IEEE Arduino Workshop series part I

The basics

Oscar Tesniere

McGill University

September 29, 2025

Outline

A word of caution (WARNING!)

- When working with electrical circuits, always disconnect the Arduino before attempting modifications / wiring the circuitry
- Never connect the +5V line to the GND line, you may short-circuit the Arduino
- If you are unsure or confused about anything, please raise your hands and we will assist you, including verification of your circuits

External resources used in this workshop

Resources

This presentation, as well as the source code used in this presentation is available on our GitHub repository. You may want to download the *master* branch by copying the following URL in your browser:

https://tinyurl.com/278v225m

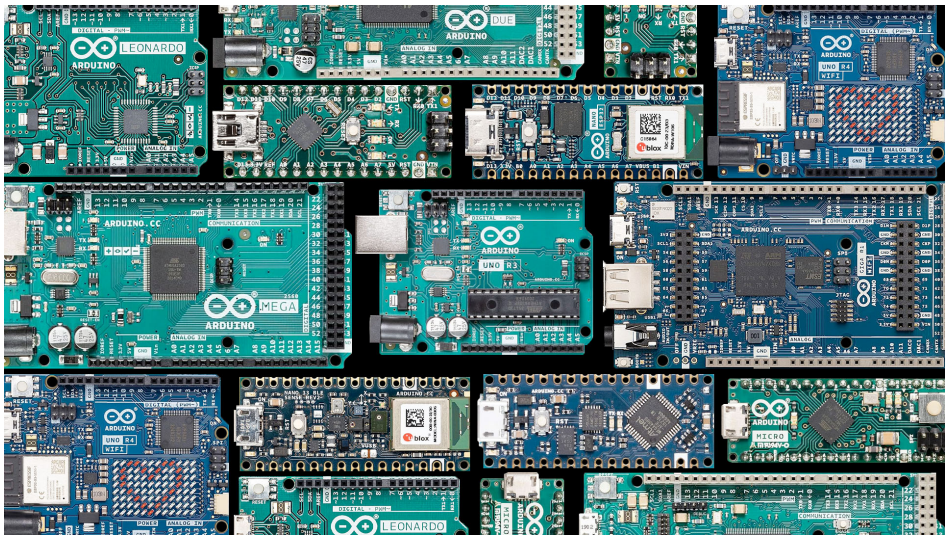
Learning outcomes

- Get hands-on experience with the C++ programming language
- Learn how to interface basic electronics components
- Build cool projects with Arduino boards

Why learn Arduino?

- Delve into electronics (and embedded systems) the easy way
- Build personal automation projects
- Participate in hackathons (RoboHacks, The Forge...)

But what is Arduino?

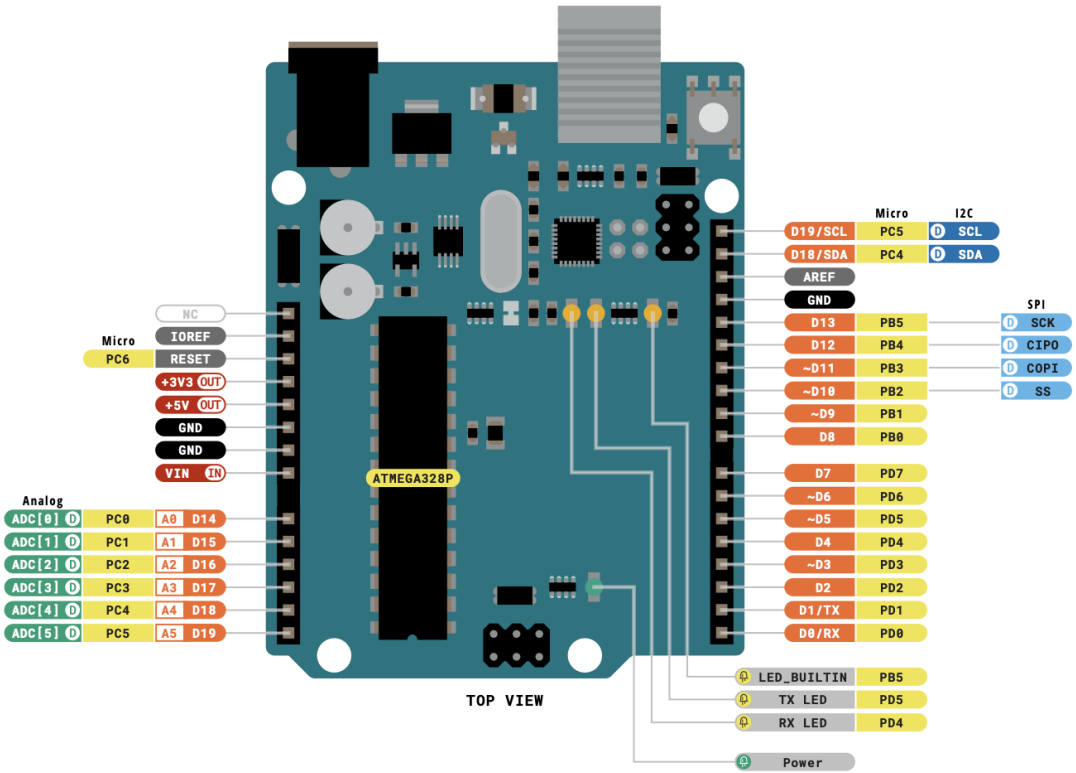


Arduino boards (hardware)

Arduino IDE (software)

Music analogy: the board is the instrument, the IDE is the sheet music

Arduino I/Os

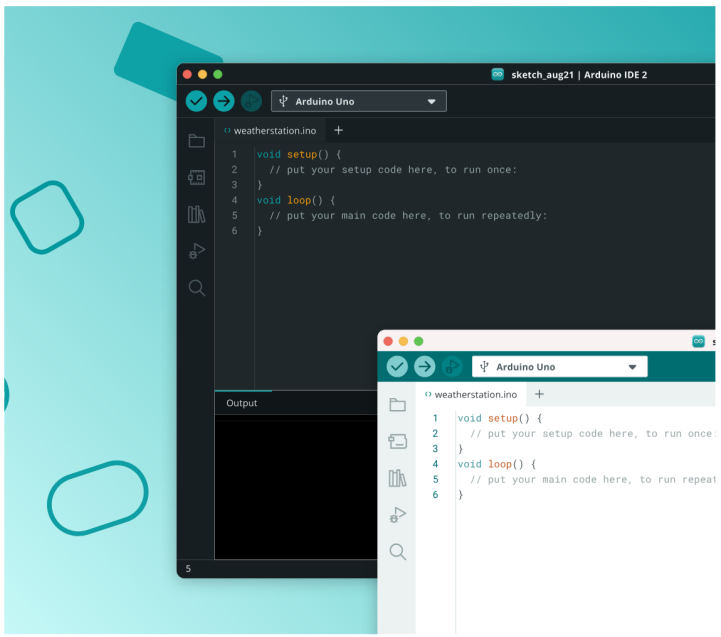


- Digital row (D2-19)
- Analog row (A0-5)
- Power row (+5V,+3V, GND)
- Serial port (USB connector)

An arduino UNO R3 board

Installing the Arduino IDE

Bring Your Projects to Life with Arduino Software



Arduino IDE 2.3.6

Release notes

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger. For more details, check the **Arduino IDE 2.0 documentation**.

macOS Intel 10.15 Catalina or newer (64-bit) | DOWNLOAD

Nightly Builds

Download a preview of the incoming release with the most updated features and bugfixes.

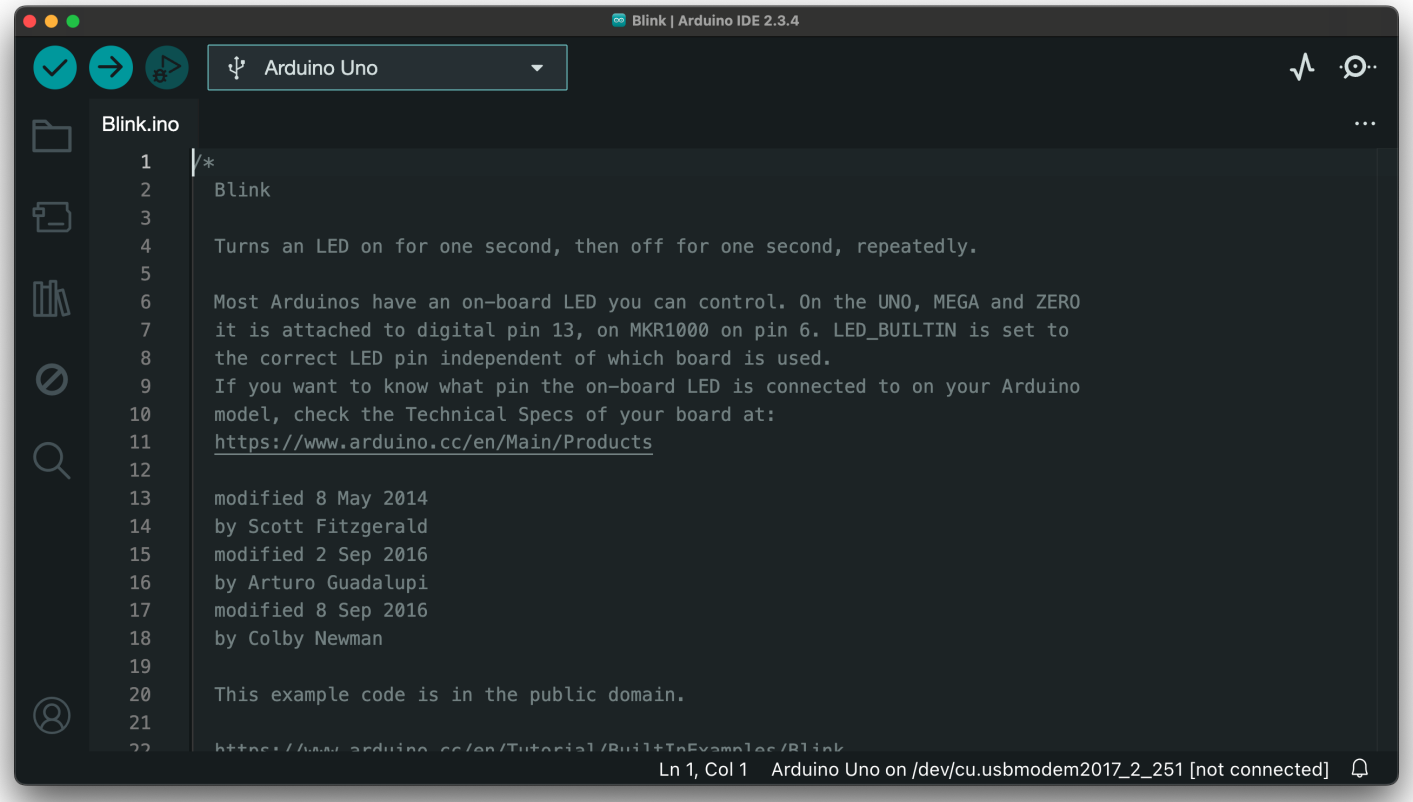
The Arduino IDE 2.0 is open source and its source code is hosted on **GitHub**.

Legacy IDE (1.8.19)

Download a legacy version of the Arduino IDE.

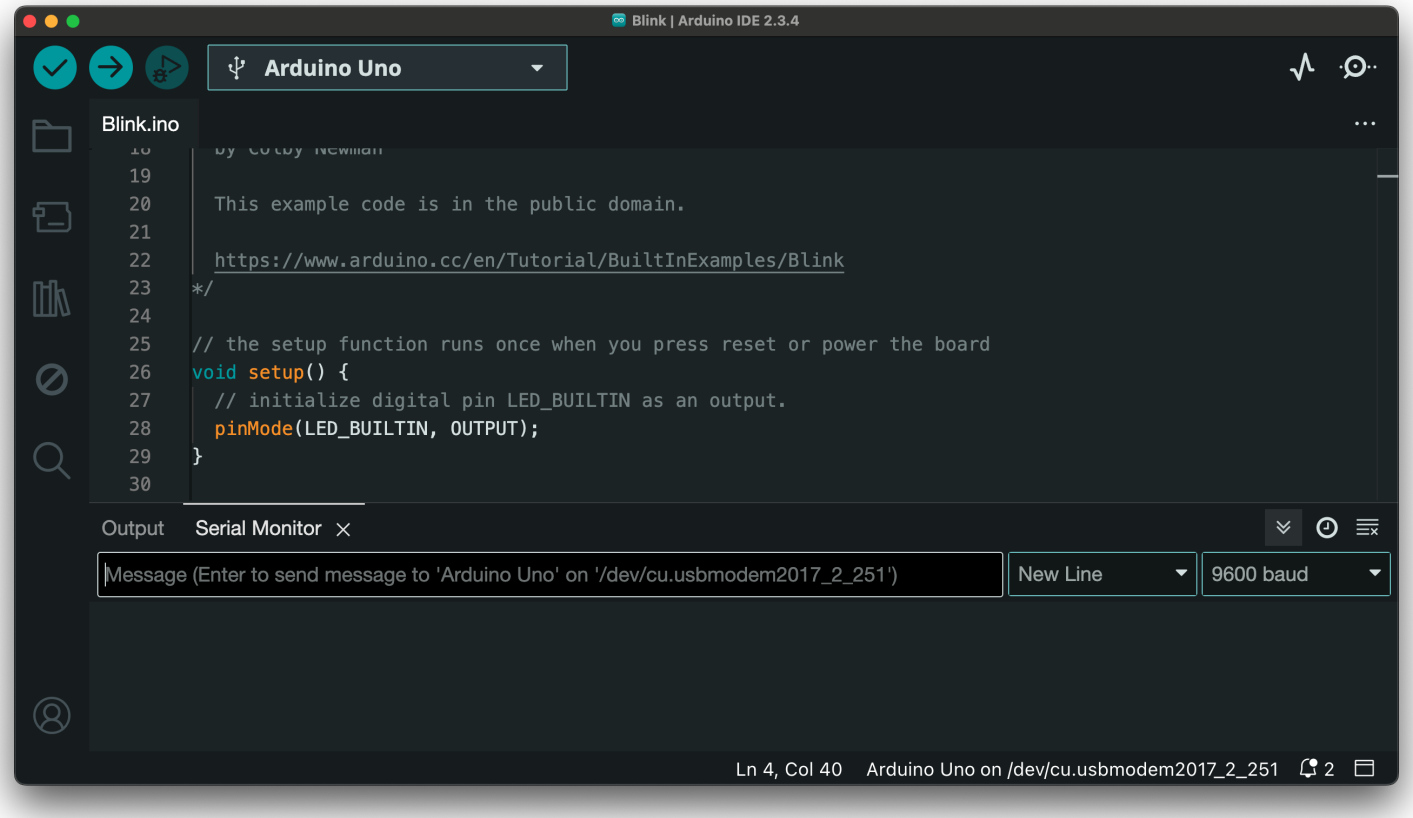
Go to Arduino official website (arduino.cc)

Installing the Arduino IDE



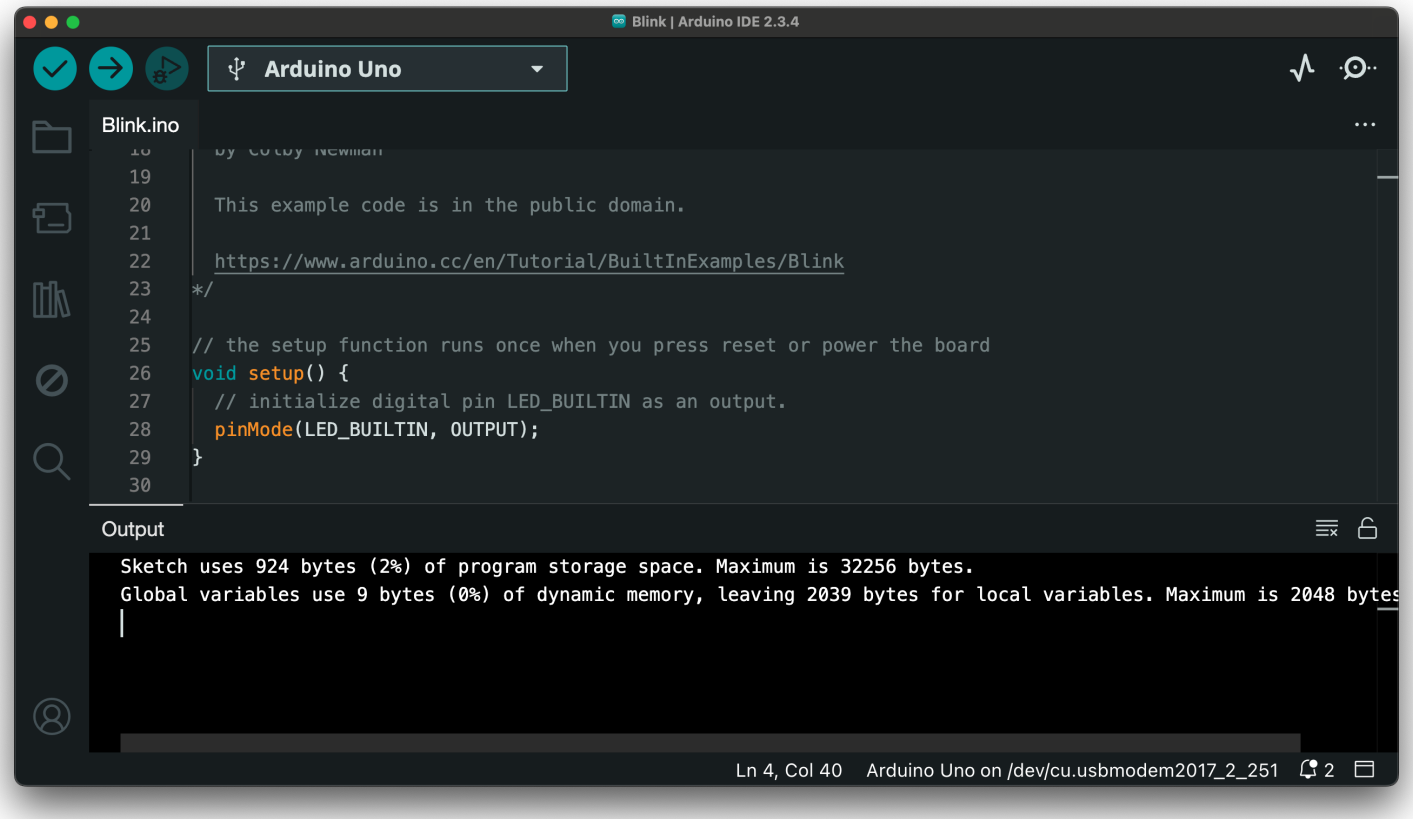
The Arduino IDE

Installing the Arduino IDE



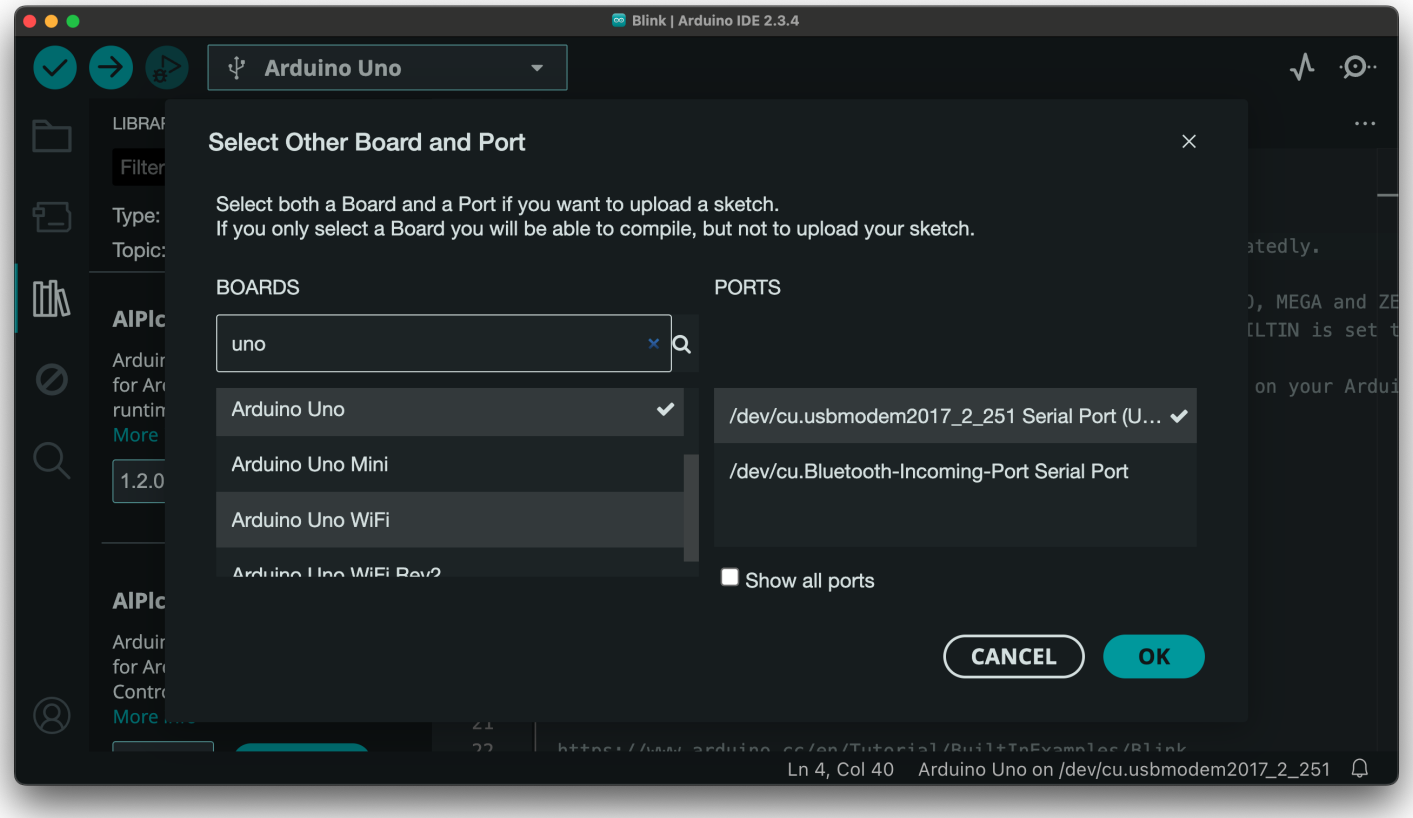
The serial monitor

Installing the Arduino IDE



The output Tab yields the result of compilation and upload

Installing the Arduino IDE



Board and Port selection tab

Analog / Digital pins

Pin type	**Pins**	**Action**
`Digital pins`	D2-15	Read/Output binary signals (1 bit)
`Analog pins`	A0-A5	Read analog signals (10 bit)

Note: Analog pins are a subset of Digital pins. All Analog pins are also digital but not all Digital pins are analog.

IEEE McGill

The Arduino language is based on C++.
Let's have a look at the syntax

The Arduino language (C++)

```
    int x = 5;
2   myFunction();
```

- All statements must end with a semicolon (;)

The Arduino language (C++)

```
while(true){
// infinite loop
}

void myFunction(){
    //function body
}
```

```
if(condition){
    //action
}
```

- All statements must end with a semicolon (;)
- All control structures (if, while, for...) and function definitions must surround inner code with { }

The Arduino language (C++)

```
// This is a single line comment

/*
this is a multiline comment
*/
```

- All statements must end with a semicolon (;)
- All control structures (if, while, for...) and function definitions must surround inner code with { }
- Single line comments can be made with // and multiline comments with /* */

The Arduino language (C++)

```
int anInteger = myValue;
float myFunction(); // functions must be typed too!
```

- All statements must end with a semicolon (;)
- All control structures (if, while, for...) and function definitions must surround inner code with { }
- Single line comments can be made with // and multiline comments with /* */
- Variables and functions (generally objects) must have a type and a scope

Variables and types

In C++, variables are typed. This means each variable is associated with a type

- int (short for integer) 16 bits

```
int myVar = 2; //initialization (all in one)

int myVar; // declaration
myVar = 2; // definition
```

Variables and types

In C++, variables are typed. This means each variable is associated with a type

- int (short for integer) 16 bits
- char (short for character) 8 bits

```
int myVar = 2; //initialization (all in one)

int myVar; // declaration
myVar = 2; // definition
```

Variables and types

In C++, variables are typed. This means each variable is associated with a type

- int (short for integer) 16 bits
- char (short for character) 8 bits
- float (floating point numbers) 32 bits

```
    int myVar = 2; //initialization (all in one)
2
    int myVar; // declaration
4   myVar = 2; // definition
```

Variables and types

In C++, variables are typed. This means each variable is associated with a type

- int (short for integer) 16 bits
- char (short for character) 8 bits
- float (floating point numbers) 32 bits
- byte 8 bits

```
    int myVar = 2; //initialization (all in one)
2
    int myVar; // declaration
4   myVar = 2; // definition
```

Variables and types

In C++, variables are typed. This means each variable is associated with a type

- int (short for integer) 16 bits
- char (short for character) 8 bits
- float (floating point numbers) 32 bits
- byte 8 bits
- etc.

```
    int myVar = 2; //initialization (all in one)
2
    int myVar; // declaration
4   myVar = 2; // definition
```

Variables and types

Variables only live within the current **scope**. The scope of a variable defines where it can be used in a program

```
    void setup() {
2   int x = 0;
    }
4   void loop() {
    x = 1; // "Compile time error : x was not
       declared in this scope"
6   }
```

A variable defined in the **global scope** (i.e. outside any block) is accessible from anywhere. A local variable can only be accessed in the local scope.

Control Structures

```
    if(condition1){
2       // do somthing if condition1 evaluates to
        true
    }
4   else if(condition2){
        // do somthing if condition1 evaluates to
        false and condition2 evaluates to true
6   }
    else{
8       // if neither condition is met
    }
```

- → if, else if and else

Control Structures

```
while(condition){
    // loops while condition evaluates to true
}

do {
 // goes through the while loop once regardless
    of the condition
}
while(condition);
```

- → While loops
- if, else if and else

Control Structures

```
    for(int i = 0; i < n; i++){
2       // loops while i < n
        // i takes the values [0;n-1] so similar to
        for i in range(n) in python
4   }
```

- → For loops
- While loops
- if, else if and else

Functions

```
int sum(int a, int b){
    return a+b;
    // function that takes two integers and
    returns another integer (int)
}

void setup(){
    // function that takes no arguments and
    returns nothing (void)
}
```

Note how both arguments and function returns are typed. The setup function is a special function as we'll see.

Arduino base sketch

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

}

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

}
```

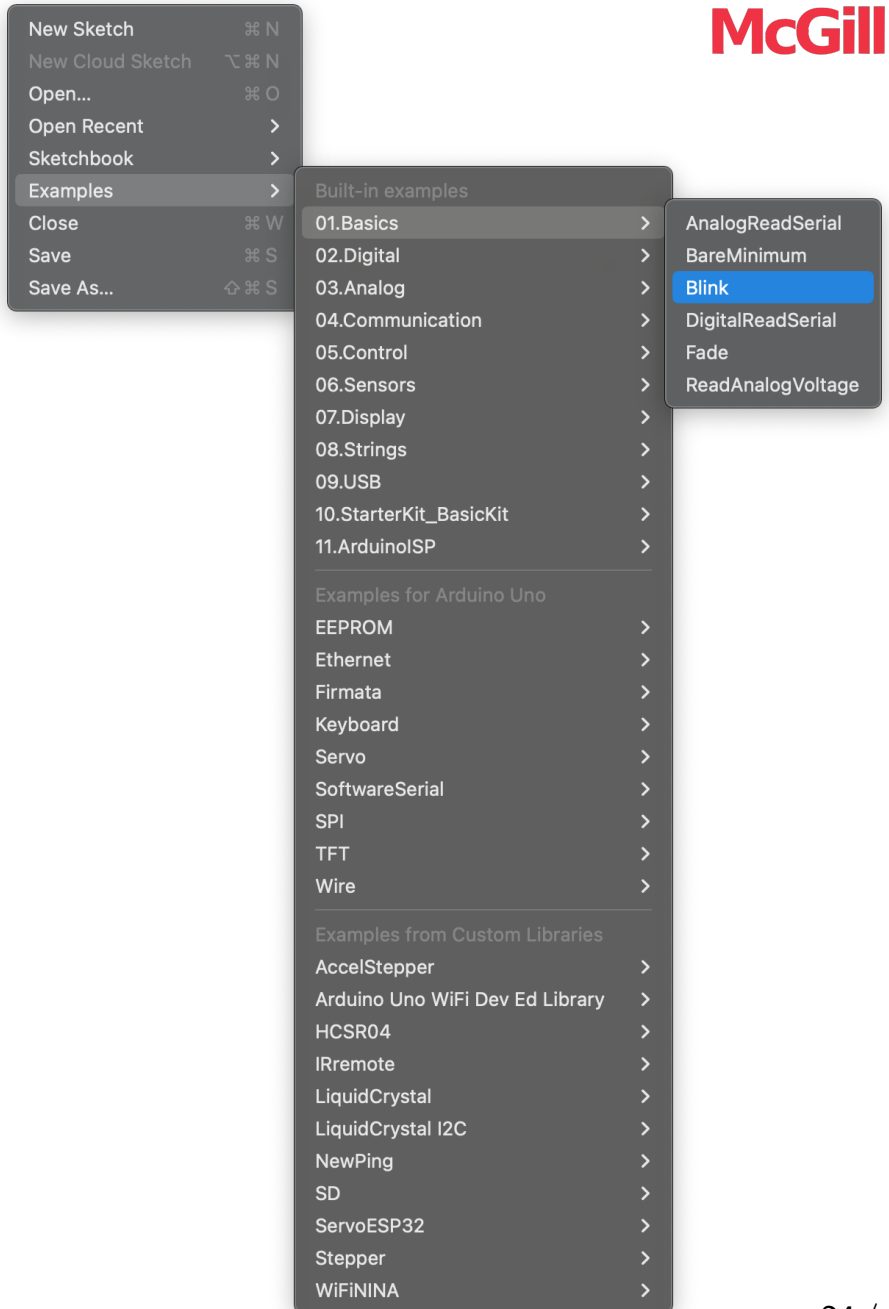
- Linear instructions are executed one after the other
- Repeating instructions repeat indefinitely or a determined number of times
- Setup runs once, while loop gets executed indefinitely
- The piece of code in the IDE is called a **sketch**

Arduino-specific syntax

- `pinMode(pin,mode)`: Configure a pin as INPUT or OUTPUT
- `digitalWrite(pin,value)`: Set a digital pin HIGH or LOW
- `digitalRead(pin)`: Read the state of a digital pin (HIGH/LOW)
- `analogRead(pin)`: Read an analog value (0–1023) from a pin
- `Serial` class:
  - `Serial.begin(baudRate)`: Initialize serial communication
  - `Serial.print()`, `Serial.println()`: Send data over serial

- **Sketch**: Your Arduino program written in simplified C++.
- **Compiler**: Converts the sketch into machine code for the microcontroller.
- **Binary (hex) file**: The compiled code uploaded to the Arduino board.
- **Uploader**: Transfers the binary to the Arduino's flash memory via USB.

Arduino Code Example (Blink) I



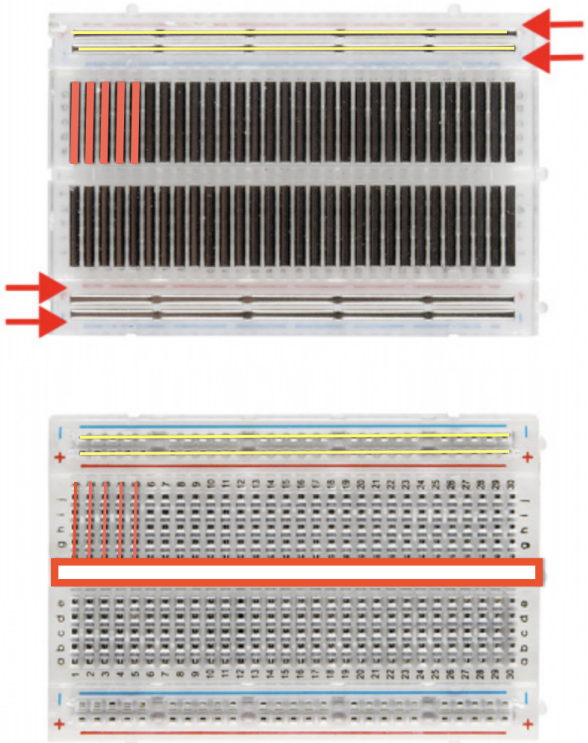
👉 Locate the Blink example in File > Examples > Basics > Blink 👈

Then, compile the code and upload it

Arduino Code Example (Blink) II

```
2   void setup() {
      // initialize digital pin LED_BUILTIN as an
       output.
4     pinMode(LED_BUILTIN, OUTPUT);
    }
6
    void loop() {
8     digitalWrite(LED_BUILTIN, HIGH);
      delay(1000);
10    digitalWrite(LED_BUILTIN, LOW);
      delay(1000);
12  }
```

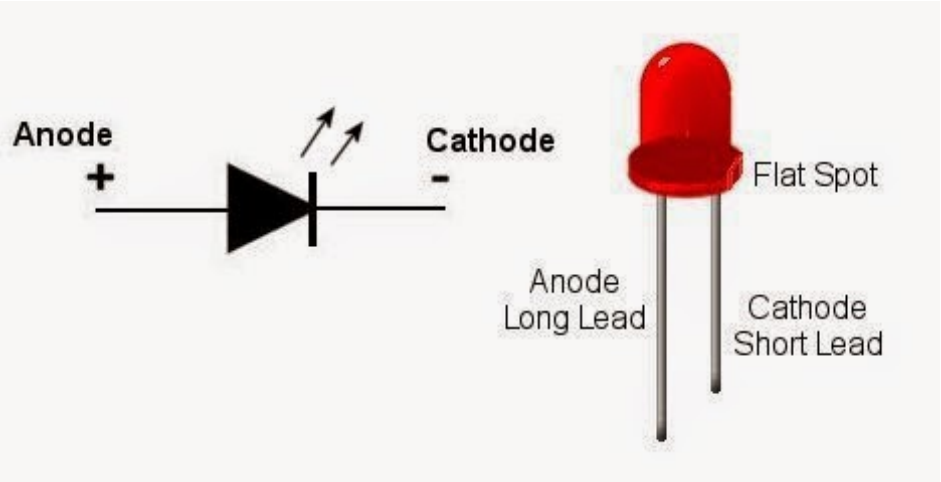
The breadboard



A breadboard (medium size)

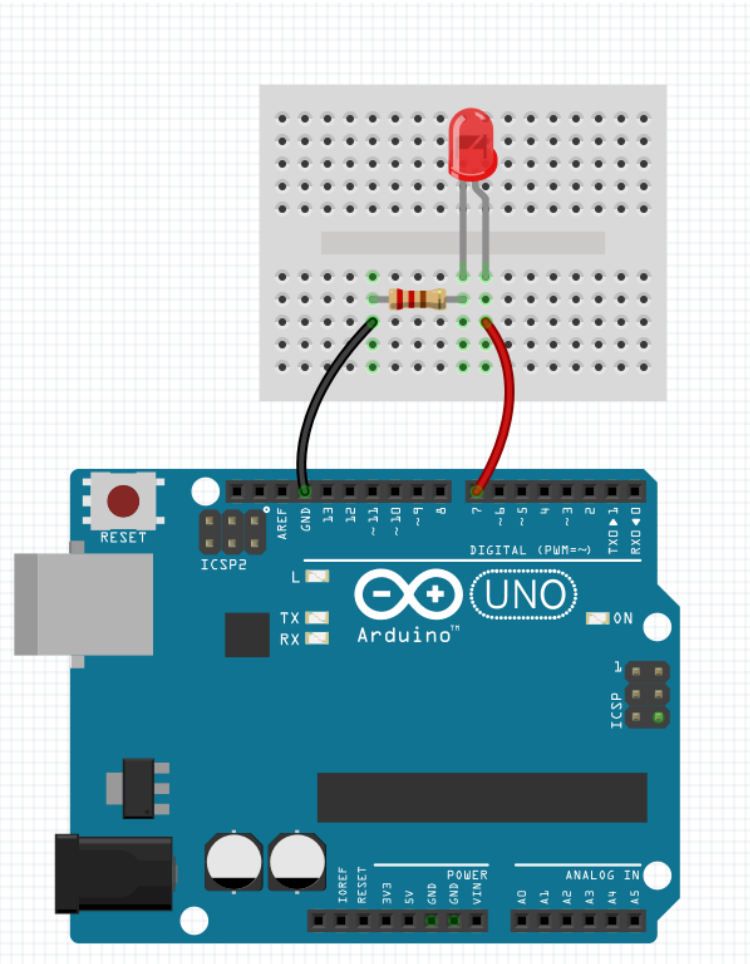
- Rows / Columns of holes connected together
- Two blocks of terminal strips and 4x power rails (usually exterior used for power)
- Used for fast prototyping with jumper wires (no soldering needed, ideal for temporary circuits)
- Convenient placement for Integrated Components over the ravine (DIP socket)
- There are multiple ways to represent a unique electrical circuit.

The LED



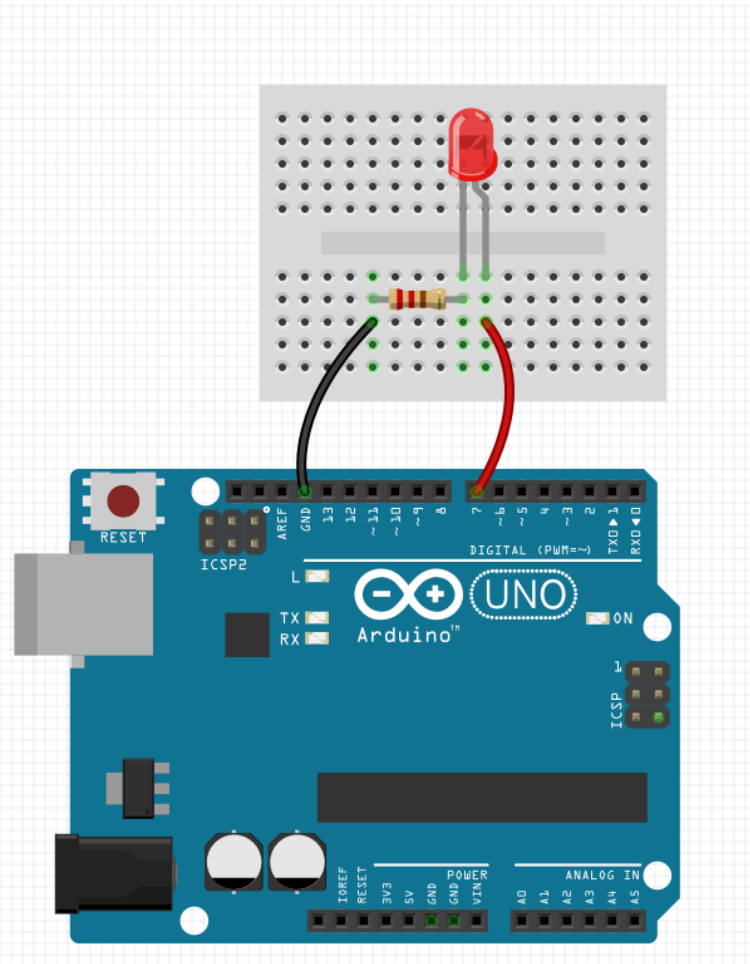
LED stands for Light Emitting Diode.

- Convert electrical energy into visible light
- Polarity matters! Think about K-cathode (–) and A-anode (+). Current flows from the anode to the cathode
- The longest lead ("leg") is the positive side
- Why do we need a current-limiting resistor? $I = \frac{V_{\text{resistor}}}{R} = \frac{V_s - V_{\text{LED}}}{R}$. Typically 220 for a 5 V power supply



👉 Connect the wiring diagram and upload the code `basic_LED_1.ino`. Once the code is uploaded, observe the behavior of the LED

- 1x 220 resistor
- 1x LED
- 2x jumper wires

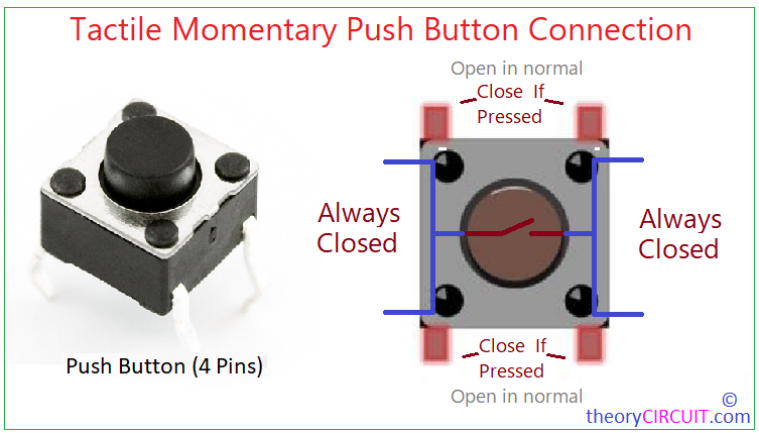


👉 Now upload the code `basic_LED_2.ino`. Observe the behavior of the LED again. What do you notice?

We used a digital pin to generate an "analog" signal that is in [0;5] V. We used Pulse Width Modulation (PWM)

Small break!

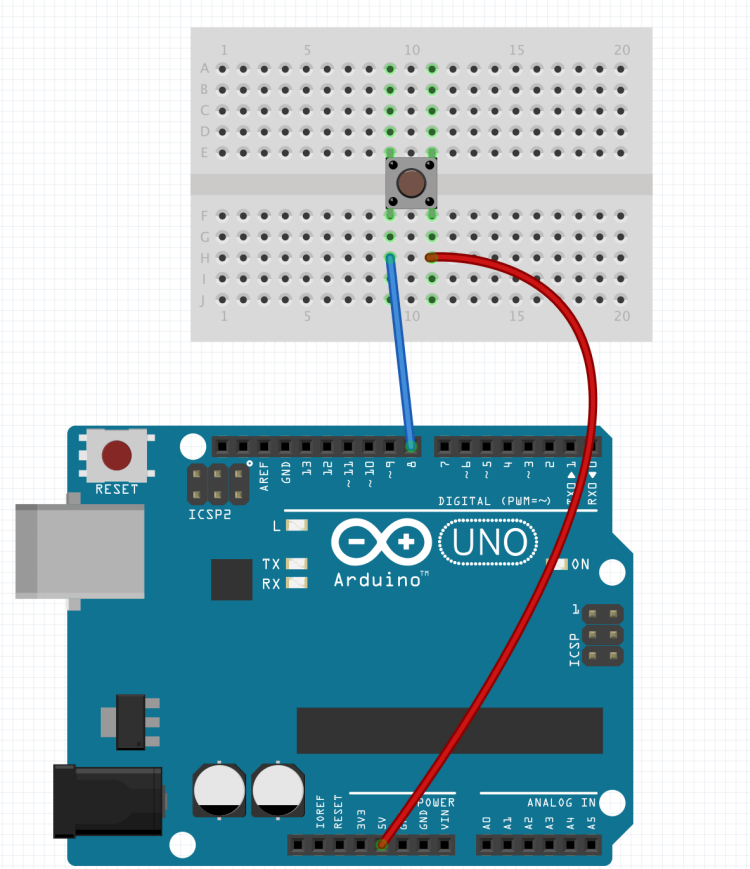
The pushbutton



A four pin pushbutton

- Four terminals (two pairs)
- The circuit is normally open and pressing the button closes the circuit
- By definition, a momentary switch (NO)

Basic button readings I



👉 Connect the wiring diagram and upload the code `basic_button.ino` Once the code is uploaded, open the Serial monitor and press the button. ▶ **Tap / Click on**

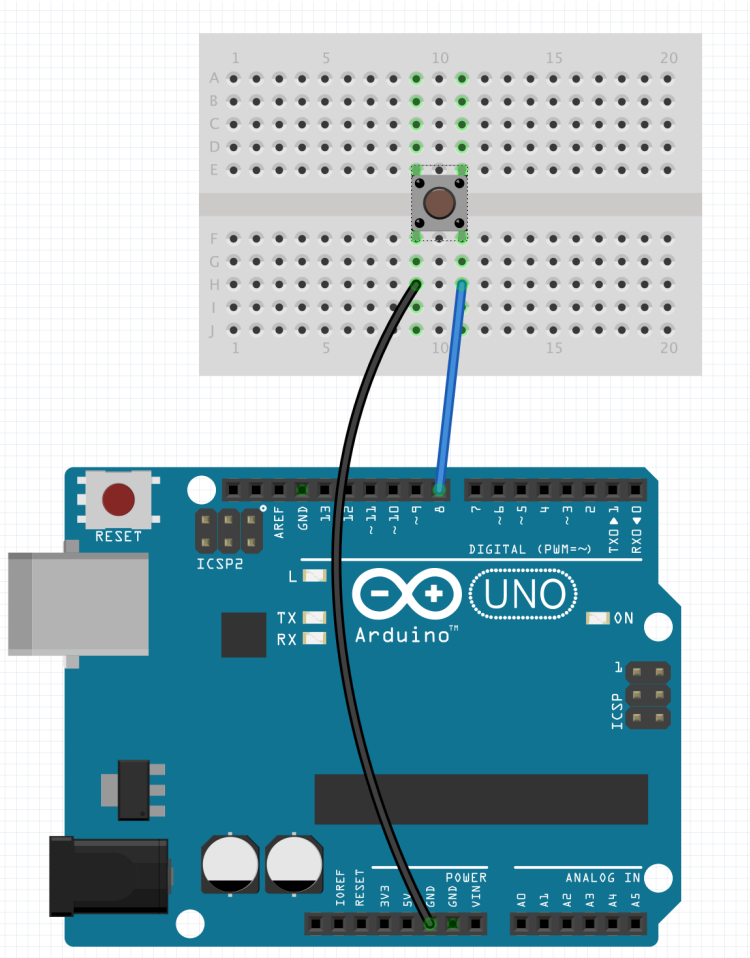
the pushbutton and monitor the output on the serial monitor

- 1x pushbutton
- 2x jumper wires

Basic button readings II

```
unsigned int pressCount;
bool onPress = false;
void setup() {
  pinMode(8,INPUT);
  Serial.begin(9600);
}
void loop() {
  if(digitalRead(8)==HIGH && onPress==false){
    pressCount++;
    onPress = true;
}
onPress = false;
delay(100); // small delay
Serial.println("You have pressed the button "+
   String(pressCount)+" times");
}
```

Basic button readings III



👉 Try different combinations!

- Swap the statement `digitalRead(8)==HIGH` with `digitalRead(8)==LOW` (line 8)
- Connect the pushbutton to +5V instead of GND

▶ **What do you notice now?**

Basic button readings IV

Now, disconnect all wires from the board and upload the code with the `digitalRead(8)==HIGH` and `digitalRead(8)==LOW` variants.

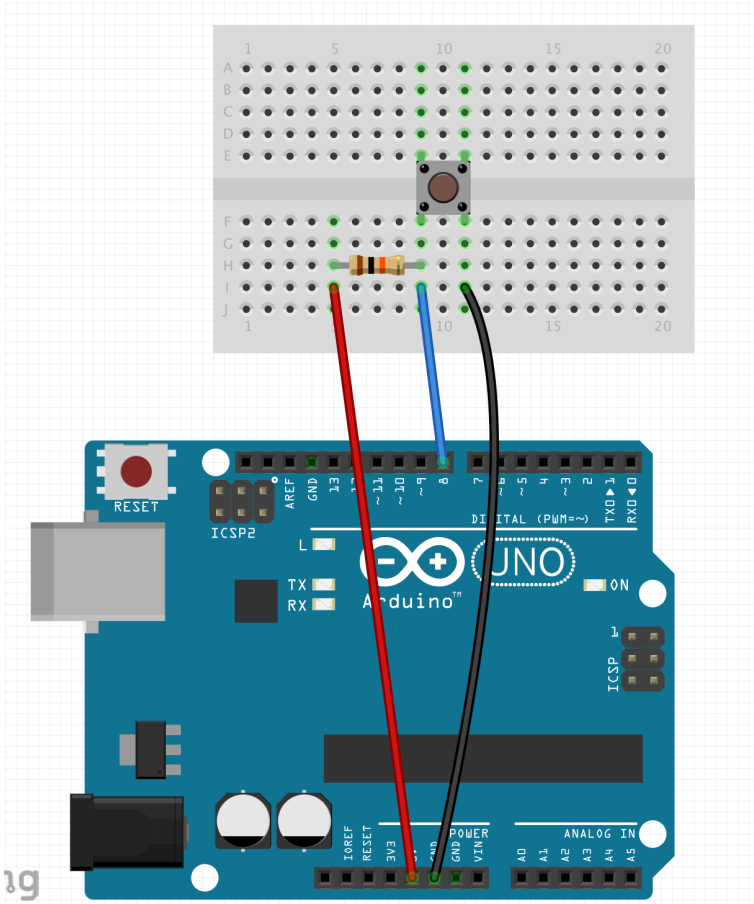
- **Does the value of pressCount change?**

Basic button readings V

Wiring configuration	**Condition (software)**	**Observation**
D8-button-GND	digitalRead(8)==HIGH	Nothing happens
D8-button-GND	digitalRead(8)==LOW	Auto increment!
D8-button-5V	digitalRead(8)==HIGH	Increment "jumps"
D8-button-5V	digitalRead(8)==LOW	A.I unless pressed
Disconnected	digitalRead(8)==LOW	Auto increment!
Disconnected	digitalRead(8)==HIGH	Nothing happens

There is a serious problem there. We can't reliably measure the number of presses. How can we do?

Basic button readings VI



👉 Now connect the following wiring diagram. You'll need a 2.2k or 2.7k resistor and an additional jumper wire. Upload the `fixed_button.ino` code.

- **Does the value of pressCount change on its own now?**
  - 1x 2.2k or 2.7k resistor
  - 1x pushbutton
  - 3x jumper wires

Why some configurations didn't work

- The digital pin is left floating and oscillates to LOW (triggering increment)
- The pushbutton bounces and generates parasite signal which increments the counter more than once / press

The pushbutton debounce problem

Solutions available

$\rightarrow$ Capacitor

The pushbutton debounce problem

Solutions available

- Capacitor

$\rightarrow$ Pull up resistor

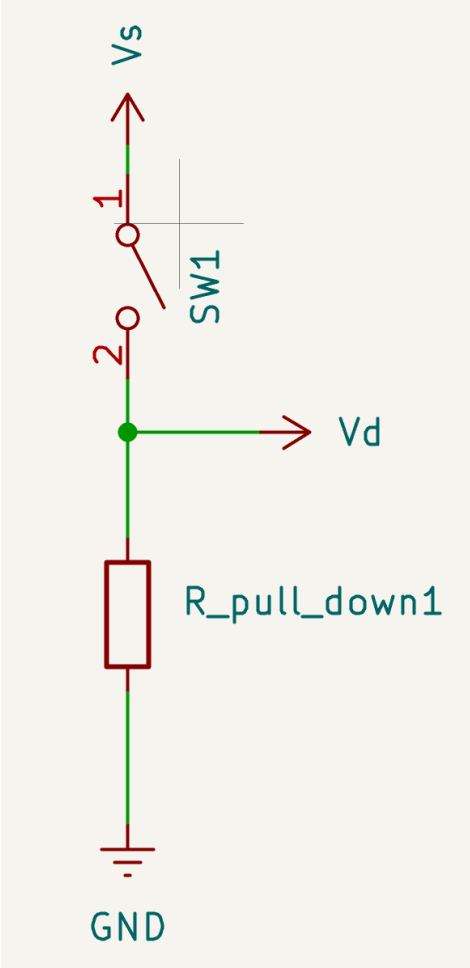
The pushbutton debounce problem

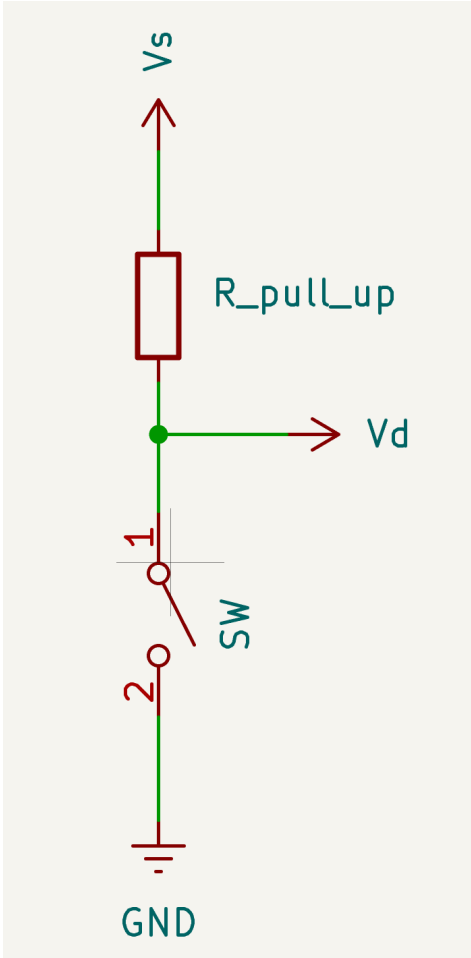
Solutions available

- Capacitor
- Pull up resistor

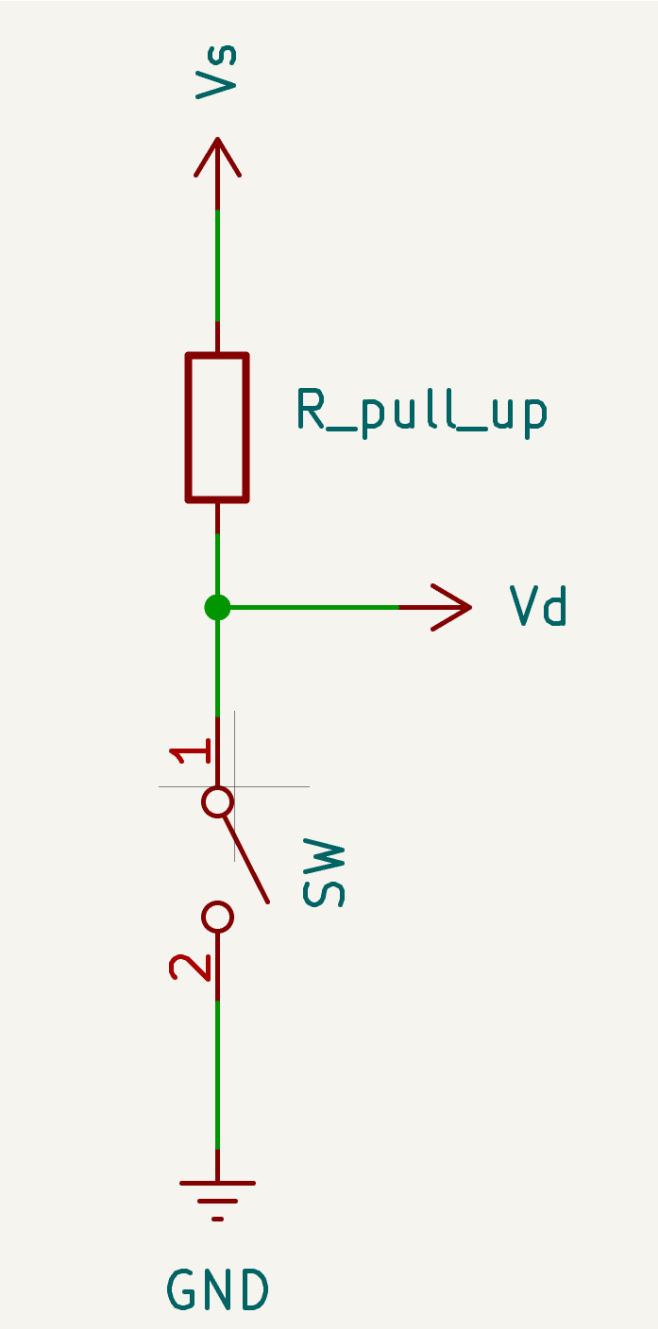
$\rightarrow$ Interrupts (Covered in WS2)

Pull UP/DOWN resistors



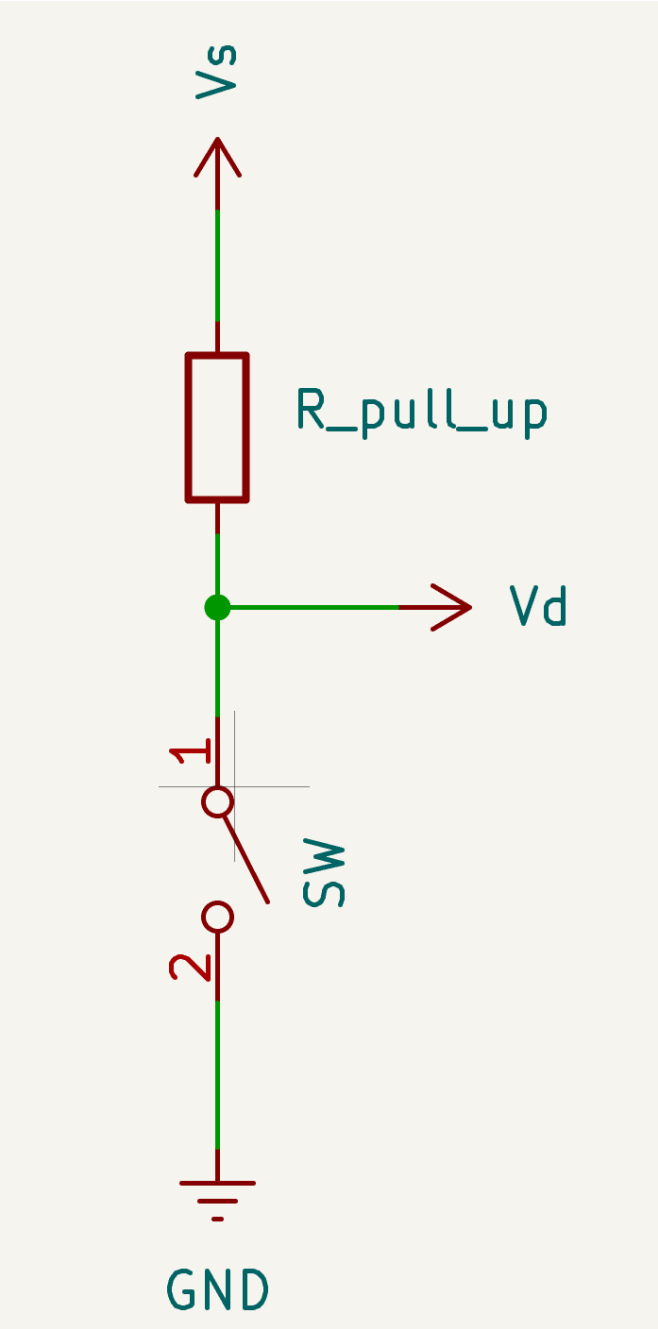


Pull-up resistors



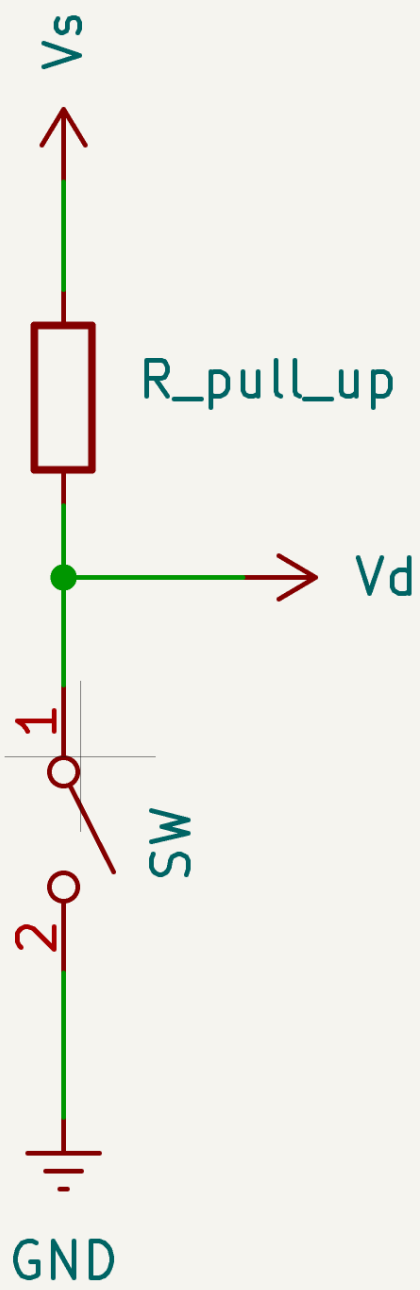
- Pull-up resistors ensure a wire is pulled to a logical HIGH in the absence of an input signal

Pull-up resistors



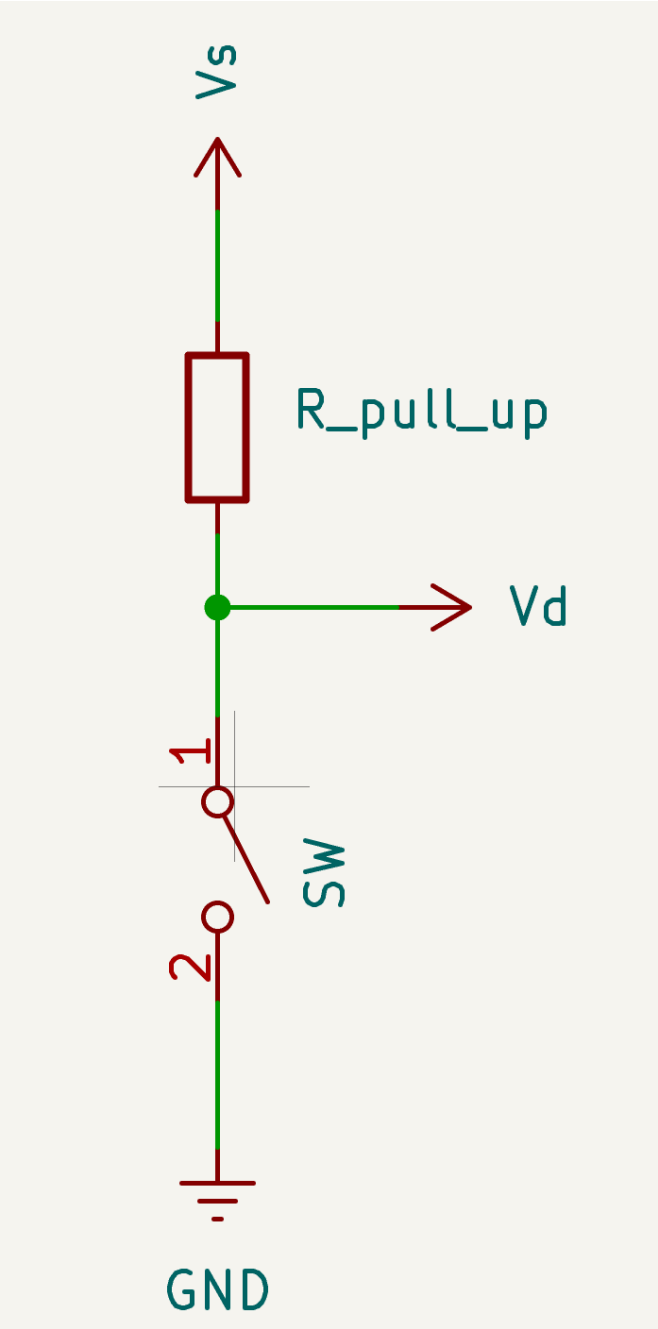
- Pull-up resistors ensure a wire is pulled to a logical HIGH in the absence of an input signal
- Digital logic circuits have three states: HIGH, LOW (when pin is OUTPUT), and FLOATING (in addition to HIGH/LOW when pin is INPUT)

Pull-up resistors



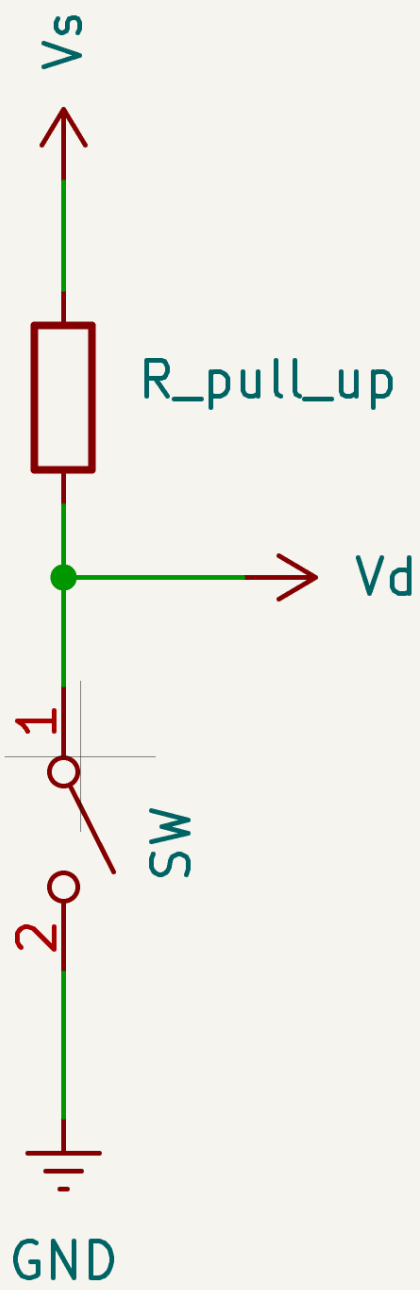
- Pull-up resistors ensure a wire is pulled to a logical HIGH in the absence of an input signal
- Digital logic circuits have three states: HIGH, LOW (when pin is OUTPUT), and FLOATING (in addition to HIGH/LOW when pin is INPUT)
- A floating input pin picks up noise $\rightarrow$ unreliable random state

Pull-up resistors



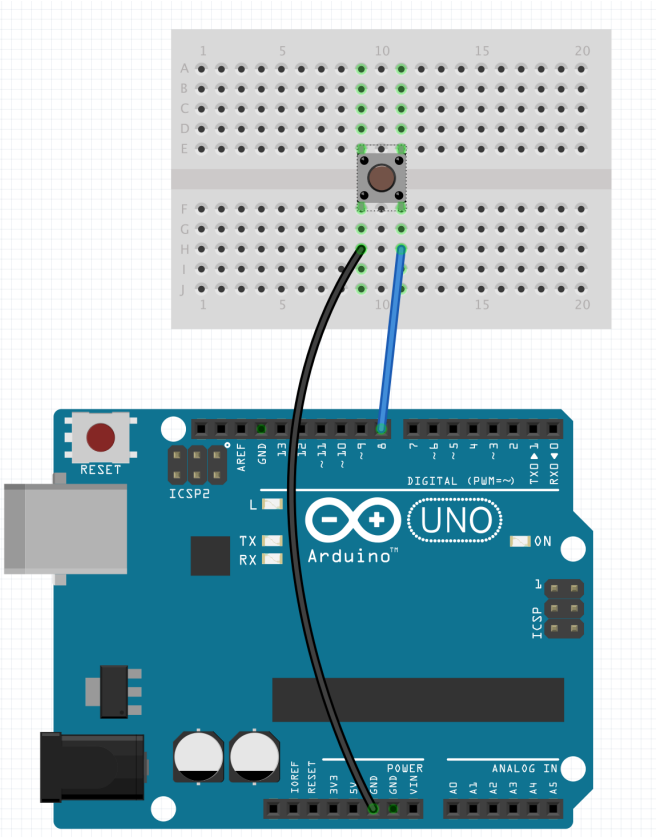
- Pull-up resistors ensure a wire is pulled to a logical HIGH in the absence of an input signal
- Digital logic circuits have three states: HIGH, LOW (when pin is OUTPUT), and FLOATING (in addition to HIGH/LOW when pin is INPUT)
- A floating input pin picks up noise $\rightarrow$ unreliable random state
- `INPUT` mode $\rightarrow$ high-impedance state (minimal current draw)

Pull-up resistors



- Pull-up resistors ensure a wire is pulled to a logical HIGH in the absence of an input signal
- Digital logic circuits have three states: HIGH, LOW (when pin is OUTPUT), and FLOATING (in addition to HIGH/LOW when pin is INPUT)
- A floating input pin picks up noise $\rightarrow$ unreliable random state
- `INPUT` mode $\rightarrow$ high-impedance state (minimal current draw)
- `INPUT_PULLUP` mode $\rightarrow$ pin tied internally to 5V (reads HIGH unpressed, LOW pressed)

Button example with `INPUT_PULLUP` I

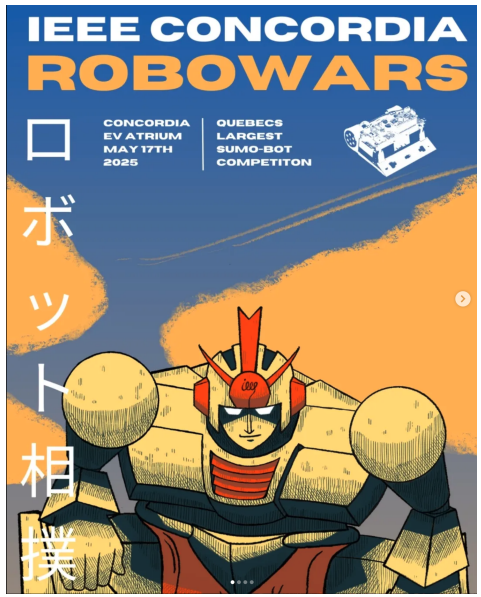


👉 Connect the wiring diagram and upload the code `builtin_pullup.ino` Once the code is uploaded, open the Serial monitor and press the button. ▷ **Tap /**

Click on the pushbutton and monitor the output on the serial monitor

Any questions?

We need you!



We're interested in forming a team for the upcoming Robowars hackathon in May 2026!

contact me at ieee.competitions@mail.mcgill.ca

Questions? Comments? Suggestions? Threats? Insults?

IEEE McGill

Feedback Form

Workshop Interest Form

Further Reading I

Arduino Cookbook
O'Reilly, 3rd Edition, 2020.

Till Tantau.
The Beamer Class.
Available at: https://ctan.org/pkg/beamer

Arduino learning
Available at:
https://docs.arduino.cc/learn/microcontrollers/digital-pins/

Available at: https://eepower.com/resistor-guide/resistor-applications/pull-up-resistor-pull-down-resistor/#

Available at:
https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all

Arduino Commands A–Z

2

A `analogRead(pin)`, `analogWrite(pin, value)`

D `digitalRead(pin)`, `digitalWrite(pin, value)`, `delay(ms)`

M `millis()`, `map(value, fromLow, fromHigh, toLow, toHigh)`

P `pinMode(pin, mode)`, `pulseIn(pin, value)`

S `Serial.begin(baud)`, `Serial.print(val)`, `Serial.println(val)`