LEARNING DOCUMENT FOR MRUDHU ELECTRONICS ARDUINO UNO R3 KIT

Chapter 1 - Blink

Description:

Turn an LED on and off every second.

This example shows the simplest thing you can do with an Arduino to see physical output: it blinks the on-board LED.

Hardware Required

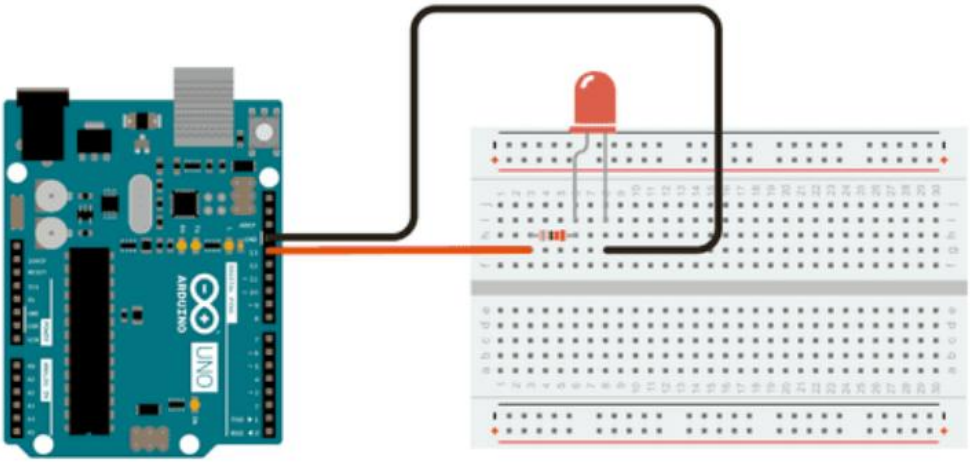
- Arduino Board
- LED (optional)
- 220 ohm resistor (optional)

Circuit:

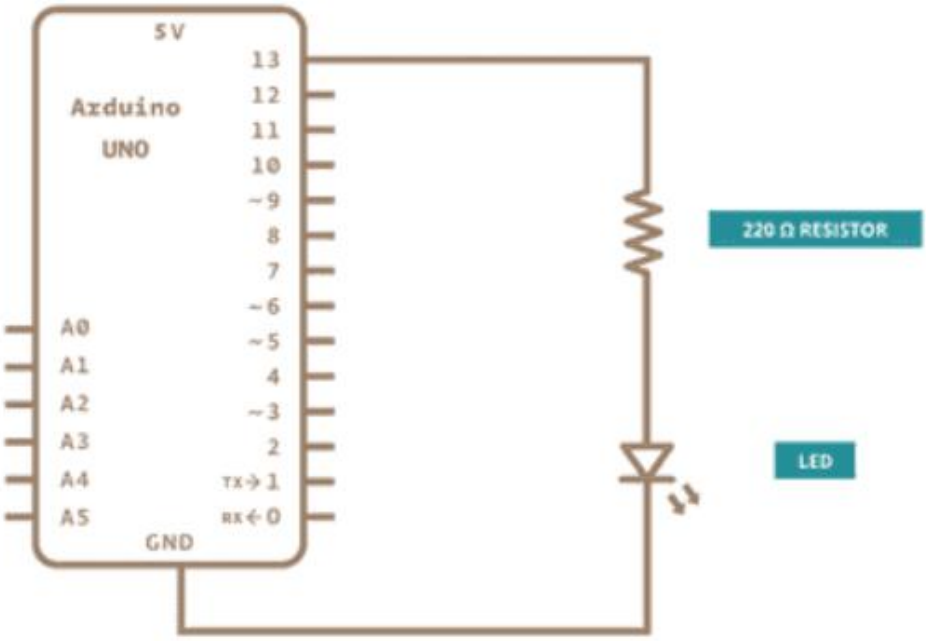
This example uses the built-in LED that most Arduino boards have. This LED is connected to a digital pin and its number may vary from board type to board type. To make your life easier, we have a constant that is specified in every board descriptor file. This constant is LED_BUILTIN and allows you to control the built-in LED easily.
Pin -- D13
If you want to light an external LED with this sketch, you need to build this circuit, where you connect one end of the resistor to the digital pin correspondent to the LED_BUILTIN constant. Connect the long leg of the LED (the positive leg, called the anode) to the other end of the resistor. Connect the short leg of the LED (the negative leg, called the cathode) to the GND.
In the diagram below we show an UNO board that has D13 as the LED_BUILTIN value.

The resistor is essential for safe operation as it limits the current flowing through the LED, preventing damage to both the LED and the Arduino's output pin. You can choose the resistor value based on the desired current using Ohm's Law (V = IR) where V is the voltage of your board (5V or 3.3V) minus the forward voltage for the LED you are using (typical for red would be 1.8 to 2.2 volts). In this case, using a 220-ohm resistor with an Arduino UNO R3 (a 5V board) limits the current to a safe level for both the LED and the Arduino pin. Adjusting the resistor value allows you to control the LED's brightness while ensuring safe operation. For 5V boards you can expect the LED to be visible to a resistor value of up to 1K Ohm.

Connection:



Schematic:



Code Explanation:

After you build the circuit plug your Arduino board into your computer, start the Arduino Software (IDE) and enter the code below. You may also load it from the menu File/Examples/01.Basics/Blink . The first thing you do is to initialize LED_BUILTIN pin as an output pin with the line

pinMode(LED_BUILTIN, OUTPUT);

In the main loop, you turn the LED on with the line:

digitalWrite(LED_BUILTIN, HIGH);

This supplies 5 volts to the LED anode. That creates a voltage difference across the pins of the LED, and lights it up. Then you turn it off with the line:

digitalWrite(LED_BUILTIN, LOW);

That takes the LED_BUILTIN pin back to 0 volts, and turns the LED off. In between the on and the off, you want enough time for a person to see the change, so the delay() commands tell the board to do nothing for 1000 milliseconds, or one second. When you use the delay() command, nothing else happens for that amount of time. Once you've understood the basic examples, check out the BlinkWithoutDelay example to learn how to create a delay while doing other things.

Once you've understood this example, check out the DigitalReadSerial example to learn how read a switch connected to the board.

Code:

```
// 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);  // change state of the LED by setting the pin
to the HIGH voltage level
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // change state of the LED by setting the pin
to the LOW voltage level
  delay(1000);                      // wait for a second
}
```


Chapter 2 - Analog Read Serial

Description:

Read a potentiometer, print its state out to the Arduino Serial Monitor.

This example shows you how to read analog input from the physical world using a potentiometer. A potentiometer is a simple mechanical device that provides a varying amount of resistance when its shaft is turned. By passing voltage through a potentiometer and into an analog input on your board, it is possible to measure the amount of resistance produced by a potentiometer (or pot for short) as an analog value. In this example you will monitor the state of your potentiometer after establishing serial communication between your Arduino and your computer running the Arduino Software (IDE).

Hardware Required

- Arduino Board
- 10k ohm Potentiometer

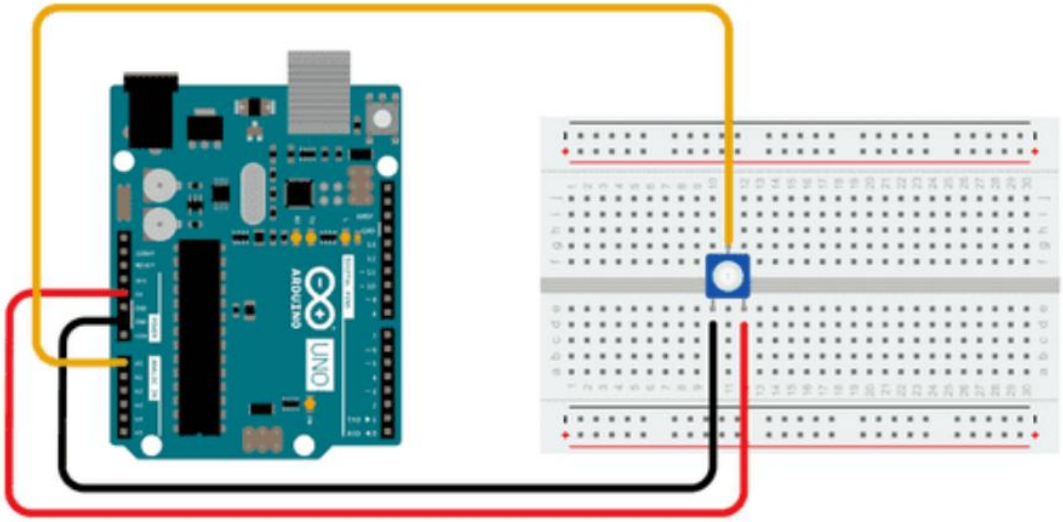
Circuit

Connect the three wires from the potentiometer to your board. The first goes from one of the outer pins of the potentiometer to ground. The second goes from the other outer pin of the potentiometer to 5 volts. The third goes from the middle pin of the potentiometer to the analog pin A0.

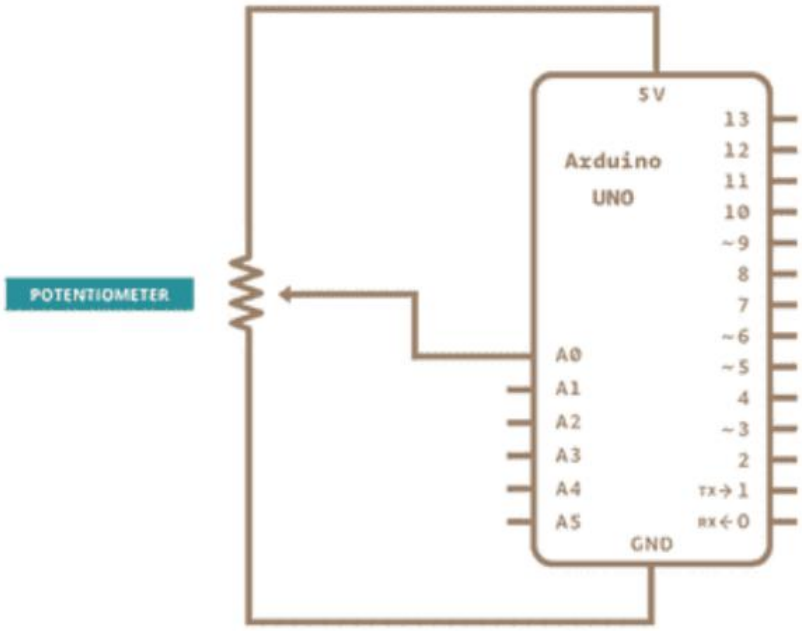
By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper, which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10k ohm), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the analog voltage that you're reading as an input.

The Arduino boards have a circuit inside called an analog-to-digital converter or ADC that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

Connection:



Schematic:



Code Explanation:

In the sketch below, the only thing that you do in the setup function is to begin serial communications, at 9600 bits of data per second, between your board and your computer with the command:

Serial.begin(9600);

Next, in the main loop of your code, you need to establish a variable to store the resistance value (which will be between 0 and 1023, perfect for an intdatatype) coming in from your potentiometer:

int sensorValue = analogRead(A0);

Finally, you need to print this information to your serial monitor window. You can do this with the command Serial.println() in your last line of code:

Serial.println(sensorValue);

Now, when you open your Serial Monitor in the Arduino Software (IDE) (by clicking the icon that looks like a lens, on the right, in the green top bar or using the keyboard shortcut Ctrl+Shift+M), you should see a steady stream of numbers ranging from 0-1023, correlating to the position of the pot. As you turn your potentiometer, these numbers will respond almost instantly.

Code:

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);  // delay in between reads for stability
}
```


Chapter 3 - Fading a LED

Description:

Demonstrates the use of analog output to fade an LED.

This example demonstrates the use of the analogWrite() function in fading an LED off and on. AnalogWrite uses pulse width modulation (PWM), turning a digital pin on and off very quickly with different ratio between on and off, to create a fading effect.

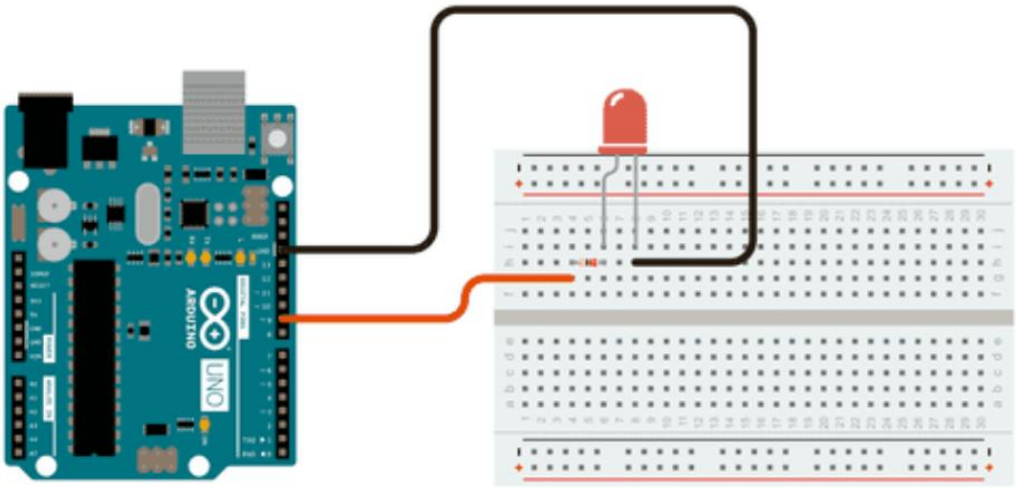
Hardware Required:

- Arduino Board
- LED
- 220 ohm resistor
- hook-up wires
- breadboard

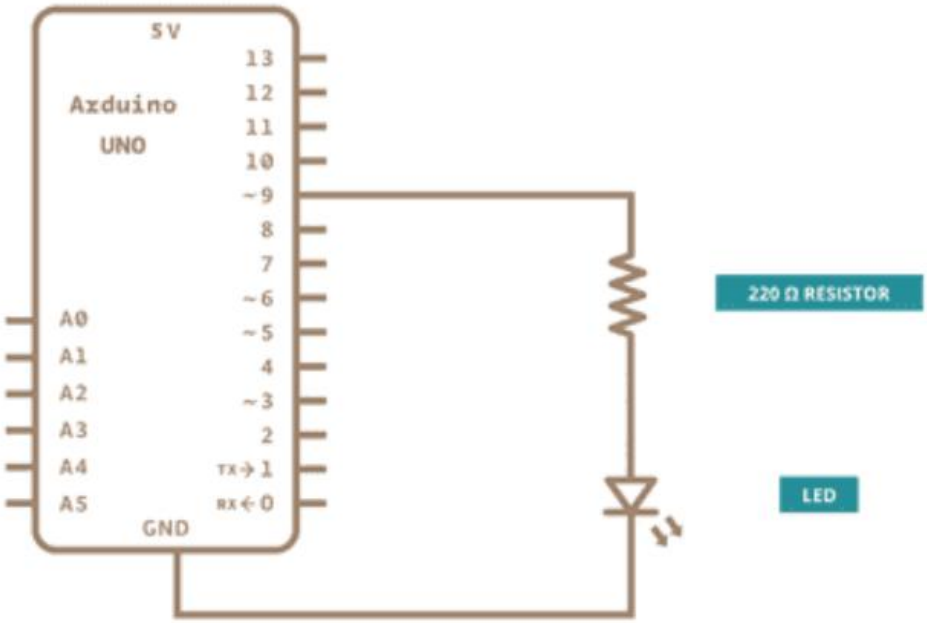
Circuit:

Connect the anode (the longer, positive leg) of your LED to digital output pin 9 on your board through a 220 ohm resistor. Connect the cathode (the shorter, negative leg) directly to ground.

Connection:



Schematic:



Code Explanation:

After declaring pin 9 to be your ledPin, there is nothing to do in the setup() function of your code.

The analogWrite() function that you will be using in the main loop of your code requires two arguments: One telling the function which pin to write to, and one indicating what PWM value to write.

In order to fade your LED off and on, gradually increase your PWM value from 0 (all the way off) to 255 (all the way on), and then back to 0 once again to complete the cycle. In the sketch below, the PWM value is set using a variable called brightness. Each time through the loop, it increases by the value of the variable fadeAmount.

If brightness is at either extreme of its value (either 0 or 255), then fadeAmount is changed to its negative. In other words, if fadeAmount is 5, then it is set to -5. If it's -5, then it's set to 5. The next time through the loop, this change causes brightness to change direction as well.

analogWrite() can change the PWM value very fast, so the delay at the end of the sketch controls the speed of the fade. Try changing the value of the delay and see how it changes the fading effect.

Code:

```
int led = 9;        // the PWM pin the LED is attached to
int brightness = 0;  // how bright the LED is
int fadeAmount = 5;  // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```


Chapter 4 - Digital Read Serial

Description:

Read a switch, print the state out to the Arduino Serial Monitor.

This example shows you how to monitor the state of a switch by establishing serial communication between your Arduino and your computer over USB.

Hardware Required:

- Arduino Board
- A momentary switch, button, or toggle switch
- 10k ohm resistor
- hook-up wires/Jumper wires
- Breadboard

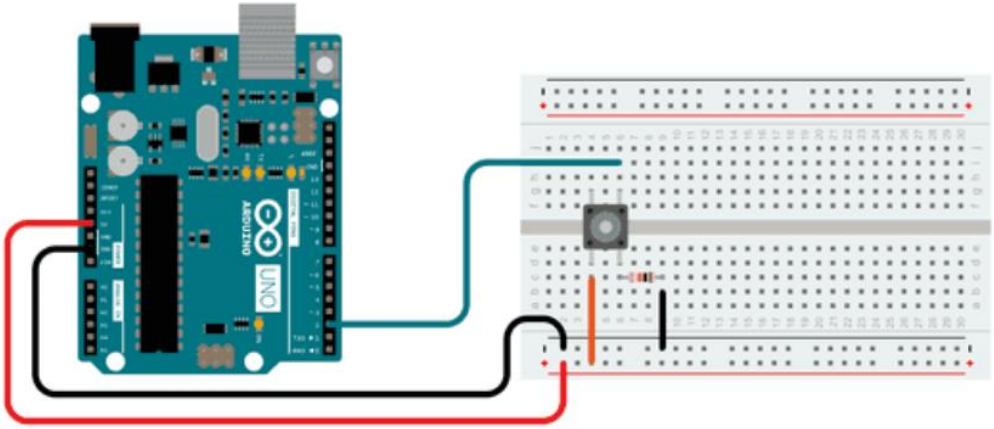
Circuit:

Connect three wires to the board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10k ohm) to ground. The other leg of the button connects to the 5 volt supply.

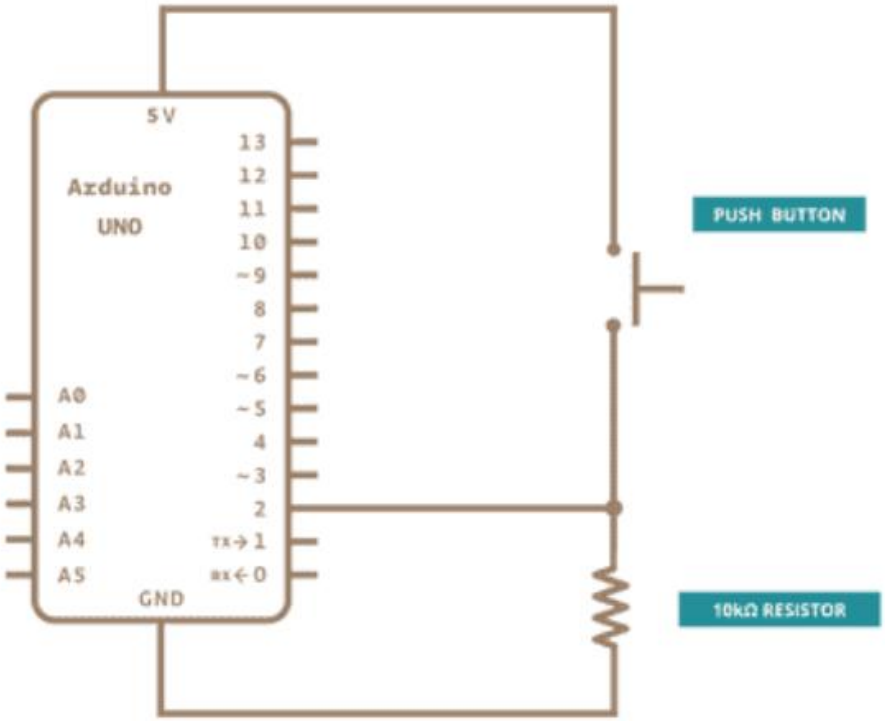
Pushbuttons or switches connect two points in a circuit when you press them. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and reads as LOW, or 0. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that the pin reads as HIGH, or 1.

If you disconnect the digital i/o pin from everything, its reading may change erratically. This is because the input is "floating" - that is, it doesn't have a solid connection to voltage or ground, and it will randomly return either HIGH or LOW. That's why you need a pull-down resistor in the circuit.

Connection:



Schematic:



Code Explanation:

In the program below, the very first thing that you do will in the setup function is to begin serial communications, at 9600 bits of data per second, between your board and your computer with the line:

Serial.begin(9600);

Next, initialize digital pin 2, the pin that will read the output from your button, as an input:

pinMode(2,INPUT);

Now that your setup has been completed, move into the main loop of your code. When your button is pressed, 5 volts will freely flow through your circuit, and when it is not pressed, the input pin will be connected to ground through the 10k ohm resistor. This is a digital input, meaning that the switch can only be in either an on state (seen by your Arduino as a "1", or HIGH) or an off state (seen by your Arduino as a "0", or LOW), with nothing in between.

The first thing you need to do in the main loop of your program is to establish a variable to hold the information coming in from your switch. Since the information coming in from the switch will be either a "1" or a "0", you can use an intdatatype. Call this variable sensorValue, and set it to equal whatever is being read on digital pin 2. You can accomplish all this with just one line of code:

int sensorValue = digitalRead(2);

Once the board has read the input, make it print this information back to the computer as a decimal value. You can do this with the command Serial.println() in our last line of code:

Serial.println(sensorValue);

Now, when you open your Serial Monitor in the Arduino Software (IDE), you will see a stream of "0"s if your switch is open, or "1"s if your switch is closed.

Code:

```
// digital pin 2 has a pushbutton attached to it. Give it a name:
int pushButton = 2;

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
  delay(1);  // delay in between reads for stability
}
```

