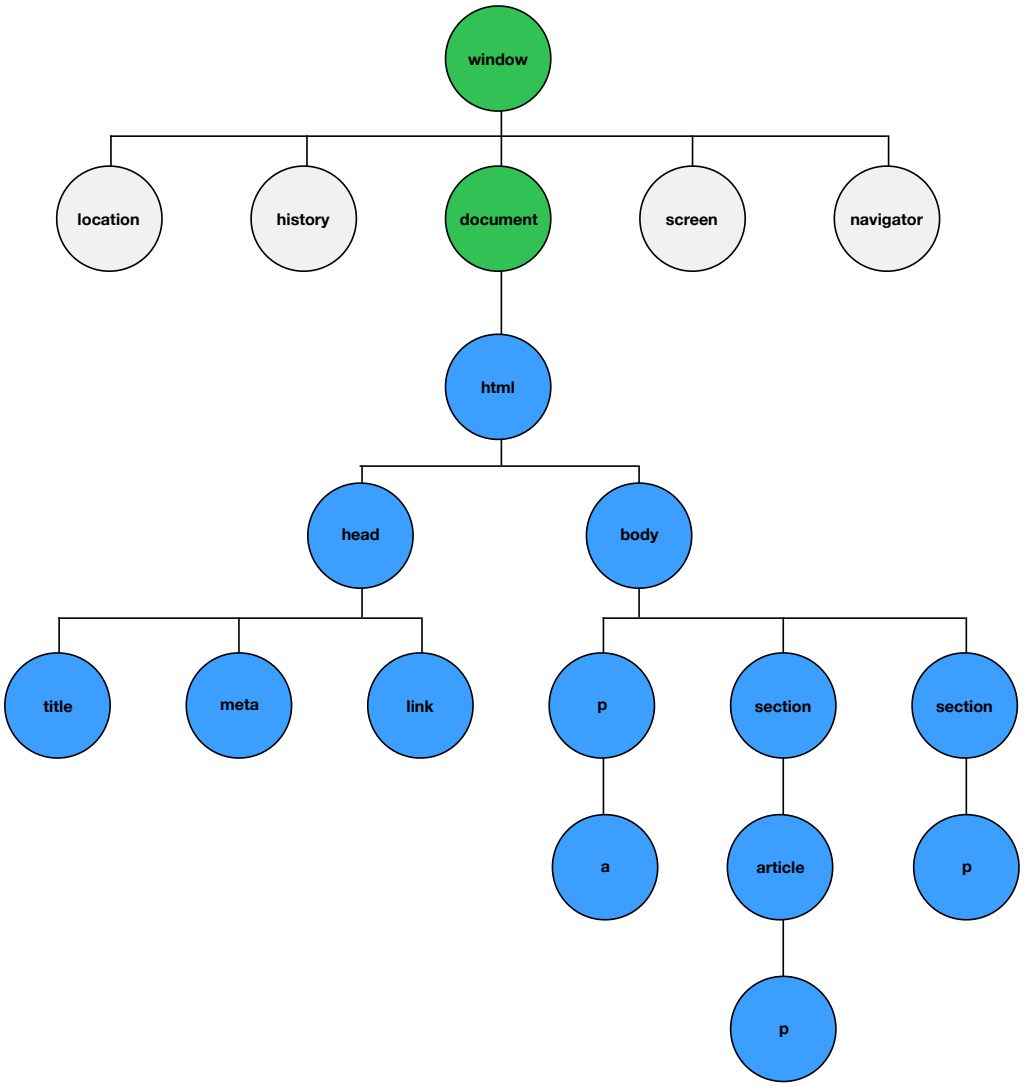
JavaScript:
The Document Object Model

When a browser receives an HTML source page, it parses the HTML into a tree structure called the *Document Object Model*, or *DOM*. This lesson covers two important components of client-side scripting: understanding how the DOM works and understanding how to manipulate it with JavaScript.

A typical DOM tree looks like the following. You'll notice that there is a **window** and a **document** object. Both are parents of the HTML tree and, in practice, you'll be working with the **document** object quite frequently. Each circle in the diagram below is called a *node* and each is an *object* having its own properties and methods.



window Object

The **window** object controls and provides access to high-level features and information (such as the current URL and the browser's history list), provides some ways to interact with users (via prompts and alerts, for example), and is the default owner of all things that seem otherwise "unattached" in the JavaScript space (like any procedural, user-written functions that are not methods on another object). Because window is the default owner of objects, JavaScript programmers often omit window when referencing things that belong there—they'll write **alert()**, for example, rather than **window.alert()**.

> *(Note:* window *is a pre-built object that already exists, so you don't need to create it or assign it to a variable.)*

Method	Description
alert()	Displays an alert box with a message and an OK button
confirm()	Displays a dialog box with a message and an OK and a Cancel button
console	The console object, to which we can log information.
prompt()	Displays a dialog box that prompts the visitor for input

document Object

The **document** object contains properties and methods that affect the page itself but that are at a higher, more abstract level than the HTML structure. The **document** object is responsible for finding nodes, creating new nodes, writing parts of the document, etc. Here are some of the commonly used **document** methods. (Note: the **document** object always exists as a variable; you don't need to create it or find it.)

> *(Note:* document *is a pre-built object that already exists, so you can always refer to the* document *object without having to create it or assign it to a variable. But you need to specify explicitly that the following methods live on the* **document** *object, otherwise JavaScript will try to find them on the default* **window** *object and you'll get an error.)*

Method	Description
document.createElement(*str* **)**	Creates an Element node of type *str*

Method	Description
document.getElementById(*id* **)**	Returns the element that has the ID attribute with the specified value (*note lowercase "d"!*)
document.getElementsByTagName(*str*)	Returns a list of all elements having that tag name.
document.write(*text*)	Writes HTML expressions or JavaScript code to a document
document.writeln(*text*)	Same as write(), but adds a newline character after each statement

Nodes

All the HTML elements are transformed into objects of type **Node**. Although there are twelve different kinds of nodes, here is a list of the three that you'll probably encounter most often. Notice that these are constants and you can access them in the *element***.nodeType** property. You can also use the *element***.nodeName** property to get a text string.

Node Types

nodeType	JavaScript Constant	nodeName
1	ELEMENT_NODE	*the HTML tag name*
2	ATTRIBUTE_NODE	*the HTML attribute name*
3	TEXT_NODE	#text

Node Objects

(Note: node objects—or elements—are either created from scratch by the document *object or are found and retrieved by asking the* document *object to search for them. Either way, you'll typically put the node into a variable of your own naming. Here, the word* element *below is italicized, signifying that you should replace it with your chosen variable name.)*

Method	Description
element.**addEventListener(** *evt, fn* **)**	Add function *fn* as a handler to the element's *evt* action. *Note: do not use the "on" prefix!*
element.**appendChild(** *node* **)**	Adds a new child node, to an element, as the last child node
element.attributes	Returns a NamedNodeMap of an element's attributes
element.childNodes	Returns a NodeList of child nodes for an element
element.className	Sets or returns the class attribute of an element
element.firstChild	Returns the first child of an element
element.getAttribute(*attr*)	Returns the specified attribute value of an element node
element.getElementsByTagName(*str*)	Returns a collection of all child elements with the specified tagname
element.id	Sets or returns the id of an element
element.innerHTML	Sets or returns the content of an element
element.**insertBefore(** *new, existing* **)**	Inserts a *new* child node before an *existing* child node
element.lastChild	Returns the last child of an element
element.nextSibling	Returns the next node at the same node tree level
element.nodeName	Returns the name of a node
element.nodeType	Returns the node type of an element
element.nodeValue	Sets or returns the value of an element
element.**parentNode**	Returns the parent node of an element
element.previousSibling	Returns the previous element at the same node tree level
element.removeAttribute(*attr*)	Removes a specified attribute from an element
element.**removeEventListener(** *evt, fn* **)**	Removes function *fn* as a handler for the specified *event*.
element.**removeChild(** *node* **)**	Removes a child *node* from an element

Method	Description
element.replaceChild(*new, old*)	Replaces an *old* child node in an element with a *new* child node
element.setAttribute(*attr, value*)	Sets or changes the specified *attribute*, to the specified *value*
element.style.*property* = *value*	Sets or returns the style *property* of an element to *value*. See <https://www.w3schools.com/jsref/dom_obj_style.asp> for properties.

A more complete list of element variables and methods is available online:

http://www.w3schools.com/jsref/dom_obj_all.asp

Collections

There are two types of collections that are sometimes returned by various methods: a **NodeList** object (which is an ordered list of nodes) and a **NamedNodeMap** object (which is an unordered list of nodes).
Collections can be slightly confusing in that different methods might return either an array, a NodeList or a NamedNodeMap. Check the online documentation to know for certain. Most of the time, interestingly, you can refer to an individual indexed location as if the list were simply a numeric array, but there's no guarantee you'll be able to iterate that array in a loop. Experiment in your code to find out for sure.

Learn more here:

NodeList: http://www.w3schools.com/dom/dom_nodelist.asp

NamedNodeMap: http://www.w3schools.com/dom/dom_namednodemap.asp

NodeList

Method	Description
item()	Returns the node at the specified index in a node list
length	Returns the number of nodes in a node list

NamedNodeMap

Method	Description
getNamedItem()	Returns the node with the specific name
length	Returns the number of nodes in a node list
removeNamedItem()	Removes the node with the specific name
setNamedItem()	Sets the specified node (by name)