Introducing Robotics

Module A

Unit #9

button

Contents

Module A Unit #9 button

Introduction to the button

The button has four legs. You connect them on the breadboard as shown in the diagram below. The button straddles the breadboard's central divide. So that two legs are on one side and two legs on the other side. This is just to help make sure we can connect correctly.

We have Leg 1 and Leg 2; it doesn't matter which is which. When the button is pressed, it connects Leg 1 and Leg 2, completing an electrical circuit. But as soon as you stop pressing the button, the circuit is broken; it is called a momentary button or a tactile button. It only works when you press it.

We connect one Leg to the ground (GND) and one Leg to digital pin 3 on the Arduino Uno. We do have to be a bit careful here because ordinarily we would also have to include a resistor in the circuit, otherwise it would short it out and maybe damage the Arduino. However, the Arduino has built-in resistors called pull-up resistors, which we can use, but we have to explicitly tell the Arduino to do so.

This next bit is a bit counterintuitive. While using a pull-up resistor, the state of the digital pin (3 in this case) is default HIGH. This means that its boolean state is HIGH when we don't press the button (and complete the circuit) but LOW when we press the button.

Just to confuse you a little bit more. If we included an external resistor in the circuit, then the opposite is true; this would be a little bit more intuitive. This is all fine as long as you know. I am just saving you the trouble of using a resistor in the circuit for simplicity.

What you will need

1 x Arduino Uno
1 x breadboard
1 x LED traffic light module
1 x button
5 x male-to-male jumper leads

Circuit Diagram for unit #9

You will be using male-to-male jumper wires.

Button	Arduino Pins
Leg 1	3
Leg 2	GND

Traffic Lights	Arduino Pins
GND	GND
R (red)	11
Y (yellow)	10
G (green)	9

figure 1: Circuit Diagram (B2) button and LED traffic lights

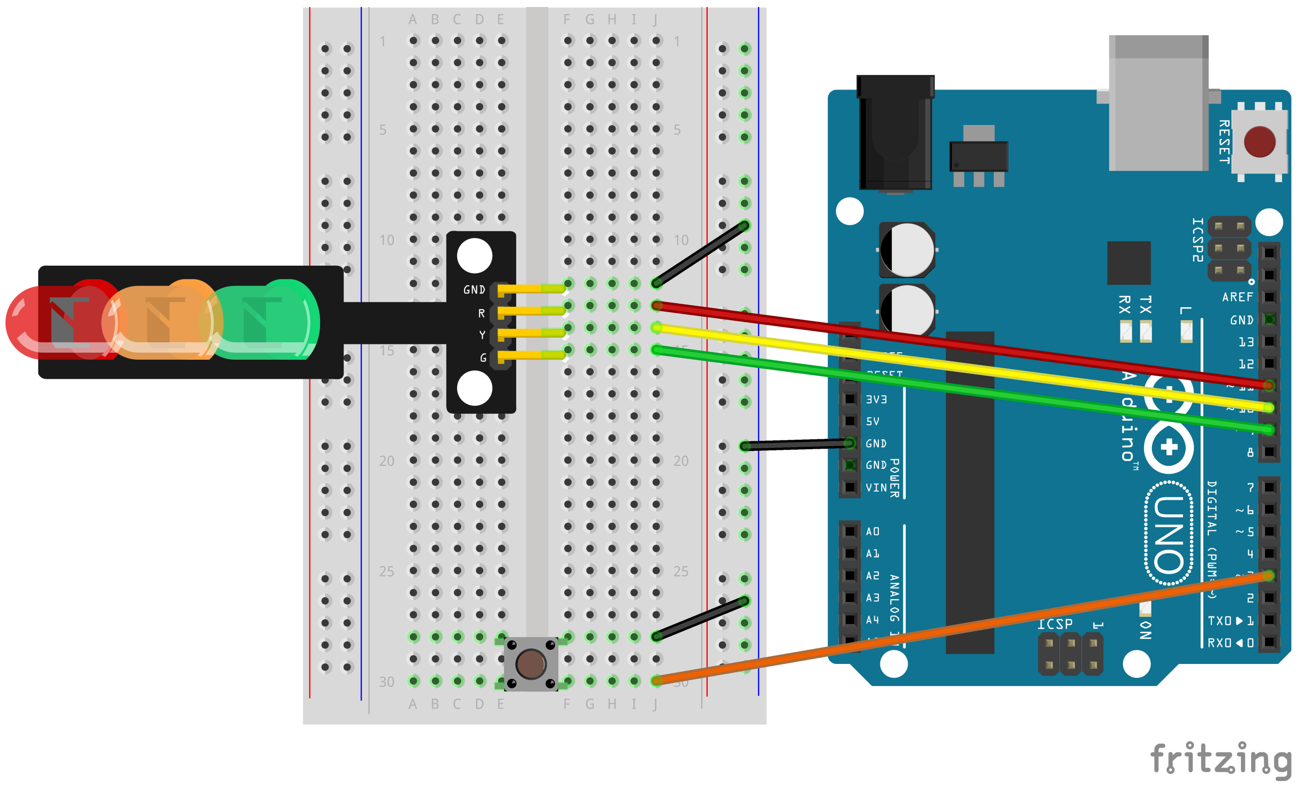
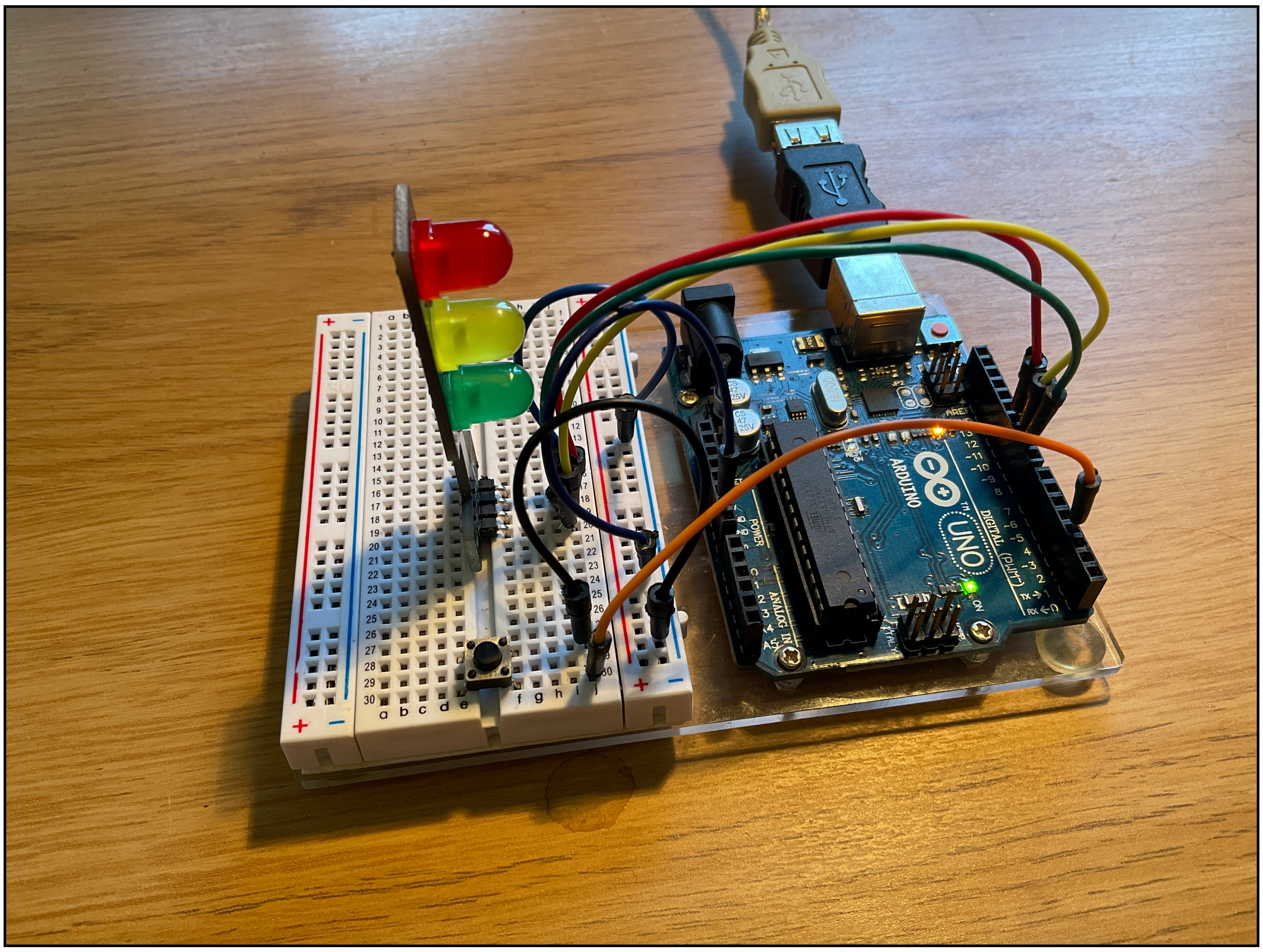


Figure 2: Button and LEDs



Sketch A9.1 LED button

Connect the button as shown, then after writing the code and uploading it to the Arduino, press the button. The red LED should come on when pressed and off when released.

```
int ledPin = 11;
int buttonPin = 3;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop()
{
  int button = digitalRead(buttonPin);
  if (button == HIGH)
  {
    digitalWrite(ledPin, LOW);
  }
  else
  {
    digitalWrite(ledPin, HIGH);
  }
}
```

🛠️ Code Explanation

`pinMode(buttonPin, INPUT_PULLUP);`	We are using digital pin 3 for the button and we need to set it to INPUT_PULLUP in setup
`if (button == HIGH)`	The == means is equal to. If the button is HIGH then there is no circuit

Sketch A9.2 LED toggle

In this sketch, we are doing more than just switching it on and off with the button but toggling it so once pressed on and once pressed off. This is more difficult than the previous sketch. Hold the button for a second each time.

Important note: this can work very badly because of a condition known as bounce (we will look at that in the next sketch).

```
int ledPin = 11;
int buttonPin = 3;
int ledState = LOW;
int lastButtonState = LOW;
int currentButtonState = LOW;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop()
{
  lastButtonState = currentButtonState;
  currentButtonState = digitalRead(buttonPin);
  if(lastButtonState == HIGH && currentButtonState == LOW)
  {
    ledState = !ledState;
    digitalWrite(ledPin, ledState);
  }
}
```

Notes

Work through the sketch to follow the logic; the logic isn't flawed, but the button is. The problem is that the contacts, as you press the button, jump or bounce and give false readings. The next sketch addresses this problem by taking into account the bounce.

Sketch A9.3 debounce

I recommend starting a new sketch. I have highlighted the changes anyway. To tackle the bounce problem, we need to introduce some sort of threshold, a timed value (50 milliseconds) to make sure that there is a proper connection and not an erratic one. We will call that debounceDelay.

```
int ledPin = 11;
int buttonPin = 3;
int ledState = LOW;
int buttonState = LOW;
int lastButtonState = LOW;
int currentButtonState = LOW;

long lastDebounceTime = 0;
Int debounceDelay = 50;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop()
{
  currentButtonState = digitalRead(buttonPin);
  if (currentButtonState != lastButtonState)
  {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay)
  {
    if (currentButtonState != buttonState)
    {
      buttonState = currentButtonState;
      if (buttonState != HIGH)
      {
        ledState = !ledState;
      }
    }
  }
  digitalWrite(ledPin, ledState);
  lastButtonState = currentButtonState;
}
```

🗒️ Notes

This is more about Boolean logic than the code. But all that code is to just toggle an LED on or off. This is what coding is all about: problem-solving. This is a work-around for a hardware issue.

🌻 Challenge

You can work through the logic and change things to give yourself a chance to follow the logic. One way is to have a list of the variables and record their state or value, and see how they change as you press the button.

🛠️ Code Explanation

`long lastDebounceTime = 0;`	This has to be long because it uses the millis() function and the number can get big very quickly.
`int debounceDelay = 50;`	This has to be long because it uses the millis() function and the number can get big very quickly.
`ledState = !ledState;`	Switches the boolean from LOW to HIGH (or not LOW) or vice versa hence !ledState