Introduction to JavaScript

JavaScript is the most commonly used client-side scripting language. It is an object scripting language and is used in web pages along with markup language HTML. JavaScript is very popular and adopted universally by every web browser. JavaScript allows dynamic content to get executed on a webpage. An HTML page is static in nature. It is JavaScript which makes this HTML page interactive i.e. dynamic.

JavaScript was created by Brendan Eich, a Netscape Communications Corporation programmer, in September 1995 for the Netscape Navigator Web browser. At that time, web pages were static, offering little user interaction beyond clicking links and loading new pages. For the first time, JavaScript enabled animation, adaptive content and form validation on the page.

How does JavaScipt work?

JavaScript is what is known as a client-side script. Most Web applications, such as a search engine, work because of an interaction between the user's device (e.g. computer, phone or tablet) and a remote server. The software on the remote server sends information to the client (i.e. the user's machine) and the software on the client side reads the information and renders a Web page on screen.

A client-side script is a programming language that performs its tasks entirely on the client's machine and does not need to interact with the server to function. For instance, if you have a Web page loaded on your computer and your Internet service provider goes down, you are still able to interact with the Web pages already loaded on your browser. You will not, however, be able to navigate to new Web pages or access any data located remotely.

Some of the dynamic website enhancements performed by JavaScript are:

- Autocomplete
- Loading new content or data onto the page without reloading the page
- Rollover effects and dropdown menus
- Animating page elements such as fading, resizing or relocating
- Playing audio and video
- Validating input from Web forms
- Repairing browser compatibility issues

While JavaScript is a client-side language, some of its most powerful features involve asynchronous interaction with a remote server. Asynchronous simply means that JavaScript is able to communicate with the server in the background without interrupting the user interaction taking place in the foreground.

Take a search engine for example. Today, search engines have an autocomplete feature. The user begins typing a word into the search box and a list of possible search terms or phrases appears below. The experience is seamless. Suggested search terms appear without reloading the page.

In the background, JavaScript reads the letters as the user types, sends those letters to a remote server. The software on the server side analyzes the words, runs algorithms to anticipate the user's search term and then sends suggestions back.

JavaScript Fundamentals

A script is a series of instructions that a computer can follow one-by-one. Each individual instruction or step is known as a statement. Statements should end with a semicolon.

JavaScript is case sensitive. So hourNow means something different to HourNow or HOURNOW.

Comments

MULTI-LINE COMMENTS

To write a comment that stretches over more than one line, you use a multi-line comment, starting with the /* characters and ending with the */ characters. Anything between these characters is not processed by the JavaScript interpreter. Multi-line comments are often used for descriptions of how the script works. Or to prevent a section of the script from running when testing it.

SINGLE-LINE COMMENTS

In a single-line comment, anything that follows the two forward slash characters // on that line will not be processed by the JavaScript interpreter. single-line comments are often used for short descriptions of what the code is doing.

Good use of comments will help you if you come back to your code after several days or months. They also help those who are new to your code.

Declaring Variable

Before you can use a variable, you need to announce that you want to use it. This involves creating the variable and giving it a name. This is called variable declaration. One example is:

```
var quantity;
```

'var' is a keyword. The JavaScript interpreter knows that this keyword is used to create a variable. If variable name is more than one word, then the convention is to give first word in lowercase and then CamelCase is used. For example,

```
var noOfBooks ;
```

Unlike some other programming languages, when declaring a variable in JavaScript, you do not need to specify what type of data it will hold.

Data Types

JavaScript distinguishes between numbers, strings and boolean values.

NUMERIC DATA TYPE

The numeric data type handles number like 0.75, 65, 245.78 etc.

STRING DATA TYPE

The strings data type consists of letters, digits and other characters e.g. 'am@5'

Note how the string data type is enclosed within a pair of quotes. These can be single or double quotes, but the opening quote must match the closing quote.

String can be used when working with any kind of text. They are frequently used to add new content into a page and they can contain HTML markup.

BOOLEAN DATA TYPE

Boolean data types can have one of two values: true or false.

In addition to these three data types, JavaScript also has others (arrays, objects, undefined and null) that you will meet in later chapters.

RULES FOR NAMING VARIABLES

Here are six rules you must always follow when giving a variable a name:

1
The name must begin with a letter, dollar sign ($), or an underscore (_). It must not start with a number.

2
The name can contain letters, numbers, dollar sign ($),or an underscore (_). Note that you must not use a dash (-) or a period (.) in a variable name.

3
You cannot use keywords or reserved words. Keywords are special words that tell the interpreter to do something. For example, var is a keyword used to declare a variable. Reserved words are ones that may be used in a future version of JavaScript.

4
All variables are case sensitive, so 'area' and 'Area' would be different variable names, but it is bad practice to create two variables that have the same name using different cases.

5
Use a name that describes the kind of information that the variable stores. For example, firstName might be used to store a person's first name, lastName for their last name, and age for their age.

6
If your variable name is made up of more than one word, use a capital letter for the first letter of every word after the first word. For example, firstName rather than firstname You can also use an underscore between each word (you cannot use a dash)

Arrays

An array is a special type of variable. It doesn't just store one value; it stores a list of values.

You should consider using an array whenever you are working with a list or a set of values that are related to each other.

Arrays are especially helpful when you do not know now many items a list will contain because, when you create the array, you don't need to specify how many values it will hold.

Creating An Array

You create an array and give it a name just like you would any other variable (using the var keyword followed by the name of the array).

Two techniques are used for creating an array:

1) Array Literal Method:

The values are assigned to the array inside a pair of square brackets, and each value is separated by a comma. The values in the array do not need to be the same data type, so you can store a string, a number and a boolean all in the same array. You can also write each value on a separate line:

2) Array Constructor Method:

This uses the new keyword followed by Array(); The values are then specified in parentheses (not square brackets), and each value is separated by a comma.
The array literal method is preferred over the array constructor when creating arrays.

Accessing an Array

Values in an array are accessed as if they are in a numbered list. It is important to know that the numbering of this list starts at zero (not one). Each array has a property called length, which holds the number of items in the array.

Refer array.js
Refer test_js.html
The above program also explains you how to integrate JS with HTML.

Functions

To create a function, you give it a name and then write the statements needed to achieve its task inside the curly braces. This is known as a function declaration.

You declare a function using the 'function' keyword. You give the function a name followed by parentheses. The statements that perform the task sit in a pair of curly braces.

Sometimes a function needs specific information to perform its task. In such cases, when you declare the function, you give it parameters. Inside the function, the parameters act like variables.

If a function needs information to work, you indicate what it needs to know in parentheses after the function name.

Having declared the function, you can then execute all of the statements between its curly braces with just one line of code. This is known as calling the function. To run the code in the function, you use the function name followed by parentheses.

Sometimes you will see a function called before it has been declared. This still works because the interpreter runs through the script before executing each statement, so it will know that a function declaration appears later in the script. When the function is called it may return a single or even multiple values using array.

Anonymous Functions

A function with no name is called an anonymous function. An anonymous function can be stored in any variable. It can be called like any function created with a function declaration.

Immediately Invoked Function Expressions (IIFE)

Pronounced “iffy”, these functions are not given a name. They are executed only once as the interpreter comes across them.

While using IIFE, the variable will hold the value returned from the function rather than storing the function itself as in case of Anonymous functions. That's why IIFE can be called only once.

function.js

```
document.write("<h2>Demonstrate syntax of Functions & how they can return multiple values</h2>");

var a = getSize(2 , 3 , 4) [0] ;
var v = getSize(2 , 3 , 4) [1] ;

function  getSize(width , height , depth)
{
	var area = width * height ;
	var volume = width * height * depth ;

	var sizes = [area , volume] ;

	return  sizes ;
}
document.write("<h4>Area is "+a+" </h4>");
document.write("<h4>Volume is "+v+" </h4>");


document.write("<h2>Demonstrate Anonymous Functions</h2>");

var area = function(width , height)
{
	return  width * height ;
}

var m = area(4 , 5);
document.write("<h4>Area is "+m+" </h4>");


document.write("<h2>Demonstrate IIFE</h2>");

var n = ( function()
{
	return  5 * 6 ;
} () ) ;

document.write("<h4>Value is "+n+" </h4>");


/*
IIFE
The final parentheses after the closing curly brace of the code block tell the interpreter to call the function immediately.
The parentheses after = and before semicolon are there to ensure that
the interpreter treats this as an expression.
*/
```

Variable Scope

The location where you declare a variable will affect where it can be used within your code. If you declare it within a function, it can only be used within that function. This is known as the variable's scope.

Local Variables

When a variable is created inside a function using the var keyword, it can only be used in that function. It is called a local variable or function-level variable. It is said to have local scope or function-level scope. It cannot be accessed outside of the function in which it was declared. Below, area is a local variable.

The interpreter creates local variables when the function is run, and removes them as soon as the function has finished its task. This means that:

- If the function runs twice,the variable can have different values each time.
- Two different functions can use variables with the same name without any kind of naming conflict.

Global Variables

If you create a variable outside of a function, then it can be used anywhere within the script. It is called a global variable and has global scope. In the example shown, wallSize is a global variable.

Global variables are stored in memory for us long as the web page is loaded into the web browser. This means they take up more memory than local variables and it also increases the risk naming conflicts. For these reasons, you should use local variables wherever possible.

If you forget to declare a variable using the var keyword, the variable will work, but it will be treated as a global variable (this is considered as bad practice).

```
function getArea(width , height)
{
  var area = width * height;
  return area;
}

var wallSize = getArea(3,2);
```

Objects

An object is a group of variables and functions. In an object, variables are known as properties and functions are called methods.

Properties tell us about the object. For example, if an object represents a Hotel then in that object we may have properties such as 'name of hotel' or 'number of rooms' it has.

Methods represent tasks that are associated with the object. For example, in object Hotel, you may have a method 'roomsAvailable()' which will check how many rooms are available by subtracting the number of booked rooms from the total number of rooms.

An object cannot have two properties with the same name. The value of a property can be a string, number, boolean, array or even another object.

Creating An Object: Literal Notation

Literal notation is the easiest and most popular way to create objects.

The object is the curly braces and their contents. The object is always stored in a variable so you would refer to it later on.

Separate each key from its value using a colon. Separate each property and method with a comma (but not after the last value).

When setting properties, treat the values like you would do for variables: strings live in quotes and arrays live in square brackets.

Example:

```
document.write("<h2>Creating object : Literal Notation</h2>");

var hotel = {
        name: "Quay" ,
        rooms: 40,
        booked: 25,
        roomsLeft: function() {
                return  this.rooms - this.booked ;
        }
};

var roomsFree = hotel.roomsLeft() ;
hotel.is5Star = true ;                          // adds a new property called is5Star
document.write("<h4>"+hotel.name+hotel['rooms']+roomsFree+hotel.is5Star+"</h4>
                ");
delete  hotel.booked ;
document.write("<h4>"+hotel.booked+"</h4>");
```

To update or access the value of properties, use dot notation or square brackets. If the object does not have the property which you are trying to update then it will be added to the object.

To delete a property, use the delete keyword followed by the object name and property name.

Creating An Object: Constructor Notation

The new keyword along with the constructor Object() creates a blank object. You can then add properties and methods to the object. Object() is part of the JavaScript language and is used to create an object.

Next, having created the blank object, you can add properties and methods to it using dot notation. Each statement that adds a property or method should end with a semicolon.

Example:

```
document.write("<h2>Creating object : Constructor Notation</h2>");

var hotel1 = new  Object() ;
hotel1.name = "Park" ;
hotel1.rooms = 50 ;
hotel1.booked = 30 ;
hotel1.roomsLeft = function() {
                return  this.rooms - this.booked ;
};
document.write("<h4>"+hotel1.name+hotel1.rooms+hotel1.roomsLeft()+"</h4>");
```

To create an empty object using literal notation use:
var hotel = {}
The curly brackets create an empty object.

Creating Many Objects: Constructor Notation

Sometimes you will need several objects to represent similar things.
Object constructors can use a function as a template for creating objects of same type.
First create the template (model/format) with the object's properties and methods.

this keyword is used inside function template to indicate that the property or method belongs to the object that this function creates. Each statement that creates a new property or method for this object ends in a semicolon (not a comma, which is used in the literal syntax).

The name of constructor function usually begins with a capital letter (unlike other functions, which tend to begin with a lowercase character). The uppercase letter is supposed to help remind developers to use the new keyword when they create an object using that function.

The new keyword followed by a call to the constructor function creates a new object.
The properties of each object are given as arguments to the function.

Although the properties are different for different objects, the methods used by different objects remain same.

Refer multiple_objects.js

Built-in Objects

Browsers come with a set of built-in objects that represent things like the browser window and the current web page shown in that window. These built-in objects act like a toolkit for creating interacting web pages.

The objects you create will usually be specifically written to suit your needs. They contain data and functions needed by your script. Whereas, the built-in objects contain functionality commonly needed by almost all scripts. As soon as a web page has loaded into the browser, these objects are available to use in your scripts.

These built-in objects help you get a wide range of information such as the width of the browser window, the content of the main heading in the page, or the length of text a user entered into a form field.

You access the properties or methods of built-in objects using dot notation, just like you would access the properties or methods of an object you had written yourself.

What is an Object Model ?

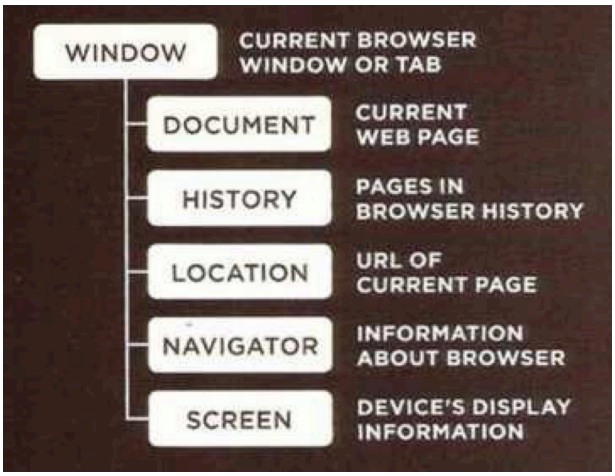
You have seen that an object can be used to create a representation of something from the real world using data. An object model is a group of objects, each of which represent related things from the real world. Together they form a model of something larger.

There are three groups of built-in objects of which we will be studying two of them:

1. Browser Object Model (BOM)
2. Document Object Model (DOM)

Browser Object Model (BOM)

The window object represents the current browser window or tab. It is the topmost object in the Browser object Model, and it contains other objects that tell you about the browser.



Here are a selection of the window object's properties and methods. You can also see some properties of the screen and history objects (which are children of the window object).

PROPERTY	DESCRIPTION
window. innerHeight	Height of window (excluding user interface) (in pixels)
window. innerWidth	Width of window (excluding user interface) (in pixels)
window. pageXoffset	Distance document has been scrolled horizontally (in pixels)
window. pageYoffset	Distance document has been scrolled vertically (in pixels)
window. screenX	X- coordinate of pointer, relative to top left corner of screen (in pixels)
window. screenY	Y-coordinate of pointer, relative to top left corner of screen (in pixels)
window. location	Current URL of window object (or local file path)
window. document	Reference to document object, which is used to represent the current page contained in window
window. history	Reference to history object for browser window or tab, which contains details of the pages that have been viewed in that window or tab
window.history.length	Number of items in history object for browser window or tab
window. screen	Reference to screen object
window. screen. width	Accesses screen object and finds value of its width property (in pixels)
window. screen. height	Accesses screen object and finds value of its height property (in pixels)

METHOD	DESCRIPTION
window. alert()	Creates dialog box with message (user must click Ok button to close it)
window. open()	Opens new browser window with URL specified as parameter (if browser has pop-up blocking software installed, this method may not work)
window. print()	Tells browser that user wants to print contents of current page (acts like user has clicked a print option in the browser's user interface)

Refer BOM.js