7.1 Using JavaScript with HTML

Basics

JavaScript allows webpages to be more interactive. JavaScript can execute in response to user interactions and alter the contents of the webpage. Ex: A user clicks on a button, and JavaScript executes and changes the color of the webpage.

The ***Document Object Model*** (or ***DOM***) is a data structure that represents all parts of an HTML document. The JavaScript object ***document*** represents the entire DOM and is created from the document's HTML. Changes made to `document` are reflected in the browser presentation and/or behavior.

Webpages add JavaScript code by using the `<script>` tag. JavaScript code between `<script></script>` tags is executed by the browser's JavaScript engine.

PARTICIPATION ACTIVITY 7.1.1: Writing JavaScript within the body of an HTML file.

The JavaScript code below uses the ***document.writeln()*** method, which outputs HTML into the document and alters the DOM.

1. Read the HTML and JavaScript below.
2. Render the webpage to run the JavaScript code that displays a randomly generated response.
3. Add more responses to the `responses` array, and render the webpage a few times until one of your new responses is displayed.

```
6  </head>
7  <body>
8     <h1>Magic 8 Ball</h1>
9     <script>
10
11    // Possible 8 Ball responses
12    let responses = [ "Without a doubt", "Ask again later", "Dor
13
14    // Display a randomly chosen response
15    let randomNum = Math.floor(Math.random() * responses.length)
16    document.writeln("<p>Magic 8 Ball says... <strong>" + respor
17
18    </script>
19 </body>
20 </html>
21
```

Render webpage Reset code

Your webpage

Magic 8 Ball

Magic 8 Ball says... **Ask again later**.

▼ View solution

Explain

```
--- START FILE: HTML ---


<!DOCTYPE html>
<html>
<head>
   <title>Magic 8 Ball</title>
   <meta charset="UTF-8">
</head>
<body>
   <h1>Magic 8 Ball</h1>
   <script>

   // Possible 8 Ball responses
   let responses = [ "Without a doubt", "Ask again later",
"Don't count on it", "I see it in your future", "Forget about
it", "Try again" ];

   // Display a randomly chosen response
   let randomNum = Math.floor(Math.random() *
responses.length);
   document.writeln("<p>Magic 8 Ball says... <strong>" +
responses[randomNum] + "</strong>.</p>");

   </script>
</body>
</html>
```

```
--- END FILE: HTML ---
```

Feedback?

PARTICIPATION ACTIVITY 7.1.2: JavaScript Basics.

1) The DOM is created from a document's HTML.
   - (•) True
   - ( ) False

Correct

The document's HTML is used to create the DOM, but the DOM may contain additional features not present in the HTML. JavaScript can alter the DOM.

2) The DOM is accessible via the global object named `document`.
   - (•) True
   - ( ) False

Correct

For ease of access, the DOM is available in any JavaScript code from the `document` object.

3) `document.writeln("<div>test</div>")` adds a `div` element to the DOM.
   - (•) True
   - ( ) False

Correct

Output from `document.writeln()` alters the DOM.

Feedback?

Window object

JavaScript running in a web browser has access to the ***window*** object, which represents an open browser window. In a tabbed browser, each tab has a `window` object. The `document` object is a property of the `window` object and can be accessed as `window.document` or just `document`. Other properties of the `window` object include:

- ***window.location*** is a location object that contains information about the window's current URL. Ex: `window.location.hostname` is the URL's hostname.
- ***window.navigator*** is a navigator object that contains information about the browser. Ex: `window.navigator.userAgent` returns the browser's user agent string.
- ***window.innerHeight*** and ***window.innerWidth*** are the height and width in pixels of the window's content area. Ex: `window.innerWidth` returns 600 if the browser's content area is 600 pixels wide.

The `window` object defines some useful methods:

- ***window.alert()*** displays an alert dialog box. Ex: `window.alert("Hello")` displays a dialog box with the message "Hello".
- ***window.confirm()*** displays a confirmation dialog box with OK and Cancel buttons. `confirm()` returns true if OK is pressed and false if Cancel is pressed. Ex: `window.confirm("Are you sure?")` displays a dialog box with the question.
- ***window.open()*** opens a new browser window. Ex: `window.open("http://www.twitter.com/")` opens a new browser that loads the Twitter webpage.

PARTICIPATION ACTIVITY 7.1.3: Using the window object.

Use the `window.confirm()` method to ask if the user would like to see a popup window:

```
let okPressed = window.confirm("Would you like to see a popup
window?");
```

Then render the webpage, and click the OK button when prompted to see a small browser window created by `window.open()`. You may need to give your browser permission to show the popup window since many browsers prevent popups from displaying by default.

```
5     <meta charset="UTF-8">
6  </head>
7  <body>
8     <h1>Popup Demo</h1>
9     <script>
10
11    let okPressed = window.confirm("Would you like to see a popu
12    if (okPressed) {
13       let myWindow = window.open("", "", "width=250, height=100
14       myWindow.document.writeln("<h1>Hello, Popup!</h1>");
15    }
16
17    </script>
18 </body>
19 </html>
20
```

Render webpage Reset code

Your webpage

Popup Demo

▼ View solution

Explain

```
--- START FILE: HTML ---


<!DOCTYPE html>
<html>
<head>
   <title>JavaScript Demo</title>
   <meta charset="UTF-8">
</head>
<body>
   <h1>Popup Demo</h1>
   <script>

   let okPressed = window.confirm("Would you like to see a
popup window?");
   if (okPressed) {
      let myWindow = window.open("", "", "width=250,
height=100");
      myWindow.document.writeln("<h1>Hello, Popup!</h1>");
   }

   </script>
</body>
</html>


--- END FILE: HTML ---
```

Feedback?

PARTICIPATION ACTIVITY 7.1.4: Window object.

1) Can window object properties and methods be accessed without putting `window.` in front of the property or method?

- (•) Yes
- ( ) No

Correct

JavaScript properties and methods that do not have an explicitly named object are associated with the window object by default. Ex: `window.alert("test")` and `alert("test")` are equivalent.

2) What window object property is useful for determining if the webpage is loaded with HTTPS or HTTP?

- (•) location
- ( ) navigator
- ( ) innerHeight

Correct

`window.location.protocol` is "https:" or "http:", depending on which protocol the URL is using.

3) What window object property likely produces the following output?

```
document.writeln(window.navigator._____);
```

```
Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 Chrome/53.0.2785.116
```

- ( ) language
- (•) userAgent
- ( ) vendor

Correct

`window.navigator.userAgent` is the browser's user agent, which differs between browsers.

4) The _____ window method is ideal for displaying a pop-up advertisement.

- ( ) alert()
- ( ) confirm()
- (•) open()

Correct

`window.open()` opens a new browser window that renders HTML, which is ideal for showing text and images in advertisements.

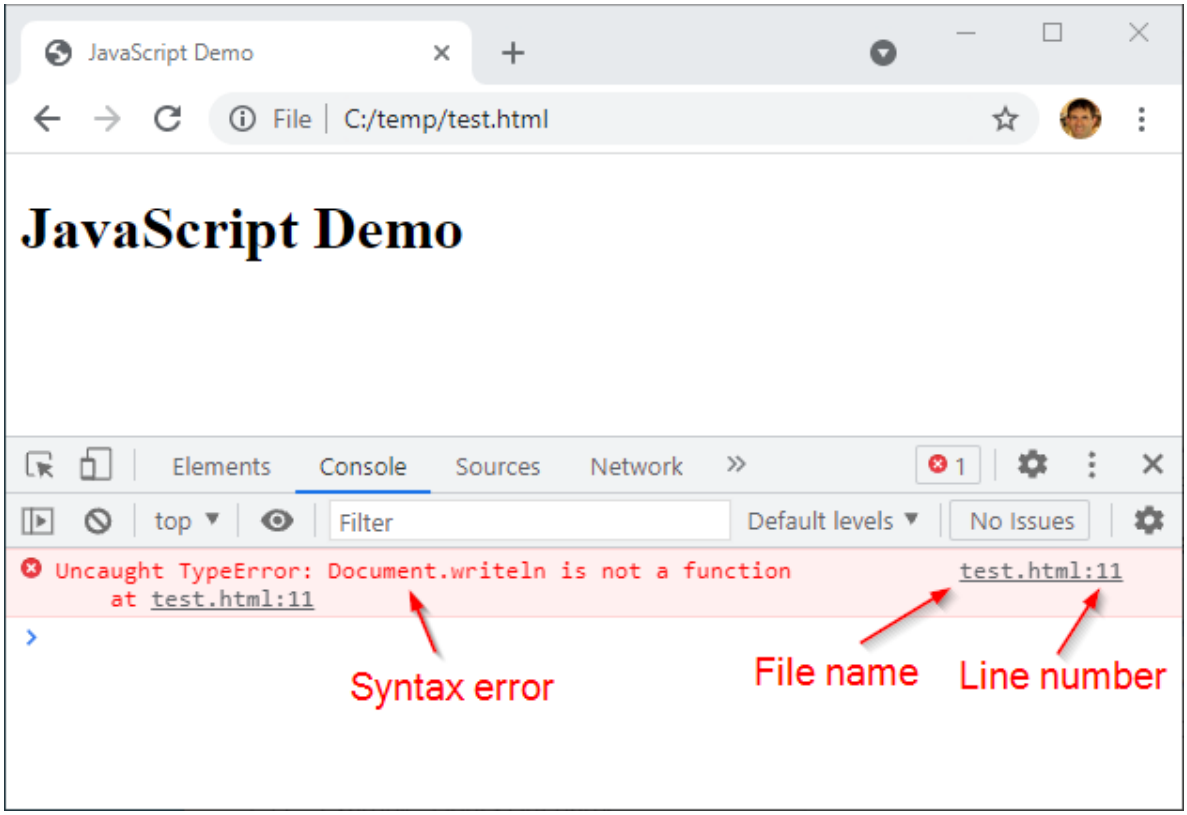
Feedback?

Using the console

Modern browsers provide a ***console*** that allows the JavaScript code to produce informational and debugging output for the web developer, which does not affect the functionality or presentation of the webpage. By default, the console is not visible. The console is viewable in Chrome by pressing Ctrl+Shift+J in Windows/Linux or Cmd+Opt+J on a Mac.

When a syntax error is present in JavaScript code or a run-time error occurs, the error is only made visible in the console. The figure below shows the syntax error created when the developer accidentally typed `Document.writeln()` with a capital "D". The console appears underneath the webpage. *Good practice is to leave the console open while writing and testing JavaScript code.*

Figure 7.1.1: Chrome console showing a syntax error on line 11 of test.html.




```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>JavaScript Demo</title>
</head>
<body>
   <h1>JavaScript Demo</h1>
   <script>

   Document.writeln("<p>Hello, JavaScript!</p>");

   </script>
</body>
</html>
```

Feedback?

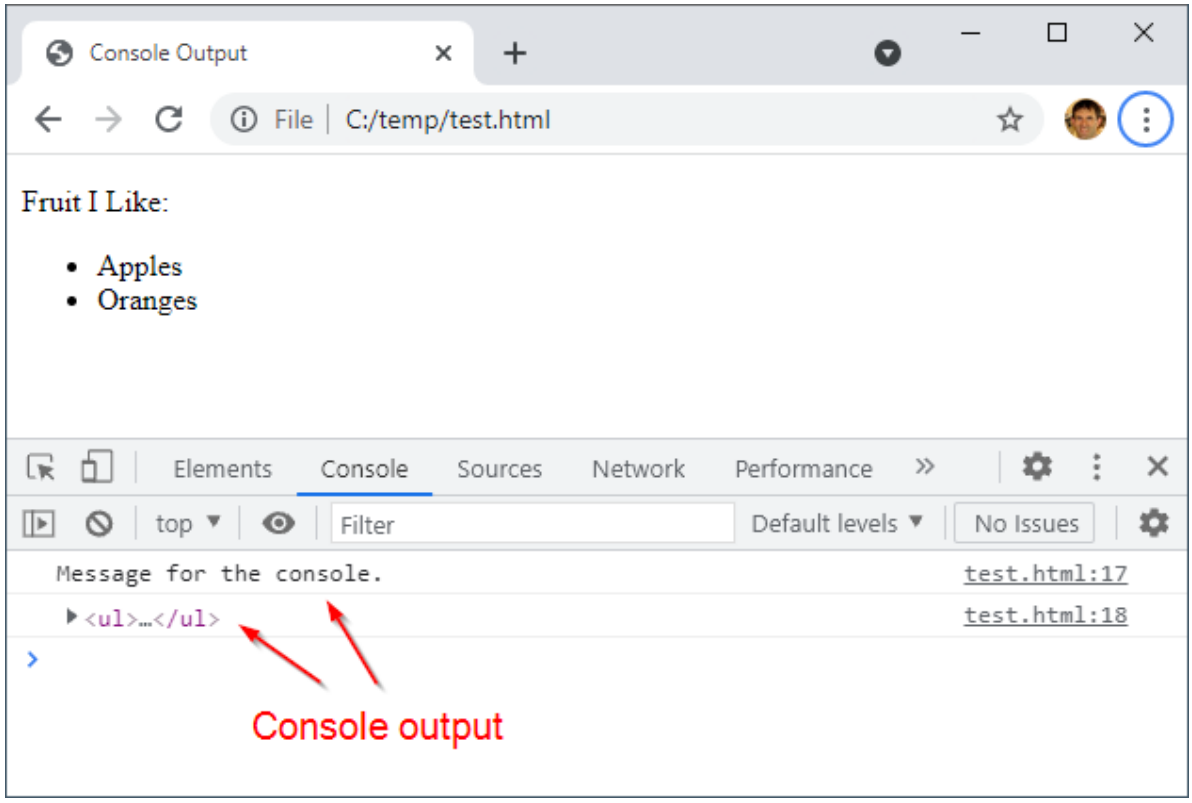
The browser provides a `console` object with a defined set of methods, or API, that the `console` object supports. An ***API*** (***Application Programming Interface***) is a specification of the methods and objects that defines how a programmer should interact with software components. The console API includes the following methods:

- ***console.log()*** displays informational data to the console.
- ***console.warn()*** displays warnings to the console. The browser usually has a special indicator to differentiate a warning from the standard log message. Ex: A yellow warning box.

- ***console.error()*** displays errors to the console. The browser usually has a special indicator to differentiate an error from a warning or the standard log message. Ex: A red error box.
- ***console.dir()*** displays a JavaScript object to the console. The browser usually supports a method for compactly representing the object. Ex: a hierarchical tree representation allowing a developer to expand and collapse the object contents.

Figure 7.1.2: console.log() output example.

When the web browser console is open, both the webpage and the console are simultaneously visible.



console.log() can print both strings and concise representations of HTML elements.

```
<body>
   <p>
      Fruit I Like:
   </p>
   <ul>
      <li>Apples</li>
      <li>Oranges</li>
   </ul>

   <script>
console.log("Message for the console.");
console.log(document.getElementsByTagName("ul")[0]);
   </script>
</body>
```

Feedback?

PARTICIPATION ACTIVITY 7.1.5: Console methods.

Match the console method with the best use for that method.

If unable to drag and drop, refresh the page.

log()	Helping determine why code isn't working as expected. console.log() is useful for printing debugging messages that help the programmer. console.log() is also helpful for seeing simplified visualizations of JavaScript data or parts of the DOM.	Correct
dir()	Displaying a structured JavaScript object. console.dir() is useful for displaying an object's properties for debug purposes.	Correct
warn()	Checking that assumptions in code are correct. Sometimes a developer may assume a variable has a certain value. The console.warn() method can be placed in strategic places in the code to indicate when those assumptions are not true.	Correct
error()	Reporting unexpected problems. The console.error() method can be called to display the state of the code for debugging. A well-written error message can be used by the developer to reproduce the problem and fix the bug.	Correct

Reset

Feedback?

Loading JavaScript from an external file

Including JavaScript directly within an HTML file is common practice when using small amounts of JavaScript. However, writing JavaScript directly within the document may lead to problems as a webpage or website gets larger.

Good practice is to use `<script>` *tags to load JavaScript from an external file rather than writing the JavaScript directly within the HTML file.* The `<script>` tag's `src` attribute specifies a JavaScript file to load.

Example 7.1.1: Loading JavaScript from an external file.

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

Feedback?

A common error when loading an external JavaScript file is to forget the closing `</script>` *tag or trying to use a self-closing* `<script />` *tag as in* `<script src="bootstrap.js" />`*. All modern browsers require a closing* `</script>` *tag.*

PARTICIPATION ACTIVITY 7.1.6: Loading an external JavaScript file.

Start 2x speed

index.html

```
<!DOCTYPE html>
<html>
  <title>JavaScript Example</title>
  <script src="file.js"></script>
  <body>
    <p>A piece of text.</p>
    <img src="dog.jpg" alt="dog">
    <p>Some more text.</p>
  </body>
</html>
```

file.js

```
alert("Press enter to continue.");
```

Web browser

A piece of text.

Some more text.

Web server

Captions

1. The web server sends index.html to the web browser.
2. Web browser reads the HTML file. The <script> tag with src attribute indicates the browser should load JavaScript from an external file.
3. Web browser requests file.js from the web server.
4. Web browser reads and executes the JavaScript file. The alert() function displays a dialog box and waits for the user to press enter.
5. After the user presses enter, web browser finishes reading the JavaScript file and continues reading the HTML file.
6. Web browser requests the image file, and the web server responds with the image file.
7. Web browser finishes reading HTML file.

Feedback?

PARTICIPATION ACTIVITY 7.1.7: Downloading JavaScript files.

1) A web browser will process the HTML following a script element that uses an external JavaScript file while the browser waits for the web server to return the JavaScript file.

- True
- False (selected)

Correct

A web browser will wait for a script element's external file to load before continuing to process the HTML. The `async` and `defer` attributes, covered later, allow the browser to continue processing HTML while the JavaScript file downloads.

2) One script element can be used to include both inline JavaScript and a reference to an external JavaScript file.

- True
- False (selected)

Correct

A script element can either be used for inline JavaScript or an external JavaScript file, not both.

3) One script element can be used to reference multiple external JavaScript files.

- True
- False (selected)

Correct

Each external JavaScript file must be referenced by separate script elements.

Feedback?

Loading JavaScript with async and defer

Although the `<script>` tag can be included anywhere in the head or body, good practice is to include the `<script>` tag in the head with the `async` or `defer` attributes set.

The `<script>` tag's ***async attribute*** allows the browser to process the webpage concurrently with loading and processing the JavaScript.

The `<script>` tag's ***defer attribute*** allows the browser to load the webpage concurrently with loading the JavaScript, but the JavaScript is not processed until the webpage is completely loaded.

PARTICIPATION ACTIVITY 7.1.8: Using the async attribute with the <script> tag.

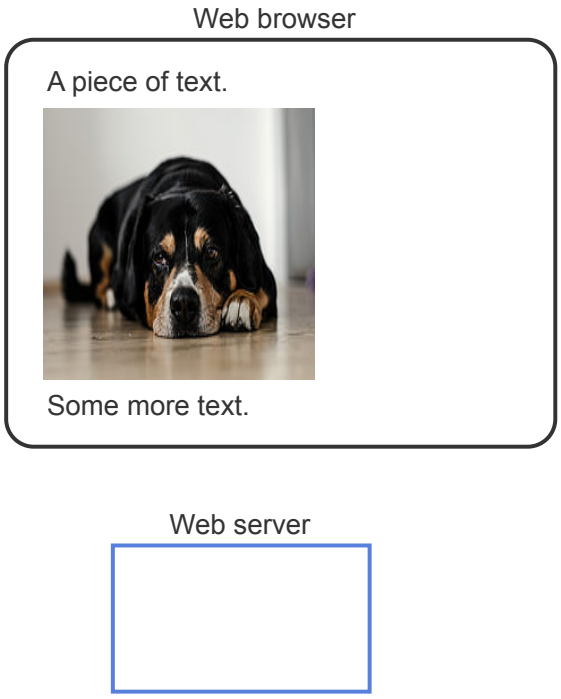
1 2 3 4 5 2x speed

index.html

```
<!DOCTYPE html>
<html>
  <title>Async Example</title>
  <script src="file.js" async></script>
  <body>
    <p>A piece of text.</p>
    <img src="dog.jpg" alt="dog">
    <p>Some more text.</p>
  </body>
</html>
```

file.js

```
alert("Press enter to continue.");
```



Web browser finishes reading HTML file.

Captions

1. Web browser reads index.html. <script> tag's async attribute causes browser to continue reading HTML without waiting for JavaScript file to load.
2. Web server responds with file.js while the browser requests the image file.
3. Web browser begins reading and executing the JavaScript file and pauses reading the HTML file. The web server concurrently responds to the image request.
4. After the user presses enter, web browser finishes reading the JavaScript file and continues processing the HTML file by displaying the dog.jpg image that was received.

5. Web browser finishes reading HTML file.

Feedback?

PARTICIPATION ACTIVITY 7.1.9: Using the defer attribute with the <script> tag.

1 2 3 4 5 2x speed

index.html

```
<!DOCTYPE html>
<html>
  <title>Defer Example</title>
  <script src="file.js" defer></script>
  <body>
    <p>A piece of text.</p>
    <img src="dog.jpg" alt="dog">
    <p>Some more text.</p>
  </body>
</html>
```

file.js

```
alert("Press enter to continue.");
```

Web browser

A piece of text.

Some more text.

Web server

After the user presses enter, web browser finishes reading the JavaScript file.

Captions

1. Web browser reads index.html. <script> tag's defer attribute causes browser to continue reading HTML without waiting for JavaScript file to load.
2. Web server responds with the JavaScript file while the browser requests the image file.
3. Web browser does not immediately process the JavaScript file due to the defer attribute. Instead, the browser continues to process the HTML.
4. After reading the HTML file, the web browser reads and executes the JavaScript file.
5. After the user presses enter, web browser finishes reading the JavaScript file.

Feedback?

PARTICIPATION ACTIVITY 7.1.10: Loading JavaScript.

1) The browser interprets the `defer` and `async` attributes for the script element the same.

Correct

The `defer` attribute indicates the browser should wait to interpret the JavaScript until the HTML has been processed, while the `async` attribute indicates the

- ○ True
- ◉ False

browser can process the HTML and JavaScript concurrently.

2) When using a third-party JavaScript library, the `defer` attribute is usually better than the `async` attribute.

- ◉ True
- ○ False

Correct

A third-party JavaScript library may assume that the DOM is already loaded. Thus, the `defer` attribute ensures the DOM is loaded before the JavaScript. However, a well written library will not make assumptions about the DOM availability.

3) When writing custom JavaScript, the `defer` attribute is usually better than the `async` attribute.

- ○ True
- ◉ False

Correct

Custom JavaScript can take advantage of concurrent processing with HTML. As a result, a developer may see better performance using the `async` attribute instead of the `defer` attribute.

4) Most webpages on the internet were written before the `defer` or `async` attributes were standardized.

- ◉ True
- ○ False

Correct

Many webpages do not use asynchronous or deferred execution of JavaScript. The load speed of these webpages could be improved by using the `async` and `defer` attributes.

Feedback?

Minification and obfuscation

To reduce the amount of JavaScript that must be downloaded from a web server, developers often minify a website's JavaScript. ***Minification*** or ***minimization*** is the process of removing unnecessary characters (like whitespace and comments) from JavaScript code so the code executes the same but with fewer characters. Minification software may also rename identifiers into shorter ones to reduce space. Ex: `let totalReturns = 10;` may be converted into `let a=10;`.

Minified JavaScript is typically stored in a file with a ".min.js" file extension. An example of minified code from the Bootstrap project is shown below.

```
// Excerpt from bootstrap.min.js
a.fn.button=b,a.fn.button.Constructor=c,a.fn.button.noConflict=function(
{
return a.fn.button=d,this},a(document).on("click.bs.button.data-api",
'[data-toggle^="button"]',function(c){let d=a(c.target).closest(".btn");
b.call(d,"toggle"),a(c.target).is('input[type="radio"],
```

A JavaScript ***obfuscator*** is software that converts JavaScript into an unreadable form that is very difficult to convert back into readable JavaScript. Developers obfuscate a website's JavaScript to prevent the code from being read or re-purposed by others. Obfuscated code may also be minified and appear in a ".min.js" file.

CHALLENGE ACTIVITY 7.1.1: JavaScript with HTML.

530096.4000608.qx3zqy7

Start

1

2

Use the writeln() method of the document object to display the current platform in a <p> tag in the webpage. Hint: The platform property of the window.navigator object contains the current platform.

```
1 <h1>Demo</h1>
2 <script>
3
4    <!-- Your solution goes here -->
5
6 </script>
```

1 2

Check Next

View your last submission

```
<!-- Your solution goes here -->
document.writeln('<p>' + window.navigator.platform + '</p>');
```

Feedback?

Exploring further:

- Window object from MDN
- Console object from MDN
- async vs defer attributes from Growing with the Web
- JavaScript minifiers: javascript-minifier.com and jscompress.com
- JavaScript obfuscators: javascriptobfuscator.com and JS-obfus

How was this section? Provide section feedback