**Fundamentals – JavaScript**

**by Todd Bagley**

**EXECUTION CONTEXT** - With respect to browsers, the JavaScript engine exists along side other engines dedicated to functionality including, but not limited to, rendering and HTTP communications.

When the JavaScript engine processes developer-generated code, essentially it makes two passes. The first pass is dedicated to construction of an Execution Context, which consists of (1) an Object to represent the context, (2) the pointer “this” that references the Object, (3) a link or relationship to the Outer Environment and lastly (4) “hoisting”, which is term representing a systematic process whereby memory is allocated… (4a) defining each declared variable as ‘undefined’ and (4b) defining each function as written by the developer. The second pass synchronously processes the available code within the current lexical level.

During this second pass, when the code encounters a function call the JavaScript engine constructs a new Execution Context and repeats this two-step process, stacking the new context on top of the current context.

Understanding this process is a critical first-step to effective coding, testing and debugging.

**EVENT HANDLING** - After all execution contexts have been processed and the stack is clear, the JavaScript engine runs a continuous loop to monitor the Event Queue. As events occur (examples: user clicks on a button, tabs to focus on an input element, makes a keystroke, etc., etc.), functions that have been registered to a relevant event suspend the event handler loop and initiate the creation of a new execution context.

**SCOPE CHAIN, ASSOCIATIVITY AND COERCION** - Variables defined within a given execution context have the potential to be available to all subsequent execution contexts. When an execution context uses a variable that is not defined within its own “scope”, the link to the Outer Environment is used to look for the variable in the parent context.

Associativity and Coercion are “automated features” of JavaScript that process code according to specific policies and procedures. These “automations” not only allow JavaScript to run when less than complete information is provided by the developer, they can also cause confusion when they are not well understood. Associativity is the order Operator functions get called in (left-to-right or right-to-left) and Coercion is how the JavaScript engine resolves processing of dissimilar variable types.

**ASYNCRONOUS CALLBACKS** - JavaScript is a single-threaded (synchronous, code processed line-by-line) technology. That said, asynchronous behavior is possible via the use of Callbacks.

**PRE-PROCESSORS AND AUTOMATION-SUPPORTED DEVELOPMENT** - Due to innovation’s need for backwards compatibility to an endless sea of ubiquitously installed web browsers, pre-processors are important new tools enabling development of cross-browser compatible code. ES6>ES5 translates JavaScript. Likewise, SASS and LESS translate CSS (cascading style sheets). Tools like Gulp automate the repetitive tasks associated with running pre-processors during the development process.

**Code Sample**

<html>

<head>

</head>

<body>

<div id="output">output</div>

<script src="app.js"></script>

</body>

</html>

// begin "app.js" //////////////////////////////////////////////////////////////

var cnt = 0;

var myVar = 'hi world';

function helloWorld() {

myVar = 'Hello World';

}

function asyncFunc(callback) {

helloWorld();

callback('here is the callback');

}

function updateDom() {

cnt++;

var eid = document.getElementById("output");

eid.innerHTML = myVar + ' (' + cnt + ')';

}

function asyncCall() {

asyncFunc(function (res) {

console.log(res + ' - wow!');

});

updateDom();

}

updateDom();

setTimeout(`asyncCall()`,2000);

