CS275 Web and Mobile App Development

JavaScript™

Material created by:
David Augenblick, Bill Mongan, Dan Ziegler, Samantha Bewley, and Matt Burlick

What is JavaScript?

- The scripting language for web pages
- Enables interactivity with web pages with capabilities beyond HTML
  - Respond to user
  - Create interactive tasks
  - Make decisions
- Runs within a browser
  - Script statements interpreted as they come up

What can JavaScript do?

- What can JavaScript do?
  - Perform calculations
  - Display text + messages
  - Automate images
  - Process data generated by a server
  - Modify a web page without having to re-load it (Ajax)
  - And much, much more

Introduction to JavaScript

- Two methods to incorporate into HTML
  1. Add directly inside an HTML document

```
<script type=”text/javascript”>
    //Place JavaScript content here
</script>
```

  2. Store JavaScript code in a separate file

```
<script type=”text/Javascript” src=”filename.js”></script>
```

     - Assumes that the JavaScript text is in the same directory as the .html file

Introduction to JavaScript

- In order to use JavaScript we will review the following:
  - Variables and data types
  - Arithmetic operations
  - String objects
  - Numeric and string arrays
  - Objects
  - DOM basics
  - Functions
  - Conditionals and loops
  - Event handling

JavaScript variables

- Like with most languages, variable names have syntax conventions.
- For JavaScript:
  - Upper + lower case letters OK
  - Numbers 0-9 OK
  - (_) - underscores OK
  - No spaces
  - 1st character must be a letter or underscore
  - Names are case sensitive
  - No size limits

JavaScript variables

- Should declare a variable before using:
  - `var justDeclareVar;`
- NOTE:
  - Omitting the `var` results in potentially using a more global version of the variable.
  - It is not necessary to define a type for a variable
  - HOWEVER, typically don't use `var` in the parameter list (in fact doing so might cause issues ☹)

JavaScript math operations

- Arithmetic operators

  + - / * and %

- The + operator concatenates 2 strings
- Incrementors and decrementors and compound operators

  ++ -- +=

- Comparison operators:

  == < > <= !=

- Logical operators:

  && ||

Controlling flow in JavaScript – conditionals and loops

If Statements

```
if(expression1) {
     //if expression 1 is true
}
else if (expression2)  {
       // if expression 2 is true
}
else
     alert ('error message');
```

Switch Statement

```
switch(myMonth) {
  case 'Jan' :
          //********
          break;
    case 'feb' :
    case 'Feb' :
          //********
          break;
    default :
          //********
}
```

Controlling flow in JavaScript – conditionals and loops

- Loops (Supports, `for, while, do..while`)

```
for (var start=0; start<end; start++)
{
    //***********
}
while(expression) {
    //**********
}
do  {
     // ***********
} while (expression)
```

JavaScript Data Types

- First, as already noted, no need to explicitly declare a type for a variable
- However there still are different data types
  - By default, JavaScript just figures it own based on what's assigned.
- 3 basic data types:
  - Numbers – int or float
  - Boolean – true or false
  - Strings (technically these are objects)
- OK to change a variable's type while in execution

JavaScript Data Types

- Converting between types
  - String to number conversion

```
var someInt = parseInt(someString)
```

  - String to float conversion

```
var someFloat = parseFloat(someString)
```

- Note: If JavaScript can't convert, then it will return NaN (not a number)
  - Which you can test with the function `isNan`

JavaScript String objects

- Two ways to create a String object:

```
1. var presString = "Barack Obama";
2. presString = new String("Barack Obama");
```

- Note that `String` is a class; thus, you can use "new" to create (construct) String objects
- Recall that the + (addition) operator will concatenate String objects

```
var firstName = "Barack";
var lastName = "Obama";
var fullName = firstName + " " + lastName;
```

JavaScript String objects

Here's some common String class methods and properties:

- `PresString.length` ==> length of the string (12 characters in this example)
- Case conversion ==> `toUpperCase(), toLowerCase()`
- Substrings ==> p`resString.substring(2,8)` = >"rack O"
  - Note: zero based subscription. Start index to end (exclusive) index.
- `presString.charAt(4)` ==> "c"
- `PresString.indexOf("Oba")` ==> 7 (starting index of substring "Oba")

JavaScript Numeric Arrays

- Array declaration:

```
codes = new Array(9);
```

- Array assignment:

```
codes[0] = 101;
codes[1] = 202;
    ……..
codes[8] = 909;
```

- Or : `codes = [101, 202, ……., 909];`
- Use `length` method to find the number of elements in the Array

`codes.length` ==> 9

- OK to change length dynamically

```
codes = new Array();
codes[0] = 101;
```
etc.

JavaScript String Arrays

- Array declaration:

```
months = new Array(12);
```

- Array assignment:

```
months[0] = "Jan";
months[1] = "Feb";
……..
months[11] = "Dec";
```

- Or ==> `months = ["Jan", "Feb", ……, "Dec"];`

JavaScript String Arrays

- Splitting a string on a specific character

```
myName = "David Harris Augenblick";
mySplitName = myName.split(" ");  ==>
    mySplitName[0] = "David"
    mySplitName[1] = "Harris"
    mySplitName[2] = "Augenblick"
```

- Join to reconstruct full name

```
myFullName = mySplitName.join(" ");
```

JavaScript Functions

- JavaScript functions serve similar purposes to functions from other languages
  - Re-use code – group together statements to define a specific activity
  - Modular programming structure
- Function definition syntax:

```
function fnName()  {
      //JavaScript statements go here
}
```

JavaScript Functions

- Like most other languages we can pass information into function via parameters/arguments:

```
function printInt(intToPrint){
        //coming soon…

}
```

- Place the function definition in the `<head>` section of the document to guarantee that it will be defined prior to being called
- To call the function ==> `printInt(7);`
- To return a value use the `return` keyword:

```
function doubleIt(x)  {
        var doubledResult = 2 * x;
        return doubledResult;
}
```

JavaScript Functions – Basic Example

- Return the larger of 2 integers

```
<html>
<head>
    <script type="text/javascript">
        function Larger(int1, int2) {
            if (int1 > int2)
                return int1;
            else
                return int2;
        }
    </script>
</head>
<body>
    <p>Function call result:</p>
    <script type="text/javascript">
        var largerVal = Larger(5, 10);
    </script>
</body>
</html>
```

JavaScript Objects

- JavaScript supports the use of objects, i.e. combination of data (properties/attributes) and methods (functions)
- We have already seen some type of objects
  - Strings
  - Arrays
- Here's some more built-in ones!

Some built-in JS objects

- The Math class have several useful properties/methods
  - `Math.PI`
  - `Math.abs(num)`
  - `Math.ceil(num)`
  - `Math.floor(num)`
  - `Math.round(num)`
  - `Math.pow(n1, n2)`
  - `Math.random()  // returns number between 0 and 1`
  - `Math.sin(angle)`
  - and many more

Some built-in JS objects

- Date object
  - Create ==>

```
moonLand = new Date(7, 20, 1969);
```

  - Can set and retrieve Date values
    - `getDate()` - retrieves day
    - `getMonth()` - retrieves month
    - `getFullYear()` - retrieves year
  - Can also create and retrieve dates containing time properties (hours, minutes, seconds)

Some built-in JS objects

- Date object – example (get today's date):

```
<script language=javascript>
    var date = new Date(); // defaults to today's date
    var m = date.getMonth() + 1;
    var d = date.getDate();  // day
    var y = date.getFullYear();
    var mydate = m + ”/” + d + “/” + y;
</script>
```

JavaScript Objects

- We can also make our own JS object types.
- JavaScript provides a lot of different ways to make objects
- To use a way that is similar to other OOP languages, we'll use the newer ECMAScript 6 way!
- This requires JavaScript *strict mode*
  - You can read more about it yourself, but basically JavaScript strict mode causes the JS engine to require that the provided JavaScript follows more strict standards
  - This allows us to use more advanced language syntax like the ES6 class structure.
- To enable strict mode, just add `'use strict'` to the top of your JS file.

JavaScript Objects

- So the basic ES6 class structure is:

```
class ClassName{
       constructor(a,b){
           this.a = a;
           this.b = b;
    }
       print(msg){
           alert('msg'+this.a);
    }
}
```

- Note a few subtitles:
  - No need for the *function* keyword before the class's method
  - The constructor method's name is actually *constructor*
- Then we can instantiate and use a class just like in many other OOP language

```
var x = new MyClass(22,7);
x.print('Good luck ');
```

JSON

- Another (simpler) way to make objects is to use the **JavaScript Object Notation** (JSON)
- JSON objects are often used to passed data between programs, functions, etc..
- It's really just an array of `<key>:<value>` pairs.
- To get the value of a key just do:

  `<varname>.<key>`

  - I.e `jsonobj.access_token`

```
var jsonobj = {
        "access_token": “ABC123DEF456GHI789"
}
```

JSON

- Here's a more extensive JSON object.
- Note the combination of the usage of JSON and arrays

```
{
 "response": {
        "recent": [
                {
                        "id": "abc123",
                        "createdAt": 1372597625,
                        "type": "checkin",
                        "timeZoneOffset": -240,
                        "user": {
                                "id": "12345",
                                "firstName": "Joe",
                                "lastName": "Smith",
                         },
                        "venue": {
                                "id": "def456",
                                "name": "Drexel University",
                                "location": {
                                        "address": "3141 Chestnut Street",
                                         "lat": 40,
                                        "lng": -75,
                                        "postalCode": "19104",
                                        "city": "Philadelphia",
                                        "state": "PA",
                                        "country": "United States",
                                 }
                         }
                }
        ]
  }
}
```

Examples of JSON Objects and Arrays

Object:

```
{
        “key”: value
}


{
        “name”: “Dan”,
        “year”: “Junior”,
        “age”: “21”,
        “major”: “Computer Science”,
        “isPassing”: true
}
```

Object with an Array

```
{
        “name”: “Dan”,
        “year”: “Junior”,
        “age”: 21,
        “major”: “Computer Science”,
        “classes”: [
                “CS 338”,
                “CS 451”,
                “SE 311”,
                “MATH 411”
        ]
}
```

JSON

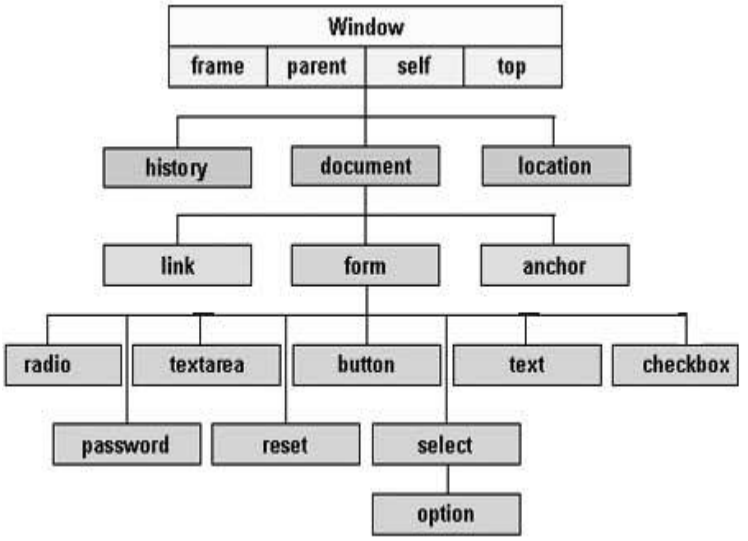
- We can also check to see if an attribute exists by comparing it to the JavaScript `undefined` object.

```
jsonobj.test==undefined
```

```
var jsonobj = {
        "access_token": “ABC123DEF456GHI789"
}
```

Diagram of the DOM Hierarchy

- Another common use of JavaScript is to manipulate the content of a webpage.
- As mentioned in the introduction, browsers organize HTML content into a hierarchical document object model (DOM)

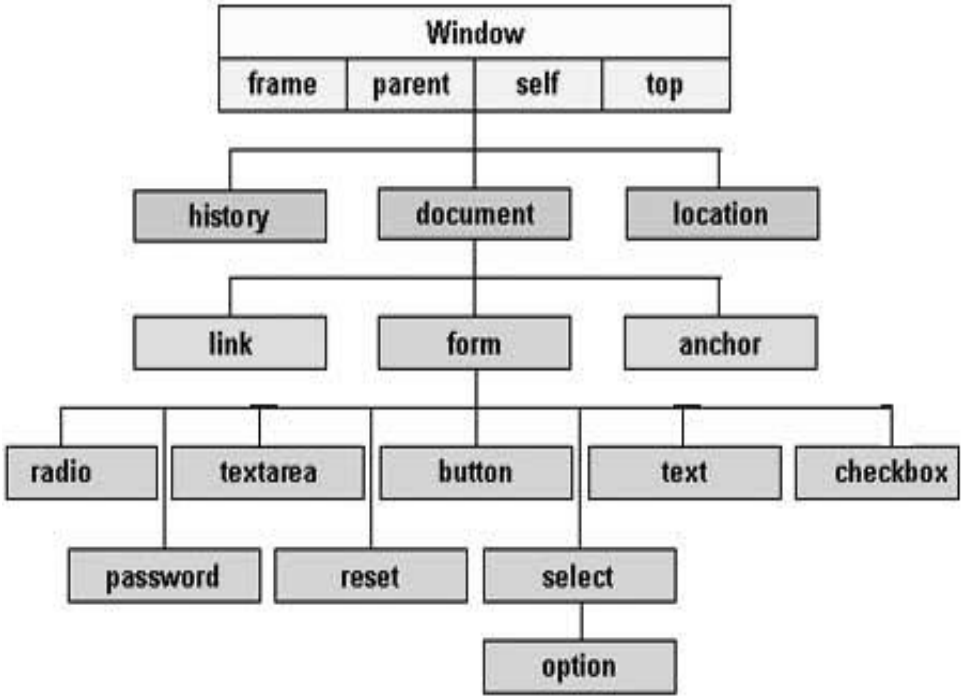


DOM

- DOM = structural framework of a web page
- Organized in a tree structure
- A DOM is created by the browser every time a (new) page loads
  - Browser breaks it's information into objects
- Renderer takes the DOM and traverses it to displays the webpage in a browser
- We can also use JavaScript to access/manipulate the DOM

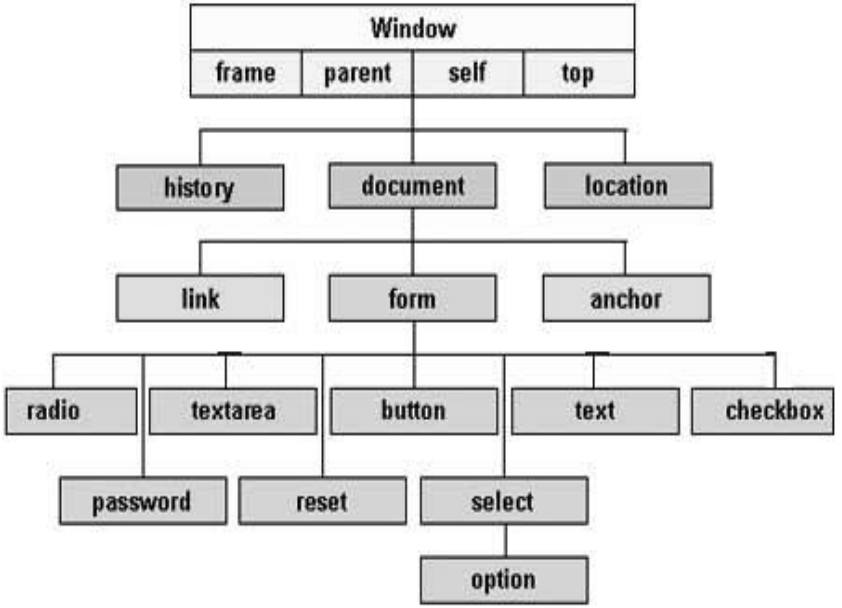
DOM

- The root of the DOM is the `window` object
- The `document` object is a child of the `window` object
  - `window.document` or `document` refers to the document in the current window



DOM

- Some important document properties include:
  - `document.URL` – the document's URL
  - `document.title` – the current page's title
  - `document.reference` – the URL that took you to the current page
  - `document.lastmodified` – date the document was last modified
  - `document.images` – the set of images within the current document
  - `document.bgcolor = 'FF0000'` – set the background color of the window.
- There are also various methods
  - `document.write(string)` – Write the content of the string at the part of the document. If the object to be written is not a string, the browser will first convert it to a string (if possible)



A simple DOM / document object example

- Lets display the current document's URL

```
<html>
<body>
      <h1>Displaying this document's URL</h1>
      <p>This document's URL is:
      <script type=”text/javascript”>
            document.write(document.URL);
      </script>
       </p>
 <body>
</html>
```

DOM

- We can also get/set attributes of DOM objects:

```
document.images[0].src
document.images[0].width
```

- And/or set their values:

```
document.images[0].src='./philliesLogo2.png'
```

A simple DOM / HTML example

- In addition we can locate and manipulate parts of the DOM by getting HTML elements from their *id*
- Let's assume we have a `div` element in our HTML

```
<div id="myElement" name="test"></div>
```

- JavaScript can locate elements of the DOM via their id

```
var newElement=document.getElementById("myElement");
```

- Or by name:

```
document.getElementsByName("test");
```

A simple DOM / HTML example

- We can now manipulate that member of the DOM using JavaScript.
- For instance the `.innerHTML` property of an HTML element can be changed.

```
<script>
    var newElement=document.getElementById("myElement");
    newElement.innerHTML="Add this Content to element" ;
</script>
```

- We could also *get* the innerHTML content:

```
var str = newElement.innerHTML;
```

- If we have an element that stores a value, like a text box, then we use `.value`:

```
var str = newElement.value;
```

Modal Windows

- For crude I/O and debugging purposes we can use JavaScript's ability to pop open messaging windows.
- These come in two common flavors:
  - Alert – Just displays a message and a button:

```
alert('Warning');
```

  - Input – Displays a message and a text field and returns the content of the text field up clicking on the button.

```
var into = input('type a number please');
```

Event handling in JavaScript

- An event usually triggers some action (event handling) on the web page when it takes pace
- Some common events include:
  - Click on something on the web page (button, image, etc.)
  - Moving the mouse over or off something on the web page (eg. Image)
  - Pressing a key (eg. Enter)
  - Loading a new web page

Event handling in JavaScript – the basics (continued)

- Elements have attributes that allow you to specify a JavaScript function to run when the event occurs.
- Attributes include:
  - `onmouseover`
  - `onmouseout`
  - `onclick`
  - `onkeyup`

Event handling in JavaScript – the basics (continued)

A simple example – changing an image after the mouse passes over it

```
<head>
<script language='javascript'>
     function warn()  {
        alert("you are now over my spouse's photo");
     }
     function change(el)   {
        el.src=' ./petsPhoto.png';
     }
</script>
</head>
<body>
    <img src='. /spousePhoto.png'  onmouseover = 'warn()'
          onmouseout = 'change(this)' />
</body>
```

Event Handler Mechanics

- In the previous examples we assigned event handlers using the attributes of an object.
- We can also bind them to an object itself:
- Example:

```
var myButton =
  document.getElementById("somebuttonName");
myButton.onclick = someFunction;
```

- If several elements have the ID "`somebuttonname`", then `someFunction` will be called if any of them are clicked.
- We can even bind a callback to the document itself!

```
function mouseClickAlert()  {
   alert("The mouse has just been clicked");
}
document.onmousedown = mouseClickAlert();
```

Event Handler Mechanics

- You can also create event handlers to deal with key board events:
- The handler "`onkeypress`" can detect when a key from the keyboard has been pressed (down + up)
- "`onkeydown`" and "`onkeyup`" can detect when either of the 2 phases of a key press has taken place
- You can also determine which specific key has been pressed
- As an example, you might want to utilize key press features to determine if the "enter" key has been pressed within a textbox object. If so, you could launch a validation process to determine if the data inputted is valid (within a certain range, etc.)

```
function response(event){
    var char = getChar(event || window.event);
    if(!char) return;  //special key
    return char;
}
document.onkeypress(response);
```