ARDUINO

NITI Aayog | ATAL INNOVATION MISSION

ARDUINO®
UNO एक R4

85K
Schools wordwide

1.5M
Students engaged

ARDUINO.CC/EDUCATION

INDEX

Chapter 1 - Introduction to Arduino

What is Arduino?

Arduino is a platform for everyone, including teachers, students, designers, engineers and just about anyone that wants to build electronic projects.

The term "Arduino" refers to the Arduino ecosystem, which has four main pillars:

- **"Arduino board"**, a circuit board with a computer that you can program.
- **"Arduino IDE**, an editor where you write code and upload it to your board.
- **"Arduino Cloud"**, an online platform where you can program your board and connect it via Internet to other boards or services (such as Google Home and Alexa).
- **"Arduino Language"**, a programming language that is based on C++, that contains specific functions for writing Arduino code.

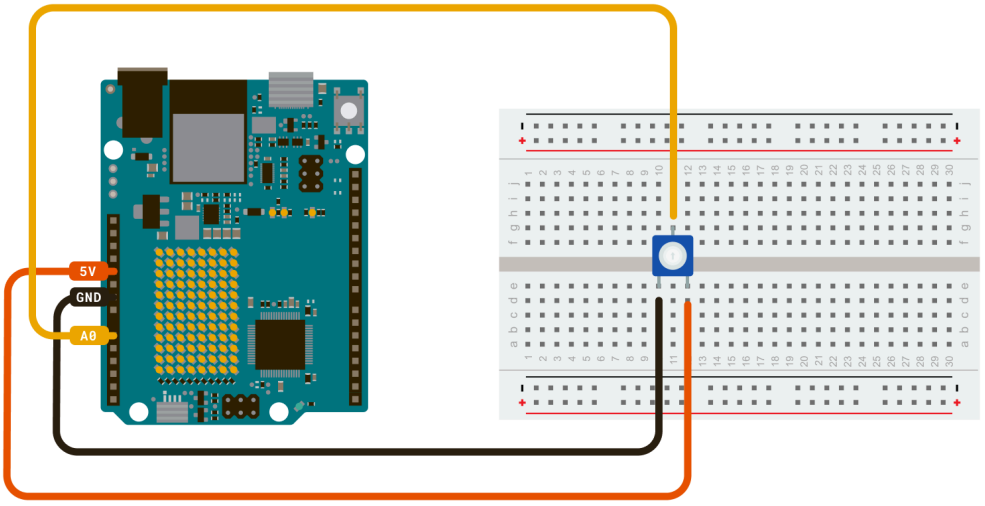
How Does it Work?

An Arduino board has a tiny computer called a **microcontroller**. This microcontroller can be instructed to perform various tasks, depending on what type of program you write.

When you write a program in the Arduino IDE (the code editor), you will be using a range of programming techniques, to make the Arduino behave a specific way.

A Circuit

An easy example to begin with, is by taking a button (an input) and a light (an output). These electronic components can be connected to an Arduino via a breadboard and wires. Here's an example of how that might look like:



When you have connected these, we can create the logic (the program) for how these components should behave when we run the program. But first, we will need to create a program!

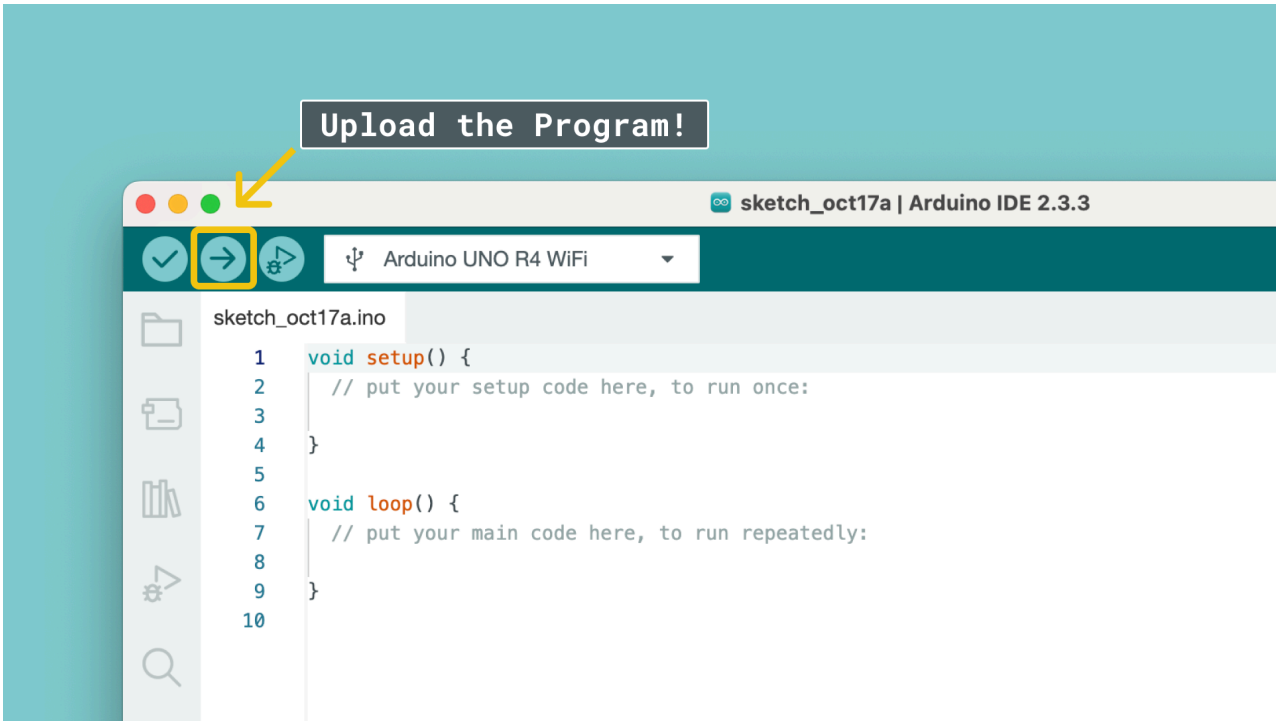
Below is a program, called a **sketch**, that we can write for the Arduino. In it, the comments will explain how it works, step by step.

Code Example

```
int button = 2; //a button connected to pin 2 on the Arduino
int light = 3; //a light (LED) connected to pin 3 on the Arduino

//the setup function will run only once
void setup(){
    pinMode(button, INPUT); //we need to tell the Arduino that it is a INPUT
    pinMode(light, OUTPUT); //and that pin 3 is a OUTPUT
}

//the loop function will run forever
void loop(){

    //first we read the button's state (pressed, or not pressed)
    int readButtonState = digitalRead(button);

    //then, we check if the button is pressed
    if(button == HIGH){
        //if the button is pressed, we activate the LED by writing "HIGH"
        digitalWrite(LED, HIGH)
    } else {
        //if the button is NOT pressed, we de-activate the LED by writing "LOW"
        digitalWrite(LED,LOW);
    }
}
```

Upload a Program

Once we have a program ready, we can upload it to our Arduino board, by connecting the board to our computer, and clicking the **"Upload"** button in the Arduino IDE.



Once the sketch has been uploaded, the board will start running the program, where the `loop()` function will run forever (or until the power is lost).

What Can You Create with an Arduino?

An Arduino is a computer that can control almost any electronic device in the world. You can build anything from a simple lamp that turns on with a click - to a satellite in space that records data from the atmosphere.

> *Fun fact, the first Guatemalan space satellite used an Arduino. You can read more about it here.*

But to give you a couple of practical examples, here are some ideas:

1. **Home Automation System** - such as a smart light that is controlled with your phone.
2. **Robotics** - like a robot arm in a factory.
3. **Weather Station** - using various sensors an Arduino can record data such as temperature, wind, pressure and rainfall.
4. **Interactive Games** - an Arduino can be used to build a game controller, or to build a physical game.

How Do I Create with Arduino?

In this course, you will learn how to set up your computer and Arduino so that you can create anything you want. However, while you can create almost anything with an Arduino, it is important to understand that you will also need to learn how to program and use it effectively. We will in the following exercises go through a series of examples and practices that will help you increase your knowledge, so that you can go on and create something!

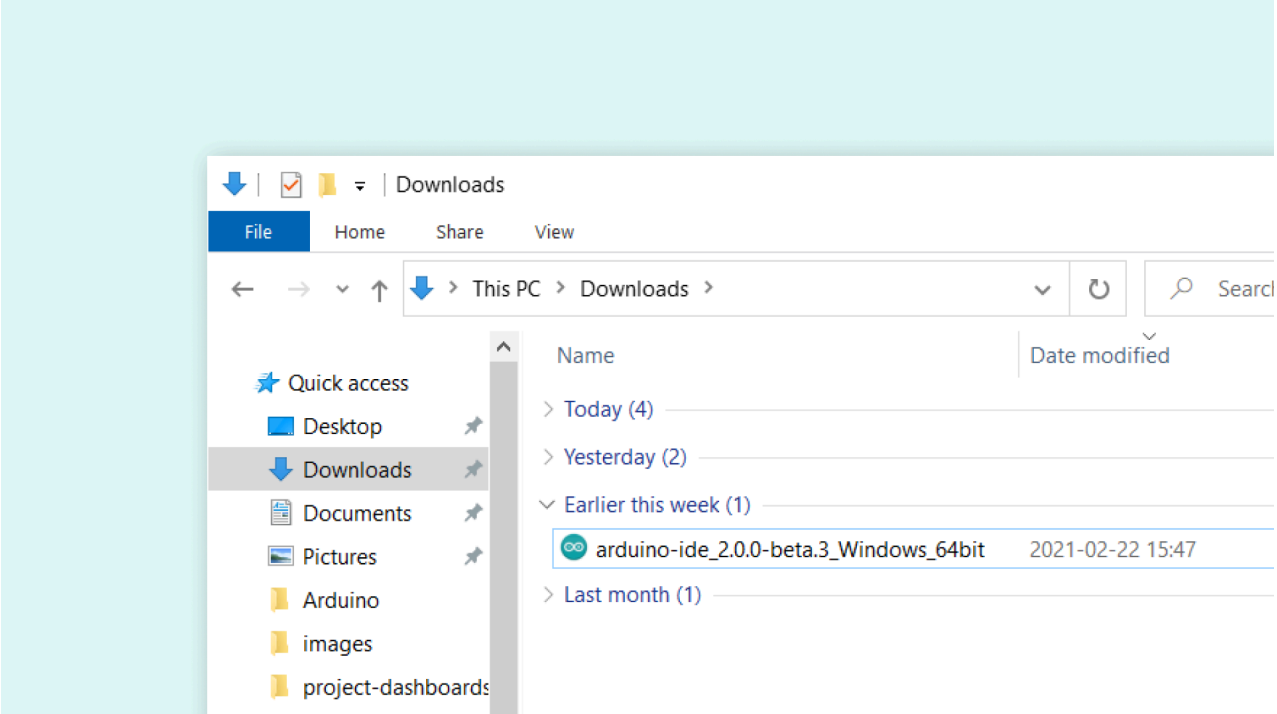
Chapter 2 - Installing Arduino IDE & Board Package

Step 1: Installing Arduino IDE

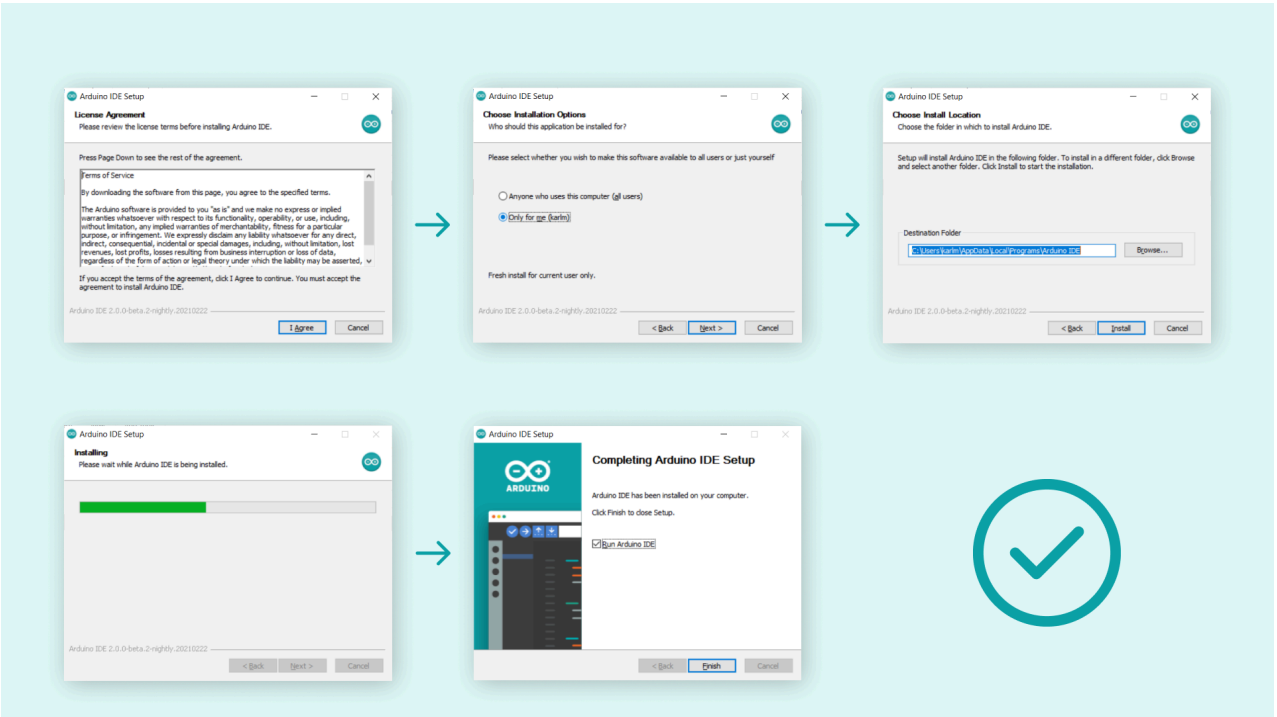
1. Download the Arduino IDE 2 from this page [Arduino Software page](#).
2. Install the editor on your computer, using the instructions for MacOS or Windows below:

Windows

To install the Arduino IDE 2 on a Windows computer, simply run the file downloaded from the software page.



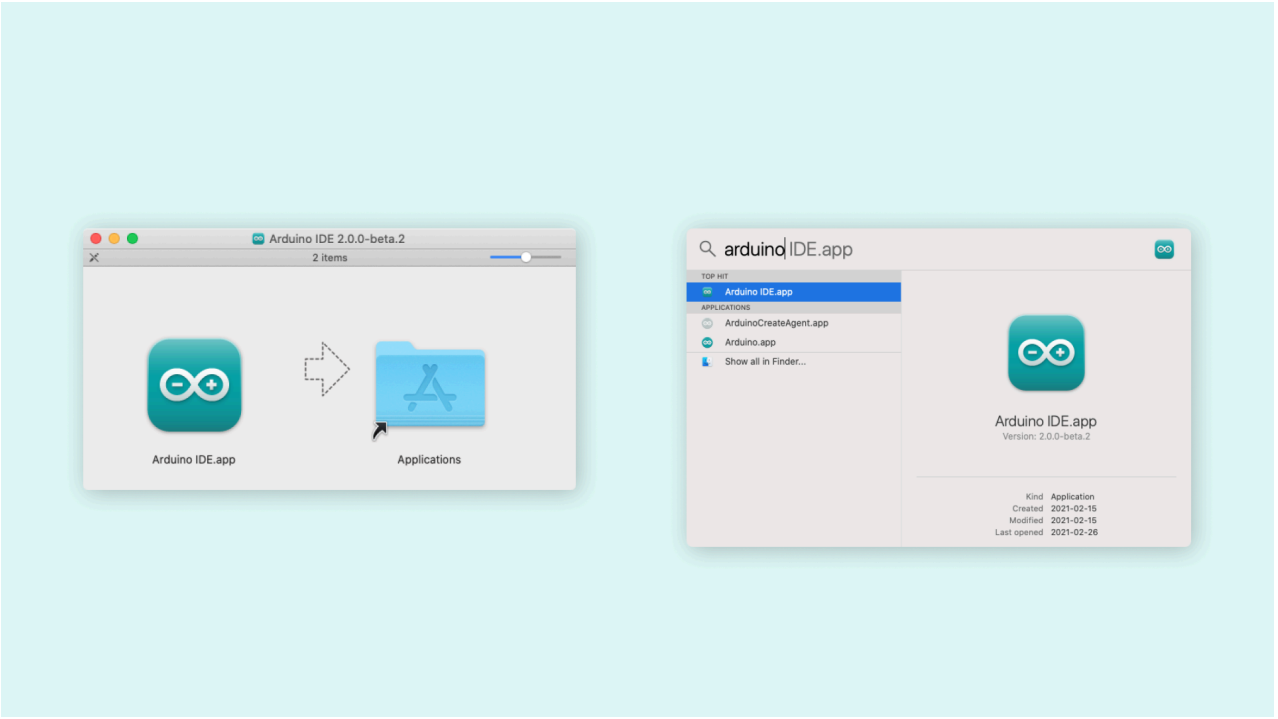
Follow the instructions in the installation guide. The installation may take several minutes.



You can now use the Arduino IDE 2 on your Windows computer!

macOS

To install the Arduino IDE 2 on a macOS computer, simply copy the downloaded file into your application folder.

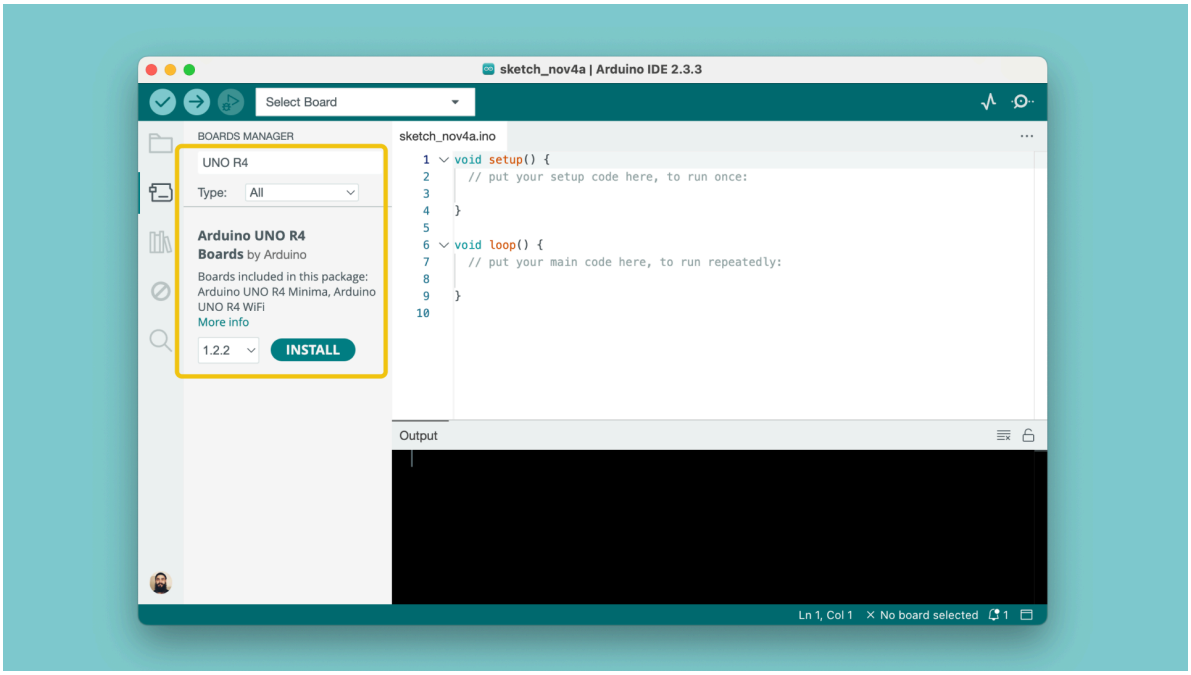


You can now use the Arduino IDE 2 on your macOS computer!

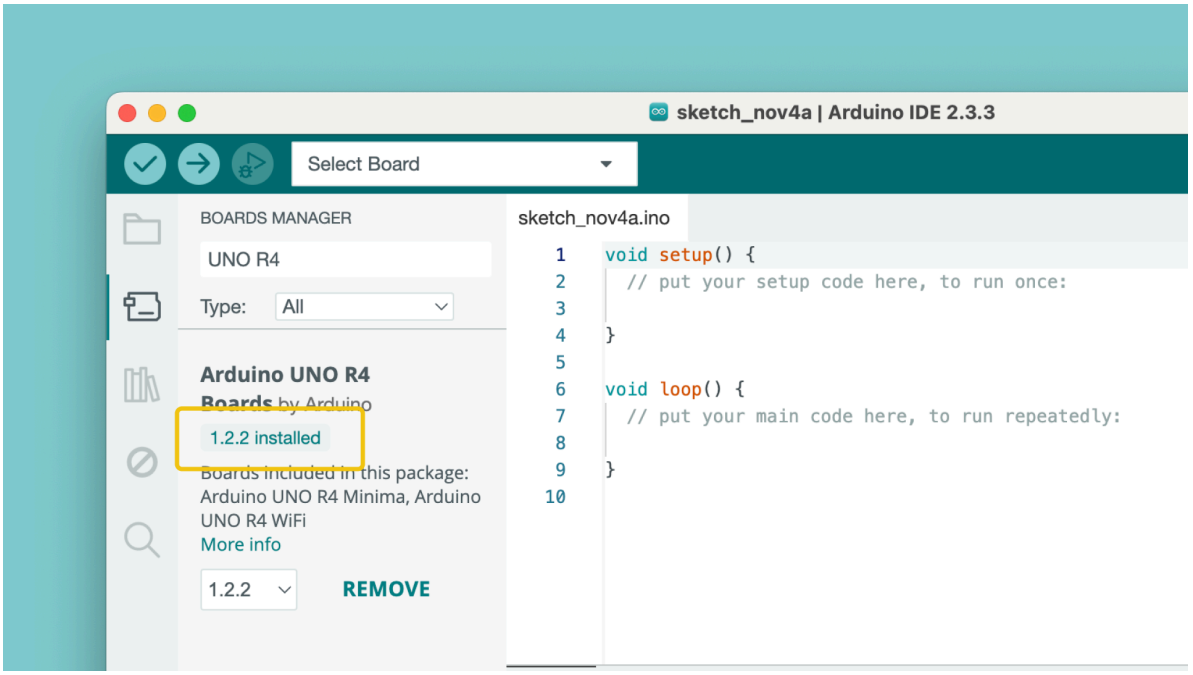
Step 2: Installing your Board Package

To use your specific Arduino board, you will need to install something called a **board package.**

1. Open the Arduino IDE.
2. Click on the "board" icon in the left menu. In the search field, enter **"UNO R4"**. The UNO R4 Board Package will now appear, and we can click the "Install" button.



3. The installation may take some minutes, and when it is finished, we should be able to see the version installed.

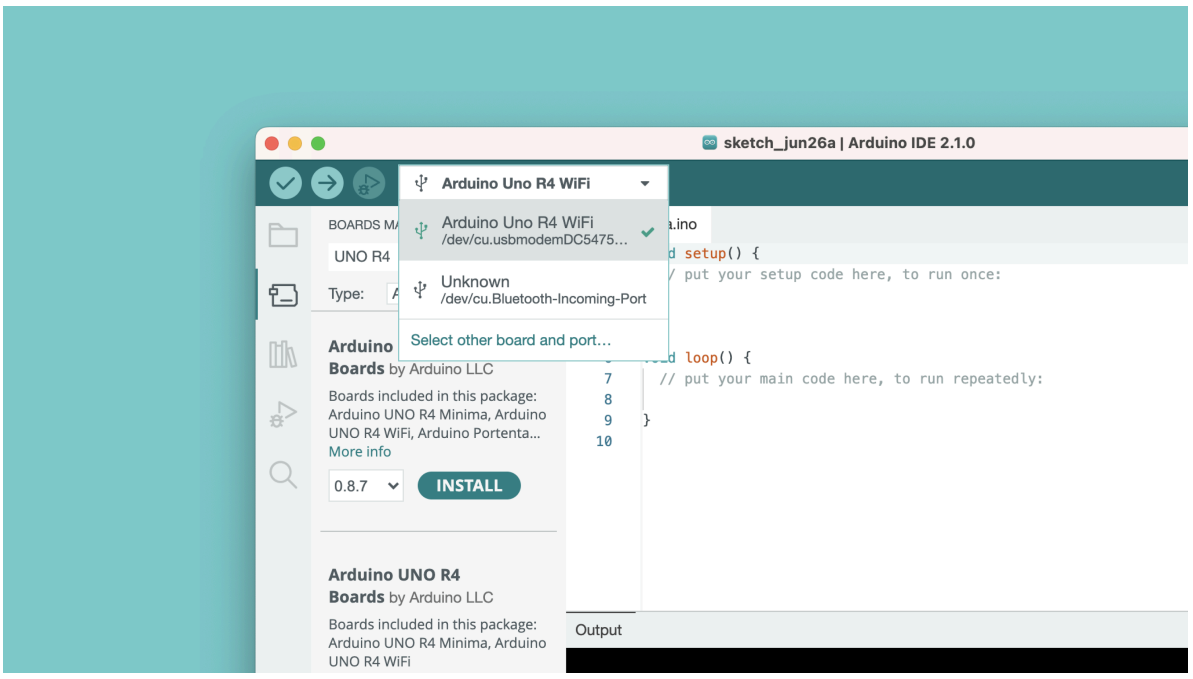


Congratulations! We have now successfully downloaded and installed a board package on your machine, and we can start using the Arduino board!

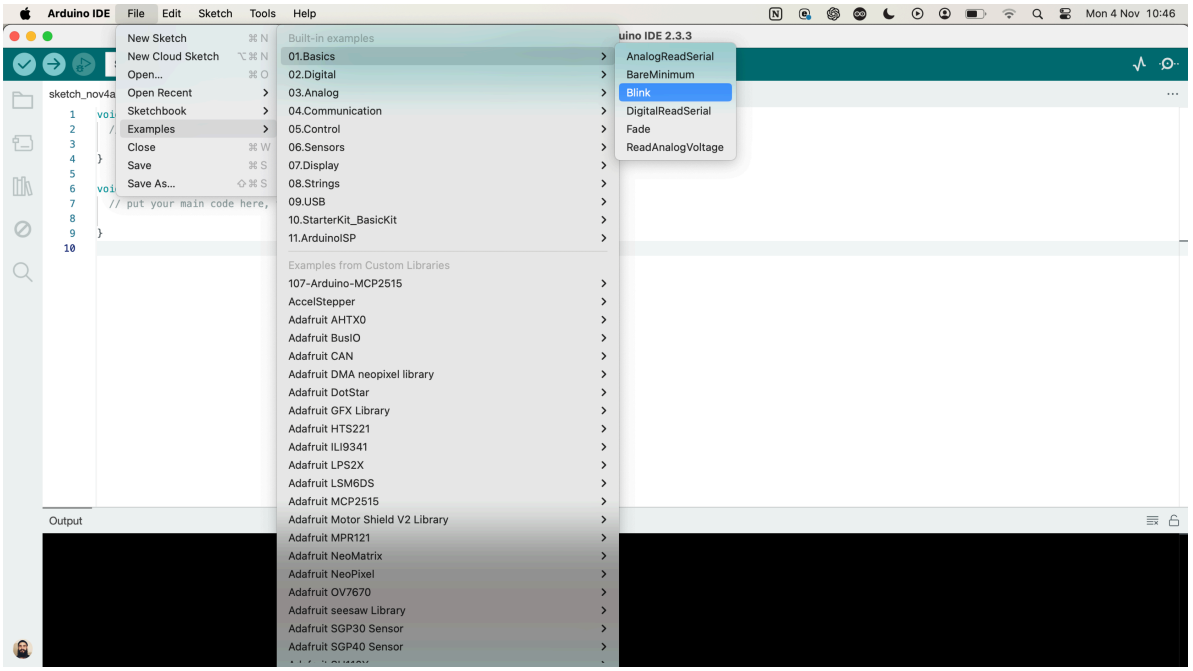
Step 3: Test the Board

Now that we have installed the board package, we can start using our board. To test it out, we will be uploading a simple example: **the blink sketch**. This will blink a light on the board every second, and is used primarily to test if everything went well.

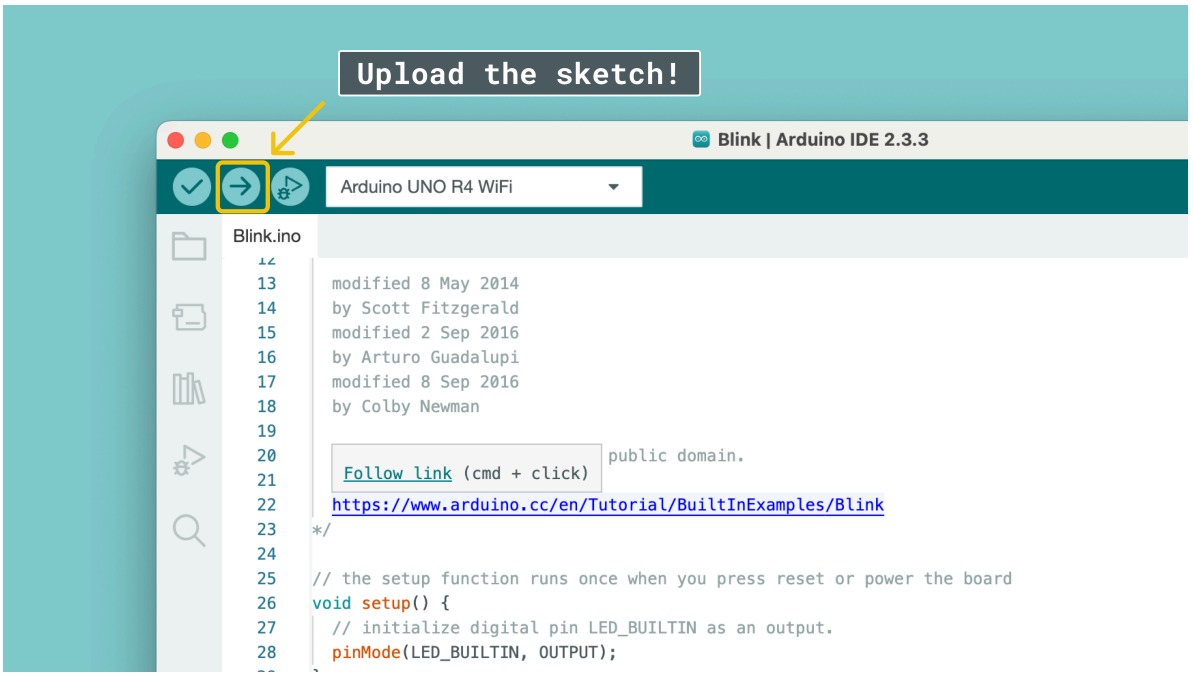
1. Connect the board to the computer. Once connected, we can click on the board drop down menu at the top left corner:



2. Now navigate to **File > Examples > Basics > Blink**. When we select it, a new window will open with a sketch example (Blink).



3. Finally, we are going to upload it to our board. Click on the "Upload" button as shown in the image below. A process will now start, and it is important to **not disconnect the board** during this process.



We should now see a light blink every second, which means the upload has been successful. We should also see a message in the black box (console) that confirms the upload was successful.

LEARN MORE: Arduino Libraries

An Arduino library is a collection of pre-written code that simplifies the process of interacting with specific hardware or implementing specific functionality in your Arduino projects. Libraries are designed to save time and effort by providing reusable and efficient code for tasks like controlling sensors, motors, displays, or even implementing communication protocols such as Wi-Fi or Bluetooth.

During this course we'll use some of them but here you can find a whole official list.

Chapter 3 - Arduino Cloud

In this chapter, we will focus on the Arduino Cloud. Arduino Cloud is a platform where you can:

- Create, edit and upload programs to your Arduino.
- Synchronize your code variables over Wi-Fi®
- View and control your board with visual dashboards.

The Arduino Cloud is **all online**, where you are only required to install a plugin to connect to your board. All your programs and configurations are automatically stored, meaning there is no risk of your project disappearing.

Inside the Arduino Cloud, there's something called the **"Cloud Editor"**. This is an editor that works exactly the same as the Arduino IDE, a software that you can download from Arduino Software page.

Create an Account

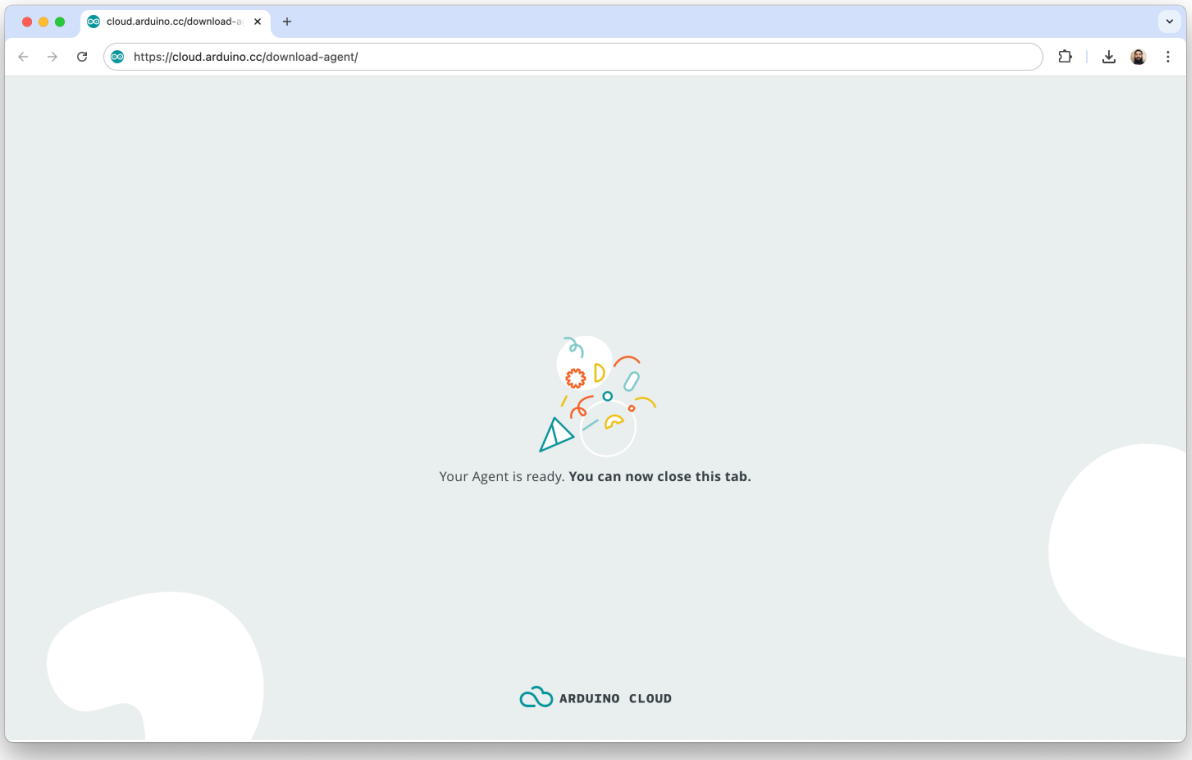
To use the Arduino Cloud, you will first need to create an account.

1. Go to the Arduino Accounts page and create an account. Note that you will need to be 14 years or older to create an account.
2. You will need to provide a valid email, create a new password, and confirm account creation in your email.

Install Cloud Agent

Once you have created an account, we are going to install the **Arduino Cloud Agent**. This plugin will allow your browser (such as Google Chrome) to communicate with your Arduino.

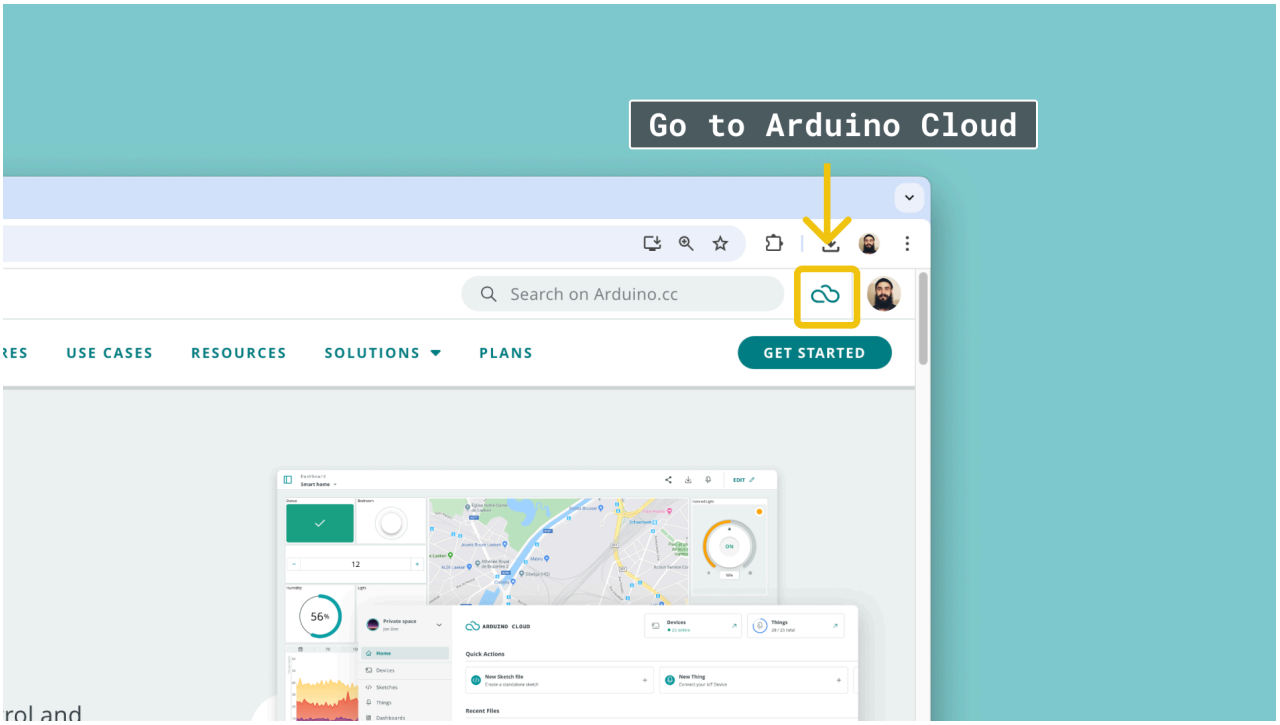
1. Go to the Cloud Agent installation page
2. Follow the installation instructions. You will need to download the plugin, and then install it on your computer.
3. When successfully installed, it will look like this:



> *If you are having problems installing the plugin, you can visit this article.*

Access Arduino Cloud App

Whenever you are on the Arduino website, you can access the Arduino Cloud App, by clicking the **"Cloud Icon"** at the top right corner.



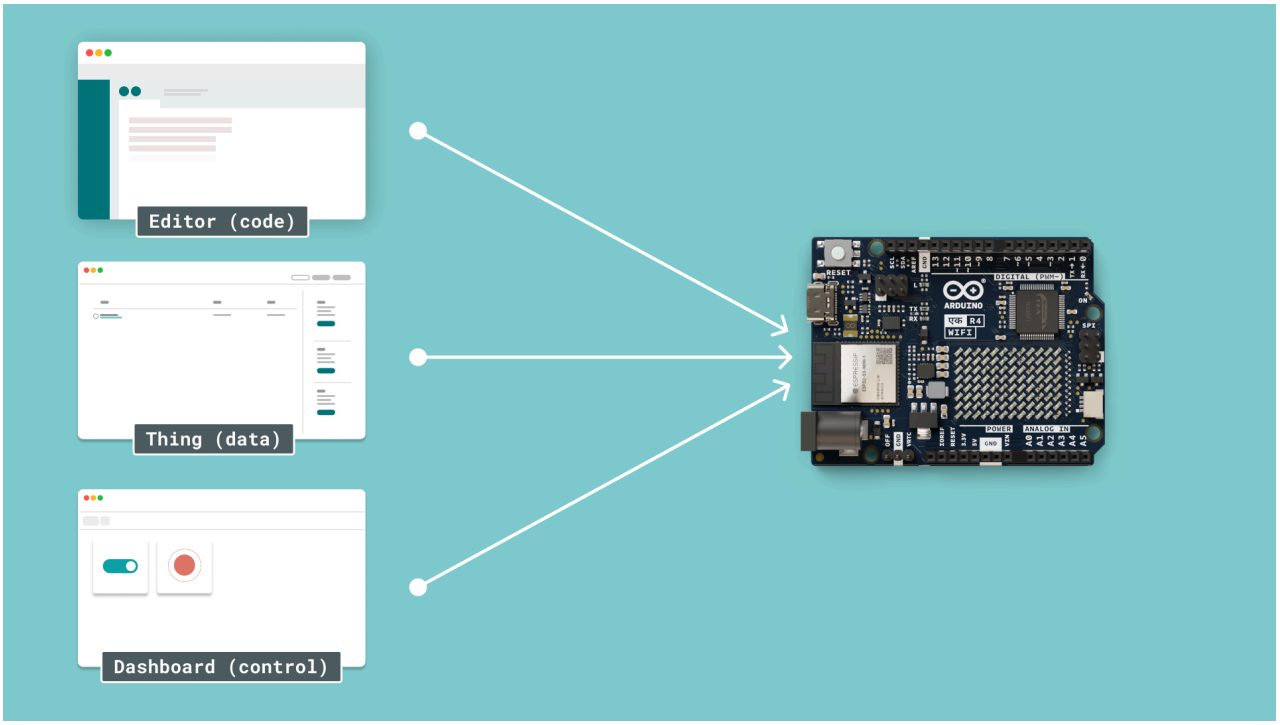
How Arduino Cloud Works

Now that your template is set up, let's take a look at how it actually works!

The Arduino Cloud consists of three main things:

- **Thing** - where we configure *data synchronisation* between board and cloud, as well as setting *Wi-Fi® credentials*.
- **Editor** - where we *create, edit and upload code* to our Arduino board.
- **Dashboard** - where we can *control* and *monitor* the data from our Arduino board.

All of these pages are accessible directly in the Arduino Cloud, from a sidemenu to the left.



Install a Cloud Template

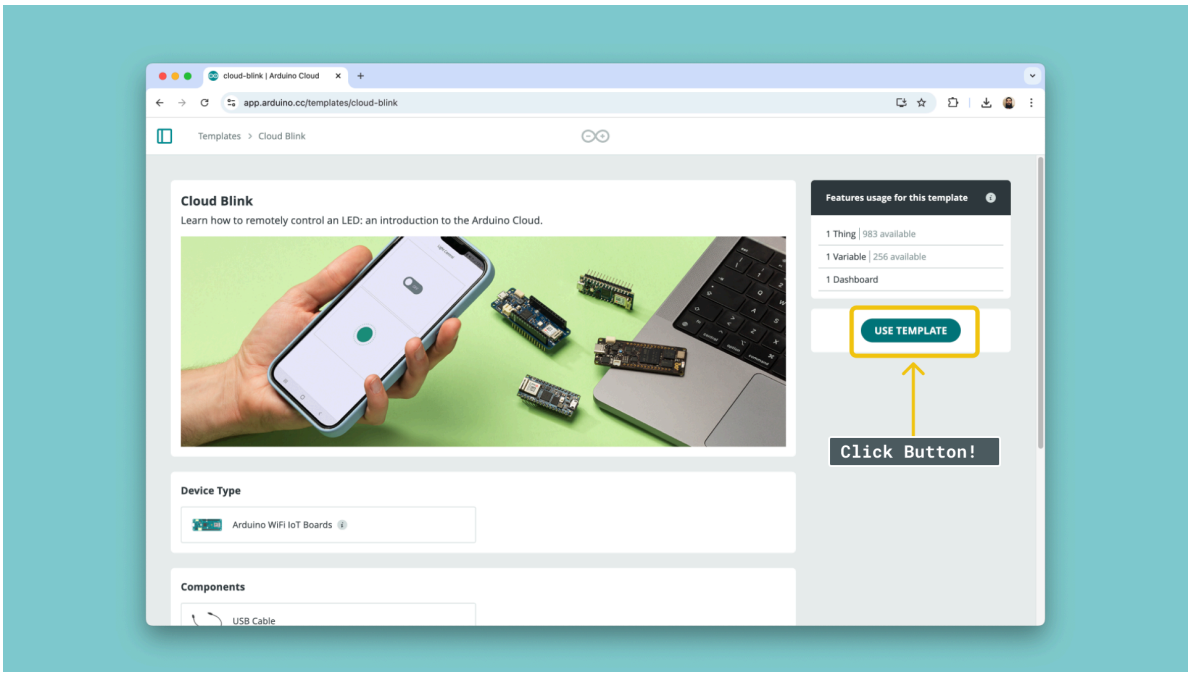
Now that we have the account set up, and plugin installed, let's finally set up our board so that we can control it.

We will install something called a **template**. A template is pre-made configuration that:

- Upload a special sketch to your board (so that it can connect to the Arduino Cloud).
- Creates a **"Thing"**, that is used to synchronize variables between the board and the Arduino Cloud.
- Creates a **"Dashboard"** that is used to control your board.

When the template has been installed, you will be able to communicate with the UNO एक R4 WiFi board, using the Arduino Cloud.

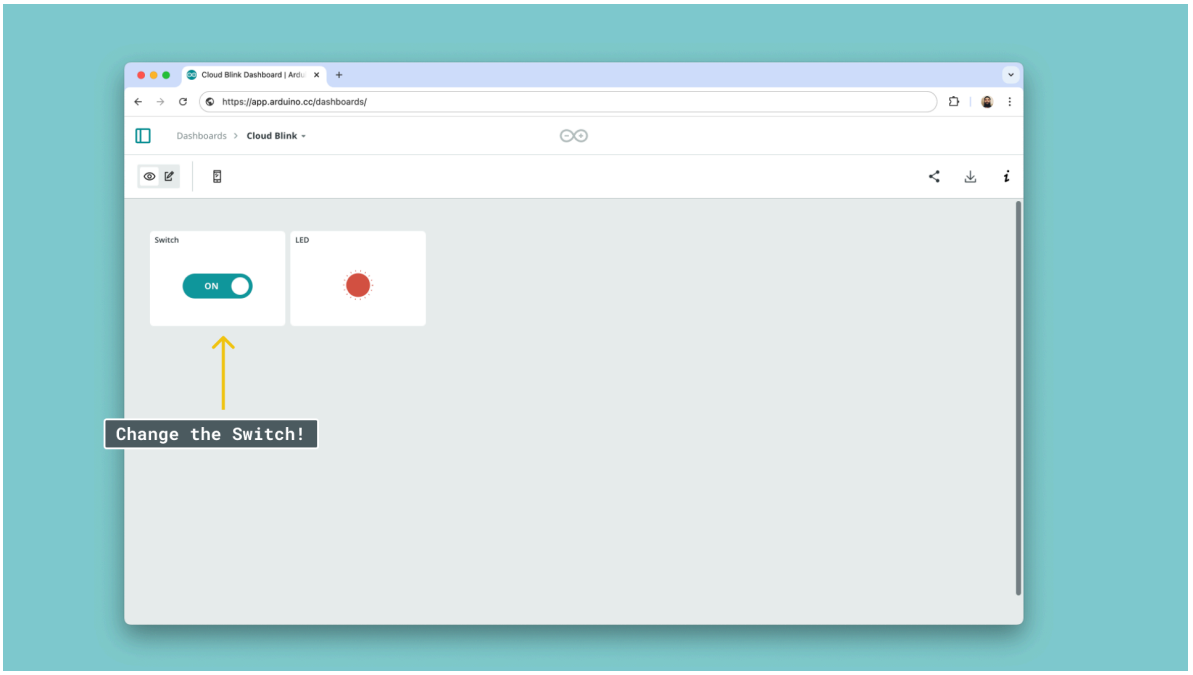
1. First, go to the Cloud Blink Template. Click on the **"Use Template"** button to the right



2. Follow the installation flow. You will need to set up your UNO एक R4 WiFi device, enter your Wi-Fi® network name and password during this installation.

3. When you finish the installation, you will then be directed to a new page, where you should see a dashboard. Wait for a while, then try to change the switch. A light on your board should now go ON / OFF depending on what you set the switch

to.

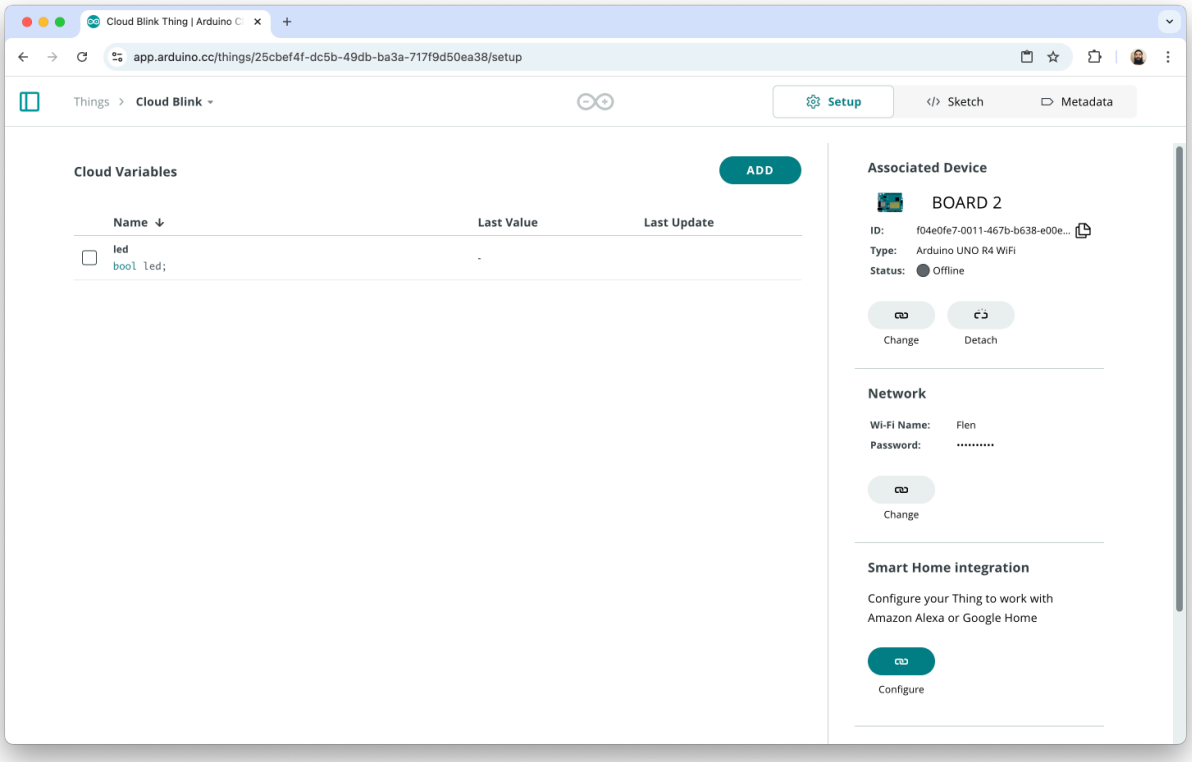


Congratulations! You have now configured your board to communicate with the Arduino Cloud. In the next section, we will take a look at how you can modify your program running on your board.

> ***Troubleshooting:*** *the Cloud Template import may fail. You can try re-running the template installation. If during the installation, you get the error "Board not found", try resetting your board by double-tapping the reset button on the board. For further support visit our Help Center.*

Thing

A "Thing" is your configuration space for your project.



Inside the **Thing** page, you will notice that there is something called **Cloud Variables.** A cloud variable is a virtual variable that will be synchronized with the variables on your board.

Variables are something you will use all the time when programing. They are a kind of container for different types of data. For each variable you need to specify the type of data it will contain; the name of the variable; and the value to assign to it.

For example:

- If the `led` variable is `true`, it will also be `true` on your board.
- If the `led` variable is changed to `false` from the dashboard, it will update your *board's* variable to `false`.

There are several **different types of variables**, including:

- `boolean` - a two state variable that can be either `true` or `false`. This is the variable used to control the LED.
- `int` - a variable that is used to store numbers, such as `123456`.
- `float` - a variable that can store values with decimal points, such as temperature: `22.25`.
- `String` - can be used to send a text message between the cloud and board, such as `"Hello, I am a string"`.

There are also two more categories of variables that you can create:

- **Specialized** - for specific use, e.g. temperature, energy, pressure.
- **Complex** - complex variables wraps *multiple* variables. For example `DimmedLight` has two variables: `boolean` and `float`.

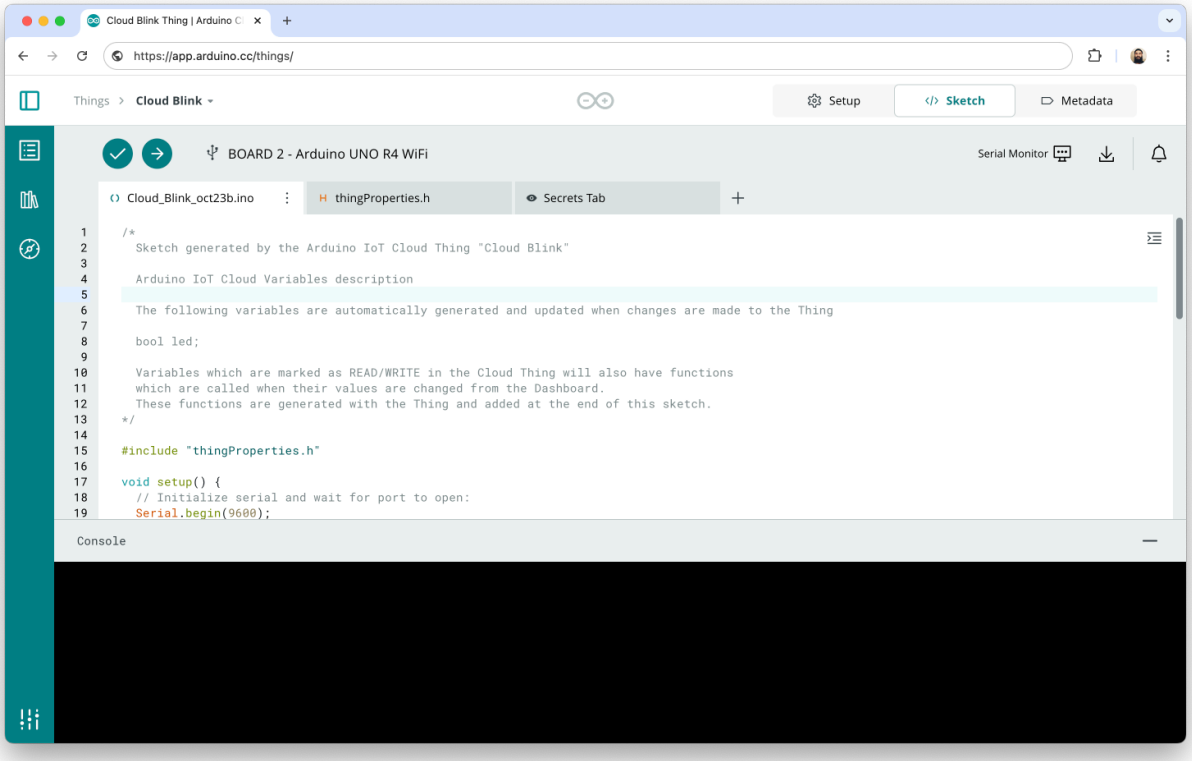
Cloud variables can also have one of the following permissions:

- **Read/Write** - data can flow both to the board from the cloud, and from the board to the cloud.
- **Read Only** - data can only flow from the board, to the cloud.

Editor

In the editor, you can write your programs, and upload them to your board. This works almost exactly like the regular Arduino IDE, but all sketches are stored online, and you do not need to install any libraries.

The Cloud Editor is accessible through your Thing page, by clicking the **"Sketch"** tab in the top right.

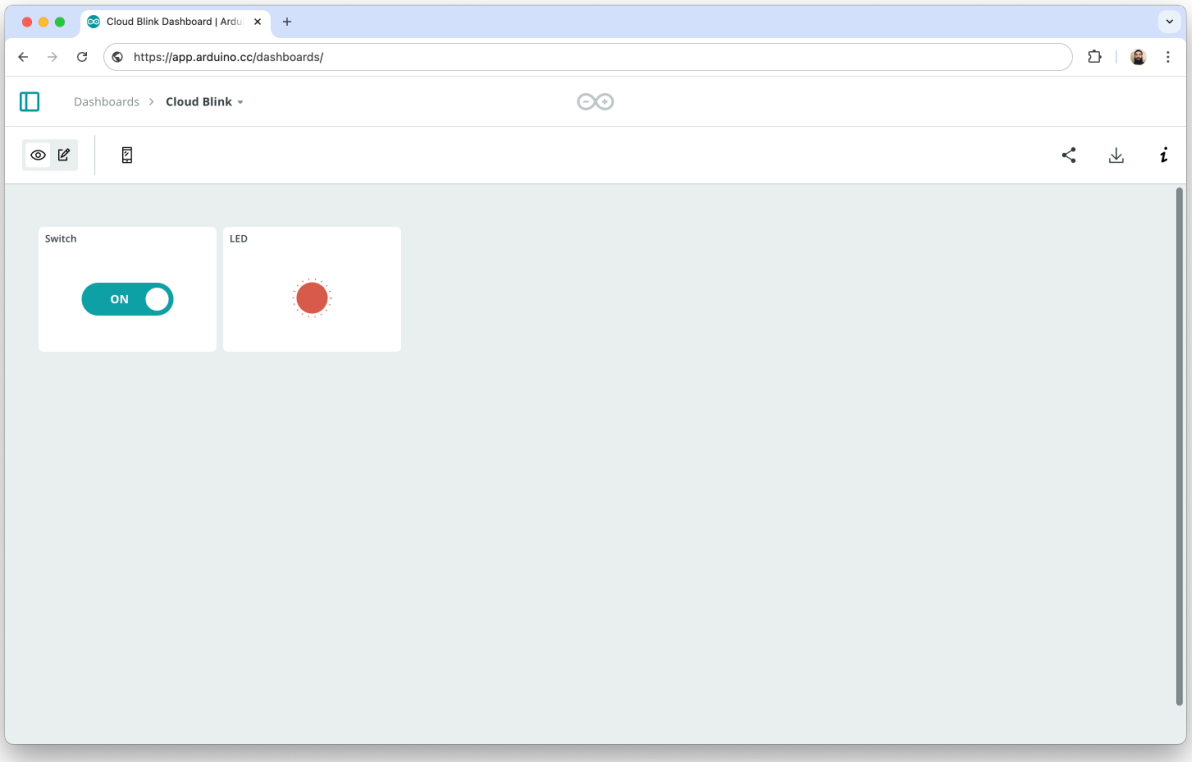


Dashboard

A dashboard is used to control and monitor your projects, via something called *widgets*. Widgets are directly linked to your variable (inside your Thing), and can either display data, or you can modify data.

A dashboard can have widgets that are linked to variables from different Things, so that you can control multiple Arduino boards from the same page.

Dashboards are accessible through the side menu on the left.



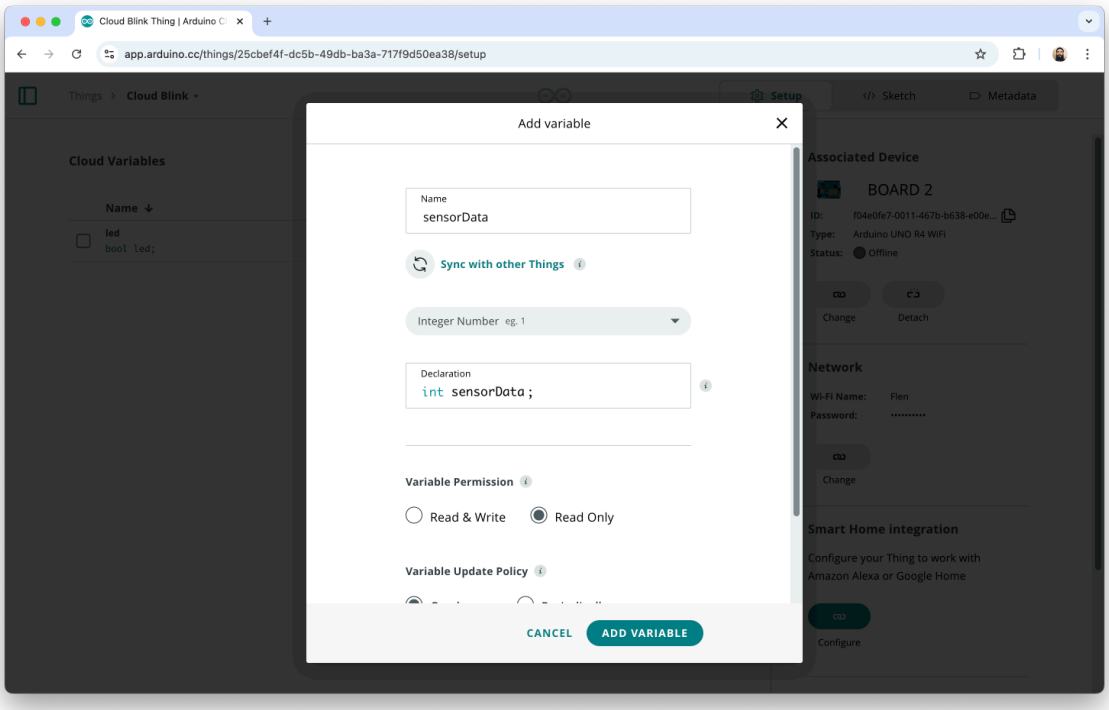
You can also access your dashboards through an app called **IoT Remote**. These can be downloaded here on your smartphone:

- Google Play Store
- Apple App store

Modifying the Arduino Cloud

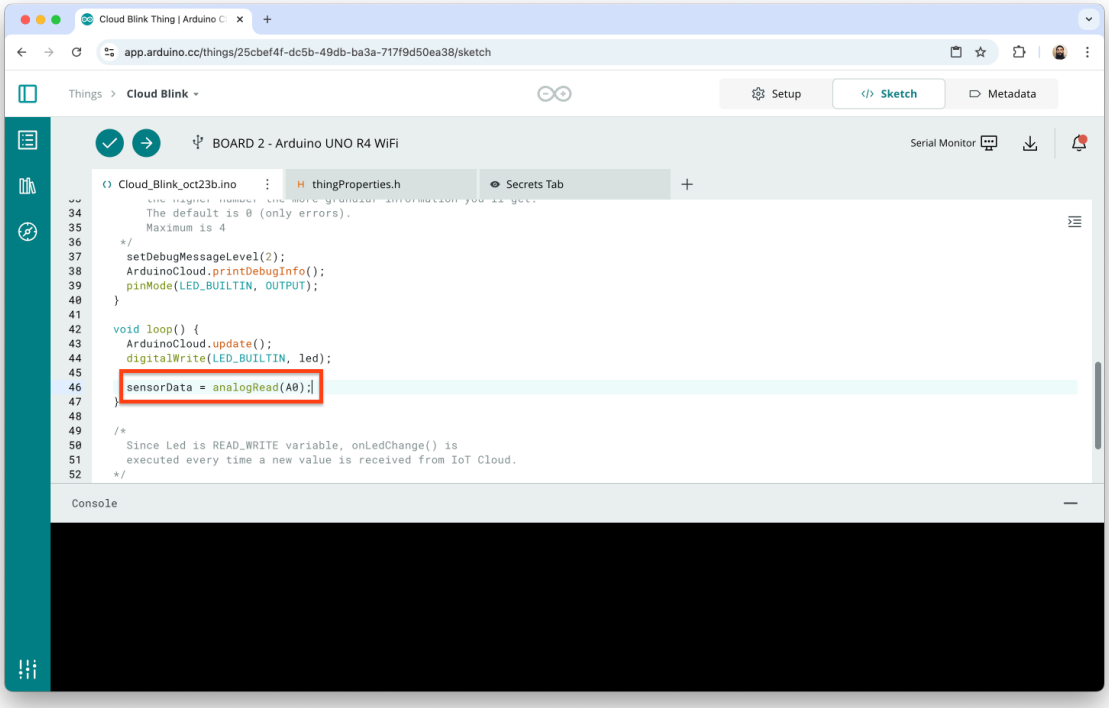
Now that you already have something working, you can start to edit and create new connected projects. This is how you do it:

1. First, add a variable inside the **Thing**, by clicking the **"Add"** button. You will find your Thing at the Arduino Cloud - Things page. Let's add a variable called `sensorData`, of the type `int`, and with a **Read Only** permission

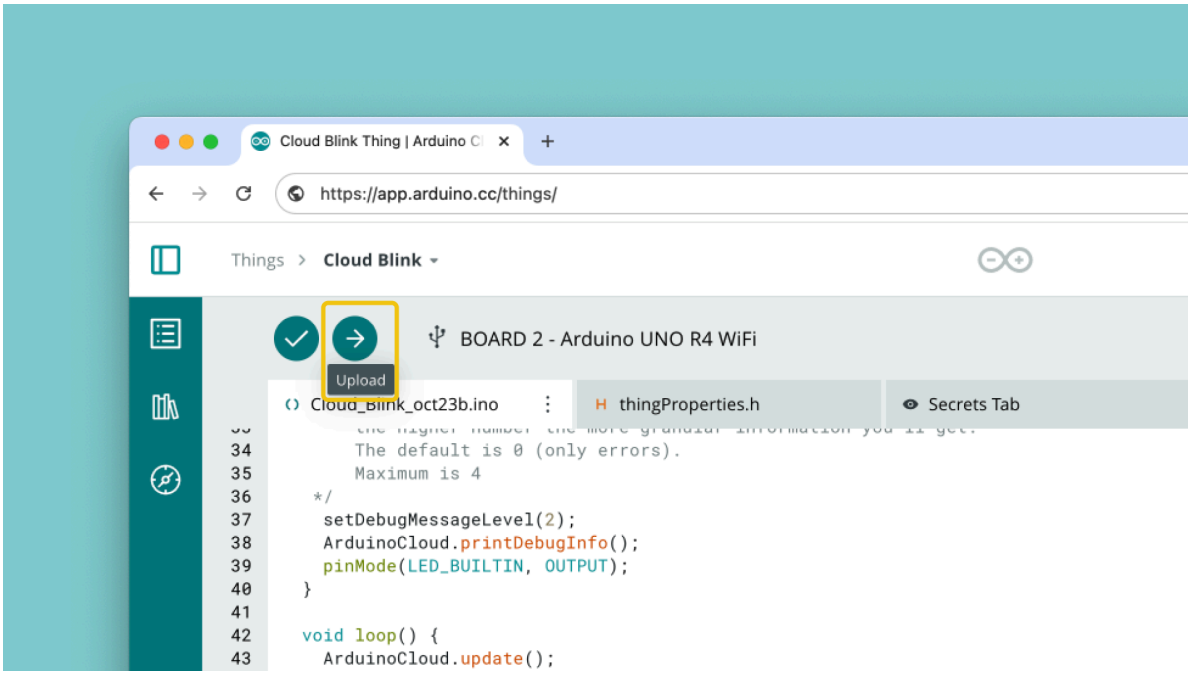


2. Then go to the **"sketch"** tab, and add the following line of code inside your `loop()` function.

```
sensorData = analogRead(A0);
```

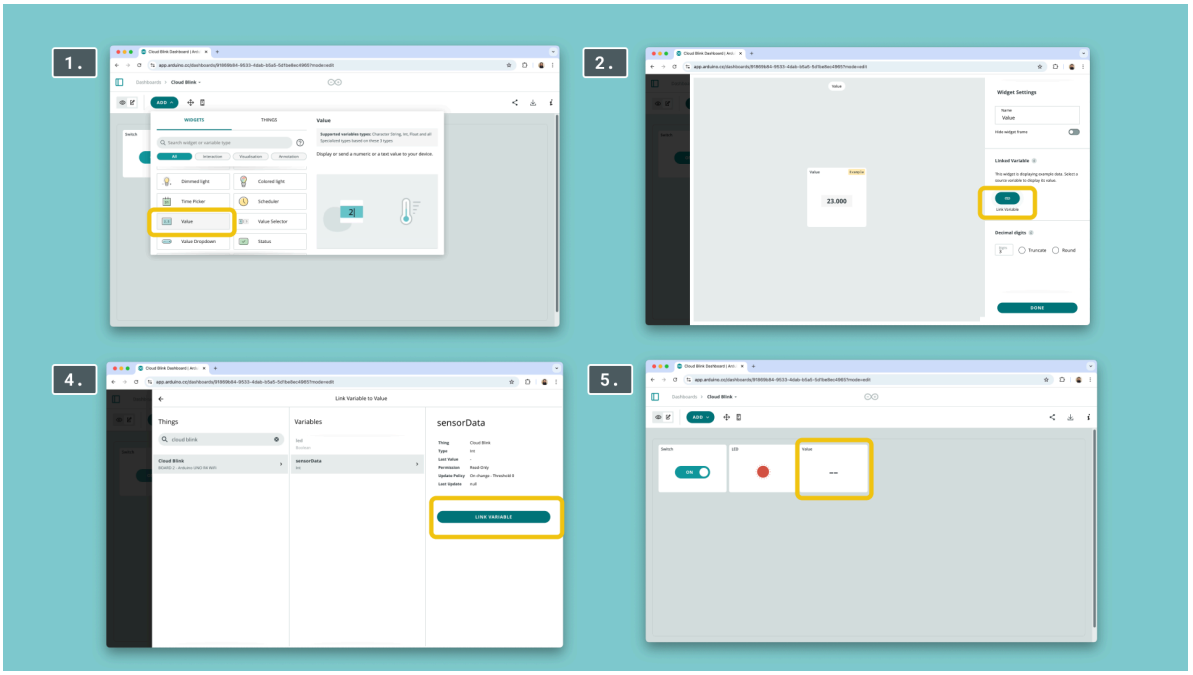


3. Upload the sketch to your board by clicking the **"Upload"** button (right arrow). Wait until it is finished.



4. Finally, let's go to the **"Dashboards"**. Navigate through the side menu, and click into your Dashboard. If you installed the template previously in this chapter, it should be named **"Cloud Blink"**.

5. We will now create a new widget that will be linked to the `sensorData` variable. Click on **"Add"**, create a **"Value"** widget, and link it to your `sensorData` variable.



6. Done! Congratulations, you have now added a new widget to your dashboard. You should now see some random value streamed to the dashboard, because we have not connected anything to the `A0` pin on the board.

Further Reading - Arduino Cloud

The Arduino Cloud is a big platform, and in this chapter we have covered only some of the basics.

If you want to learn more, visit Arduino Docs - Arduino Cloud, where you will find 50+ pages that will help you navigate the platform!

Chapter 4 - Arduino Basics

In this chapter, we will learn a little bit about basic functionalities of an Arduino, focusing on three key aspects:

- Digital inputs and outputs
- Analog inputs and outputs
- Serial communication

These are fundamental to any Arduino project, and you will learn more in-depth about these concepts through a set of examples.

Digital vs Analog Signals

All communication between electronic components are facilitated by electronic signals. There are two main types of electronic signals: analog & digital.

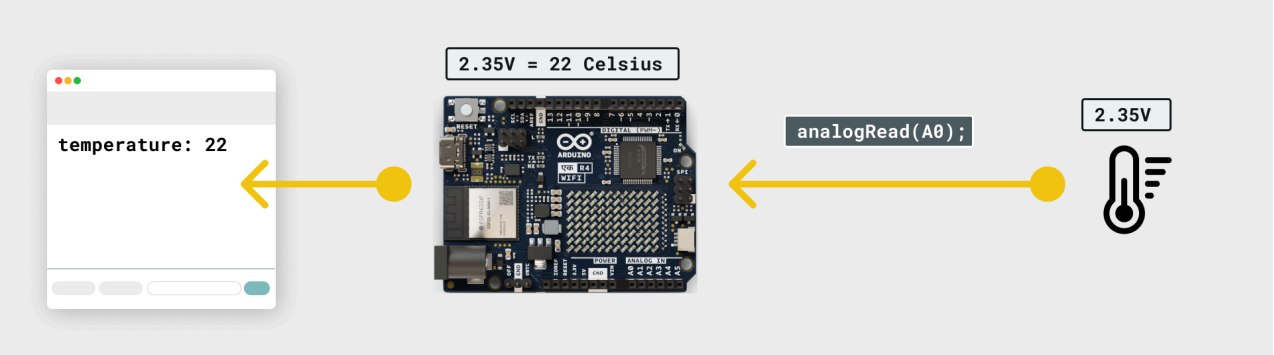
- An analog signal is generally bound to a range. In an Arduino, that range is typically 0-5V, or 0-3.3V. For example, when measuring a physical property, such as light, we can record it with an analog component.
- A digital signal works a bit different, representing only two binary states (0 or 1). These are read as high or low states in the program. This is the most common signal type in modern technology.

Analog Input/Output

To process an analog signal using an Arduino, we use something called an Analog-digital Converter (ADC). Simply put, an ADC takes an analog value (or the voltage), and converts it into a digital signal (numbers).

For example, in a range between 0-5V, the representation in numbers can be 0-1023, when using something called "10-bit logic".

- Let's say we have a temperature sensor, and the value range is 0-50.
- To find the current temperature, we would first read the analog value, and then re-map it to that range.
- This can be done with for example a map function:
- `temperature = map(analogReading, 0,1023, 0,50)`



In a similar fashion, if we want to generate an analog signal, we use something called Pulse-width Modulation (PWM). PWM allows you to control the voltage output (for example the speed of a motor), by rapidly turning something on and off very fast.

Digital Input/Output

Processing a digital signal is easier, as you only need to read a 0 or a 1 (a boolean state).

For example, if we want to read a button, we can check if it is ON/OFF by using:

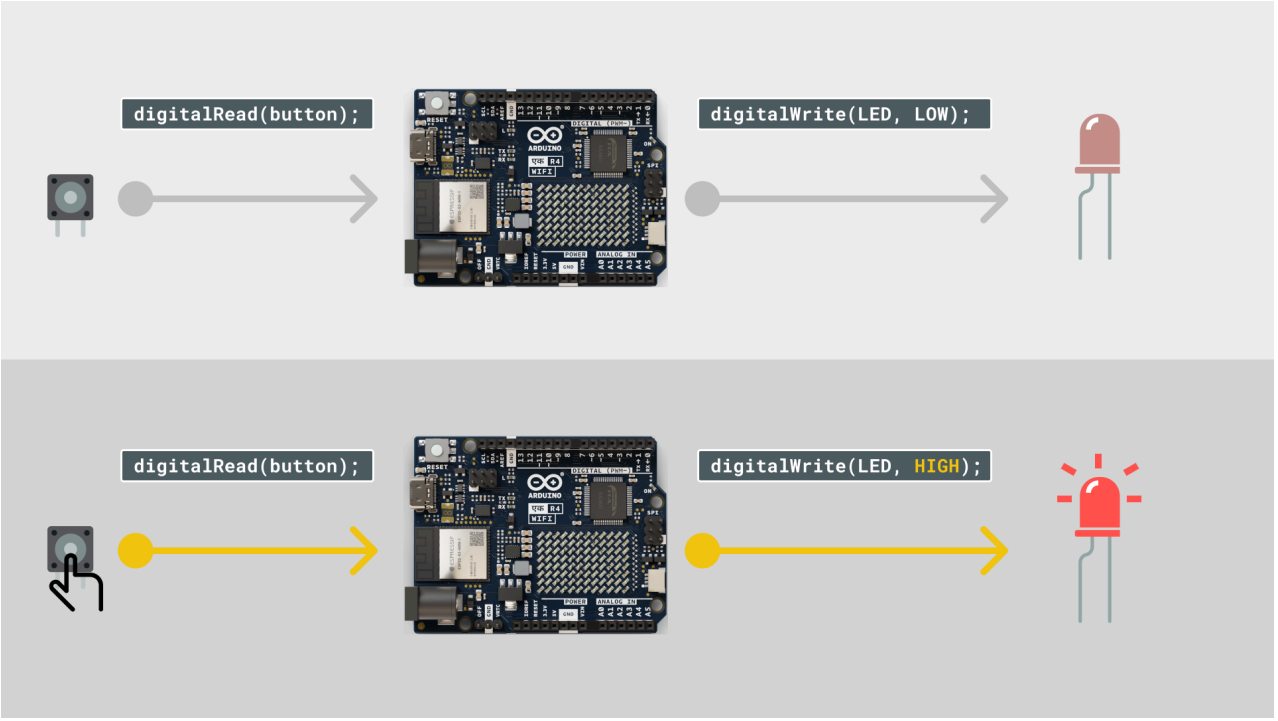
- `buttonState = digitalRead(button)`

And if we want write a digital signal, to for example turn on an LED, we can use:

- `digitalWrite(LED, HIGH)`

When you write a digital signal, you are essentially activiting a "path", that allows *current* to travel. If you have a component connected to that path (pin), it will activate!

Take a look at the image below to understand what is happening. We will revisit this example later on in this chapter.



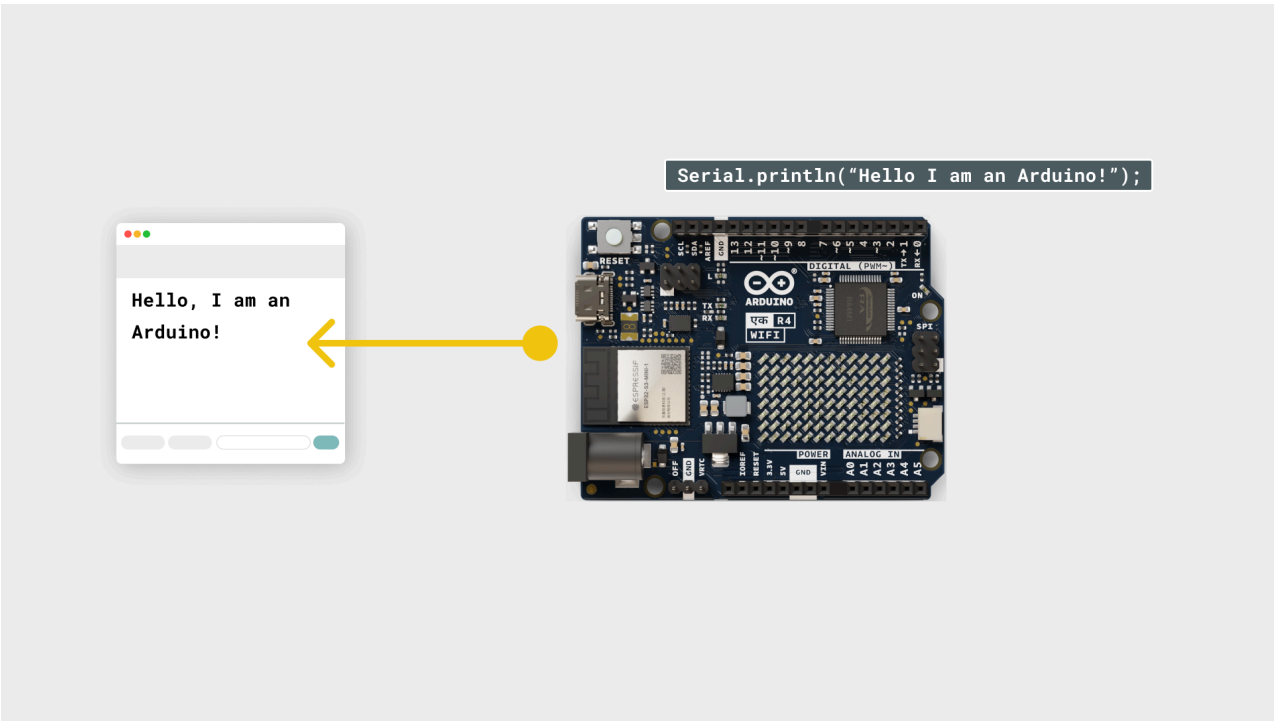
Serial Communication

Serial communication is also an incredibly important concept in Arduino - or in computer science in general - as it is how the Arduino can talk to other devices, such as your computer, using something called a **Serial Protocol**.

Let's take the example of reading temperature (see the section above). When the Arduino has recorded a value from its environment, how can we display that information?

The answer lies in the function `Serial.print()`. Whenever we want to send something from the board to a computer, we use that. A command can look something like the following:

- `Serial.print("Hello, I am an Arduino)` - we send a text message to the computer. Note that we use citation marks `""`, this indicates that we want to send only the characters between the citation marks.
- `Serial.print(temperature)` - we send a temperature value to the computer. Notice that we *do not* use citation marks, because we want to send the value of the `temperature` variable.



Basic Examples

Now that we know how some of the basic things works, we will dive into some practical examples that let's you try out all of the things you've learned so far.

Blink Example

The Blink Example makes the built-in LED on the board blink every second, by using something called delays.

Requirements

- Arduino board

Circuit

No circuit is required for this example.

Code Example

Upload the following code example to your board, using the Arduino IDE.

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

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
}
```

How it Works

After the program has been uploaded, a light on your board should be blinking every second. This is a classic test example to make sure everything works well. The light will continue to blink forever, and will do nothing else.

It works as follows:

- First, we write a HIGH state to the component called `LED_BUILTIN`. This is a small LED on the Arduino board.
- Then, we pause the program for one second using `delay(1000)`. 1000 represents milliseconds, where 1000 milliseconds = 1 second.
- After one second has passed, the program resumes, and writes a LOW state to `LED_BUILTIN`.
- We then pause the program for another second, and the loop starts over.

Analog Write

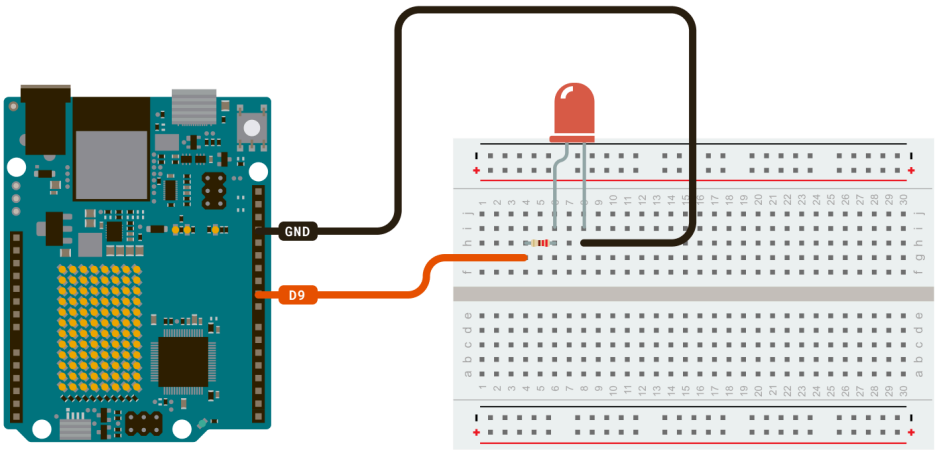
In this example, we will learn how to use `analogWrite()`, which allows you to control the intensity of an LED.

Requirements

- Arduino Board
- LED
- 220 Ohm resistor
- Breadboard
- Jumper wires

Circuit

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



Code Example

Upload the following code example to your board, using the Arduino IDE.

```
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);
}
```

How it Works

After uploading, you should observe the LED's behavior. The LED will slowly fade in (get more bright), before fading out (get less bright). It will keep on doing this forever.

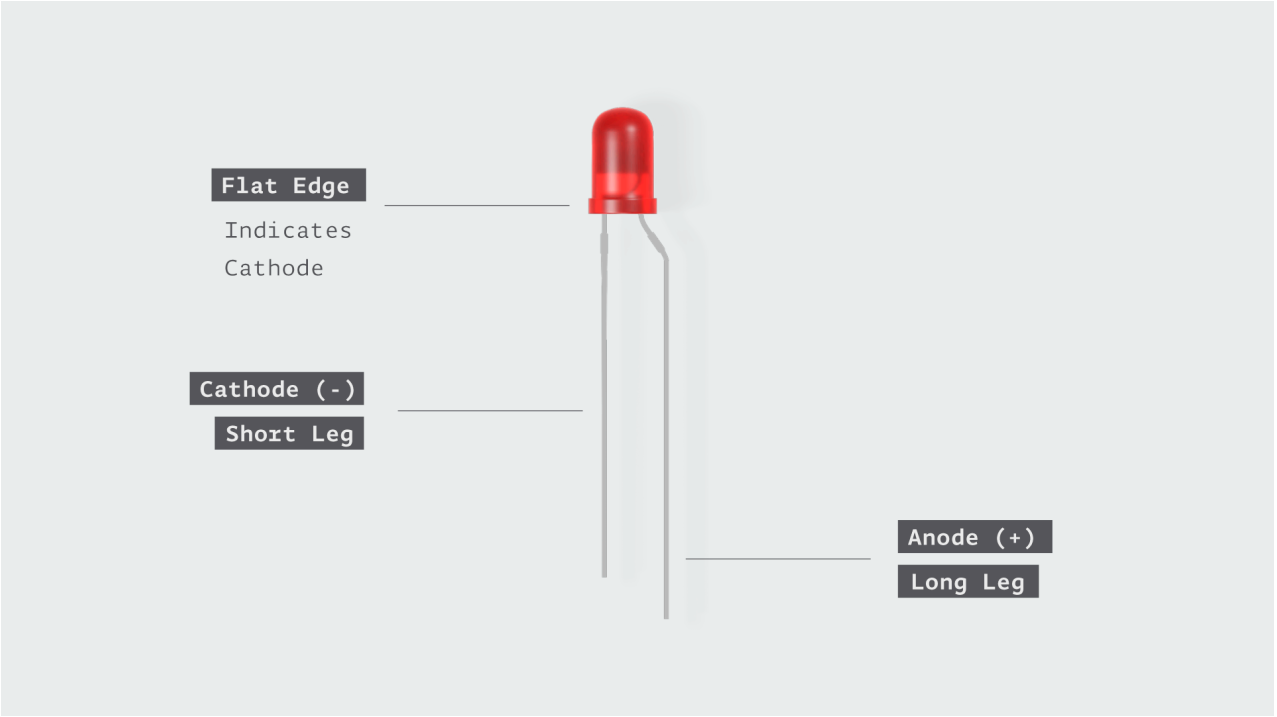
How we make this happen is by everytime the `loop()` function runs, we either increase/decrease the brightness by `5`. When using `analogWrite()`, we have a range of 0-255 (0-5V). This is because we use something called an 8-bit logic. Whenever the value is at 255 it will be at its brightest, and at 0, it will be off.

Note that once the `brightness` variable reaches 255, we switch the direction, instead decreasing the brightness with every loop.

Since the `delay(30)` is used at the end of the loop, it means it will get brighter every 30 milliseconds.

LEARN MORE: How a LED work

The LEDs usually have two wires, one short and one long. The long wire is called the **anode**, and the short wire is called the **cathode**. On some LEDs, the cathode side of the LED has a flat edge. Current must enter the LED through the anode and exit through the cathode. Therefore, the anode (long wire) should be toward the positive terminal of the breadboard, and the cathode (short wire) should be toward to ground (negative terminal).



LEDs come in different shapes, sizes, and colors. In the activity in the introduction, you caused the built-in LED on the Arduino UNO R4 board to blink. Although that LED is much smaller and a different shape, it works the same way.

Analog Read

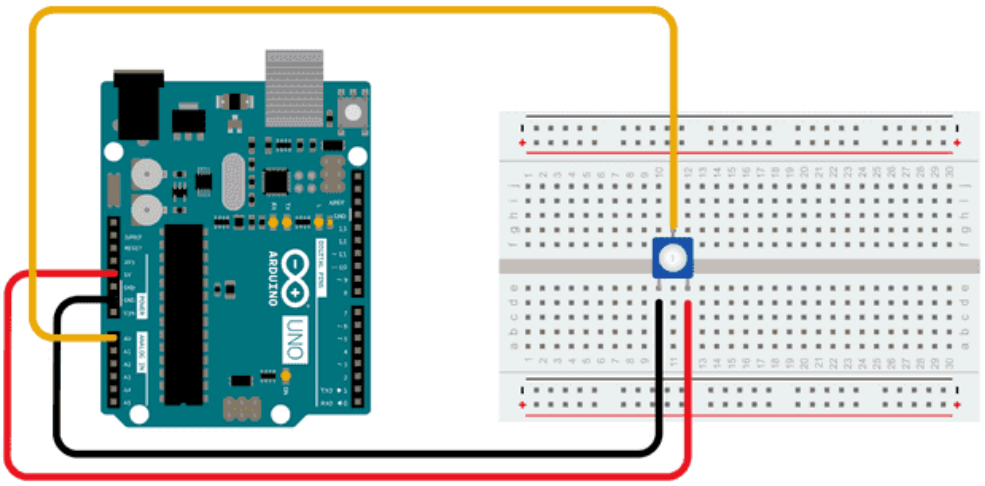
In this example, we will use something called a **potentiometer** to record analog values and send them to the computer using serial communication.

Requirements

- Arduino Board
- Potentiometer
- Breadboard
- Jumper Wires

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.



Code Example

Upload the following code example to your board, using the Arduino IDE.

```
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

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
}
```

After uploading, open the **Serial Monitor**, and see the values change when you turn the potentiometer.

How it Works

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.

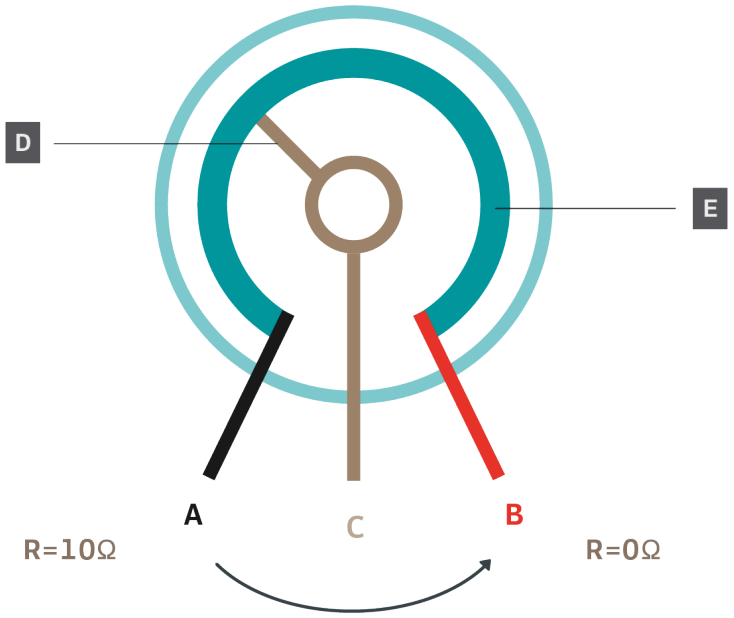
The Arduino boards have a circuit inside called an *analog-to-digital converter or ADC* that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft 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.

LEARN MORE: What is a potentiometer

A potentiometer is a type of variable resistor. In other words, it can be changed to have no resistance or to have a great deal of resistance. The potentiometer has two pins on one side and a single pin on the other. For some of them you might need to attach a

white knob to the peg of the potentiometer.

Inside a potentiometer is a resistive material and wiper that slides along this material. Each end of the resistive material is connected to a pin or terminal. In the following graphic, these would be Pins A and B. The resistance between Pins A and B is fixed and is the maximum amount of resistance a potentiometer can add to the circuit.



In the image above *D* is the wiper and *E* the resistive strip.

A third pin (Pin *C* in the graphic) is connected to a wiper. The resistance through the wiper, or Pin C, depends on the position of the wiper as it contacts the resistive material. The more of the resistive material the current must pass through before exiting the potentiometer through the wiper, the higher the resistance through Pin C.

So, potentiometers change the resistance in the circuit. However, microcontrollers don't read resistance; they read voltage and we use a digital pin to read the voltage as either HIGH or LOW.

Read Button Press

In this example, we will be reading the state of a pushbutton connected to a digital pin, and send the information to the computer.

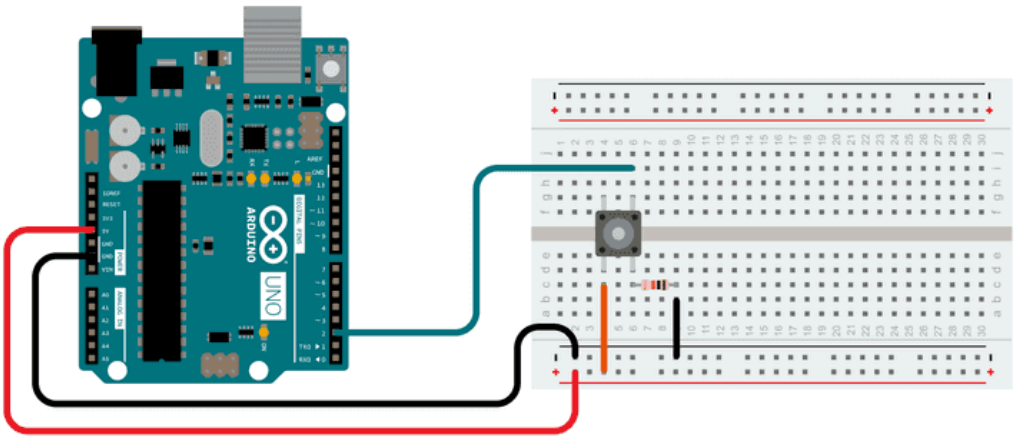
Requirements

- Arduino Board
- A button
- 10k ohm resistor
- hook-up wires
- breadboard

Circuit

Connect three wires to the board. Connect the first two wires (red and black) 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 (blue) 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.



Code Example

Upload the following code example to your board, using the Arduino IDE.

```
int pushButton = 2;

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

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
}
```

After the program has been uploaded, open the Serial Monitor. Now, try pushing the button. You should see that the state printed will change from 0 to 1.

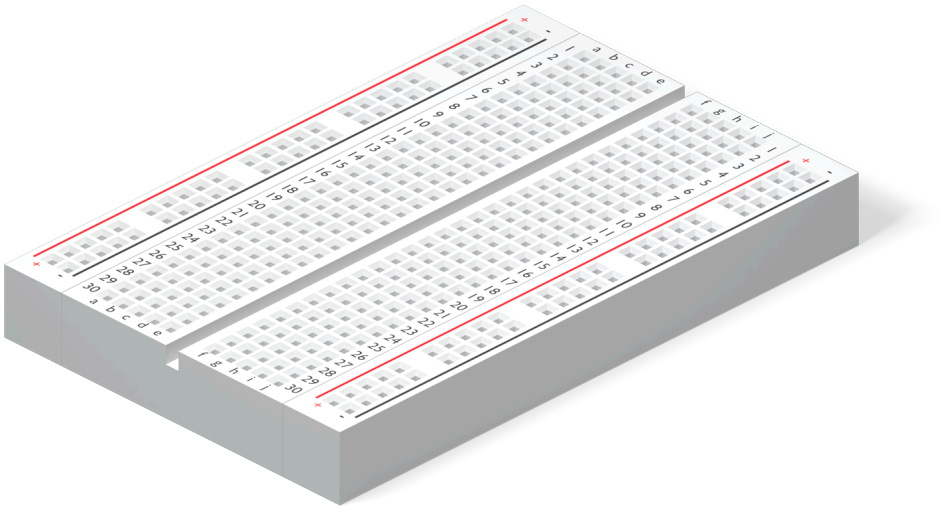
How it Works

In this example, we read the digital state from the button with `digitalRead(pushButton)`, and print it to the computer directly using `Serial.println(buttonState)`.

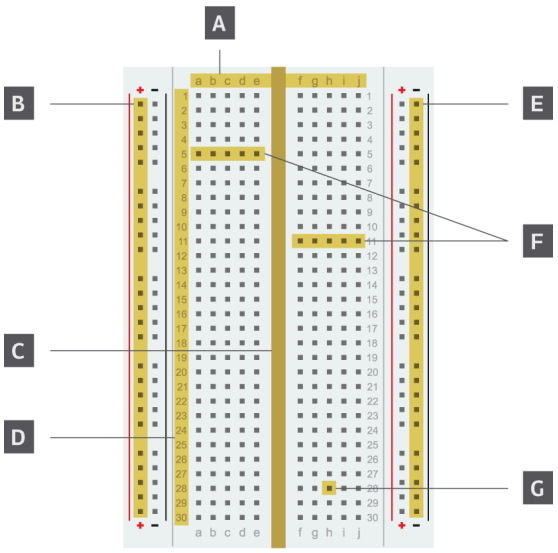
LEARN MORE: a Breadboard

A **breadboard** is a tool you can use to build electronic circuits. It gets its name from long ago when people used to build electronic circuits on actual pieces of wood that were used for cutting loaves of bread. Circuit breadboards have come a long way and have become useful tools for creating electronic **prototypes**.

The breadboard you'll use is called a solderless breadboard. Each hole on the breadboard contains a metal connector that pinches wires when wires are inserted into the holes. This helps keep the wires from pulling out and gives the circuit secure connections.



Holes in the breadboard are connected in different ways:



Location	Component	Description
A	Column IDs	Each vertical column in the prototyping area is labeled with a letter.
B	Positive terminal	All the holes next to the red vertical stripe with a + sign at the top are connected. These holes are used to supply electricity to the circuit.
C	Non-conductive Gap	This middle section divides the board into two unconnected sides.
D	Row IDs	Each horizontal row in the prototyping area is labeled with a number.

E	Negative terminal	All the holes next to the black (or blue) vertical stripe with a - sign at the top are connected. These holes are used to ground the circuit.
F	Connected Rows	In each horizontal row of five holes, the holes are connected.
G	Prototyping area	The center section of the board is the prototyping area. Any hole in the prototyping area can be referenced by a number-letter combination. The designation for this hole is 28h.

Chapter 5 - Arduino UNO एक R4 Series

The Arduino UNO R4 Series has many features compared to its earlier versions. These features can help us create more advanced projects without needing any additional electrical components.

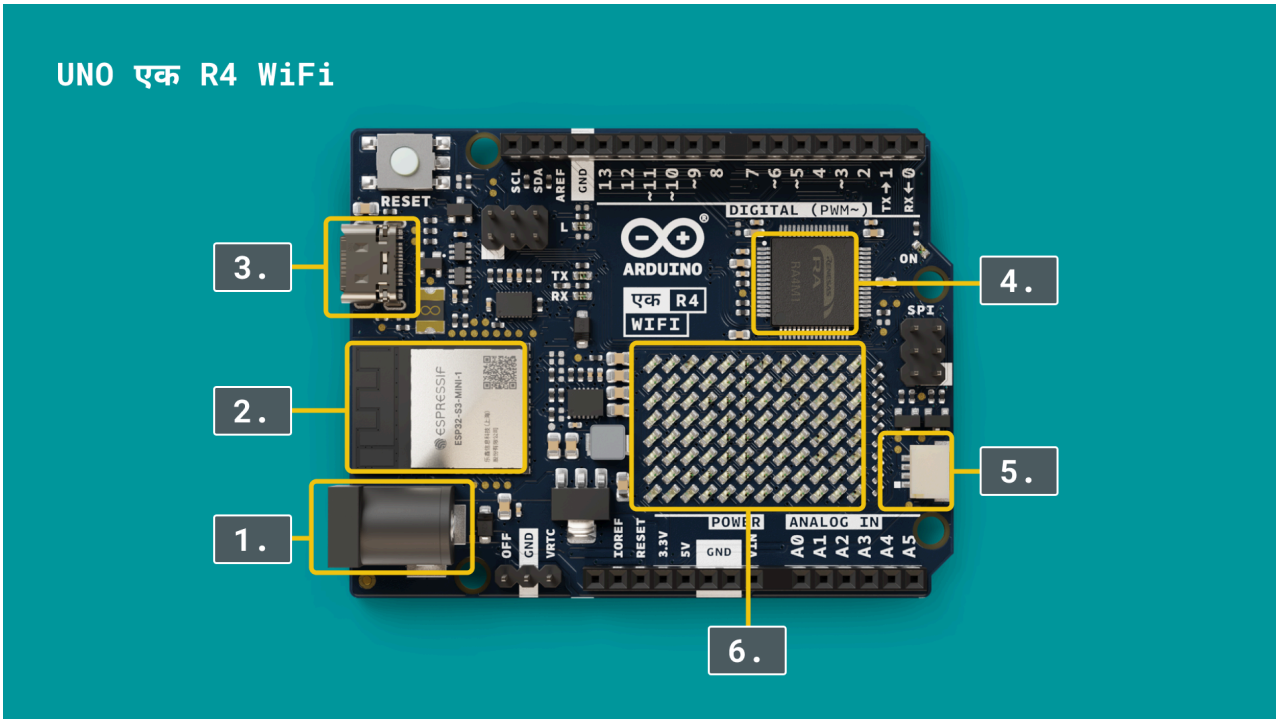
The R4 Series consists of two boards:

- UNO एक R4 WiFi - includes a USB-C® connector, Wi-Fi®/Bluetooth® chip, LED Matrix, Qwiic Connector and support for RTC, HID.
- UNO R4 Minima - includes USB-C® connector, support for RTC, HID.

Both boards works very similar, and are based on the same microcontroller, but the UNO एक R4 WiFi has more features.

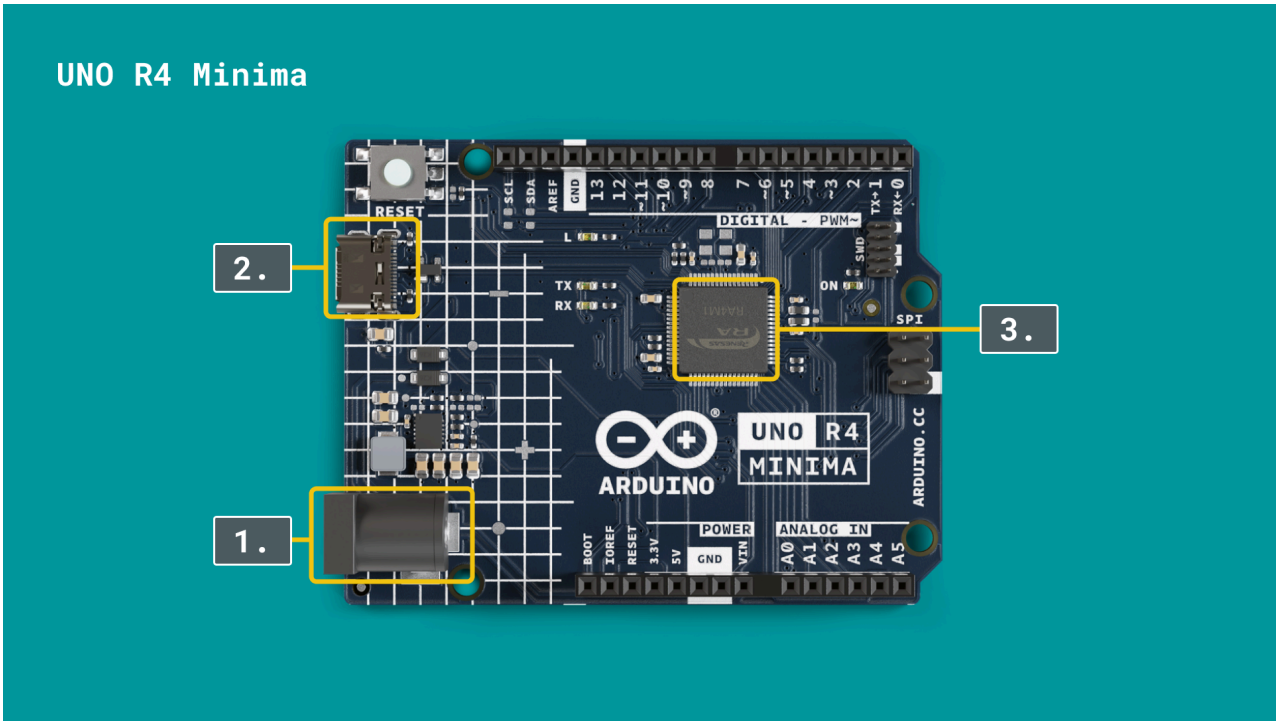
Board Overview

UNO एक R4 WiFi



1. **Power Connector** - this can be used to power the board with an external power adapter.
2. **Wi-Fi®/Bluetooth® chip (ESP32)** - a radio module can be used to connect to Internet and Bluetooth.
3. **USB-C connector** - used to power the board, and to transfer programs from the computer.
4. **Microcontroller (RA4M1)** - the brain of the Arduino.
5. **Qwiic Connector** - used to connect modules using I2C protocol (like Modulinos).
6. **LED Matrix** - a 96 LED display that can be used to display text, icons, animations or games.

UNO R4 Minima



1. **Power Connector** - this can be used to power the board with an external power adapter.
2. **USB-C connector** - used to power the board, and to transfer programs from the computer.
3. **Microcontroller (RA4M1)** - the brain of the Arduino.

Learn How To Use Features

Now that we have an overview of the UNO R4 boards, let's go through each of the features, learn what they do, and how we can program our board to use them.

> *Many of the features will not be available on the UNO R4 Minima.*

LED Matrix

The Arduino UNO एक R4 WiFi comes with a built in 12x8 LED Matrix, that is available to be programmed to display graphics, animations, act as an interface, or even play games on.

Requirements

- UNO एक R4 WiFi
- USB-C® cable

Icon Blinking

This example show how to create icons on the LED matrix display. We will display a **heart icon**, followed by a **smiley icon**.

Code Example

Upload the following code to your board:

```
#include "Arduino_LED_Matrix.h"

ArduinoLEDMatrix matrix;

void setup() {
  Serial.begin(115200);
  matrix.begin();
}

const uint32_t happy[] = {
```

```
    0x19819,
    0x80000001,
    0x81f8000
};
const uint32_t heart[] = {
    0x3184a444,
    0x44042081,
    0x100a0040
};

void loop(){
  matrix.loadFrame(happy);
  delay(1000);

  matrix.loadFrame(heart);
  delay(1000);
}
```

How it Works

After uploading, you will see a heart icon on the LED Matrix for 1 second. It then changes to a smiley icon.

So how does this work?

- Two arrays are created, one that holds a "heart", and one that holds a "smiley".
- The arrays contain binary data, which is pretty hard to understand, but the data is used to activate a specific pixel.

Animation

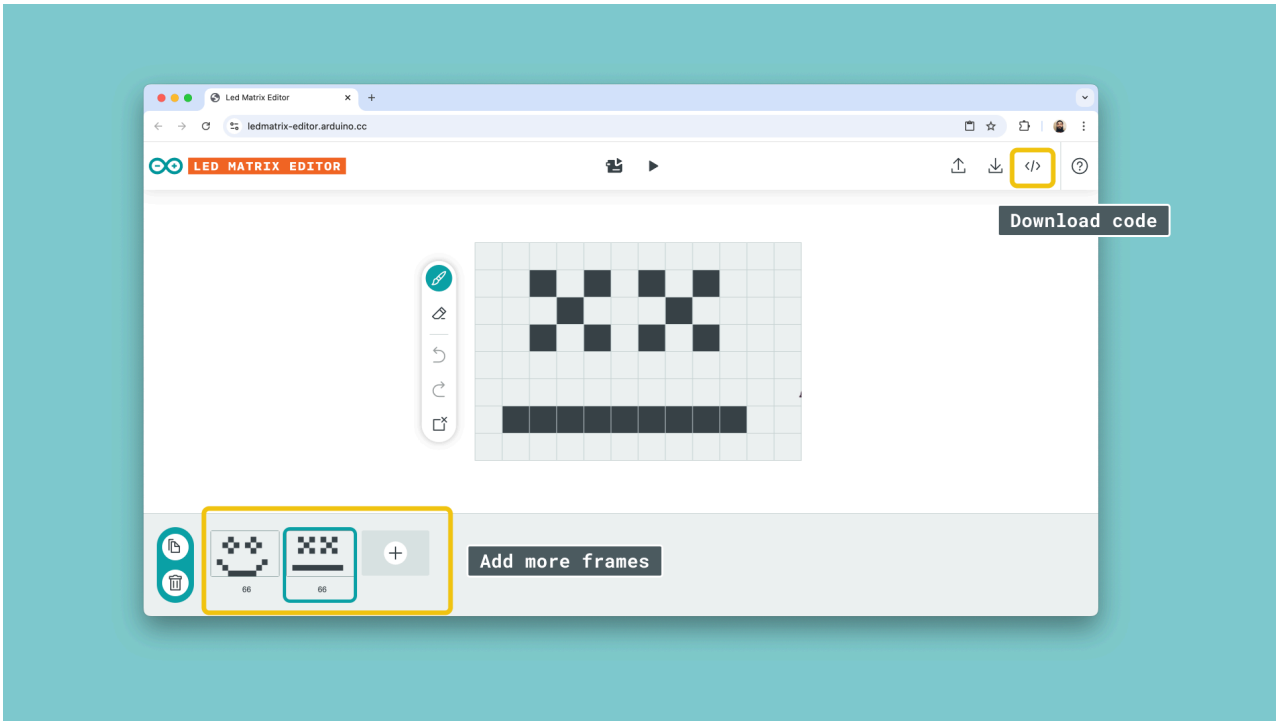
In this next example, we will create **animations** on the LED Matrix. To do this, we will need to use:

- LED Matrix Editor - an editor that allows you to create animations, frame by frame.
- We will also need to create something called a "header file" inside the editor. This will be explained in the next step.

Create Animation

Go to LED Matrix Editor. This is a pretty basic tool, where you can create exactly what you want to display on the LED Matrix.

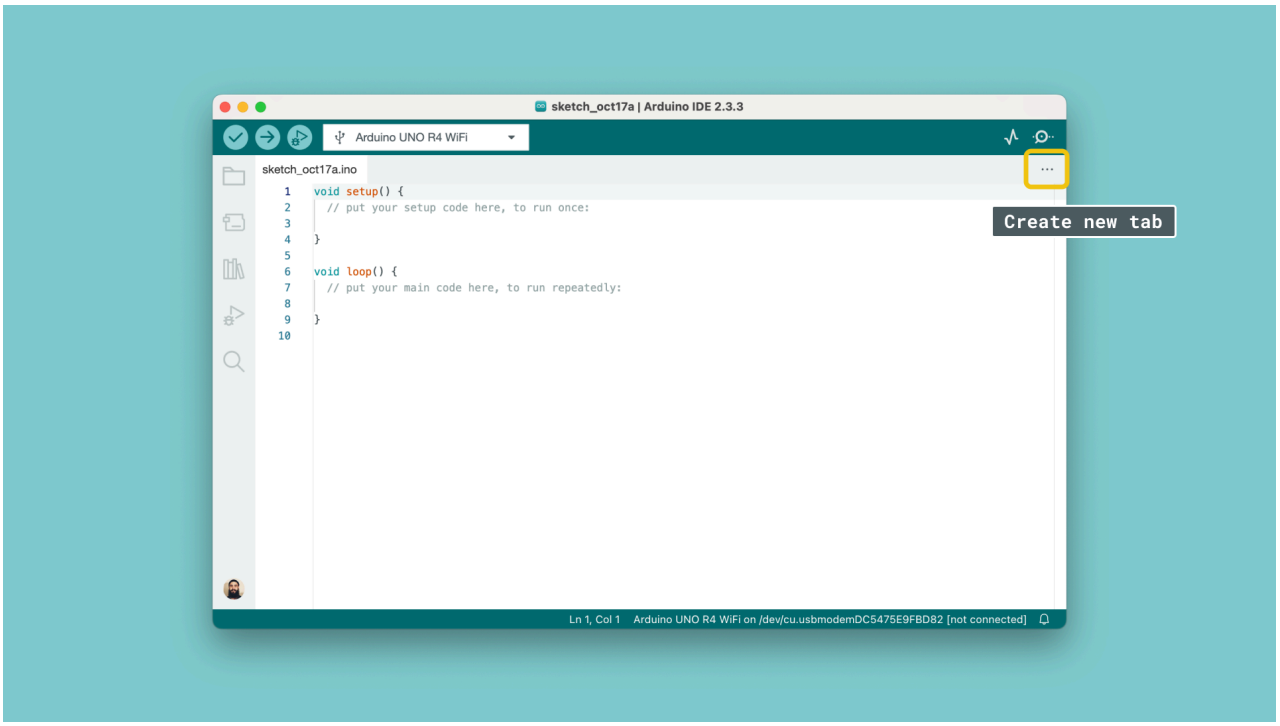
- First, you can draw something in the middle of the screen (each square represents a pixel).
- Second, in the bottom menu, click the **"+"** button, to add a new frame.
- Add as many frames as you want.
- When you are finished, click the "Code" symbol at the top right, to export the frames.



Create Header File

Now, to use the animation, we will need to create a new sketch file in the Arduino IDE.

- Open the Arduino IDE, and create a new sketch.
- In the top right corner, click on the three dots (...), and select "New Tab". Name it `animation.h`



- Now, go to your downloads and open the `animation.h` file. Copy the contents from this file, into the new file we created inside the Arduino IDE.
- Then, go back to your sketch file, by clicking on the tab.

> *Note: you can also locate your sketch folder and move the file you downloaded into the sketch folder. Either method works.*

Code Example

Copy this code example into your sketch file, and upload it to your board:

```
#include "Arduino_LED_Matrix.h"   //Include the LED_Matrix library
#include "animation.h" //include the new animation.h file
// Create an instance of the ArduinoLEDMatrix class
ArduinoLEDMatrix matrix;

void setup() {
  Serial.begin(115200);
  // you can also load frames at runtime, without stopping the refresh
  matrix.loadSequence(animation);
  matrix.begin();
  matrix.play(true);
}

void loop() {
}
```

How it Works

After uploading the code, you will notice that the animation starts playing, displaying all the frames you created.

- In the **LED Matrix Editor**, we create frames for the LED Matrix.
- These frames are stored in an array, that is then loaded to the LED Matrix, using the `matrix.loadSequence(animation)` function.
- After the frames are loaded, we initialize the matrix, and set the `matrix.play()` to `true`.
- This animation will now run forever and ever.

Modify the Animation

A simple modification you can make is to change the duration of how long a frame should be displayed. If you open the `animation.h` file, we can change this.

```
const uint32_t animation[][4] = {
    {
        0x3184a444,
        0x44042081,
        0x100a0040,
        66 //change this to for example 500
    },
    {
        0x81d,
        0x40084022,
        0x41f8000,
        66 //change this to for example 1000
    }
};
```

Upload the code again, and watch as the animation's duration will change depending on the value you input.

- The value is **milliseconds (ms)**, where 1000 ms = 1 second.

Real-time Clock (RTC)

A real-time clock (RTC) is a built-in component inside the microcontroller, it is so small and hidden that you cannot see it. But what it can do, is pretty cool.

An RTC keeps track of **time**, and can be used for keeping track of time, timers, automatic triggers and so on.

Requirements

- UNO एक R4 WiFi or
- UNO R4 Minima
- USB-C Cable

Counting

In this example we will initialize the RTC when starting the board, with a starting date. As long as the board is running, the time will be accurate. Modify this below with the your current date and time:

`RTCTime startTime(17, Month::OCTOBER, 2024, 13, 37, 00, DayOfWeek::THURSDAY, SaveLight::SAVING_TIME_ACTIVE);`

The time will be sent to the computer, and can be viewed in the **Serial Monitor** tool.

Code Example

Upload the following code to the board.

```
#include "RTC.h"

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

  RTC.begin();

  RTCTime startTime(17, Month::OCTOBER, 2024, 13, 37, 00, DayOfWeek::THURSDAY, SaveLight::SAVING_TIME_ACTIVE);

  RTC.setTime(startTime);
}

void loop() {
  RTCTime currentTime;

  // Get current time from RTC
  RTC.getTime(currentTime);

  // Print out date (DD/MM//YYYY)
  Serial.print(currentTime.getDayOfMonth());
  Serial.print("/");
  Serial.print(Month2int(currentTime.getMonth()));
  Serial.print("/");
  Serial.print(currentTime.getYear());
  Serial.print(" - ");

  // Print time (HH/MM/SS)
  Serial.print(currentTime.getHour());
  Serial.print(":");
  Serial.print(currentTime.getMinutes());
  Serial.print(":");
  Serial.println(currentTime.getSeconds());

  delay(1000);
}
```

Human Interface Device (HID)

A "Human Interface Device" (HID) is a term used for accessories used by *humans* to interact with a computer, such as a mouse or a keyboard. The Arduino can actually become a mouse or a keyboard, by using the `HID` library, where it can send "keypresses" to your computer, emulating a keyboard, or `x` and `y` coordinates to emulate a mouse.

Requirements

- UNO एक R4 WiFi or
- UNO R4 Minima
- USB-C Cable

Keyboard

In this example, we will send a keypress to our computer, every second.

Code Example

Upload the following example to your board, and open a regular text editor to observe the results.

> *Please note that after uploading this sketch, your Arduino will act as a keyboard. To stop this, double tap the "reset" button on your board, and upload a different sketch.*

```
#include <Keyboard.h>

void setup() {
  Keyboard.begin();
  delay(1000);
}

void loop() {
  Keyboard.press('w');
  delay(100);
  Keyboard.releaseAll();
  delay(1000);
}
```

How it Works

- When the code is uploaded, the board will immediately start emualting a keyboard.
- The `Keyboard.press('w')` will emulate pressing down the **w** character on your keyboard.
- The `Keyboard.releaseAll()` function emulates **releasing** the character (like lifting your finger).

Mouse

In this example, we will emulate a mouse. We will only do something simple: move both axis of mouse just slightly (10 points), back and forth.

Code Example

Upload the following example to your board, and observe the results on the screen.

> *Note that this example will take control of the mouse cursor on your computer. To stop this, double tap the "reset" button on your board, and upload a different sketch.*

```
#include <Mouse.h>

void setup() {
  Mouse.begin();
  delay(1000);
}

void loop() {
  Mouse.move(10,10);
  delay(1000);
  Mouse.move(-10,-10);
```

```
  delay(1000);
}
```

How it Works

- After uploading, the mouse on your screen will move slightly back and forward.
- This is done by using `Mouse.move(10,10)` (10 pixels in x/y direction), and `Mouse.move(-10,-10)` (10 pixels in the reverse direction).

Wi-Fi®

Wi-Fi® is a well known wireless protocol used by billions of people every day. The UNO एक R4 WiFi has the possibility to connect to these networks, and send data in different ways across the Internet!

However, compared to the examples we have done so far, the Wi-Fi® examples are a little bit more complex.

Requirements

- UNO एक R4 WiFi
- USB-C cable
- Access to a Wi-Fi® network

Web Server

The first example is something called a "Web Server". This uses the same functionality as when you visit any webpage on the Internet.

This example will demonstrate how to send data from the board, so that you can view it directly in your web browser (for example Google Chrome, Edge, Safari).

Code Example

Copy and paste this example into the Arduino IDE, but before uploading, you will need to edit two fields:

- `char ssid[] = "your_network";` - replace `"your_network"` with your Wi-Fi® network name.
- `char pass[] = "your_password";` - replace `"your_password"` with your Wi-Fi® network password.

```
#include "WiFiS3.h"

///////please enter your sensitive data in the Secret tab/arduino_secrets.h
char ssid[] = "your_network";        // your network SSID (name)
char pass[] = "your_password";    // your network password (use for WPA, or use as key for
WEP)
int keyIndex = 0;                 // your network key index number (needed only for WEP)

int status = WL_IDLE_STATUS;

WiFiServer server(80);

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    // don't continue
    while (true);
  }
```

```
  String fv = WiFi.firmwareVersion();
  if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
    Serial.println("Please upgrade the firmware");
  }

  // attempt to connect to WiFi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }
  server.begin();
  // you're connected now, so print out the status:
  printWifiStatus();
}


void loop() {
  // listen for incoming clients
  WiFiClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an HTTP request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the HTTP request has ended,
        // so you can send a reply
        if (c == '\n' && currentLineIsBlank) {
          // send a standard HTTP response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");  // the connection will be closed after
completion of the response
          client.println("Refresh: 5");  // refresh the page automatically every 5 sec
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
          // output the value of each analog input pin
          for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
            int sensorReading = analogRead(analogChannel);
            client.print("analog input ");
            client.print(analogChannel);
            client.print(" is ");
            client.print(sensorReading);
            client.println("<br />");
          }
          client.println("</html>");
          break;
        }
        if (c == '\n') {
          // you're starting a new line
```

```
          currentLineIsBlank = true;
        } else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(1);

    // close the connection:
    client.stop();
    Serial.println("client disconnected");
  }
}


void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your board's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Accessing the Web Server

After uploading the code, open the Serial Monitor. If it is successful in connecting to your Wi-Fi® network, it will look like this:

- Copy the **IP address** and enter it in your browser.
- You should now see data in the web page. This is data from the board, the readings from all analog pins on the board (they are not connected to anything so they don't have much meaning).
- This web page is hosted directly on your Arduino, but requires you to be connected to the **same network** on your computer to see (it is not available on the Internet, only on the local network).

How it Works

- When the program starts, it first connects to Wi-Fi®, using `status = WiFi.begin(ssid, pass)`
- The UNO एक R4 WiFi is used to host a very minimal web page, written in HTML. This is first initialized by `server.begin()` .
- Inside the `loop()` , it get's a little bit more complicated:
  - `WiFiClient client = server.available()` checks for incoming clients (a client in this case is someone accessing the web page from the browser).
  - `while (client.connected())` keeps running *while* a client is connected.
  - `client.println("<html>")` there is a lot of these lines, which prints out an entire HTML site to the client, because that is the format the browser requires.
  - `for (int analogChannel = 0; analogChannel < 6; analogChannel++)` this for loop reads all the analog channels.
  - `client.print(analogChannel)` prints the analog data to the client.

Network Time

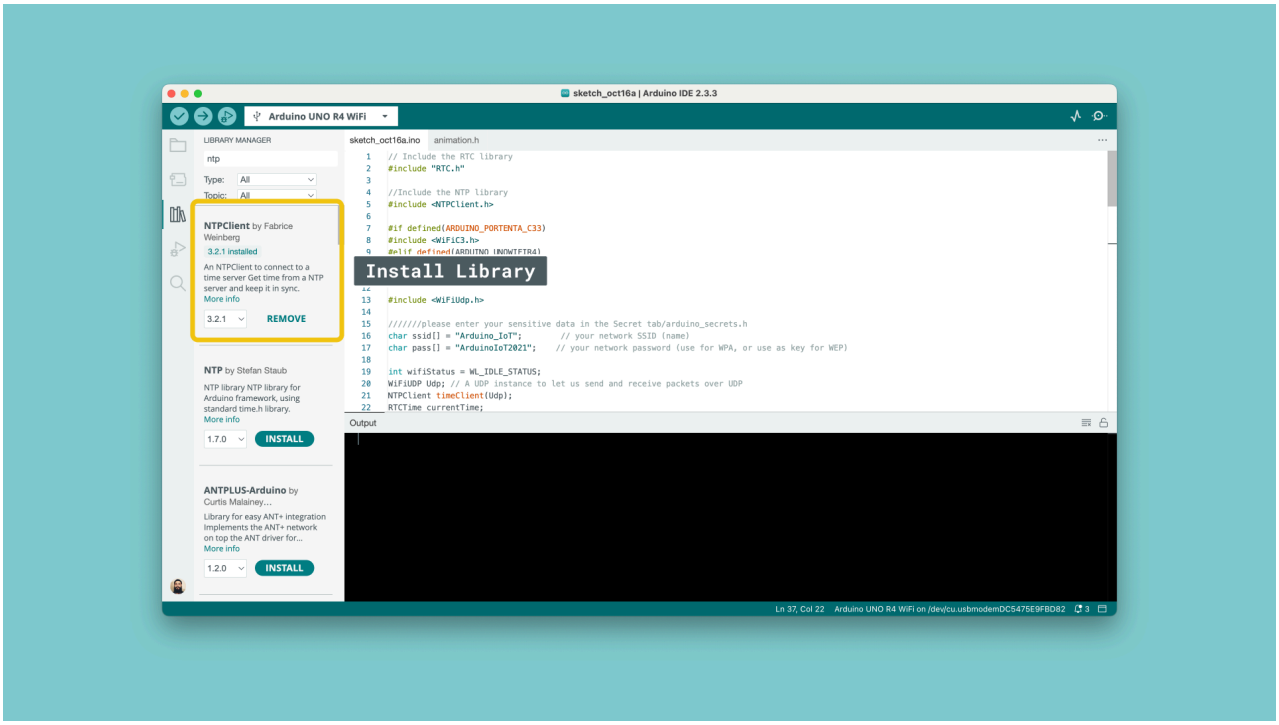
> *This only works on the UNO एक R4 WiFi, as it requires Wi-Fi® connectivity.*

In this next example, we will fetch the current time from Internet, and set the clock when starting the board.

Please note that this is a rather advanced example.

Install NTP Library

To use this example, you will need to install the **NTP Library**. In the editor, open the library manager in the left side menu, and search for **NTPClient**, then install it.



Code Example

Copy and paste this example into the Arduino IDE, but before uploading, you will need to edit two fields:

- `char ssid[] = "your_network";` - replace `"your_network"` with your Wi-Fi® network name.
- `char pass[] = "your_password";` - replace `"your_password"` with your Wi-Fi® network password.

> *Note that the both network and password are case sensitive.*

```
// Include the RTC library
#include "RTC.h"

//Include the NTP library
#include <NTPClient.h>

//include WiFiS3 library
#include <WiFiS3.h>

//include WiFiUdp library
#include <WiFiUdp.h>

///////please enter your sensitive data in the Secret tab/arduino_secrets.h
char ssid[] = "your_network";        // your network SSID (name)
char pass[] = "your_password";    // your network password (use for WPA, or use as key for
WEP)

int wifiStatus = WL_IDLE_STATUS;
WiFiUDP Udp; // A UDP instance to let us send and receive packets over UDP
NTPClient timeClient(Udp);
RTCTime currentTime;

void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your board's IP address:
```

```
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}

void connectToWiFi(){
  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    // don't continue
    while (true);
  }

  String fv = WiFi.firmwareVersion();
  if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
    Serial.println("Please upgrade the firmware");
  }

  // attempt to connect to WiFi network:
  while (wifiStatus != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
    wifiStatus = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }

  Serial.println("Connected to WiFi");
  printWifiStatus();
}

void setup(){
  Serial.begin(9600);
  while (!Serial);

  connectToWiFi();
  RTC.begin();
  Serial.println("\nStarting connection to server...");
  timeClient.begin();
  timeClient.update();

  // Get the current date and time from an NTP server and convert
  // it to UTC +2 by passing the time zone offset in hours.
  // You may change the time zone offset to your local one.
  auto timeZoneOffsetHours = 2;
  auto unixTime = timeClient.getEpochTime() + (timeZoneOffsetHours * 3600);
  Serial.print("Unix time = ");
  Serial.println(unixTime);
  RTCTime timeToSet = RTCTime(unixTime);
  RTC.setTime(timeToSet);

  // Retrieve the date and time from the RTC and print them
```

```
  RTC.getTime(currentTime);
  Serial.println("The RTC was just set to: " + String(currentTime));
}

void loop(){

  RTC.getTime(currentTime);
  Serial.println(String(currentTime));
  delay(1000);

}
```

How it Works

After uploading, open the Serial Monitor.

- First, the sketch will attempt to connect to the Wi-Fi® network specified.
- After connecting, it will make a request to an NTP server, and fetch the current time.
- The new time fetched will be added to the RTC, using the `RTC.setTime()` function.
- In the loop, every second, the time will be printed to the Serial Monitor.

> *Note that you may need to adjust the `auto timeZoneOffsetHours = 2` variable so that it matches your current time (for example, you can change it to `-5` or `3`).*

Bluetooth®

Bluetooth® is a network protocol that allows for data transfer in short ranges. Bluetooth® is available on the UNO एक R4 WiFi's via the radio module.

Bluetooth® applications are very common: it is used for mouses, keyboards, headphones, speakers and other useful applications.

The way that modern Bluetooth applications work is by having a **publisher** and **subscriber**. In simple terms, a device publishes something, and one device can subscribe to it. In other terms, they are named **central** and **peripheral** devices.

Requirements

- UNO एक R4 WiFi
- USB-C cable
- Smartphone with Bluetooth® (for example an iPhone or Samsung)

You will also need to download a mobile app that you can test this example with. We recommend the **nRF Connect for Mobile**, available in the iOS store and Play Store:

- iPhone (iOS store)
- Android (Play Store)

Install ArduinoBLE Library

To use this example, you will need to install the BLE Library. In the editor, open the library manager in the left side menu, and search for ArduinoBLE, then install it.

Code Example

Upload this example to your board. When finished, open the Serial Monitor.

```
#include <ArduinoBLE.h>
BLEService newService("180A"); // creating the service

BLEUnsignedCharCharacteristic randomReading("2A58", BLERead | BLENotify); // creating the
Analog Value characteristic
BLEByteCharacteristic switchChar("2A57", BLERead | BLEWrite); // creating the LED
characteristic
```

```
const int ledPin = 2;
long previousMillis = 0;


void setup() {
  Serial.begin(9600);    // initialize serial communication
  while (!Serial);       //starts the program if we open the serial monitor.

  pinMode(LED_BUILTIN, OUTPUT); // initialize the built-in LED pin to indicate when a
central is connected
  pinMode(ledPin, OUTPUT); // initialize the built-in LED pin to indicate when a central is
connected

  //initialize ArduinoBLE library
  if (!BLE.begin()) {
    Serial.println("starting Bluetooth® Low Energy failed!");
    while (1);
  }

  BLE.setLocalName("UNO R4 WiFi"); //Setting a name that will appear when scanning for
Bluetooth® devices
  BLE.setAdvertisedService(newService);

  newService.addCharacteristic(switchChar); //add characteristics to a service
  newService.addCharacteristic(randomReading);

  BLE.addService(newService);  // adding the service

  switchChar.writeValue(0); //set initial value for characteristics
  randomReading.writeValue(0);

  BLE.advertise(); //start advertising the service
  Serial.println(" Bluetooth® device active, waiting for connections...");
}

void loop() {

  BLEDevice central = BLE.central(); // wait for a Bluetooth® Low Energy central

  if (central) {  // if a central is connected to the peripheral
    Serial.print("Connected to central: ");

    Serial.println(central.address()); // print the central's BT address

    digitalWrite(LED_BUILTIN, HIGH); // turn on the LED to indicate the connection

    // check the battery level every 200ms
    // while the central is connected:
    while (central.connected()) {
      long currentMillis = millis();

      if (currentMillis - previousMillis >= 200) { // if 200ms have passed, we check the
battery level
        previousMillis = currentMillis;

        int randomValue = analogRead(A1);
        randomReading.writeValue(randomValue);

        if (switchChar.written()) {
```

```
          if (switchChar.value()) {   // any value other than 0
            Serial.println("LED on");
            digitalWrite(ledPin, HIGH);         // will turn the LED on
          } else {                              // a 0 value
            Serial.println(F("LED off"));
            digitalWrite(ledPin, LOW);          // will turn the LED off
          }
        }

      }
    }

    digitalWrite(LED_BUILTIN, LOW); // when the central disconnects, turn off the LED
    Serial.print("Disconnected from central: ");
    Serial.println(central.address());
  }
}
```

How it Works

The program works like this:

- We advertise a "Bluetooth® Service".
- This can be found in the **nRF Bluetooth App**, and we can connect to it.
- Inside it, we can read values from the board, and control an LED by sending a 0 or a 1 to the board.

Inside the code, this happens:

- We create a new "Bluetooth® Service", using `BLEService newService("180A")`
- Then, we wait for a client to connect, using `while (central.connected())`
- If a client connects, we advertise (publish) a value recorded on the board (in this case, it is from analog pin A1), using `randomReading.writeValue(randomValue)` .
- We also listen for incoming data (this can be sent from the Bluetooth app), using `if (switchChar.value())`
  - If a value is anything other than 0 (for example, 1), the LED on the board will turn ON.
  - If a value is 0, it will turn OFF.

Chapter 6 - Arduino Common Components

There are some commonly used electronic parts that can be used with your Arduino UNO ϙ R4 WiFi to add extra functionality to your projects. These include sensors and actuators such as temperature sensors, RGB LEDs, motion sensors, and more. These components are often part of an Arduino starter kit or available in most maker spaces.

You can combine multiple components to create cool and useful projects! In this chapter, we'll explore how to use those components, and at the end, we'll share project ideas to get you started.

Requirements

For this chapter, you will need:

- Arduino UNO ϙ R4 WiFi
- USB-C® Cable
- Components
  - Temperature & humidity sensor (e.g., DHT11 or DHT22)
  - RGB LEDs
  - Push buttons
  - Ultrasonic distance sensor (e.g., HC-SR04)
  - Piezo buzzer
  - Rotary encoder
  - Accelerometer (e.g., ADXL345)
  - 220 Ω resistor
- Breadboard and jumper wires

Libraries

Some of these components require specific libraries. Use the Arduino IDE's Library Manager to install them:

1. Open the Arduino IDE.
2. Navigate to **Tools > Manage Libraries**.
3. Search for the required libraries:
   - `Adafruit Unified Sensor` and `DHT sensor library` by Adafruit for the DHT11/DHT22.
   - `Adafruit ADXL345` for the accelerometer.

Install these libraries to get started.

Temperature Sensor

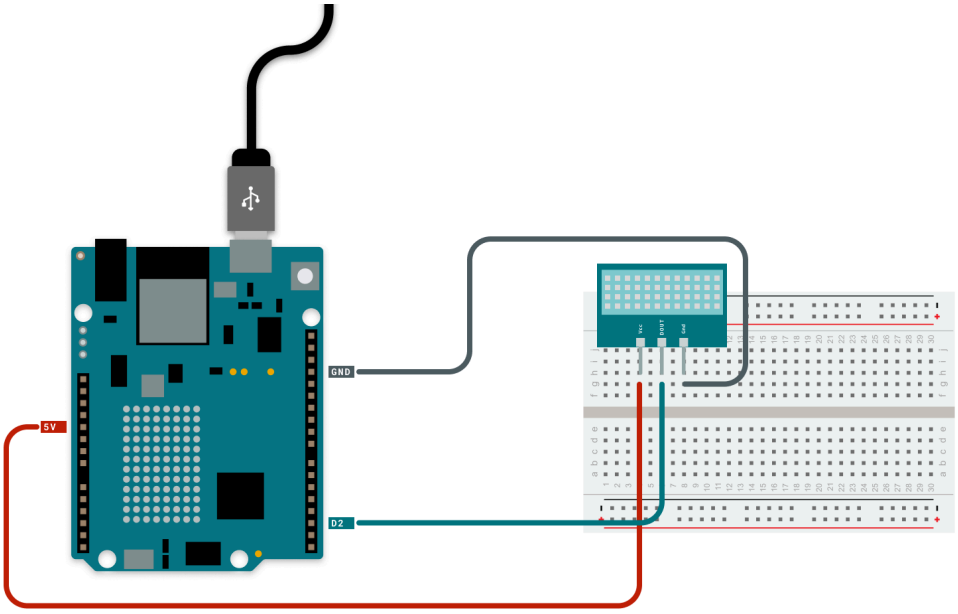
A temperature sensor like the DHT11 or DHT22 can measure temperature and humidity, making it useful for monitoring environmental conditions.

Requirements

- Arduino UNO ϙ R4 WiFi
- USB-C cable
- DHT11 or DHT22 sensor

Circuit

1. Connect **VCC** to 5V and **GND** to GND.
2. Connect the data pin of the sensor to pin 2 on the Arduino.



Note: if your DHT sensor does not come with a support circuit, you need to add an external 10K pull-up resistor on the output pin for proper communication between the sensor and the Arduino board. For more information click here.

Code Example

```
#include <DHT.h>

#define DHTPIN 2     // Pin connected to the data pin of the sensor
#define DHTTYPE DHT11 // Replace with DHT22 if using that model

DHT dht(DHTPIN, DHTTYPE);

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

void loop() {
  float temperature = dht.readTemperature(); // Read temperature in Celsius
  float humidity = dht.readHumidity();       // Read relative humidity

  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");

  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
```

```
  delay(2000); // Wait 2 seconds between readings
}
```

How it Works

After uploading the code and connecting your sensor, the Arduino regularly reads temperature and humidity, displaying the results in the Serial Monitor every two seconds.

- **Sensor Initialization**: Initiated with `dht.begin()`, setting up communication between the Arduino and the sensor.
- **Reading Data**: The `loop()` function continuously retrieves temperature and humidity using `readTemperature()` and `readHumidity()`.
- **Data Validation**: The readings are checked for validity, and an error message is displayed if the data is `NaN`.
- **Serial Output**: Valid temperature and humidity data are printed to the Serial Monitor, showing real-time environmental conditions.
- **Delay**: A preset `delay(2000)` spaces out readings every two seconds to avoid excessive data output.

Piezo Buzzer

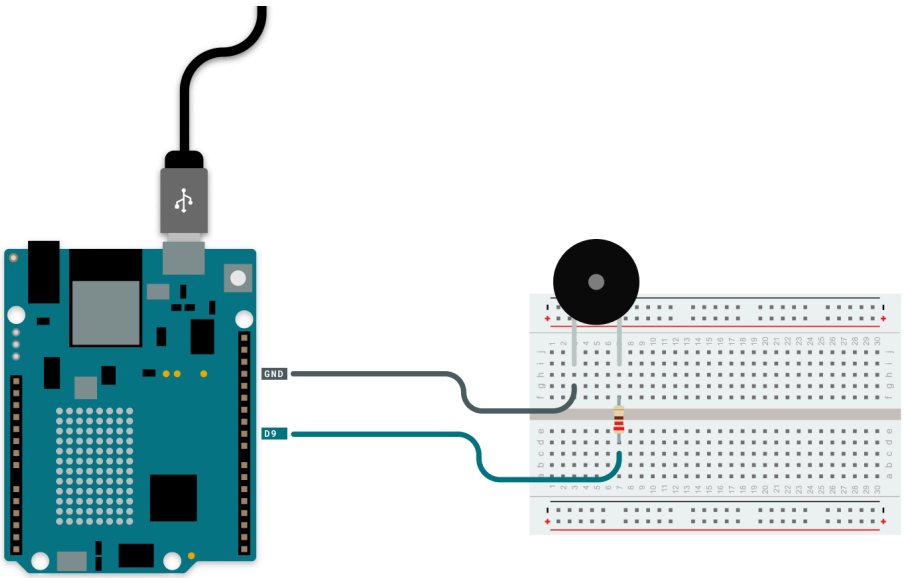
A piezo buzzer generates sound by applying an electric signal to a piezoelectric element. You can control the frequency to create different tones.

Requirements

- Arduino UNO ए R4 WiFi
- USB-C cable
- Piezo buzzer
- 220 Ω resistor

Circuit

1. Connect the positive terminal of the buzzer to **pin 9** on the Arduino board through a 220 Ω resistor.
2. Connect the negative terminal of the buzzer to **GND**.



Code Example

```
int buzzerPin = 9;
int frequency = 440; // Frequency in Hz, determines the pitch of the sound
```

```
int duration = 1000; // Duration in ms

void setup() {
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  tone(buzzerPin, frequency, duration);
  delay(1000);
  noTone(buzzerPin);
  delay(1000);
}
```

How it Works

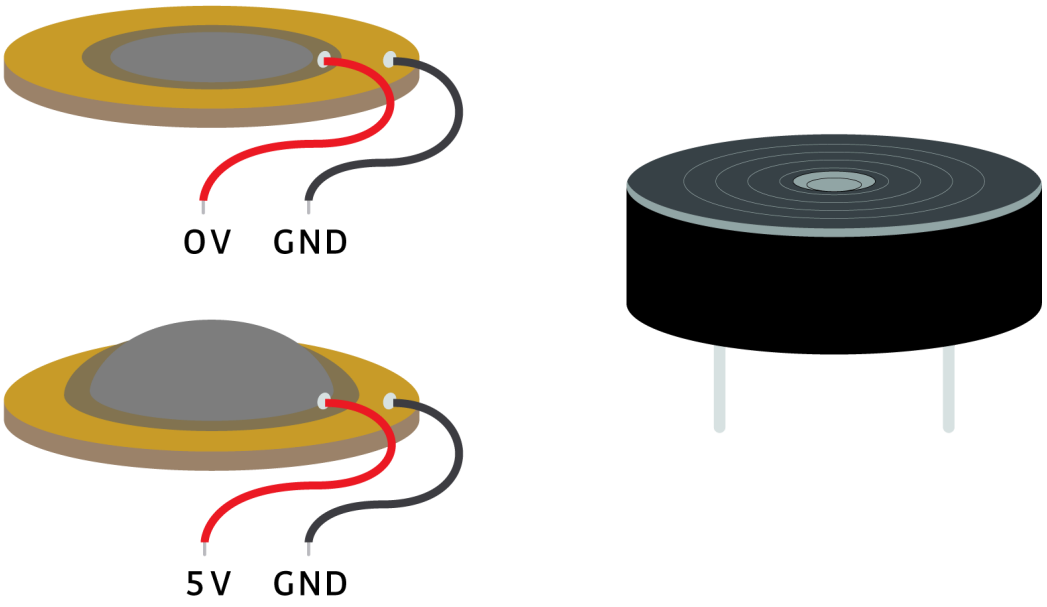
Upon code execution and circuit setup, the buzzer emits a tone every second, illustrating its capability to generate sounds at specific frequencies.

- **Initialization**: The buzzer pin is configured as an `OUTPUT` in `setup()`.
- **Tone Generation**: The `tone()` function activates the buzzer, producing a sound at 440 Hz by sending an electrical signal.
- **Duration Management**: The tone persists for one second before being turned off with `noTone()`.
- **Pausing**: A `delay(1000)` creates a one-second pause between tones, resulting in a consistent on-off cycle.
- **Cycle Repetition**: The loop repeats this procedure, playing the note intermittently.

Learn More: How a Piezo Buzzer work

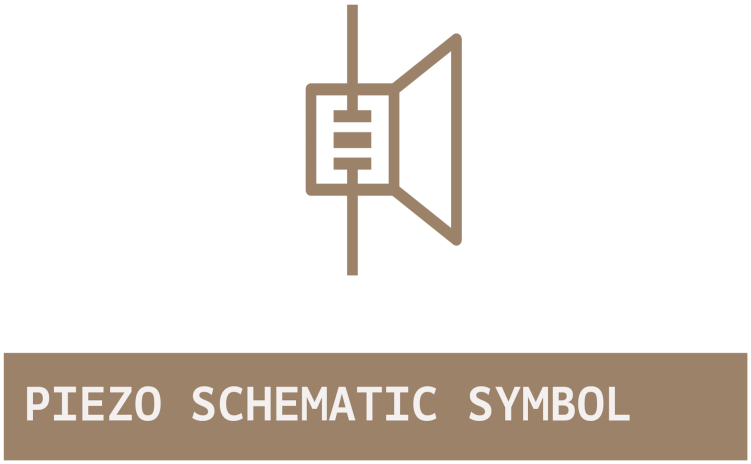
A **piezo buzzer** is kind of like a drum. When you hit a drum, the drumhead vibrates back and forth. These vibrations create sound waves that your ears can hear.

Inside the hard plastic case of the piezo buzzer is a membrane that vibrates like a drumhead. Most piezo membranes are made of a ceramic disk placed on top of a metal disk. Rather than being struck by a drumstick, a piezo membrane vibrates due to electric current. When current passes through the membrane, it deforms. When the current stops, the membrane goes back to its original shape. When current is turned on and off very quickly, the membrane vibrates back and forth, creating sound waves that you can hear.



By changing how fast the current is turned on and off, you can control the speed of the vibration. Faster vibrations create higher pitches while slower vibrations create lower pitches.

There is not a standard schematic symbol for piezo buzzers. Because piezo buzzers generate sound, a symbol that looks like a speaker is often used.



RGB LEDs

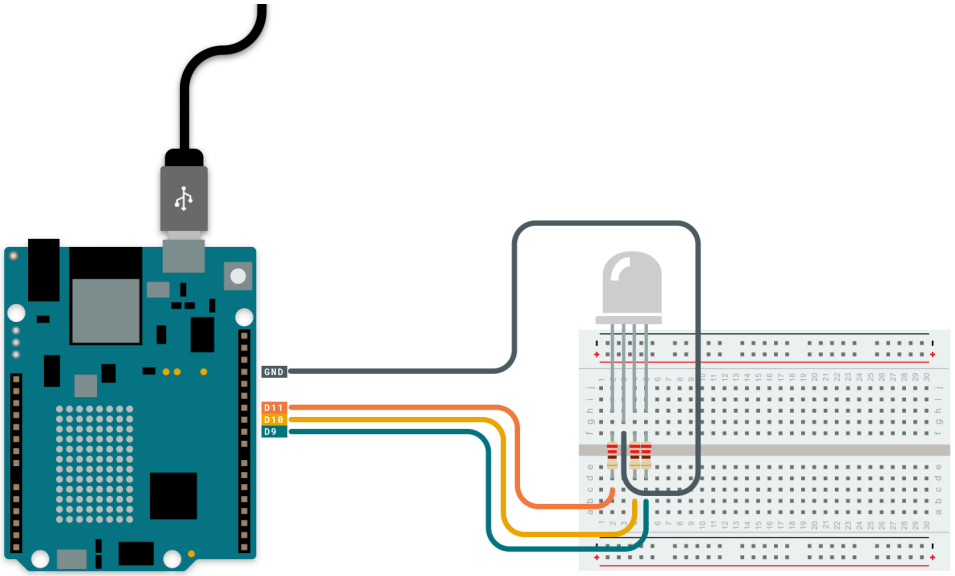
An RGB LED combines red, green, and blue LEDs in one package. You can control their brightness to create various colors.

Requirements

- Arduino UNO ए R4 WiFi
- USB-C cable
- Discrete RGB LED with resistors

Circuit

1. Connect the cathode of the RGB LED to **GND**.
2. Connect the red anode to **pin 11** through a 220 Ω resistor.
3. Connect the green anode to **pin 10** through a 220 Ω resistor.
4. Connect the blue anode to **pin 9** through a 220 Ω resistor.



Code Example

```
// Define RGB LED pins
const int redPin = 11;
const int greenPin = 10;
const int bluePin = 9;

// Function to set RGB color
void setColor(int red, int green, int blue) {
  analogWrite(redPin, red);
  analogWrite(greenPin, green);
  analogWrite(bluePin, blue);
}

void setup() {
  // Set pins as outputs
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  // Cycle through colors

  // Red
  setColor(255, 0, 0);
  delay(1000);

  // Green
  setColor(0, 255, 0);
  delay(1000);

  // Blue
  setColor(0, 0, 255);
  delay(1000);
```

```
  // Yellow
  setColor(255, 255, 0);
  delay(1000);

  // Cyan
  setColor(0, 255, 255);
  delay(1000);

  // Magenta
  setColor(255, 0, 255);
  delay(1000);

  // White
  setColor(255, 255, 255);
  delay(1000);

  // Off
  setColor(0, 0, 0);
  delay(1000);
}
```

RGB LED: How it Works

Following code upload and circuit configuration, the RGB LED cycles through diverse colors, demonstrating the combination of different light intensities.

- **Setup and Control**: Pin outputs for red, green, and blue allow color adjustment via PWM signals determined by `analogWrite()`.
- **Color Function**: `setColor()` modifies each color channel's brightness to create various hues.
- **Color Cycling**: Within `loop()`, the LED transitions through several predefined colors with `delay(1000)` pauses.
- **Blending Colors**: By altering PWM values, standard and mixed colors (e.g., yellow, cyan) are generated.
- **Cycle Repetition**: The LED turns off after all colors, before restarting the cycle to loop through them again.

Push Button

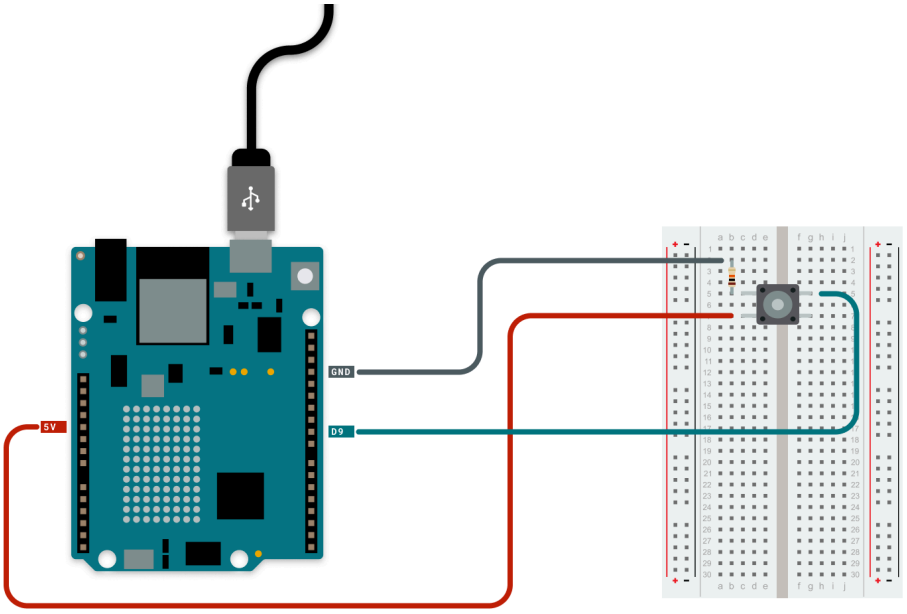
Push buttons can be used to input commands or interact with your project.

Requirements

- Arduino UNO ए R4 WiFi
- USB-C cable
- Push button
- 10 kΩ pull-down resistor

Circuit

1. Connect one side of the button to **pin 2**.
2. Connect the other side to **GND** through a 10 kΩ resistor.
3. Connect the same side to **5V** through a jumper wire.



Code Example

```
int buttonPin = 2;
int lastState = LOW; // Tracks the previous state of the button
int currentState = LOW; // Tracks the current state of the button

void setup() {
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  currentState = digitalRead(buttonPin);

  if (currentState == HIGH && lastState == LOW) {
    Serial.println("Button Pressed!");
  }
  if (currentState == HIGH) {
    Serial.println("Button Held!");
  }
  if (currentState == LOW && lastState == HIGH) {
    Serial.println("Button Released!");
  }

  lastState = currentState;
  delay(50); // Debounce delay
}
```

How it Works

Once the code is running and the push button is connected, pressing the button causes the Serial Monitor to reflect different button states.

- **Input Setup**: The button pin is set as an `INPUT`, ready to detect changes in its electrical state.

- **State Tracking**: `digitalRead(buttonPin)` checks whether the button is pressed or released, storing the result in `currentState`.
- **Press Detection**: Transitions from LOW to HIGH output "Button Pressed!" to the Serial Monitor.
- **Hold and Release**: High states print "Button Held!", while transitions back to LOW are acknowledged with "Button Released!".
- **Debouncing**: A `delay(50)` stabilizes readings, reducing noise from rapid button toggles.
- **State Updating**: `lastState` captures `currentState` for future detection in the next loop iteration.

Ultrasonic Distance Sensor

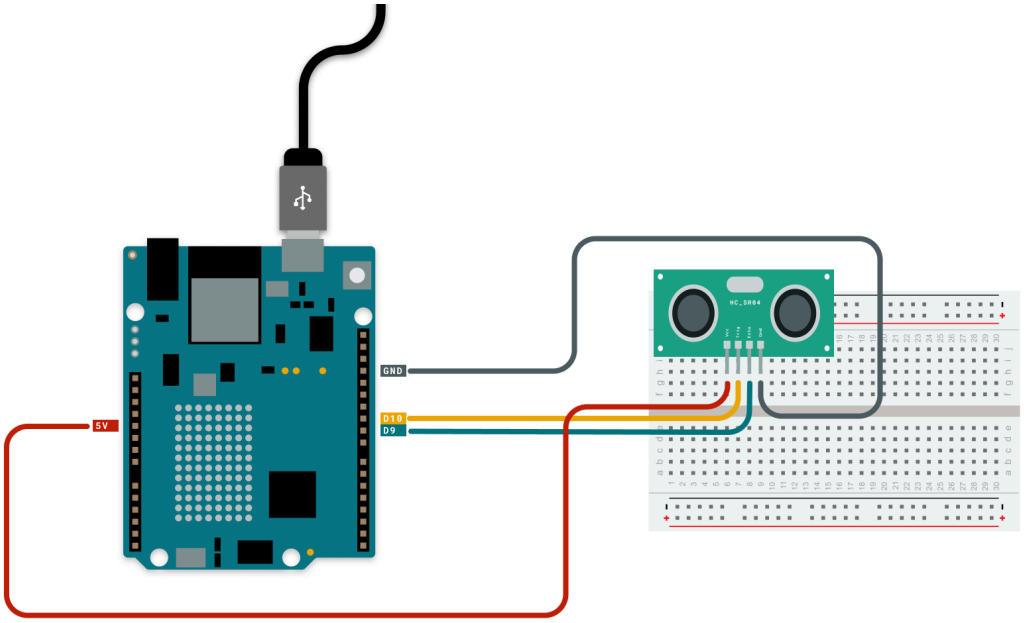
An ultrasonic sensor like the HC-SR04 measures distance by sending out sound waves and measuring their reflection time.

Requirements

- Arduino UNO ए R4 WiFi
- USB-C cable
- HC-SR04 sensor

Circuit

1. Connect **VCC** to 5V and **GND** to GND.
2. Connect **Trig** to pin 10 and **Echo** to **pin 9**.



Code Example

```
#define TRIG_PIN 10
#define ECHO_PIN 9

void setup() {
  Serial.begin(9600);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}

void loop() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH); // Measures the time in microseconds for the
echo pulse to return, which is proportional to the distance of the object
  int distance = duration * 0.034 / 2;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
  delay(500);
}
```

How it Works

After uploading the code and connecting the HC-SR04 sensor, it continuously measures distance by using sound waves and then reports the distance to the Serial Monitor.

- **Ping Setup**: The Trig pin is set to pulse HIGH briefly, emitting a sound wave.
- **Echo Measurement**: `pulseIn(ECHO_PIN, HIGH)` measures the time taken for the wave to bounce back.
- **Distance Conversion**: This time is calculated into distance using the formula `distance = duration * 0.034 / 2` , accounting for the speed of sound and wave return.
- **Output**: The measured distance is printed in real-time to the Serial Monitor, providing continuous updates.
- **Reading Delay**: `delay(500)` spaces out readings to ensure sensor accuracy and prevent noise interference.

Accelerometer

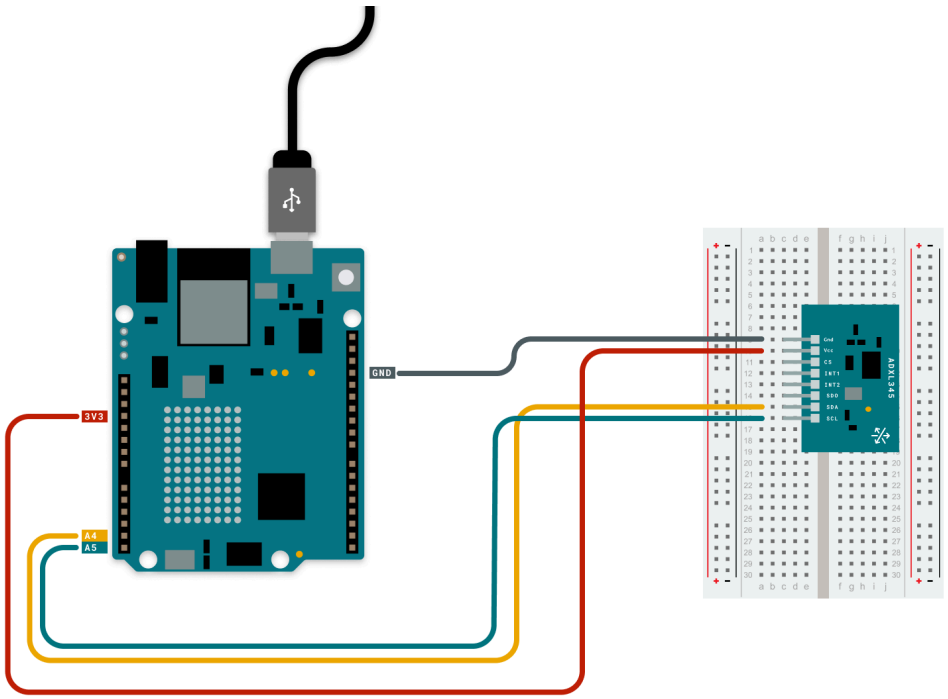
An accelerometer measures acceleration along the X, Y, and Z axes. The ADXL345 is a popular model.

Requirements

- Arduino UNO ए R4 WiFi
- USB-C cable
- ADXL345 accelerometer

Circuit

1. Connect **VCC** to 3.3V and **GND** from the sensor to **GND** on the board.
2. Connect **SDA** to A4 and **SCL** to **A5**.



Code Example

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>

Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);

void setup() {
  Serial.begin(9600);
  if (!accel.begin()) {
    Serial.println("ADXL345 not detected");
    while (1);
  }
}

void loop() {
  sensors_event_t event;
  accel.getEvent(&event); // Retrieves acceleration data on X, Y, and Z axes, providing a
snapshot of movement

  Serial.print("X: ");
  Serial.print(event.acceleration.x);
  Serial.print(" m/s^2");
  Serial.print(" Y: ");
  Serial.print(event.acceleration.y);
  Serial.print(" m/s^2");
  Serial.print(" Z: ");
  Serial.print(event.acceleration.z);
  Serial.println(" m/s^2");
  delay(500);
}
```

How it Works

Once the code is uploaded and the ADXL345 accelerometer is connected, the Arduino continuously measures and outputs the acceleration data along the X, Y, and Z axes to the Serial Monitor.

- **Sensor Initialization**: The `setup()` function initializes serial communication and tries to begin data interaction with the accelerometer using `accel.begin()`. If the sensor isn't detected, a failure message is printed.
- **Event Retrieval**: Within the `loop()`, the `accel.getEvent(&event)` function captures the acceleration data across all three axes, providing real-time movement readings.
- **Data Output**: The acceleration values for the X, Y, and Z axes are printed to the Serial Monitor, formatted with appropriate units (m/s²), giving users insight into the sensor's readings.
- **Reading Interval**: A `delay(500)` ensures the data updates in half-second intervals, making the output easy to read while reducing unnecessary processing load.

Project Ideas

Now that you've learned how to use these components, try combining them into projects. Here are some ideas:

Temperature Lamp

Change the color of an RGB LED based on the temperature.

- Read temperature from a DHT11/DHT22 sensor.
- Set RGB colors to blue (cold) or red (hot).

Mini Synth

Create a mini synthesizer with push buttons and a piezo buzzer.

- Assign a different frequency to each button press.
- Use `tone()` to play notes.

With these components, the possibilities are endless!

Chapter 7 - Arduino Cloud Templates

In the previous chapter, we explored how to use common components with the Arduino UNO ϙ R4 WiFi. Now, let's take it a step further by integrating these components with the **Arduino Cloud** to create powerful IoT applications!

Templates are pre-configured projects that allow users to quickly set up Arduino devices for the Arduino Cloud, creating a dashboard based on the project in two minutes.

Monitor and control your projects remotely via a customizable dashboard. For each example in this chapter, you'll find clear instructions to set up your components and connect them to the Arduino Cloud.

> ***Note**: Each example is standalone, so you can start with any component without depending on the others.*

LEARN MORE: What is IoT?

Internet of Things, or IoT, is referring to a giant network of connected devices collecting and sharing data from all over the world. Billions of devices are connected to the internet sharing information with each other and are used by us in our everyday lives.

Internet of Things is a way of understanding the digitalization of our world built on two fundamental concepts:

- everything can be marked digitally and therefore be distinguished from the rest by means of this identifier,
- any digital object that could be connected to a global network (in our case, the internet) will be connected.

This vision of a connected world brings a new computing paradigm where computation happens in a distributed manner, but where data is later shared and processed at different centralized pieces of infrastructure. This network of interconnected entities includes lamps, cellphones, smart watches, washing machines, cars, houses, buildings and people. All of these interconnected entities "talk" to each other using various technologies and protocols and exchange information over the internet. The nature of applications varies from a simple phone-controlled light bulb, to a complex garbage-management system.

Cloud-Integrated Components

Each example includes:

- A circuit diagram
- A link to a Cloud Template for quick setup
- An explanation of how it works

> ***Note**: This chapter cannot be followed with an Arduino UNO ϙ R4 Minima.*

Temperature Sensor

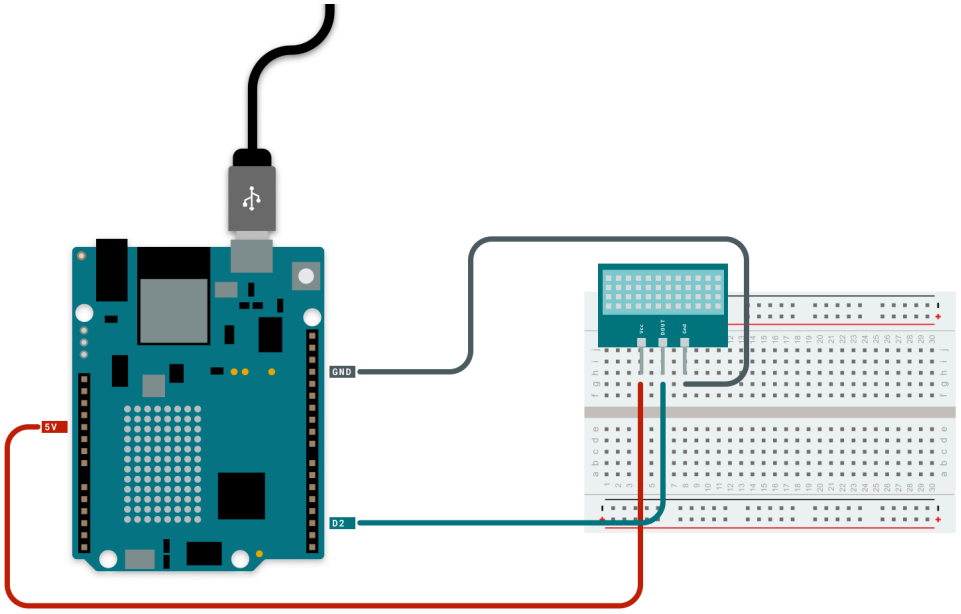
Use a temperature sensor (e.g., DHT11 or DHT22) to monitor environmental conditions remotely.

Requirements

- Arduino UNO ϙ R4 WiFi
- USB-C® cable
- DHT11 or DHT22 sensor

Circuit

1. Connect **VCC** to 5V and **GND** to GND.
2. Connect the data pin of the sensor to pin 2 on the Arduino.



Cloud Integration

Follow the link below to set up the Cloud Template:

- Temperature Sensor Cloud Template

Once set up, you'll have a dashboard to monitor temperature and humidity in real-time.

Piezo Buzzer

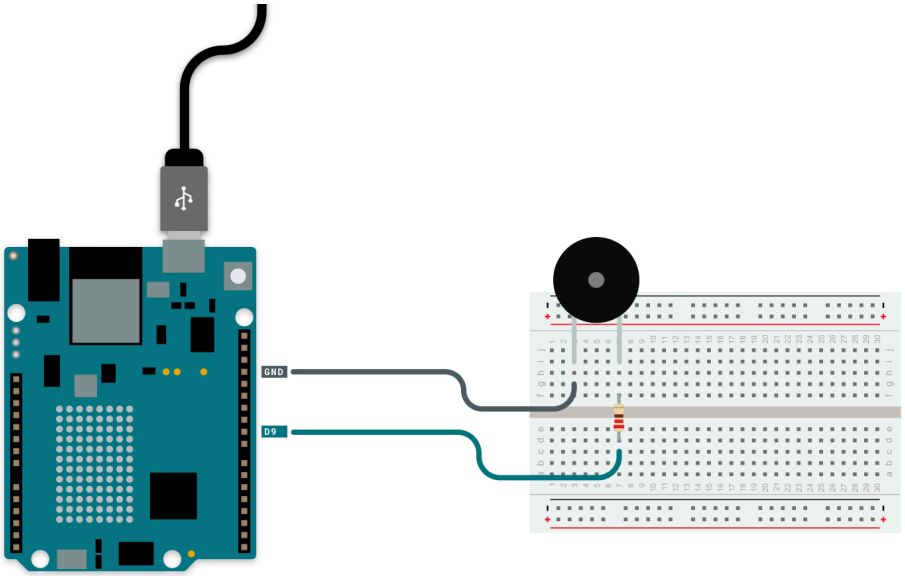
A piezo buzzer can generate sound remotely, making it perfect for alerts or notifications.

Requirements

- Arduino UNO ϟ R4 WiFi
- USB-C® cable
- Piezo buzzer
- 220 Ω resistor

Circuit

1. Connect the positive terminal of the buzzer to pin 9 on the Arduino through a 220 Ω resistor.
2. Connect the negative terminal of the buzzer to GND.



Cloud Integration

Follow the link below to set up the Cloud Template:

- Piezo Buzzer Cloud Template

Once set up, you'll have a dashboard to control sound output remotely.

RGB LEDs

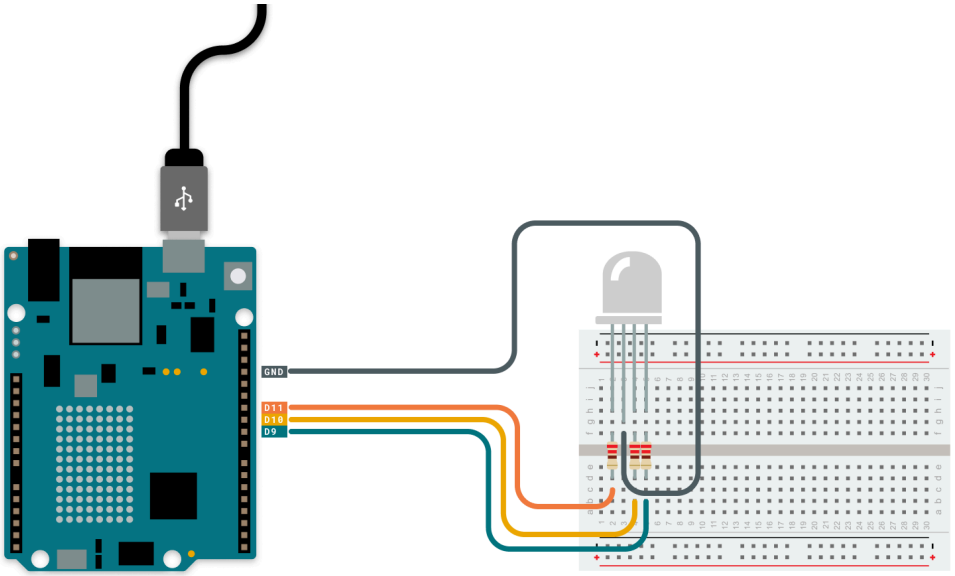
Control the color of an RGB LED remotely to create dynamic lighting effects.

Requirements

- Arduino UNO ϟ R4 WiFi
- USB-C® cable
- Discrete RGB LED with resistors

Circuit

1. Connect the cathode of the RGB LED to GND.
2. Connect the red anode to pin 11 through a 220 Ω resistor.
3. Connect the green anode to pin 10 through a 20 Ω resistor.
4. Connect the blue anode to pin 9 through a 20 Ω resistor.



Cloud Integration

Follow the link below to set up the Cloud Template:

- RGB LED Template (Cloud)

Once set up, you'll have a dashboard to control the LED's colors remotely.

Ultrasonic Distance Sensor

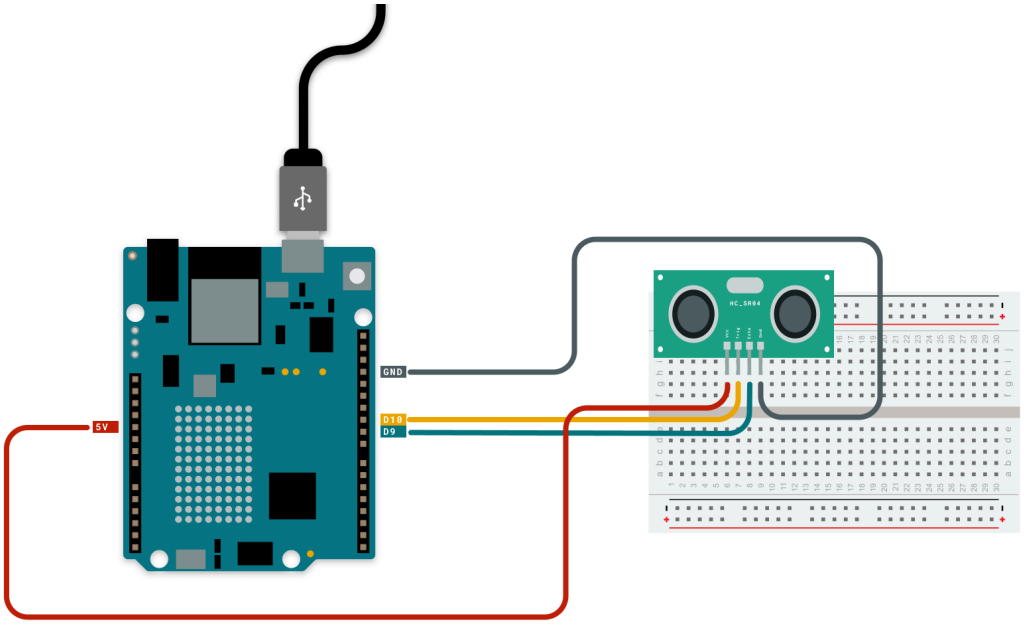
Monitor distances using an ultrasonic sensor like the HC-SR04, integrated with the Arduino Cloud.

Requirements

- Arduino UNO ϙ R4 WiFi
- USB-C® cable
- HC-SR04 sensor

Circuit

1. Connect **VCC** to 5V and **GND** to GND.
2. Connect **Trig** to pin 10 and **Echo** to pin 9.



Cloud Integration

Follow the link below to set up the Cloud Template:

- Ultrasonic Distance Sensor Template (Cloud)

Once set up, you'll have a dashboard to monitor distance measurements in real-time.

Accelerometer

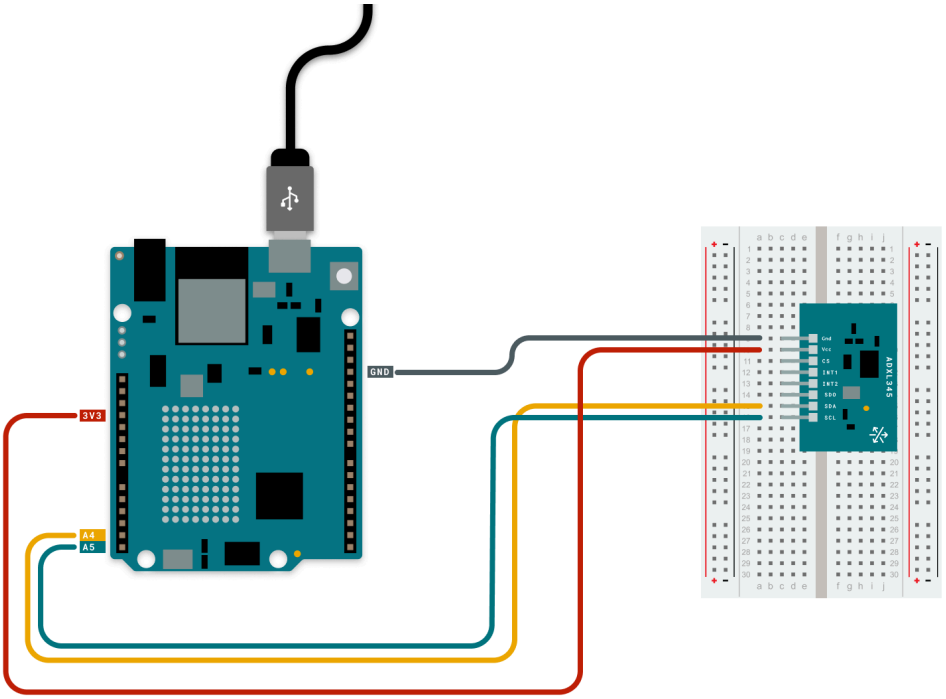
Use an accelerometer like the ADXL345 to monitor movement and acceleration remotely.

Requirements

- Arduino UNO ϙ R4 WiFi
- USB-C® cable
- ADXL345 accelerometer

Circuit

1. Connect **VCC** to 3.3V and **GND** to GND.
2. Connect **SDA** to A4 and **SCL** to A5.



Cloud Integration

Follow the link below to set up the Cloud Template:

- Accelerometer Template (Cloud)

Once set up, you'll have a dashboard to monitor movement and acceleration in real-time.

Troubleshooting

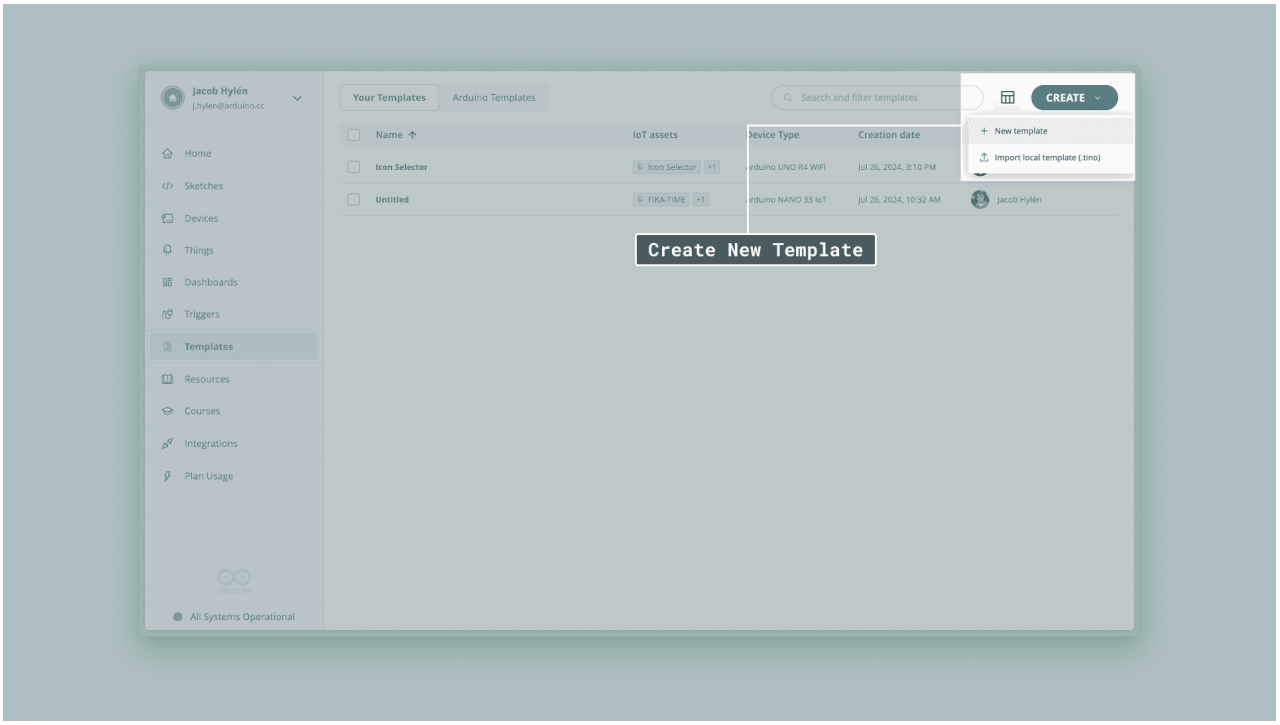
If your project isn't working as expected, try these solutions:

- **Wi-Fi Network/Password**: Ensure the Wi-Fi credentials are entered correctly. Both are case-sensitive.
- **Template Installation**: If your dashboard doesn't appear, re-run the template installation.
- **Board Connection Issues**:
  - Unplug and reconnect the board, then retry.
  - Double-tap the reset button quickly to enter "default mode" (indicated by a fading orange light).
- **Arduino Cloud Agent**: Ensure the Agent plugin is installed. Refer to this guide.

With the Arduino Cloud, these common components become powerful IoT tools. Start experimenting and unleash your creativity!

Further Reading - Custom Templates

Within a few minutes, this feature allows you to transform any of your existing cloud projects into a reusable template within minutes.



Once created, the template can either be saved for personal use or exported as a `.tino` file to share with others in any way you prefer.

To get started, you'll need an existing project that you can use as your starting point. Read more about it here.

Chapter 8 - Introduction to Machine Learning with Arduino

Artificial Intelligence (AI), and more specifically Machine Learning (ML), is revolutionizing the way devices understand and interact with the world. ML enables machines to learn from data and make decisions, making it a powerful tool for enhancing Arduino projects. This chapter will introduce you to ML concepts and demonstrate how you can use it in your Arduino projects using Edge Impulse.

What is Machine Learning?

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that allows machines to learn from data rather than being explicitly programmed. While AI covers broader tasks like speech recognition and computer vision, ML is the specific technique enabling devices, such as Arduino boards, to recognize patterns in sensor data. Key concepts in ML include:

- **Training:** Teaching a model using a dataset.
- **Inference:** Using a trained model to make predictions.
- **Features:** Attributes or patterns extracted from raw data.
- **Models:** Mathematical representations of learned patterns.

In the context of Arduino, ML can be used to interpret sensor data and add intelligence to projects. Common use cases include:

- **Gesture Recognition:** Detect specific movements using accelerometers.
- **Sound Classification:** Identify sounds like claps or voice commands.
- **Anomaly Detection:** Detect unusual patterns in data.

Arduino and ML Integration

Arduino boards like the UNO एक R4 WiFi can process sensor data and use pre-trained ML models for inference. Training and model deployment are made easy with platforms like Edge Impulse, which provide tools for data collection, model training, and deployment.

If you want to train your own model, you can collect data from your sensors, label it, and use Edge Impulse to generate an optimized model. However, if you want to get started quickly, you can also use the pre-trained model and library we have already prepared. Simply include it as a library in your Arduino project, and you can immediately start making predictions using real-time sensor data.

How Arduino and AI Work Together

AI can be integrated with Arduino projects by combining the board’s hardware capabilities with the computational power of cloud-based AI services. However, it’s important to distinguish between:

- **On-Device ML (TinyML):** This involves running ML models directly on the Arduino board. This approach is useful for applications like gesture recognition and anomaly detection because it doesn’t require internet connectivity.
- **Cloud-Based AI:** Here, Arduino acts as a sensor, collecting data and sending it to external AI services (e.g., OpenAI, Google AI) for advanced processing, such as natural language understanding or complex image recognition.

Some example applications include:

- **Voice-controlled projects:** Using AI for speech recognition and converting commands into actions on the Arduino.
- **Predictive systems:** Using AI to analyze sensor data and make predictions (e.g., weather forecasting).
- **Interactive displays:** Using AI to generate dynamic, meaningful content for displays, like an LED matrix.

Gesture Recognition with Edge Impulse Example

In this example, we’ll create a project to recognize gestures using the ADXL345 accelerometer and an ML model trained on Edge Impulse.

Requirements

- Arduino UNO एक R4 WiFi
- ADXL345 Accelerometer

- USB-C® Cable
- Edge Impulse Library

Steps

- Connect the ADXL345 Accelerometer as shown in the previous chapters.

1. **Option 1: Train Your Own Model**
    - Collect accelerometer data using the Edge Impulse Data Acquisition tool.
    - Train a model on Edge Impulse to recognize gestures like circles, up-down, and left-right movements.
    - Export the trained model as an Arduino library.
    - Deploy the model on the Arduino UNO R4 WiFi.
2. **Option 2: Use the Pre-Trained Model**
    - Instead of training from scratch, use our pre-trained model.
    - Download the library.
    - Install the library by clicking on "Sketch > Add .ZIP Library... >" and select the library(.zip).

You can download our pretrained model as a library here.

Code Example

The following example demonstrates gesture recognition using a pre-trained model. While this model is functional, it is not the most advanced. For the best results, ensure that the sensor is held facing upwards. If you want even better accuracy, you can train your own model using Edge Impulse.

The following gestures are included in the pretrained model:

- Up / Down
- Left / Right
- Circle
- Idle

Below is the complete code for running gesture recognition on the Arduino using the pre-trained Edge Impulse model:

> *Note: Compiling time is much longer than in the previous exercises.*

```
#include <Wire.h>
#include <Adafruit_ADXL345_U.h>       // Library for ADXL345
#include <movement_model.h> // Include your Edge Impulse Arduino library

#define CONVERT_G_TO_MS2    9.80665f
#define MAX_ACCEPTED_RANGE  2.0f

Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);
bool debug_nn = false; // Set this to true for detailed debug info

void setup() {
  Serial.begin(115200);
  while (!Serial); // Wait for USB connection
  Serial.println("Edge Impulse Inference – Movement Detection");

  // Initialize the ADXL345 sensor
  if (!accel.begin()) {
    Serial.println("Failed to initialize ADXL345! Check wiring.");
    while (1);
  }
  Serial.println("ADXL345 initialized.");

  // Ensure the model expects 3-axis accelerometer data
  if (EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME != 3) {
    Serial.println("ERR: Model requires 3 sensor axes!");
```

```
    while (1);
  }
}

float get_sign(float number) {
  return (number >= 0.0) ? 1.0 : -1.0;
}

void loop() {
  Serial.println("\nStarting inferencing in 2 seconds...");
  delay(2000);

  Serial.println("Sampling...");

  float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };

  // Collect accelerometer data
  for (size_t ix = 0; ix < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; ix += 3) {
    sensors_event_t event;
    accel.getEvent(&event);

    buffer[ix] = event.acceleration.x;
    buffer[ix + 1] = event.acceleration.y;
    buffer[ix + 2] = event.acceleration.z;
  }

  // Convert raw accelerometer data to a signal object
  signal_t signal;
  int err = numpy::signal_from_buffer(buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
  if (err != 0) {
    Serial.println("Failed to create signal from buffer");
    return;
  }

  // Run the classifier to detect movement patterns
  ei_impulse_result_t result = { 0 };
  err = run_classifier(&signal, &result, debug_nn);
  if (err != EI_IMPULSE_OK) {
    Serial.println("ERR: Failed to run classifier");
    return;
  }
}
```

How It Works

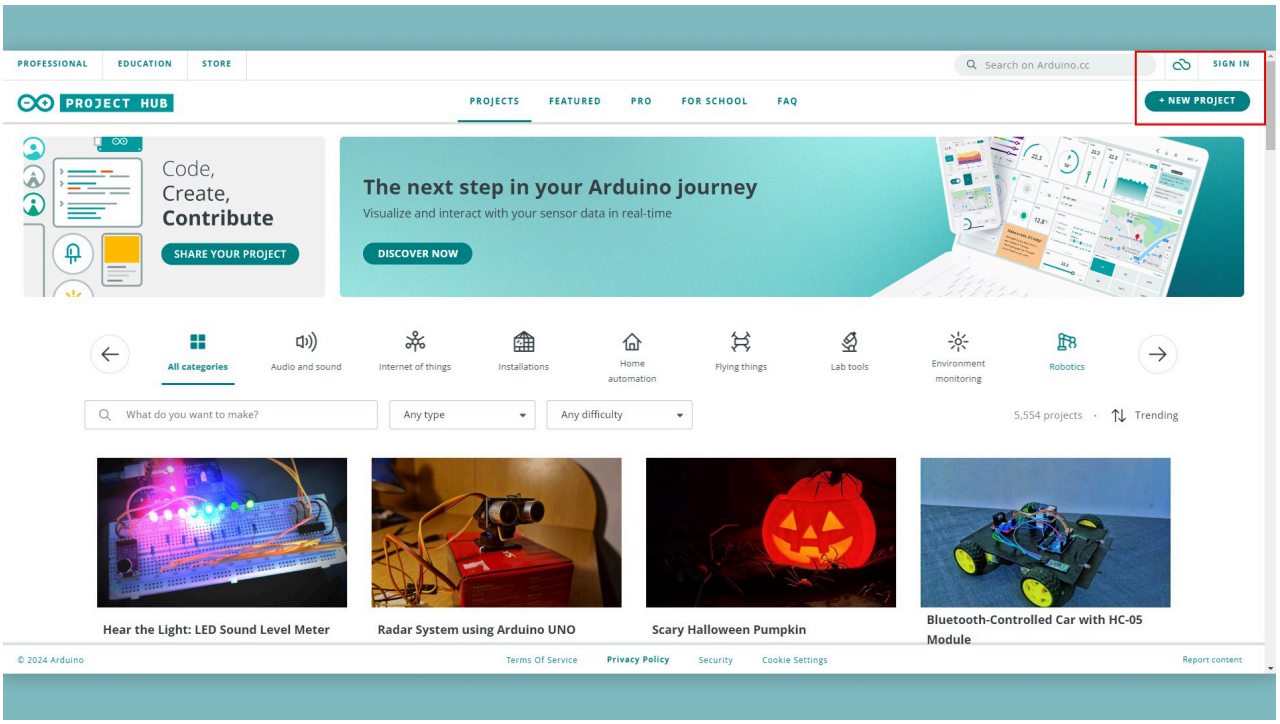
- This system collects real-time accelerometer data, processes it into meaningful signals, and runs inference using the trained ML model. The model then predicts the movement pattern based on predefined classes.
- The predictions are displayed in the Serial Monitor, allowing you to integrate gesture recognition into various Arduino projects such as interactive control systems or smart devices. The code includes beginner-friendly comments to guide you through each step, making it easier to understand how data is collected, processed, and classified.

Chapter 9 - Creating a Project Hub Project

In this chapter, we will be creating a project on **Project Hub**. Project Hub allows you to share your inventions with other people around the world, and is a great way of showcasing your projects!

Project Hub

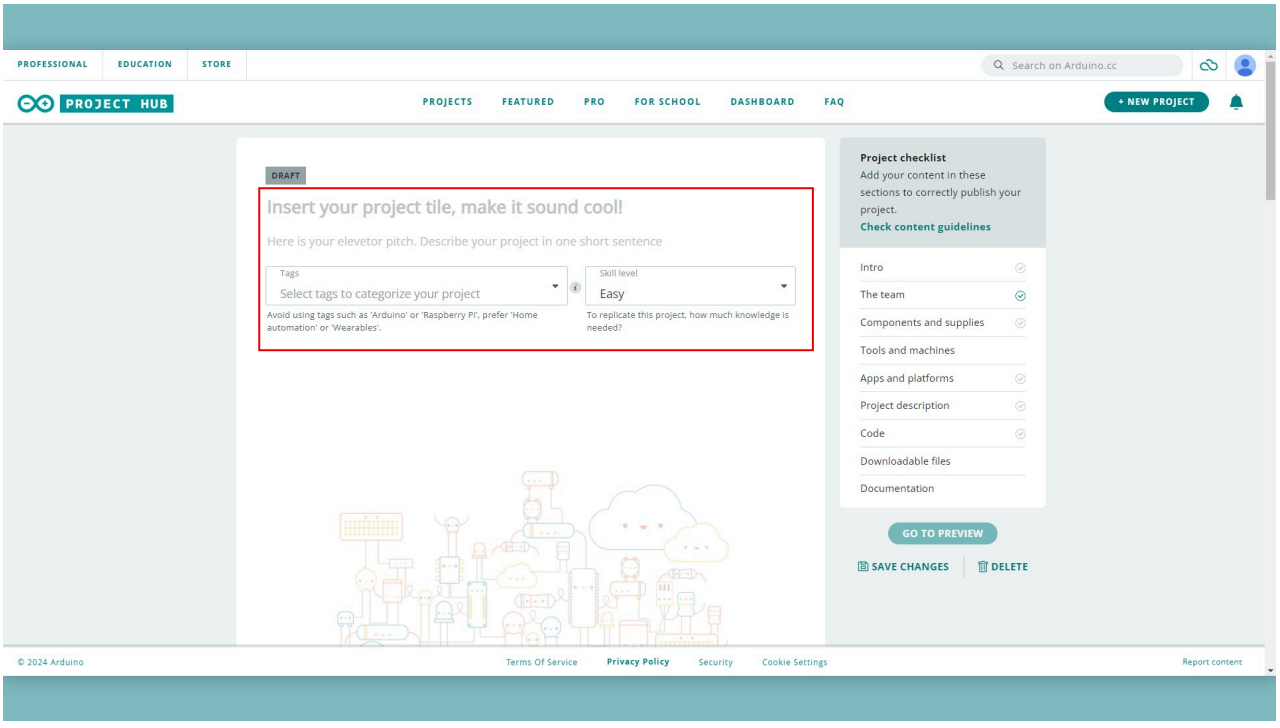
Go to [Arduino Project Hub] and log in. Press the **New Project** button, as shown in the image below:



This will take you to the project page.

> *Note that you need an Arduino account to create projects in Project Hub.*

The Project Page

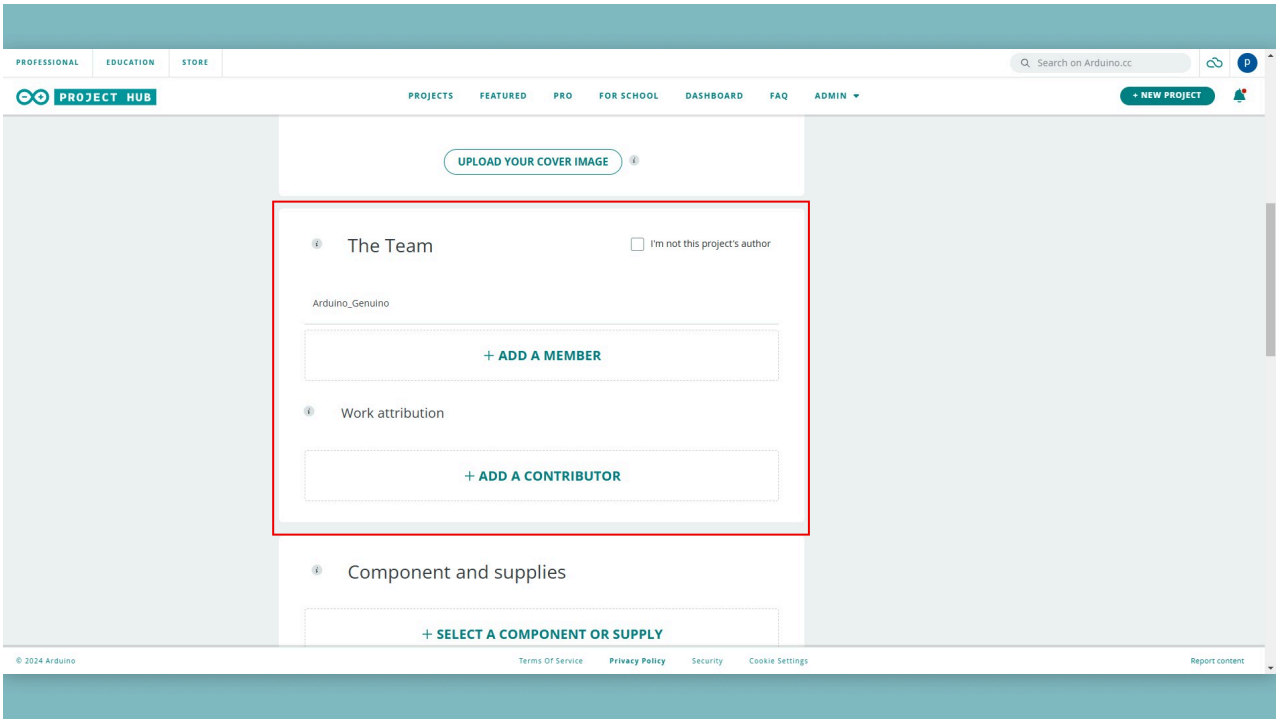


Here you can start creating your project. First enter the projects title, and remember to keep it short, descriptive to make your project sound interesting!

Below the title we have the elevator pitch, where you can describe your project in one or two sentences. Then you can browse from the **Tags** list and pick the ones that apply to your project.

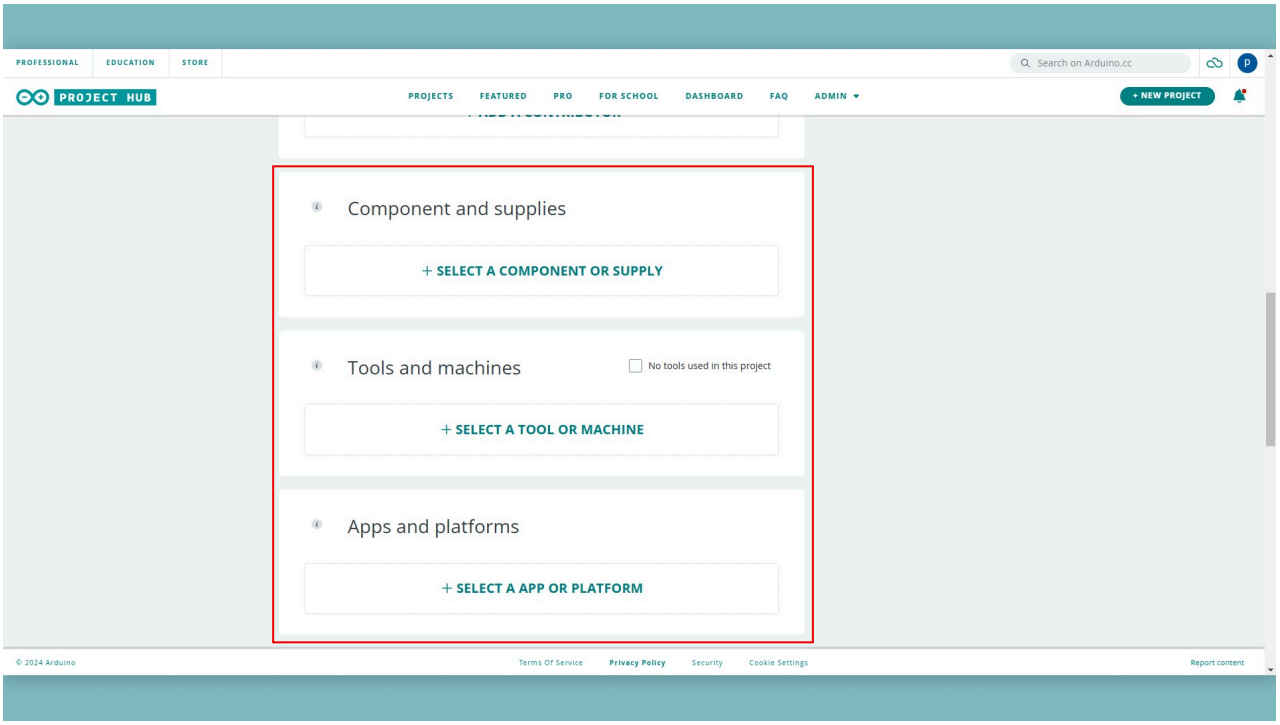
We can then pick a skill level you think is appropriate. Then finish this section with uploading a cover image of your project, take a nice picture of your assembled project and put it here.

Team Section



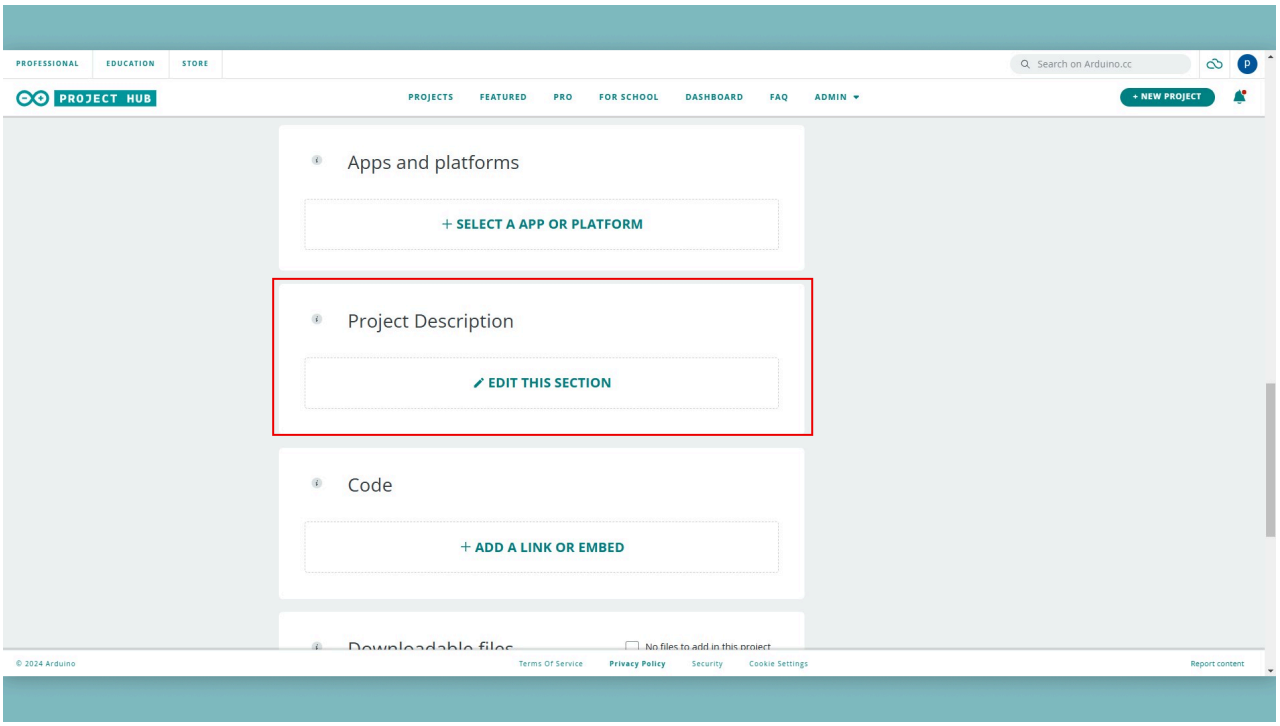
In **The Team** section you can add other users if you were working on a team project!

Components, Tools and Apps



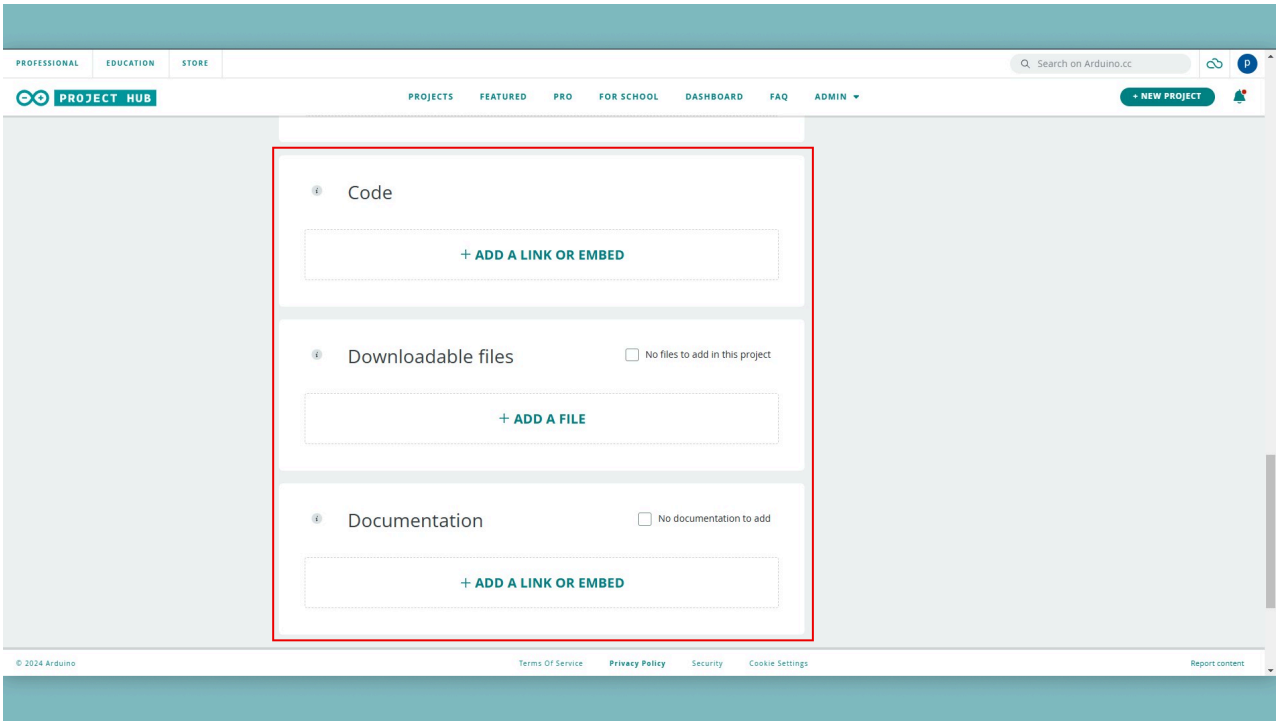
In the **Components and supplies**, **Tools and Machines** and **Apps and Platforms** sections you can add the board, components, tools, machines and software that you used in your project. If the component or board does not already exist in the list then you can create your own entry.

Project Description



The **Project Description** is the largest part of your project. Here you can describe the process of building the project, the background of why you made this project and add other images or videos you might have of your project. It is important that here you describe how to create the project in detail so other users can follow along and create their own version. Make it easier for other users to read with adding titles and text formatting.

Code & Files



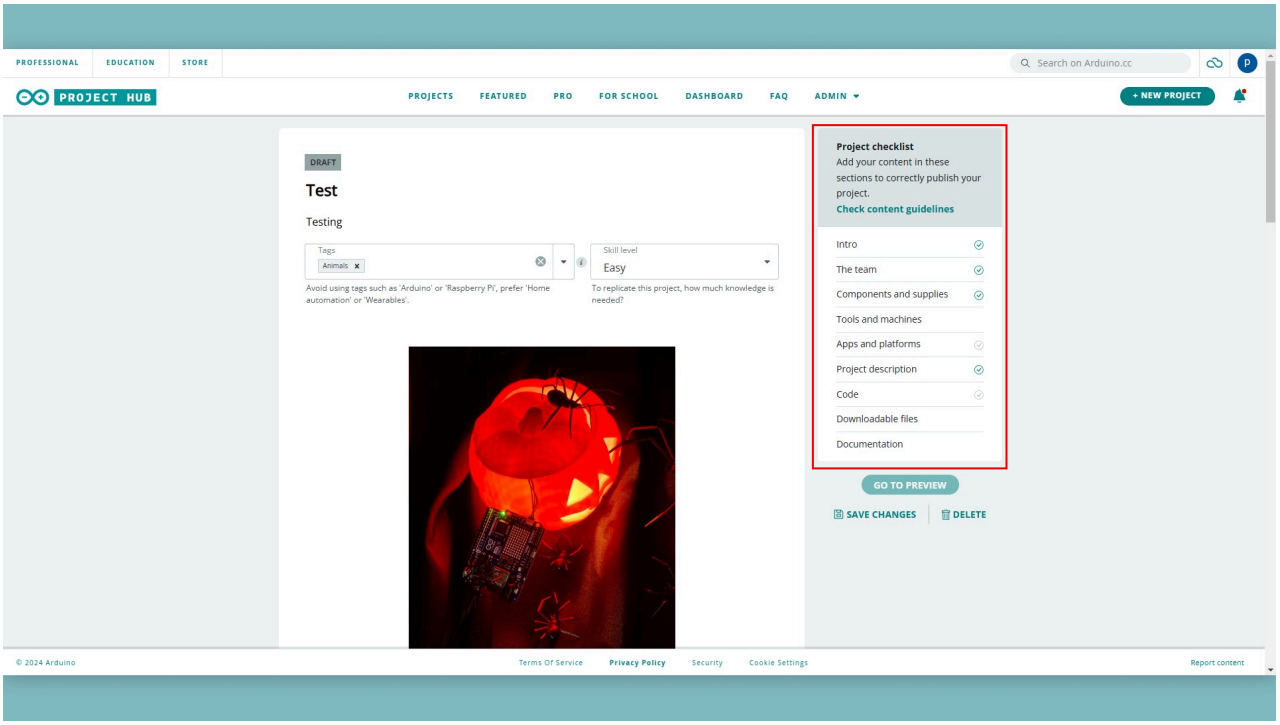
In the **Code** section you should put all the code that is needed to make the project run. You can upload code directly with a file, link your repository (GitHub repo for example) or you can embed your code directly onto the page.

In the **Downloadable files** section you should include a schematic of the project, showing how everything needs to be connected. Here you can also add *stl* files if you used anything 3D printed or any other files you think is relevant.

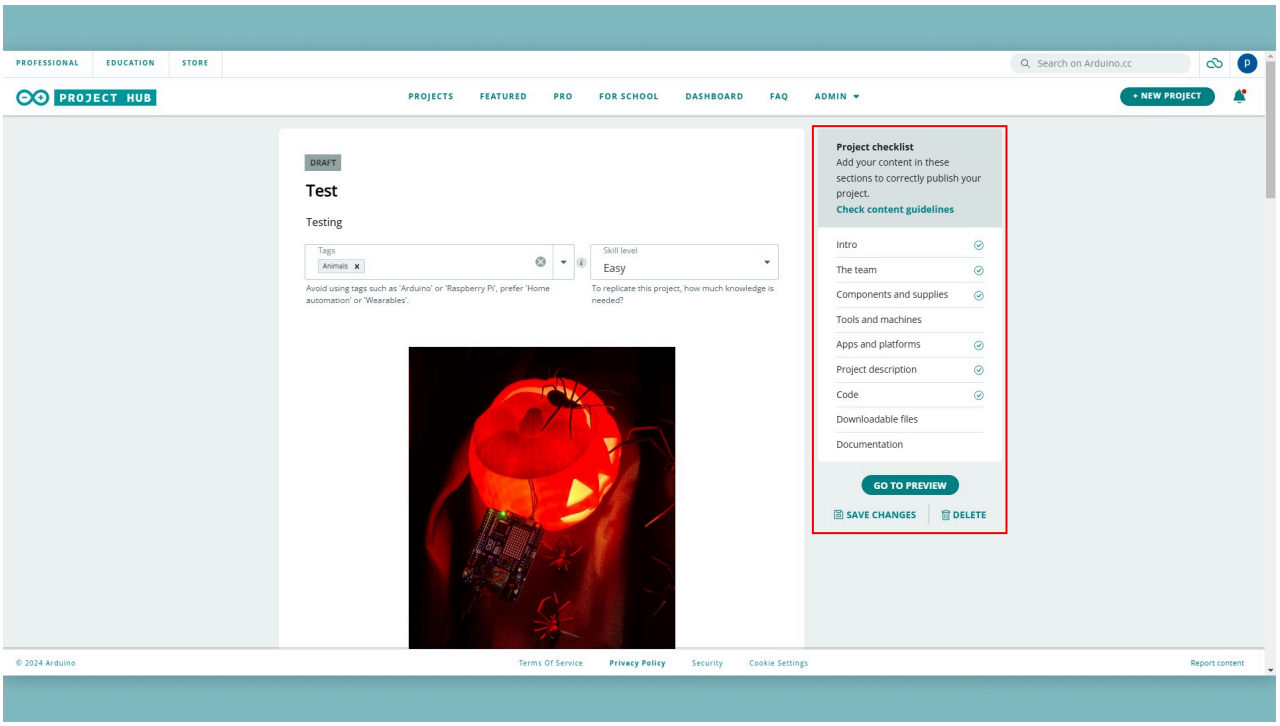
The **Documentation** section is for additional files you wish to include but it is optional.

Finishing the Project

You can check if everything that needs to be in the project is there with the checklist on the right, if the content is not there the checkmark will not be filled in. As you can see in the image here:



If you filled out all the sections with the appropriate content you can now submit the project for approval. To submit the project for approval press the **GO TO PREVIEW** button.



On the preview page press the **PUBLISH PROJECT** button to submit the project for approval.

When you submit it for approval a member of our Arduino team will review the project and see if it is complete and does not violate any of our Project Hub rules.

If you have something missing from your project your project will be marked with **"Needs revision"** and you will receive a reason for what specifically needs revision, if you can not find the reason then please check your email. If you then fix the issue you can re-submit the project for review with the same process.

Chapter 10 - External Components

An Arduino board can be connected to many different electronic components, and through different software libraries, it makes it easy to control them.

In previous chapters, we have connected some simpler components, such as LEDs and pushbuttons. In this we will focus on some slightly more advanced components.

A couple of examples are:

- Servo motors that can control for example robotic arms.
- Sensor modules that can capture data from your surroundings (such as temperature sensors).

Requirements

- Online code editor (Arduino Cloud) or offline code editor (Arduino IDE).
- Arduino UNO एक R4 WiFi / Arduino UNO एक R4 Minima
- USB-C® Cable

For each of the examples you will also need some specific components. These will be listed in the examples.

Note on Arduino Libraries

When using external components, you will most likely need to install a **library**. This is done through the library manager in the Arduino IDE.

If you are using the online editor (Arduino Cloud), all available Arduino libraries are already installed.

These libraries are written by individuals all over the world - only a small percentage are written by the official Arduino team.

It is important to understand that libraries may or may not be compatible with a specific Arduino board. If you are using an older version, it might work, while on a newer version, support has not been added.

In general, the **Arduino UNO एक R4 WiFi / Minima** boards are supported by many libraries.

Small Servo Motor

A servo motor is a motor that can be controlled with pulses using PWM. Most servo motors can be set to a specific angle (for example 90°).

There are two types of servo motors:

- **Positional rotation** - can typically rotate from 0-180°, where the angle can be configured.
- **Continuous rotation** - rotates continuously, where the speed can configured.

The most commonly available servo motor is the **positional rotation**, they are usually quite small.

Requirements

- Arduino UNO R4
- Servo Motor
- hook-up wires
- 100 µF capacitor** (read below)
- power supply (for the servo motor)
- breadboard

** When using a small servo motor, without any extra loads such as added weight or an object blocking its movement, we recommend using a **100 µF** capacitor.

Powering a Servo Motor

While it is possible to power a servo motor directly through your Arduino board, it is **not recommended.** Why is that? See a servo motor draws a lot of *current*, sometimes a lot more than the Arduino board can handle. This is bad for two reasons:

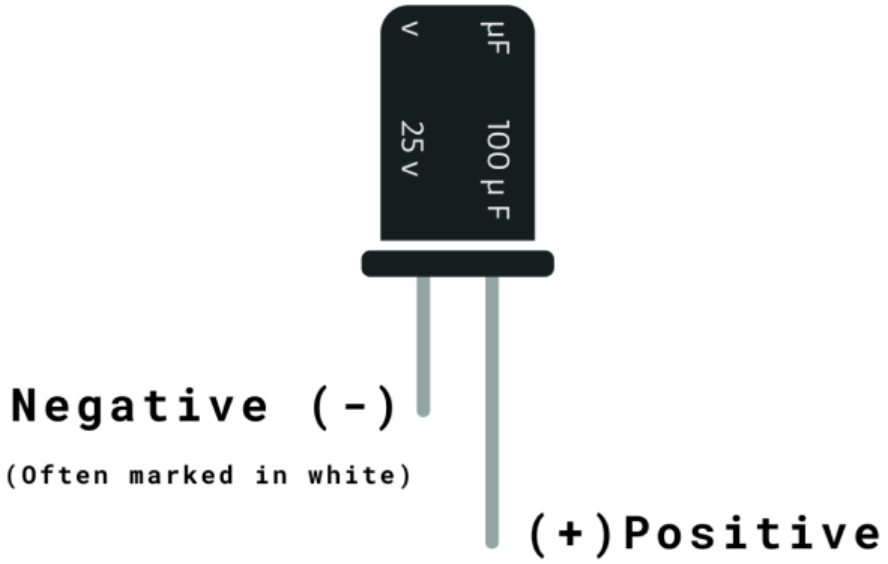
- When the servo motor has a high load (for example, it is connected to a heavier object), it draws more current, which in the best case scenario will only reset your board. This is of course not a good thing either.
- In the worse case scenario, it may actually draw so much current that it can permanently damage your board.
- Imagine a water pipe that has water running through it. Now, if you were to connect a waterfall to it, the pipe would most likely not hold on for so long. Same thing applies when using motors.

The solution to this is to **power the servo motor independently**. This way, we only connect the signal cable to the Arduino, but we draw the power from somewhere else.

To achieve this, there's a number of options:

- We can use a battery pack (that is rated with the same voltage as your servo motor).
- We can use a power supply (that is also rated with the same voltage as your servo motor).

Capacitors



Another component that is very useful when controlling servo motors is a **capacitor**. A capacitor is not strictly needed, but it makes sure your motor behaves more accurately.

As a servo motor draws high current, a capacitor can help to store energy that is not needed. This makes your servo motor movements more "smooth".

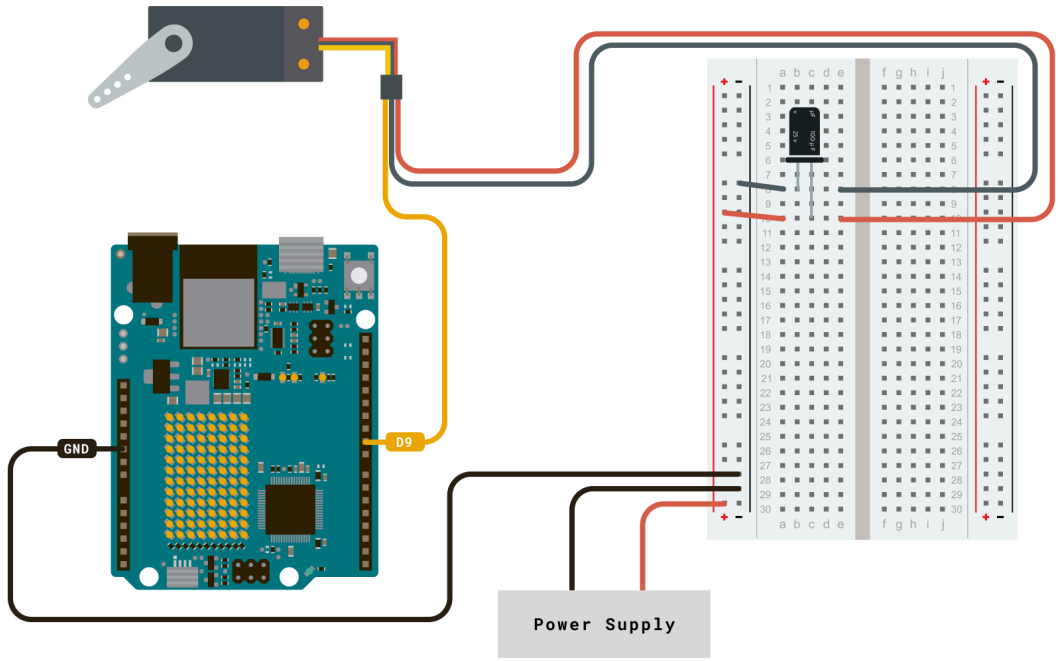
A capacitor is simply connected in parallel with your servo motor, with the **positive (+)** and **negative (-)**.

Circuit

Now that we have gone through some of the important powering options, let's see how we can actually control it using an Arduino. First, we need to connect it to the board. Follow the circuit diagram below:

- The power supply is connected to the breadboards positive and negative blocks.
- The servo motor is connected (via the capacitor) to the positive and negative blocks.
- The servo motors signal cable (typically yellow or white) is connected to a digital pin on the Arduino.
- The Arduino's GND pin is connected to the negative block on the breadboard.

Note that the power supply needs to be of the same rating as your servo motor. A common example is using 4x 1.5V batteries = 6V.



Install Servo Library

To install the servo motor library, open the Arduino IDE, and in the library manager, search for **"Servo"**. Install it.

Note that your library may already be installed. If you are using Arduino Cloud Editor, you will not need to install it.

Code

To control the servo, we can use the following code example. This example will simply move your servo motor back and forth.

```
#include <Servo.h>

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

```
int pos = 0;    // variable to store the servo position

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

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```

Upload the code to your Arduino board by clicking the **"Upload"** button.

How it Works

When you upload the code, the servo motor should start moving back and forth. Let's take a look at some of the most important aspects of the code:

- `#include <Servo.h>` - we include the Servo library.
- `Servo myservo;` - we create a "servo" object, so we can access some additional functions.
- `myservo.attach(9)` - we attach the servo motor to a pin. The pin needs to have PWM support.
- `for (pos = 0; pos <= 180; pos += 1) {}` - this for loop will increase `pos` by `1` for `180` times. This is because the max angle is `180`, and we want to run through every angle.
- `myservo.write(pos)` - this is where the magic happens. This command will send a position to the servo motor that it needs to move to.
- `for (pos = 180; pos >= 0; pos -= 1){}` - in a similar fashion to the previous for loop, we also have one that will decrease `pos` by `1` until it reaches 0.

Troubleshooting

- Is the servo not moving or making any sound? Make sure that it is powered properly.
- Is the servo moving, but sometimes seems to get stuck? It may be so that the current the motor draws is higher than what the battery / power supply can provide.
- If using a battery, you could also check the specifications of the battery. Some batteries may have a lower capacity (mAh), and you may need to replace it with a battery with a higher capacity.

BMP280 (I2C) Example

I2C is a protocol that allows you to read and send data using just two data wires. It is used for many sensors and modules, and in this example, we will take a look at the **BMP280 pressure & temperature sensor**.

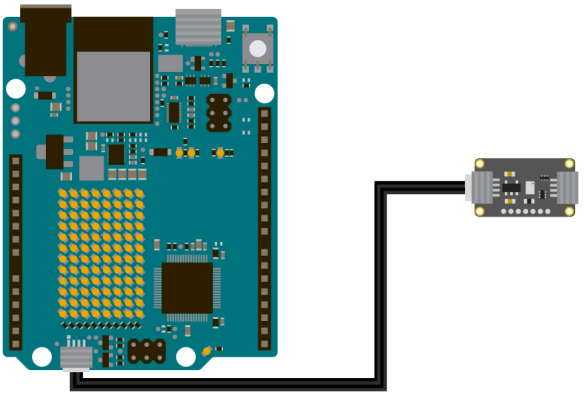
By learning this example, you will get a good idea of how to use any sensors that uses I2C for communication.

Requirements

- BMP280 Sensor (I2C)
- Qwiic Cable
- Arduino UNO एक R4 WiFi
- USB-C® Cable

Circuit

Connect the BMP280 module to the UNO एक R4 WiFi board:



Additional Note: many sensor modules does not have a Qwiic connector. Qwiic was designed to make connections easier, but you can also connect it using other pins on the UNO एक R4 WiFi.

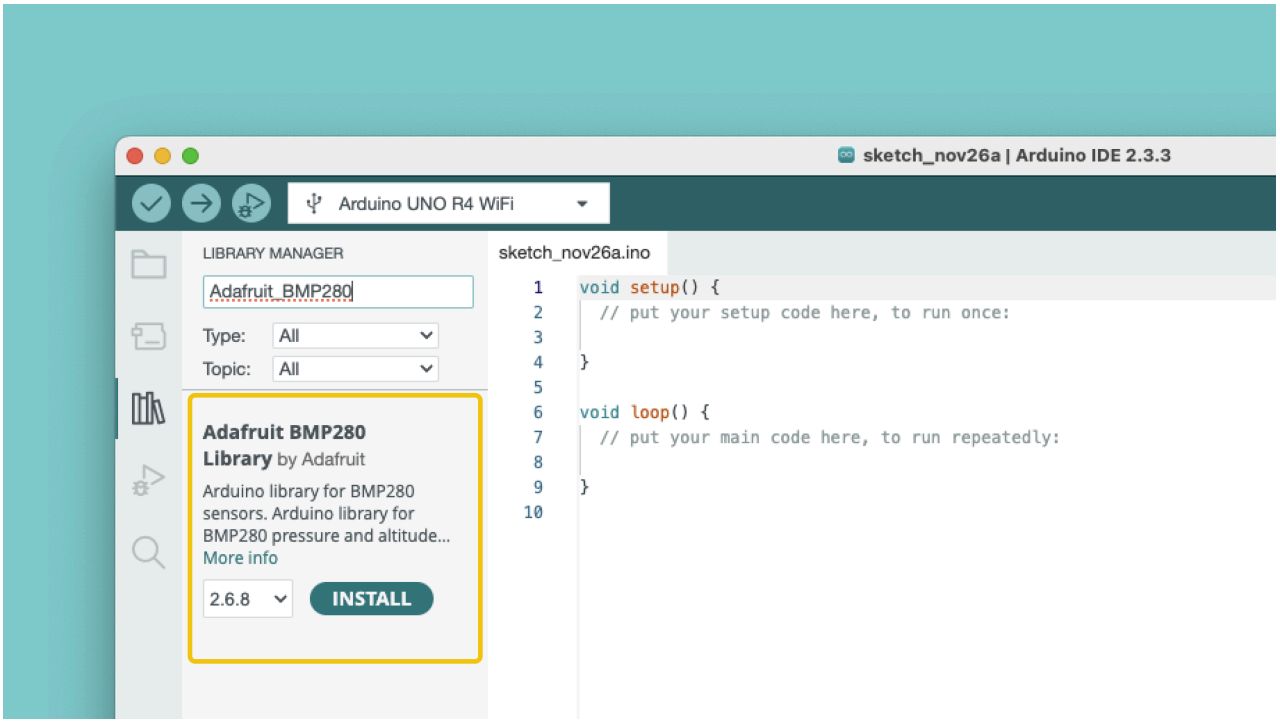
An I2C module usually have 4 available pins that we can use:

- SDA (Serial Data) - you can connect the **SDA** pin on a module to the **A4** pin on the UNO एक R4 WiFi.
- SCL (Serial Clock) - you can connect the **SCL** pin on a module to the **A5** pin on the UNO एक R4 WiFi.
- VIN/Power - connect this to the **3.3V** pin on the UNO एक R4 WiFi.
- GND - connect this to the **GND** pin on the UNO एक R4 WiFi.

Knowing the above connections will help you greatly when using any I2C based modules, not just the BMP280!

Install BMP280

First, we need to install the **Adafruit_BMP280** library. Open the library manager inside the Arduino IDE, and search for it, and install it.



Note that your library may already be installed. If you are using Arduino Cloud Editor, you will not need to install it.

Code

In this code example, we will read the **temperature** from the BMP280 sensor, and print out the values in the Serial Monitor.

```
#include <Wire.h>
#include <Adafruit_BMP280.h>

//Create an instance of the BMP280 sensor
Adafruit_BMP280 bmp;

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

  // Start the sensor, and verify that it was found
  if (!bmp.begin()) {
    Serial.println("Sensor not found");
    while (1){}
  }

}

void loop() {
  // Read the values
  float temperature = bmp.readTemperature();

  // Print to the Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" C");

  Serial.println();
  delay(2000);
}
```

Upload the code to your Arduino board by clicking the **"Upload"** button.

How it Works

Once you upload the code, you should see the values being printed in the serial monitor.

Let's take a look at some of the important aspects of the code:

- `if (!bmp.begin()) {}` - this initializes the library.
- `float temperature = bmp.readTemperature()` - this reads the temperature, and stores it in a float variable.
- `Serial.print(temperature)` - prints the `temperature` variable to the serial monitor.

Troubleshooting

If you get the `"Sensor not found"` error in the serial monitor, you can do the following:

- Check that the sensor is connected properly. If you are not using a Qwiic connector, check that SDA / SCL is connected properly (A4 - SDA, A5 - SCL).

If you are unable to upload the code to the board, make sure that:

- The board is selected.
- The library is installed.

Sometimes after installing a library, you may need to restart the Arduino IDE.