

PUSH BUTTON SWITCH

In this complete tutorial you will learn how to use a push button with Arduino, with different circuit configurations. You will also see how to use the push button for various applications, and take advantage of some of the Arduino capabilities, for example interrupts.

I'm going to use an Arduino Uno board, but this tutorial also works for any other Arduino board you can find.

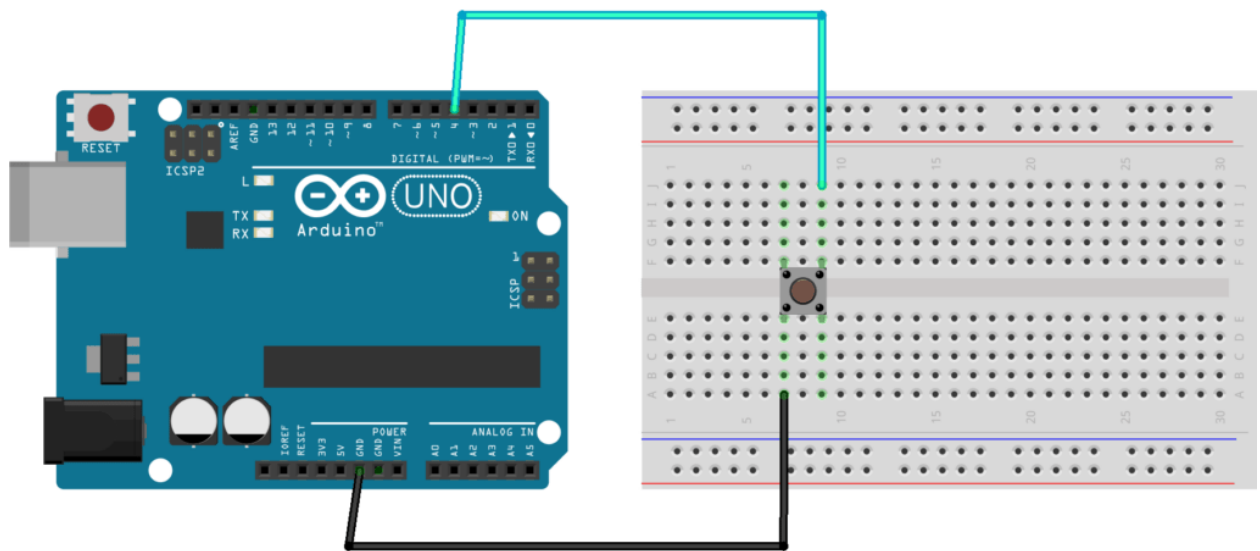
Let's start!

PUSH BUTTON CIRCUIT

For the circuit you will need:

- Arduino board
- Breadboard
- Some wires
- Push button

Here's the circuit we'll make.



fritzing

STEPS TO BUILD THE CIRCUIT:

Make sure to power off the Arduino.

Plug the push button in the middle of the breadboard, like on the picture.

On one button's leg, plug a wire (black if possible) to a GND pin on the Arduino board.

The top left and bottom left legs of the button are connected together, and the top right and bottom right legs are connected together. If you have plugged the GND wire on the left side, then plug another wire on the right side, so they are not connected together. This other wire goes to a digital pin, for example 4.

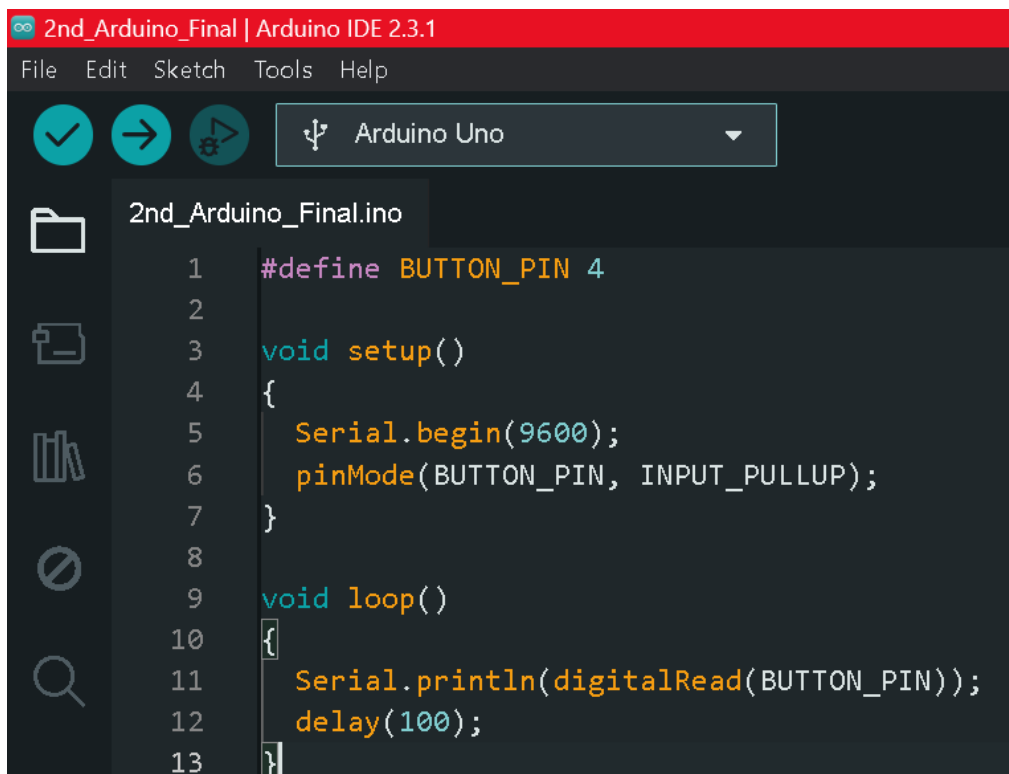
As for now there is no resistor involved in the circuit. In the following of this post we'll talk more about this. Also, if you want to know more about Arduino pins, check out this [Arduino Uno pinout guide](#).

ARDUINO CODE TO READ PUSH BUTTON'S STATE

What we're trying to do here is to simply read the state from the button and print it 10 times per second.

Print the push button's state

Here is the code to print the button's state 10 times a second.



```
2nd_Arduino_Final | Arduino IDE 2.3.1
File Edit Sketch Tools Help

[Icons] Arduino Uno

2nd_Arduino_Final.ino
1  #define BUTTON_PIN 4
2
3  void setup()
4  {
5      Serial.begin(9600);
6      pinMode(BUTTON_PIN, INPUT_PULLUP);
7  }
8
9  void loop()
10 {
11     Serial.println(digitalRead(BUTTON_PIN));
12     delay(100);
13 }
```

LET'S BREAK DOWN THIS CODE LINE BY LINE.

Code to setup the push button

```
#define BUTTON_PIN 4
```

First we create a #define for the button pin, so we don't need to write "4" every time we want to use this pin in our code. This will make your life easier, especially if you want to plug the button to another pin later on. This way you just need to modify one line of the code, which is at the top of the program – and thus easy to find.

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

The void setup() function will be called first, only once. What we do here is to initialize Serial communication, so that we can use the Serial Monitor to print the data we got from the button.

```
  pinMode(BUTTON_PIN, INPUT_PULLUP);  
}
```

This is how you initialize the push button in your code. In the void setup(), you use the pinMode() function with 2 arguments: first the button's pin – here BUTTON_PIN will be replaced by "4" – and then the mode we want for the pin.

As we are going to read data from the button – not write – we choose an input mode. And then we still have 2 options: INPUT or INPUT_PULLUP. As we don't have any resistor in our circuit, we will use INPUT_PULLUP to use the internal pull up resistor of the Arduino board. Note that in this tutorial I'm not going to dive deep into the explanation of INPUT_PULLUP, but if you want to know more, checkout this [Arduino INPUT_PULLUP tutorial](#).

Code to read the push button's state

```
void loop()
{
  Serial.println(digitalRead(BUTTON_PIN));
```

Now the pin for the button is initialized as input, and we enter the void loop(), which will be called again and again an infinite number of times.

To read the button's state, we use the digitalRead() function, with one argument: the button's pin. The result of that function is going to be either HIGH or LOW. Here, because we are in a pull up configuration (with INPUT_PULLUP), when the button is not pressed you will read HIGH. When the button is pressed you will read LOW.

Then, we put the result of digitalRead() into the function Serial.println(), which will simply print the data on the Serial Monitor.

```
    delay(100);
}
```

Just after reading the button's state and printing it, we add a delay() of 100 milliseconds, which means that we're going to do this action roughly 10 times per second. After we exit the void loop(), it is called again, etc etc.

Now you can test the code by compiling it and uploading it to your Arduino.

Open the Serial Monitor, and press/release the push button several times.

Every 100 milliseconds you will have a new line, with either 0 or 1.

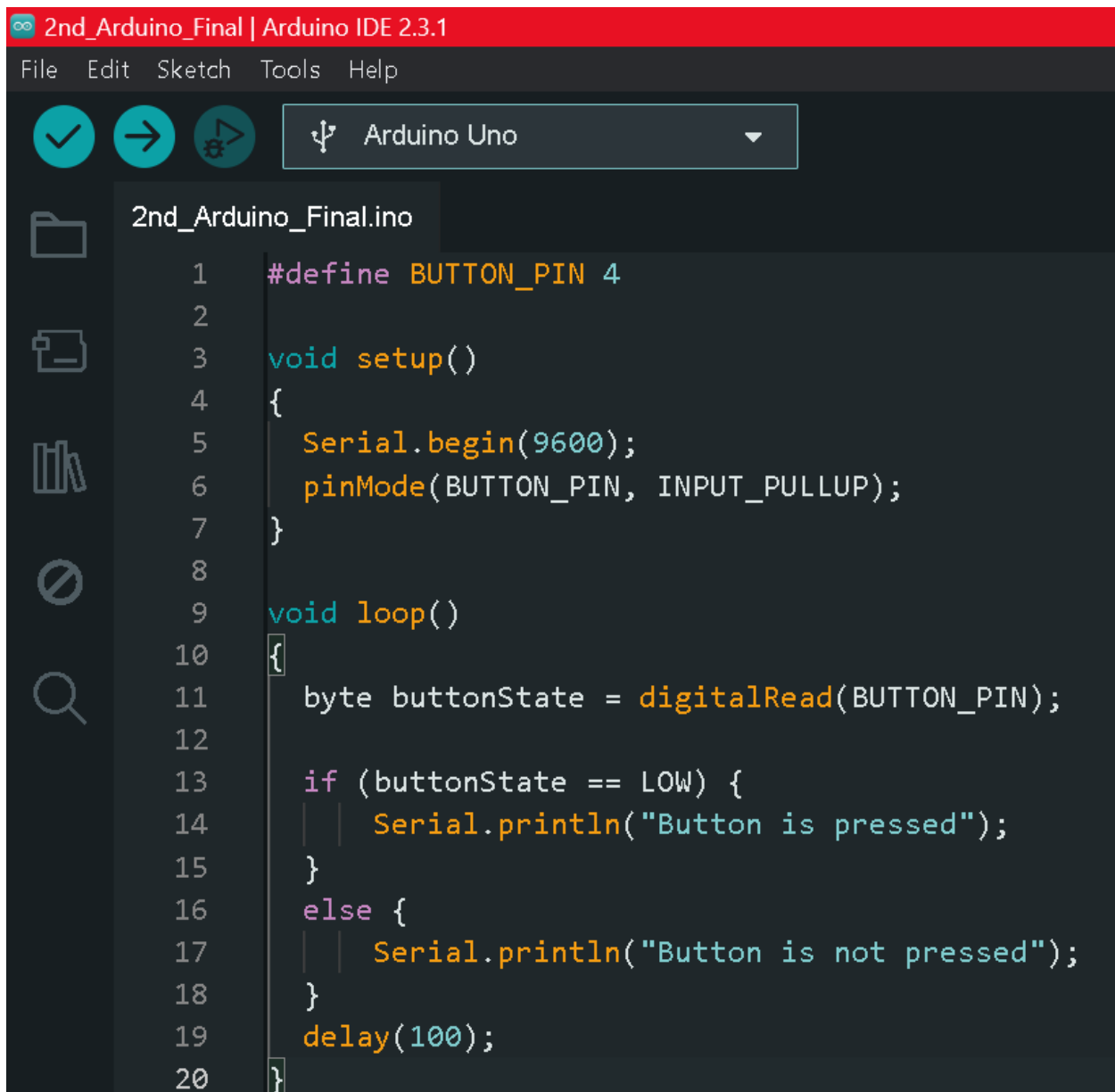
As I previously wrote, the value you get out of digitalRead() is HIGH or LOW, but when you send it through Serial, this is going to be converted to 1 (HIGH) or 0 (LOW).

So, when the button is not pressed you will see "1" printed, and when the button is pressed, you will see "0".

IMPROVEMENT: PRINT A MORE EXPLICIT MESSAGE

This first code is great as a minimal example to see the data coming from the push button. But in a real application, you may want to do more interesting things than just printing the button's state. For example, you could read the button's state, keep it inside a variable, and then depending on the value, do a specific action.

Let's use the following Arduino code.



```
2nd_Arduino_Final | Arduino IDE 2.3.1
File Edit Sketch Tools Help

[Checkmark] [Next] [Previous] [USB] Arduino Uno

2nd_Arduino_Final.ino
1  #define BUTTON_PIN 4
2
3  void setup()
4  {
5      Serial.begin(9600);
6      pinMode(BUTTON_PIN, INPUT_PULLUP);
7  }
8
9  void loop()
10 {
11     byte buttonState = digitalRead(BUTTON_PIN);
12
13     if (buttonState == LOW) {
14         Serial.println("Button is pressed");
15     }
16     else {
17         Serial.println("Button is not pressed");
18     }
19     delay(100);
20 }
```

The setup phase is the same. And as you can see in the loop(), now we store the value we got from digitalRead() into a variable, that we name buttonState. The data type for this variable is byte, which is what you need for the values HIGH and LOW.

Then, with a simple if structure, we check if the state is LOW, which means that, in our configuration with the internal Arduino pull up resistor, the button is pressed. When we detect this, we can decide to do any action we want. And if the condition is false, which means that the state is HIGH (because there are just 2 possible states), we know that the button is not pressed.

Now you can run this code, and see "Button is not pressed" printed every 100 milliseconds, and when you press on the push button, "Button is pressed".

```
Button is not pressed
```

```
Button is not pressed
```

```
Button is not pressed
```

```
Button is not pressed
```

```
Button is pressed
```

```
Button is pressed
```

```
Button is pressed
```

```
Button is not pressed
```

```
Button is not pressed
```

This is a nice improvement to the previous code, which gives us more context and allows the code to grow with more functionalities.

REFERENCE

https://roboticsbackend.com/arduino-push-button-tutorial/#Using_pull_updown_resistors