DOM: Document Object Model

Using alerts and prompts is really fun, but javascript is way more powerful than just alerts and prompts! Most every site that you encounter uses a lot of javascript, and that javascript is used to manipulate the DOM.

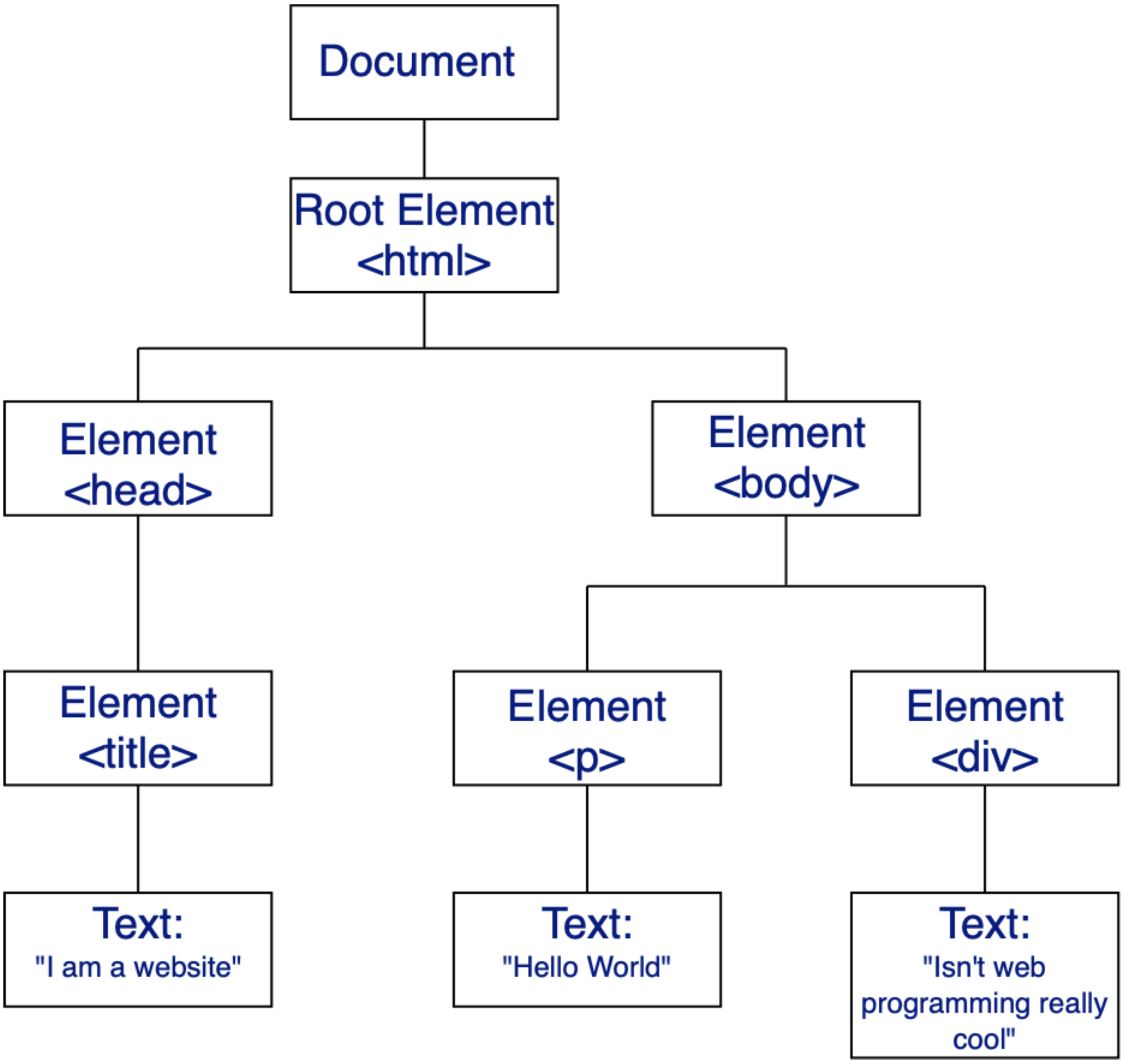
What is the DOM?

DOM stands for "Document Object Model", which is more or less just a super fancy name for a bunch of javascript variables and methods that interact with our HTML and CSS. By using this javascript & html/css bridge, we can easily interact with the HTML by using javascript, which lets us change colors, text sizes, validate forms, etc.

The web browser (e.g. Firefox, Chrome, etc.) turns every html tag into a javascript object! Let's take a very basic website:

```html
<!doctype html>
<html>
  <head>
    <title> I am a website </title>
  </head>
  <body>
    <p>
      Hello World
    </p>
    <div>
      Isn't web programming really cool?
    </div>
  </body>
</html>
```

What happens behind the scenes is that our browsers actually create a "model" of our site's elements. Each HTML element gets modelled as a javascript object, and each element inside of our elements has a smaller object! So, if we start at doctype html (our document) and inside of that object we have a head and a body. Inside of the head we have a title, and inside of our body we have a paragraph element and a div element. Each of those in turn have text inside them!



Now, looking at diagrams is fun, but let's see an actual document. Let's look at how the html above works. Click here, and then open your developer console (in chrome it's under developer, in firefox it's under window). In your console, type: `console.dir(document)`.

`console.dir()` is a way for us to click through javascript objects in our developer console. Now, the document we see here is really big! That's because there's a lot going on behind the scenes other than just a few elements.

In your document click on `body`. We can see that body has a lot of elements inside of it. If you click then on `childNodes` you can see the children of `<body>`:

```
childNodes: NodeList (5)
  0 #text " "
  1 <p> Hello World </p>
  2 #text " "
  3 <div> Isn't web programming really cool? </div>
  4 #text " "
```

This is the document object model in a nutshell! This is what lets us access parts of our web page with javascript!

Let's explore just a little more. Let's take our earlier example, and look at what nested elements look like:

```html
<!doctype html>
<html>
  <head>
    <title> I am a website </title>
  </head>
  <body>
    <p>
      Hello World
    </p>
    <div>
      <p>
        You could go visit google if you want by clicking below!
      </p>
      <a href='www.google.com'> Go to google! </a>
    </div>
  </body>
</html>
```

Now let's take a look. If you go to the console and write `console.dir(document)`, then open up `body` and click on `childNodes`, you'll see that one of our children ALSO has child nodes!

```
childNodes: NodeList (5)
  0 #text " "
  1 <p> Hello World </p>
  2 #text " "
  3 <div>
      <p> You could go visit google if you want by clicking below! </p>
      <a href="www.google.com"> Go to google! </a>
    </div>
  4 #text " "
```

There's a lot to the DOM that we'll not cover (and you'll probably never cover - there's so much to the DOM)! It's ok not to know everything. That's what google is for!

Working with the DOM

So now that we know about what the DOM is and how it works (or at least have a general understanding of a little bit), let's look at how to work with it!

Let's take a look back at our original (very complex) page. Open the page and go to your developer console. In your console type:

```
var myLink = document.querySelector('a');
```

Now, type `myLink` in your console. What does it return? You should see:

```
<a href="www.google.com"> Go to google! </a>
```

What we can then do, is use the DOM to manipulate what's being displayed! Try typing:

```
myLink.style.fontSize = '500%'
```

Your link should've gotten a whole lot bigger!

That's not all either! We can use functions to manipulate our doms as well! Go back to our page, and this time, in your console, type:

```
// just to set our link back to its original
myLink.style.fontSize = '100%'
var size = 10

setInterval( function() {
  myLink.style.fontSize = size + 'px'
  size = size + 1
}, 50)
```

You might want to refresh your page (or else the link will just keep growing)!

That function we used for setInterval may just look like fun and games, but what that shows us is that we can tie javascript functions directly to our DOM, which means, we can tie buttons (and so much more!) to our DOM!

Selecting and Manipulating DOM elements

There are a lot of DOM selectors. You can ultimately select anything in the document by looking at what's inside of the document (you can view with `console.dir(document)`). Try that again on our basic web page above.

Now if you go to your console and type:

```
var myURL = document.URL
```

You'll find that myURL has a URL inside of it! Everything we'll be working with lives inside the document! We'll use `document.SOMETHING` for everything!

We've already looked at `document.querySelector()`, but there are some other document methods we'll dive into as well:

- `document.getElementById()`: for when you want to get one specific element.
- `document.getElementsByClassName()`: for when you want to get all elements of a particular class.
- `document.getElementsByTagName()`: for when you want to get all elements of a certain tag (e.g. all `<p>` elements, or all `<h3>` elements).
- `document.querySelector()`: for when you want to search by a css style'd query and grab the very first element.
- `document.querySelectorAll()`: for when you want to search by a css style'd query and grab *all* the elements in the DOM.

document.getElementById()

getElementById is pretty self explanatory. It grabs a specific element by its ID. If you recall, back when we were coding our sites and learning CSS, we saw that you could technically style a webpage using multiple of the same IDs. An ID is supposed to be a unique thing. Let's see why:

```html
<!doctype html>
<html>
  <head>
    <title> I am a website </title>
    <style>
      #greyDiv {
         background-color: lightgrey;
      }

      .coolClass {
        font-size: 2em;
        font-weight:bold;
      }
    </style>
 </head>
  <body>
    <p class='coolClass'>
      Hello World
    </p>
    <div id='greyDiv'>
      <p>
        You could go visit google if you want by clicking below!
      </p>
      <a href='www.google.com'> Go to google! </a>
    </div>
    <p class='coolClass'>
      I'm just a boring paragraph.
    </p>
    <div id='greyDiv'>
      I like to steal IDs!
    </div>

  </body>
</html>
```

See how both `<div>` elements have the id "`greyDiv`"? Styling wise, our browser allows for multiple of the same IDs. Now, let's try selecting our IDs with `document.getElementById()`. Open the browser's console and type:

```
var myDiv = document.getElementById('greyDiv')
```

Because IDs are supposed to be unique, the document's function `getElementById()` only grabs the element with the first ID it recognizes. Notice, though, that it not only grabs the `<div>` but also everything inside of it. That's because the elements inside of the `<div>` are `child nodes` of the `<div>` itself. **So when we select a specific DOM element, we're not only grabbing the element itself, we're also grabbing all of its child elements!**

Reminder: When you type `myDiv` into the console, notice how it gives you a bunch of HTML. Type `console.dir(myDiv)` to see the object itself.

document.getElementsByClassName()

Suppose that you didn't want to only grab a specific element, but wanted every single element using a specific class (say you wanted to change some class styling), you can grab elements by their class names!

Go into our page, and type:

```
var classElements = document.getElementsByClassName('coolClass')
```

Up to now, we've only seen individual elements returned (with some nested elements inside). Now, we actually return an "HTML Collection" which is a fancy way of saying that `classElements` is kind of like an array of html elements that use `coolClass` as a class.

document.getElementsByTagName()

Sometimes you need to change specific types of elements on your page (say you want to change your list elements from all bullets to circles, because why not?), then you'd use `getElementsByTagName()`. Take a look at our boring list below:

```
<!doctype html>
<html>
  <head>
    <title> I am a website </title>
 </head>
  <body>
  <div>
    I am a boring site.
  </div>
  <ul>
```

```
      <li>Boring List</li>
      <li>Boring List 2 </li>
      <li>Boring List 3 </li>
      <li>Boring List 4 </li>
    </ul>
  </body>
</html>
```

What if we wanted to spice our list up? Remember when we turned our list bullets into cats? Let's do that, but with javascript:

First, select all of the list items `<li>`s:

```
var myList = document.getElementsByTagName('li')
```

Now, let's take our cat style and store it in a variable!

```
myStyle = `background-image: url('https://media.giphy.com/media/V0YMxvqOXOkEw/giphy.gif');     background-repeat: no-repeat;
  list-style-type: none;
  padding-left:75px;
  padding-bottom:50px;
`
```

Note: Above we used the tick ` (to the left of the 1) to lets us create multi-line strings!.

Now that we have a special style, let's create a loop in the console to loop over all of our list elements and add our new style!

```
for(var i = 0; i < myList.length; i++) {
  myList[i].style = myStyle
}
```

BAM! Now we've made our list significantly better with cats.

document.querySelector()

Query selector is a method that basically does what we've done above, but ignores whether or not we're looking for an ID or a given class or a tag. Query selector returns the **first** element that matches what query it's given. So let's take our site from earlier with all of the ids and classes:

```
<!doctype html>
```

```html
<html>
  <head>
    <title> I am a website </title>
    <style>
      #greyDiv {
         background-color: lightgrey;
      }

      .coolClass {
        font-size: 2em;
        font-weight:bold;
      }
    </style>
 </head>
  <body>
    <p class='coolClass'>
      Hello World
    </p>
    <div id='greyDiv'>
      <p>
        You could go visit google if you want by clicking below!
      </p>
      <a href='www.google.com'> Go to google! </a>
    </div>
    <p class='coolClass'>
      I'm just a boring paragraph.
    </p>
    <div id='greyDiv'>
      I like to steal IDs!
    </div>

  </body>
</html>
```

Query selector uses the same types of notation as CSS, so if you want to select an id (such as `greyDiv`), you'd have to use the css syntax: `#greyDiv`, and similarly, if you wanted to select a class, you'd have ot use the syntax `.coolClass`.

Now to select an ID with query selector we can write:

```javascript
var firstClass = document.querySelector('.coolClass')
```

Did you notice that the only element inside of `firstClass` is:

```
<p class="coolClass">
      Hello World
</p>
```

This is because query selector only selects the first item it comes across.

As a quick note, when you change the style of a selected ID or a class, you're not changing the ID or the Class styling itself, you're only changing the elements that you've selected.

Try writing:

```
firstClass.style = 'background-color:red'
```

If you look inside firstClass again, now it is:

```
<p class="coolClass" style="background-color: red;">
      Hello World
</p>
```

It still is inheriting its colors from from `coolClass`, but the styling on the element itself is what changed.

document.querySelectorAll()

Our Query selector only selects a single item, but if we want to get all items of a specific type, we can use `document.querySelectorAll()`. This grabs not just the first item, but all items that match the query! Let's try what we did above, again!

On the page above, go to your console, and type:

```
var allClasses = document.querySelectorAll('.coolClass')
```

This will select not just the first class like above, but **all** the elements that use `coolClass`:

```
NodeList (2) = $1
0  <p class="coolClass"> Hello World </p>
1  <p class="coolClass"> I'm just a boring paragraph. </p>
```

Now, just like above if we write a quick for loop:

```
for(var i = 0; i < allClasses.length; i++ ){
  allClasses[i].style = 'background-color:red'
}
```

Now we've turned all of our cool classes red!

Styling Through the DOM:

We've already worked with a number of styling methods, however, we've only really accessed the styling in a specific way, like above where we wrote:

```
allClasses[i].style = 'background-color:red'
```

Our initial choice of styling in the previous examples was entirely for familiarity. We could just as easily have written:

```
allClasses[i].style.background = 'red'
```

The DOM lets us access styles with the javascript dot operator. While we may go back and forth in our styling, it's good to note that you can access not only the 'style' attribute, but also specific types of stylings based entirely off of the DOM's dot operators!

So far we've only worried about colration with our styling. Any styling that we've done with CSS, we can also do here:

```
<!doctype html>
<html>
  <head>
    <title> I am a website </title>
 </head>
  <body>
  <div>
    I am a boring site.
  </div>
  <ul>
    <li>Boring List</li>
    <li>Boring List 2 </li>
    <li>Boring List 3 </li>
    <li>Boring List 4 </li>
  </ul>
  </body>
</html>
```

Let's wrap our list in a border. First we need to select the list:

```
var myList = document.querySelector('ul')
```

Now let's add a border:

```
myList.style.border = '1px solid blue'
```

Styling with CSS through DOM Manipulation

So far we've styled entirely by adding single styles to a given class, ID, or element, but what if we wanted to give multiple styles? We could do what we did before an add one giant string of styles to our elements. That can be time consuming.

Above, when we changed our background bullets to cats, we used this simple for loop with a style string:

```
myStyle = `background-image: url('https://media.giphy.com/media/V0YMxvqOXOkEw/giphy.gif');     background-repeat: no-repeat;
  list-style-type: none;
  padding-left:75px;
  padding-bottom:50px;
`
for(var i = 0; i < myList.length; i++) {
  myList[i].style = myStyle
}
```

Without the for loop that whole style sting and loop would look like:

```
myList[0].style.backgroundImage = "url('https://media.giphy.com/media/V0YMxvqOXOkEw/giphy.gif')"
myList[0].style.backgroundRepeat = 'no-repeat'
myList[0].style.listStyleType = 'none'
myList[0].style.paddingLeft = '75px'
myList[0].style.paddingBottom = '50px'
myList[1].style.backgroundImage = "url('https://media.giphy.com/media/V0YMxvqOXOkEw/giphy.gif')"
myList[1].style.backgroundRepeat = 'no-repeat'
myList[1].style.listStyleType = 'none'
myList[1].style.paddingLeft = '75px'
myList[1].style.paddingBottom = '50px'
```

```
myList[2].style.backgroundImage =
"url('https://media.giphy.com/media/V0YMxvqOXOkEw/giphy.gif')"
myList[2].style.backgroundRepeat = 'no-repeat'
myList[2].style.listStyleType = 'none'
myList[2].style.paddingLeft = '75px'
myList[2].style.paddingBottom = '50px'
myList[3].style.backgroundImage =
"url('https://media.giphy.com/media/V0YMxvqOXOkEw/giphy.gif')"
myList[3].style.backgroundRepeat = 'no-repeat'
myList[3].style.listStyleType = 'none'
myList[3].style.paddingLeft = '75px'
myList[3].style.paddingBottom = '50px'
```

While both options work, we don't have to waste our time with either option! They're both significantly more time consuming than createing a brand new class.

Let's assume we already had a cat-class defined in our code:

```
.cat-class {background-image:
url('https://media.giphy.com/media/V0YMxvqOXOkEw/giphy.gif');
background-repeat: no-repeat;
  list-style-type: none;
  padding-left:75px;
  padding-bottom:50px;
}
```

Now, when we want to change our bullet points, we no longer have to do all of this extra work! All we need to do is add a class to our elements:

```
 myList.classList.add('cat-class')
```

Now when we go to update our styles we can just:

```
for(var i = 0; i < myList.length; i++) {
  myList[i].classList.add('cat-class')
}
```

Problem solved!

EXERCISE:

Take the list items on our "boring list" site, and select items and try updating them. You can update by color, font (size, family, etc.), opacity, style, etc. The goal of this is to get used to using the DOM to manipulate our sites!

Events

What makes websites reactive is that there are special things called `events`. Each event is something that happens on the website like a mouse moving over a portion of your site or clicking on a button! A site reacts to these events by using **event handlers**.

Event handlers are javascript functions that you can attach to specific elements that handle specific user interactions! Let's take a look at a basic list from our very first lecture:

HTML:

```html
<!doctype html>
<html>
  <head>
  </head>
  <body>
    <h1> Welcome to my site! </h1>
    <hr />
     <ul type='square'>
       <li>I really like to write code</li>
       <li>Does that make me a square?</li>
       <li>Some people do call me a block head...</li>
       <li>Maybe they're just not seeing me at the right ANGLE?</li>
     </ul>
  </body>
</html>
```

Now let's add some CSS to our site:

```
    .done {
    text-decoration: line-through;
    opacity: 0.5;
  }

  .selected {
    color: green;
  }
```

Notice how nothing happened! That's because we haven't implemented any of our css classes! We don't want to hard code our classes into our site! We want to attach them to the DOM's event listeners!

What we'll want to do is grab every single list item `<li>` and then attach an event listener to each. Let's grab an event:

```
var lis = document.querySelectorAll("li")
```

Now we've selected all of our `<li>`s. We'll want to add an event listener to each. But what event listeners? We want to have a `mouseover`, a `mouseout`, and a `click`. Let's try adding those event listeners to each `<li>` with a for loop:

```
for(var i = 0; i < lis.length; i++){
  lis[i].addEventListener("mouseover", function(){
    this.classList.add("selected");
  });

  lis[i].addEventListener("mouseout", function(){
    this.classList.remove("selected");
  });

  lis[i].addEventListener("click", function(){
    this.classList.toggle("done");
  });
}

```

Now, let's see what those all look like together!

Why Use Event Handlers?

You may wonder why we should bother using event handlers if they're just going to put lines through list items, change colors of pages, change font sizes, but using event handlers can be an incredibly powerful tool! Let's spend the rest of today talking about making a game!

```html
<!DOCTYPE html>
<html>
<head>
  <title>Color Game</title>

<body>
<h1>
  The Great
  <br>
  <span id="colorDisplay">RGB</span>
  <br>
  Color Game
</h1>

<div id="stripe">
  <button id="reset">New Colors</button>
  <span id="message"></span>
  <button class="mode">Easy</button>
  <button class="mode selected">Hard</button>
</div>

  <div id="container">
    <div class="square"></div>
    <div class="square"></div>
    <div class="square"></div>
    <div class="square"></div>
    <div class="square"></div>
    <div class="square"></div>
  </div>
</body>
</html>
```

CSS:

```css
body {
  background-color: #232323;
  margin: 0;
  font-family: "Montserrat", "Avenir";
}

.square {
  width: 30%;
  background: purple;
  padding-bottom: 30%;
  float: left;
  margin: 1.66%;
  border-radius: 15%;
  transition: background 0.6s;
  -webkit-transition: background 0.6s;
  -moz-transition: background 0.6s;
}

#container {
  margin: 20px auto;
  max-width: 600px;
}

h1 {
  text-align: center;
  line-height: 1.1;
  font-weight: normal;
  color: white;
  background: steelblue;
  margin: 0;
  text-transform: uppercase;
  padding: 20px 0;
}

#colorDisplay {
  font-size: 200%;
}

#message {
  display: inline-block;
  width: 20%;
}

#stripe {
  background: white;
  height: 30px;
  text-align: center;
  color: black;
}

```

```css
.selected {
  color: white;
  background: steelblue;
}

button {
  border: none;
  background: none;
  text-transform: uppercase;
  height: 100%;
  font-weight: 700;
  color: steelblue;
  letter-spacing: 1px;
  font-size: inherit;
  transition: all 0.3s;
  -webkit-transition: all 0.3s;
  -moz-transition: all 0.3s;
  outline: none;
}

button:hover {
  color: white;
  background: steelblue;
}

```

Javascript:

```javascript
var numSquares = 6;
var colors = [];
var pickedColor;
var squares = document.querySelectorAll(".square");
var colorDisplay = document.getElementById("colorDisplay");
var messageDisplay = document.querySelector("#message");
var h1 = document.querySelector("h1");
var resetButton = document.querySelector("#reset");
var modeButtons = document.querySelectorAll(".mode");


init();

function init(){
  setupModeButtons();
  setupSquares();
  reset();
}

function setupModeButtons(){
  for(var i = 0; i < modeButtons.length; i++){
    modeButtons[i].addEventListener("click", function(){
      modeButtons[0].classList.remove("selected");
      modeButtons[1].classList.remove("selected");
      this.classList.add("selected");
      this.textContent === "Easy" ? numSquares = 3: numSquares = 6;
      reset();
    });
  }
}

function setupSquares(){
  for(var i = 0; i < squares.length; i++){
  //add click listeners to squares
    squares[i].addEventListener("click", function(){
      //grab color of clicked square
      var clickedColor = this.style.background;
      //compare color to pickedColor
      if(clickedColor === pickedColor){
        messageDisplay.textContent = "Correct!";
        resetButton.textContent = "Play Again?"
        changeColors(clickedColor);
        h1.style.background = clickedColor;
      } else {
        this.style.background = "#232323";
        messageDisplay.textContent = "Try Again"
      }
    });
  }
}


function reset(){
  colors = generateRandomColors(numSquares);
  //pick a new random color from array
  pickedColor = pickColor();
  //change colorDisplay to match picked Color
  colorDisplay.textContent = pickedColor;
  resetButton.textContent = "New Colors"
  messageDisplay.textContent = "";
  //change colors of squares
  for(var i = 0; i < squares.length; i++){
    if(colors[i]){
      squares[i].style.display = "block"
      squares[i].style.background = colors[i];
    } else {
      squares[i].style.display = "none";
    }
  }
  h1.style.background = "steelblue";
}

resetButton.addEventListener("click", function(){
  reset();
})

function changeColors(color){
  //loop through all squares
  for(var i = 0; i < squares.length; i++){
    //change each color to match given color
    squares[i].style.background = color;
  }
}

function pickColor(){
  var random = Math.floor(Math.random() * colors.length);
  return colors[random];
}

function generateRandomColors(num){
  //make an array
  var arr = []
  //repeat num times
  for(var i = 0; i < num; i++){
    //get random color and push into arr
    arr.push(randomColor())
  }
  //return that array
  return arr;
}

function randomColor(){
  //pick a "red" from 0 - 255
  var r = Math.floor(Math.random() * 256);
  //pick a "green" from  0 -255
  var g = Math.floor(Math.random() * 256);
  //pick a "blue" from  0 -255
  var b = Math.floor(Math.random() * 256);
  return "rgb(" + r + ", " + g + ", " + b + ")";
}
```