



Exported for Brian O'Connell on Tue, 27 May 2025 19:29:13 GMT

P3L: Digital Signals & Introductory Programming Concepts

The following is the pre-lab lesson content for Programming Lab 3. It will cover the following:

- Digital Signals
- Buttons
- if statements
- State Variables
- millis() function

P2 Review

The previous lab was about getting introduced to Arduino/C++, creating your first few simple circuits, learning more about some of the basic concepts, and exploring the possibilities to look forward to. It covered the following topics:

- Software
 - The Arduino IDE
 - Arduino Language
 - As a variation of C++
 - Arduino Sketches
 - Basic Variables, Functions, and Structures
 - Arduino specific basic commands
- Hardware
 - The SparkFun Redboard
 - Solderless Breadboard
 - Basic electrical concepts
 - Jump Wires
 - Resistors
 - LEDs



Question 1

[Show Correct Answer](#)[Show Responses](#)

What modifier used when declaring a data type ensures that the value can not be changed later in the script?



Question 2

[Show Correct Answer](#)[Show Responses](#)

There are two required functions in any Arduino program. They are, in order of their standard

appearance in an Arduino script,

and

.

Microcontroller/Circuit Concepts

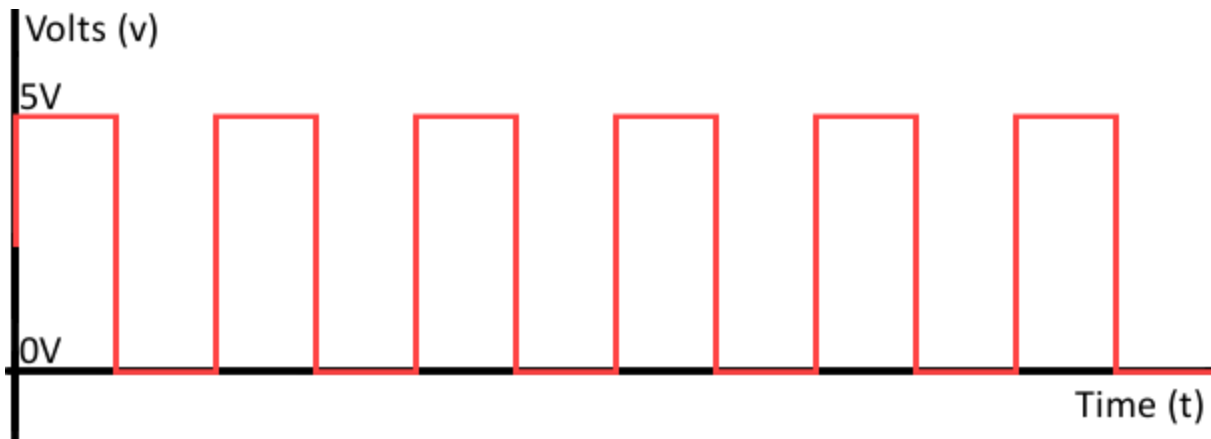
This week we're going to get more into the specific concepts that drive the functionality of the microcontroller. Physically, this means the different types of signals it handles/utilizes but we will also get into an organizational paradigm for when working with microcontrollers.

We'll start with the electrical concepts we will be utilizing to achieve our activity goals in this lab.

Signals

Earlier, we mentioned digital and analog signals. These are the primary signals used in circuit design.

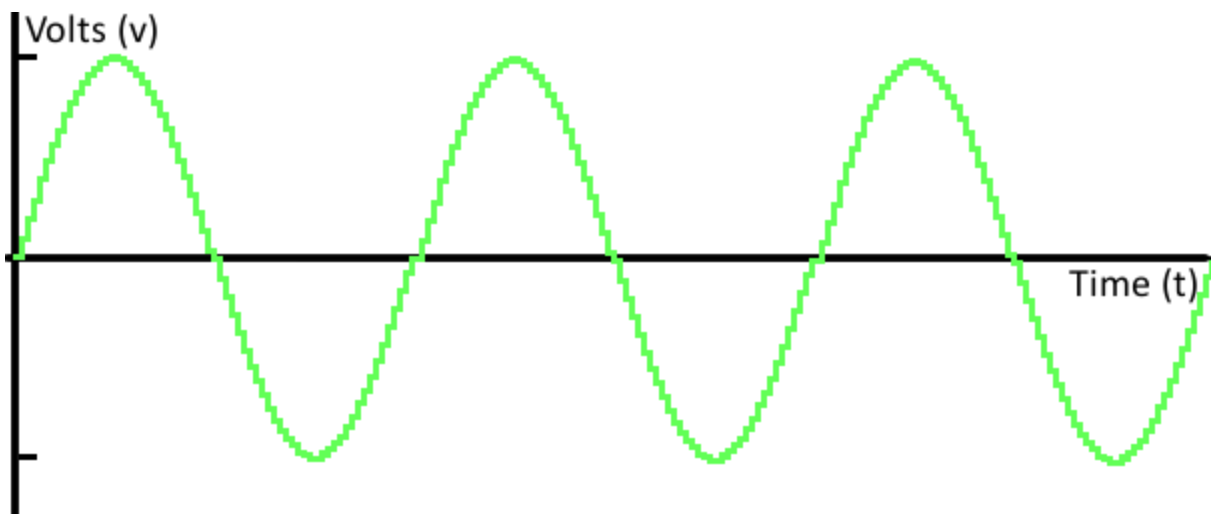
- **Digital**
 - A Digital signal has a finite set of possible values
 - Either **0V (LOW)** or **5V (HIGH)**



Digital signal can switch only between their HIGH and LOW setting

- **Analog**

- Analog signal can be tuned to a value between 0 and 5V
 - These signals are not perfectly smooth but a collection of discrete steps
 - We will get into this in more detail in the next lesson



Analog signals can be a value between 0 and 5V

We will only cover digital signals this week but for a more detailed and nuanced explanation of the difference, you can [see Sparkfun's tutorial](#). Just knowing the general difference though should be sufficient of our needs. In the next lab, we will also get into the details about analog signals.

I/O

I/O is a common term in circuits with microcontrollers. It means Input/Output, referring to a pin's capability to take in a signal or send out a signal. An example of an output would be lighting an LED. In that case, the pin connected to an LED would light up the LED every time it is set to HIGH, ie 5V are being sent out along that branch of the circuit. Last week, we just used them as outputs, but this week we'll work with the input capability.

Question 3

What do the terms 'HIGH' and 'LOW' represent in the context of a Digital signal in microcontroller circuit design?

- A HIGH refers to 0V and LOW refers to 5V.
- B HIGH refers to 5V and LOW refers to 0V.
- C HIGH refers to input signals and LOW refers to output signals.
- D HIGH refers to peak signal frequency and LOW refers to the least signal frequency.

 HINT

 EXPLANATION

 Show Responses

 Show Answer

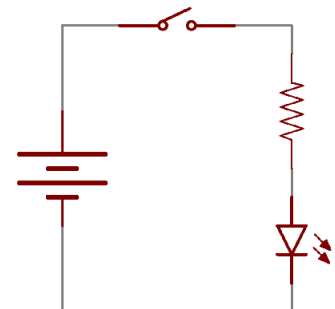
New Components

Buttons

Buttons are another common component. The tactile buttons included in your kit are SPST momentary [switches](#).

When you engage the button, it completes the circuit and allows current to flow. We can use these components to control a voltage source's connection to

one of the RedBoard pins. That way, we can control the digital signal for that input pin with our touch. When you connect the input pin to a ground, it reads as **LOW**, and when you connect it to the voltage source, it reads as **HIGH**.

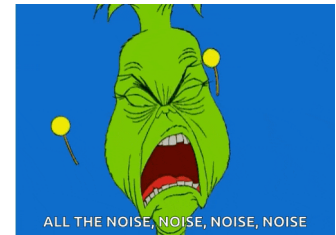


There are some things to keep in mind with buttons or any input for that matter. All sensors have to deal with some form and level of signal noise.

Signal noise

Noise is any unwanted disturbance in the signal. Noise in your system can be quite problematic, and annoying, to deal with.

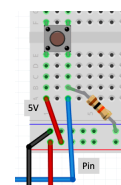
Some components have some inherent signal noise concerns; buttons are one of these. Built up static electricity in the environment can be enough to momentarily change the signal applied to a pin. It could be as simple an issue as occasionally blinking a light when you don't want it to or setting off something more drastic or even dangerous. We don't want our circuits to ever read a signal that isn't there.



For buttons, we can use [pull-up/down resistors](#) for this. You can cause a very weak pull to either a **HIGH** or **LOW** signal by connecting the input pin to either the **voltage source** or the **ground** using a large resistor. If you're looking for a **HIGH** signal, you want the default to be pulled **LOW**. If you're looking for a **LOW** signal, you want its default state to be pulled **HIGH**. A pull-up resistor allows for some current draw along that path although a very weak amount. When you engage the button, connecting the pin to the other signal, that is the path of least resistance and the current flow that way, intentionally changing the voltage across the input pin. Think of this like a spring on a door. It helps keep it shut when you want it shut but doesn't take much force to overcome its pull and allows you open the door when you push on it.

Northeastern University

Pulldown Resistors

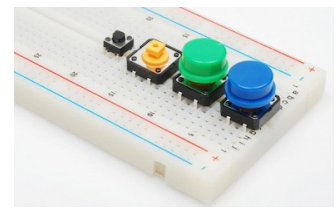


- Connects the signal pin to the opposite signal source

We will discuss and demo this phenomenon in more detail in class. To see this in action though, when you make a button circuit, before adding the pull-up/down resistor, try waving your hand over the running RedBoard and see what effect that static has.

Breadboard fit

This issue is not with every button but is with the tactile buttons you have. They tend to be difficult to insert into the bread board. You may need to



bend the pins using the pliers to help them fit properly. Be sure to press them in with enough force so that they fully engage with the breadboard's internal clips.

Troubleshooting

Like anything, Arduino is going to take time and experience to get used to and be able to easily debug. Arduino has provided their own [troubleshooting FAQ](#) but the following may also help. They may not make much sense now but will once you get into the lesson it may.

Software issues:

- Keep your eye on the debug window
 - It is not as straight forward as MATLAB's as it's far more verbose and jargon filled but you can isolate many issues and their location reading the debug window output
- Check your Setup
 - Have you setup your Values correctly?
- Check your Loop
 - Are you reading all the inputs? Repeatedly and at the right times?
 - Are you sending out the information you think you are?
- Flip and reverse it
 - Try scanning through your code from right to left and bottom to top. It sometimes reveals things you missed going through it the normal way.

Hardware issues:

Troubleshooting hardware is not a completely different issue than troubleshooting code. There are common issues and techniques to keep in mind. For instance, examining things in reverse can be helpful. Going through the steps you took in reverse can help you realize something was skipped or not connected properly.

The following troubleshooting questions will make more sense once you've had some experience setting up hardware. These are the first questions to go through whenever you have a circuit issue.

- Power
 - Are all components connected to power?
- Grounded
 - Are all components connected to ground?
- Resistors
 - Do you have properly sized resistors?
- Holes
 - Are all components fully engaged in the breadboard holes?
 - This is a notorious problem with the buttons
- Polarization
 - Are all components arranged in the right direction?
 - This is a notorious issue with LEDs

A good order of questions would be:

- Is it turned on?
- Is the code doing what I think it should be doing?
- Is everything connected properly?
- Who might be more familiar with this issue?



Question 4

[Show Correct Answer](#)[Show Responses](#)

Let's assume your LED connected to pin 13 isn't working as expected and just isn't turning on. What's the order of actions you should take to debug this issue?

A

Ask for help

B

Check if all the other components and wires are properly plugged into the RedBoard

C

Check if the built-in LED connected to pin 13 on the RedBoard is blinking

D

Check if the LED's polarity is correct

Arduino Code/Commands

This week, we'll be building off of the example **Button**. Go to **File / Examples / 01.Digital / Button** and open it. We'll do a general breakdown for now so you can recognize the new commands and then we'll build a circuit to illustrate their functionality.

What's Old?

You should already be familiar with the 2 required functions in Arduino, **setup()** and **void()**. There are elements you should recognize as well:

```
const int buttonPin = 2;
const int ledPin = 13;

int buttonState = 0;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

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);
  }
}
```

- Global Variables
 - Variables established for use anywhere in the script
- **Const** = Constant Value
 - Used in conjunction with establishing the data type to lock that variable's value so it will not change
- **digitalWrite(pin, value)**
 - Sets **pin** to a digital signal **value**, **HIGH** (5V) or **LOW** (0V).
- **delay(number)**
 - Delays for a set number of milliseconds

What's familiar?

There are also familiar elements but used differently:

- `pinMode(pin, direction)`
 - Sets a pin as an `output` or as an `input` based on how it is going to handle signals

In the last lab, we just used this as an output. For this lab, we will be using both since we have both an actuator, something that takes an electrical signal and turns it into a physical reaction, and a sensor, something that turns a physical input into an electrical signal. Our LED is the actuator since it takes the output signal from a pin and turns it into light, hence it is set to `output`. Our button will be the sensor since it completes a circuit, applying a readable signal to a pin, hence it is set to `input`.

What's new?

The only new Arduino specific command would be `digitalRead(pin)`. We can infer what `digitalRead(pin)` does based on our familiarity with `digitalWrite(pin, value)` does. It is at this point in the script where the code will check the signal at the pin indicated, in this case the pin connected to the button. It outputs a value based on that reading which we can store in a variable. In this example, the integer `buttonState` was created for that purpose and, with each run of `digitalRead()`, we assign the output value to that variable.

- `PinState = digitalRead(pin);`
 - Reads a digital signal value from a set pin
 - Returns a `1` or `0` corresponding to a `HIGH` (5V) or `LOW` (0V) input

The last new concept would be the implementation of an `IF Statements`. Here we can see how that is done in C++ syntax. The general syntax for an `IF-ELSE Statement` is

```
1. if ( condition1 )
2.     { Code A; }
3. else
4.     { Code B; }
```

In our pseudocode, we have been using:

```
1. IF condition1 THEN
2.     Code A;
3. ELSE
4.     Code B;
5. ENDIF
```

Note the differences, specifically the use of `()` to contain the condition and `{ }` to contain the code. Those are required elements of C++ syntax. Not properly bracketing your code and forgetting `;` at the end of lines are 2 of the most common issues. We will go over some debugging tricks in class to help avoid that. The commands are also case sensitive so be sure to use lower-case in C++/Arduino scripts.



Question 5

[Show Correct Answer](#)[Show Responses](#)

What is the command to configure a pin (identified by the variable “MagPin”) to accept a signal from a magnetic door sensor? A magnetic door sensor is a type of digital sensor.

P3L1: Button Switch

We'll illustrate as we work on the lesson activity. You should already have the example **Button** open. Save a copy as your script for this assignment, P3L1. Don't forget to update the title block as well to include the assignment, your name and date, name of the script, appropriate credit when it is due, and a description of the program. Some of that you'll update later as you learn what we're going to do.

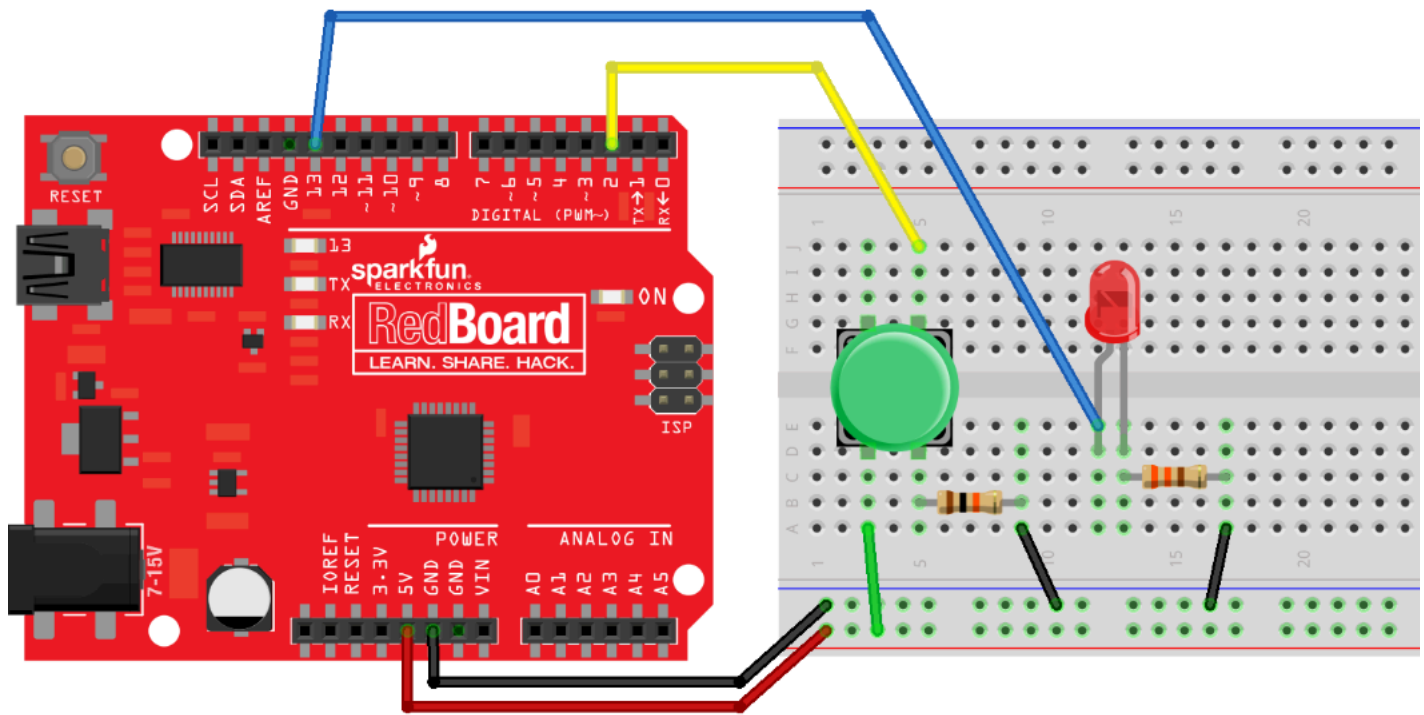
Create your Circuit

For this assignment, we're just going to need a button and a switch. We'll start with the basic example for a button, described in the Arduino example. Do not hesitate to reach out for help from Prof. O'Connell or utilize the Red Vests

SAFETY CONCERN!!

Don't forget to disconnect any power source before you begin working on a circuit. We're working at low voltages but it's a good habit to unplug from your power source before working directly with exposed wires. At these voltages, is more a concern for your hardware but that includes your laptop so be safe and careful.

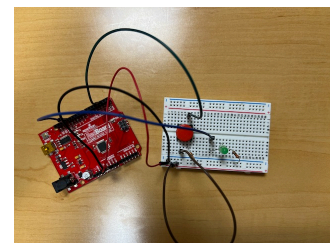
Create the following circuit:



You do not have to have your components in the exact rows shown. You just need to complete the electrical connections shown in this wiring diagram. Here's a description of the connections:

- **LED**
 - Connect **Pin 13** to the **Anode** pin of an **LED**
 - Connect the **Cathode** pin of the **LED** to one side of a **330 Ohm resistor**
 - Connect the other side of **330 Ohm resistor** to **GND**
- **Button**
 - Connect **Pin 2** to one side of a **Button**
 - Connect the other side of the **Button** to **5V**
- **Pull-Down Resistor**
 - Connect a **10 KOhm resistor** to the **Pin 2** and its side of the **Button**
 - Connect the other end of the **10 KOhm resistor** to **GND**

The short wires shown in the wire diagram are just simple representations. You can see in this image that you can use a different color button or LED and even connect the resistors directly to the ground column.



Example of a completed circuit for this assignment

Now plug your board into your computer and upload your script.

Run the Script

- Select the Upload button
 - The right facing arrow
- This begins uploading the code to the board. It will take a minute.
 - If you get errors, make sure to check the following:
 - Is your RedBoard plugged in?
 - Under Tools, have you selected 'Arduino/Genuino Uno'?
 - Under Tools, do you have the correct port selected?
 - If you have multiple port options and aren't sure which, close the drop-down, unplug the RedBoard, check again, close the drop-down, plug in the RedBoard, then select the newest one.
- Once it's uploaded, press the button to see what happens
 - The LED should be on when you press the button and off when the button is not pressed

Understand the Script/Circuit

Let's break down the logic a bit to understand what's happening, when, and why. Here is the example **Button** broken down into the acceptable course pseudocode:

1. **PROGRAM Button:**
2. **Setup Button Pin as an input, use for component Button;**
3. **Setup LED Pin as an output, use for component LED;**
4. **WHILE 1**
5. **Check Button's state;**
6. **IF Button State is HIGH THEN**
7. **Turn LED On;**
8. **ELSE**
9. **Turn LED Off;**

```
const int buttonPin = 2;
const int ledPin = 13;

int buttonState = 0;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

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);
  }
}
```

9. **ENDIF**
10. **ENDWHILE**
11. **END**

The Setup

- Global Variables
 - There are two variables marked as constants
 - This makes sense since they're associated with components and they won't change while the program runs
 - It's also best practice to use descriptive variables rather than rely on just using the pin numbers. It makes clear that you're talking about the LED or the button and if you move a component to a new pin number, you only have to update 1 line in the script.
 - There's a new one that's not constant though
 - Its name suggests it has something to do with the button state, so you can see the value of a descriptive variable name.
- Initialize digital pins in **setup()**
 - We have one **output** for the LED
 - The button pin is setup as an **input**
 - Makes sense since that will take in information; someone pressing the button connecting the pin to a voltage source

The Loop

- **buttonState = digitalRead(button1Pin);**
 - This is when the script actually checks the button, if it's pressed or not. The value of **buttonState** will only change when the line of code runs. If you're pressing the button and then release the button after this line runs, the value of **buttonState** remains **HIGH**. The program doesn't know until it reads it again.



Question 6

Why is it beneficial to use descriptive variable names in a script when setting up hardware components like LEDs and buttons in Arduino?



A

Descriptive names clarify the component's role and simplify script updates.

B	Descriptive names increase the processing speed of the Arduino.
C	Descriptive names allow the Arduino to detect hardware changes automatically.
D	Descriptive names reserve memory space for future variable additions.

 HINT



 EXPLANATION



Show Responses



Show Answer

Let's illustrate this phenomena.

- Add a delay after the digital read of 2.5 seconds
 - To do this, add a new line of code after `buttonState = digitalRead\(button1Pin\)`; with a delay of 2500 milliseconds.
- Upload your code
- Press the button quickly and release it
 - Did it turn on at all? If it did, did it shut off when you released it? Try pressing it a couple of times.
- Press the button and hold until the light turns on
 - How long did you have to hold before it turned on?

Now when you press the button, you have to hold it for at least 2.5 seconds to guarantee a read and the light turning on. You may luck out and becoming up on that 2.5 sec cycle the instant you press it, but there's no guarantee you won't have to wait. The same goes for when you release the button. It may not immediately react. If you press it for less than 2.5 seconds, the program may not run that line of code while you're holding it down and would not turn on the LED since, during that time, the command `buttonState = digitalRead\(button1Pin\)`; did not run.

It's the equivalent of only seeing when your eyes are open. We blink periodically but our eyes are almost constantly open. Imagine trying to navigate the world if it was the opposite, that your eyes are closed the majority of the time and you open your eyes periodically. How hard would it be to move around safely if you could only peep into the visible world for 1 millisecond per stride as you walked around.

A microcontroller will only 'open its eyes' when directly commanded too. It's why we store that `digitalRead()` output to a variable to keep track of the state of the button and why we didn't initially have `delay`. We wanted the `loop()` to repeat as often as possible to get that momentary look at the button's state to happen very often. With the delay, if you push the button for a micro-second and release it before the code gets to that command, then the microcontroller does not know that the button was pushed. So if you try pressing the button while it's in the middle of going through the delay, it won't register that button press because `digitalRead()` wasn't run during that time.

Leave the `delay(2500);` there but just comment it out (Add `//` at the beginning of the line) so it's clear you engaged with this phenomena.

Moving along. We can now see how Arduino handles IF Statements through this example

- Arduino uses C++ syntax and is similar to other languages in readability but keep in mind that it's
 - Case sensitive
 - Sections of code are contained by these brackets: `{ }`
 - The general syntax is

```
if ( condition1 )
{ Code A }
else if ( condition2 )
{ Code B }
else
{ Code C }
```
 - We will go over these in greater detail later, for now we're just making use of examples


In pseudocode:

1. **PROGRAM Button:**
2. **Setup Pin 2 as an input, use for component Button;**
3. **Setup Pin 13 as an output, use for component LED;**
4. **WHILE 1**
5. **Check Button's state;**
6. **IF Button State is HIGH THEN**
7. **Turn LED On;**
- ELSE**

8. Turn LED Off;
9. ENDIF
10. ENDWHILE
11. END

Question 7

According to the passage, which condition best describes when the LED light will turn on?


- 
- A When the Button is pressed and applies a voltage source to the Button Pin.
 - B When the RedBoard detects a voltage source on the Button Pin.
 - C When the button is pressed and at least 2.5 seconds have passed.
 - D When the button is released and as least 2.5 seconds have passed.

 HINT



 EXPLANATION



 Show Responses

 Show Answer

Modify the project

We are now going to modify the script so that it the button acts more like a switch. When you press it, it will change the LED from either OFF to ON or from ON to OFF.

1. PROGRAM P3L1:
2. Setup Pin 2 as an input, use for component Button;
3. Setup Pin 13 as an output, use for component LED;
4. WHILE 1
5. Check Button's state;
6. IF Button State is HIGH THEN
7. Turn LED from OFF to ON or from ON to OFF

8. **ENDIF**
9. **ENDWHILE**
10. **END**

This is a simplified version of the pseudocode we'll be using, so don't make changes yet. We're going to create a 'Button Switch,' a fairly common feature that takes advantage of binary data types and digital signals. It's very useful and beneficial in a lot of microcontroller activities.

State Variables

How this will work is by taking advantage of what's called a state variable.

- **State Variable -**
 - A variable used to keep track of the current "state" or condition of a system or component in a program. It is often used to manage behavior that depends on past actions or inputs, making it critical in implementing logic for decision-making or controlling sequences of actions.

Key Characteristics:

1. **Tracks System State:** It represents whether something is on/off, active/inactive, or in a particular mode. For example, a state variable might indicate whether an LED is currently lit or a button has been pressed.
2. **Persists Across Iterations:** Since Arduino runs its **loop()** function repeatedly, state variables maintain their value between iterations, enabling the program to "remember" past states.
3. **Often Boolean or Integer:** In many cases, state variables are booleans (true/false) or integers (representing a mode or step in a process).
4. **Enables Conditional Logic:** State variables are often used in **if** or **switch** statements to determine what actions should occur based on the current state.

How to use a State Variable

Currently, we're already using one. ButtonState is a variable that keeps track of the current state of the button. We're going to expand on that use by, instead of using a State Variable to track an external state, we're going to use it to set an output.

Using them for a Button Switch

To set the LED state, we use the command `digitalWrite(pin, value);`. This writes a value to the pin. In our current code, it's implemented as `digitalWrite(ledPin, HIGH)` and `digitalWrite(ledPin, LOW)`. There is nothing keeping us from using a **State Variable** as our value instead though.

- Create a State Variable called **ledState**.
 - Give it an initial value of **0**.

So now we can use this to instead of **LOW** or **HIGH**, `digitalWrite(ledPin, ledState)`. **ledPin** will then be written with whatever value **ledState** holds. Our code will no be updated so that the buttonState doesn't directly control when the digitalWrite() command is run but rather the value of ledState. We'll then run digitalWrite() after. Update your code with that to match this updated pseudocode:

```
1. PROGRAM P3L1:
2. Setup Pin 2 as an input, use for component Button;
3. Setup Pin 13 as an output, use for component LED;
4. WHILE 1
5.     Check Button's state;
6.     IF Button State is HIGH THEN
7.         Set ledState = 1;
8.     ELSE
9.         Set ledState = 0;
10.    ENDIF
11.    digitalWrite(ledPin, ledState);
12. ENDWHILE
13. END
```

Now this won't change the overall behavior, we'll get to that, but it does use a state variable to set the output rather than doing it directly. Run your script and see if you've got it working correctly.

So here's where you can make it so it stays on or off after a quick button press and release.

In programming, binaries have opposites. True's opposite is false, 0's opposite is 1, LOW's opposite is HIGH. This also means that the NOT of one equals its opposite. By that logic, I could set a variable equal to not itself to have it switch to its opposite.

So if I wanted **ledState** to switch from 0 to 1 and vice versa, I need to set it equal to **NOT ledState**.

Try replacing:

1. `Set ledState = 1;`
2. `ELSE`
3. `Set ledState = 0;`


with

1. `Set ledState = NOT ledState;`
2. `delay for 25 milliseconds;`

Remember in C++/Arduino, `!` serves as the not symbol. So **NOT X** is **!X**. The delay is just so you have enough time to release the button. We'll fix that too but I want you to see the phenomena now. Run your code and quickly press the button. The LED should switch ON or OFF with each quick button press. If you're having trouble releasing fast enough before it switches again, increase the delay a bit.

Question 8

How does using a state variable enhance the functionality in this case?

- 
- A It toggles the LED its state upon each button press.
 - B It indefinitely holds the LED's initial state regardless of button input.
 - C It replaces HIGH and LOW with something more descriptive.
 - D It takes advantage of binary logic to make circumstantial updates.

 HINT



 EXPLANATION



 Show Responses

 Show Answer

Debounce

The last thing we're going to address is adding a debounce.

- **Debounce**

- The process of filtering out unintended or rapid, noisy fluctuations in a signal, usually caused by physical or mechanical switches or inputs. It ensures that only deliberate, stable signals are registered by a microcontroller, avoiding false or repeated triggers.

To see *why we need this*, hold down the button. The LED should flicker based on your delay in the IF-statement. If you don't release the button fast enough, the code takes another unintentional reading of that button and then acts accordingly. You don't want to have to worry about quick button presses here.

There are lots of different ways to incorporate a Debounce into your system. You can do some with hardware, the pull-down resistor is an example of that, or through software, which we're about to do using a State Variable.

Here's a pseudocode for our final version of this:

1. **PROGRAM P3L1:**
2. **Setup Pin 2 as an input, use for component Button;**
3. **Setup Pin 13 as an output, use for component LED;**
4. **WHILE 1**
5. **Check Button's state;**
6. **IF Button State is HIGH AND Last Button State NOT EQUAL TO Button State THEN**
7. **Set ledState = NOT ledState;**
8. **ENDIF**
9. **digitalWrite(ledPin, ledState);**
10. **Last Button State = Button State;**
11. **ENDWHILE**
12. **END**

This works by adding another element to the if statement to force it to only work when there's a change in button state. If the button's current state is the same as the state it was in the last time the loop ran, then that segment of code will not run. With that in place, it will only react to a change and, with the first operand in place, only when that change is from a button press, not release.

Try implementing this yourself first. Again, this is not meant to break you or anything, but you should work to challenge yourself at times.

- We've added another variable that's just storing a state for later reference
- You're also going to need to add boolean operators to your control structure: [&&](#).

Give it a try or two. You can do this. KEEP IN MIND that there are diminishing returns. You're not meant to be banging your head against the wall figuring it all out on your own. Read along for a more detailed walkthrough along with a video if you need it. If you need help, ask.



Going into the
Debounce, you
should have just
had the button
switch
implemented

```
// Constants won't change. They're used here to set pin numbers:  
const int buttonPin = 2; // the number of the pushbutton pin  
const int ledPin = 13; // the number of the LED pin  
  
// variables will change:  
int buttonState = 0; // variable for reading the pushbutton status  
int ledState = 0; // Variable for tracking/setting the LED state  
  
void setup() {  
  // initialize the LED pin as an output:  
  pinMode(ledPin, OUTPUT);  
  // initialize the pushbutton pin as an input:  
  pinMode(buttonPin, INPUT);  
}  
  
void loop() {  
  // read the state of the pushbutton value:  
  buttonState = digitalRead(buttonPin);  
  // delay(2500) // Delay of 2.5 seconds to illustrate a phenomena  
  
  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:  
  if (buttonState == HIGH) {  
    // Set LED state value  
    ledState = !ledState;  
    delay(25) // Small delay to allow for button release  
  }  
  // Set LED state:  
  digitalWrite(ledPin, ledState);  
}
```

with something like this. You can always open these images in a new tab if you need them to be bigger to read.

Also, I left a typo in there from when I went out of order of taking the screen grab and then ran it, instead of checking it first. It errored right away and I fixed it. Instead of replacing the image though, I thought I'd leave it and include that, like every programmer out there professional or not, I still sometimes forget the **;** after adding a new line of code. We all do it, so just be thankful the error notifications will point you right to it. Also, my delay ended up being bigger than 25 as well.

Ok, let's update to add the debounce, first we need to add a state variable to track the last button state and then actually capture that add the end of every loop. Now run your code just to make sure you don't have any errors. It's not going to do anything functional yet, but it's good practice to save and run between steps.

- Create a state variable, **lastButtonState**, for tracking the state.
- Set **lastButtonState** equal to **buttonState** at the end of **loop()**
 - This saves that information for the next loop

Next we need to use that information. We already have an if-statement that's based on `buttonState == HIGH` but now we also want it to factor in the `lastButtonState` to react to a change.

That means to only react if the last button state is NOT equal to the current button state. To compound conditions, we can use AND logic in how we call it out. In Arduino, the boolean AND operator is `&&`. The double ampersand is important so don't forget it.

- Add the condition `lastButtonState != buttonState` to the if-statement's condition
 - Use `&&` to combine with the current conditional
- Comment out the delay as we no longer need it.
 - Just comment it out so the TA knows you followed through with the whole exercise.

Now you should have something that switches the LED off or on the instant you touch the button, but you don't have to worry about quickly releasing it.

Here's the code for the new if-statement. I've included it super small intentionally to help encourage you to try implementing the AND logic yourself first.

```
// check if the pushbutton is pressed. If it is, the buttonState is HIGH:
if (buttonState == HIGH && lastButtonState != buttonState) {
  // Button and State value
  ledState = !ledState;
  // delay(100); // small delay to allow for button release
}
```

Video

Please visit the textbook on a web or mobile device to view video content.



Question 9

What kind of signals do microcontrollers handle?



A

They only handle digital signals.

B	They only handle analog signals.
C	They handle both digital and analog signals.
D	They do not handle any signals.

 HINT



 EXPLANATION



Show Responses



Show Answer



Question 10

Show Correct Answer

Show Responses

In C++, what do you need to terminate each code statement with?

Submit P3L1

For Arduino assignments you will need the following:

- The Raw Code – The .ino file
 - This is the Arduino file
- A PDF of your code and circuit – The .pdf file
 - This should be a single PDF and contain all your content. It's necessary to make reviewing your work easier for the TAs because .ino files need to be downloaded while .pdf files can be viewed directly in Canvas.
 - It should contain:
 - Your raw code - be careful to maintain code format for easier review
 - Pseudocode or flowchart - as required by the assignment
 - Image(s) of your circuit

- Circuit diagram - as required by the assignment
- The video of your working circuit – The .mp4 file
 - This shows you uploading the script to your board and you describing the run of your circuit to help reassure that this represents your own work
 - Please attach to your assignment as a **media comment**. You can upload it as a regular file but then the TAs have to download it to view it. For their convenience, if you, after you've uploaded the other files and even the movie file if you want to ensure it's there, add a media comment with the video then the TAs can review everything directly in Canvas without having to download anything unless there's a major issue or concern.

NOTE: A previous version of this said to upload the video under the MEDIA TAB, not the "File Upload" tab. This has been changed because of a back end issue on Canvas. Apparently Canvas will now only accept submissions from one of their tabs, not multiple. The above use of a media comment seems to be the best workaround to ensure all files are easily reviewable within canvas.

Film your circuit

You will need to make and submit a video for each of your SparkFun assignments of your circuit running.

With the prevalence of camera phones, I assume this is something you are capable of doing. If that assumption isn't valid in your case, talk to Prof. O'Connell and we will figure out another method.

- Record your circuit functioning with the following included in the video
 - **Show your face or Husky ID** at some point in the video
 - We want to confirm that this is your work
 - **Show you uploading the script to the board**
 - Again, confirming that this is your work
 - Show the circuit functioning
 - Narrate what the circuit is doing and why
 - If it is not functioning as you think it should and are just turning in for partial credit, explain what it is doing and what it is intended to do
- Save the video using the proper naming convention

The video only has to show functionality and the narration just needs to show understanding. That typically only requires one or two runs of the circuit. It should not run more than 1 minute in length.