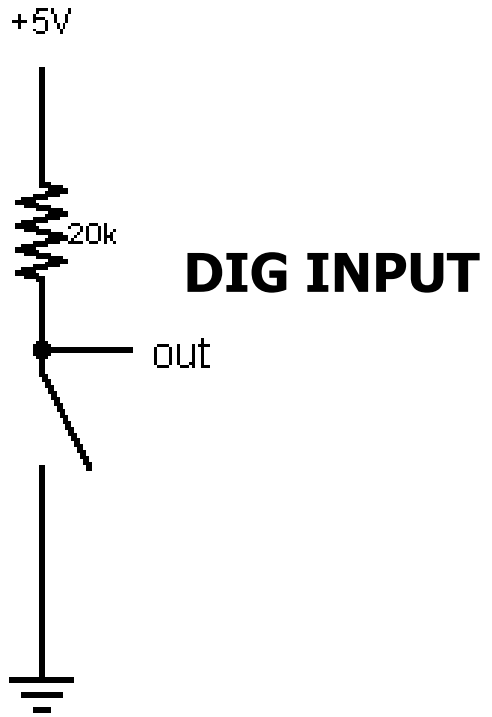
Recall that the pushButton variable stores the number 2

Programming: Conditional Statements `if()`



Programming: Conditional Statements `if()`

```
void loop()
{
   int buttonState = digitalRead(5);
   if(buttonState == LOW)
   {    // do something
   }
   else
   {   // do something else
   }
}
```

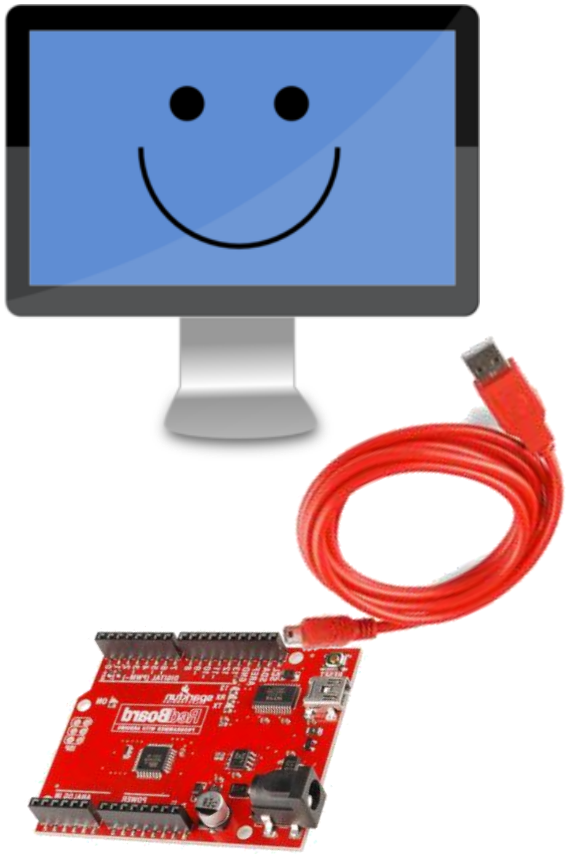


Boolean Operators

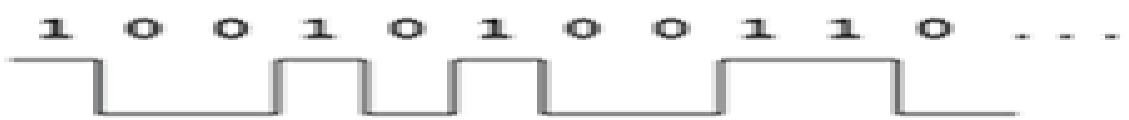
<Boolean>	Description
( ) == ( )	is equal?
( ) != ( )	is not equal?
( ) > ( )	greater than
( ) >= ( )	greater than or equal
( ) < ( )	less than
( ) <= ( )	less than or equal

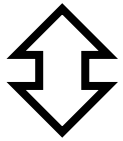
Using Serial Communication

Method used to transfer data between two devices.



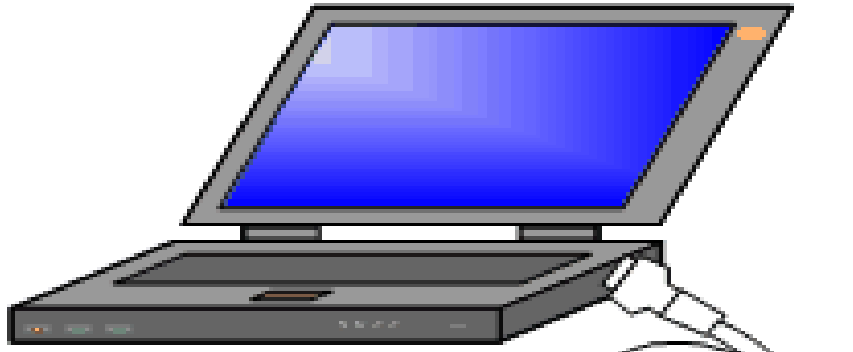
Data passes between the computer and Arduino through the USB cable. Data is transmitted as zeros (‘0’) and ones (‘1’) sequentially.



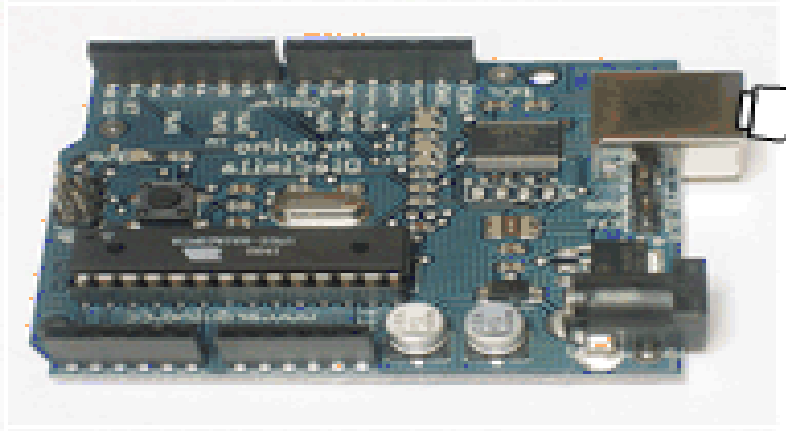


Arduino dedicates Digital I/O pin # 0 to receiving and Digital I/O pin #1 to transmit.

Topic 4: Serial Communication



Cable



1 0 0 1 0 1 0 0 1 1 0 . . .

Information passes between the computer and Arduino through the USB cable. Information is transmitted as zeros (‘0’) and ones (‘1’)… also known as **bits**!

Serial Communications

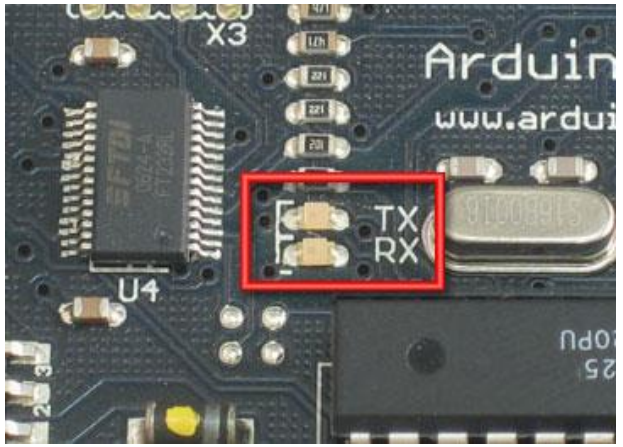
- "Serial" because data is broken down into bits, each sent one after the other down a single wire.
- The single ASCII character 'B' is sent as:

```
'B' =   0 1 0 0 0 0 1 0
    =   L H L L L L H L
    =
```

- Toggle a pin to send data, just like blinking an LED
- You could implement sending serial data with `digitalWrite()` and `delay()`
- A single data wire needed to send data. One other to receive.

Serial Communication

- ***Compiling*** turns your program into binary data (ones and zeros)
- ***Uploading sends the bits through USB cable to the Arduino***
- ***The two LEDs near the USB connector blink when data is transmitted***
  - ***RX blinks when the Arduino is receiving data***
  - ***TX blinks when the Arduino is transmitting data***

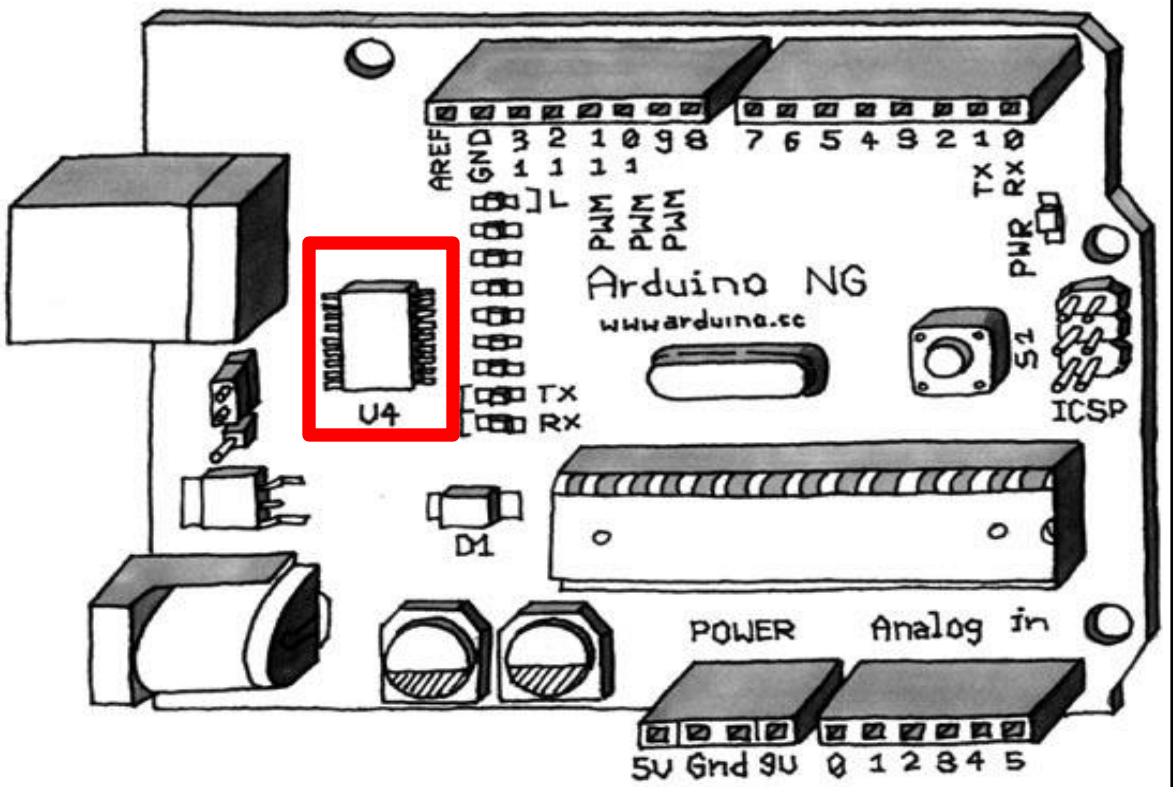


Some Commands

- Serial.begin()
  - e.g., Serial.begin(9600)
- Serial.print() or Serial.println()
  - e.g., Serial.print(value)
- Serial.read()
- Serial.available()
- Serial.write()
- Serial.parseInt()

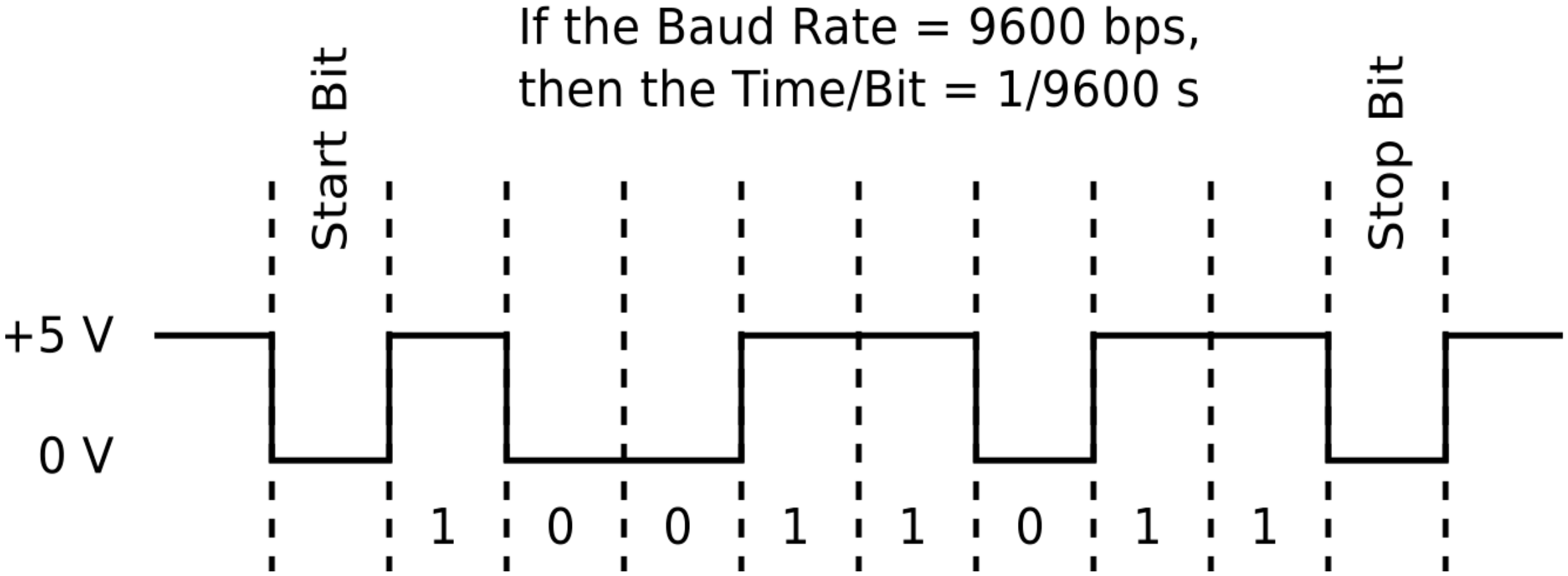
Serial-to-USB chip---what does it do?

The LilyPad and Fio Arduino require an external USB to TTY connector, such as an FTDI “cable”.
In the Arduino Leonardo a single microcontroller runs the Arduino programs and handles the USB connection.

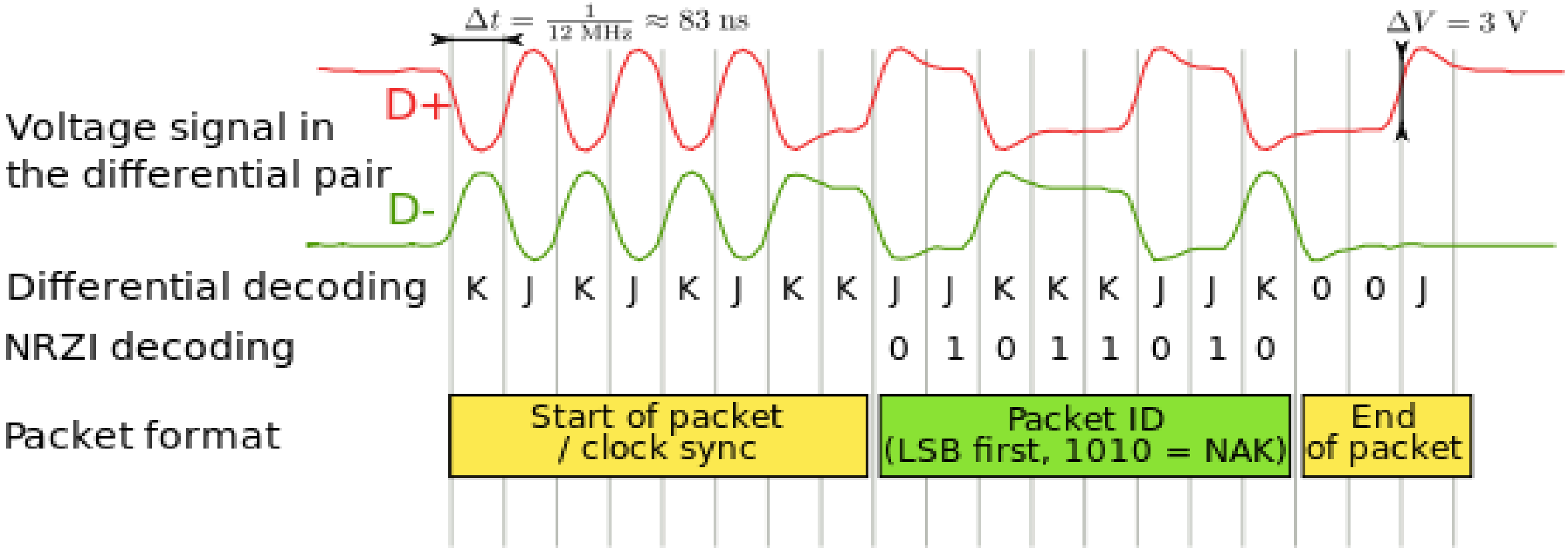


Two different communication protocols

Serial (TTL):



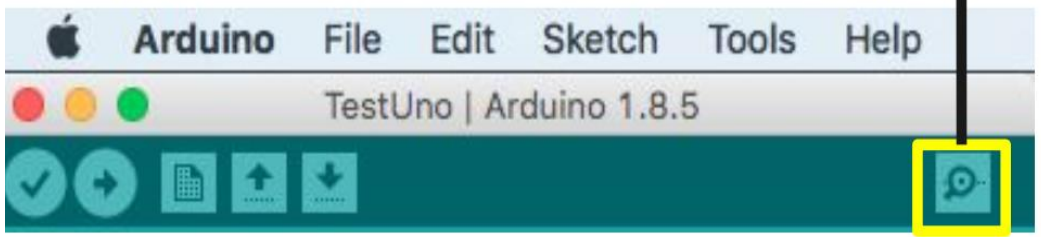
USB Protocol



- Much more complicated

Using the Serial Monitor

- The serial monitor allows you to see the output from the program




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

void loop() {
  int x = 5;
  float y = 2.56;
  float sum = x+y;
  Serial.println(sum);
  delay(1000);
}
```

1. insert Serial.begin(baudRate); to initialize the serial port

   baudRate = Speed of Connection (higher is faster; must match workstation baud). // i want that baud

   default baud rate is 9600;

2. Printing to the serial monitor:

   Serial.print(sum) // does not start a new line
   Serial.println(sum) //starts a new line

3. Working with time
   - delay(x): pauses the sketch for x milliseconds
   - delayMicroseconds(x): pauses the sketch for x microseconds
   - micros(): Returns the number of microseconds since the arduino was reset
   - millis(): returns the number of milliseconds since the Arduino was reset

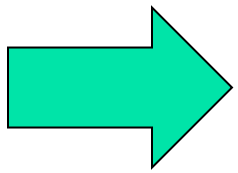
Output on the Serial Monitor

- **Note: You cannot concatenate the Serial output**
  Ex.this won't work
  Serial.println("The sum is" + sum);

TestUno | Arduino 1.8.5

TestUno

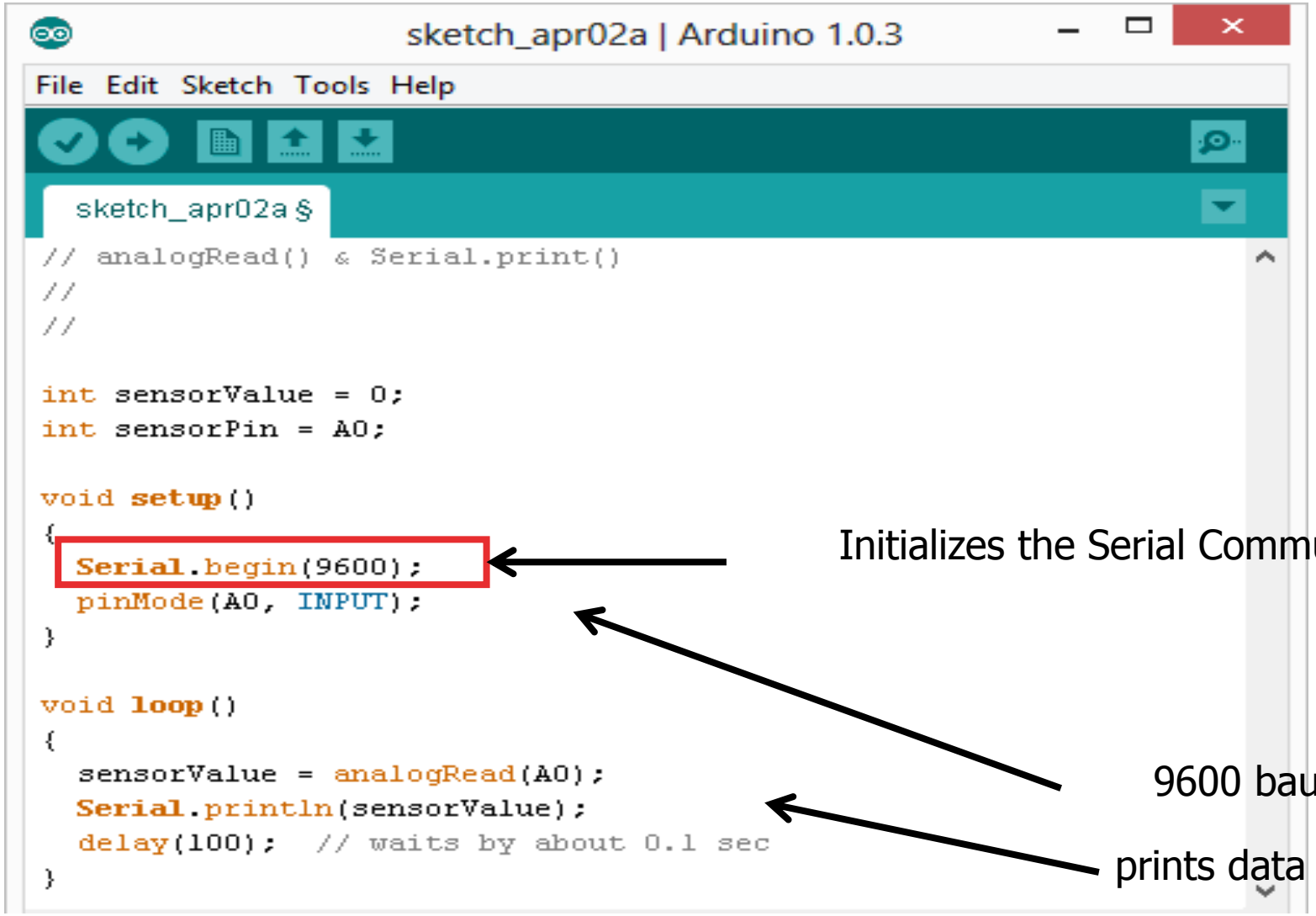
```
void setup() {
  //This is a comment
  Serial.begin(9600); //Sets baud Rate
}

void loop() {
  int x = 5;
  float y = 2.56;
  float sum = x+y;
  Serial.println(sum); //prints a new line
  delay(1000); //waits 1 second before re-iterating the loop
}
```



/dev/cu.usbmodemFA1

```
7.56
7.56
7.56
7.56
7.56
7.56
7.56
7.56
7.56
7.56
```

Serial Monitor & analogRead()



Initializes the Serial Communication

9600 baud data rate

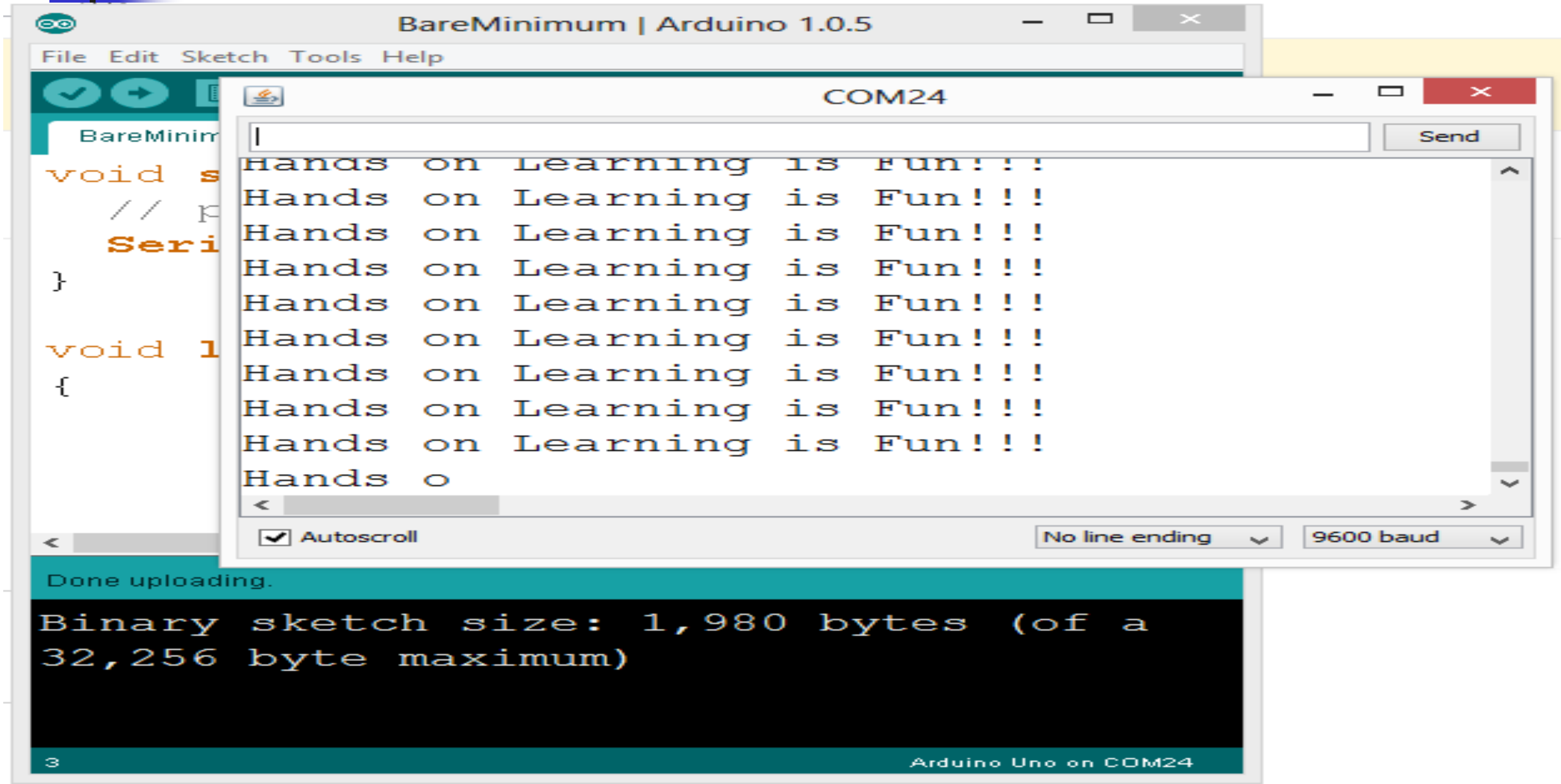
prints data to serial bus

Serial Monitor & analogRead()

Opens up a Serial Terminal Window

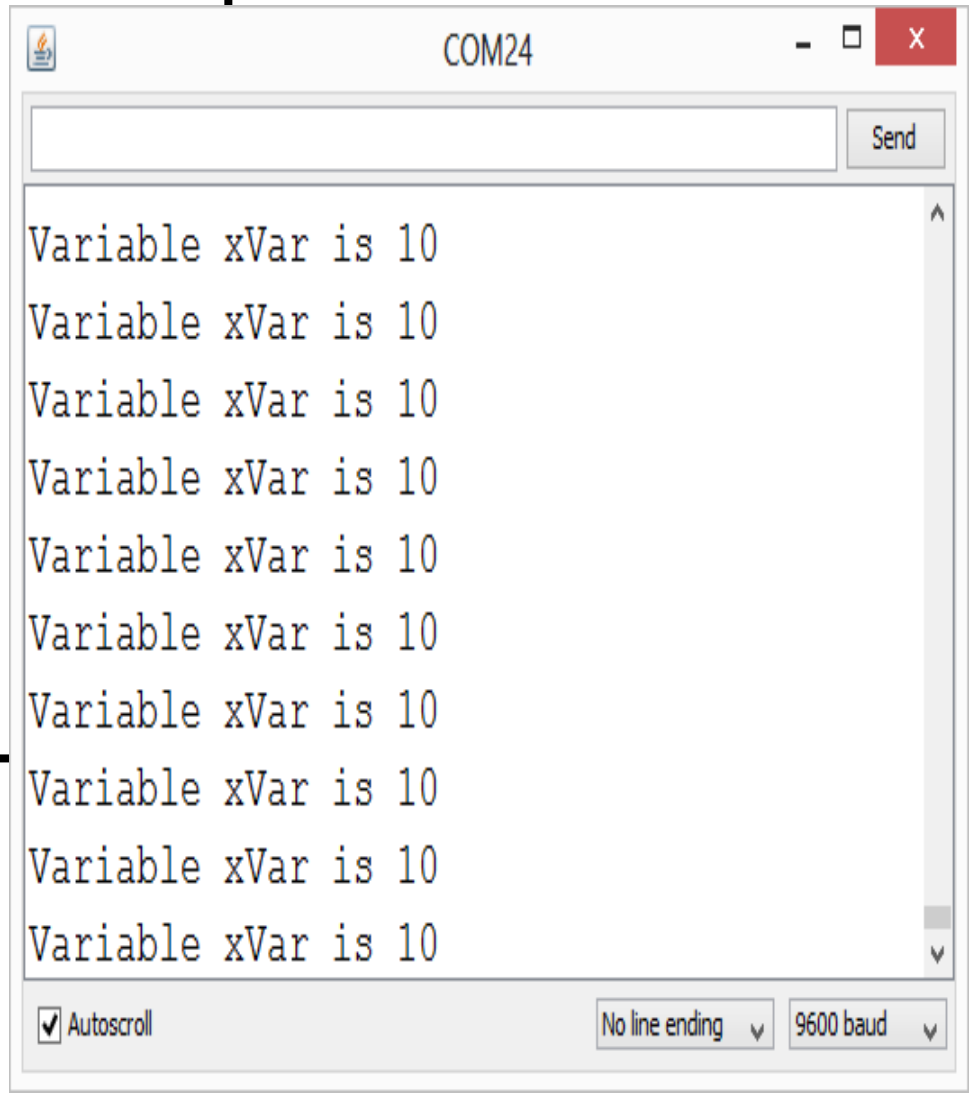
Additional Serial Communication Sending a Message

```
void loop ( )
{
  Serial.print(“Hands on “) ;
  Serial.print(“Learning ”) ;
 Serial.println(“is Fun!!!”) ;
}
```


BareMinimum | Arduino 1.0.5
File Edit Sketch Tools Help
BareMinim
void s
// p
Seri
}
void l
{
COM24
Send
Hands on Learning is Fun!!!
Hands on Learning is Fun!!!
Hands on Learning is Fun!!!
Hands on Learning is Fun!!!
Hands on Learning is Fun!!!
Hands on Learning is Fun!!!
Hands on Learning is Fun!!!
Hands on Learning is Fun!!!
Hands on Learning is Fun!!!
Hands o
Autoscroll
No line ending
9600 baud
Done uploading.
Binary sketch size: 1,980 bytes (of a
32,256 byte maximum)
3
Arduino Uno on COM24

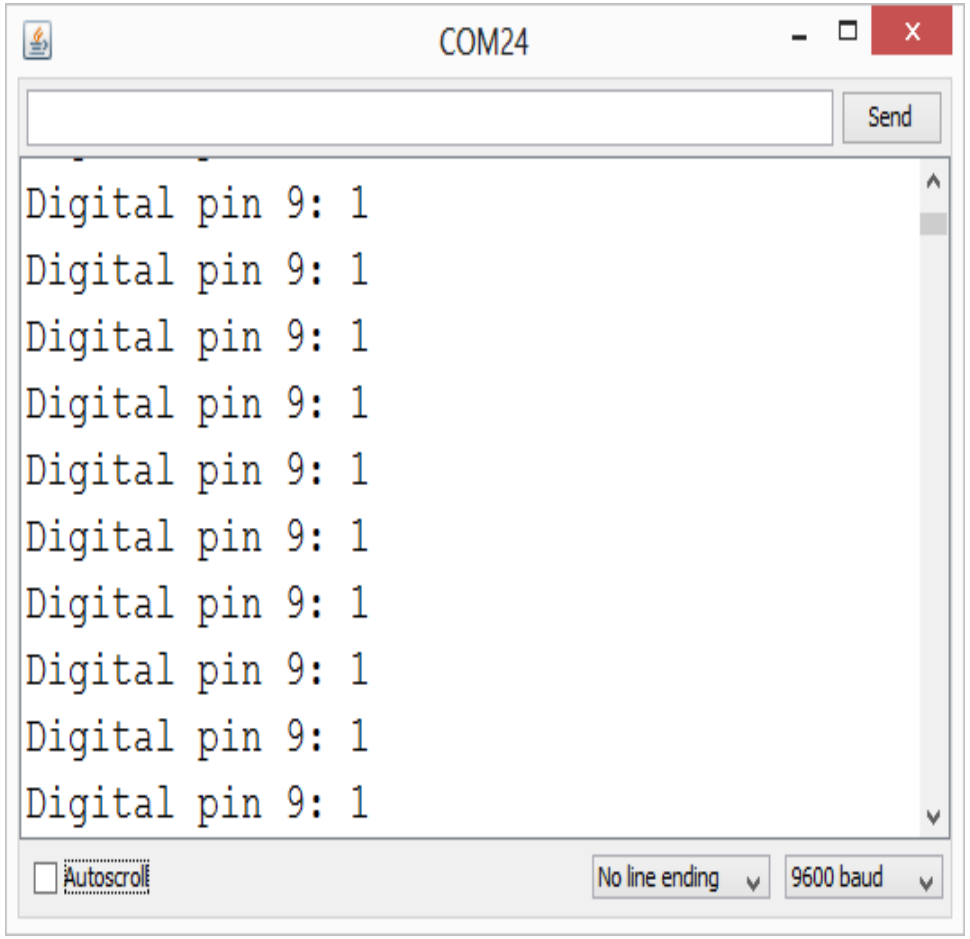
Serial Communication: Serial Debugging

```
void loop()
{
  int xVar = 10;
  Serial.print ( "Variable xVar is " ) ;
  Serial.println ( xVar ) ;
}
```



Serial Communication: Serial Troubleshooting

```
void loop ( )
{
  Serial.print ("Digital pin 9: ");
  Serial.println (digitalRead(9));
}
```



Advanced Math Functions

Function	Description
`abs(x)`	Returns the absolute value of *x*
`constrain(x, a, b)`	Returns *x* if *x* is between *a* and *b* (otherwise returns *a* if *x* is lower than *a*, or *b* if *x* is higher than *b*)
`cos(x)`	Returns the cosine of *x* (specified in radians)
`map(x, fromLow, fromHigh, toLow, toHigh)`	Remaps the value *x* from the range *fromLow* to *fromHigh* to the range *toLow* to *toHigh*
`max(x, y)`	Returns the larger value of *x* or *y*
`min(x, y)`	Returns the smaller value of *x* or *y*
`pow(x, y)`	Returns the value of *x* raised to the power of *y*
`sin(x)`	Returns the sine of *x* (specified in radians)
`sqrt(x)`	Returns the square root of *x*
`tan(x)`	Returns the tangent of *x* (specified in radians)

Example:

- int Temperature = -7;
  int value = abs(temperature);
  Serial.println(value);
- What does it print above?
- Note: The map() and contrain() functions are mostly used with sensors. They allow you to keep values returned by the sensors within a specific range that your sketch can manage

Generating Random Numbers

- Two functions are available for working with random numbers. random() and randomSeed()
  - random(min, max) : returns a random number between min and max -1
  - random(max) : returns a random number between 0 and max -1
  - randomSeed(seed): Initializes the random number generator, causing it to restart at an arbitrary point in random sequence

```
//this variable will hold a random number generated by the random() functi
long randomNumber;

//Set up - this is where you get things "set-up". It will only run once
void setup() {

  //setup serial communications through the USB
  Serial.begin(9600);

  //Let's make it more random
  randomSeed(42);

}//close setup

void loop() {

  //generate a random number
  randomNumber = random(2,5);

  //display the random number on the serial monitor
  Serial.print("The Random Number is = ");
  Serial.println(randomNumber);

}
```

Control flows

if control

```
if (condition) {
        Statement 1;
        Statement 2;
        etc.
}
```

Example

```
if (temperature > 100) {
        Serial.println("wow"!)
        Serial.println("that's hot!")
        }
```

if/else control

```
if (condition) {
        Statement 1;
        Statement 2;
        etc.
}
else {
Statements;
}
```

Example

```
if (grade > 92) {
        myGrade = 'A';
  else
        myGrade = 'F';
```

if/else if control

```
if (condition)
        Statement;
else if (condition)
        Statement;
else if (condition)
        Statement;
else
        Statement;
```

Example

```
if (grade > 92) {
        myGrade = 'A';
  else if (grade  > 83)
        myGrade = 'B';
  else
        myGrade = 'C';
```

Numeric Comparisons

- **Compound Conditions**
- **Example 1**
  If ((a == 1) || (b == 1)){
  statements;}
- Example 2
  If ((a == 1) && (b == 2)) {
  statements;
  }
- **Negating a condition check**
  int a == 1;
  If (!(a == 1))
  Serial.println("The 'a' variable is not equal to 1");
  if(!(a ==2))
  Serial.println("The 'a' variable is not equal to 2");

Operator	Description
==	Equal
!=	Not equal
<>	Not equal
>	Greater than
>=	Greater than or equal
>	Less than
<=	Less than or equal

Using a switch statement

- Instead of doing a lot of if and else statement, it may be useful to do a switch
- Format:
- switch (var) {
  case 23:
  //do something when var equals 23
  break;
  case 64:
  //do something when var equals 64
  break;
  default:
  // if nothing else matches, do the default
  // default is optional
  break;
  }

- Example
- switch (grade) {
  case ‘A’:
  Serial.println(“you got higher than a 92”);
  break;
  case ‘B’:
  Serial.println(“You got higher than a 80”);
  break;
  default:
  Serial.println(“You have dishonored the family”);
  break;
  }

Loops

for

for (statement1; condition; statement 2){
Code statements
}

executes until condition is met

Example

```
int values[5] = {10, 20, 30, 40, 50};
for (int counter = 0; counter < 5; counter ++){
        Serial.print("one value in the array is " );
        Serial.println(values[counter]);
}
```

while

while (condition){
Code statements
}

executes until condition is met

Example

```
int i = 0;
while (i<3) {//hearts for days
Serial.println(" i is : " i);
i++;
}
```

do while

do {
Code statements
} while (condition);

executes at least once, and until condition is met

Example

```
int i = 0;
Serial.println("Hi");
do {
 Serial.println("my name is");
if(i==0) Serial.println("What");
if(i==1) Serial.println("Who");
 i++;
} while (i < 2);
Serial.println("slicka chika slim shady");
```

Loops Continued

- **Using Multiple Variables**
- You can initialize multiples variables in a for statement
- **Example**

```
int a,b;
For (a = 0, b = 0; a < 10; a++, b++){
Serial.print("One value is ");
Serial.println(a);
Serial.print(" and the other value is ");
Serial.println(b);
}
```

- **Nesting Loops**
- You can place loops inside of another loop. The trick to using inner loop sis that you must complete the inner loop before you complete the outer loop
- **Example**

```
int a,b;
for (a = 0; a < 10; a++){
for (b = 0; b < 10; b++){
Serial.print("one value is ");
Serial.print(a);
Serial.print("and the other value is ");
Serial.println(b);
}
}
```

Controlling Loops

- **Break Statement**
- You can use the break statement when you need to break out of a loop before the condition would normally stop the loop
- **Example:**

```
int i;
for (i = 0; i <= 20; i++) {
if (i == 15)
Break;
Serial.print("currently on iteration:");
Serial.println(i);
}
Serial.println("This is the end of the test");
}
```

- **Continue Statement**
- You can use the continue statement to control loops. Instead of telling the Arduino to jump out of a loop, it tells the Arduino to stop
processing code inside the loop, but still jumps back to the start of
the loop
- **Example**

```
int i;
for (i = 0; i <= 10; i++){
If ((i > 5) && (i < 10))
Continue;
Serial.print("The value of the counter at");
Serial.println(i);
}
Serial.println("this is the end of the test");
```

Arrays

- An array stores multiple data values of the same data type in a block of memory, allowing you to reference the variables using the same variable name. It does it through an index value.
- **Format**
- *datatype variableName[size];*
- ***Example 1:***
  int myarray[10]; //able to store 10 values
  myarray[0] = 20; //stores values in index 0
  myarray[1] = 30; // stores values in index 1
- ***Example 2:***
  // assigns values to first 5 data locations (index 0-4)
  Int myarray[10] = {20, 30, 40, 50, 100};
- ***Example 3***
  int myarray[ ] = {20, 30, 40, 50 100};

- **Using Loops with Arrays**

```
int values[5] = {10, 20, 30, 40, 50};
for (int counter = 0; counter < 5; counter ++){
Serial.print("one value in the array is " );
Serial.print(values[counter]);
}
```

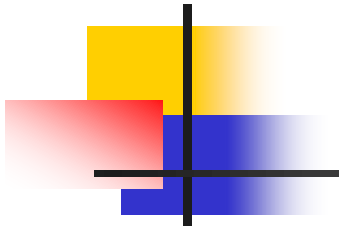
Arrays Continued

- **Determining the size of an Array**
  You may not remember how many data points are in your array
- You can use the handy *sizeof* function
- *size = sizeof(myArray) / sizeof(int);*
- **Example:**
  for (counter =0; counter < (sizeof(value)/sizeof(int)); counter++){
  //This will only iterate through number of points in an array
  statements;
  }

- **Challenge Question 1: What is the array 'myArray' look like in the program below?**
  int myArray[3][4];
  for (int i = 0; counter < 3; i ++){
  for (int j = 0; j < 4; i++) {
  myArray[i] [j] = 1;
  }
  }
- **Challenge Question 2: What is syntactically wrong with the array below? How do i fix it?**
  char myArray[ ] = { "mon", "tue", "wed", "thu", "fri"};

Strings

Method	Description
`charAt(n)`	Returns the character at the *n*th position in the string.
`compareTo(string2)`	Returns `0` if the string is equal to *`string2`*, a negative number if the string is less than *`string2`*, or a positive number if the string is greater than *`string2`*.
`concat(string1, string2)`	Appends the *`string2`* value to the end of *`string1`*, and creates a new string value.
`endsWith(string2)`	Returns `true` if the string ends with the *`string2`* value.
`equals(string2)`	Returns `true` if the string is equal to *`string2`*.
`equalsIgnoreCase(string2)`	Returns `true` if the string is equal to *`string2`*, ignoring character case.
`getBytes(buf, len)`	Copies *`len`* string characters into the *`buf`* variable.
`indexOf(val, [,from])`	Returns the index location where the string *`val`* starts in the string. By default, it starts at index 0, or you can specify a starting location using the *`from`* parameter. Returns `-1` if *`val`* is not found in the string.
`lastIndexOf(val [, from])`	Returns the index location where the string *`val`* starts in a string. By default, it starts at the end of the string, working toward the front of the string, or you can specify a starting location using the *`from`* parameter. Returns `-1` if *`val`* is not found in the string.



- A string value is a series of characters put together to create a word or sentence
- **Format**
  String name = “my text here”;
- **Example**
  String myName = “Jackie Chan”;

- **Manipulating Strings**
  String myName = “Jackie Chan”;
  myName.toUpperCase();
- Output: JACKIE CHAN
- Although you can just create a char array, Strings are much easier to work with! They have many more supported functions

More String Functions

`length()`	Returns the number of characters in the string (not counting the terminating null character).
`replace(substring1, substring2)`	Returns a new string with the *substring1* value with *substring2* in the original string value.
`reserve(n)`	Reserves a space of *n* characters in memory for a string value.
`startsWith(string2)`	Returns `true` if the string starts with *string2*.
`substring(from [,to])`	Returns a substring of the original string value, starting at the *from* index location. By default, it returns the rest of the string from that location, or you can specify the *to* index value.
`toCharArray(buf, len)`	Copies *len* characters in the string to the character array variable *buf*.
`toInt()`	Returns an integer value created from the string value. The string must start with a numeric character, and the conversion stops at the first non-numeric character in the string.

More string functions

Method	Description
`setCharAt(index, c)`	Replaces the character at *index* with the character *c*.
`toLowerCase()`	Converts the string value to all lowercase letters.
`toUpperCase()`	Converts the string value to all uppercase letters.
`trim()`	Removes any leading and trailing space or tab characters from the string value.

Functions

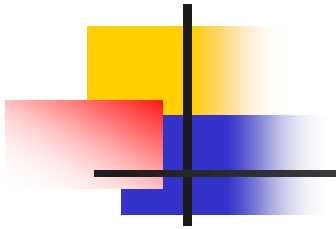
- You'll often find yourself using the same code in multiple locations. Doing large chunks of these is a hassle. However, functions make these a lot easier. You can encapsulate your C code into a function and then use it as many times as you want.
- ***Structure***

```
datatype functionName() {
// code statements
}
Void myFunction() {
Serial.println("This is my first function");
}
```

- **TIP:** Make sure you define your function outside of the setup and loop functions in your arduino code sketch. If you define a function inside another function, the inner function becomes a local function, and you can't use it outside the outer function To create a function that does not return any data values to the calling program, you use the ***void*** data type for the function definition

Using the function/Returning a value

- **Using the Function**
- To use a function you defined in your sketch, just reference it by the function name you assigned, followed by parentheses
- void setup() {
  Serial.begin(9600)
  MyFunction();
  Serial.println("Now we're back to the main program");
  }
- **Returning a Value**
  To return a value from the function back to the main program, you end the function with a return statement return value;



- The value can either a constant numeric or string value, or a variable that contains a numeric or string value. However, in either case, the data type of the returned value **must match** the data type that you use to define the function
- int myFunction2() {
  int value = 10*20;
  return (value);
  }

Passing Values to Functions

- You will most likely want to pass values into function. In the main program code, you specify the values passed to a function in what are called arguments, specific inside the function parenthesis
- returnValue = area(10,20);
  The 10 and 20 are value arguments separated by a comma. To retrieve the arguments passed to a function, the function definition must declare what are called parameters. You do that in the main function declaration line

Example

```
void setup() {
Int returnValue;
Serial.begin(9600);
Serial.print("The area of a 10 x 20
size room is ");
returnValue = area(10,20);
Serial.println(returnValue);
}
void loop() {
}
int area (int width, int height) {
int result = width * height;
Return result;
}
```

Handling Variables inside Functions

- One thing that causes problem for beginning sketch writers is the scope of a variable. He scope is where the variable can be referenced within the sketch. Variables defined in function can have a different scope than regular variables. That is, they can be hidden from the rest of the sketch. Functions use two types of variables:
  - Global Variables
  - Local Variables
- Defining Global Variables
- Write them before the setup() loop.
- Ex:
  const float pi = 3.14;
- Be careful in modifying global variables.
- Declaring local variables
- Local variables are declared in the function code itself, separate from the rest of the sketch code. What's interesting is that a local variable can override a global variable (but not good practice)

Calling Functions Recursively

- Recursion is when the function calls itself to reach an answer. Usually a recursion function has a base value that it eventually iterates down to. Many advanced algorithms use recursion to reduce a complex equation down one level repeatedly until they get to the level defined by the base value
- int factorial (int x) {
  if (x <=1) return 1;
  else return x * factorial(x-1);
  }
- Remember you are allowed to call other functions from inside a function

Structs

- Data structures allow us to define custom data types that group related data elements together into a single object.
- Before you can use a data structure in your sketch, you need to define it. To define a data structure in the Arduino, you can use the *struct* statement. Here is the generic format for declaration
- **Format**
  struct *name* {
  variable list
  };
- ***Example of declaration***
  struct sensorinfo {
  char date[9];
  int indoortemp;
  int outdoortemp;
  }morningTemps, noonTemps, eveningTemps;

Full Example: Declaring and Calling

- struct sensorinfo {
  char date[9];
  int indoortemp;
  int outdoortemp;
  }morningTemps
- void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  strcpy(morningTemps.date, "01/01/14");
  morningTemps.indoortemp = 72;
  morningTemps.outdoortemp = 25;
  Serial.print ("Today's date is ");
  Serial.println(morningTemps.date);
  Serial.print("The morning outdoor temperate is ");
  Serial.println(morningTemps.outdoortemp);
  }

Unions

- Unions allow you to have a variable take on different data types
- **Format**

  union {
  //variable list
  };
- **Example**

  Union {
  float analogInput;
  int digitalInput;
  } sensorInput;

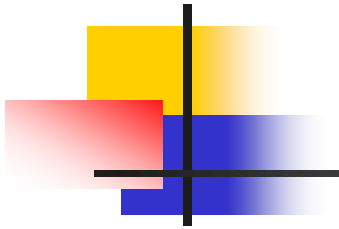
- **//Full Example:**

  Union{
  float analogInput;
  Int digitalInput;
  }sensorInput;
  //Saving a value
  sensorInput.analogInput =
  myFunction1();
  sensorInput.digitalInput =
  myFunction2();
  //Calling a value;
  Serial.println(sensorInput.analogInput);
  Serial.println(sensorInput.DigitalInput);

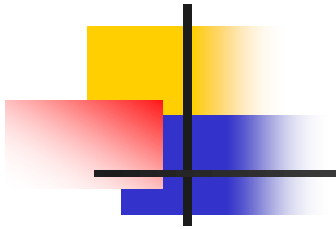
Using Libraries

Library	Description
EEPROM	Functions to read and write data to EEPROM memory.
Esplora	Functions for using the game features of the Esplora unit.
Ethernet	Functions for accessing networks using the Ethernet shield.
Firmata	Functions for communicating with a host computer.
GSM	Functions for connecting to mobile phone networks using the GSM shield.
LiquidCrystal	Functions for writing text to an LCD display.
Robot_Control	Functions for the Arduino Robot.
SD	Functions for reading and writing data on an SD card.
Servo	Functions for controlling a servo motor.
SoftwareSerial	Functions for communicating using a serial port.
SPI	Functions for communicating across the SPI port.
Stepper	Functions for using a stepper motor.
TFT	Functions for drawing using a TFT screen.
Wifi	Functions for accessing a wireless network interface.
Wire	Functions for communicating using the TWI or I2C interfaces.

TABLE 13.1 The Standard Arduino Libraries

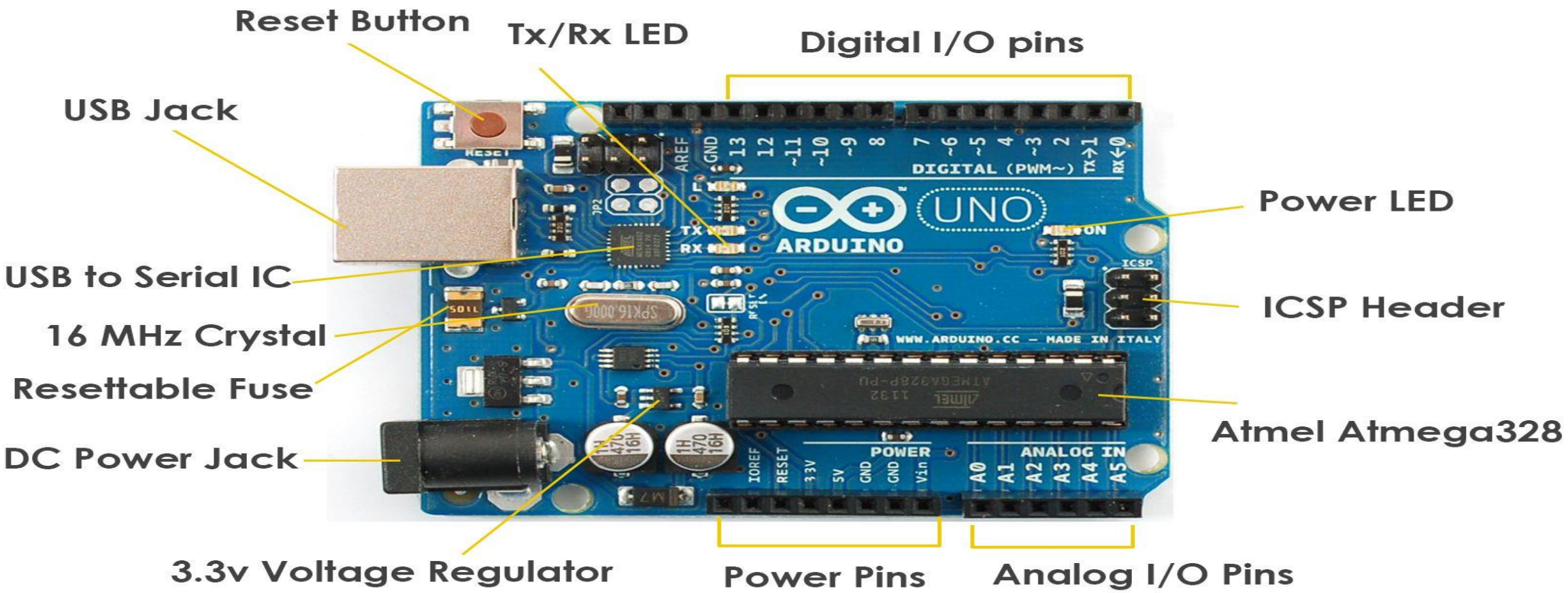


- Libraries allow you to bundle related functions into a single file that the Arduino IDE can compile into your sketches. Instead of having to rewrite the functions in your code, you just reference the library file from your code, and all the library functions become available. This is handy for Arduino Shields
- Defining the library in your sketch
  #include ‘libraryheader’
  #include ‘EEPROM’
- Referencing the Library Functions
  Library.function()
- Ex. for the EEPROM library
  EEPROM.read(0)

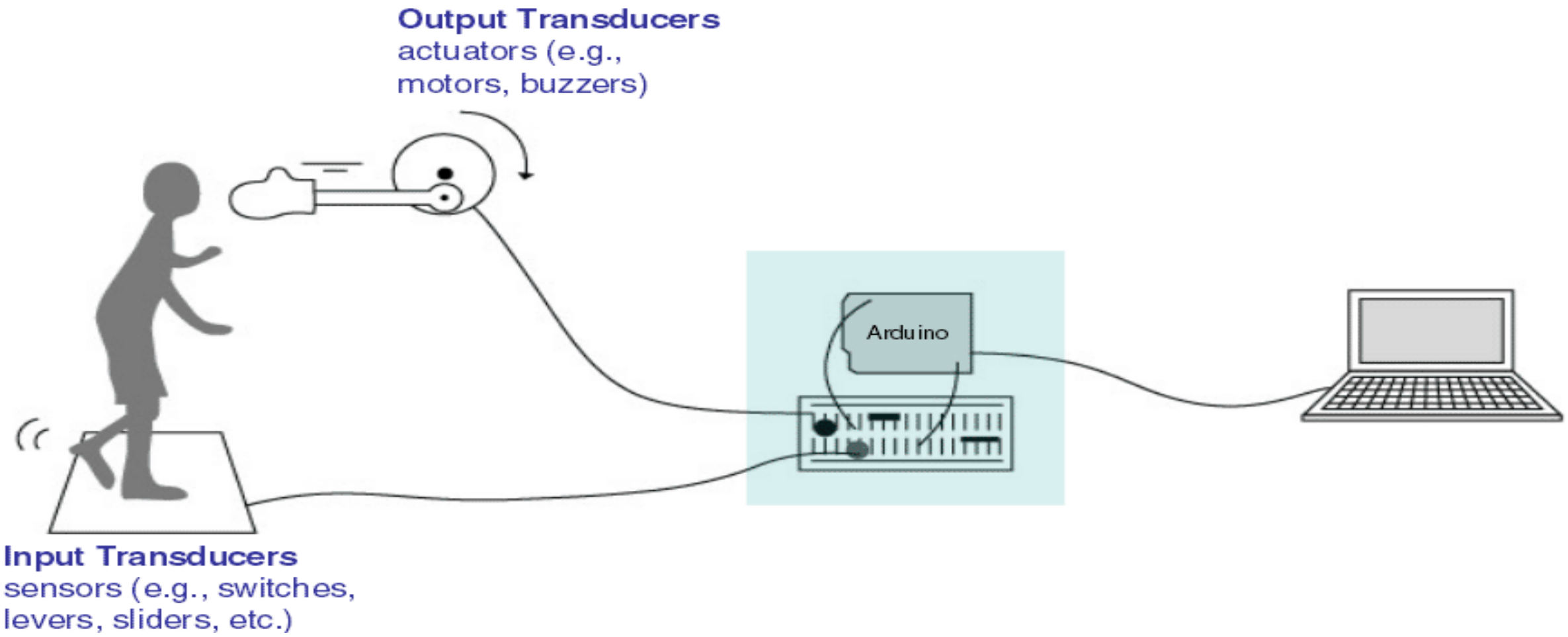


- Installing your library
  - 1. Open the Arduino IDE
  - 2. Select the sketch from the menu bar
  - 3. Select Import libraries from the submenu
  - 4. Select Add library from the list of menu options

Input/Output Layout on Arduino



Input/Output



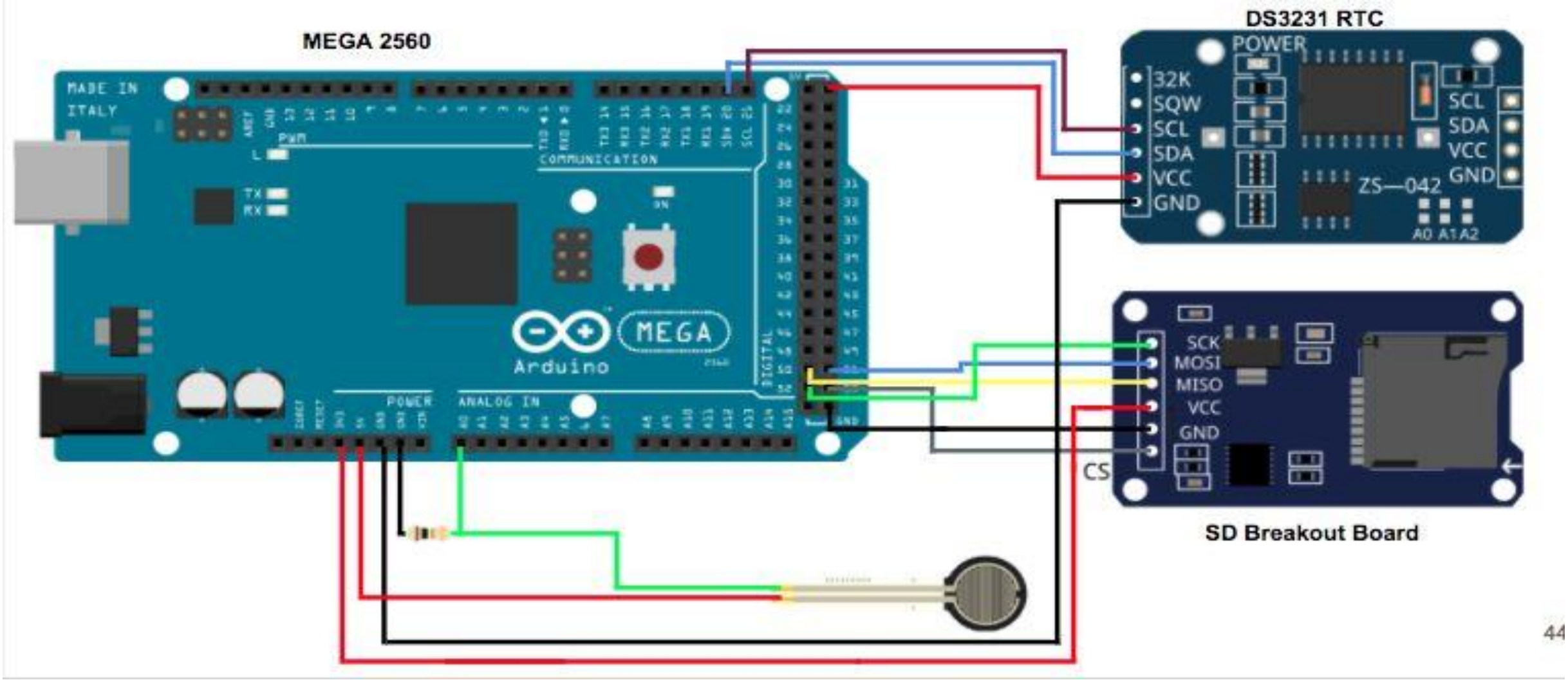
Digital & Analog I/O

- **Format**
  pinMode(pin, MODE);
- The pinMode function requires two parameters. The pin parameter determines the digital interface number to set. The mode parameter determines whether the pin operates input or output mode. There are 3 values for interface mode setting:
  - INPUT - To set an interface for normal input mode
  - OUTPUT - To set an interface for output mode
  - **Getting a reading from an input device:**
    analogRead(pin)

Complete Example

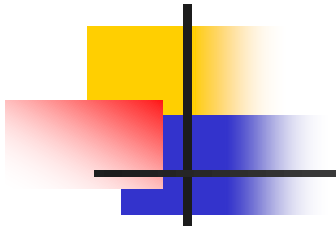
```
void setup (){
Serial.begin(9600);
pinMode(A1, INPUT);
}
void loop() {
float reading = analogRead(A1);
}
```

Reading Connection Diagrams



Writing to EEPROM

- EEPROM is the long term memory in the Arduino. It keeps data stored even when powered off, like a USB flash drive. Important Note: EEPROM becomes unreliable after 100,000 writes
- You'll first need to include a library file in your sketch **#include <EEPROM.h>**
- After you include the standard EEPROM library file, you can use the two standard EEPROM functions in your code:
- ***read (address)*** - to read the data value stored at the EEPROM location specified by address *write (address, value)* - to read values to the EEPROM location specified by address



```
#include <EEPROM.h>
void setup()
{
for (int i = 0; i < 255; i++)
EEPROM.write(i, i);
}
void loop()
{}
```

Debugging Applications

- Compiler will give you the line code(s)
- Compiler will give you the error
- The line causing problems may get highlighted
- Look up errors online