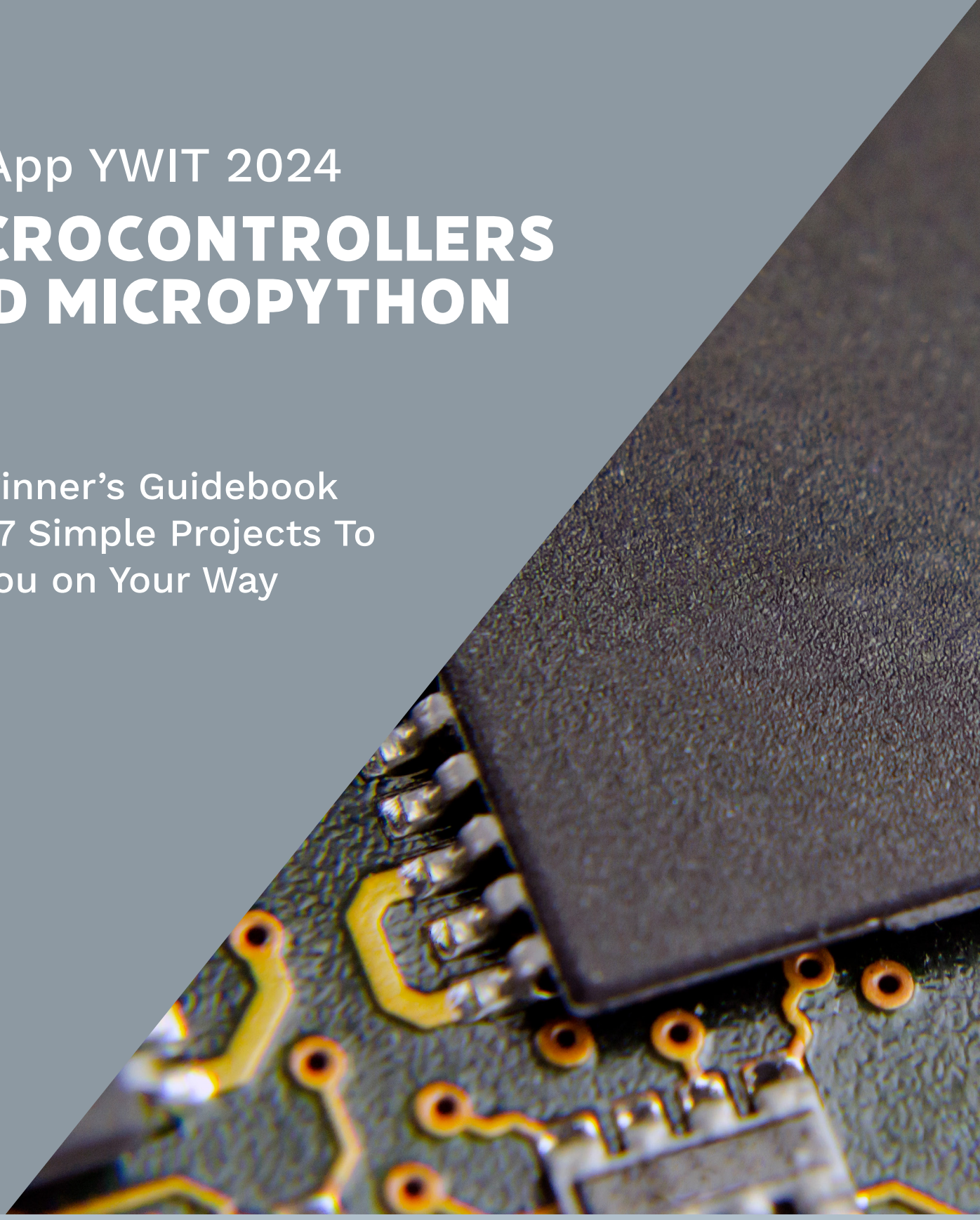
NetApp YWIT 2024

MICROCONTROLLERS AND MICROPYTHON

A Beginner’s Guidebook With 7 Simple Projects To Get You on Your Way



September 26, 2024

Contents

Chapter 1: Introduction

1.1 What this course will teach

This workshop will introduce the student to Python coding, electronics, and project design. We will be building several projects ranging from simple (less components and simpler code) to more complex (more components or more complicated code). These projects are based on the ESP32-C3 microcontroller (specifically the development board produced by Seeed Studio) which is running MicroPython and they depend on some other electronics components such as LEDs, buttons, and more.

Students are encouraged to make use of the written instructions and diagrams and to explore and play with the components for themselves to see how they work and what they do. There are many online resources as well for learning.

1.2 Project Kit Contents

1. Project Case (Color may vary)
2. USB Cable
3. Jumper Wires and WiFi Antenna
4. Breadboard
5. Microcontroller
6. Speaker
7. OLED Display
8. LEDs
9. Capacitive Button and Temperature/Humidity Sensor
10. Resistors and Buttons

Figure 1.1: This image shows all of the contents of the project kit and labels what each one is

1.3 Working with the microcontroller

1.3.1 Installing the driver

Connecting the microcontroller to your computer may require the installation of a USB driver. If so, download the driver for your OS from this page and install as instructed: `https://ftdichip.com/drivers/vcp-drivers/`. Once installed successfully, unplug and replug the USB cable into the microcontroller and then press the small reset button on the microcontroller board labeled "R" (for reset).

1.3.2 Pinout Reference

On any integrated circuit, that is a chip that contains electronic logic, a pinout is a diagram and description of what each of the pins of that circuit are used for. This is a very useful document for any electronics engineer to have handy and to reference. The pinout diagram for the microcontroller that we are using as part of this workshop is replicated below:

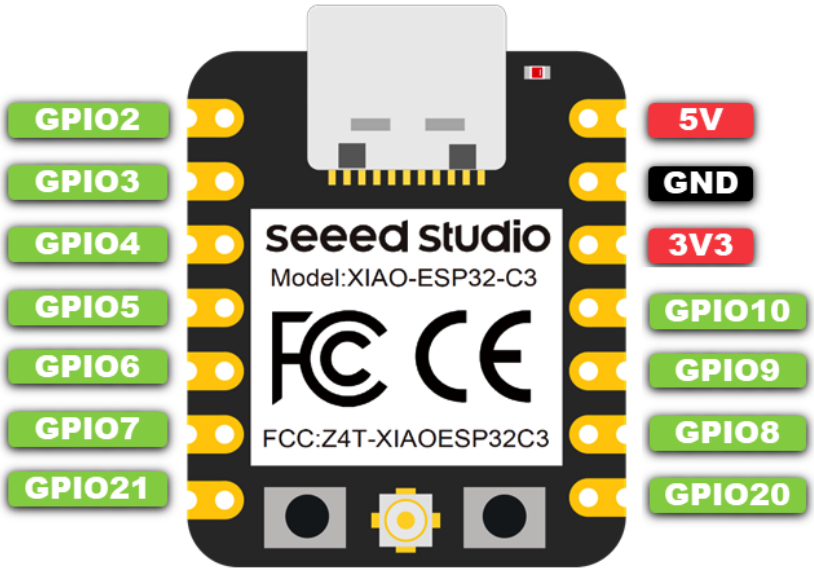


Figure 1.2: A simplified pinout of the Seeed Studio ESP32-C3 development board

In this diagram, you'll notice several things:

- An image of the microcontroller board in the center
- A label for each pin indicating its function

In the code throughout this guide, you will see references to pin numbers like **Pin(5)**. Wherever you see this, it means that the code is expecting a connection to, in this example, the 4th pin down on the left. If you find that your code is not working as expected, double check the pin numbers in the code vs. the placement of your wires.

1.3.3 Project Guide Updates

In case you are reading a out of date version of this guide, you can find the latest at `https://netapp-ptc.github.io/YWIT-2024/project_guide.pdf`.

You can also find the source code for all of the workshops and other useful information in the GitHub repository at `https://github.com/NetApp-PTC/YWIT-2024`.

Chapter 2: Electronics Essentials

Electronics is a facinating world. This beginner's guide will introduce you to the essential components and tools you'll need to start building your own simple circuits. We'll cover the basics of breadboards, LEDs, buttons, resistors, and jumper cables. You should feel free to explore on your own as well using the components in your kit. While this guide will be useful, it is not even close to everything you'll need to know. There is always more to learn and if electronics interests you, there are lots of online tutorials as well as formal education to guide you further.

2.1 The Breadboard: Your Circuit's Foundation

2.1.1 What is a breadboard?

A breadboard is a reusable platform for prototyping electronic circuits without the need for soldering. It's a plastic board with numerous tiny holes arranged in rows and columns. These holes are connected internally, making it easy to connect components together.

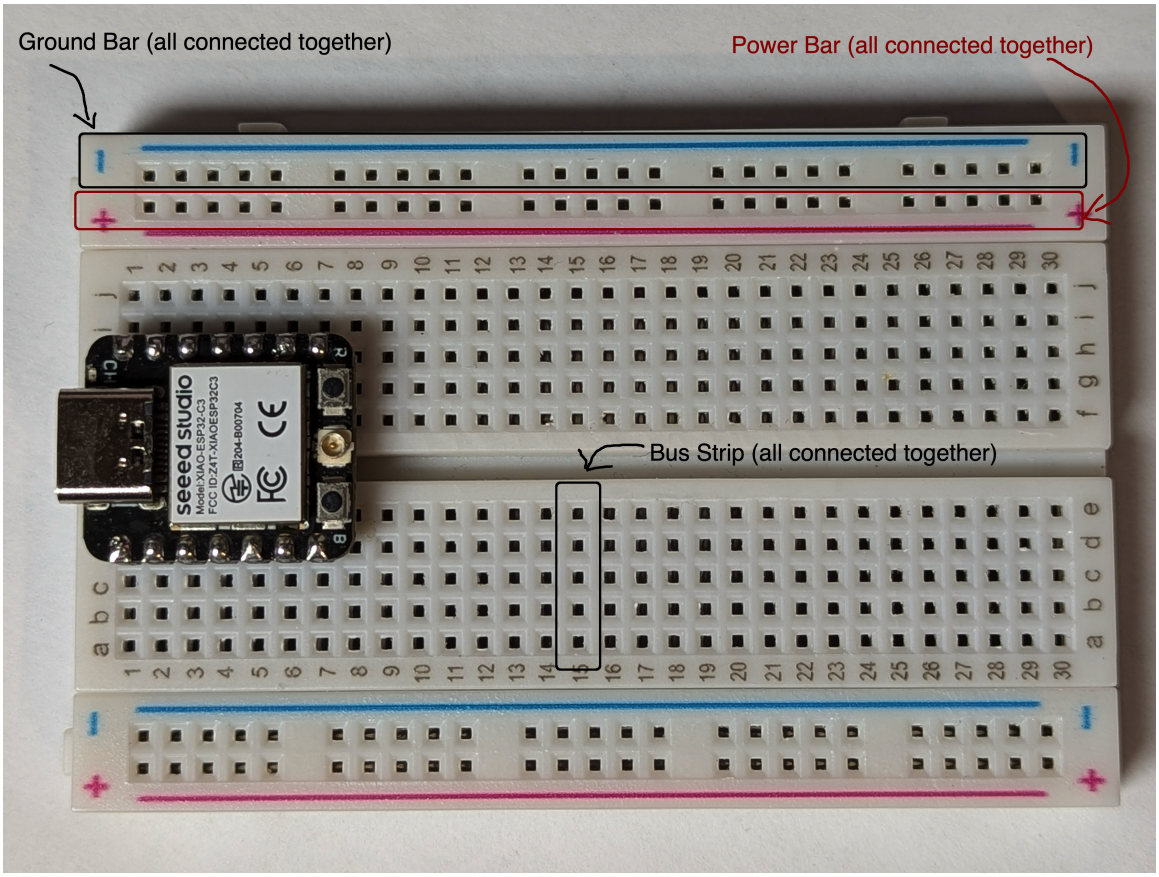
2.1.2 Understanding the layout

Power rails

The long horizontal rows along the top and bottom edges are typically used as power rails to distribute power (positive and negative) throughout the circuit.

Internal connections

The remaining rows are divided into vertical columns, with each column having five holes connected internally. These connections allow you to easily connect multiple components within the same column.



2.2 LEDs: Lighting Up Your Circuits

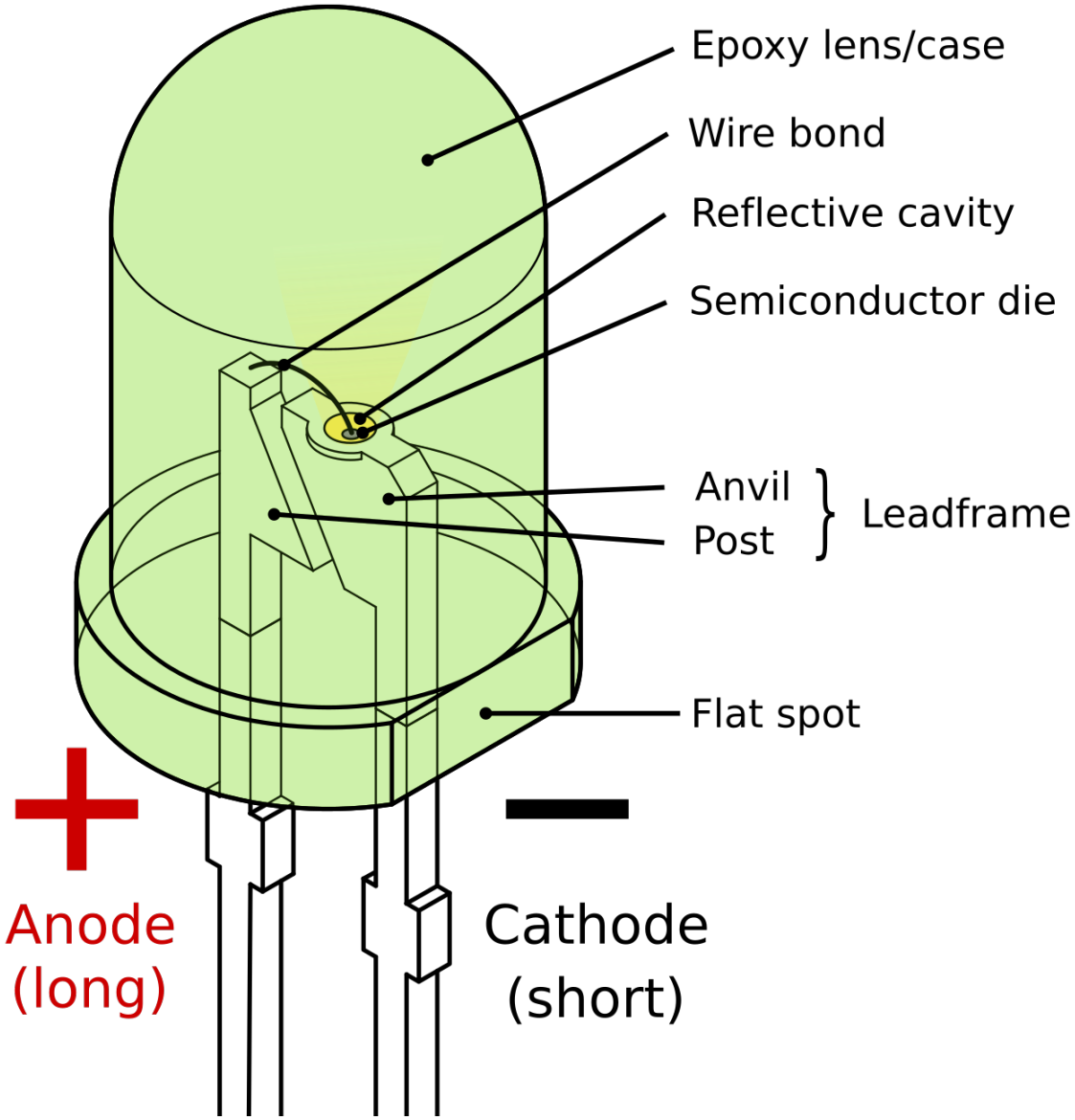
2.2.1 What is an LED?

An LED (Light-Emitting Diode) is a semiconductor device that emits light when an electric current passes through it. They are available in various colors and are commonly used as indicators or displays in electronic circuits.

2.2.2 Polarity

LEDs have two leads:

- **Anode (longer lead):** Connected to the positive (+) side of the power supply.
- **Cathode (shorter lead):** Connected to the negative (-) side of the power supply.



2.2.3 Current limiting resistor

It's crucial to connect a resistor in series with an LED to limit the current flowing through it. Otherwise, the LED may get damaged or burn out. The value of the resistor depends on the LED's specifications and the power supply voltage.

2.3 Buttons: Interacting with Your Circuits

2.3.1 What is a button?

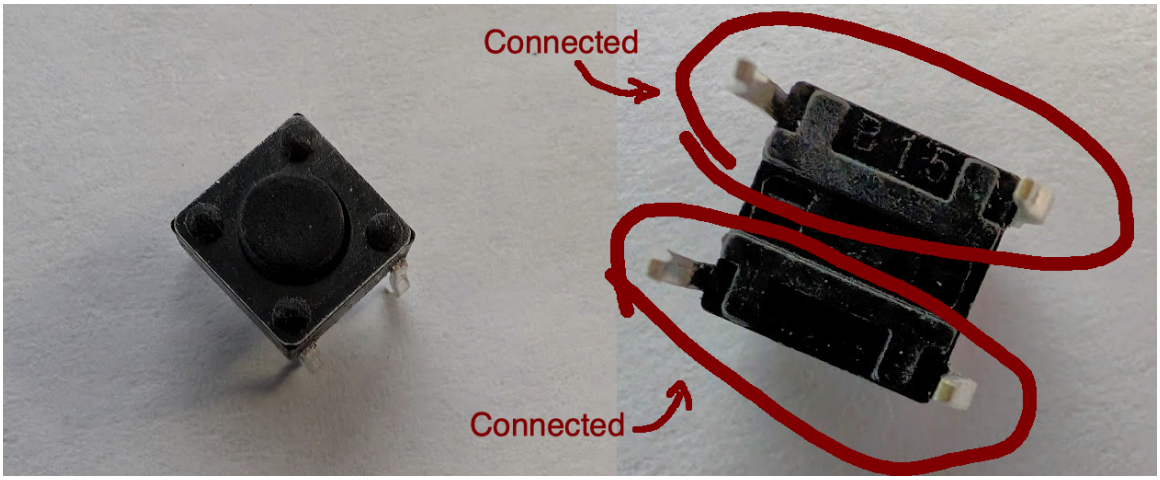
A button, also known as a pushbutton switch, is a simple mechanical device that completes or breaks an electrical circuit when pressed. They are commonly used to trigger actions or provide input to electronic circuits.

2.3.2 Types of buttons

- **Normally open (NO):** The circuit is open (no connection) when the button is not pressed and closes (connection made) when pressed.
- **Normally closed (NC):** The circuit is closed (connection made) when the button is not pressed and opens (no connection) when pressed.

2.3.3 Connected Pins

Many buttons will have more than two pins (often 4). If they do, then often you can see on the bottom that some of those pins will already be connected together. Pay attention to this so that you wire it up the right way.



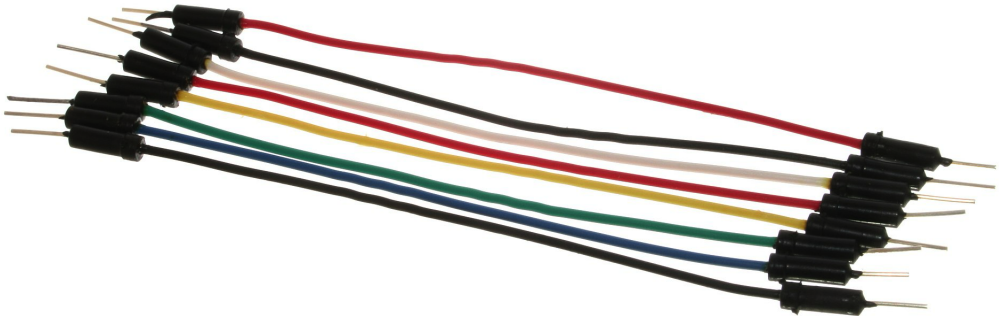
2.4 Jumper Cables: Making Connections

2.4.1 What are jumper cables?

Jumper cables are short wires with connectors at both ends used to make temporary connections between components on a breadboard or between a breadboard and other devices.

2.4.2 Using jumper cables

Insert the connectors into the appropriate holes on the breadboard or device to establish a connection.



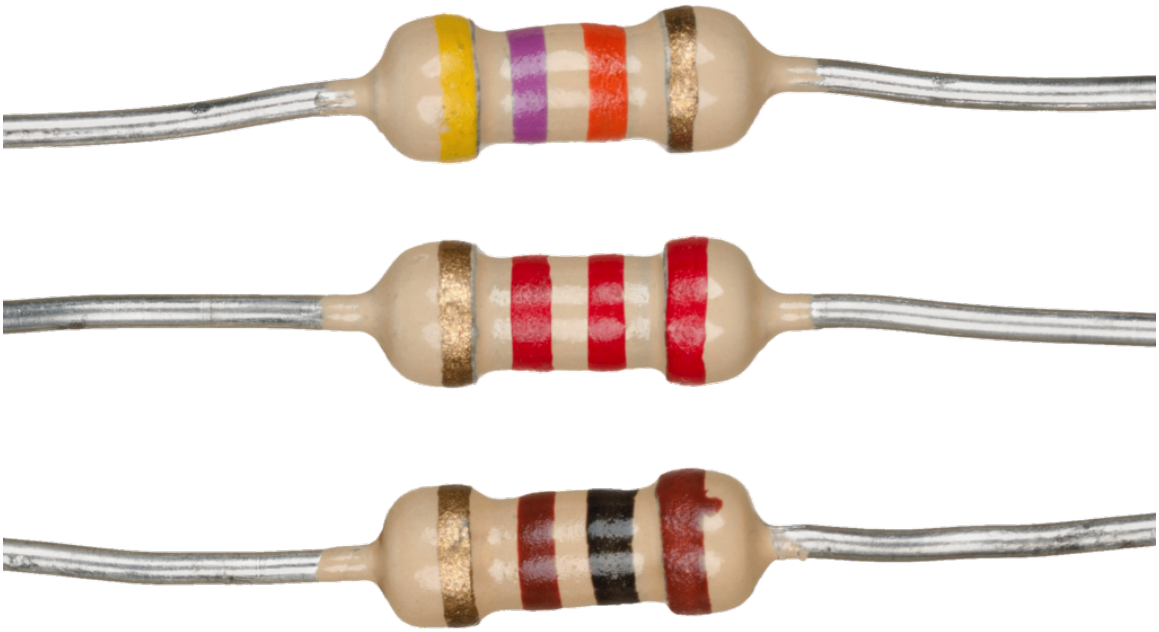
2.5 Resistors: Controlling Current Flow

2.5.1 What is a resistor?

A resistor is a passive component that limits the flow of electric current in a circuit. They are essential for protecting components (like LEDs) from excessive current and for controlling voltage levels in circuits.

2.5.2 Resistance value

The resistance value is measured in ohms (Ω) and is indicated by colored bands on the resistor body. You can use a resistor color code chart to decode the value.



2.6 Safety Tips:

- Always disconnect the power supply before making any changes to the circuit.
- Be careful not to short-circuit the power supply by connecting the positive and negative rails directly.
- If you smell burning or see smoke, immediately disconnect the power supply.

Chapter 3: MicroPython IDE

3.1 Connecting Your Microcontroller

Throughout this guide, we will be making use of a free online IDE for connecting our microcontroller to our computer and loading, editing, and running code on it. You can access the IDE from your browser by going to `https://viper-ide.org/`. Click on the USB icon in the top right and choose your microcontroller from the list:

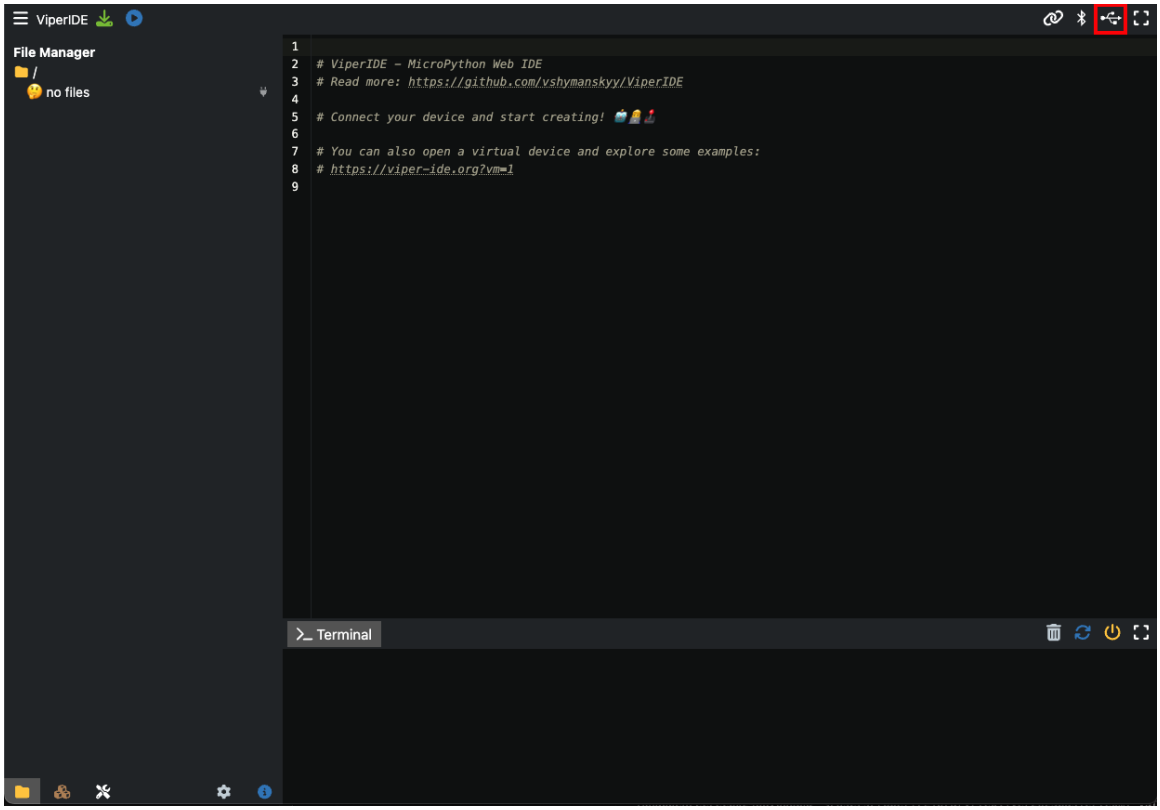


Figure 3.1: Click the button highlighted in red.

If you see multiple items in the dialog that pops up, choose the one that starts with "USB JTAG". See below for an example:

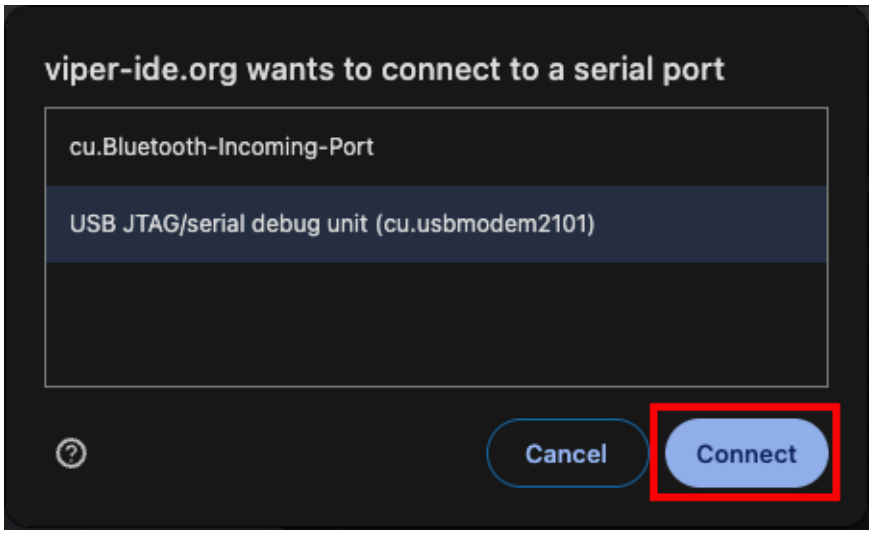


Figure 3.2: Click the button highlighted in red.

Once you have connected, you will see a green dialog labeled "Device connected" and the file manager on the list will populate with the list of files installed on the device:

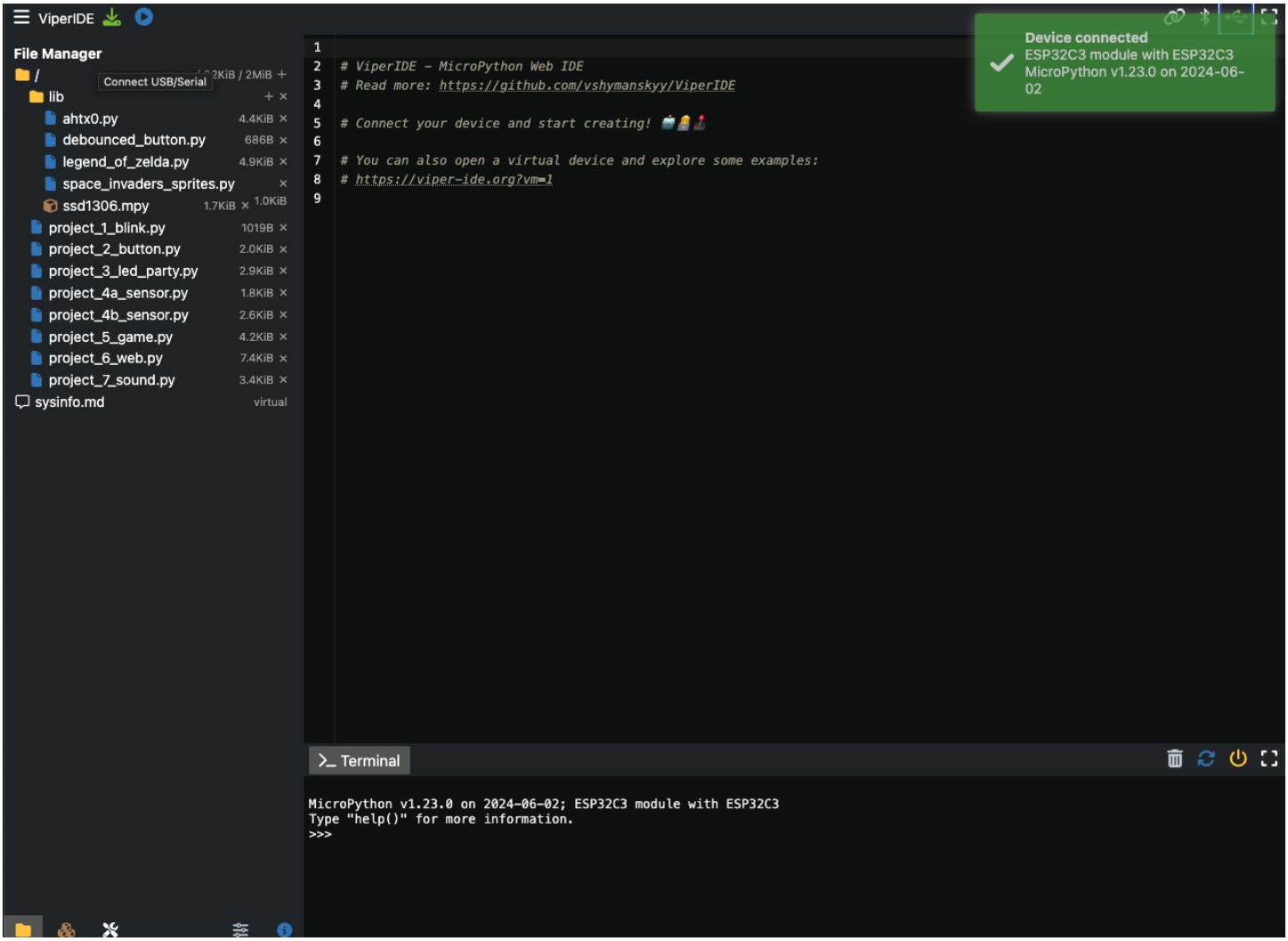


Figure 3.3: Make sure to open the right file for this project

After your device is connected, refer back to the chapter you are working on for the next steps.

Chapter 4: Project 1: Blink

4.1 Overview

This project is designed to provide a foundation for subsequent projects in this book (**and beyond**). Over the course of this project, you will:

- Create a simple circuit using your breadboard
- Write a program that runs in a loop
- Use MicroPython in your program to interact with your microcontroller's GPIO pins.

At the end of this project, your microcontroller should run a MicroPython program which alternates a light between its ON and OFF states. Let's get started!

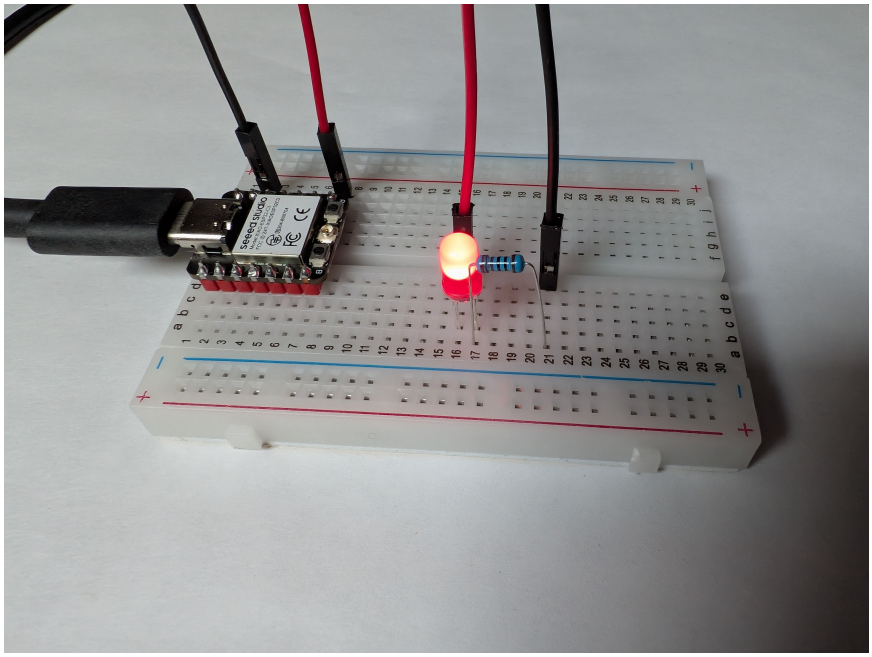


Figure 4.1: The end result should look something like this

4.2 Directions

Remove previous components

Before beginning, remove any components from prior chapters including LEDs, buttons, and wires. You may leave the microcontroller attached to the breadboard.

4.2.1 Creating the circuit

Using jumper cables, you will be assembling a circuit between your microcontroller, your breadboard, an LED, and a 220Ω resistor.

Attach the microcontroller to the breadboard

Carefully insert the pins at the bottom of your microcontroller into the breadboard, making sure that the microcontroller is oriented such that:

- The pin labeled **5V** is inserted in hole at **Column H, Row 1** of the breadboard (or **H1**, for short)
- The pin labeled **GPIO2** is inserted in hole **D1** of the breadboard
- The pin labeled **GPIO20** is inserted in hole **H7** of the breadboard
- the pin labeled **GPIO21** is inserted in hole **D7** of the breadboard

You may need to apply more pressure than expected to seat the microcontroller properly in the breadboard. When its over, it should look like this:

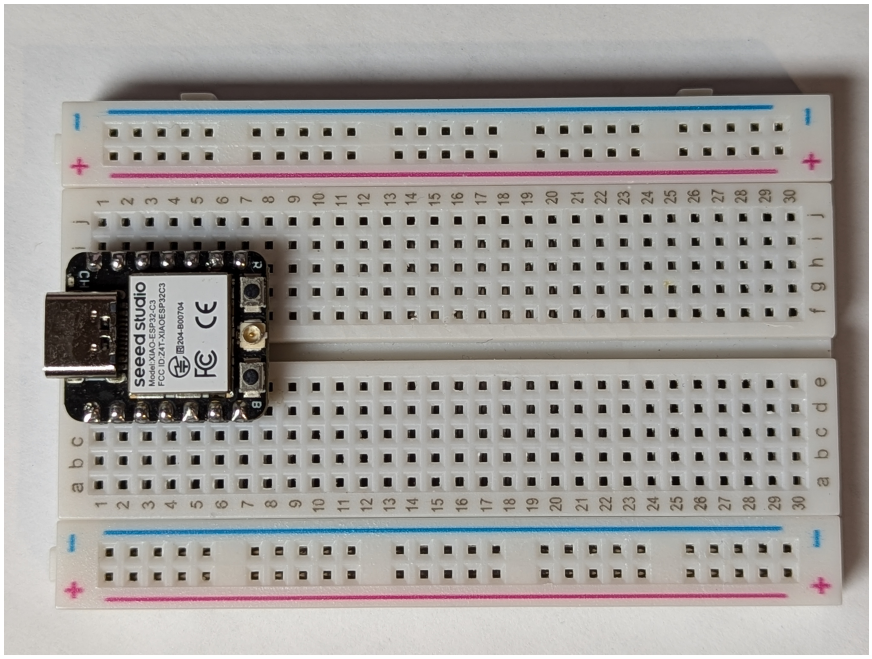


Figure 4.2: So far, so good!

Connect the LED and Resistor

Place an LED of your choice (the example image below uses RED) into the breadboard. The longer leg should be placed in **B16** and the shorter leg should be placed in **B17**. Then place one leg of a resistor (doesn't matter which one) in **A17** and the other in **A21**.

You should be left with something that looks like this:

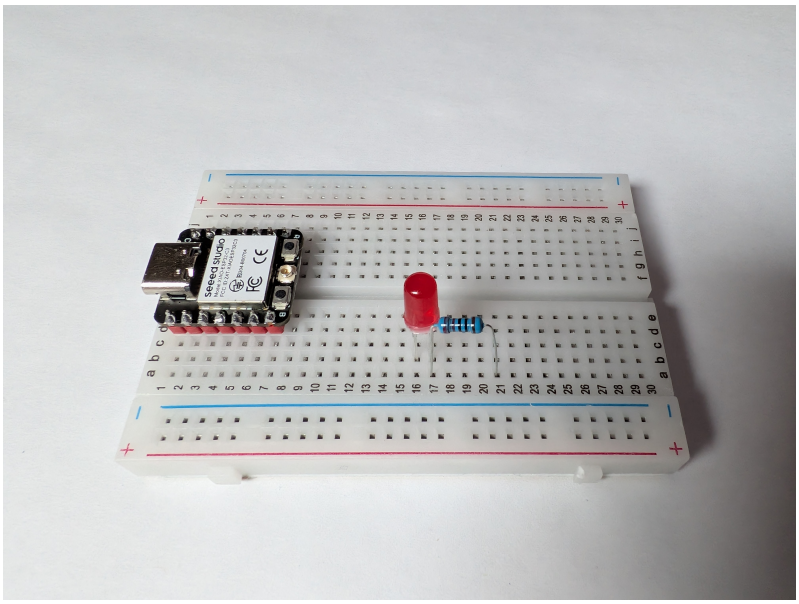


Figure 4.3: All of the components except for the jumper wires are now placed.

NOTE: Whenever you connect an LED to a microcontroller, you must always connect a resistor in series with it. In series means that the power goes in one leg of the LED, the other leg of the LED is attached to one leg of the resistor, and the ground completes the circuit from the other leg of the resistor. This is important so that you do not burn out the LED and/or the microcontroller pin.

Connect the necessary jumper wires

- Place one end of a red jumper wire into hole **J7** of the breadboard and the other end into **E16**. This will provide **3.3** volts of power to the LED when the program turns it on.
- Using a black jumper wire, place one end of the wire into hole **J2** of the breadboard and the other end into **E21**. This will provide a ground path for the LED through the resistor to complete the circuit.

You should be left with something that looks like this:

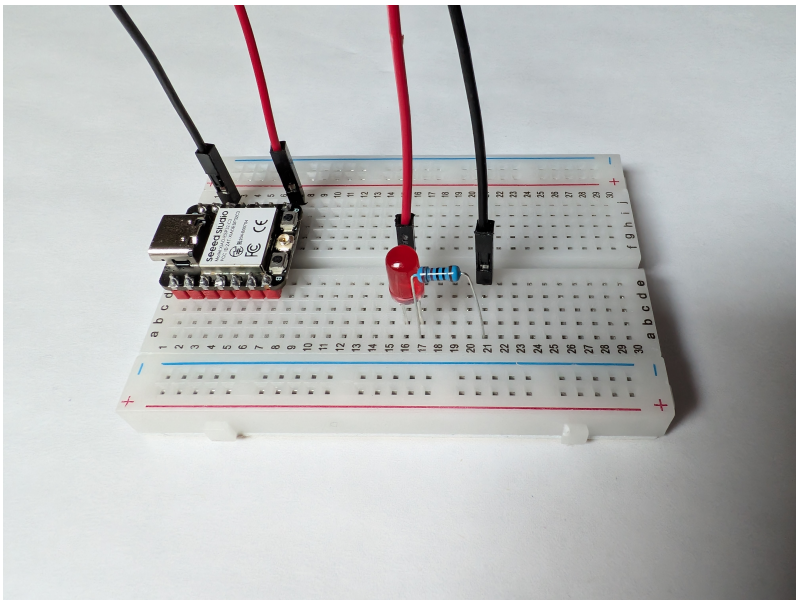


Figure 4.4: I'm absolutely POSITIVE I connected everything correctly!

4.2.2 Programming the microcontroller

Once all of the wiring is correct, connect the USB cable to the microcontroller and load the IDE to access it. Refer back to Chapter 3 for instructions.

Click on the file named "project_1_blink.py". This will load the code in the editor for this section. Read through the comments and the code to get a sense for how it works. Once you are ready, you can click the blue play button in the upper left of the window to start the script.

While the script is running, the LED will blink on and off waiting a half second inbetween each state. You can stop the script by pressing the red stop button located where the blue play button used to be.

4.3 Review

In this project, we learned the basics of setting up a simple circuit and running some code to interact with that circuit. We learned that some pins on the microcontroller can be programmed to turn connected devices on and off. We also learned that connecting an LED in a circuit always requires adding a resistor in series to prevent burning out the components.

4.4 Possible Extensions

If you want to do some experimentation, try these:

- Update the code so that it sleeps for a random amount of time each loop
- Translate a message into morse code and have the LED blink out the message

Chapter 5: Project 2: Button

5.1 Overview

This project adds on to the lessons from the last project. In addition to our LED, we will now also have a button in our circuit. Over the course of this project, you will:

- Create a simple circuit using your breadboard
- Write a program that runs in a loop and listens for user input (the button)
- Respond to user input by changing the brightness of the LED

At the end of this project, your microcontroller should run a MicroPython program which changes the brightness of the LED when the button is pushed. Let's get started!

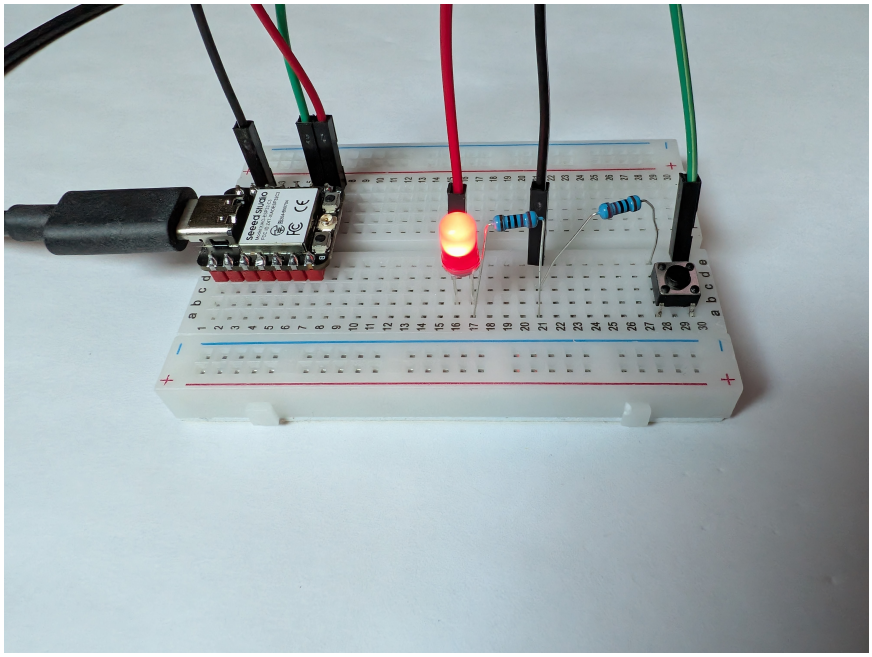


Figure 5.1: The end result should look something like this

5.2 Directions

Remove previous components

Before beginning, remove any components from prior chapters including LEDs, buttons, and wires. You may leave the microcontroller attached to the breadboard.

5.2.1 Creating the circuit

Using jumper cables, you will be assembling a circuit between your microcontroller, your breadboard, an LED, a push-button, and 220Ω resistors for the LED and button.

Attach the microcontroller to the breadboard

Carefully insert the pins at the bottom of your microcontroller into the breadboard, making sure that the microcontroller is oriented such that:

- The pin labeled **5V** is inserted in hole at **Column H, Row 1** of the breadboard (or **H1**, for short)
- The pin labeled **GPIO2** is inserted in hole **D1** of the breadboard
- The pin labeled **GPIO20** is inserted in hole **H7** of the breadboard
- the pin labeled **GPIO21** is inserted in hole **D7** of the breadboard

You may need to apply more pressure than expected to seat the microcontroller properly in the breadboard. When its over, it should look like this:

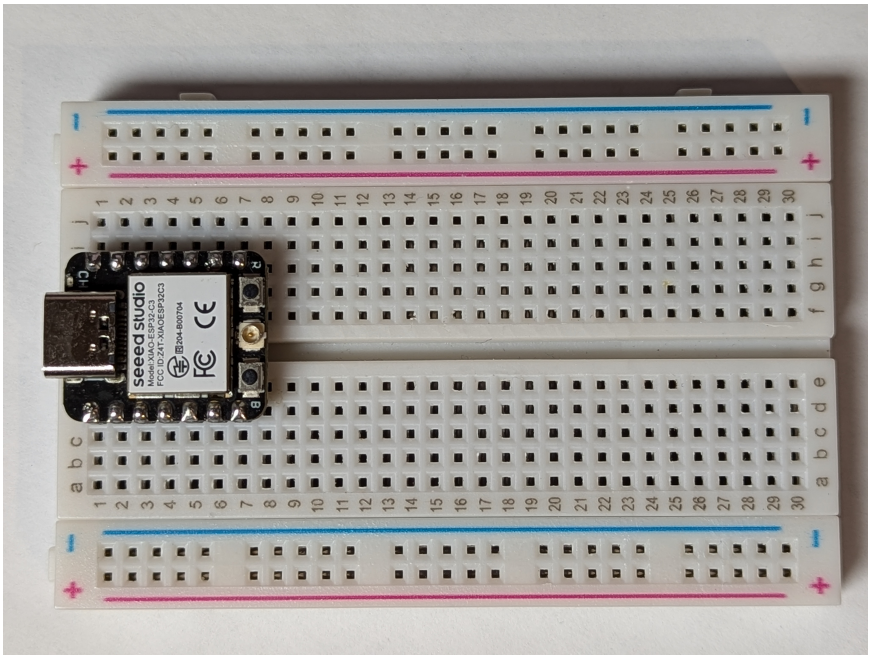


Figure 5.2: So far, so good!

Connect the LED, Button, and Resistors

- Place an LED of your choice (the example image below uses RED) into the breadboard. The longer leg should be placed in **B16** and the shorter leg should be placed in **B17**.
- Place one leg of a resistor (doesn't matter which one) in **A17** and the other in **A21**.
- Place the button so that one connected set of pins (refer to 2.3 for an example) is in **C28** and **A28** and the other set is in **C30** and **A30**.
- Place a resistor between **E28** and **C21**. This will provide the low signal that the button can switch.
- Finally place a resistor between **E30** and the top negative rail (the blue one).

You should be left with something that looks like this:

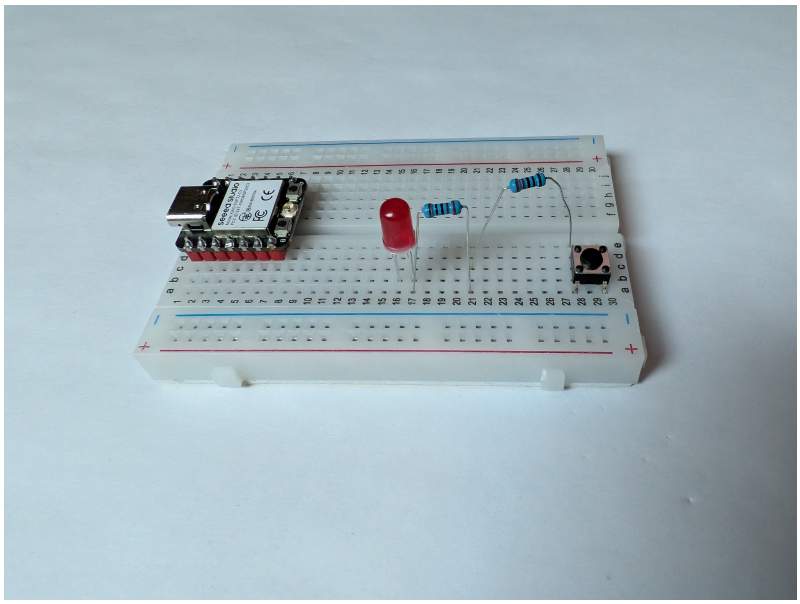


Figure 5.3: All of the components except for the jumper wires are now placed.

Connect the necessary jumper wires

- Place one end of a red jumper wire into hole **J7** of the breadboard and the other end into **E16**. This will provide **3.3** volts of power to the LED when the program turns it on.
- Using a black jumper wire, place one end of the wire into hole **J2** of the breadboard and the other end into **E21**. This will provide a ground path for the LED through the resistor to complete the circuit.
- Place a green jumper wire into hole **J6** and the other end into **E30**. This will provide the signal to the microcontroller that the button has been pressed.

You should be left with something that looks like this:

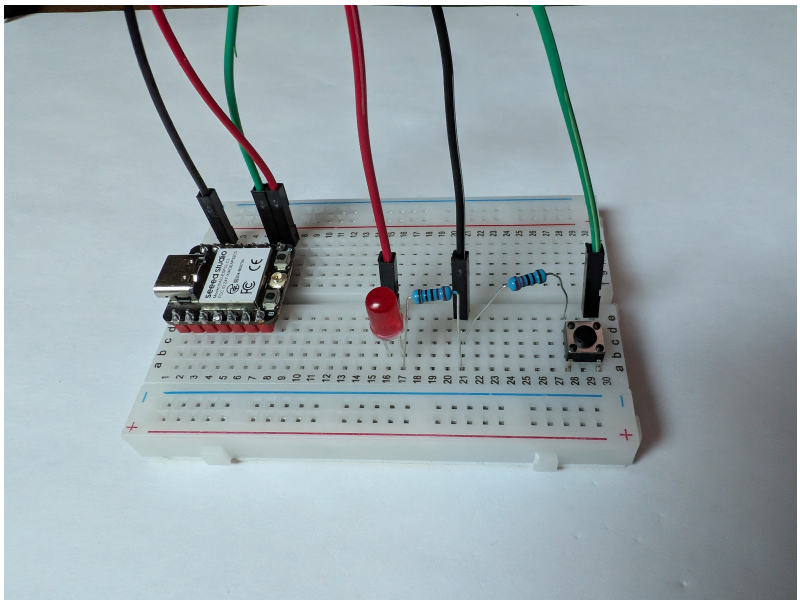


Figure 5.4: All of the components except for the jumper wires are now placed.

5.2.2 Programming the microcontroller

Once all of the wiring is correct, connect the USB cable to the microcontroller and load the IDE to access it. Refer back to Chapter 3 for instructions.

Click on the file named "project_2_button.py". This will load the code in the editor for this section. Read through the comments and the code to get a sense for how it works. Once you are ready, you can click the blue play button in the upper left of the window to start the script.

While the script is running, the LED will toggle between off and 3 different brightness levels. It will change whenever you press the button.

5.3 Review

In this project, we learned how we can add interactivity to a circuit by using momentary pushbuttons. These buttons can be read from the microcontroller's code to run a piece of code (called an interrupt) whenever they are pressed.

5.4 Possible Extensions

If you want to do some experimentation, try these:

- Add more brightness levels for the LED to change between
- Update the code so that it listens for a secret series of button presses before it will turn the LED on

Chapter 6: Project 3: Pig Game

6.1 Overview

This project adds on to the lessons from the last project. In addition to our single LED and button, we'll now have a more complex seven-segment display and a second button to work with. Over the course of this project, you will:

- Expand the circuits from previous projects with more components
- Wire up a complex component that requires many input wires
- Write a program that will play a simple two-player game using the components

At the end of this project, your microcontroller should run a MicroPython program that allows two players to take turns rolling a die. The first player to reach 100 points wins the game.

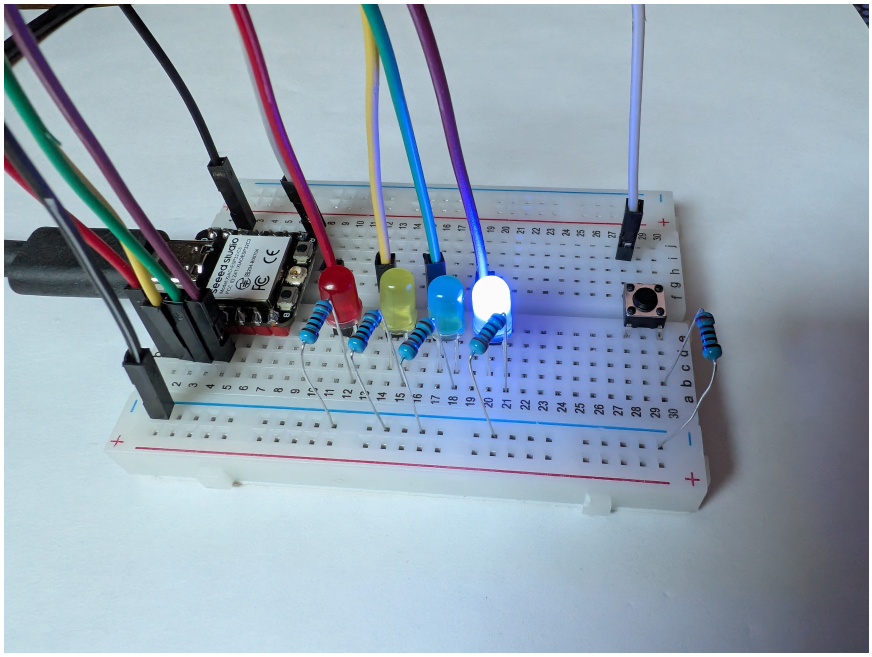


Figure 6.1: The end result should look something like this

6.2 Directions

Remove previous components

Before beginning, remove any components from prior chapters including LEDs, buttons, and wires. You may leave the microcontroller attached to the breadboard.

6.2.1 Creating the circuit

Using jumper cables, you will be assembling a circuit between your microcontroller, your breadboard, some LEDs, a pushbutton, and a 220Ω resistor for the display.

Attach the microcontroller to the breadboard

Carefully insert the pins at the bottom of your microcontroller into the breadboard, making sure that the microcontroller is oriented such that:

- The pin labeled **5V** is inserted in hole at **Column H, Row 1** of the breadboard (or **H1**, for short)
- The pin labeled **GPIO2** is inserted in hole **D1** of the breadboard
- The pin labeled **GPIO20** is inserted in hole **H7** of the breadboard
- the pin labeled **GPIO21** is inserted in hole **D7** of the breadboard

You may need to apply more pressure than expected to seat the microcontroller properly in the breadboard. When its over, it should look like this:

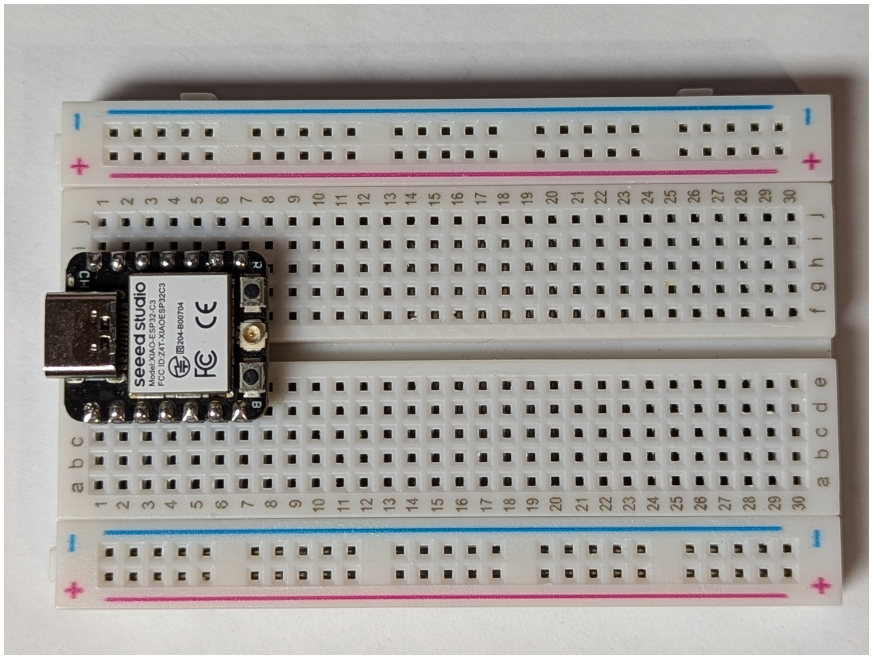


Figure 6.2: So far, so good!

Connect the Display, Buttons, and Resistor

- Place the seven-segment display in the board with the top left pin in **G13**, the top right pin in **G17**, the bottom left pin in **C13**, and the bottom right pin in **C17**.
- Place a button button so that one connected set of pins (refer to 2.3 for an example) is in **E24** and **F24** and the other set is in **E26** and **F26**.
- Place a second button so that one connected set of pins is in **E28** and **F28** and the other set is in **E30** and **F30**.
- Finally place a resistor between **I15** and the top negative rail.

You should be left with something that looks like this:

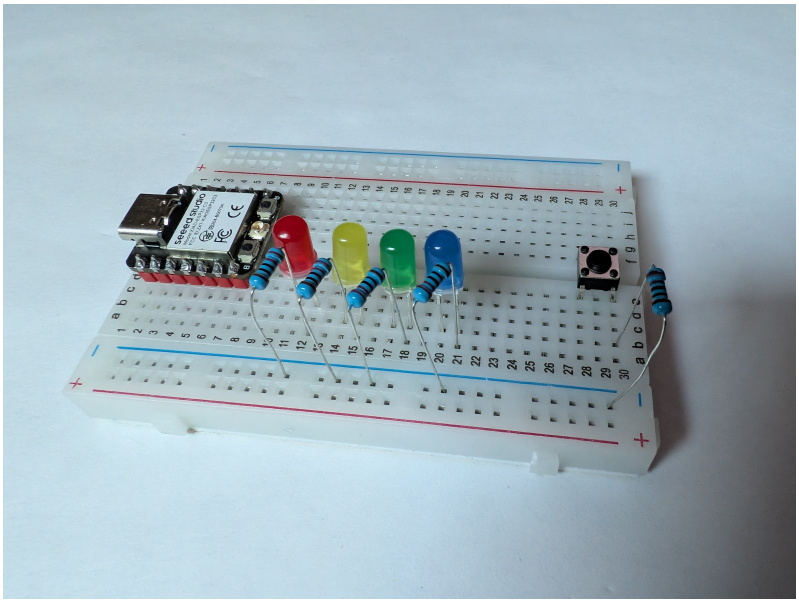


Figure 6.3: All of the components except for the jumper wires are now placed.

Connect the necessary jumper wires

- Place one end of a black jumper wire into hole **J2** of the breadboard and the other end into the top negative rail (the blue one). This will provide a ground path for all of the components.
- Place a yellow jumper wire between **J4** and **J13**.
- Place a green jumper wire between **J5** and **J14**.
- Place a blue jumper wire between **J6** and **J16**.
- Place a purple jumper wire between **B6** and **J17**.
- Place a yellow jumper wire between **B5** and **A16**.
- Place a green jumper wire between **B4** and **A14**.
- Place a purple jumper wire between **B3** and **A13**.
- Place a white jumper wire between **B2** and **J30**.
- Place a white jumper wire between **B1** and **J26**.
- Place a black jumper wire between **J24** and the top negative rail.
- Place a black jumper wire between **J28** and the top negative rail.

You should be left with something that looks like this:

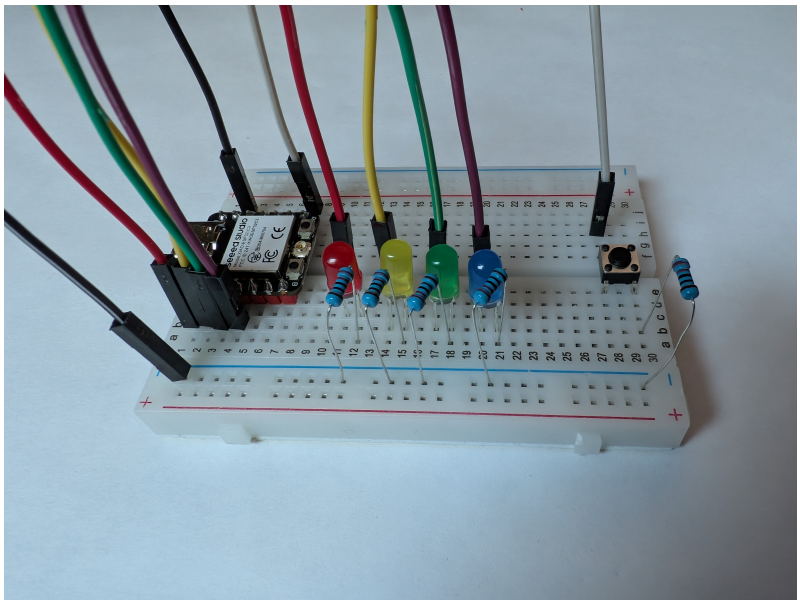


Figure 6.4: All components and wires are now installed

6.2.2 Programming the microcontroller

Once all of the wiring is correct, connect the USB cable to the microcontroller and load the IDE to access it. Refer back to Chapter 3 for instructions.

Click on the file named "project_3_pig_game.py". This will load the code in the editor for this section. Read through the comments and the code to get a sense for how it works. Once you are ready, you can click the blue play button in the upper left of the window to start the game.

The game will start with player 1 rolling the die using the right-most button. As long as the player does not roll a 1, they can continue to accumulate points. If they roll a 1, they lose all of their points accumulated during that turn and the next player gets a turn. If they roll any other number, they can choose to end their turn and save the points by pressing the left button. Play continues to pass back and forth until one player reaches the win limit of 100 saved points and wins the game.

6.3 Review

In this project, we learned how we can add interactivity to a circuit by using momentary pushbuttons. These buttons can be read from the microcontroller's code to run a piece of code (called an interrupt) whenever they are pressed.

6.4 Possible Extensions

If you want to do some experimentation, try these:

- Add an indicator to show on the board which player's turn it is. You might add some LEDs from the kit, or maybe use the decimal point on the seven-segment display.
- Try changing the code to implement one or more of the variations on the Pig game: `https://en.wikipedia.org/wiki/Pig_(dice_game)#Variations`

Appendix A: Python Primer

If you're new to Python, this section will give you a few things you should know in order to better understand the projects in this guide. This is by no means a complete or comprehensive look at the Python language. For that, we recommend looking at the official Python site and reading through the tutorial there.

> Note: for the projects being used here, we are using an implementation of Python known as MicroPython. This version is meant to run on microcontrollers with limited resources. It also has built into it libraries for dealing with hardware devices that are not part of the standard CPython distribution. Therefore, not all Python examples you find online will run on your microcontroller and not all projects for a microcontroller can be run on your computer. But a lot of the code can be shared so the lessons you learn here can apply to other Python projects.

Here is a sample of a small Python script. We will disect and explain what each section does below:

Listing A.1: An example Python script

```
1  def show(message, repeat=1):
2      """This function prints the given message to the
3      console as smany times as specified in the
4      srepeat parameter.
5      """
6
7      for iteration in range(0, repeat):
8          print(iteration, message)
9
10 name = input("What is your name: ")
11 show(name)
12 show(name, repeat=3)
```

On line 1, we are defining a function named show. This function accepts two parameters, message and repeat. The message parameter is required and the repeat parameter is optional with a default value of 1.

Lines 2 through 5 comprise the docstring for the function. This information is meant for programmers to read and explains what the function does. It does not affect how the function works.

Line 7 starts a loop. The loop will repeat the statements in the loop body until a condition is met. In this case, it will loop until it has performed the operation for each repeat.

Line 8 is the body of the loop. This statement will print the message that the user passed in to the console along with the iteration number of the loop.

Line 10 prompts the user for their name and saves the result in a variable called name.

Line 11 calls our show function which will print the user's name once (the default).

Line 12 calls our show function again, this time saying that we want to repeat the loop of printing the name twice.

Running the program, we will see output like this:

```
$ python program.py
What is your name: Emily
0 Emily
0 Emily
1 Emily
2 Emily
$
```

Another feature that you'll see used often in Python are classes. Classes are a convienent way to model something in your program that holds state and implements functionality. For example, let's say that we are writing a game about racing go- karts. We need to allow each player to have their own kart and keep track of how fast it is going, which way they are turning, and allow the kart to speed up and slow down. Here is a small class that will help us do that:

Listing A.2: An example of a Python class

```
1  class Kart:
2      MAXIMUM_SPEED = 100
```

```
3
4       def __init__(self):
5           """The kart starts motionless at the beginning"""
6           self._speed = 0
7           self._direction = 0
8           self._acceleration = 0
9
10      def brake(self):
11          """This is called when the user presses the brake button"""
12          self._acceleration = -5
13
14      def accelerate(self):
15          """This is called when the user presses the accelerator button"""
16          self._acceleration = 5
17
18      def steer(self, direction):
19          """This is called when the user presses left or right"""
20          self._direction = direction
21
22      def update(self, ticks):
23          """Update will be called by our game engine and will be
24          provided the number of ticks since it was last called.
25          """
26
27          self._speed += self._acceleration * ticks
28
29          # limit our speed so that we don't go faster than our
30          # kart is allowed to, or slower than 0
31          if self._speed > Kart.MAXIMUM_SPEED:
32              self._speed = Kart.MAXIMUM_SPEED
33          if self._speed < 0:
34              self._speed = 0
```

Looking at this class, there are 5 methods. The first one (on line 4) is a special method that is called by the Python interpreter whenever a new Kart is created. It will initialize some variables for this particular Kart object.

You may have noticed that the first method takes a parameter called "self". This is the first parameter of all methods in a class in Python. It is automatically passed by the interpreter and is a reference to the current object. It lets us access the variables that belong to the class, like those we defined in the __init__ method.

Speaking of the variables in the __init__ method. Notice how we named them all with an underscore? This is a convention in Python that says they are private to our class and that code written outside of the class shouldn't access them directly. That means that our class should provide ways to modify or read these variables via other methods.

The second method starts on line 10. This is called when the player presses the brake button on their controller and will set our Kart's acceleration to a negative value so that we start to slow down. It modifies the private _acceleration member of the class.

The third starts on line 14. It is the opposite of braking and will start speeding our Kart up when the user presses the accelerator. It aslo modifies the private _acceleration member of the class.

The fourth method, line 18, is again something to deal with user input. This time we can see that it takes a second parameter, direction. If the user presses left on their controller, then we can expect left to be passed here. The same for right. We will modify the private _direction member here.

Finally, we have a fifth method starting on line 22. This method is called by our game engine and uses the class members to determine what happens to the Kart throughout the game. That is, it is asking the Kart to update itself at a certain moment in time (usually once per frame) so that next time it draws it to the screen, it will be in the updated location.

Notice in the last method, we are accessing not only our own variables, _speed, and _acceleration, but we are also reading a class variable, Kart.MAXIMUM_SPEED. Unlike our member variables, a class variable is the same for all instances of a class. It is useful here to keep the game fair so that all Karts have the same limitation on their speed.