



Exported for Brian O'Connell on Tue, 27 May 2025 19:30:11 GMT

Programming Lab 4 - Analog Signals

You should currently be able to

- Access the Arduino IDE
- Flash a Program to your RedBoard
- Save your work and recover your work
- Properly set up and submit documents for SparkFun assignments
- Use the Arduino IDE
- Break down a basic Arduino Program
- Utilize Digital Signals
- Use Basic Hardware
 - LEDs
 - Buttons

This assignment will focus on:

- Utilize Analog Signals
- Use Analog signals with Basic Hardware
 - LEDs
 - Buttons
- Use new hardware:
 - Potentiometer

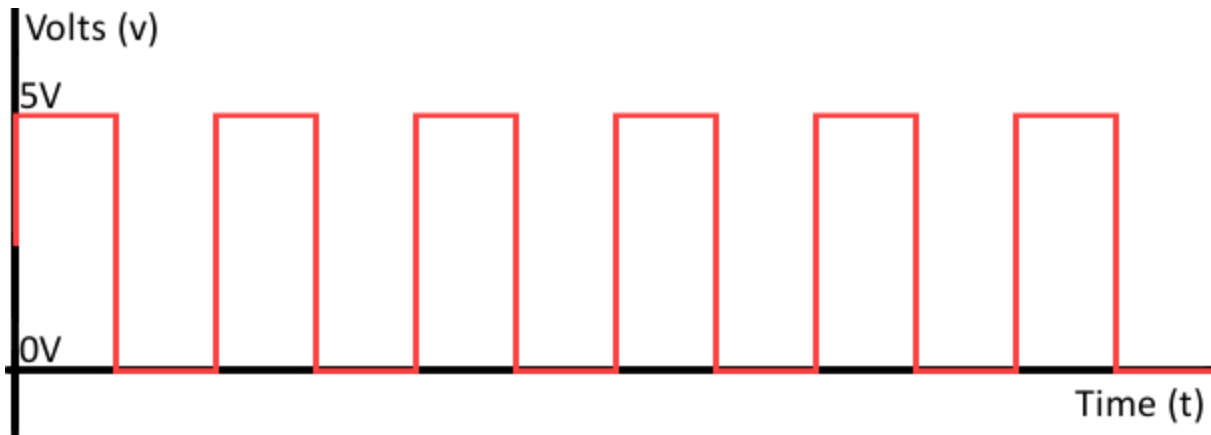
Review

In the last lesson, we focused on Digital signals. In the this lesson, we will make use of analog signals. For more detailed information on the difference, [see Sparkfun's tutorial](#), but you can wait until later to utilize that resource. I just want you to be aware of other resources if any of the below is insufficient. Let's review digital signals as well as some of the basic Arduino elements before diving into analog signals.

Digital Signal Review

We all have turned a light off or on. That is the essential example of a digital signal, either you're sending 100% power or 0% power.

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



Digital signal can switch only between their HIGH and LOW setting

C++ Review

We now have some experience with some basic C++ syntax elements

- [Variables](#)
 - When declared, specify the **Data Type** and then the **variable name**. You can then assign it a value at declaration or wait until later.
- **;**
 - All lines have to end with semi-colons
- **{ }**
 - We need to fence off segments of code using curly brackets
- IFTTT
 - We've worked with **If This Then That** programming logic already, one of the core fundamentals of any programming logic
 - **if (condition1) {**
 do Thing A;
 }
 else if (condition2) {

```
    do Thing B;
}
else {
    do Thing C;
}
```

Arduino Review

There was also some key elements of the Arduino IDE that we should review as well.

- Variables
 - Declared anywhere in the program but before `setup()`, and they're **global**. When declared in `setup()`, `loop()`, or any other function, they are **local** and can only be used in that specific function.
 - `const`
 - If you declare a variable as Constant, then its value can not be changed.
- `void setup()`
 - This is the first of two required functions in any Arduino script.
 - Runs once at power-up or reset to configure your board and initialize variables, pin modes, libraries, etc.
- `void loop()`
 - This is the second of the two required functions in any Arduino script.
 - This is the primary part of your script that runs on a continuous loop.
- `pinMode(pin, direction)`
 - Used during `setup()`
 - Sets a pin as an input or an output based on how it is going to handle signals
 - We're sending out a signal so it makes sense that it's an **output**
- `digitalWrite(pin, signal)`
 - Sets a pin value to HIGH (5V) or LOW (0V)
 - When it writes the pin low, no signal (0 V) goes out
 - When it writes the pin high, the max signal (5 V) goes out
- `digitalRead(pin)`
 - Reads the value of a digital pin as HIGH (5V) or LOW (0V)
 - When connected to a voltage source of 5V, reads **HIGH**
 - When connected to ground, reads **LOW**

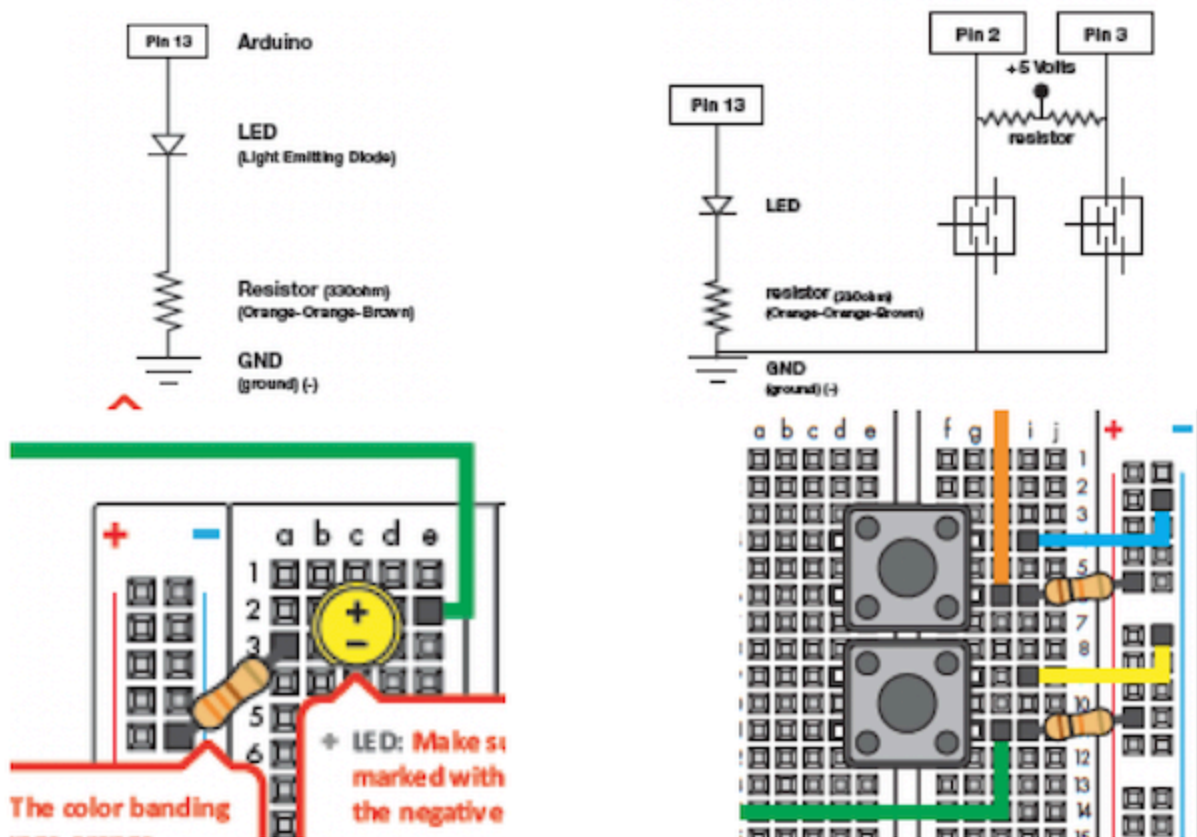
Hardware Review

LEDs

- One of the more common components you'll deal with
- This is a **polarized** component, meaning it matters which direction current flows
 - If it's not on, try reversing it
- Make sure the **ballast resistor** is included and sized correctly to help prevent the LED from burning out

Buttons

- Another common component
- Non-polarized component but typically difficult to engage with the breadboard
- Make use of a **pull-up or pull-down resistor** to help smooth out your signal and avoid false positives.



SAFETY REVIEW!!

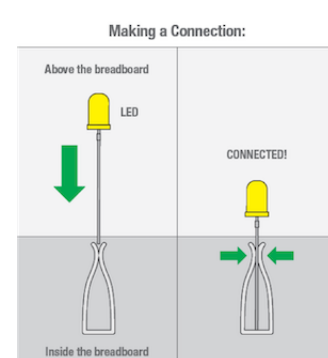
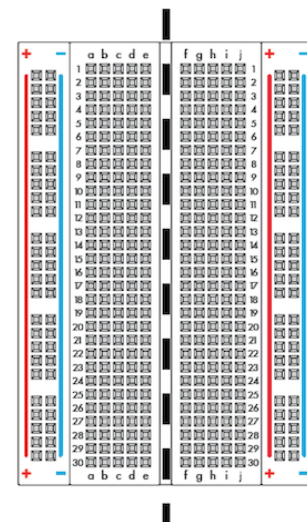
We are going to now begin working with the circuit. There is a safety practice you should get in the habit of first. At the voltages we're working with, it is less about harm to you but rather to your computer,

microcontroller, and the rest of the hardware. This also applies for larger voltages when they can cause you harm.

- WHENEVER YOU ARE BUILDING YOUR CIRCUIT, DISCONNECT THE REDBOARD FROM THE POWER SOURCE.
 - This will be the USB cable throughout this assignment
- WHENEVER YOU ARE MODIFYING YOUR CIRCUIT, DISCONNECT THE REDBOARD FROM THE POWER SOURCE.
 - This will be the USB cable throughout this assignment
- WHENEVER YOU ARE IN DOUBT ABOUT WHETHER YOU SHOULD BE WORKING WITH A POWERED CIRCUIT OR AN UNPOWERED CIRCUIT, DISCONNECT THE REDBOARD FROM THE POWER SOURCE.
 - This will be the USB cable throughout this assignment.
 - It could also be a battery or plug connected to the barrel jack connector.

Solderless Breadboards

- **Center Valley** (Marked with Dashed Line ->)
 - Separates the board into two mirror halves
 - No electrical connections across the valley
- **+ - Power**
 - Runs power along that vertical column
- **-- Ground**
 - Runs ground along that vertical column
- Rows 1-30
 - Two sets of 5 horizontally connected sockets
 - ABCDE are connected
 - FGHIJ are connected
 - They are separated by the Center Valley



The biggest issue with solderless breadboards is not fully engaging the pins with the clip inside the breadboard. Make sure to fully press your hardware and wires in until they're fully engaged.

Troubleshooting Review

Like anything, Arduino is going to take time and experience to get used to and be able to easily debug. They have provided their own [troubleshooting FAQ](#) but the following may also help. They may not make much sense now but will once you get into Part 1 and Part 2 of this assignment.

Software issues:

- Keep your eye on the debug window
 - It is not as straight forward as MATLAB 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 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 find the issue.

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



Share your experience

What's a mistake you made in your circuit or code that you had to fix? Like or comment on ones that other post that you experienced or have a better solution for.



Responses



Reply

Ordered by **Newest Responses** ▼



Justin Ogongo

3 months ago

semicolon

Comments 0 1



Charles Wamester

4 months ago

forgot parentheses or a semicolon

Comments 0 1



Jonah Wolk

4 months ago

forgot a ;

Comments 0 1



Owen Weiss

4 months ago

missed a ;

Comments 0 1



Benjamin Anderson

4 months ago

forgetting ;

Comments 0 2



 Luis Ferreira

4 months ago

forgetting a ;

Comments  0  2



 Juan-Mario Vuksan

4 months ago

i forgot the ;

Comments  0  2



 Nathan Yost

4 months ago

tried to change a variable I accidentally set as const

Comments  0  1



 Jonah Wang

4 months ago

blowing up a resistor

Comments  0  2



 Albert Lin

4 months ago

I mixed up the variable names because they were too arbitrary and then it took forever

Comments  0  1



 John Wolfe

4 months ago

I forget the ;

Comments  0  1



 Jacob Hathcock

4 months ago

Didnt realize my pin numbers were off by one so my RGB values were wrong, spent an hour and a half trying to fix the LED itself instead of the code.

Comments  0  1



 Liam Yin

4 months ago

lots of semi colons

Comments  0  1

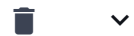


 Foster Ward

4 months ago

comments

Comments  0  1

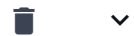


 Korel Unluer

4 months ago

I forgot to add semi colons

Comments  0  1



 Kyra Pallod

4 months ago

Forgetting to add semi colons & brackets

Comments  0  1



 Aditya Chowdhury

4 months ago

forgetting to add semi colons and comments

Comments  0  1



 Harrison Zabar

4 months ago

I usually forget to add the semi colons or brackets

Comments  0  1



 Mark Yang

4 months ago

I forget to add comments

Comments  0  1



 Roman Ton

4 months ago

I forget to do semi colons

Comments  0  1



 Benjamin Anderson

4 months ago

I'm really bad with remembering semi colons

Comments  0  1



 Timothy Wong

4 months ago

Sometimes i will forget how to correctly wire the button and connect it to the write ground / output signals.

Comments  0  1



 Hana Wright

4 months ago

sometimes I'll forget to have the correct orientation of my LED lights and I'll forget to fix it which makes my code not work.

Comments  0  1



 James Wheeler

4 months ago

Getting the correct orientation of wires on the breadboard

Comments  0  3



 Joshua Wolf

4 months ago

I always forget a bracket when coding if statements

Comments  0  2



 Katherine Woodbury

4 months ago

I have made syntax errors like forgetting a semi-colon or a bracket

Comments  0  1



 Dante Varano

4 months ago

I've missed semicolons at the ends of lines or forgot to forward slash before commenting

Comments  0  1



 Zoe Zhao

4 months ago

I have made small mistakes with my circuit such as not properly grounding a component, and usually fix this pretty quickly by reviewing my circuit and making sure everything is in the right place.

Comments  0  2



 David Hamman

4 months ago

I have made the mistake of forgetting small syntax errors with the ; or the parentheses. I have accidentally put the led in backwards and have missed the right pin on stuff too.

Comments  0  2



 Taylor Luks

4 months ago

One mistake I make in my codes is forgetting the semicolon at the end of a line, Arduino does point this out which helps, but it caused an error when I tried to compile. It took me a few minutes but once I added the missing semicolon the code ran correctly.

Comments  0  1



 Bella Wong

4 months ago

A mistake I have previously made is forgetting to add the proper syntax in my code, such as {} and ;. A solution to this problem is being more conscious while writing my code and keeping in mind the proper syntax.

Comments  0  1



 Brady O'Keefe


4 months ago

One of my major issues is that I forget about the polarity of the LEDs, and I put them in the wrong way. Another issue I commonly have is that I forget to put my code in {} when I am typing it.

Comments  0  3



 P3 Review 1

 100% Correct 31/31

What does the pinMode(pin, direction) function do in an Arduino script?

?

- A Sets the voltage value for a pin
- B Sets a pin as input or output based on signal handling
- C Establishes a continuous loop for the script

D

Sets a pin value to HIGH (5V) or LOW (0V)

 HINT EXPLANATION Show Responses Show Answer

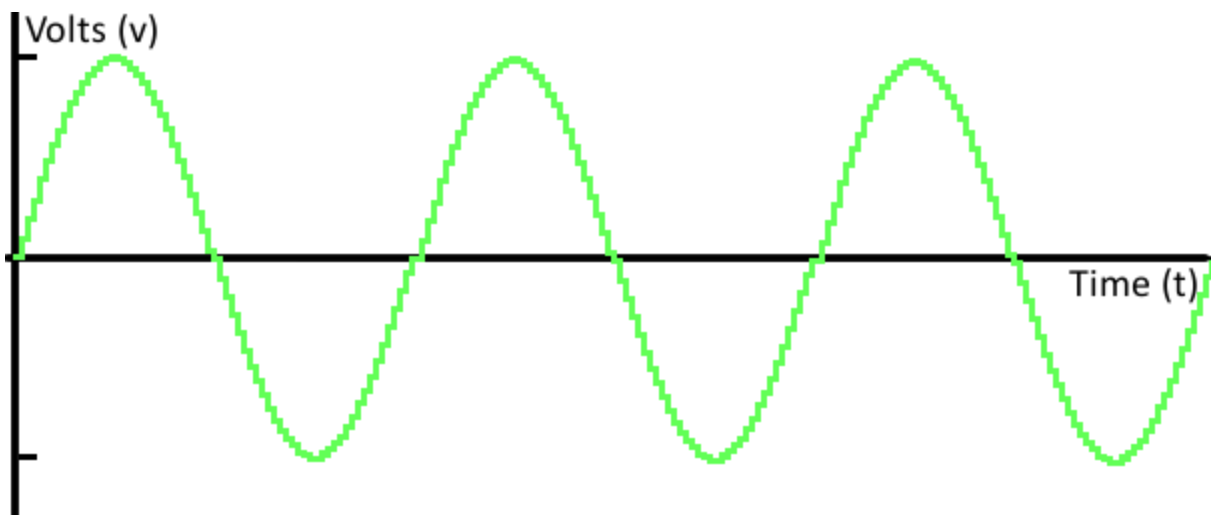
What's New

P4 will focus on Analog signals.

Analog Signals

Where digital signals only handle the extremes of signals, giving you binary control or response, analog signal provides the nuance, letting you read all the values between. It's the difference between only working with integers values and then being handle floating decimal points.

An analog signal can be tuned to a value between 0 and 5V.

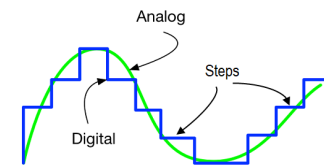


Analog signals can be a value between 0 and 5V

Arduino handles the input and output of Analog signals differently though. Analog inputs are taken by the **Analog In** pins. Analog outputs are handled by the digital pins indicated as PWM using the ~. Why this is done by digital pins is explained. How they work is explained below. The

major difference is that they function at different resolutions, meaning the minimum voltage difference that can be sensed or outputted are different.

When an analog signal is read into Arduino, it is sampled and converted to a digital signal which leads to resolution loss. So, all analog signals will be

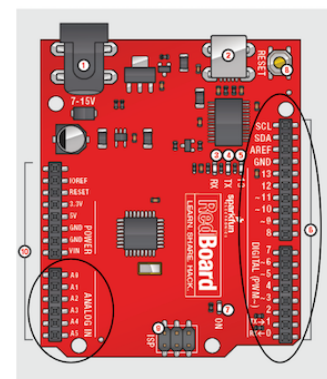


converted to digital by your RedBoard's Analog to Digital converter (ADC). The resulting signal will no longer be perfectly smooth as it will be composed of a series of discrete steps or values (See Figure 6).

Analog Inputs

Arduino boards take **in** Analog signals from the Analog Pins. They are **analog inputs** only.

- Analog Pins (In section 10)
 - **A0, A1, A2, A3, A4, A5**
 - Default as input
- **pinMode(...)**
 - Not necessary for Analog inputs
 - Only for Digital
- **analogRead(pin)**
 - Used to read from analog pins
 - Must call by **A#**, not just the **#**



NOTE: If you are running out of digital pins, you can use an Analog In pin as a digital I/O by setting the pinMode and using the digitalWrite or digitalWrite commands. This is uncommon but available if needed.

- Resolution
 - The signal is not perfectly smooth, it's read in as a series of discrete but fine steps.
 - Arduino's Analog input runs on a 10 bit resolution
 - This means 10 bits are used to define the discrete steps.
 - This allows for readings from 0000000000 to 1111111111 in binary
 - Which translates from 0-1023 in base-10
 - This means you can only recognize a difference of 1/1024 V or 0.00098 V
 - This is more than fine enough for our needs

For an analog output, you must use Pulse Width Modulation(PWM).

Pulse Width Modulation (PWM)

Electronics always pare down to digital. Even for the analog input, there's a built-in digital to analog converter on the board that uses digital processing to provide you with a Arduino, and most other microcontrollers, use a technique called **Pulse Width Modulation(PWM)** to simulate analog signals by a controlled use of digital signals.



PWM Pins

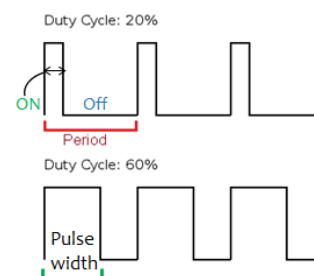
[Show Correct Answer](#)[Show Responses](#)

Using just the information on the board, look at your RedBoard and see if you can determine how many PWM pins are built into an Arduino Uno/RedBoard?

Think about a strobe light. It slows things down by flickering a light off and on at high speeds. You register an image every few milliseconds instead of at your normal frame rate so things look like they've slowed down. That is how PWM works. It sets the signal to HIGH for a few milliseconds and then low for a few milliseconds. The component registers the average voltage. So at 5 V for 2 milliseconds and then at 0 V for 3 milliseconds would result in the component reacting as if it was sent 2 V.

- PWM turns the signal from High to Low quickly.
 - At Microsecond transitions
- Outputs the average value

The duration when the signal is on is called the **pulse width** and the time it takes for a signal to complete an on-and-off cycle is called a **period**. Consequently, the **duty cycle** expressed as a percentage is the ratio of the pulse width by the period, which is the percentage of time the signal is on in a period.



These images help illustrate this phenomena:

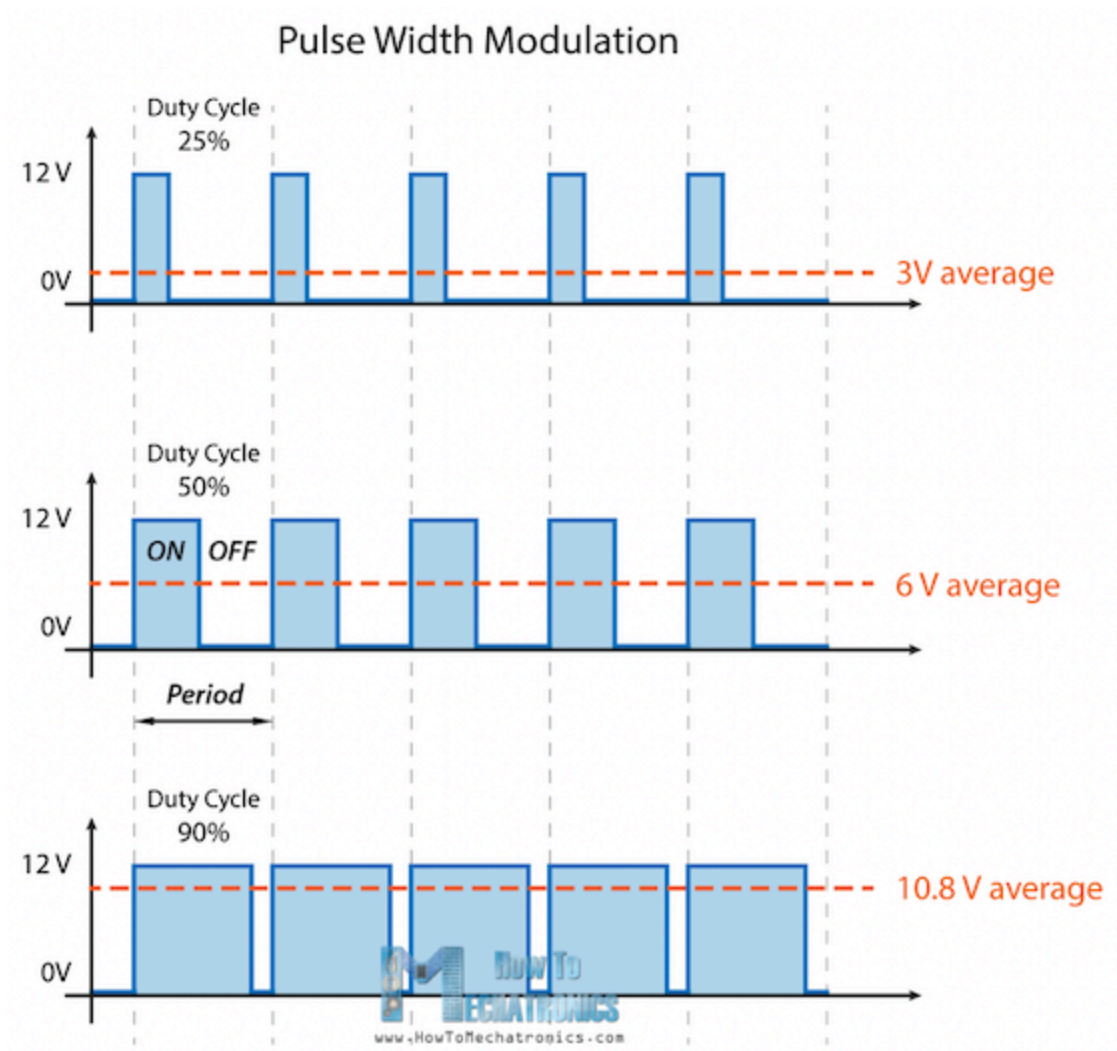


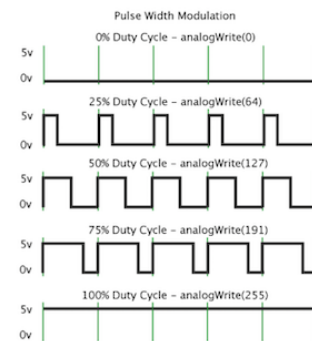
Image courtesy of www.HowToMechatronics.com

Like with inputs, PWM is not capable of a perfectly smooth read. There is a resolution involved. It is only available on a subset of the digital pins.

- Resolution
 - The signal is not perfectly smooth, it's taken at a series of discrete steps.
 - PWM on Arduino Uno's works on an 8 bit Resolution
 - This means 8 bits are used to define the discrete steps.
 - This outputs values of 00000000 to 11111111 in binary
 - Which translates to 0-255 in Base-10
 - This means you can send signals based on discrete steps of $1/255$ V or 0.0039 V
 - Again, more than fine enough for our needs
 - BUT do note the input and output resolutions are different. 2 bits makes a big difference.
- Pins
 - Only available on the pins marked by a tilde (~)
 - Pins **3, 5, 6, 9, 10, & 11**

Here are some examples of how **PWM** would be called in your Arduino script:

- use `analogWrite(pin, value)`
 - Use a value of 0-255
 - Corresponds to 0-100% Duty Cycle
- `analogWrite(pin, 63)`
 - 25% Duty Cycle
 - Quarter Strength signal average
- `analogWrite(pin, 127)`
 - 50% Duty Cycle
 - Half Strength signal average
- `analogWrite(pin, 255)`
 - 100% Duty Cycle
 - Full Strength signal average



Arduino Analog Facts



100% Correct 31/31

Which of the following statements about Analog signals and Arduino boards is NOT true?



- A Analog signals can be a value between 0 and 5V
- B Arduino boards can take in Analog signals from the Analog Pins
- C Arduino's Analog input runs on an 8 bit resolution
- D PWM on Arduino Uno's works on an 8 bit resolution
- E PWM pins are indicated using a ~ marking next to the pin number



HINT



EXPLANATION



Show Responses



Show Answer

Different Resolutions

Because of the different resolutions, you can't just take an analog input signal from something like a potentiometer or slider (think of the speed control for a fan or light in a house) and convert it directly to the analog output to control the brightness of an LED. Since the input range is 0-1023 and the output range is 0-255, once your input is about 25% of the max, the output would be maxed out. The command `map()` was created by Arduino to handle this issue. The map command re-maps a value on one range to the corresponding value on another. This allows for some signal processing to convert a signal that comes in on a range of 0-1023 to the output range of 0-255.

- `map(value, fromLow, fromHigh, toLow, toHigh);`
 - Re-maps a number from one range to another.
 - `value`
 - The number to map
 - `fromLow`
 - The lower bound of the value's current range
 - `fromHigh`
 - The upper bound of the value's current range
 - `toLow`
 - The lower bound of the value's target range
 - `toHigh`
 - The upper bound of the value's target range

For example, given:

- `I = 255; // Input value`
- `O = map(I, 0, 1023, 0, 255); // Convert to an output value`

The value of `O` will become `64`. So even though `I` was initially at the max for an analog output on an Arduino Uno but ~25% of the input, the value got converted to ~25% of the output.



Map command

[Show Correct Answer](#)

[Show Responses](#)

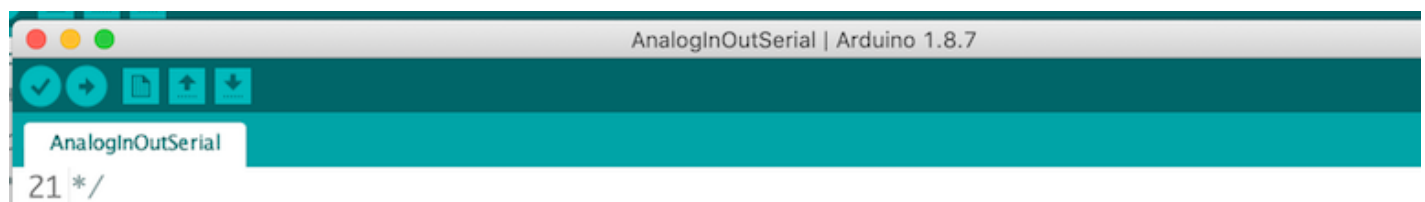
I have a variable X that's currently on a range of -10 to 10 but I want to remap it to a range of 15 to 75. Give me the line of code to reset the value of X using the map command.
DO NOT USE ANY SPACES IN YOUR RESPONSE.

Serial Monitor

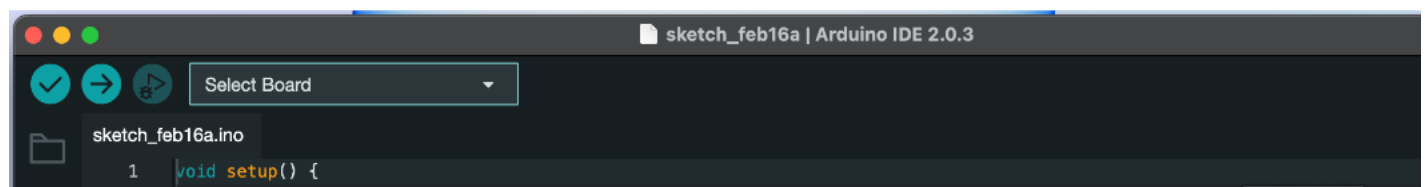
The last thing we should introduce is the [Serial Monitor](#). This is the method that your RedBoard can use to talk back and forth to your computer. We'll just use it for sending information to the computer for now. It becomes great for debugging by reporting out any information the Arduino is reading/writing or otherwise handling. This is particularly useful to be used with Analog signals since it's not as clear what their exact value is.

You access the serial monitor, only when connected to a running Arduino, by selecting the icon on the upper right of the Arduino IDE.

In 1.8 -



In 2.0 -



There are 2 basic commands to learn when using the Arduino Serial Monitor. For anything above and beyond the basics, there are several available tutorials from groups like [SparkFun](#) and [Adafruit](#).

Basic Commands:

- [Serial.begin\(speed\)](#)
 - Begins the Serial communication at a specified baud rate
 - Default/typical speed (baud rate) is 9600
 - To be used in the **setup()**
- [Serial.print\(...\)](#)

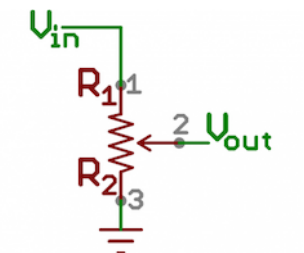
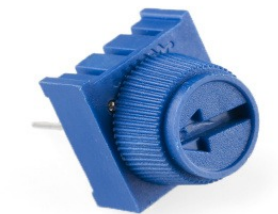
- Prints data to the serial port as human-readable ASCII text
- Use `Serial.println(...)` to print data and include a line-break after the data.

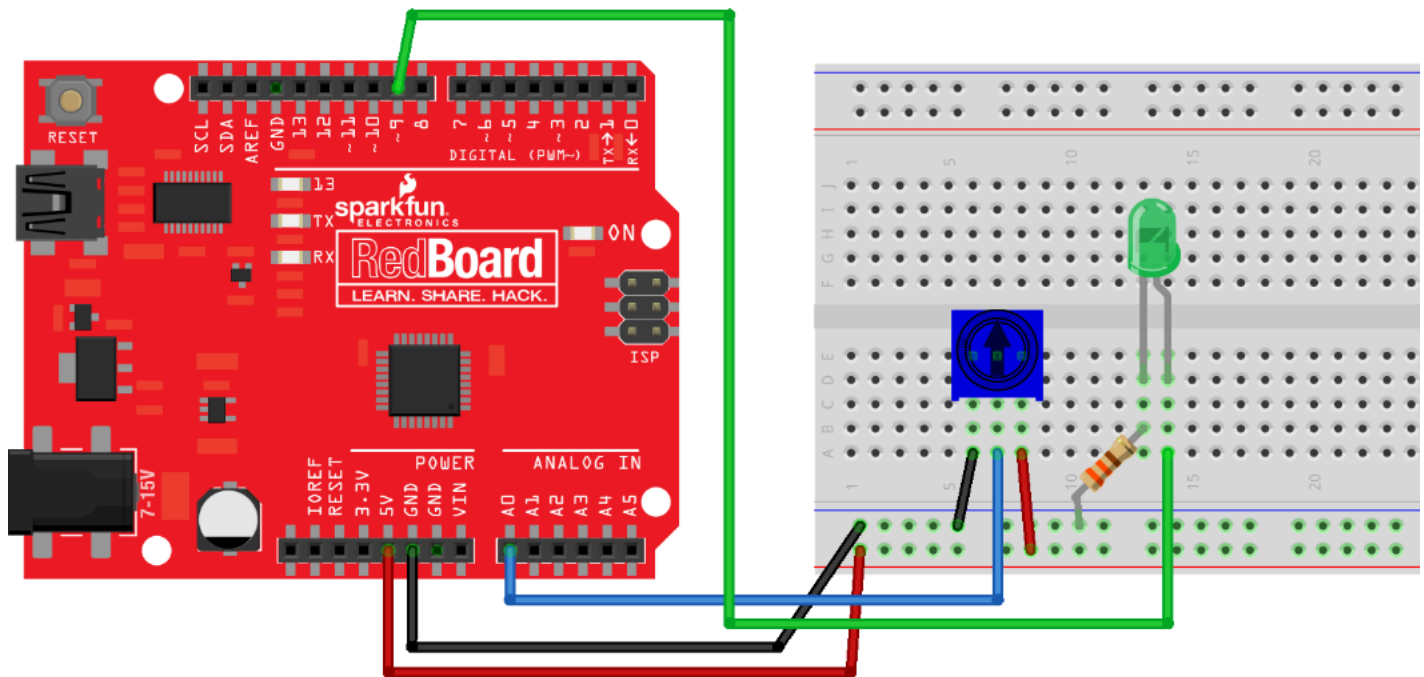
New Hardware

In P4, we will be adding in some analog sensors. You can experiment with them using the example [AnalogInOutSerial](#) and the wire diagrams provided below. We'll be starting with it for the part 1 activity. This example showcases all of the commands and concepts covered so far in this lesson.

Potentiometer

A potentiometer (also referred to as a trimpot or pot for short) is a variable [resistor](#). This 3 pin component, when a voltage source of X Volts is connected to one side and the ground is connected to the other, the middle pin will output 0 V to X V depending on the knob's position. Rotate towards ground and the output will go to 0V, rotate towards the voltage source and the output goes to X V. In our case, you'll see a range of 0 to 5 V.





Upload the code

- Open the built-in **Analog In Out Serial** example
 - Go to **File / Examples / 03.Analog / AnalogInOutSerial**
- Run the serial monitor and see the values
 - Cover and uncover the Photoresistor to see the changes
 - It's likely running faster than you can easily follow. Increase the delay to slow down the serial monitor updates. A delay around 100 is usually sufficient but you can increase it further if you need to. I wouldn't increase it more than 500 though to ensure reasonable read times.

Now you should see that the sensor value and output varies, relative to one another, based on the position of the potentiometer. You should also notice that they go across 2 different ranges. This is because, as previously mentioned, they have a different input and output range.

Question

✓ 100% Correct 30/30

How does the `map()` function in Arduino assist with PWM signal processing?

?

A	It smooths out PWM signals for more precise output.
B	It bypasses the need for a resolution difference between input and output.
C	It re-maps an input range to a different output range, allowing for more accurate signal conversion.
D	It increases the PWM resolution from 8-bit to 10-bit for better signal precision.

 HINT



 EXPLANATION



 Show Responses

 Show Answer

Part 2: Multiple LED Control

Before beginning, please keep in mind what amount of consideration and experimentation is productive for you and recognize when it's just unproductive struggle and grasping at straws. Everything you need is here. You are going to be given a detailed pseudocode, *use it. This means actually copy it into your script as comments and build off it.* There will also be a more scaffolded starter code later on if you need it. I've noticed those who struggle the most tend to take the pseudocode and start typing out commands above, quickly ignoring the pseudocode. Copy the pseudocode into your script as comments. Work with each line, folding your code into the pseudocode, rather than treating them as completely separate. It'll help you keep on task.

You can also reach out via Teams if you have any questions. This is meant to integrate new concepts using your expected current capabilities, not for you to completely synthesize understanding from scratch. If you need more clarity, need me to add something for your understanding, that's fine and useful for me to know. I want these to be robust enough for everyone and it helps to know where more scaffold, different approaches, and more reinforcement would be useful.

In this activity, we will be using a potentiometer.

Build an Example

We will be building the built-in Arduino example **Analog In Out Serial** and the circuit for a potentiometer (see above)

Build the circuit

Build the above circuit for the potentiometer. The following circuit requires:

- A Potentiometer
- An LED
- A 330 Ohm Resistor
- Wires

Upload the code

- Open the built-in **Analog In Out Serial** example
 - Go to **File / Examples / 03.Analog / AnalogInOutSerial**
- Save a copy as this part of the assignment
 - Update the title block to be yours for this assignment

Understand the Example

Once you've uploaded the example and have the circuit running, observe how it works. You can use the knob on the potentiometer to control the output of the LED. You can also open the **Serial Monitor** by clicking the Serial Monitor button in the upper right of the Arduino window. It will be scrolling very fast but you can see a print out of the sensor and output values. If you need to slow it down, increase the delay in the example, re-upload, and try again. A value of approximately 100-250 should be sufficient. Just keep in mind that the delay here puts a delay in reading the sensor and updating the output so you want to keep it minimized.

Let's break down how it works:

The Circuit

In the same way that the ballast resistor decreases the voltage slightly to protect the LED, the potentiometer gives you a variable resistance to control the voltage allowed to the analog input

pin A0. That will determine the signal being read by that pin. The PWM-enabled pin 9 then outputs a PWM signal with a duty cycle dependent on that potentiometer position.

The Code

Let's understand the code by breaking down what's new about it. This'll make it easier to work with later.

The Setup

- **No `pinMode(...)`**

in **`setup()`**

- You ***do NOT need*** to declare **pin mode** for **analog pins**
- **Declaring a pin mode will force a digital signal**

response, so outputs and inputs will only be treated as their extremes of 0 V and 5 V.

- **`Serial.begin(9600);`**

- Establish a serial connection during the setup.
- You can access it by hitting the **Serial Monitor** icon in the **Arduino IDE**.

```
const int analogInPin = A0;
const int analogOutPin = 9;
int sensorValue = 0;
int outputValue = 0;

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

void loop() {
  sensorValue = analogRead(analogInPin);
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  analogWrite(analogOutPin, outputValue);

  Serial.print("sensor = ");
  Serial.print(sensorValue);
  Serial.print("\t\t output = ");
  Serial.println(outputValue);

  delay(2);
}
```

Code only from the
Arduino example

The Loop

- **`analogRead(pin);`**

- Reads an analog signal from that pin
- Value reads as 0-1023

- **`analogWrite(pin,value);`**

- Writes a PWM signal from that pin
- Value reads as 0-255

- **`Serial.print(...);`**

- **`println(...)`** adds newline at end
 - Like **`disp()`** in MATLAB
- Special Characters
 - **`/t`** to tab
 - **`/n`** for a new line

- **`map(value, fromLow, fromHigh, toLow, toHigh);`**

- Re-maps a number from one range to another, providing some signal processing. This allow for converting a signal that comes in on a range of 0-1023 to the output range of 0-

255.

- **value**
 - The number to map
- **fromLow**
 - The lower bound of the value's current range
- **fromHigh**
 - The upper bound of the value's current range
- **toLow**
 - The lower bound of the value's target range
- **toHigh**
 - The upper bound of the value's target range

Modify the Example: Multiple LEDs

WARNING: I have tried to scaffold this to help you develop it on your own. There is a video later to help visualize the intended behavior and it goes through some pseudocode to code breakdowns. I want you to try a little on your own to push your capabilities but I also only want you to put in some **productive** struggle for this. Recognize when you start engaging with **unproductive** struggle and get the help you need to make be productive again.

IF YOU NEED HELP, don't hesitate to ask. There are lots of opportunities and resources:

- Feel free to work in small groups on these assignments
- Go to FYELIC for some Red Vest assistance
- Chat with Top Hat's ACE or
- Stuck? Start a Thread in Top Hat
- Reach out to me over MS Teams.

Currently, the example goes like this:

1. **Read the analog in value;**
2. **Map it to the range of the analog out;**
3. **Change the analog out value;**
4. **Print the results to the Serial Monitor;**

This follows our standard paradigm of **Sense** (Line 1), **Think** (Line 2), and **Act**(Line 3). We have an added paradigm of **Check** (line 4) to give us some feedback on what the microprocessor is doing. We're going to stick with that paradigm going forward so think in terms of **Sense, Think, Act,**

Check as you incorporate the following pseudocode. I've included some sub-headings for Sense, Think, Act, Check to clarify

1. **SENSE -**
2. Read the analog in value
3. **THINK -**
4. IF analog in value < 1/3 signal range
5. Map LED1 output level from min to 1/3 of signal range
 to the range of the analog out;
6. Set LED2 output level to 0;
7. Set LED3 output level to 0;
8. ELSE IF analog in value between 1/3 and 2/3 signal range
9. Set LED1 output level to full;
10. Map LED2 output level from 1/3 to 2/3 of signal range
 to the range of the analog out;
11. Set LED3 output levels to 0;
12. ELSE IF Analog in value > 2/3 signal range
13. Set LED1 output level to full;
14. Set LED2 output level to full;
15. Map LED3 output level from 2/3 to max of signal range
 to the range of the analog out;
16. **ENDIF**
17. **ACT -**
18. Write the LED Output values for LED1, LED2, & LED3 to the new levels
19. **CHECK -**
20. Print the results to serial monitor:
 signal level, LED1 Output level,
 LED2 Output Level, & LED3 Output level

Here, we still maintain that paradigm, but we've added 2 LEDs and are controlling them based on the potentiometer range; **Sense** (Line 2), **Think** (Line 4-16), **Act**(Line 18), & **Check** (Line 20).

You can more easily copy it from here:

<https://raw.githubusercontent.com/boconn7782/CourseCode/main/P4L>

This code is partially complete. You'll be filling in the gaps as you go. You'll have to comment out incomplete elements though if you want to run tests or check it. Keep that in mind as you build it out.

Modify the Circuit:

- Add 2 more LEDs to the circuit using PWM pins

It's useful to check your circuit whenever you modify it. You can do this 3 ways:

1. Plug each new LED's signal wire the signal pin into the voltage column to check that the LED lights up, then move it to the PWM pins you intend to use.
2. Plug each new LED's signal wire into the already used Pin 9 to check if it lights up like the existing LED did, then move that jumper wire back to the PWM pins you intend to use.
3. Update your code to add the new LEDs and have them all react the exact same as the first with some quick copy/paste updates to the code before implementing the pseudocode above.

Modify the Code:

Let's see what a similar pseudocode for the **Analog In Out Signal** looks like:

1. **Read the analog in value;**
2. **Map it to the range of the analog out;**
3. **Change the analog out value;**
4. **Print the results to the Serial Monitor;**

Understand the Example

Translating the example pseudocode:

Sense

Read the analog in value; refers to taking the input signal in. This isn't limited to one command. It involves some preparation and setup to enable the Arduino to do this. This associates with setting up the following variables and commands:

- Create a variable to keep track of what pin the sensor is connected to
 - **const int analogInPin = A0;**
- Create a variable to store the sensor value in
 - **int sensorValue = 0;**
- Then you can use the Arduino Command to read the sensor and store its value

- `loop(){...`
`sensorValue = analogRead(analogInPin);`
`...}`

Note that one line required some establishment of variables before `setup()` and then some code in the `loop()`.

Think

Map it to the range of the analog out; refers to re-mapping that signal value from it's 0-1023 range to an appropriate output range and then set the LED to that output level. This means you need all of the information for that LED as well as writing the output to it. This associates with setting up the following variables and commands:

- Create a variable to store the output value in
 - `int outputValue = 0;`
- Then you can use the Arduino Command to convert the value sensor value to an appropriate range and store the new value
 - `loop(){...`
`outputValue = map(sensorValue, 0, 1023, 0, 255);`
`...}`

So not only do we need to establish the LED pin, but to maintain that **sense think act** paradigm, we need a variable to store the output during the **think** phase and we don't actually write that value to the LED until later. Also note that we use that output variable in our **check** phase so it does double duty.

Act

Change the analog out value; is where action takes place. This is referring to writing that value to the LED using the following command

- Create a variable to track of what pin the LED is connected to
 - `const int analogOutPin = 9;`
- Then you can use the Arduino Command to change the output voltage on that pin based on the desired output value
 - `loop(){...`
`analogWrite(analogOutPin,sensorValue);`
`...}`

Note that we stored that information previously so those variables are being used in multiple elements as we noted earlier.

Check

Print the results to the Serial Monitor; is where that check is occurring, referring to the printing out of sensor and output values to the serial monitor. This is good for debugging and knowing what the values you're actually working with. For this, you'll need to start the serial monitor, hold onto all the information you want to print, hence using `outputValue`, then print it out to the serial monitor. It's also important to the print out of details for the human user to monitor the program, making sure it's not accidentally creating SkyNet or the like. This associates with setting up the following variables and commands:

- Establish the connection to the Serial monitor
 - `void setup() {
 Serial.begin(9600);
}`
- Then you can send information to the Serial Monitor using the appropriate Arduino Commands
 - `loop(){...
 Serial.print("sensor = ");
 Serial.print(sensorValue);
 Serial.print("\t output = ");
 Serial.println(outputValue);}`

Note that we stored that information previously so those variables are being used in multiple elements as we noted earlier.

Understand the Updated Pseudocode

Use the above as a reference for translating the below pseudocode and implementing it. Here it is again for your convenience:

1. **Read the analog in value**
2. **IF analog in value < 1/3 signal range**
3. **Map LED1 output level from min to 1/3 of signal range to the range of the analog out;**
4. **Set LED2 output level to 0;**
5. **Set LED3 output level to 0;**
6. **ELSE IF analog in value between 1/3 and 2/3 signal range**

7. Set LED1 output level to full;
8. Map LED2 output level from 1/3 to 2/3 of signal range to the range of the analog out;
9. Set LED3 output levels to 0;
10. ELSE IF analog in value > 2/3 signal range
11. Set LED1 output level to full;
12. Set LED2 output level to full;
13. Map LED3 output level from 2/3 to max of signal range to the range of the analog out;
14. ENDIF
15. Write the LED Output values
for LED1, LED2, & LED3 to the new levels
16. Print the results to serial monitor:
signal level, LED1 Output level,
LED2 Output Level, & LED3 Output level

For microcontrollers, the pseudocode/flowchart doesn't need to cover all required elements like establishing the Pin and setting up the PinMode. That can be assumed as a necessity. For instance:

- Because there's a reference to 3 LEDs, in your code you will need to:
 - Establish a constant integer variable for each connected pin #
 - Establish an output variable to store information to be used in Act and Check
- Because there's a reference to serial monitor, in your code you will need to:
 - Start the serial monitor running in the setup().



pinMode for analog outputs

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

We're adding 2 LEDs to use as analog outputs. Since your original LED had a variable for it's pin number and a variable for it's output value, you should be creating new ones for the 2 new LEDs. Do you need to declare pinModes for the new LEDs?

A

Yes

B

No

This may help in visualizing the changes:

Analog In Out Serial

1. Read the analog in value;
2. Map it to the range of the analog out;
3. Change the analog out value;
4. Print the results to the Serial Monitor;

CHECK

SENSE

THINK

ACT

Multiple LEDs

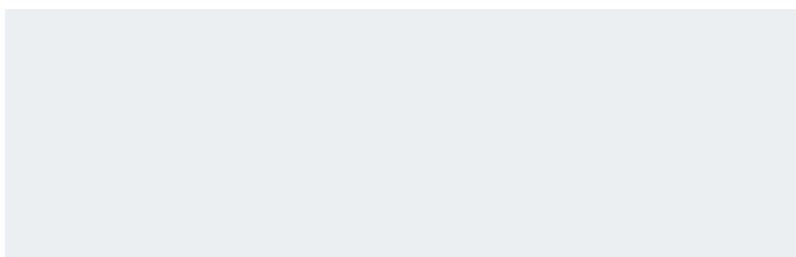
1. Read the analog in value
2. IF analog in value < 1/3 signal range
3. Map LED1 output level from min t signal range to the range of the out;
4. Set LED2 & LED3 output levels to
5. ELSE IF analog in value between 1/2/3 signal range
6. Set LED1 output level to full;
7. Map LED2 output level from 1/3 t signal range to the range of the out;
8. Set LED3 output levels to 0;
9. ELSE IF PotSig > 2/3 signal range
10. Set LED1 & LED2 output levels to
11. Map LED3 output level from 2/3 t signal range to the range of the out;
12. ENDIF
13. Change the LED Output values for I LED2, & LED3 to the new levels
14. Print the results to serial monito signal level, LED1 Output level, I Output Level, & LED3 Output level

This pseudocode also makes use of an [if/else if](#) statement. As a reminder of the C++ syntax:

```
if (condition1) {  
    // do Thing A  
}  
else if (condition2) {  
    // do Thing B  
}  
else {  
    // do Thing C  
}
```

Final Behavior

The following video showcases the intended behavior:



Video

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

Try on your own first

I recommend making a first pass on your own but don't spend a significant amount of time. Remember, you're looking for productive struggle to help in understanding. If you're just trying random things without thoughtful implementation or just generally getting frustrated, it's no longer productive. If you need a little more guidance, follow along with the rest of the walkthrough. If you need further help, don't hesitate to reach out.

Walkthrough

Let's start with the original code that's set up for 1 potentiometer and 1 LED:

```
1. const int analogInPin = A0; // Analog input pin that the
   potentiometer is attached to
2. const int analogOutPin = 9; // Analog output pin that the LED is
   attached to
3. int sensorValue = 0; // value read from the pot
4. int outputValue = 0; // value output to the PWM (analog out)
5.
6. void setup() {
7. // initialize serial communications at 9600 bps:
8. Serial.begin(9600);
9. }
10.
11. void loop() {
12. // read the analog in value:
13. sensorValue = analogRead(analogInPin);
14. // map it to the range of the analog out:
15. outputValue = map(sensorValue, 0, 1023, 0, 255);
16. // change the analog out value:
```

```
17. analogWrite(analogOutPin, outputValue);
18. // print the results to the Serial Monitor:
19. Serial.print("sensor = ");
20. Serial.print(sensorValue);
21. Serial.print("\t output = ");
22. Serial.println(outputValue);
23. // wait 2 milliseconds before the next loop for the analog-to-digital
24. // converter to settle after the last reading:
25. delay(2);
26. }
```

The Potentiometer

Let's focus on the potentiometer first. In the code there is the following that pertains to it:

Associated Variables

- `const int analogInPin = A0; // Analog input pin that the potentiometer is attached to`
- `int sensorValue = 0; // value read from the pot`

We're not adding an sensors so we're good here. We still have 1 pin associated with a sensor and only reading in one value so only need one variable to capture that.

Setup

Nothing associated with the potentiometer here. We do **not** need pinMode for analog signals.

Sense

- `// read the analog in value:`
- `sensorValue = analogRead(analogInPin);`

We're not adding an sensors so we're good here. We still have 1 pin associated with a sensor and only reading in one value so only need one variable to capture that.

Think

- `// map it to the range of the analog out:`
- `outputValue = map(sensorValue, 0, 1023, 0, 255);`

This is going to be more complicated. We're going to be dealing with the sensor along a few different ranges so instead of just being interested in the range of 0 to 1023 but we're now interested in the first third (0 to 341), the second third(342 to 682) and the last third(683-1023). This will also be involving the setting the output values to our LEDs so let's deal with that when we get to the LEDs. Just know that we will no longer be dealing with 1 `outputValue` but rather 3, one for each of the LEDs.

Act

Nothing associated with the potentiometer here. All action will be associated with the output components, the LEDs.

Check

- `// print the results to the Serial Monitor:`
- `Serial.print("sensor = ");`
- `Serial.print(sensorValue);`

We're not adding a new sensor value to be concerned with so we're good here as well.

The LEDs

Now we're adding 2 LEDs to have a total of 3.

Associated Variables

- `const int analogOutPin = 9; // Analog output pin that the LED is attached to`
- `int outputValue = 0; // value output to the PWM (analog out)`



For 1 LED, we have the `analogOutPin` and `outputValue`. Now that we're using 3 LEDs, we should have 3 of each. So you should make sure you have


- `analogOutPin1`
- `analogOutPin2`
- `analogOutPin3`
- `outputValue1`
- `outputValue2`
- `outputValue3`

You can also use a different naming convention if you like. You can use something like **LED1pin** or **LED1val** or have it reference the LED color rather than a number. Whatever makes more sense for you but make sure you're consistent throughout the rest of your script and it's descriptive enough as to be clear for any one else reading your script. I will stick with the above just since it's built off the example even though I, personally, tend to go with a component reference in my descriptive variable names.



Setup



Nothing associated with the LEDs here. We do **not** need pinMode for analog signals.



 **To pinMode or not to pinMode**  100% Correct 29/29

If you use pinMode, the LEDs will respond correctly, providing a range of light intensity as an output. 

A	True
B	False

 **HINT** 

 **EXPLANATION** 

 Show Responses  Show Answer

Sense

We don't need to sense with the output components, so we're good here too.

Think

Currently we take the input of the potentiometer and map it's entire range to one output.

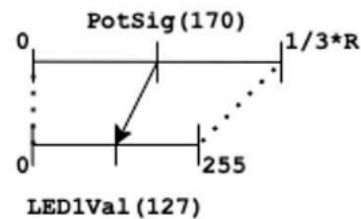
- **// map it to the range of the analog out:**
- **outputValue = map(sensorValue, 0, 1023, 0, 255);**

This is where the logic comes into play of the if statement. Now we want to map partial ranges to the full output of each LED.

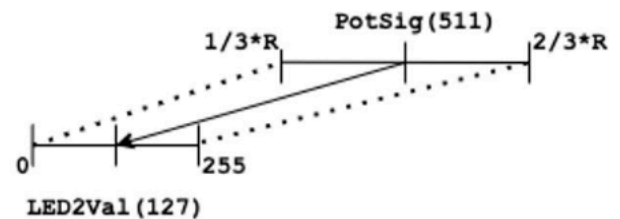
This graphic might help to illustrate what it should be doing. I've updated the names to be more descriptive to the component.

```
if (PotSig < 1/3 R)
{
  Map LED1 output level
  Set LED2 & LED3 output
  levels to 0
}
else if (PotSig < 2/3 R)
{
  Map LED1 output level to Full
  Map LED2 output level
  Set LED3 output level to 0
}
else
{
  Set LED1 & LED2 output
  levels to Full
  Map LED3 output level
}
```

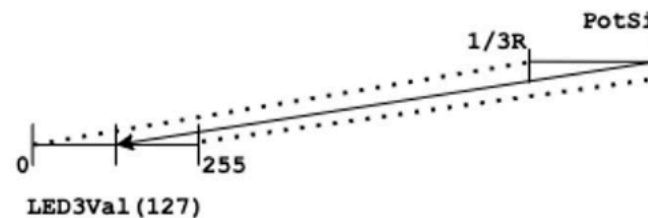
IF PotSig < 1/3 signal range (R)



ELSE IF PotSig < 2/3 signal range (R)



ELSE



END

This uses PotSig instead of analog in signal and LED1output level instead of outputValue1

So in our if statements, we want it broken into the 3 ranges

1. `if (sensorValue <= 341) {`
2. `// do Thing A`
3. `} else if (sensorValue > 341) && (sensorValue <= 683) {`
4. `// do Thing B }`
5. `else if (sensorValue > 683) && (sensorValue <= 1023){`
6. `// do Thing C`
7. `}`

Remember though, with IFTTT, the first thing that's true will be what occurs. We can simplify this to:

```

1. if (sensorValue <= 341) {
2.   // do Thing A
3. } else if (sensorValue <= 683) {
4.   // do Thing B }
5. else {
6.   // do Thing C
7. }

```

Even though when `(sensorValue <= 341)` is true and therefore `(sensorValue <= 683)` will also always then be true, we don't have to concern ourselves with defining the second circumstance to include `(sensorValue > 341)`. Since the first statement was true, the second statement is not even considered.

We want to be mapping our output values based on their associated ranges. So the `outputValue` for an LED gets mapped from the `sensorValue` within its associated sensor range to the full range of the LED output.

- `outputValue1 = map(sensorValue, 0, 341, 0, 255);`
- `outputValue2 = map(sensorValue, 342, 682, 0, 255);`
- `outputValue3 = map(sensorValue, 683, 1023, 0, 255);`

But if it's just on full or off, it should just be 0 or 255.

- `outputValue1 = 0;`
- `outputValue1 = 255;`

All the code and conversions are there but I do want you to map it back to the pseudocode yourself. Don't hesitate to reach out for help from peers, FYELIC, the TAs, or me if it's not coming together for you quickly.

For your reference , here's the **Think** portion of the pseudocode:

```

1. ....
2. IF analog in value < 1/3 signal range
3.   Map LED1 output level from min to 1/3 of signal range
   to the range of the analog out;
4.   Set LED2 output level to 0;
5.   Set LED3 output level to 0;
6. ELSE IF analog in value between 1/3 and 2/3 signal range
7.   Set LED1 output level to full;

```

8. Map LED2 output level from 1/3 to 2/3 of signal range
 to the range of the analog out;
9. Set LED3 output levels to 0;
10. ELSE IF analog in value > 2/3 signal range
11. Set LED1 output level to full;
12. Set LED2 output level to full;
13. Map LED3 output level from 2/3 to max of signal range
 to the range of the analog out;
14. ENDIF
15. ...

THE MOST COMMON ISSUE I've found with this assignment is people integrate the `analogWrite` within the if-else if statement, even though the pseudocode clearly never references it there. You should not call `analogWrite` until you are in the **Act** phase of your script.

Act

After the output value had been mapped from the sensor value, it's time to update the setting for the LED.

- `// change the analog out value:`
- `analogWrite(analogOutPin, outputValue)`

Now it's time to update it so it's doing the same for each of the associated `analogOutPins` and their associated `outputValues`. Most young coders first impulse will be to attempt the `analogWrite` inside the if statements. But we're sticking with a sense, think, act, check paradigm here and there's a benefit that will become apparent in Check.

Check

- `// print the results to the Serial Monitor:`
- `...`
- `Serial.print("\t output = ");`
- `Serial.println(outputValue);`

This is why we stored the output value. We get to use it in 2 places, once to Act and once to Check. We're going to do the same thing with each of our individual output values. Remember that you only want to use `serial.println()` for the last one since that creates the linebreak. So our print out now becomes:

- `// print the results to the Serial Monitor:`

- `Serial.print("sensor = ");`
- `Serial.print(sensorValue);`
- `Serial.print("\t output 1 = ");`
- `Serial.print(outputValue1);`
- `Serial.print("\t output 2 = ");`
- `Serial.print(outputValue2);`
- `Serial.print("\t output 3 = ");`
- `Serial.println(outputValue3);`

Submit your Assignment

As you've done for other assignments, you should submit the files in the corresponding assignment in Canvas. You need the following files for this assignment.

- The Raw Code –
 - The .ino file
- PDF of code and extras
 - A copy of the raw code
 - An image of the circuit
- The video of your working circuit – The .mp4 file
 - A standard movie file - .mp4, .mov, [etc.](#)
 - As a media comment so the TA can view it in Canvas

Make sure you're using the correct naming conventions



P4 Review 1



100% Correct

29/29

What is the function of the `map()` command in the Arduino programming language?



A

It converts an analog input signal to an analog output signal.

B

It re-maps a value from one range to another range.

C

It allows the Arduino to talk back and forth to the computer.

D

It measures or reacts to changes in ambient-light.

 HINT



 EXPLANATION



 Show Responses

 Show Answer

The following survey is for my reference to help improve future assignments. The results are not checked until after the semester is completed and participation is not required.


Assignment Feedback

The following helps with updating assignments as well as the development of new assignments. The results of this survey are not checked until after the completion of the semester. These have no effect on your grade and participation is not required. They just help with updating assignments and with the development of new assignments.

This survey tracks NU emails in order to be able to correlate with performance and establish trends in the data. Again, you do not need to participate.

b.oconnell@northeastern.edu [Switch account](#)



 Not shared

* Indicates required question

What is the assignment code? Ex. G1H *

Your answer

Exported for Brian O'Connell on Tue, 27 May 2025 19:30:11 GMT



Study with Ace