Set 20: Document Object Model

Skill 20.01: Explain the Document Object Model (DOM)

Skill 20.01 Concepts

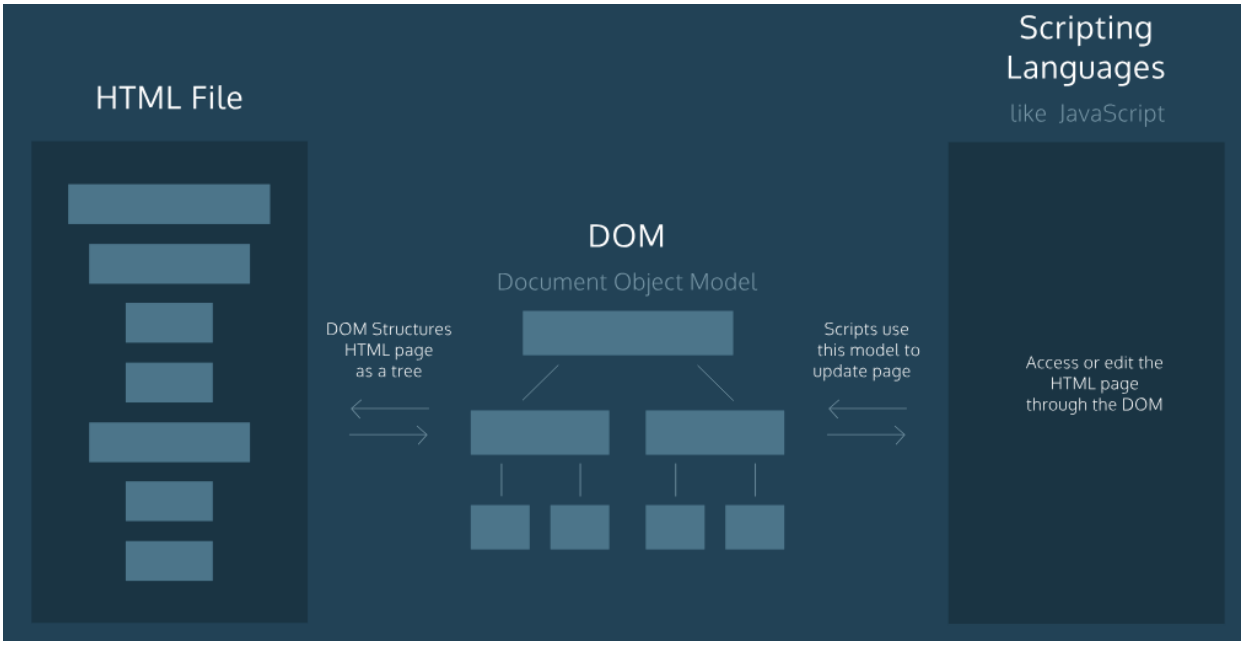
The Document Object Model, abbreviated DOM, is a powerful tree-like structure that allows programmers to conceptualize hierarchy and access the elements on a web page.

The DOM is one of the better-named acronyms in the field of Web Development. In fact, a useful way to understand what DOM does is by breaking down the acronym but out of order:

> The DOM is a logical tree-like **M**odel that organizes a web page's HTML **D**ocument as an **O**bject.

The DOM is a language-agnostic structure implemented by browsers to allow for web scripting languages, like JavaScript, to access, modify, and update the structure of an HTML web page in an organized way.

For this reason, we like to think of the DOM as the link between an HTML web page and scripting languages.

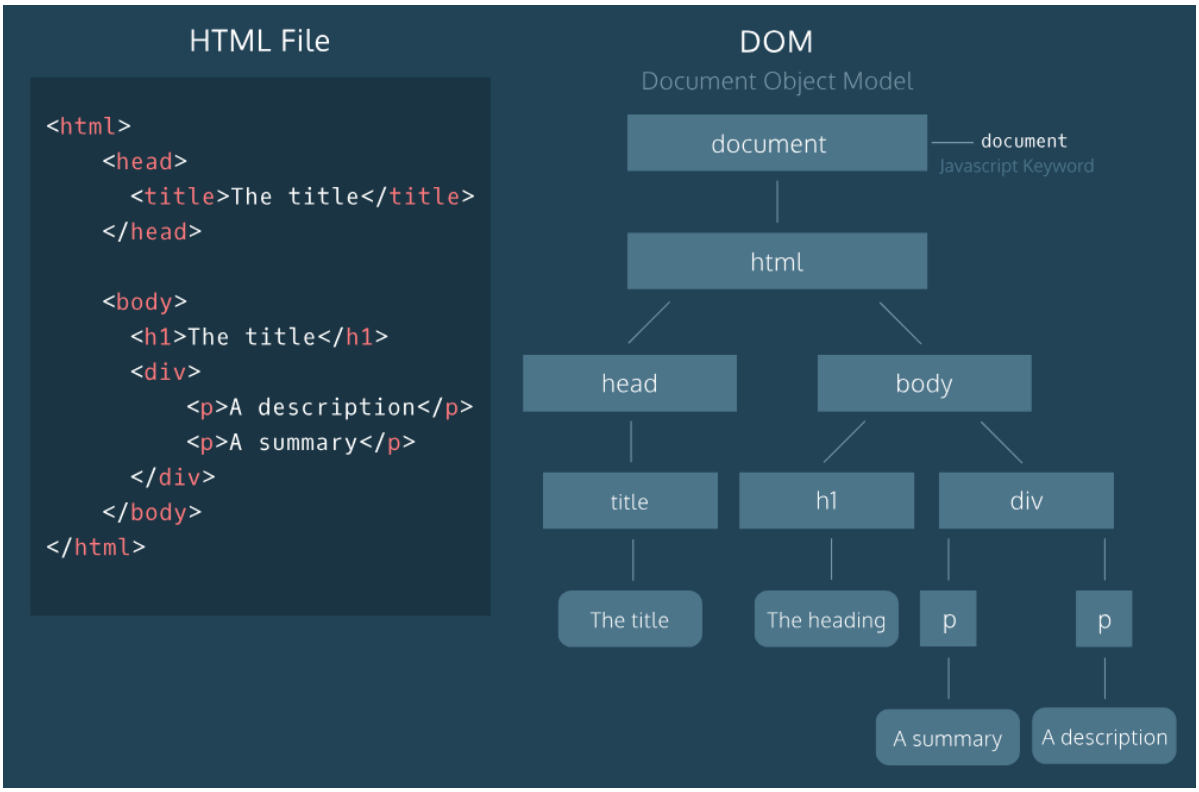


Tree-like modeling is used in many fields, including evolutionary science and data analytics. Perhaps you're already familiar with the concept of family trees: these charts represent the familial relationships amongst the descendants of a given family name.

The DOM tree follows similar logic to that of a family tree. A family tree is made up of family members and their relationships to the family name. In computer science, we would call each family member a *node*.

We define a *node* as an intersecting point in a tree that contains data. In the DOM tree, the top-most node is called the root node, and it represents the HTML document. The descendants of the root node are the HTML tags in the document, starting with the <html> tag followed by the <head> and <body> tags and so on.

The diagram below models the HTML document and labels the root element, which is the document. Observe the difference in the rectangular boxes and the curved boxes. These denote a difference in the types of nodes in the DOM structure.



There are nine different types of node objects in the DOM tree above. In our diagram, the node objects with the sharp-edge rectangles are of the type *Element*, while the rounded edge rectangles are of type *Text*, because they represent the text inside the HTML paragraph elements.

When trying to modify a web page, the script will mostly interact with the DOM nodes of type *Element*. Elements are the building units of HTML web pages, they contain everything between an opening tag and a closing tag. If the tag is a self-closing tag, then that is the element itself.

Skill 20.01 Exercise 1

Skill 20.02: Use JavaScript to access DOM elements

Skill 20.02 Concepts

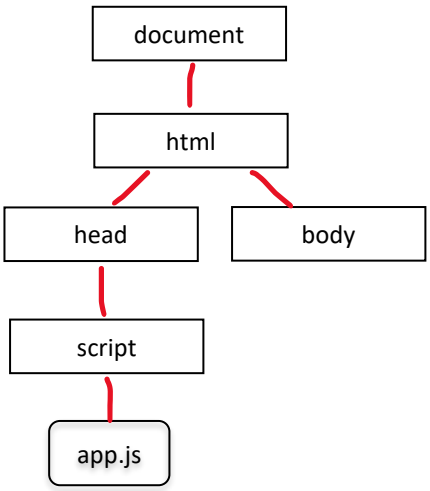
The *document* object is JavaScript's "door" to the HTML page. Using javascript we can access nodes of the DOM tree and once they are accessed, we can manipulate them. Using javascript, we can also create elements on the DOM tree and add them to our HTML page.

In the example below, the DOM for a basic HTML page is shown. Using javascript to interact with the DOM produces the following output. While it isn't necessary to know what all the code in the app.js file means, you can probably read the code and make out what is going on.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<script src = "app.js"
defer></script>
</head>
<body>

</body>
</html>
```

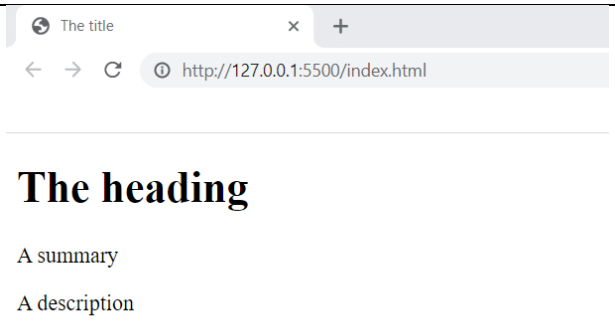
Document Object Model (DOM)



app.js

```
var t = document.createElement("title");
t.innerHTML = "The title";
document.head.append(t);

var h = document.createElement("h1");
h.innerHTML = "The heading";
document.body.append(h);

var d = document.createElement("div");
document.body.append(d);

var summary = document.createElement("p");
summary.innerHTML = "A summary";
d.append(summary);

var description =
document.createElement("p");
description.innerHTML = "A description";
d.append(description);
```

Browser output



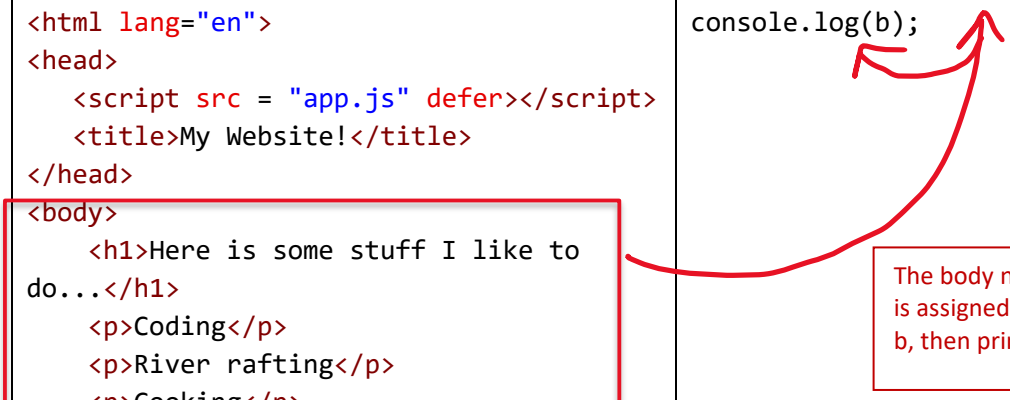
The *document* object allows scripts to access elements or nodes of the DOM.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <script src = "app.js" defer></script>
   <title>My Website!</title>
</head>
<body>
    <h1>Here is some stuff I like to
do...</h1>
    <p>Coding</p>
    <p>River rafting</p>
    <p>Cooking</p>
    <p>Travelling</p>
</body>
</html>
```

app.js

```
var b = document.body;
console.log(b);
```



The body node of the html page is assigned to a variable called b, then printed to the console.

Console output

```
▼<body data-new-gr-c-s-check-loaded="14.1108.0" data-gr-
 ext-installed>
   <h1>Here is some stuff I like to do...</h1>
   <p>Coding</p>
   <p>River rafting</p>
   <p>Cooking</p>
   <p>Travelling</p>
   <!-- Code injected by live-server -->
  ▶<script> ... </script>
 </body>
```

[Skill 20.02 Exercises 1]

Skill 20.03: Use JavaScript to access elements by id

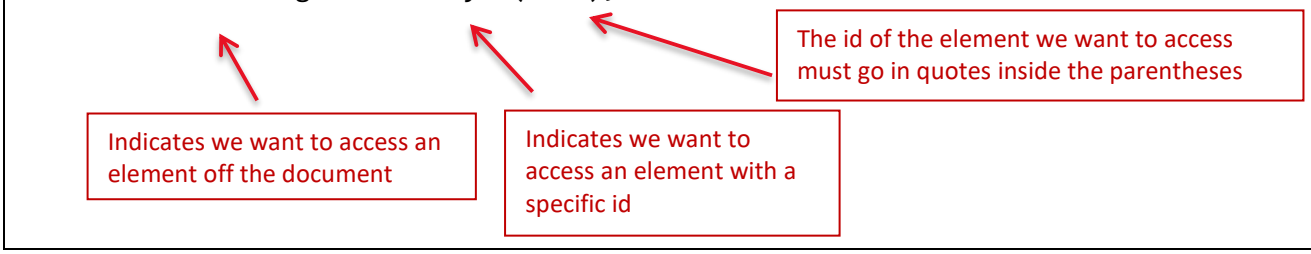
Skill 20.03 Concepts

In the above examples, we saw how to access entire nodes of our webpage. Using an id attribute allows us more control of what we can access. Below illustrates how to add an id attribute to an HTML element.

```
<h1> My Hobbies </h1>
<p id = "hobbie1">Soccer</p>
<p id = "hobbie2">Chess</p>
<p id = "hobbie3">Cooking</p>
```

To access each element we can use the *getElementById* command,

```
var h1 = document.getElementById("h1");
```



Below is a complete example,

Index.html

```
<!DOCTYPE html>
<html>
<head>
    <script src="app.js"
defer></script>
    <title>About Me</title>
</head>

<body>
    <h1> My Hobbies </h1>
    <p id = "h1">Soccer</p>
    <p id = "h2">Chess</p>
    <p id = "h3">Kayaking</p>
</body>
</html>
```

App.js

```
var hobbie1 = document.getElementById("h1");
var hobbie2 = document.getElementById("h2");
var hobbie3 = document.getElementById("h3");
console.log(hobbie3);
console.log(hobbie2);
console.log(hobbie1);
```

Console output

```
    <p id="h3">Kayaking</p>                    app.js:4
    <p id="h2">Chess</p>                       app.js:5
    <p id="h1">Soccer</p>                      app.js:6
```

[Skill 20.03 Exercises 1]

Skill 20.04: Change the contents of an element

Skill 20.04 Concepts

We have learned how to access elements from an HTML page using JavaScript. Once you have accessed an element, you can change the element's properties and attributes - this can range from modifying the text inside an element to styling the border, background, or position of the element. Below will discuss how to change the text of an element.

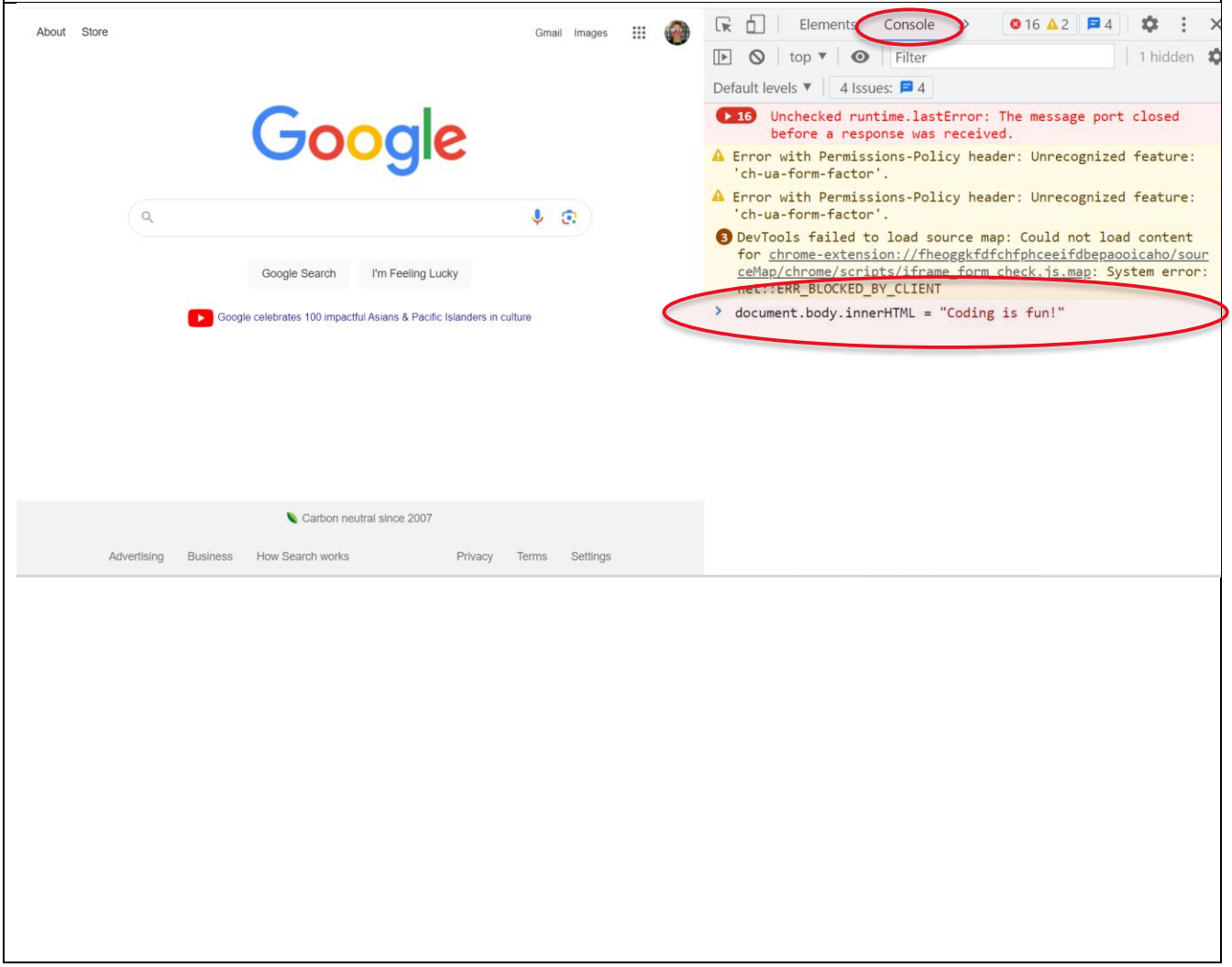
You can set or change the contents of an element with the *.innerHTML* property.

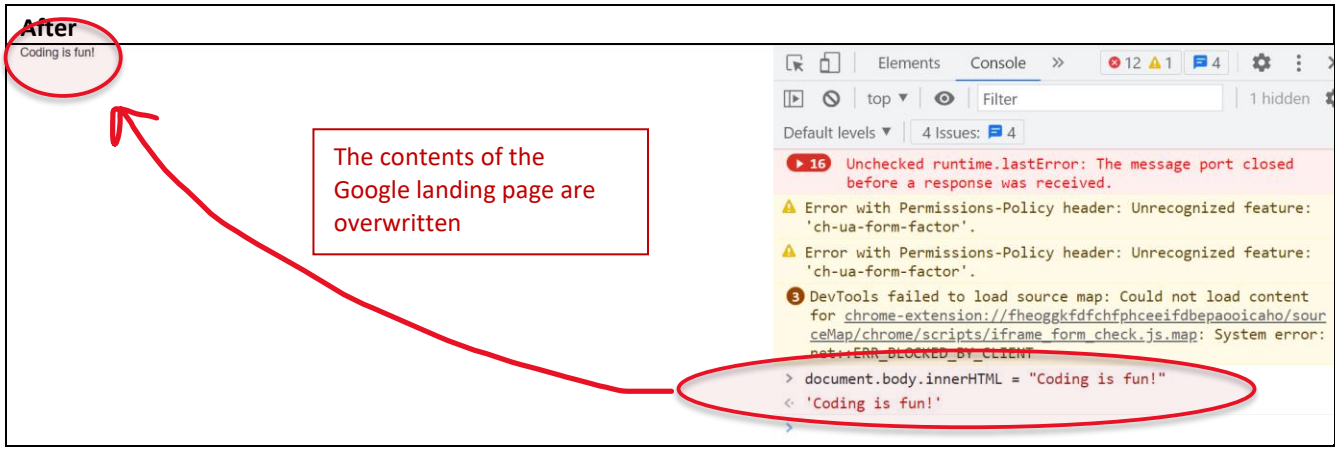
For example, the following code reassigns the inner HTML of the body element to the text 'Coding is fun!':

```
document.body.innerHTML = "Coding is fun!"
```

In fact, you can change the contents of any page on the internet. Check how we have overwritten the Google landing page,

Original page





The *.innerHTML* property can also add any valid HTML, including properly formatted elements. The following example assigns an *h2* inside the *<body>* element:

```
document.body.innerHTML = '<h2>This is a heading</h2>';
```

As another option, you can also change the contents of an element with a specific id,

```
document.getElementById('bio').innerHTML = 'The description';
```

Skill 20.04 Exercises 1 & 2