Digital Inputs, if and while branches / loops

Objectives

- Understand how digital inputs are configured on the Arduino
- Understand how to wire digital inputs
- Understand the importance of pulldown / pullup resistors
- Configuring internal pullups
- Understand math operations and looping structures

Part 1: Digital Inputs

- A digital input refers to a single value that can be read as a zero (low or off) or a one (high or on) from a pin.

- Digital inputs are configured in a similar method to digital outputs in the setup function:

```
const int pushButton = 6;
void setup()
{
  pinMode (pushButton, INPUT); // setup pin D6 as an input
}
```

Reading the switch

Once the input has been configured, it can be read in a program using the **digitalRead** function:

```
void loop()
{
  inputVal = digitalRead(startSwitch);
}
```

Putting it all together

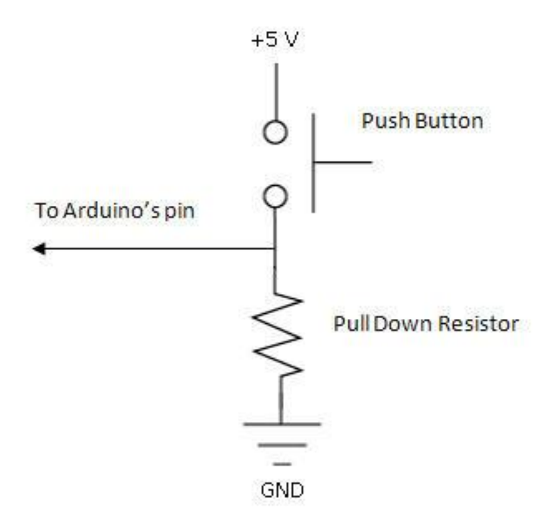
```
int startSwitch = 6;        // define pin 6 to be an input called “start”
int inputValue;
void setup()
{
   pinMode (startSwitch, INPUT);  // initialize digital pin 6 as an input
}

void loop()
{
   inputValue = digitalRead(startSwitch);
}
```

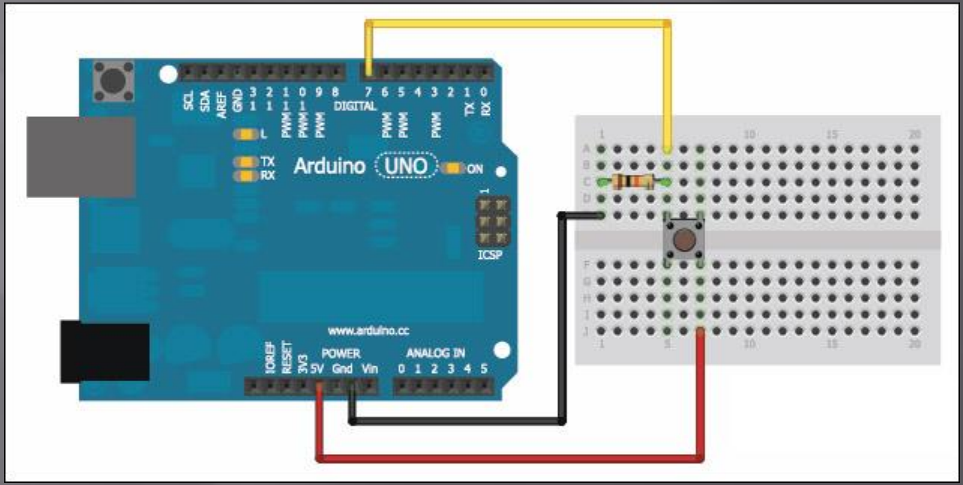
Now, let’s combine inputs AND outputs…

```
int startSwitch = 6;          // define Input Switch “start” as pin 6
int LED = 13;                 // define LED as 13
boolean switchVal;            // define a variable as type boolean

void setup()
{
   pinMode (startSwitch, INPUT);     // initialize digital pin 6 as an input
   pinMode (LED, OUTPUT);            // initialize digital pin 13 as output
}

void loop()
{
   switchVal = digitalRead(startSwitch);   // read the switch (0 or 1)
   digitalWrite (LED, switchVal);          // copy this value to the LED
}
```

Notice – since switchVal is type boolean, it is read as either HIGH or LOW

How do we wire switches?

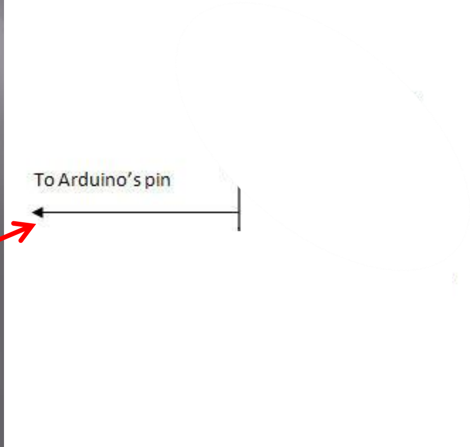




What's up with the resistor?

- A resistor connected between the input pin and ground is called a "Pulldown" resistor.
- Pulldown resistors are used to establish a base voltage if the switch is off.

If the resistor is not attached and the switch is open (not pressed), what is the voltage at the input pin???



Without any connection, the pin is "floating" and could be read as either a high or a low value. By connecting the resistor, we establish a current to ground if the switch is open, in this case a "low."

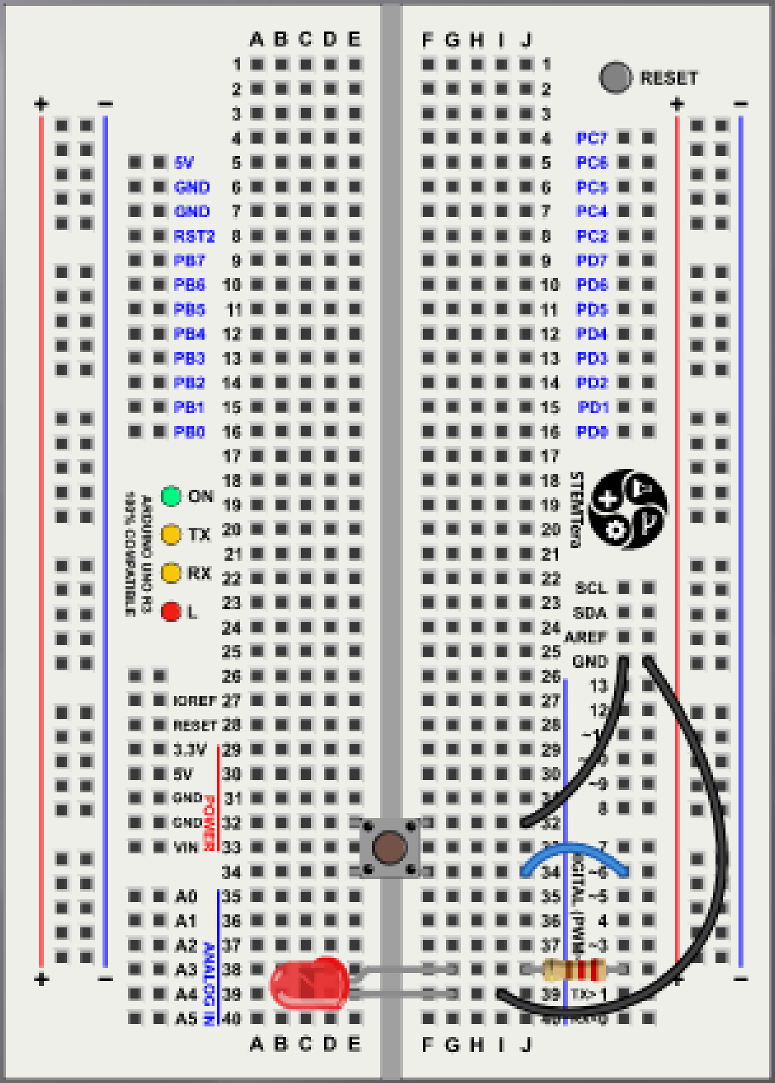
Another way…

- If we don’t want to use a resistor, we can use an *internal* pullup.
- This is done when defining the input:

```
pinMode (6, INPUT_PULLUP);
```

- This will connect an internal pull-up resistor to the input.
- However, in this case, the button will go LOW when pressed.

Wiring a switch



Part 2: Basic math operations and branching structures

The C programming language uses some basic math symbols, with a couple of exceptions:

+ *Addition*	value = x + y;
- *Subtraction*	value = y – x;
/ *Division*	value = y/x;
***** *Multiplication*	value = x * y;
++ *Add one to a value*	value++ (also called *increment*)
-- *Subtract one from value*	value-- (also called *decrement*)

New topic: Basic math and branching structures

The C programming language uses other symbols for comparing values:

==	*Equality*	value == 100;	** Note!! Two equals signs!
!=	*Inequality*	value != 100;	
<	*Less than*	value < 100;	
>	*Greater than*	value > 100;	
<=	*Less or equal*	value <= 100;	
>=	*Greater or equal*	value >= 100;	

Branching & Looping Structures

In order to process a change based on an input condition, we can use the if and while commands:

NOTE: no ";" after if condition!!

```
count++; // increment variable count
if (count > 100)     // if this condition is true, run the next line
  digitalWrite (BlueLED, HIGH);
digitalWrite (BlueLED, LOW);  // Otherwise, run this line of code
```

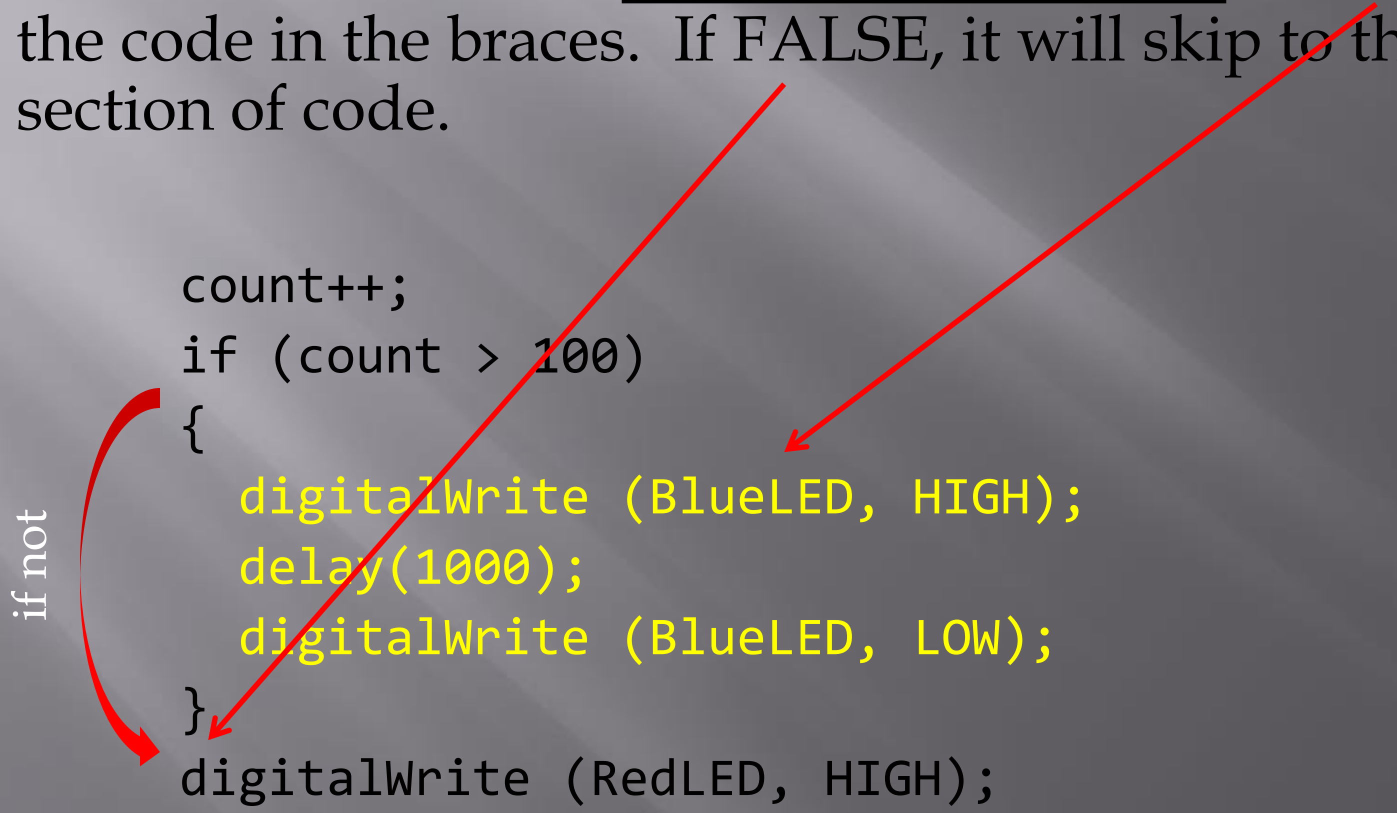
If there are multiple lines of code that need to be run, they can be placed within curly braces:

```
count++;
if (count > 100)
{
  digitalWrite (BlueLED, HIGH);
  delay(1000);
  digitalWrite (BlueLED, LOW);
}
```

Branching Statements

IMPORTANT:

The if command will test for the condition – if TRUE, it runs the code in the braces. If FALSE, it will skip to the next section of code.



```
count++;
if (count > 100)
{
  digitalWrite (BlueLED, HIGH);
  delay(1000);
  digitalWrite (BlueLED, LOW);
}
digitalWrite (RedLED, HIGH);
```

Branching Structures

if - else

if can be combined with *else* command to perform some alternate code if false. If FALSE, it will skip to the code in the *else* section of code.

```
count++;
if (count >= 100)
{
  digitalWrite (BlueLED, HIGH);
}
else   // count < 100
{
  digitalWrite (BlueLED, LOW);
}
```

Looping Statements: *while*

The if command performs *branching* – it will execute the code one time if true, or skip to the next section of code if false.

In some cases, we want to have the program wait until a condition has changed before moving on. To do this we use the *while* command:

```
while (input == TRUE)
{
  digitalWrite (BlueLED, HIGH);
  input = digitalRead (pushButton);
}
digitalWrite (BlueLED, LOW);
```

While : CAUTION

Be careful when using while!

- If the condition is not read in the loop, or never changes, the program will “hang” at that line of code.
- The program hasn’t stopped; it is just waiting for an input.
- Next week we’ll look at another way to do this…