2. Hello World!

A *"Hello World!"* in the Arduino sphere is a blinking LED.
You just need an Arduino board and a USB cable.
pen a new file in the IDE. The lines of code below are pre-written and form the basis of every program. We'll explain more about that later.

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

}

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

}
```

Give the file a name and save it.
Then type the following text into the Arduino sketch editor, but you can skip the lines starting with a `//` as they are comments.

```
// LED connected to digital pin 13 > works for UNO & Every
const int ledPin = 13;

// the setup function runs once when you press reset
// or power the board
void setup() {
    // initialize digital pin 13 as an output.
  pinMode(ledPin, OUTPUT);

}

  // the loop function runs over and over
void loop() {
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(ledPin, HIGH);
  // wait for 1000 milliseconds or 1 second
  delay(1000);
  // turn the LED off by making the voltage LOW
  digitalWrite(ledPin, LOW);
  // wait for another second
  delay(1000);
}
```

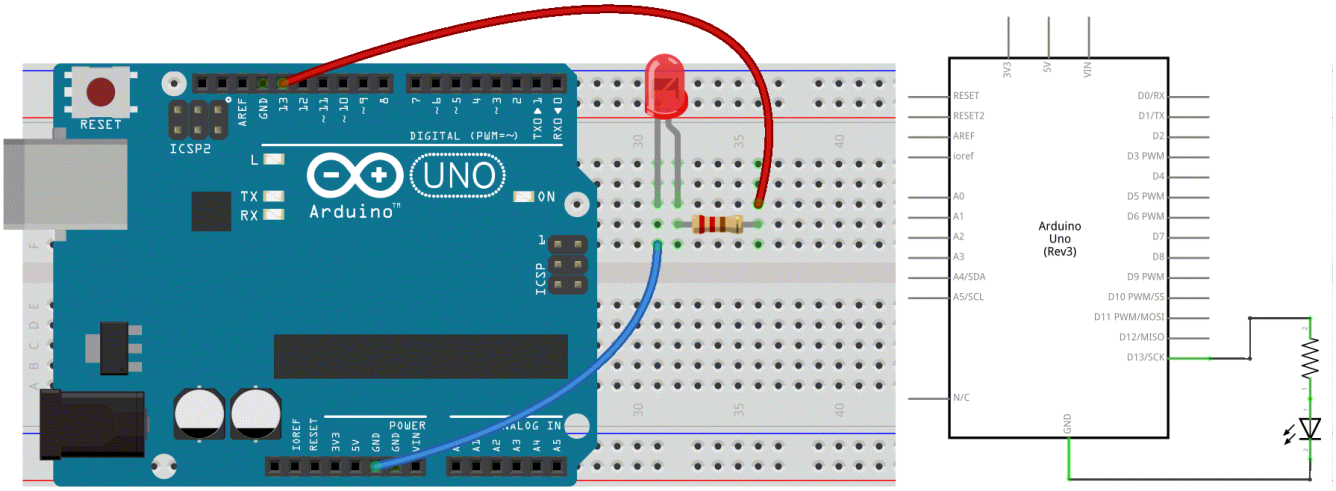
Click the **Verify** button to check whether your code is correct.
If everything is fine, you'll see the message **"Done compiling"** appear at the bottom of the Arduino IDE. The Arduino IDE has translated your sketch into an executable program that can be run by the board.

Now, click the **Upload** button. This will reset the board and stop its current functions. The compiled sketch is then sent to the board and stored in its memory. Afterwards, the board will run it. Subsequently the board will run it.

If everything went well you'll see the notifications **"Done compiling."** and **"Done uploading."** appear to let you know the process has completed correctly.

You can adjust the two delay values to change the blinking rhythm. Don't forget to compile and upload the code after making changes.

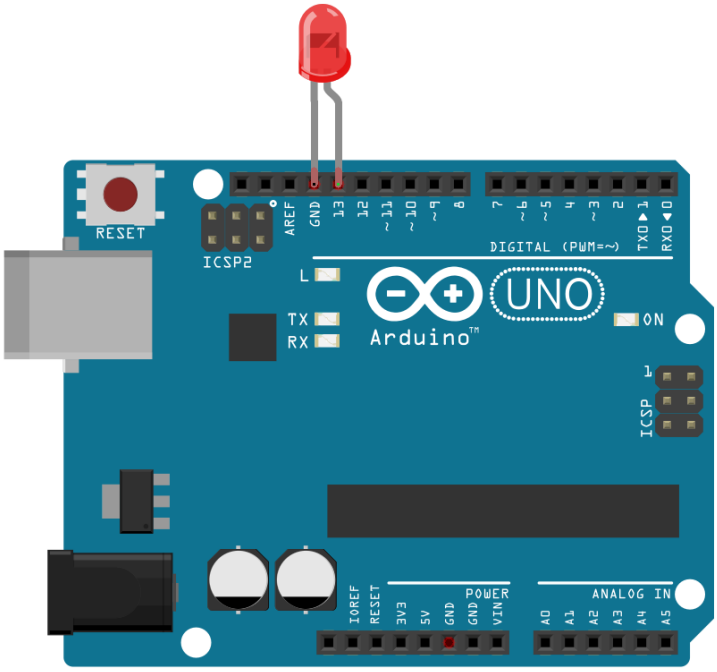
3b. An LED on a Breadboard

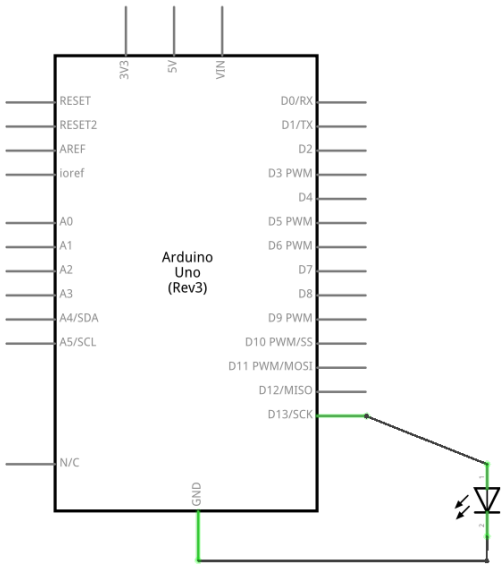


Next, let's try some other actuators:

- A **buzzer** (or beeper) is a small device that produces a buzzing sound and is commonly used for signaling.
- A **relay** is an electrically operated switch. It uses a low-voltage control signal to switch a circuit, usually at a higher voltage. Relays can be used to control lighting, electrical appliances, and other equipment.
- ...

3. An External LED





4b. Sticky On/Off Button

Lets program **a second behavior** that to make the button "stick".

Code

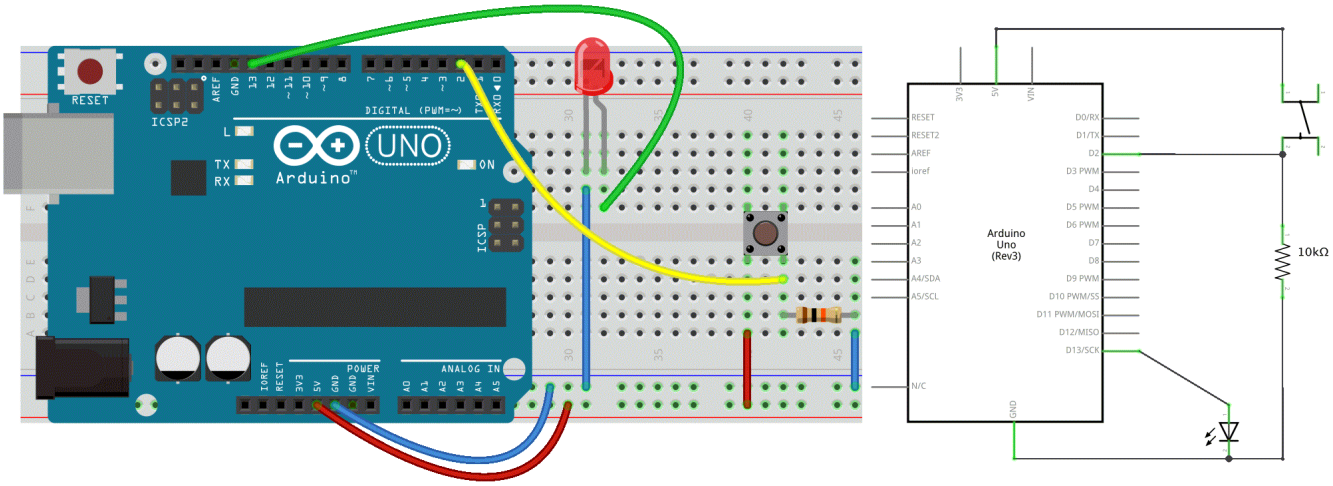
```
/* Turn on LED when the button is pressed
  and keep it on after it is released */

const int buttonPin = 2;
const int ledPin =  13;

int val = 0;          // val will be used to store the state of the input pin
int old_val = 0;      // this variable stores the previous value of "val"
int buttonState = 0;  // variable that will store the pushbutton status

void setup() {
  // initialize the LED pin as an output
  // & the pushbutton pin as an input
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  val = digitalRead(buttonPin);

  // check if there was a transition
  if ((val == HIGH) && (old_val == LOW)) {
    buttonState = 1 - buttonState;
    delay(10);    // small delay for debouncing
  }

  old_val = val;  // val is now old, let's store it

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == 1) {
    digitalWrite(ledPin, HIGH); // turn LED on
  } else {
    digitalWrite(ledPin, LOW);  // turn LED off
  }
}
```

4. Push the Button

In our first example, the LED was our actuator, and the Arduino controlled it. Now, imagine using an external input—like your finger—to control that LED. For this, we need **a sensor**. The simplest sensor available is **a pushbutton**.

Circuit

- LED connected from pin 13 to ground
- Pushbutton connected to pin 2 and +5V
- 10K resistor connected from pin 2 to ground



Code

```
// constants don't change:
const int buttonPin = 2;
const int ledPin =  13;

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output
  // & the pushbutton pin as an input
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

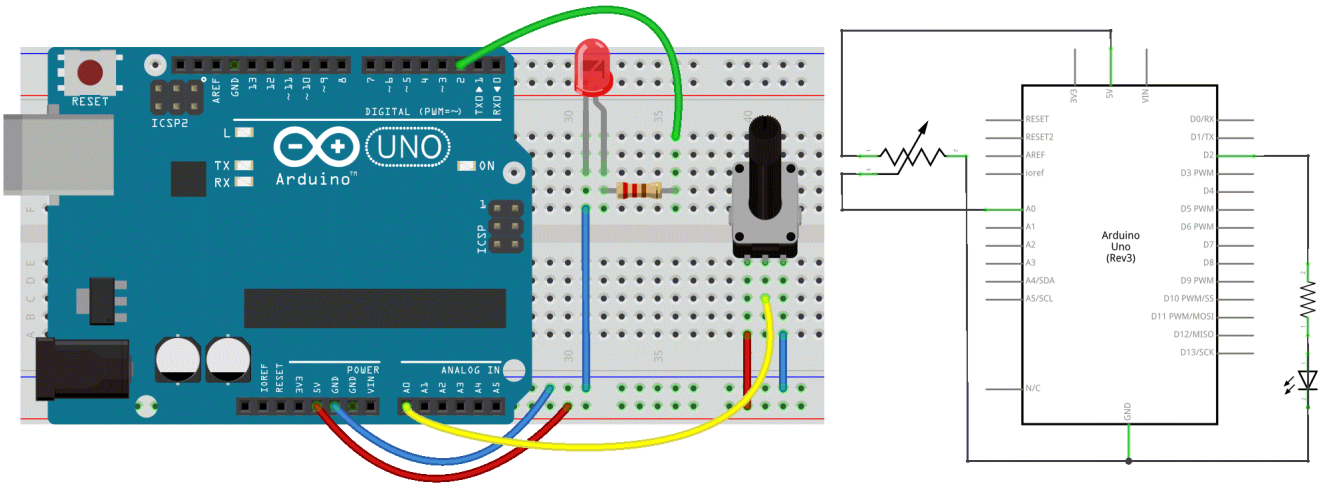
If everything is correct, the LED will light up when you press the button.
Yes? Great!

5a. Analog Sensors (for UNO)

The next sketch and circuit diagram demonstrate analog input by reading an analog sensor — a potentiometer (or trimpot)— on analog pin 0 and turning an LED on and off using digital pin 2. The amount of time the LED stays on and off depends on the value obtained from `analogRead()`.

Circuit

- Potentiometer: Connect the center pin of the potentiometer to analog input A0, one side pin to ground, and the other side pin to +5V.
- LED: A 220Ω resistor connects digital output pin 2 to the anode (long leg) of the LED, while the cathode (short leg) is connected to ground.
  (The resistor can also be placed between the cathode and ground, since in a series circuit the order of components does not matter—the current must pass through all parts.)



Code

```
int sensorPin = A0;   // select the input pin for the potentiometer
int ledPin = 2;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT
  pinMode(ledPin, OUTPUT);
  // there is no need to set our analog in pin
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

5b. talk2me

Let's set up a **serial communication** between our Arduino and the computer to monitor changing values.

In the code below, we'll map the range 0–1023 to a custom range of 10–500, send both variables over the serial port, and use the Arduino Serial Monitor to view them. Click the Serial Monitor button in the toolbar and select the same baud rate specified in the `Serial.begin()` call.

The circuit remains unchanged.

Code

```
int sensorPin = A0;    // select the input pin for the potentiometer
int ledPin = 2;        // select the pin for the LED
int sensorValue = 0;   // variable to store the value coming from the sensor
int outputValue = 0;   // variable to store a scaled value of the sensorvalue

void setup() {
  // declare the ledPin as an OUTPUT
  pinMode(ledPin, OUTPUT);
  // initialize serial communications at 9600 bps
  Serial.begin(9600);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);

  // map or scale it to a custom range:
  outputValue = map(sensorValue, 0, 1023, 10, 500);

  // print the results to the Serial Monitor:
  Serial.print("sensor = ");
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);

  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

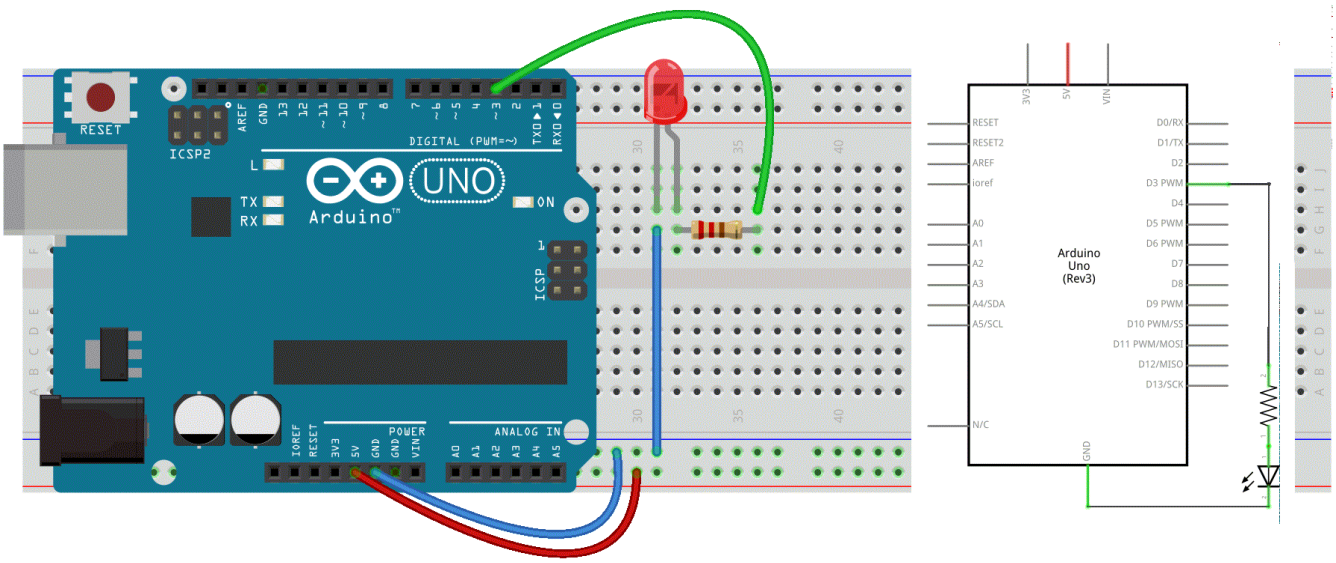
6a. Analog Out, a fading LED

PWM, short for **Pulse Width Modulation**, is a technique used to represent an analog signal level using a digital output.

On an Arduino Uno, there are six PWM pins: digital pins 3, 5, 6, 9, 10, and 11, each marked with a tilde (~) symbol.

We'll explore this PWM feature by gradually changing the brightness of an LED over time.

Circuit



Code

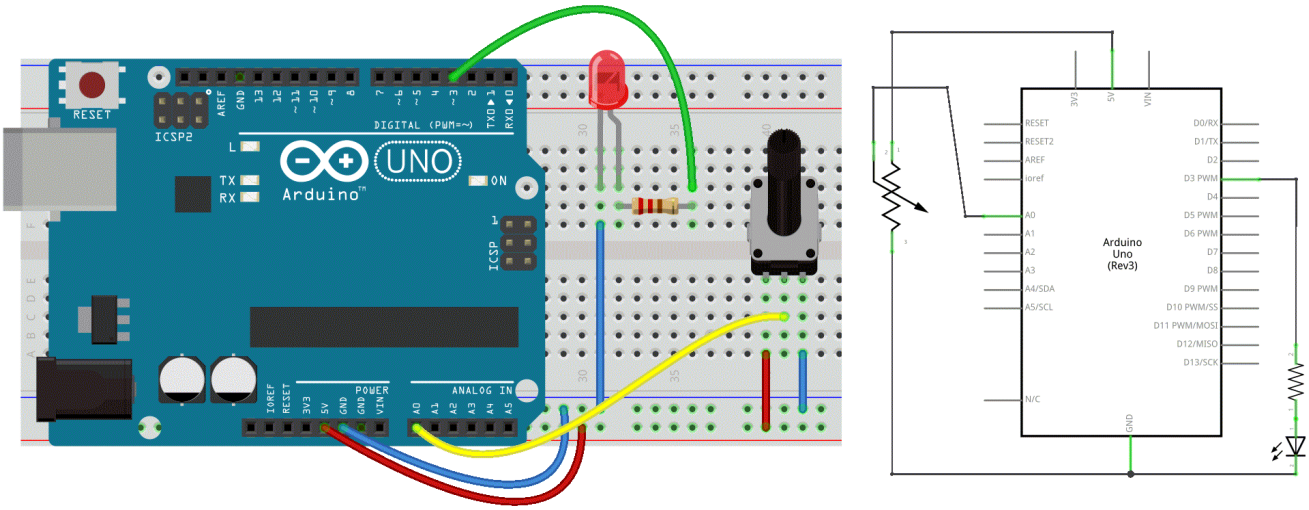
```
int ledPin = 3;      // LED connected to digital pin 3
int fadeAmount = 5;  // how many steps to fade the LED by


void setup() {
  // nothing happens in setup
}

void loop() {
  // fade in from min to max in increments of ? points:
  for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += fadeAmount) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }

  // fade out from max to min in increments of ? points:
  for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= fadeAmount) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }
}
```

6b. Analog Input 2 Output

Now let's connect our input to the output.
In a previous experiment, we created a button-controlled LED using a digital signal to control a digital pin. This time, we'll use a potentiometer to control the brightness of the LED.

The Arduino will read the analog value from the potentiometer and apply this value to a PWM pin. Since PWM only works within the range 0–255, we need to map the sensor input (which ranges from 0–1023) down to this smaller range.

Circuit



Code

```
/* Set the brightness of ledPin to a brightness specified by the
  value of the analog input */

const int ledPin = 3;      // LED connected to digital pin 9
const int analogPin = A0;  // potentiometer connected to analog pin 0

int val = 0;               // variable to store the read value
int ledVal;                // variable to store the output value


void setup() {
  // Noting here as: Analog pins are automatically set as inputs &
  // it is not needed to set the pin as an output before calling analogWrite()
}
void loop() {
  // read the value from the sensor
  val = analogRead(analogPin);
  // turn the ledpin on at the brightness set by the sensor
  //Mapping the Values between 0 to 255 because we can give output
  //from 0 -255 using the analogwrite funtion
  ledVal = map(val, 0, 1023, 0, 255);
  analogWrite(ledPin, ledVal);
  delay(10);
}
```

6c. Servo Motor Control

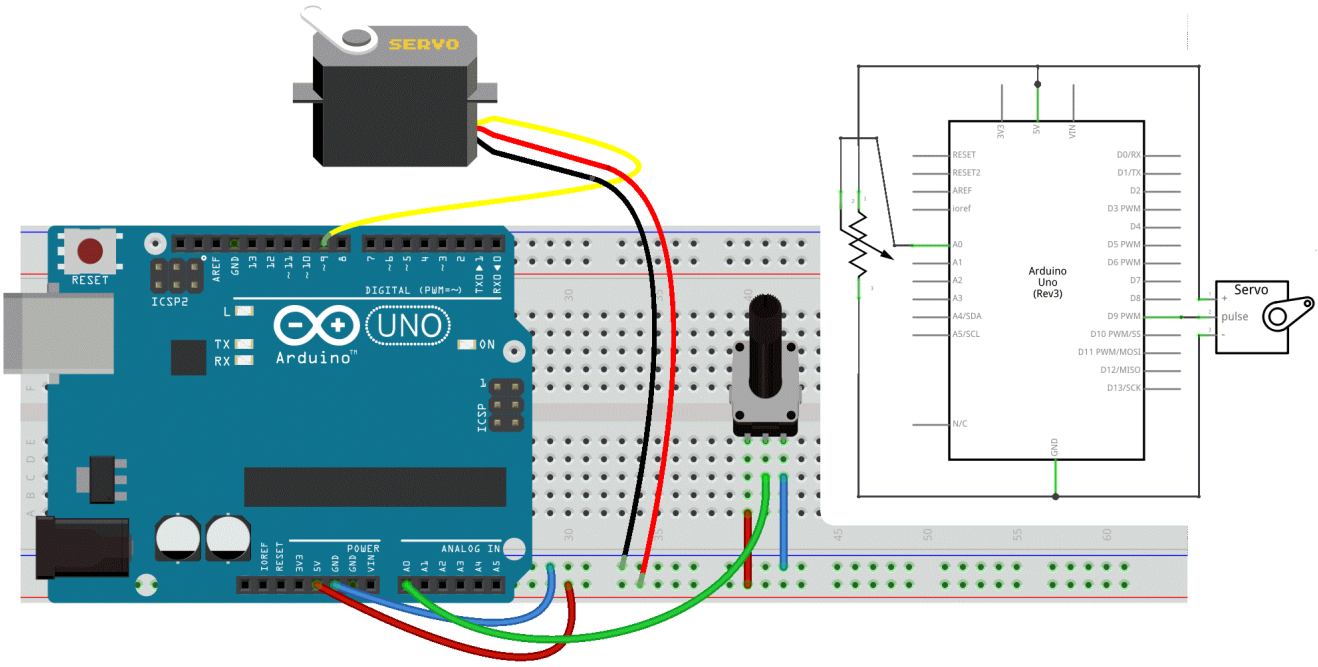
Now, let's replace the LED with a **Servo Motor**.

Servos are motors with a shaft that can rotate to a specified position, typically within a range of 0 to 180 degrees. With an Arduino, we can command the servo to move to a specific angle. In this example, we'll learn how to connect a servo motor and control its position based on the value read from a potentiometer.

Circuit

A servo motor usually has a three-pin female connector:

- The darkest wire (brown in this case) is typically ground. Connect it to the Arduino GND.
- The power wire, usually red, connects to +5V on the Arduino.
- The remaining wire (signal) connects to digital pin 9 (or 10) on the Arduino.



Note: Servos can draw significant current. If you need to control more than one or two servos, use an external power supply instead of the Arduino's +5V pin.

Code

In this example, we'll use **the Servo library**, which simplifies servo control.

To include a library in your sketch, go to Sketch > Include Library, or type the `#include <name_of_library>` directive at the top of your code.

```
#include <Servo.h>

Servo myservo;    // create servo object to control a servo

int potpin = A0;  // analog pin used to connect the potentiometer
int val;          // variable to read the value from the analog pin

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  // read the value of the potentiometer
  val = analogRead(potpin);
```

```
    // scale it to use it with the servo (value between 0 and 180)
    val = map(val, 0, 1023, 0, 180);
    // sets the servo position according to the scaled value
    myservo.write(val);
    // waits for the servo to get there
    delay(15);
  }
```

See the servo reference page for more details.

Key Functions

- `Servo objectName;` — creates a servo object.
- `objectName.attach(pin);` — assigns the servo to a specific pin (usually 9 or 10).
- `objectName.write(angle);` — sets the servo angle (0 to 180 degrees).