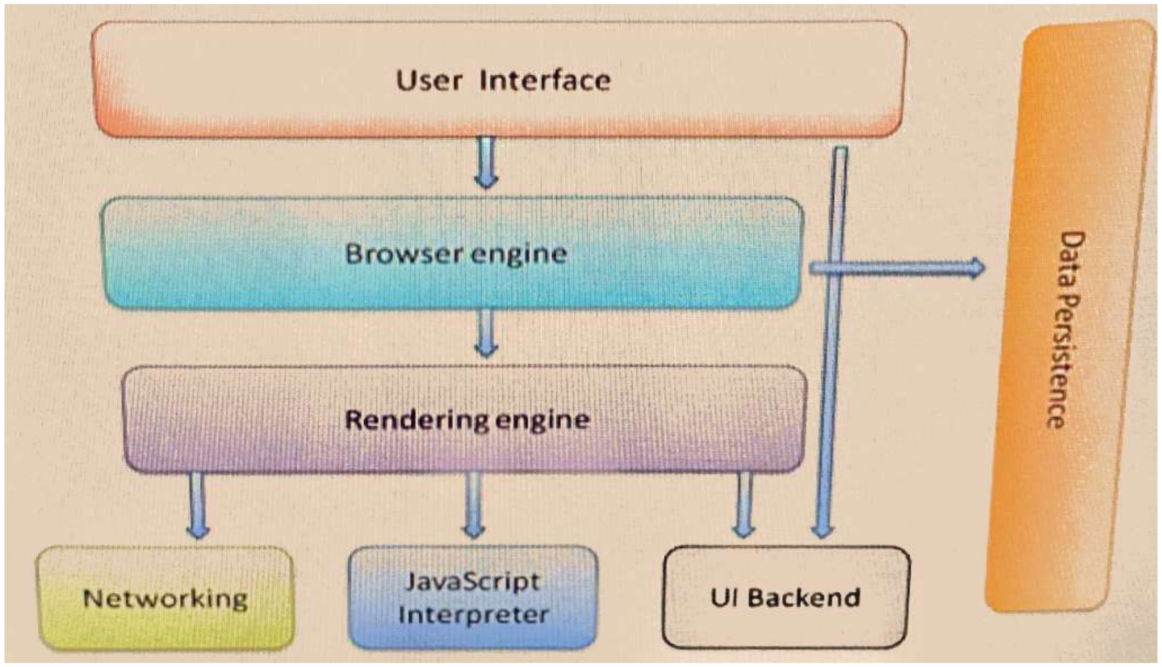
a/A Week 4 day1

Notebook:	Week4		
Created:	9/7/2020 1:50 AM	**Updated:**	9/7/2020 3:13 AM
Author:	gunerb1@tcnj.edu		
URL:	https://open.appacademy.io/learn/js-py---aug-2020-online/week-3-aug-2020-online/running-js-scripts-in-the-browser		

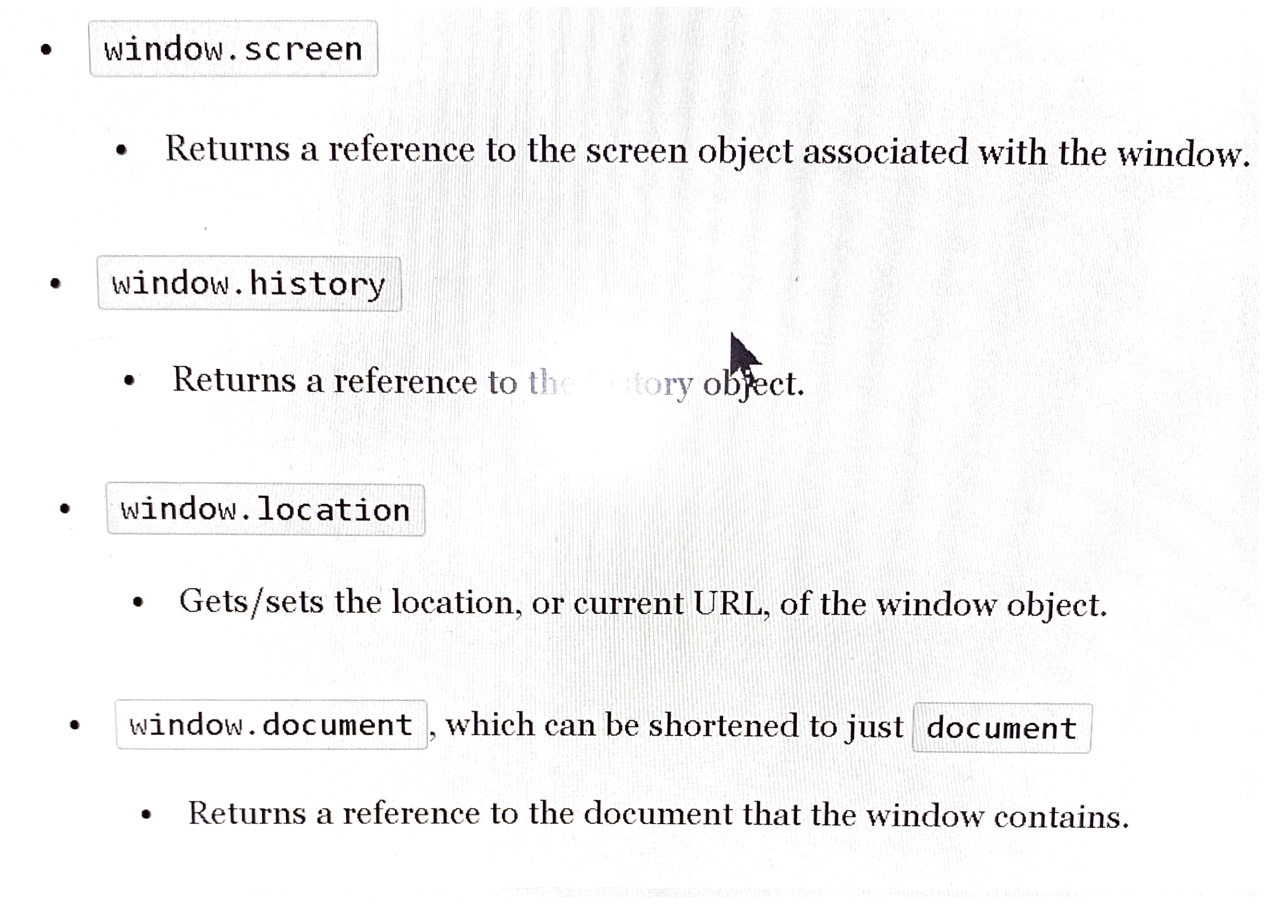
The BOM Contains the DOM



The DOM contains a collection of notes which are HTML elements.

The document object is a webpage and the dog represents the object hierarchy of that document.

The window object contains properties and methods we could use to access different objects within the window.

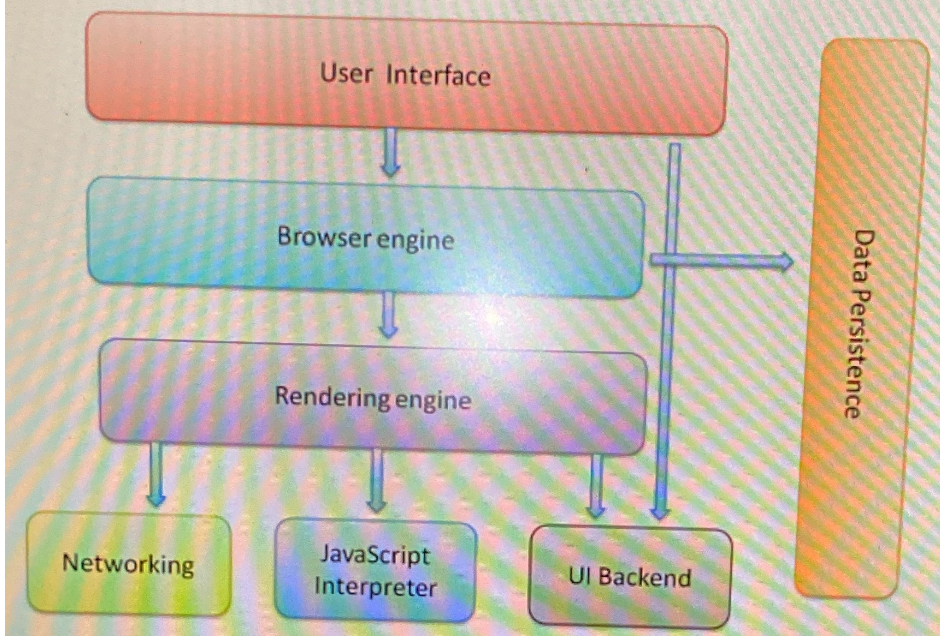


- `window.screen`
  - Returns a reference to the screen object associated with the window.
- `window.history`
  - Returns a reference to the history object.
- `window.location`
  - Gets/sets the location, or current URL, of the window object.
- `window.document`, which can be shortened to just `document`
  - Returns a reference to the document that the window contains.

window.document can be shortened to document.

The Browser Diagram :

depicting a high-level structure of the browser, from html5rocks.com:



- **User interface**: This is the browser interface

The user interface Includes the address bar back and forward buttons bookmark menu

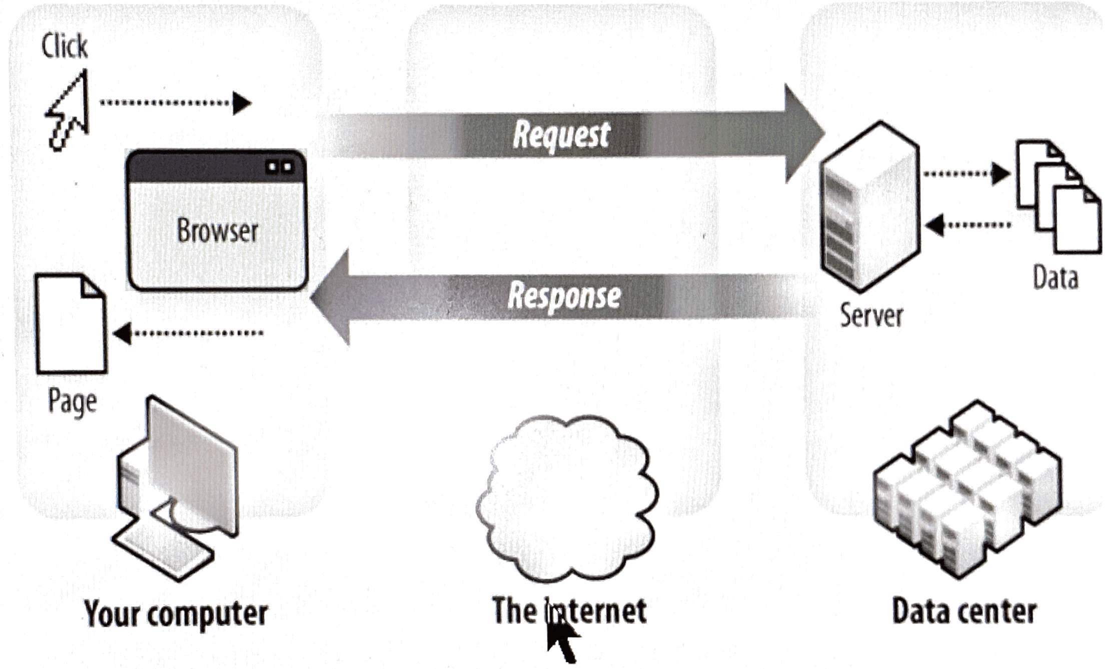
The browser engine manages it directions between the user interface and the rendering engine

The request Dash response cycle is a communication pattern between the client otherwise known as the browser and a server.

Whenever we tape a URL into the address bar of a browser we are making a request to the server for information… a common form of this is an HTTP request.

The request-response cycle diagram

Let's take a look at this diagram of the request-response cycle from O'Reily:



On the left is the **client** side, or the browser. On the right is the **server** side with a database where data is stored. The internet, in the middle, is a series of these client requests and server responses. We'll be going into more dept with servers soon, but for right now we are focusing on the client side.

Window Properties and Methods: https://developer.mozilla.org/en-US/docs/Web/API/Window

Context, scope, and anonymous functions

Two important terms to understand when you're developing in Javascript are **context** and **scope**. Ryan Morr has a great write-up about the differences between the two here: "Understanding Scope and Context in Javascript".

The important things to note about **context** are:

1. **Every function has a context (as well as a scope).**
2. **Context refers to the object that *owns* the function (i.e. the value of *this* inside a given function).**
3. **Context is most often determined by how a function is invoked.**

Take, for example, the following code:

```
const foo = {
bar: function() {
return this;
}};console.log(foo.bar() === foo);// returns true
```

The anonymous function above is a method of the `foo` object, which means that `this` returns the object itself — the context, in this case.

What about functions that are unbound to an object, or not scoped inside of another function? Try running this anonymous function, and see what happens.

```
(function() {
console.log(this);})();
```

When you open your console in the browser and run this code, you should see the `window` object printed. When a function is called in the global scope, `this` defaults to the global context, or in the case of running code in the browser, the `window` object.

DOMContentLoaded ensures that the JavaScript script will run when the document has been loaded without waiting for CSS stylesheets images in subframes to load however if we wanted to wait for everything in the document to load we could instead use the window object method called window.onload

Another way of ensuring that the script will not run until the document has been loaded is to insert your script tag right before the closing of the body tag in the HTML file itself.

If you want to include your script in the head tags rather than the body tags you could use the async or deferred method in the script tag.

Using async, A script is fetched asynchronously after the script is fetched HTML parsing is paused to execute the script and then is resumed.

Using the deferred method a script is fetched asynchronously and is executed only after the HTML parsing has been completed.

It is common practice to use both methods simultaneously as new or browsers generally recognized async and older browsers commonly only recognize the deferred method.