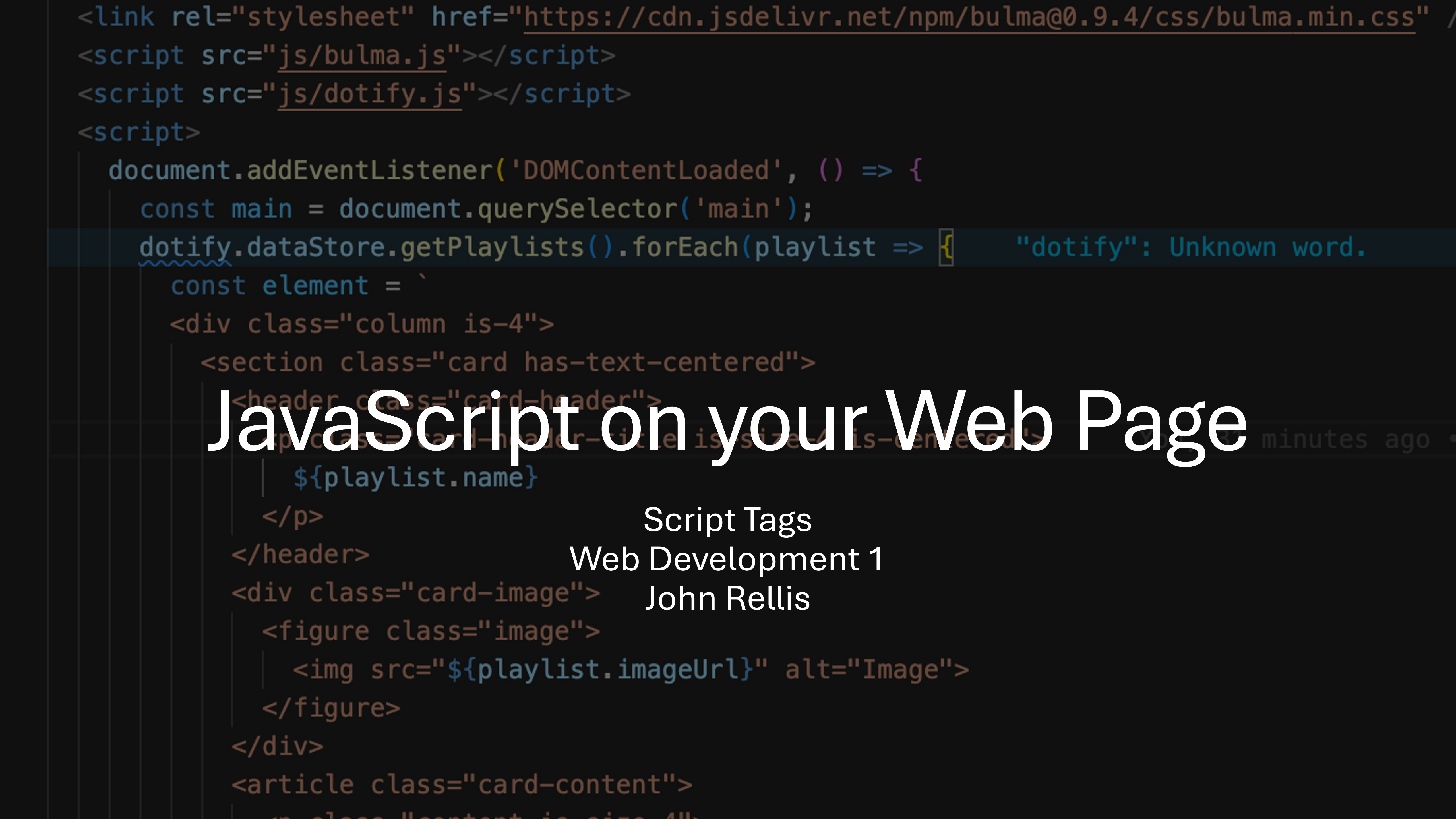
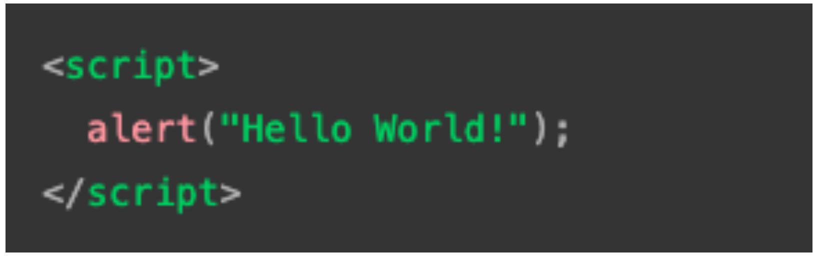


```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css"
<script src="js/bulma.js"></script>
<script src="js/dotify.js"></script>
<script>
  document.addEventListener('DOMContentLoaded', () => {
    const main = document.querySelector('main');
    dotify.dataStore.getPlaylists().forEach(playlist => {     "dotify": Unknown word.
      const element = `
      <div class="column is-4">
        <section class="card has-text-centered">
          <header class="card-header">
            <p class="card-header-title is-size-4 is-centered">          minutes ago
              ${playlist.name}
            </p>
          </header>
          <div class="card-image">
            <figure class="image">
              <img src="${playlist.imageUrl}" alt="Image">
            </figure>
          </div>
          <article class="card-content">
```

JavaScript on your Web Page

Script Tags
Web Development 1
John Rellis

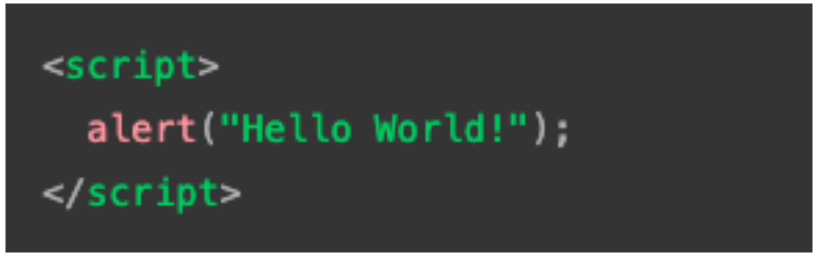
The Script Element (tag)



```
<script>
  alert("Hello World!");
</script>
```

- The <script> HTML element is used to embed executable code or data; this is typically used to embed or refer to JavaScript code.
- The <script> element can also be used with other languages, such as WebGL's GLSL shader programming language and JSON.

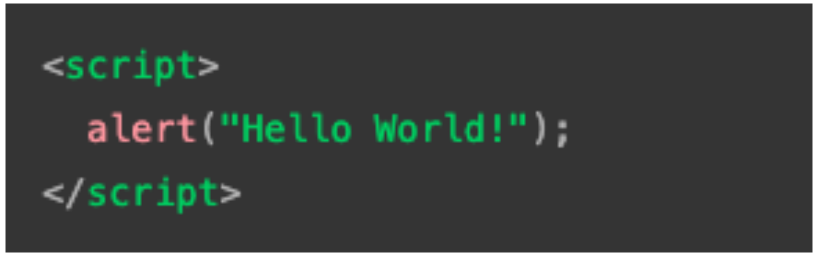
The Script Element – Inline Script



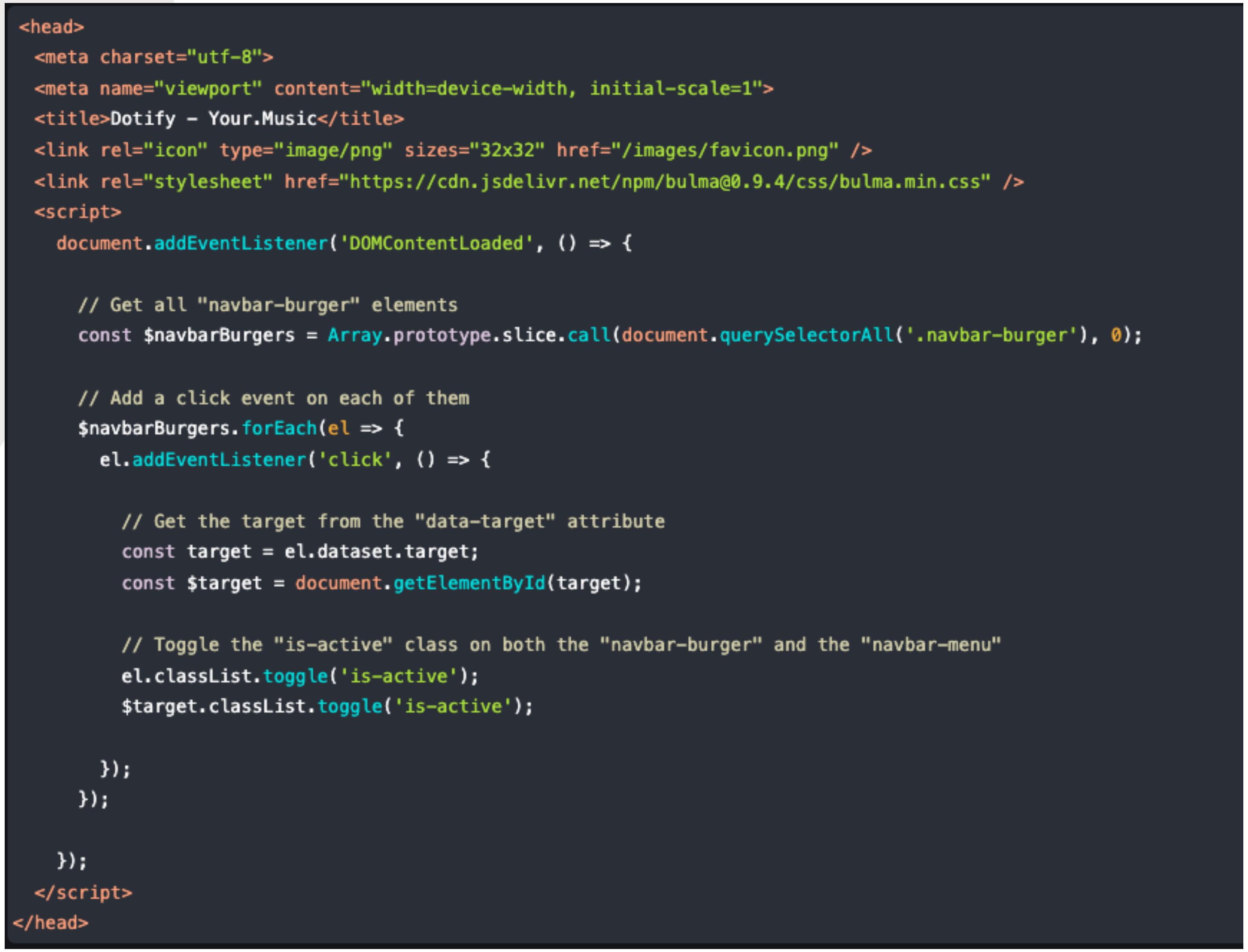
```
<script>
  alert("Hello World!");
</script>
```

- The <script> element can hold JavaScript within its content
- This allows you to put JavaScript right in your HTML file

The Script Element – Inline Script



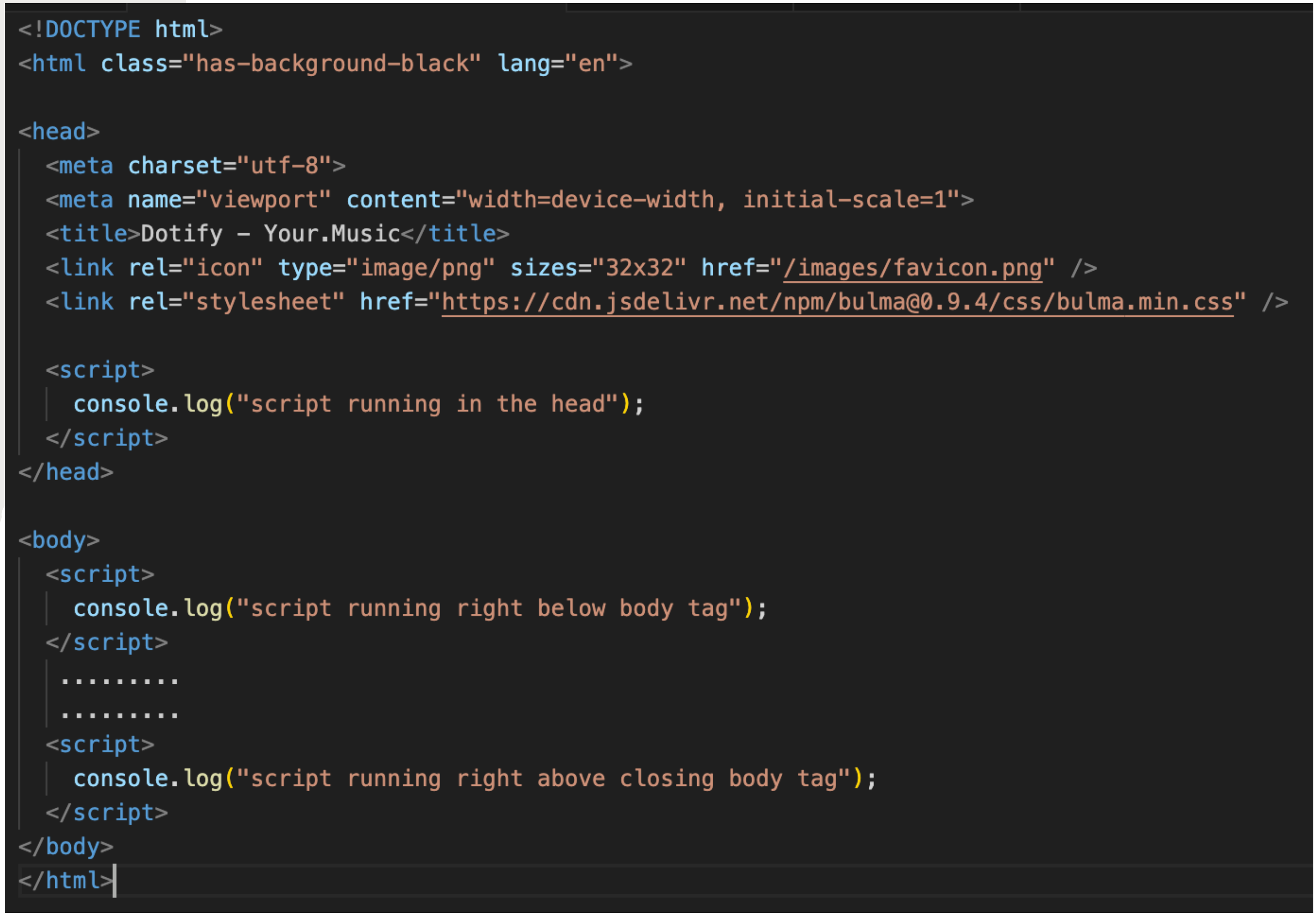
```html
<script>
  alert("Hello World!");
</script>
```



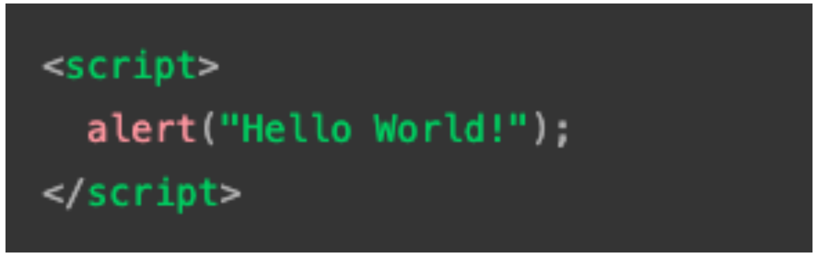
```html
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Dotify - Your.Music</title>
  <link rel="icon" type="image/png" sizes="32x32" href="/images/favicon.png" />
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css" />
  <script>
    document.addEventListener('DOMContentLoaded', () => {

      // Get all "navbar-burger" elements
      const $navbarBurgers = Array.prototype.slice.call(document.querySelectorAll('.navbar-burger'), 0);

      // Add a click event on each of them
      $navbarBurgers.forEach(el => {
        el.addEventListener('click', () => {

          // Get the target from the "data-target" attribute
          const target = el.dataset.target;
          const $target = document.getElementById(target);

          // Toggle the "is-active" class on both the "navbar-burger" and the "navbar-menu"
          el.classList.toggle('is-active');
          $target.classList.toggle('is-active');

        });
      });

    });
  </script>
</head>
```

- Inline scripts run in the order they tag is placed on the page

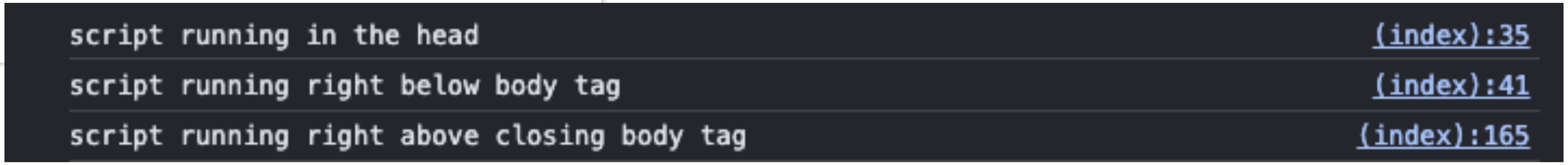
The Script Element – Inline Script



```html
<!DOCTYPE html>
<html class="has-background-black" lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Dotify - Your.Music</title>
  <link rel="icon" type="image/png" sizes="32x32" href="/images/favicon.png" />
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css" />

  <script>
    console.log("script running in the head");
  </script>
</head>

<body>
  <script>
    console.log("script running right below body tag");
  </script>
  .........
  .........
  <script>
    console.log("script running right above closing body tag");
  </script>
</body>
</html>
```

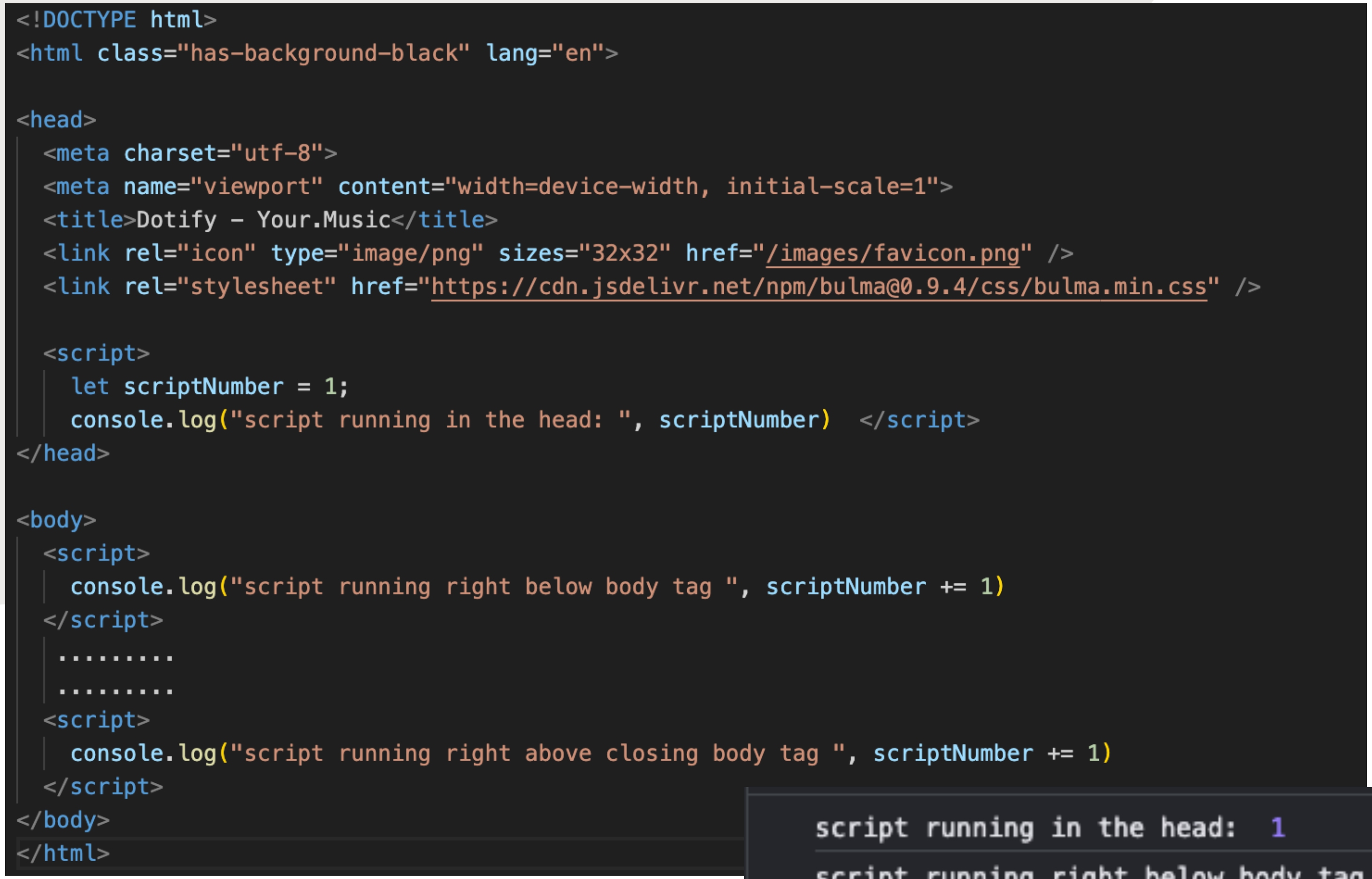


```html
<script>
  alert("Hello World!");
</script>
```

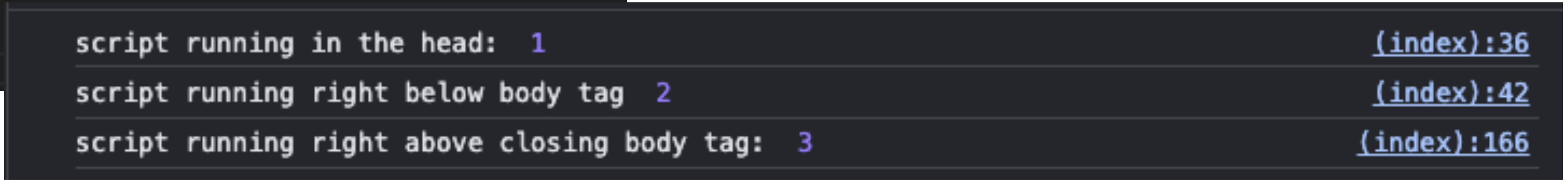


```
script running in the head                                  (index):35
script running right below body tag                         (index):41
script running right above closing body tag                 (index):165
```

The Script Element



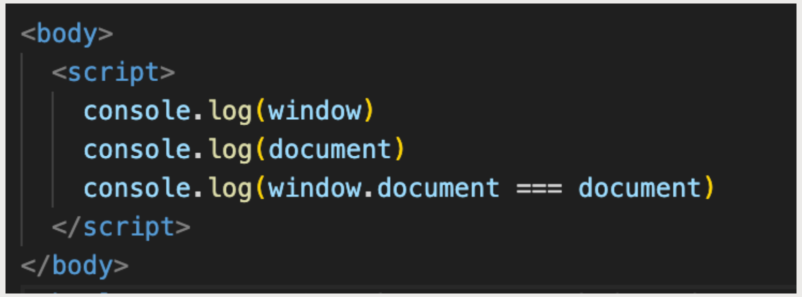
```
<!DOCTYPE html>
<html class="has-background-black" lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Dotify - Your.Music</title>
  <link rel="icon" type="image/png" sizes="32x32" href="/images/favicon.png" />
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css" />

  <script>
    let scriptNumber = 1;
    console.log("script running in the head: ", scriptNumber)  </script>
</head>

<body>
  <script>
    console.log("script running right below body tag ", scriptNumber += 1)
  </script>
   .........
   .........
  <script>
    console.log("script running right above closing body tag ", scriptNumber += 1)
  </script>
</body>
</html>
```



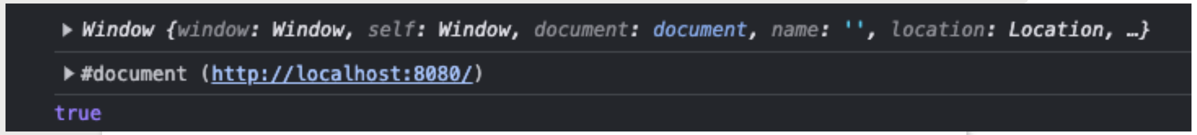
```
script running in the head:  1                                (index):36
script running right below body tag  2                        (index):42
script running right above closing body tag:  3               (index):166
```

- Scripts that run on the same page are all executed within the same scope.
- Scope is the term used to describe where the script is running, what variables it can see and potentially augment
- This can cause some confusion and it is understandable
- Scripts are not executed in isolated environments, they are executed within the context, or scope, of the page

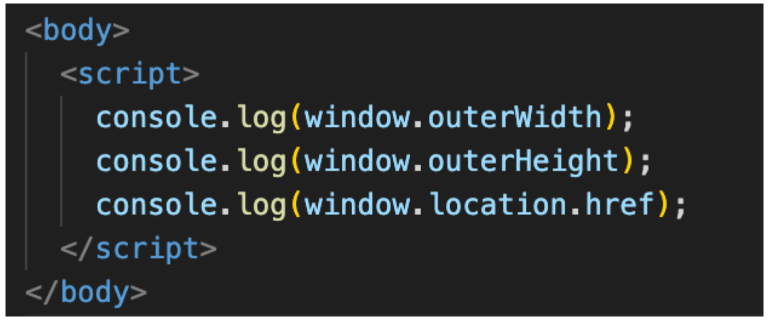
Tell me more about scope



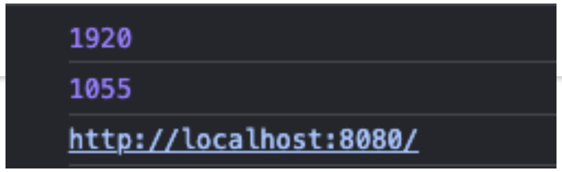
```
<body>
  <script>
    console.log(window)
    console.log(document)
    console.log(window.document === document)
  </script>
</body>
```



```
▸ Window {window: Window, self: Window, document: document, name: '', location: Location, …}
▸ #document (http://localhost:8080/)
true
```



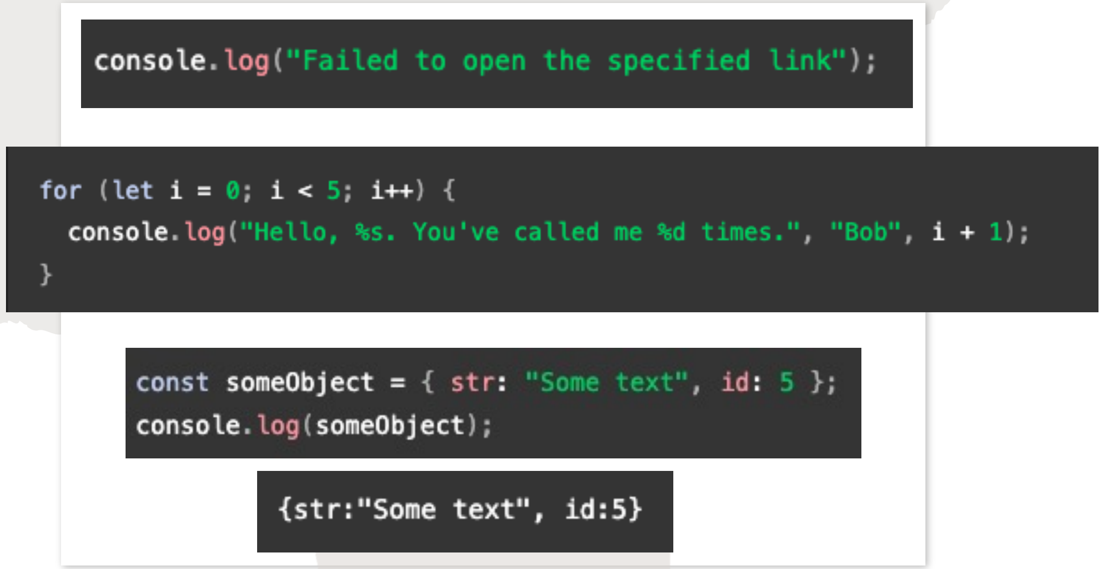
```
<body>
  <script>
    console.log(window.outerWidth);
    console.log(window.outerHeight);
    console.log(window.location.href);
  </script>
</body>
```



```
1920
1055
http://localhost:8080/
```

- Within the scope of the web page, we have access to
  - window - The Window interface represents a window containing a DOM document; the document property points to the DOM document loaded in that window.
  - document - The Document interface represents any web page loaded in the browser and serves as an entry point into the web page's content, which is the DOM tree.
- Window is typically talked about as the “global scope” and global variables can exist here
- It is generally bad to rely on the global scope

Wait, what is console.log?



```
console.log("Failed to open the specified link");
```

```
for (let i = 0; i < 5; i++) {
  console.log("Hello, %s. You've called me %d times.", "Bob", i + 1);
}
```

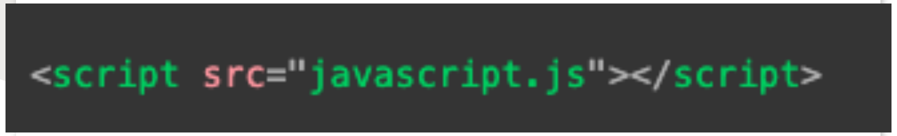
```
const someObject = { str: "Some text", id: 5 };
console.log(someObject);
```

```
{str:"Some text", id:5}
```

- The console.log() static method outputs a message to the console.
  The message may be a single string (with optional substitution values), or it may be any one or more JavaScript objects.
- The console object provides access to the debugging console.
- The specifics of how it works vary from browser to browser or server runtimes (Node.js, for example), but there is a de facto set of features that are typically provided.

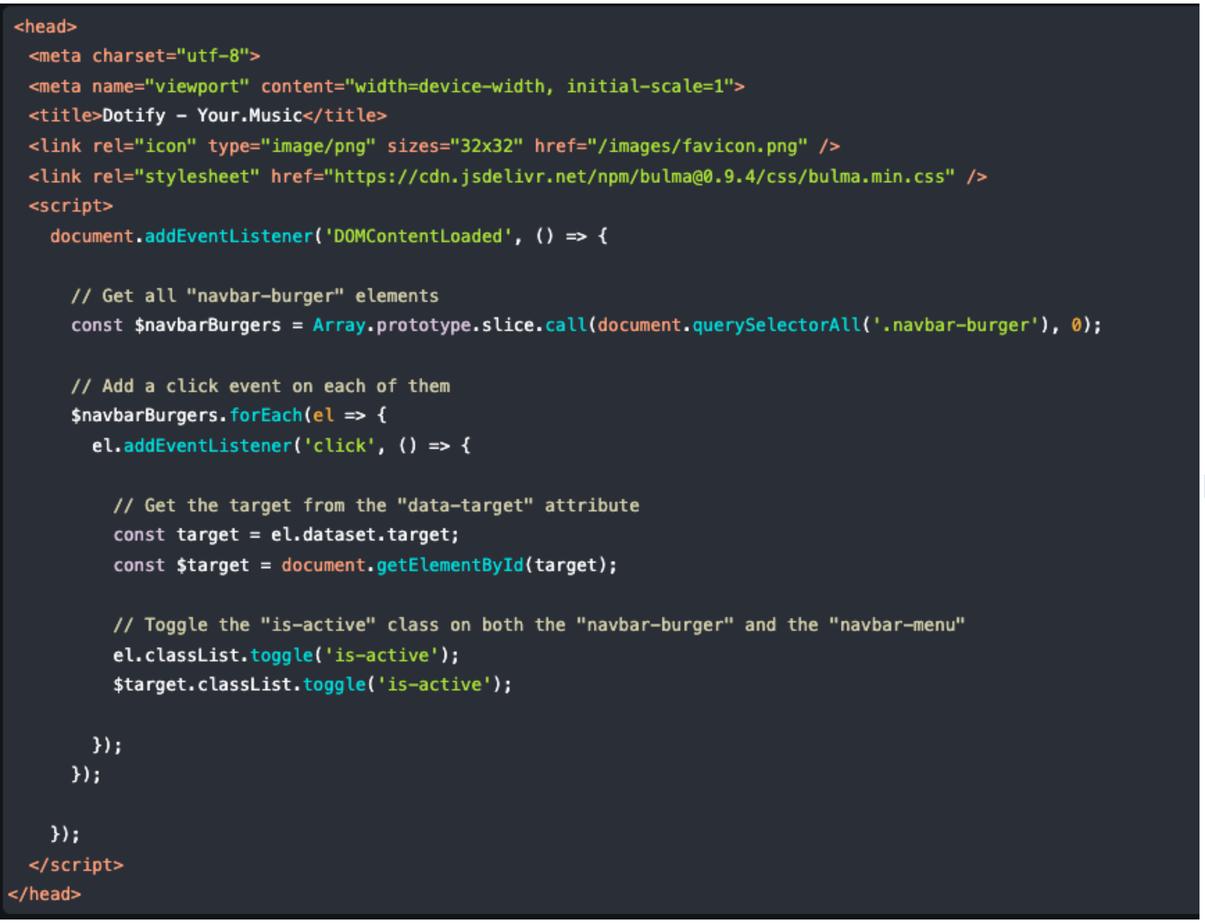
https://developer.mozilla.org/en-US/docs/Web/API/console/log_static

The Script Element – src file

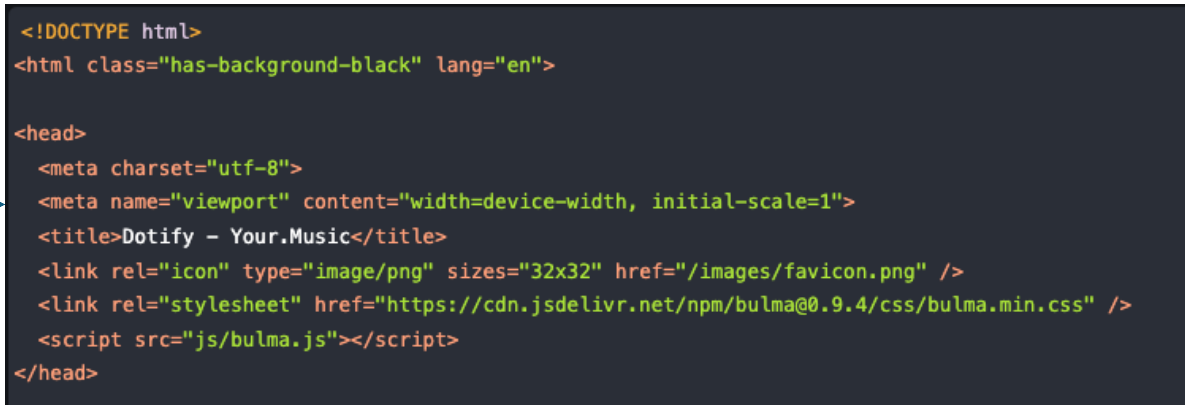


```
<script src="javascript.js"></script>
```

- The <script> element can also hold a path to a file in the src attribute
- This allows you to separate your JavaScript from your HTML

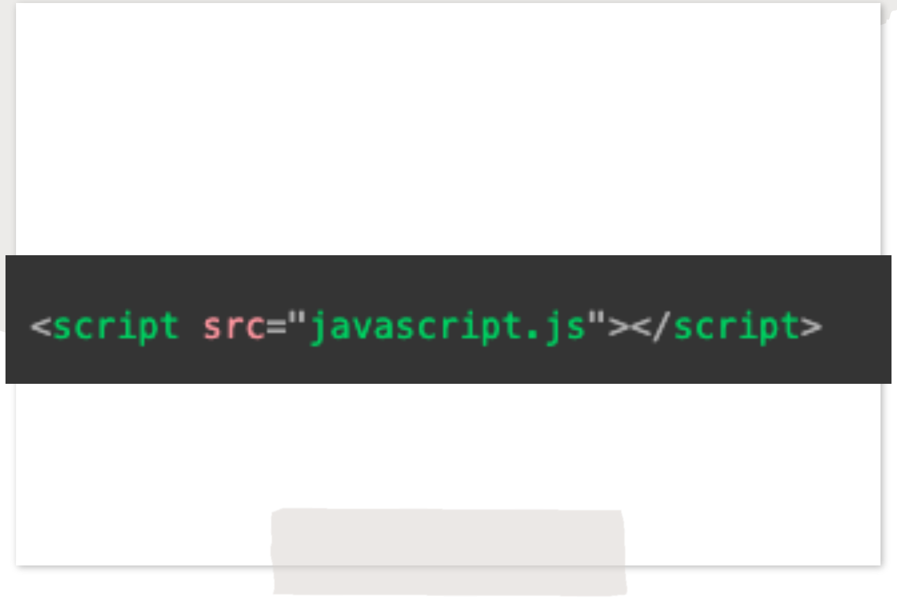


```html
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Dotify - Your.Music</title>
  <link rel="icon" type="image/png" sizes="32x32" href="/images/favicon.png" />
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css" />
  <script>
    document.addEventListener('DOMContentLoaded', () => {

      // Get all "navbar-burger" elements
      const $navbarBurgers = Array.prototype.slice.call(document.querySelectorAll('.navbar-burger'), 0);

      // Add a click event on each of them
      $navbarBurgers.forEach(el => {
        el.addEventListener('click', () => {

          // Get the target from the "data-target" attribute
          const target = el.dataset.target;
          const $target = document.getElementById(target);

          // Toggle the "is-active" class on both the "navbar-burger" and the "navbar-menu"
          el.classList.toggle('is-active');
          $target.classList.toggle('is-active');

        });
      });

    });
  </script>
</head>
```



```html
<!DOCTYPE html>
<html class="has-background-black" lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Dotify - Your.Music</title>
  <link rel="icon" type="image/png" sizes="32x32" href="/images/favicon.png" />
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css" />
  <script src="js/bulma.js"></script>
</head>
```

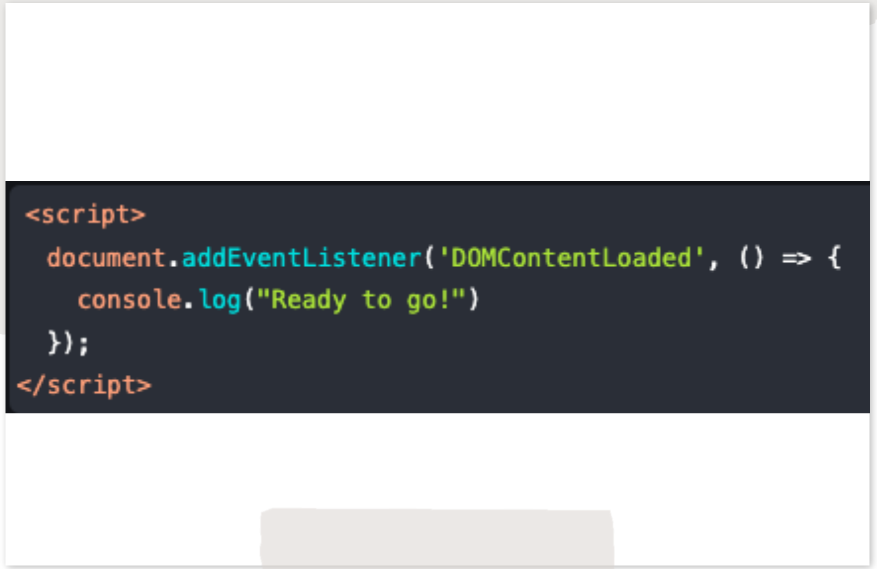
The Script Element – src file



```html
<script src="javascript.js"></script>
```

- Scripts run this way also share the same scope, the document variable is available to you.
- This can be convenient but can introduce a lot of bugs so beware

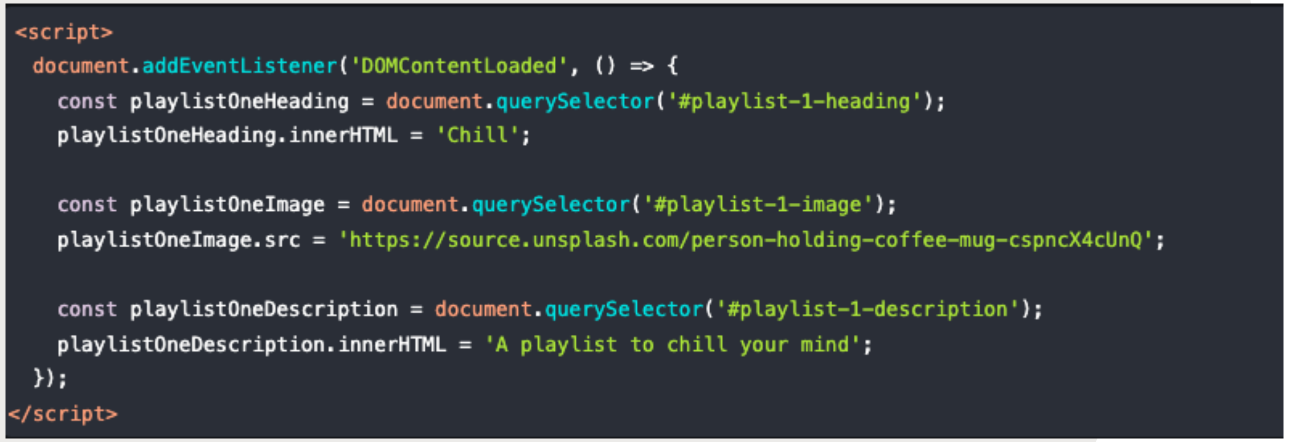
The Script Element - DOMContentLoaded



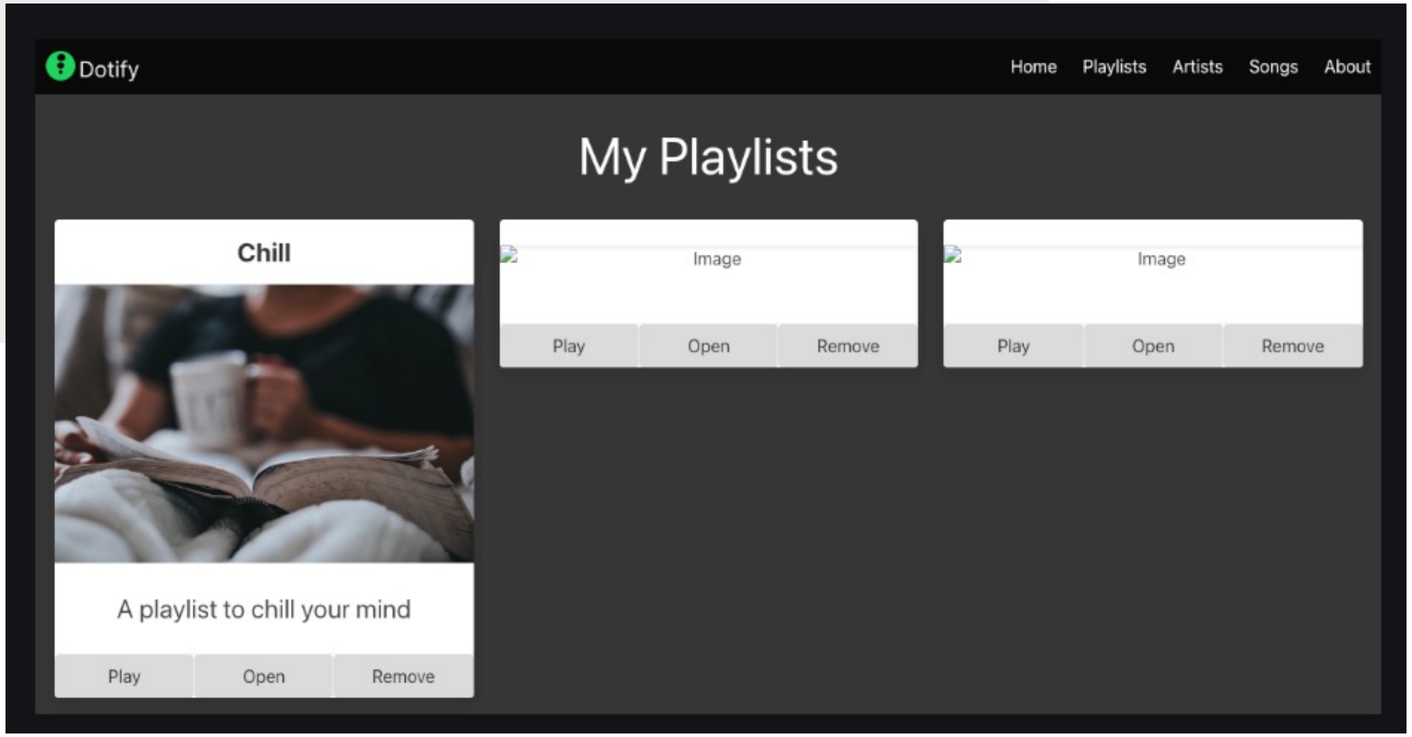
```
<script>
  document.addEventListener('DOMContentLoaded', () => {
    console.log("Ready to go!")
  });
</script>
```

- You may notice in the JavaScript that we added in the last lab that we added an event listener called "DOMContentLoaded"
- Don't worry, we haven't really covered events or listeners
- However, using this event listener it is possible to wait until the entire HTML page is loaded before running your JavaScript
- This can be useful to populate dynamic values

JS in Dotify Lab



```
<script>
  document.addEventListener('DOMContentLoaded', () => {
    const playlistOneHeading = document.querySelector('#playlist-1-heading');
    playlistOneHeading.innerHTML = 'Chill';

    const playlistOneImage = document.querySelector('#playlist-1-image');
    playlistOneImage.src = 'https://source.unsplash.com/person-holding-coffee-mug-cspncX4cUnQ';

    const playlistOneDescription = document.querySelector('#playlist-1-description');
    playlistOneDescription.innerHTML = 'A playlist to chill your mind';
  });
</script>
```



- Remember, in previous lectures we learned about DOM manipulation using the JS console in our browser
- In this lecture we have seen how we can add JS to a webpage
- We will now combine both ideas to add JavaScript to our Dotify playlists page
- In our lab we will
  - Refactor Dofity so that it’s Hamburger menu JavaScript is in a separate file
  - Debug issues with our JavaScript directly in the browser
  - Learn about and use DOMContentLoaded
  - Add content to our playlist cards using JavaScript