**COURSE CONTENT FOR JAVASCRIPT**

**Introduction JavaScript**

1. More code, less words
2. Exhaustive code and repetition
3. Color-coding conventions
4. Code examples

**JavaScript Objects**

1. Creating objects
2. JavaScript constructors create and return object instances
3. The native JavaScript object constructors
4. User-defined/non-native object constructor functions
5. Instantiating constructors using the new operator
6. Creating shorthand or literal values from constructors
7. Primitive (aka simple) values
8. The primitive values null, undefined, "string", 10, true, and false are not

Objects

1. How primitive values are stored/copied in JavaScript
2. Primitive values are equal by value
3. The string, number, and Boolean primitive values act like objects when used like

Objects

1. Complex (aka composite) values
2. How complex values are stored/copied in JavaScript
3. Complex objects are equal by reference
4. Complex objects have dynamic properties
5. The typeof operator used on primitive and complex values
6. Dynamic properties allow for mutable objects
7. All constructor instances have constructor properties that point to their constructor

Function

1. Verify that an object is an instance of a particular constructor function
2. An instance created from a constructor can have its own independent properties (aka

instance properties)

1. The semantics of "JavaScript objects" and "Object() objects"

**Working with Objects and Properties**

1. Complex objects can contain most of the JavaScript values as properties
2. Encapsulating complex objects in a programmatically beneficial way

Getting, setting, and updating an object's properties using dot notation or bracket

Notation

1. Deleting object properties
2. How references to object properties are resolved
3. Using hasOwnProperty to verify that an object property is not from the prototype

Chain

1. Checking if an object contains a given property using the in operator
2. Enumerate (loop over) an object’s properties using the for in loop
3. Enhancing and extending objects with Underscore.js

**String()**

1. Conceptual overview of using the String() object
2. String() parameters
3. String() properties and methods
4. String object instance properties and methods

**Number()**

1. Conceptual overview of using the Number() object
2. Integers and floating-point numbers
3. Number() parameters
4. Number() properties
5. Number object instance properties and methods

**Boolean()**

1. Conceptual overview of using the Boolean() object
2. Boolean() parameters
3. Boolean() properties and methods
4. Boolean object instance properties and methods
5. Non-primitive false Boolean objects convert to true
6. Certain things are false, everything else is true

**Working with Primitive String, Number, and Boolean Values**

1. Primitive/literal values are converted to objects when properties are accessed
2. You should typically use primitive string, number, and Boolean values

**Null**

1. Conceptual overview of using the null value
2. typeof returns null values as "object"

**Undefined**

1. Conceptual overview of the undefined value
2. JavaScript ECMA-262 Edition 3 (and later) declares the undefined variable in the

global scope

**The Head/Global Object**

1. Conceptual overview of the head object
2. Global functions contained within the head object
3. The head object vs. global properties and global variables
4. Referring to the head object
5. The head object is implied and typically not referenced explicitly

**Object()**

1. Conceptual overview of using Object() objects
2. Object() parameters
3. Object() properties and methods
4. Object() object instance properties and methods
5. Creating Object() objects using "object literals"
6. All objects inherit from Object.prototype

**Function()**

1. Conceptual overview of using Function() objects
2. Function() parameters
3. Function() properties and methods
4. Function object instance properties and methods
5. Functions always return a value
6. Functions are first-class citizens (not just syntax, but values)
7. Passing parameters to a function
8. this and arguments values are available to all functions
9. The arguments.callee property
10. The function instance length property and arguments.length
11. Redefining function parameters
12. Return a function before it is done (i.e. cancel function execution)
13. Defining a function (statement, expression, or constructor)
14. Invoking a function (function, method, constructor, or call() and apply())
15. Anonymous functions
16. Self-invoking function expression
17. Self-invoking anonymous function statements
18. Functions can be nested
19. Passing functions to functions and returning functions from functions
20. Invoking function statements before they are defined (aka function hoisting)
21. A function can call itself (aka recursion)

**The this Keyword**

1. Conceptual overview of this and how it refers to objects
2. How is the value of this determined?
3. The this keyword refers to the head object in nested functions
4. Working around the nested function issue by leveraging the scope chain
5. Controlling the value of this using call() or apply()
6. Using the this keyword inside a user-defined constructor function
7. The keyword this inside a prototype method refers to a constructor instance

**Scope and Closures**

1. Conceptual overview of JavaScript scope
2. JavaScript does not have block scope
3. Use var inside of functions to declare variables and avoid scope gotchas
4. The scope chain (aka lexical scoping)
5. The scope chain lookup returns the first found value
6. Scope is determined during function definition, not invocation
7. Closures are caused by the scope chain

**Function Prototype Property**

1. Conceptual overview of the prototype chain
2. Why care about the prototype property?
3. Prototype is standard on all Function() instances
4. The default prototype property is an Object() object

**DOM/CSS Scripting**

1. Introduction to the Document Object Model (DOM)
2. Using the getElementById method
3. Modifying Page Content with the DOM
4. Manipulating CSS using JavaScript
5. Programmatic Access to CSS

**Common Applications**

1. Form Validation and Testing
2. Working with Regular Expressions
3. User Interaction
4. Local Form Processing
5. Object Detection
6. Creating New Windows
7. Adding Content to a Window
8. Rollover Images