## Chapter 9: The Window Object

### The Browser Object Model (BOM):

The BOM is a collection of properties and methods that contain information about the browser and computer screen, like finding out the dimensions of the screen it is viewed on, and which pages have been visited before the current page.

All properties and methods are made available through the *window* object; every *window*, *tab*, *popup*, *frame*, and *iframe* has a *window* object.

The BOM only makes sense in the browser environment, unlike in Node.js; Node.js, however, has an object called *global* that is similar to the *window* object.

### Going Global:

*Note: global variables are created without using the* ***const****,* ***let****, or* ***var*** *keywords, and can be accessed throughout all parts of a program.*

Global variables are actual properties of a global object, i.e., they are properties of the *window* object.

### Halts in the Code:

*Window.alert()* - pauses execution of code; returns *undefined*.

*Window.confirm()* - Stops the execution of a program, showing the message provided as an argument. Returns the Boolean of *true* for a click of OK, and *false* for Cancel.

*Window.prompt()* - stops execution; like *confirm()*, but asks for an input from the user to a provided field. Returns the string entered when OK is clicked, otherwise, *null* for Cancel.

### Browser Information:

*Window.navigator* - returns a reference to the *Navigator* object, which contains information about the browser being used. The *userAgent* property will return information about the browser and operating system being used. **This information can be changed (it is a *read-write* property), so it is not reliable!**

*Window.location* - property object that contains information about the URL of the current page. Can get different parts of the URL:

*href* - returns the full URL as a string. Is a read/write property  
 *protocol* - returns a string describing the protocol (*http, https, pop2, ftp, etc)* used. There is a colon at the end {:}.  
 *host* - returns a string describing the domain of the URL *and* the port number (omitted if default port of 80) is used).  
 *hostname* - returns a string describing the domain of the current URL **only**.  
 *port* - returns a string describing the port number, empty if not explicitly stated in the URL.  
 *pathname* - returns a string of the path that follows after the domain.  
 *search* - returns a string starting with a ? followed by the query string parameters; empty string if there are no string parameters.  
 *hash* - returns a string that starts with a # followed by the fragment identifier, empty if there is none.  
 *origin* - returns a string that shows the protocol and domain where the current page originated from. **This property is *read-only*, so it cannot be changed.**  
  
The *window.location* property also has the *reload()*, *assign()*, *replace()*, and *toString()* methods, which do the following, respectively:

* Can force a reload of the current webpage. If *true*, forces the browser to reload the page from the server, instead of the cached page.
* Used to load another resource from a URL provided as a parameter.
* Similar to the *assign()* method, but the current page will not be stored in the session history, therefore the user will be unable to navigate back using the back button.
* Returns a string containing the whole URL.

### Cookies:

To create a cookie, assign it to the JavaScript cookie jar, using the *document.cookie* property.

This acts like a special type of string, where assigning another cookie to it will not overwrite the entire property, but append it to the end of the string.  
{document.cookie = {{ hero=true }}; document.cookie = {{ city=Metropolis }};}

The cookie value can be changed.  
{document.cookie = {{ city=Gotham }};}

To see the current contents of the cookie jar, simply enter *document.cookie*.

You can use the *split()* method to separate the cookie string into an array containing each *name/value* pair, then use a *for-of* loop to iterate through the array:

*Const cookies = document.cookie.split({{ ; }});  
 for (crumb of cookies) {  
 const [key, value] = crumb.split({{ = }});  
 console.log(`The value of ${key} is ${value}.`);  
 }*

Cookies do expire, though: they are only session cookies by default; when a user closes the browser window/tab, they are deleted. They can be made persistent by adding *{{ ; expires=date }}*, where *date* is the date value in the UTC String format *{Day, DD-Mon-YYYY HH:MM:SS GMT}*. ***(Note: 86,400,000 = 1 day later.)***

## Chapter 14: HTML5 APIs

These have already been reviewed; I will mainly be looking at Multimedia and Notification APIs.