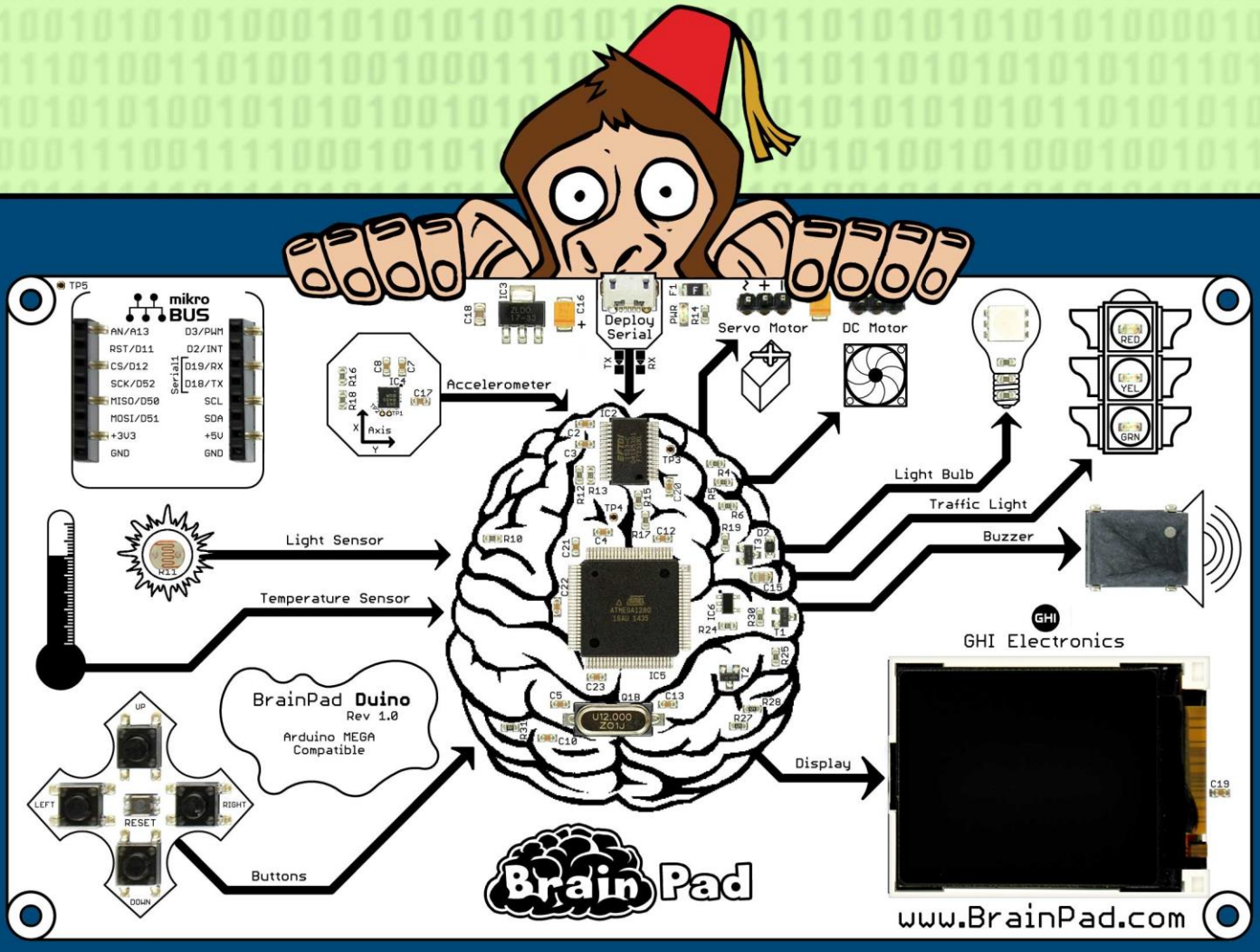
GHI electronics

BrainPad

C++

INTRODUCTION



Contents

Introduction

The BrainPad circuit board is designed as a powerful educational tool that can be used to teach everyone from kids, to college students and professionals. Kids will start to learn programming using Visual Studio, one of the most widely used professional tools. College students and professionals that already know programming can use the BrainPad circuit board to learn about digital electronics and the connection between computing and the physical world.

Overview

Students will learn how to create projects in Arduino Software (IDE) along with programming basics. Programs in this lesson will be limited to a `while` loop, an `if` statement and the `BrainPad` object.

Guidelines

- Prerequisites: None
- Ages 12 and up
- PC setup with Arduino Software (IDE) and the BrainPad Driver zip file.
- Supplies: BrainPad

Starting a New Project

Open Arduino and you'll automatically be presented with a Sketch file named after the current date. In order to use the BrainPad, you'll need to include Servo.h, SPI.h, Wire.h and BrainPad.h as shown in Example 1.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>
```

Example 1 – Include these files at the top of every BrainPad Duino Sketch.

These files will provide the functionality needed to use the `BrainPad` object. So far, your Sketch should look similar to Figure 1.

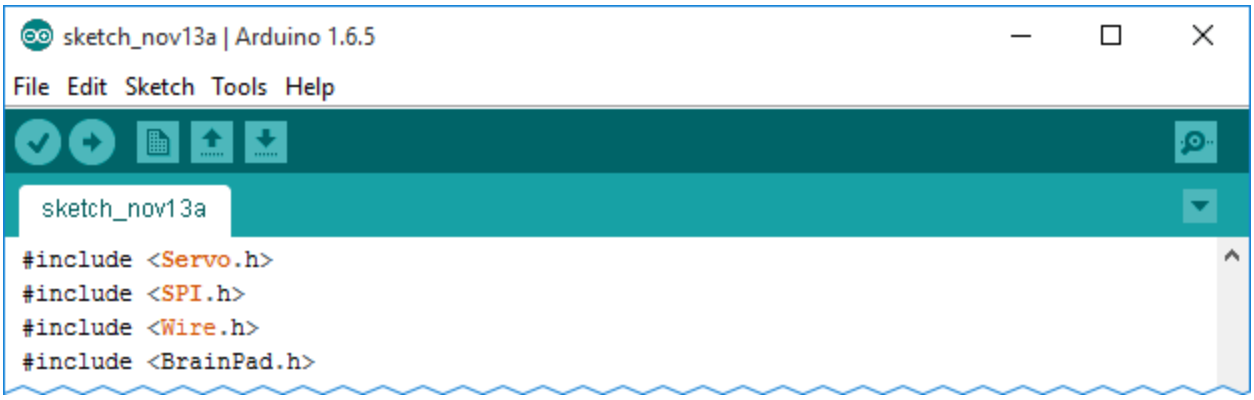


Figure 1 – Include these files to use the `BrainPad` object.

In every Sketch you'll find two functions, one named `setup()` and the other named `loop()`. The `setup()` function is where you place code to set things up. Once the `setup()` function is executed, the `loop()` function starts being called continuously (looping).

We only need to use the `setup()` function (Example 2) at this time, so inside of it add the highlighted code to setup the serial communication and to print the string “Hello World!”

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>

void setup() {
  Serial.begin(9600);
  Serial.print("Hello world!");
}

void loop() {
}
```

Example 2 – Here we tell the program to print “Hello World!” during setup.

Now we’re ready to run our program, but before we can run this code, we’ll need to make sure the Arduino Software has the proper settings for BrainPad Duino. In the menu under Tools, make sure the *Board* is set to “Arduino/Genuino Mega or Mega 2560” and the *Processor* is set to “ATmega1280” as shown in Figure 2.

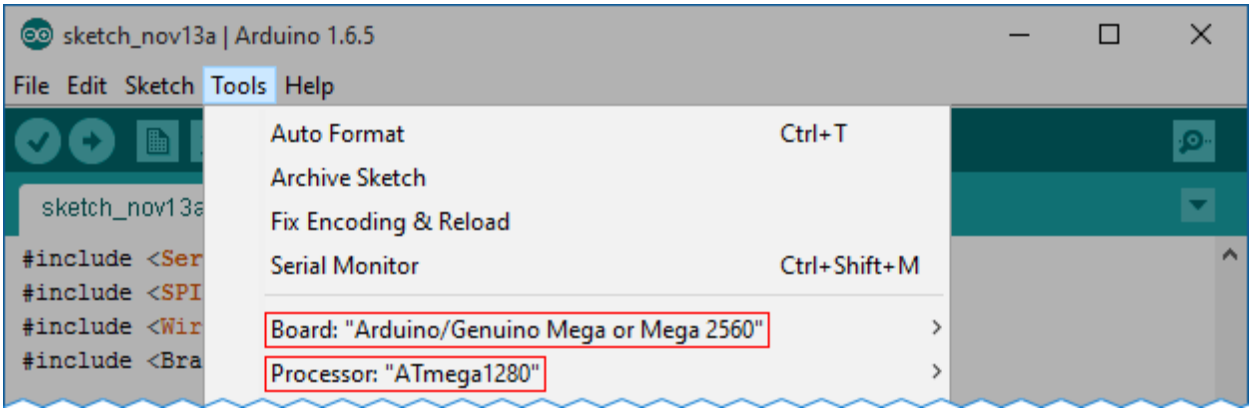


Figure 2 – Set the Board and Processor settings to the values above.

Next, under Sketch select Verify / Compile (Figure 3) to assemble our code and check for errors. In general, it's useful to run every now and then while coding.

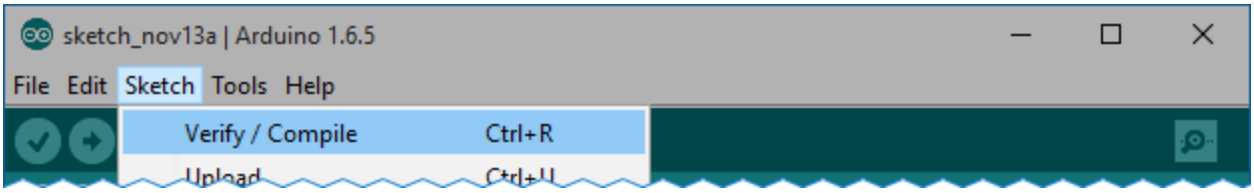


Figure 3 – Select Verify / Compile to assemble the code and send it to the BrainPad Duino.

When compiling finishes, you should see the same output as shown in Figure 4.

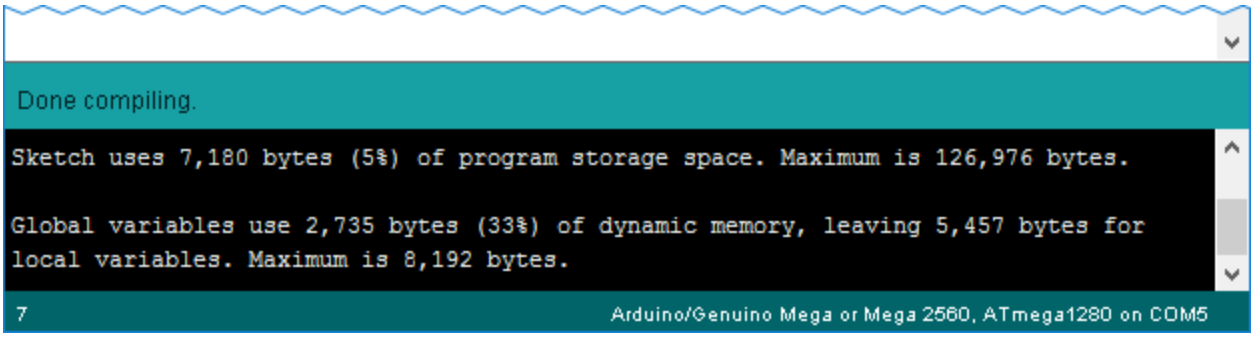


Figure 4 – Successful compiling shows no error messages.

Great! Now we're ready to upload our code. In order to see the string this code prints, we need to open the Serial Monitor. Go to Tools > Serial Monitor as shown in Figure 5.

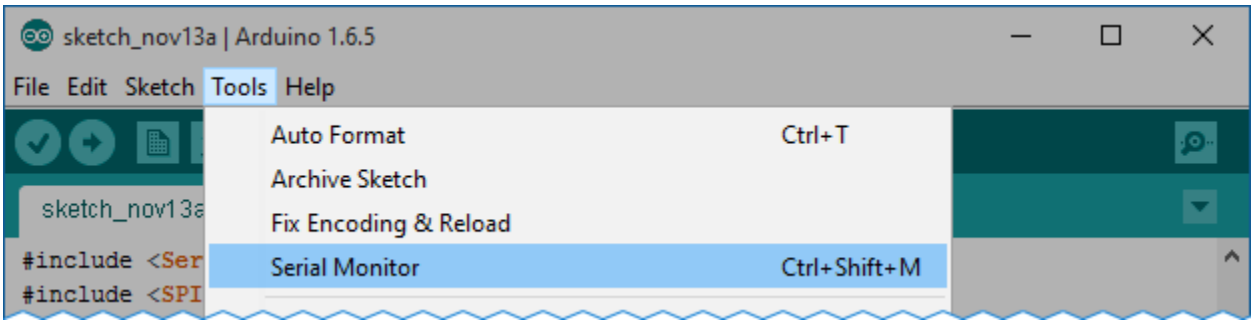


Figure 5 – Select Serial Monitor to open the corresponding window.

This will open another window (Figure 6) that will display serial output.

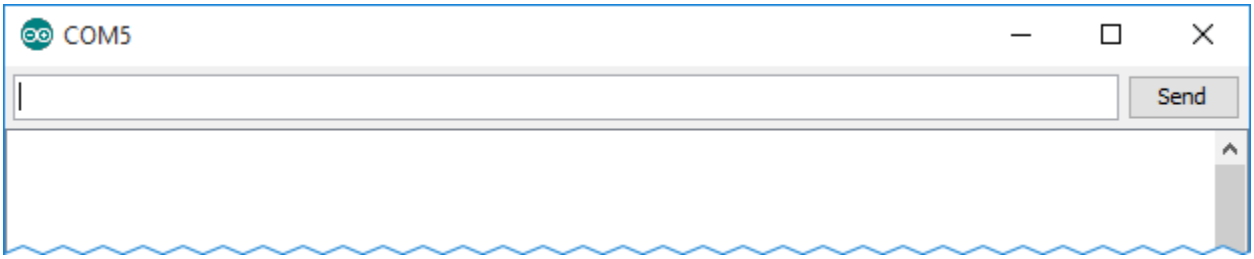


Figure 6 – The Serial Monitor window will show all serial output.

In the menu select Sketch > Upload (Figure 7) to send your code to the BrainPad Duino.

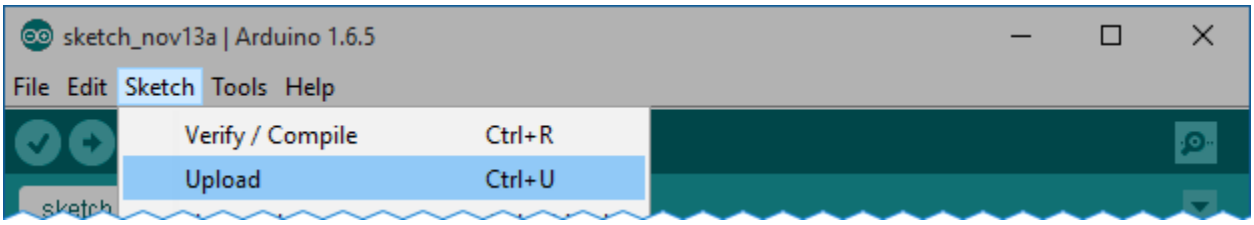


Figure 7 – Select Upload to transfer your code to the BrainPad Duino.

Uploading first recompiles the code to check for errors. If no errors are found, the code begins to upload.

During this time, you'll see the TX and RX lights blinking on the BrainPad Duino. This indicates the code is being transmitted as shown in Figure 8.

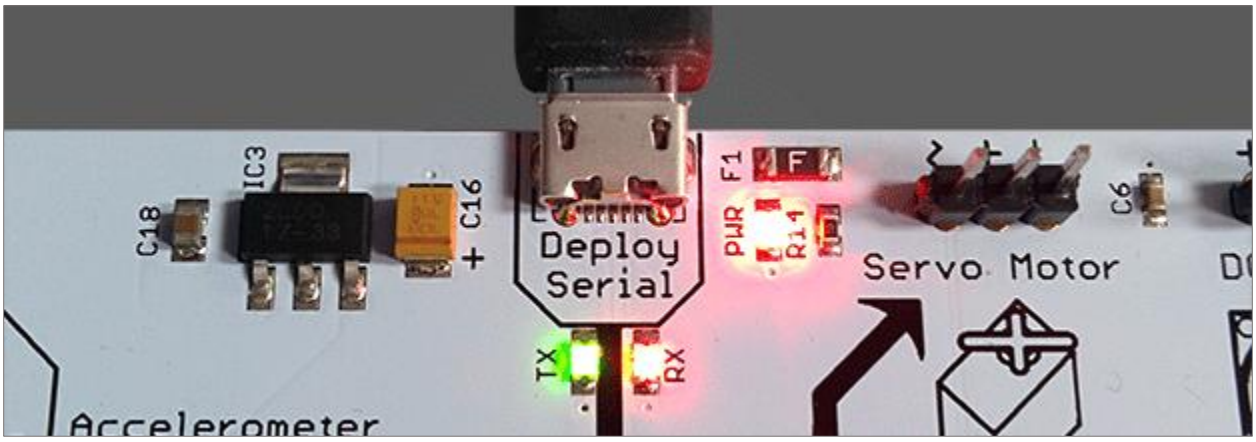


Figure 8 – The TX and RX lights blink as data is transferred.

Once the code is transferred (about six seconds) the board will reset and shortly after you will see "Hello World!" appear in the Serial Monitor window (Figure 9).

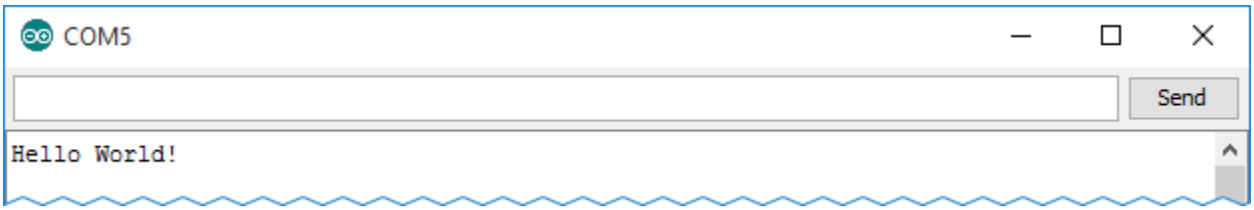


Figure 9 – Here we see "Hello World!" printed by our code.

Congratulations, you just wrote your first program! Next, we'll learn how to control the BrainPad Duino using the `BrainPad` object.

Exercise

Change the text to print your name in the Serial Monitor window.

The BrainPad Object

The `BrainPad` object is a combination of code that was developed by GHI Electronics to cover the internals of the BrainPad. To the student, the `BrainPad` is just an object that can be easily controlled. For example, let's open a new Sketch by clicking File > New. Now we can use the code below in Example 3 to turn the green light on.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>

void setup() {
  BrainPad.TrafficLight.TurnGreenLightOn();
}

void loop() {
}
```

Example 3 – This code turns the traffic light's green light on.

The program can now be uploaded just like before by selecting Sketch > Upload. In a few seconds, the green light will come on as shown in Figure 10.

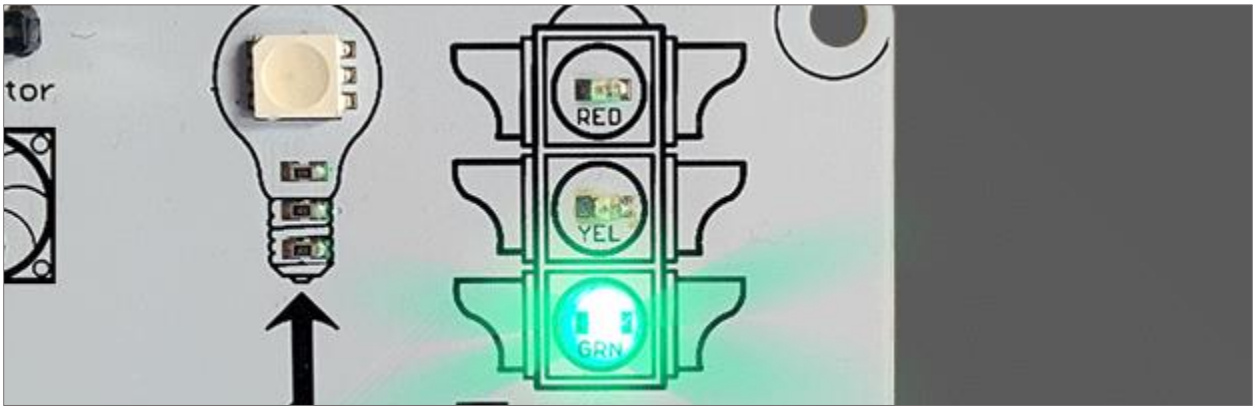


Figure 10 – The traffic light's green light is on.

Exercise

Turn the green light on, then off.

Slowing Things Down

If you did the last exercise you'd notice that the green light never came on. This is because programs execute faster than our eyes can see. That's why we need to slow things down so we can see what's happening. This allows us to confirm the results match what we expect to happen.

To fix this, we'll need to wait after turning the green light on. This is accomplished by adding the wait code as shown in Example 4.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>

void setup() {
  BrainPad.TrafficLight.TurnGreenLightOn();
  BrainPad.Wait.Seconds(1);
  BrainPad.TrafficLight.TurnGreenLightOff();
}

void loop() {
}
```

Example 4 – Wait one second between turning the green light off.

Uploading the code from Example 4 will result in the green light coming on for a second then shutting off.

Exercise

Create a real traffic light using the following logic:

1. Yellow and red lights turns off.
2. Green light turns on.
3. Wait 5 seconds.
4. Green light turns off.
5. Yellow light turns on.
6. Wait 1 second.
7. Yellow light turns off.
8. Red light turns on.
9. Wait 5 seconds.

The While Loop

The previous exercise simulated a traffic light but it only did it once. We could repeat the code over and over to make the traffic light run a few more times but what if we wanted this to run forever? This is where loops come in and the ideal loop for this task is the `while` loop as shown in Example 5.

```
while (BrainPad.Looping) {
  // Do something
}
```

Example 5 – This code will loop indefinitely.

Any code between the two curly brackets will be executed indefinitely. We will keep things simple and blink the green light (Example 6) for half a second on and half a second off.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>

void setup() {
  while (BrainPad.Looping) {
    BrainPad.TrafficLight.TurnGreenLightOn();
    BrainPad.Wait.Seconds(0.5);
    BrainPad.TrafficLight.TurnGreenLightOff();
    BrainPad.Wait.Seconds(0.5);
  }
}

void loop() {
}
```

Example 6 – This code will blink the green light on and off for half a second indefinitely.

Note: Code after the `while` loop will never be reached and the program will never end.

In our Sketch, we also have the predefined function called `loop()`. This function acts the same as a `while` loop. If we simply move our code from the `while` loop to the `loop()` function, we will get the same results as shown in Example 7.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>

void setup() {
}

void loop() {
    BrainPad.TrafficLight.TurnGreenLightOn();
    BrainPad.Wait.Seconds(0.5);
    BrainPad.TrafficLight.TurnGreenLightOff();
    BrainPad.Wait.Seconds(0.5);
}
```

Example 7 – Here we use the predefined `loop()` function to achieve the same results.

Exercise

Make the traffic light simulate a real traffic light indefinitely.

The If Statement

An `if` statement (or conditional statement) checks to see if a statement is true or false and then does one of two things depending on the result. In Example 8 below, the continuous `loop()` function checks if the down button is pressed, and if it is, turns the green LED on.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>

void setup() {
  BrainPad.TrafficLight.TurnGreenLightOff();
}

void loop() {
  if (BrainPad.Button.IsDownPressed())
    BrainPad.TrafficLight.TurnGreenLightOn();
}
```

Example 8 – Pressing the down button will turn the green light on permanently.

However, this program still has a problem or what programmers call a bug. When the down button is released, the green light stays on.

Can you guess why?

Computers are strict on following orders. In the previous example, the green LED never turned off because we never told the program to turn it off.

We can accomplish this by using the exclamation symbol which represents “not” or false in an `if` statement. While `BrainPad.Button` doesn’t have a method called `IsDownNotPressed()` we can continue to use `BrainPad.Button.IsDownPressed()` but with an exclamation before it to represent “if the button is not pressed” as shown in Example 10.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>

void setup() {
  BrainPad.TrafficLight.TurnGreenLightOff();
}

void loop() {
  if (BrainPad.Button.IsDownPressed())
    BrainPad.TrafficLight.TurnGreenLightOn();
  if (!BrainPad.Button.IsDownPressed())
    BrainPad.TrafficLight.TurnGreenLightOff();
}
```

Example 9 – This code will turn the green light on when the down button is pressed, and off when the down button is not pressed.

Exercise

- Turn the green light on when the up button is pressed.
- Turn the green light off when the down button is pressed.
- Turn the green and red lights on when the left button is pressed.

The Else Statement

The `else` statement is always used with the `if` statement and has a very useful purpose that would be perfect for the last example. Previously, we needed to check if the button is pressed or not pressed. We can simplify this by catching when an `if` statement isn't true using `else` as shown in Example 11.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>

void setup() {
  BrainPad.TrafficLight.TurnGreenLightOff();
}

void loop() {
  if (BrainPad.Button.IsDownPressed())
    BrainPad.TrafficLight.TurnGreenLightOn();
  else
    BrainPad.TrafficLight.TurnGreenLightOff();
}
```

Example 10 – This code uses an `else` statement to check if a button is pressed or not to turn the traffic light's green LED on and off.

Or & And Operators

The most used logical operators are the `||` (or) and `&&` (and) operators. These are typically used within the `if` statement. In Example 12, we want the buzzer to generate a 5,000 Hz sound when both the up and down buttons are pressed. This is accomplished by using an `&&` operator between `IsUpPressed()` and `IsDownPressed()`.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>

void setup() {
}

void loop() {
  if (BrainPad.Button.IsUpPressed() && BrainPad.Button.IsDownPressed()) {
    BrainPad.Buzzer.PlayFrequency(5000);
  } else {
    BrainPad.Buzzer.Stop();
  }
}
```

Example 11 – This code checks if both up and down are pressed, if so it plays a high pitch sound.

White Space

Spaces and lines are used to make our code more readable. C++ programs completely ignore white space as shown in Example 12.

```
#include <Servo.h>
#include <SPI.h>
#include <Wire.h>
#include <BrainPad.h>
void setup(){}
void loop(){if(BrainPad.Button.IsUpPressed()&&BrainPad.Button.IsDownPressed()){
BrainPad.Buzzer.PlayFrequency(5000);}else{BrainPad.Buzzer.Stop();}}
```

Example 12 – This code shows how white space is ignored when the program runs. White space simply makes code readable by humans. However, includes must be kept on their own lines. Functions must also have spaces between their return type and name.

The above program is perfectly valid. Can you guess what the program does?

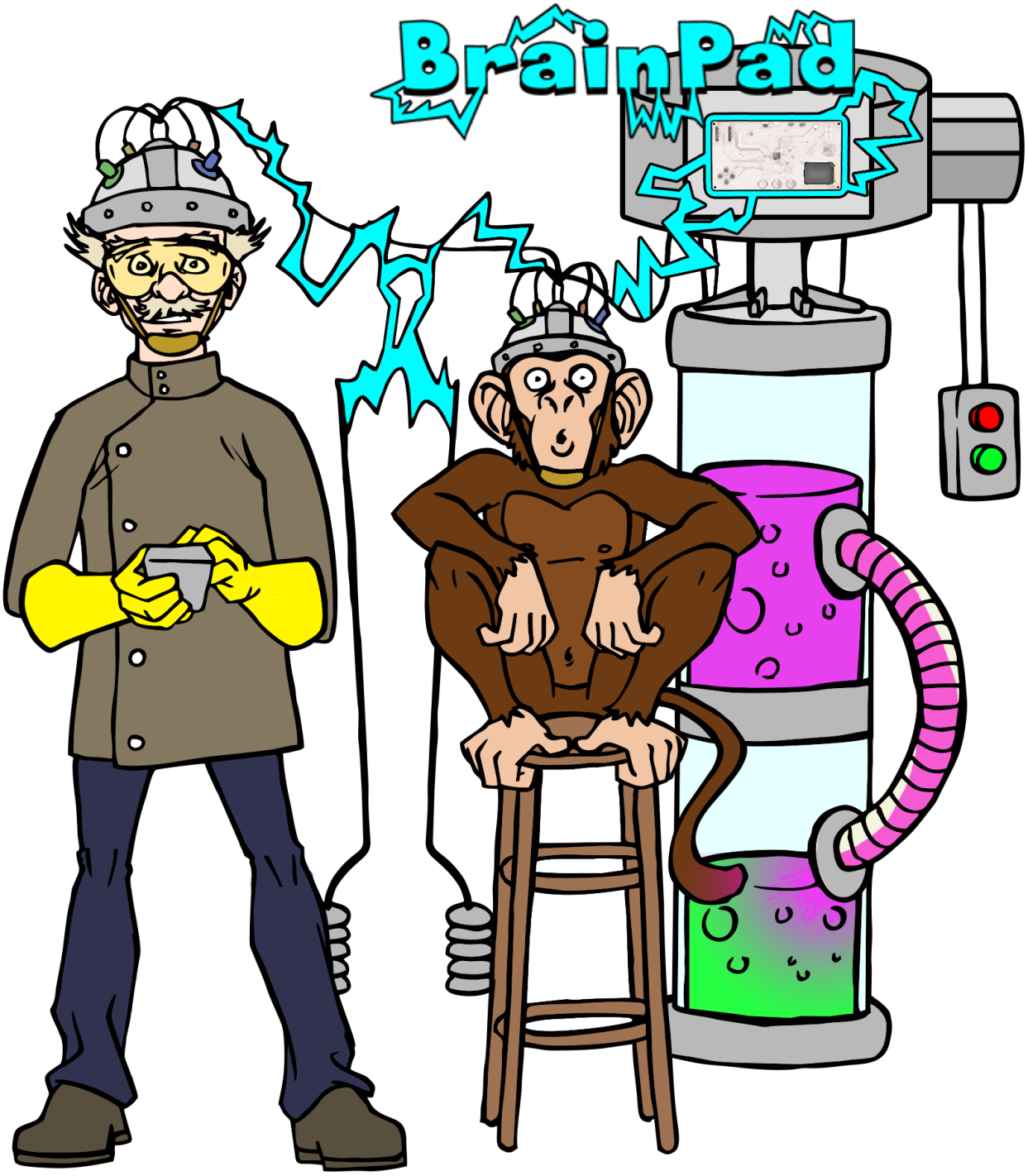
Exercise

Turn the green light on and play the buzzer at 5,000 Hz whenever the up or down buttons are pressed. Pressing both or either one will have the same effect.

Extra Credit

If the up and down buttons are pressed together, turn the red light on. If the up or down buttons are pressed, then play the buzzer at 5,000 Hz. You must have two `if` statements in your code.

Tip: Use nested statements such as an `if` statement inside another `if` statement.



GHIElectronics.com/support/brainpad