Front End Web

JavaScript
Review and Discussion

Three Layers of Web Pages

- Structure // HTML
- Style // CSS
- Interactive // JavaScript

What is JavaScript?

JavaScript is a cross-platform, object-oriented scripting language. It is a small and lightweight language. Inside a host environment (for example, a web browser), JavaScript can be connected to the objects of its environment to provide programmatic control over them.

Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

JavaScript is not Java

Advantages of JavaScript

- JavaScript is very easy to implement. All you need to do is put your code in the HTML document and tell the browser that it is JavaScript.
- JavaScript works on web users' computers — even when they are offline!
- JavaScript allows you to create highly responsive interfaces that improve the user experience and provide dynamic functionality, without having to wait for the server to react and show another page.
- JavaScript can load content into the document if and when the user needs it, without reloading the entire page — this is commonly referred to as Ajax.
- JavaScript can test for what is possible in your browser and react accordingly — this is called Principles of unobtrusive JavaScript or sometimes defensive Scripting.
- JavaScript can help fix browser problems or patch holes in browser support — for example fixing CSS layout issues in certain browsers.

Source: https://www.w3.org/community/webed/wiki/What_can_you_do_with_JavaScript

Uses of JavaScript

- A sign-up form can check if your user name is available when you enter it, preventing you from having to endure a frustrating reload of the page.
- A search box can give you suggested results while you type, based on what you've entered so far (for example "bi" could bring up suggestions to choose from that contain this string, such as "bird", "big" and "bicycle"). This usage pattern is called autocomplete.
- Information that changes constantly can be loaded periodically without the need for user interaction, for example sports match results or stock market tickers.
- Information that is a nice-to-have and runs the risk of being redundant to some users can be loaded when and if the user chooses to access it. For example the navigation menu of a site could be 6 links but display links to deeper pages on-demand when the user activates a menu item.

Uses of JavaScript

- JavaScript can fix layout issues. Using JavaScript you can find the position and area of any element on the page, and the dimensions of the browser window. Using this information you can prevent overlapping elements and other such issues. Say for example you have a menu with several levels; by checking that there is space for the sub-menu to appear before showing it you can prevent scroll-bars or overlapping menu items.
- JavaScript can enhance the interfaces HTML gives us. While it is nice to have a text input box you might want to have a combo box allowing you to choose from a list of preset values or enter your own. Using JavaScript you can enhance a normal input box to do that.
- You can use JavaScript to animate elements on a page — for example to show and hide information, or highlight specific sections of a page — this can make for a more usable, richer user experience. There is more information on this in the JavaScript animation article later on in the course.

Source: https://www.w3.org/community/webed/wiki/What_can_you_do_with_JavaScript

Be aware

JavaScript might not be available — this is easy to test for so not really a problem. Things that depend on JavaScript should be created with this in mind however, and you should be careful that your site does not break (ie essential functionality is not available) if JavaScript is not available.

If the use of JavaScript does not aid the user in reaching a goal more quickly and efficiently you are probably using it wrong.

Source: https://www.w3.org/community/webed/wiki/What_can_you_do_with_JavaScript

Be aware

Using JavaScript we often break conventions that people have got used to over the years of using the web (for example clicking links to go to other pages, or a little basket icon meaning “shopping cart”).

We like being in control and once we understand something, it is hard for us to deal with change. Your JavaScript solutions should feel naturally better than the previous interaction, but not so different that the user cannot relate to it via their previous experience. If you manage to get a site visitor saying “ah ha — this means I don’t have to wait” or “Cool — now I don’t have to take this extra annoying step”— you have got yourself a great use for JavaScript.

JavaScript should never be a security measure. If you need to prevent users from accessing data or you are likely to handle sensitive data then don’t rely on JavaScript. Any JavaScript protection can easily be reverse engineered and overcome, as all the code is available to read on the client machine. Also, users can just turn JavaScript off in their browsers.

Source: https://www.w3.org/community/webed/wiki/What_can_you_do_with_JavaScript

Security

Buggy implementations used to crash the browser or execute code by using heap spraying attacks or similar. These can be kind of mitigated within modern browsers by using ASLR, DEP, sandboxes and similar techniques.

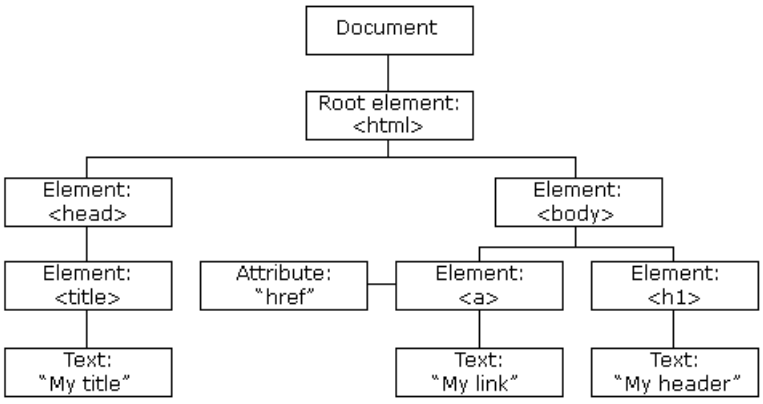
Cross Site Scripting (XSS) attacks can be used to steal authorization credentials (i.e. passwords) or hijack existing sessions. This way they can be used for identity theft or misuse. They work because the same browser is used for different web sites (i.e. banking and looking at cats) and this way it tears down security borders between these sites. Same Origin Policy is only of limited use in these cases.

A typical case of such attacks is the inclusion of third party scripts into a web site. This can be external Javascript libraries, tracking code like google analytics, buttons for social networks or advertisement. This included code is out of the control of the original web site but still has full access to the site inside the browser and can thus read information from it, fill in forms or read their content, navigate the user to different sites, press like buttons, inject advertisements or malware etc.

Apart from that Javascript can be used to fingerprint the browser in a more detailed way which might be a problem especially when trying to be anonymous.

Source: https://security.stackexchange.com/questions/110850/what-are-the-exact-security-risks-of-having-javascript-enabled

The D.O.M.



JavaScript and the D.O.M.

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

Source: https://www.w3schools.com/js/js_htmldom.asp

JavaScript accesses the D.O.M.

```
<html>
<body>

<p id="demo"></p>

<script>
  document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

The B.O.M.

The Browser Object Model (BOM)

- There are no official standards for the **B**rowser **O**bject **M**odel (BOM).
- Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

The B.O.M.

- Window - browser's window
- Screen
  - screen.width
  - screen.height
  - screen.availWidth
  - screen.availHeight
  - screen.colorDepth
  - screen.pixelDepth
- Location
  - window.location.href returns the href (URL) of the current page
  - window.location.hostname returns the domain name of the web host
  - window.location.pathname returns the path and filename of the current page
  - window.location.protocol returns the web protocol used (http: or https:)
  - window.location.assign loads a new document

The B.O.M.

- History
  - history.back() - same as clicking back in the browser
  - history.forward() - same as clicking forward in the browser
- Navigator
  - navigator.appName
  - navigator.appCodeName
  - navigator.platform
- Pop-up Alert
  - Alert box
  - Confirm box
  - Prompt box
- Timing
  - setTimeout(*function, milliseconds*) Executes a function, after waiting a specified number of milliseconds.
  - setInterval(*function, milliseconds*) Same as setTimeout(), but repeats the execution of the function continuously.
- Cookies

JavaScript and APIs

Browser APIs

The DOM (Document Object Model) API allows you to manipulate HTML and CSS, creating, removing and changing HTML, dynamically applying new styles to your page, etc.

The Geolocation API retrieves geographical information. This is how Google Maps is able to find your location, and plot it on a map.

The Canvas and WebGL APIs allow you to create animated 2D and 3D graphics.

Audio and Video APIs like HTMLMediaElement and WebRTC allow you to do really interesting things with multimedia, such as play audio and video right in a web page, or grab video from your web camera and display it on someone else's computer.

Source: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

JavaScript and APIs

3rd party APIs

- The Twitter API allows you to do things like displaying your latest tweets on your website.
- The Google Maps API allows you to embed custom maps into your website, and other such functionality.

Code Example

Source: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash

JavaScript Coding Basics

There are three kinds of declarations in JavaScript:

- var Declares a variable, optionally initializing it to a value.
- let Declares a block-scoped, local variable, optionally initializing it to a value.
- const Declares a block-scoped, read-only named constant.

What is a variable?

In a programming language, **variables** are used to **store** data values.

JavaScript uses the **var** keyword to **declare** variables.

An **equal sign** is used to **assign values** to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

- var x;
  x = 6;
- var x = 6;

Source: https://www.w3schools.com/js/js_syntax.asp

Declare Variables

Creating a variable in JavaScript is called "declaring" a variable.

var x (undefined)

var x = 52

Variables

You use variables as symbolic names for values in your application. The names of variables, called identifiers, conform to certain rules.

- A JavaScript identifier must start with a letter, underscore (_), or dollar sign ($); subsequent characters can also be digits (0-9). Because JavaScript is case sensitive, letters include the characters "A" through "Z" (uppercase) and the characters "a" through "z" (lowercase).
- You can use most of ISO 8859-1 or Unicode letters such as å and ü in identifiers. You can also use the Unicode escape sequences as characters in identifiers.
- Some examples of legal names are Number_hits, temp99, $credit, and _name.

Variable Scope

When you declare a variable outside of any function, it is called a *global* variable, because it is available to any other code in the current document.

When you declare a variable within a function, it is called a *local* variable, because it is available only within that function.

Variable hoisting

Another unusual thing about variables in JavaScript is that you can refer to a variable declared later, without getting an exception.

This concept is known as **hoisting**; variables in JavaScript are in a sense "hoisted" or lifted to the top of the function or statement.

However, variables that are hoisted will return a value of undefined. So even if you declare and initialize after you use or refer to this variable, it will still return undefined.

Because of hoisting, all var statements in a function should be placed as near to the top of the function as possible. This best practice increases the clarity of the code.

```
/**
 * Example 1
 */
console.log(x === undefined); // true
var x = 3;

/**
 * Example 2
 */
// will return a value of undefined
var myvar = 'my value';

(function() {
  console.log(myvar); // undefined
  var myvar = 'local value';
})();
```

Global variables

- Global variables are in fact properties of the global object. In web pages the global object is window, so you can set and access global variables using the window.variable syntax.
- Consequently, you can access global variables declared in one window or frame from another window or frame by specifying the window or frame name. For example, if a variable called phoneNumber is declared in a document, you can refer to this variable from an iframe as parent.phoneNumber.

Constants

You can create a read-only, named constant with the const keyword. The syntax of a constant identifier is the same as for a variable identifier: it must start with a letter, underscore or dollar sign ($) and can contain alphabetic, numeric, or underscore characters.

```
1 const PI = 3.14;
```

Data Types

Six data types that are primitives: Boolean. true and false.

- null. A special keyword denoting a null value. Because JavaScript is case-sensitive, null is not the same as Null, NULL, or any other variant.
- undefined. A top-level property whose value is undefined.
- Number. 42 or 3.14159.
- String. "Howdy"
- Symbol (new in ECMAScript 2015). A data type whose instances are unique and immutable.

You don’t have to declare the data type. It is done automatically with JavaScript.

Working with JavaScript

```
'37' - 7 // 30
'37' + 7 // "377"
```

```
x = 'The answer is ' + 42 // "The answer is 42"
y = 42 + ' is the answer' // "42 is the answer"
```

Working with JavaScript

Objects and functions are the other fundamental elements in the language.

You can think of objects as named containers for values, and functions as procedures that your application can perform.

Literal Types: Arrays

An array literal is a list of zero or more expressions, each of which represents an array element, enclosed in square brackets ([]). When you create an array using an array literal, it is initialized with the specified values as its elements, and its length is set to the number of arguments specified.

```
1 | var coffees = ['French Roast', 'Colombian', 'Kona'];
```

Count arrays starting with 0

```
1 | var coffees = ['French Roast', 'Colombian', 'Kona'];
```

coffees[0] = French Roast
coffees[1] = Colombian
coffees[2] = Kona

Other Literal Types

Boolean : True || False

Integers : Numbers

Floating-point : Numbers with decimals and/or (+/-)

RegExp : A regexp literal is a pattern enclosed between slashes.
var re = /ab+c/;

String : Characters enclosed in single or double quotes
'foo'

Object : A list of 0 or more pairs of names and values
myObj = { "name":"John", "age":30, "car":null };

Objects

Objects are like variables with many data points

var car = "Fiat";

var car = {type:"Fiat", model:"500", color:"white"};

Try it: https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_properties_1

Escaping Characters

var quote = "He read \"The Cremation of Sam McGee\" by R.W. Service.";

OUTPUT of quote: He read "The Cremation of Sam McGee" by R.W. Service.

Escape line breaks

```
var str = 'this string \
 is broken \
 across multiple \
 lines.' console.log(str); // this string is broken across multiplelines.
```

Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_Types#Array_literals

JSON

JSON is Object Data

- JSON: **J**ava**S**cript **O**bject **N**otation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.

```
var person = { "name":"John", "age":31, "city":"New York" };
```

NOTE: JSON names require double quotes. JavaScript names don't.

Source: https://www.w3schools.com/js/js_json_intro.asp

JSON Data

```
{
  "name":"John",
  "age":30,
  "cars":[  "Ford", "BMW", "Fiat"  ]
}
```

CSS - Data-* attributes

The data-* attributes is used to store custom data private to the page or application.

- The data-* attributes gives us the ability to embed custom data attributes on all HTML elements.
- The stored (custom) data can then be used in the page's JavaScript to create a more engaging user experience (without any Ajax calls or server-side database queries).
- The data-* attributes consist of two parts:
- The attribute name should not contain any uppercase letters, and must be at least one character long after the prefix "data-"
- The attribute value can be any string

Note: Custom attributes prefixed with "data-" will be completely ignored by the user agent.

Try It: https://www.w3schools.com/tags/att_data-.asp

Code Example

Resources:
https://www.w3schools.com/js/tryit.asp?filename=tryjson_parse

https://developers.facebook.com/tools/explorer/145634995501895/?method=GET&path=me%3Ffields%3Did%2Cname&version=v2.10

Events

- Event handlers can be used to handle, and verify, user input, user actions, and browser actions:
  - Things that should be done every time a page loads
  - Things that should be done when the page is closed
  - Action that should be performed when a user clicks a button
  - Content that should be verified when a user inputs data
  - And more ...
- Many different methods can be used to let JavaScript work with events:
  - HTML event attributes can execute JavaScript code directly
  - HTML event attributes can call JavaScript functions
  - You can assign your own event handler functions to HTML elements
  - You can prevent events from being sent or being handled
  - And more ...

Try It: https://www.w3schools.com/js/js_events.asp

Conditional Statements

Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions.
- You can use conditional statements in your code to do this.
- In JavaScript we have the following conditional statements:
- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed

Try It: https://www.w3schools.com/js/js_if_else.asp

Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Instead of this:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

----------vv or better this vv----------

```
for (i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
}
```

Error Handling

- Errors will happen.
- Important to deliver error-free code

Try It: https://www.w3schools.com/js/js_errors.asp

Debugging Tools

Console

http://www.jslint.com/

Performance

Reduce Activity in Loops

Statements or assignments that can be placed outside the loop will make the loop run faster.

```
var i;
for (i = 0; i < arr.length; i++) {
```

// -- vv better vv -- //

```
var i;
var l = arr.length;
for (i = 0; i < l; i++) {
```

Reduce DOM Access

If you expect to access a DOM element several times, access it once, and use it as a local variable

Performance

Reduce DOM Size

Avoid Unnecessary Variables

Delay JavaScript Loading

Putting your scripts at the bottom of the page body lets the browser load the page first.

The HTTP specification defines that browsers should not download more than two components in parallel.

An alternative is to use **defer="true"** in the script tag. The defer attribute specifies that the script should be executed after the page has finished parsing, but it only works for external scripts.

JavaScript Style

Make JavaScript more readable by:

- Use white space. JavaScript ignores white space.
- Use line breaks. JavaScript ignores line breaks.
- Block code with line breaks and tabs.

```
function myFunction() {
    document.getElementById("demo1").innerHTML = "Hello Dolly!";
    document.getElementById("demo2").innerHTML = "How are you?";
}
```

- Camel Case

Camel Case

JavaScript programmers tend to use camel case that starts with a lowercase letter:

- firstName, lastName, masterCard, interCity.

NOTE: Hyphens are not allowed in JavaScript. It is reserved for subtractions.

Style Guides to Consider

https://google.github.io/styleguide/jsguide.html

https://contribute.jquery.org/style-guide/js/

https://github.com/felixge/node-style-guide/

https://make.wordpress.org/core/handbook/best-practices/coding-standards/javascript/

https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Coding_Style/

Why conform to one format (or style)?

Why have coding guidelines? Why not let each developer use their own style and patterns at will? In our experience, there are a few good reasons:

1. reading code becomes easier. You can actually discern patterns visually without having to read each token. This greatly speeds up reading each others' code
2. It speeds up writing code. If you have simple rules to follow when writing code, you don't have to make judgements about how it looks. You have more brainpower left to think about the important things (such as if you have an off-by-1 error..., potential NPEs, semantic bugs, ...)
3. they should encapsulate shared experience of many developers, and hence help one developer not make the same error another did in the past

Source: https://github.com/facebook/jcommon/wiki/Coding-Standards

W3C JS Quiz

https://www.w3schools.com/js/js_quiz.asp