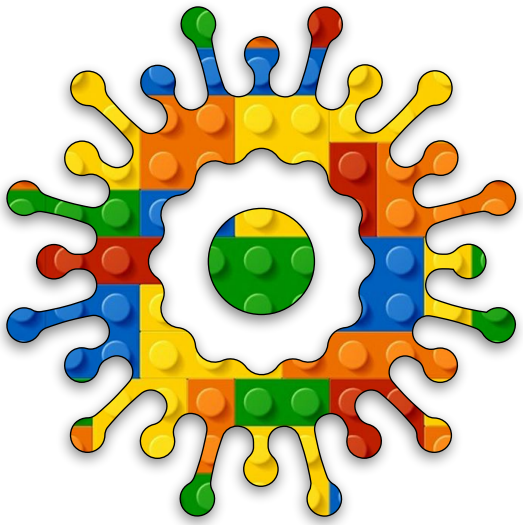
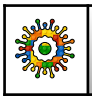
Internet of Things

Module A

Unit #8

button





Contents

Module A Unit #8 button

Introduction to the button

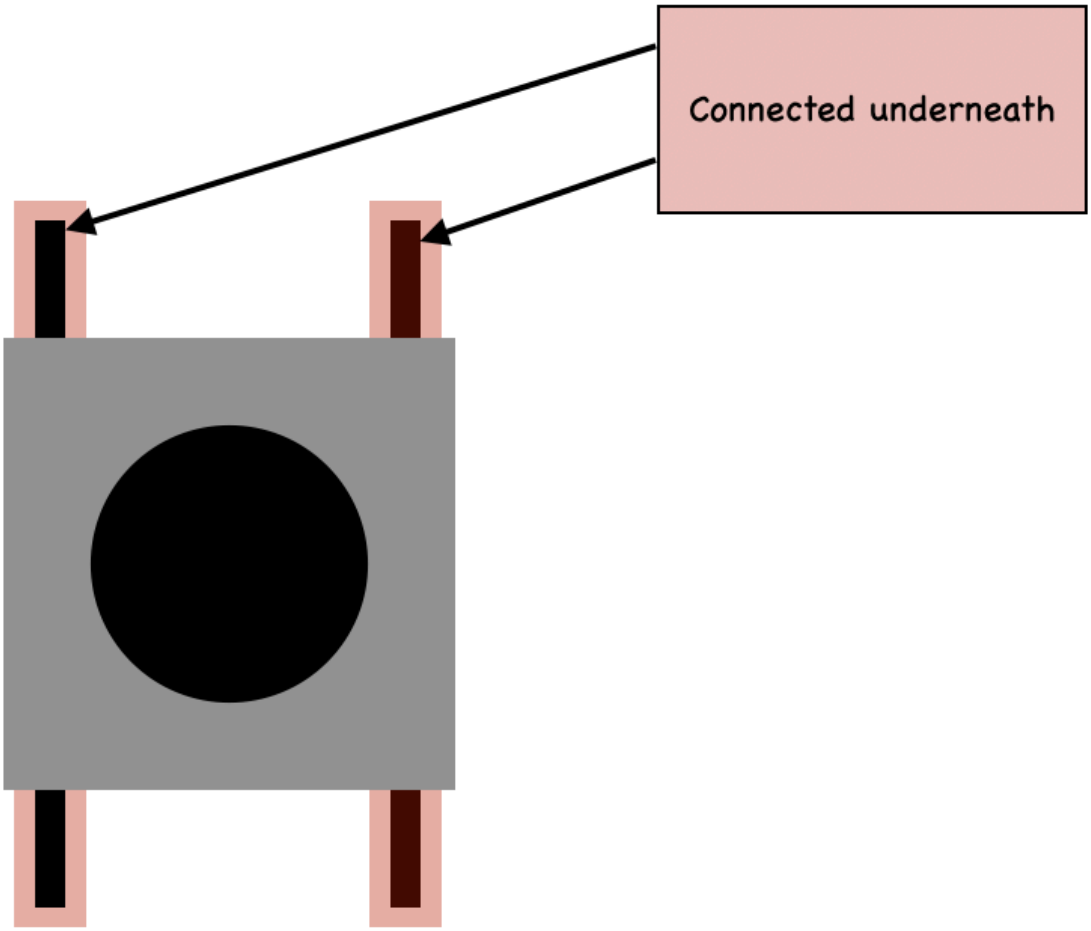
The button is a tactile or momentary push button. It makes contact with a metal plate when you push down and completes the circuit. When you stop pushing, it releases and is no longer in contact and hence breaks the circuit. See Fig. 2.

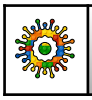
As you can see, it has four pins protruding from the body of the button and a black button on top. This kind of button fits nicely on the breadboard. The pins are connected as shown in Fig. 2. When you press the button, you connect the pins from left to right (as seen in the diagram below). The pins top to bottom are already connected internally.

To use this button, see Fig.1 (rather than a button module), we need a resistor. The beauty of the Arduino is that it has a built-in resistor we can use with the button (but not with an LED!). It is called a pull-up resistor, more on that later.

Figure 1: button

Figure 2: the button





What you will need

1 x Arduino Nano 33 IoT
1 x breadboard
1 x LED traffic lights
1 x button
2 x male to male jumper leads

We connect (on the same side of the protruding pin) one pin to a ground GND pin and the other pin to D5 (Arduino pin 5). You can use any digital pin; using pin 5 is just a suggestion.

One of the pins -> GND
The other pin -> D5 (pin 5)

Circuit Diagram for the button

You will need two wires to connect the button to the device.

Button	Arduino Pins
OUT	5
GND	GND

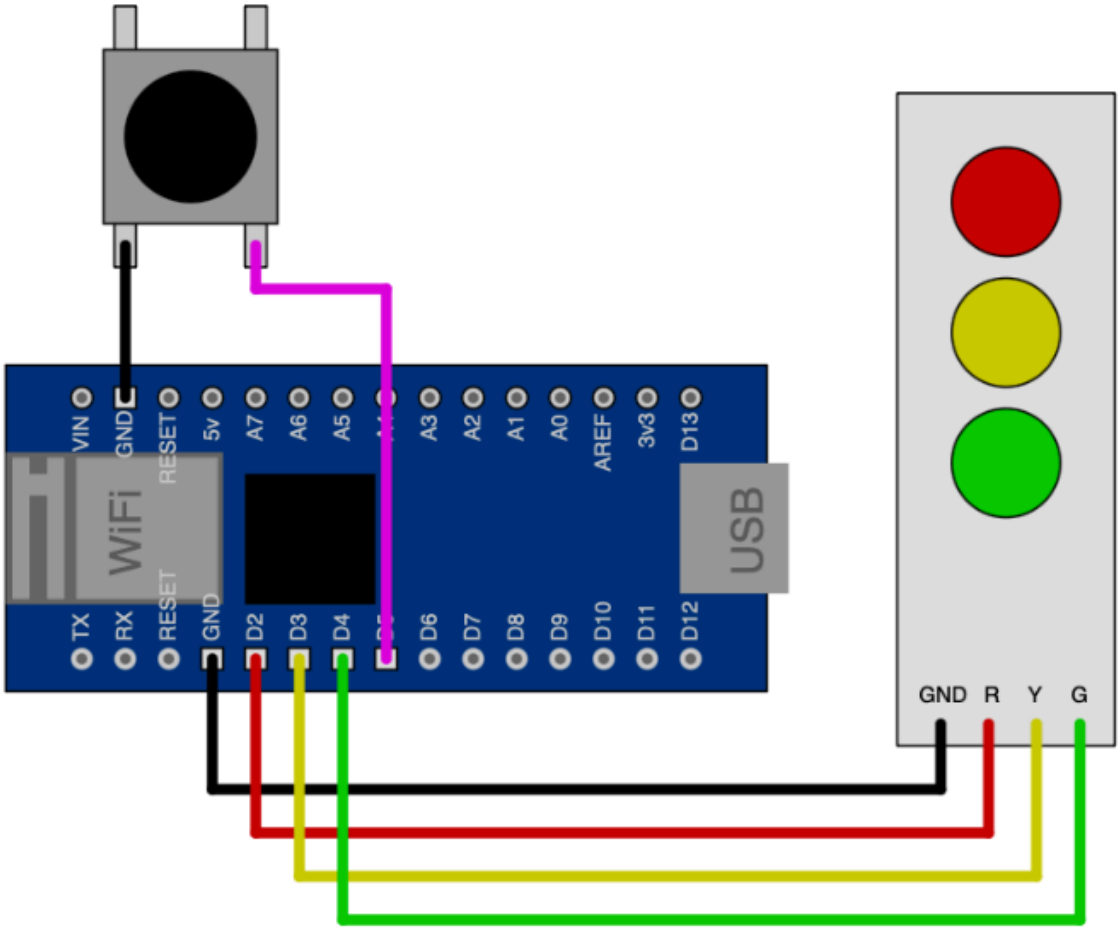
LEDs as before.

Traffic Lights	Arduino Pins
GND	GND
R (red)	2
Y (yellow)	3
G (green)	4

The circuit diagram below shows the traffic light LEDs as we had them before (no change). We are going to add the button to the breadboard. You can put it anywhere away from the board itself. I generally place it across the central divide running the length of the breadboard.

See Fig. 3 below. The wiring diagram shows the connections; your board will look quite different.

Figure 3: circuit diagram



Sketch A8.1 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. When using the pull-up resistor, the default (not pressed) state is HIGH. This is a little bit counterintuitive; if you use an external resistor, the opposite is true.

```
int ledPin = 2;
int buttonPin = 5;

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

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

🛠️ Code Explanation

`pinMode(buttonPin, INPUT_PULLUP);`	Using the pull-up (internal) resistor
`if (button != LOW)`	The != means not. If the button is not LOW (pressed)...

Sketch A8.2 LED toggle

In this sketch, we are doing more than just switching it on and off with the button, but toggling it so that on one press the LED is on and on the next press of the button the LED switches off. This is more difficult than the previous sketch. Hold the button for a second each time.

! this works very badly because of a condition known as bounce (we will look at that in the next sketch).

```
int ledPin = 2;
int buttonPin = 5;
int ledState = LOW;
int lastButtonState = HIGH;
int currentButtonState = HIGH;

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

void loop()
{
  lastButtonState = currentButtonState;
  currentButtonState = digitalRead(buttonPin);
  if(lastButtonState == LOW && currentButtonState == HIGH)
  {
    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 A8.3 debounce

I recommend starting a new sketch; too many changes to highlight. To tackle the bounce problem, we need to introduce some sort of delay to counter that. We will call that debounceDelay.

```
int ledPin = 2;
int buttonPin = 5;
int ledState = LOW;
int buttonState = LOW;
int lastButtonState = LOW;
int currentButtonState = LOW;

long lastDebounceTime = 0;
long debounceDelay = 50;

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

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 just to toggle an LED on or off. This is what coding is all about: problem solving. This is a workaround for a hardware issue.

🛠️ Code Explanation

`long lastDebounceTime = 0;`	These two lines of code are the key to this working, they have to be long because they use the millis() function and the number can get big very quickly.
`long debounceDelay = 50;`	
`Serial.available()`	gets the number of bytes (characters) available from reading the serial port
`Serial.setTimeout()`	sets the maximum milliseconds to wait for serial data
`Serial.parseInt()`	returns the first integer number
`Serial.Read()`	reads incoming serial data