Working with Digital Input

Now that you've explored and experimented with *Digital Output*, it's time to jump into it's complimentary concept: *Digital Input*. Before starting, let's do a quick refresher about the basics of Digital Output.

Digital Output Recap

While working with the LED you used *digitalWrite()* to send a HIGH or LOW signal to an LED pin. The HIGH signal corresponded to the LED turning ON, whereas the LOW message corresponded to the LED turning OFF. This is the core aspect of Digital Output: you're sending either an ON or an OFF signal- there is no middle ground.

The most basic example of this ON/OFF output is the LED changing states. Next you'll be exploring and experimenting with an ON or OFF input.

Basics of Digital Input

Next, let's continue with the LED example, but instead of looking at the output you're going to now consider the input.

Recall that digital has two states, the form that this takes for input would again be ON and OFF. The most basic example of digital input is a *light switch*. A typical, straight forward light switch has two states: ON and OFF. This is digital input.

Digital Input with Arduino

Recall that *digitalWrite()* is the command linked with *digital output*. Now let's introduce the corresponding command for *digital input* : *digitalRead()*.

digitalRead() works similarly to its complimentary pair in that it requires a pin as a parameter. You'll also need to configure **pinMode()** to be set for *input* instead of output like you've used thus far.

The *digital input* "Hello World" project is to use a pushbutton to turn an LED on and off. Previously, the LED state was controlled entirely by the code that you sent to the board. This time, a physical button is going to be controlling the state.

Let's examine some pushbutton code from the Arduino Reference:

```
int ledPin = 13; // LED connected to digital pin 13
int inPin = 7;   // pushbutton connected to digital pin 7
int val = 0;     // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin 13 a
  pinMode(inPin, INPUT);      // sets the digital pin 7 as i
}

void loop()
{
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);    // sets the LED to the butto
}
```

Now, let's break this code down by section:

Code Breakdown

- **Global Variables** :
  - 3 global variables are initialized: *ledPin*, *inPin*, and *val*. Note that these are all type *int*
  - inPin is given a value of 7, meaning that digital pin 7 is the pin where the pushbutton will be attached.
- **void setup()** :
  - **pinMode()** is used to set the states of each pin. Note that the ledPin is set to OUTPUT and the inPin is set to INPUT.
- **void loop()** :
  - the *val* variable is now used to store the value from the button. Remember that *inPin* is the pin that the pushbutton is connected to and that **digitalRead()** is used to grab the value.
  - **digitalRead(inPin)** will return either a HIGH (on) or LOW (off) value based on the state of the button.
  - **digitalWrite(ledPin)** then sends the output message to the LED based on the button's state.

The Push Button

Now that you've gotten a glimpse at the code for a basic pushbutton project, let's examine the pushbutton itself.

The concept behind a pushbutton is that when the button is pushed, it connects two points on a circuit. In the above example, when the button is pressed down it completes the circuit resulting in the LED turning on.

When the button is released, the circuit is no longer complete and the LED turns off.

Essentially, the button connects to the **5V (POWER)** when the button is pressed. This results in a HIGH reading, which then turns ON the LED.

Once the button is released, **5V** is essentially disconnected, which then turns OFF the LED.

Activity: Button Practice

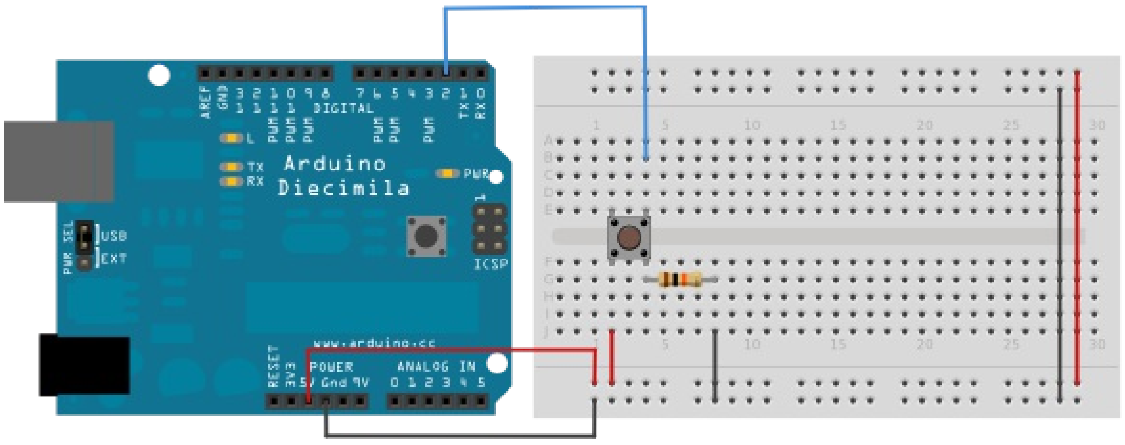
It's time to level up your LED skills! Instead of the state of the LED being solely determined by your code (blinking on and off in a loop), you're going to attach a button so that you can control whether the LED is on or off.

Supplies:

- Uno board and breadboard
- Jumper wires (at least 4)
- Pushbutton
- LED
- 10K Ohm resistor (for the button)
- 220 or 560 Ohm resistor (for the LED)

Steps (Board):

Refer to this *Fritzing Diagram* from Arduino Reference: Buttons page:



1. Once your supplies are gathered, connect the breadboard to the

Arduino:
    -
        - rail on the breadboard connects to GND on Uno
    -
        - rail on the breadboard connects to 5V on Uno
    - Connect both sides of the breadboard: - to - and + to +
2. Connect the pushbutton so that it straddles the middle notch in the breadboard.
3. Connect the top leg of the button to the + rail.
4. Connect the bottom leg of the button to the resistor.
5. Connect the other leg of the resistor to the - rail on the breadboard.
6. Connect the opposite leg of the button/resistor to PIN 2 (or whatever pin you set) of the Uno.
7. Connect the LED to PIN 13 (or whatever pin you set) using the resistor, as you did in previous projects.

Steps (Code):

1. Navigate to this link and copy the code:
    - CMD + C to copy
2. Make a new sketch and paste the *entire* code into the sketch
3. Make changes to the code, but make sure to pay attention to the *code comments* so that you know what you can and can't change.
    - If you change the pin placement of the LED or the pushbutton, make sure that your code matches.
4. Upload your code to the board and then test your button!

Beyond the Basics: digitalRead()

You've now practiced the basics of *digital input* and *digital output*, and you've honed your **digitalWrite()** skills. But you haven't dived into

digitalRead() yet!

digitalRead() can be used to *read* the state of a digital pin. This means that you can use this to *check* whether or not a certain state is HIGH or LOW.

Here is some basic code (from the same *Arduino Reference* page) outlining the process:

Note that these are snippets from the entire code.

```
int buttonState = 0;         // variable for reading the pus

...

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

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

In this example, the variable *buttonState* is declared globally. This is going to have either a 1 or a 0 as a value. It's initialized as a 0.

In the **void loop()** code, there is a conditional statement being used to check the state of the button. This is done with **digitalRead(buttonPin)** which checks to see if the button is currently reading HIGH or LOW. If it's HIGH, then the LED should be ON, and if the button state is LOW, then the LED should be OFF.

Resources

- Arduino Reference Documentation : This has more details about buttons in Arduino, and is where the example code in this lesson are sourced from.