Web Design Course

JAVASCRIPT BASICS

Lecture by Mr. Kaung Sett

IU #	IU Description	Required / Optional
01	Web Design Concepts	Required
02	HTML Basics	Required
03	Advanced HTML & Web Browsers	Required
04	Structuring & Styling with CSS	Required
05	Working with CSS : An Example	Required
06	**Javascript Basics**	**Required**
07	Advanced Javascript	Required

IU Contents

S. No.	Topic Description
01	JavaScript Introduction
02	JavaScript Values
03	Variables
04	Operators
05	Type Conversions
06	Arrays
07	Conditional Statements
08	Objects
09	Loop Statements

- JavaScript is a dynamic computer programming language.
- It is lightweight and most commonly used as a part of web pages
- allow client-side script to interact with the user and make dynamic pages.
- It is an interpreted programming language with object-oriented capabilities.
- It is designed for creating network-centric applications.
- Open and cross-platform

- **Less server interaction**
  - validate user input before sending the page off to the server.
  - saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors**
  - Users don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity**
  - can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces**
  - can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

- cannot treat JavaScript as a full-fledged programming language.
- has the following important limitations −
  - ¶ cannot read or write the file using Client-side JavaScript . This has been kept for security reason.
  - ¶ cannot be used for networking applications because there is no such support available.
  - ¶ doesn't have any multithreading or multiprocessor capabilities.

❑ The <script> Tag

- JavaScript can be implemented by placing JavaScript statements within the **<script>... </script>** HTML tags in a web page.
- You can place the **<script>** tags, containing your JavaScript, anywhere within you web page, but it is better to keep it within the **<head>** tags.

❑ The script tag takes two important attributes –

- **Language** – This attribute specifies what scripting language you are using. Typically, its value will be JavaScript.
- **Type** – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/JavaScript".

```
<script language="javascript" type="text/javascript">
JavaScript code
 </script>
```

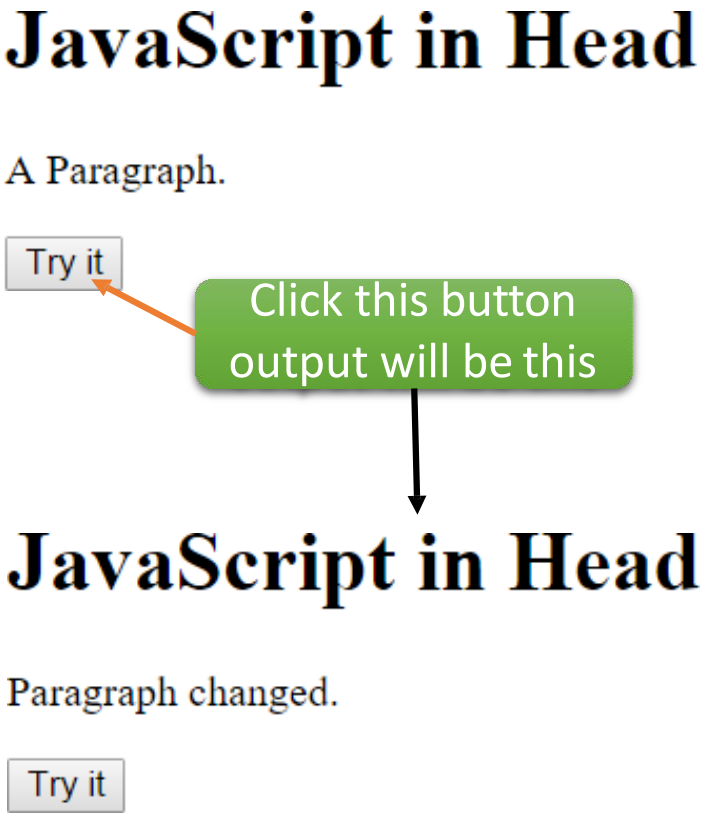
❑ JavaScript in <head>

- In this example, a JavaScript function is placed in the <head> section of an HTML page.
- The function is invoked (called) when a button is clicked.

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
   document.getElementById("demo").innerHTML =
"Paragraph changed.";
}
</script></head>
<body>
<h1>JavaScript in Head</h1>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try
it</button>
</body>
</html>
```

o/p



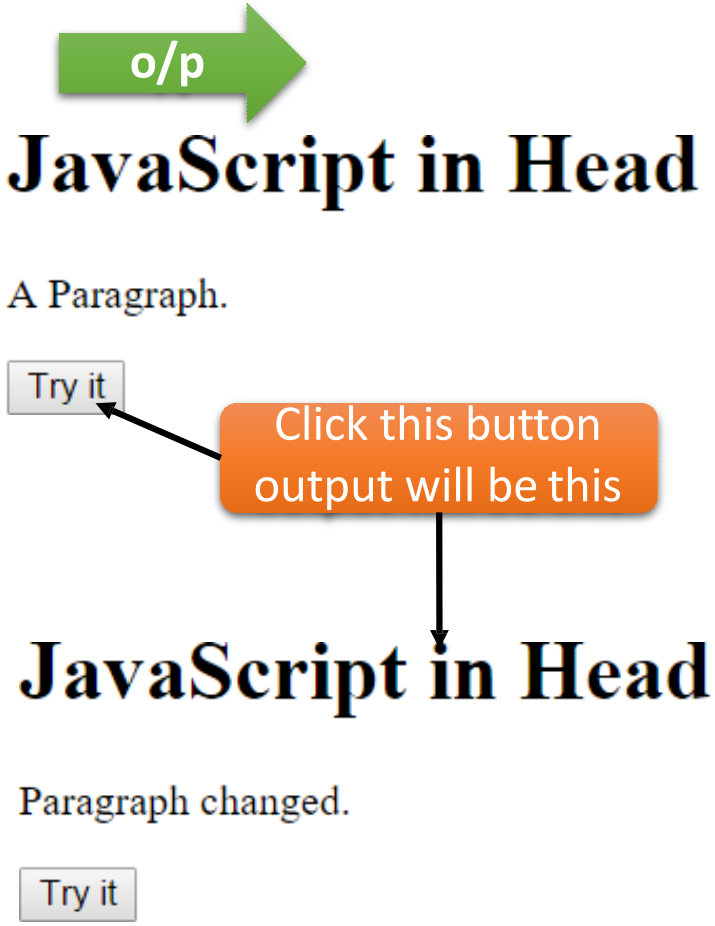
❑ External JavaScript

- Scripts can also be placed in external files.
- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the **file extension .js**.

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>External JavaScript</h1>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try
it</button>
<p><strong>Note:</strong> myFunction is stored in
an external file called "myScript.js".</p>
<script src="myScript.js"></script>
</body>
</html>
```

o/p



JavaScript Datatypes

- important characteristics of a programming language is the set of data types it supports.
- are the type of values that can be represented and manipulated in a programming language.
- supports following three primitive data types −
  - **Numbers,** eg. 123, 120.50 etc.
  - **Strings** of text e.g. "This text string" etc.
  - **Boolean** e.g. true or false.
- also supports two trivial data types, **null** and **undefined,** each of which defines only a single value and composite data type known as **object**.

❑ JavaScript Variables

- Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.
- Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
<script type="text/javascript
 <!--
    var money;
    var name;
  //-->
</script>
```

```
<script type="text/javascript">
<!—
 var money, name;
 //-->
</script>
```

You can also declare multiple variables with the same **var** keyword

❑ JavaScript Variable Scope

- The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.
  - ¶ **Global Variables** − A global variable has global scope which means it can be defined anywhere in your JavaScript code.
  - ¶ **Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.
- If you declare a local variable or function parameter with the same name as a global variable you effectively hide the global variable.

```
<html>  <body onload = checkscope();><script type =
"text/javascript">
      <!--
        var myVar = "global"; // Declare a global variable
        function checkscope( ) {
          var myVar = "local"; // Declare a local variable
          document.write(myVar);
        }
      //--></script></body></html>
```

local

❑ JavaScript Variable Names

- While naming your variables in JavaScript, keep the following rules in mind.
  - should not use any of the JavaScript reserved keywords as a variable name. For example, **break** or **boolean** variable names are not valid.
  - variable names should not start with a numeral (0-9). They must start with a letter or an underscore character. For example,**123test** is an invalid variable name but **_123test** is a valid one.
  - variable names are case-sensitive. For example, **Name** and **name** are two different variables.

❑ JavaScript Variable Names

- A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

❑ JavaScript Values

- Creating a variable in JavaScript is called "declaring" a variable.
- You declare a JavaScript variable with the var keyword:

```
var carName;
```

- After the declaration variable has no value.
- To assign value to a variable use the following syntax:

```
carName = " Volvo ";
```

- You can also assign a value to the variable when you declare it:

```
var carName = " Volvo ";
```

❑ JavaScript Values Example

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Variables</h1>
<p>Create a variable, assign a value to it, and display it:</p>
<p id="demo"></p>
<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
</body>
</html>
```

Result

JavaScript Variables

Create a variable, assign a value to it, and display it:

Volvo

- One Statement, Many Variables
  - can declare many variables in one statement.
  - Start the statement with var and separate the variables by comma:

var person=" John Doe ", carName = " Volvo", price = 200;

❑ JavaScript Operators

- **What is an operator?**
  - ¶ Let us take a simple expression 2+5 is equal to 7. Here 2 and 5 are called operands and '+' is called the operator.

- **JavaScript supports the following types of operators.**
  - ¶ Arithmetic Operators
  - ¶ Comparison Operators
  - ¶ Logical (or Relational) Operators
  - ¶ Assignment Operators
  - ¶ Conditional (or ternary) Operators

❑ JavaScript Arithmetic Operators

- Arithmetic operators are used to perform arithmetic on numbers

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

❑ JavaScript Comparison Operators

- JavaScript supports the following comparison operators –

Operator	Description	Example
= =	**Equal**	A == B
!=	Not Equal	A !=B
>	**Greater than**	A > B
<	Less than	A<B
>=	**Greater than or Equal to**	A >= B
<=	**Less than or Equal to**	A <= B

❑ JavaScript Logical Operators

- JavaScript supports the following logical operators −

Operator	Description	Example
&&	Logical AND	A && B
||	Logical OR	A || B
!	Logical NOT	A && B

❑ JavaScript Assignment Operators

- Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	X = y	X = y
+ =	X += y	X=X + y
- =	X -= y	X=X – y
* =	X *= y	X=X * y
/ =	X /= y	X=X / y
% =	X %= y	X=X % y

❑ JavaScript Assignment Operators

- supports the following bitwise operators –

Operator	Description	Example
&	Bitwise AND	A & B
|	BitWise OR	A | B
^	Bitwise XOR	A ^ B
~	Bitwise Not	~B
<<	Left Shift	A << 1
>>	Right Shift	A >> 1
>>>	Right shift with Zero	A >>> 1

- The conditional operator (? :) and the typeof operator.−
  - The conditional operator first evaluates an expression for a true or false value and then depending upon the result of the evaluation executes one of the two given statements.

Operator and Description

? : (Conditional)

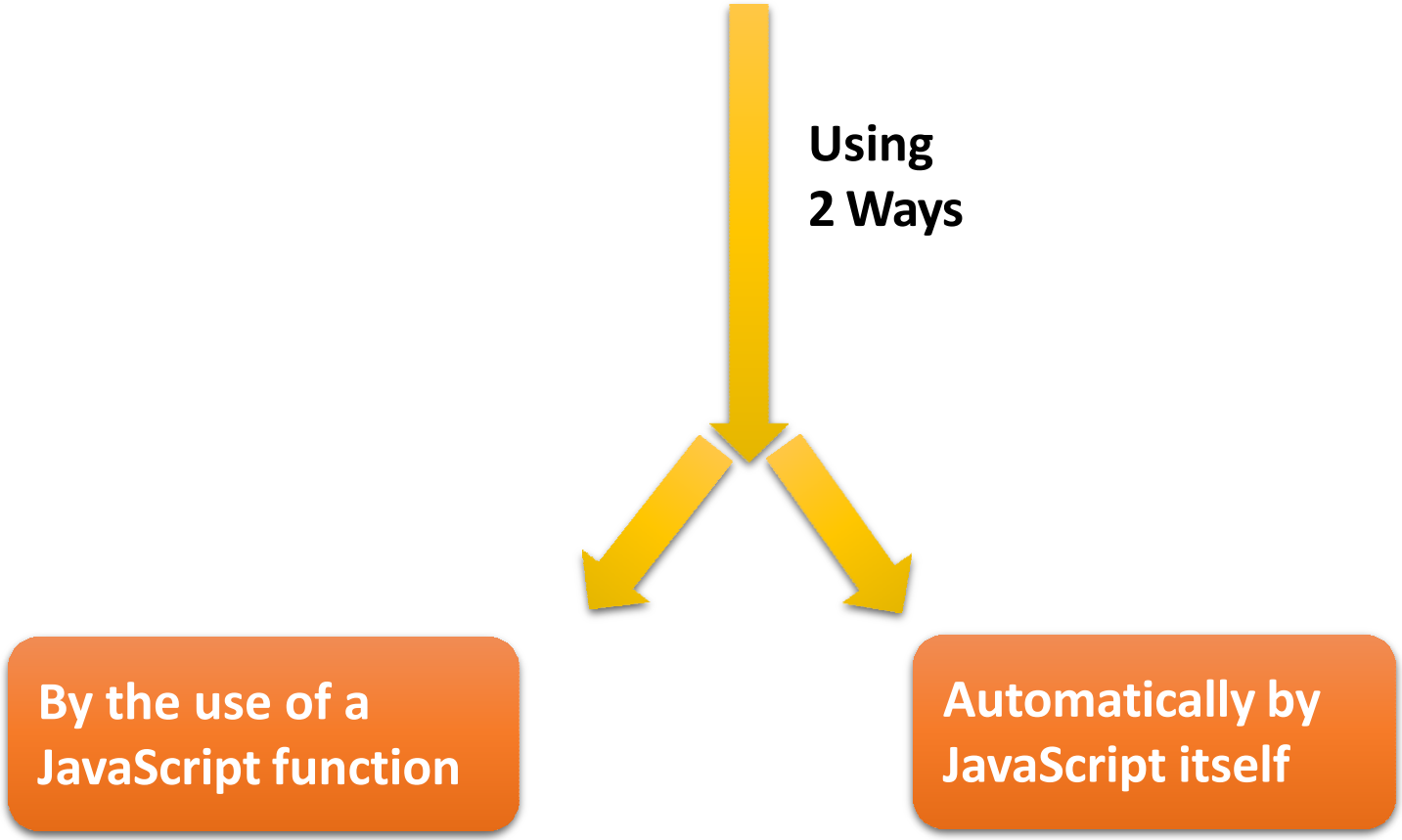
If Condition is true? Then value x : Otherwise value y

❑ JavaScript typeof Operator

- is a unary operator that appears before its single operand, which can be of any type.
- Its value is a string indicating the data type of the operand.
- evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.
- Following is the list of return values of typeof operator

Type	String Returned by typeof
Number	“number””
String	“string”
Boolean	“boolean”
Object	“object”
Function	“function”
Undefined	“undefined”
Null	“object”

- **JavaScript variables can be converted to a new variable and another data type:**



❑ Converting Numbers to Strings

- The global method **String()** can convert numbers to strings.
- It can be used on any type of numbers, literals, variables, or expressions:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>The String() method can convert a number to a string.</p>

<p id="demo"></p>

<script>
var x = 123;
document.getElementById("demo").innerHTML =
  String(x) + "<br>" +
  String(123) + "<br>" +
  String(100 + 23);
</script>
</body>
</html>
```

Result

The String() method can convert a number to a string

123
123
123

❑ Converting Booleans to Strings

- The global method String() can convert boolean to string.

```
String(false)          //returns “false”
String(true)           //returns  “true”
```

- The Boolean method toString() does the same.

```
false.toString()       //returns  “false”
True.toString()        //returns    “true”
```

❑ Converting Dates to Strings

- The global method String() can convert dates to strings.

```
String(Date())          //returns Thu Jul 17 2015 12:30:35 GMT+0200
(w.Europe Daylight Time)
```

- The Date method toString() does the same.

```
Date().toString()          //returns Thu Jul 17 2015 12:30:35
GMT+0200 (w.Europe Daylight Time)
```

❑ Converting Strings to Numbers

- The global method Number() can convert strings to numbers.
- Strings containing numbers (like "3.14") convert to numbers (like 3.14).
- Empty strings convert to 0.
- Anything else converts to NaN (Not a number).

```
Number(”3.14”)          //returns  3.14

Number(“    “)          //returns    0

Number(“ “)             //returns    0

Number(“99  88“)        //returns      NaN
```

The Unary + Operator

- The **unary + operator** can be used to convert a variable to a number:

```
var y = "4";          // y is string
var x = +y;           // x is a number
```

- If the variable cannot be converted, it will still become a number, but with the value NaN (Not a number):

```
var y = "Johny";      // y is string
var x = +y;           // x is a number (NaN)
```

❑ Converting Booleans to Numbers

- The global method **Number()** can also convert booleans to numbers.

```
Number(false)        // returns 0
Number(true)         // returns  1
```

❑ Converting Dates to Numbers

- The global method **Number()** can be used to convert dates to numbers.

```
e = new Date();      // returns  1404568027739
Number(e)
```

- The date method getTime() does the same.

```
e = new Date();      // returns  1404568027739
e.getTime()
```

❑ Automatic Type Conversion

- When JavaScript tries to operate on a "wrong" data type, it will try to convert the value to a "right" type.

```
4 + null          //returns    4          because null is converted to 0
“4” + null        //returns  “4null”      because null is converted to “null”
“4” + 2           //returns    42         because 2 is converted to “2”
“4” -  2          //returns    2          because “4” is converted to 4
“4” * “2”         //returns    8          because “4” and “2” are converted into 4 and 2
```

❑ Automatic String Conversion

- JavaScript automatically calls the variable's toString() function when you try to "output" an object or a variable:

```
Document.getElementById(“demo”).innerHTML = mycar;

// if myVar = {name:”Jhony”}      //toString converts to “[object object]”
// if myVar = [1,2,3,4]           //toString converts to “1,2,3,4”
// if myVar = new Date()          //toString converts to “Fri Jul 17 2015
                                    10:12:15 GMT+0200”
```

- Numbers and booleans are also converted, but this is not very visible:

```
// if myVar = 123          // toString converts to “123”
// if myVar = true         // toString converts to “true”
// if myVar = false        // toString converts to “false”
```

What is an Array?

- is a special variable, which can hold more than one value at a time.
- stores a fixed-size sequential collection of elements of the same type.
- If have a list of items (a list of car names, for example), and storing a cars in a single variable could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
- The solution is an array!

❑ Creating an Array

- Using an array literal is the easiest way to create a JavaScript Array.

  Syntax:

```
var array-name = [item1, item2,…….];
```

  Example:

```
var cars = [“Maruti” ,”BMW”];
```

- **Using the JavaScript Keyword new**
  - ¶ The following example also creates an Array, and assigns values to it:

  Example:

```
var cars = new array(“Maruti” ,”BMW”);
```

❑ Access the Elements of an Array

- You refer to an array element by referring to the index number.
- This statement accesses the value of the first element in cars:

```
var name = cars[0];
```

- This statement modifies the first element in cars:

```
cars[0] = “Maruti”;
```

- [0] is the element in array.[1] is the second. Array indexes start with 0.

❑ Arrays are Objects

- Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.
- But, JavaScript arrays are best described as arrays.
- Arrays use **numbers** to access its "elements". In this example, person[0] returns John:

Array :

```
var person = [“Johny” , “Doey” , 45];
```

Object:

```
var person = {first name:“Johny” , lastname:“Doey” ,age= 45}
```

❑ Associative Arrays

- Many programming languages support arrays with named indexes.
- Arrays with named indexes are called associative arrays (or hashes).
- JavaScript does **not** support arrays with named indexes.
- In JavaScript, **arrays** always use **numbered indexes**.

```
var person = [ ];
person[0] = “Johny”
person[1] = “Doey”
person[2] = “45”
var x = person.length;          //person.length will return 3
var y = person[0];              // person[0] will return “johny”
```

WARNING !!

If you use a named index, JavaScript will redefine the array to a standard object. After that, **all array methods and properties will produce incorrect results**.

❑ The Array properties

Property	Description
constructor	Returns the function that created the Array object's prototype
length	Sets or returns the number of elements in an array
prototype	Allows you to add properties and methods to an Array object

❑ Prototype

- Make a new array method that transforms array values into upper case:

```
Array.prototype.myUcase = function() {
   for (i = 0; i < this.length; i++) {
      this[i] = this[i].toUpperCase();
   }
};
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.myUcase();
```

- The values in the *fruits* array is now:

  BANANA,ORANGE,APPLE,MANGO
- When constructing a property, ALL arrays will be given the property, and its value, as default.
- When constructing a method, ALL arrays will have this method available.

❑ The Difference Between Arrays and Objects

- **arrays** use **numbered indexes**.
- **objects** use **named indexes**.

❑ When to Use Arrays. When to use Objects.

- JavaScript does not support associative arrays.
- should use **objects** when you want the element names to be **strings (text)**.
- should use **arrays** when you want the element names to be **numbers**.

Conditional Statements

- are used when you want to perform different actions based on different conditions.

Conditional statements list:

- **if :** Used to specify a block of code to be executed, if a specified condition is true
- **Else :** Used to specify a block of code to be executed, if the same condition is false
- **else if :** Used to specify a new condition to test, if the first condition is false
- **switch :** Used to specify many alternative blocks of code to be executed

❑ The if Statement

Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax :

```
If (condition) {
    block of code to be executed if condition is true
}
```

Example

```
<!DOCTYPE html>
<html>
<body>
<p>Display "Good day!" if the hour is less than 16:00:</p>
<p id="demo">Good Evening!</p>
<script>
if (new Date().getHours() < 16) {
    document.getElementById("demo").innerHTML = "Good
day!";
}
</script>
</body>
</html>
```

o/p

Display ”Good day!” if the hour is less than 16:00

Good Evening

❑ The else Statement

Use the **else** statement to specify a block of code to be executed if the condition is false.

Syntax :

```
If (condition) {
    block of code to be executed if condition is true
}

else {
    block of code to be executed if condition is false
}
```

Example

```
if (hour < 16) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

o/p

Good Evening

❑ The else if Statement

Use the **else if** statement to specify a new condition if the first condition is false.

Syntax :

```
If (condition1) {
```

Example

```
if (time < 10) {
    greeting = "Good morning";
} else if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

o/p

Good Evening

❑ Switch Statement

The switch statement is used to perform different actions based on different conditions.

Syntax :

```
switch(expression)
{
   case 1:
        code block
        break;
    case n:
        code block
        break;
     default:
         default code block
}
```

- This is how it works:
  - ¶ The switch expression is evaluated once.
  - ¶ The value of the expression is compared with the values of each case.
  - ¶ If there is a match, the associated block of code is executed.

Example

- The getDay() method returns the weekday as a number between 0 and 6. (Sunday=0, Monday=1, Tuesday=2 ..)
- Use the weekday number to calculate weekday name:

```
Switch(new Date().getDay()} {
Case 0:
   Day="Sunday";
   Break;
Case 1:
   Day="Monday";
   Break;
Case 2:
   Day="Tuesday";
   Break;
Case 3:
   Day="Wensday";
   Break;
Case 4:
   Day="Thursday";
   Break;
Case 5:
   Day="Friday";
   Break;
Case 6:
   Day="Saturday";
   Break;
}
```

Sunday

❑ The break Keyword

- When the JavaScript code interpreter reaches a **break** keyword, it breaks out of the switch block.
- This will stop the execution of remaining code and case testing inside the block.

❑ The default Keyword

- The default keyword describes the code to run if there is no case match.

JavaScript-Looping Statements

Loops can execute a block of code many number of times.

JavaScript supports different kinds of loops

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - loops through a block of code while a specified condition is true

❑ The For Loop

The for loop is often the tool you will use when you want to create a loop.

Syntax:

```
for (statement 1; statement 2; statement 3)
{
    code to be executed
}
```

- **Statement 1** is executed before the loop (the code block) starts.
- **Statement 2** defines the condition for running the loop (the code block).
- **Statement 3** is executed each time after the loop (the code block) has been executed.

❑ The For Loop

Example

```
var txt=“;
for (i = 0; i < 4; i++)
{
      txt   +=   “The number is “ +i;
}
alert(txt);
```

o/p

```
The number is 0
The number is 1
The number is 2
The number is 3
```

- From the example above, you can read :
- Statement 1 - Sets a variable before the loop starts (var i=0).
- Statement 2 - Defines the condition for the loop to run (i must be less than 5)
- Statement 3 - Increases a value i++ each time the code block in the loop has been executed

❑ The For/In Loop

The JavaScript for/in statement loops through the properties of an object:

Example

```
var person = {fname: “Jhony” , lname:”Doey” , age:25};

var text = “ “;
var x;
for (x in person)
{
    text += person[x];
}
```

o/p

Jhony Doey 25

❑ The While Loop

The while loop loops through a block of code as long as a specified condition is true.

Syntax :

```
While(condition)
{
    code block to be executed
}
```

Example

```
while (i < 9)
{
     text   +=   “The number is “ +i;
    i++;
}
```

o/p

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
```

❑ The Do/While Loop

The do/while loop is a variant of the while loop.

This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax :

```
do
{
    code block to be executed
}
while (condition)
```

Example

```
do
{
    text   +=   "The number is " +i;
     i++;
}
while (i < 9);
```

o/p

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
```

❑ The break Statement

- The break statement can be used to jump out of a loop.
- The break statement breaks the loop and continues executing the code after the loop (if any):

Example

```
for (i = 0; i < 10;i ++)
{
    if(i ===4)
    {
       break;
     }
    text += “The number is” + i +  “<br>”;
}
```

o/p

```
The number is 0
The number is 1
The number is 3
The number is 3
```

❑ The continue Statement

The **continue statement** breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

❑ Following example skips the value of 3:

Example

```
var txt='';
for(i = 0;   i < 9; i++)
{
      if(i == 3)
      {
         continue;
       }
       txt += "The number is" + i +  "<br>";
}
alert(txt);
```

o/p

The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8

THANK YOU