

Unit 4: Client Side Scripting with Javascript

Introduction

Javascript (JS) is a scripting languages, primarily used on the Web. It is used to enhance HTML pages and is commonly found embedded in HTML code. JavaScript is an interpreted language. Thus, it doesn't need to be compiled. JavaScript renders web pages in an interactive and dynamic fashion. This allowing the pages to react to events, exhibit special effects, accept variable text, validate data, create cookies, detect a user's browser, etc.

There are two ways to use JavaScript in an HTML file. The first one involves embedding all the JavaScript code in the HTML code, while the second method makes use of a separate JavaScript file that's called from within a Script element, i.e., enclosed by Script tags. JavaScript files are identified by the .js extension. Although JavaScript is mostly used to interact with HTML objects, it can also be made to interact with other non-HTML objects such as browser plugins, CSS (Cascading Style Sheets) properties, the current date, or the browser itself.

Structure of Javascript Program

JavaScript is written in plain text. All the Javascript program are written inside the `<script type="text/javascript">` tag. The `<script type="text/javascript">` tag tells the browser that the code inside this tag are Javascript program and browser render accordingly.

The `<script>` tag is written inside the `<body>` tag of the HTML page. JavaScript supports both C-style and C++-style comments, thus:

- Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence `<!--`. JavaScript treats this as a single-line comment, just as it does the `//` comment.-->
- The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `//-->`.

Variable and Data Types:

Variables are referred as named containers for storing information. We can place data into these containers and then refer to the data simply by naming the container. Here are the important rules that must be followed while declaring a variable in JavaScript.

No particular character identifies a variable in JavaScript as the dollar sign does in PHP. Instead, variables use the following naming rules:

- A variable may include only the letters a-z, A-Z, 0-9, the \$ symbol, and the underscore (_).
- No other characters, such as spaces or punctuation, are allowed in a variable name.

- The first character of a variable name can be only a-z, A-Z, \$, or _ (no numbers).
- Names are case-sensitive. Count, count, and COUNT are all different variables.
- There is no set limit on variable name lengths.

However, var keyword is used to define variable as follows:

Example

```
<script type="text/javascript">
  var money;
  var name, age;
</script>
```

String Variables

JavaScript string variables should be enclosed in either single or double quotation marks, like this:

Example:

```
<script type="text/javascript">>
  sub= 'microprocessor';
  program= "csit";
</script>
```

Numeric Variables

Creating a numeric variable is as simple as assigning a value

Example:

```
<script type="text/javascript">>
  mark = 42;
  percentage = 42;
</script>
```

The following functions are used in numbers in javascript

parseInt():The parseInt() function parses a string and returns an integer.

Syntax:

parseInt(string)

parseFloat(): The parseFloat() function parses a string and returns a floating point number.

Syntax:

parseFloat(string)

Example:

```
<html>
<head>
<html>
<head>
<script type="text/javascript">
  var ten = parseInt("10");
  document.write(ten + " is a integer <br>");
  var a = parseFloat("10.25");
  document.write(a + " is a float");
</script>
</head>
</body>
</html>
```

Output:

10 is a integer
10.25 is a float

eval()

This function evaluates or executes an arguments

Syntax:

eval(string)

Example:

```
<html>
<head>
  <script>
    var x = 10;
    var y = 20;
    var a = eval("x * y") + "<br>";
    var b = eval("2 + 2") + "<br>";
    var c = eval("x + 17") + "<br>";

    var res = a + b + c;
    document.write(res);
```

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

Output:

200
4
27

Variables can be initialized at time of declaration or after declaration as follows

```
<script type="text/javascript">  
    var name = "Ali";  
    var money;  
    money = 2000.50;  
</script>
```

Data Types:

The following are the data types of Javascript

- **String:** Can contain groups of character as single value. It is represented in double quotes. E.g. `var x= "Hello"`.
- **Numbers:** Contains the numbers with or without decimal. E.g. `var x=44, y=44.56;`
- **Booleans:** Contain only two values either true or false. E.g. `var x=true, y= false.`
- **Undefined:** Variable with no value is called Undefined. E.g. `var x;`
- **Null:** If we assign null to a variable, it becomes empty. E.g. `var x=null;`
- **Array:** Can contain groups of values of same type. E.g. `var x={1,2,3,55};`
- **Objects:** Objects are stored in property and value pair. E.g. `var rectangle = { length: 5, breadth: 3};`

Statements

In a computer language, a group of words, numbers, and operators that performs a specific task is a statement. But in JavaScript, a statement might look as follows:

```
var a = var b * 2;
```

The characters `a` and `b` are called variables which are like simple boxes you can store any of your stuff in. In programs, variables hold values (like the number 42) to be used by the program. Think of them as symbolic placeholders for the values themselves. By contrast, the `2` is just a value itself, called a literal value, because it stands alone without being stored in a variable. The `=` and `*` characters are operators, they perform actions with the values and variables such as assignment and mathematic multiplication. Most statements in JavaScript conclude with a semicolon (`;`) at the end. The statement `a = b * 2;` tells the computer, roughly, to get the current value stored in the variable `b`, multiply that value by 2, then store the result back into another variable we call `a`.

Expressions:

Statements are made up of one or more expressions. An expression is any reference to a variable or value, or a set of variable(s) and value(s) combined with operators.

For example:

```
a = b * 2;
```

This statement has four expressions in it:

- 2 is a literal value expression.
- b is a variable expression, which means to retrieve its current value.
- b * 2 is an arithmetic expression, which means to do the multiplication.
- a = b * 2 is an assignment expression, which means to assign the result of the b * 2 expression to the variable a (more on assignments later).

A general expression that stands alone is also called an expression statement, such as the following:

```
b * 2;
```

This flavor of expression statement is not very common or useful, as generally it wouldn't have any effect on the running of the program it would retrieve the value of b and multiply it by 2, but then wouldn't do anything with that result.

Keywords:

Keywords are reserved and cannot be used as variable or function names. Here is the complete list of JavaScript keywords:

break, case, catch, continue, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with

Block Statement in JavaScript

A block statement groups zero or more statements. In languages other than JavaScript, it is known as a compound statement.

Here's the syntax:

```
{  
    //List of Statements  
}
```

Variables with a block get scoped to the containing function. Block statement never introduce scope and using var to declare variables don't have block scope.

Example:

```
var a= 20; {  
    var b= 40;
```

}

Now, when you will print the value of a, it will print 40, not 20. This is because variable declared with a var within the block has the same scope like var before the block.

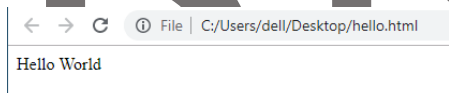
Printing Statements or Variables in JavaScript

To write a strings of text to a document use, document.write(). This function can also be used to write the variables directly in the HTML. The write() method writes HTML expressions or JavaScript code to a document.

Example 1: Printing string using document.write()

```
<script type="text/javascript">
  document.write("Hello World");
</script>
```

Output:



Example 2: Printing variable using document.write()

```
<script type="text/javascript">
  var world = "Hello World"
  document.write(world);
</script>
```

In the above two example the output is same the only difference is inside the document.write() function, in the example 1 the Hello World is a string which is kept inside the double quoted but in example 2 the Hello World is stored in variable named as world and the world variable is printed without using any quoted marks. i.e. double quote or single quote.

Operators

Operators in JavaScript, as in PHP, can involve mathematics, changes to strings, and comparison and logical operations (and, or, etc.). JavaScript mathematical operators look a lot like plain arithmetic for instance, the following statement outputs 7:

Example

```
<script type="text/javascript">
  document.write(2+5);
</script>
```

Arithmetic Operators

Arithmetic operators are used to perform mathematics. It can be used for the main four operations (addition, subtraction, multiplication, and division) as well as to find the modulus (the remainder after a division) and to increment or decrement a value.

The following table shows the arithmetic operators list

Operator	Description	Example
+	Addition	j + 12
-	Subtraction	j - 22
*	Multiplication	j * 7
/	Division	j / 3.13
%	Modulus remainder) (division	j % 6
++	Increment	++j
--	Decrement	

Assignment Operators

The assignment operators are used to assign values to variables. They start with the very simple =, and move on to +=, -=, and so on. The operator += adds the value on the right side to the variable on the left, instead of totally replacing the value on the left. The following table lists the various assignment operator variable

Operator	Example	Equivalent to
=	<code>j = 99</code>	<code>j = 99</code>
+=	<code>j += 2</code>	<code>j = j + 2</code>
+=	<code>j += 'string'</code>	<code>j = j + 'string'</code>
-=	<code>j -= 12</code>	<code>j = j - 12</code>
*=	<code>j *= 2</code>	<code>j = j * 2</code>
/=	<code>j /= 6</code>	<code>j = j / 6</code>
%=	<code>j %= 7</code>	<code>j = j % 7</code>

Comparison Operators

Comparison operators are generally used inside a construct such as an if statement, where you need to compare two items. For example, you may wish to know whether a variable you have been incrementing has reached a specific value, or whether another variable is less than a set value, and so on. The following table shows the list of comparison operators used in JavaScript:

Operator	Description	Example
==	Is <i>equal</i> to	<code>j == 42</code>
!=	Is <i>not equal</i> to	<code>j != 17</code>
>	Is <i>greater than</i>	<code>j > 0</code>
<	Is <i>less than</i>	<code>j < 100</code>
>=	Is <i>greater than or equal</i> to	<code>j >= 23</code>
<=	Is <i>less than or equal</i> to	<code>j <= 13</code>
===	Is <i>equal</i> to (and of the same type)	<code>j === 56</code>
!==	Is <i>not equal</i> to (and of the same type)	<code>j !== '1'</code>

Logical Operators

Unlike PHP, JavaScript's logical operators do not include and and or equivalents to `&&` and `||`, and there is no xor operator. The following table shows the list of logical operator used in JavaScript

Operator	Description	Example
<code>&&</code>	And	<code>j == 1 && k == 2</code>
<code> </code>	Or	<code>j < 100 j > 0</code>
<code>!</code>	Not	<code>! (j == k)</code>

Incrementing, Decrementing, and Shorthand Assignment

The following forms of post- and pre-incrementing and decrementing you learned to use in PHP are also supported by JavaScript, as are shorthand assignment operators:

```
++x
--y
x += 22
y -= 3
```

String Concatenation

JavaScript handles string concatenation slightly differently from PHP. Instead of the `.` (period) operator, it uses the plus sign (+), like this:

```
document.write("You have " + messages + " messages.")
```

Example:

```
<script type="text/javascript">
  var name = "John";
  document.write("Hello " + name + ".");
</script>
```

Output:

```
Hello John.
```

Escape Characters

Escape characters, which you've seen used to insert quotation marks in strings, can also be used to insert various special characters such as tabs, newlines, and carriage returns. Here is an example using tabs to lay out a heading—it is included here merely to illustrate escapes, because in web pages, there are better ways to do layout.

Example

```
<script type="text/javascript">
  var title = "Name\tAge\tLocation"
  document.write(title);
</script>
```

Output

Name Age Location

Here the backslash(\) is escaped. The following table shows the JavaScript's escape characters

Character	Meaning
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	Newline
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\'</code>	Single quote (or apostrophe)
<code>\"</code>	Double quote
<code>\\</code>	Backslash
<code>\XXX</code>	An octal number between 000 and 377 that represents the Latin-1 character equivalent (such as <code>\251</code> for the © symbol)
<code>\xXX</code>	A hexadecimal number between 00 and FF that represents the Latin-1 character equivalent (such as <code>\xA9</code> for the © symbol)
<code>\uXXXX</code>	A hexadecimal number between 0000 and FFFF that represents the Unicode character equivalent (such as <code>\u00A9</code> for the © symbol)

DRAFT

Flow Controls

Conditionals alter program flow. They enable you to ask questions about certain things and respond to the answers you get in different ways. There are three types of non-looping conditionals: the if statement, the switch statement, and the ? operator.

The if Statement

Several examples in this chapter have already made use of if statements. The code within such a statement is executed only if the given expression evaluates to true.

Syntax:

```
if(condition){  
  //statements  
}
```

The else Statement

When a condition has not been met, you can execute an alternative by using an else statement

Syntax:

```
if(condition){  
  //statement  
}  
else{  
  //statement  
}
```

JavaScript has no elseif statement, but that's not a problem because you can use an else followed by another if to form the equivalent of an elseif statement

Syntax:

```
if(condition){  
  //statement  
}  
else{  
  //statement  
}
```

Example:

```
<script type = "text/javascript">  
if (a > 100)  
{  
  document.write("a is greater than 100")  
}  
else if(a < 100)
```

```

{
  document.write("a is less than 100")
}
else
{
  document.write("a is equal to 100")
}
</script>

```

The switch Statement

The switch statement is useful when one variable or the result of an expression can have multiple values, and you want to perform a different function for each value.

It works by passing a single string to the main menu code according to what the user requests. Let's say the options are Home, About, News, Login, and Links, and we set the variable `page` to one of these according to the user's input.

Syntax:

```

switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}

```

Example:

```

<script type = "text/javascript">
switch (page)
{
  case "Home":
    document.write("You selected Home")
    break
  case "About":
    document.write("You selected About")
    break
  case "News":
    document.write("You selected News")
    break
  case "Login":
    document.write("You selected Login")

```

```
break
case "Links":
    document.write("You selected Links")
    break
}
</script>
```

The variable page is mentioned only once at the start of the switch statement. Thereafter, the case command checks for matches. When one occurs, the matching conditional statement is executed. Of course, a real program would have code here to display or jump to a page, rather than simply telling the user what was selected.

Breaking out

The break command allows your code to break out of the switch statement once a condition has been satisfied. Remember to include the break unless you want to continue executing the statements under the next case.

Default Action

When no condition is satisfied, you can specify a default action for a switch statement by using the default keyword.

The ? Operator

The ternary operator (?), combined with the : character, provides a quick way of doing if...else tests. With it you can write an expression to evaluate, and then follow it with a ? symbol and the code to execute if the expression is true. After that, place a : and the code to execute if the expression evaluates to false.

Example:

```
<script type = "text/javascript">
<script>
    document.write(
        a <= 5 ?
        "a is less than or equal to 5" :
        "a is greater than 5"
    )
</script>
</script>
```

Looping in Javascript

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

There are mainly two types of loops:

1. **Entry Controlled loops:** In this type of loops the test condition is tested before entering the loop body. For Loop and While Loop are entry controlled loops.
2. **Exit Controlled Loops:** In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. do – while loop is exit controlled loop.

3.

JavaScript mainly provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time

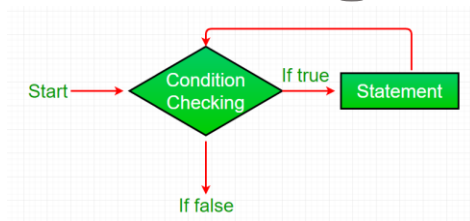
while loop

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax:

```
while(boolean condition)
{
    Loop statements...
}
```

Flow Chart



Explanation:

- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called Entry control loop
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

Example:

```
<script type = "text/javascript">
  // JavaScript program to illustrate while loop
  var x = 1;
  // Exit when x becomes greater than 4
  while (x <= 4)
  {
    document.write("Value of x:" + x + "<br />");
    // increment the value of x for
    // next iteration
    x++;
  }
</script>
```

Output:

```
Value of x:1
Value of x:2
Value of x:3
Value of x:4
```

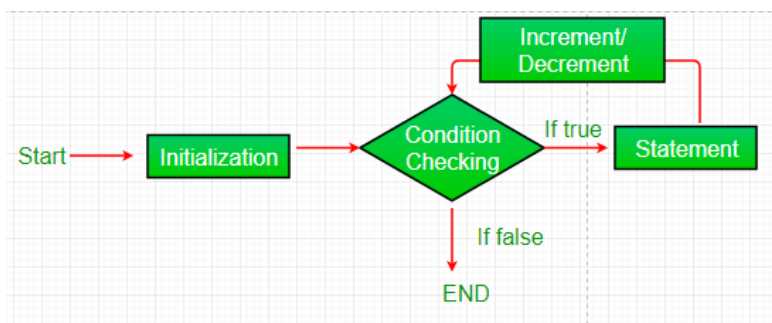
for loop

For loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

```
for (initialization condition; testing condition; increment/decrement)
{
  statement(s)
}
```

Flow Chart:



Explanation:

- **Initialization condition:** Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
- **Testing Condition:** It is used for testing the exit condition for a loop. It must return a Boolean value. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.
- **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
- **Increment/ Decrement:** It is used for updating the variable for next iteration.
- **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

Example

```
<script type = "text/javascript">
// JavaScript program to illustrate for loop
var x;
// for loop begins when x=2
// and runs till x <=4
for (x = 2; x <= 4; x++)
{
    document.write("Value of x:" + x + "<br />");
}
</script>
```

Output:

```
Value of x:2
Value of x:3
Value of x:4
```

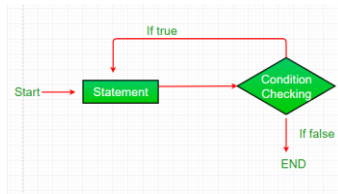
do while:

do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.

Syntax:

```
do
{
    statements..
}
while (condition);
```

Flow Chart



Explanation

- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked, and therefore is an example of exit control loop.

Example:

```
<script type = "text/javascript">
// JavaScript program to illustrate do-while loop
var x = 21;
do
{
    // The line while be printer even
    // if the condition is false
    document.write("Value of x:" + x + "<br />");
    x++;
} while (x < 20);
</script>
```

Output:

Value of x:21

Functions in JavaScript

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax:

```
<script type = "text/javascript">  
    function functionname(parameterlist) {  
        statements  
    }  
</script>
```

To create a function in JavaScript, we have to first use the keyword *function*, separated by name of function and parameters within parenthesis. The part of function inside the curly braces {} is the body of the function. Below are the rules for creating a function in JavaScript:

- Every function should begin with the keyword *function* followed by,
- A user defined function name which should be unique,
- A list of parameters enclosed within parenthesis and separated by commas,
- A list of statement composing the body of the function enclosed within curly braces {}.

Example

```
<script type = "text/javascript">  
function printHello()  
{  
    document.write("Hello");  
}  
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

Example:

```
<script type = "text/javascript">  
function printHello()  
{
```

```
document.write("Hello");
}
printHello();
</script>
```

Output:

Hello

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example:

```
<script type = "text/javascript">
function printParameters(name, age) {
    document.write (name + " is " + age + " years old.");
}
printParameters('John',37);
</script>
```

Output:

John is 37 years old.

The return Statement

A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

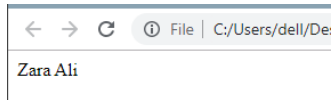
For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example:

```
<script type = "text/javascript">
function concatenateName(first, last) {
    var full;
    full = first + last;
    return full;
}
function printsName() {
```

```
var result;  
result = concatenateName('Zara', 'Ali');  
document.write (result );  
}  
printsName();  
</script>
```

Output:



The following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

Events in Javascript

Popup Boxes

In JavaScript, popup boxes are used to display the message or notification to the user. There are three types of pop up boxes in JavaScript namely Alert Box, Confirm Box and Prompt Box.

Alert Box

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed

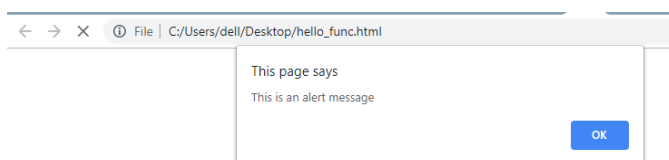
Syntax:

`alert("Alert message")`

Example:

```
<script type = "text/javascript">  
  alert("This is an alert message");  
</script>
```

Output:



Confirmation Box

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method `confirm()` will return true. If the user clicks on the Cancel button, then `confirm()` returns false. You can use a confirmation dialog box as follows.

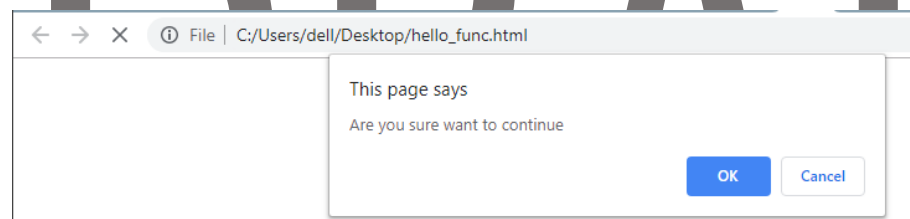
Syntax:

`confirm("Message")`

Example:

```
<script type = "text/javascript">
  confirm("Are you sure want to continue");
</script>
```

Output:



Prompt Box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

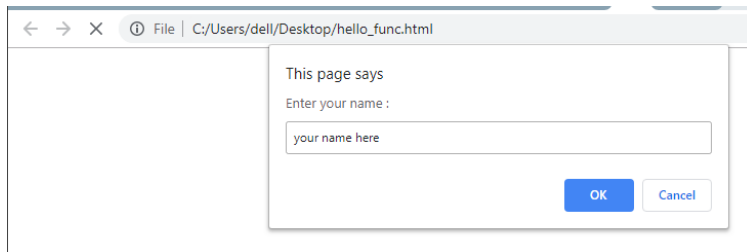
This dialog box is displayed using a method called `prompt()` which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method `prompt()` will return the entered value from the text box. If the user clicks the Cancel button, the window method `prompt()` returns null.

Example

```
<script type = "text/javascript">
  var name = prompt("Enter your name : ", "your name here");
  document.write("You have entered : " + name);
</script>
```

Output:



Object and Properties

Objects, in JavaScript, is its most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript's primitive data-types (Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

- Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.
- An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don't actually store the value.
- Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of "key: value" pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

An object can be created with figure brackets {...} with an optional list of properties. A property is a "key: value" pair, where a key is a string (also called a "property name"), and value can be anything.

Example:

```
<script type = "text/javascript">  
  // Create an object:  
var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};  
</script>
```

In the above example the college is an object and the faculty:"Bsc.Csit", year:"2017", subject:"Web" is called property and property value of the object.

Accessing Object Properties

The object property can be accessed by following ways:

`objectName.propertyName`

or

`objectName["propertyName"]`

Example1: Using `objectName.propertyName`

```
<script type = "text/javascript">
  // Create an object:
  var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};
  // Display some data from the object:
  document.write("The college has " + college.faculty);
</script>
```

Output:



Example 2: Using `objectName["propertyName"]`

```
<script type = "text/javascript">
  // Create an object:
  var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};
  // Display some data from the object:
  document.write("The college has " + college['faculty']);
</script>
```

Output:



Adding New Properties

New properties to an existing object can be added by simply giving a value

Example:

```
<script type = "text/javascript">
  // Create an object:
  var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};
  college.location = "Lalitpur";
```

```
// Display some data from the object:
document.write("The college is located at " + college['location']);
</script>
```

Output:

A screenshot of a web browser window. The address bar shows the file path: file:///C:/Users/dell/Desktop/add_obj.html. The main content area displays the text: "The college is located at Lalitpur".

The college is located at Lalitpur

Deleting Properties

The delete keyword deletes a property from an object.

Example

```
<script type = "text/javascript">
  // Create an object:
var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};
college.location = "Lalitpur";
delete college.faculty;
// Display some data from the object:
document.write("The college is faculty is deleted " + college.faculty);
</script>
```

Output

A screenshot of a web browser window. The address bar shows the file path: file:///C:/Users/dell/Desktop/deleteobj.html. The main content area displays the text: "The college is faculty is deleted undefined".

The college is faculty is deleted undefined

In this example we delete the faculty properties so when we tried to get the property value we get undefined message

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called Object.

The new Operator

The new operator is used to create an instance of an object. To create an object, the new operator is followed by the constructor method.

The Object() Constructor

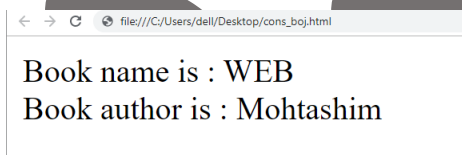
A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called Object() to build the object. The return value of the Object() constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the var keyword.

Example

```
<script type = "text/javascript">
    var book = new Object(); // Create the object
    book.subject = "WEB"; // Assign properties to the object
    book.author = "Mohtashim";
    document.write("Book name is : " + book.subject + "<br>");
    document.write("Book author is : " + book.author + "<br>");
</script>
```

Output



JavaScript Arrays

The Array object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Array Declaration

An array can be declared using any one of the following techniques

Syntax:

```
var arrayname = new Array();
```

Or

```
var arrayname = [];
```

Array in JavaScript are indexed from 0.

Example: Creating an array and accessing array elements

```
<script type = "text/javascript">
    // Create an array:
    var college = ["NCIT", "NCC", "PMC"];
    document.write(college[0] + "<br>");//accessing the array elements
    document.write(college[1] + "<br>");
    document.write(college[2] + "<br>");
```

```
</script>
```

Output

← → 🌐 file:///C:/Users/dell/Desktop/array.js.html

NCIT
NCC
PMC

Array Methods

1. **concat():** Returns a new Array comprised of this array joined with other arrays or values

Syntax:

array1.concat(array2)

Example:

```
<script type = "text/javascript">
  var alpha = ["a", "b", "c"];
  var numeric = [1, 2, 3];
  var alphaNumeric = alpha.concat(numeric);
  document.write("alphaNumeric : " + alphaNumeric );
</script>
```

Output:

alphaNumeric : a,b,c,1,2,3

2. **length:** Returns the length of the array

Syntax:

arrayname.length

Example

```
<script type = "text/javascript">
  var subject = new Array("Web", "DBMS", "C Programming");
  document.write("The length of the array is : " + subject.length );
</script>
```

Output:

The length of the array is : 3

3. **join():** Joins all elements of an array into a string.

Syntax:

arrayname.join(separator)

Separator: Specifies a string to separate each element of the arrayname. If omitted, the array elements are separated with a comma.

Example:

```
<script type = "text/javascript">
    var arr = new Array("First","Second","Third");
    var str = arr.join();
    document.write("str : " + str );
    var str = arr.join(", ");
    document.write("<br />str : " + str );
    var str = arr.join(" + ");
    document.write("<br />str : " + str );
</script>
```

Output:

```
str : First,Second,Third
str : First, Second, Third
str : First + Second + Third
```

4. **indexOf():** Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.

Syntax:

arrayname.indexOf(item, start)

The first parameter is required and the second parameter is optional, it is used for start point of the search. Negative values will start at the given position counting from the end, and search to the end.

Example:

```
<script type = "text/javascript">
    var subject = new Array("Web","DBMS","Web","MicroProcessor",'C++');
    var a = subject.indexOf("Web"); //starts the searching for beginning
    var b = subject.indexOf("Web",2);
    document.write("The first search is in " +a);
    document.write("The second search is in " +b);
</script>
```

Output:

```
The first search is in 0The second search is in 2
```

5. **lastIndexOf():** The **lastIndexOf()** method searches the array for the specified item, and returns its position. The search will start at the specified position, or at the end if no start position is specified, and end the search at the beginning of the arrayname.

Syntax:

arrayname.lastIndexOf(item,start)

Example:

```
<script type = "text/javascript">
  var str = "This is string one and again string.";
  var index = str.lastIndexOf("string");
  document.write("lastIndexOf found String" +index);
  document.write("<br/>");
  var index = str.lastIndexOf("one",5);
  document.write("The lastIndexOf found string" +index);
</script>
```

Output:

```
lastIndexOf found String :29
lastIndexOf found String :-1
```

The first parameter is required and the second parameter is optional, it is used for start point of the search. Negative values will start at the given position counting from the end, and search to the end.

6. **pop():** Removes the last element from an array and returns that element

Syntax:

arrayname.pop()

Example:

```
<script type = "text/javascript">
  var num = new Array(6, 2, 9, 10);
  var element = num.pop();
  document.write("Element is" + element);
  document.write("<br/>");
  var element = num.pop();
  document.write("Element is " +element);
</script>
```

Output:

```
element is : 10  
element is : 9
```

7. **push():** Adds one or more elements to the end of an array and returns the new length of the array.

Syntax:

arrayname.push(element1,.....,elementN)

Example:

```
<script type = "text/javascript">  
  var numbers = new Array(6, 2, 9, 10);  
  var element = numbers.push(15);  
  document.write("new Array is" +numbers);  
  document.write("<br/>");  
  var element = numbers.push(20);  
  document.write("new Array is " +numbers);  
</script>
```

Output:

```
new array is : 6,2,9,10,15  
new array is : 6,2,9,10,15,20
```

8. **reverse() :** Reverses the order of the elements of an array. The first becomes the last, and the last becomes the first.

Syntax:

arrayname.reverse()

Example:

```
<script type = "text/javascript">  
  var numbers = new Array(6, 2, 9, 10);  
  var arr = numbers.reverse();  
  document.write("Reverse array is: " +numbers);  
</script>
```

Output:

```
Reverse array is : 10,9,2,6
```

9. **shift():** Removes the first element from an array and returns that element.

Syntax:

arrayname.shift()

Example:

```
<script type = "text/javascript">
  var numbers = new Array(6, 2, 9, 10);
  var arr = numbers.shift();
  document.write("Removed element is: " +numbers);
</script>
```

Output:

Removed element is : 2,9,10

10. unshift() : Adds one or more elements to the front of an array and returns the new length of the array.

Syntax:

arrayname.unshift(element1,.....,elementN)

Example:

```
<script type = "text/javascript">
  var numbers = new Array(6, 2, 9, 10);
  var arr = numbers.unshift(15);
  document.write("Added element is: " +numbers);
</script>
```

Output:

Added element is : 15,6,2,9,10

11. sort(): Sorts the elements of an array

Syntax:

arrayname.sort()

Example:

```
<script type = "text/javascript">
  var arr = new Array(6, 2, 9, 10);
  var sorted = arr.sort();
  document.write("Sorted array is: " +sorted);
```



```
</script>
```

Output:

2.6,9,10

12. slice(): Extracts a section of an array and returns a new Array.

Syntax:

arrayname.slice(begin,end)

Example:

```
<script type = "text/javascript">
    var arr = new Array("orange", "mango", "banana", "sugar", "tea");
    document.write("arr.slice(1, 2): " + arr.slice( 1, 2));
    document.write("<br/> arr.slice(1, 2): " + arr.slice( 1, 3));
</script>
```

Output:

```
arr.slice( 1, 2) : mango
arr.slice( 1, 3) : mango,banana
```

13. toString(): Returns a string representing the array and its elements.

Syntax

arrayname.toString()

Example:

```
<script type = "text/javascript">
    var arr = new Array("orange", "mango", "banana", "sugar");
    var str = arr.toString();
    document.write("Returned string is: " + str);
</script>
```

Output:

Returned string is : orange,mango,banana,sugar

String In JavaScript:

The String object lets you work with a series of characters; it wraps JavaScript's string primitive data type with a number of helper methods. JavaScript automatically converts between string primitives and String objects

Syntax:

`var name = new String(string_name)`

String Methods and Properties

1. **length** : The length property returns the length of a string

Syntax:

`string.length`

Example

```
<script type = "text/javascript">
  var text = "HTML is not a programming language";
  var length = text.length;
  document.write("The length of the string is " + length);
</script>
```

Output:

The length of the string is 34

2. **charAt()**:Returns the character at the specified index

Syntax:

`string.charAt(index)`

The index of the first character is 0, the second character is 1, and so on.

Example:

```
<script type = "text/javascript">
  var text = "This is a string.";
  document.write("text.charAt(5) is: " + text.charAt(5));
</script>
```

Output:

str.charAt(5) is: i

3. **concat():** Combines the text of two strings and returns a new string. This method does not change the existing strings, but returns a new string containing the text of the joined strings.

Syntax:

`string.concat(string2,.....string3,.....,stringN)`

Example:

```
<script type = "text/javascript">
  var str1 = "Hello";
  var str2 = "World";
  document.write("Combined string is: " + str1.concat(str2));
</script>
```

Output:

Combined string is HelloWorld

4. **indexOf():** Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.

Syntax:

`string.indexOf(searchvalue, start)`

Example:

```
<script type = "text/javascript">
  var str = ("This is a String.");
  var index = str.indexOf("a");
  document.write("indexOf(a) is: " + index);
</script>
```

Output:

indexOf(a) is: 8

5. **lastIndexOf():** Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.

Syntax:

`string.lastIndexOf(searchvalue, start)`

Example:

```
<script type = "text/javascript">
    var str = ("This is a String and also a text.");
    var index = str.lastIndexOf("a");
    document.write("lastIndexOf(a) is: " +index);
</script>
```

Output:

lastIndexOf(a) is: 26

6. **localeCompare():** Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.

- Returns -1 if string1 is sorted before string 2
- Returns 0 if the two strings are equal
- Returns 1 if string1 is sorted after string 2

Syntax:

string.localeCompare(*compareString*)

Example:

```
<script type = "text/javascript">
    var str1 = new String( "This is a string" );
    var index1 = str1.localeCompare( "is a" );
    document.write("localeCompare first :" + index1 );
    var index = str1.localeCompare( "What" );
    document.write("<br/>localeCompare second :" + index );
</script>
```

Output:

localeCompare first :1
localeCompare second :-1

7. **match():**Used to match a regular expression against a string.

Syntax:

string.match(*regex*)

Example:

```
<script type = "text/javascript">
```

```
var string = "This is a string";  
var result = string.match(/trin/i);  
document.write("Match() : " + result);  
</script>
```

Output:

Match() : trin

8. **replace():** Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.

Syntax:

string.replace(searchvalue, newvalue)

Example:

```
<script type = "text/javascript">  
    var str = 'Hello This is Web Technology';  
    var newstring = str.replace('Technology', 'Programmming');  
    document.write(newstring);  
</script>
```

Output:

Hello This is Web Programmming

9. **search():** Executes the search for a match between a regular expression and a specified string

Syntax:

string.search(searchvalue)

Example:

```
<script type = "text/javascript">  
    var str = "This is a string."  
    var res = str.search("a");  
    document.write(res);  
</script>
```

Output:

8

10. **slice():** Extracts a section of a string and returns a new string.

Syntax:

string.slice(start, end)

Example:

```
<script type = "text/javascript">
  var str = "Hello world!";
  var res = str.slice(0, 5);
  document.write(res);
</script>
```

Output:

Hello

11. split(): Splits a String object into an array of strings by separating the string into substrings.

Syntax:

string.split(separator, limit)

Example:

```
<script type = "text/javascript">
  var str = "How are you doing today?";
  var res = str.split(" ");
  document.write(res);
</script>
```

Output:

How,are,you,doing,today?

12. substr(): Returns the characters in a string beginning at the specified location through the specified number of characters.

Syntax:

string.substr(start, length)

Example:

```
<script type = "text/javascript">
  var str = "Hello world!";
  var res = str.substr(0, 5);
  document.write(res);
</script>
```

Output:

Hello

13. substring(): Returns the characters in a string between two indexes into the string.

Syntax:

`string.substring(start, end)`

Example:

```
<script type = "text/javascript">
    var str = "Hello world!";
    var res = str.substring(1, 4);
    document.write(res);
</script>
```

Output:

ello

The difference is in the second argument. The **second argument** to `substring` is the index to stop at (but not include), but the second argument to `substr` is the maximum **length** to return. Moreover, `substr()` accepts a negative starting position as an offset from the end of the string. `substring()` does not.

14. toLocaleLowerCase(): The characters within a string are converted to lower case while respecting the current locale.

Syntax:

`string.toLocaleLowerCase()`

Example:

```
<script type = "text/javascript">
    var str = "Hello World!";
    var res = str.toLocaleLowerCase();
    document.write(res);
</script>
```

Output:

hello world!

The locale is based on the language settings of the browser.

- 15. toLocaleUpperCase():** The characters within a string are converted to upper case while respecting the current locale.

Syntax:

string.toLocaleUpperCase ()

Example:

```
<script type = "text/javascript">
    var str = "Hello World!";
    var res = str.toLocaleUpperCase();
    document.write(res);
</script>
```

Output:

HELLO WORLD!

- 16. toLowerCase():** Returns the calling string value converted to lower case.

Syntax:

string.toLowerCase()

Example:

```
<script type = "text/javascript">
    var str = "Hello World!";
    var res = str.toLowerCase();
    document.write(res);
</script>
```

Output:

hello world!

- 17. toUpperCase():** Returns the calling string value converted to uppercase.

Syntax:

string.toUpperCase()

Example:


```
<script type = "text/javascript">
    var str = "Hello World!";
    var res = str.toUpperCase();
    document.write(res);
</script>
```

Output:

HELLO WORLD!

18. toString(): converts a number to a string.

Syntax:

number.toString()

Example:

```
<script type = "text/javascript">
    var num = 321;
    var res = num.toString(8);
    document.write(res);
</script>
```

Output:

501

19. valueOf(): Returns the primitive value of the specified object.

Syntax:

string.valueOf()

Example:

```
<script type="text/javascript">
var str = "Hello World!";
var res = str.valueOf();
document.write(res);
</script>
```

Output:

Hello World!

Number in JavaScript

The Number object represents numerical data, either integers or floating-point numbers. In general, you do not need to worry about Number objects because the browser automatically converts number literals to instances of the number class.

Syntax:

```
var n = new Number(number)
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns NaN (Not-a-Number).

Number Methods in JavaScript

1. **toExponential():** Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.

Syntax:

```
number.toExponential(x)
```

where x is optional and an integer between 0 and 20 representing the number of digits in the notation after the decimal point. If omitted, it is set to as many digits as necessary to represent the value

Example:

```
<script type="text/javascript">
  var num = 5.56789;
  var res = num.toExponential();
  document.write(res);
</script>
```

Output:

5.56789e+0

2. **toFixed() :** Formats a number with a specific number of digits to the right of the decimal.

Syntax:

```
number.toFixed(x)
```

where x is optional and the number of digits after the decimal point. Default is 0 (no digits after the decimal point)

Example:

```
<script type="text/javascript">
```

```
var num = 5.56789;  
var res = num.toFixed(2);  
document.write(res);  
</script>
```

Output:

5.57

3. **toLocaleString()** : Returns a string value version of the current number in a format that may vary according to a browser's local settings.

Syntax

number.toLocaleString(*locales*, *options*)

Example:

```
<script type="text/javascript">  
var num = 3325;  
var result = num.toLocaleString('en-US');  
document.write(result);  
</script>
```

Output:

3,325

4. **toPrecision()** : Defines how many total digits (including digits to the left and right of the decimal) to display of a number.

Syntax:

number.toPrecision(x)

where x is Optional. The number of digits. If omitted, it returns the entire number (without any formatting)

Example:

```
<script type="text/javascript">  
var num=213.45689;  
document.write(num.toPrecision(4));  
</script>
```

Output:

213.5

5. **toString():** Returns the string representation of the number's value.

Example:

```
<script>
    var num = 321;
    var res = num.toString(8);
    document.write(res);
</script>
```

Output:

501

6. **valueOf():** Returns the number's value.

Syntax:

number.valueOf()

Example:

```
<script type="text/javascript">
    var num=0/0;
    document.write("Output : " + num.valueOf());
</script>
```

Output:

Output : NaN

Boolean in JavaScript

The Boolean object represents two values, either "true" or "false". If value parameter is omitted or is 0, -0, null, false, NaN, undefined, or the empty string (""), the object has an initial value of false.

Syntax:

var name= new Boolean(value)

Boolean Methods

1. **toSource():** Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.

Syntax:

boolean.toSource()

2. **toString():** Returns a string of either "true" or "false" depending upon the value of the object.

Syntax:

boolean.toString()

3. **valueOf():** Returns the primitive value of the Boolean object.

DRAFT

Date In JavaScript

1. **getFullYear():** returns the year in the specified date according to local time. Use **getFullYear** instead.

Syntax:

```
date.getFullYear();
```

Example:

```
<script type="text/javascript">
```

```
function myFunction() {  
    var d = new Date("March 29, 1999 01:15:00");  
    var n = d.getFullYear();  
    document.write("Full Year: "+n);  
}
```

Output:

Full Year: 1999

2. **setDate():** Sets the day of the month for a specified date according to local time.

Syntax:

```
date.setDate(day_number);
```

Example:

```
<script type="text/javascript">
```

```
function myFunction() {  
    var d = new Date("March 03, 1999 01:15:00");  
    d.setDate(29);  
    document.write("demo").innerHTML = d;  
}
```

Output:

New Date: Mon Mar 29 1999 01:15:00 GMT+0545 (Nepal Time)

3. **setFullYear():** Sets the full year for a specified date according to local time.

Syntax:

```
date.setFullYear(year_number);
```

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 03, 1999");
  d.setFullYear(2019);
  document.write("demo").innerHTML = d;
}
</script>
```

Output:

New Full Year: Sun Mar 03 2019 00:00:00 GMT+0545 (Nepal Time)

4. **setHours()** : Sets the hours for a specified date according to local time.

Syntax:

```
date.setHours(Hrs, minutes,seconds);
```

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date();
  d.setHours(15, 35, 1);
  document.write("demo").innerHTML = d;
}
</script>
```

Output:

Sun Oct 20 2019 15:35:01 GMT+0545 (Nepal Time)

5. **setMilliseconds()** : Sets the milliseconds for a specified date according to local time.

Syntax:

```
date.setMilliseconds
```

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 03, 1999");
  d.setFullYear(2019);
  document.write("demo").innerHTML = d;
}
```

`</script>`

Output:

New set Milliseconds: 192

6. **setMinutes()** : Sets the minutes for a specified date according to local time.

Syntax:

`date.setMinutes();`

Example:

```
<script type="text/javascript">
function myFunction() {
    var d = new Date("March 29, 1999 8:44:00");
    d.setMinutes(17);
    document.write("demo").innerHTML = d;
}
</script>
```

Output:

New set Minutes: Mon Mar 29 1999 08:17:00 GMT+0545 (Nepal Time)

7. **setMonth()** : Sets the month for a specified date according to local time.

Syntax:

`date.setMonth();`

Example:

```
<script type="text/javascript">
function myFunction() {
    var d = new Date("March 29, 1999 8:44:00");
    d.setMonth(01);
    document.write("demo").innerHTML = d;
}
</script>
```

Output:

New set Months: Mon Mar 01 1999 08:44:00 GMT+0545 (Nepal Time)

8. **setSeconds()** : Sets the seconds for a specified date according to local time.

Syntax:

`date.setSeconds();`

Example:

```
<script type="text/javascript">
function myFunction() {
```



```
var d = new Date("March 29, 1999 8:44:00");
d.setSeconds(44);
document.write("demo").innerHTML = d;
}
</script>
```

Output:

New set Seconds: Mon Mar 29 1999 08:44:44 GMT+0545 (Nepal Time)

9. **setTime()** : Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.

Syntax:

```
date.setTime();
```

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 29, 1999 8:44:00");
  d.setTime(151214171819);
  document.write= d;
}
</script>
```

Output:

New set Time: Thu Oct 17 1974 09:26:11 GMT+0530 (Nepal Time)

37. **toString()**: Returns the "date" portion of the Date as a human-readable string.

Syntax:

```
Date.toString();
```

Example:

```
<script>
function myFunction() {
  var d = new Date("March 29, 1999 8:44:00");
  var n = d.toString();
  document.write("Date String: "+n);
}
</script>
```

Output:

Date String: Mon Mar 29 1999

- 10. toLocaleDateString():**Returns the "date" portion of the Date as a string, using the current locale's conventions.

Syntax:

date.toLocaleDateString();

Example:

<script type="text/javascript">

```
function myFunction() {  
    var d = new Date();  
    var n = d.toLocaleDateString();  
    document.write("Locale Date String: "+n);  
}
```

</script>

Output:

Locale Date String: 10/20/2019

- 11. toLocaleFormat():**Converts a date to a string, using a format string.

Syntax:

date.toLocaleFormat();

Example:

<script type = "text/javascript">

```
function format(){  
    var dt = new Date(1993, 6, 28, 14, 39, 7);  
    document.write( "Formatted Date : " + dt.toLocaleFormat( "%A, %B %e,  
    %Y" ) );  
}
```

`</script>`

Output:

```
Formatted Date : Wed Jul 28 1993 14:39:07
```

12. `toLocaleString()`: Converts a date to a string, using the current locale's conventions.

Syntax:

`date.toLocaleString();`

Example:

`<script>`

```
function myFunction() {  
    var d = new Date("March 29, 1999 8:44:00");  
    var n = d.toLocaleString();  
    document.write("Local String: "+n);  
}
```

`</script>`

Output:

```
Local String: 3/29/1999, 8:44:00 AM
```

13. `toLocaleTimeString()`: Returns the "time" portion of the Date as a string, using the current locale's conventions.

Syntax:

`date.toLocaleTimeString();`

Example:

`<script type="text/javascript">`

```
function myFunction() {  
    var d = new Date();  
    var n = d.toLocaleTimeString();  
    document.write("Local Time String: "+n);  
}
```

`</script>`

Output:

Local Time String: 4:06:05 PM

14. **toSource():** Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.

Syntax:

`date.toSource`

Example:

Output:

15. **toString():** Returns a string representing the specified Date object.

Syntax:

`date.toString();`

Example:

`<script type="text/javascript">`

```
function myFunction() {  
    var d = new Date();  
    var n = d.toString();  
    document.write("To String: "+n);  
}
```

`</script>`

Output:

To String: Sun Oct 20 2019 16:10:23 GMT+0545 (Nepal Time)

16. **toTimeString():** Returns the "time" portion of the Date as a human-readable string.

Syntax:

date.toString();

Example:

```
<script type="text/javascript">  
function myFunction() {  
    var d = new Date();  
    var n = d.toString();  
    document.write("To Time String: "+n);  
}  
</script>
```

Output:

To Time String: 16:12:35 GMT+0545 (Nepal Time)

17. valueOf():Returns the primitive value of a Date object.

Syntax:

date.valueOf();

Example:

```
<script type="text/javascript">  
function myFunction() {  
    var d = new Date();  
    var n = d.valueOf();  
    document.write("Value Of: "+n);  
}  
</script>
```

Output:

Value Of: 1571567375060

Math in JavaScript

The math object provides you properties and methods for mathematical constants and functions. Unlike other global objects, Math is not a constructor. All the properties and methods of Math are static and can be called by using Math as an object without creating it.

Math Methods in JavaScript

1. **abs():** Returns the absolute value of a number
2. **ceil():** Returns the smallest integer greater than or equal to a number
3. **exp():** Returns E^N , where N is the argument, and E is Euler's constant, the base of the natural logarithm.
4. **floor():** Returns the largest integer less than or equal to a number
5. **log():** Returns the natural logarithm of a number
6. **max():** Returns the largest of zero or more numbers
7. **min():** Returns the smallest of zero or more numbers
8. **pow():** Returns base to the exponent power, that is base exponent
9. **random():** Returns a pseudo-random number between 0 and 1
10. **round():** Returns the value of a number rounded to the nearest integer
11. **sqrt():** Returns the square root of a number

Regular Expressions and RegExp Object

A regular expression is an object that describes a pattern of characters.

The JavaScript RegExp class represents regular expressions, and both String and RegExp define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

Syntax

A regular expression could be defined with the RegExp () constructor, as follows

```
var pattern = new RegExp(pattern, attributes);  
or simply  
var pattern = /pattern/attributes;
```

JavaScript - Document Object Model or DOM

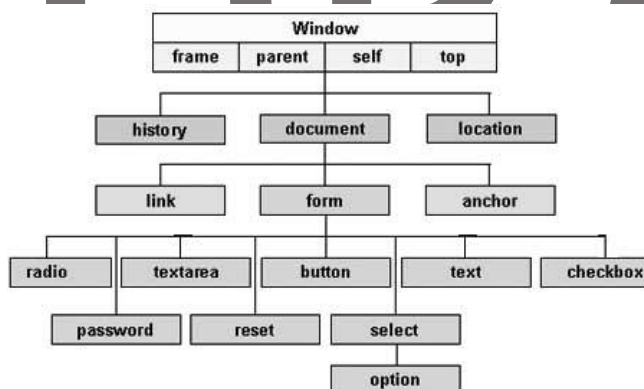
Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the Document Object Model, or DOM. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- Window object – Top of the hierarchy. It is the outmost element of the object hierarchy.
- Document object – Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- Form object – Everything enclosed in the <form>...</form> tags sets the form object.
- Form control elements – The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

The following figure is a simple hierarchy of few objects:



HTML DOM

`document.getElementById()`

The `getElementById()` method returns the element that has the ID attribute with the specified value. This method is one of the most common methods in the HTML DOM, and is used almost every time you want to manipulate, or get info from, an element on your document. Returns *null* if no elements with the specified ID exists. An ID should be unique within a page. However, if more than one element with the specified ID exists, the `getElementById()` method returns the first element in the source code.

Syntax:

document.getElementById(elementID)

document.getElementById("ElementId").value

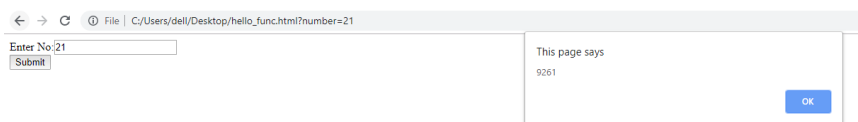
The value property sets or returns the value of the value attribute of a text field. The value property contains the default value OR the value a user types in (or a value set by a script).

Example:

```
<html>
<head>
<body>
<form>
  Enter No:<input type="text" id="number" name="number"/><br/>
  <input type="button" value="Submit" onclick="getcube()"/>
</form>
</body>
<script type="text/javascript">
  function getcube(){
    var number=document.getElementById("number").value;
    alert(number*number*number);
  }
</script>
</html>
```

Output:

Enter No:



document.getElementsByName()

The document.getElementsByName() method returns all the element of specified name.

Syntax:

document.getElementsByName("name")

Example:

```
<html>
<head>
```



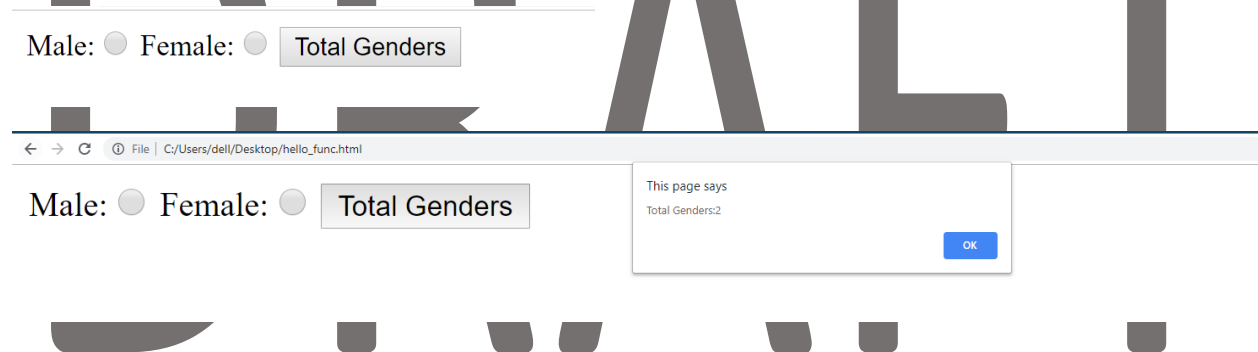
```

<script type="text/javascript">
  function totalelements()
  {
    var allgenders=document.getElementsByName("gender");
    alert("Total Genders:"+allgenders.length);
  }
</script>
<form>
Male:<input type="radio" name="gender" value="male">
Female:<input type="radio" name="gender" value="female">
<input type="button" onclick="totalelements()" value="Total Genders">
</form>

</body>
</html>

```

Output:



Javascript - innerHTML

The innerHTML property can be used to write the dynamic html on the html document. It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Example

```

<html>
  <script type="text/javascript" >
    function showcommentform() {
      var data="Javascript is fun";
      document.getElementById('mylocation').innerHTML=data;
    }
  </script>
  <form name="myForm">
    <input type="button" value="Show Magic" onclick="showcommentform()">

```

```
<div id="mylocation"></div>
</form>
</html>
```

Output:

comment

comment

Javascript is fun

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using `document.form1.name.value` to get the value of name field. Here, document is the root element that represents the html document.

`form1` is the name of the form.

`name` is the attribute name of the input text.

`value` is the property, that returns the value of the input text.

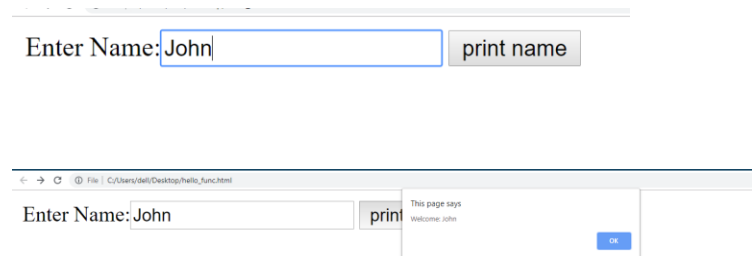
Let's see the simple example of document object that prints name with welcome message.

Example:

```
<html>
  <script type="text/javascript">
    function printvalue(){
      var name=document.form1.name.value;
      alert("Welcome: "+name);
    }
  </script>

  <form name="form1">
    Enter Name:<input type="text" name="name"/>
    <input type="button" onclick="printvalue()" value="print name"/>
  </form>
</html>
```

Output:



Enter Name:

Enter Name:

This page says
Welcome: John

Form Validation

Forms are used in webpages for the user to enter their required details that are further send it to the server for processing. A form is also known as web form or HTML form.

If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server. JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

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

Some terms used in Form validation

Check for empty: To check if all the fields are filled or not, check for empty which can be done by checking with empty string

Number validation: To check for the numbers , use regular expression with match properties

Example:

```
<script type="text/javascript">
function printvalue(){
var pattern = /^[0-9]+$;/
var number = document.getElementById("num").value;
if(number.match(pattern))
{
alert('Your Registration number has accepted....');
document.form1.text1.focus();
return true;
}
}
```

```
}  
</script>
```

Email validation:

Validating email is a very important point while validating an HTML form. In this page we have discussed how to validate an email using JavaScript. An email is a string (a subset of ASCII characters) separated into two parts by @ symbol. a "personal_info" and a domain, that is personal_info@domain. The length of the personal_info part may be up to 64 characters long and domain name may be up to 253 characters

Example:

```
<script type = "text/javascript">  
    function validateEmail() {  
        var emailID = document.getElementById("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>
```

Letters and Number validation

```
<script type = "text/javascript">  
function alphanumeric(inputtxt)  
{  
var letters = /^[0-9a-zA-Z]+$/;  
var string = document.getElementById('name').value;  
if(string.match(letters))  
{  
alert('Your registration number have accepted : you can try another');  
document.form1.text1.focus();  
return true;  
}  
else  
{  
alert('Please input alphanumeric characters only');  
return false;  
}  
}
```

```
}  
</script>
```

Example 1: Combining all for form validation

```
<html>  
<head>  
<title>Form Validation</title>  
</head>  
<body>  
<label>Name:</label>  
<input type="text" id="name"><br>  
<label>Number:</label>  
<input type="text" id="num"><br>  
<label>Gender:</label>  
<input type="radio" name="gender" value="Male" id="male">M  
<input type="radio" name="gender" value="Female" id="female">F<br>  
<input type="submit" onclick="check();">  
</body>  
<script>  
function check(){  
var n=document.getElementById("num").value;  
if(n == ""){  
alert("Enter number");  
}  
else if(isNaN(n)){  
alert('is not a number');  
}  
var condition=/^[A-Za-z]+$/;  
var name=document.getElementById("name").value;  
var result=name.match(condition);  
if(name == ""){  
alert("Enter name");  
}  
else if(!result){  
alert("Name format not valid");  
}  
var female=document.getElementById("female").checked;  
var male=document.getElementById("male").checked;  
if (male||female){  
alert("Gender selected");  
}  
else{  
alert("Gender required");  
}
```

```
}  
}  
</script>  
</html>
```

Output:

Name:
Number:
Gender: ☐ M ☐ F

Name:
Number:
Gender: ☐ M ☐ F

This page says
Name format not valid

OK

Error Handling

There are three types of errors in programming:

- Syntax Errors,
- Runtime Errors
- Logical Errors.

Syntax Errors

Syntax errors, also called parsing errors, occur at compile time in traditional programming languages and at interpret time in JavaScript. For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script type = "text/javascript">  
    window.print(  
</script>
```

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

Runtime errors, also called exceptions, occur during execution (after compilation/interpretation). For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script type = "text/javascript">
    window.printme();
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

Logical Errors

Logic 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 result you expected. You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions. You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

Here is the try...catch...finally block syntax

```
<script type = "text/javascript">
    try {
        // Code to run
        [break;]
    }

    catch ( e ) {
        // Code to run if an exception occurs
        [break;]
    }

    [ finally {
        // Code that is always executed regardless of
        // an exception occurring
    } ]
</script>
```

The try block must be followed by either exactly one catch block or one finally block (or one of both). When an exception occurs in the try block, the exception is placed in e and the catch block is executed. The optional finally block executes unconditionally after try/catch.

JavaScript Cookies

A cookie is variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JavaScript, you can both create and retrieve cookie values

Example of Cookies:

- a. Name cookie
- b. Date cookie

How It Works ?

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields :

- Expires – The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- Domain – The domain name of your site.
- Path – The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- Secure – If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- Name=Value – Cookies are set and retrieved in the form of key-value pairs

Cookies were originally designed for CGI programming. The data contained in a cookie is automatically transmitted between the web browser and the web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the cookie property of the Document object. JavaScript can read, create, modify, and delete the cookies that apply to the current web page.

Storing Cookies

The simplest way to create a cookie is to assign a string value to the document.cookie object, which looks like this.

```
document.cookie = "key1 = value1;key2 = value2;expires = date";
```

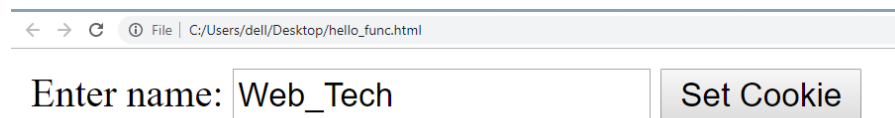

Here the expires attribute is optional. If you provide this attribute with a valid date or time, then the cookie will expire on a given date or time and thereafter, the cookies' value will not be accessible.

Example:

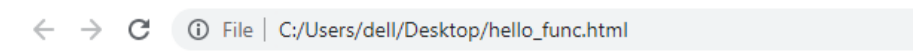
```
<html>
  <head>
    <script type = "text/javascript">
      function WriteCookie() {
        if( document.myform.customer.value == "" ) {
          alert("Enter some value!");
          return;
        }
        cookievalue = escape(document.myform.customer.value) + ";";
        document.cookie = "name=" + cookievalue;
        document.write ("Setting Cookies : " + "name=" + cookievalue );
      }
    </script>
  </head>

  <body>
    <form name = "myform" action = "">
      Enter name: <input type = "text" name = "customer"/>
      <input type = "button" value = "Set Cookie" onclick = "WriteCookie();"/>
    </form>
  </body>
</html>
```

Output:

A screenshot of a web browser window. The address bar shows the file path C:/Users/dell/Desktop/hello_func.html. The page content displays the text "Enter name:" followed by a text input field containing the value "Web_Tech". To the right of the input field is a button labeled "Set Cookie".

Enter name:

A screenshot of a web browser window. The address bar shows the file path C:/Users/dell/Desktop/hello_func.html. The page content displays the text "Setting Cookies : name=Web_Tech;".

Setting Cookies : name=Web_Tech;

Reading Cookies

Reading a cookie is just as simple as writing one, because the value of the document.cookie object is the cookie. So you can use this string whenever you want to access the cookie. The

`document.cookie` string will keep a list of name=value pairs separated by semicolons, where name is the name of a cookie and value is its string value.

You can use strings' `split()` function to break a string into key and values

Example:

```
<html>
  <head>
    <script type = "text/javascript">
      function ReadCookie() {
        var allcookies = document.cookie;
        document.write ("All Cookies : " + allcookies );

        // Get all the cookies pairs in an array
        cookiearray = allcookies.split(';');

        // Now take key value pair out of this array
        for(var i=0; i<cookiearray.length; i++) {
          name = cookiearray[i].split('=')[0];
          value = cookiearray[i].split('=')[1];
          document.write ("Key is : " + name + " and Value is : " + value
);
        }
      }
    </script>
  </head>
  <body>
    <form name = "myform" action = "">
      <p> click the following button and see the result:</p>
      <input type = "button" value = "Get Cookie" onclick = "ReadCookie()"/>
    </form>
  </body>
</html>
```

Output:

click the following button and see the result:

Get Cookie

Setting Cookies Expiry Date

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiry date within the cookie. This can be done by setting the 'expires' attribute to a date and time.

Example

```
<html>
  <head>
    <script type = "text/javascript">
      function WriteCookie() {
        var now = new Date();
        now.setMonth( now.getMonth() + 1 );
        cookievalue = escape(document.myform.customer.value) + ";";
        document.cookie = "name=" + cookievalue;
        document.cookie = "expires=" + now.toUTCString() + ";";
        document.write ("Setting Cookies : " + "name=" + cookievalue );
      }
    </script>
  </head>
  <body>
    <form name = "myform" action = "">
      Enter name: <input type = "text" name = "customer"/>
      <input type = "button" value = "Set Cookie" onclick = "WriteCookie()"/>
    </form>
  </body>
</html>
```

Deleting a Cookie

Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiry date to a time in the past.

Example:

```
<html>
  <head>
    <script type = "text/javascript">
      function WriteCookie() {
        var now = new Date();
        now.setMonth( now.getMonth() - 1 );
        cookievalue = escape(document.myform.customer.value) + ";";
        document.cookie = "name=" + cookievalue;
        document.cookie = "expires=" + now.toUTCString() + ";";
        document.write("Setting Cookies : " + "name=" + cookievalue );
      }
    </script>
  </head>
  <body>
    <form name = "myform" action = "">
      Enter name: <input type = "text" name = "customer"/>
      <input type = "button" value = "Set Cookie" onclick = "WriteCookie()"/>
    </form>
  </body>
</html>
```

```
</script>
</head>

<body>
  <form name = "myform" action = "">
    Enter name: <input type = "text" name = "customer"/>
    <input type = "button" value = "Set Cookie" onclick = "WriteCookie()"/>
  </form>
</body>
</html>
```

jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

Adding jQuery to Your Web Pages

There are several ways to start using jQuery on your web site. You can:

Download the jQuery library from [jQuery.com](http://jquery.com)

Include jQuery from a CDN, like Google

Downloading jQuery

There are two versions of jQuery available for downloading:

- Production version - this is for your live website because it has been minified and compressed
- Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from [jQuery.com](http://jquery.com).

The jQuery library is a single JavaScript file, and you reference it with the HTML `<script>` tag (notice that the `<script>` tag should be inside the `<head>` section):

```
<head>
<script src="jquery-3.4.1.min.js"></script>
</head>
```

or just use

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
</head>
```

jQuery Syntax

It is used for selecting elements in HTML and performing the action on those elements.

Syntax:

`$(selector).action()`

The selector may be id selector or class selector

- `$` sign: It grants access to jQuery.
- `(selector)`: It is used to find HTML elements.
- `jQuery action()`: It is used to perform actions on the elements

Examples:

`$(this).hide()` - hides the current element.

`$('p').hide()` - hides all `<p>` elements.

`$('.test').hide()` - hides all elements with `class="test"`.

`$('#test').hide()` - hides the element with `id="test"`.

Events and Effects

jQuery Methods are inside a Document ready event for easy reading of code.

Syntax:

```
$(document).ready(function(){
  // jQuery Method
});
```

This is to check and stop the jquery before the document is finished loading. This method also allows you to have JavaScript code before the body of your document, in the head section.

What are Events?

All the different visitors' actions that a web page can respond to are called events. An event represents the precise moment when something happens.

Examples:

- moving a mouse over an element

- selecting a radio button
- clicking on an element

The term "fires/fired" is often used with events. Example: "The keypress event is fired, the moment you press a key".

Here are some common DOM events:

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

Effects

jQuery hide() and show()

Example:

```
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("#hide").click(function(){
    $("p").hide();
  });
  $("#show").click(function(){
    $("p").show();
  });
});
</script>
</head>
<body>

<p>If you click on the "Hide" button, I will disappear.</p>

<button id="hide">Hide</button>
<button id="show">Show</button>

</body>
</html>
```

Output:

If you click on the "Hide" button, I will disappear.

The some other jQuery Effects are:

animate()	Runs a custom animation on the selected elements
clearQueue()	Removes all remaining queued functions from the selected elements
delay()	Sets a delay for all queued functions on the selected elements
dequeue()	Removes the next function from the queue, and then executes the function
fadeIn()	Fades in the selected elements
fadeOut()	Fades out the selected elements
fadeTo()	Fades in/out the selected elements to a given opacity
fadeToggle()	Toggles between the fadeIn() and fadeOut() methods
finish()	Stops, removes and completes all queued animations for the selected elements
hide()	Hides the selected elements
queue()	Shows the queued functions on the selected elements
show()	Shows the selected elements
slideDown()	Slides-down (shows) the selected elements
slideToggle()	Toggles between the slideUp() and slideDown() methods
slideUp()	Slides-up (hides) the selected elements
stop()	Stops the currently running animation for the selected elements
toggle()	Toggles between the hide() and show() methods

Introduction to JSON

JSON: short for JavaScript Object Notation, is a format for sharing data. As its name suggests, JSON is derived from the JavaScript programming language, but it's available for use by many languages including Python, Ruby, PHP, and Java. JSON is usually pronounced like the name "Jason."

JSON uses the .json extension when it stands alone. When it's defined in another file format (as in .html), it can appear inside of quotes as a JSON string, or it can be an object assigned to a variable. This format is easy to transmit between web server and client or browser.

Very readable and lightweight, JSON offers a good alternative to XML and requires much less formatting. This informational guide will get you up to speed with the data you can use in JSON files, and the general structure and syntax of this format.

Syntax and Structure

A JSON object is a key-value data format that is typically rendered in curly braces. When you're working with JSON, you'll likely see JSON objects in a .json file, but they can also exist as a JSON object or string within the context of a program.

A JSON object looks something like this:

```
{  
  "first_name" : "Sammy",  
  "last_name" : "Shark",  
  "location" : "Ocean",  
  "online" : true,  
  "followers" : 987  
}
```

Although this is a very short example, and JSON could be many lines long, this shows that the format is generally set up with two curly braces (or curly brackets) that look like this { } on either end of it, and with key-value pairs populating the space between. Most data used in JSON ends up being encapsulated in a JSON object.

Key-value pairs have a colon between them as in "key" : "value". Each key-value pair is separated by a comma, so the middle of a JSON looks like this: "key" : "value", "key" : "value", "key": "value". In our example above, the first key-value pair is "first_name" : "Sammy".

JSON keys are on the left side of the colon. They need to be wrapped in double quotation marks, as in "key", and can be any valid string. Within each object, keys need to be unique. These key strings *can* include whitespaces, as in "first name", but that can make it harder to access when you're programming, so it's best to use underscores, as in "first_name".

JSON values are found to the right of the colon. At the granular level, these need to be one of 6 simple data types:

- strings
- numbers
- objects
- arrays
- Booleans (true or false)
- null

At the broader level, values can also be made up of the complex data types of JSON object or array, which is covered in the next section.

Each of the data types that are passed in as values into JSON will maintain their own syntax, so strings will be in quotes, but numbers will not be.

Though in .json files, we'll typically see the format expanded over multiple lines, JSON can also be written all in one line.

```
{ "first_name" : "Sammy", "last_name": "Shark", "online" : true, }
```

DRAFT

Exercise:

1. What is JavaScript? List out some features of Javascript.
2. How can you define a named function in JavaScript?
3. What is anonymous function?
4. Define `parseInt`, `parseFloat` along with examples
5. If we want to return the character from a specific index which method is used?
6. What is the use of window object?
7. Explain the differences between `==` and `===` .
8. Explain `document.getElementsByName()`.
9. 7. What is DOM in Javascript?
10. 8. Difference between `substr()` and `substring()`.
11. 9. Explain `join()` and `concat()`.
12. 10. Explain `trunc()` and `floor()`.
13. Write a source code to create the login form that contains a textbox for username, a text box for password and a submit button. Make the following validations(both username and password is required and cannot be of same value and password length should be greater than 5 characters long which should not accept other than alphabets and numbers
14. Write a JS function that multiplies two prime number only. Check if the numbers entered are prime or not