CHAPTER-6

JavaScript

JavaScript makes HTML pages more dynamic and interactive.

The HTML <script> Tag

The HTML <script> tag is used to define a client-side script (JavaScript).

The <script> element either contains script statements, or it points to an external script file through the src attribute. Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content. To select an HTML element, JavaScript most often uses the document.getElementById() method. This JavaScript example writes "Hello JavaScript!" into an HTML element with id="demo":

```
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First JavaScript</h1>
<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Click me to display Date and Time.</button>
<p id="demo"></p>
</body>
</html>
```

JavaScript Functions and Events

A JavaScript function is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an event occurs, like when the user clicks a button.

JavaScript in <head>

A JavaScript function is placed in the <head> section of an HTML page.

The function is invoked (called) when a button is clicked:

Example:

```
<!DOCTYPE html>
```

```
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>
<h2>Demo JavaScript in Head</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

JavaScript in <body>

A JavaScript function is placed in the <body> section of an HTML page.

The function is invoked (called) when a button is clicked:

Example:

```
<!DOCTYPE html>

<html>

<body>

<h2>Demo JavaScript in Body</h2>

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

<button type="button" onclick="myFunction()">Try it</button>

<script>

function myFunction() {

  document.getElementById("demo").innerHTML = "Paragraph changed.";

}

</script>

</body>

</html>
```

External JavaScript:

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension .js.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

MyScript.js

```
function myFunction() {
```

```
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

```
<!DOCTYPE html>
<html>
<body>
<h2>Demo External JavaScript</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
<p>This example links to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>
<script src="myScript.js"></script>
</body>
</html>
```

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

The HTML <noscript> Tag

The HTML <noscript> tag defines an alternate content to be displayed to users that have disabled scripts in their browser or have a browser that doesn't support scripts:

```
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
<noscript>Sorry, your browser does not support JavaScript!</noscript>
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
<noscript>Sorry, your browser does not support JavaScript!</noscript>
<p>A browser without support for JavaScript will show the text written inside the noscript element.</p>
 </body>
</html>
```

Variables in Javascript:

Variables are containers for storing data (storing data values).
In this example, x, y, and z, are variables, declared with the var keyword:
Always declare JavaScript variables with var,let, orconst.

The var keyword is used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015.

If you want your code to run in older browsers, you must use var.

4 Ways to Declare a JavaScript Variable:

- Using var
- Using let
- Using const
- Using nothing

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, x, y, and z are variables.</p>
<p id="demo"></p>
<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML = "The value of z is: " + z;
</script>
</body>
</html
```

```
<script>
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
document.getElementById("demo").innerHTML ="The total is: " + total;
</script>
```

Scope of a Variable:

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

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

Operators:

In JavaScript, an operator is a special symbol used to perform operations on operands (values and variables). For example, 2 + 3; // 5. Here + is an operator that performs addition, and 2 and 3 are operands.
There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.
In JavaScript we have the following conditional statements:

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

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

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

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

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

```
  greeting = "Good evening";
}
```

The else if Statement

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

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

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript if .. else</h2>
<p>A time-based greeting:</p>
<p id="demo"></p>
<script>
const time = new Date().getHours();
let greeting;
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
document.getElementById("demo").innerHTML = greeting;
</script>
</body>
</html>
```

Arrays: An array is a special variable, which can hold more than one value:

```
const cars = ["Saab", "Volvo", "BMW"];
```

Why Use Arrays?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

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

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Accessing Array Elements

You access an array element by referring to the index number:

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

```
<h2>JavaScript Arrays</h2>
<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>
<p id="demo"></p>
<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>
```

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

```
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

Changing an Array Element

This statement changes the value of the first element in cars:

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

Adding Array Elements

The easiest way to add a new element to an array is using the push() method:

```
const fruits = ["Banana", "Orange", "Apple"];
fruits.push("Lemon");
```

Removing last element :

The pop() method removes the last element from an array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

let fruit = fruits.pop();
```

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

JavaScript Function Syntax

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses (). Function names can contain letters, digits, underscores, and dollar signs (same rules as variables). The parentheses may include parameter names separated by commas:

(parameter1, parameter2, ...)

The code to be executed, by the function, is placed inside curly brackets: {}

```
function name(parameter1, parameter2, parameter3) {
```

```
  // code to be executed
}
```

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

```
let x = myFunction(4, 3);   // Function is called, return value will end up in x

function myFunction(a, b) {

  return a * b;           // Function returns the product of a and b

}
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function to convert from Fahrenheit to Celsius:</p>
<p id="demo"></p>
<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
</script>
</body>
</html>
```

Accessing function without()

```
<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius();
</script>
```

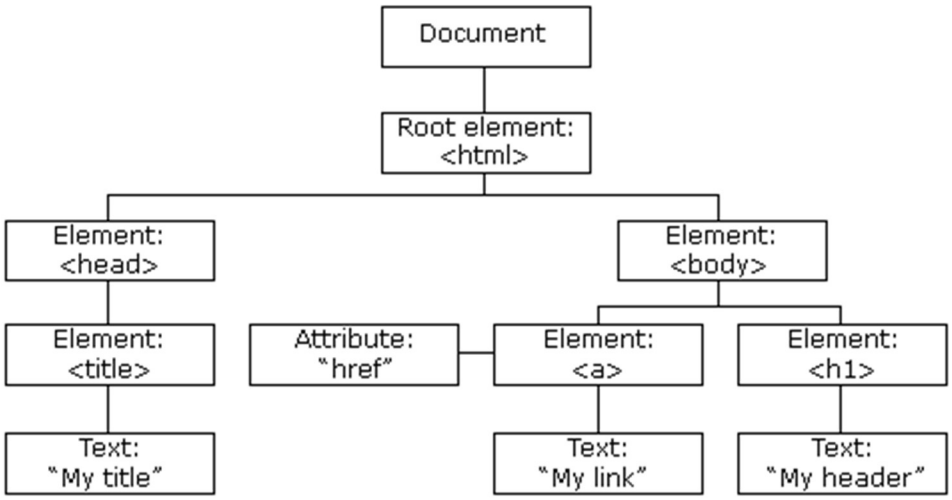
It will return function definition

The HTML DOM (Document Object Model)

The HTML DOM is a standard object model and programming interface for HTML. When a web page is loaded, the browser creates a Document Object Model of the page. 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

The HTML DOM model is constructed as a 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