Digital Input

Digital inputs are useful if you want to determine when a button has been pressed (i.e., a bump sensor),whether a switch is on or off, or if you want to read a pulse from a sensor to determine its hidden value.To determine whether an input is HIGH or LOW, you use the digitalRead(pin) command. Sometimes a digital input signal might not always have a full 5v available, so the threshold to drive an input pin HIGH is around 3v, and anything below this threshold is considered to be LOW. R/C receivers used for hobby airplanes/boats/cars output "servo signals," which are pulses of electricity that are driven HIGH for a short but specific length of time before going back to LOW. The duration of the pulse specifies the position of the R/C transmitter control sticks. If you try to check this type of signal with your voltage-meter, you won't see the needle move. That's because the pulse is too short to register on the meter, but any digital input on the Arduino can read a pulse length like a servo signal using the pulseIn() command. We can read information from a digital input, not only by whether it is HIGH or LOW, but by *how long* it is HIGH or LOW. The Arduino is good at precisely measuring the length of short electrical pulses, down to about 10 microseconds! This means that quite a bit of information can be encoded into a digital input in the form of a pulse or Serial command.

Circuitry

We will now understand the basic working of taking input through pins with the help of 'push buttons'. Get hold of one (these are pretty cheap) and connect it like the following circuit.

Note the use of a pull-down resistor (150 ohms) in conjunction with the push button. To understand how pull-down and pull-up resistors work, refer to (this)2 (http://francisshanahan.com/index.php/2009/what-are-pull-up-and-pull-down-resistors/).

Code

Now enter the code: Plug in the Arduino, click Verify, then Upload if no errors are found. The breadboard LED should be turned off by default and should glow as long as the button is depressed.

Understanding the Code

1. Most of the code should be easy to gure out once you have understood the previous examples
2. Note the pins from which output has to be gathered are set to the INPUT pinMode in the setup block.
3. digitalRead(pin) This** function returns a boolean value - HIGH if the pin is high and LOW if the pin is low**.

Internal Pull-Up Resistors

Conveniently, the Arduino contains pull-up resistors that are connected to the pins (the analog pins have pull-up resistors also). These have a value of 20K ohms and need to be activated within software to use them. To activate an internal pull-up resistor on a pin, you rst need to change the pinMode of the pin to an INPUT and then write a HIGH to that pin using a digitalWrite command. Make sure that your button is connected across the pin and the ground in this case:

```
pinMode(pin, INPUT);

digitalWrite(pin, HIGH);
```

Note that now pressing the button will cause the state variable to read LOW and otherwise HIGH by default. The wire connecting the push button to +5V is no longer needed. The circuit should be like this:

Digital Outputs

A digital output is equally simple, yet can be used to do complicated tasks. If you have an Arduino, you have seen the Hello World! sketch, which simply blinks the LED on pin D13 that is built in to the board—this is the most simple use of a digital output. Each pin on the Arduino is capable of supplying or sourcing about 40mA of current at 5v. Often the current supplied by an Arduino pin is not sufficient to power anything more than an LED, so a level-shifter or amplifier can be used to increase the voltage and current that is switched ON and OFF by the Arduino to a more usable level for controlling motors, lights, or relays. Digital pins are also the basis for serial data transfer, which can send multiple commands through a single digital output.

Circuitry

Now we'll dive into some hardware and code for a primer on the working of the Arduino. Get a** breadboard, LED, few resistors, and wires** ready. Unplug the Arduino and connect everything as shown:

Make sure the LED is connected correctly with the** longer leg (positive terminal) to Digital Pin 10, and the **shorter leg to Ground. Use elementary Ohm's Law and the rating of the LED to calculate the rating of resistor to use. A 100 ohms one should sufce here though.

Code

Now plug in the Arduino and enter the code:

Click Verify, then Upload if no errors are found. The breadboard LED should now ash on and off every second.

Understanding the Code

1. // Project 1 - LED Blinky This is a comment. Comments start with a double slash, and are ignored by the compiler. They are essential to understand how the code works, especially if you are distributing your code
2. int ledPin = 10; This line denes a **variable **called ledPin of type integer. A variable is a place to store data. The data can be of type integer (int), oating point (oat), character (char) etc.
3. void setup() This is a function. A function is a bunch of code assembled into one convenient block. Variables are passed to the function with parentheses. The function code lies within curly braces. An Arduino sketch must have a setup() and a loop() function, otherwise it will not work. The** setup() function runs once at the start of the program** and issues instructions like setting pin modes to input or output (e.g Pin 10 with the LED is set as output), setting serial baud rates etc
4. void loop()The** loop() function is the main program function and runs continuously as long as the Arduino is on** (after the setup() function though). Every statement within the loop()function (within the curly braces) is carried out, one by one, step by step, until the bottom of the function is reached, then the loop starts again at the top of the function
5. digitalWrite(ledPin, HIGH); digitalWrite** writes a HIGH or a LOW value to the pin within the statement** (in this case ledPin, which is Digital Pin 10). When you set a pin to HIGH, you are sending out 5 volts to that pin. When you set it to LOW, the pin becomes 0 volts, or ground. This statement, therefore, sends out 5v to pin 10 and** turns the LED on**.
6. delay(1000); tells the Arduino to wait for 1000 milliseconds, i.e. 1 second.

Setting up a digital input and output in the same sketch

```
// Code Example: Input and Output
// This code will set up a digital input on Arduino pin 2 and a digital output on
Arduino pin 13.
// If the input is HIGH the output LED will be LOW
int switch_pin = 2; // this tells the Arduino that we want to name digital
pin 2 "switch_pin"
int switch_value; // we need a variable to store the value of switch_pin, so we make "switch_value"
int my_led = 13; // tell Arduino to name digital pin 13 = "my_led"
void setup()
{
pinMode(switch_pin, INPUT); // let Arduino know to use switch_pin (pin 2) as an Input
pinMode(my_led, OUTPUT); // let Arduino know to use my_led (pin 13) as an Output
}
void loop(){
switch_value = digitalRead(switch_pin); // read switch_pin and record the value to switch_value
if (switch_value == HIGH){ // if that value "is equal to (==)" HIGH...
digitalWrite(my_led, LOW); // ... then turn the LED off
}
else { // otherwise...
digitalWrite(my_led, HIGH); // ...turn the LED on.
}
}
// end code
```