Iris E-course

Innovation through Education

# Introduction to the Iris Robot Platform

Presented By:

Higher Order Innovation

www.higherorderinnovation.com

**Authors: Tom Frederick, Kristin Frederick, Lauren Reinsvold**

Updated November 1, 2021

# Table of Contents

# Preface

Before jumping into robotics and engineering, we would like to take a minute to describe the authors' vision for this ebook. This Iris e-course is not meant to be a stand-alone curriculum, but rather a guide: ***a guide to help teachers teach and to help students learn.***

Robotics education can be intimidating. Our goal is to make robotics education more accessible for both teachers and students. Teachers need not be engineers and students need not be future engineers. In our experience, the best teachers and coaches are ones who facilitate learning by providing resources to students rather than only providing answers. Giving students exposure to the vast field of robotics and opportunities to actively engage is more important than a teacher who knows all the ins-and-outs of specific robotics concepts.

The integration of multiple STEM topics into one cohesive course is a daunting, but important, task. We want to show students that the disciplines of applied physics, mechanical engineering, computer programming, electrical engineering, mechatronics, and robotics are all interconnected. To do this without overwhelming both students and teachers, for every chapter we have chosen one concept as the background information to provide context. There are hundreds of equally important concepts relevant in the world of robotics. ***The Iris ebook is our attempt to provide one way to teach robotics; this is by no means the only way to teach using the Iris Robot Platform.***

Rather than write our own curriculum for the majority of the background sections, we have organized a list of websites, online tutorials, and links to lessons that we recommend. We hope this decreases the amount of time teachers need to spend sifting through all of the resources available on the Internet. We will do our best to keep the links in this ebook updated but we encourage teachers and parents to test the links for themselves before offering this ebook to students. Please check our website (**www.higherorderinnovation.com**) periodically to make sure you have the most updated version of this ebook.

# Introduction

Higher Order Innovation was founded to increase STEM education opportunities by supporting robotics programs at elementary through high school levels. Witnessing the creativity, exhilaration, and dedication of students at competitive robotics events has always renewed our passion for pushing the limits of STEM education events. Our products have been designed with educational robotics competitions in mind but can be used in more generic settings such as classrooms, extracurricular programs, continuing education workshops, and much more.

The core principle of Higher Order Innovation is to provide the expertise and capabilities for everyone to be innovative in their own way. We believe in the notion that truly novel creation is the result of thinking on a level higher than mere manufacturing and that the future of industry and growth is based on how we educate our students. By using our background and experience in mechanical, electrical, and software engineering, we are working to bring the art of innovation to people of all ages. ***Innovation through education*** is what drives us to accomplish this mission.

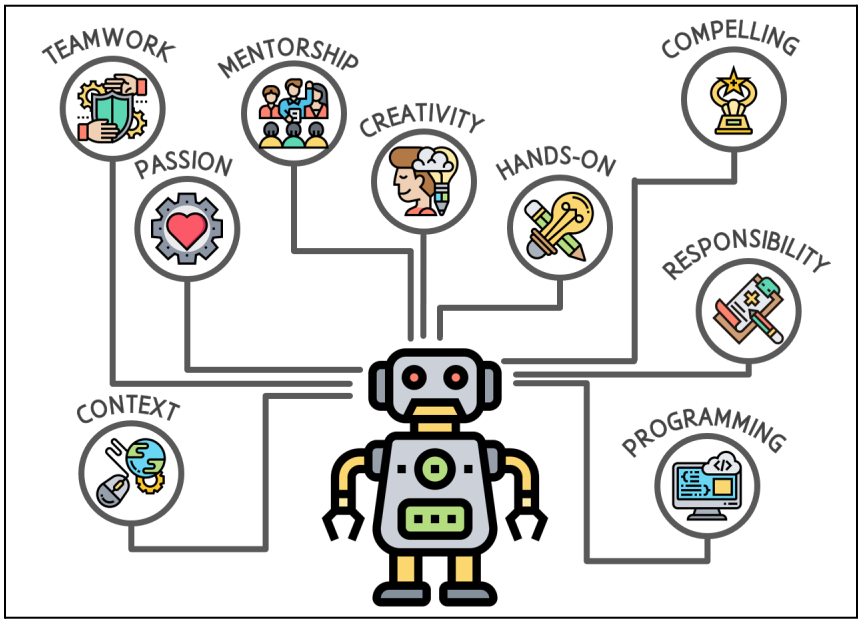


***Figure 1: Benefits of competitive robotics***

The benefits of competitive robotics programs extend far beyond the classroom, as shown in Figure 1. Robotics makes STEM accessible by providing a ***context*** for physics and computer science concepts. Many students can, and should, work together on a single robot as they learn ***teamwork*** skills and to rely on each other's strengths. Students can find new ***passions***, whether it is different types of engineering or computer skills. Teams are the perfect environment for ***mentorship*** from professionals and older robotics students. It encourages ***creativity*** and the development of problem-solving skills. Building a robot forces a ***hands-on*** learning approach and can be instrumental in increasing comprehension. Robotics tournaments are ***compelling*** competitions that can engage students who may otherwise lean towards sports in favor of academics. These competitions require the teams to be prepared, teaching

***responsibility*** and flexibility as situations arise. Robotics education also teaches computer ***programming*** skills that are becoming increasingly important in all aspects of technology.

The **Iris Robot Platform** is a great introduction to robotics and the perfect challenge for anyone's next project. It is small enough to drive around on your desk while powerful enough to drive throughout the room. Controlling the Iris Robot Platform through an Android app allows the robot to explore the room and accomplish tasks. For many people, bringing software into the physical world through hardware can make programming easier to learn and easier to remember. It is a low cost, mobile robot platform that can be programmed to operate on its own or in response to a wide variety of remote controls.

Our goal with our curriculum, **Iris E-courses**, is to maximize learning opportunities at multiple levels. The Iris Robot Platform is an engaging introduction to microprocessor programming, mobile app development, as well as the potential expansion into web development. It is ideal for teaching programming: the robot is palm sized and can be operated and tested on the desk next to the computer. Engaging users with programming can be difficult at times, but through the mobile robotic platform, programming methods and techniques will keep the engagement higher for longer.

This ebook is specifically the **Introduction to Iris Robot Platform**. As you can see on our website **www.higherorderinnovation.com**, we have a variety of ebooks and tutorials available which we have grouped together as the Iris E-course. This ebook is the first in the series and lays the foundation. In this ebook you will learn about the components on the Iris Robot Platform and walk through an example Arduino program for each chapter.

Iris has been designed so students and teachers can focus on programming without having to build the electrical and mechanical systems. However, we believe programming should not be taught in a vacuum. This curriculum has been intentionally set up to learn physics and engineering principles alongside programming principles. Many robotics curricula teach how to use a single robotics platform; this ebook aims to show where robotics fits in the wider context of science and technology. Each chapter has four distinct sections. The first section offers an introduction to relevant physics topics. The second section builds the foundation even further by delving into the hardware on Iris and basic electrical engineering concepts. The third section teaches programming fundamentals through example programs and libraries. The fourth section gives students an opportunity to synthesize the hardware, software, and physics topics by presenting a challenge topic.

Higher Order Innovation wants you to learn the "how" as well as "why" questions by integrating foundational concepts of physics with software concepts. The field of robotics is ever-changing as technology continues to evolve. The main objective of this Iris E-course curriculum is more than to simply learn how to program the Iris Robot Platform: robotics education can and should be much more than that. The logic and reasoning skills necessary for robotics are invaluable regardless of future career. We hope this curriculum will foster creativity and out-of-the-box thinking for all participants.

Let's get started!

# Chapter 1: Getting started with Iris and Arduino

## 1.1 The Iris Robot Platform and Arduino Electronics Platforms

The Iris Robot Platform was created by Higher Order Innovation specifically with education in mind. It is a combination of an Arduino-compatible microcontroller, built-in sensors, and motors designed to minimize the barrier of entry into robotics. Arduino is an open-source electronics platform that is freely available for download on Mac OS X, Windows, Linux, and ChromeOS, or online via a web-based editor. The Arduino programming language is how the inputs from the Iris sensors are transformed into outputs from the robot, such as changing the LED lights or moving the robot.

Thousands of forums, tutorials, and resources are available by the vast number of people worldwide using Arduino. This curriculum contains all the information necessary for the example programs provided with the Iris Robot Platform; however it is our hope that students utilize additional resources as they continue to explore the vast possibilities of Iris and Arduino.

For more information regarding the Arduino Electronics Platform: **http://www.arduino.cc**

## 1.2 Overview of the Iris Robot Platform

One of the most engaging methods of teaching programming is through interactions with the real world. This can be achieved through the lens of a camera and computer vision algorithms or through a robotic platform such as Iris. The transformation of code into the physical brings life to the topics in a way that purely computer based methods cannot. The Iris Robot Platform was designed to introduce as many interesting and useful components as possible while achieving that goal. These components allow the robot to sense its environment and respond, all while students learn programming concepts.

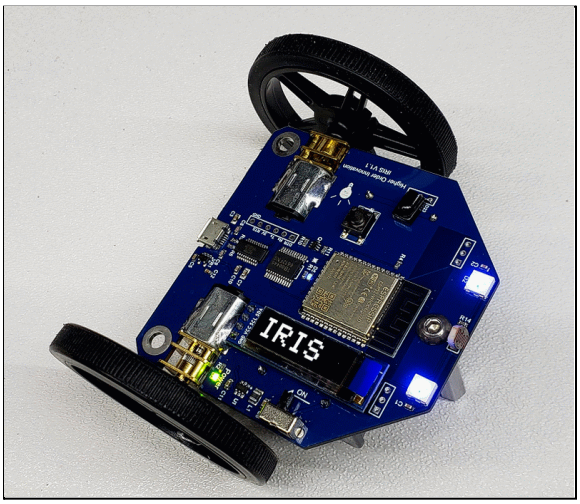


***Figure 2: Iris Robot Platform with LEDs***

At first glance, the Iris Robot Platform looks like a simple circuit board with wheels. Iris contains the following components:

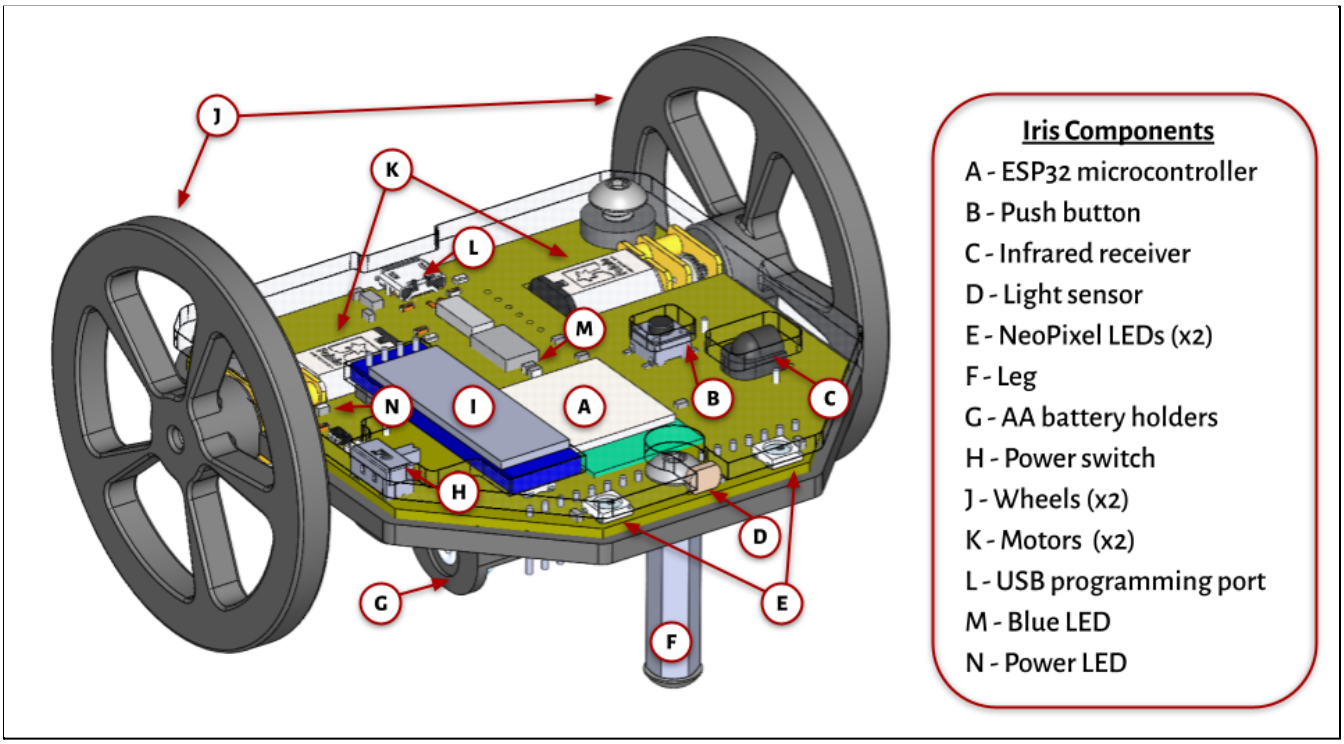


***Figure 3: Iris Robot Platform component list***

A. **ESP32 microcontroller** - This is the brain of the robot and can be reprogrammed to control all parts of the robot. Bluetooth, Wifi, and IR communication capabilities enable remote control through a variety of options such as a mobile phone or standard internet browser.
B. **Push button** - When this is pressed, a HIGH/LOW signal change can be registered by the microcontroller.
C. **Infrared receiver** - Most TVs and stereos use infrared (IR) remotes which can be used to control the Iris robot using the IR decoder.
D. **Light sensor** - This analog sensor can be read by the microcontroller as a signal between 0 and 4095. It can detect bright lights or shadows and everything in between.
E. **Neopixel LEDs** - These special components easily allow control of the two LEDs separately, changing their precise color and brightness while only using one pin to communicate.
F. **Leg** - A solid leg holds up the front of the robot and allows it to glide across the surface.
G. **AA battery holders** - Two AA batteries power the robot when USB is not attached. Motors are directly powered from the batteries.
H. **Power switch** - This switch turns the power on/off for the robot and controls the power to the motors when the USB cable is plugged in.
I. **OLED Screen** - Display pictures and/or text as a way to communicate with the robot when it is not directly plugged into the computer.
J. **Wheels** - These rubber tread tires help the robot grip its surface.

K. **Motors** - Two brushed-DC gear motors move the robot around.
L. **USB programming port** - This port powers everything except the motors and allows custom programs to be loaded onto the robot.
M. **Blue LED** - This blue light in the center of the robot can be controlled by Arduino programming.
N. **Power LED** - This light will turn on when the robot is powered either through USB or battery power. Remember to turn off the robot to save batteries.

## 1.3 Assembly of the Iris robot

The Iris Robot Platform kit includes the following pieces:

1. (1) Iris Robot printed circuit board
2. (2) wheels
3. (1) aluminum hex standoff
4. (2) UNC 8-32 bolts
5. (1) USB micro power and programming cable
6. (2) rechargeable AA batteries
7. (1) AA battery charger
8. (1) Infrared remote

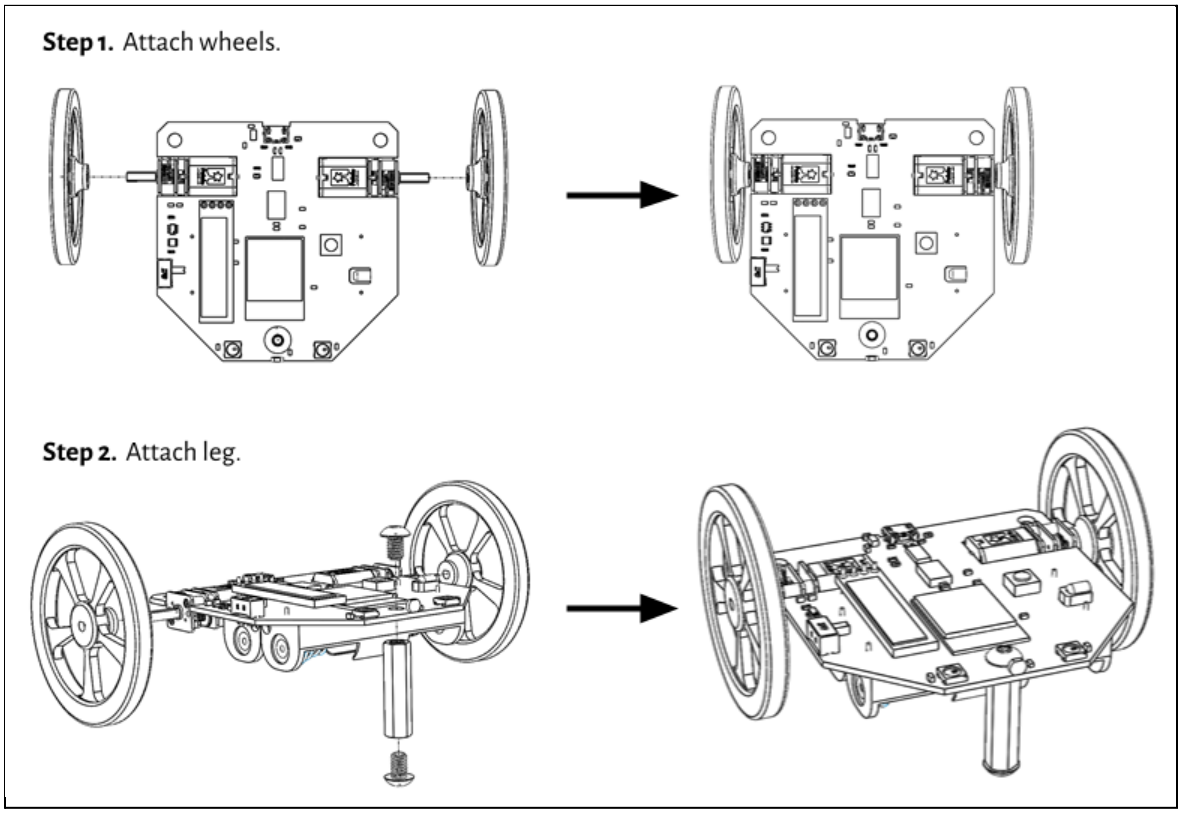


***Figure 4: Assembly of the Iris Robot***

The Iris robot comes with its motors and sensors already attached. Assembly consists of three simple steps:

1. **Attach wheels** - Align the D shape of the motor and the cutout in the wheel. The flat surface of the wheel should be facing away from the body of the robot. Holding the motor in one hand and the wheel in the other, press the wheel into place until the hub is nearly touching the motor.

2. **Attach leg** - Two (2) UNC 8-32 bolts and one (1) aluminum hex threaded standoff are included. One bolt passes, from the top, through the single hole at the front of the robot. The threaded standoff then twists on, locking this bolt in place. The second bolt can be threaded into the bottom of the standoff, providing a slightly rounded front "leg" which will drag across the surface as it moves. Explore using different materials at the bottom of this leg to see how the interaction affects the movement of the robot.

3. **Power the robot** - There are two methods of powering the robot, each with its own purpose. Power using the USB micro cable will provide power to everything except for the motors whenever the cable is plugged in. This method can be used for exploring the board, interacting with the sensors, and testing code. The second method for powering Iris is the AA batteries. When installed, these batteries can be switched ON/OFF with the power switch. When ON, these batteries provide power to everything on the board, including the motors. Both batteries and USB can be used at the same time.

## 1.4 Setting up Arduino platform

For users who are familiar with Arduino, we suggest skipping to step 3 of the set-up process described below.

**Step 1: Install the Arduino Integrated Development Environment (Arduino IDE).**

- An *Integrated Development Environment (IDE)* is a software application specifically built for computer programming. The Arduino IDE is what you will use to program the software for the circuit board on Iris. There are two options: the online IDE or the desktop IDE.
- Go to the following link to learn more about IDE: **https://www.codecademy.com/articles/what-is-an-ide**
- For the online IDE you will use the Arduino Web Editor at **https://create.arduino.cc/editor**. You will need to create an account with Arduino in order to save your programs.
- The desktop IDE can be downloaded from the following link: **https://www.arduino.cc/en/software**. We recommend installing a desktop version of the Arduino IDE so you do not need to worry about a constant and reliable internet connection while programming. Arduino Desktop IDE is available for Windows, Mac OS X, Linux, and ChromeOS operating systems. Please note this ebook was written using the desktop Arduino IDE version 1.8. Arduino IDE version 2.0 may be available at the time you are reading this ebook, but all these instructions were written with version 1.8 in mind.
- More specific information regarding each of these versions can be found at: **https://www.arduino.cc/en/Guide**.

**Step 2: Learn the basics of Arduino IDE.**

- We suggest reading the Getting Started Guide on the Arduino website: **https://www.arduino.cc/en/Guide/Environment**.
- At this time focus on how to use the Arduino IDE software, not necessarily the hardware of the Iris microcontroller board or the programming language. Make sure you understand the basics such as:
  - ✓ What is a *sketch*?
  - ✓ How do you save your sketches in a *sketchbook*?
  - ✓ How do you edit sketches?
  - ✓ How do you upload a sketch to the board? Note: before attempting to upload, make sure you have finished steps 3 and 4 below and have selected the appropriate board and computer port.
- As you progress through this Iris curriculum, we encourage you to think outside-the-box as you come up with solutions to the challenges presented in each chapter. If you know what you want to do but aren't quite sure how to do it, we suggest the Arduino forum: **https://forum.arduino.cc/**.

**Step 3: Install the Iris board cores.**

- In the world of Arduino programming, *board cores* are the link between the components on the circuit board (specifically the microcontroller) and the programs you write in the Arduino IDE. Before beginning any new project you need to make sure the Arduino IDE is correctly set up for the board you are using.
- Click on Tools > Boards > Board Manager to get more information on the board core. After you download the Arduino IDE the default board core will be set to Arduino Uno.
- To install a core read the section "How to install a third party core" from the Arduino website: **https://www.arduino.cc/en/Guide/Cores**
- The Iris board uses the Sparkfun ESP32 microcontroller core, which requires two board URLs that need to be installed prior to successful programming. Go to File > Preferences > Additional Boards Manager URLs to add the two links below. Select the icon circled in red to add both links, one per line:
  - ➢ **https://raw.githubusercontent.com/sparkfun/Arduino_Boards/master/IDE_Board_Manager/package_sparkfun_index.json** (Make sure you correctly copy this link from both lines without any extra characters or spaces in between)
  - ➢ **https://dl.espressif.com/dl/package_esp32_index.json**

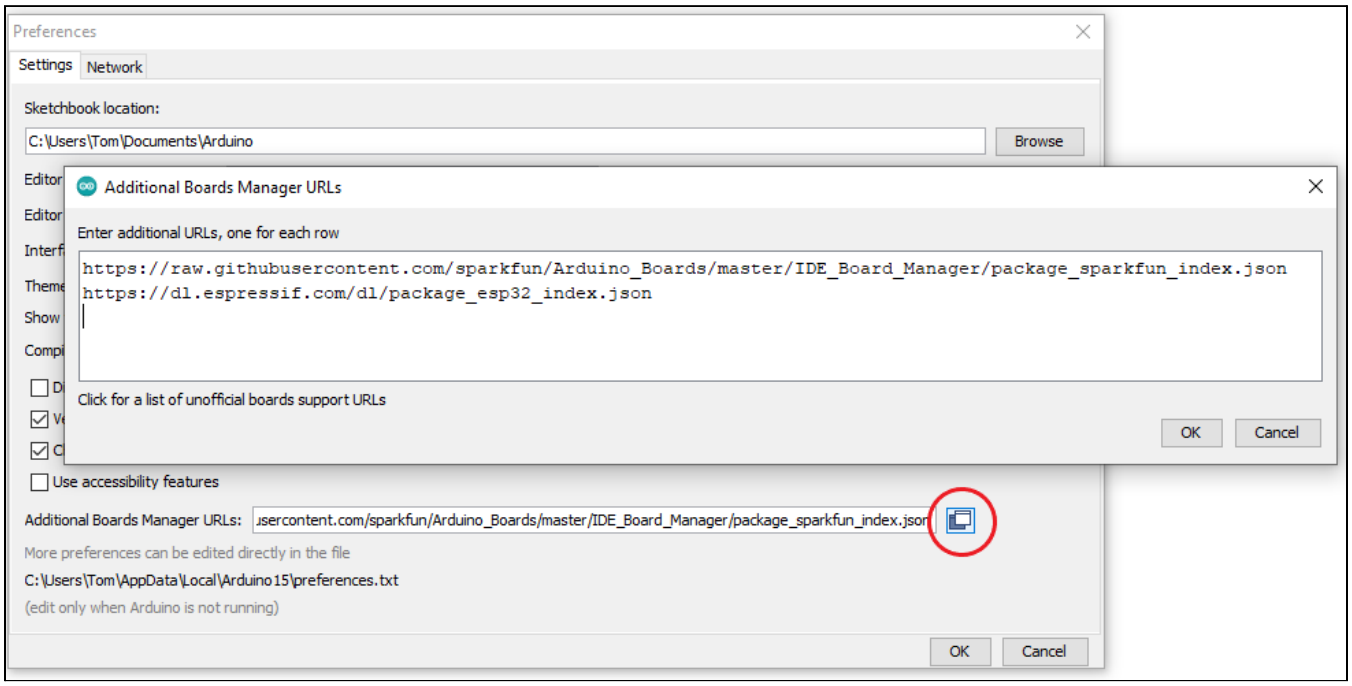


***Figure 5: Additional boards manager URLs screenshot***

- After adding the Additional Board URLs, go to Tools>Boards>Boards Manager and search for ESP32. Install the Sparkfun ESP32 Boards and esp32 options as shown in Figure 6 below. Click the Install button next to each of the two boards.

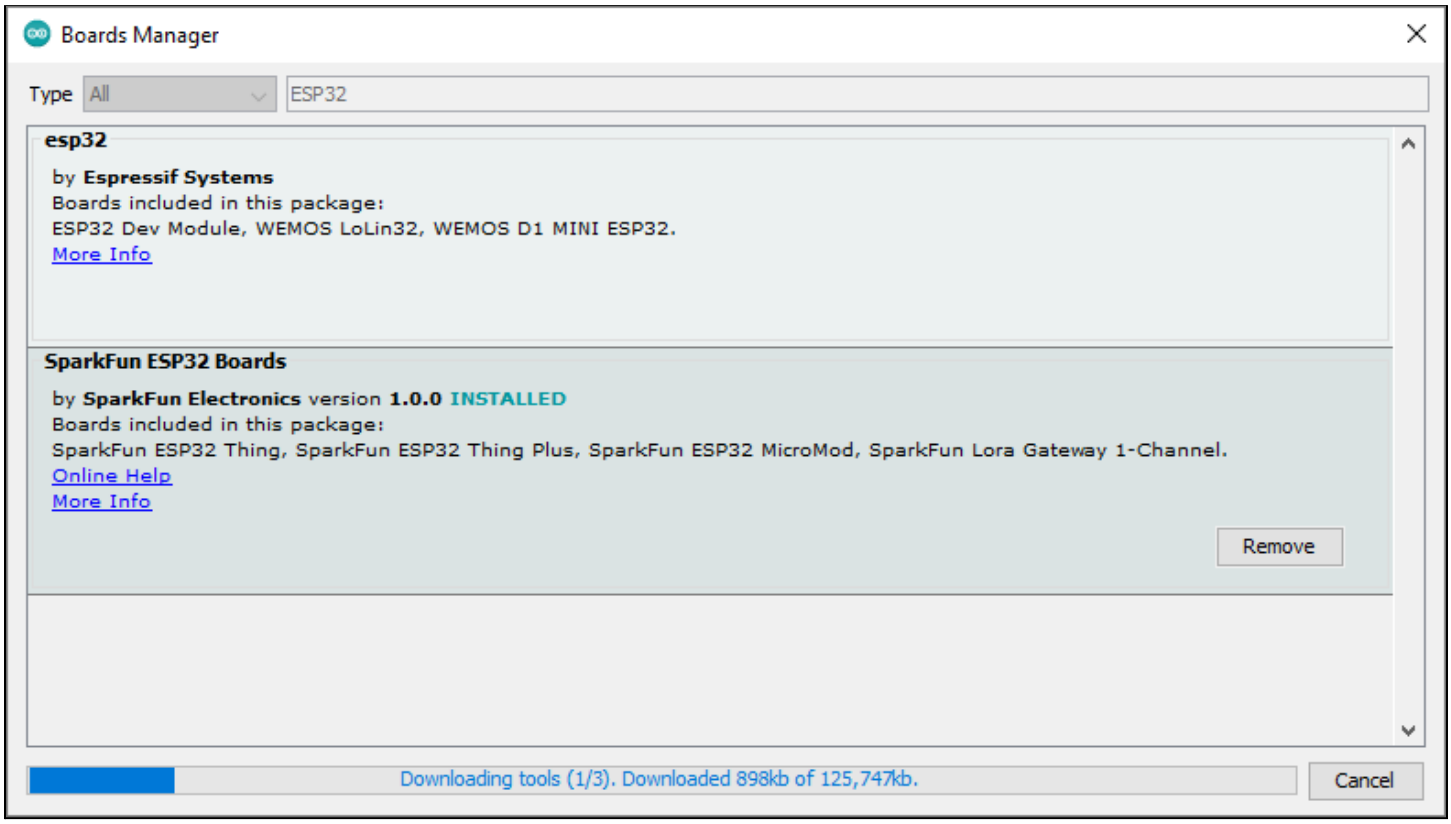

***Figure 6: Arduino boards manager screenshot***

- To program the Iris robot, navigate to Tools>Board>Sparkfun ESP32 Arduino and select the "Sparkfun ESP32 Thing". This selection will tell your Arduino Development Environment the correct way to communicate with your robot.

**Step 4: Choose the correct port on your computer.**

- The *computer port* is how the computer communicates with the Iris board when the Iris board is plugged in via the USB cord. Computers often have multiple ports so make sure you have the correct connection between the board and the computer.
- The small end of the USB micro cable plugs into the port on the back of Iris (labeled L in Figure 3) and the large end of the USB micro cable plugs into your computer.
- For more information on choosing the correct port on your computer: **https://www.arduino.cc/en/Guide/Environment#uploading**

**Step 5: Install the Iris library.**

- A *library* in software programming is a set of prewritten code for a distinct purpose in order to help your program run more efficiently. The Arduino IDE already has a number of libraries installed to perform tasks such as connecting to the default Arduino Uno board, processing and storing data, and controlling certain types of motors. The Iris library contains Arduino code that has been optimized to the specific components on the Iris circuit board.
- For more information on Arduino libraries: **https://www.arduino.cc/reference/en/libraries/**
- For more information on how to install new libraries: **https://www.arduino.cc/en/Guide/Libraries**
- To install the Iris library:
    1. Go to **https://github.com/HigherOrderInnovation/IRIS_robot**
    2. Click on Code (green button) and download a zip file
    3. In Arduino go to Sketch->Include Library->Add .ZIP Library. The ZIP file will be named IRIS_robot_master.zip

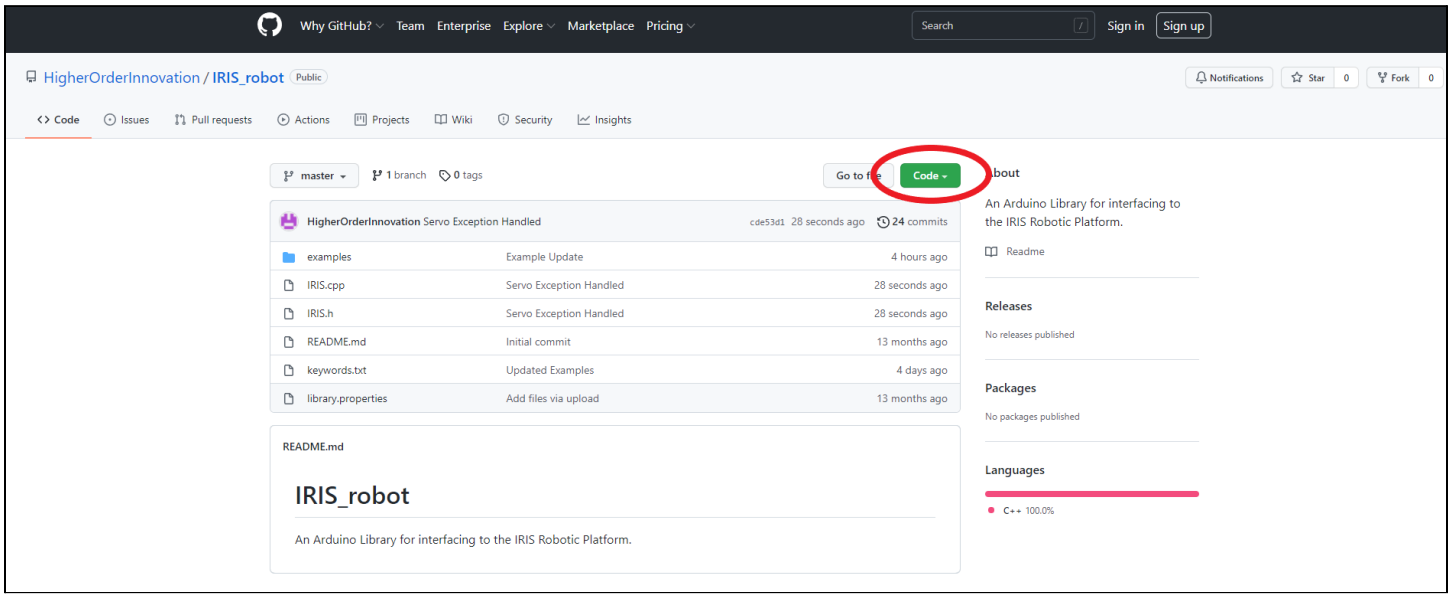


***Figure 7: Iris library GitHub screenshot***

- The Iris library contains the example programs that will be used in this curriculum. To open an example program go to File > Examples > IRIS Robot. Note you will need to scroll down to Examples from Custom Libraries to find the Iris example programs.

**Step 6: Start your first project!**

- Each of the remaining chapters of this curriculum contains an example program and programming challenge. The knowledge acquired is cumulative; we do not recommend skipping a chapter without making sure you thoroughly understand the information presented. The programming skills, physics principles, and engineering concepts will build upon themselves as you continue through the curriculum.
- Table 1 contains an overview of the chapters, example programs provided, and the purpose of each software program.

***Table 1: Overview of example programs***

| Chapter and Code | Goal |
|---|---|
| Ch02-Blink | Blink the on-board LED. |
| Ch02-Blink-Challenge | Modify the duration of the ON period of the LED |

| | |
|---|---|
| Ch03-Button | Perform an action when the button is pressed |
| Ch03-Button-Challenge | Measure the length of a button press |
| Ch04-Light | Read the light intensity of the environment |
| Ch04-Light-Challenge | Modify a blinking LED with respect to environmental lighting |
| Ch05-Screen | Display text on the on-board screen |
| Ch05-Screen-Challenge | Display dynamic information to the user on the on-board screen |
| Ch06-Neopixel | Fade a single color in and out |
| Ch06-NeoPixel-Challenge | Fade through all of the colors in a loop |
| Ch07-Motors | Make the robot move |
| Ch07-Motors-Challenge | Drive the robot in a square pattern |
| Ch08-WiFi | Create a wireless access point and use a web browser to control the LED |
| Ch08-WiFi-Challenge | Control the NeoPixels through the web interface |
| Ch09-Infrared | Read the signals being sent by the remote control. |
| Ch09-Infrared-Challenge | Drive your robot with the remote |
| Ch10-Bluetooth | Connect the robot to the Android app to control the LED |
| Ch10-Bluetooth-Challenge | Use the slider on the Android app to control Neopixel brightness |

## 1.5 Basics of Arduino programming language

In the Arduino programming language a program is called a *sketch*. Sketches consist of three pieces: *functions*, *variables*, and *structure*.

*Functions* are how you interact with the board; think of them like a set of instructions and rules for the board. Each function is a piece of code that performs a defined task. You can write functions to perform mathematical operations, communicate with the board by reading or writing to specific pins on the board, as well as many other types of computations. Libraries contain functions so you don't have to define and create every function every time.

- ➢ Read this for more information on how to use and create functions: **https://www.arduino.cc/en/Reference/FunctionDeclaration**

*Variables* are how you store pieces of information (also known as data) in the sketch. The variable is defined by the type of data, a name, and what data it contains (value).

- ➢ Read this for more information on how to define variables: **https://www.arduino.cc/en/Tutorial/Foundations/Variables**

*Structure* components are how the functions and variables work together to create the sketch. One type of structure is mathematical operators such as * for multiplication and + for addition. Another type of structure are comparison operators such as < for less than, <= for less than or equal to, and != for not equal to. Other structure components define the rules for the code, known as syntax. An example of syntax would be curly braces {} used to define functions. The screenshot shown in Figure 8 below outlines what your Arduino IDE window will look like once you have gone through all of the set-up and opened up the first example program:

A. **Upload button** - Hit the right arrow button to upload the code to your Iris robot to run the program. You will need to use this button after every time you change the code to reupload to the Iris board.
B. **Library** - Make sure you downloaded the Iris library otherwise this line of code will fail. See section 1.4 step 5 for more information.
C. **Board core** - Needs to be set for SparkFun ESP32 Thing. See section 1.4 step 3 for more information.
D. **Compiling parameters** - Leave these at the default configuration.
E. **Computer port** - This may be different for every computer; for this example it is set to COM3. Your Iris robot will need to be plugged in. See section 1.4 step 4 for more information.
F. **Serial monitor** - This magnifying glass button opens up the serial terminal monitor. You will learn about this in chapter 3.

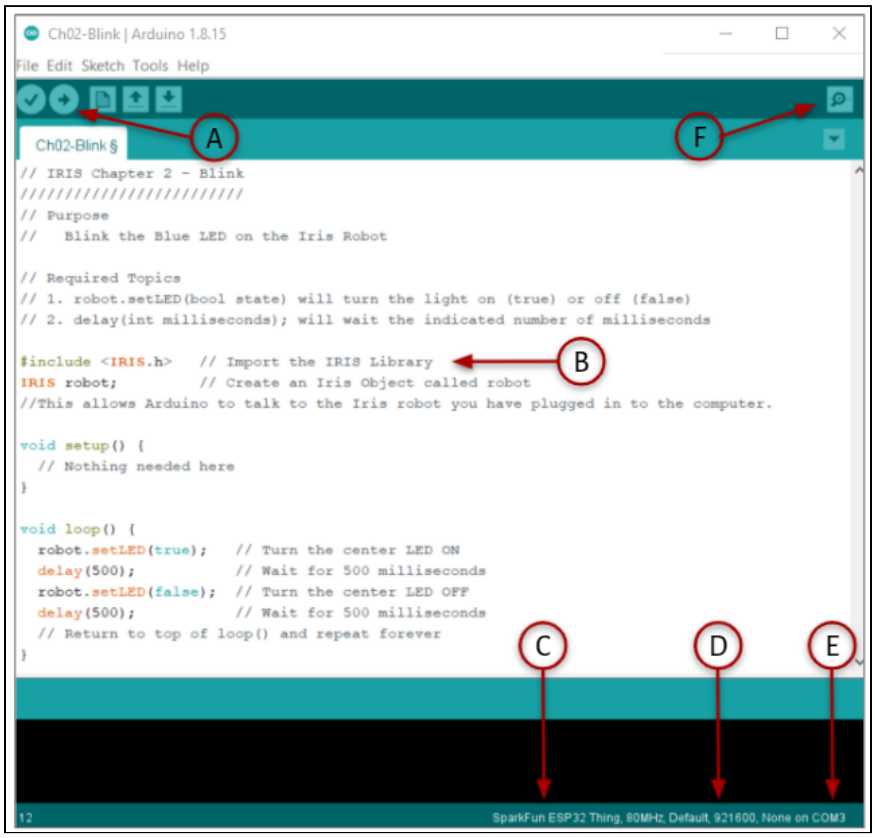


***Figure 8: Arduino IDE screenshot***

Each coding section of this ebook will begin with a list of the new structures, functions, and value types that are introduced in the example code.

# Chapter 2: Blink

## 2.1 Circuits and Ohm's Law

Before we dive too deep into electrical engineering, mechatronics, and the hardware on the Iris board, it is important to have a firm grasp on circuits. Work through the following tutorial to make sure you understand how to read a basic circuit diagram:

- Sparkfun circuits tutorial: **https://learn.sparkfun.com/tutorials/what-is-a-circuit/**

Electrical circuits are connected in a continuous path, much like the complete loop of a running track circuit. The term "short-circuit" refers to instances when that path is connected in a way that was not intended because the new path is shorter than the intended path. While circuits come in many different shapes and sizes, they all follow the same principles of having complete paths for the electricity to flow. Many times the first version of a circuit is done on a breadboard and large wires. This tests the design and allows a person to hand-build the circuit.

Once the circuit is proven, that circuit can be converted into a digital design on the computer to produce a Printed Circuit Board (PCB). Whereas the wires in a breadboard are large and looping, PCBs use thin layers of copper to carry the electricity around the board. The Iris robot is built using a 2-layer circuit board (top and bottom), but more complex designs like a computer motherboard can be made up of more than 8 layers. Each layer carries the copper wires (traces) and the extra layers allow designs to prevent short-circuits.

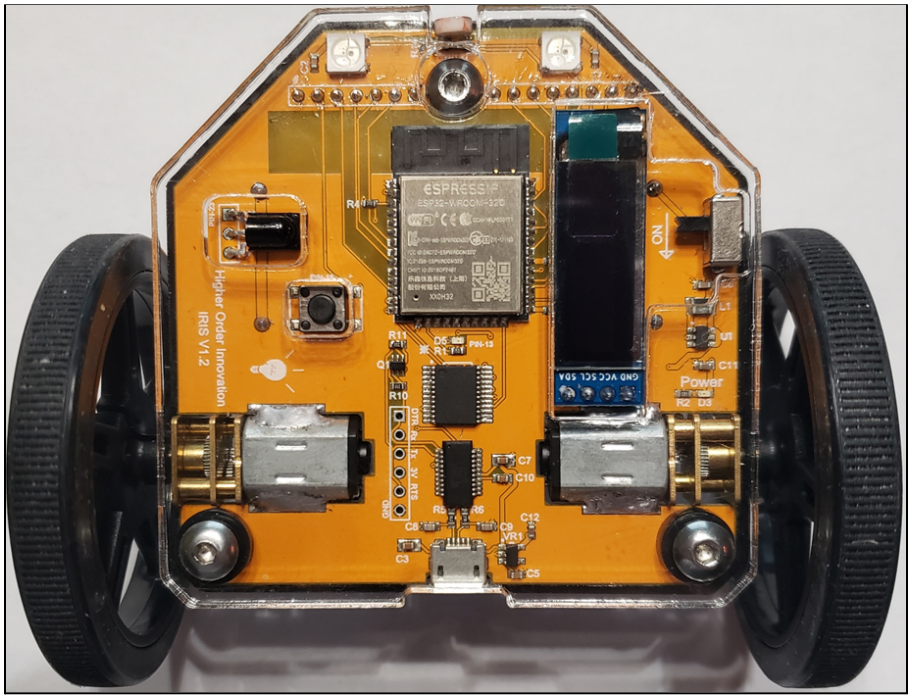


***Figure 9: An assembled Iris robot***

Every path that electricity takes, whether large or small wires, will follow certain laws that govern how they behave. One of these is Ohm's Law, which describes the relationship between voltage, current, and resistance in a circuit.

Ohm's Law: $V = IR$

Voltage ($V$) is defined as the electric potential difference and is measured in volts. Current ($I$) describes the flow of electrons, measured in amperes. Resistance ($R$) is the opposition to the flow of electrons, measured in ohms. The following Khan Academy tutorials go more in depth on voltage, current, and resistance:

- Khan Academy introduction to circuits tutorial: **https://www.khanacademy.org/science/high-school-physics/dc-circuits/electric-current-resistivity-and-ohms-law/v/circuits-part-1**, including the set of 4 practice questions after the videos/articles.

- Khan Academy electric power tutorial: **https://www.khanacademy.org/science/high-school-physics/dc-circuits/electric-power-and-dc-circuits/v/electric-power**, including the sets of 7 and 4 practice questions after the videos/articles.

## 2.2 Light Emitting Diodes (LEDs)

The Iris board contains two LEDs. One is the green power LED that is always on as long as the Iris is receiving power. The other LED is blue and is attached to the microcontroller so that you can control its state, either on or off, through your code. Both of these LEDs are surface-mount components.

Work through the following tutorial. Note that the tutorial only shows through-hole LEDs instead of surface-mount LEDs, however they operate identically. Surface-mount LEDs are smaller and used more often on boards.

- Adafruit LEDs tutorial: **ttps://learn.adafruit.com/all-about-leds/overview**

The first LED on your board is the power LED. Below are the different representations of the circuit and its operation:

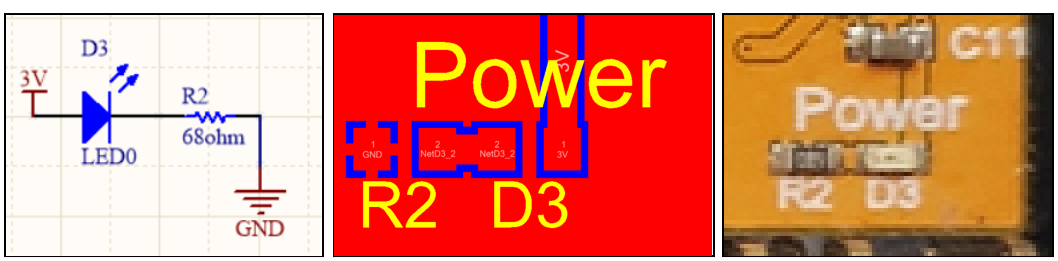


***Figure 10:The power LED in schematic representation (left), pcb design (center), and assembled on Iris (right).***

Here is the specification sheet for the power LED that is on your Iris board: **https://www.we-online.de/katalog/datasheet/150060BS75000.pdf**. The microcontroller on the Iris board operates at 3.3 volts. Based on the datasheet (see page 2), the forward voltage is 2 volts at a forward current of 20 mA.

What size of resistor is necessary to achieve this? We can use Ohm's Law to figure it out:

$$V = IR \rightarrow R = \frac{V}{I}$$

$$\begin{aligned} R &= \frac{V_s - V_f}{I} \\ &= \frac{3.3V - 2V}{20mA} \\ &= \frac{1.3V}{0.02A} \\ &= 65\,\Omega \end{aligned}$$

The engineering reality is that not every exact size of resistor exists, but rather standard sizes are used. In the case of the green power LED on the Iris board, the closest size resistor of 68 Ohms was chosen.

When the current flowing through an LED is increased, the intensity of the light also increases. LEDs are designed with typical and maximum forward currents. The LED above was calculated for a forward current of 20mA. If the resistor in the circuit was replaced with a 55 Ohm resistor, would the current go up or down? Would the light get brighter or dimmer?

Using Ohm's Law we can see that if the resistance decreases the current will increase because we are not changing the voltage. An increased current will cause the LED to get brighter.

Now let's look at the blue LED on the Iris robot. Below are the different representations of the circuit and its operation:

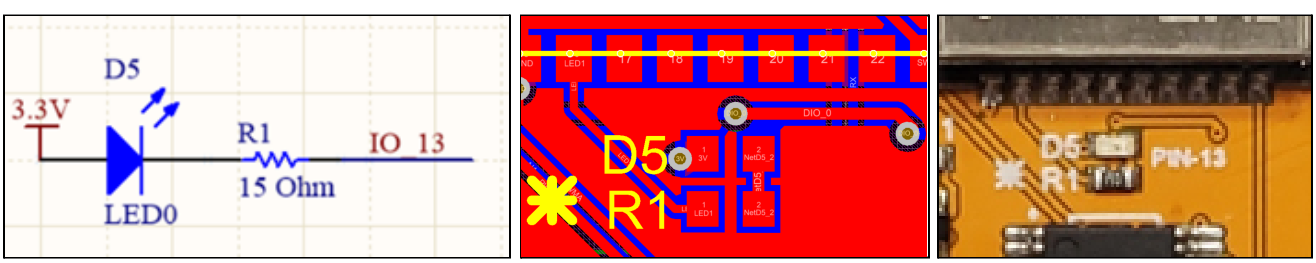


***Figure 11:The center blue LED in schematic representation (left), pcb design (center), and assembled on Iris (right).***

What do you notice about the resistor that is paired with the blue LED compared to the power LED? The blue LED requires a much smaller resistor (15 Ohm compared to 65 Ohm). Why do you think this is? Note that the current has not changed and the forward voltage has not changed because we are still connected to the same power source (3.3V from the batteries).

Using Ohm's Law we can see that if the resistance decreases but the current stays the same, there must be a smaller change in voltage. Since the forward voltage is the same, this must mean we have a large source voltage. By looking at the part specification sheet we see that yes, the blue LED requires more voltage at 3.2V: **https://www.we-online.com/katalog/datasheet/150060BS75000.pdf**

## 2.3 Blink program

The first Arduino code we will work through is the Blink program. To open this example program go to File > Examples > IRIS Robot > Ch02-Blink. Note you will need to scroll down to Examples from Custom Libraries to find the Iris example programs. The purpose of this program is simple: blink the blue LED on the Iris robot.

When you open the example program in the Arduino IDE you will notice there are a lot of double forward slashes // followed by readable text. These are *comments*, which are sections of the code that are meant for human readers to better understand the code and are ignored by the computer or microcontroller. All of the example programs in this ebook begin with a large section of comments, including the purpose of the program and information on the included functions. Read through the following links to learn more about comments:

- Single line comments: **https://www.arduino.cc/reference/en/language/structure/further-syntax/singlelinecomment/**
- Block comments: **https://www.arduino.cc/reference/en/language/structure/further-syntax/blockcomment/**

All Arduino programs (called *sketches*) always contain two specific functions: *setup*() and *loop*(). The *setup*() function runs once in the program and the *loop*() function runs continuously until told to stop. You can see in the Blink example code that this program has a blank *setup*() function; however, it is still necessary for Arduino to understand how to run the program. Read through the following links to learn more about these important functions:

- Setup function: **https://www.arduino.cc/reference/en/language/structure/sketch/setup/**
- Loop function: **https://www.arduino.cc/reference/en/language/structure/sketch/loop/**

Once you have the Iris robot plugged into the computer and have gone through all of the set-up steps in section 1.4, you are ready to run the program. Hit the arrow button at the top of the Arduino IDE screen to compile the code on the Iris board (robot). See Figure 8 for more information. Remember that every time you change the code in the Arduino IDE, you will need to recompile and upload the code to the Iris board using the arrow button.

After you upload the program to the Iris board to run the program answer the following questions:

1. How would you make the LED blink less frequently? Have Iris pause for one second between half-second blinks. Show how you would change the `void loop(){}` section.

2. How would you make the LED blink quicker? Have the LED stay on for only a quarter of a second between half-second pauses. Show how you would change the `void loop(){}` section.

***Blink code:***

```
// IRIS Chapter 2 - Blink
//////////////////////////
// Purpose
//   Blink the Blue LED on the Iris Robot

// Required Topics
// 1. robot.setLED(bool state) will turn the light on (true) or off (false)
// 2. delay(int milliseconds); will wait the indicated number of milliseconds

#include <IRIS.h>   // Import the IRIS Library
IRIS robot;         // Create an Iris Object called robot

void setup() {
  // Nothing needed here
}

void loop() {
  robot.setLED(true);   // Turn the center LED ON
  delay(500);           // Wait for 500 milliseconds
  robot.setLED(false);  // Turn the center LED OFF
  delay(500);           // Wait for 500 milliseconds
  // Return to top of loop() and repeat forever
}
```

***Answers:***

1. Change the length of the second delay to make Iris blink less frequently.

```
void loop() {
  robot.setLED(true);
  delay(500);
  robot.setLED(false);
  delay(1000);        // Pause for one second
}
```

2. Change the length of the first delay to make Iris blink quicker.

```
void loop() {
  robot.setLED(true);
  delay(250);          // LED stays on for 250 milliseconds
  robot.setLED(false);
  delay(500);
}
```

## 2.4 Challenge - Declare, use, and assign variables

In the Blink example code we used the pre-written functions *robot.setLED*() and *delay*(), both of which require a number as input. What if we wanted that value to change while the program is being run? We cannot change the code while the program is being run so we cannot change the number. We will need to use a variable. In Arduino, creating a variable is called a *declaration*. Read through the following link to learn more about declaring, using, and assigning a variable:

➢ Arduino variable declaration: **https://www.arduino.cc/en/Reference/VariableDeclaration**

Your first challenge is to write a program in which the LED blinks get progressively longer by one half second each blink. The LED needs to turn on for 0.5 seconds, then 1 second, then 1.5 second, then 2 seconds, then 2.5 seconds, etc., with a constant half second pause between blinks. Declare and use a variable named duration to accomplish this challenge.

Try to write this challenge program on your own. Remember that there are often multiple computer programming solutions for a given task. Once you are ready, you can go to File > Examples > IRIS Robot > Ch02-Blink-Challenge to see one solution.

# Chapter 3: Button

## 3.1 Pull-up resistors

The push button on the Iris board is connected to a pull-up resistor. This is necessary for Arduino to reliably determine whether the push button is pressed or released. Work through the following tutorial to learn more about pull-up resistors:

- Sparkfun pull-up resistors tutorial: **https://learn.sparkfun.com/tutorials/pull-up-resistors**

Below are the different representations of the push button. Can you identify the pull-up resistor in each of them? This is a trick question because this button does not use a separate pull-up resistor but rather it becomes part of the microcontroller. It is important to know that electronically it still requires a pull-up resistor but there are less obvious ways for one to be used.

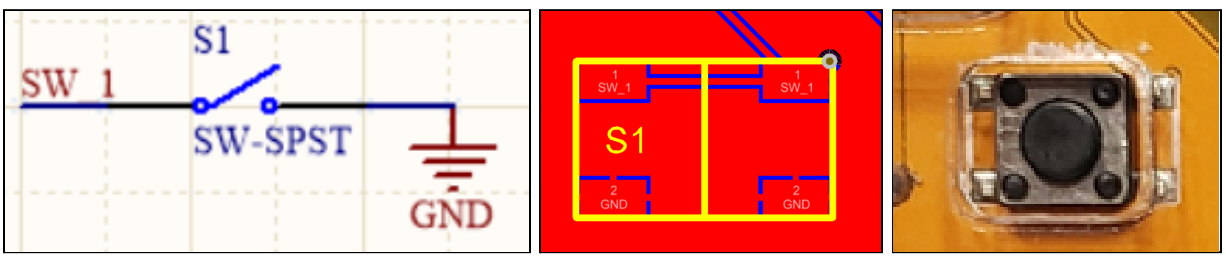


***Figure 12:The push button in schematic representation (left), pcb design (center), and assembled on Iris (right).***

## 3.2 Types of buttons

The Iris board contains one button (push button) and one switch (power switch). Buttons and switches are used to alternate between an open circuit and a closed circuit - an easy enough purpose but there are many types of buttons and switches. Work through this tutorial to learn the basic types:

- Sparkfun buttons and switches tutorial: **https://learn.sparkfun.com/tutorials/button-and-switch-basics**

Here is the specific part for the button on your Iris board: **https://www.digikey.com/en/products/detail/c-k/PTS645SM43SMTR92-LFS/1146840?s=N4IgTCBcDaIMIGkByBOAjGscAqBaJAIiALoC%2BQA.** What type of button is it? Be as specific as possible. By reading through the product attributes descriptions you will find it is a momentary single-pole single-throw (SPST) button. This type of button has a single input and a single output that are only connected when the button is pressed, breaking when it is released.

Here is the specific part for the power switch on your Iris board: **https://www.digikey.com/en/products/detail/c-k/OS102011MS2QN1/411602?s=N4IgTCBcDaIMIGkByBOArANjQWiQERAFoBfIA.** What type of switch is it? Be as specific as possible. By reading through the product attributes descriptions you will find it is a maintained single-pole double-throw (SPDT) switch. This type of switch has a single

input that can be connected to either of two outputs. The state of this switch maintains its connection even after the user releases it.

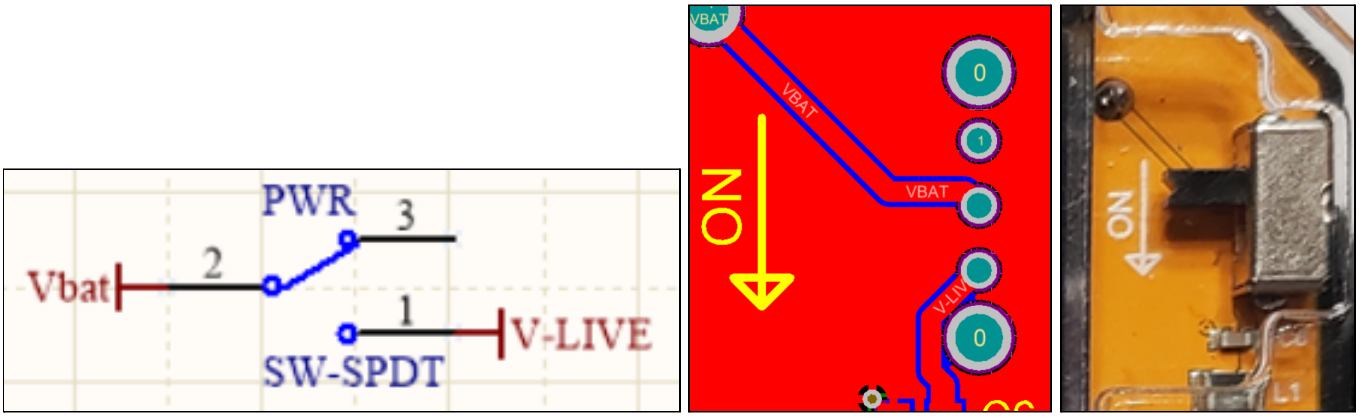


***Figure 13:The power switch in schematic representation (left), pcb design (center), and assembled on Iris (right).***

## 3.3 Button program

For this program we will be using the push button on the Iris robot. Go to File > Examples > IRIS Robot > Ch03-Button to open the program. The function *robot.getButtonState*() checks to see if the button is being pressed (TRUE state) or not (FALSE state). We can use this information to affect other parts of the robot by using the if() and else() functions:

- *if*() function: **https://www.arduino.cc/reference/en/language/structure/control-structure/if/**
- *else*() function: **https://www.arduino.cc/reference/en/language/structure/control-structure/else/**

The Button example program also introduces the important topic of *serial communication* in Arduino programming. The serial port and serial terminal are ways that the Arduino board (your Iris board) communicates with your computer. In order to use this form of communication, we need to begin the serial communication using *Serial.begin*() function. This is added as part of the *setup*() function of the program to specify the baud rate to indicate how quickly the information should go between Arduino and the board. For this program, we are using the *Serial.println*() function to tell us whether the LED is on or off. In order to see what is being "printed" to the serial terminal, you will need to open the serial monitor by using the magnifying glass button in the upper right corner of the Arduino IDE (see Figure 8). The following link contains an overview of serial communication in Arduino:

- Serial communication: **https://www.arduino.cc/reference/en/language/functions/communication/serial/**

After you upload the program to the Iris board to run the program answer the following questions:

1. Why does the serial monitor show multiple lines in a row of either "The LED is on" or "The LED is off"?

2. How would we change the code if we wanted the LED to be off when the button is pressed? Show how you would change the `void loop(){}` section. Hint: take a look at boolean operators.

***Button code:***

```
// Iris Chapter 3 - Button
//////////////////////////
// Purpose:
//  When the button is pressed, turn the LED on.  Use the serial monitor to
print whether the LED is on or off.

// Required Topics:
// 1. robot.setLED(bool state) will turn the light on (true) or off (false)
// 2. robot.getButtonState() returns true (pressed) or false (released) for the
state of the button.

#include <IRIS.h>   // Import the Iris library
IRIS robot;         // Create an Iris object called robot
//This allows Arduino to talk to the Iris robot you have plugged in to the
computer.

void setup() {
  Serial.begin(9600); // Begin the Serial Communication at 9600 baud
// Open the Serial Monitor using the magnifying glass in the upper right corner
}

void loop() {
  if (robot.getButtonState()) {       // Check the state of the button
    robot.setLED(true);               // If the button is TRUE then the LED
                                         turns on
    Serial.println("The LED is on");  // Print a new line to the Serial Monitor
  }
  else {
    robot.setLED(false);              // If the button is FALSE then the LED
                                         turns off
    Serial.println("The LED is off"); // Print a new line to the Serial Monitor
  }
  delay(10);  // This delay is necessary for the button to reset
}
```

***Answers:***

1. The answer is found in the *loop*() function. *loop*() will run over and over forever, faster than your human finger can press the button. The Iris robot is running *loop*() hundreds of times per second. Every time the loop function runs, it either prints "The LED is on" or "The LED is off", depending on the state of the button. The way the code is currently written it will print over and over until the state of the button changes.

2. The simplest way to change the code such that the LED is continuously on until the button is pressed is to add one character: an exclamation mark. The ! character in boolean logic means "not" and is used to make TRUE values into FALSE and vice versa.

```
void loop() {
  if(!robot.getButtonState()){        // Exclamation mark means "not"
    robot.setLED(true);               // If the button is NOT TRUE (i.e. false)
                                      // then the LED turns on
  }
  else{
    robot.setLED(false);              // If the button is NOT FALSE (i.e. true)
                                      // then the LED turns off
  }
}
```

See the following link for more information on various types of operators (boolean/logic, mathematical, comparison, etc.) in Arduino programming language:

- Types of Arduino structure: **https://www.arduino.cc/reference/en/#structure**

## 3.4 Challenge - Calculate how long the button is pressed

We know how to print to the serial monitor when the button is pressed but let's see if we can print something more useful than "The LED is on". What if the serial monitor was able to tell us how long the button was pressed? Then the Iris robot can be used as a reaction timer: how quickly can you press and let go of the button? There are a few new functions, structures, and Arduino topics you will need to incorporate together for this challenge:

- *while* loop control structure to pause the code for as long as the button is pressed: **https://www.arduino.cc/reference/en/language/structure/control-structure/while/**

- *break* control structure to leave the while loop: **https://www.arduino.cc/reference/en/language/structure/control-structure/break/**

- *long* variable to hold the large number returned by the millis() function:
  **https://www.arduino.cc/reference/en/language/variables/data-types/long/**

- *millis*() function to record the start and stop times:
  **https://www.arduino.cc/reference/en/language/functions/time/millis/**

Once you are ready, you can go to File > Examples > IRIS Robot > Ch03-Button-Challenge to see one solution to this challenge.

# Chapter 4: Light

## 4.1 Parallel and series circuits

There are a multitude of ways to connect buttons, switches, batteries, resistors, LEDs, sensors, motors, and more on a circuit. As additional pieces are added to a circuit we can describe them as being added in series or in parallel. Work through the following tutorial to understand the difference between a series circuit and a parallel circuit:

- All About Circuits tutorial: **https://www.allaboutcircuits.com/textbook/direct-current/chpt-5/what-are-series-and-parallel-circuits/**

If a voltage is applied across two resistors that are connected in series it can be used as a voltage divider circuit. The output voltage will be smaller than the input voltage (this can be calculated using Ohm's Law). Work through the following tutorial for more information on voltage dividers:

- Sparkfun tutorial: **https://learn.sparkfun.com/tutorials/voltage-dividers**

## 4.2 Photoresistors

One important example of the use of a voltage divider is the light sensor on the Iris robot. The light sensor is known as a photoresistor or a photocell. Photocells change resistance in response to the amount of light. However, most microcontrollers can't sense a change in resistance but rather measure changes in voltage. The combination of the photocell resistor and another resistor can be used to create a voltage divider circuit in which the output voltage is measured by the microcontroller. The following link provides more information on photoresistors:

- EE Power tutorial: **https://eepower.com/resistor-guide/resistor-types/photo-resistor/**

The circuit representations for the photocell are included below. Can you find the resistor that forms the voltage divider with the photoresistor? What value does it have? It has a value of 10,000 Ohms.

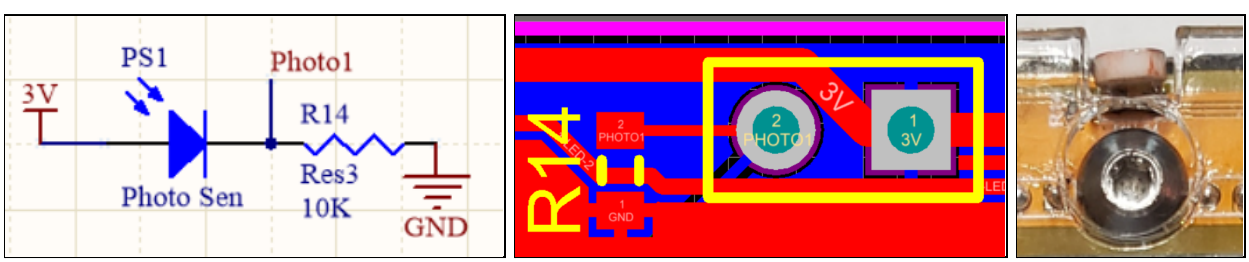


***Figure 14:The photoresistor in schematic representation (left), pcb design (center), and assembled on Iris (right).***

## 4.3 Light program

The program using the light sensor on Iris can be found under File > Examples > IRIS Robot > Ch04-Light. There are only two new Arduino programming topics needed to understand this program:

- Compound addition +=: **https://www.arduino.cc/reference/en/language/structure/compound-operators/compoundaddition/**
- *String* variable type: **https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/**

Note that different data types will be discussed more in-depth in the following chapter. For now it is important to know that *String* is a type of variable like *int* or *long* and that variable declaration works the same for all data types.

After you upload the program to the Iris board to run the program answer the following questions:

1. What is the highest light sensor reading you can get? What is the lowest?

2. What do you think will happen if you take away the *delay*(500) at the end of the *loop*() function?

***Light code:***

```
// IRIS Chapter 4 - Light
//////////////////////////////////////
// Purpose
//   Read the intensity of light reading from the sensor
//   Display the light reading on the serial monitor

// Required Topics:
// 1. robot.getLightReading() will return a value proportional to the light
intensity

#include <IRIS.h>   // Import the Iris library
IRIS robot;         // Create an Iris object called robot
//This allows Arduino to talk to the Iris robot you have plugged in to the
computer.

void setup() {
  Serial.begin(9600);       // Begin Serial communication at 9600 baud
// Open the Serial Monitor using the magnifying glass in the upper right corner
}

void loop() {
  int lightSensorReading = robot.getLightReading();
// Read the light sensor on the robot
  String output = "Light Reading is: ";
// Assemble the output message
  output += lightSensorReading;
// += adds the variable to the message
  Serial.println(output);
  delay(500);
// Delay 500ms between readings
}
```

***Answers:***

1. The light sensor value is higher in bright light. The highest possible value is 4095, although it would be very difficult to achieve this. The light sensor value lowers as it gets darker, with a lowest possible value of 0.

2. *delay*(500) is necessary for the photoresistor in the light sensor to reset. Without the half-second delay, the *loop*() function runs too quickly for both the light sensor and the serial port. If you try to run the program without the delay, the serial monitor will get overwhelmed with how fast the light sensor readings are being printed.

## 4.4 Challenge - Blink in response to light

Reading the light sensor value from the serial monitor on the computer is one way to tell if the Iris robot is in a bright or dark environment. Can you use the LED to provide similar information? Write a program in which the LED on Iris blinks faster if the robot is in lower light and slower blinks if it is in bright light. We suggest using the *map*() function:

➢ *map*() function: **https://www.arduino.cc/reference/en/language/functions/math/map/**

Once you are ready, you can go to File > Examples > IRIS Robot > Ch04-Light-Challenge to see one solution to this challenge.

# Chapter 5: Screen

## 5.1 Computer data storage

Let's take a break from physics and electrical engineering to focus on some important topics in software engineering. The Merriam-Webster dictionary defines a computer as "an electronic device that can store and work with large amounts of information" (https://www.merriam-webster.com/dictionary/computer). How is this information stored? The answer is using binary numbers. The binary number system consists of 0 and 1. Work through the following tutorial to learn how zeros and ones can be combined to create bits and bytes:

- How stuff works tutorial on bytes: **https://computer.howstuffworks.com/bytes.htm**

Computers communicate using bits and bytes but we communicate using the alphabet and words. The American Standard Code for Information Interchange (ASCII, pronounced "ASSK-ee") is the conventional way to encode zeros and ones into characters using seven bits. Characters include symbols such as punctuation and mathematical operators, the digits 0-9, uppercase and lowercase letters, as well as control characters that perform specific functions. The following link contains a table that lists all 128 seven-bit characters and what character it represents:

- W3Schools ASCII reference chart: **https://www.w3schools.com/charsets/ref_html_ascii.asp**

## 5.2 Data types

In computer programming bits and bytes and ASCII all describe *how* the data is stored but Arduino also needs to know *what* is being stored. There are many different formats data can be created to use in your program. These include whole numbers (such as 1 or 3), decimal numbers (such as 1.2 or 33.4), boolean statements (true/false), and strings of text characters (such as "This is a String"). A full list and description of the different data types that are accessible in Arduino can be found below:

- Arduino variable types: **https://www.arduino.cc/reference/en/#variables**

The guide above is primarily directed towards the original Arduino boards which are based on an 8-bit architecture. The ESP32 microcontroller (the brain of Iris) is a 32-bit processor. The effect on this is that the sizes of each variable may be increased. For instance, an *int* on an 8-bit processor may be saved in 16-bits of memory (-32,768 to 32,767) whereas an *int* on the 32-bit ESP32 is held in 32-bits of memory (-2,147,483,648 to 2,147,483,647). While the sizes may vary, the type of data held in each type is the same. An *int* still holds a whole number integer value.

The primary data types that you have seen used in the code examples so far have included the *int*, *long*, *boolean*, and *String*. The types of *int* and *long* both hold whole number integers with capacity being the only difference (*long* is 64-bit for the ESP32). You may be asking, is there any reason not to just define everything as type *long*? The practical answer is that 99% of your programs will never know the difference. Some of the considerations are below:

1. If you are storing a lot of data (many thousands of data points), storage and memory may become a concern.

2. If you are performing a lot of math, the size of the variables involved as well as the operations being performed can create delays in your program. These delays can be measured in microseconds (0.000001 seconds). Small but present if you are trying to do something very precise.

The *boolean* data type can be used to hold true or false. This same sort of ON/OFF could just as easily be held as a 0/1 in an *int*, but *boolean* logic has the distinct advantage of being more clear than some arbitrary setting of 0 = false and 1 = true. *Boolean* data is also smaller (8-bits) rather than the 32-bits of an *int*.

The last main data type is the *String*. It is true that *String* variables take up a lot of memory, but this is the easiest method for manipulating a series of characters. For instance a *String* with a value of "ABCD" would be ASCII encoded and take up 4 bytes of memory. By declaring "ABCD" as a *String* you can use built-in functions to make substrings or find which character is at a specific point. Note that *String* is different from *string* because case (capital vs lower) matters in the Arduino programming language. The data type *string* does not have all of the capabilities of *String*, thus we will always use the uppercase *String*. A detailed list of the *String* functions can be found below:

- *String* variables: **https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/**

## 5.3 Screen program

In this coding section we will be using various data types and ASCII data encoding to communicate with the Iris robot using an OLED (Organic Light-Emitting Diode) screen. An OLED screen contains an electroluminescent layer that lights up in response to an electric current. As you see on your Iris board, it is a small digital display that will allow the Iris robot to communicate.

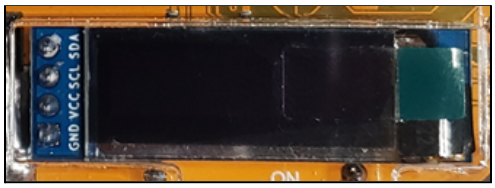


***Figure 15: Built-in OLED Screen***

Specifically, the Iris robot contains the Adafruit Monochrome 128x32 I2C OLED display. Monochrome means it can display a single color (white). 128x32 is the size of the screen: 128 pixels wide by 32 pixels high. I2C indicates how the screen is wired onto the board. For more information see the following link:

- Adafruit monochrome OLED tutorial: **https://learn.adafruit.com/monochrome-oled-breakouts/**

The example program for this chapter can be found at File > Examples > IRIS Robot > Ch05-Screen. The purpose of this program is to use the screen to show how long the robot has been awake. There are no additional Arduino programming topics to learn, only the new functions for the screen. We'll walk through the most useful functions in the example code. However, in order to use the program with the screen we will need to download three additional Arduino libraries:

1. In the Arduino IDE, go to Tools > Manage Libraries. Search for "Adafruit SSD1306". Click Install for the Adafruit SSD1306 library by Adafruit, version 2.4.6. See figure below.

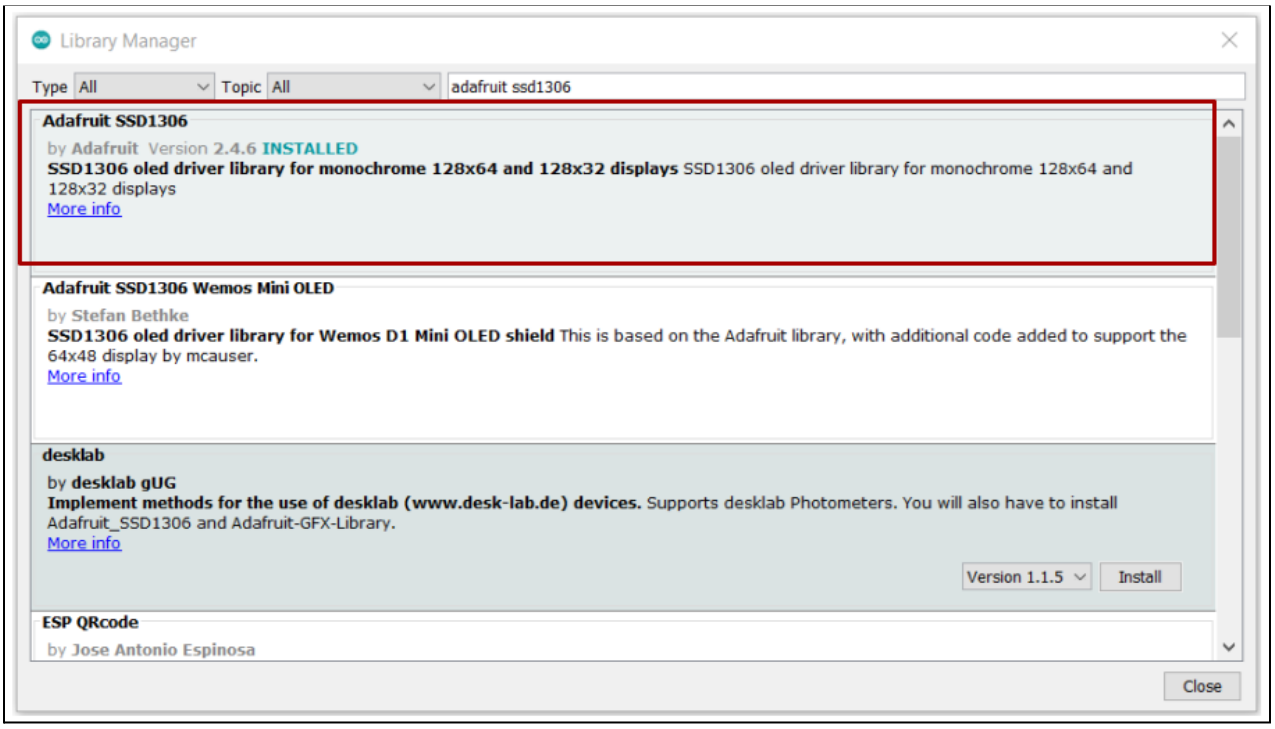


***Figure 16: Adafruit SSD1306 library***

2. In the Arduino IDE, go to Tools > Manage Libraries. Search for "Adafruit GFX". Click Install for the Adafruit GFX library by Adafruit, version 1.10.12. See figure below.

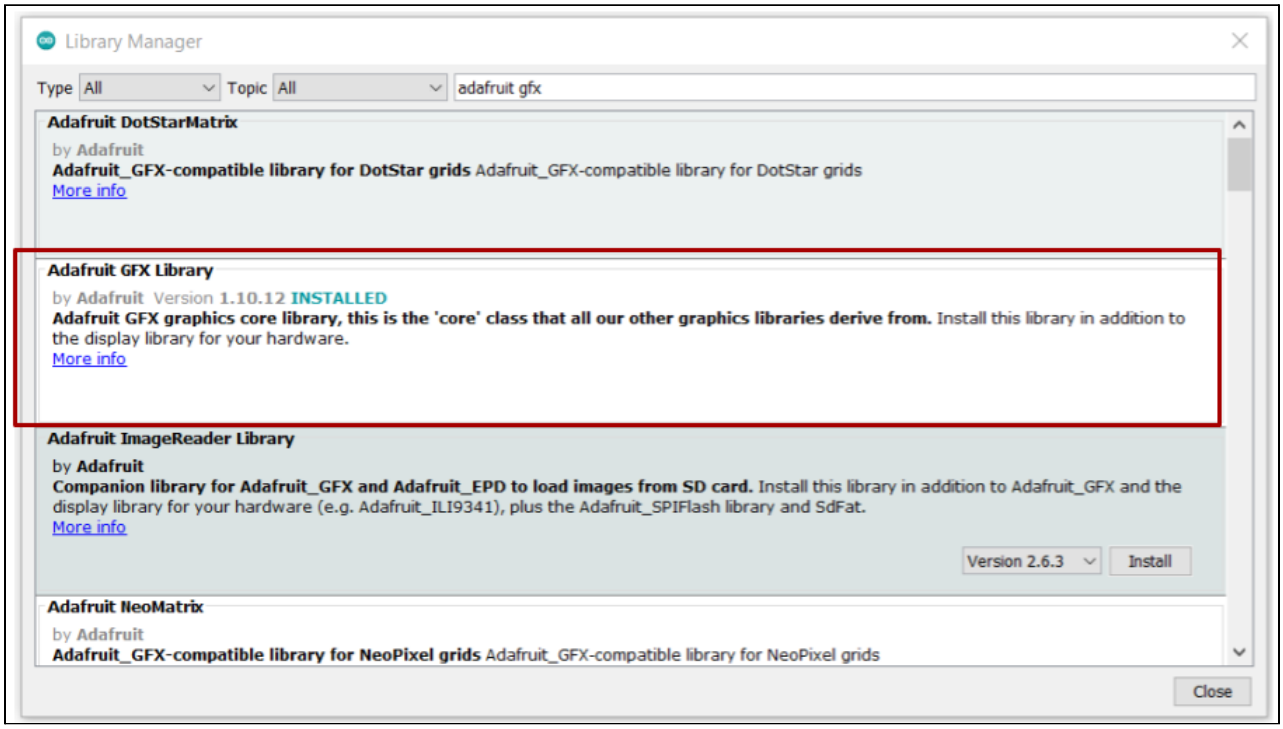


***Figure 17: Adafruit GFX library***

3. In the Arduino IDE, go to Tools > Manage Libraries. Search for "Adafruit BusIO". Click Install for the Adafruit BusIO library by Adafruit, version 1.9.3. See figure below.

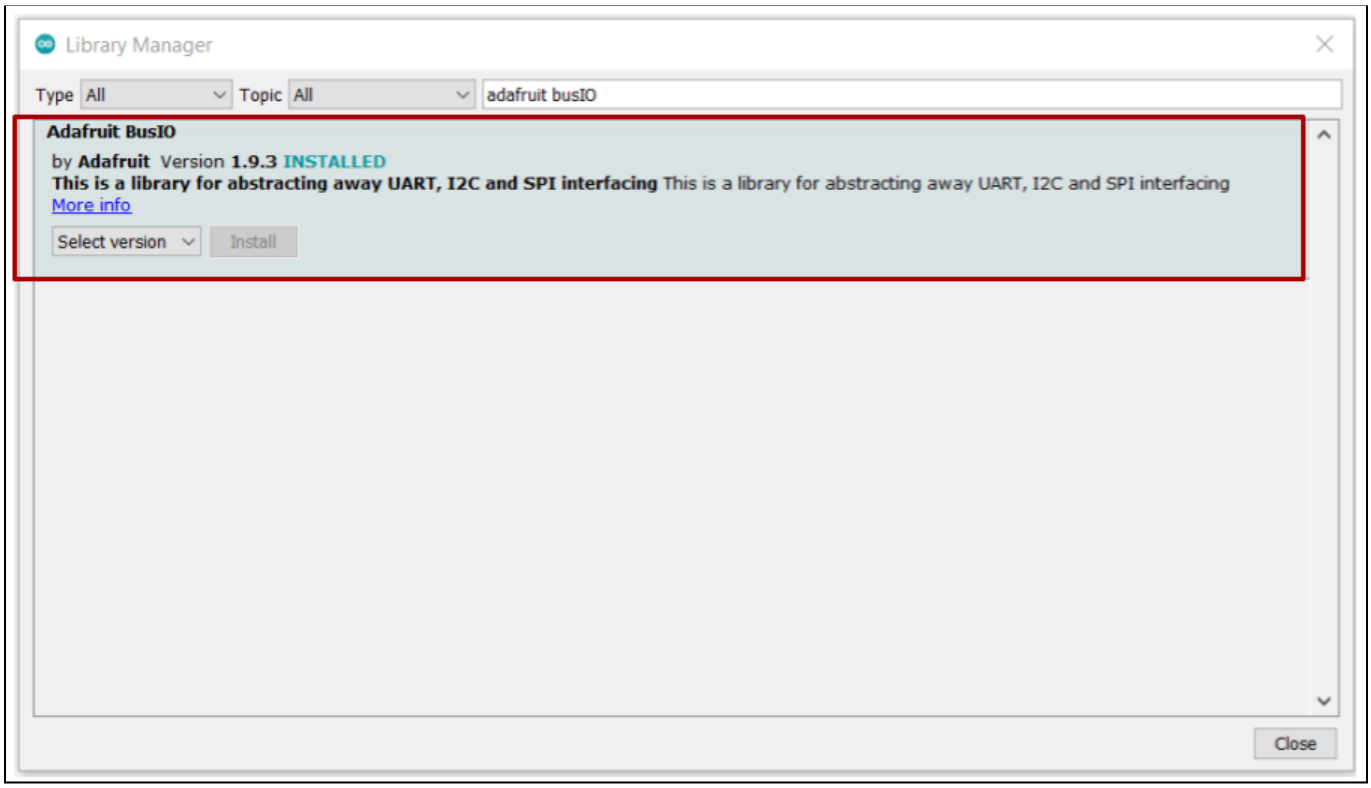


***Figure 18: Adafruit SSD1306 library***

***Screen code, part 1 of 2:***

```
// IRIS Chapter 5 - Screen
////////////////////////////////////
// Purpose:  Use the screen to communicate a counter

// Required Libraries
// 1. Adafruit_GFX
// 2. Adafruit_SSD13067

#include <IRIS.h>   // Import the Iris library
IRIS robot;         // Create an Iris object called robot
//This allows Arduino to talk to the Iris robot you have plugged in to the
computer.

// Additional Include statements for screen libraries
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     4 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// These functions are required to initialize the screen
void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.display();
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
}
```

What is missing in the code above? The code above is approximately half of the full code for the Screen program. You should notice that it is missing one of the required functions for an Arduino sketch: the *loop*() function. The *loop*() section is on the following page. The section on this page includes all of the necessary code to get the screen to work. You don't need to understand every line, but if you want to use the screen as part of another program make sure you include this section of code.

***Screen code, part 2 of 2:***

```
// Declare the variable output.  We will use this to count the seconds.
int output = 0;

void loop() {
  display.clearDisplay();                    // Clear the Screen
  display.setTextSize(1);                    // Normal 1:1 pixel scale
  display.setCursor(0,0);                    // Start at top-left corner
  display.println("My name is Iris.");       // Print to screen then a new line
  display.println("I've been awake for");    // Print to screen then a new line
  display.setTextSize(2);                    // Change the text size
  display.print(output);                     // Print to screen
  display.println(" seconds");               // Print to screen then a new line
  display.display();                         // Render the screen
  output += 1;                               // Assign a new value to output
  delay(1000);                               // Wait one second
}
```

The first half of the code (previous page) told the screen how to work but the second half of the code (this page) tells the screen what to say. We created a variable called *output* which holds the number of seconds since the program began. Looking at this sample code, you should now be able to answer the following questions:

1. How many lines of text can fit on the Iris screen? Play around with the text size. What happens if you change the text size to 3 for the first line? What if you change it to 1 for the last line?

2. What happens when you get to >99 seconds? Can you fix this problem?

There is a wide variety of possibilities when it comes to using the Iris screen. This screen is capable of graphics in addition to text. You can draw shapes, show bitmap images, make animations, and more. We suggest going through the Adafruit GFX graphics tutorial below:

- ➢ Adafruit_GFX library tutorial: **https://learn.adafruit.com/adafruit-gfx-graphics-library**

***Answers:***

1. The size of text is assigned using the *display.setTextSize*() function. If the first line of text is size 3 then you do not see the bottom line. If the last line is size 1 then there is a space between the last line and the bottom of the screen. Both of these show you that there are 4 lines of text possible on the Iris screen.

2. Once there are triple-digits in the number of seconds, there is no longer room on a single line and you will see that the second "s" in "seconds" will disappear. One solution would be to decrease the text size of the third line. Additionally, you could make the number of seconds and the word seconds on separate lines.

## 5.4 Challenge - Communicate the light sensor value on the robot

The primary advantage of the screen on the Iris robot is it allows the robot to communicate. In previous chapters, we have used the LED as one way to provide information, but the screen allows for a much broader range of data transmission. The challenge for this section is to write a program that combines your coding knowledge from the previous two chapters. Can you write a program such that when the push button is pressed the screen shows the value from the light sensor?

We suggest doing this in two steps:

1. Look at the example code for Ch04-Light. Use this to print the light sensor value to the screen.

2. Look at the example code for Ch03-Button. Use this to make the screen only show the value when the push button is pressed.

Once you are ready, you can go to File > Examples > IRIS Robot > Ch05-Screen-Challenge to see one solution to this challenge.

# Chapter 6: Neopixel

## 6.1 Electromagnetic spectrum

In Chapter 4 we used a photoresistor to sense how much light the Iris robot could see (i.e. the brightness). Now we're going to discuss different types of light, specifically colors of light. Work through the following tutorial to learn about the electromagnetic spectrum:

- Khan Academy tutorial: **https://www.khanacademy.org/science/ap-physics-2/ap-light-waves/ap-introduction-to-light-waves/a/light-and-the-electromagnetic-spectrum**

Work through the following tutorial to learn more specifically about the visible light portion of the EM spectrum:

- NASA tutorial: **https://science.nasa.gov/ems/09_visiblelight**

Do you know why red laser pointers are the most common? To find the answer we can use Planck's equation, which describes the relationship between the energy and frequency, and the speed of light equation, which describes the relationship between wavelength and frequency.

Planck's equation: $E = h\nu$ where $E =$energy, $h =$ Planck's constant, and $\nu =$ frequency.
Speed of light equation: $c = \lambda\nu$ where $c =$speed of light constant, $\lambda =$ wavelength, and $\nu =$ frequency.

As the wavelength of an EM wave decreases the frequency will increase. As the frequency increases the energy also increases. Red colored light has the highest wavelength and the lowest frequency in the visible light spectrum, thus it requires the lowest amount of energy. Blue colored light, on the other hand, is at the opposite end of the visible light spectrum and requires more energy.

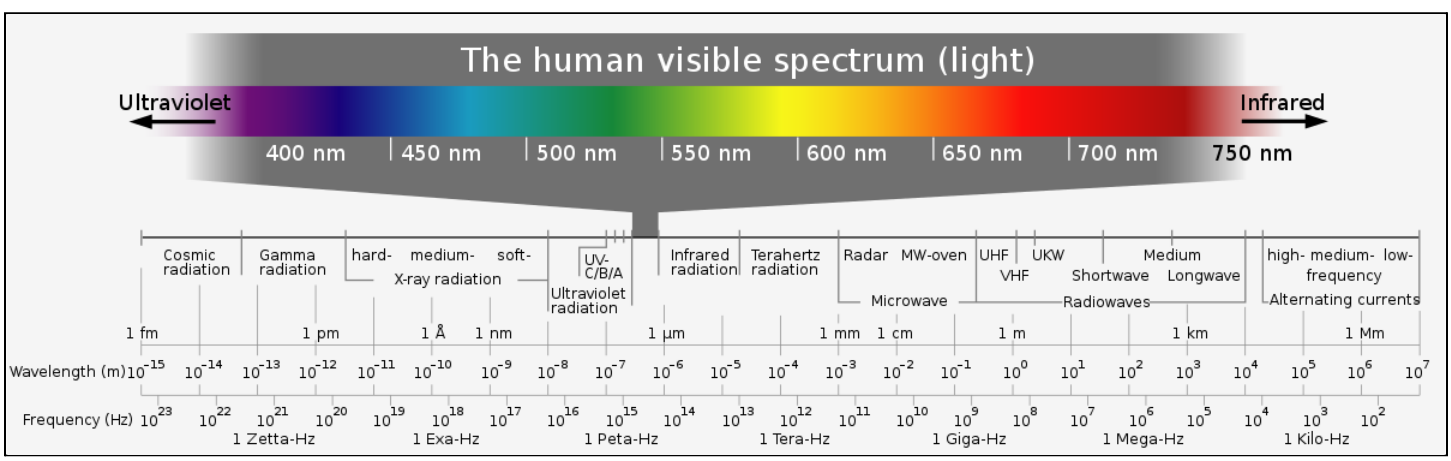


***Figure 19: Visible light spectrum.*** Image source.

## 6.2 Neopixel LED

In Chapter 2 we learned about LEDs and used two specific LEDs: one glowed yellow and one glowed blue. What if we wanted a single light to glow a variety of colors? One way to do this is using the product from Adafruit known as a Neopixel. There are many types of Neopixels available, depending on the type of project you want to create. The following link describes the capabilities of Neopixels:

- Adafruit Neopixel guide: **https://learn.adafruit.com/adafruit-neopixel-uberguide/the-magic-of-neopixels**

The two Neopixels on the Iris robot are individual surface-mount RGB 5mm x 5mm. RGB-type NeoPixels means that the single Neopixel actually contains three LEDS: one red diode, one green diode, and one blue diode. The main benefit of the Neopixel LED is that a string of these lights can be controlled with a single connection (called a pin) from the microcontroller. A more traditional RGB (red-green-blue) LED would require three pins for each one.

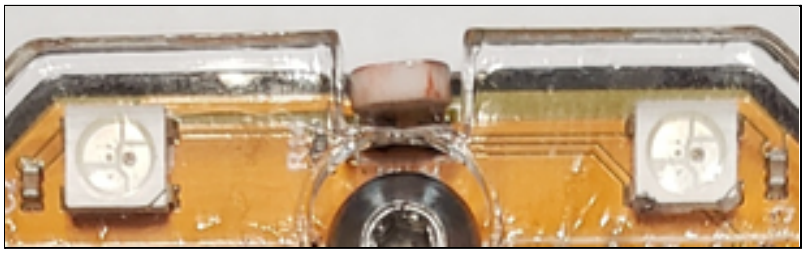

***Figure 20: A pair of Neopixel LEDs assembled on the front of Iris.***

In the Neopixel code, the color is specified using three bytes: one byte for the red diode, one byte for the green diode, and one byte for the blue diode. There are 256 possible choices (0 - 255) for the amount of light shown by each of the diodes. Colors are shown as (R, G, B). The following link shows how the combination of red, green, and blue can be used to create any color in the visible spectrum:

- W3Schools RGB color widget: **https://www.w3schools.com/colors/colors_rgb.asp**

If we wanted the Neopixels to match the blue LED on the Iris robot, what color value would we specify? What about if we wanted it to match the yellow power LED on the Iris robot? The blue LED has a value of (0,0,255) and the yellow power LED has a value of (255,255,0).

## 6.3 Neopixel program

Similar to the OLED screen from the previous chapter, using the Neopixels requires a specific library (which has been included in the Iris library). Go to File > Examples > IRIS Robot > Ch06-Neopixel. Read through the following guide:

- ➢ Neopixel library tutorial: **https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-use**

Turning the Neopixels to a specific color is straightforward code; let's do something more exciting and have the Neopixels fade. In order to use this, we will adjust the *strip.setBrightness*() function using a *for*() loop structure. Read through the following link to get a better understanding of for() loops before looking at the example code:

- ➢ *for*() loop: **https://www.arduino.cc/reference/en/language/structure/control-structure/for/**

As you read through the code over the next two pages, try to answer the following questions:

1. Right now the color is set for red. How would you fade white colored lights?
2. What would you change if you wanted the lights to fade more quickly?
3. What is the maximum brightness the lights will shine?

***Neopixel code:***

```
// IRIS Chapter 6 - Neopixel
//////////////////////////////////////
// Purpose
//   Fade Neopixel LEDs

// Required Libraries
// 1. Adafruit_NeoPixel.h

#include <IRIS.h>   // Import the Iris library
IRIS robot;         // Create an Iris object called robot

// Additional set-up required for Neopixel
#include <Adafruit_NeoPixel.h>
#define LED_PIN    robot.getNeoPin()
#define LED_COUNT 2
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);

void setup() {    // Initialize NeoPixel strip object
  strip.begin();
  strip.show();
}
```

```
uint32_t color = strip.Color(255,0,0);  // Choose the color here.

void loop() {
// The variable i will INCREASE from 0 to 50 by an increment of 1
  for (int i = 0; i <= 50; i = i+1){
    strip.setBrightness(i);
    strip.setPixelColor(0, color);      // Use the color variable
    strip.setPixelColor(1, color);      // Use the color variable
    strip.show();                       // Display the new colors
    delay(50);                          // Wait 50 milliseconds
  }

// Reverse the fading sequence:
// The variable i will DECREASE from 50 to 0 by an increment of 1
  for (int i = 50; i >= 0; i = i-1){
    strip.setBrightness(i);
    strip.setPixelColor(0, color);      // Use the color variable
    strip.setPixelColor(1, color);      // Use the color variable
    strip.show();                       // Display the new colors
    delay(50);                          // Wait 50 milliseconds
  }
}
```

***Answers:***

1. To change the color you need to adjust the RGB value in the *strip.Color()* function. White light has an RGB value of (255, 255, 255).

2. Each of the *for()* loops currently ends with a 50 milliseconds delay. If you want the color to fade more quickly, decrease the delay. If you want the color to fade slower, increase the delay.

3. The maximum brightness has been set to 50 in this example program. The maximum value for these Neopixels is 255, but it is best practice to stay at about 20% when working indoors.

## 6.4 Challenge - Cycle through the rainbow

We have learned how to use a *for*() loop to fade the brightness for a single color. What if we wanted to fade through colors? Write a program that cycles through the rainbow. Hint: look at the following figure. Can you see the six *for*() loops you will need to create? Remember that you will need to set the brightness to a constant value, since it is no longer dynamically changing. Include *strip.setBrightness*(50) in your code.

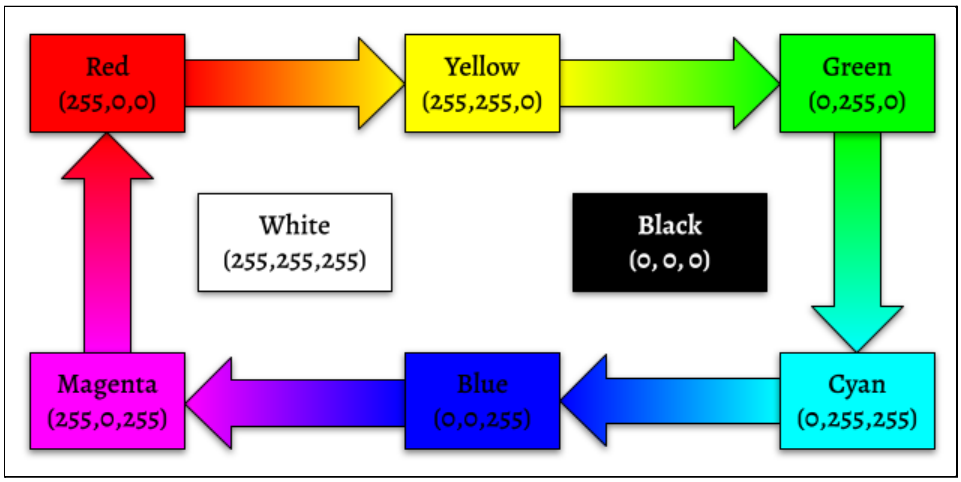


***Figure 21: RGB rainbow***

Once you are ready, you can go to File > Examples > IRIS Robot > Ch06-Neopixel-Challenge to see one solution to this challenge.

If you are interested in learning more about colors, we recommend the following set of tutorials:

- Khan Academy color science: **https://www.khanacademy.org/computing/pixar/color**

# Chapter 7: Motors

## 7.1 Motor selection

Circuit boards can be fun, but making them mobile and something that you can drive is a bonus. In order to have a firm grasp on how motors work, we suggest working through all three videos found in the link below:

- Khan Academy electric motors:
  **https://www.khanacademy.org/science/ap-physics-2/ap-magnetic-forces-and-magnetic-fields/ap-electric-motors/v/magnetism-9-electric-motors**

When deciding how to make your robot move, you are faced with several different types of electric motors capable of turning your wheels. While they all generate rotary motion, the electrical design that allows them to do this can significantly affect the project's complexity and cost. When looking at your Iris robot, you will notice two large silver and gold, rectangular motors attached to the circuit board (Figure 22 below). These two motors are brushed DC, gearmotors. A gearmotor is simply a motor that has a gearbox attached to it (gold part of the motor). Brushed DC motors are the simplest of motors as you simply apply power in one direction or another to generate motion. The link below describes these motors on your Iris robot:

- Sparkfun dc brush motors tutorial:
  **https://learn.sparkfun.com/tutorials/motors-and-selecting-the-right-one/dc-brush-motors---the-classic**

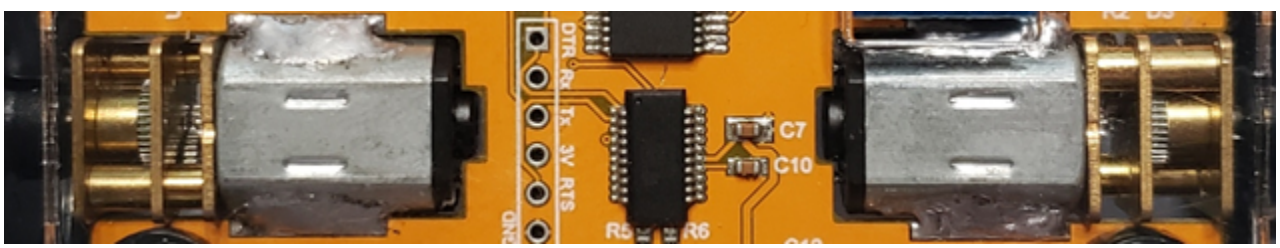


***Figure 22: A pair of brushed DC gearmotors allow Iris to move.***

## 7.2 Precision control

Most motors are able to turn clockwise or counterclockwise. Some can only go in one direction. For the brushed DC gearmotors on Iris, we can control three variables: speed, direction, and duration. For instance, we can tell the robot to spin its left motor 57% of full speed, clockwise, and to run it for 2 seconds before stopping. This is enough control for many things, but when trying to do precision motion, it is only enough to do what's called brute force control. "Brute force" refers to the fact that the best way to control these motors is to test one set of parameters, then adjust, then try again, then adjust again, and so on, in a very inefficient manner.

Brute force control is necessary because there are environmental and robot variables that make precision impossible to maintain. There are several different ways this will happen:

1. As your batteries drain, 57% speed will not necessarily remain the same rotational speed. This will make straight motions shorter and can affect the efficiency of turns.
2. Depending on your surface, the tires may slip or the front leg may drag, changing how the robot moves for a given command. Surface roughness and the ability of the wheels to gain traction will especially affect the turning speed; on hard surfaces it may take 500 milliseconds to perform a turn but on a rug it may take 900 milliseconds to turn the same amount.
3. Manufacturing differences between the left and right motors, even though they are identical models, can mean that sending both motors 100% forward will result in a path that curves away from straight. Adjusting this in your code so that a straight motion is 98%/100% for Left/Right can improve performance.

These factors can make precision movements frustrating as no two runs are the same, but being mindful of battery power and keeping the surface in mind can help get closer to that goal. When more precise applications are necessary, stepper motors can be used as they can easily keep track of their position (and therefore speed).

## 7.3 Motor program

So far, the Arduino programs we have written for the Iris robot have used pre-existing functions. Now we are going to work through writing our own function. As with variables, this is called *function declaration*. Work through the following guide from Arduino:

➢ Function declaration: **https://www.arduino.cc/en/Reference/FunctionDeclaration**

The example program for motor control can be found at File > Examples > IRIS Robot > Ch07-Motor. You will see that we only need one new function in order to drive the motors: *robot.setMotors*(). We have used this function to write a new function called *turn*() that specifically allows the robot to turn. Once you upload the program to the Iris board, make sure the power switch is turned to "on" and unplug the Iris robot. We suggest placing Iris on the floor so it doesn't fall off. Now you are ready to hit the push button which will start the program on Iris. As you work through the program make sure you can answer the following questions:

1. In section 7.2 we mentioned that the Iris robot motors require three variables: speed, direction, duration. Can you see in the code where you are affecting each of those variables?

2. How many degrees does your Iris turn when it's on a hard surface? Now put it on a softer surface (like a tablecloth or rug). How much does it turn now? Don't change the code to test the surfaces.

3. What if we wanted to be able to easily change the speeds of multiple turns? How would you change the *turn*() function?

***Motors code:***

```
// IRIS Chapter 6 - Motors
//////////////////////////
// Purpose
//   When the push button is pressed, drive forward and back

// Required Topics
// 1. robot.setMotors(int left, int right) left and right range from -100....100
with 0 being stop

#include <IRIS.h>
IRIS robot;

void setup() {
}

void turn(int turnDuration){
  robot.setMotors(-50,50);          // Adjusting will change speed of turn
  delay(turnDuration);              // Adjusting will change duration of turn
  robot.setMotors(0,0);             // Stop all motors
  delay(500);                       // Wait for 500 milliseconds
}

void loop() {
  if(robot.getButtonState()){       // Read the current state of the button
    delay(1000);                    // Wait for one second before moving
    robot.setMotors(-100,-100);     // Drive backwards Left = -100, Right = -100
    delay(1000);                    // Continue for 1000 milliseconds
    robot.setMotors(0,0);           // Stop both motors
    delay(1000);                    // Pause for 1000 milliseconds
    robot.setMotors(100,100);       // Drive forward Left = 100 and Right = 100
    delay(1000);                    // Continue for 1000 milliseconds
    robot.setMotors(0,0);           // Stop both motors
    turn(500);                      // Turn for 500 milliseconds
  }
  delay(100);                       // Wait for 100 milliseconds
}
```

***Answers:***

1. The speed for the left and right motors is the absolute value of the inputs to the function *robot.setMotors*(). The direction the motors will move is whether the input is positive or negative. The duration the motors will run is set by the value of the *delay*() function used after the *robot.setMotors*() function.

2. Your Iris will probably turn close to 180 degrees (turn around) on a hard surface, whereas it may be closer to 90 degrees (right angle) on a softer surface. We wanted you to do this to point out that if you want to make a particular type of turn, you will need to adjust and test on the specific surface.

3. Right now the *turn*() function tells the left motor to move counter-clockwise at 50% power and, at the same time, the right motor to move clockwise at 50% power. If we wanted to adjust the speed of the turn we would want the ability to change these values. We can do this easily by rewriting the *turn*() function:

```
void turn(int turnDuration, int left, int right){
  robot.setMotors(-left,right);
  delay(turnDuration);
  robot.setMotors(0,0);
  delay(500);
}
```

## 7.4 Challenge - Drive in a square pattern

Can you get your Iris robot to drive in a square? Try to end up as close to the starting location as possible. You already know how to drive forwards, backwards, and turn. However, this challenge will require a level of precision that may take many iterations of adjustment in the code. To increase your efficiency, we encourage you to write two new functions: one for the corners and one for the edges.

Once you are ready, you can go to File > Examples > IRIS Robot > Ch07-Motor-Challenge to see one solution to this challenge.

# Chapter 8: WiFi

## 8.1 Digital versus analog

Every electrical signal in microcontrollers can be classified as either a digital or analog signal. Digital signals have two states that are either a 0 (LOW/OFF) or a 1 (HIGH/ON). Analog signals can smoothly transition between low and high to produce something like 0.37 for instance. Read through the following guide for a better understanding of how digital and analog signals are different:

- Sparkfun tutorial: **https://learn.sparkfun.com/tutorials/analog-vs-digital/all**

Just as signals can be classified as either analog or digital, so can devices that use them. See the guide below for some of the ways that analog and digital devices both have benefits and drawbacks:

- How Stuff Works tutorial: **https://electronics.howstuffworks.com/question7.htm**

The Iris robot is based on the ESP32 microcontroller which is a digital device. It stores all of its memory as 0's and 1's and that is also how it manipulates the data to perform operations. The ESP32 is capable of interfacing with digital components such as producing a digital output as in chapter 2 with the blinking LED and capturing a digital input from something like the push button from chapter 3. When the microcontroller captures an analog input in chapter 4 with the light sensor, it converts that analog signal into a digital form, specifically 12 bits of digital information. These 12 bits of the analog to digital converter allow the incoming analog signal to be stored as a value between 0 and 4095.

## 8.2 WiFi

The internet has been a driving force behind the rapid development in so many fields. There is a good chance that you are surrounded by computers, phones, and other devices that are constantly communicating with the internet. While some computers still use an Ethernet cable to do this communication, WiFi has grown to handle most of it. Read through the following guide for a more detailed look into what makes up WiFi as a technology:

- How Stuff Works tutorial: **https://computer.howstuffworks.com/wireless-network.htm**

We also recommend the following short video describing the Internet:

- Khan Academy "What is the Internet?": **https://www.khanacademy.org/computing/ap-computer-science-principles/the-internet/introducing-the-internet/v/what-is-the-internet**

The Iris robot is capable of communicating on the same standards as your computer and phone. This means that if you wanted your robot to check the weather for you, it could. There are many different settings and configurations

that your Iris robot can take as it jumps into the WiFi world. Below is a guide that walks through some of these options:

- ➢ ESP32 and WiFi tutorial: **https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/**

One of the most useful configurations for the ESP32 microcontroller on your Iris board is that of a Wireless Access Point. This is the mode that will be used to interact with your robot in the next section.

## 8.3 WiFi program

In order to send data directly to and from Iris, we will be setting Iris up as a wireless access point. This means that your robot will appear in your phone/tablet/computer as a WiFi network. When you connect to that network you will be connecting directly to your own Iris. Work through the following steps (in order):

1. **Open the example program in Arduino**. Go to File > Examples > IRIS Robot > Ch08-WiFi.

2. **Compile the program and upload to your board**. Make sure Iris is plugged in and hit the arrow button on the Arduino IDE (see Figure 8).

3. **On your computer (or phone or tablet) connect to the Iris WiFi network**. If you are currently on a different WiFi network, for instance if you are reading this ebook while connected to the Internet, you will need to disconnect from that network in order to connect to the Iris WiFi network. Below are links to help you find your wireless network settings depending on the type of device:
   - Mac computer: https://support.apple.com/en-us/HT201974
   - iPhone or iPad: https://support.apple.com/en-us/HT202639
   - Windows 10 computer: https://support.microsoft.com/en-us/windows/connect-to-a-wi-fi-network-in-windows-1f881677-b569-0cd5-010d-e3cd3579d263
   - Chromebook: https://support.google.com/chromebook/answer/1047420?hl=en
   - Android phone or tablet: https://support.google.com/android/answer/9075847?hl=en

4. **Open a new browser window** (Google Chrome, Microsoft Edge, Apple Safari, etc.). In the web address bar type in 192.168.4.1. This will take you to the webpage created by the Arduino code you uploaded to Iris.

5. **Control the robot from the webpage**. The example program contains links that will turn the blue LED on and off on the Iris robot.

When you opened the Ch08-WiFi example program in Arduino, you probably noticed that there is a lot going on. Most of this code is required by the ESP32 microcontroller to turn the Iris robot into a wireless access point. Rather than go through all the lines of code, we are going to focus on two specific sections of code, which are spotlighted below:

***WiFi code:***

```
// IRIS Chapter 8 - WiFi
//////////////////////
// Purpose
//   Create an access point with Iris
//   Using a phone/tablet/computer connect to the "Iris" wifi network
//   Open a browser (Chrome/Edge/Safari etc) and go to the webpage 192.168.4.1
//   Click on the different options to change the state of the LED

#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiAP.h>

#include <IRIS.h>   // Include the Iris library
IRIS robot;         // Create an instance of the Iris Robot

const char *ssid = "Iris";
// Set the name of your WiFi network
const char *password = "password";
// Set the password of your WiFi network

WiFiServer server(80);

void setup() {
  Serial.begin(115200);
// Start communication with a baud of 115200
  Serial.println();
// Print a blank line
  Serial.println("Iris Access Point Test");

  //WiFi.softAP(ssid, password);
// Uncomment this line and comment out next line to add the password
  WiFi.softAP(ssid);
// Start a network with a password that will be named ssid
  server.begin();
```

```
// Start up the server that will handle requests

  Serial.println("Server started");
// Print out the notice of the server starting
}

void loop() {
  WiFiClient client = server.available();   // listen for incoming clients

  if (client) {
    Serial.println("New Client.");
// print a message out the serial port
    String currentLine = "";
// make a String to hold incoming data from the client
    while (client.connected()) {            // loop while the client's connected

if (client.available()) {
// if there's bytes to read from the client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
// if the byte is a newline character
        if (c == '\n') {
// if the current line is blank, you got two newline characters in a row.
// that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
// HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
// and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();

  // the content of the HTTP response follows the header:
            client.print("<br><br>Click <a href=\"/H\">here</a> to turn ON the
                         LED.<br><br><br>");
            client.print("Click <a href=\"/L\">here</a> to turn OFF the
                         LED.<br>");

            // The HTTP response ends with another blank line:
            client.println();
            // break out of the while loop:
```

1. Change the text and links on the webpage Iris creates

```
            break;
          } else {    // if you got a newline, then clear currentLine:
            currentLine = "";
          }
        } else if (c != '\r') {
// if you got anything else but a carriage return character,
          currentLine += c;      // add it to the end of the currentLine
        }

  // Check to see if the client request was "GET /H" or "GET /L":
        if (currentLine.endsWith("GET /H")) {
          robot.setLED(true);
        }
        if (currentLine.endsWith("GET /L")) {
          robot.setLED(false);
        }
      }
    }
    // close the connection:
    client.stop();
    Serial.println("Client Disconnected.");
  }
}
```

2. Change what happens when the links are clicked

1. **Change the text and the links on the webpage Iris creates.** This section uses a combination of Arduino programming language and HTML programming language. HTML (HyperText Markup Language) gives structure and specifies the content for webpages. There are four important pieces to the HTML used here:
   - **<br>**: Single line break. Multiple in a row results in empty lines, which can make the webpage easier to read.
   - **<a href = "/..."><br>**: Hyperlink. The text in the quotation marks is the web address of the page the link will take you to. In this case, we want it to be an addition to the current web address so we must begin with a forward slash to indicate it is a part of the current web address.
   - **</a>**: This is the end of the hyperlink tag. All of the text between the <a> and </a> tags is the text that will become the link on the webpage. In this case, the word "here" is the link that you will click to be taken to the new address.
   - **Text**: The remaining text that is not between the <a> and </a> will show up on our webpage as normal text, instead of hyperlinked text.

2. **Change what happens when the links are clicked.** HTML documents (such as the webpage 192.168.4.1 that Iris creates) are transmitted using HTTP, which stands for HyperText Transfer Protocol. HTTP is the rules that govern how the information on the webpage is communicated. The hyperlinks that were created above get encoded as HTTP messages that will end with "GET /*link*" where *link* is the name you gave the new web address. We can use that to tell Iris what to do when the link is clicked in a series of *if*() loops. You will notice that this section is made up of the Arduino programming language, rather than HTML, and contains the *robot.setLED*() function you learned back in chapter 2.

## 8.4 Challenge - WiFi control of Neopixels

You can now use the coding knowledge you have learned in this book to make the Iris robot do whatever you'd like and control it from a webpage. We decided to create a webpage to change the Neopixels on the Iris robot, with the options of red, yellow, green, cyan, blue, and magenta. You can find it at File > Examples > IRIS Robot > Ch08-Wifi-Challenge. A couple of notes:

- The majority of the WiFi example code will not need to be changed; focus on the two sections highlighted in the previous section. You will need to change the HTML links and text using the *client.print*() functions. You will need to change the *if*() loops that contain the *currentLine.endsWith*() functions.

- Remember to add the additional set-up sections for the Neopixels. Look at the Ch06-Neopixel-Challenge example for the necessary lines of code to include the Neopixel library, initialize the Neopixels, and set the brightness levels.

Do you want to make your webpage look nicer? HTML is the structure used to create webpages and it uses CSS (Cascading Style Sheets) for formatting. Here are some links to learn more about HTML and CSS:

➢ W3Schools HTML tutorial: **https://www.w3schools.com/html/default.asp**

➢ W3Schools CSS tutorial: **https://www.w3schools.com/html/html_css.asp**

# Chapter 9: InfraRed

## 9.1 Pulse width modulation

Microcontrollers are digital devices. They work best when switching a signal from ON to OFF, but sometimes an analog signal needs to be input or output. When there is an analog input that the microcontroller needs to work with, the analog to digital converter reads the signal and does its best to map that value to 12-bits (for the ESP32 on Iris). For the reverse case when an analog signal needs to be output, the microcontroller can "fake" the analog output in most cases by using a technique called pulse width modulation (PWM). The guide below discusses the use of PWM signals to mimic a varying analog signal to dim an LED.

- Adafruit tutorial: **https://learn.adafruit.com/circuitpython-basics-analog-inputs-and-outputs/pulse-width-modulation-outputs**

Another common use of PWM signalling is in the control of servo motors. Servo motors are generally a combination of a brushed DC gearmotor with a position sensor and control circuitry. This combination allows the microcontroller to control the precise position of the rotary output rather than having speed control. A discussion of PWM for servo control can be found at the link below.

- Sparkfun tutorial: **https://learn.sparkfun.com/tutorials/pulse-width-modulation**

## 9.2 Infrared

Remote control access to television, our garages, and our music is a common element to everyday life. This was not always the case as control units started out requiring a wire to connect them to the device to control. The specific type of remote control that we will focus on here is the Infrared (IR) remote. Below is a good summary of remote control history and some of the science behind IR remotes:

- How Stuff Works tutorial: **https://electronics.howstuffworks.com/remote-control.htm**

Televisions and stereos are one thing, but what does it take to remote control your Iris robot with one of these? Some background regarding the receiver hardware that is built into the Iris robot can be found below:

- Adafruit tutorial: **https://learn.adafruit.com/ir-sensor?view=all#ir-remote-signals**

Just as a TV remote of the past would not always adjust the volume when you wanted until the remote was held "just right," controlling the Iris robot will be the same way. Since communication is done via IR light pulses, line-of-sight is important, otherwise there may be surfaces that can reflect away the emitted IR signals. There is a balance between the inexpensiveness of IR remotes and the convenience born from the WiFi and Bluetooth controllers of modern technology that do not require you to even be in the same country (in the case of WiFi).

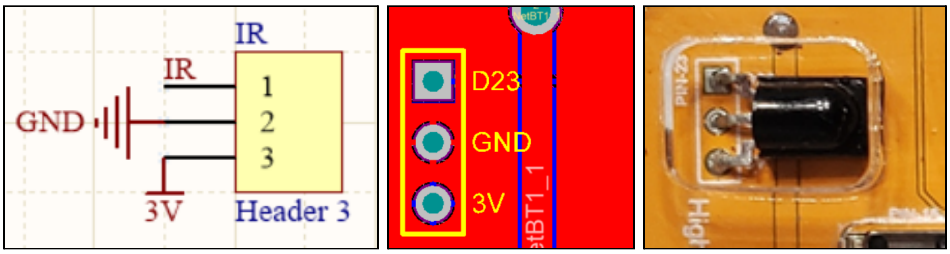


*Figure 23: The IR receiver in schematic representation (left), pcb design (center), and assembled on Iris (right).*

## 9.3 IR program

There is an additional Arduino library you will need to download using the library manager. In the Arduino IDE, go to Tools > Manage Libraries. Search for "IRremote" (no space). Click Install for the IRremote library version by Armin Joachimsmeyer, version 3.3.0. See figure below.

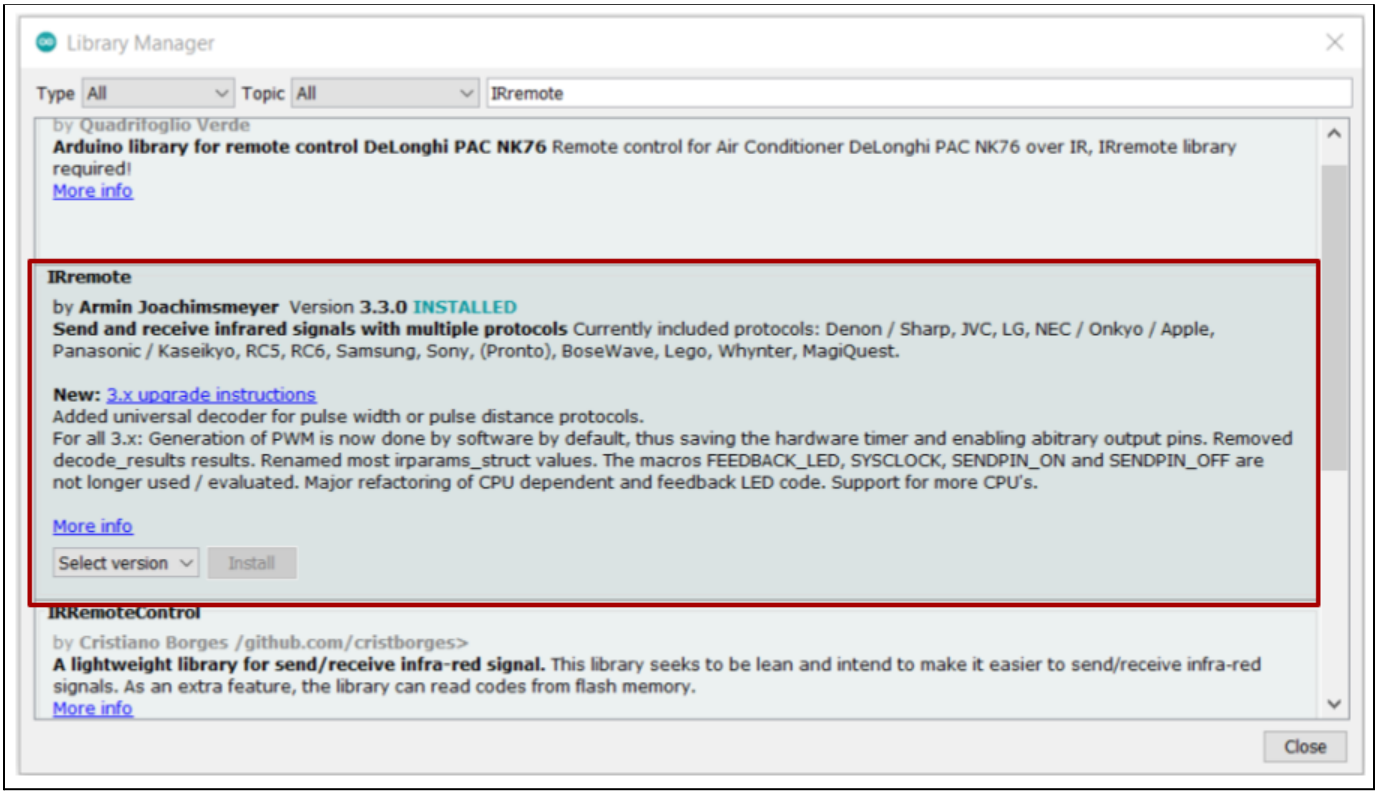


*Figure 24: IRremote library*

Open the IR example program by going to File > Examples > IRIS Robot > Ch09-IR. The goal of this program is for the Iris robot to turn on the blue LED when the UP arrow button on an IR remote is pressed and turn off the blue LED when the DOWN arrow button is pressed. The beauty and difficulty in using an infrared receiver is that IR remotes are ubiquitous. An IR remote has been included with your Iris robot; however, you can use any IR remote that you would like (such as one from your television).

We need the IR remote to send messages to the Iris robot and the Iris robot to understand those messages. The code in the example program Arduino file will need to be adjusted for the specific IR remote you are using. This can be done using the steps outlined below:

1. **Compile and upload the example program to your Iris robot.** Unlike other example programs in the Iris library, this program is probably not going to work the first time you try. This is because different IR remotes send different message codes for the buttons.

2. **See what messages Iris is receiving from the IR remote.** Open the serial monitor in the Arduino IDE using the magnifying glass button. Click the UP button on your remote. Then click the DOWN button on your remote. On the author's computer the UP arrow printed the message "25" on the serial monitor and the DOWN arrow printed the message "28" on the serial monitor - yours may be different. See figure below.

3. **Change the code** in order for Iris to understand the messages sent by the IR remote. You will only need to change a few characters in the code, as highlighted below. Change the "25" to the message printed on your serial monitor when you hit the UP arrow button. Change the "28" to the message printed on your serial monitor when you hit the DOWN arrow button.

4. **Compile and upload the program again.** This time you should be able to turn on the blue LED on the Iris robot using the UP arrow on your IR remote and turn off the blue LED using the DOWN arrow.

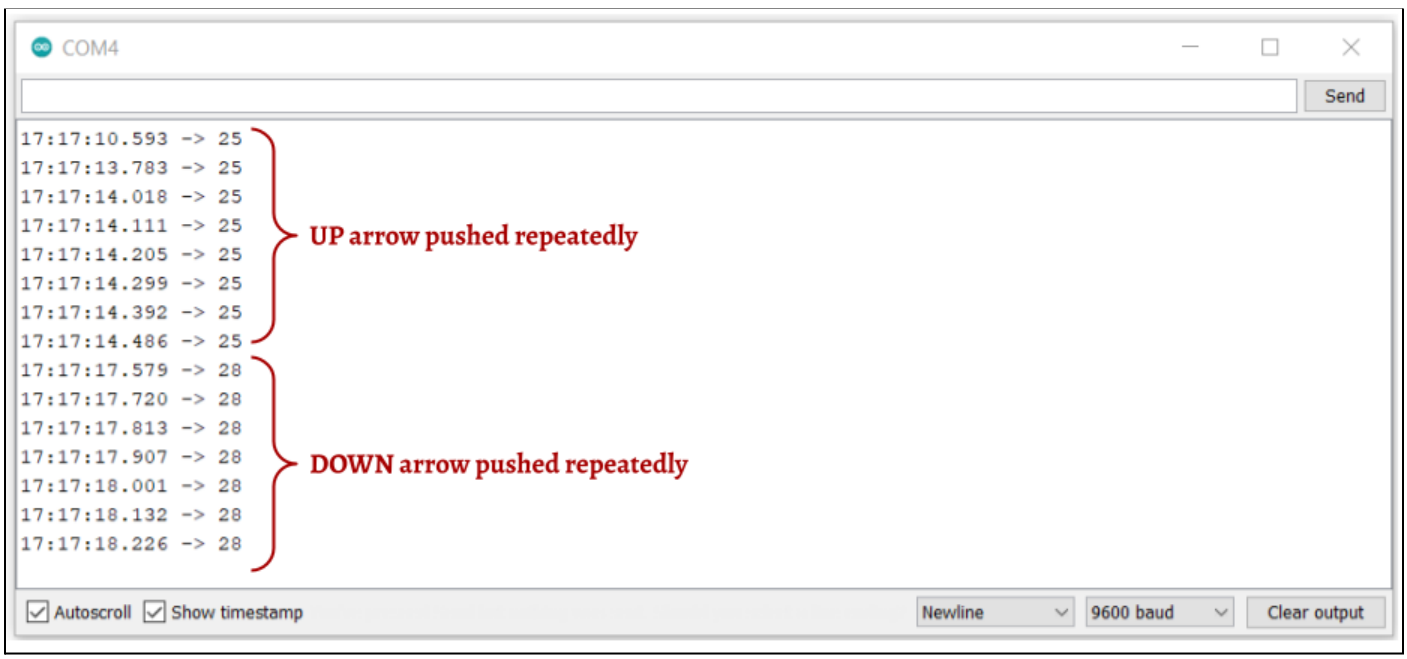


***Figure 25: Serial monitor containing IR remote messages***

***IR code:***

```
// IRIS Chapter 9 - IR - Purpose
//////////////////////////////////
// Purpose
//   Turn the center LED on and off using the remote control input

// Required Libraries - Add through Sketch>Include Libraries>Manage Libraries...
// 1. IRremote.h

#include <IRIS.h>     // Include the Iris library
IRIS robot;
#include <IRremote.h> // Include the IR decoding library
IRrecv irrecv(robot.getIrPin(),13);   // Initialize the IR receiver
decode_results results;

void setup()
{
  Serial.begin(9600);   // Begin serial communication
  irrecv.enableIRIn();  // Start the receiver
  robot.setLED(false);  // Turn the center LED off to start
}
void loop() {
  if (irrecv.decode()) {
// Print the result to the serial monitor
// Use this to find the correct message for each button
    Serial.println(irrecv.decodedIRData.command);

    if (irrecv.decodedIRData.command == 25) { // UP arrow button
       robot.setLED(true);                    // Turn the center LED ON
    }

    if (irrecv.decodedIRData.command == 28) { // DOWN arrow button
       robot.setLED(false);                   // Turn the center LED off
    }
                                                Change to match messages sent by your IR remote
    irrecv.resume(); // Receive the next value
  }
  delay(100);       // Slow down the loop to allow the IR receiver to reset
}
```

## 9.4 Challenge - Remote control of Iris's motors

It's finally time to do everyone's favorite robotics task: drive the robot around the room using a remote control! Write a program that uses the UP, DOWN, LEFT, and RIGHT arrow buttons to control the motors on the Iris robot. We suggest adding an *if*() loop that will keep track of how long it has been since the last message was received and stop the motors if it has been 100 milliseconds since a button was pressed. We also suggest adding a button that stops the motors.

Once you are ready, you can go to File > Examples > IRIS Robot > Ch09-IR-Challenge to see one solution to this challenge.

# Chapter 10: Bluetooth

## 10.1 ISM band

The Industrial, Scientific, and Medical Band (ISM band) refers to portions of the electromagnetic spectrum that have been allocated for special usage, specifically parts of the EM spectrum that contain radio frequencies. When we are talking about Bluetooth and WiFi communication, the ISM band that we generally consider is the 2.4GHz band. For a more detailed look at how crowded and segmented the US radio frequency allocations are, take a look at the breakdown below:

- Radio spectrum: **https://www.transportation.gov/pnt/what-radio-spectrum**

When you consider that nearly all Wifi and all Bluetooth communication takes place in this narrow range of the spectrum more appreciation can be found for just how well everything works on top of each other. The ISM band at 2.4GHz also represents many other radio devices such as cordless phones, baby monitors, toys, and microwave ovens. Part of building a product in this ISM band is that you must accept interference from all of the other devices that also operate there. This interference can be why your bluetooth headphones disconnect or WiFi is sometimes slower than normal.

## 10.2 Bluetooth

Compared to the usage of WiFi, Bluetooth is likely a close second in how often you likely are faced with it. Whether it is Bluetooth connectivity to a pair of headphones or a keyboard to a computer, Bluetooth devices continue to grow in popularity all around us. To understand more about what Bluetooth is and how it differs from other wireless communication like WiFi, read through the following guides:

- Sparkfun tutorial: **https://learn.sparkfun.com/tutorials/bluetooth-basics**

- How Stuff Works tutorial: **https://electronics.howstuffworks.com/bluetooth.htm**

Bluetooth was designed to require less power than WiFi. The trade off is that Bluetooth has a shorter range and operates at a lower data transfer rate. The lower data transfer rate is why Bluetooth devices generally peak at transmission of high quality audio whereas WiFI is capable of streaming high quality video. The ESP32 has Bluetooth built-in, so we are ready to connect to the Iris robot via Bluetooth.

## 10.3 Bluetooth program

To proceed and fully test out the program presented in this section, we have created a companion Android app which is available to download from the Google Play Store. You can use the Bluetooth capabilities on the Iris Robot to connect to any Bluetooth device; however, in order to run the example program for this chapter you will need an

Android device. Please visit **www.higherorderinnovation.com** to purchase an Android phone if you would like a dedicated device to control your Iris robot.

Before we walk through the example code that will allow us to control the robot via Bluetooth, let's get the Iris Robot connected to the app using the following steps:

1. **Open and compile the example program on Arduino.** Go to File > Examples > IRIS Robot > Ch10-Bluetooth then upload the code to your Iris Robot board.

2. **Turn on Bluetooth on the Android mobile device.** Connect to the device called "Iris". See the following link for more specific instructions:

   ➢ Bluetooth connectivity: **https://support.google.com/android/answer/9075925?hl=en**

3. **Download the Iris Robot Platform app.** This app can be found by opening Google Play Store on your Android device and searching for Iris Robot Platform. Download the app distributed by Higher-Order Innovation. See Figure 26 below.

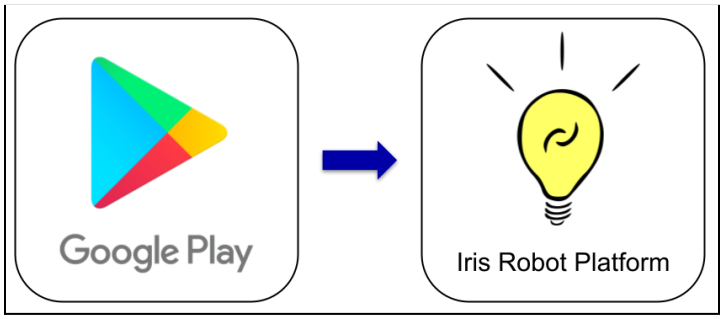


***Figure 26: App icons for Google Play Store and Iris Robot Platform***

4. **Connect the app to the robot.** On the mobile app, use the dropdown menu to choose the device Iris then hit the button labeled "Connect to robot". Once the robot has been connected this button will turn green with the label "Disconnect". See Figure 27 on the next page.

This app will enable your robot in a similar fashion as the WiFi or Infrared examples in that the remote will interact with your robot to make it perform different actions. What these actions are is up to you. Let's first walk through the functionality of the app itself.

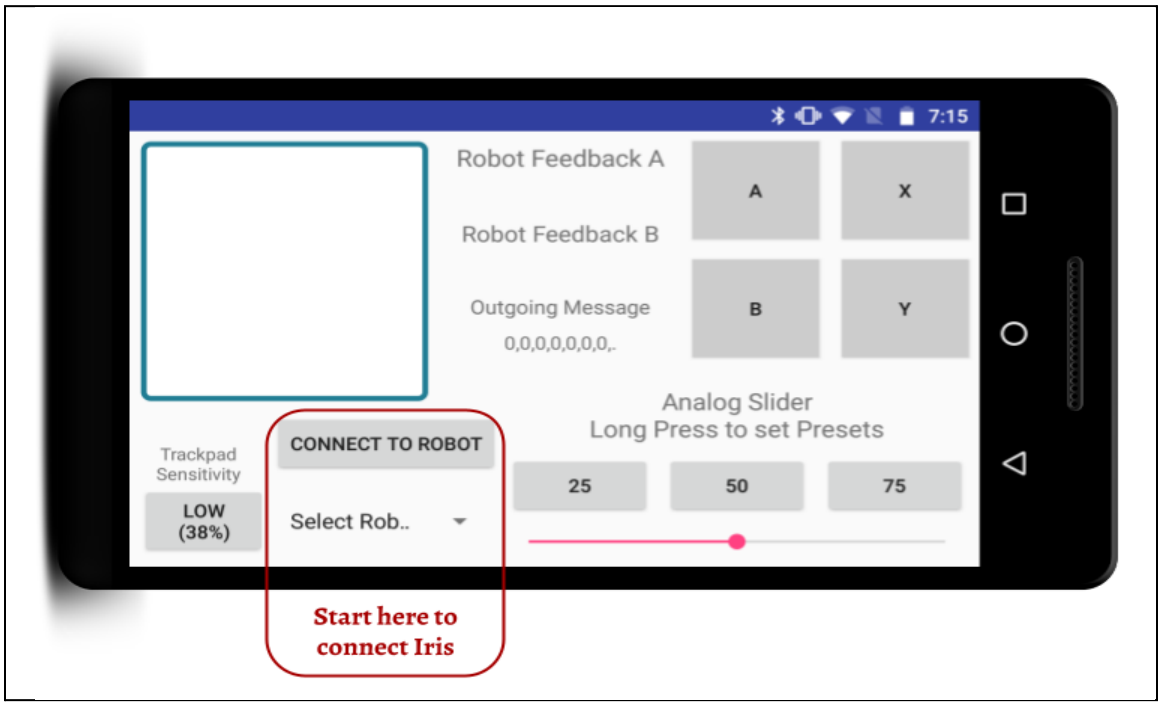


***Figure 27: Iris Robot Platform app***

Iris Robot Platform Android app features list:

1. **Trackpad** - Click and drag within the large square on the upper left side of the screen to create a varying output for your robot.
2. **Trackpad sensitivity button** - Select different degrees of sensitivity to allow different levels of control.
3. **"Select Robot" dropdown** - Select your robot from a list of Bluetooth devices connected to your device
4. **Connect button** - Once your robot is selected on the dropdown, pressing this button will connect your device to Iris. Once your Iris has been connected, this button will turn green and display "Disconnect". Click the disconnect button when you want to unpair the Iris robot from the app. You will need to use this button to disconnect/connect every time you upload new Arduino code to the Iris robot.
5. **Robot feedback A/B** - Iris can send text data to be displayed on the mobile device.
6. **Outgoing Message** - These are the commands that are being sent to Iris. Watch how they change as you interact with the controller. You can also see these messages in the serial monitor in the Arduino IDE.
7. **Buttons labeled A/B/X/Y** - These four buttons on the upper right side of the screen offer a chance to have a digital version of the push button on your robot.
8. **Analog slider and presets** - These three buttons and the slider work together to set a single value. Clicking the buttons will set the value to the number displayed on the button. To change these presets, drag the slider to the intended location and hold down any of the three preset buttons.

What do you notice about the field Outgoing Message? You can see it in the center of the app or on your computer in the serial monitor in the Arduino IDE. These are the commands that are being sent to Iris from the app. The message changes depending on whether any buttons are being pressed on the app, how the analog slider is positioned, or if the trackpad is in use. The outgoing message is in the form of an *array*. For the purposes of this program, we will be using an *array* of *int* (integer) values, meaning that we will have a list of integers that we can access. More information about arrays can be found below:

- Array reference: **https://www.arduino.cc/reference/en/language/variables/data-types/array/**

The specific values of the outgoing message array and their assigned roles can be found in Table 3 below. These values represent the 7 different actions that are communicated from the phone to the robot. Accessing these values enable you to command your robot through the remote interface of the app. In order to access any of these values, you can use remote[i] where i can be any number from 0 to 6, representing the index of the value that you want to use.

***Table 2: Outgoing message array***

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Mapping | Button A | Button B | Button Y | Button X | Trackpad X | Trackpad Y | Slider |
| Range | 0 or 1 | 0 or 1 | 0 or 1 | 0 or 1 | -100 to 100 | -100 to 100 | 0 to 100 |
| Type | Digital | Digital | Digital | Digital | Analog | Analog | Analog |

Just as you have to declare single variables as integers (*int*) or *Strings*, so too do you have to declare *arrays*. When you declare an *array* you will notice that there are sets of square brackets "[ ]" immediately after the variable name as well as a set of curly brackets "{ }" containing the initial 7 values of the *array*. The type and usage of these brackets is required for declaring an array without error. For example, in this program you can use *remote*[1] to access the state of button A on the app.

Data can be collected and sent from the robot to the app to be displayed in the fields Robot Feedback A and Robot Feedback B. In order to tell the phone whether to display the message in position A or position B, the characters "A" and "B" are added to the front of the messages. The function that sends these messages to the phone is *SerialBT.println*() which functions the same way that *Serial.println*() does when sending data to the serial monitor on your computer.

We have highlighted these important sections of the code on the last page of the Bluetooth example program:

***Bluetooth code:***

```
// IRIS Chapter 10 - Bluetooth
////////////////////////////////////
// Purpose
//   Connect the Iris robot to the Higher Order Innovation Iris Remote Control
Android App (available in the Google Play Store)
//   Remember to pair your Iris robot first with your device through the Android
Bluetooth Menu at the top of your screen
//   Next connect the Iris robot to the app using the Connect Robot button
//   Once connected button commands will be sent from the phone to the robot and
feedback sent from your robot to the phone

// Required Topics:
// 1. Arrays -
(https://www.arduino.cc/reference/en/language/variables/data-types/array/)

#include "BluetoothSerial.h"
// This Library allows for Serial Communication between the Robot and your Phone
BluetoothSerial SerialBT;
// Create a BluetoothSerial Object named SerialBT

#include <IRIS.h>   // Import the Iris library
IRIS robot;         // Create an Iris object called robot
//This allows Arduino to talk to the Iris robot you have plugged in to the
computer.

// Remote Control Input
// Index  Mapping     Description
//   0    Button A    1 if pressed, 0 if released
//   1    Button B    1 if pressed, 0 if released
//   2    Button Y    1 if pressed, 0 if released
//   3    Button X    1 if pressed, 0 if released
//   4    Trackpad X  Range from -100 to 100 for horizontal tracking
//   5    Trackpad Y  Range from -100 to 100 for vertical tracking
//   6    Slider      Range from 0 to 100

// For example remote[3] represents the fourth value in the array (numbering
starts from 0) and corresponds to the button marked X in the app
int remote[] = {0, 0, 0, 0, 0, 0, 0};
```

```
boolean stringComplete = false;
// This flag is used to indicate when a full message is received
String inputString = "";
// This String holds the message as it is being read
char tempChar[5];
// This is a buffer that helps in converting the incoming message to integers to
go into remote[]

#include "esp_bt_main.h"
#include "esp_bt_device.h"

void setup() {
  // Start Bluetooth Advertising so that your phone can see it
  // You can name the robot nearly anything by replacing Iris below
  SerialBT.begin("Iris");

  Serial.begin(115200);       // Begin Serial communication at 115200 baud
// Open the Serial Monitor using the magnifying glass in the upper right corner

}

int counter = 0;
void readRemoteInput() {
  // While there are characters waiting to be read
  while (SerialBT.available()) {
    // Read the next character
    char inChar = (char)SerialBT.read();
    // Handle the end of the message (The phone ends the message with a '.')
    if (inChar == '.') {
      stringComplete = true;
    }
    else if (inChar == ',') {  // Each time a comma is hit, save previous value
      // Convert the String to charArray
      inputString.toCharArray(tempChar, 5);
      // Using the function atoi, convert the charArray to an integer
      remote[counter] = atoi(tempChar);
      // Clear the inputString for the next number
      inputString = "";
      // increment the counter to the next position in remote[]
      counter++;
```

```
    }
    else {
      // If not the end of a value (,) or the end of the message (.), add the
incoming character to the end of inputString
      inputString += inChar;
    }
  }
  // Each Time there is a new message, this will trigger
  if (stringComplete) {
    // Reset the end of message flag
    stringComplete = false;
    // Reset the counter that iterates through the remote[] array
    counter = 0;
  }
}

// Keep track of the last message that was printed to the screen
int lastMessagePrintTime = 0;

void loop() {

  readRemoteInput();  // Check for incoming messages

// Every 100 milliseconds, print out the most recent message and act on it
  if (millis() - lastMessagePrintTime > 100) {
    // Update the lastMessagePrintTime to the current milliseconds count
    lastMessagePrintTime = millis();
    // Print out all of the incoming values from the remote
    Serial.print("[");
    for (int i = 0; i < 7 ; i = i + 1) {
      Serial.print(remote[i]);
      if (i != 6) { // Add a comma for readability on all but the last
        Serial.print(",");
      }
    }
    Serial.println("]");
```

```
// If the A Button is pressed, change the state of the LED
    if (remote[0] == 1) {
      robot.setLED(true);
    }
    else {
      robot.setLED(false);
    }
```

Actions sent from app to robot

```
// Send data to the app
// To get your data to show up in Robot Feedback A, the message needs to start
with a capital "A"
// This example sends the light sensor reading to position A on the app
    String outputA = "A";
    outputA = outputA + robot.getLightReading();
    SerialBT.println(outputA);       // Send the message to the phone
// To get your data to show up in Robot Feedback A, the message needs to start
with a capital "B"
// This example sends the state of the button to position B on the app
    String outputB = "B";
    outputB = outputB + robot.getButtonState();
    SerialBT.println(outputB);       // Send the message to the phone
  }
```

Data sent by robot to app

```
  delay(5); // Slow down reading.
}
```

## 10.4 Challenge - Use the slider as input

Using the analog slider as the input, write a program that changes the Neopixels from blue to magenta using the analog slider. Hint: change the value of red in the RGB value of the NeoPixels as the value of the slider increases while keeping the value of green constant at 0 and the value of blue constant at 255. What if your favorite shade of purple has the RGB value (175, 0, 255)? You could hold down one of the buttons above the analog slider to make a preset for the number 175 in order to quickly get back to your favorite color.

Once you are ready, you can go to File > Examples > IRIS Robot > Ch10-Bluetooth-Challenge to see one solution to this challenge.

# Chapter 11: What's Next?

Congratulations on making it through the Introduction to the Iris Robot Platform ebook! Now you know how to write Arduino programs that can:

- ✓ Turn an LED on, off, and blink
- ✓ Use the feedback from a button
- ✓ Learn about the robot's environment using a light sensor
- ✓ Communicate using the on-board OLED screen
- ✓ Play with color using the Neopixels
- ✓ Drive the Iris Robot using two motors
- ✓ Utilize WiFi to create a remote connection to the robot
- ✓ Control the robot using an infrared remote
- ✓ Operate the robot with a Bluetooth-connected device

What do you want to do now? Here are some suggestions:

- ➔ **Play rock, paper, scissors.** Our next mini-ebook will build off of everything you learned in this introductory course to program Iris to play rock, paper, scissors in a variety of ways. Iris can use the screen, an IR remote, and a web browser to attempt to beat you in this classic game.

- ➔ **Compete against other Iris robots.** The CREATE Foundation will be hosting an exhibition at the 2022 U.S. Open Robotics Championship for the brand-new Iris Competition. This competition will test your Iris's ability to race through a maze-like environment. **Contact us** for more information.

- ➔ **Make Iris into a larger robot using the Iris dock.** Continue to program your hand-held Iris robot while increasing it's capabilities. The Iris docking station can be used to add motors, sensors, and larger batteries to a bigger frame. See our website for more information: **www.higherorderinnovation.com**

- ➔ **Hone your critical thinking skills.** Computer programming can be intimidating due to differences in software languages. Block-based coding environments take away those barriers while encouraging out-of-the-box thinking and imagination. We suggest beginning with the Scratch programming environment: **https://scratch.mit.edu/**

- ➔ **Write your own mobile app to control Iris.** For those getting started with mobile app development we suggest the block-based coding environment of MIT App Inventor: **https://appinventor.mit.edu/**

- ➔ **Be creative!**

We hope this Introduction to the Iris Robot Platform has provided you with the tools to ***innovate*** and ***educate***.

Higher Order Innovation

Innovation through Education

www.higherorderinnovation.com