INT222 - Internet Fundamentals

Week 8: Introduction to JavaScript

Agenda

- What is JavaScript?
- Variables and Data Types
- Document Object Model (DOM)
- Functions
- JavaScript Statements

What is JavaScript?

- JavaScript (sometimes shortened to JS) is a lightweight, interpreted, high-level language used along with html code.

- The language syntax is somewhat similar but not the same as the C language. Today, JavaScript is the scripting language for Web pages.

JavaScript Features

- an interpreted language interprets and executes each statement - one-by-one - in the order they appear.
- always runs inside a host environment (mostly the browser) and integrates with its HTML/CSS content.
  - e.g. Firefox Scratchpad
- relaxed about syntax, rules, and types
  - fewer and "looser" data types
  - variables don't need to be declared
  - errors often silent (few exceptions)
- key construct is the function rather than the class
- JS focuses on user interfaces and interacting with a document
- a powerful prototypal object-oriented language

The Use of JavaScript

- JS is used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - react to events (e.g. page load user click)
  - get information about a user's computer (e.g. browser type)
  - perform calculations on user's computer (e.g. form validation)

The History of JavaScript

- JavaScript was developed in 1995 by Brendan Eich at Netscape.
- JavaScript started with simple one-liners embedded in HTML.
- The JavaScript backlash (incompatibility, …) caused some web projects to completely ban any client-side programming and trust only their predictable and reliable server.
- ECMAScript (European Computer Manufacturers Association) was born, becoming the standard of JavaScript.
  - As of 2012, all modern browsers fully support ECMAScript 5.1.
- JavaScript is now used in much more sophisticated ways.
  - Developers leverage the object-oriented nature of the language to build scalable code architectures made up of reusable pieces.
  - JavaScript provides behavior, the third pillar in today's paradigm
    - web pages consist of three clearly distinguishable parts: content (HTML), presentation (CSS), and behavior (JavaScript).

Using JavaScript with HTML

- The ways to use JavaScript in an HTML document:
  1. use the <script> and </script> tags to include JavaScript code directly into an HTML file – internal JavaScript code.
  2. use the <script> and </script> tags to include a separate external JavaScript file into an HTML file – external JavaScript code.
     - The external file will only contain JavaScript statements and must have a .js extension.
     - e.g. <script src="myscript.js"></script>

Using JavaScript with HTML

3. The original way of using JavaScript – Inline (embedded) JavaScript code.

   - e.g.

     ```
     <body onLoad="alert('The document has been loaded.');>
     ```

Where to Add JavaScript Code

- Four (4) very basic javaScript examples:
  - JavaScript embedded in the <head>...</head> part of an html document
  - JavaScript embedded in the <body>...</body> part of an html document
  - External JavaScript load in the <head>...</head> part of an html document
  - External JavaScript load in the <body>...</body> part of an html document
- Browsers display HTML in a "top-down" fashion, so the execution of the javascript code is determined by its location in the html code.

Data Type in JavaScript

There are 3 main data types:

- strings
  - must be enclosed in single or double quotes
- numbers
  - can be integers or floating point
- boolean
  - values are binary, with the values (1) "true" and (0) "false" (without the quotes)

Special values

- Infinity
  - Number Data Type
- NaN
  - Number Data Type
- null
  - both a value and a data type
- undefined
  - both a value and a data type

Variables

- Variable naming rules are:
  - Must start with a letter, underscore (_), or dollar sign ($)
  - Cannot be a reserved (key) word
  - Subsequent characters can be letters - upper case (A...Z) or lower case (a...z), numbers or underscores

Declaring Variables

- JS is a loosely typed language

Declaration	Type	Value
var identOne = "some text";	String	some text
var identone = 'some text';	String	some text
var IdentOne = '172';	String	172
var _identOne = 25;	Number (Integer)	25
var _identTwo = 56.2564;	Number (float)	56.2564
var ident_A = true;	Boolean	true (1)
var ident_B = false;	Boolean	false (0)
var ident_C;	undefined	undefined
var ident_D="Yes", ident_E="No";	String / String	Yes / No

Variable Scope

- In JavaScript, variable scope can be global or local. Scope is determined by where and how a variable is declared.

1. Global variable
   A variable that is declared outside any functions is global. A global variable can be referenced anywhere in the current document.
   - Declared outside any functions, with or without the var keyword
   - Declared inside a function without using the var keyword, but only after the function has been called once

Variable Scope

2. Local variable
   A variable that is declared inside a function is local. A local variable can only be referenced inside the function it is declared in.
   - Declared in a function with the var keyword.
   - If you reference a local variable globally or in another function, JavaScript will trigger the "is not defined" error.
     - this is different error from the "**undefined**" JavaScript error for a variable that is not initialized.

Example

```
var display = "";      // Global variable
ident_A = 5;           // Global variable - bad practice

function someFunction() {   // Start of function

    var ident_B = 15;      // Local  variable
    ident_C     = 34;       // Global variable - bad practice
    var ident_A = 0;
    ident_C++;                // increment ident_C by 1
    ident_A     = ident_B + ident_C;
    alert(ident_A);         // show the value of ident_A inside the function

} // End of function

someFunction();     // call the function
alert(ident_A);       // show the value of ident_A outside the function
alert(ident_C);       // show the value of ident_C
alert(ident_B);       // what happens here?
```

About Variable Declaration

- It is recommended that you
  - avoid using global variables.
  - always use the var keyword when declaring variables.
- Notes
  - Functions are the only construct that can be used to limit scope of variables.
  - In JavaScript, Code blocks do not determine variable scope.

Examples

Local Block in C	Block scope in JavaScript

#include <stdio.h>
 	
int main()
{
int x = 10;
{
int x = 30;
printf(%d ”, x);
}
printf(“%d”, x);
}
	var a = 10;
	{
	var a = 30;
	b= 20;
	}
	for (var i = 0; i < 5; i++) {
	var c = i;
	}
	alert(a);
	alert(b);
	alert(c);

Output: 30 10	Output?

JavaScript Object

- Javascript is referred to it as an object-based, event drive scripting language.
  - The object-based means that Javascript works with items known as objects.
- Objects are things in a browser, a window, a form field, a document, a submit button, etc.
  - In JavaScript, almost everything is an object. All primitive types except null and undefined are treated as objects.
- An object is just a special kind of data, with properties and methods.

Object Properties

- A JavaScript object has properties associated with it.
- A property of an object can be explained as a variable that is attached to the object.
- Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects.
- The properties of an object define the states of the object.
- You access the properties of an object with a simple dot-notation:
  - objectName.propertyName

Object Methods

- Objects have methods which are really tasks and/or functions that the object performs.
  - Similar to a function or subroutine or procedure, a method has a name and parameters
- Methods define the behaviour of an objects.
- Methods may change an object's properties, thus changing its state.
  - methods may be called using dot-notation. e.g. objectName.methodName(para1, para2, ...)

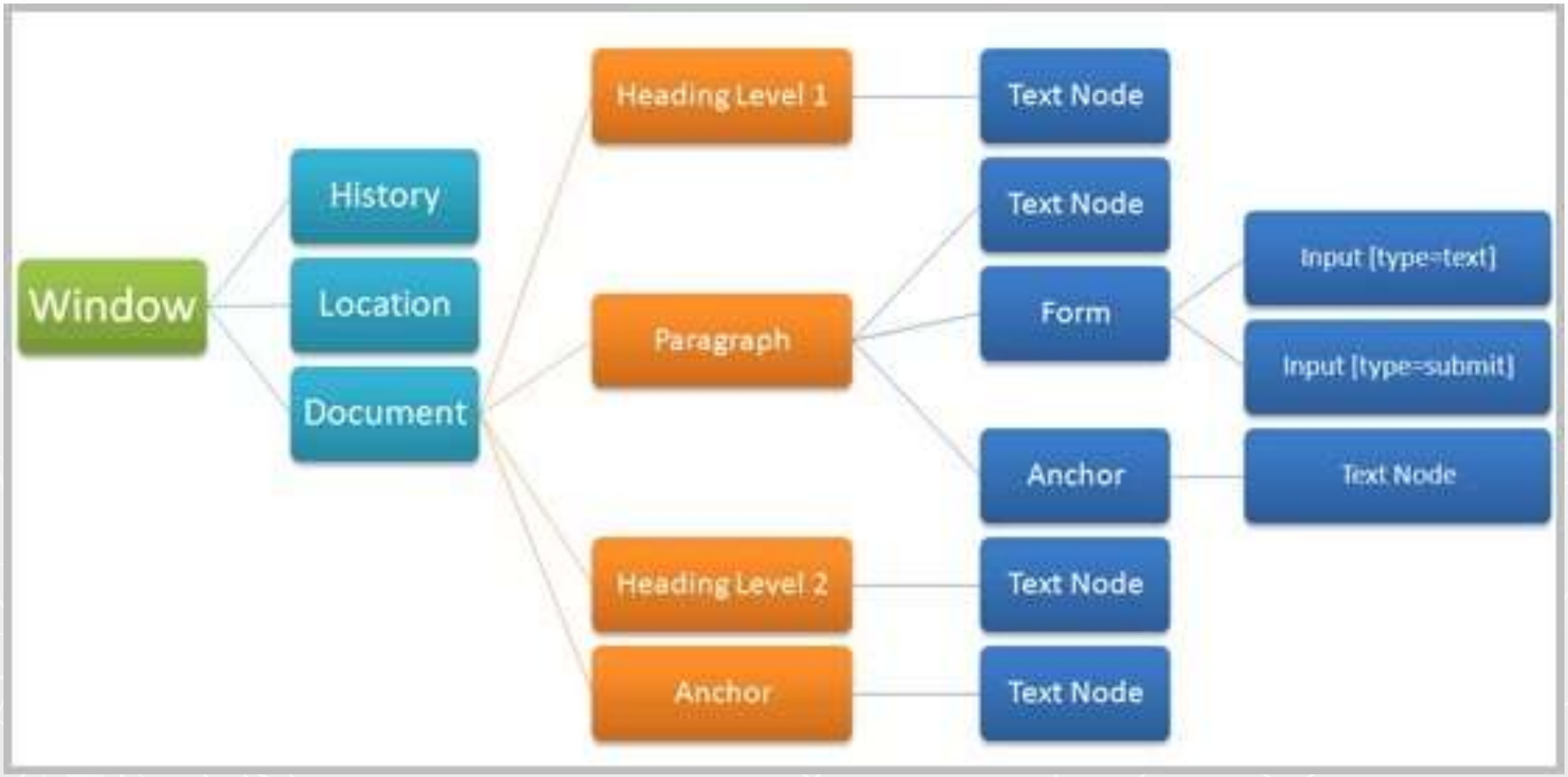
Objects are Hierarchical

- A property of an object may itself be an object
- There is a parent-child relationship.
- An object is a parent if it has a property which is an object.
- The property which is an object is a child of that parent.
- These relationships are often thought of as an upside-down tree structure, with the root at the top being the ancestor of all the descendants below.

The DOM (Document Object Model)

- The DOM is a programming interface for HTML (and XML) documents.
  - It provides a structured representation of the document (a tree), and it defines a way that the structure can be accessed from programs so that they can change the document structure, style and content.
- Modern web browsers use an object-oriented paradigm:
  - where the top-level object is the window object
  - the window object contains 3 main objects: document, location and history.

DOM Tree



http://www.sitepoint.com/forums/showthread.php?627986-What-exactly-is-the-DOM

Document Object

- Document object contains information about the currently displayed document.
- Document object properties include:
  - domain: name of the server of the current document, read-only
  - URL: the url of the current document, read-only
  - referrer: the url of the previous document, read-only
  - title: defined in <title> tags, read-write)

More Document Object Properties

- bgColor (background color, read-write)
- fgColor (text color, read-write)
- lists of links
- lists of forms
- lists of images

Document Object Methods

- document.write()
  - OR: window.document.write()
- document.writeln()
- document.getElementById()
  - The getElementById() can be used to retrieve an element by its "id"
  - The innerHTML property of an element can be used to change its contents

Location Object

- Location object properties include:
  - href : complete URL
    - e.g. redirect a webpage:

```
<a href="javascript:void(0)"
    onclick="location.href = prompt('Please enter URL',
    'https://scs.senecac.on.ca');">Load a new page</a>
```

  - hostname :other parts of the URL, stored separately
- All location object properties can be changed.

History Object

- History object contains the list of previously visited URLs.
- History object methods include:
  - back() - same as browser "Back" button
  - forward() - same as browser "Forward" button
  - go(n)
    where (n) is a positive or negative integer negative number will move back, positive number will move forward in the history list

Functions

- A function is a "subprogram" that can be *called* by code external (or internal in the case of recursion) to the function.
- Like the program itself, a function is composed of a sequence of statements - *function body*.
- Values can be *passed* to a function, and the function can *return* a value.

Functions

- A function is a method for the window object
- A function is not executed until it is called
- Functions are declared using the "function" keyword
- Function names must adhere to variable name rules
- Function declaration:

```
function functionName ( parameter1, parameter2, …) { functionBody }
```

- In JavaSript, function is object – function statement:

```
var functionName = function(parameter1, parameter2, …) {
        functionBody} ;
```

More about Functions

- Functions are usually declared in the <head> element of the html document, to ensure that the function is defined to the browser before any activation by an event handler.
- Functions may be called within other functions.
- Functions are executed when they are called, not in the order in which they are declared.

Example

```
<head>
  <meta charset="utf-8">
  <title>INT222</title>
  <script>
     //Here is the function declaration
       function myFirstFunction() {
           alert("Hello out there");
       }  //End of myFirstFunction
  </script>
</head>
<body>
     Here's some HTML code just for starters
     <br />
     <script>
        myFirstFunction();  //This is a function call
     </script>
     Here's some more HTML code just for show
...
 </body>
```

click here to see how this displays in a browser

click same as above with an external JavaScript

Parameters

- Parameters are also referred to as arguments
- Parameters are used to pass values to functions
- Multiple parameters can be used within each function

An Example

```
<head>
  <meta charset="utf-8">
  <title>INT222</title>
  <script>
        function greetings(first,last) {
           document.write("Hello " + first + " " + last);
        }
  </script>
</head>
<body>
  This page shows how parameters are passed to JavaScript functions ...
  <br />   <br />
  <script>
        var firstName = prompt("Enter your first name", "");
        var lastName = prompt("Enter your last name", "");

        greetings(firstName, lastName);
  </script>
 </body>
```

Try it out

An Other Example

- The return value from a function can be used to directly pass a value to another function.

```
<head>
  <meta charset="utf-8">
  <title>INT222</title>
  <script>
        function greetings(name) {
           document.write("Hello " + name);
        }
  </script>
</head>
<body>
     <mark>This page shows how parameters are passed in a slightly different way
...</mark>
     <br /><br />
     <script>
       greetings(prompt("Please enter your name"))
     </script>
</body>
```

Try it out

Two Types of Functions

- Custom functions / user-defined functions.
  - Return value is optional. Return may only exit the function.

- JS built-in functions/ global functions
  - are the methods of the window object.

User-defined Functions

- Example

```
function addTwoNumbers(a, b) {return a + b;}
var add2numbers = function(a, b){return a + b;};
function addNumbers() {
   var sum = 0;
   for (var i=0; i<arguments.length; i++) {
                sum += arguments[i];
   }
   return sum;
}
alert( addTwoNumbers(2, 3) ); // 5
alert( add2numbers(2, 4) );   // 6
alert( addNumbers(2, 6, 8) ); //16
```

JS Built-in / Global Functions

- They are built into the JavaScript language.
  - **Common window object methods** (Methods of the window object)
- They have already been defined and the logic behind them has already been coded for you to use.
  - alert() or window.alert(), confirm(), prompt()
  - parseInt(), parseFloat()
  - Number(), String()
  - isNaN(), inFinite(), eval(),
  - Open(), close(), Focus(), scrollTo(), …

JS Modal Dialog Windows

- window.alert()

```
<head>
  <meta charset="utf-8">
  <title>INT222</title>
 </head>
 <body>
   <script>
     alert("some text");  // alert a string
     var info_1 ="more text";
     var info_2 = 15;
     var info_3 = 55;
     alert("This is using a string  variable: " + info_1);
     alert("This is using a number variable: " + info_2 );
     alert("This is using two number variables: " + info_2 +  info_3);
     alert("This is using two number variables: " + eval(info_2 + info_3));
     alert(info_2 +  info_3);
   </script>
 </body>
```

try it out

JS Modal Dialog Windows

- window.confirm()

```
<head>
  <meta charset="utf-8">
  <title>INT222</title>
 </head>
 <body>
  <script>
    var decision = confirm("Last chance. Are you sure you want to leave?");

     if (decision) {   // they pressed OK
       location.replace("http://www.cnn.com");
     }
     else {  // pressed cancel
       alert("I'm glad you are staying.")
     }
  </script>
 </body>                                                    //  Try it out
```

JS Modal Dialog Windows

- window.prompt()

```
<head>
   <meta charset="utf-8">
   <title>INT222</title>
 </head>
 <body>
   <script>
      var info_1 = prompt("shows in the dialogue box - required",
"default - optional");
      var info_2 = prompt("enter something", "");
      var info_3 = prompt("enter something");

     </script>
 </body>                                                    // Try it
out
```

JavaScript statement

- A JavaScript typically consists of a series of statements.
- A statement is a single line of instruction to the computer made up of objects, expressions, variables, and events /event handlers.
- Every statement has the same structure:
  - A statement starts on its own line (recommended but not required)
  - The first item in a statement is a command
  - The second part following the command is some information about that command
  - The last part of a statement is a terminating semi-colon;

JavaScript Execution Flow

- The four (4) basic control structures:
  - Sequential - an instruction is executed when the previous one is finished
  - Conditional - a logical condition is used to determine which instruction will be executed next - similar to the "if" and "switch" statements in C
  - Looping - a series of instructions are repeatedly executed until some condition is satisfied - similar to the "for" and "while" statements in C
  - Transfer - jump to a different part of the code - similar to calling a function in C

Programming Constructs (1) – Sequence

```
var a = 3;
var b = 6;
var c = a + b;
alert(c);
```

Programming Constructs (2) – Selection

- Make decisions and perform single or multiple tasks based on the outcome of the decision (true or false).
- Types of conditional statements :
  - ***if***
  - ***if / else***
  - ***switch / case***

Conditional Statements

- Conditional Statements give JavaScript scripts the ability to make decisions and perform single or multiple tasks based on the outcome of the decision (true or false).
- **The if-else**
  - expression / condition is in parentheses (expression)
  - relational operators include: == != > < >= <=
  - && (and) and || (or) can be used to create compound conditions
  - ! (not) can be used to invert a condition
  - the else clause is optional
  - if statements may be nested
  - multiple action statements must be enclosed in brace brackets { }

The General Format for an If Statement

```
if (expression)
      statement;
```

```
if (expression) {
    statement;
    statement;
    statement;
 }
```

The general format for an if / else statement is as follows:

```
if (expression) {
      statement;
}
else {
   statement;
}
```

```
if (expression) {
        statement;
        statement;
        statement;
}
                else {
        statement;
        statement;
  }
```

The general format for an **if / else / if** statement:

```
if (expression-a) {
        statement1;                    /* If expression-a is True
}
else {
        if (expression-b) {
            statement2;            /* If expression-b is True
        }
        else {
            statement3;            /* If expression-b is False
        }
} // end of the if statement
```

or

```
if (expression-a) {
        statement1;  }              /* If expression-a is True
else if (expression-b) {
        statement2; }               /* If expression-b is True
else {  statement3; }               /* If expression-b is False
```

```
if (mark>=90)
  {
     grade = 'A+';
  }
else if ((mark<90)&&(mark>80))
  {
     grade = 'A';
  }
else
  {
     grade = 'B';
  }
```

expression-a	expression-b	Action
true	true	statement1
true	false	statement1
false	true	statement2
false	false	statement

Conditional Statement

- The switch / case statement:
  - select one of many blocks of code to be executed.

```
switch (expression)
{
   case  401:
            statement1;
            break;
   case  403:
            statement2;
            break;
   case  407:
            statement3;
            break;
   default:
            statement4;
}
```

Evaluation the above expression

- The value in the first case (401) is compared to the result.
  - if the comparison outcome is true, statement1 is executed
    control is passed to the end of the switch
- The value in the second case (403) is compared to the result.
  - if the comparison outcome is true, statement2 is executed
    control is passed to the end of the switch.
- and so on....
- The default action (statement4) is executed if non of the case comparisons are true.

Programming Constructs (3) – Iteration

- loop - an action that occurs again and again until a certain condition is met.
- Continuously check a condition and based on the outcome, either terminate the loop or repeat a set of statements.
- Three basic types of loop structures:
  - The for loop
  - The for / in loop
  - The while loop
  - The do-while loop

The *for* loop

```
for (initialExp ; condition ; updateExp) {  « first part
        statement(s)                        « second part
}
```

- The *initialExp* is a variable which defines the initial starting value for the loop.
  - The variable can be defined within the loop statement or by a separate variable declaration prior to the loop statement.
- The *condition* is the section in which you'll use an operator (usually a comparison operator) to provide the number of times to iterate through the loop.
- The *updateExp* is usually used to adjust the *initialExp*. The adjustment may be any increment, decrement, or any other operation you may need to use.
- The statement(s) are the action that may be repeated.

The *for* loop

- How it works:
  1. the initialExp is set (once)
  2. the condition is checked:
     - if the condition is true, the action is taken
       - the updateExp is executed
       - back to step 2 (check the condition)
     - if the condition is false, exit the *for* loop
- Make sure that the condition being checked to terminate the loop is met at one point
  - otherwise your script will be in an infinite loop.

The *for* loop example 1

```
document.write("<table>");
document.write(" <caption>2 times table</caption>");
for (var ident = 1 ; ident <= 12 ; ident++) {
   document.write(" <tr>");
   document.write("  <td>2</td>");
   document.write("  ..........");
   document.write(" </tr>");
}
document.write("</table>");
```

Try it out

The *for* loop example 2

```
var ident;
document.write("<table>");
document.write(" <caption>2 times table</caption>");
for (ident = 1 ; ident <= 12 ; ident++) {
   document.write(" <tr>");
   document.write("  <td>2</td>");
   document.write("  ..........");
   document.write(" </tr>");
}
document.write("</table>")
```

Try it out

The *for* loop example 3

```
var htmlCode="";

htmlCode += "<table>";
htmlCode += " <caption>2 times table</caption>";

for (var ident = 1 ; ident <= 12 ; ident++) {
   htmlCode += " <tr>";
   htmlCode += "  <td>2</td>";
   htmlCode += "  ..........";
   htmlCode += " </tr>";
}
htmlCode += "</table>";
document.write(htmlCode);          // Try it out
```

The *for in* loop

- Iterates over the enumerable properties of an object, in arbitrary order. For each distinct property, statements can be executed.

```
var student = {name:"John", program:"CPD", semester:2};

for (var x in student)
{
  alert(x + ": " + student[x]);
}
```

The *while* loop

- The while loop loops through a block of code while a specified condition is true.
- The while loop tests the supplied condition *first* and continues to execute the action statement(s) until the condition is met.
- Here is the syntax for the while loop:

```
while (condition) {
        action statement(s)
}
```

The *while* loop

- How it works:
- the condition is checked:
  - if the condition is true
    the action is taken
    back to step 1 (check the condition)
  - if the condition is false
    exit the while loop
- Again, make sure that the condition being checked to terminate the loop is met at one point - otherwise your script will be in an infinite loop.

An Example of a while Loop

```
var ident = 1;
document.write("<table");
document.write(" <caption>2 times table</caption>");
while (ident <= 12) {
   document.write(" <tr>");
   document.write("  <td>2</td>");
   document.write("  .....");
   document.write(" </tr>");
   ident++;
}
document.write("</table>");
```

Try in out

The do...while Loop

- A variant of the *while* loop.
- Execute the block of code at least ONCE, and then it will repeat the loop as long as the specified condition is true.
- Here is the syntax for the do while loop:

```
do {
        action statement;
        action statement;
        action statement;
} while (condition);
```

- How it works:
  1. the action is taken
  2. the condition is checked:
     - if the condition is true
       the action is taken
       back to step 2 (check the condition)
     - if the condition is false
       exit the do while loop
- Again, make sure that the condition being checked to terminate the loop is met at one point - otherwise your script will be in an infinite loop.

An example of a while loop

```
var ident = 1;
document.write("<table");
document.write(" <caption>2 times table</caption>");

do {
   document.write(" <tr align='center'>");
   document.write("  <td>2</td>");
   document.write("  ..........");
   document.write(" </tr>");
   ident++;
} while (ident <= 12);

document.write("</table>");
```

Try in out

Break Statements

- Break the loop and continue executing the code that follows after the loop (if any).

```
var i=1;
while (i<5)
{
   alert("week "+i);
   if (i==3)
            break;
   else
           i++;
}
```

Break Statements

- Break the current loop and continue with the next value. (nested loops).

```
var i=1;
var j=1;
while (i<5)
{
   alert('week: '+i );
   for (j=1; j<=7; j++)
   {
            alert('day:'+j +'of week:'+ i);
             if (j==3)
            break;
   } // for
   i++;
} // while
```

Conditional expression

- A conditional expression can have one of two values based on a condition. The syntax:

  (condition) ? val1 : val2;

- If the condition is true, the expression has the value of val1, Otherwise it has the value of val2.

	condition	When True	When False

```
var status = (age >= 18) ? "adult" : "minor";
```

Resourceful Links

- (JS) Statement (MDN)

  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements

- (JS)Values, variables, and literals (MDN)

  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Values,_variables,_and_literals#Integers

- (JS) Functions and function scope(MDN)

  https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions_and_function_scope

- Introduction to DOM (MDN)

  https://developer.mozilla.org/en-US/docs/DOM/DOM_Reference/Introduction#Core_Interfaces_in_the_DOM

Thank You!