Exploration — The Document Object Model

Introduction

In this exploration, we will study the Document Object Model, from here on out referred to as the DOM. The DOM is the model that a browser uses to render a web page. As a first step to rendering a page, the browser parses the HTML document and creates a DOM tree from it. From there the browser renders the page by rendering each node of the DOM tree. The DOM gives a representation of a document as a logical tree of nodes and includes an API for interacting with this tree representation. This allows us to write programs to dynamically access and modify the content, structure, and style of an HTML document.

Introduction to trees

The topic of trees is typically covered in a class on data structures. However, if you have not yet taken data structures, in the following sections we have something of a crash course in the tree data structure.

Nodes

A tree is made up of **nodes**. Each node has exactly one **parent** (with one exception noted below). However, a node can have an unlimited number of **children**. If a node has no children it is called a **leaf**. Nodes that are children of the same parent are called **siblings**. In an HTML DOM tree, the order of the siblings is significant as it determines which gets rendered first (and thus usually higher) in the page.

Root node

Every tree has a single **root** node that does not have a parent node. If we start from any node and continue to visit parents of those nodes we end up at the root node. Thus the root node is the **ancestor** of every other node in the tree and all other nodes are the **descendants** of the root node.

Navigating a Tree

If we are at a node in a tree, there are three primary ways of navigating to other nodes in the tree.

1. Up the tree: We can move up to the parent node of the current node. Since the root node has no parent node, this navigation is not possible when we are at the root node.

2. Down the tree: The next common movement is moving to a child node. This is a little more complex as there can be several children of the current node. So we often need to get a list of children and find the node we want to move to in that list. If we are at a leaf node, there are no children and this movement is not possible.
3. Sideways in the tree: Finally we can visit sibling nodes. This is a more complex movements. If the nodes do not contain references to their siblings, we need to move up to the parent of the current node and then from there get the list of children which includes the original node and all of its siblings.

No Cycles

Another important property of trees is that a tree cannot have **cycles**. This means that:

1. If we are at a node `x` and we navigate up the tree to `x`'s parent, and then to `x`'s parent's parent, and so on, we will never arrive back to `x`. Our navigation will stop at the root node.
2. If we are at a node `x` and we navigate down the tree of any one `x`'s children, then keep navigating down the tree to any one of that node's children, and so on, we will never arrive back at `x`. Our navigation will stop at a leaf node.

DOM Spec & Levels

The DOM is a W3C (World Wide Web Consortium) standard. DOM provides APIs for manipulating not just HTML, but also other types of documents, such as CSS and XML (a markup language popular for data exchange) . Versions of DOM are referred to as levels. When we read the DOM specification on **the W3C website (https://dom.spec.whatwg.org/)**, we will find a description of **interfaces** rather than the description of the API in a particular programming language. This is because the DOM specification is programming language neutral. Many programming languages have implemented the DOM specification so that programs written in that language can use the DOM API. We will study and use JavaScript's implementation of the DOM API. The **MDN website (https://developer.mozilla.org/en-US/docs/Web/API)** provides documentation on the programming language neutral DOM specification, as well as **JavaScript's implementation of this API (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction#dom_and_javascript)**.

JavaScript and DOM

When JavaScript is running in the browser, two global objects `window` and `document` are available to the JavaScript code. The object `window` represents the browser window and the object `document` represents the webpage currently loaded in this browser window. The object `document` gives us an entry point into the DOM tree that the browser has built up after it parsed the webpage. `document` is also available as a property of `window`, i.e., `window.document`. The properties `document.head` and `document.body` respectively correspond to the `head` and `body` element of the document.

Nodes in a DOM Tree

A DOM tree contains nodes of various types. Objects corresponding to these nodes in JavaScript all inherit from the prototype `Node` and share the attributes and methods defined on `Node` [recall from Module 2, Exploration - Object-Oriented Programming that JavaScript supported only prototype-based inheritance until class-based inheritance was added to the language). These properties and methods allow us to navigate the DOM tree in various ways.

Example

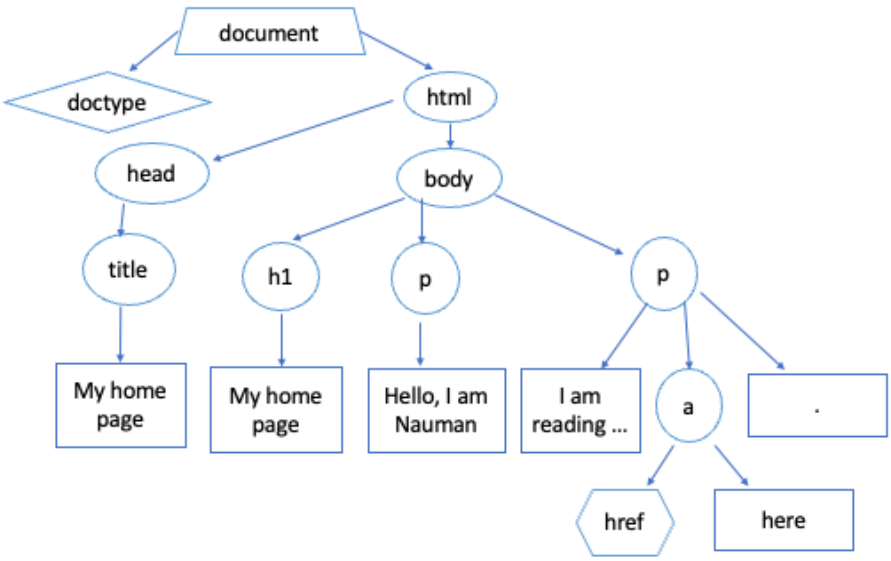
Here is a simple HTML document.

```
<!doctype html>
<html>

<head>
  <title>My home page</title>
</head>

<body>
  <h1>My home page</h1>
  <p>Hello, I am Nauman and this is my home page.</p>
  <p>I am reading a JavaScript book. You can find it
    <a href="https://www.amazon.com/Modern-JavaScript-Impatient-Cay-Horstmann/dp/0136502148">her
e</a>.
  </p>
</body>
</html>
```

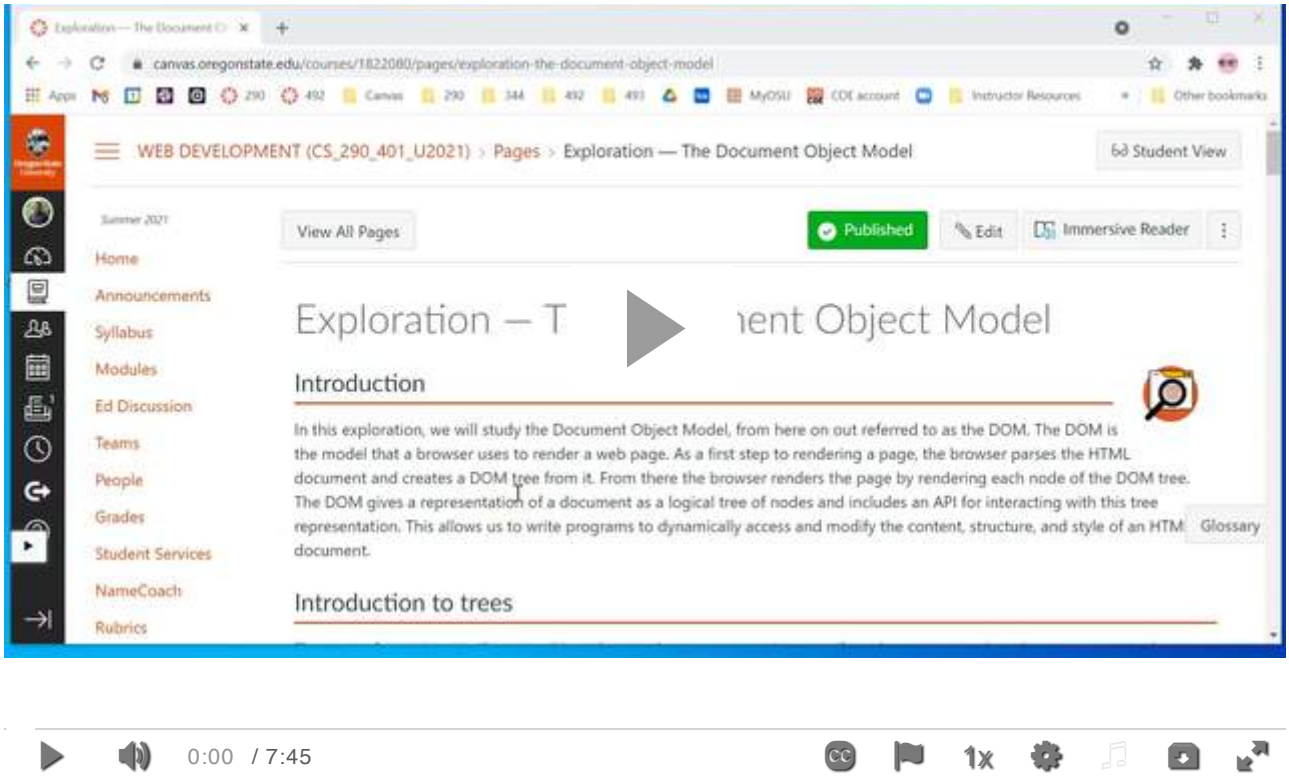
The DOM tree for this document is shown in the following figure:



This DOM tree includes nodes of the following type:

- Document: This node type is at the root of the DOM tree.
- Document Type: This node type corresponds to the `doctype` declaration in the HTML document and it is the first child of the Document node.
- Element: All elements in the HTML document are represented by Element nodes. These nodes are depicted as circles in the above diagram, with the name of the tag displayed inside the circle.
- Attribute: There is one attribute node in the tree which corresponds to the `href` attribute of the `a` element.
- Text: Text nodes correspond to the text content in the HTML document. These nodes are depicted with rectangles in the above figure.

Note: The DOM tree in the above figure does not include all nodes in the actual DOM tree. A complete DOM tree will also include additional text nodes, one for each whitespace or newline that occurs between the HTML elements. We have not shown these additional nodes because they do not impact how this HTML document is rendered.



Traversing the Elements in a DOM Tree

Any node that has one or more children has the following properties:

- `firstElementChild`: Move down the tree to the first child node of type element.
- `lastElementChild`: Just like `firstElementChild` but move to the last child node which is of type element.
- `nextElementSibling`: Move sideways to the next node which is both an element and a child of the same parent as the current node.

- `previousElementSibling`: Move sideways to the node which is both an element and a child of the same parent as the current node, but is the previous child of the parent.
- `children`: This contains all the children which are element nodes.

To move up the tree, we can use the property `parentNode`. For the attributes of an element node, DOM API provides the methods `getAttribute(attributeName)` and `setAttribute(attributeName, value)`, while the property `attributes` returns all the attributes of an element.

Note that the API contains additional methods and properties that allow traversing other node types, including the text nodes of the HTML document.

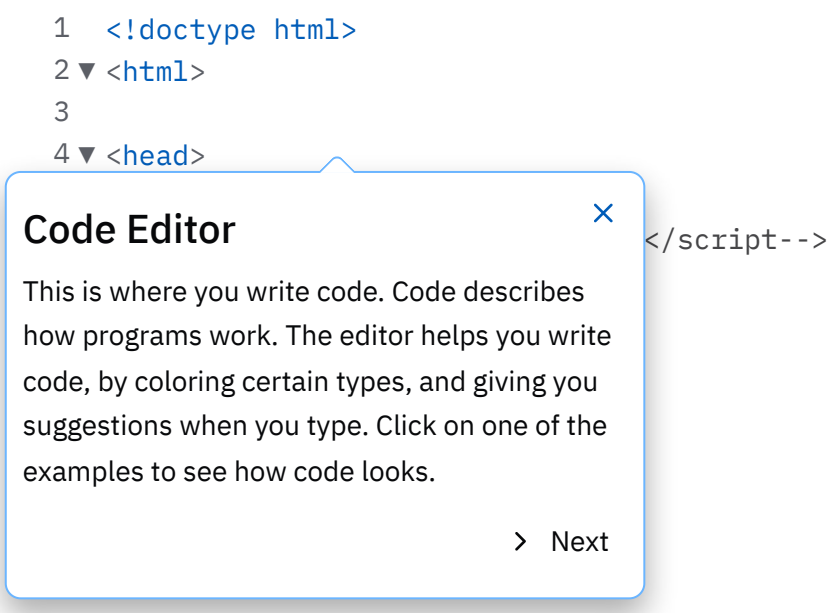
Example

In the following example, code in `rainbow.js` calls different methods to traverse the DOM tree and change the background color of various elements as follows:

- The body element becomes red,
- The `h1` heading becomes orange,
- The paragraph element “Hello, I am Nauman and this is my home page.” becomes yellow,
- The anchor element becomes green.

Note: The first line in our JavaScript code assigns a function to `window.onload`. This has been done so that this function executes after everything in the webpage has been loaded in the browser window. We will discuss this further in the **Exploration DOM Events (https://canvas.oregonstate.edu/courses/1879154/pages/exploration-dom-events)** **(https://canvas.oregonstate.edu/courses/1879154/pages/exploration-dom-events)** .

> **Note:** The `index.html` file in the following repl has the `rainbow.js` script commented out at line 6. Therefore, the changes to the background colors of each node are not updated. Open the code in Replit and fork it to edit the HTML file. Then uncomment line 6, run the edited code and you will see the color changes.



Searching for Elements

Many times it is more convenient to search for elements based on their class, their id, or their type, or using CSS selectors.

getElementById

The method **document.getElementById(elementId)** (https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById) takes a string for the id of the element we are looking for and returns this element.

Note that an Id should be unique in an HTML document, whereas a document can have multiple elements with the same tag name or the same class name. Because of this, the word "Element" is singular in the name of the method `getElementById`, while "Elements" is plural in the name of the methods `getElementsByTagName` and `getElementsByClassName` that we describe next.

getElementsByTagName

The method **getElementsByTagName(tagName)** (https://developer.mozilla.org/en-US/docs/Web/API/Element/getElementsByTagName) is defined on nodes of type `element` and returns a

collection of all descendant nodes with a particular tag name. For example, `document.body.getElementsByTagName('div')` will return a collection of all the `div` elements in the body of the document.

getElementsByClassName

The method **getElementsByClassName(className) (https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByClassName)** is available for nodes of type `element` and on the document node. It selects all descendant elements which have the provided class name. If we wanted to find everything in a page that has a `warning` class, we would call `document.body.getElementsByClassName('warning')`.

querySelector

The method **querySelctor(selectors) (https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector)** is defined on the `document` node. It returns the first element which matches the CSS selector or selectors provided as an argument to the method.

Example

In the following example, we use the API methods to search nodes based on their class, type and id values, and change the styles of these nodes. Specifically, code in the JavaScript file calls

- `document.getElementsByClassName('make-me-red')` to find elements in the DOM with class `make-me-red` and then for each of these elements, sets the value of the property `style.color` to red.
- `document.getElementsByTagName('button')` to find button elements in the DOM and then sets the border style of each button element.
- `document.getElementById('best')` to find the element with ID `best` and sets the value of its property `style.backgroundColor` to pink.

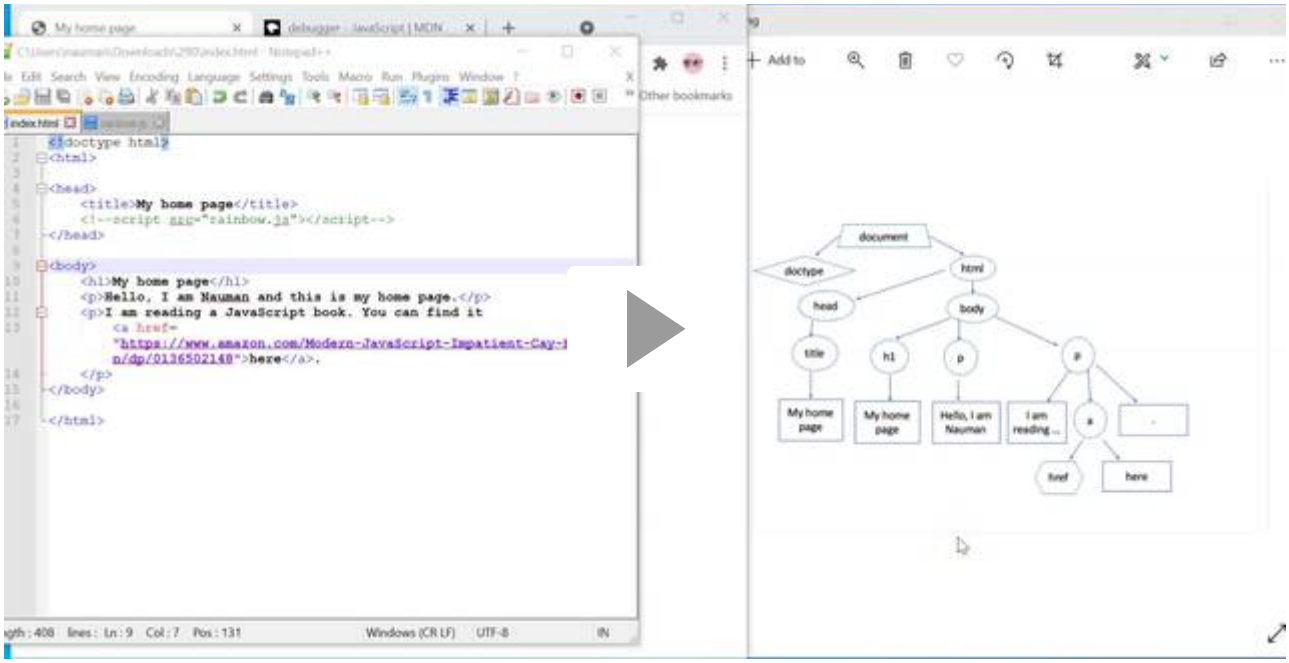
```
1  <!doctype html>
2 ▼<html>
3
4 ▼<head>
```

</script>

Code Editor

This is where you write code. Code describes how programs work. The editor helps you write code, by coloring certain types, and giving you suggestions when you type. Click on one of the examples to see how code looks.

Next



0:00 / 10:37

Summary

This exploration we looked at what is the DOM tree and how we can navigate and search this tree using the DOM API. However, modern web applications typically do not use this low level DOM API. Later in the course, we will study higher level frameworks that hide the details of the DOM API from us, and which are widely used to develop web applications. So be sure to have a good conceptual understanding of how the browser parses a webpage to create a DOM tree and how an API is available to navigate and search the DOM tree. However, remembering the names or details of the methods presented in this exploration is not particularly important for developing modern web applications.

Additional Resources

Here are some references to learn more about the topics we discussed in this exploration.

- DOM specification is available **at W3C** (https://www.w3.org/DOM/DOMTR) .
- **W3C reference** (https://www.w3schools.com/js/js_htmldom.asp) on JavaScript and DOM.
- **MDN reference** (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction) on JavaScript and DOM.
- Trees and DOM are discussed in **Chapter 14** (https://eloquentjavascript.net/14_dom.html) of the book *Eloquent JavaScript*.