**Client-side web scripting**

* It associates a script with an HTML.
* It allows program (i.e. scripts) to be downloaded from a web server and executed in the client environment (e.g. browser).
* **Common client-side scripting technologies**
* JavaScript, ECMAScript, Jscript
* VBScript, ActionScript
* Java Applets, ActiveX Controls, Flash Animations, Microsoft Silverlight, Adobe Integrated Runtime (AIR)
* **Common uses**
* Dynamic (X)HTML
* Client-side form data validation
* Asynchronous content retrieval, RIA Technologies (e.g. AJAX)
* **Common issues**
* Browser support
* No scripting support
* Scripting disabled
* Plug-in availability
* Version incompatibilities, non-standard implementations
* Capabilities restrictions
* Scripting languages are not for general-purpose programming
* E.g. JavaScript is restricted by the sandbox execution model and the same origin policy
* Security risks
* Browser implementation defects (e.g. buffer overflows)
* E.g. JavaScript, cross-site scripting (XSS) or cross-site request forgery (XSRF) issues
* Malicious ActiveX controls

**JavaScript**

* It is developed by **Brendan Eich** at Netscape Communications as the scripting language for the Netscape Navigator Browser.
* It is formerly called **Mocha**, then **LiveScript**, then **JavaScript**
* Standardized by ECMA International as ECMAScript
* JavaScript Frameworks
* script.aculo.us, jQuery, MooTools, Prototype, Dojo, Toolkit, etc.
* Linked/Embedded in web pages using the <script> element
* **Linked**

<script type=”text/javascript” src=”script.js”> </script>

* **Embedded** (either in the <head> or the <body> element)

<script type=”text/javascript”>

<!--hide script from non-JavaScript browsers…

/\* script code goes here \*/

//end of script hiding … -->

</script>

<noscript>

/\* content to show when scripting not available \*/

</noscript>

* JavaScript + DOM/BOM +CSS +(X)HTML = DHTML
* JavaScript code in (X)HTML pages can be executed “on the fly” as the document is rendered (i.e. code outside of functions executes as the <script> element is encountered); in most cases though, JavaScript code is executed in response to document events (e.g. clicking a page element).
* Basic language features:
* Paradigm
* Object-oriented (prototype-based), functional, imperative scripting language
* Java-/C like syntax
* Implicit semicolon insertion for statement termination
* Identifiers are alphanumeric, \_, and $ characters
* Single-line (//) or block (/\* \*/) comments
* Type system and variables scoping rules
* Dynamic (aka loose or weak) typing
* Global (aka top-level) or local scopes
* Data Types
* Primitive Types
* **Numbers** (binary, decimal, hexadecimal, octal notation and exponential)
* **Boolean** (true, false)
* **Strings** (single or double quote delimited)
* **Undefined and Null** (empty string, false, 0)
* Composite (object) Types
* **Core JavaScript Objects** (Object, Number, Boolean, String, Date, Math, Global, RegExp, Error, Arrays (Array), Functions (Function, Arguments))
* **Browser Object Model (BOM)** (Window, Screen, Map, Location, etc.)
* **Document Object Model (DOM)** (Attr, Comment, Element, Node, Text, Form, Image, Input, Layer, Link, Option, Select, Style, TextArea, Applet, Anchor)
* Keywords
* break, case, catch, continue, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, with
* Reserved words (currently unused)
* abstract, Boolean, byte, char, class, const, debugger, float
* Statements and control structures
* var
* used to declare global/local scoped variables
* if-else
* condition expressions having values of 0, “”, null and undefined evaluate to false
* switch-case-default-break
* allows any expression type to be used as the switch expression
* case labels may be different types
* case labels may be expressions
* case execution falls-through, unless terminated by a break
* while, do-while, for, for-in, break, continue
* for while and do-while, false condition expressions similar to if-else
* for-in used for property enumeration
* allows labeled break/continue
* try-catch-finally, throw
* throw and catch can handle any expression type
* function, return
* JavaScript functions are similar to Java methods except for the following differences:

- No return value type is specified, and return is optional within the function body

- Functions may return a value on one invocation and not return a value (i.e. have an undefined return value) on another invocation

-Functions may return different types of values on different invocations

-Function parameters are dynamically typed

- Functions can be invoked with an arbitrary number of arguments, regardless of the actual parameters specified in the function definition (the Arguments object can be used to access unnamed arguments passed to the function invocation)

-Functions are first-class objects

* Functions can be invoked as global functions (i.e. as methods of the Global object) as methods of specific objects, or as object constructors
* with
* used to access object properties without having to explicitly qualify the property with the object name (serves as shorthand notation for accessing object properties, at the expense of program readability)
* **Truthy and Falsy Things**

You need to understand the importance of truthy values and falsy values

and what’s the difference between this two.

**Values that evaluate to true:**

'0';

'any string';

[]; // an empty array

{}; // an empty object

1; // any non - zero number

**Values that evaluate to false:**

'0';

' '; //empty string

NaN; // JavaScript’s “not-a-number” variable null;

undefined; // undefined can be redefined !

* **Four Types of Basic Data Structures**

1. **Array Lists**:

* these are the special objects in JavaScript called the ‘arrays’.

for example, ["one","two","three"],

1. **Records**:

* also the special JavaScript objects,

for example, {firstName: "Juan", lastName: "Delacruz"}

1. **Maps**:

* which are also special JavaScript objects,

for example, {"one":1,"two":2,"three":3}

1. **Entity Tables**:

* Table: A collection of entities, meaning a single table can have different

sets of properties

* Entity: This is the set of properties it is similar to the database row.
* **Asynchronous JavaScript and XML (AJAX)**
* Synchronous Request
* Recommended for few small requests. JavaScript will not continue to execute, until the server response is ready. If the server is busy or slow, the application will hang or stop.

var xhr = new XMLHttpRequest();

xhr.open(“GET”, “somepage.txt”, false);

xhr.send(null);

if(xhr.status==200){

var str = xhr.responseText;

}

* Asynchronous Request
* JavaScript does not have to wait for the server response. It can execute other scripts while waiting for server response, and deal with the response when the response is ready.
* Three important properties of the XMLHttpRequest Object
* Onreadystatechange - Stores a function to be called automatically each time ready state changes
* Readystate
* 0 : request not initialized
* 1 : server connection established
* 2 : request received
* 3 : processing request
* 4 : request finished and response is ready
* STATUS
* 202: OK
* 404 : Not Found
* **XMLHttpRequest** – to request HTTP from the server
* **JavaScript Object Notation (JSON)**

var vehicle = {

“plateno”: “abc123”,

“brand”: “civic”

}

*property* = “plateno”

*value* = “abc123”

* JSON.stringify – used to convert an object file into string
* JSON.parse – used to parse JSON into an object

**DOCUMENT OBJECT MODEL**

* Document Object Collections
* anchors[] forms[] images[] links[]
* Document Object Properties
* Cookie, document Mode, domain, lastModified, readyState, referrer, title, URL
* Document Object Methods
* close() getElementById() getElementsByName() getElementsByTagName() open() write() writeln()