Set 18: Document Object Model Part 1

Skill 18.01: Explain the Document Object Model (DOM)
Skill 18.02: Interpret the parent-child relationship of the DOM
Skill 18.03: Use JavaScript to access DOM elements
Skill 18.04: Manipulate DOM elements

Skill 18.01: Explain the Document Object Model (DOM)

Skill 18.01 Concepts

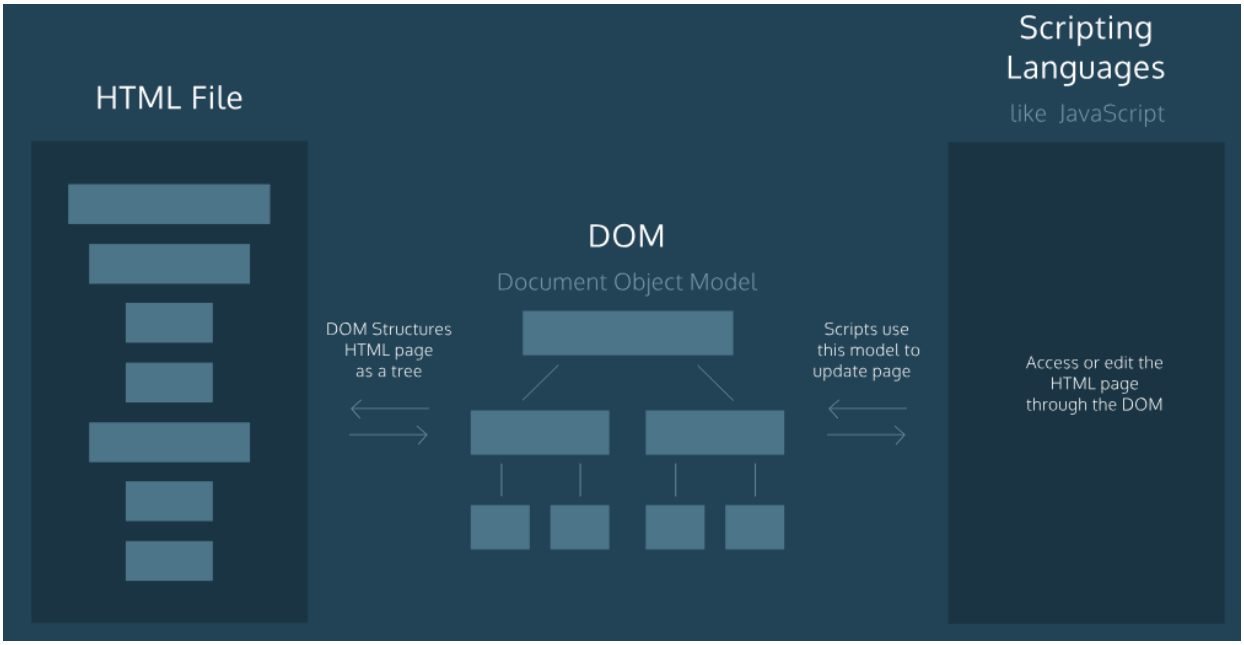
The Document Object Model, abbreviated DOM, is a powerful tree-like structure that allows programmers to conceptualize hierarchy and access the elements on a web page.

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.

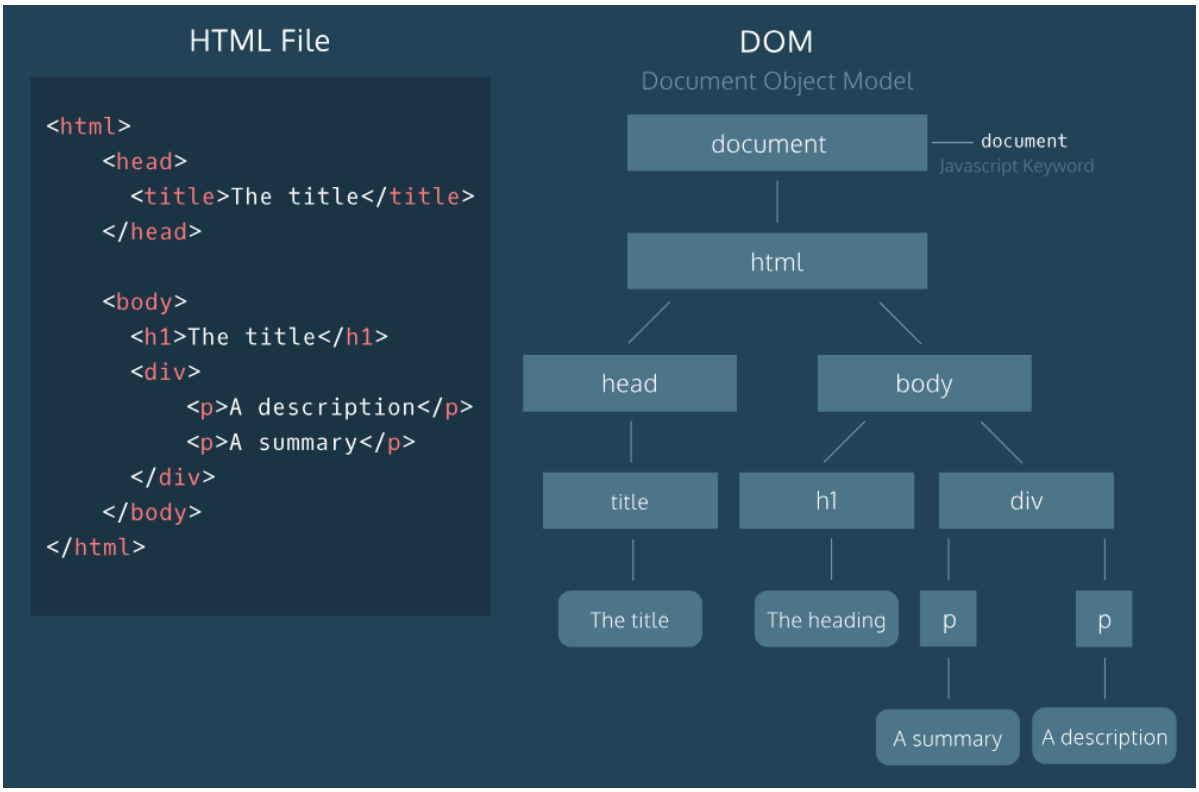


Tree-like modeling is used in many fields, including evolutionary science and data analytics. Perhaps you're already familiar with the concept of family trees: these charts represent the familial relationships amongst the descendants of a given family name.

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 18.01 Exercise 1

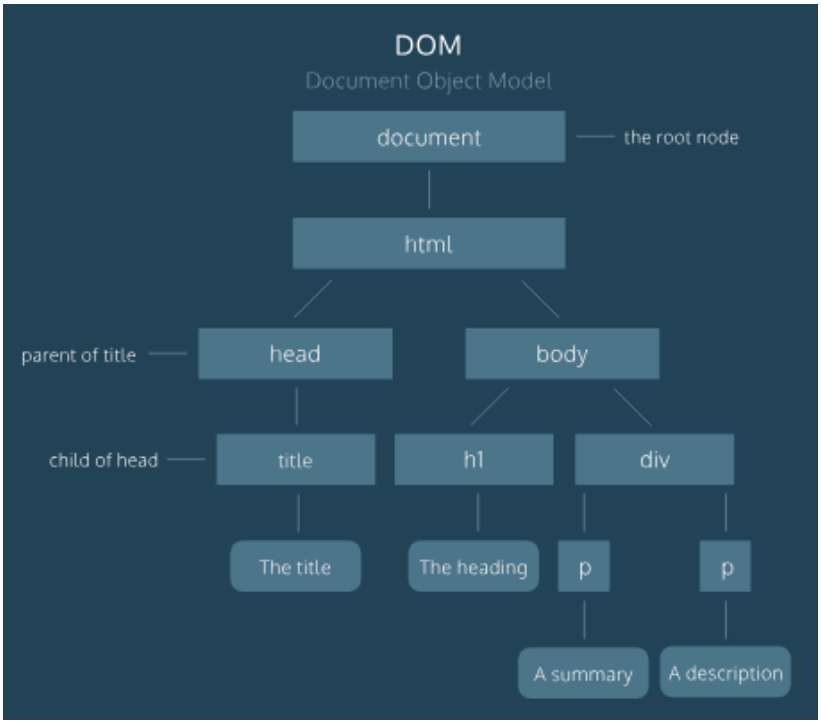
Skill 18.02: Interpret the parent-child relationship of the DOM

Skill 18.02 Concepts

Following the metaphor of a family tree, let's define some key terminology in the DOM hierarchy:

- A *parent node* is the closest connected node to another node in the direction towards the root.
- A *child node* is the closest connected node to another node in the direction away from the root.

Knowing these terms will allow you to understand and discuss the DOM as a tree-like structure. In fact, you will also see this terminology used when referring to the nesting structure of HTML code. Programmers refer to elements nested inside other elements as the children elements and parent elements respectively.



[Skill 18.02 Exercise 1]

Skill 18.03: Use JavaScript to access DOM elements

Skill 18.03 Concepts

The *document* object in JavaScript is the door to the DOM structure. The *document* object allows you to access the root node of the DOM tree. Before you can access a specific element in the page, first you must access the document structure itself.

The *document* object allows scripts to access children of the DOM as properties.

For example, if you wanted to access the <body> element of a page in your script, you could access it as a property of the document by typing the code below,

```
var b = document.body;
```

Similarly, you could access the <title> element with the .titleproperty.

```
var b = document.title;
```

As another option, if you want to access elements directly by their *id*, you can use the aptly named *.getElementByID()* function:

```
var d = document.getElementById('bio');
```

Many elements of the same class can also be selected using the *.getElementByClassName()* function.

```
var c = document.getElementByClassName('favorites');
```

See a comprehensive list (https://developer.mozilla.org/en-US/docs/Web/API/Document) of all document properties.

Skill 18.03 Exercises 1

Skill 18.04: Changing the contents of an element

Skill 18.04 Concepts

When using the DOM in your script to access an HTML element, you also have access to all of that element's properties. This includes the ability to modify the contents of the element as well as its attributes and properties— that can range from modifying the text inside a *p* element to assigning a new background color to a *div*.

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 'The cat loves the dog':

```
document.body.innerHTML = "The cat hates the dog";
```

The *.innerHTML* property can also add any valid HTML, including properly formatted elements. The following example assigns an *h2* element as a child 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';
```

It is also possible to change the content of many elements at the same time using the .getElementByClassName, but more on this later!

Skill 18.04 Exercises 1