Intro to Arduino

No Experience and No Hardware Required Tutorial

Spring 2020

Prepared by Katelyn Brinker (katelyn.brinker@ieee.org)

Background and Setup

Needing to purchase hardware is often an obstacle for getting started with tinkering with Microcontrollers. In this tutorial we'll be doing a no experience required, no hardware required, introduction to Arduino. We'll be using an Arduino simulator (TinkerCAD) for this tutorial, but parallel instructions will also be provided for those who do have the hardware. While there are many different microcontroller simulators out there, TinkerCAD was chosen because it provides a good parallel to working with real hardware (lights blink, motors move, circuits can be built with breadboards, etc.) and it's free.

Objectives:

- Get familiar with common functions and the programming syntax of the Arduino language
- Work with different microcontroller accessories to build a foundation for future tinkering

Hardware List (Optional):
If you plan to use hardware during this workshop, here's a list of what you will need:

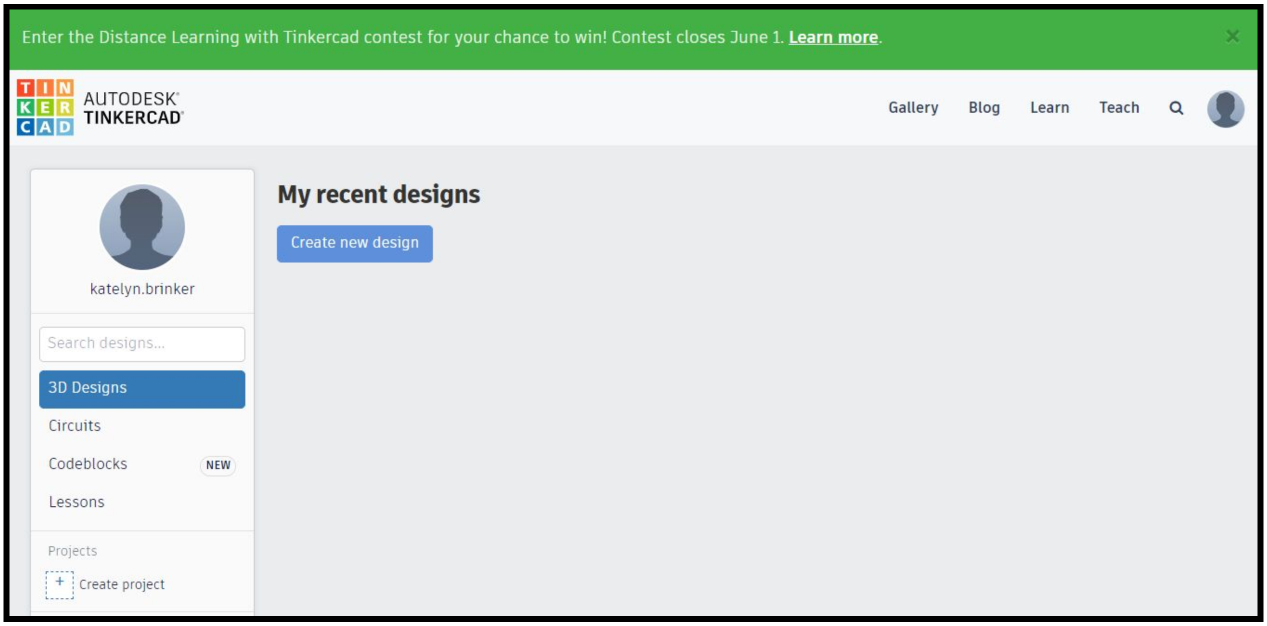
1. Arduino Uno
2. Breadboard
3. 1 LED
4. 1 resistor (anything from 100 to 1k will work)
5. 1 10k resistor
6. Push button
7. Servo motor
8. Jumper wires
9. DHT11 sensor

Note: There's a wide variety of Arduino starter kits out there that can be purchased that contain all the components you would need for this tutorial and more. Here are two options available on Amazon:

- Starter kit 1
- Starter kit 2

Setting up the Arduino Simulator - TinkerCAD:
TinkerCAD is an Autodesk online simulator which allows for Arduino simulations. To get started, login with an Autodesk account or create one: https://www.tinkercad.com/

After logging in you'll be brought to the TinkerCAD dashboard.



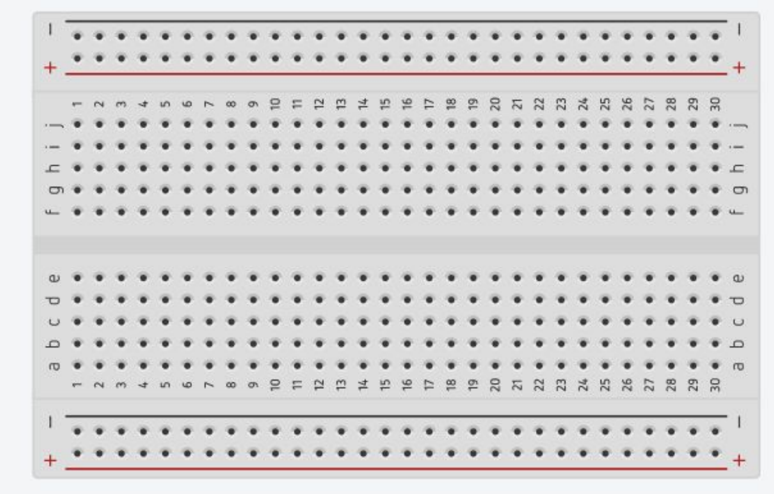
Enter the Distance Learning with Tinkercad contest for your chance to win! Contest closes June 1. Learn more.

Setting up the Arduino software:
For both using the simulator and using real hardware, we will be using the Arduino integrated development environment (IDE). Download and install it from this link: https://www.arduino.cc/en/main/software

Inside TinkerCAD, you can write and upload code to your virtual circuit. However, TinkerCAD creates basic code for you to start with based on the components you're using. In the case of this tutorial, that's not really conducive for learning how to code in Arduino so we're going to write the code in the Arduino IDE and then copy it into TinkerCAD.

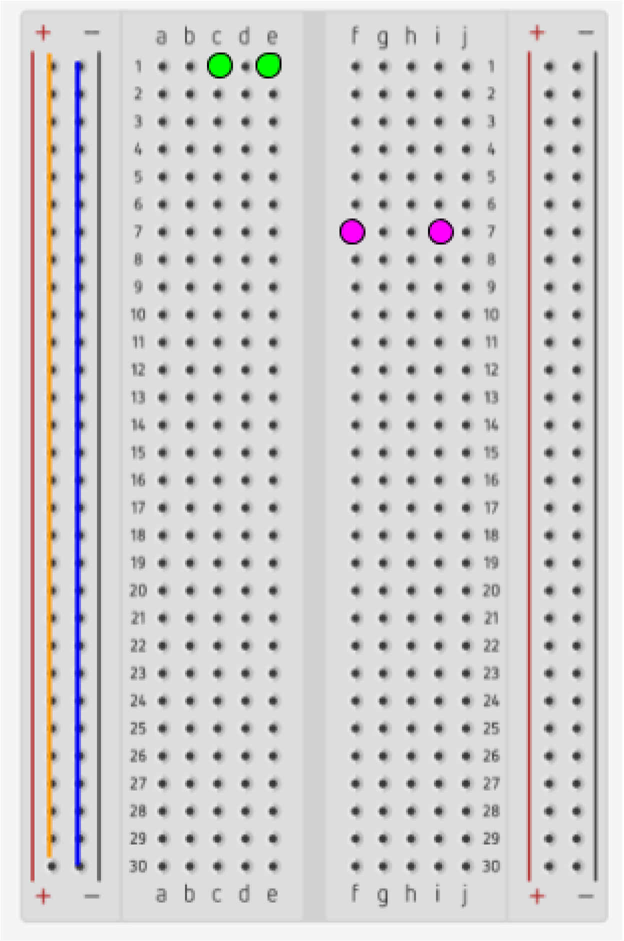
Circuit 1: Turning on an LED

A brief intro to breadboarding:



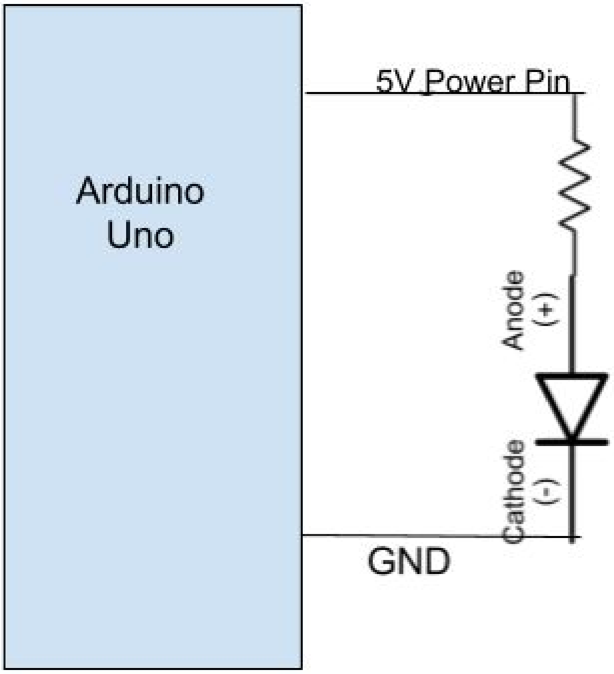
On a breadboard, there's a couple conventions that you need to remember in order to make electrical connections between components:

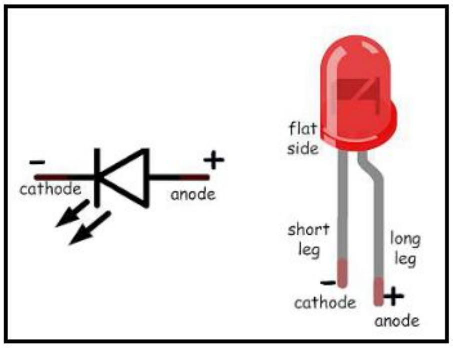
1. Each column on the sides of the breadboard labeled with a '+' or '-' is electrically connected. These are referred to as the power columns. This means that all the holes in the first column (marked with an orange line below) are internally electrically connected to each other. Similarly all the holes in the second column (marked with a blue line) are internally electrically connected.
    a. These power columns are not connected to each other - there is no internal electrical connection between a hole in the orange column and a hole in the blue column.
2. In the columns labeled a-e and f-j, the holes in each **row** are connected. For example hole 1c is connected to hold 1e (green circles), and hole 7f is connected to hole 7i (pink circles).
    a. The break in the board between columns e and f is also typically a break in the internal electrical connections in the board, so in a row a hold in column d won't be connected to a hole in column h in the same row.



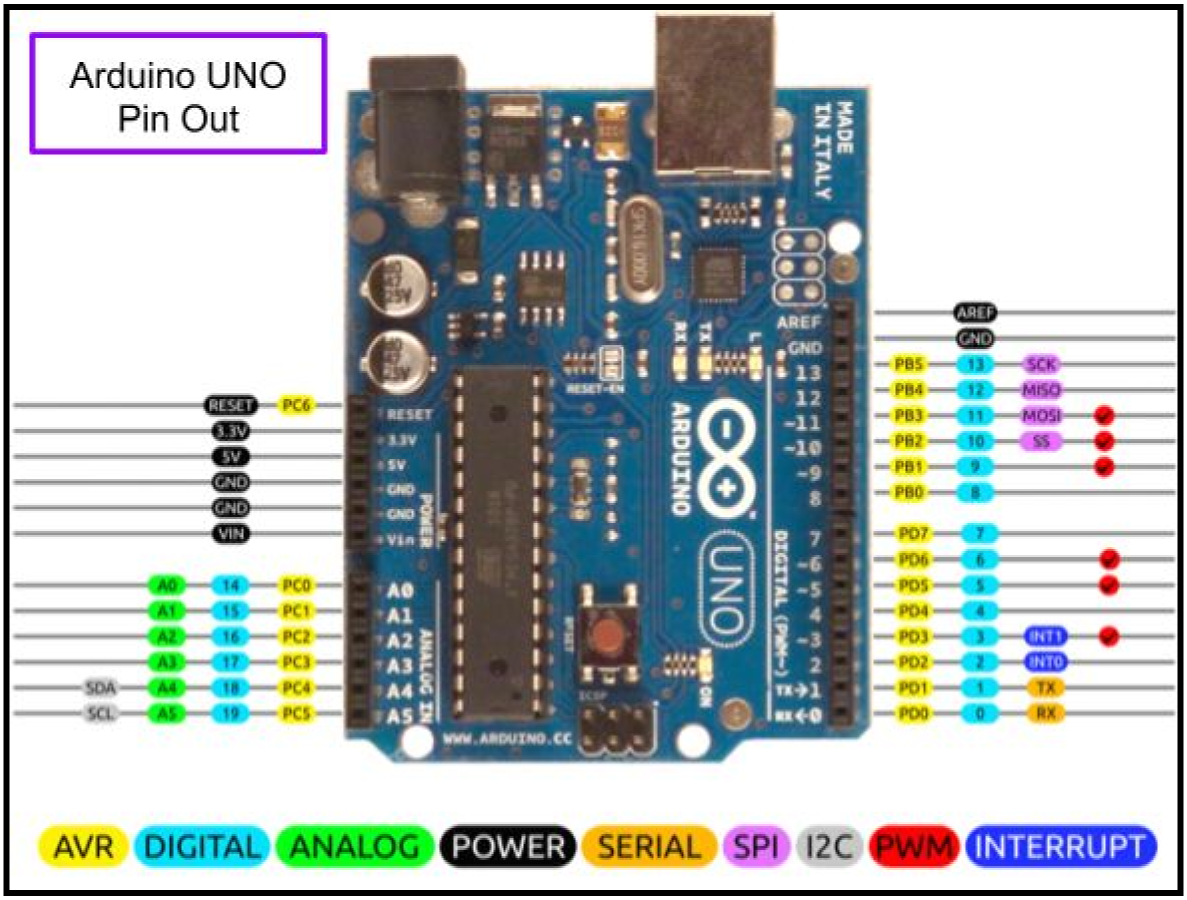
Building the Circuit:

We're going to build the following circuit in TinkerCAD/on hardware. This circuit will allow us to control when the LED turns on and off through code.

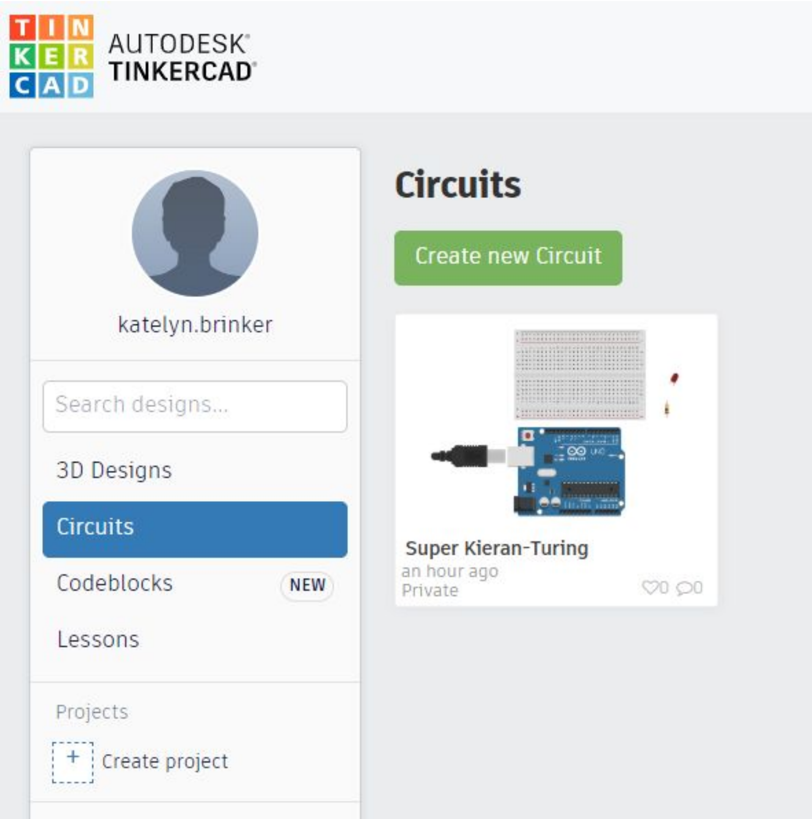




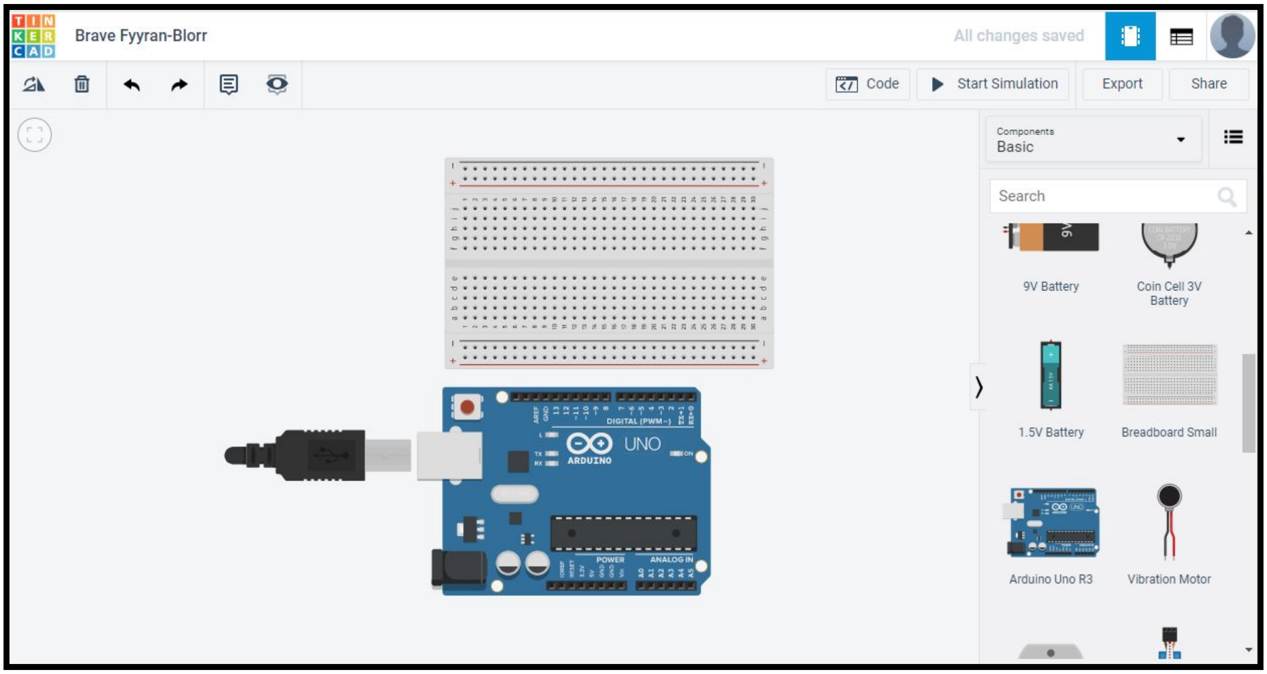
On the Arduino Uno there are pins (physical inputs and outputs that we can connect to) that have different capabilities. There are power, analog, digital, pwm, interrupt, I2C, Serial, and SPI pins as described by the diagram below, which is commonly referred to as the pin out. We're primarily going to be working with the Power, Digital, and Analog pins in this tutoria, which you'll also noticed are labeled on the board in the pin outl.



1. To start the circuit in TinkerCAD, click on “Circuits” in the menu on the left of the Dashboard and then select “Create New Circuit”

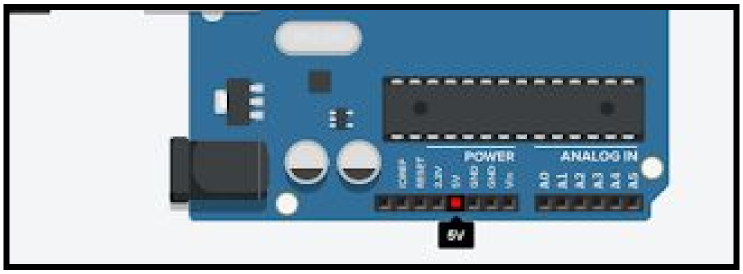


2. To add components to the canvas, click on them in the components menu on the right. Start by placing an Arduino Uno and a breadboard into the canvas.
    a. Once you place a component you can select and move it by clicking and dragging.
    b. Use your mouse scroll wheel to zoom in and out. Hold down the scroll wheel to pan.

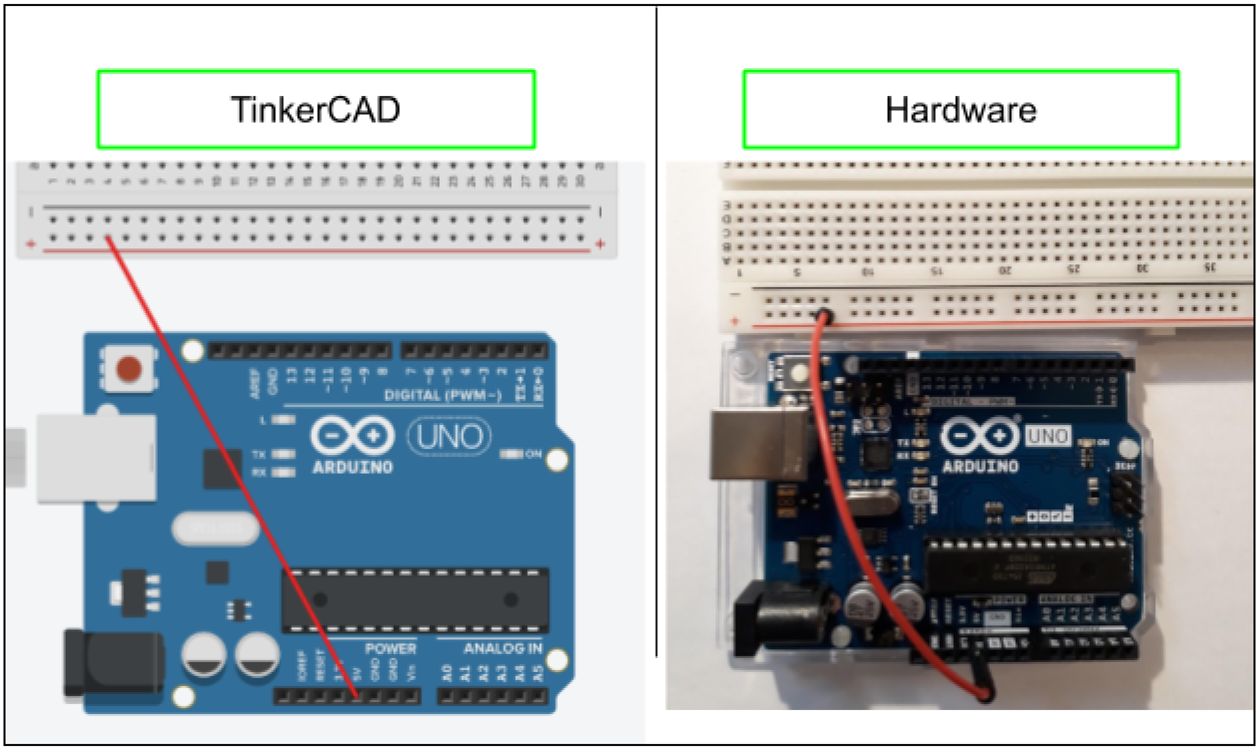


We’re going to start by making our power/control and ground connections between the Arduino and the breadboard.

3. Make a connection between the 5V pin on the Arduino and the + power column on the breadboard.
    a. TinkerCAD: click on the pin on the arduino (the pin will turn red and display the label “5V” when you hover over it with your mouse) and then click on one of the holes in the + power column on your breadboard. This will make a wire. You can change the color of the wires in TinkerCAD after you place them. In this tutorial power wires are going to be red for easy identification.

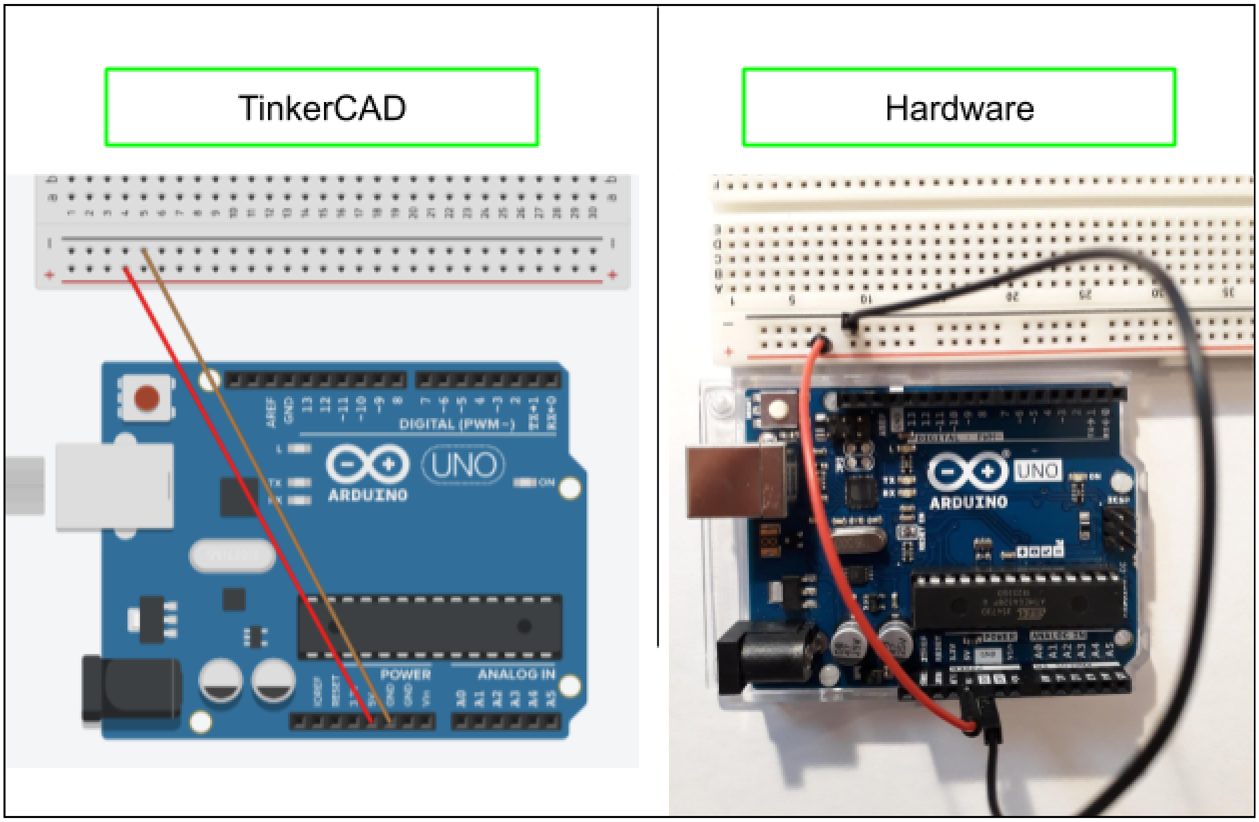


    b. Hardware: plug a jumper wire into the 5V pin on the arduino and into the + power column on the breadboard.


TinkerCAD

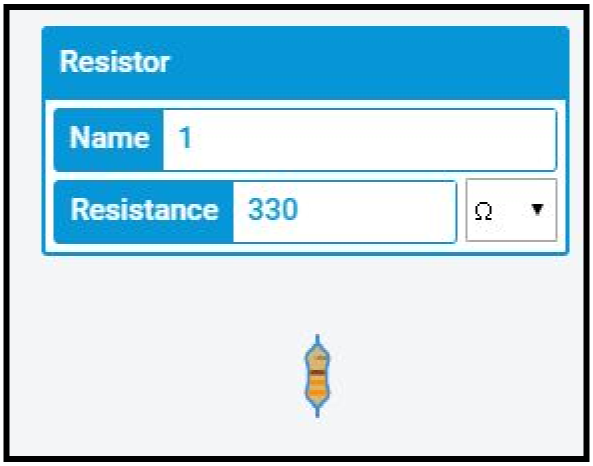
Hardware

4. Make a connection between one of the ground (GND) pins on the Arduino Uno and the - power column on the breadboard.
    a. TinkerCAD: brown wires will be used for gnd connections for ease of identification.
    b. Hardware: use another jumper wire to make this connection.



Next, we're going to add a 330Ω resistor and an LED to the circuit. The resistor is used to limit the current flowing to the LED so that it doesn't burn out. The higher the resistance of the resistor, the less current that will flow to the LED and the dimmer it will appear.

5. Add the 330Ω resistor and LED to the canvas.
    a. TinkerCAD: once you place the resistor in the canvas you'll have the option to change its value. Notice that the stripes on the resistor change color to match how the physical resistor of that value would look.



Resistor

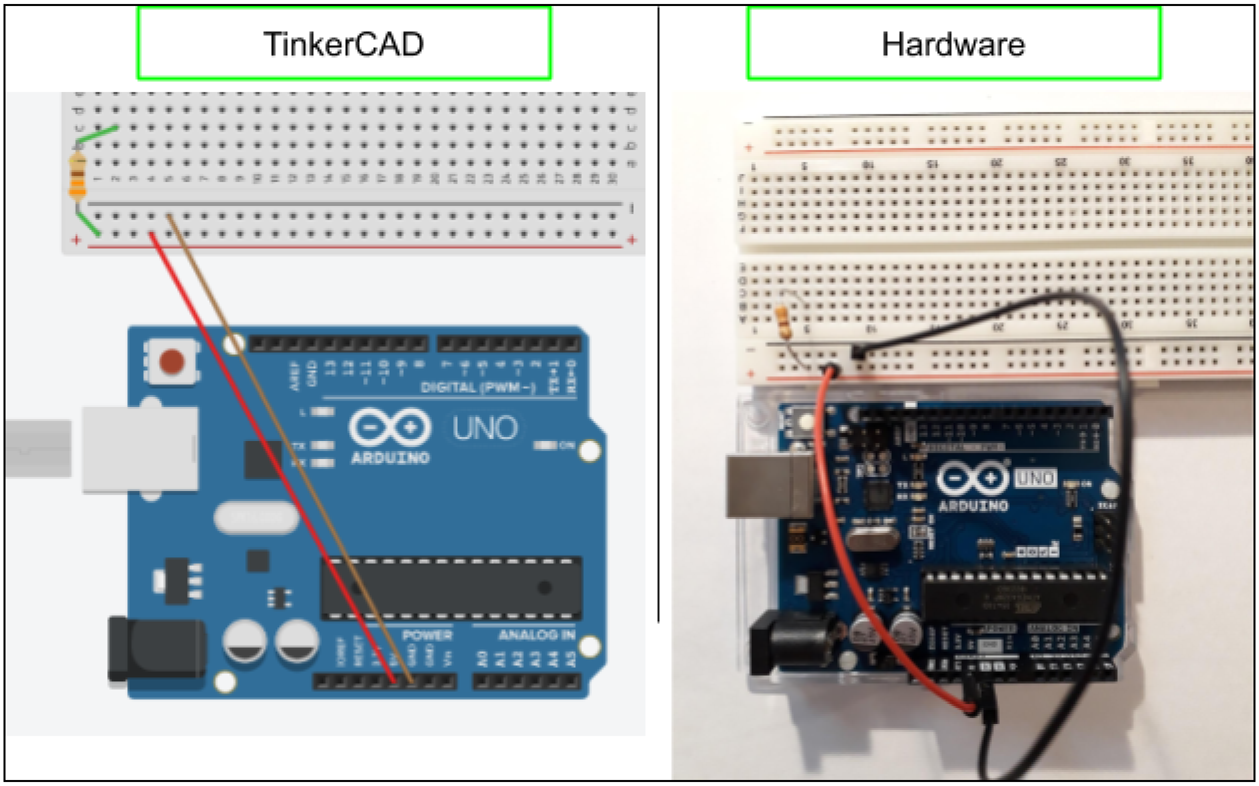
Name: 1

Resistance: 330 Ω ▼

6. Connect the resistor to the breadboard between the + power column and one of the holes in the a-e columns of the breadboard. Since all the holes in the + power column are connected, this is making the electrical connection between the 5V pin on our Arduino and the resistor.

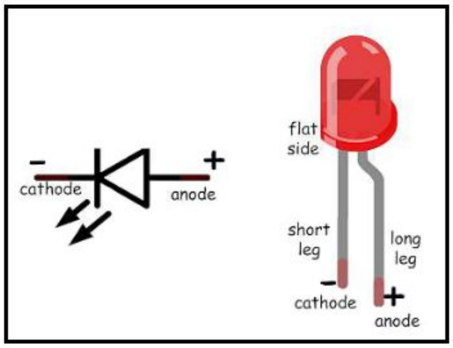
a. TinkerCAD: Use wires to make the connections. It helps to place the resistor to the side of the breadboard, draw the connections, and then move the resistor onto the breadboard so that you can see where you're making connections. The wires will move with the resistor when you move it.

b. Hardware: plug one leg of the resistor into the + power column and one leg into the middle of the breadboard.



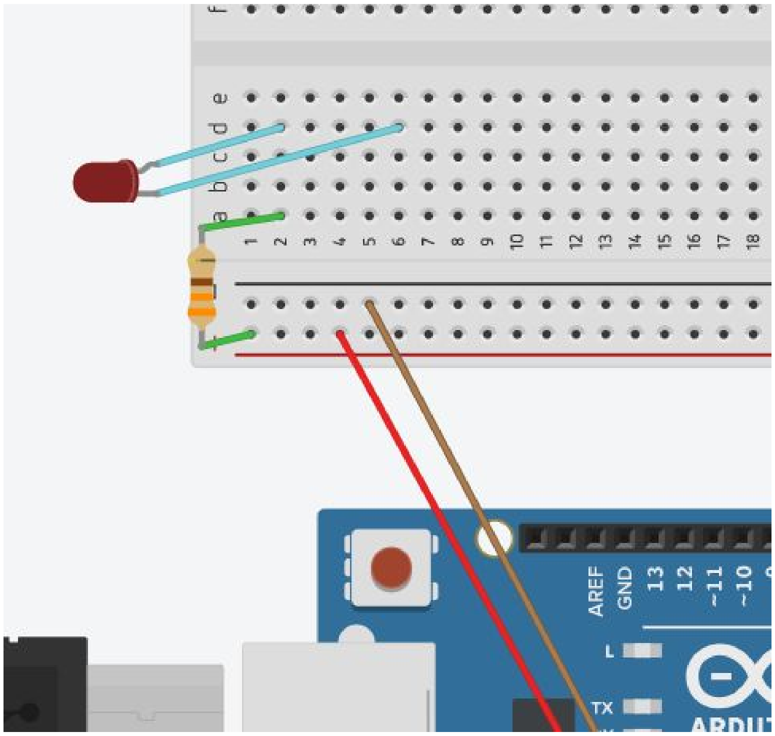
LEDs are polarized, meaning they only work in one direction, while resistors are unpolarized. In a circuit, current is going to flow in the direction of the LED/diode symbol arrow (i.e., in the anode (+) and out the cathode (-)). On an LED, what's the anode and what's the cathode is denoted in a coupled ways:

- The legs are two different lengths - the longer in the anode
- One of the sides of the "light bulb" portion of the LED is flat. It's hard to see but you can feel it. This is the cathode side.
- To remember which way to put an LED in the circuit, you can think of the symbol like a funnel.

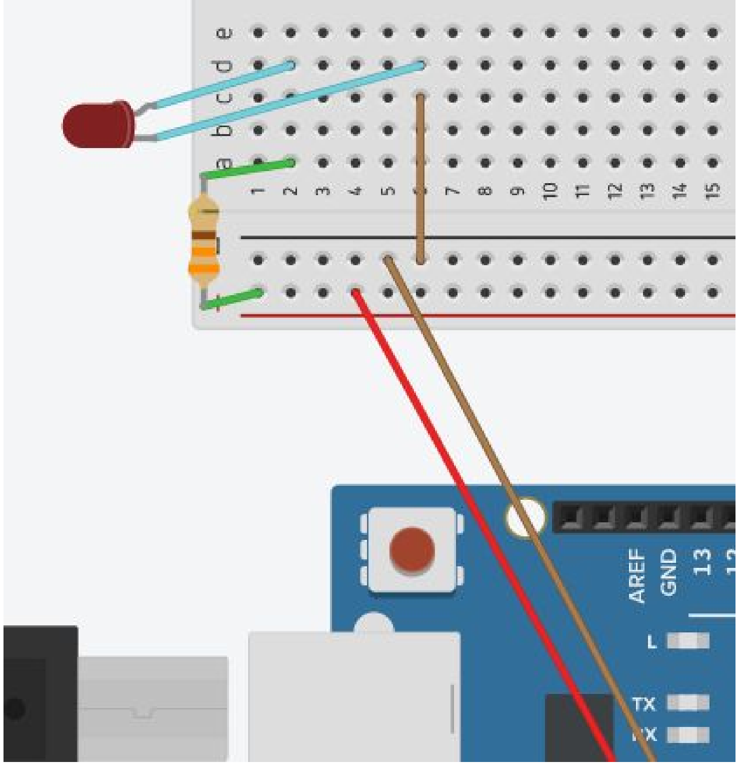


In TinkerCAD when you hover over one of the legs, it tells you whether it's the anode or the cathode. You can rotate components by selecting them and clicking the rotate button in the upper left.

7. Connect the anode of the LED to the same row of the breadboard as the resistor and connect the cathode to any other row. This will make a connection between the resistor and the LED.



8. Complete the circuit by connecting the cathode of the LED to the - power column.



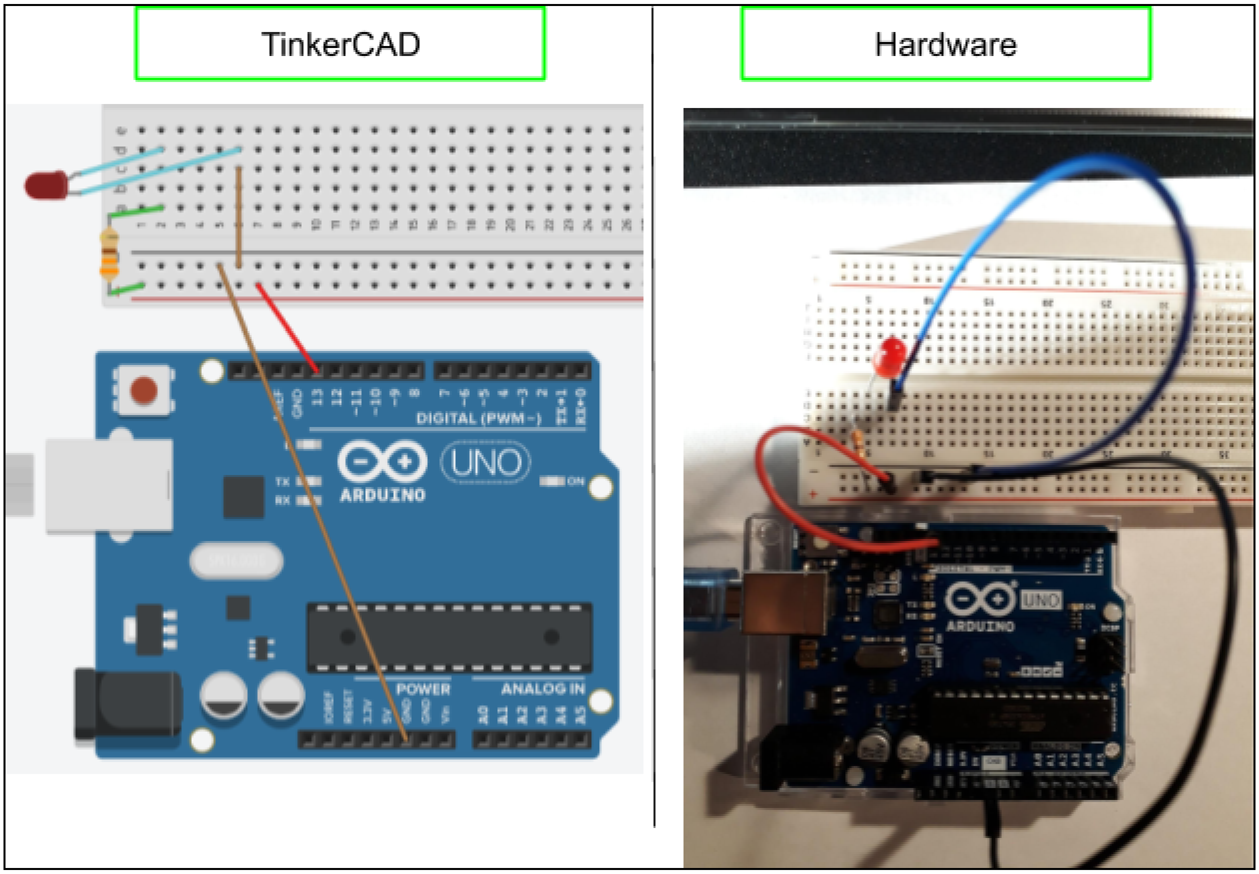
9. Test your circuit. If it’s wired correctly the LED will turn on.
    a. TinkerCAD: Click “Start Simulation” in the upper right of TinkerCAD. This will plug in the USB cable and power the circuit.
    b. Hardware: plug in the USB cable to the arduino and your computer. The Arduino power light will turn off if the Arduino is receiving power from your computer.

Circuit 2: Blinking LED

Next, we’re going to program the LED to blink by slightly changing the previous circuit.

Setup the Circuit:

1. Move the connection to the 5V pin on the arduino to digital pin 13 (D13).



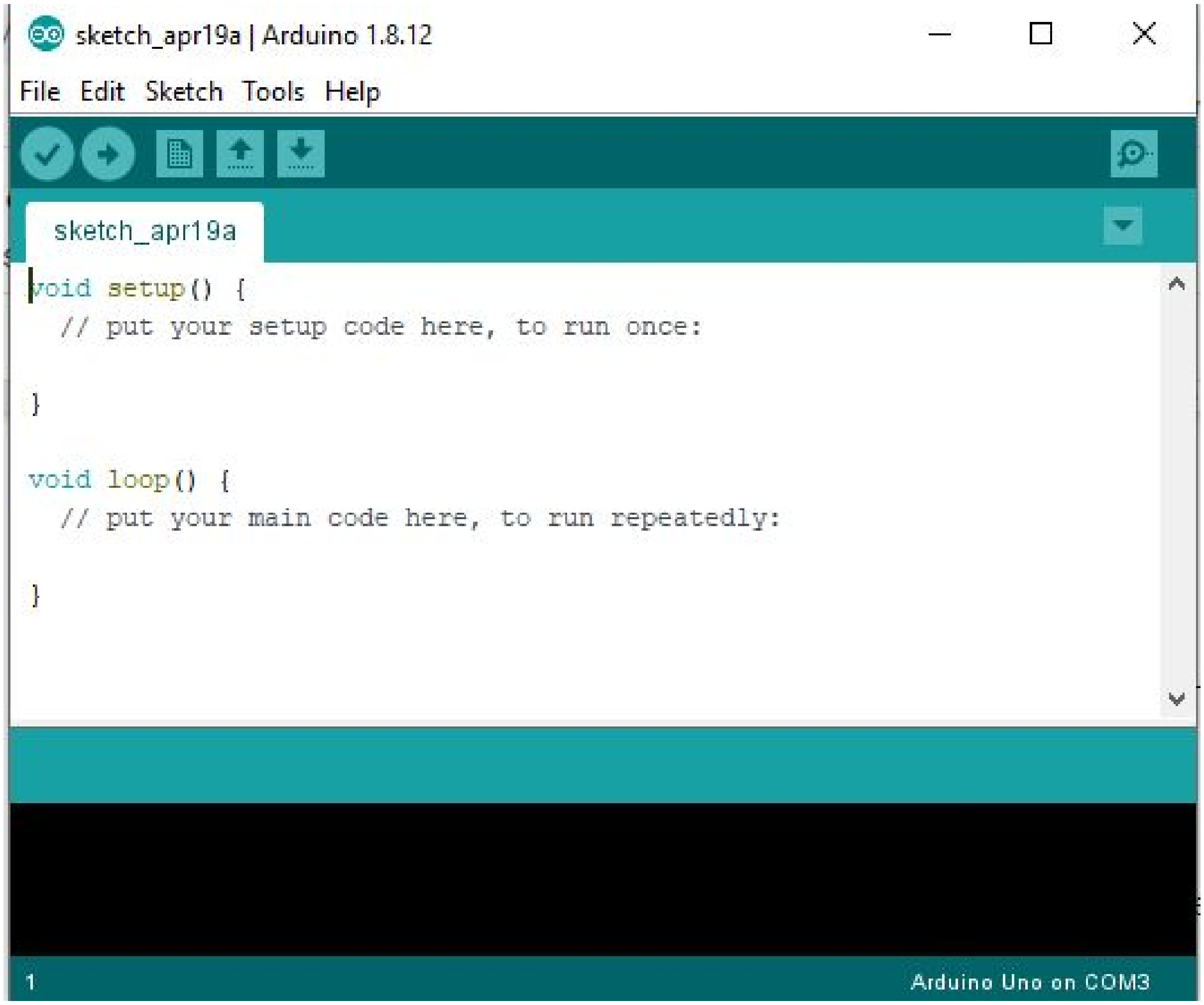
Write the Code:

It is suggested that you write the code in the Arduino IDE and then copy it into TinkerCAD since TinkerCAD will auto-generate a lot of the code for you and that kind of defeats the purpose of this tutorial.

1. Open up the Arduino IDE and start a new sketch (File - New).

In the Arduino IDE and language (the language you code in is also called Arduino), programs are referred to as sketches. In the sketch there are two main components/functions:

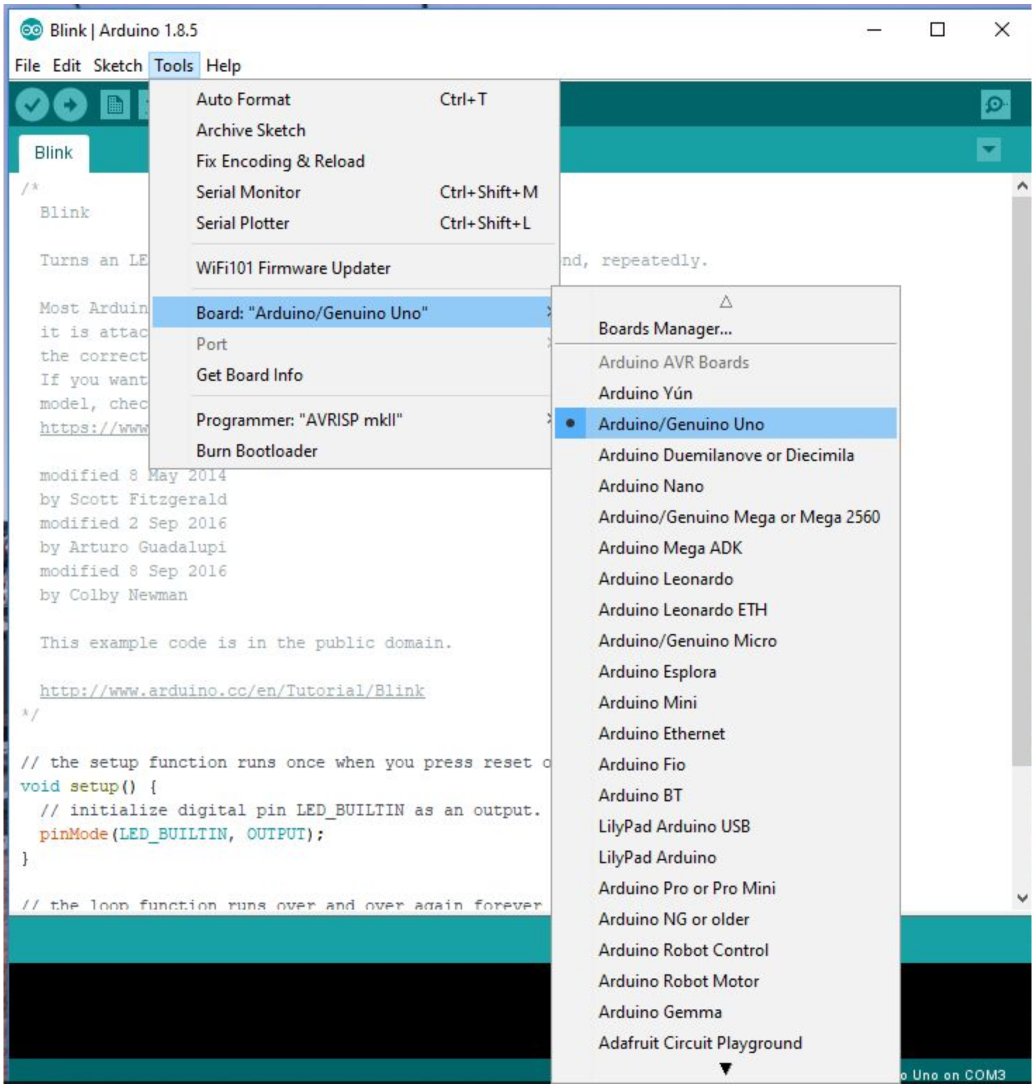
- Void setup(): This is where we lay the foundation of the code by telling the IDE what we're going to use each of the pins we're using for. This only runs once when the code is uploaded to the physical (or virtual) Arduino.
- Void loop(): this is where the body of the code goes. This is where we're going to tell the pins what we want them to do so that we can control the accessories, like the LED, that are attached to the pins. This part of the code will run over and over again until you either remove power from the Arduino or change the code.



```
void setup() {
  // put your setup code here, to run once:

}

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

}
```

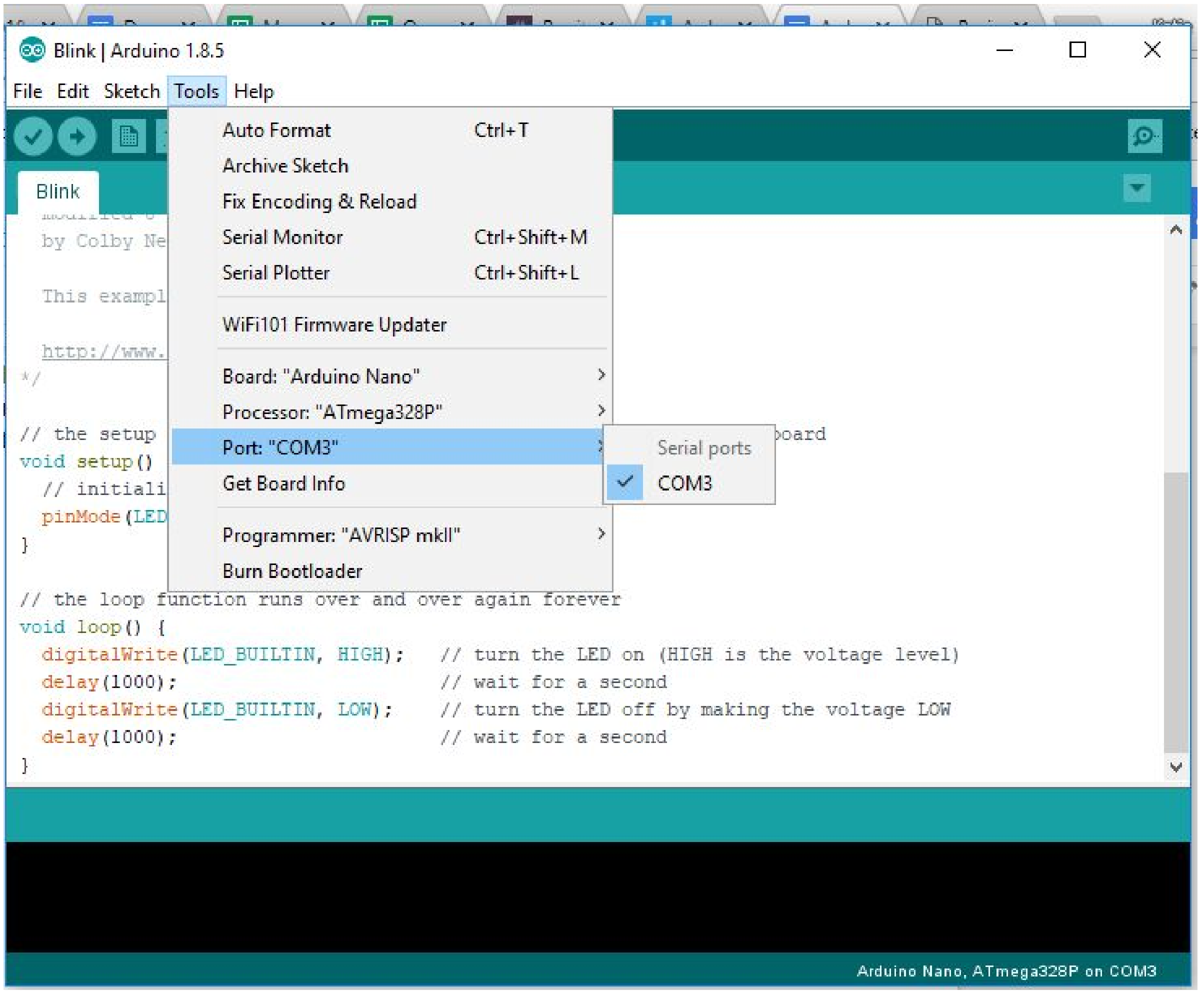
In Arduino, comments are designated by double slashes "//"

2. Select the Arduino Uno as the board you will be using by going into the "Tools" menu at the top of the IDE. This informs the IDE so it knows what pin is which.



```
/*
  Blink

  Turns an LE                                          nd, repeatedly.

  Most Arduin
  it is attac
  the correct
  If you want
  model, chec
  https://www

  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 o
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
```

3. If you are using hardware, plug in your Arduino (the power light on the board should be on) and check your Port through the “Tools” menu. If you don’t have the proper com port selected, your computer won’t be able to upload code onto the Arduino.



```
  modified 8
  by Colby Ne

  This exampl

  http://www.
*/

// the setup                                  board
void setup()
  // initiali
  pinMode(LED
}

// 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
}
```

The Arduino language is similar to C and has a lot of built in functions that we'll be working with. Builtin function definitions are provided below. The function names are in blue. Note the capitalization.

Functions:

pinMode(pin, type)

- Pin: this is the arduino pin you are referencing
- Type: INPUT, OUTPUT, INPUT_PULLUP
    - INPUT_PULLUP: pin is an input but will be pulled high by default
- Example: pinMode(LED_BUILTIN, OUTPUT);

digitalWrite(pin, level)

- Pin: pin you want to control
- Level: HIGH or LOW
    - Pay attention to the device you are controlling. If it is active low it will be on when the pin is set LOW
- Example: digitalWrite(13,LOW);

digitalRead(pin)

- Pin: the pin whose value you want to determine
- Often used in if statements to see if a pin meets certain conditions

- Example: digitalRead(2);

analogWrite(pin, value)

- Pin: pin you want to control
- Value: between 0 and 255. The value varies the duty cycle of the PWM signal being written to a pin
- Example: analogWrite(A0, 27);

analogRead(pin)

- Pin: the pin whose value you want to determine
- Example: analogRead(A2);

delay(x)

- x: millisecond value that determines the duration of the delay. The next command won’t be executed until the delay is complete.
- Example: delay(200);

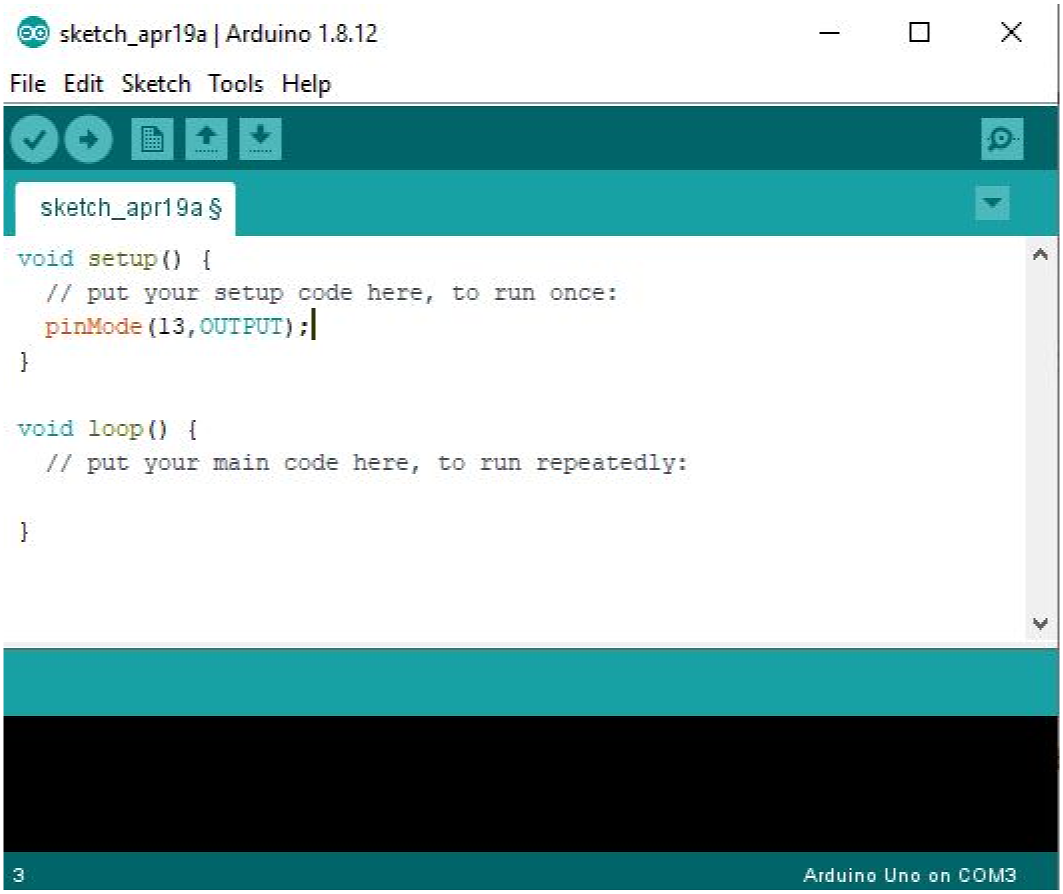
Serial.print(val) or Serial.print(val, format)

- Examples:
  Serial.print(78) gives "78"
  Serial.print(1.23456) gives "1.23"
  Serial.print('N') gives "N"
  Serial.print("Hello world.") gives "Hello world."
  Serial.print(78, BIN) gives "1001110"
  Serial.print(78, OCT) gives "116"
  Serial.print(78, DEC) gives "78"
  Serial.print(78, HEX) gives "4E"

Code tips:

- Make sure you’re including semi-colons. Every code line you add needs to end with a semi-colon.
- Watch your capitalization
- Make sure you’re referencing the correct pin

4. Setup digital pin 13 as an OUTPUT pin in the setup function using the pinMode() function. Try to do this without checking the answer in the Figure below.



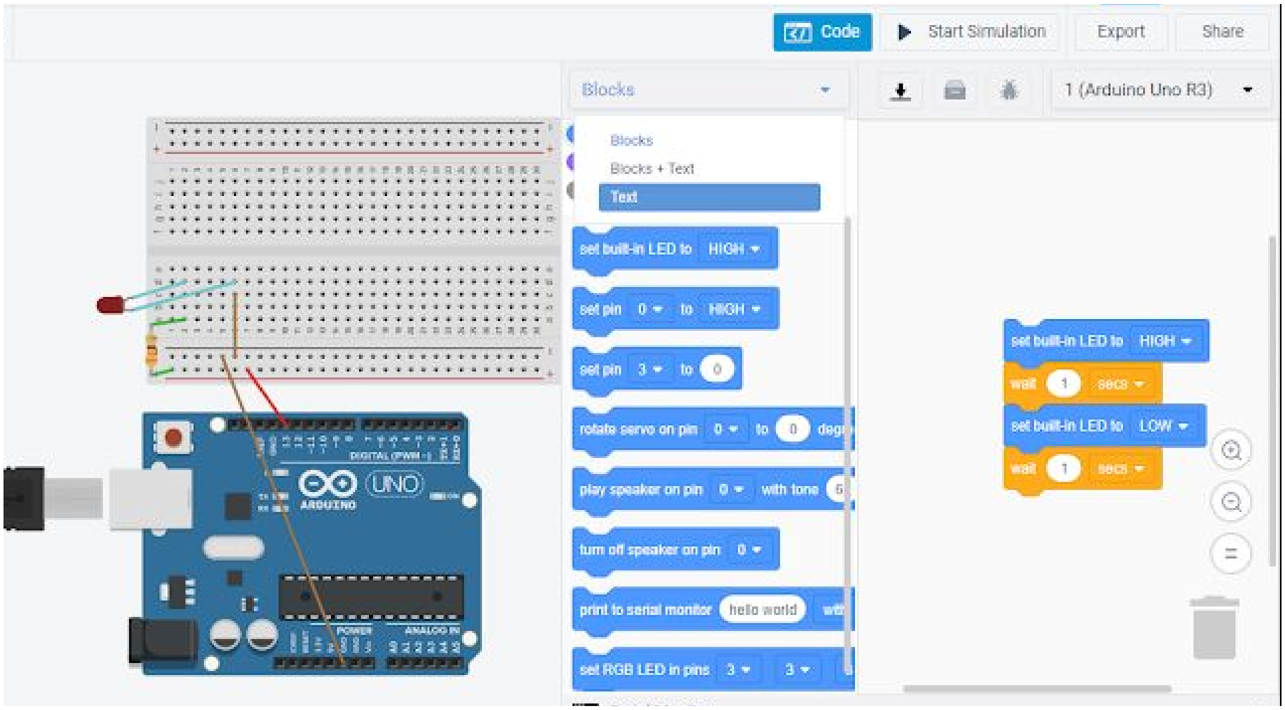
```
void setup() {
  // put your setup code here, to run once:
  pinMode(13,OUTPUT);
}

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

}
```

In order to make the LED blink we need to turn it on and off through pin 13. Since the code in the loop() functions repeats over and over again and runs really quickly, we need a delay between turning the LED on and turning the LED off and turning it back on again so that we can actually see it blinking.

5. Write the code in the loop function to make the LED blink using the digitalWrite() and delay() functions. Use a delay of 500 milliseconds to start.

6. Hit verify (checkmark in the upper lefthand corner) to save and compile your code. If you have any mistakes like a missing semicolon, you'll get an error print out in the bottom message section of the IDE.

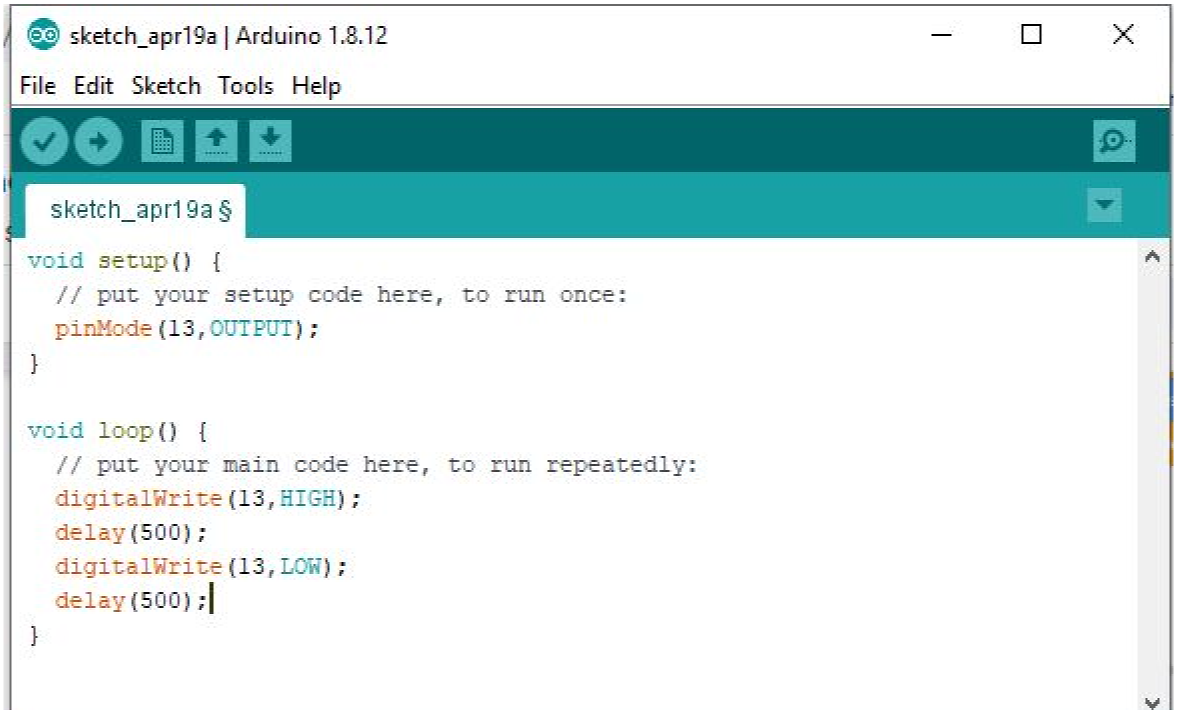
7. When your code compiles, upload it to the Arduino.
    a. TinkerCAD: Select the Code button in the upper right and switch from "Blocks" to "Text." Copy in your code from the Arduino IDE into the TinkerCAD code window. Then hit "Start Simulation" to run your code.



b. Hardware: Plug in your Arduino and then hit the Upload button that's right by the Verify button. Hitting the upload button will compile and then upload so you don't need to verify before uploading every time.

8. Confirm that your hardware is acting as you would expect. The LED should turn on and off repeatedly.

9. Play with the delays and check out the effect on your simulation or hardware. What's the smallest delay you can use and still tell that the LED is turning on and off?



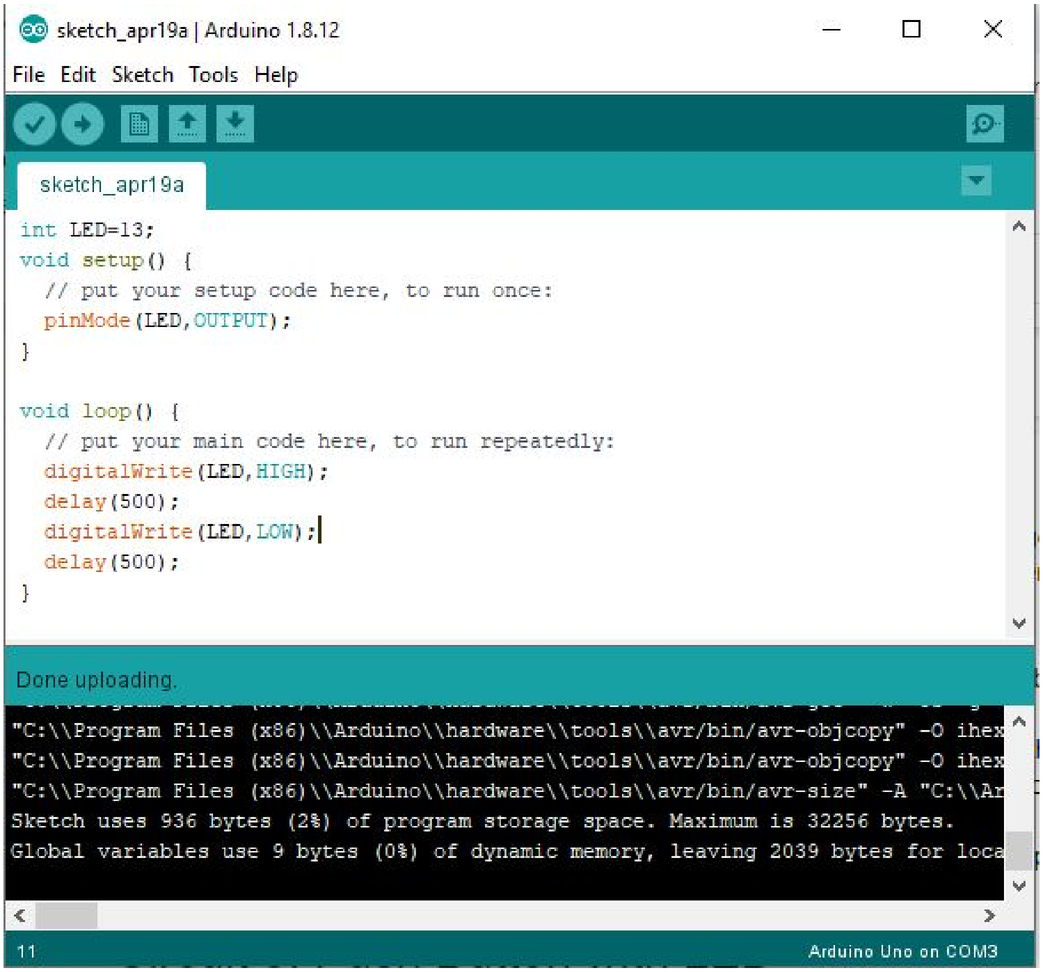
```
void setup() {
  // put your setup code here, to run once:
  pinMode(13,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(13,HIGH);
  delay(500);
  digitalWrite(13,LOW);
  delay(500);
}
```

Instead of using the default pin names, we can use variables to make pin usage more intuitive. This is helpful when you start to utilize a lot of pins and are trying to remember which is doing what in your code.

10. Add a variable called LED to represent digital pin 13. This code goes above the setup() function and the syntax will be "int LED=13;" In TinkerCAD you'll need to stop the simulation before you can modify the code.

11. Replace the instances of "13" with "LED" in your code. You can do this in either the TinkerCAD code window or the Arduino IDE since this is your own code that you're working with.

12. Upload and check that your hardware is reacting to the code as you'd expect.



```
int LED=13;
void setup() {
  // put your setup code here, to run once:
  pinMode(LED,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED,HIGH);
  delay(500);
  digitalWrite(LED,LOW);
  delay(500);
}
```

Done uploading.

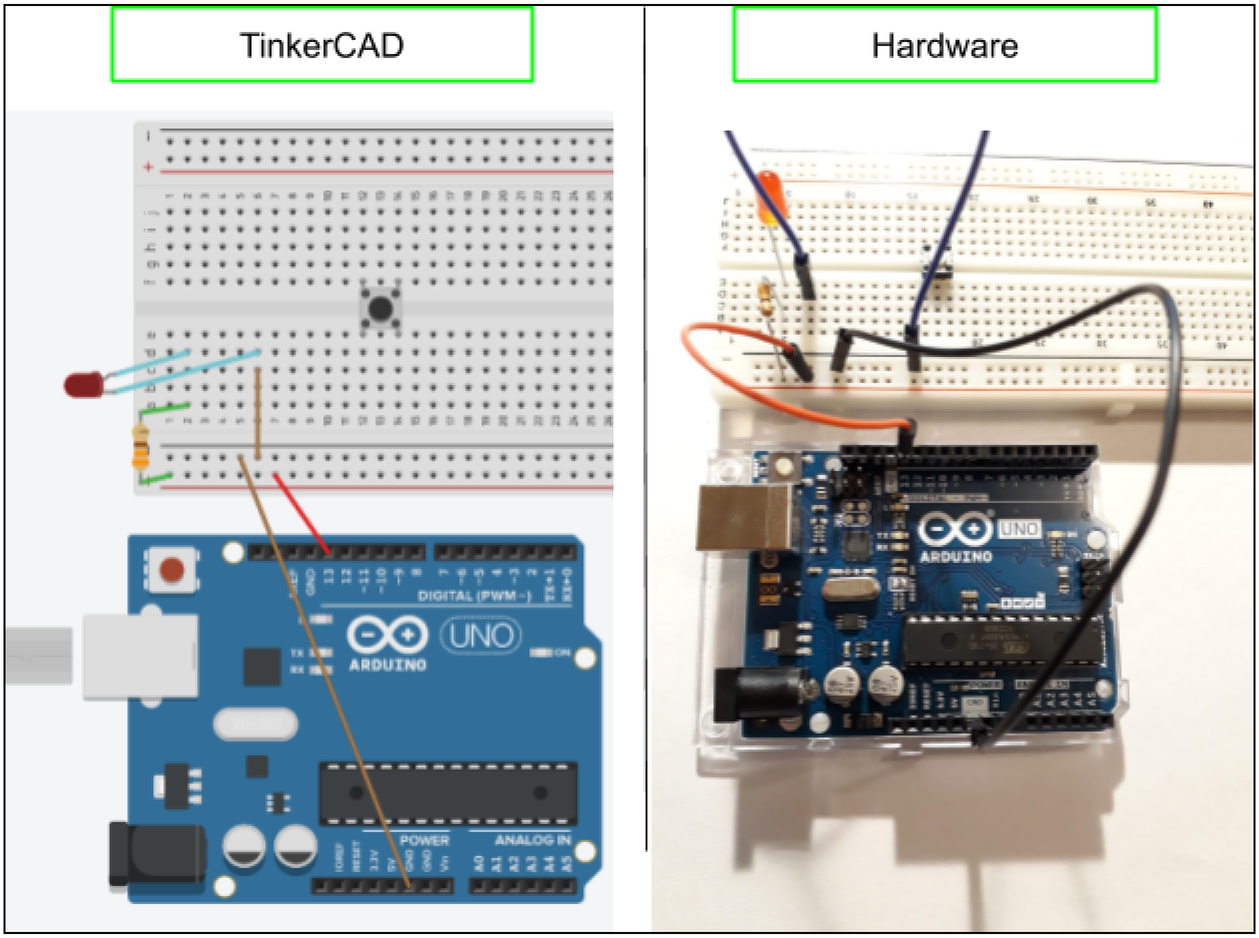
```
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-objcopy" -O ihex
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-objcopy" -O ihex
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-size" -A "C:\\Ar
Sketch uses 936 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for loca
```

Circuit 3: Push Button with LED

Next we're going to add a push button to our circuit so that when the button is pushed the LED turns on. This will require adding a little more breadboarding to our previous circuit and using some control structures in our code.

Build the Circuit:

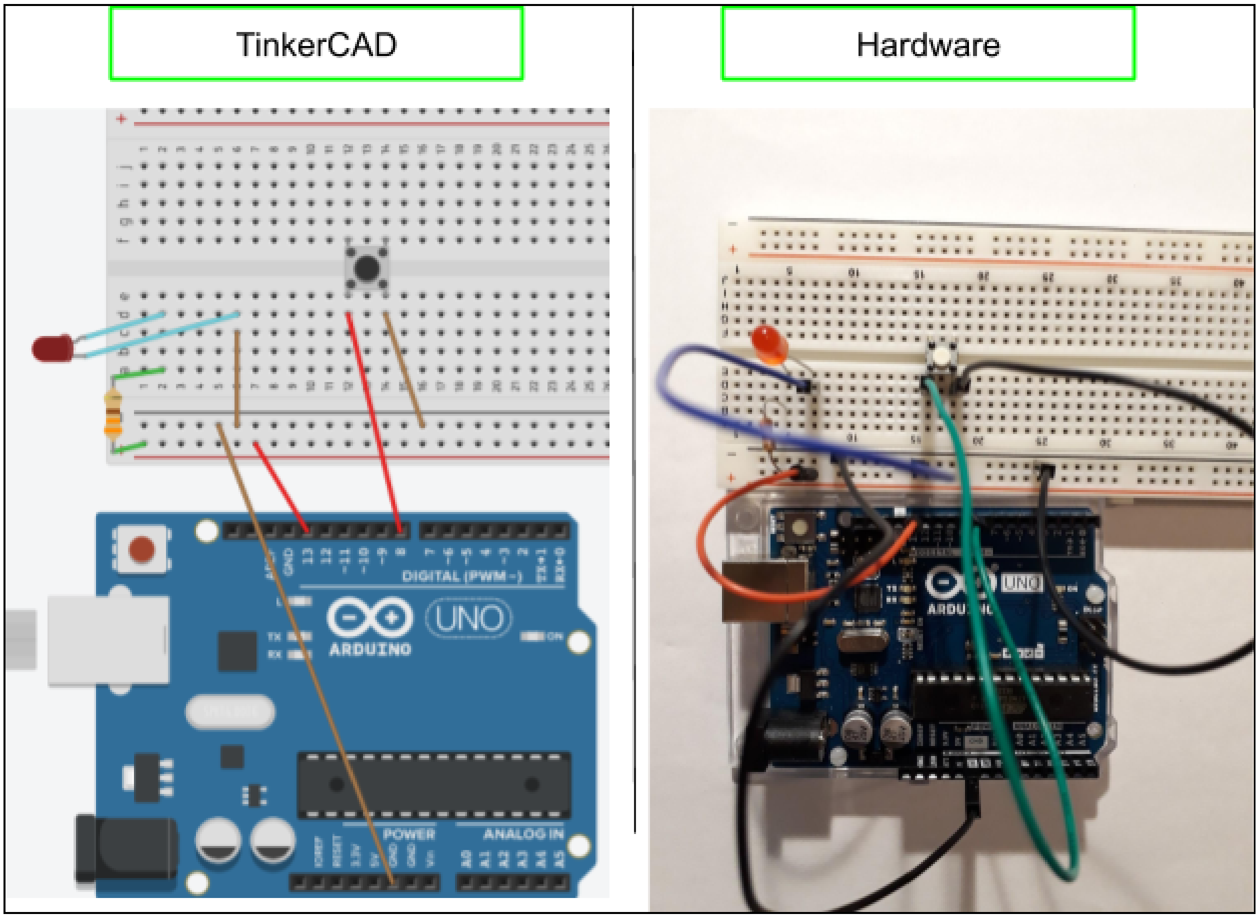
1. Leave your blinking LED circuit as is. If you're using hardware, unplug it before building the pushbutton part of the circuit. It's bad practice to work with live hardware. In TinkerCAD, you can simply stop the simulation.
    a. TinkerCAD: close the code window by pressing the Code button.
2. Add a pushbutton to the circuit. It's easiest to place it straddling the break in the breadboard.



TinkerCAD

Hardware

1. Connect the bottom left pin of the push button to digital pin 8 on the arduino (red wire in TinkerCAD). Connect the bottom right pin of the push button to the - power column on the breadboard (brown wire in TinkerCAD).



TinkerCAD

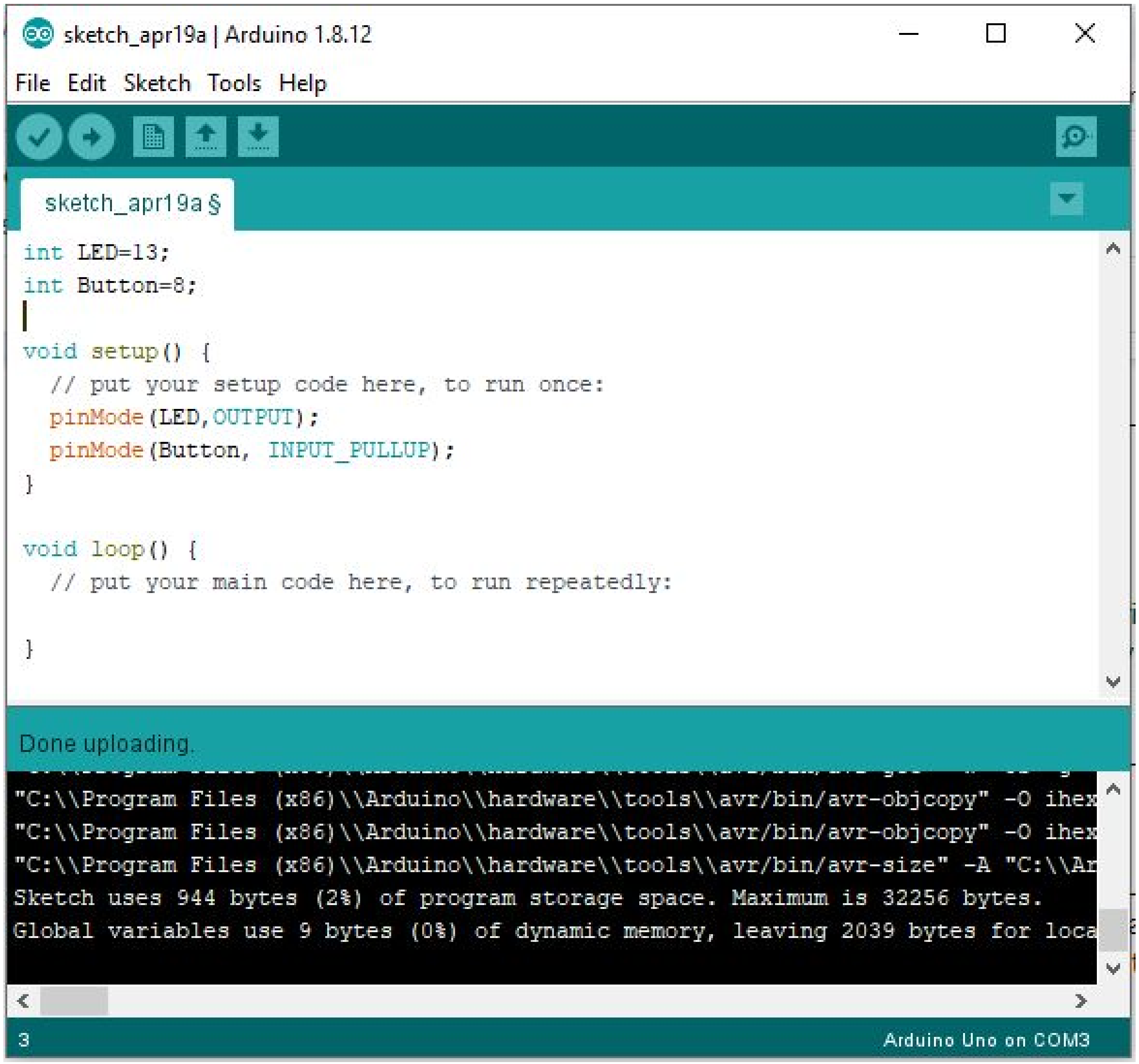
Hardware

Write the Code:

Since we're now using digital pin 8 to control the button, we need to define a variable for the button and declare its pin type in the setup() function. You can modify the code you wrote for the Blinking LED circuit or start a new sketch.

1. Add a variable called "Button" that's set to digital pin 8. You will also still need a variable called "LED" that's set to digital pin 13.

2. Set the LED pin to be an OUTPUT and the Button pin to be of type "INPUT_PULLUP". Push buttons are active low devices so when they are given a LOW signal they are on. By setting the pin type to INPUT_PULLUP, the pin is HIGH by default so that the button is off by default.



```
int LED=13;
int Button=8;

void setup() {
  // put your setup code here, to run once:
  pinMode(LED,OUTPUT);
  pinMode(Button, INPUT_PULLUP);
}

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

}
```

Done uploading.

```
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-objcopy" -O ihex
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-objcopy" -O ihex
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-size" -A "C:\\Ar
Sketch uses 944 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for loca
```

Control structures allow the code to branch or repeat based on conditions. Examples of control structures are if/elseif/else statement, do/while loops, and for loops.

For if/elseif/else statements we use conditions to determine what the code should do. If the condition is met then the code under it is executed.

```
if(condition)
{
        Then do this code
}
elseif(condition)
{
        Then do this code
}
else
{
        Do this code
}
```

Conditions are often equalities or inequalities and can utilize digitalRead or analogRead to access the value of a pin.

- Example: if(digitalRead(8)==LOW)
- Example: elseif(x<12)
- Example: if(analogRead(A2)>12)

For a do/while statement, we tell the code to run over and over again until a condition is met.

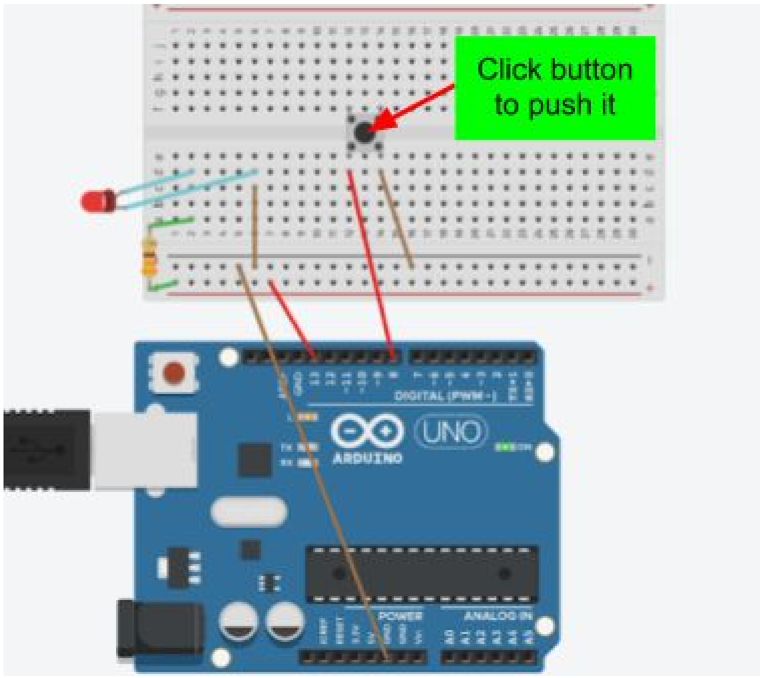
```
do
{
        This code
}while(condition)
```

A for loop is a control structure that allows for iteration, such as incemenenting to look through a data set or increasing speed one notch at a time.

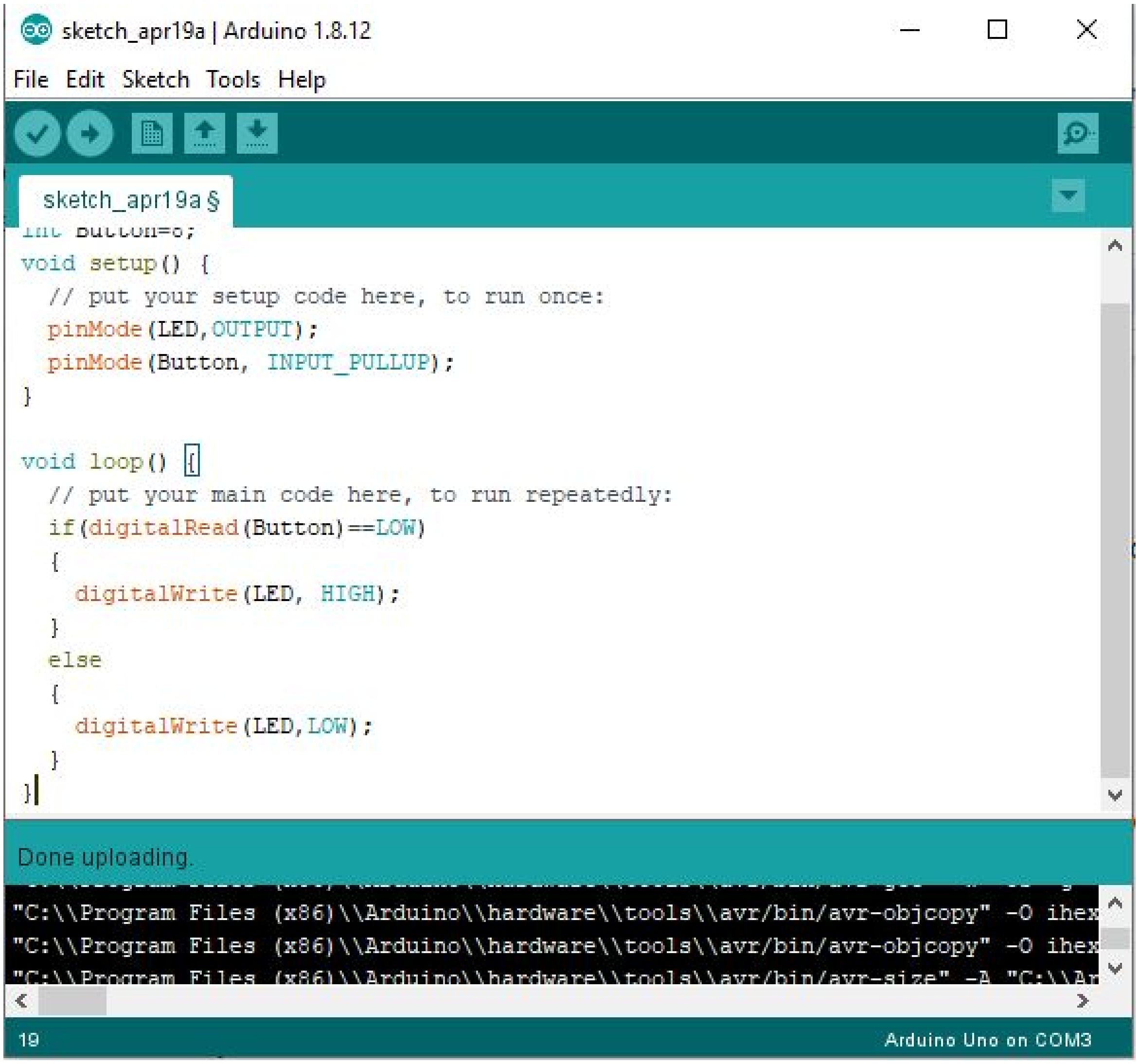
```
for (initialization; condition; increment)
{
         //statements
}
```

The initialization should include the data type (e.g., int i=0; i<5; i++)

3. In the loop() function, use an if/else statement so that if the button is pushed (digitalRead(Button)==LOW) then the LED is turned on, else then the LED is turned off. Use digitalWrite to turn the LED on and off.
4. Upload and test your code.
    a. TinkerCAD: click on the push button to press it. Holding down your left mouse button on the push button will hold it down.



Click button to push it



```
int Button=8;
void setup() {
  // put your setup code here, to run once:
  pinMode(LED,OUTPUT);
  pinMode(Button, INPUT_PULLUP);
}

void loop() {
  // put your main code here, to run repeatedly:
  if(digitalRead(Button)==LOW)
  {
    digitalWrite(LED, HIGH);
  }
  else
  {
    digitalWrite(LED,LOW);
  }
}
```

Done uploading.

```
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-objcopy" -O ihex
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-objcopy" -O ihex
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-size" -A "C:\\Ar
```

Circuit 4: Servo Motor

Next, we're going to move to working with servos. This circuit won't require a breadboard so you can either start a new circuit in TinkerCAD or disconnect your Arduino from the breadboard. Select and then hit the delete key on your keyboard to delete a wire or component in TinkerCAD.

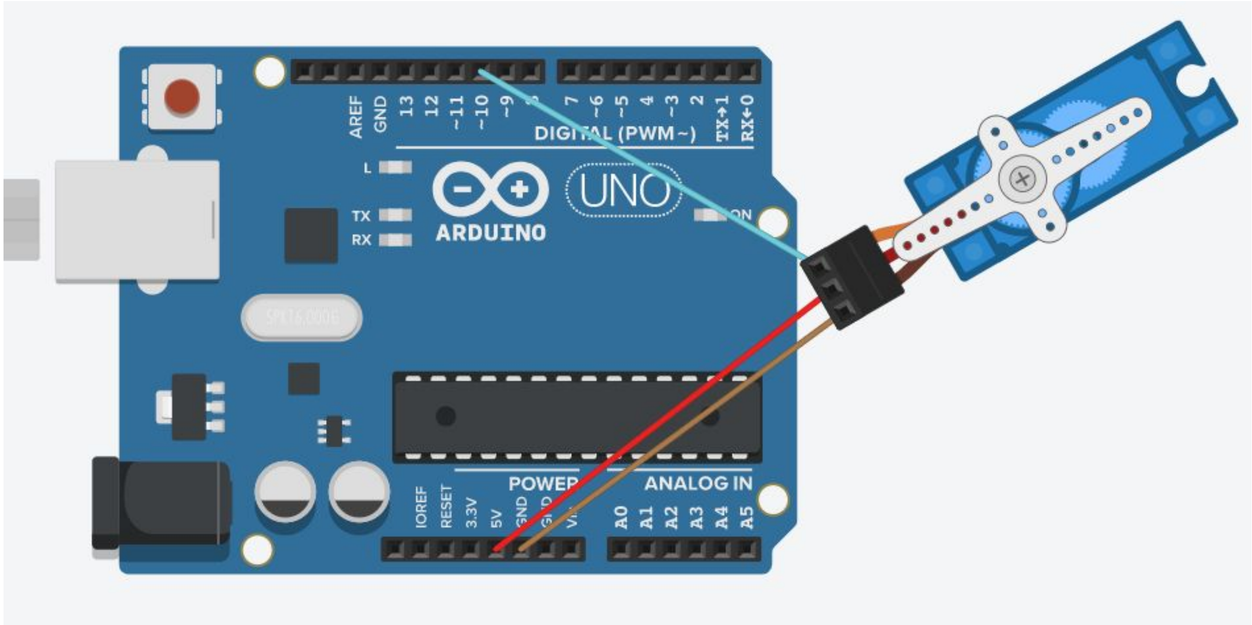
Servos are rotary actuators with position control. Through code you can set their angle between 0° and 180°. Note that they are not capable of rotating continuously.

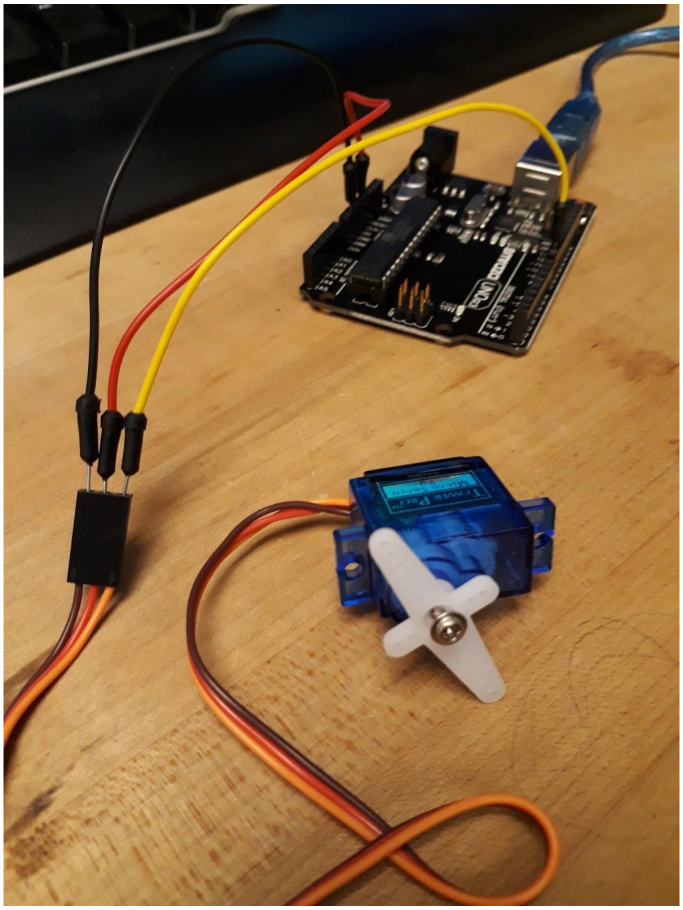
Building the Circuit

1. Wire the Arduino to the servo.
    a. TinkerCAD: We're going to use the micro servo in the components window.
    b. Wire designations/connections:
        i. Brown: ground
        ii. Red: 5V

iii. Orange: signal - this should go to a pulse width modulation (PWM) pin on the Arduino which is designated with a ‘~’. Connect it to digital pin 11.

   1. Note: on some servos the signal line is white rather than orange.

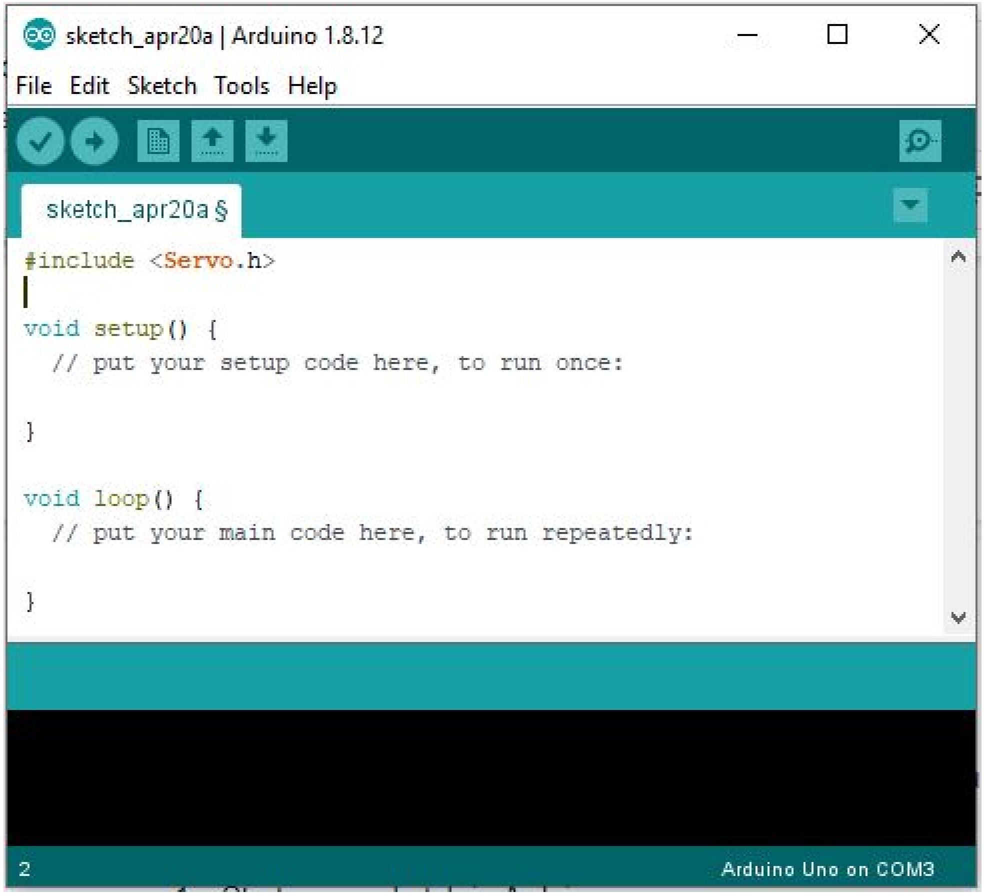




Write the Code

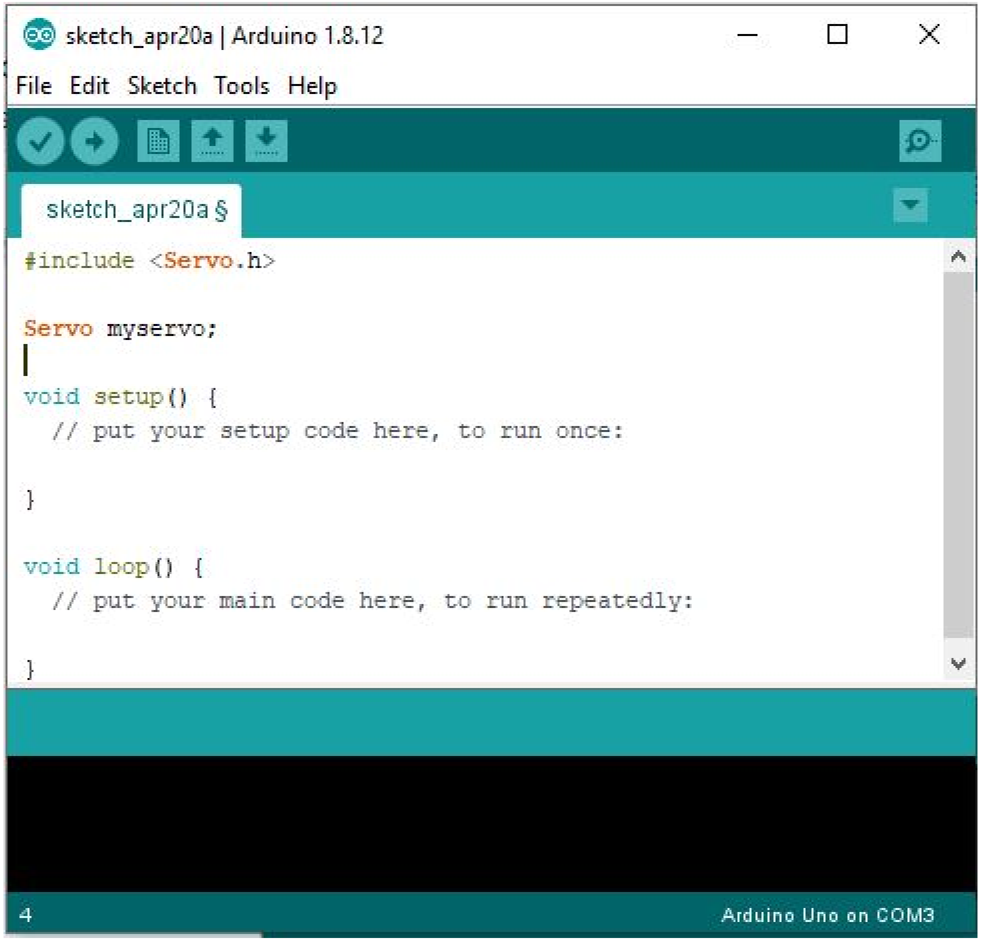
For the code, we’re going to use the servo library included in Arduino. This will provide us with specialized functions that allow us to easily control the servo.

1. Start a new sketch in Arduino
2. Include the servo library using this syntax: “#include <Servo.h>”
    a. This statement goes at the top of the code before the setup() function.



```
#include <Servo.h>

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

}

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

}
```

3. Create a servo object in your code. We’ll be able to then access the properties of this object to set and read angles. The syntax is “Servo myservo;” and it goes below the inclusion of the servo library. “myservo” is a chosen variable name so you can set it to whatever you would like.



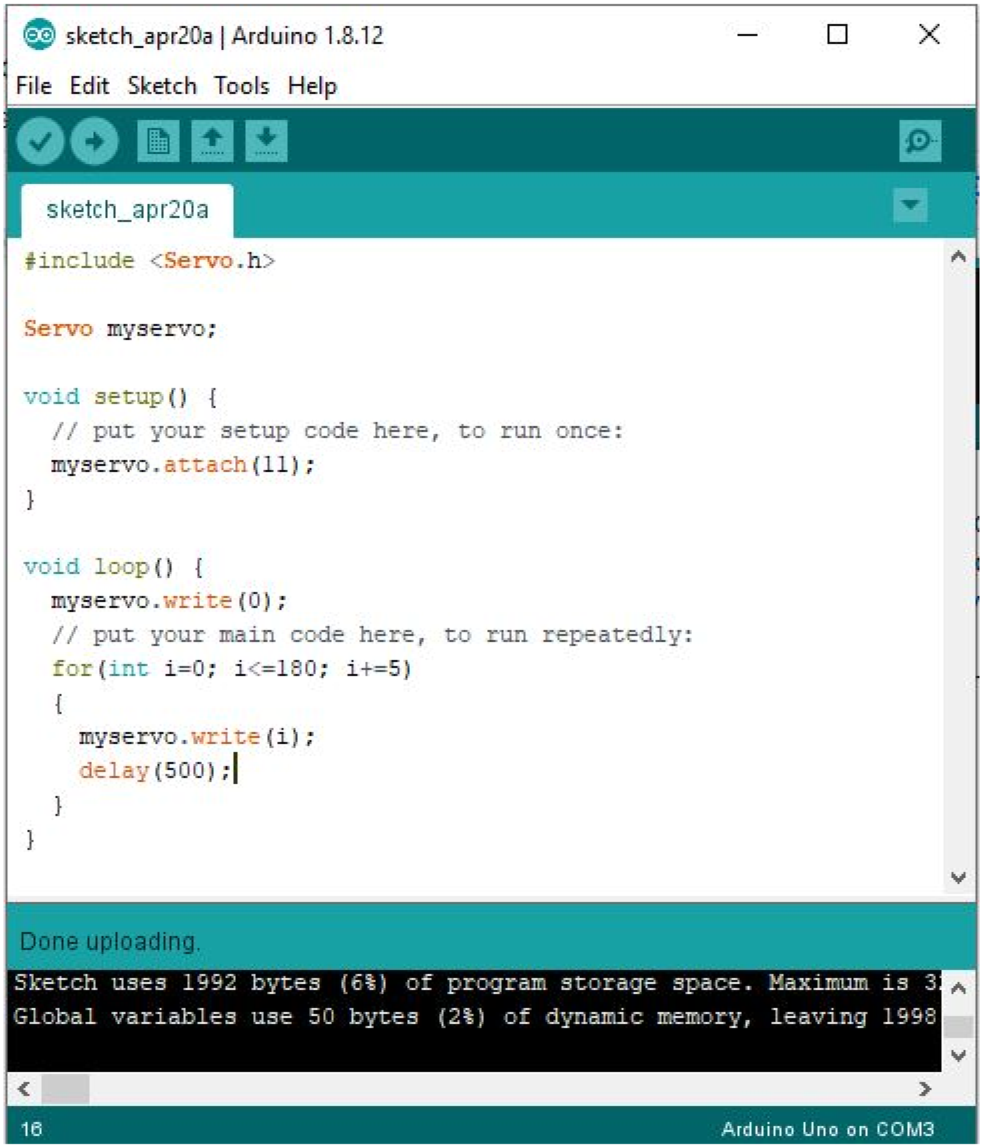
```
#include <Servo.h>

Servo myservo;

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

}

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

}
```

4. In the setup, instead of using pinMode, we're going to attach the servo to a PWM pin. In the setup function, use the following syntax: "myservo.attach(11);"

5. In the loop function, start by setting the position of the servo to 0. The syntax to do so is "myservo.write(0);" This will also bring the servo back to the 0° position at the start of every execution of the loop function.

6. Make the servo increment angles from 0° to 180° in 5 degree increments by using a for loop.
   a. For loop initialization: int i=0;
   b. For loop condition: i<=180;
   c. For loop increment: i+=5
   d. For loop body: myservo.write(i); delay(500);
      i. A delay is used so that you can see the increments in angle.

7. Upload the code and see if the servo moves as expected.



```
#include <Servo.h>

Servo myservo;

void setup() {
  // put your setup code here, to run once:
  myservo.attach(11);
}

void loop() {
  myservo.write(0);
  // put your main code here, to run repeatedly:
  for(int i=0; i<=180; i+=5)
  {
    myservo.write(i);
    delay(500);
  }
}
```

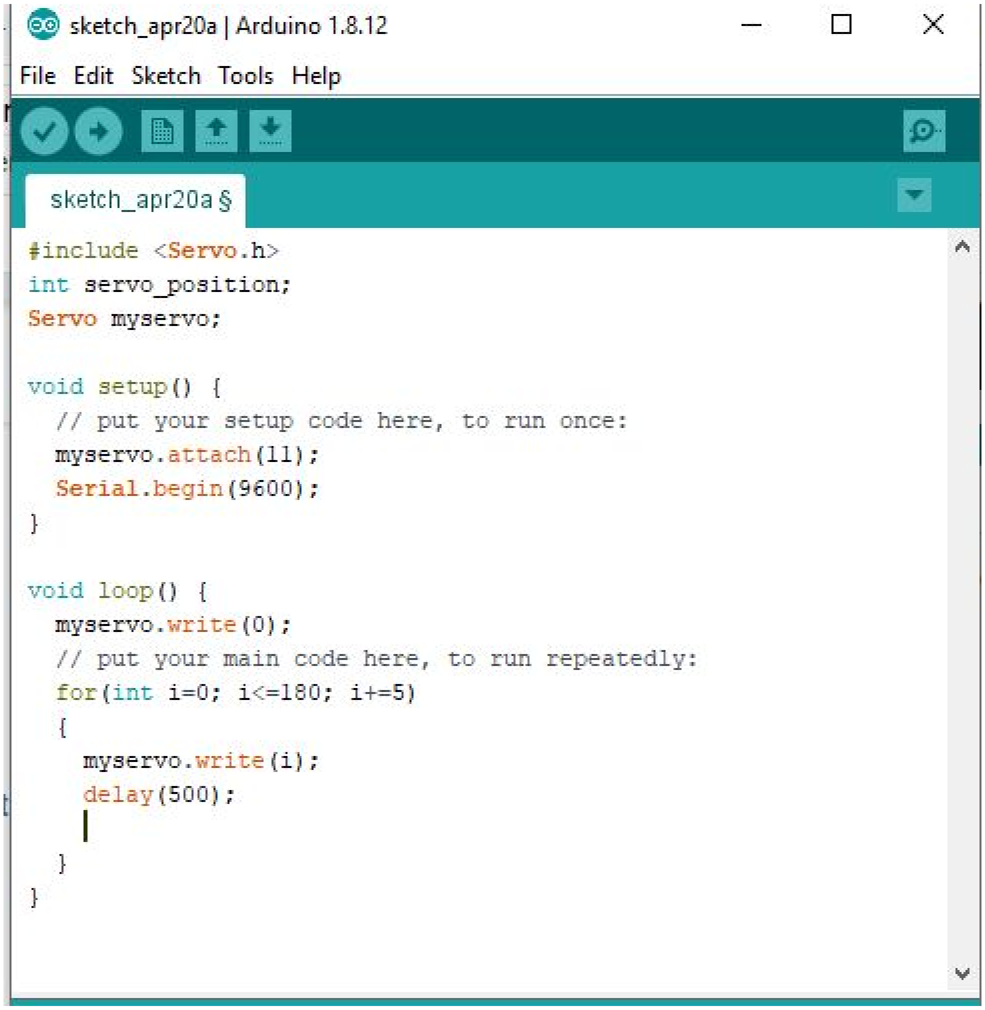
Done uploading.

```
Sketch uses 1992 bytes (6%) of program storage space. Maximum is 3
Global variables use 50 bytes (2%) of dynamic memory, leaving 1998
```

Now, we’re going to add a little more code so that the position of the servo prints to the serial monitor.

8. Add a global variable of type int called servo_position.
   a. Note: when we created variables for the pins (LED and Button) previously, those were global variables. This one goes in the same spot.

9. In the setup function add “Serial.begin(9600);” This will allow you to use the serial port. 9600 is the baud rate, which is the rate at which info is transferred to a communications channel. Having a baud rate of 9600 means that the maximum rate of info transfer is 9600 bits per second.

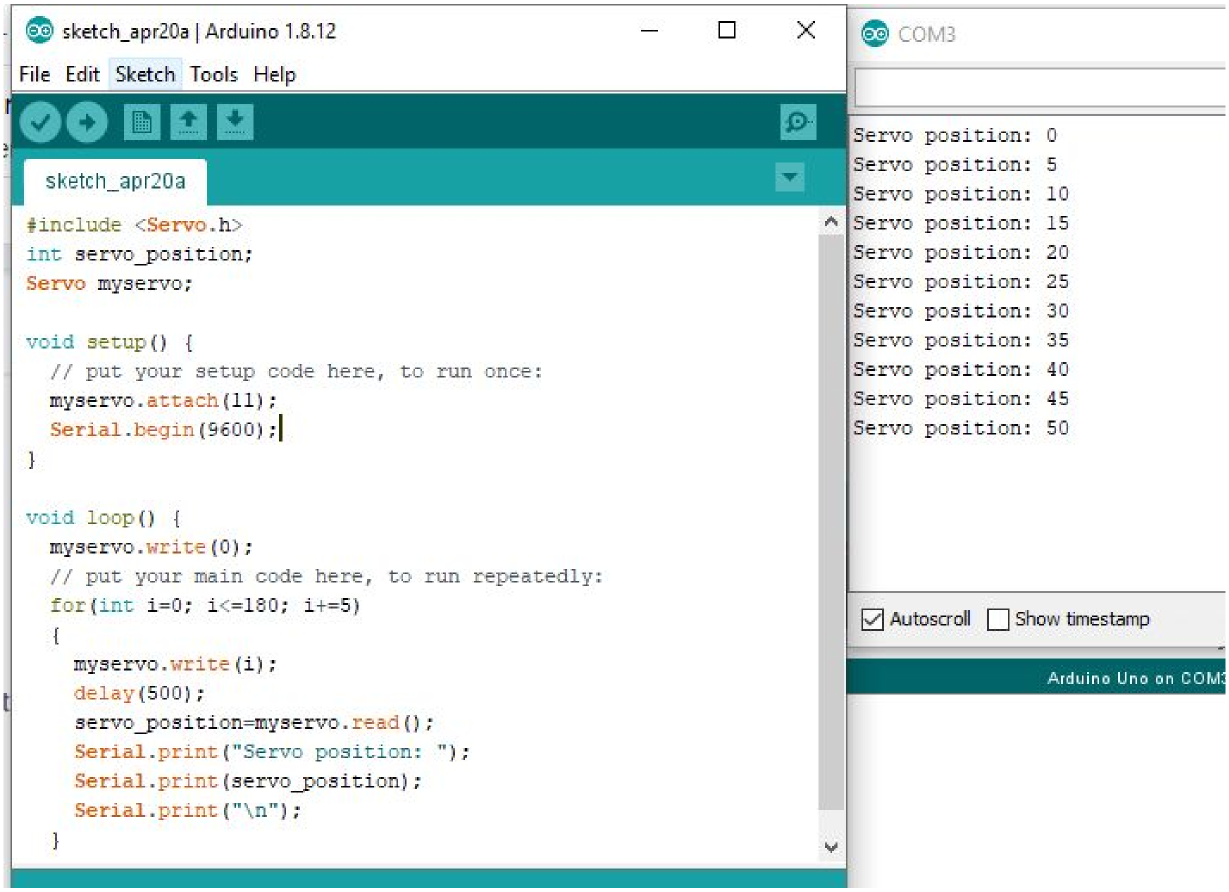


```
#include <Servo.h>
int servo_position;
Servo myservo;

void setup() {
  // put your setup code here, to run once:
  myservo.attach(11);
  Serial.begin(9600);
}

void loop() {
  myservo.write(0);
  // put your main code here, to run repeatedly:
  for(int i=0; i<=180; i+=5)
  {
    myservo.write(i);
    delay(500);
    
  }
}
```

10. In the for loop set “servo_position” to myservo.read(). This should go below the delay.

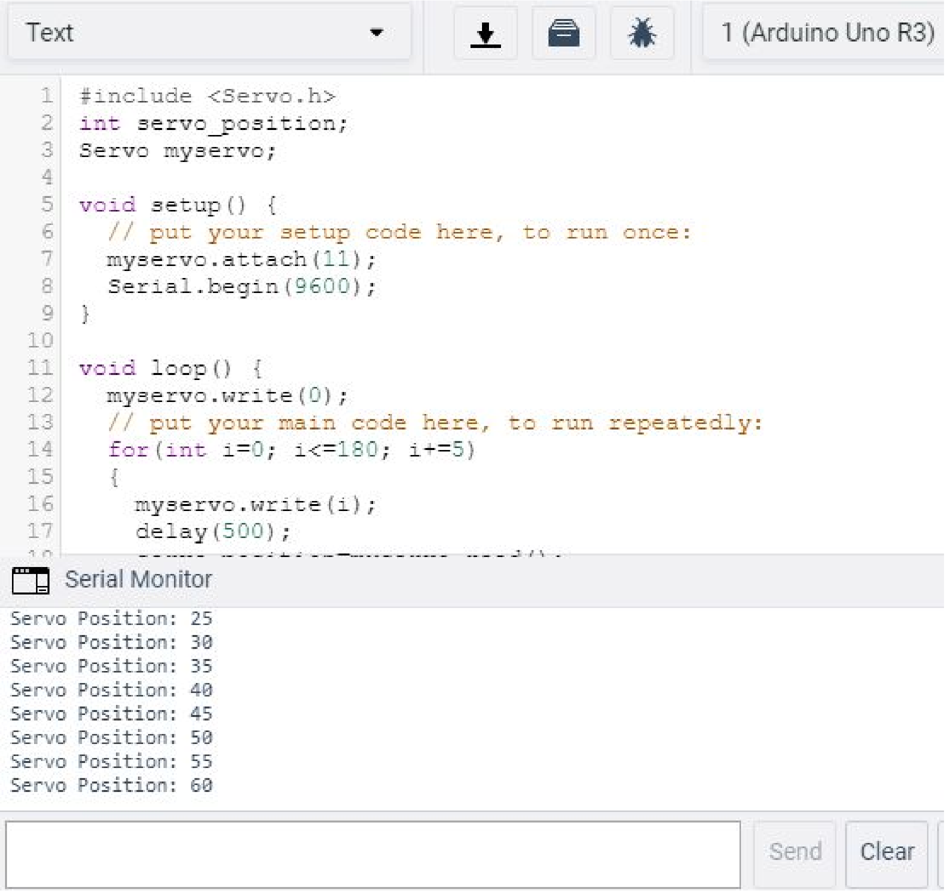
11. Use Serial.print to print the value of servo_position. Refer to the examples in the Function definitions above.
    a. “Serial.print(servo_position);” will print the positions one after another in the serial monitor like words on a page. If you want each position printed on a newline, you need to include a second print command: Serial.print(“\n”);

12. Upload the code and check the serial monitor:
    a. TinkerCAD: The serial monitor button is in the bottom left of the code window. Click this button to view the output.
    b. Hardware: In the Arduino IDE, the serial monitor is found under the Tools menu.


sketch_apr20a | Arduino 1.8.12
File Edit Sketch Tools Help
sketch_apr20a
COM3
☑ Autoscroll ☐ Show timestamp
Arduino Uno on COM3

```
#include <Servo.h>
int servo_position;
Servo myservo;

void setup() {
  // put your setup code here, to run once:
  myservo.attach(11);
  Serial.begin(9600);
}

void loop() {
  myservo.write(0);
  // put your main code here, to run repeatedly:
  for(int i=0; i<=180; i+=5)
  {
    myservo.write(i);
    delay(500);
    servo_position=myservo.read();
    Serial.print("Servo position: ");
    Serial.print(servo_position);
    Serial.print("\n");
  }
```

```
Servo position: 0
Servo position: 5
Servo position: 10
Servo position: 15
Servo position: 20
Servo position: 25
Servo position: 30
Servo position: 35
Servo position: 40
Servo position: 45
Servo position: 50
```



```
#include <Servo.h>
int servo_position;
Servo myservo;

void setup() {
  // put your setup code here, to run once:
  myservo.attach(11);
  Serial.begin(9600);
}

void loop() {
  myservo.write(0);
  // put your main code here, to run repeatedly:
  for(int i=0; i<=180; i+=5)
  {
    myservo.write(i);
    delay(500);
```

```
Servo Position: 25
Servo Position: 30
Servo Position: 35
Servo Position: 40
Servo Position: 45
Servo Position: 50
Servo Position: 55
Servo Position: 60
```

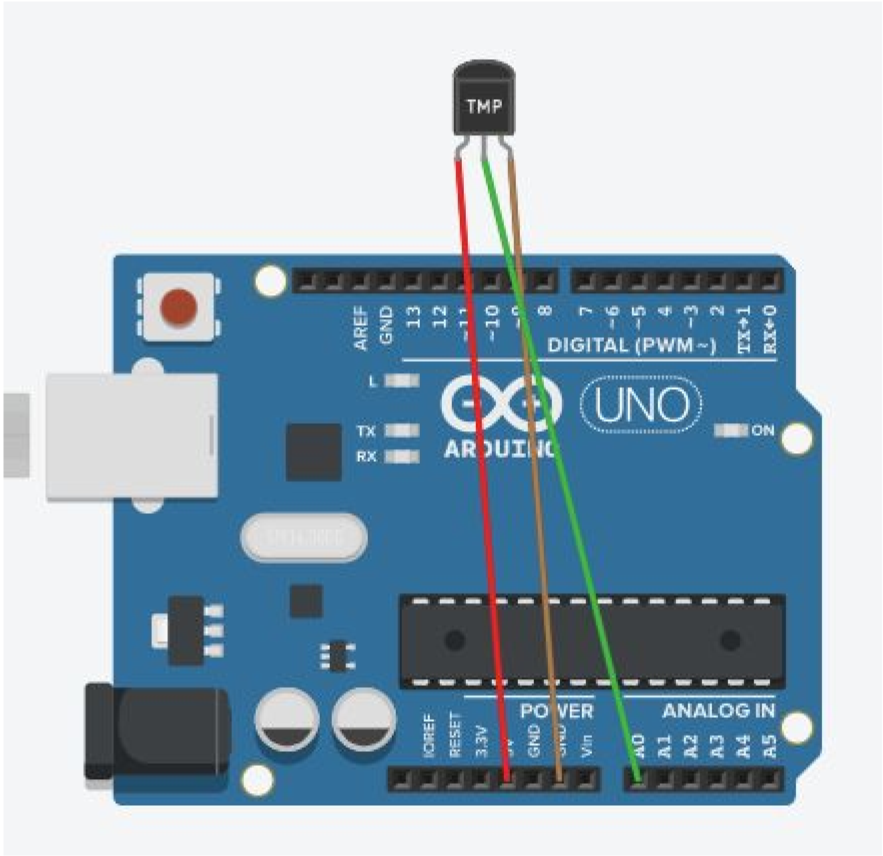
Circuit 5: Temperature/Humidity Sensor

The last circuit we're going to work with is a temperature/humidity sensor setup. Unfortunately, TinkerCAD doesn't have a ton of environmental sensor options so we're going to use the TMP36 temperature sensor in TinkerCAD and the DHT11 temperature/humidity sensor for hardware.

TMP36 Sensor - TinkerCAD Circuit

The TMP36 sensor will provide an analog output that is related to the temperature.

1. Start a new TinkerCAD circuit. Place an Arduino Uno and a TMP36 sensor into the canvas. Connect the legs of the sensor to pins on the Arduino. Power will go to 5V, ground to ground, and Vout will go to an analog pin - A0.



2. Create the following global variables:
    a. int sensorPin=A0;
    b. int reading;
    c. float voltage;
    d. float tempC;

3. In the setup, use Serial.begin(9600) to set the baud rate.

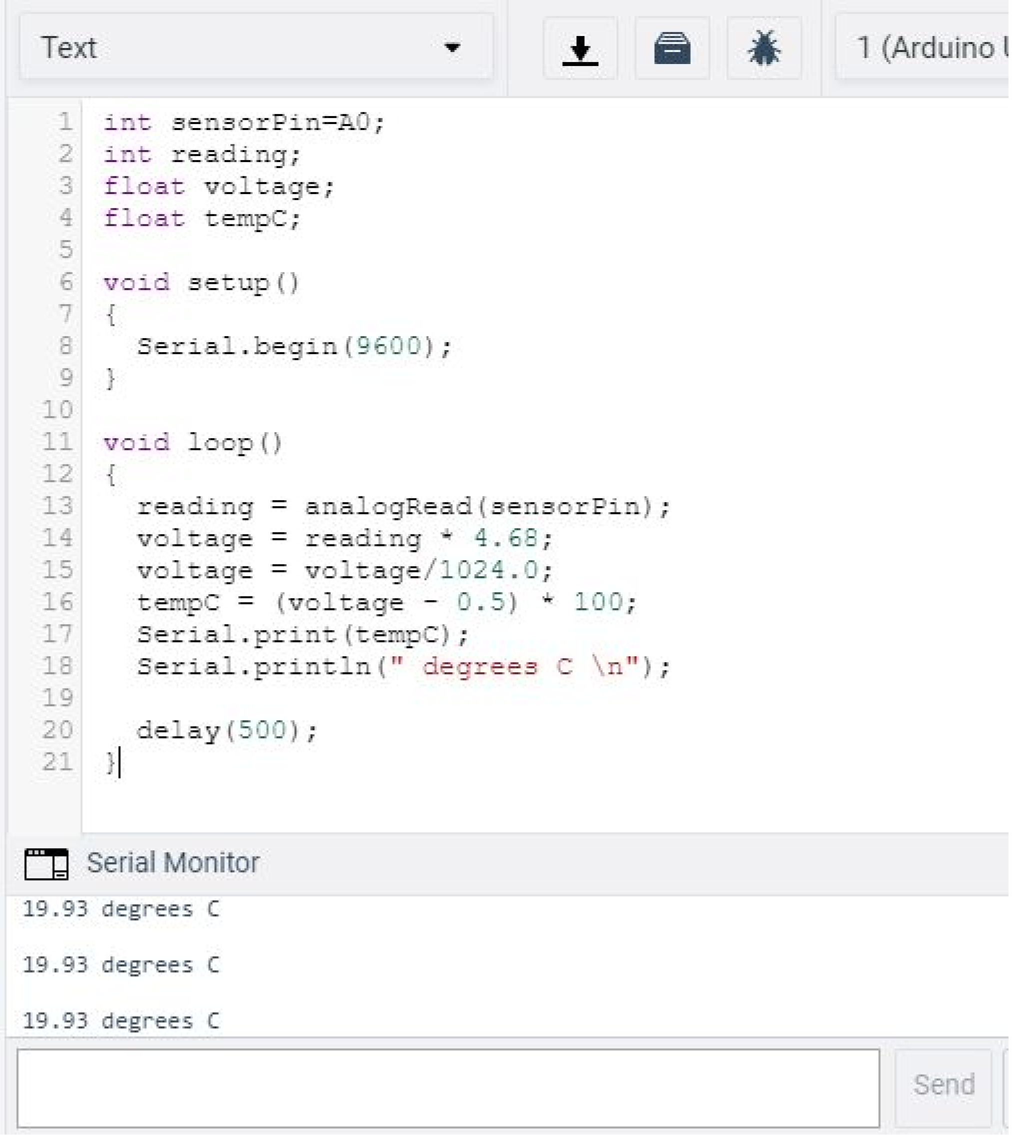
4. In the loop function, set the global variable “reading” to the value seen on pin A0 using analogRead().

5. Use the following formula to convert the value in the variable “reading” to a temperature in celsius. You can break the forumla up into multiple lines with intermediate varialbes like “voltage”

$$tempC = ((\frac{reading*4.68}{1024}) - 0.5) * 100$$

6. Print the temperature to the serial monitor and then include a 0.5 second delay to create a pause between sensor readings.

7. Upload the code and look at the values on the Serial monitor.

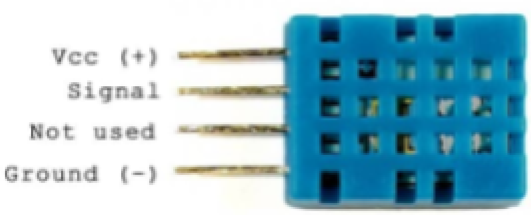


```
int sensorPin=A0;
int reading;
float voltage;
float tempC;

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

void loop()
{
  reading = analogRead(sensorPin);
  voltage = reading * 4.68;
  voltage = voltage/1024.0;
  tempC = (voltage - 0.5) * 100;
  Serial.print(tempC);
  Serial.println(" degrees C \n");

  delay(500);
}
```

```
19.93 degrees C

19.93 degrees C

19.93 degrees C
```

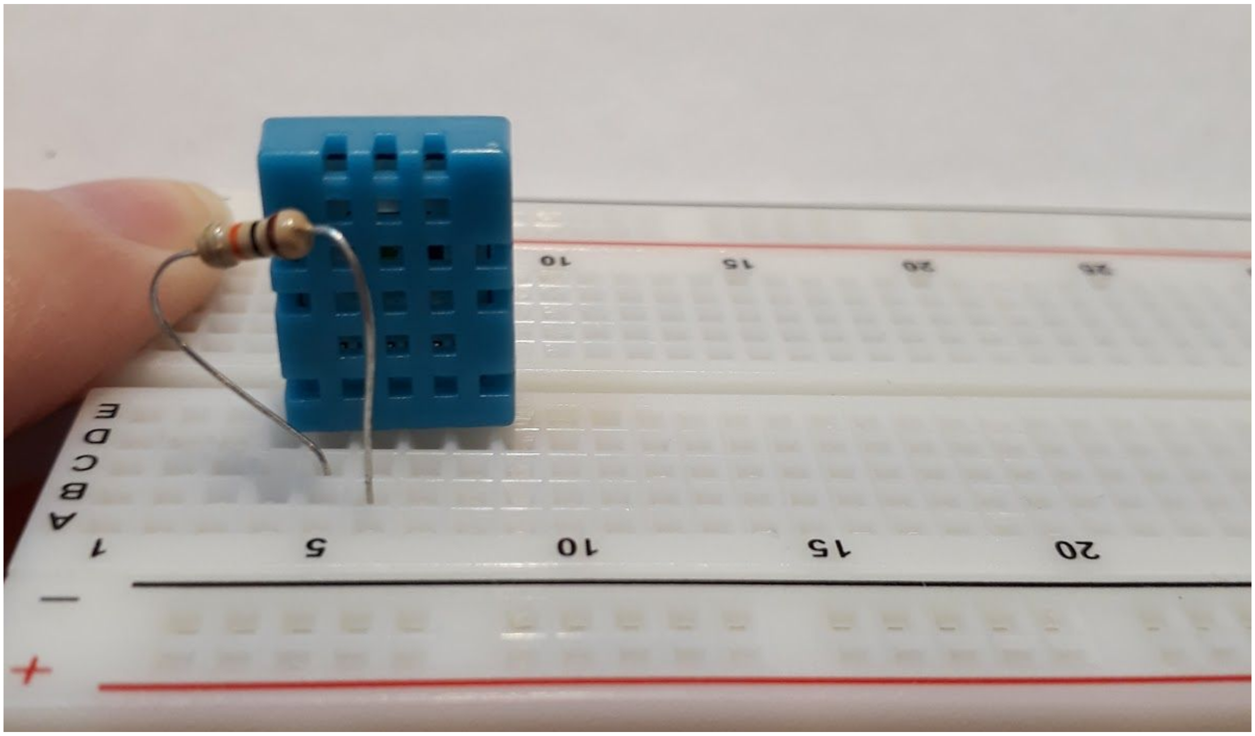
DHT11 Sensor: Hardware

The DHT11 sensor comes in two common configurations: a 4 pin version like the one shown below and a 3 pin version that's mounted on a small PCB. The PCB mounted version includes a 10kΩ pull up resistor. When using the 4 pin version, this resistor just has to be included in the circuit.

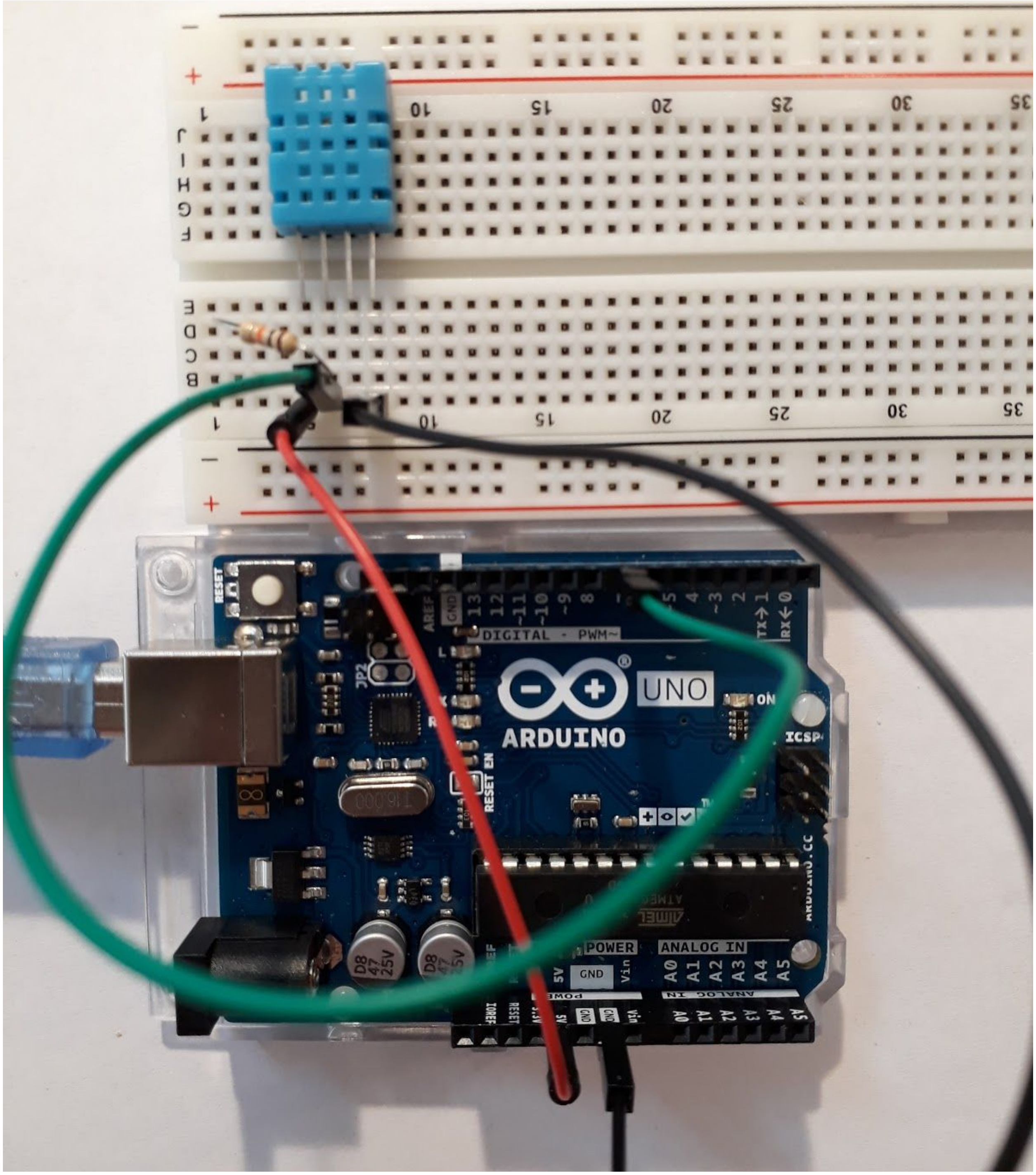


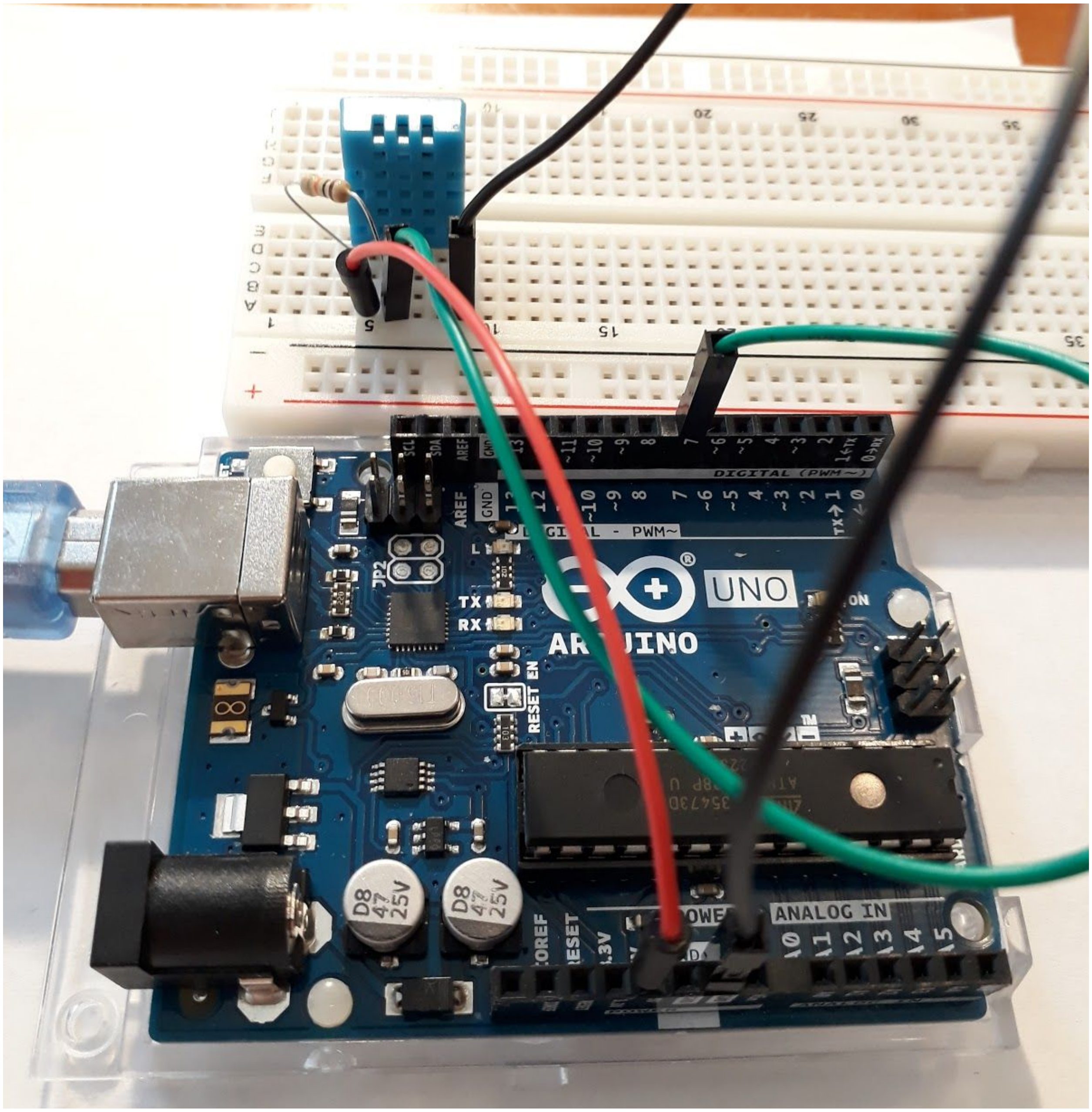
1. Place the sensor in a breadboard so that each pin is in a different row.

2. Connect the 10kΩ resistor between the Vcc and Signal pin rows on the breadboard.

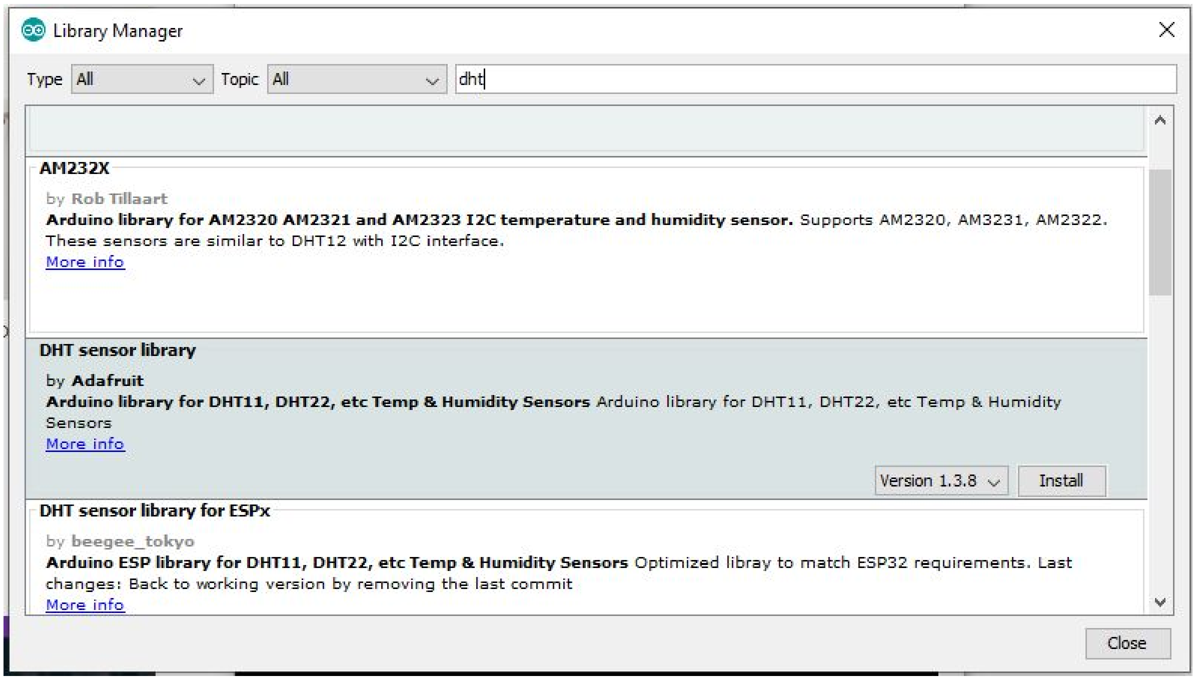


3. Connect the VCC sensor pin to the 5V pin on the Arduino. Connect the ground sensor pin to ground on the Arduino. Connect the signal sensor pin to digital pin 7 on the Arduino.


ARDUINO
UNO
DIGITAL - PWM~
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
RESET
ON
ICSP
RESET EN
POWER
ANALOG IN
5V
GND
Vin
A0
A1
A2
A3
A4
A5
1
10
15
20
25
30
35
J
I
H
G
F
E
D
C
B
+
-



4. Start a new sketch in Arduino. If you don't already have the dht.h library installed, go to Library Manager through Tools - manage libraries in the top menu. Search dht and install the DHT sensor library by Adafruit.



Library Manager

Type All Topic All dht

AM232X

by **Rob Tillaart**

Arduino library for AM2320 AM2321 and AM2323 I2C temperature and humidity sensor. Supports AM2320, AM3231, AM2322. These sensors are similar to DHT12 with I2C interface.

More info

DHT sensor library

by **Adafruit**

Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors

More info

Version 1.3.8 Install

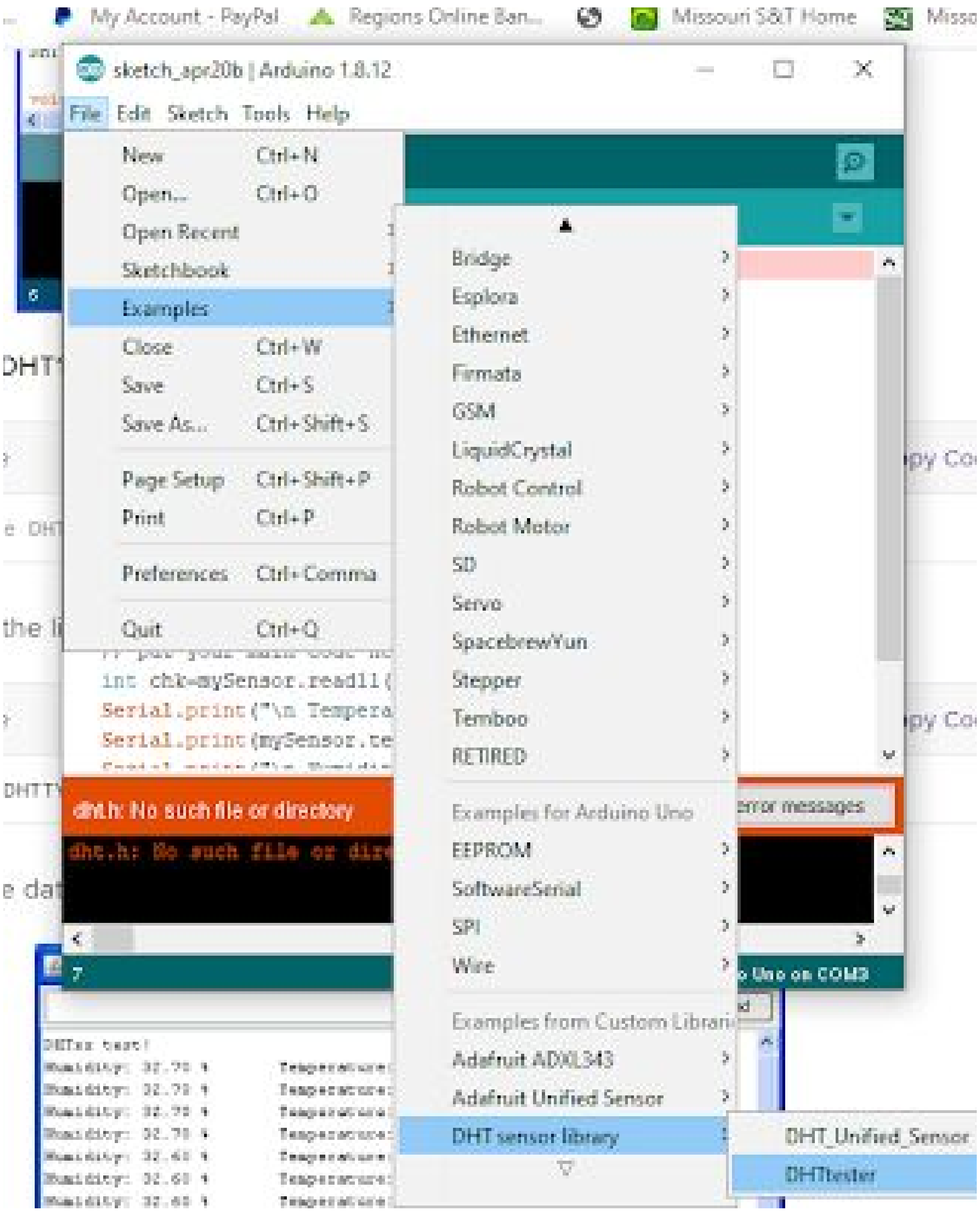
DHT sensor library for ESPx

by **beegee_tokyo**

Arduino ESP library for DHT11, DHT22, etc Temp & Humidity Sensors Optimized libray to match ESP32 requirements. Last changes: Back to working version by removing the last commit

More info

5. Open the DHT tester example sketch after installing the Adafruit DHT library.



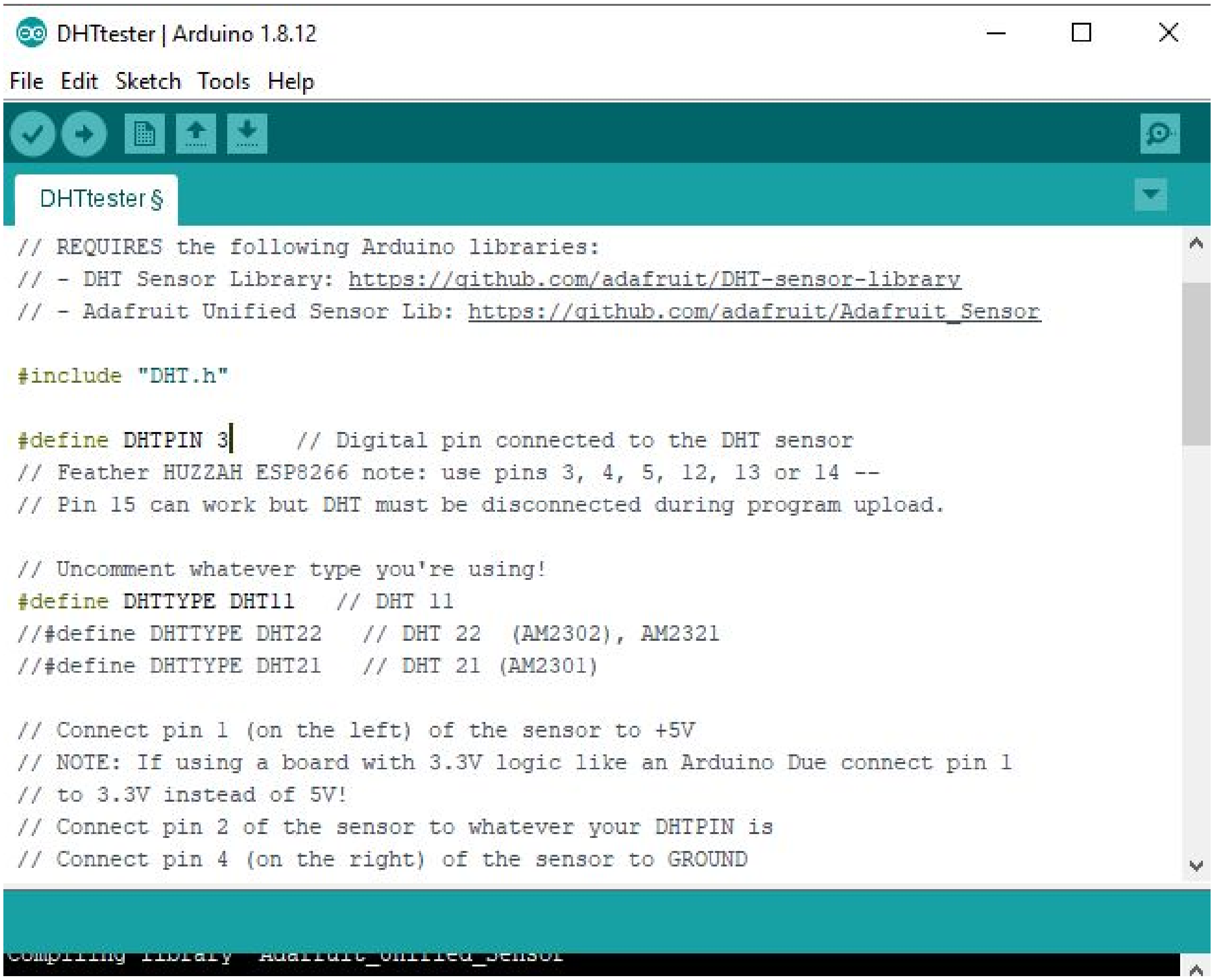
```
int chk=mySensor.read11(
Serial.print("\n Tempera
Serial.print(mySensor.te
```

```
dht.h: No such file or directory
dht.h: No such file or dire
```

```
DHTxx test!
Humidity: 32.70 %        Temperature:
Humidity: 32.70 %        Temperature:
Humidity: 32.70 %        Temperature:
Humidity: 32.70 %        Temperature:
Humidity: 32.60 %        Temperature:
Humidity: 32.60 %        Temperature:
Humidity: 32.60 %        Temperature:
```

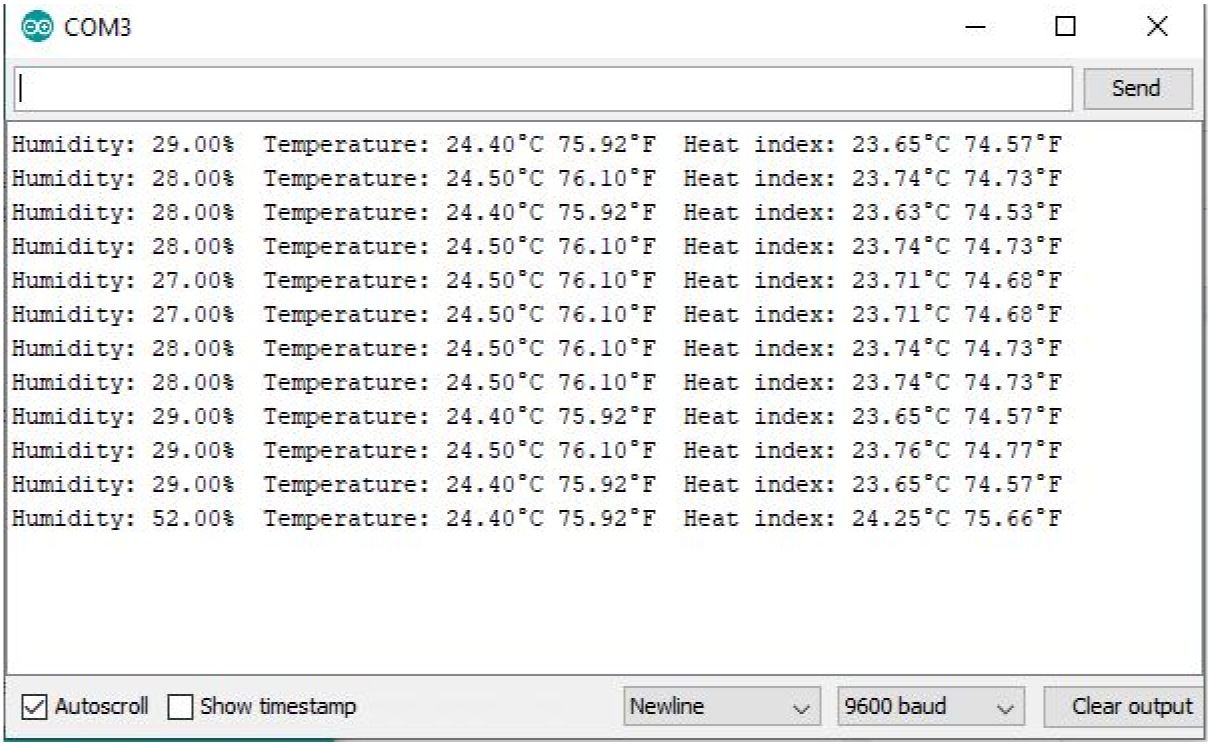
6. Change the pin and sensor type in the example sketch so that digital pin 7 is declared as the signal pin of the sensor and the DHT11 sensor is selected.

7. Upload the code and check the output on the serial monitor. An easy way to check if the sensor is reacting is to breathe on the sensor. The humidity and temperature should rise when you do this.



```
// REQUIRES the following Arduino libraries:
// - DHT Sensor Library: https://github.com/adafruit/DHT-sensor-library
// - Adafruit Unified Sensor Lib: https://github.com/adafruit/Adafruit_Sensor

#include "DHT.h"

#define DHTPIN 3     // Digital pin connected to the DHT sensor
// Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --
// Pin 15 can work but DHT must be disconnected during program upload.

// Uncomment whatever type you're using!
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
```

```
Compiling library "Adafruit_Unified_Sensor"
```



```
Humidity: 29.00%  Temperature: 24.40°C 75.92°F  Heat index: 23.65°C 74.57°F
Humidity: 28.00%  Temperature: 24.50°C 76.10°F  Heat index: 23.74°C 74.73°F
Humidity: 28.00%  Temperature: 24.40°C 75.92°F  Heat index: 23.63°C 74.53°F
Humidity: 28.00%  Temperature: 24.50°C 76.10°F  Heat index: 23.74°C 74.73°F
Humidity: 27.00%  Temperature: 24.50°C 76.10°F  Heat index: 23.71°C 74.68°F
Humidity: 27.00%  Temperature: 24.50°C 76.10°F  Heat index: 23.71°C 74.68°F
Humidity: 28.00%  Temperature: 24.50°C 76.10°F  Heat index: 23.74°C 74.73°F
Humidity: 28.00%  Temperature: 24.50°C 76.10°F  Heat index: 23.74°C 74.73°F
Humidity: 29.00%  Temperature: 24.40°C 75.92°F  Heat index: 23.65°C 74.57°F
Humidity: 29.00%  Temperature: 24.50°C 76.10°F  Heat index: 23.76°C 74.77°F
Humidity: 29.00%  Temperature: 24.40°C 75.92°F  Heat index: 23.65°C 74.57°F
Humidity: 52.00%  Temperature: 24.40°C 75.92°F  Heat index: 24.25°C 75.66°F
```

☑ Autoscroll □ Show timestamp Newline 9600 baud