# What is the DOM?

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.

Note: There are other types of documents, such as XML and SVG, that are also modeled as DOM structures.

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.

# The DOM as a Tree Structure

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.

# Parent Child Relationships in the DOM

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

A parent node is the closet 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.

# Nodes and Elements in the DOM

As mentioned, a node is the equivalent of each family member in a family tree. A node is an intersecting point in a tree that also contains data.

There are nine different types of node objects in the DOM tree. In our diagram, the node objects with the sharp-edge rectangles are of the type [Element](https://developer.mozilla.org/en-US/docs/Web/API/Element), while the rounded edge rectangles are of type [Text](https://developer.mozilla.org/en-US/docs/Web/API/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.

# Attributes of Element Node

DOM element nodes model elements in an HTML document.

Much like an element in an HTML page, the DOM allows us to access a node's attributes, such as its class, id, and inline style.

In the diagram to the right, we have highlighted the paragraph element with an id of "bio" in the HTML document. If we were accessing that element node in our script, the DOM would allow us to tweak each of those attributes, or simply access them to check their value in the code.

# Review

Congratulations on completing our introduction to the Document Object Model, or DOM, as a structure!

Let's review what you've learned so far:

* The DOM is a structural model of a web page that allows for scripting languages to access that page.
* The system of organization in the DOM mimics the nesting structure of an HTML document.
* Elements nested within another are referred to as the children of that element. The element they are nested within is called the parent element of those elements.
* The DOM also allows access to the regular attributes of an HTML element such as its style, id, etc.

With this understanding, you can begin to leverage the power of scripting languages to create, update, and maintain webpages!

# The document keyword

The Document Object Model, abbreviated DOM, is a powerful tree-like structure that organizes the elements on a web page and allows scripting languages to access them. This lesson will focus on some of the most useful methods and properties of the [DOM Interface](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model) in JavaScript. This interface is implemented by every modern browser.

First things first! The document object in JavaScript is the door to the DOM structure. The document 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 allows scripts to access children of the DOM as properties.

For example, if you wanted to access the <body> element in your script, you could access it as a property of the document by typing document.body. This property will return the body element of that DOM.

Similarly, you could access the <title> element with the .title property. See a [comprehensive list](https://developer.mozilla.org/en-US/docs/Web/API/Document) of all document properties.

**Tweak an Element**

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 access and set 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 loves 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>';

# Style an element

Another way to modify an element is by changing its CSS style. The .style property of a DOM element provides access to the inline style of that HTML tag.

The syntax follows an element.style.propertyformat, with the property representing a CSS property.

For example, the following code selects the first element with a class of blue and assigns blue as the background-color:

let blueElement = document.querySelector('.blue'); blueElement.style.backgroundColor = 'blue';

Unlike CSS, the DOM style property does not implement a hyphen such as background-color, but rather camel case notation backgroundColor.

The following chaining syntax would also work:

document.querySelector('.blue').style.fontFamily = 'Roboto';

# Create and Insert Elements

Just as the DOM allows scripts to modify existing elements, it also allows for the creation of new ones. The .createElement(tagName)method creates a new element based on the specified tag name. However, it does not append it to the document. It creates an empty element with no inner HTML.

In order to create an element and add it to the web page, you must assign it to be the child of an element that already exists on the DOM. We call this process appending. The .appendChild()method will add a child element as the last child node.

The following code creates a new paragraph element, adds text to the new element's innerHTML, and appends it to the body of the document:

let paragraph = document.createElement('p'); paragraph.innerHTML = 'The text inside paragraph'; document.body.appendChild(paragraph);

Unlike .innerHTML the .appendChild() method does not replace the content inside of the parent, in this case body. Rather, it appends the element as the last child of that parent.

# Remove an Element

In addition to modifying or creating an element from scratch, the DOM also allows for the removal of an element. The .removeChild()method removes a specified child from a parent.

Because the .querySelector() method returns the first paragraph, the following code would remove the first paragraph in the document:

let paragraph = document.querySelector('p'); document.body.removeChild(paragraph);

It's possible to also specify a different parent with the .querySelector() method, as long as you remove an element nested within that parent element.

If you want to hide an element because it does not need to be loaded initially, the .hiddenproperty allows you to hide it by assigning it as true or false:

document.getElementById('sign').hidden = true;

The code above did not remove the element from the DOM, but rather hid it. This is not the same as setting the CSS visibility property to hidden. For a list of the best use cases for this property, read a list in the [MDN documentation](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/hidden).

# Interactivity with onclick

You can add interactivity to DOM elements by assigning a function to run based on an [event](https://developer.mozilla.org/en-US/docs/Web/Events).

Events can include anything from a click to a user mousing over an element.

The .onclick property allows you to assign a function to run on a click event on an element:

let element = document.getElementById('interact'); element.onclick = function() { element.style.backgroundColor = 'blue' };

# Traversing the DOM

In the DOM hierarchy, parent and children relationships are defined in relation to the position of the root node.

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.

These relationships follow the nesting structure present in HTML code. Elements nested within one HTML tag are children of that parent element.

Each DOM element node has a .parentNode and .children property. The property will return a list of the element's children and return null if the element has no children.

The .firstChild property will grant access to the first child of that parent element.

# Review

In this lesson, you manipulated a webpage structure by leveraging the Document Object Model interface in JavaScript.

Let's review what we learned:

* The document keyword grants access to the root of the DOM in JavaScript
* The DOM Interface allows you to select a specific element with CSS selectors by using the .querySelector() method
* You can also access an element directly by its ID with .getElementById()
* The .innerHTML and .style properties allow you to modify an element by changing its contents or style respectively
* You can create, append, and remove elements by using the .createElement(),.appendChild() and .removeChild() methods respectively
* The .onclick property can add interactivity to a DOM element based on a click event