Unit-1 Java Script Concepts

Introduction to JavaScript

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are −

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

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>

  JavaScript code

</script>
```

```
<html>
  <body>
    <script language = "javascript" type = "text/javascript">
      <!--
        document.write("Hello World!")
      //-->
    </script>
  </body>
</html>
```

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your

statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language = "javascript" type = "text/javascript">
  <!--
    var1 = 10
    var2 = 20
  //-->
</script>
```

But when formatted in a single line as follows, you must use semicolons −

```
<script language = "javascript" type = "text/javascript">
  <!--
    var1 = 10; var2 = 20;
  //-->
</script>
```

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus −

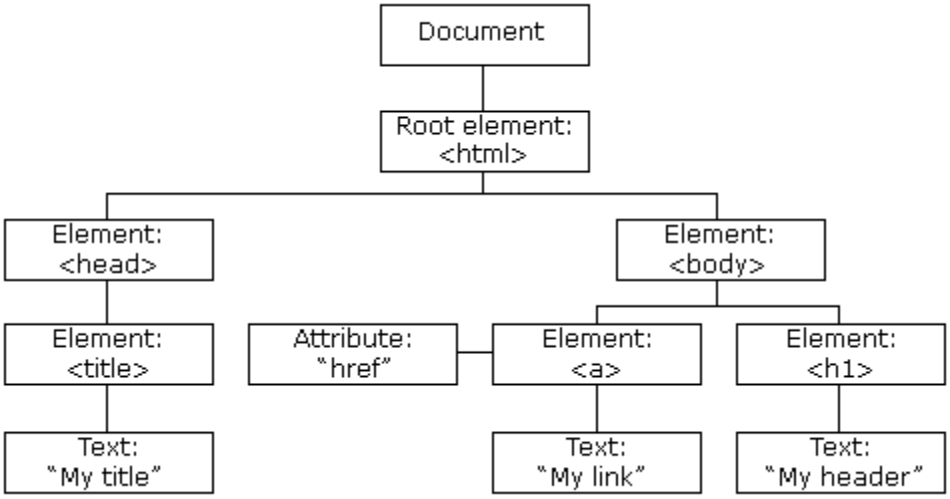
- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line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

Document Object:

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



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 is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements

- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

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

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 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

Adding Events Handlers

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

Window Object

The window object represents an open window in a browser.

Window Object Methods

Method	Description
addEventListener()	Attaches an event handler to the window
alert()	Displays an alert box with a message and an OK button

atob()	Decodes a base-64 encoded string
blur()	Removes focus from the current window
btoa()	Encodes a string in base-64
clearInterval()	Clears a timer set with setInterval()
clearTimeout()	Clears a timer set with setTimeout()
close()	Closes the current window
confirm()	Displays a dialog box with a message and an OK and a Cancel button
focus()	Sets focus to the current window
getComputedStyle()	Gets the current computed CSS styles applied to an element
getSelection()	Returns a Selection object representing the range of text selected by the user
matchMedia()	Returns a MediaQueryList object representing the specified CSS media query string
moveBy()	Moves a window relative to its current position
moveTo()	Moves a window to the specified position
open()	Opens a new browser window

print()	Prints the content of the current window
prompt()	Displays a dialog box that prompts the visitor for input

The Math Object

Unlike other objects, the Math object has no constructor.

The Math object is static.

All methods and properties can be used without creating a Math object first.

Number to Integer

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)

JavaScript String Methods:

https://www.w3schools.com/js/js_string_methods.asp

Objects:

Objects are variables too. But objects can contain many values.

Object values are written as name:value pairs (name and value separated by a colon).

Example:

let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

A JavaScript object is a collection of named values.

Object Properties:

The named values, in JavaScript objects, are called properties.

Property	**Value**
firstName	John
lastName	Doe
age	50
eyeColor	blue

Object Methods:

An object method is an object property containing a function definition.

```
const person = {
firstName: "John",
lastName: "Doe",
id: 5566,
fullName: function() {
returnthis.firstName + " " + this.lastName;
  }
};
```

Example:

```
const person = {
  name: "John",
  age: 30,
```

```
  city: "New York"
};

let txt = "";
for (let x in person) {
txt += person[x] + " ";
};

document.getElementById("demo").innerHTML = txt;
```

Date Object

We can create date object simply with the use of Date().

```
const d = new Date();
```

By default, JavaScript will use the browser's time zone and display a date as a full text string:

Fri Jul 02 2021 10:44:45 GMT+0530 (India Standard Time)

Creating Object:

Date objects are created with the new Date() constructor.

There are 4 ways to create a new date object:

new Date()

new Date(year, month, day, hours, minutes, seconds, milliseconds)

new Date(milliseconds)

new Date(date string)

new Date()

new Date() creates a new date object with the current date and time:

Example

```
const d = new Date();
```

Date objects are static. The computer time is ticking, but date objects are not.

new Date(year, month, ...)

new Date(year, month, ...)

creates a new date object with a specified date and time.

7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order):

Example

const d = new Date(2018, 11, 24, 10, 33, 30, 0);

Note: JavaScript counts months from 0 to 11.

January is 0. December is 11.

6 numbers specify year, month, day, hour, minute, second:

Example

const d = new Date(2018, 11, 24, 10, 33, 30);

5 numbers specify year, month, day, hour, and minute:

Example

const d = new Date(2018, 11, 24, 10, 33);

4 numbers specify year, month, day, and hour:

Example

const d = new Date(2018, 11, 24, 10);

3 numbers specify year, month, and day:

Example

const d = new Date(2018, 11, 24);

2 numbers specify year and month:

Example

const d = new Date(2018, 11);

You cannot omit month. If you supply only one parameter it will be treated as milliseconds.

Example

```
const d = new Date(2018);
```

new Date(dateString)

new Date(dateString) creates a new date object from a date string:

Example

```
const d = new Date("October 13, 2014 11:13:00");
```

Date Methods

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

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)

Example:

```
const d = new Date();
d.setFullYear(2020);
```

The setFullYear() method can optionally set month and day:

Example:

```
const d = new Date();
d.setFullYear(2020, 11, 3);
```

Conditional Statement:

1. If Statement:

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

Example:

```
if (hour < 18) {
greeting = "Good day";
}
```

2.If..Else Statement:

The If..else statement contains two blocks, one as if statement and second is else statement where if statement specify a block of JavaScript code to be executed if a condition is true and else specify a block of code to be executed if the condition is false.

Example:

```
if (hour < 18) {s
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

3. If..ElseIf.. Statement:

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

Example:

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

4. Switch Statement

In Switch statement, switch block is used to select one of many code blocks to be executed.

Syntax:

```
switch(expression) {

case x:

   // code block

break;

case y:

   // code block
```

```
break;

default:

   // code block

}
```

Looping Structure

1. While Loop:

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

Syntax:

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

Example:

```
while (i< 10) {
  text += "The number is " + i;
  i++;
}
```

2. Do While Loop:

The do while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

```
do {

  // code block to be executed

}

while (condition);
```

Example:

The example below uses a do while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

```
do {
  text += "The number is " + i;
  i++;
}
while (i< 10);
```

3. For Loop:

The for loop has the following syntax:

Syntax:

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

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

Example:

```
for (let i = 0; i< 5; i++) {
  text += "The number is " + i + "<br>";
}
```

4. For In Loop:

The for in statement loops through the properties of an Object:

Syntax:

```
for (key in object) {
  // code block to be executed
}
```

Example:

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
for (let x in numbers) {
  txt += numbers[x];
}
```

Function

JavaScript functions are defined with the **function** keyword.You can use a function declaration or a function expression.

Function Declarations

functions are declared with the following syntax:

```
functionfunctionName(parameters) {
  // code to be executed
}
```

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

Example

```
functionmyFunction(a, b) {
return a * b;
}
```

Semicolons are used to separate executable JavaScript statements.

Since a function declaration is not an executable statement, it is not common to end it with a semicolon.

Function Expressions

A JavaScript function can also be defined using an expression.

A function expression can be stored in a variable:

Example

```
const x = function (a, b) {return a * b};
```

After a function expression has been stored in a variable, the variable can be used as a function:

Example

```
const x = function (a, b) {return a * b};

let z = x(4, 3);
```

Function Parameters and Arguments

Function parameters are the names listed in the function definition.Function arguments are the real values passed to (and received by) the function.

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

Invoking a JavaScript Function

The code inside a function is not executed when the function is defined.

The code inside a function is executed when the function is invoked.

It is common to use the term "call a function" instead of "invoke a function".

It is also common to say "call upon a function", "start a function", or "execute a function".

Example

```
functionmyFunction(a, b) {
return a * b;
}
myFunction(10, 2);          // Will return 20
```

Example

```
functionmyFunction(a, b) {
return a * b;
}
window.myFunction(10, 2);
```

JavaScript Dialog Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
window.alert("sometext");
```

The window.alert() method can be written without the window prefix.

Example

```
alert("I am an alert box!");
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

window.confirm("sometext");

The window.confirm() method can be written without the window prefix.

Example

```
if (confirm("Press a button!")) {

txt = "You pressed OK!";

} else {

txt = "You pressed Cancel!";

}
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

window.prompt("sometext","defaultText");

The window.prompt() method can be written without the window prefix.

Example

```
let person = prompt("Please enter your name", "Harry Potter");

let text;

if (person == null || person == "") {

text = "User cancelled the prompt.";

} else {

text = "Hello " + person + "! How are you today?";

}
```