Exported for Brian O'Connell on Tue, 27 May 2025 19:27:34 GMT

P2L: Introducing Arduino/C++

The following is the pre-lab lesson content for Programming 2: Introducing Arduino/C++. We will be learning Arduino and C++ simultaneously since the Arduino programming is a C++ variant, meaning that it is a specialized version of C++ for use with Arduino Microcontrollers. The pre-lab will focus on just getting started with Arduino and have some foundational knowledge about microcontrollers and circuits going into class. In class, we will cover more of the fundamentals of C++, basic usage, and best practices.

In it, you will go over the following:

- Install Arduino
- Go over basic elements
- Review some Basics about electricity
- Basic Circuit elements
- Run a basic example
- Submit an Arduino AssignmentA

Install Arduino

This first portion just goes over getting Arduino installed on your computer and ensuring connection to your Arduino. You may have already done this if you completed some of the activities included with the booklet included in the Sparkfun Inventor's Kit (SIK). This document will point you to some of the resources in that since you should have it on hand but will also link to Sparkfun's digital version as well as some of the Arduino documentation because those are kept up to date by the vendors.

The purpose of this part is to instruct and guide you on how to get started with the Arduino IDE & the Sparkfun Inventor’s Kit and be prepared for the in-class activities and assignments. In it, you will be guided on how to:

- Access the Arduino IDE
  - Through VLAB or personal installation
- Flash a Script to your RedBoard
- Save your work and recover your work
- Properly set up and submit documents for SparkFun assignments

NOTE: Most people in the First Year Engineering program refer to Arduino products and assignments that use them as SparkFun. It's a colloquial expression due to our heavy use of their products and that all students use the Sparkfun Inventor's Kit. I've organized the assignments differently, under the more generic 'programming' title, and I focus more on them as generic Arduino and microcontroller lessons. I wanted you to be aware of that variance particularly if you go to FYELIC for help since 'Arduino', 'microcontroller', and 'Sparkfun' become a bit interchangeable there.

The SparkFun Inventor's Kit (SIK)

You need to have the SIK or equivalent for the Arduino lessons and homeworks. If you haven't, do so ASAP -https://www.sparkfun.com/northeastern. If you are borrowing an older version from a friend or a different brand's kit, that's fine but feel free to ask Prof. O'Connell to be sure it has everything you'll need.

YOU CAN STILL COMPLETE THIS ASSIGNMENT BY GOING TO FYELIC AND USING THEIR SPARE PARTS IN THERE BUT YOU STILL NEED TO GET THE SPARKFUN INVENTOR'S KIT ASAP.

I will also allow for this, and only this, Arduino assignment to be completed using TinkerCAD. You still must install Arduino on your computer and install the drivers as instructed in the online version (since you don't have the physical copy since you're still waiting on your kit) of the SparkFun Inventor's Kit Guide.

First and foremost, DO NOT use the black backer board. The provided booklet instructs you to essentially glue the breadboard to it. This becomes an issue later on if you want to use the breadboard in any other configuration like to help with wiring up your project. You'll be fine to just not use that black backer board ever. If you've already done so, it's not a big deal. The double sided tape is not so strong that the breadboard can't be peeled off if you want to later on or you can just work with that constraint in your designs.

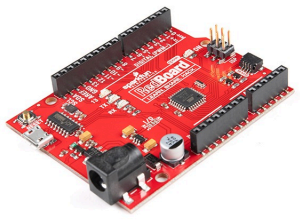
The only minor thing you need to do regarding the physical kit is to cut the zip tie on the jumper wires. This obviously only applies if your kit came with them zip tied. They also sometimes package them loose in a bag which means you don't have to worry about it. The important thing is to have them available for use later in this pre-lab and for the in-class activities.

Question 1 Show Correct Answer Show Responses

Have you ensured that your jumper wires are freely available to be used and not trapped together?

A	I cut the zip tie
B	They were in a resealable bag
C	I don't have my kit yet but promise to do this when I get it. I'm using FYELIC resources for now.
D	I'm just going to ignore that and do it later. Probably won't forget and then not be able to complete this assignment or participate fully in class because I'll have to run and find scissors somewhere...

We will later go over more of the specific contents of the SIK later but for this task, all you need is the RedBoard (See the image included here) and the USB cable. The RedBoard is SparkFun's version of the Arduino Uno, Arduino's flagship microcontroller and one of the more commonly available ones on the market.



SparkFun RedBoard

Getting Started with Arduino

For this we will be focusing on just making sure you can access scripts for your Arduino based microcontroller and upload those to the RedBoard. We won't be getting into the electronics in the kit until the later tasks.

IT IS STRONGLY RECOMMENDED that you download and install the Arduino IDE rather than use VLab for this. You can access it through VLab but there's an added level of complexity to making sure you can maintain serial communication through your computer and into VLab that has been problematic in the past. The Arduino IDE is relatively small (less than 600 mb) and is not processor heavy. Therefore, I'm not bothering with the Pros and Cons and skip to just installing it on your personal computer. There is a description of how to enable using USB components in VLAB included though. If you insist on using VLAB rather than installing Arduino locally, you still need to install the USB drivers so that your computer recognizes the microcontroller.

Installing Arduino IDE

Before starting, know which Arduino you have. You most likely have the most recent kit from Sparkfun and and therefore have the newer SparkFun Redboard Quiic. If you have an older kit, you'll have the SparkFun Redboard. If you're unsure which you have, go to the links and compare. They are visibly different. Both are Arduino Uno clones and function with the Arduino IDE, they just have some different setup elements. Some of you may have a generic Arduino Uno which is allowed but you'll have to ensure that you have the required components for any activities and assignments. That includes the proper drivers.

- SparkFun Redboard Quiic
  - Uses a USB Micro-B cable
  - Needs you to install the CH340 drivers for communication
- SparkFun Redboard or a generic Arduino Uno
  - Uses a USB Mini-B cable or a USB A cable
  - Needs you to install the FTDI drivers for communication

Installation

1. Look to your SparkFun Inventor's Kit Guide.
    1. There is an online version. It's not a pdf of the guide, but a version formatted for the web.
2. Follow the instructions from page 6, 8, & 9 (You can download the SIK Examples from page 7 but they are not required for Prof. O'Connell's courses))
    1. Download and install the Arduino IDE
    2. Install USB drivers
        1. You may not need this step depending on your computer. You may wait until after the 'Select your Board and Device' step. If it didn't recognize the board, you likely need these drivers.
        2. For older arduinos: install the FTDI drivers
        3. For newer arduinos: install the CH340 drivers (Most of you will need these)
    3. Connect the RedBoard to your computer
    4. Open the Arduino IDE
        1. Page 8 of the guide goes over the User Interface
    5. Select your Board (Arduino Uno) and port (Dependent on computer type)
        1. Page 9 of the Guide or this online resource
3. You've now connected your RedBoard to your Arduino IDE

Accessing through VLAB

You may also access through VLAB like you would any other application on VLAB. THIS IS NOT recommended since Arduino IDE doesn't take up that much space and connecting to USB through VMWARE adds an unnecessary level of complexity. If you do want to access it through VLAB, take note of a few things though:

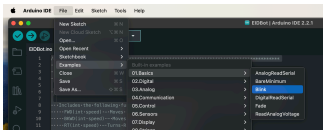
1. You need to enable the USB connection through VLab
    1. This is different for every operating system so I will discuss it generally.
    2. For more specifics about your operating system, you may need to search VMware's help options or other internet sources
2. **While NOT logged** into one of the VLab COE_open_# computers
    1. go to *Connection* in the menu bar
    2. This may also be in preferences or settings, depending on VMware version.
       It may also have the word USB in it.
3. Select "Start remote USB services"
    1. It may be called something different on your OS but likely similar
4. Select an option similar to "Automatically connect when inserted"

1. If you select “Automatically connect at startup”, you will have to restart VLab every time you disconnect your RedBoard.

These directions are very general and may not specifically apply to your version. For more assistance, see Prof. O’Connell during office hours or visit FYELIC for redshirt assistance.

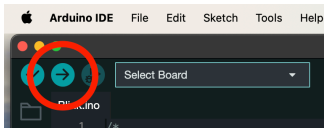
Let's check to see if that worked

- Open the built-in Blink example
  - Go to *File / Examples / 01.Basics / Blink*



- This will open up a new window for the example script Blink

- Select the Upload button
- 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?



- See what your board is now doing
  - If you see no difference, ie the light is still blinking, it’s likely because your board was pre-flashed with this script. Some manufacturers use this as the default script they use when testing new Uno type boards.
  - Try uploading "BareMinimum" instead (This should turn the LED off since its a script that does nothing) and then re-uploading Blink to see a difference in behavior.
  - Try changing the value of the delays (`delay(1000)` to `delay(500)` or another significantly different value), re-upload, and see what happens. You have to re-upload after changing the values for the Arduino's actions to reflect that change.

IF YOU HAVE ANY ISSUES, remember that the TAs and Prof. O'Connell have office hours, FYELIC is open weekdays with plenty of red vests to help, and you can also message Prof. O'Connell on Teams for real time help or to at least schedule some help outside of office hours.

TinkerCAD Simulations

IF YOU DO NOT HAVE AN ARDUINO KIT AVAILABLE, you can still engage with the following using TinkerCAD. TinkerCAD Circuits is an online simulator for basic circuits. You can set up a virtual Arduino circuit and program using the Arduino language in it. It's a useful tool for the short term but long term you need to have a physical board and electronics for the later assignments.

You can access TinkerCAD circuits here: https://www.tinkercad.com/circuits and log in using a student account. You can either use your AutoDESK credentials, the same one you used when registering your AutoCAD. Or you can log in using google and your northeastern.edu credentials. Use this to get started: https://www.tinkercad.com/things/kud6j7TjAXy-g2l1template. I've already populated it with an Arduino and loaded in the Blink example in the code editor from the Arduino IDE. Make your own copy to be able to edit and experiment with it.

To add components, just drag and drop components from the right side ribbon into the workspace. You can search for components too. Try searching for 'Arduino Uno', 'LED', 'Resistor', and 'breadboard' to find the things you need for todays pre-lab. You can connect components by selecting their pin or hole and then dragging to the connecting pin or hole. This will form a wire between them. Play around to add components and make sone connections. There's nothing to submit here so this is purely to familiarize yourself with the platform. We'll discuss the actual function of those components later in the lesson.

Learn

Now that we have the Arduino running, let's learn a little bit about what's it's doing and how it works. The following introduces a variety of Arduino/C++ elements that you will need to understand and be able to use for the in-class content. It references some of the built-in Arduino examples that you can setup and run to illustrate the concepts. There are periodic questions on this content that you will need to respond to. They are heavily weighted towards participation. Full credit for this task doesn't require 100% correctness but does require 100% participation.

If you want more information on any of the Arduino elements, commands, or concepts, links are included to Arduino help documentation as well as some SparkFun's resources.

What is Arduino?

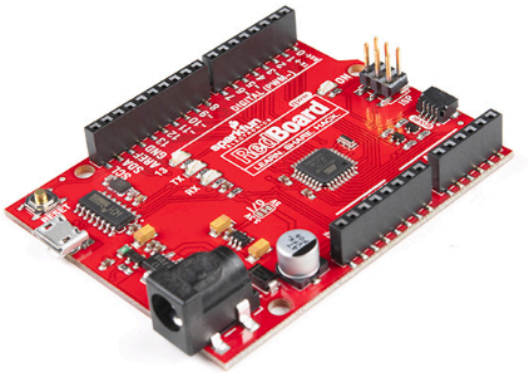
Arduino is an open-source electronics platform. It has been specifically designed to be (relatively) easy to use and get started with. It is used for a wide range of projects by both amateurs and professionals alike. It is inexpensive, cross-platform compatible, open source with a large development community, and extensible hardware and software options and support.

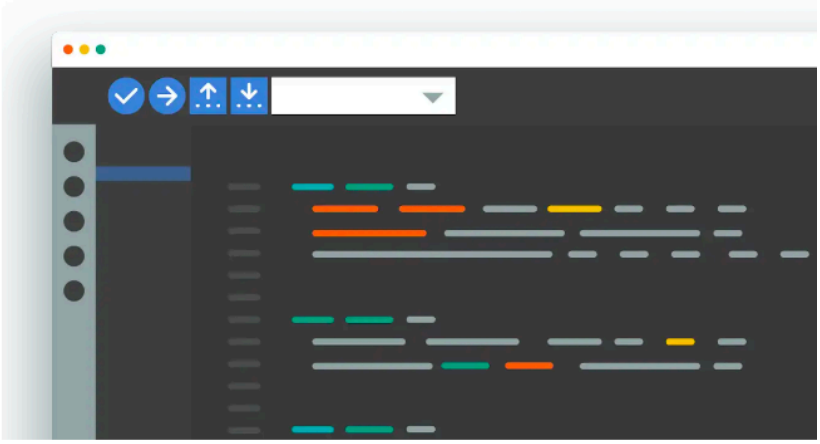
The Arduino System

With the hardware and software capabilities of the Arduino system, it is capable of a whole universe of actions. It is a system that can connect the physical and digital worlds within moments through a combination of digital and analog inputs and outputs

The Arduino System consists of two parts:

- Software
  - A simplified version of C++
- Hardware
  - A full line of microcontroller boards of varying levels of complexity





Together they can:

- Read inputs (sensors)
  - Sense change in light or sound

    - Sense A finger pressing a button
    - A Twitter message, etc.
- Send outputs (Actuators)
    - Activating a motor
    - Turning on an LED
    - Publishing something online, etc.

Through the thoughtful combination of those activities, you can achieve a wide range of tasks with just the kit required by this class and do even more by expanding the connected resources and components in your design.

The Software

Arduino IDE

IDE is the standard industry acronym for an Integrated Development Environment. Arduino created their own IDE to work with their version of the programming language C++ and to easily connect with their hardware. It contains a text editor for writing and developing your code, a message area for feedback, a secondary text console and a toolbar and menu bar for common functions and other menus. The Arduino IDE will compile your sketch and any required support libraries into a single unit of code for upload to your Arduino hardware. They also purposely tried to maintain consistency across platforms with their user environment so there is very little variation between Windows, Max, and Linux.

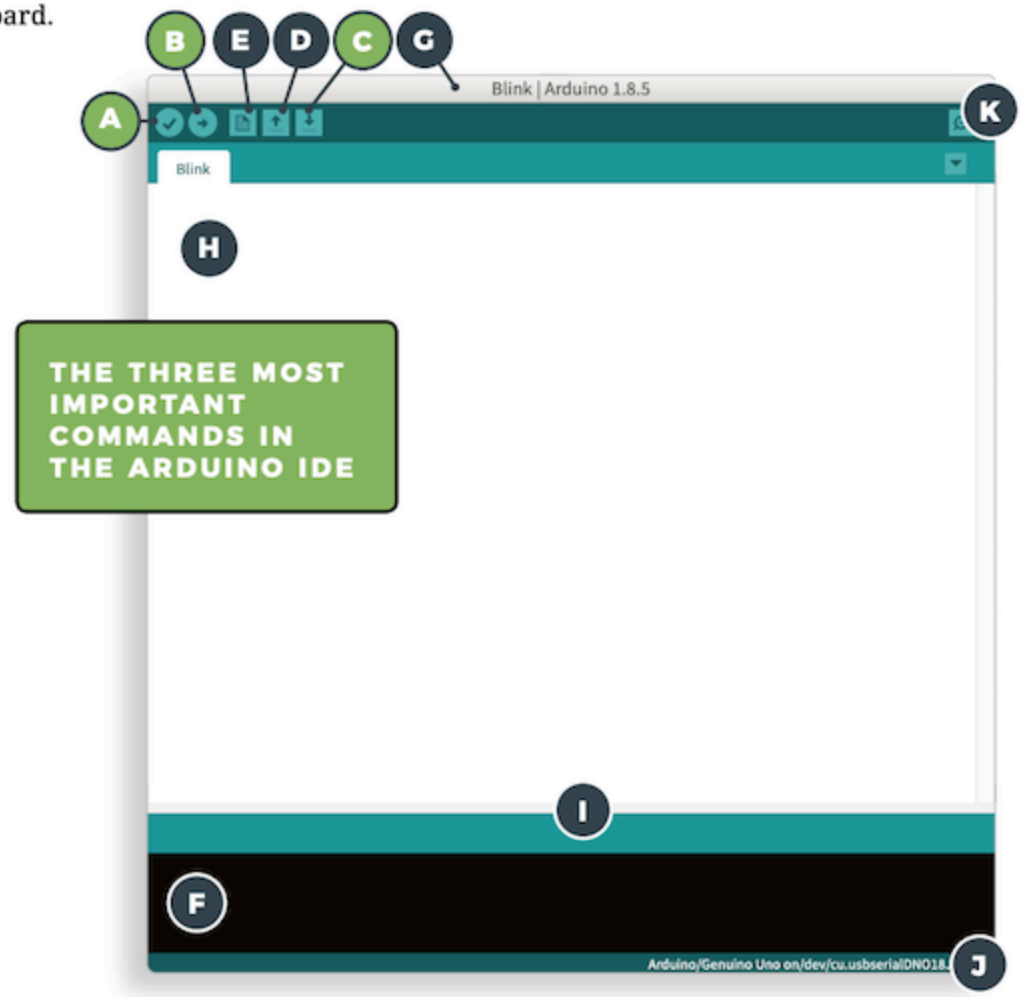
You can also reference the Intro of your Sparkfun Inventor's Kit (SIK) Guide for more information about the Arduino IDE.

General User Interface

The SIK Guide does an excellent job illustrating the GUI:

5. OPEN THE ARDUINO IDE:

Open the Arduino IDE software on your computer. Poke around and get to know the interface. We aren't going to code right away; this step is to set your IDE to identify your RedBoard.



GRAPHICAL USER INTERFACE (GUI)

A	VERIFY	Compiles and approves your code. It will catch errors in syntax (like missing semicolons or parentheses).
B	UPLOAD	Sends your code to the RedBoard. When you click it, you should see the lights on your board blink rapidly.
C	SAVE	Saves the currently active sketch.
D	OPEN	Opens an existing sketch.
E	NEW	Opens up a new code window tab.
F	DEBUG WINDOW	Displays any errors generated by your sketch.
G	SKETCH NAME	Displays the name of the sketch you are currently working on.
H	CODE AREA	Where you compose or edit the code for your sketch.
I	MESSAGE AREA	Indicates if the code is compiling, uploading or has errors.
J	CONNECTION AREA	Displays the board and serial port currently selected.
K	SERIAL MONITOR	Opens a window that displays any serial information your RedBoard is transmitting (useful for debugging).

Taken from the Sparkfun Inventor's Kit User Guide

Menu

The application menu bar, which may not be part of the window but rather at the top of your screen, includes drop-downs for **File**, **Edit**, **Sketch**, **Tools**, and **Help**. There are many common commands among them but I'll showcase a few that will be useful now and not already available

- **File / Examples**
  - Examples provided by the Arduino Software or other libraries, organized into drop-down tree organized by topic or 3rd party library
- **Edit / Comment/Uncomment**
  - Comment or Uncomment the selected code
- **Tools / Auto Format**
  - Formats the code nicely, lining up indentations based on your use of curly brackets.
- **Tools / Board**
  - Select your board type
- **Tools / Port**
  - Select the port connected to your board from the system available ports

Programming in Arduino

We'll be covering some C++ here as well. Arduino is a variant of C++ so it shares many of the same structures and commands. That means we can take advantage of the overlap with our C++ curriculum and cover both at times. We will still cover some pure C++ in a dedicated C++ development environment later in the semester but for now, understand that a lot of the concepts, commands, and syntax will apply for both. I will try and be explicit when that is **not** the case and something is an Arduino specific command or programming concept.

Also, keep in mind that these pre-labs just introduce elements and concepts. There will be examples to better showcase these elements and we will also go over them in more detail, more specifically about how to best utilize them, in class. So if you feel you don't fully understand an element, you can click on the embedded links for more information but I also recommend waiting until you've gone through the pre-lab to see how much understanding is necessary at this point. You can always come back to something to see if now makes more sense with the full picture in mind.

For the following, it may be helpful to have another example code available as well as `Blink`. Look at the example code `Button` in the Arduino IDE (It has more elements to it than `Blink` so it's a better reference for now). Along the top of the IDE, go to `File / Examples / 01.Digital / Button` to open the example. You **DO NOT** need to implement `Button`, it's just useful to have another script available for reference to showcase all of the Arduino/C++ elements.

Arduino Sketches

Arduino refers to their programs as sketches. As noted, the language is a modified version of C++ that has been somewhat simplified and customized for interfacing with physical hardware. I will provide a context in how Arduino fits within C++ or where the programming languages differ. The Arduino Language itself can be broken into 3-parts: Variables, Functions, and Structures.

- **Variables**
  - Data types and Arduino constants
- **Functions**
  - Control the Arduino board and perform computations
- **Structure**
  - Computational and syntax elements of Arduino (C++) code

We will go over these a bit more in class and introduce components of them as we begin to use them.

Code Breakdown

In most programming languages, the program is not just the script that you write. A program written in C++ actually exists in concert with other elements of the program. It is actually a compilation of various support “Source Files” that a compiler interprets to create an executable program. We will discuss these in more detail later when we work in pure C++ but for now, understand that when you’re actually coding, you're writing a script that will be compiled with other elements to the program to become something the computer can understand and execute. The Arduino IDE handles those other files and compiles them for you.

Every Arduino sketch requires 2 things: **Setup** and **Loop**. There are also **Definitions** and **Custom Functions**, but these are not required. We will get into these later, but I just like to clarify in case you come across these in any personal research you do. A purely C++ script has different requirements, but the Arduino IDE handles those on the backend for you.

Variables

In Arduino, variables are typically established before `setup()`, but can be created in other places. Variables are useful for containing information to be used in your program, even more so if you use descriptive names. When creating a variable though, you need to establish its date type and name.

Here are some key considerations with variables in Arduino. We'll expand upon this as we start using them more often and using more data types:

- Names matter
  - Best practice is to give a descriptive name so when you use it throughout your program, it's clear to you what it's referring to
  - There are a limited amount of allowed characters to use in naming a variable
    - **CAN** consist of both uppercase (`A–Z`) letters, lowercase(`a–z`) letters, and numbers (`0–9`)
    - **CAN NOT** start with a number
    - **CAN NOT** have the same name as Arduino language keywords, e.g. **CAN NOT** name a variable `SETUP`, `LOOP`, or other built-in keywords
    - **CAN NOT** have multiple variables with the same name, but just unique within your script
    - Names are case-sensitive, e.g. `Count` and `count` are different variables
    - **CAN NOT** containt any special character **EXCEPT** the underscore (_).
- To use a variable, you must create it first
  - We declare variables first to create them. We can then use them later in the program but they can't be
- The Arduino IDE has some built-in variables
  - For instance, in the example `Blink` you see `LED_BUILTIN` referenced. This is a built-in variable of the data type `int` and the value `13` because it is associated with the LED built-in to the Arduino board that's attached to Pin 13.
- Data type matters
  - You need to declare the data type of a variable
  - Once you create the data type, you can't change its type later
- Global Variables
  - Variables established outside any function including `setup()` and `void()`
  - Can be used anywhere, hence global
  - Usually established before `setup()`
  - We'll get into local variables later
- `Const` = Constant Value

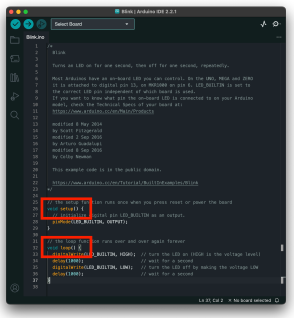
- Used in conjunction with establishing the data type to lock that variable's value so it will not change
- This is an Arduino specific modifier to variables
- Commonly used to ensure variables associated with pin numbers or other unchanging external elements do not accidentally get changed later in your code
- ex: `const int buttonPin = 2` creates a variable `buttonPin` with a constant value `2`. Likely because a button is connected to pin #2 on the microcontroller.

Functions

Arduino are run by establishing functions within the script. There are 2 required functions, `setup()` and `loop()`, but you can create other custom functions. We will discuss custom functions later.

The Setup

The setup does exactly what it sounds like, it **sets up** the sketch with everything it needs. Here is where you define anything you need in the code.



```
1. void setup()
2. {
3.     < setup code >
4. }
```

This will run whatever you have for `< setup code >` once at the start of your script. So right when you apply power to the microcontroller or reset it, this will run. The Setup Code is typically used to set up circuit elements or before the circuit's actual purpose begins. For instance, you only need to tell the microcontroller what's connected to it or establish other system settings just once.

An important thing to note in C++/Arduino syntax is the use of `{...}`. The use of curly brackets denotes a block of related code. In this case, it denotes the code associated with Setup. You'll see this used in other places as well as we progress, but wanted to note this element of syntax early.

You will see the following Arduino command today. As new commands are introduced in the pre-labs or in the lessons, they will be further explained.

For instance, you see the Arduino function `pinMode()` in the example. This is what it does:

- `pinMode(pin, direction)`
  - Sets a microcontroller pin as an `input` or an `output` based on how it is going to handle signals. We are only using it as an output for now.

This is your first instance of a built-in function within Arduino/C++. They take the form of `<Function Name>(<input1>, <input2>, ...)`. You are not privy to the specific underlying code within it, but like other functions, it takes in provided information as inputs included in the parentheses, `pin` and `direction` in this case, and then runs that underlying code based on it. We will also deal with functions that return some information as well. Like with most concepts though, we'll address that when it comes up.

Question 2

How do built-in functions in Arduino/C++ typically operate when invoked?

A	They process the inputs given in parentheses and execute underlying code accordingly.
B	They are directly editable by the programmer to suit different needs.
C	They automatically handle all inputs without needing specification in the code.
D	They require manual integration of their code into the main script.

HINT

EXPLANATION

Show Responses Show Answer

The Loop

The loop similarly is named for what it does. It loops through the code forever, repeating it until the microcontroller is no longer powered. As the program loops consecutively, your code can respond to and control your hardware.

```
1. void loop()
2. {
3. ...
4. }
```

This will run continuously, essentially as part of a `while(1){...}` forever loop. Whatever the circuit is supposed to be doing repeatedly and long term.

In the example *Blink*, you see some other common Arduino functions that we will work with later. They are not limited to use in the loop, but just happend to be only used there in the referenced example.

- `digitalWrite(pin, signal)`
  - Sets pin to a digital signal value, `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
- `delay(number)`
  - Delays for a set number of milliseconds

Structures

These are elements like if, for, and while statements. We will discuss these as they come up in class.

Commenting

Commenting allows for the inclusion of human readable content to inform the user of how the program works. They are ignored by the compiler so they do not effect the program itself. As a tool though, you can use comments to debug a program or try ideas without eliminating the original code.

To comment in C++ and Arduino:

- `//`
  - Comments out one line of code
- `/* ... */`
  - Comments out all lines between the `/*` and `*/`

Semicolon

When coding in c++, every line needs to end with a semi-colon. One of the most common errors you'll come across is when this is forgotten. We will go over some debugging techniques to identify this issue but it's best to just get in the habit to remember this important structural component of programming.

Question 3

In Arduino programming, what is the significance of using the 'const' keyword when establishing a variable?

A	It enables the variable to be used globally throughout the program.
B	It makes the variable's name case-sensitive.
C	It locks the variable's value so that it will not change.
D	It allows a variable name to start with a number.

HINT

EXPLANATION

Show Responses Show Answer

The Hardware

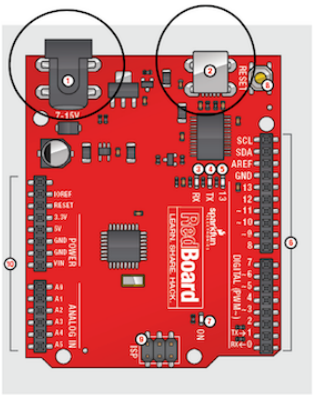
The second half of the system is the Arduino hardware. That includes the microcontroller controlling the system, handling all data and decisions, and the other components that enable the creation of the circuit to take in data and act on those decisions.

Microcontrollers

The microcontroller is the brains of the system. There are 100s of options for microcontrollers. We will be using the RedBoard. It is based on the Arduino Uno , arguably the most common microcontroller currently on the market. The RedBoard is an Arduino Uno clone with the same form fit and function albeit some minor details and a different color.

Power

The RedBoard can be powered by a few different sources. You will typically power it through the USB port though, so having it tethered to your computer through the USB and running off of the 5V 500mA power that provides. If you want it to run untethered from your computer or need more power than the USB, you can also power it through the Barrel Jack.



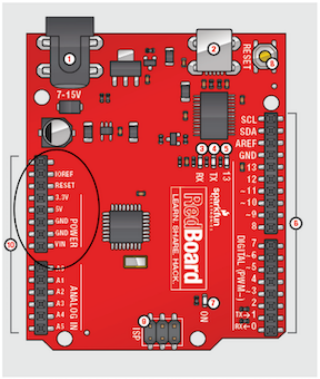
- Barrel Jack (1)
  - Can be used with either a wall-plug power adapter (commonly referred to as a *wall-wart*) or battery back.
- Power In (USB Port) (2)
  - Provides power and communicates with your board when plugged into your computer via USB.

Power Pins

There are several pins you can use to power hardware.

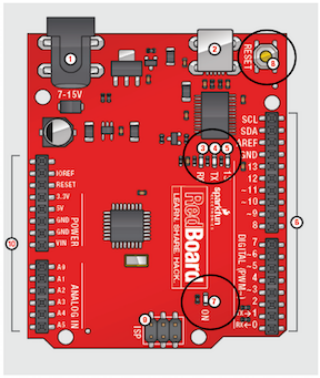
- IOREF
  - Reference voltage for the I/O Pins
- 3.3V

    - Regulated power at 3.3 Volts
- 5V
    - Regulated power at 5V
- GND
    - Connects to the source ground of the power source
    - All ground pins are connected
- VIN
    - Unregulated power equal to voltage of the power source
    - You can also power the RedBoard from this pin if you have an appropriate power source with a jumper wire connection.



Indicators and Reset

There are also several indicator light on the RedBoard (And the Arduino Uno). These can tell you information about whether information is being transmitted to the RedBoard, being returned by it, or even as simple as whether it is powered or not.



- RX(3)
    - Indicates that information is being received(RX) by the RedBoard
- TX(4)
    - Indicates that information is being transmitted(TX) by the RedBoard
- Pin 13 (5)
    - A built-in LED connected to Pin 13
- ON (7)
    - This is a power indicator, engaged directly if a power source is present
- Reset Button (8)
    - This is a way to manually reset your RedBoard, which makes your code restart

Question 4 Show Correct Answer Show Responses

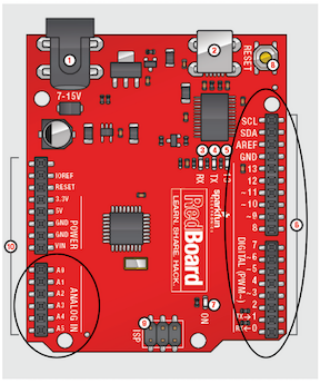
If you wanted to see if your RedBoard was plugged into a power source, which built-in LED would you check?

A	TX

B	RX
C	Pin 13
D	ON

I/O Pins

There are 14 digital Input/Output (I/O) pins and 6 analog I/O pins available on the Arduino Uno style boards like the RedBoard. Arduino does have other boards with more available I/O pins and even some with fewer.



- Digital Pins (6)
  - Used for digital signal inputs and outputs
  - Can be used for controlled power at 5V and ground
  - Can be reconfigured for analog output (We'll explain this concept in a future lesson)
- Analog Pins (10)
  - Used for analog signal inputs
    - We will cover this in the next lab
  - Can be configured to act as digital pins

IMPORTANT: 0 and 1 are also used for information transfer with the computer. If you have them connected to a component while you try to upload your code, you'll get an error since the upload is being interfered with. It is best to avoid using them until absolutely required by your project need and even then there's a trick that I can show you for how to do so.

We will get into the difference between analog and digital signals later.

Question 5 Show Correct Answer Show Responses

How many ground pins are readily available on the RedBoard?

The Other Components

These make up the organs of the circuit. The wires, substrates, sensors, and actuators allow the microcontroller to interact with the real world by taking in data and then triggering actions based upon it. The pre-labs only provide an overview for the components that will be required for use in that Lab so this one will only cover a few. Other components will be introduced and covered in subsequent labs.

Electricity

We are going to be making and controlling electrical circuits in these labs. A basic refresher on electrical concepts should be helpful before we start putting things together. A useful analogy for electricity is plumbing. The flow of electricity through a circuit is very much like the flow of water through pipes. Each electron moving through a wire is like a water molecule moving through a pipe.

Voltage is like the water pressure in the pipe. High voltage means that the electrons are being pushed with high pressure—they really "want" to flow. The most common voltage used with the RedBoard is 5 Volts. This is a very safe, very low voltage, but still enough to operate many small electronic devices. If you start working with higher voltage components, then you will need to ensure you're taking the proper safety precautions.

Current is like the flow rate of the water in the pipe. Larger current means more electrons are flowing through the wire every second. If a pipe is sealed at both ends, then no matter how much we pressurize the water at one end, all of the water will just sit there; no water will flow. We have to open the pipe so that there's a place for the water to go—there must be a difference in pressure from one end to the other. Analogously, the electrons won't move if the entire circuit is at one single voltage; we must "open" the pipe by connecting the ends of a circuit to two different voltages.

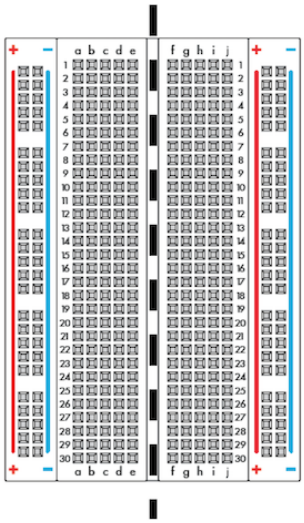
Ground is another name for 0 Volts, and it is like a no-pressure, open end of a pipe where the water drains out to the sea. Your RedBoard provides a few pins with a ground connection. A complete circuit must start at one voltage, like 5 Volts, and end at a different voltage, like ground.

In plumbing, the relationship between the pressure and flow rate is determined by the pipe's diameter. If a pipe has a small diameter, then less water will flow through it. The analogous

electrical quantity is **resistance**. A large resistance is like a very skinny pipe. If something has a large resistance, then less current will flow through it when the same voltage is applied. A wire is like an extremely large diameter pipe; it has extremely low resistance. Resistance is measured in units of Ohms, with symbol Ω.

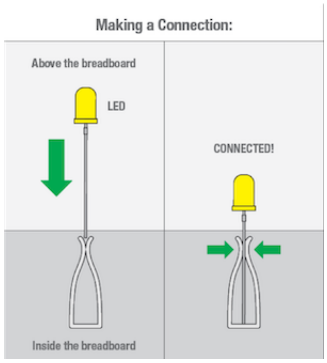
Solderless Breadboards

Solderless Breadboards are one of the most fundamental elements to developing circuits. The name actually originates from prototyping a circuit by hammering nails into a bread board and then wrapping wires around them to connect them. It creates a quick and dirty electrical connection that can be setup, changed, and disconnected quickly. The solderless breadboard allows for the same quick and easy prototyping. You can plug electrical wires and board compatible hardware into the holes. Holes are electrically connected depending on their locations.



- **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 electrically separated by the Center Valley



One of the most common issues people have 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.

For the Pre-Lab, you will use this and you will be given a very clear diagram on what to connect where. In class, we will go over in more detail how these work so you'll be more prepared when you make your own circuits from scratch.

Question 6 Show Correct Answer Show Responses

Multiple answers: Multiple answers are accepted for this question

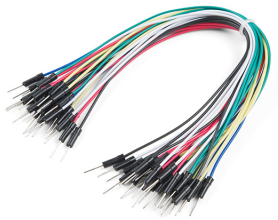
Which of the following are electrically connected?

A	A pin in 1A to a pin in 1G
B	A pin in 12B to a pin in 12E
C	10 pins evenly spaced in the left + column
D	A pin in the right - column to a pin in the right + column
E	A pin anywhere in the left + column to one in the right + column
F	A pin in 8H to a pin in 22H

Jump Wires

A **jump wire** is a length of wire coated with plastic insulation and with a short pin at each end. They are designed to work with a breadboard. Inserting these pins into any two breadboard holes

creates an electrical connection between the two holes. By convention, black wires are often used to make connections to ground and red wires are often used to make connections to power. However, the colors make no electrical difference. Your jump wires may be attached to each other in a strip that can be peeled apart as needed.



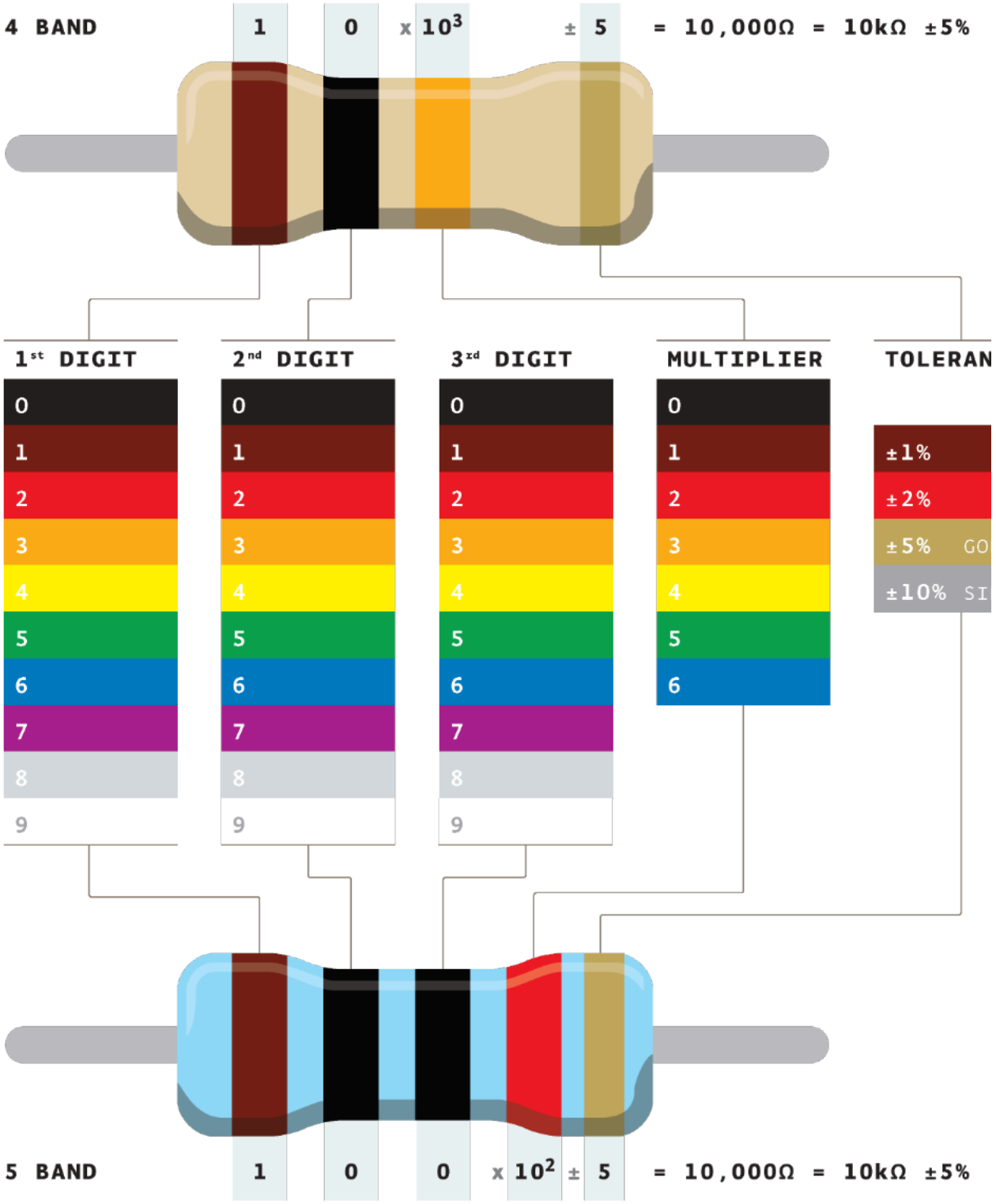
Resistors

Resistors are passive passive electrical component that creates resistance in the flow of electric current, causing voltage drops and serving a variety of purpose through that function. They are typically defined by how they are used and we will go over that as they come up.

Resistors are easy to recognize as being a resistor, usually is cylindrical with a lead coming out either side but there are other forms as well. They are available in wide range of resistance values but your kit has 2 of the most commonly used ones. Knowing the actual resistance of a resistor is a little complicated though. The only way to tell the difference between them is by the markings on them, those colored bands. Make sure to select the correct resistors when instructed. Grabbing a resistor that is too large (10 KOhm) may reduce the voltage enough that it looks like your circuit is not working. If you use a resistor too small (330 Ohm), you may damage other components in the circuit that were relying on there being a minimum level of resistance to reduce the voltage applied across them. Be aware when selecting resistors.

You can determine their resistance based on the color bands. The following chart covers the breakdown of what color means what. Each color band corresponds to a factor in calculating the resistance. For now, we will only use the 2 provided in your kit though so you just need to be able to delineate between the two:

- 330 Ohm - Orange, Orange, Brown, Gold
- 10k Ohm - Brown, Black, Orange, Gold



Question 7

Show Correct Answer Show Responses

Match the resistance to it's color bands

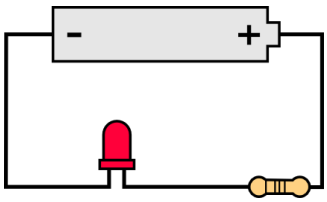
Premise			Response	
1	Orange - Orange - Brown - Gold	→	A	330 Ohm

2	Brown - Black - Orange - Gold	→	B	67 Ohm
3	Red - Green - Yellow - Gold	→	C	250 KOhm
			D	10 KOhm

LEDs

In this lab, we will be setting up and blinking a Light Emitting Diode (LED). They are one of the most common components you will work with. Their name is a direct description of their function. They emit light. It also describes their limitation, they are a diode so polarity matters.

Current Limits



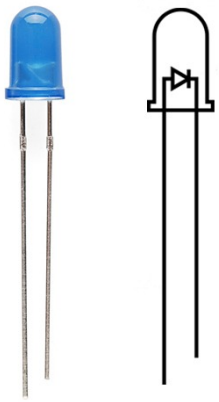
When you see LEDs depicted, they are typically shown with a resistor. LEDs are relatively simple electrical components but they are also capable of burning themselves out if too much voltage is applied. Hooking them directly to a power source may harm it. The best way to avoid this is to calculate the size resistor by determining the specific forward voltage and maximum forward current of the LED. You can also do this with some good sense, experience, and trial & error.

Luckily for us, the SparkFun kit provides the correct size resistors for use with the provided LEDS, **330 Ohm** resistors.

Polarity

Another concern with LEDs is that they are polarized components. This means that it matters which side is connected to ground and which side is connected to the power source. Notice the LED in this

pic or the ones in your kit have different length pins. The longer pin is referred to as the **anode**. This should be connected to the voltage source(**+**). The shorter pin is the **cathode**. It should be connected to the ground(**-**). That side is also marked with a flat edge on the LED itself. The current will run from the anode to the cathode and cannot run in the other direction.

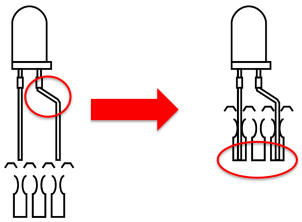


I always remember with the phrase "**Shorter** is closer to **ground"**

If my circuit isn't working, one of the first things I check is the LEDs. Usually flipping that solves the issue. They're very easy to put in backwards accidentally, so just take it out flip it and reverse it.

Working with a Breadboard

You should notice that one wire is longer than the other. If you plug directly into the breadboard without bending them, you may bottom out the long one and not be able to engage the short one. It helps to bend the longer lead wire so that they bottom out at the same time.



Question 8 Show Correct Answer Show Responses

For polar components that will only allow current to move in one direction, the most common reason why they're not working is that you accidentally installed it _____ .

P2L1 - Your First Circuit

Now we are going to create your first Arduino script and circuit. While we do that, we will breakdown an example to better understand what's going on. It will repeat some of what you read above but that's to just reinforce what's happening and better connect it to the circuit you build and its behaviors.

Naming Conventions and Assignment Submissions

Before we start with the example which we'll be renaming, let's go over the naming convention so you can **save early and save often**. There are also several files required for Arduino assignments to capture all of the work. The required files will be described in each assignment but are generally detailed below. They continue to follow the standard naming convention though unless otherwise specified.

Assignment Submissions

For each Arduino based assignment, you will need to submit 3 documents per part:

1. .ino file
    1. The raw code
2. .pdf file
    1. Your pseudocode or flowchart
    2. image of your circuit
    3. a copy of your raw code
3. A video of your working circuit – a .mp4 or .mov file (Or other Canvas viewable formats.)

For the pdf, you can create multiple pdfs then combine them or you can create a document and include the PC/FC and image of your circuits as images, copy in the code, then print the document to a pdf. For the latter option, be aware of format for the code. If you copy it in and lose all the formatting, such as the tabs and the font changes to a non-monosized font, then you're better off printing to pdf and combining pdfs. The point is to make it easiest for the TA to be able to review and assess your work. If you lose the formatting or spread it over multiple files, it makes it harder for the TA to review your work, particularly when they're looking to give you as much credit as they can. If it's hard to review then it's hard to find reasons to provide partial credit.

Naming Conventions

All lab assignments are setup with a similar name: <Subject> <Number of the lab> - <Pre-Lab Lesson or Homework>. Assignments should be submitted using the following naming convention:

<Assignment Topic><Topic Number><Assignment Type><Part number>_<First Initial of your First Name><First 4 Letters of your Last Name>.<Filetype>

For example, Prof. O'Connell would submit the following for this assignment:

- P2L1_BOCON.ino
- P2L1_BOCON.pdf
- P2L1_BOCON.mp4

If you do not submit the correct files using the correct naming convention, the assignment will not be accepted. You will be asked to resubmit correctly. As of the moment of that resubmission request, the assignment is considered 1 day late (in case the error is not noticed until several days later).

Start a new Script

Now let's begin the activity by creating your first script. It doesn't matter that we're building off an existing example, lots of programs start that way.

- Open the built-in Blink example
  - Go to `File / Examples / 01.Basics / Blink`
    - You may already have this open
  - This will open up a new script window for '`Blink`'
- Use 'Save As…' to create your own script
  - Use the correct naming convention
    - `P2L1_...`
  - This will create a file folder ⓘ of that name with a .ino file inside of it with the same name. This is your script.

Add a title block

For all of our scripts and programs, we will need to include information about what it is, who the author is, etc. This is simply a best practice in programming. You can see the example already has

this information. We're going to co-opt it for our purposes.

- Add your title block
  - Replace the current comment block (denoted by `/* */` and should start at the very top of the example) with your own title block. Use the following as a template for creating yours.

```
/*
Programming #<Lab #> <Pre-lab Lesson or Homework> Part <Part # of the Lab>
<First Initial>.<Last Name>
<Name of Script>
<Appropriate credit where credit is due>
<Description of what the program will do.>
*/
```

For this assignment, you should have the following:

```
/*
Programming Lab #2 Lesson Part 1
<First Initial>.<Last Name>
Blinks
Developed from Arduino IDE Sample - Blink
Blinks a pattern via an LED
*/
```

- Save your updated script

Understand the Code

Let's work on Understanding the code in context by breaking down the examples we used to check if your Arduino connection worked. This'll make it easier to work with later. Look at the example code given in the image here, based on the Arduino example `Blink`. It functions exactly the same but i've made a change to help illustrate some elements.

There is a
difference

between your example in Arduino and the version I'm providing here. The built-in example uses `LED_BUILTIN` instead of `LED1`. The variable `LED_BUILTIN` is inherent to the IDE so it does not need to be declared. It will always equal 13 since it's associated with that on-board LED that's connected to pin 13 and labelled as such. I've included const int `LED1 = 13`; just to showcase the use of a variable in Arduino that includes both declaring it as well as using it.

```
const int LED1 = 13;

// the setup function runs once when
you press reset or power the board
void setup()
{
    // initialize digital pin
LED_BUILTIN as an output.
    pinMode(LED1, OUTPUT);
}

// the loop function runs over and over
again forever
void loop() {
  // turn the LED on (HIGH is the
     voltage level)
  digitalWrite(LED1, HIGH);
  // wait for a second
  delay(1000);
  // turn the LED off by making the
     voltage LOW
  digitalWrite(LED1, LOW);
  // wait for a second
  delay(1000);
}
```

CHANGE YOUR SCRIPT to replace the `LED_BUILTIN` with a defined variable. You can use `LED1` as shown in this example.

Variables

`LED1` has been created as a global variable since it's before `setup()`. It can now be used anywhere in the code. It also includes the **Constant** modifier. It makes sense to use a `const` here since the LED is always connected to pin 13 so you will always want `LED1` to be associated with 13.

Functions

`setup()` is used to do just that, setup anything we need in our script. Remember that this will run once. Here, we are telling the Arduino, using pinMode(), that we have something connected to a specific pin and we want to define which direction a signal will run at that pin.

- Initialize digital pin in `setup()`
  - `pinMode(pin, direction)`
    - We're sending out a signal to the LED so it makes sense that it's an `output`
    - We'll use `input` when we get to sensing signals

`loop()` will run repeatedly. The purpose of this program is to blink the LED. So it's going to turn on the LED by sending it a HIGH signal, waiting a period, sending it a LOW signal, waiting a period, then repeating.

- `digitalWrite(pin, signal)`
  - Sets pin to `HIGH` (5V) or `LOW` (0V)

    - When it writes the pin low, no signal (0 V) goes out and the LED is turned off
    - When it writes the pin high, the max signal (5 V) goes out and the LED is turned on
- `delay(number)`
    - Delays for a set number of milliseconds, the equivalent of 1 second in this case
    - Try changing the values then re-uploading to see how this effects the LED behavior

In pseudocode, this would be translated to:

```
PROGRAM Blink:
Setup Pin 13 as an output, used for component LED;
WHILE 1
    Turn LED On;
    wait 1 second;
    Turn LED Off;
    wait 1 second;
ENDWHILE
END
```

The most difficult concept for most students is the delay. Not regarding what it does but rather what it's used for. When you use `digitalWrite()` to turn something off or on, that action takes place in less than 1 millisecond and then it moves on to the next line of code. Try commenting out the second delay (Put `//` in front of it) then run it again. It may look like it stays on but that's not the case. It does turn off but the next line of code, which would be a return to the beginning of `loop()`, is to turn it back on. It's not off for long enough for the human eye to react but it does turn off for that millisecond before being turned back on.

Question 9

Which of the following is a correct statement about the use of delay in Arduino scripting based on the passage?

A	The delay function does not allow the LED to turn off.
B	Without the delay function, the LED appears constantly on due to the speed of script execution.

C	The delay function is used to slow down the speed of the loop.
D	The purpose of delay is to keep the LED on for a longer duration.

HINT

EXPLANATION

Show Responses Show Answer

Create your first Circuit

We are now going to create your first circuit. Do not hesitate to reach out for help from Prof. O'Connell or utilize the Red Vests if you have any issues or concerns.

SAFETY CONCERN!!

We are going to now begin working with the circuit. There is a basic safety practice you should get in the habit of first. At the voltages we're working with, it is more of a **minor** concern for your computer, microcontroller, and the rest of the hardware rather than your own personal safety. But it is a good habit to get into when dealing with circuits, particularly as you begin working at higher voltages where your personal well being is more at risk.

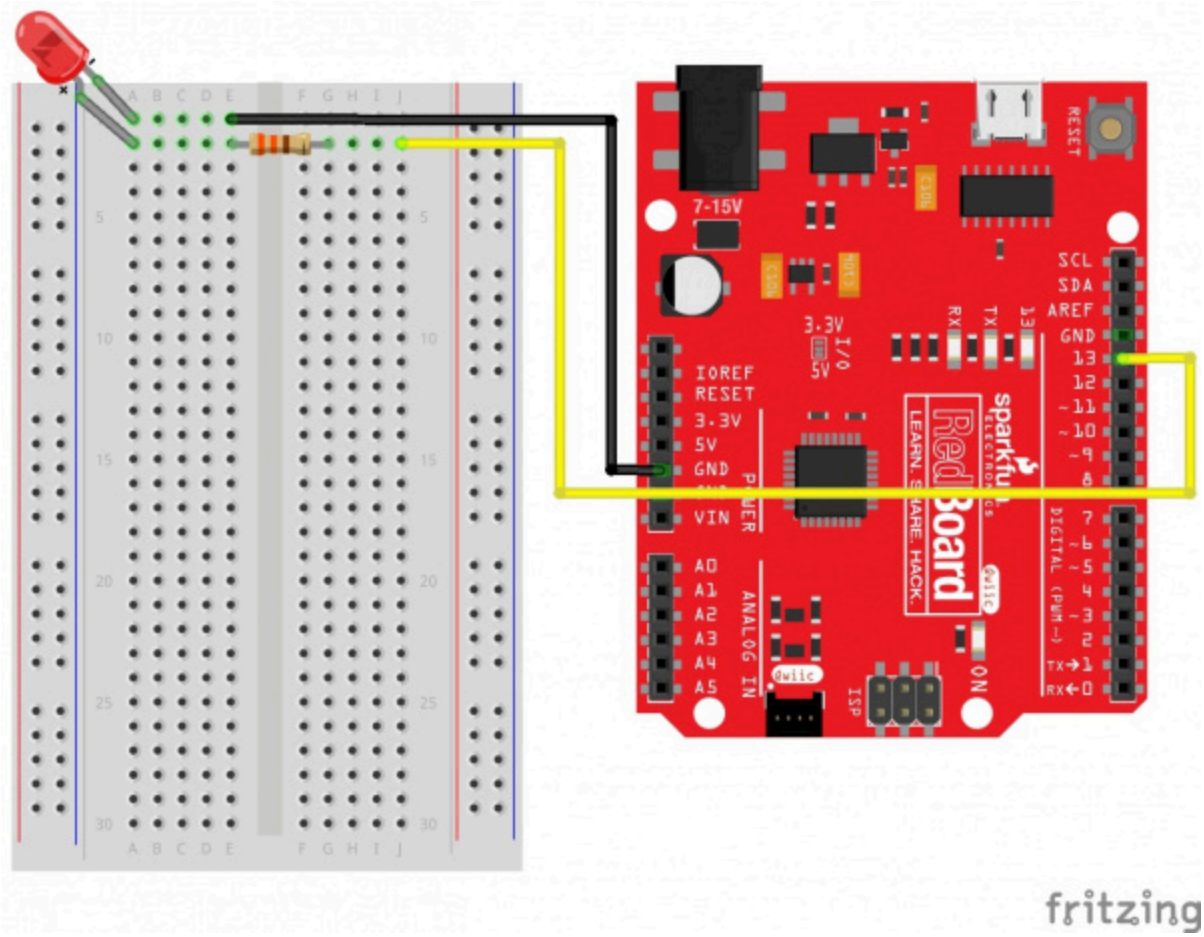
- WHENEVER YOU ARE BUILDING YOUR CIRCUIT, DISCONNECT FROM THE POWER SOURCE WHEN YOU MAKE CHANGES.
  - This will be the USB cable throughout this assignment
  - If you're using a wall power supply (in the future) disconnect that as well
- IF YOU DO NOT, YOU COULD...
  - Short the circuit and trip the breaker in your USB port
    - This may require you to restart your computer but usually unplugging the USB cable, waiting about 10 seconds, and then plugging it back in works to reset it
  - Short a component and burn it out

    - For most of these, that will mean having to replace it from FYELIC
- Short the Arduino
    - This is very uncommon, they're pretty tough. But if you do, it will need to be re-flashed by Prof. O'Connell or someone at FYELIC, if it's recoverable at all.

Question 10 Show Correct Answer Show Responses

Where on the RedBoard do you look to make sure, no matter what program is running on it, that it's not receiving power and therefore now safe to work on adding a circuit to it?

Create the following circuit:



This is the same circuit as the one in your SIK booklet on page 15 or [in their online copy](). I do not typically use the booklet but for this first Arduino activity, I'm using it since that'll mean you have more resources to help you complete it. Keep in mind that the example code they reference is the equivalent of the built-in example **Blink**. We will not use the booklet extensively going forward.

You do not have to have your components in the exact rows shown but for a first attempt it's not a bad idea. You must though:

- **LED**
  - Connect **Pin 13** to a **330 Ohm resistor**
  - Connect the **330 Ohm resistor** to the **Anode** pin of an **LED**
  - Connect the **Cathode** pin of the **LED** to **GND**

REMEMBER, the wire colors don't matter nor do the lengths. You can use whatever jumper wire colors and length you want as long as the electrical connections are made.

Take a photo of your circuit, you'll need it for submitting the assignment.

Run the Script

Your example may already be running on your board so, when you plug it back in, the LED might start blinking already. If not:

- 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, the LED should be blinking. If not:
  - Is the LED backwards?
  - Are the connections sound? Are things supposed to be connected in the same rows?
  - Are you connected to the correct pins on the RedBoard?

Understand the Code

Normally, I start with having you create the circuit then run the script so you can see the behavior as we break down the code. In this case, since the Redboard has that built in LED at Pin 13, we were able to do it in the opposite order. So we can just move on.

Modify the project

We are now going to modify the script slightly to showcase some understanding. We are going to:

- Move the LED to a different pin
- Change the blinking pattern

Move the LED

First let's move the LED to a different pin. Disconnect it from Pin 13 and move it to another one of the digital pins, except for 0 or 1. So pick from Pins 2 through 12 and move it there. Since you moved the physical location of the pin connected to the LED on the circuit, in the script you now need to in the ...

... change the `LED1` value to match the new pin value. Do that and re-upload your script.

Change the blinking pattern

Lastly, we're going to change the blinking pattern. Right now, our loop looks like:

```
// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
    delay(1000);                      // wait for a second
    digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
    delay(1000);                      // wait for a second
}
```

I want you to add at least 2 blinks to the pattern. It can be anything. Maybe a short blink, a long blink then repeat. You can even get it to blink SOS in morse code repeatedly, which would be 3 short blinks, 3 long blinks, 3 short blinks then repeat (`... ––– ...`), so 9 blinks in total, Or something longer like:

`–– .– –.–– / – .... . / ..–. ––– .–. –.–. . / –... . / .–– .. – .... / –.–– ––– ..–` (Brownie points for deciphering this)

So your code would become, if we were just going to add a single blink, something like this:

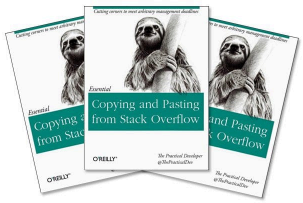
```
// the loop function runs over and over again forever
void loop() {
    // Short Blink - dot
    digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
    delay(500);                      // wait for .5 seconds
    digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
    delay(500);                      // wait for .5 seconds

    // Long Blink - dash
```

```
10.     digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the
    voltage level)
11.     delay(1000);                      // wait for a second
12.     digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making
    the voltage LOW
13.     delay(2000);                      // wait for 2 seconds
14. }
```

I just copied those 4 lines of code (even included a linebreak in between to showcase that) and then changed it to shorten the first blink. I also made the last delay longer to emphasize the repeating pattern. Yours shouldn't be exactly like this and should include at least 1 more blink. Anything more is just to challenge yourself and experiment further.

Note that I also **included descriptive comments** before the segments of code to clarify what they're doing.

You can learn a lot about Arduino, and about programming in general, by taking an example or a subset of one and then modifying it to see how your changes work. A lot of programming is just the copying and repurposing of older code, either existing examples or projects of your own.



Now go ahead and try it yourself. You can do it!

Requirements:

- A blinking pattern of at least 3 blinks
  - So your loop should be at least 12 lines of code, 4 for each blink.
  - Yours doesn't have to be morse code, I just did it as something to latch onto for a pattern. Here's a morse code translator is you're interested but limit yourself to your initials or something else short - https://morsecode.world/international/translator.html.
- Must include comments to indicate
  - Single line comments proceeding a segment of code to say what it's supposed to do
  - What the intended behavior of each line is
    - This is a bit of over-commenting for now but it's to start building habits

Be aware of your time. The requirement is only a 3 blink pattern. If you want to do something longer, that's fine, but keep in mind your time as you do.

Requirements:

- A blinking pattern of at least 3 blinks
  - So your loop should be at least 12 lines of code, 4 for each blink.
- Must include comments to indicate
  - What the intended behavior of each line is
  - This is a bit of over-commenting for now but it's to start building habits

Be aware of your time. The requirement is only a 3 blink pattern. If you want to do something longer, that's fine, but keep in mind your time as you do.

Submit your first Arduino Assignment

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 to review 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