***Web App Highlights***

The Architecture of a Web Application

**1.1.1 The Request**

The process starts when the user (at a web browser running on a computer) types a URL (Uniform Resource Locator, i.e. a web site address) into their browser and hits enter or presses the "go" button. The browser then sends a request over the network to a web server (another computer). The request is formatted by the browser according to a special set of rules (a "protocol") called HTTP (Hypertext Transfer Protocol).

**1.1.2 The Response**

At the other end, the web server receives the request and interprets it. Usually it is a request for a file, and if all goes well the server finds the file the client asked for and retrieves it to use in an HTTP- formatted response. (If the file is not found, the server will send back an error code as its response.)

The file requested might be a web page written in HTML (Hypertext Markup Language), possibly including some CSS (Cascading Style Sheet) commands and/or some JavaScript commands. In this case, the file is just sent back as is.

1.1.3 Server-Side Scripts

But the file might also contain a script (a computer program) written in PHP (PHP: Hypertext Preprocessor), Java, ASP.NET (Microsoft’s Active Server Pages framework), or some other language. If so, the server executes the script and the script’s output is sent as the response. Typically this output includes a web page (containing HTML, possibly with JavaScript and/or CSS) that is tailored to suit the particular user who sent the request.

1.1.4 Server-Side Database Interaction

If the web server has a database, the script might also interact with this database as part of its operation (e.g. to retrieve previously stored user information, to store the contents of a form the user filled out, etc.). This database interaction is usually performed in a language called SQL (Structured Query Language).

1.1.5 Client-Side Response Processing

When the response from the server arrives, the browser reads the HTML and constructs a Document Object Model (DOM), which serves as the browser’s internal model of the web page. It also fetches and applies styles from any associated CSS information, and it might run some associated JavaScript as well, which can cause it to add, remove, and modify parts of the DOM. Then it draws the DOM to the screen.

1.1.6 Client-Side Scripts

If the page contains JavaScript functions, that information might be used to modify the appearance of the page in response to user interaction (e.g. drop-down menus or hover menus, button presses, gestures, etc.) before a link is clicked or button is clicked that launches the next HTTP request and causes the cycle to repeat.

***JavaScript Highlights***

**Introduction**

According to w3schools.com:

JavaScript Can Change HTML Content

JavaScript Can Change HTML Attributes

JavaScript Can Change HTML Styles (CSS)

JavaScript Can Validate Data

JavaScript can make web pages more interactive with some special effects

JavaScript is a prototype-based, client-side scripting language

JavaScript runs in the browser by being added into an existing HTML document

JavaScript code can also be called as an external file (with extension .js)

**Basic I/O**

alert(..)

console.log(…)

confirm(….)

prompt(….)

document.write(…..)

Variables

Case sensitive

Must begin with a letter, underscore ( \_ ), or dollar character ($)

Cannot begin with a number or any other character that is not a letter (other than the underscore and dollar)

Avoid keywords and reserved words

Can hold value of any type such as number, string, array, boolean, object, etc.

var lastName, firstName = "Wilson";

var stdID; stdID = 2288900122;

var firstTime = true, lastTime = false, empty = null;

Operations

Common Arithmetic Operators

Practice 2

Common Comparison Operators

Practice 3

Math Methods

Type Conversion

parseInt() returns the integer equivalent of its argument (or NaN)

parseFloat() returns the float equivalent of its argument (or NaN)

isNaN() return true if the argument cannot be converted to a number.

parseInt("39")

parseInt("batman888")

parseInt("+007Bond")

parseInt("-57ooo")

parseFloat("28")

parseFloat("62.32")

parseFloat("9.9.9")

parseFloat("-17.88")

Arrays

Arrays in Javascript are typeless which means you can mix data types within an array

var a = new Array();

var a = [ ];

var a = ["Andy", "Male", 992100233, true];

a.length displays the number of elements in the array a.

Array Creation

var provinces = new Array("ON", "NS", "BC", …);

//Creates Empty Array

var provinces = new Array( ); var provinces = [ ];

//Adds to Array

provinces.push("ON");

provinces.push("NS");

provinces.push("BC");

Array of Objects

Array Methods

Associate Arrays

Associate Arrays are arrays with string as index instead of numeric index.

var users = [ ];

users['name'] = "Andy";

users['age'] = 25;

users['id'] = "AA999";

console.log(users['name']);

console.log(users.name);

console.log(users.length);

console.log(users[0]);

Arrays - filer, find, reduce functions

The filter function is used to return an array that contains only matching elements from the original array base on a condition.

let stocks = [{ id: "A1", count: 45, prov: "ON" },

{ id: "A2", count: 89, prov: "MB" },

{ id: "A3", count: 97, prov: "AB" },

{ id: "A4", count: 81, prov: "ON" },

{ id: "A5", count: 66, prov: "MB" }];

var results = []

var results = stocks.filter((stock)=> stock.count < 80);

The function find does something similar to the filter function with two major differences. It returns a single element instead of an array and the returned element is the first element that satisfies the condition being tested.

let stocks = [{ id: "A1", count: 45, prov: "ON" },

{ id: "A2", count: 89, prov: "MB" },

{ id: "A3", count: 97, prov: "AB" },

{ id: "A4", count: 81, prov: "ON" },

{ id: "A5", count: 66, prov: "MB" }];

var results = stocks.find((stock)=> stock.count < 80);

The reduce function executes a predefined function for array element and returns the function's accumulated result. The items of the array are executed in the left-to-right sequence.

let stocks = [{ id: "A1", count: 45, prov: "ON" },

{ id: "A2", count: 89, prov: "MB" },

{ id: "A3", count: 97, prov: "AB" },

{ id: "A4", count: 81, prov: "ON" },

{ id: "A5", count: 66, prov: "MB" }];

var result = stocks.reduce((total, x) => total + x.count, 0 );

var items = [ 0, 1, 2, 4, 5 ];

var result = items.reduce((total, x) => total + x );

Loops

For Loop

For/Of Loop - for iterable objects such as arrays, strings, etc.

While Loop

Do/While Loop

forEach Loop

var shapes = ["Square", "Circle", "Rectangle", "Triangle", "Ploygons"];

shapes.forEach(x => {

console.log(x);

})

DOM (Document Object Model...)

DOM is an arrangement of the content of an HTML document into a tree-like structure (the DOM tree). Each part of the HTML document is represented by a node on the tree including elements, attributes, and text.

The document object provides properties and methods which allow access and modification of document content.

DOM Properties

<html>

<head>

<title>DOM Exercise</title>

<meta charset='UTF-8'>

</head>

<body>

<h1>My Name</h1>

<h2>My ID</h2>

<section>

<p>My Program</p>

<p>My Campus</p>

</section>

</body>

</html>

x.innerHTML -

x.nodeName -

x.parentNode -

x.childNodes -

x.attributes - the text value of x

the name of x

the parent node of x

the child nodes of x

the attributes nodes of x

Properties available to navigate between nodes:

parentNode

childNodes

firstChild

lastChild

nextSibling

previousSibling

Download and run exercise\_1.html from SLATE.

In browser console execute the above comments...

document.getElementsByTagName('body')[0].xxxxx

Try executing these commands...

document.contentType

document.title

document.all

document.body.childElementCount

Retrieve a Node (HTML element)...

By ID

document.getElementById('xxx')

By Tag Name

document.getElementsByTagName('xxx')

By Class Name

document.getElementsByClassName('xxx')

By CSS Selectors

CSS Selectors

Change a Node ...

Changing the HTML Content

Changing the value of attribute

Changing CSS Properties

Add a Node ...

Creating New Nodes

Property/Method Action

createElement(new\_node) Create a new element node

createTextNode(new\_node) Create a new text node

Inserting Nodes into the DOM

Property/Method Action

parent\_node.appendChild(new\_node) Add a node as the last child of a parent element

parent\_node.insertBefore(new\_node, next\_sibling) Insert a node into the parent element before a specified sibling node

node.replaceChild(new\_node, existing\_node) Replace an existing node with a new node

Removing Nodes into the DOM

Property/Method Action

parent\_node.removeChild(child\_node) Remove child node

existing\_node.remove() Remove node

ECMAScript...

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.

ECMAScript is the official name of the language.

ECMAScript Versions:

Functions...

A function is basically a script that can be called to perform a task

They can be used more than once within a script to perform their task

A function needs to be declared with or without parameters

A function can return a value back to the calling script

A function has to be called to perform its task

…...

Example...

Functions can be defined anywhere within the script due to function declaration hoisting - function declarations are hoisted to the top of the scope in which they are defined. This means it's possible to call a function before it's actually been declared in your code.

However, it is always a good idea to have them declared at the top of the document or may be inside the <head> element.

Example...

What is the output of this script?

ES6 Function Features...

Default Parameter Values

JavaScript functions have a unique feature that allows for you to pass any number of parameters during function call (actual parameters) regardless of the number of parameters present in the function definition (formal parameters). So you need to include default parameters just in case someone forgets to pass one of the parameter.

In ES5:

function counts(number1,number2) {

number1 = number1 || 5;

number2 = number2 || 10;

console.log(number1 + ',' + number2);

}

counts(4);

In ES6:

function counts(number1=5, number2=10) {

console.log(number1 + ',' + number2);

}

counts(4);

OR

function counts(number1=5, number2=getParm()) {

console.log(number1 + ',' + number2);

}

function getParm() {

return 30;

}

counts(4);

counts(undefined, 90);

Rest Parameters

A rest parameter is simply a named parameter which is preceded by three dots(...) – the rest operator. This named parameter becomes an array which contains rest of the parameters (i.e apart from the named parameters) passed during function call.

function counts(number1, ...allOthers) {

console.log(number1 + ',' + allOthers);

}

counts(4,5,6,7,8);

Arrow Functions =>

Arrow Functions, or "fat arrow functions," introduce a new syntax for defining functions that is very concise. We can avoid typing keywords like function, return and even curly brackets { } and parentheses ().

let mirror = value => value;

Equivalent to:

let mirror = function(value) {

return value;

}

let myAdd = (n1,n2) => n1+n2;

Equivalent to:

let myAdd = function(n1,n2) {

return n1+n2;

}

let Hello = () => 'Hello Andy';

Equivalent to:

let Hello = function() {

return 'Hello Andy';

}

For a function body having more than just a return statement, you need to wrap the body in curly braces just like traditional functions.

let calculate = (n1,...numN) => {

let result = n1;

for (var i in numN)

result += numN[i];

return result;

}

map Functions =>

The map function returns an array of the same length with the results of calling a function on every element in the calling array.

let stocks = [{ id: "A1", count: 45, prov: "ON" },

{ id: "A2", count: 89, prov: "MB" },

{ id: "A3", count: 97, prov: "AB" },

{ id: "A4", count: 81, prov: "ON" },

{ id: "A5", count: 66, prov: "MB" } ];

let result = stocks.map(stock => stock.count \* 2);

Variables...

Similar to function declaration hoisting, JavaScript moves all variable declarations to the top.

JavaScript doesn’t require variables to be declared.

In ES5 variables have two scopes: global and local.

Inside a function variables not declared will have global scope. Variables declared or listed as parameters are local to the function.

Variables declared outside of functions are always global, which means they are accessible to all functions on the page.

If a function contains a variable name that is the same as a global variable name, references to that name within the function will always be to the local variable.

var xx = 0;

function sum(zz) {

var xx = 1;

yy = xx + zz;

console.log(xx + " " + yy + " " + zz);

}

sum(50);

What are xx, yy, zz before and after calling the function?

function Car (model) {

var owner = "Bob Smith"; //local

var model = model; //local

this.getOwner = function( ) {

return owner; } }

var cObj = new Car ();

console.log(cObj.owner); ?

console.log(cObj.model); ?

console.log(cObj.getOwner()); ?

Download w3\_exercise\_1.html from SLATE and examine the codes.

ES6 Features...

let and const

Both let and const create variables that are block-scoped – they only exist within the innermost block that surrounds them.

Both let and const are NOT hoisted and must be declared before using.

Your can update a let variable but you cannot update a const variable.

Accessing a let or const variable before its declaraton (within its scope) causes a ReferenceError.

function func() {

if (true) {

var tmp = 123;

}

console.log(tmp);

}

function func2() {

if (true) {

let tmp = 123;

}

console.log(tmp);

}

function func3() {

let tmp = 20;

if (true) {

let tmp = 123;

console.log(tmp);

}

console.log(tmp);

}

Events...

HTML events are "things" that happen to HTML elements

When JavaScript is used in HTML pages, JavaScript can "react" on these events. (W3schools.com)

<button onclick="document.write('Hi')">Click Me</button>

<input type="button" value="Press me" onclick="alert('Press me')">

Download w3\_exercise\_2.html from SLATE and examine the codes.

Download w3\_exercise\_3.html from SLATE and examine the codes.

Download catchTheRabbit.html from SLATE and try to catch some rabbits.

More Functions...

Hoisting

JavaScript builds its execution environment in two passes.

The declaration pass sets up the runtime environment, where it scans for all variable and function declarations and creates the identifiers.

The second pass is the execution pass. After the first pass, all declared functions are available, but variables are still undefined until values are assigned to them.

Undefined is NOT the same as 'not defined'.

showMe();

function showMe() {

var x = 2;

console.log(x);

}

Function Expression (Anonymous Functions)

Function expression are functions that don't have names.

They are associated with a variable, an event, or something similar to run.

They can accept inputs and return outputs.

const x = 90;

const y = 3;

const showMe = () => {

console.log(x/y);

};

showMe(x,y);

const showMe2 = (a,b) => {

console.log(a/b);

};

showMe2(100,2);

Self-Invoking Functions

Function expressions can be made "self-invoking".

A self-invoking expression is invoked (started) automatically, without being called.

Self-invoking functions are useful for initialization tasks and for one-time code executions.

Function expressions will execute automatically if the expression is followed by ().

You have to add parentheses around the function to indicate that it is a function expression

( () => {

console.log("Hello!! I'm self-invoking.");

}) ();

More Examples:

console.log('Come before self-invoking');

( () => {

console.log('--- This is self-invoking.');

})();

console.log("After self-invoking.");

( () => {

console.log('--- Another self-invoking.');

})();

More Self-Invoking Functions

Parameters can be passed to self-invoking functions.

( function (a,b,...rest) {

console.log(arguments.length + ' arguments.\n\n');

return arguments.length;

})(1,2,3,4,5,6);

A self-invoking function can have variables and methods but they cannot be accessed from outside of it. To access them, the global window object has to be used.

(function() {

var me = 'Andy';

function meter(x) {

return 3.28084 \* x;

};

window.me = me;

window.meter = meter;

})(window);

console.log(me);

console.log(meter(3));

Nested Functions

JavaScript supports nested functions and "nested" functions have access to the scope "above" them.

In JavaScript, all functions have access to the scopes "above" them including the global scope.

function showName(firstName, lastName) {

var intro = "my name is ";

function addGreeting () {

var greeting = "Good day,";

function doIt() {

return greeting+" "+intro+" "+firstName +" "+lastName;

}

return doIt();

}

return addGreeting();

}

let show = showName("Michael", "Jackson");

console.log(show);

More Examples:

function createAdder(a,b) {

function addNumbers(a, b) {

let ret = a + b;

return ret

}

return addNumbers;

}

let adder = createAdder;

console.log(adder);

console.log('-------');

let adder2 = createAdder();

console.log(adder2);

console.log('-------');

console.log(adder2(1,3));

More Events...

HTML events are "things" that happen to HTML elements

An event handler is a JavaScript property of an object / element that is used to handle an event on a Web page

When events happen, the JavaScript event handlers will be able to identify the events and react appropriately with the predefined actions

Event handlers can be used directly within HTML elements by adding special attributes to those elements. They can also be JavaScript codes between the <script> tags

Event Handler - within HTML

<p id="clickme" onclick="function();"> You can click here</p>

<p id="clickme" onclick="alert('You clicked me');"> You can click here</p>

Event Handler - between <script>

<body>

<p id="clickme">You can click here</p>

<script>

var \_click = document.getElementById("clickme");

\_click.onclick = () => { alert("Hello"); };

\_click.addEventListener("click", () => {

alert("hello"); });

</script>

</body>

Try running this script:

<body>

<script>

var \_click = document.getElementById("clickme");

\_click.onclick = () => { alert("Hello") };

</script>

<p id="clickme">You can click here</p>

</body>

w4\_event\_ex1.html

Click here if you need help!!!

You can resolve the problem by moving the script to the bottom of the body section.

<body>

<p id="clickme">You can click here</p>

<script>

var \_click = document.getElementById("clickme");

\_click.onclick = () => { alert("Hello") };

</script>

</body>

Or, use window.onload / window.addEventListener to make sure the DOM has been loaded.

<body>

<script>

window.onload = () => { // OR

window.addEventListener("load", () => {

var \_click = document.getElementById("clickme");

\_click.onclick = () => { alert("Hello"); };

};

});

</script>

<p id="clickme">You can click here</p>

</body>

Try running this script:

<body onclick="clickme()" onkeypress="keypress()">

<script>

function clickme() {

alert("You clicked on " + window.event.srcElement.tagName); }

function keypress() {

\_key = window.event;

alert(\_key.type+" " + String.fromCharCode(\_key.keyCode)); }

</script>

</body>

w4\_event\_ex2.html

See this Example

<body>

<ul id="ul"></ul>

<script>

var ul = document.getElementById("ul");

for (let i=0; i<5; i++) {

const li=document.createElement('li');

li.textContent='first'+i;

li.addEventListener("click", (event) => {

console.log(event.srcElement.innerHTML);

console.log(li.innerHTML);

});

ul.appendChild(li)

}

</script>

</body>

More Events...

HTML events are "things" that happen to HTML elements

An event handler is a JavaScript property of an object / element that is used to handle an event on a Web page

When events happen, the JavaScript event handlers will be able to identify the events and react appropriately with the predefined actions

Event handlers can be used directly within HTML elements by adding special attributes to those elements. They can also be JavaScript codes between the <script> tags

Event Handler - within HTML

<p id="clickme" onclick="function();"> You can click here</p>

<p id="clickme" onclick="alert('You clicked me');"> You can click here</p>

Event Handler - between <script>

<body>

<p id="clickme">You can click here</p>

<script>

var \_click = document.getElementById("clickme");

\_click.onclick = () => { alert("Hello"); };

\_click.addEventListener("click", () => {

alert("hello"); });

</script>

</body>

Try running this script:

<body>

<script>

var \_click = document.getElementById("clickme");

\_click.onclick = () => { alert("Hello") };

</script>

<p id="clickme">You can click here</p>

</body>

w4\_event\_ex1.html

Click here if you need help!!!

You can resolve the problem by moving the script to the bottom of the body section.

<body>

<p id="clickme">You can click here</p>

<script>

var \_click = document.getElementById("clickme");

\_click.onclick = () => { alert("Hello") };

</script>

</body>

Or, use window.onload / window.addEventListener to make sure the DOM has been loaded.

<body>

<script>

window.onload = () => { // OR

window.addEventListener("load", () => {

var \_click = document.getElementById("clickme");

\_click.onclick = () => { alert("Hello"); };

};

});

</script>

<p id="clickme">You can click here</p>

</body>

Try running this script:

<body onclick="clickme()" onkeypress="keypress()">

<script>

function clickme() {

alert("You clicked on " + window.event.srcElement.tagName); }

function keypress() {

\_key = window.event;

alert(\_key.type+" " + String.fromCharCode(\_key.keyCode)); }

</script>

</body>

w4\_event\_ex2.html

See this Example

<body>

<ul id="ul"></ul>

<script>

var ul = document.getElementById("ul");

for (let i=0; i<5; i++) {

const li=document.createElement('li');

li.textContent='first'+i;

li.addEventListener("click", (event) => {

console.log(event.srcElement.innerHTML);

console.log(li.innerHTML);

});

ul.appendChild(li)

}

</script>

</body>

Forms...

Forms are used to collect inputs and feed them to other parts of the script or send them to remote servers.

To create a form object, you add a set of <form> and </form> tags and a number of HTML tags.

The most important form element is the <input> element.

The <table> element can work with the <form> element to present a structured format for input/output of data.

HTML Tags Used to Create Forms ...

Tags / Function

<fieldset></fieldset> Used to group related controls on a form Used for readability (optional)

<form></form> Creates the form for user input Required

<input /> Defines controls and attributes Required

<legend></legend> Caption that displays in the grouping border Used with <fieldset> (optional)

<textarea></textarea> Creates a mulit-line input area for text Required for text needed on multiple lines

<input> Element ...

The <input> Element - Types

text Free-form text field (no link breaks) Ex. Name field, Address field, ... HTML4

password Free-form text field for sensitive information Ex. Password entry (entry is masked) HTML4

checkbox none, part, or all can be chosen in a set of choices

Ex. Pizza toppings, Shopping list, .... HTML4

radio Only one option in a group of choices can be selected Ex. Gender, Marital status, ... HTML4

button Generic button that can be used it initiate a script HTML4

submit Button that initiates form submission to send the data from the user to the server HTML4

datetime Date and time year, month, day, hour, minute, second, fractions of a second HTML5

date Date year, month, day HTML5

month Date consisting of year and a month year, month HTML5

week Date consisting of a year and a week number Sunday = 0, Saturday = 6 HTML5

time Time hour, minute, seconds, fractional seconds HTML5

number Only numerical value accepted HTML5

range Range type is used for input fields that contains a value from a range of numbers HTML5

email Email value Format example: email@example.com HTML5

url URL value Format example: http://www.url.com HTML5

<form> Attributes

method POST versus GET

name Attaches a unique name to the form. This can then be used as a path to finding data in a specific input field (ex. using Javascript)

onSubmit Calls a client-side function. Will not go to the action page/script if this function returns false. When this function returns true, the action page/script is then called (ex. an age fields would use the number type to validate that the user does not enter anything but a number but a function could then be used to find out if the age is in a specific range)

action Used to tell the browser what page/script to call once the form is submitted (server-side)

<form name="entry" method="post" onSubmit="return checkform()" action="entry.php">

placeholder attribute ...

A new element in HTML5 which is used to provide a hint to ther user of what can be entered into a field.

Syntax: <input type="text" name="search" placeholder="search the web" />

Placeholder text must not contain line breaks

autofocus attribute ...

A new element in HTML5 which is used to automatically focus on one particular form field

Syntax: <input type="text" name="search" autofocus />

required attribute ...

A new element in HTML5 which is used to make a field required (will not submit without a value)

Syntax: <input type="text" name="search" required />

Download w5\_basicForm.html from SLATE and examine the codes.

Accessing Forms...

You can use any one of the following options to access forms:

Use the forms array of the document object

Use the name attribute

Use the id attribute

Form Arrays

You can access a form from the form array by:

document.forms.length

document.forms[n]

And for each form you can find out the number of elements.

document.forms[n].length

Name Attribute

You need a name attribute in the opening form tag.

<form name='xxxx'>

document.getElementsByName('xxxx')[n]

OR

document.xxxx

To find out the number of elements:

document.getElementsByName('xxxx')[n].length

OR

document.xxxx.length

What if you have more than one form with the same name?

Id Attribute

You need an id attribute in the opening form tag.

<form id='xxxx'>

document.getElementById('xxxx')

To find out the number of elements:

document.getElementById('xxxx').length

What if you have more than one form with the same id?

Accessing Form Contents

document.getElementById('...')

Requires an id attribute on every input element

document.forms["form name"]["input element name"]

Requires a name attribute on the <form> and on every input element

document.forms['form name'][input element number]

Only requires a name attribute on the <form>

document.forms[form number][input element number]

No names required

document.querySelectorAll("input")

Retrieves an array of <input>element

Accessing Radio Button and Checkbox Groups

For radio button and check box groups you will need to check for individual value of each item.

The radio button and check box will need to have the same name attribute value.

var box = document.getElementsByName("xxxx");

for (let x of box) {

var y = x.checked;

....

}

Download and complete w5\_arithmetic.html