Computer Programming – CS 6011
Lecture 11: JavaScript

Fall 2023

- Topics
  - JavaScript
  - Document Object Model (DOM)

Web page

- HTML: Describes the structure of a web page
- CSS: Describes the styling of a web page

Web page

- HTML: Describes the structure of a web page
- CSS: Describes the styling of a web page
- JavaScript (JS): Makes the webpage interactive

Introduction to JavaScript

- While not actually “Java” – the syntax / usage is similar
- Designed to allow programmers to make web pages interactive
- JavaScript:
  - Scripting language
    - No need for a compiler
    - Rendering is handled by the browser
  - Uses mostly objects (in the same way Java does)
  - Garbage collected
  - Recently added “Classes” – but for now we are going to use just ‘basic’ JS

Dynamic Typing

- Unlike C++/Java, JavaScript variables don't have explicit types!
- Note, values DO have types.
  - "123" is a string
  - 123 is an int
  - true is a Boolean, etc.
- Declaring variables:
  - `let x;` // x is now a variable – note we did not declare a type for it.
  - `let y = 10;`
    - `y = "hello";` // Legal – but DON'T do it!!! Why?
  - `var x;` // Old version – better not to use it.

Don’t use Var, use Let

Some differences

- `var` has different scoping then `let`
  - `Var` has function scope
  - `Let` has block scope
- Hoisting

```
x = 3

var x;  // Legal to ‘declare x’ after assigning it (if you use var).

// Can not do that with let
```

Data types in JavaScript

- let varName;
- x = 3; // Number
- xStr = “hello”; // String
- xArray = [4, 7.2, “hello”]; // Array
  - Notice something different about this type of array?
- empObj = {}; // x is an empty object
- stdObj = { “name”: “John”, “gpa”: 3.7, “year”: 3 }; // Create a “complete” object

JavaScript Objects

- Objects in JavaScript are very similar to Maps:
  - Map: A list of `Key - Value` pairs.
  - And can be accessed in either of two ways:
    - `myObject.field` // Let's use this one for the most part.
    - `myObject[ 'field' ]`
- Objects have no pre-set structure (no class definition as seen in C++ or Java)
  - And thus (in bad programs), the "same" type of object could have some different fields…

JavaScript Objects Example

- In C++/Java, we define classes, then create objects from them.
- In JavaScript, it is a little different.

```
let xObj = {}; // xObj is an object – Haven't created a class for xObj
xObj.someField = 4; // xObj now has a field named "someField".
console.log(xObj.someField ); // Prints 4 to the CONSOLE!!!
```

- Add a method to object x:

```
xObj.aMethod = function(){ console.log( "Hello" ); };
xObj.aMethod(); // Calls the method (and prints hello).
```

JavaScript Objects Example

- Note: `" "` and `''` mean the same thing in JavaScript: both delimit strings.
- `xObj = {};` // Creates an empty object.
- Approach 1

  `xObj.a = 7;` // Adds a field (member variable) to the object.
- Approach 2

  `xObj[ 'b' ] = "Hello";` // Adds another field.

- `xObj.a == xObj[ 'a' ] ==7` // Two ways to access the a field that is in x.
- `xObj.b == xObj[ 'b' ] == "Hello"` // True
- If you know your fields ahead of time, you can create an object literal like this:

  `xObj = { a: 7, b: "Hello"};`

JSON

- JavaScript Object Notation (JSON)

```
'{ a: 4, b: "Hello", "2" : "two" }'
```

- Used to represent objects and pass them (their data) around in many different applications.
- Exchanging structured data
  - Write it to / read it from disk
  - Across the network (Typically between a server and web applications)

JSON Example

- Comments // are NOT allowed in JSON. Below are just for informational purposes!!!

```
{
   "book": [  // <- Notice the [ (square bracket)… what does it mean?
      {       // [ ] means array… so this is an array of what?
         "id": "444",                    // The { } means object, so an array of objects…
         "language": "C",                // id, language, edition, author are all what?
         "edition": "First",             // Fields of the object.
         "author": "Dennis Ritchie"
      },
      {
         "id": "555",
         "language": "C++",
         "edition": "second",
         "author": "Bjarne Stroustrup"
      }
   ]
}
```

- To use the above JSON, it would be written as a String (surrounded with “ ”).

Methods

- How do we add a method to an object?
  - Add a new field to the object (where the field is a function):

```
let myObj = { };
myObj.methodX = function( a, b ) { … }
```

- Methods are treated the same way as member variables (fields) – you just add them to your object.
  - Note: just like variable types in JavaScript, you don't specify function (return) types (or parameter types).

More JS syntax

- `this`
  - The current object
  - Print out the value of *`this`* to see which object you are currently working with
- for each loops
  - `for( let prop in obj )` // Iterate through the *properties* of an object.
  - `for( let val of array )` // Iterate through the items in an array.
  - `for( let i = 0; i < arr.length; i++ )` // Can also use normal for loop.
- `3 == '3'` // true – JS tries to convert things to match
  - `3 === '3'` // false – strict equality (probably should just use this)

More JS syntax

- In JavaScript, Strings are compared using ==

```
let s1 = "hello"
let s2 = "hello"
s1 == s2 // true
```

- Create constants using:

```
const MAX_NUMBER = 1000;
```

- String or Number?
- `let x = Math.random().toFixed( 2 );` // x => 0.35
- In the above case, x is a string, NOT a number, so:
- `x += 10;` The result would be: 0.3510
- `x = Number( x );` // Turn x back into a number.

JavaScript Functions

- JS Functions are similar to C++ functions (with a couple of differences).

1) Declare/Define a function:

```
function myFunctionName( param1, param2 ) { … }
```

2) Create a function and store it in a variable:

```
let myFunctionName = function( param1, param2 ) { … }
```

- What differences do you see in the above function declarations vs a Java/C++ function declaration?
  - Parameters do not have a type specified.
  - The return type is not listed.
  - Can create a variable of type “function” directly.
  - We can store functions in variables!
- Functions can be defined anywhere – including within other functions.

Passing Function Parameters

```
function doit( x, y, z ) {
}
```

- Calling doit

```
doit( 1 )
doit( 1, 2 );
doit( 1, 2, 3 );
```

- All valid ways to call `doit`… however, y and z will be *undefined* if not passed in.
- Very flexible
- Note: *`arguments`* is a hidden variable that is an array of parameters passed to the function.
  - `x == arguments[ 0 ]`

Example

```
function doit( x1, x2, x3 ) {
    console.log( arguments.length );
    console.log( arguments[0] );
    console.log( x2 );
}
```

- `doit( 99 )` displays:

```
1
99
undefined  // Kind of null.
```

Using “use strict”

- “use strict”; // At the beginning of your script! Note: “ ” are required.
  - This mode changes previously allowed “bad syntax” into real errors.
  - x = 3.14; // ERROR: x was not declared.
  - function doit(p1, p1) { } // ERROR – don’t name your parameters the same…
  - let MAX_VALUES = 123 // (Sometimes) ERROR: No semicolon

Loading/Running JS code

- No ***Main***, JavaScript just starts in your file and runs the code.
- JavaScript runs on the browser – after the HTML has been loaded.
- When does JavaScript code run?
  - As soon as it is seen.
  - After all the code / images / css / etc is loaded…

```
function main() {
      console.log( “Everything is loaded!” );
}
window.onload = main;
```

- Note “main” is not a key word. You may also write:

```
window.onload = function() {
      console.log( “Everything is loaded!” );
}
```

Adding JS to your Webpage

- Option 1 (Recommended)
  - Place your JS code in a separate file and link it to your html file through the “script” tag:

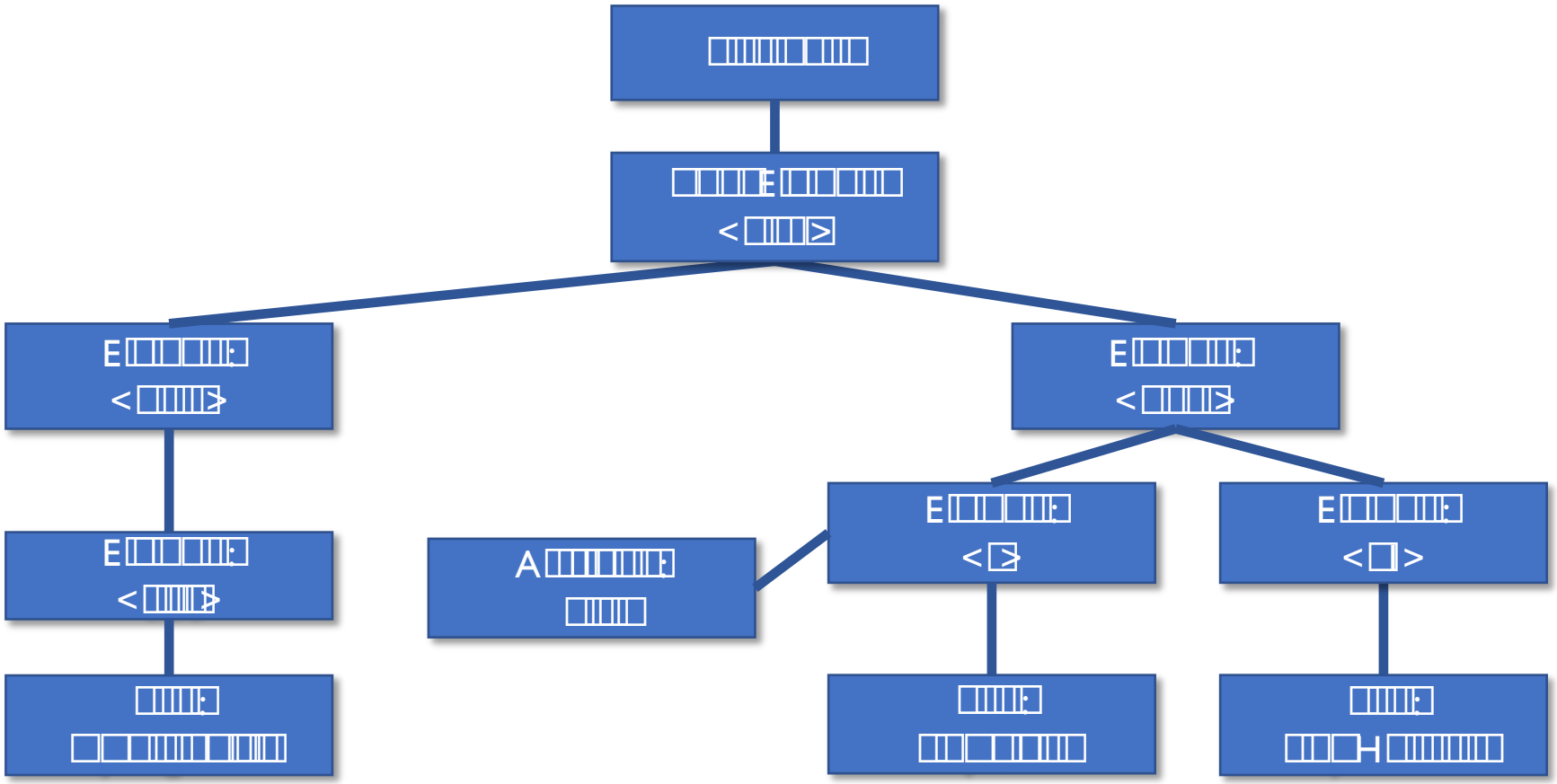
```
<head>
    <script type=“text/javascript” src=“myScript.js”/>
</head>
```

- Option 2
  - Place your JS code directly in your html file
- To see the console output, look at your browser’s developer tools panel. This panel has a lot of useful info about what is happening in your web page.
  - console.log(‘My debug message’);
  - Note, you can write and run code in the console.

HTML Document

```
<html>
        <head>
                <title> My Page Title </title>
        </head>
        <body>
                <p> A paragraph with a
                        <a href="www.google.com">link to google</a>
                </p>
        </body>
</html>
```

HTML representation



- This does not exactly match the previous slide but gives you the idea.

Document Object Model (DOM)

- When your browser requests and receives a webpage, it doesn't just display the HTML as it receives it.
- The page is stored internally as an object. Specifically, as a "Tree" of "Nodes"
  - Similar to the GUI layout / widgets from the Synthesizer assignment.
  - The DOM is the representation of the current webpage.
- A node is an HTML element (basically a tag, or an attribute of a tag).
- A "Tree" is a way of connecting the nodes such that each node has exactly one parent, and possibly several children. (Similar to a family tree.)
- Remember, the purpose of JavaScript is to manipulate webpages – or more specifically to manipulate the DOM.

JavaScript + DOM

- JavaScript has built-in methods for interacting with the DOM
  - Selecting elements
  - Traversing the tree
  - Modifying nodes
  - Modifying the tree

document

- *`document`* is a variable that refers to the page you are viewing in your browser.
  - *`document`* is created automatically for you – it already exists as your code starts executing.
- `document.writeln( “<p> CS 6011 </p> “ );`
  - Adds the text directly to the documents body.
  - Note: Deletes anything that was in the document from the HTML file.
- Most of the JavaScript API methods for interacting with the DOM are part of the *document* object.
- document.body – The <body> element of the DOM.

Selecting Elements (JS)

```
let elems = document.getElementsByTagName( tagName ); // eg: 'p'
```

- returns an array of nodes

```
let elem = document.getElementById( someId ); // returns 1 element.
```

- returns a single node with that ID (as IDs should be unique!)

```
let elems = document.getElementsByClassName( someClassName );
```

- returns a list of nodes with the class you gave to them (as can be referenced by CSS)

```
let elem = document.querySelector( "#elemId" );
```

```
let elems = document.querySelectorAll( ".myclass" );
```

- Take a "css like selector string" and return the first/all matching elements.
- Like a combination of the get functions.
- #idName — Get element with id "idName"
- p.example — Get a <p class="example"> element.

Traversing the Tree

- Once you have an element (using one of the methods from the previous slide)…
- Elements have a `.children` field which is a list of child nodes.
  - Note: Text nodes are not listed in the children…
    - Use `.childNodes` if you wish to see the Text nodes too.
- `querySelector()` works with a node and returns matching nodes that are descendants of a given node.

```
let firstHeading = myDiv.querySelector( 'h1' );
let allH2sInMyDiv = myDiv.querySelectorAll( 'h2' );
```

Modifying Node Examples

- Modifying a CSS property:

  myElement.style.background = “rgb(255, 0, 0)”;

  myElement.style.background = “red”;

- You can change (replace all of) the actual HTML too:

  myElement.innerHTML = “ The new HTML code ”;

- To just update the text of a node:

  myElement.textContent = “New text…”;

Modifying the Tree

- Create and return a new element:

```
let myNewElem = document.createElement( tagName );
let myPar = document.createElement( 'p' );
```

- Add a new child to an element:

```
element.appendChild( someElement );
```

- Create a text node to add to an element with text inside:

```
let myText = document.createTextNode( "Some Text" );
```

- Removing elements:

```
element.remove();  // Removes element from its parent.
node.removeChild( child );  // Removes child from node.
```

Monday Assignment(s)

- In class example…
- HW 4 – Synthesizer GUI – Final
- Lab – JS Hello World
- Lab – DOM Manipulation