Unit – 2
JavaScript

Prepared by: Dr. Susheela Dahiya

JavaScript

- JavaScript was created by **Brendan Eich** and it came into existence in September 1995, when Netscape 2.0 (a web browser) was released.
- JavaScript was designed with a purpose to make web pages dynamic and more interactive.
- JavaScript is one of the popular scripting languages having following features :
  1. It can be used for client and server applications.
  2. It is platform independent which means it can run on any operating systems (i.e. Linux, Microsoft Windows, Mac OS X etc.).
  3. JavaScript codes are needed to be embedded or referenced into HTML documents then only it can run on a web browser.
  4. It is an interpreted language.
  5. It is a case-sensitive language and its keywords are in lowercase only.

CLIENT–SIDE JAVASCRIPT

- Client-side JavaScript refers to JavaScript code that gets executed by the web browser on the client machine. JavaScript statements can be embedded in a HTML document that can respond to events such as mouse clicks, form input, and page navigation etc. without any network connection.

SERVER–SIDE JAVASCRIPT

- Server-side JavaScript is also known as “LiveWire”. Like client-side JavaScript, server-side JavaScript is also embedded within a HTML document. When a query is made by the client, web server executes the script after interpreting it.

PLACING THE JAVASCRIPT CODE

1. **Embedded/Inline JavaScript :** JavaScript code can be placed either in the HEAD or in the BODY section of a HTML document.
   a) It is advised to place JavaScript code in HEAD section when it is required to be used more than once.
   b) If the JavaScript code is small in size and used only once, it is advisable to put it in the BODY section of the HTML document.
2. **External JavaScript :** In case, same JavaScript code needs to be used in multiple documents then it is the best approach to place JavaScript code in external files having extension as **" .js"**. To do so, we will use ***src*** attribute in <SCRIPT> tag to indicate the link for the source JavaScript file.

STATEMENTS IN JAVASCRIPT

- A collection of statements to accomplish a job, is called as a script or program. The JavaScript statements will be as follows :

```
a = 100;        // stores value 100 in variable a
b = 200;        // stores value 200 in variable b
c = a + b;      // stores the sum of a and b in variable c
document.write("Sum of A and B : ");   // displays the string
document.write(c);   // displays the value of c
```

- In JavaScript, semicolon(;) is used to end a statement but if two statements are written in separate lines then semicolon can be omitted.
- Some valid statements are :

  (i) p=10
      q=20

  (ii) x=12; y=25 // semicolon(;) separating two statements.

- Some invalid statements are :

  1. x=12 y=25 // statements within the same line not separated by semicolon (;)

Comments

- Comments are the statements that are always ignored by the interpreter. They are used to give remarks to the statement making it more readable and understandable to other programmers. There are two types of comments :
- Single line comment using double-slash (//).
- Multiple lines comment using /* and */ .

For example :

```
// This is a single-line comment.
/* This is a multiple-line comment.
        It can be of any length. */
```

IDENTIFIERS

- Identifiers refer to the name of variables, functions, arrays, etc. created by the programmer.
- Identifiers may be any sequence of characters in uppercase and lowercase letters including numbers or underscore and dollar sign.
- An identifier must not begin with a number and cannot have same name as any of the keywords of the JavaScript.
- Some valid identifiers are :

```
RollNo
bus_fee
_vp
$amt
```

- Some invalid identifiers are :

```
to day          // Space is not allowed
17nov           // must not begin with a number
%age            // no special character is allowed
```

VARIABLES

- A variable is an identifier that can store values. These values can be changed during the execution of script. Once a value is stored in a variable it can be accessed using the variable name. Variable declaration is not compulsory, though it is a good practice to use variable declaration. Keyword var is used to declare a variable.

  Syntax: var var-name [= value] [..., var-name [= value]]

  Example

  ```
  var name = “Sachin”;          // Here ‘name’ is variable
  document.write(name);         // Prints Sachin
  ```

- A JavaScript variable can hold a value of any data type.

  For example :

  ```
  i = 7;
  document.write(i);
  i = “seven”;
  document.write(i);
  ```

- Some valid examples of variable declaration:

  ```
  var cost;
  var num, cust_no = 0;
  var amount = 2000;
  ```

VARIABLES

- Naming Conventions
  - We should use meaningful name for a variable.
  - A variable name must start with a letter, underscore (_), or dollar sign ($).
  - The subsequent characters can be the digits (0-9).
  - JavaScript is case sensitive, so the variable name my_school is not the same as My_School.
- Some valid variable names

  f_name

  India123

  _sumof
- Some invalid variable names

  10_numbers // must not begin with any number.

  rate% // '%' is not a valid character.

  my name // Space is not allowed.

Scope of Variables

- Block Scope

```
{
  let x = 2;
}
// x can NOT be used here\
```

- Global Scope

```
{
  var x = 2;
}
// x CAN be used here
```

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

<h2>Redeclaring a Variable Using var</h2>

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

<script>
var  x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Redeclaring a Variable Using var

2

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

<h2>Redeclaring a Variable Using let</h2>

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

<script>
let  x = 10;
// Here x is 10

{
  let x = 2;
  // Here x is 2
}

// Here x is 10
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Redeclaring a Variable Using let

10

DATA TYPES

- JavaScript supports three basic data types – number, string, boolean and two composite data types – arrays and objects.
- NUMBER
  - The number variable holds any type of number, either an integer or a real number. Some examples of numbers are: 29, -43, 3.40, 3.4323
- STRING
  - A string is a collection of letters, digits, punctuation characters, and so on. A string literal is enclosed within single quotes or double quotes (‘or “). Examples of string literals are: ‘welcome’, “7.86” , “wouldn’t you exit now”, country=”India”
- JavaScript also allows us to use Escape Sequence within string literals. The escape sequence starts with a backslash (\), followed by another character. This backslash tells browser to represent a special action or character representation. For example, \” is an escape sequence that represents a double quote (“).

Escape Sequence	*Action/ Character Represented*
\b	Backspace
\n	New line
\r	Carriage return
\t	Tab
\'	Single quote (')
\"	Double quote (")
\\	Backslash (\)

- Example :

  document.write ("Abhinav said, \"Earth doesn\'t revolve round the sun\". But teacher corrected him.");

- Here, two types of escape characters are used \" and \' in this example.
- Output

  Abhinav said, "Earth doesn't revolve round the sun". But teacher corrected him.

DATA TYPES

- BOOLEAN VALUES
  - A Boolean variable can store only two possible values either true or false.
  - Internally it is stored as 1 for true and 0 for false.
  - It is used to get the output of conditions, whether a condition results in true or false.
  - Example x == 100; // results true if x=100 otherwise false.
- ARRAYS
  - An array is a collection of data values of same types having a common name.
  - Each data element in array is referenced by its position in the array also called its index number.
  - Individual array elements can be referenced by the array name followed by the pair of square brackets having its index number.
  - The index number starts with zero in JavaScript i.e. the first element in JavaScript has it index value as 0, second has its index value as 1 and so on.
  - An array can be declared in any of the following ways :

```
var a =new a();
var x = [ ];
var m = [2,4,”sun”];
```

DATA TYPES

- ARRAYS (Contd.)
  - An array is initialised with the specified values as its elements, and its length is set to the number of arguments specified.

    Example - This creates an array name games with three elements.

    games = [“Hockey”, “Cricket”, “Football”];
  - We can also store different types of values in an array. For example :

```
var arr = new Array();        // creation of an array
arr[0] =”JAVASCIPT”;          // stores String literal at index 0
arr[1] = 49.5;                // stores real number at index 1
arr[2] = true;                // stores Boolean value
```

- NULL VALUE
  - JavaScript supports a special data type known as null that indicates “no value or blank”. Note that null is not equal to 0.

    Example –

```
var distance = new object();
distance = null;
```

EXPRESSIONS AND OPERATORS

- An expression is a combination of operators operands that can be evaluated. It may also include function calls which return values.
  - `x = 7.5`
  - `v= m + n;`
- **ARITHMETIC OPERATORS:**
  - These are used to perform arithmetic/mathematical operations like subtraction, division, multiplication etc.
  - Arithmetic operators work on one or more numerical values (either literals or variables) and return a single numerical value.
  - `var s = 10 + 20;`
  - `var h = 50 * 4;`
- **ASSIGNMENT OPERATORS**
  - It assigns the value of its right operand to its left operand.
  - This operator is represented by equal sign(=).
  - JavaScript also supports shorthand operator for standard operations.
- **RELATIONAL (COMPARISON) OPERATORS**
  - Relational Operators are some symbols which return a Boolean value true or false after evaluating the condition.
  - `10 < 5  // false, numeric comparison`

ARITHMETIC OPERATORS

+	(Addition)	-	(Subtraction)
*	(Multiplication)	/	(Division)
%	(Modulus)	++	(Increment by 1)
—	(Decrement by 1)		

SHORTHAND ASSIGNMENT OPERATORS

Shorthand operator	*Example*	*is equivalent to*
+ =	a + = b	a = a + b
– =	a – = b	a = a – b
* =	a * = b	a = a * b
/ =	a / = b	a = a / b
% =	a % = b	a = a % b

RELATIONAL (COMPARISON) OPERATORS

Operator	*Description*	*Example*
= =	is equal to	4 = = 8 returns false
! =	is not equal to	4 ! = 8 returns true
>	is greater than	8 > 4 returns true
<	is less than	8 > 4 returns false
< =	is less than or equal to	8 < = 4 returns false
> =	is greater than or equal to	8 > = 4 returns true

EXPRESSIONS AND OPERATORS

- **LOGICAL OPERATORS**
  - Logical operators are used for combining two or more conditions
  - JavaScript has three logical operators :

Operator	*Description with Example*
&& (AND)	returns *true* if both operands are *true* else it return *false*.
| | (OR)	returns *false* if both operands are *false* else it returns *true*.
! (NOT)	returns *true* if the operand is *false* and false if operand is *true*.

- **CONCATENATION OPERATOR**
  - The + operator concatenates two string operands.
  - The + operator gives priority to string operands over numeric operands It works from left to right.
  - The results depend on the order in which operations are performed.
  - “Good” + “Morning” = “GoodMorning”
  - “5” + “10” = “510”

- **Conditional Operator (? :)**
  - The conditional operator is a special JavaScript operator that takes three operands. Hence, it is also called ternary operator. A conditional operator assigns a value to a variable based on the condition.

Syntax - var_name = (condition) ? v_1 : v_2

- If (condition) is true, the value v_1 is assigned to the variable, otherwise, it assigns the value v_2 to the variable.

For example

status = (age >= 18) ? “adult” : “minor”

This statement assigns the value “adult” to the variable status if age is eighteen or more. Otherwise, it assigns the value “minor” to status.

FUNCTIONS

- A JavaScript function is a block of code designed to perform a particular task.
- Functions are First class data types (It means is whatever you could do with the variable, whatever you could do with an object you could also do with the function)
- A JavaScript function is executed when "something" invokes it (calls it).
- You can reuse code: Define the code once, and use it many times.
- You can use the same code many times with different arguments, to produce different results.
- SYNTAX:

```
function name(parameter1, parameter2, parameter3)
{
  // code to be executed
}
```

FUNCTIONS

- Function Invocation
  - The code inside the function will execute when "something" **invokes** (calls) the function:
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)
- Function Return
  - When JavaScript reaches a return statement, the function will stop executing.
  - If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
  - Functions often compute a **return value**. The return value is "returned" back to the "caller"

```
// Function to compute the product of a and b
var x = myFunction(4, 3);   // Function is called, return value
will end up in x

function myFunction(a, b)
{
  return a * b;     // Function returns the product of a and b
}
```

FUNCTIONS Factory

- Function inside another function & can be used to perform multiple tasks

```
// Function factory
function makeMultiplier(multiplier) {
  var myFunc = function (x) {
    return multiplier * x;
  };

  return myFunc;
}

var multiplyBy3 = makeMultiplier(3);
console.log(multiplyBy3(10));
var doubleAll = makeMultiplier(2);
console.log(doubleAll(100));
```

```
// Passing functions as arguments
function doOperationOn(x, operation) {
  return operation(x);
}

var result = doOperationOn(5, multiplyBy3);
console.log(result);
result = doOperationOn(100, doubleAll);
console.log(result);
```

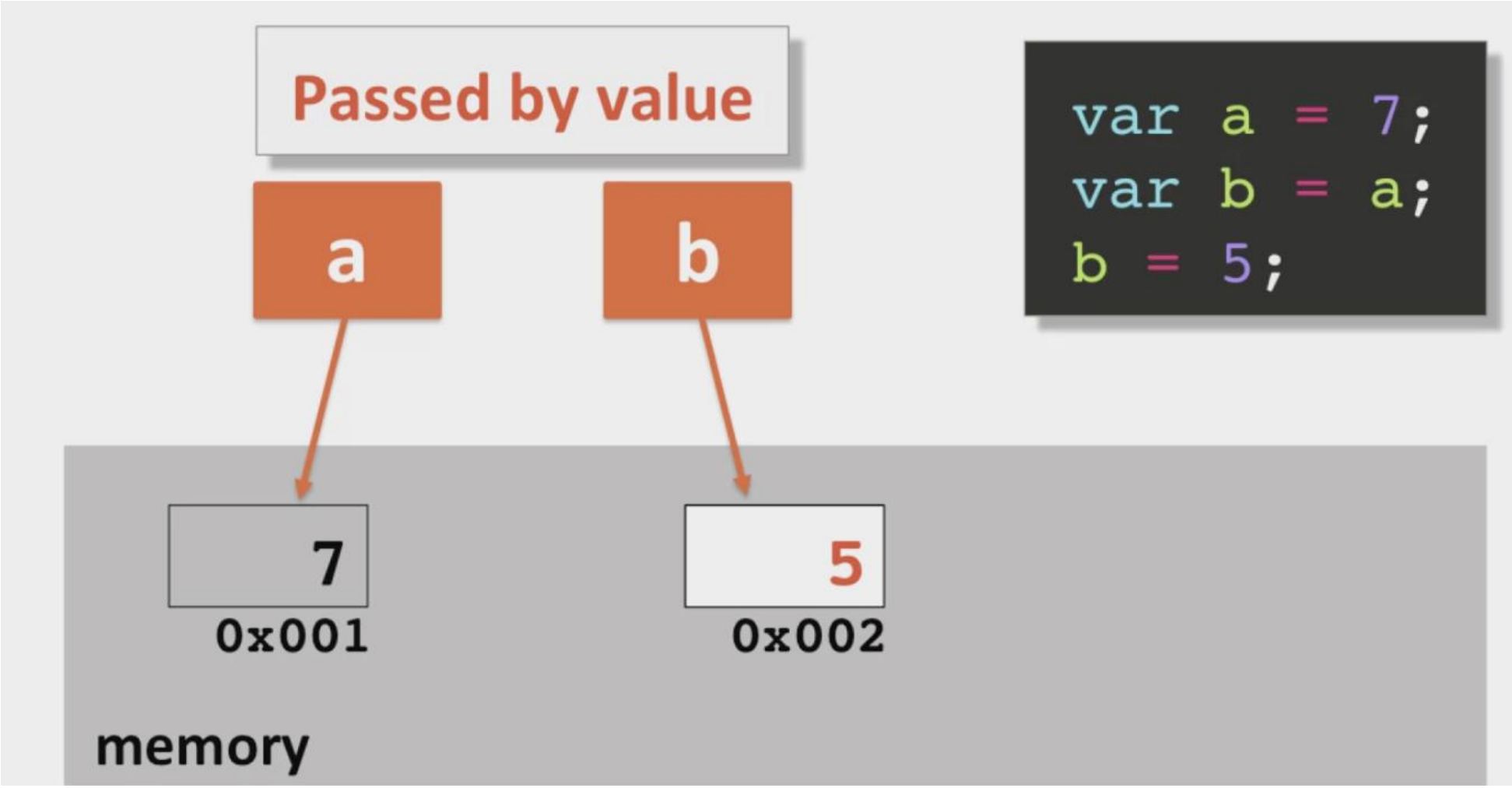
Output

```
30
200
```

```
15
200
```

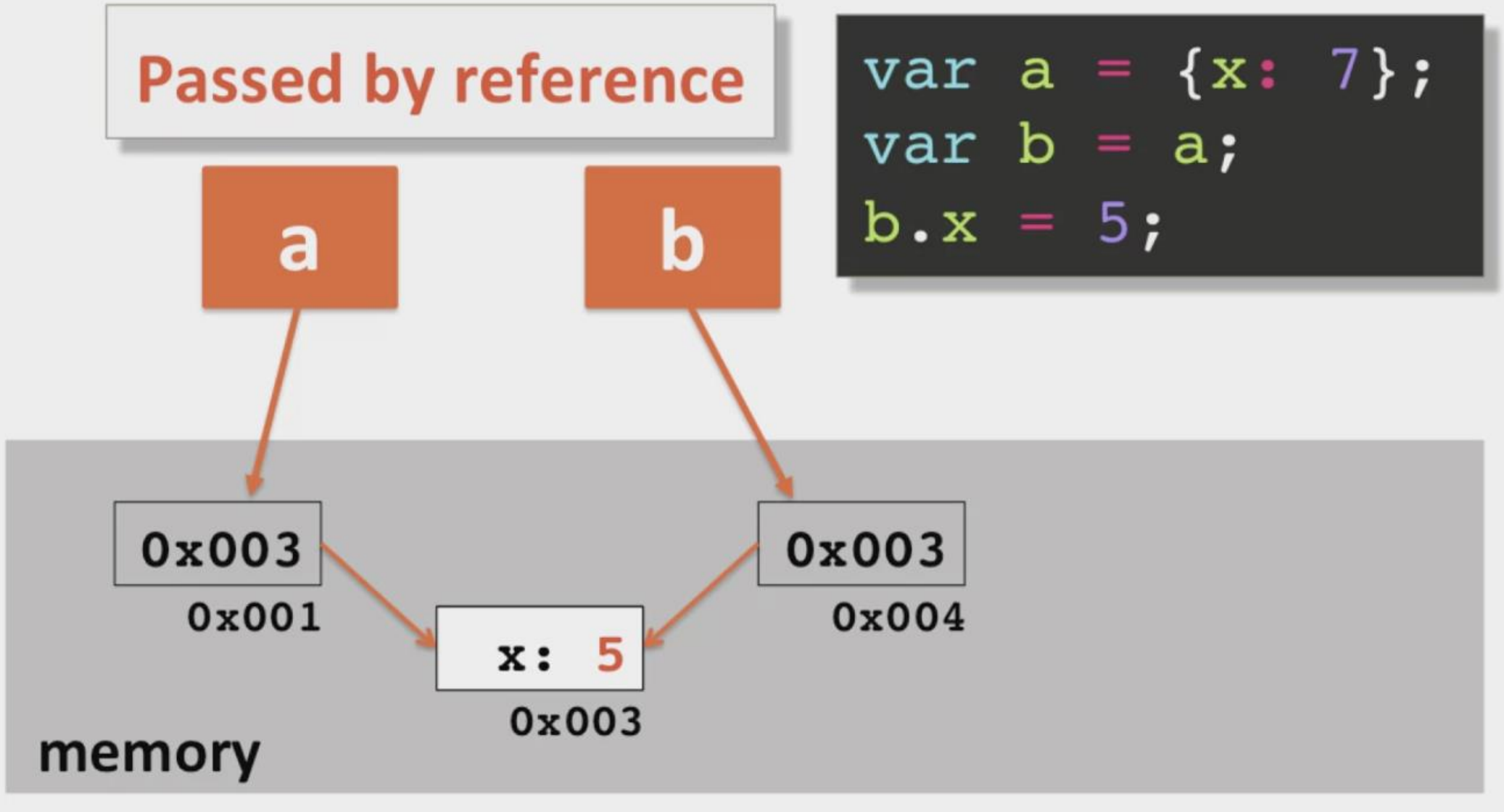
```
function makeMultiplier(multiplier) {
  var myFunFunc = function (x) {
    return multiplier * x;
  };


  return myFunFunc;
}


var operation = makeMultiplier(10);
console.log(operation(10));
```

Passing variables by value v/s by reference



Passed by reference

```
var a = {x: 7};
var b = a;
b.x = 5;
```



```
// Copy by Reference vs by Value
var a = 7;
var b = a;
console.log("a: " + a);
console.log("b: " + b);

b = 5;
console.log("after b update:");
console.log("a: " + a);
console.log("b: " + b);
```

```
a: 7
b: 7
after b update:
a: 7
b: 5
```

```
// Pass by reference vs by value
function changePrimitive(primValue) {
  console.log("in changePrimitive...");
  console.log("before:");
  console.log(primValue);

  primValue = 5;
  console.log("after:");
  console.log(primValue);
}

var value = 7;
changePrimitive(value); // primValue = value
console.log("after changePrimitive, orig value:")
console.log(value);
```

```
in changePrimitive...
before:
7
after:
5
after changePrimitive,
orig value:
7
>
```

```
var a = { x: 7 };
var b = a;
console.log(a);
console.log(b);

b.x = 5;
console.log("after b.x update:");
console.log(a);
console.log(b);
```

```
Object {x: 7}
Object {x: 7}
after b.x update:
Object {x: 5}
Object {x: 5}
```

```
// Pass by reference vs by value
function changePrimitive(primValue) {
  console.log("in changePrimitive...");
  console.log("before:");
  console.log(primValue);

  primValue = 5;
  console.log("after:");
  console.log(primValue);
}

var value = 7;
changePrimitive(value); // primValue = value
console.log("after changePrimitive, orig value:")
console.log(value);
```

```
in changePrimitive...
before:
7
after:
5
after changePrimitive,
orig value:
7
```

```
function changeObject(objValue) {
  console.log("in changeObject...");
  console.log("before:");
  console.log(objValue);

  objValue.x = 5;
  console.log("after:");
  console.log(objValue);
}

value = { x: 7 };
changeObject(value);
console.log("after changeObject, orig value:");
console.log(value);
```

```
in changeObject...
before:
Object {x: 7}
after:
Object {x: 5}
after changeObject, orig
value:
Object {x: 5}
```

setInterval method

- The setInterval Javascript method is used to call a function repeatedly at a specified interval of time. This time interval at which the function will be called is provided by the user in milliseconds.
- Syntax

let intervalID = setInterval(func, delay)

- Parameter - func is the name of the function on which we want to apply the setInterval javascript method, and delay is the time in milliseconds, i.e the interval at which the function is to be called.

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

<head>
    <title>
        setInterval() method
    </title>
</head>

<body>

    <button onclick="setInterval(demo, 5000);">
      Click here
    </button>

    <script>
        function demo() {
            alert('Here is an example of setInterval Javascript');
        }
    </script>

</body>

</html>
```

Output:

"Click Here" button

As soon as we press the "Click Here" button an alert box will pop up after every 5000ms or 5 seconds, saying that "Here is an example of setInterval Javascript.

clearInterval method

- We can cancel the interval by using the clearInterval() function in javascript. For this we need to store the interval identifier inside a variable and pass this variable as an argument to the clearInterval() method and call it.

Syntax for canceling setInterval() with clearInterval():

```
let intervalID = setInterval(...);

clearInterval(intervalID);
```

setTimeout method

- The setTimeout Javascript method is used to call a function after a certain period of time. The time after which the function will be called is given by the user in milliseconds.

Syntax

let timerID = setTimeout(func, delay)

- func is the name of the function on which we want to apply the setTimeout javascript method, and the delay is the time in milliseconds after which the above function is to be called.

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

<head>
    <title>
        setTimeout() method
    </title>
</head>

<body>

    <button onclick="setTimeout(test, 5000);">
      Click here
    </button>

    <script>
        function test() {
            alert('Here is an example of setTimeout javascript');
        }
    </script>

</body>

</html>
```

Output:

"Click Here" button

As soon as we press the "Click Here" button an alert box will pop up

after 5000ms or 5seconds, saying that "Here is an example of setTimeout javascript".

clearTimeout method

- We can cancel the timer by using the clearTimeout() function in javascript.
- For this, we need to store the timer identifier inside a variable, and pass this variable as an argument to the clearTimeout() method, and call it.
- Syntax for canceling setTimeout() with clearTimeout():

```
let timerID = setTimeout(...);

clearTimer(timerID);
```

Conditional Statements

- Conditional statements are used to perform different actions based on different conditions.
- In JavaScript we have the following conditional statements:
  - Use if to specify a block of code to be executed, if a specified condition is true
  - Use else to specify a block of code to be executed, if the same condition is false
  - Use else if to specify a new condition to test, if the first condition is false
  - Use switch to specify many alternative blocks of code to be executed

JavaScript if, else, and else if

- 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 the condition is
true
}
if (hour < 12) {
  greeting = "Good Morning";
}
else if ((hour < 20)
{
  greeting = "Good Evening";
}
else {
  greeting = "Good Day";
}
```

- The switch statement is used to perform different actions based on different conditions.
- switch statement is used to select one of many code blocks to be executed.
- 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.
  - If there is no match, the default code block is executed.
-
```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

LOOP STATEMENTS

- Loop statements are the primary mechanism for telling a JavaScript interpreter to execute statements again and again until a specified condition is met. JavaScript supports following looping statements :
  - for
  - do ... while
  - while loop
- Most loops have a counter variable which is initialised before the loop starts and then it is tested as part of the condition (expression) evaluated before every iteration of the loop. Finally, the counter variable is incremented or updated at the end of the loop body just before the condition is evaluated again.

FOR LOOP

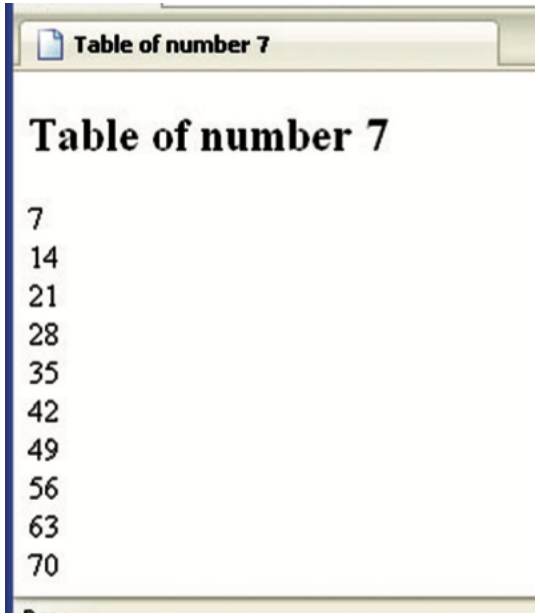
- The for loop consists of three optional expressions separated by semicolon, followed by a block of statements executed in the loop. Loop statements executed repeatedly again and again until the condition is false. The for loop is used when we know in advance how many times the script code should run.
- The Syntax is

```
for([initial-expression]; [condition]; [increment-expression])
{
statements
}
```

- Parameters
  - Initial-expression – used to initialise a counter variable.
  - Condition – If condition evaluates to true, the statements are executed.
  - Incr.-expression – to increment the counter variable.

- Examples
- The following for statement declares variable i and initialising it to 1. It checks that i is less than 20, performs the two succeeding statements, and increments i by 2 after each pass through the loop.

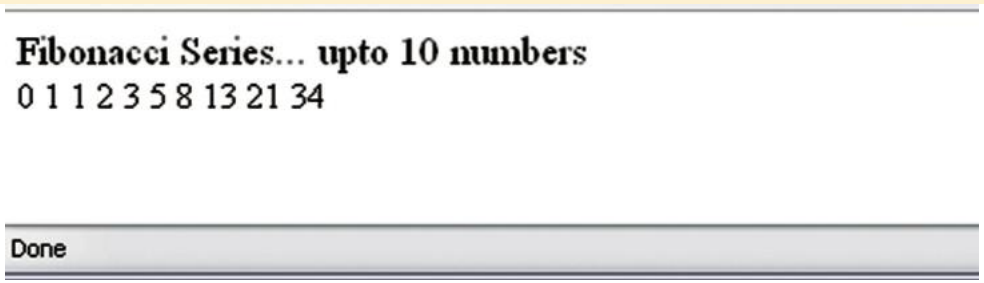
```
<HTML>
<HEAD>
<TITLE> Table of 7 </title>
</HEAD>
<BODY>
<SCRIPT language="JavaScript" type="text/JavaScript">
document.write("<H2> Table of number 7 </H2>");
// for loop to display Table of number 7
for (var i = 1; i <= 10; i++)
{ document.write(i*7);
document.write("<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```



WHILE LOOP

- The while loop statement is simpler than the for loop. It consists of a condition and block statement. The condition is evaluated before each pass through the loop. If the condition is true then it executes block statement.
- The Syntax is
- while (condition) {
  statements
  }
- Example : The following while loop gives same output as for loop in previous example.

```
<HTML>
<HEAD>
<TITLE>Fibonacci Series</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/JavaScript">
// Program to print the Fibonacci series upto 10 numbers
document.write("Fibonacci Series... upto 10 numbers <BR>".fontsize(4));
//.fontsize to increase the font size of the string
i=0;
document.write(i + "  ");
j=1;
document.writeln(j + "  ");
var x = 3;
while (x <= 10)
{
    t = i + j;
    document.write(t + "  ");
    i = j;
    j = t;
    x++;
}
</SCRIPT>
</BODY>
</HTML>
```



Do...While

- The do...while loop is much like a while loop. It will repeat the loop until the specified condition is false. This loop will always be executed at least once, even if the condition is false, because the block of statements is executed before the condition is tested. In this loop statement, curly braces are optional.
- The Syntax is
- do
  {
  statements
  }
  while (condition);

Example : The following do...while loop gives same output as while loop in previous example.

```
// do...while loop to display Odd numbers between 1 to 20 var i = 1;
// Initialization of counter variable
do
{
document.write(i);
document.write("<BR>");
I += 2; // Updation
} while (i < 20); // Condition
```

Condition lies between the parentheses after the block of statements with while keyword.

LABEL

- A label is an identifier followed by a colon that can be helpful in directing the flow of program.
- The Syntax is

  label: statement

- The value of label may be any JavaScript identifier. The statement that you identify with a label may be any statement.
- Example

  In this example, the label “whileloop” identifies a while loop.

```
var x=1;
whileloop:
while(x<=10)
{
    document.write(x);
    x++;
}
```

BREAK

- Break statement is used to exit from the innermost loop, switch statement, or from the statement named by label. It terminates the current loop and transfers control to the statement following the terminated loop.
- The Syntax is

  break [label]
- The break statement includes an optional label that allows the control to exit out of a labeled statement.
- Example : The following program segment has a break statement that terminates the while loop when it is equal to 3.

```
var i = 0;
while (i < 6)
{
      if (i == 3)
                  break; //the control moves out of loop in first iteration
      i++;
}
document.write(i);
```

CONTINUE

- The continue statement skips the statement following it and executes the loop with next iteration. It is used along with an if statement inside while, do-while, for, or label statements.
- The Syntax is

  continue [label]

- The continue statement does not terminate the loop. Instead, in a while loop, it jumps back to the condition and in a for loop, it jumps to the update expression. The continue statement can include an optional label that allows the program to terminate a labeled statement and continue to the specified labeled statement.
- Example: A program to input 50 elements using prompt() and then compute sum of marks more than 40 using continue statement.

```
var marks = new Array();
var i = 0, sum=0;
while (i < 50)
{
        i++;
        marks[i]= parseInt(prompt("Enter marks")); // parseInt converts
string value                                          into a number.
        if (marks[i] <= 40) // when the condition is true then
                continue;     // control goes to while condition expression.
        sum = sum + marks[i];
}

document.write(sum+"\n");
```

JAVASCRIPT POPUP BOXES (DIALOG BOXES)

- In JavaScript, three kinds of popup boxes – Alert box, Confirm box, and Prompt box can be created using three methods of window object.
- ALERT BOX
  - alert( ) method of window object creates a small dialog box with a short text message and “OK” command button called alert box. Alert box contains an icon indicating a warning.
  - An alert box is used if we want to display some information to the user. When an alert box appears, the user needs to click “OK” button to proceed.
  - Syntax

```
[window].alert(“Text to be displayed on the popup box”);
```

The word window is optional.

Example

```
window.alert(“I am to alert you about ....”);
or
alert(“I am to alert you about ....”);
```

JAVASCRIPT POPUP BOXES (DIALOG BOXES)

- CONFIRM BOX
  - Confirm box is used if we want the user to verify and confirm the information. The user will have to click either “OK” or “Cancel” buttons.
  - Confirm box returns a Boolean value. If the user clicks on “OK”, it returns true. If the user clicks on “Cancel”, it returns false.

Syntax: `[window].confirm(“Text to be confirmed”);`

Example: `confirm(“Do you want to quit now?”);`

- PROMPT BOX
  - Prompt box allows getting input from the user. We can specify the default text for the text field. The information submitted by the user from *prompt( )* can be stored in a variable.
  - A prompt box returns input string value when the user clicks “OK”. If the user clicks “Cancel”, it returns null value.

Syntax: `prompt(“Message” [, “default value in the text field”]);`

Example: `var name = prompt(“What’s your name? “, “Your name please...”);`

OBJECTS

- JavaScript is an object-based scripting language. It allows us to define our own objects and make our own variable types.
- It also offers a set of predefined objects. The tables, forms, buttons, images, or links on our web page are examples of objects.
- The values associated with object are properties and the actions that can perform on objects are methods or behaviour.
- Property associated to an object can be accessed as follows:
  - ObjectName.PropertyName

Predefined Objects

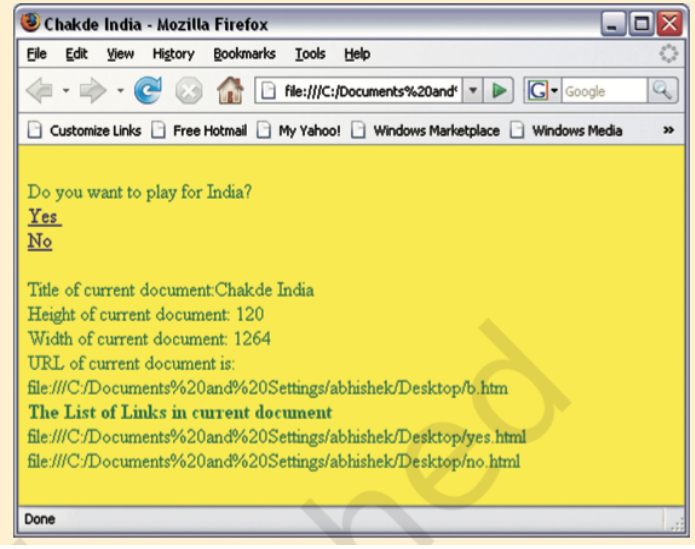
- DOCUMENT OBJECT
  - The Document object is one of the parts of the Window object.
  - It can be accessed through the window.document property.
  - The document object represents a HTML document, and it allows one to access all the elements in a HTML document.
  - For example: title of current document can be accessed by document.title property.

Some of the common properties of document object are :

Properties	***Purposes***
Title	returns/ sets title of the current document.
bgColor	returns/ sets the background color of the current document.
fgColor	returns/ sets the text color of the current document.
linkColor	returns/ sets the color of hyperlinks in the document.
alinkColor	returns/ sets the color of active links in the document.
vlinkColor	returns/ sets the color of visited hyperlinks.
height	returns the height of the current document.
width	returns the width of the current document.
Forms	returns a list of the FORM elements within the current document.
Images	returns a list of the images in the current document.
URL	returns a string containing the URL of the current document.
Location	to load another URL in current document window.

Methods	***Purposes***
open()	Opens a document for writing.
write()	Writes string/data to a document.
writeln()	Writes string/data followed by a newline character to a document.
close()	Closes a document stream for writing.

```
<HTML>
<HEAD>
<TITLE>Document Properties</TITLE>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
document.fgColor = "green"; // sets text color
document.bgColor = "yellow";      // background color
document.title = "Chakde India";  // change title
document.linkColor = "navy";      // hyperlinks color
document.alinkColor = "red";      // active links
document.vlinkColor = "lime";     // visited hyperlinks
document.write("<BR>Do you want to play for India?");
document.write("<BR> <A href='yes.html'> Yes </A>");
document.writeln("<BR> <A href='no.html'>No</A><BR>");
document.write("<BR>Title of current document: " + document.title);
document.write("<BR>Height of current document: " + document.height);
document.write("<BR>Width of current document: " + document.width);
document.write("<BR> URL of current document is: " + document.URL);
//Use of document.links to list of all the hyperlinks
document.write("<BR><B>The List of Links in current document</B>");
var links = document.links;
for(var i = 0; i < links.length; i++)
{
document.write("<BR>"+ document.links[i]);
}
</SCRIPT>
</BODY>
</HTML>
```



DATE OBJECT

- Date object is used to set and manipulate date and time. JavaScript dates are stored as the number of milliseconds since midnight, January 1, 1970. This date is called the epoch. Dates before 1970 are represented by negative numbers. A date object can be created by using the new keyword with Date().
- Date objects are created with the new Date() constructor.

Syntax

- ```
  new Date()
  new Date(date string)
  new Date(year,month)
  new Date(year,month,day)
  new Date(year,month,day,hours)
  new Date(year,month,day,hours,minutes)
  new Date(year,month,day,hours,minutes,seconds)
  new Date(year,month,day,hours,minutes,seconds,ms)
  new Date(milliseconds)
  ```
  ```

DATE OBJECT

Different examples of using a date()

- `today = new Date();`
- `dob = new Date("October 5, 2007 12:50:00");`
- `doj = new Date(2007,10,5);`
- `bday = new Date(2007,10,5,12,50,0);`

Methods to read date values

We can use the ***get*** methods to get values from a Date object. Some get methods examples that returns some value according to local time are:

- getDate( ) - Returns the day of the month
- getDay( ) - Returns the day of the week
- getFullYear( ) - Returns the full year
- getHours( ) - Returns the hour
- getMinutes( ) - Returns the minutes
- getMonth( ) - Returns the month
- getSeconds( ) - Returns the seconds
- getTime( ) - Returns the numeric value corresponding to the time
- getYear( ) - Returns the year

```html
<!DOCTYPE HTML>
<HTML>
<HEAD>
    <TITLE>Displaying Time</TITLE>
</HEAD>
<BODY>
    <CENTER>
          <H1>Today's Date and Current Time</H1>
    </CENTER>
    <SCRIPT TYPE = "TEXT/JAVASCRIPT">
        var today = new Date();
        var dob = new Date("October 13, 2014 11:13:00");
        var doj = new Date("2022-03-25");
        var bday = new Date(2007,10,5,12,50,0);
        document.write("Today: ");
        document.write(today);
        document.write("<BR>");
        document.write(dob);
        document.write("<BR>");
        document.write(doj);
        document.write("<BR>");
        document.write(bday);
        document.write("<BR>");
    </SCRIPT>
</BODY>
</HTML>
```

Today's Date and Current Time

Today: Thu Mar 23 2023 13:33:17 GMT+0530 (India Standard Time)
Mon Oct 13 2014 11:13:00 GMT+0530 (India Standard Time)
Fri Mar 25 2022 05:30:00 GMT+0530 (India Standard Time)
Mon Nov 05 2007 12:50:00 GMT+0530 (India Standard Time)

Date Get Methods

Method	Description
getFullYear()	Get **year** as a four digit number (yyyy)
getMonth()	Get **month** as a number (0-11)
getDate()	Get **day** as a number (1-31)
getDay()	Get **weekday** as a number (0-6)
getHours()	Get **hour** (0-23)
getMinutes()	Get **minute** (0-59)
getSeconds()	Get **second** (0-59)
getMilliseconds()	Get **millisecond** (0-999)
getTime()	Get **time** (milliseconds since January 1, 1970)

```html
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Date get Mthods</h1>

<script>
    const d = new Date("2023-03-23");
    document.write("Date d = ");
    document.write(d);
    document.write("<BR>");
    document.write("getFullYear = ");
    document.write(d.getFullYear());
    document.write("<BR>");
    document.write("getMonth = ");
    document.write(d.getMonth());
    document.write("<BR>");
    document.write("getDate = ");
    document.write(d.getDate());
    document.write("<BR>");
    document.write("getDay = ");
    document.write(d.getDay());
    document.write("<BR>");
    document.write("getHours = ");
    document.write(d.getHours());
    document.write("<BR>");
    document.write("getMinutes = ");
    document.write(d.getMinutes());
    document.write("<BR>");
    document.write("getSeconds = ");
    document.write(d.getSeconds());
    document.write("<BR>");
    document.write("getMilliseconds = ");
    document.write(d.getMilliseconds());
    document.write("<BR>");
    document.write("getTime = ");
    document.write(d.getTime());
    document.write("<BR>");
</script>

</body>
</html>
```

JavaScript Date get Mthods

Date d = Thu Mar 23 2023 05:30:00 GMT+0530 (India Standard Time)
getFullYear = 2023
getMonth = 2
getDate = 23
getDay = 4
getHours = 5
getMinutes = 30
getSeconds = 0
getMilliseconds = 0
getTime = 1679529600000

Set Date Methods

Set Date methods are used for setting a part of a date:

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

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

<h2>JavaScript Date set Methods</h2>
<p>Please note that month counts from 0. December is month 11:</p>

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

<script>
const d = new Date();
d.setFullYear(2020, 11, 3);
document.write("date = ");  document.write(d);
document.write("<BR>");
d.setMonth(10);
document.write("date = ");  document.write(d);
document.write("<BR>");
d.setDate(15);
document.write("date = ");  document.write(d);
document.write("<BR>");
d.setDate(d.getDate() + 50);
document.write("date = ");  document.write(d);
document.write("<BR>");
d.setHours(22);
document.write("date = ");  document.write(d);
document.write("<BR>");
d.setMinutes(30);
document.write("date = ");  document.write(d);
document.write("<BR>");
d.setSeconds(30);
document.write("date = ");  document.write(d);
document.write("<BR>");
</script>

</body>
</html>
```

JavaScript Date set Methods

Please note that month counts from 0. December is month 11:

date = Thu Dec 03 2020 13:55:16 GMT+0530 (India Standard Time)
date = Tue Nov 03 2020 13:55:16 GMT+0530 (India Standard Time)
date = Sun Nov 15 2020 13:55:16 GMT+0530 (India Standard Time)
date = Mon Jan 04 2021 13:55:16 GMT+0530 (India Standard Time)
date = Mon Jan 04 2021 22:55:16 GMT+0530 (India Standard Time)
date = Mon Jan 04 2021 22:30:16 GMT+0530 (India Standard Time)
date = Mon Jan 04 2021 22:30:30 GMT+0530 (India Standard Time)

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

<script>
let text;
const today = new Date();
const someday = new Date();
someday.setFullYear(2100, 0, 14);

if (someday > today) {
  text = "Today is before January 14, 2100.";
} else {
  text = "Today is after January 14, 2100.";
}

document.write(text);

</script>

</body>
</html>
```

OUTPUT:

Today is before January 14, 2100.

MATH OBJECT

- This object contains methods and constants to carry more complex mathematical operations.
- This object cannot be instantiated like other objects.
- All properties and methods of Math are static.
- We can refer to the constant p as Math.PI and the sine function as Math.sin(x), where x is the method’s argument.

There are 4 common methods to round a number to an integer:

Math.round(x)	Returns x rounded to its nearest integer
Math.ceil(x)	Returns x rounded up to its nearest integer
Math.floor(x)	Returns x rounded down to its nearest integer
Math.trunc(x)	Returns the integer part of x (new in ES6)

Methods	***+Description***
pow(x, p)	Returns X^P
abs(x)	Returns absolute value of x.
exp(x)	Returns e^x
log(x)	Returns the natural logarithm of x.
sqrt(x)	Returns the square root of x.
random()	Returns a random number between 0 and 1.
ceil(x)	Returns the smallest integer greater than or equal to x.
floor(x)	Returns the largest integer less than or equal to x.
min(x, y)	Returns the lesser of x and y.
max(x, y)	Returns the larger of x and y.
round(x)	Rounds x up or down to the nearest integer.
sin(x)	Returns the sin of x, where x is in radians.
cos(x)	Returns the cosine of x, where x is in radians.
tan(x)	Returns the tan of x, where x is in radians.


```
Math.E       // returns Euler's number
Math.PI      // returns PI
Math.SQRT2   // returns the square root of 2
Math.SQRT1_2 // returns the square root of 1/2
Math.LN2     // returns the natural logarithm of 2
Math.LN10    // returns the natural logarithm of 10
Math.LOG2E   // returns base 2 logarithm of E
Math.LOG10E  // returns base 10 logarithm of E
```

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE>JavaScript Math Object</TITLE>
</HEAD>
<BODY>
<SCRIPT>
    document.write("Value of PI :");
    document.write(Math.PI);
    document.write("<BR>");
    document.write("Random value:" + Math.random() + "<BR>");
    document.write("Rounded value of 0.69 :" + Math.round(0.69) + "<br>");
    document.write("Value of 5 <SUP>2</SUP> :" +  Math.pow(5,2) + "<br>");
    document.write("Square root of 2 :" + Math.SQRT2 );
</SCRIPT>
</BODY>
</HTML>
```

Value of PI :3.141592653589793
Random value:0.8203466828422956
Rounded value of 0.69 :1
Value of 5^{2} :25
Square root of 2 :1.4142135623730951

JavaScript Const

- Variables defined with const cannot be Redeclared.
- Variables defined with const cannot be Reassigned.
- Variables defined with const have Block Scope.
- JavaScript const variables must be assigned a value when they are declared
- Constant Objects and Arrays
  - const does not define a constant value. It defines a constant reference to a value.
  - You can NOT:
    - Reassign a constant value
    - Reassign a constant array
    - Reassign a constant object
  - But you CAN:
    - Change the elements of constant array
    - Change the properties of constant object

Constant Objects and Arrays

- Constant Arrays

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// You can change an element:
cars[0] = "Toyota";


// You can add an element:
cars.push("Audi");


//But you can NOT reassign the array:
cars = ["Toyota", "Volvo", "Audi"];    // ERROR
```

- Constant Objects

```
// You can create a const object:
const car = {type:"Fiat", model:"500", color:"white"};

// You can change a property:
car.color = "red";

// You can add a property:
car.owner = "Johnson";
 // You can NOT reassign the object:
car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR
```

JavaScript Can Change HTML Content

- One of many JavaScript HTML methods is getElementById().
- Syntax - `document.getElementById(id)`
- The `id` attribute defines the HTML element.
- The `innerHTML` property defines the HTML content.
- The following statement tells the browser to write "Hello JavaScript" inside an HTML element with id="demo":

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

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

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>

<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the
light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the
light</button>

</body>
</html>
```

What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.

Turn on the light Turn off the light

What Can JavaScript Do?

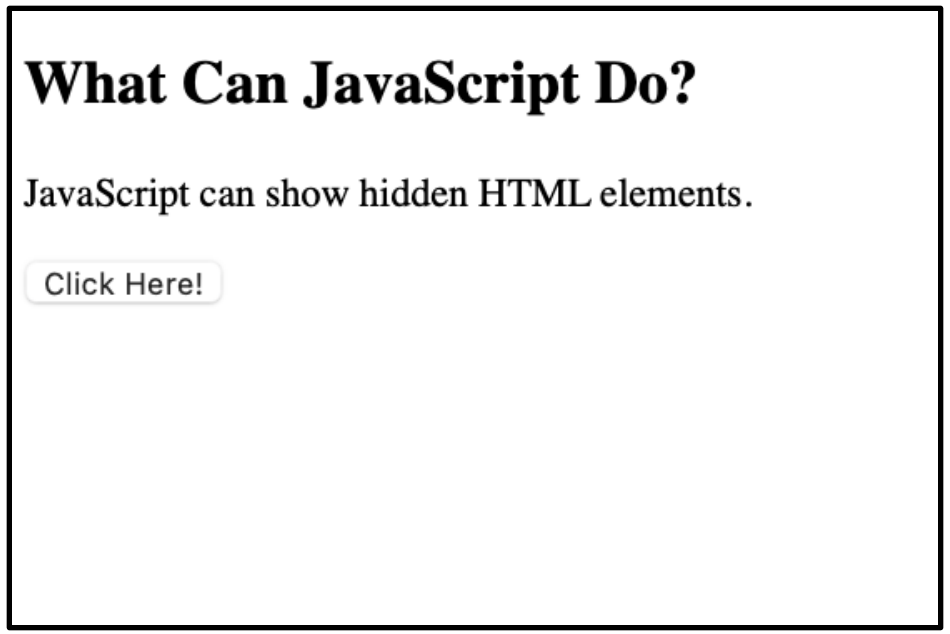
JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.

Turn on the light Turn off the light

JavaScript Can Show HTML Elements

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

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can show hidden HTML elements.</p>

<p id="demo" style="display:none">Hello JavaScript!</p>

<button type="button" onclick="document.getElementById('demo').style.display='block'">Click Here!
</button>

</body>
</html>
```



JavaScript String Methods

- String length

  let str = “JavaScript”;

  let str_length = str.length;
- String slice(start, end)
  - string indexing starts from 0
  - end position will not be inclusive
  - let str_slice = str.slice(3, 8)
  - let str_slice = str.slice(7) – (end of the string will be the 2nd parameter)
  - If a parameter is negative, the position is counted from the end of the string (indexing from end starts from -1):

    let str_slice = str.slice(-6); let str_slice = str.slice(-6, -3);
- String substring(start, end)
  - The difference between slice and substr is that - in substring() start and end values less than 0 are treated as 0
- String substr(start, length)
  - The second parameter specifies the length of the extracted part.
  - If the value for second parameter is not specified, substr() will slice out the rest of the string.

Indexing	0	1	2	3	4	5	6	7	8	9	
	J	a	v	a	S	c	r	i	p	t	
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	Indexing

JavaScript String Methods

- String replace(old, new) - case-sensitive, returns a new string, replaces only the first match
  - let str = ”JavaScript Programming!”;
    let newStr = str.replace(“JavaScript”, ”Python”);
  - To replace all matches, use a regular expression with a /g flag (global match):
    str.replace(/Java/g, ”Python”);
- String replaceAll()
  - let str = ”Java Programming and JavaScript Programming!”;
    let newStr = str.replaceAll(“Java”, ”Python”);
- String toUpperCase()
- String toLowerCase() - str.toLowerCase();
- String concat() - let str3 = str1.concat(" ", str2);
  - str = ”Java" + " " + ”Script!"; // Output – Java Script
    str = "Java".concat(" ", "Script!"); // Output – Java Script
- String trim() - removes whitespace from both sides of a string
  - let str1 = " JavaScript. ";
    let str2 = str1.trim();
- String trimStart() - removes whitespace only from the start of a string
- String trimEnd() - removes whitespace only from the start of a string

JavaScript String Methods

- String padStart() –
  - pads a string with another string - To pad a number, convert the number to a string first

    let num = 15; let str = num.toString();

    str.padStart(4,”0") // output - 0015
- String padEnd()

    let num = 15; let str = num.toString();

    str.padEnd(4,”0") // output - 1500
- String charAt() - returns the character at a specified index (position) in a string
  - let str = “JavaScript”;
  - let char = str.charAt(4); // output - S
- String charCodeAt() - returns the unicode of the character at a specified index in a string
  - let str = “JavaScript”;
  - let char = str.charCodeAt(0); // output - 72
- String split() - A string get converted to an array –
  - text.split(",") // Split on commas
  - If the separator is omitted, the returned array will contain the whole string in index [0].

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

<h1>JavaScript String Methods</h1>
<h2>The split().Method</h2>

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

<script>
let str = "JavaScript";
const myArr = str.split("");

text = "";
for (let i = 0; i < myArr.length; i++) {
  text += myArr[i] + "<br>"
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript String Methods

The split().Method

J
a
v
a
S
c
r
i
p
t

JavaScript Search Methods

- String indexOf() –
  - returns the index of (position of) the first occurrence of a string in a string
  - return -1 if the text is not found
  - ```
    let str = "Please click on ‘click here’";
    str.indexOf("click");              //output - 7
    str.indexOf("click",15);        //output - 17
    ```
- String lastIndexOf()
  - returns the index of (position of) the last occurrence of a string in a string
  - return -1 if the text is not found
  - ```
    let str = "Please click on ‘click here’";
    str.lastIndexOf(”click”);       //output - 17
    str.lastIndexOf(”click", 15); //output - 7
    //searches backwards (from the end to the beginning)
    ```
- String search()
  - returns the position of the first match
  - ```
    str.search(”click”);        //output - 7
    ```
    ```

JavaScript Search Methods

- String match()
  - returns an array containing the results of matching a string against a string
  - `let str = "Please click on ‘click here’";`
  - `str.match(“click”); //output – 1 click`
  - `str.match(/click/g); //global search //output – 2 click,click`
  - `str.match(/click/gi); //global case-insensitive search`
- String matchAll() `text.matchAll(“click”);`
- String includes() –
  - returns true if a string contains a specified value otherwise returns false
  - `let str = " Please click on ‘click here’";`
    `str.includes(”click");`
    `str.includes(”click", 12);`
- String startsWith()
  - returns true if a string begins with a specified value. case-sensitive
  - `str.startsWith("Hello");`
- String endsWith()
  - returns true if a string ends with a specified value
  - `str.endssWith("Hello");`

Types of Errors

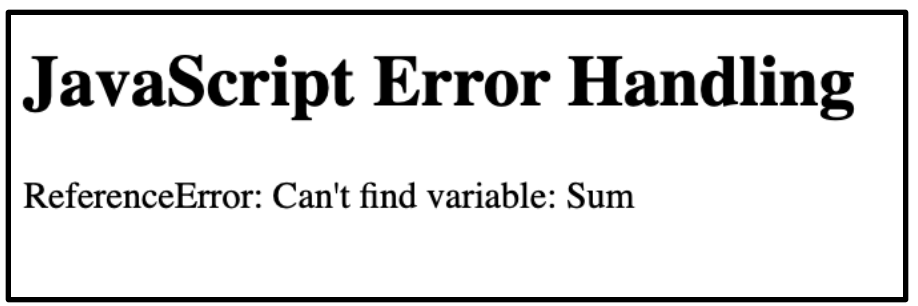
- Three types of errors:
  - Syntax Errors
  - Runtime Errors
  - Logical Errors
- Syntax errors (Parsing errors) occur at interpret time in JavaScript. Ex: missing a closing parenthesis. When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.
- Runtime Errors are also called exceptions, occur during execution (after interpretation). Ex: try to call a method that does not exist.
- Logical errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the expected result. You cannot catch those errors, because it depends on your requirement what type of logic you want to put in your program.

JavaScript Error Handling

- Try...Catch...Finally
- The try statement defines a code block to run that may throw an error in try block.
- The catch statement defines a code block to handle an error.
  - The catch block can have parameters that will give you error information.
  - Generally, catch block is used to log an error or display specific messages to the user.
- The finally statement defines a code block to be executed regardless of the occurrence of an error.
- The throw statement defines a custom error.

```
try
{
        // code that may throw an error
}
catch(ex)
{
        // code to be executed if an error occurs
}
finally
{
        // code to be executed regardless of an error occurs or not
}
```

```html
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Error Handling</h1>

    <p id="errorMessage"></p>

    <script>
        try
        {
            var result  =  Sum(10, 20); // Sum is not defined yet
        }
        catch(ex)
        {
            document.getElementById("errorMessage").innerHTML = ex;
        }
    </script>
</body>
</html>
```



JavaScript Error Handling

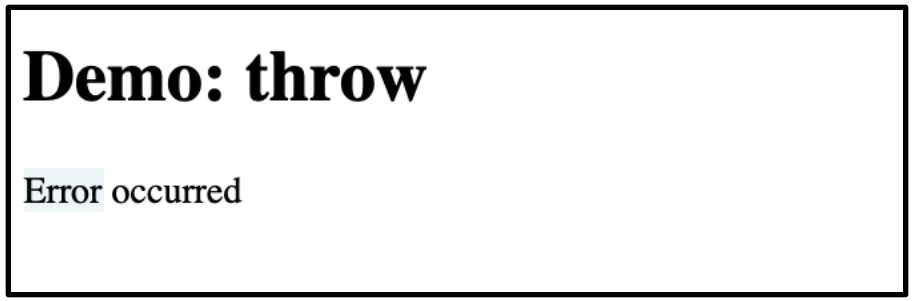
Throw

- throw keyword is used to raise a custom error.

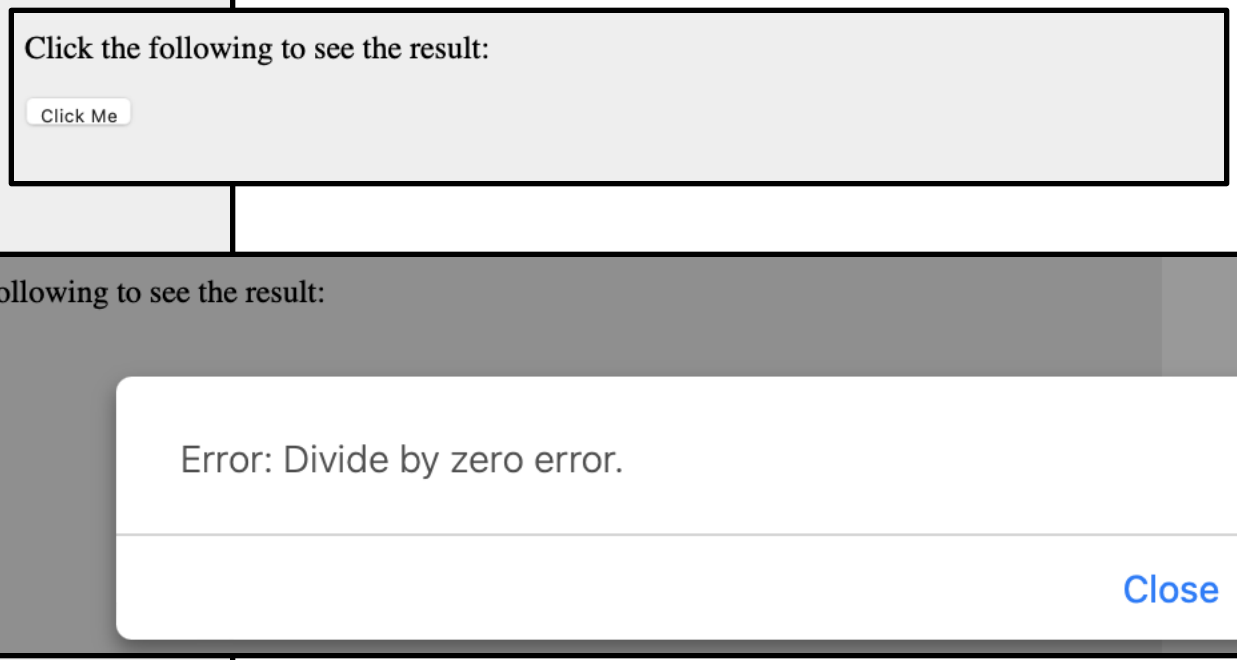
The onerror() Method

- The **onerror** event handler was the first feature to facilitate error handling in JavaScript. The **error** event is fired on the window object whenever an exception occurs on the page.
- The **onerror** event handler provides three pieces of information to identify the exact nature of the error –
  - **Error message** – The same message that the browser would display for the given error
  - **URL** – The file in which the error occurred
  - **Line number**– The line number in the given URL that caused the error

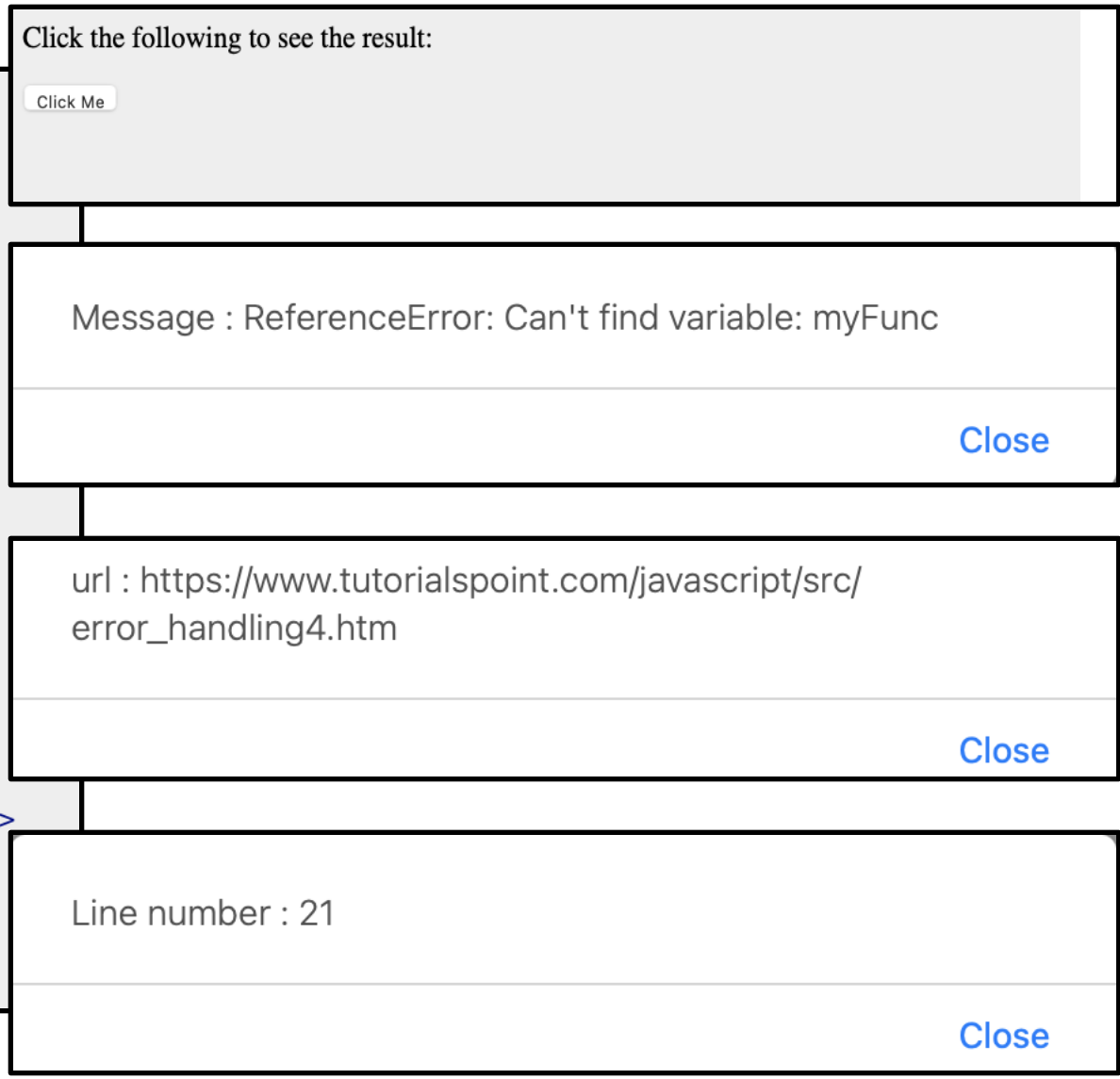
```
<html>
<body>
    <h1>Demo: throw</h1>

    <p id="errorMessage"></p>

    <script>
        try
        {
            throw "Error occurred";
        }
        catch(ex)
        {
            document.getElementById("errorMessage").innerHTML = ex;
        }

    </script>
</body>
</html>
```



```
<html>
   <head>

      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a = 100;
               var b = 0;

               try {
                  if ( b == 0 ) {
                     throw( "Divide by zero error." );
                  } else {
                     var c = a / b;
                  }
               }
               catch ( e ) {
                  alert("Error: " + e );
               }
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```



```html
<html>
   <head>

      <script type = "text/javascript">
         <!--
            window.onerror = function (msg, url, line) {
               alert("Message : " + msg );
               alert("url : " + url );
               alert("Line number : " + line );
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```



JavaScript User-defined Objects

- An object is a collection of name-value pairs.
- In JavaScript, when a property doesn't exist and you simply mention that property in an object, JavaScript engine creates that property for you.

```
// Object creation
var company = new Object();
company.name = "Facebook";
company.ceo = new Object();
company.ceo.firstName = "Mark";
company.ceo.favColor = "blue";

console.log(company);
console.log("Company CEO name is: "
  + company.ceo.firstName);

console.log(company["name"]);
var stockPropName = "stock of company";
company[stockPropName] = 110;

console.log("Stock price is: " +
  company[stockPropName]);
```

```
company.$stock = 110;
```

```
company["stock of company"] = 110;
```

```
// Better way: object literal
var facebook = {
  name: "Facebook",
  ceo: {
    firstName: "Mark",
    favColor: "blue"
  },
  $stock: 110
};
```

```
// Better way: object literal
var facebook = {
  name: "Facebook",
  ceo: {
    firstName: "Mark",
    favColor: "blue"
  },
  "stock of company": 110
};

console.log(facebook);
```

```
console.log(facebook.ceo.firstName);
```

JavaScript - Form Validation

- **Basic Validation** –
  - The form must be checked to make sure all the mandatory fields are filled in.
  - Basic validation would require just a loop through each field in the form to check for data.
- **Data Format Validation** –
  - The data that is entered must be checked for correct form and value.
  - The code must include appropriate logic to test correctness of data.

```
<html>

   <head>
      <title>Form Validation</title>

      <script type = "text/javascript">
         <!--
            // Form validation code will come here.
         //-->
      </script>

   </head>

   <body>
      <form action = "/cgi-bin/test.cgi" name = "myForm" onsubmit = "return(validate());">
         <table cellspacing = "2" cellpadding = "2" border = "1">

            <tr>
               <td align = "right">Name</td>
               <td><input type = "text" name = "Name" /></td>
            </tr>

            <tr>
               <td align = "right">EMail</td>
               <td><input type = "text" name = "EMail" /></td>
            </tr>

            <tr>
               <td align = "right">Zip Code</td>
               <td><input type = "text" name = "Zip" /></td>
            </tr>

            <tr>
               <td align = "right">Country</td>
               <td>
                  <select name = "Country">
                     <option value = "-1" selected>[choose yours]</option>
                     <option value = "1">USA</option>
                     <option value = "2">UK</option>
                     <option value = "3">INDIA</option>
                  </select>
               </td>
            </tr>

            <tr>
               <td align = "right"></td>
               <td><input type = "submit" value = "Submit" /></td>
            </tr>

         </table>
      </form>

   </body>
</html>
```

Name	
EMail	
Zip Code	
Country	[choose yours]
	Submit

Basic Form Validation

```
<script type = "text/javascript">
   <!--
      // Form validation code will come here.
      function validate() {

         if( document.myForm.Name.value == "" ) {
            alert( "Please provide your name!" );
            document.myForm.Name.focus() ;
            return false;
         }
         if( document.myForm.EMail.value == "" ) {
            alert( "Please provide your Email!" );
            document.myForm.EMail.focus() ;
            return false;
         }
         if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value ) ||
            document.myForm.Zip.value.length != 5 ) {

            alert( "Please provide a zip in the format #####." );
            document.myForm.Zip.focus() ;
            return false;
         }
         if( document.myForm.Country.value == "-1" ) {
            alert( "Please provide your country!" );
            return false;
         }
         return( true );
      }
   //-->
</script>
```

Data Format Validation

```
<script type = "text/javascript">
   <!--
      function validateEmail() {
         var emailID = document.myForm.EMail.value;
         atpos = emailID.indexOf("@");
         dotpos = emailID.lastIndexOf(".");

         if (atpos < 1 || ( dotpos - atpos < 2 )) {
            alert("Please enter correct email ID")
            document.myForm.EMail.focus() ;
            return false;
         }
         return( true );
      }
   //-->
</script>
```

HTML DOM (Document Object Model)

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
- It defines the logical structure of documents and the way a document is accessed and manipulated.
- DOM is a way to represent the webpage in a structured hierarchical way so that it will become easier for programmers and users to glide through the document. With DOM, we can easily access and manipulate tags, IDs, classes, Attributes, or Elements of HTML using commands or methods provided by the Document object.

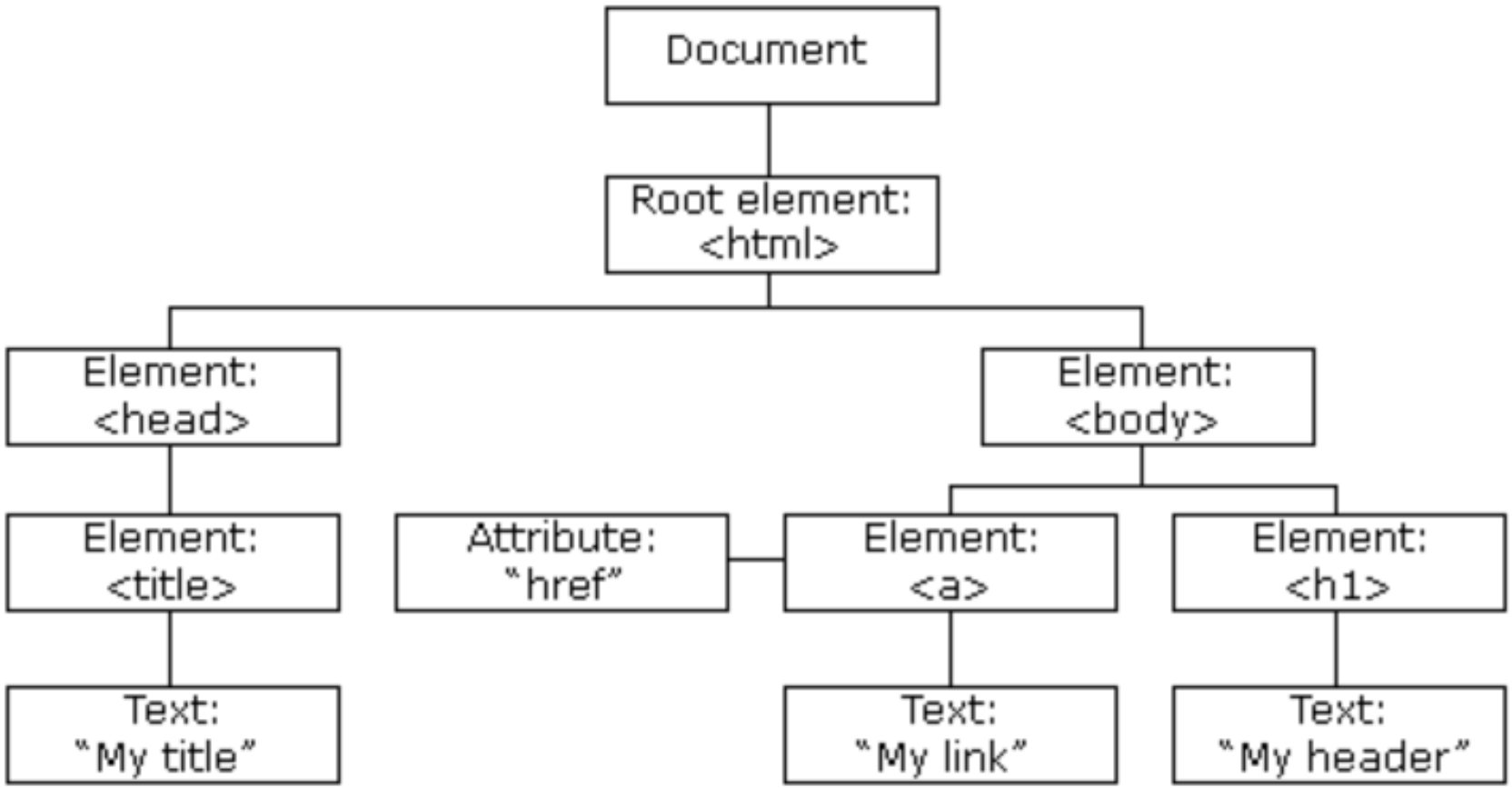
HTML DOM (Document Object Model)

- In the DOM, all HTML elements are defined as **objects**.
- The **HTML DOM** model is constructed as a tree of **Objects**
- The HTML DOM is a standard for how to get, change, add, or delete HTML elements
- HTML DOM methods are **actions** you can perform on HTML Elements (like adding or deleting an HTML element).
- HTML DOM properties are **values** (of HTML Elements) that you can get or set (like changing the content of an HTML element).

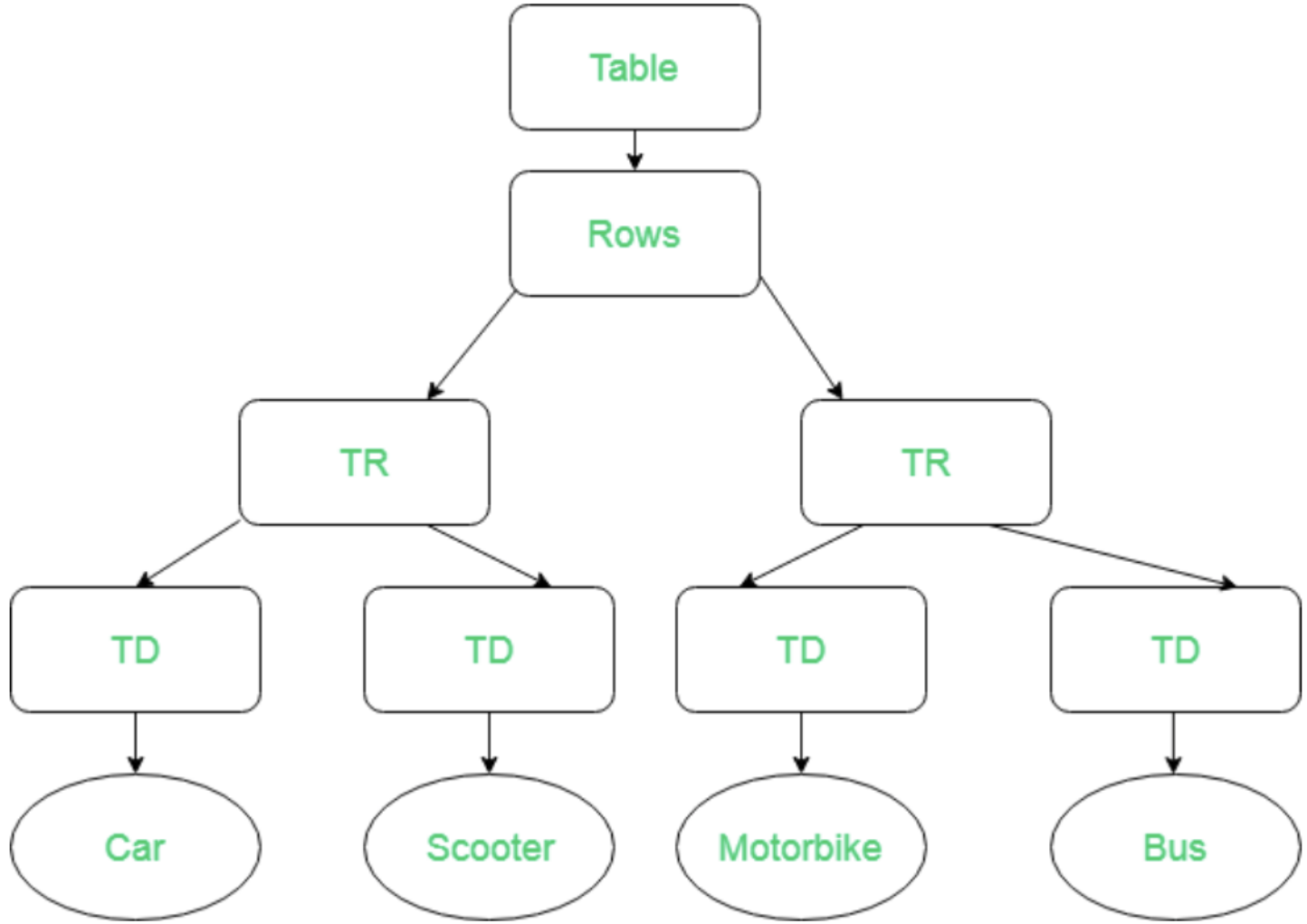
```
document.getElementById("demo").innerHTML="Hello World!";
```

- getElementById is a **method**, while innerHTML is a **property**.
- The innerHTML property is useful for getting or replacing the content of HTML elements

The HTML DOM Tree of Objects



```
<table>
  <ROWS>
    <tr>
      <td>Car</td>
      <td>Scooter</td>
    </tr>
    <tr>
      <td>MotorBike</td>
      <td>Bus</td>
    </tr>
  </ROWS>
</table>
```



Document Object Model Manipulation

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <script src="js/script.js"></script>
  </head>
<body>
  <h1 id="title">Lecture 53</h1>

  <p>
    Say hello to
    <input id="name" type="text">
    <button>Say it!</button>
  </p>


  <div id="content"></div>
</body>
</html>
```

script.js

```js
// DOM manipulation
console.log(document.getElementById("title"));
```

Output: null

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
  </head>
<body>
  <h1 id="title">Lecture 53</h1>

  <p>
    Say hello to
    <input id="name" type="text">
    <button>Say it!</button>
  </p>


  <div id="content"></div>

  <script src="js/script.js"></script>
</body>
</html>
```

We have inserted the script at the end

Output:

```
                                        script.js:2
  <h1 id="title">Lecture 53</h1>
```

Reason: We have placed the script tag before the h1 element.
So, when the script executed (sequential execution) there was no HTML element with id="title

Document Object Model Manipulation

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
  </head>
<body>
  <h1 id="title">Lecture 53</h1>

  <p>
    Say hello to
    <input id="name" type="text">
    <button onclick="sayHello();">
      Say it!
    </button>
  </p>


  <div id="content"></div>

  <script src="js/script.js"></script>
</body>
</html>
```

script.js

```js
function sayHello () {
  console.log(
    document.getElementById("name")
  );
}
```

Output:

```
                                        script.js:6
<input id="name" type="text">
```

Dynamic HTML with DOM

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

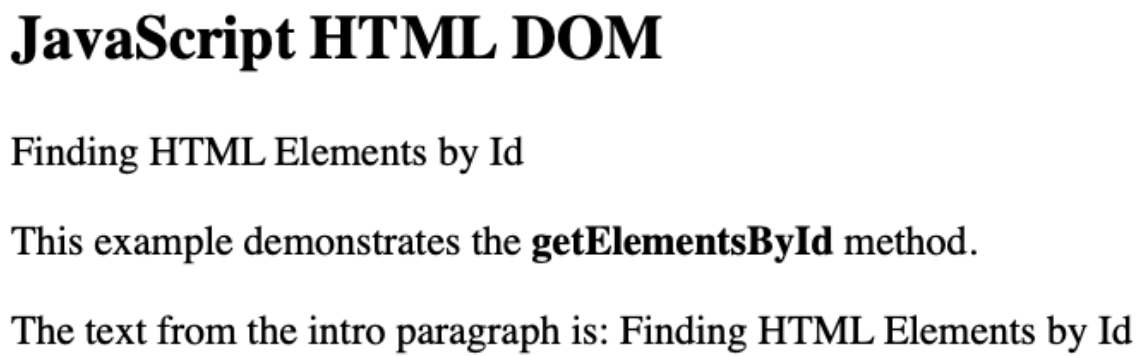
The HTML DOM Document Object

- The document object represents a web page.
- If you want to access any element in an HTML page, you always start with accessing the document object.
- Methods used to access and manipulate HTML are given below:.

Finding HTML Elements

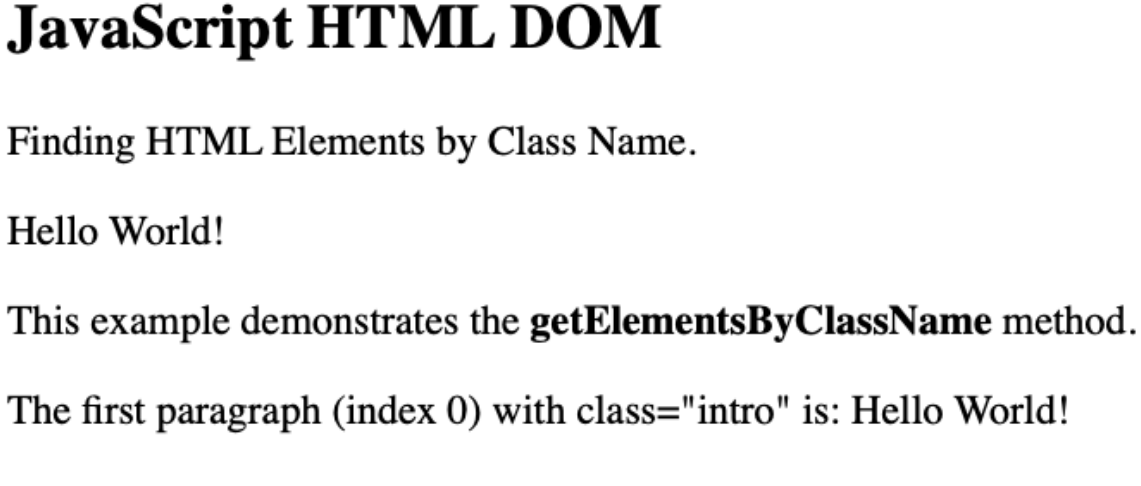
Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

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

<h2>JavaScript HTML DOM</h2>

<p id="intro">Finding HTML Elements by Id</p>
<p>This example demonstrates the <b>getElementsById</b> method.</p>

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

<script>
const element = document.getElementById("intro");

document.getElementById("demo").innerHTML =
"The text from the intro paragraph is: " + element.innerHTML;

</script>

</body>
</html>
```



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

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Tag Name.</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>

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

<script>
const element = document.getElementsByTagName("p");

document.getElementById("demo").innerHTML = 'The text in first paragraph (index 0) is: ' + element[0].innerHTML;

</script>

</body>
</html>
```

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

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Class Name.</p>
<p class="intro">Hello World!</p>
<p class="intro">This example demonstrates the <b>getElementsByClassName</b> method.</p>

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

<script>
const x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>

</body>
</html>
```



Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = *new value*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Finding HTML Objects

Property	Description
document.anchors	Returns all <a> elements that have a name attribute
document.embeds	Returns all <embed> elements
document.forms	Returns all <form> elements
document.head	Returns the <head> element
document.images	Returns all <img> elements
document.implementation	Returns the DOM implementation
document.inputEncoding	Returns the document's encoding (character set)
document.lastModified	Returns the date and time the document was updated

Finding HTML Objects

Property	Description
document.links	Returns all <area> and <a> elements that have a href attribute
document.readyState	Returns the (loading) status of the document
document.referrer	Returns the URI of the referrer (the linking document)
document.scripts	Returns all <script> elements
document.strictErrorChecking	Returns if error checking is enforced
document.title	Returns the <title> element
document.URL	Returns the complete URL of the document

Finding HTML Objects

Property	Description
document.baseURI	Returns the absolute base URI of the document
document.body	Returns the <body> element
document.cookie	Returns the document's cookie
document.doctype	Returns the document's doctype
document.documentElement	Returns the <html> element
document.documentMode	Returns the mode used by the browser
document.documentURI	Returns the URI of the document
document.domain	Returns the domain name of the document server

DHTML

- Dynamic HyperText Markup Language or Dynamic HTML
- DHTML is made up of four languages or components: HTML, CSS, JavaScript, and DOM
- DHTML allows creating dynamic web pages with ease and simplicity.
- DHTML provides dynamic styles by which the appearance and styles of elements can be changed dynamically (Tags and their properties, fonts, color, style, the content of web pages can be changed/altered using DHTML).
- Using DHTML, users can easily create animations and dynamic fonts for their web sites or web pages.
- DHTML provides the facility for using the events, methods, and properties.
- DHTML provides the feature of code reusability.
- DHTML is used for real-time positioning.
- DHTML is also used for data binding.
- DHTML makes a webpage dynamic and be used to create animations, games, applications along with providing new ways of navigating through websites.
- The functionality of a webpage is enhanced due to the usage of low-bandwidth effect by DHTML.
- DHTML also facilitates the use of methods, events, properties, and codes.

HTML	DHTML
HTML is simply a markup language.	DHTML is not a language, but it is a collection of technologies of web development.
It is used for developing and creating web pages.	It is used for creating and designing the animated and interactive web sites or pages.
HTML is used to build a static document page and to specify the hyperlinks.	DHTML describes the technologies used to build dynamic and interactive web pages.
HTML does not contain any server-side scripting code.	DHTML may contain the code of server-side scripting.
The files of HTML are stored with the .html or .htm extension in a system.	The files of DHTML are stored with the .dhtm extension in a system.
A simple page which is created by a user without using the scripts or styles called as an HTML page.	A page which is created by a user using the HTML, CSS, DOM, and JavaScript technologies called a DHTML page.
HTML does not require database connectivity.	DHTML may require database connectivity as it interacts with the user.
HTML pages do not use event methods.	DHTML pages make use of event methods.
HTML provides tags such as <body>, , <form>, etc., to control the presentation of information on the web pages.	DHTML enables the incorporation of minor animations and dynamic menus into Web pages. It employs events, methods, and properties to provide dynamism to HTML pages.
HTML does not permit changes to the current pages without returning to the webserver first.	DHTML allows you to modify the current pages at any time. Without initially returning to the webserver.

Advantages of DHTML

- DHTML allows creating more interactive dynamic web pages as compared to static web pages.
- As compared to other multimedia software like Flash and Shockwave, DHTML has a compact size of files and can be downloaded quickly on the client machine, thus saving bandwidth and time.
- The content can be changed on the client machine without the need for reloading or refreshing the web page.
- It is supported by almost all browsers on today's date, viewer requires no extra plug-ins for browsing through the webpage that uses DHTML and there is no need for any extra software to run.
- It is efficient, and files are maintainable. Web designers can have more flexibility due to this.
- Highly flexible and easy to make changes.
- User time is saved by sending less number of requests to the server. As it is possible to modify and replace elements even after a page is loaded, it is not required to create separate pages for changing styles which in turn saves time in building pages and also reduces the number of requests that are sent to the server.
- It has more advanced functionality than a static HTML. it is capable of holding more content on the web page at the same time.

Disadvantages of DHTML

- As DHTML combines multiple technologies such as HTML, CSS, JS. Web designers require command all over them.
- It is required to take care of a web page to be properly working in different browsers.
- The browser compatibility check is a must while using DHTML. It is not guaranteed that DHTML will perform the same on every platform.
- It is not supported by all the browsers.
- Implementation of different browsers are different. So, if it worked in one browser, it might not necessarily work the same way in another browser.
- Even after being great with functionality, DHTML requires a few tools and utilities that are some expensive. For example, the DHTML text editor, Dreamweaver. Along with it the improvement cost of transferring from HTML to DHTML makes cost rise much higher.

String Handling in DHTML

```
<html>
<head>
<script>
        var x = "DHTML";
        var len = x.length;
        document.write("The length of the string is: " + len + "<br>");

        var ch = x.chatAt(0);
        document.write("Char at 0th position is: " + ch + "<br>");

        var pos = x.indexOf("M");
        document.write("The position od M is: " + pos + "<br>");
</script>
</head>
<body>
        <p>String Handling in DHTML</p>
</body>
</html>
```

Numbers in DHTML

```
<html>
<head><script>
        var age = prompt("Enter your age: ");
        if (isNaN(age)) {
                alert("Age should be numeric.");
        }
        else {
                alert("Your age is: " + age);
        }
</script></head>
<body>
        <p>Numbers in DHTML</p>
</body>
</html>
```

Events in DHTML

- An event is defined as changing the occurrence of an object.
- The event is a term in the HTML, which triggers the actions in the web browsers.
- Events are the actions that the user performs when they visit your page.
- It is compulsory to add the events in the DHTML page. Without events, there will be no dynamic content on the HTML page.
- When an event happens it triggers objects (through JavaScript code) that are associated with that kind of event
- Actually, the event handlers catch these events performed by the user and then execute the code in response.
- **Example of events:**
  1. Click a button.
  2. Move the mouse around
  3. Submitting a form.
  4. An image loading or a web page loading, etc.

Types of Events

- Events can be classified into 4 categories:

1. Window Events
2. Mouse Events
3. Keyboard Events
4. Form Events

Window & Mouse Events

- Window Events
  - The Window itself has its own events, which trigger when a new page is starting up (onLoad), shutting down (onUnload),being resized (onResize), moved (onMove), canceled (onAbort) or when an error occurs (onError).
  - There is also an event triggered when the window moves to foreground (onFocus) or changes to background (onBlur).
- Mouse Events
  - The mouse has a few events associated with it when a button is pressed (onMousedown) on top of an element and when it is released (onMouseup);
  - When the mouse moves and the pointer is already over an element (onMousemove) and when it moves away from the element (onMouseout).
  - Events are triggered also when the pointer is over an element (onMouseover) and when it is clicked once (onClick) or twice (onDblclick).

Keyboard & Form Events

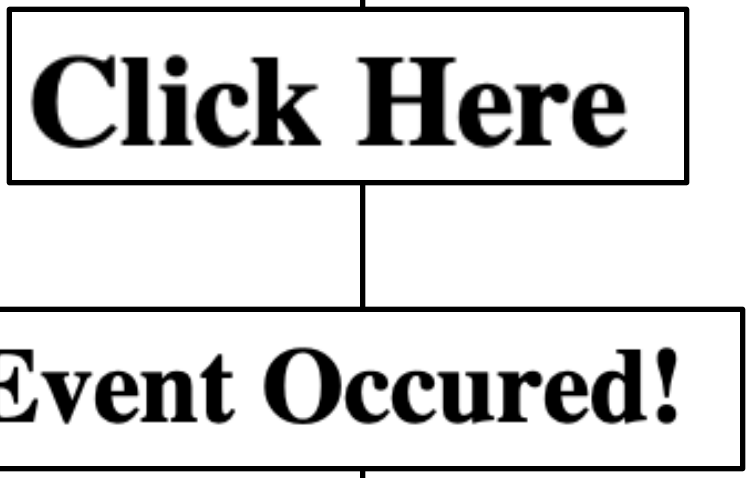
- Keyboard Events
  - Events can be triggered when the key is pressed down (onKeydown) and when it is released (onKeyup).
  - The complete key sequence, down press and up release, triggers another event (onKeypress).
- Form Events
  - Events can be triggered when the reset button on the form is clicked (onReset) or when the submit button is clicked (onSubmit).
  - Even when a content is selected on a page (onSelect) event is generated.

S.No.	Event	When it occurs
1.	onabort	It occurs when the user aborts the page or media file loading.
2.	onblur	It occurs when the user leaves an HTML object.
3.	onchange	It occurs when the user changes or updates the value of an object.
4.	onclick	It occurs or triggers when any user clicks on an HTML element.
5.	ondblclick	It occurs when the user clicks on an HTML element two times together.
6.	onfocus	It occurs when the user focuses on an HTML element. This event handler works opposite to onblur.
7.	onkeydown	It triggers when a user is pressing a key on a keyboard device. This event handler works for all the keys.
8.	onkeypress	It triggers when the users press a key on a keyboard. This event handler is not triggered for all the keys.
9.	onkeyup	It occurs when a user released a key from a keyboard after pressing on an object or element.
10.	onload	It occurs when an object is completely loaded.

S.No.	Event	When it occurs
11.	onmousedown	It occurs when a user presses the button of a mouse over an HTML element.
12.	onmousemove	It occurs when a user moves the cursor on an HTML object.
13.	onmouseover	It occurs when a user moves the cursor over an HTML object.
14.	onmouseout	It occurs or triggers when the mouse pointer is moved out of an HTML element.
15.	onmouseup	It occurs or triggers when the mouse button is released over an HTML element.
16.	onreset	It is used by the user to reset the form.
17.	onselect	It occurs after selecting the content or text on a web page.
18.	onsubmit	It is triggered when the user clicks a button after the submission of a form.
19.	onunload	It is triggered when the user closes a web page.

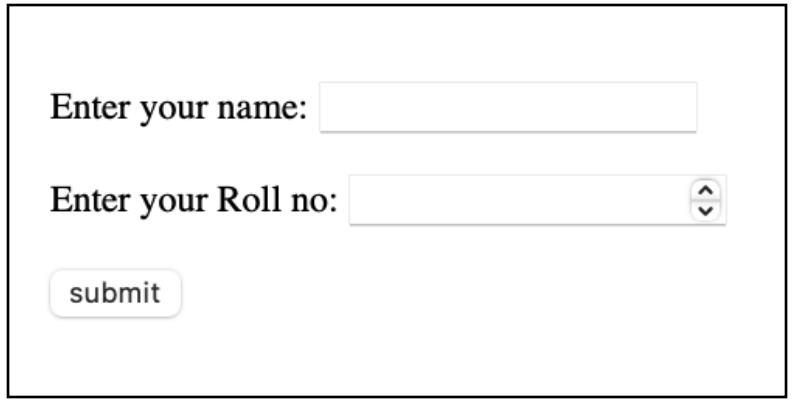
Handlers	Can be used with these tags:
onAbort	images
onBlur	windows, all form elements, frames
onClick	buttons, radio buttons,checkboxes, submit buttons, links
onChange	text fields, textareas, select lists
onError	windows, images
onFocus	windows, frames, and all form elements
onLoad	body, images
onMouseover	areas, links
onMouseout	links
onReset	forms
onSelect	text fields, textareas
onSubmit	submit button
onUnload	body

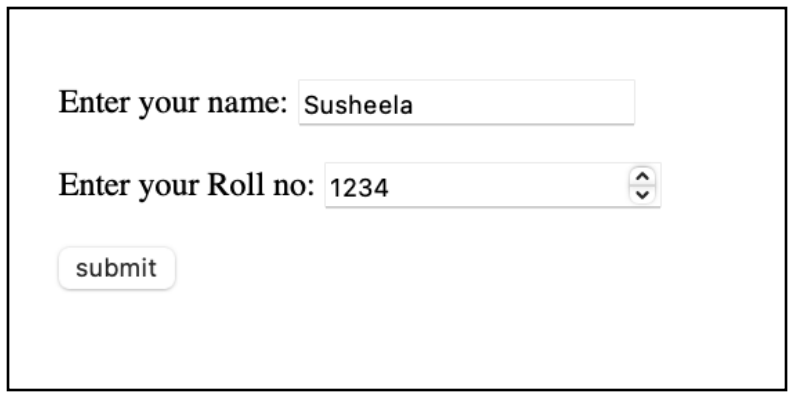
```
<html>
<head>
<script type="text/javascript">
function changetext(id)
{
id.innerHTML="Event Occured!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click Here</h1>
</body>
</html>
```

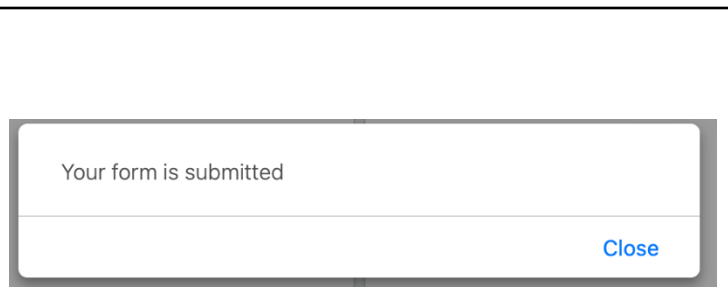


```
<html>
<head>
    <title>
        Example of onclick event
    </title>
    <script>
        function ChangeText(ctext)
        {
            ctext.innerHTML=" Events in DHTML! ";
        }
    </script>
</head>
<body>
    <font color="red"> Click on the Given text for changing it: <br>
    </font>
    <font color="blue">
    <h1 onclick="ChangeText(this)"> Hello World! </h1>
    </font>
</body>
</html>
```

```html
<html>
<head>
    <title>
        Example of onsubmit event
    </title>
    <script>
        function Submit_Form()
        {
            alert(" Your form is submitted");
        }
    </script>
</head>
<body>
    <form onsubmit="Submit_Form()">
        <label> Enter your name: </label>
        <input type="text" name = "name">  <br><br>
        <label> Enter your Roll no: </label>
        <input type="Number" name = "roll">  <br><br>
        <input type="submit" value="submit">
    </form>
</body>
</html>
```

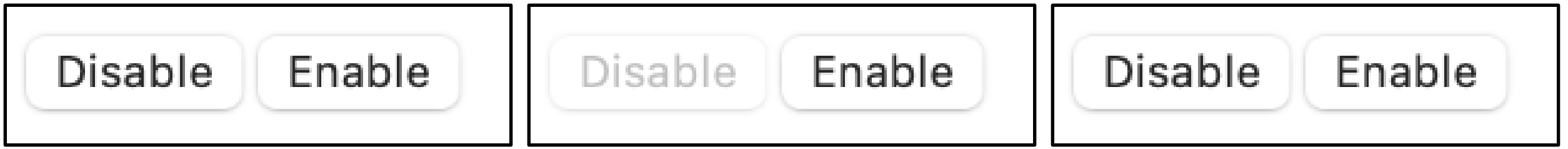






```
<html>
<body>
<button onclick="this.disabled='true'; alert(this.isDisabled);">Disable Me</button>
</body>
</html>
</html>
```

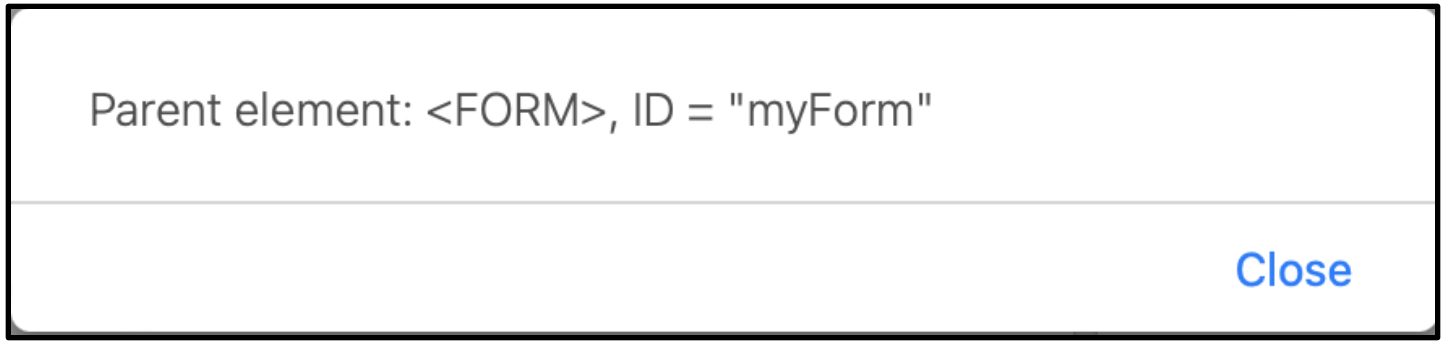
```html
<html>
<body>
<script>
    function function1() {
        document.all.myButton.disabled = true;
    }
    function function2() {
        document.all.myButton.disabled = false;
    }
</script>
<input id="myButton" type="button" value="Disable" onClick="function1();">
<input type="button" value="Enable" onClick="function2();">
</body>
</html>
```



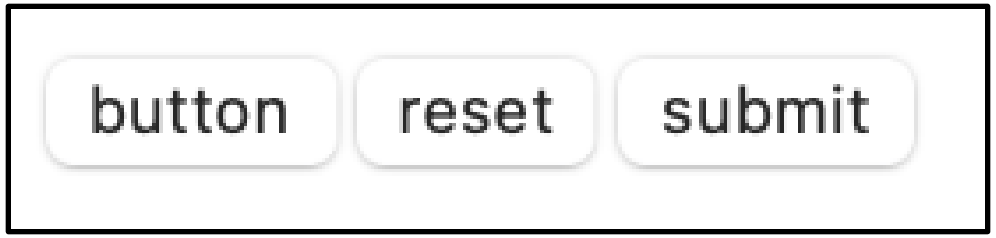
Button Parent Element

```
<html>
<body>
<form id="myForm">
<input id="myButton"
       type="button" value="Show my parent" onclick="function1();">
</form>
<script language="JavaScript">
    function function1() {
        var m = document.all.myButton.parentElement.id;
        alert("Parent element: <FORM>, ID = " + '"' + m + '"');
    }
</script>
```

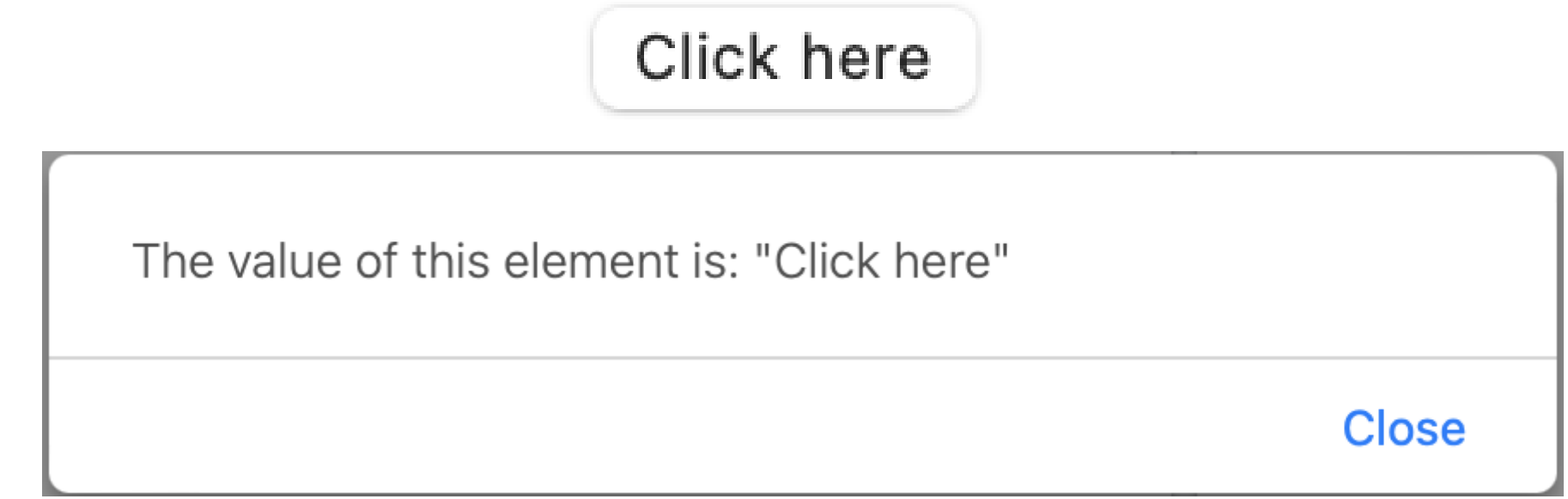
```
<script>
function function1(form) {
    var elem = document.getElementById("myButton").form.id;
    alert(elem);
    return true;
}
</script>
```



```
<html>
<body>
<input type="button"
       value="button"
       onClick="alert('&quot;'+this.type+'&quot;');">
<input type="reset"
       value="reset"
       onClick="alert('&quot;'+this.type+'&quot;');">
<input type="submit"
       value="submit"
       onClick="alert('&quot;'+this.type+'&quot;');">
</body>
</html>
```



```
<html>
<body>
<script>
    function function1() {
      var m = document.getElementById("myButton").defaultValue;
      alert("The value of this element is: "+'"'+m+'"');
    }
</script>
<input id="myButton" type="button" value='Click here' onclick="function1();">
</body>
</html>
```



jQuery

- jQuery is a feature-rich, short, and compact JavaScript library.
- It's cross-platform and works with a variety of browsers.
- jQuery has changed the way a lot of people write JavaScript because of its versatility and extensibility.
- It's also known as "write less, do more." because it links a lot of common functions that require a large number of lines of JavaScript code to be converted into methods that may be invoked with a single line of code whenever they're needed.
- jQuery has changed the way a lot of people write JavaScript because of its versatility and extensibility.
- The primary goal of jQuery is to make it simple to use JavaScript on your website to make it more interactive and appealing. It's also utilized to add animation.
- It is also used to simplify a lot of the more challenging aspects of JavaScript, such as AJAX calls and DOM manipulation.

Features of jQuery

- HTML manipulation
- DOM manipulation
- DOM element selection
- CSS manipulation
- Effects and Animations
- Utilities
- AJAX
- HTML event methods
- JSON Parsing
- Extensibility through plug-ins

jQuery

- jQuery is relatively easier and faster to code than JavaScript.
- JavaScript Code Snippet:

```
function backgroundColorChange(color)
{
    document.body.style.background = color;
}
Onload = changeColor('green’);
```

- jQuery Syntax:

```
$('body').css('background', 'green');
```

Advantages of jQuery

- Simple to Code
- Easily to Integrated with other IDE (Visual Studio and NuGet)
- Easy animation
- Fast (due to less lines of code)
- SEO friendly
- Run in all major browsers

XML

- Extensible Markup Language
- XML is a markup language used to describe the content and structure of data in a document.
- It is a simplified version of Standard Generalized Markup Language (SGML).
- XML is an industry standard for delivering content on the Internet.
- XML is also extensible, because it provides a facility to define new tags, XML is also extensible.
- Like HTML, XML uses tags to describe content. However, rather than focusing on the presentation of content, the tags in XML describe the meaning and hierarchical structure of data.
- This functionality allows for the sophisticated data types that are required for efficient data interchange between different programs and systems.
- XML is portable across heterogeneous systems, because it enables separation of content and presentation, the content, or data,
- The XML syntax uses matching start and end tags (such as <name> and </name>) to mark up information. Information delimited by tags is called an element.

XML

- XML stands for Extensible Markup Language
- XML is a markup language much like HTML.
- XML was designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is designed to be self-descriptive.
- XML is W3C recommendation.
- XML was developed by W3c in 1996.
- XML 1.0 was officially adopted as a W3c recommendation in 1998.

Goals of XML

- The user must to able to define and use his own tags.
- Allows the user to build his own tag library, based on his web requirement.
- Allow user to define the formatting rules for the user defined tags.
- XML must support storage or transport of data.

XML Syntax

- XML tags are as simple as in HTML but in XML tags are more flexible.
- There are no predefined XML tags, You must define your personalized tags (user defined tags).
- The simple XML document must contain opening tag and closing tag.
- The XML tags are case sensitive.
- The XML tags are used to define the scope of elements in XML document.
- The first element of XML document is called root element.

XML

- The XML syntax uses matching start and end tags (such as <name> and </name>) to mark up information. Information delimited by tags is called an element.
- Every XML document has a single root element, which is the top-level element that contains all the other elements.
- Elements that are contained by other elements are often referred to as sub-elements.
- An element can optionally have attributes, structured as name-value pairs, that are part of the element and are used to further define it.
- XML is called universal data interchange language
  - It provides a simple and universal way of storing any textual data. Data stored in XML documents can be electronically distributed and processed by any number of different applications. These applications are relatively easy to write because of the standard way in which the data is stored. Therefore, XML is a universal data interchange language.

XML Prolog

- The first part of a document is the prolog.
- It is optional so you won’t see it every time, but if it does exist it must come first.
- The prolog begins with an XML declaration which, in its simplest form, looks like the following:

Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
```

- This declaration contains the version number (either 1.0 or 1.1) and also contain information about the encoding used in the document. Here the encoding is specified as UTF-8.
-

XML Elements

- Elements are the basic building blocks of XML and all documents will have at least one element called root element that is the parent of all other elements.
- Once the XML prolog is finished you need to create the root element of the document. Everything else in the document lies under this element to form a hierarchical tree.
- The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree. All elements can have sub elements (child elements).

```
<department>
        <student>
                <name>.....</name>
                <rollno>....</rollno>
        </student>
</department>
```

XML Attributes

- Attributes are name-value pairs associated with an element.
- Attributes are part of XML elements.
- An element can have multiple unique attributes.
- Attribute gives more information about XML elements, they define properties of elements.

```
<element-name attribute1="value1" attribute2="value2">
        ....content..
</element-name>
```

Attributes Rules

- Attributes consist of a name and a value separated by an equals sign. The name follows the same rules as element names.
- The attribute value must be in quotes. You can use either single or double quotes, but you can’t mix them in a single attribute
- There must be a value part, even you cant left value just empty.
- Attribute names must be unique per element.

```
<department>
        <student gender="female">
                <name>.....</name>
                <rollno>....</rollno>
        </student>
</department>
```

- The following sample XML file describes the contents of an address book:

```
<?xml version="1.0"?>

<address_book>
        <person gender="f">
                <name>Jane Doe</name>
                <address>
                        <street>123 Main St.</street>
                        <city>San Francisco</city>
                        <state>CA</state>
                        <zip>94117</zip>
                </address>
                <phone area_code=415>555-1212</phone>
        </person>
        <person gender="m">
                <name>John Smith</name>
                <phone area_code=510>555-1234</phone>
                <email>johnsmith@somewhere.com</email>
        </person>
</address_book>
```

- The root element - address_book.
- The address book currently contains two entries in the form of person elements: Jane Doe and John Smith.

Document Type Declaration(DTD)

- DTD is an XML technique used to define the structure of a XML document.
- It's useful to make sure the structure and vocabulary of XML documents are valid against the grammatical rules of the appropriate XML language.
- A DTD can contain declarations that define elements, attributes, notations, and entities for any XML files that reference the DTD file.
- It also establishes constraints for how each element, attribute, notation, and entity can be used within any of the XML files that reference the DTD file.
- To be considered a valid XML file, the document must be accompanied by a DTD (or an XML schema) and conform to all of the declarations in the DTD (or XML schema).
- DTDs consist of three basic parts:
  - Elements declarations
  - Attributes declarations
  - Entities declarations

XML Schema

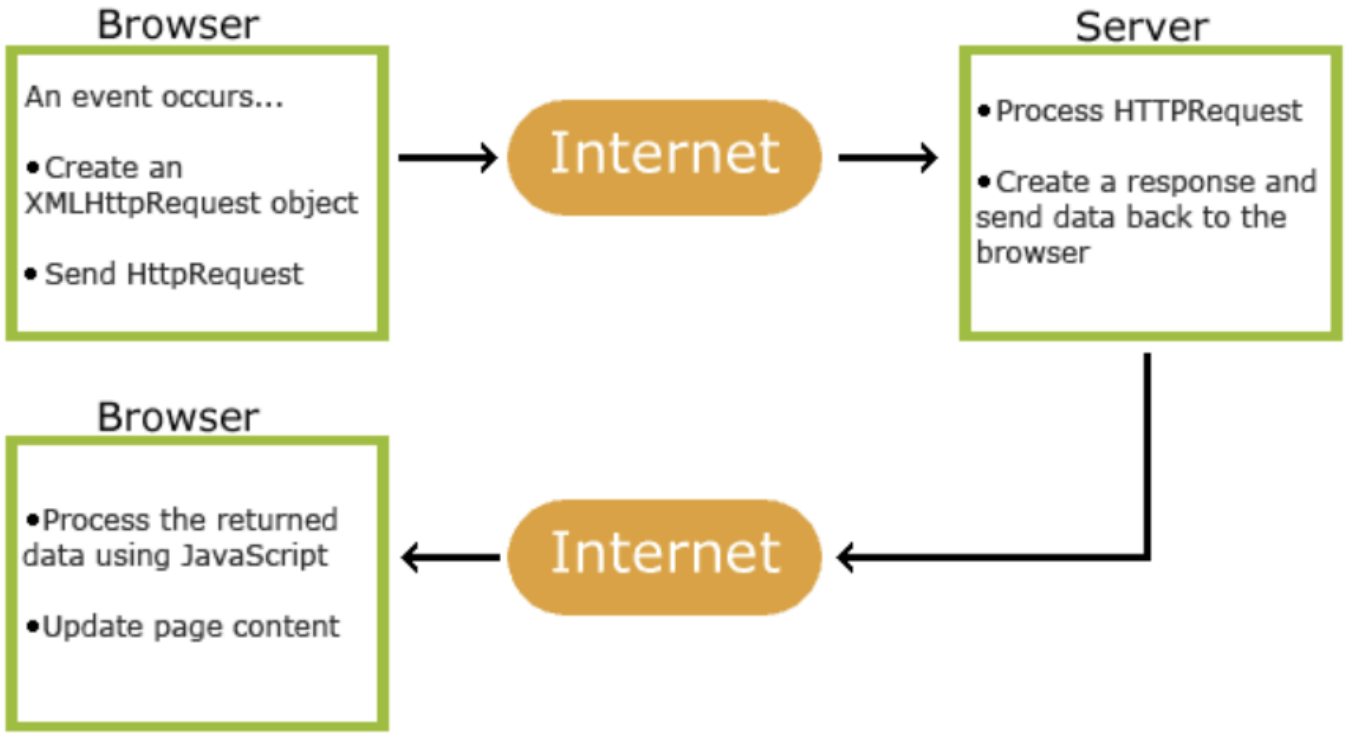
- The short name for XML Schema is XSD(XML Schema Definition).
- An XML schema is a document or a set of documents that defines the structure and legal elements of the XML file. XML schemas can be used to ensure that an XML file is syntactically correct and conforms to the defined schema.
- XML Schema have a number of advantages over DTDs.
- Schema use XML syntax, so the same tools for processing XML data (e.g., parsers, DOM, SAX, etc.) can be used for processing Schema.
- Schema support a richer set of data types, allows users to derive their own data types.
- Schema allow more flexible and powerful definitions of content models.
- Schema are extensible, supporting easier reuse of parts of schemas in other schemas (e.g., easier use of Namespaces).
- Schema have annotation elements that provide greater documentation of structural design.

Ajax (Asynchronous JavaScript **A**nd **X**ML)

- AJAX is not a programming language.
- AJAX is a web browser technology independent of web server software.
- AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.
- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- With AJAX, when you click on submit button, JavaScript will make a request to the server, interpret the results, and update the current screen (rather than directed to a new page). The user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.
- AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
- JavaScript object – XMLHttpRequest, performs asynchronous interaction with the server.

Working of Ajax

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript



Advantages of Ajax

1. Speed is enhanced as there is no need to reload the page again.
2. AJAX make asynchronous calls to a webserver; this means client browsers avoid waiting for all the data to arrive before starting of rendering.
3. Form validation can be done successfully through it.
4. Bandwidth utilization – It saves memory when the data is fetched from the same page.
5. More interactive.

Disadvantages of Ajax

1. Ajax is dependent on JavaScript. If there is some JavaScript problem with the browser or in the OS, Ajax will not support.
2. Ajax can be problematic in Search engines as it uses JavaScript for most of its parts.
3. Source code written in AJAX is easily human readable. There will be some security issues in Ajax.
4. Debugging is difficult.
5. Problem with browser back button when using AJAX enabled pages.

REFERENCES

- https://www.w3schools.com/html/tryit.asp?filename=tryhtml_default_default
- https://www.tutorialspoint.com/javascript/javascript_form_validations.htm
- http://www.java2s.com/Code/JavaScript/Form-Control/Button.htm
- https://www.codingninjas.com/codestudio/library/jquery-introduction
- https://docs.oracle.com/cd/E11035_01/wls100/xml/intro.html