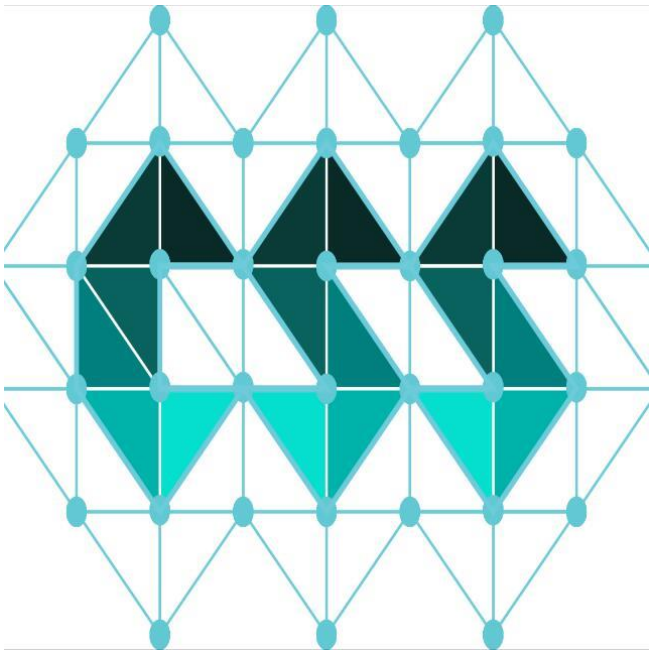

CSS PROJECT GUIDE

Bit Byte Bug

SUMMARY

This document will serve as a guide to developing the Bit Byte Bug project as designed by CSS' Team.

Isaiah Carrington

Contents

Introduction

Welcome to the CSS Bit Byte Bug Guide. In this document I aim to guide you down the process of creating a simple Rock Paper Scissors like game, using HTML, CSS, and JavaScript.

This project should take on average, about an hour to complete.

Overview

Bit Byte Bug is a web game designed using HTML, CSS, and JavaScript, designed to teach students basic usages of each of the three (3) languages. The game will be a replica of the infamous Rock Paper Scissors game but using Computer Science terms instead.

Brief Explanation

Bit or Binary Digit, is the smallest unit of digital data, representing 0 or 1.

A Byte is a group of 8 bits used to store data.

Lastly, a bug is an error or unintended behavior in a computer program.

Rules

- Bit Defeats Byte
- Byte Defeats Bug
- Bug Defeats Bit

Gameplay

The user will select their choice of Bit, Byte or Bug. The program will then randomly choose one of those same options. Winner will be determined based on the aforementioned rules. If both players happen to choose the same option, then the game will be determined as a draw. The game results will then be displayed in an alert box.

Design

The game will be designed using a minimum approach, which students can then expand on as they see fit. We will use just the minimum number of elements to get a functioning result on the screen.

Aim

In designing this game, the aim is to teach students the following concepts:

- Basic HTML
    - Tags
    - Attributes
- Basic JavaScript
    - Document
    - Functions (Arrow)
    - Variables
- CSS Frameworks
    - Bulma
    - Classes

Requirements

To take part in the exercise, students will require the following requirements:

- Laptop / Computer
- Web Browser
- Text editor (such as the default Notepad)

Methodology

We will begin with HTML, building out the elements that we will need.

Next, we will add in JavaScript, to get the core functionality and logic of the game.

Finally, we will include Bulma CSS, and add styling to our page.

Implementation

In this section, we will examine the implementation of the project.

File Structure

For the simplicity of this project, we will be using two (2) files. These will be:

- index.html
- script.js

Note: For students not using a text editor, they will have to enable “Show File extensions” in their file explorer.

HTML Part 1

HTML defines the structure for the web page. For the HTML, we will first create the index.html file.

Getting Started

Before we do any real thinking, we will begin by creating our overall structure as follows.

Figure 1: Image showing core structure of website

What we've done here, is state that we are going to be making an HTML document. That's what <!DOCTYPE html> does.

From there, we create the html, head, title, and body tags as needed. These will be explained more in depth during the session.

The Basics

Given that we'll be making the most basic version of the game, we will need three (3) clickable buttons, each representing one of the three (3) options:

- Bit
- Byte
- Bug

We can create these, using the button tags inside our body, giving them names as follows:

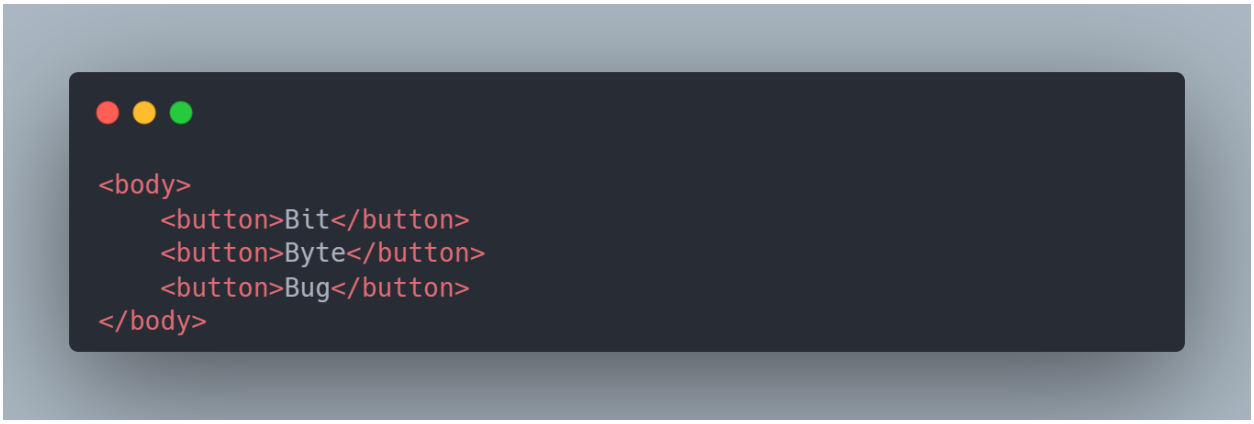

Figure 2: Image showing button code

If we look at the webpage within the browser, it should look like:

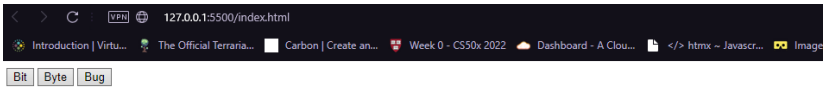

Figure 3: Image showing html page at this point

Final Steps

Last thing we need to do, is setup our buttons to do something when we click them. For now, we'll have them alert a simple message, stating their value. We can do this by adding an onclick attribute as follows:

```
<body>
    <button onclick="alert(this.innerText)">Bit</button>
    <button onclick="alert(this.innerText)">Byte</button>
    <button onclick="alert(this.innerText)">Bug</button>
  </body>
```

Figure 4: Image showing the onclick attribute with the alert function

Now whenever you click the button, it will show whatever text is in between the button tags. This will be discussed more in depth during the session.

With this working, we can start to prepare for the JavaScript section.

Preparing For JavaScript

Before we can use JavaScript in our code, we must make some preparations.

First, we need to create our JavaScript file. This file is where we will store our JavaScript code. In the same folder as our HTML file, we'll just create a file called "script.js".

With the file created, we can start working on the HTML. What we need to do, is include this script file in our HTML document by adding the script tag as follows:

```
<body>
  ...
</body>
<script src="script.js"></script>
```

Figure 5: Image adding JavaScript to HTML document

Next, in our JavaScript, we will be creating a function called buttonPressed, which we want to run every time we press a button. So, we will modify our button code to call this function instead of the alert. We will make the following change:

```
<body>
  <button onclick="buttonPressed(this.innerText)">Bit</button>
  <button onclick="buttonPressed(this.innerText)">Byte</button>
  <button onclick="buttonPressed(this.innerText)">Bug</button>
</body>
```

Figure 6: Image replacing alert with buttonPressed

With this, we have completed the HTML, and can now move on to creating our JavaScript.

JavaScript

With the raw HTML completed, we'll be looking at adding some functionality. Using the rules defined, we can begin to reason out our logic.

1. When the user clicks one of the options (Bit, Byte or Bug), we want this choice of theirs to be saved somehow.
2. The program must then randomly make a choice of Bit, Byte or Bug.
3. We need to then store this choice.
4. We then need to determine the outcome using the rules we mentioned.
5. Finally, we will output this outcome to the user.

We'll build out this functionality step by step.

First, we'll create out "script.js" file, where we will be writing our JavaScript.

Step 1

First, we'll create a function that will be run when the user presses the button. We can do this as follows:

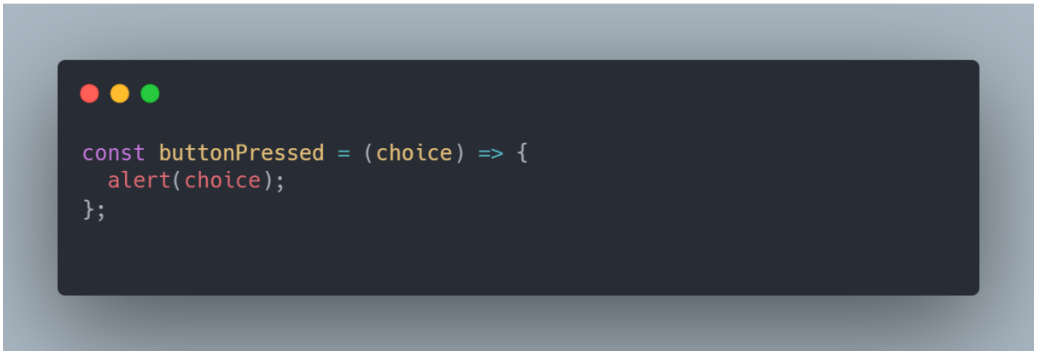

Figure 7: Image showing JavaScript Function

If we reload our webpage, and try out our buttons, we should get something like:

Figure 8: Image of pressing Byte Button after linking JavaScript

With this, we have stored the user's choice inside of the variable choice. Therefore, step 1 is complete!

Step 2

For step 2, we want our program to randomly make a choice of Bit, Byte or Bug. For this, we will generate a random number between 0 and 2 (inclusive) and choose the option from there. We can generate a random number in this range using the following code:

```
const buttonPressed = (choice) => {
  let number = Math.floor(Math.random() * 3);
};
```

Figure 9: Image showing generating a random number between 0 and 2 inclusive

The code will be explained during the session of course.

Step 3

Now that we have a random number, we need to decide what value it will be. We can do this using if statements as follows:

```
const buttonPressed = (choice) => {
  let number = Math.floor(Math.random() * 3);

  let compChoice = "";
  if (number == 0) compChoice = "Bit";
  else if (number == 1) compChoice = "Byte";
  else compChoice = "Bug";
};
```

Figure 10: Image using if statements to decide computer's choice

And just like that, the computer has decided a random choice, and we've stored it in the variable compChoice.

Step 4

For step 4, we need to determine the outcome based on the rules we defined. We can do this using MORE if statements!

First, we'll create a variable that will store the outcome as text.

```
const buttonPressed = (choice) => {
  let number = Math.floor(Math.random() * 3);

  let compChoice = "";
  if (number == 0) compChoice = "Bit";
  else if (number == 1) compChoice = "Byte";
  else compChoice = "Bug";

  let outcome = "";
};
```

Figure 11: Image creating variable to store outcome

Now, we can use our comparisons to determine the outcome. For simplicity, we'll break this into sub steps. First, we'll check to see if the player and the computer chose the same value. If so, then our outcome will be a draw.

```
const buttonPressed = (choice) => {
  let number = Math.floor(Math.random() * 3);

  let compChoice = "";
  if (number == 0) compChoice = "Bit";
  else if (number == 1) compChoice = "Byte";
  else compChoice = "Bug";

  let outcome = "";

  if (choice == compChoice) outcome = "DRAW";
};
```

Figure 12: Image determining if the outcome is a draw

Next, we'll check to see if the player one, using the rules we defined before.

```
const buttonPressed = (choice) => {
  let number = Math.floor(Math.random() * 3);

  let compChoice = "";
  if (number == 0) compChoice = "Bit";
  else if (number == 1) compChoice = "Byte";
  else compChoice = "Bug";

  let outcome = "";

  if (choice == compChoice) outcome = "DRAW";
  else if (
    (choice == "Bit" && compChoice == "Byte") ||
    (choice == "Byte" && compChoice == "Bug") ||
    (choice == "Bug" && compChoice == "Bit")
  )
    outcome = "Player Wins!";
};
```

Figure 13: Image showing code checking to see if the player won.

This code will be explained during the session of course.

Finally, if the game is not a draw, and the player didn't win, then the computer must have won. We can show this using the following code:

```
const buttonPressed = (choice) => {
  let number = Math.floor(Math.random() * 3);

  let compChoice = "";
  if (number == 0) compChoice = "Bit";
  else if (number == 1) compChoice = "Byte";
  else compChoice = "Bug";

  let outcome = "";

  if (choice == compChoice) outcome = "DRAW";
  else if (
    (choice == "Bit" && compChoice == "Byte") ||
    (choice == "Byte" && compChoice == "Bug") ||
    (choice == "Bug" && compChoice == "Bit")
  )
    outcome = "Player Wins!";
  else outcome = "Computer Wins!";
};
```

Figure 14: Image showing code if computer won

Step 5

Finally, we can output this result to the user using the alert function we looked at earlier.

This is as simple as:

```
const buttonPressed = (choice) => {
  ...
  alert(outcome);
};
```

Figure 15: Image showing the output of the outcome.

The overall final JavaScript code should look like:

```
const buttonPressed = (choice) => {
  let number = Math.floor(Math.random() * 3);

  let compChoice = "";
  if (number == 0) compChoice = "Bit";
  else if (number == 1) compChoice = "Byte";
  else compChoice = "Bug";

  let outcome = "";

  if (choice == compChoice) outcome = "DRAW";
  else if (
    (choice == "Bit" && compChoice == "Byte") ||
    (choice == "Byte" && compChoice == "Bug") ||
    (choice == "Bug" && compChoice == "Bit")
  )
    outcome = "Player Wins!";
  else outcome = "Computer Wins!";

  alert(outcome);
};
```

Figure 16: Image showing full JavaScript Code

This completes the JavaScript section of this course.

CSS

CSS or Cascading Style Sheets is responsible for the overall style and look of the page. Normally, we can create our own styles, however, for this demonstration, we will be making use of a Framework called Bulma.

To use Bulma, we can visit their website here or just type in Bulma CSS in your browser.

If going to it via google, click on this link:

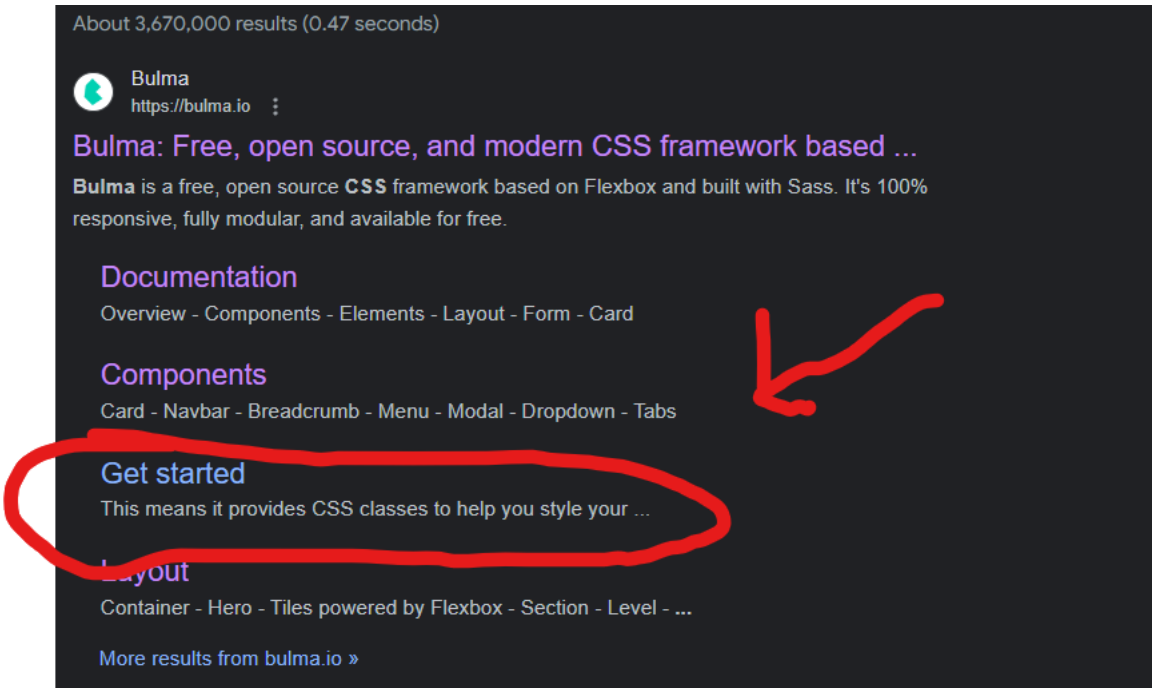

Figure 17: Image showing which Bulma link to click

This will take you to a page resembling:

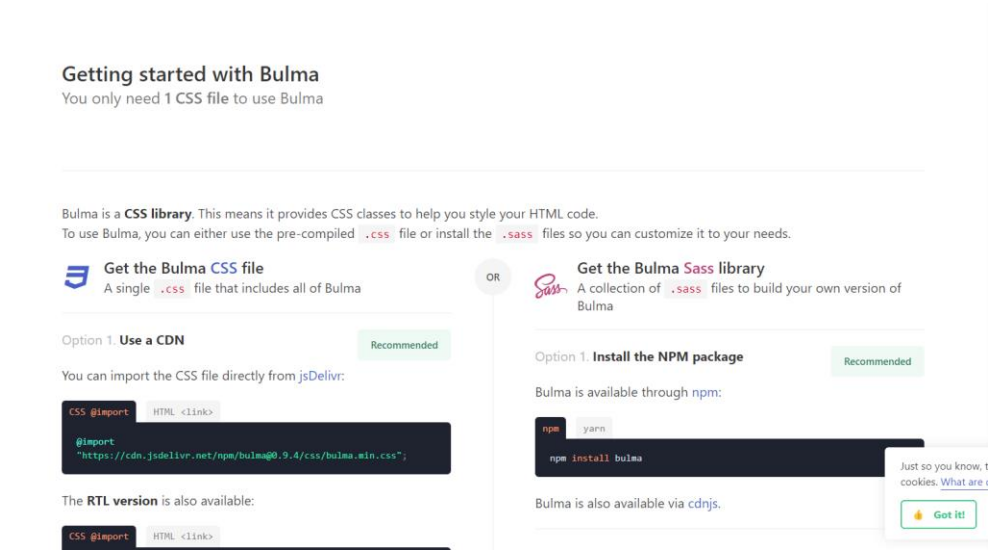

Figure 18: Image showing Bulma Getting Started page

Getting Started with Bulma

Using a CSS Framework is very easy. First, we just need to include the style sheet in our HTML, and then use the provided classes in our code. To include the stylesheet, we need to press on “HTML <link>” under “Use a CDN”:

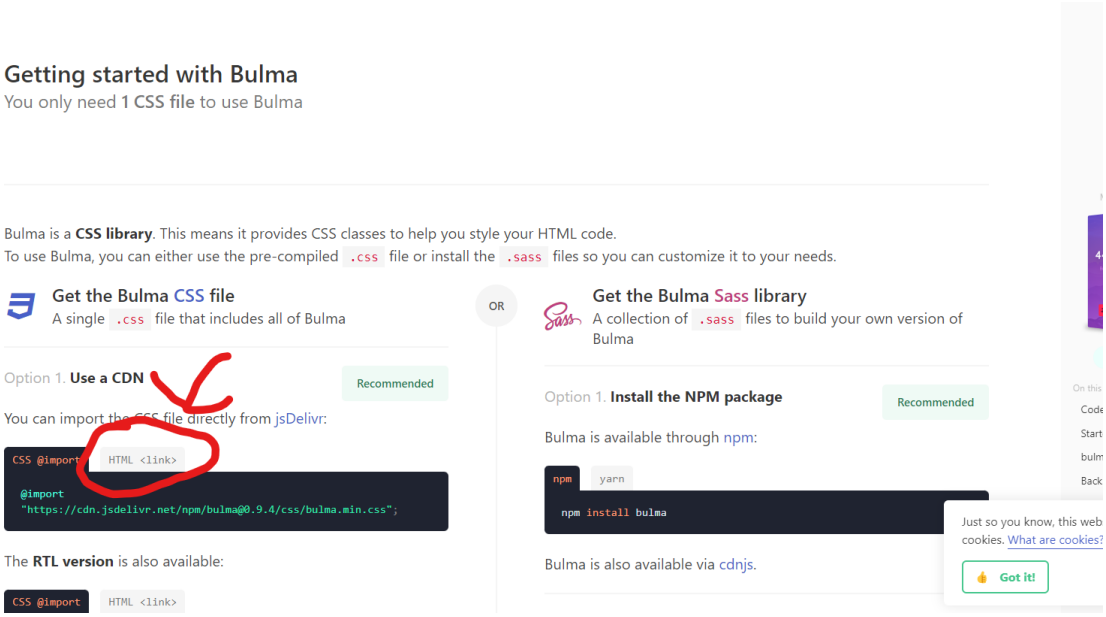

Figure 19: Image showing what I said it shows.

And then copy the displayed link.

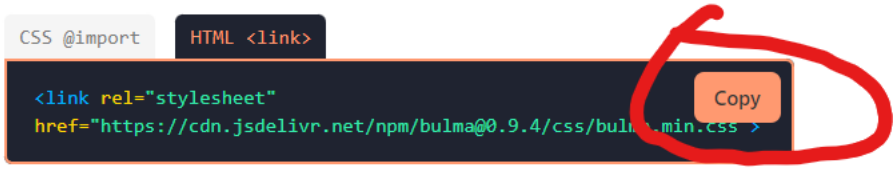

Figure 20: Image showing the bright orange copy button.

With the link copied, we want to put it in the head of our HTML document as follows:

Figure 21: Image showing the Link in the Head of the HTML document.

Using Bulma

If you were to save, and reload your HTML page in your browser, you will see the effect of Bulma immediately. Our buttons go from looking like:

Bit Byte Bug

To looking like:

Bit Byte Bug

Might not be extremely obvious from the images, but the font has changed. Now we can style our buttons further.

By using either the search bar on the Bulma website and typing “Button”, or navigating to this link, we can see the list of classes Bulma provides for us to style our buttons along with examples:

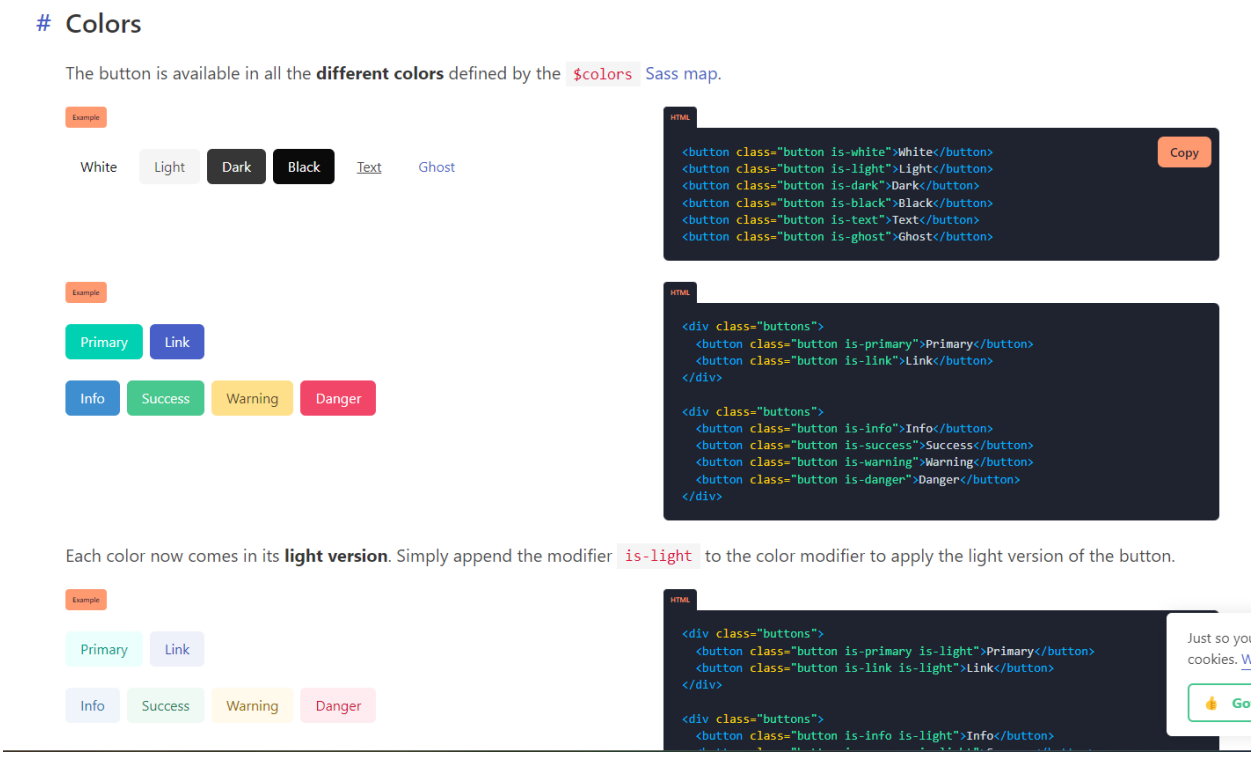

Figure 22: Image showing various button coloring styles

Using the examples there as a reference, add a class attribute to your buttons to style them as you please!

Example:

Figure 23: Image showing buttons styled using Bulma Classes

Which results in:

Figure 24: Image showing Bulma styled Buttons

Have fun and customize as you please!

Closing Notes

Congratulations, you've successfully built out Bit, Byte, Bug! You've also Styled it and added functionality to make it work!

For extra Challenge, consider adding more stylings to the page, like maybe centering the buttons 🙄.

Hope you enjoyed!