

Lecture 3

Java script. Basics.
JS DOM

Introduction

JavaScript is a programming language, mostly used to define the interactive behavior of web pages. It was invented by Brendan Eich.

JavaScript allows you to make web pages more interactive by accessing and modifying the contents and styling in a web page while it is being viewed in the browser.

JavaScript, along with HTML and CSS, forms the foundation of modern web browsers and the internet.

JavaScript is considered, unofficially, to be the most popular programming language in the world .

Numerous open-source mapping and data visualization libraries are written in JavaScript, including **Leaflet** and **MapLibre GL JS**, as well as **OpenLayers**, **D3**, and many others.

Many commercial services of web mapping also provide a JavaScript API for building web maps with their tools, such as **Google Maps JavaScript API**, **Mapbox GL JS** and **ArcGIS API for JavaScript**

Java script. Basics.

JavaScript was initially created to “make web pages alive”.

The programs in this language are called *scripts*. They can be written right in a web page’s HTML and run automatically as the page loads.

JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.

The browser has an embedded engine sometimes called a “JavaScript virtual machine”.

Different engines have different “codenames”.

V8 – in Chrome, Opera and Edge.

SpiderMonkey – in Firefox.

There are at least *three* great things about JavaScript:

- Full integration with HTML/CSS.
- Simple things are done simply.
- Supported by all major browsers and enabled by default.

JavaScript is the only browser technology that combines these three things.

JavaScript also allows to create servers, mobile applications, etc.

Examples of what JavaScript can do

1. JavaScript Can Change HTML Content:

One of many JavaScript HTML methods is `getElementById()`.

The example "finds" an HTML element (with `id="demo"`), and changes the element content (`innerHTML`) to "Hello JavaScript":

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

(The Element property **innerHTML** gets or sets the HTML markup contained within the element.)

2. JavaScript Can Change HTML Attribute Values.

3. JavaScript Can Change HTML Styles (CSS):

```
document.getElementById("demo").style.fontSize = "35px";
```

4. JavaScript Can Hide HTML Elements:

```
document.getElementById("demo").style.display = "none";
```

5. JavaScript Can Show HTML Elements (showing hidden HTML elements can also be done by changing the display style):

```
document.getElementById("demo").style.display = "block";
```

How to Create .Js

1. Create a new folder named scripts.
2. Within the scripts folder, create a new text document called **main.js**, and save it.

In your **index.html** file, enter this code on a new line, just before the closing `</body>` tag:

```
<script src="scripts/main.js">  
</script>
```

This is doing the same job as the `<link>` element for CSS. It applies the JavaScript to the page, so it can have an effect on the HTML (along with the CSS, and anything else on the page).

3. Add this code to the **main.js** file:

```
const myHeading = document.querySelector("h1");  
myHeading.textContent = "Hello world!";
```

Make sure the HTML and JavaScript files are saved. Then load **index.html** in your browser.

1. The heading text changed to *Hello world!* using JavaScript. It is done by using a function called `querySelector()` to grab a reference to your heading, and then store it in a variable called **myHeading**. This is similar to what was done when using CSS selectors. When you want to do something to an element, you need to select it first.
2. Following that, the code set the value of the **myHeading** variable's `textContent` property (which represents the content of the heading) to *Hello world!*.



Adding JavaScript to Your Web Pages

There are typically three ways to add JavaScript to a web page:

- Embedding the JavaScript code between a pair of `<script>` and `</script>` tag.
- Creating an external JavaScript file with the .js extension and then load it within the page through the `src` attribute of the `<script>` tag.
- Placing the JavaScript code directly inside an HTML tag using the special tag attributes such as `onclick`, `onmouseover`, `onkeypress`, `onload`, etc.

Embedding the JavaScript Code

You can embed the JavaScript code directly within your web pages by placing it between the `<script>` and `</script>` tags.

The `<script>` tag indicates the browser that the contained statements are to be interpreted as executable script and not HTML.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Embedding JavaScript</title>
</head>
<body>
  <script>
    var greet = "Hello World!";
    document.write(greet); // Prints: Hello World!
  </script>
</body>
</html>
```

Calling an External JavaScript File

You can also place your JavaScript code into a separate file with a **.js** extension, and then call that file in your document through the **src** attribute of the **<script>** tag, like this:

```
<script src="js/hello.js"> </script>
```

This is useful if you want the same scripts available to multiple documents. It saves you from repeating the same task over and over again, and makes your website much easier to maintain.

1. JavaScript file named "hello.js" and place the following code in it (Pic.1)
2. Call this external JavaScript file within a web page using the **<script>** tag (Pic.2)

```
// A function to display a message
function sayHello() {
    alert("Hello World!");
}

// Call function on click of the button
document.getElementById("myBtn").onclick = sayHello;
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Including External JavaScript File</title>
</head>
<body>
    <button type="button" id="myBtn">Click Me</button>
    <script src="js/hello.js"></script>
</body>
</html>
```


External JavaScript

External file: myScript.js

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

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:

Example

```
<script src="myScript.js"></script>
```

- You can place an external script reference in `<head>` or `<body>` as you like.
- The script will behave as if it was located exactly where the `<script>` tag is located.
- External scripts cannot contain `<script>` tags.

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

To add several script files to one page - use several script tags:

Example

```
<script src="myScript1.js"></script>  
<script src="myScript2.js"></script>
```

External References

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
- With a file path (like /js/)
- Without any path

This example uses a **full URL** to link to myScript.js:

Example

```
<script src="https://www.....com/js/myScript.js"></script>
```

Placing the JavaScript Code Inline

- You can also place JavaScript code inline by inserting it directly inside the HTML tag using the special tag attributes such as *onclick*, *onmouseover*, *onkeypress*, *onload*, etc.
- However, you should avoid placing large amount of JavaScript code inline as it clutters up your HTML with JavaScript and makes your JavaScript code difficult to maintain.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Inlining JavaScript</title>
</head>
<body>
  <button onclick="alert('Hello World!')">Click Me</button>
</body>
</html>
```

Positioning of Script inside HTML Document

The `<script>` element can be placed in the **`<head>`**, or **`<body>`** section of an HTML document.

Ideally, scripts should be placed at the end of the body section, just before the closing **`</body>`** tag, it will make your web pages load faster, since it prevents obstruction of initial page rendering.

Each **`<script>`** tag blocks the page rendering process until it has fully downloaded and executed the JavaScript code, so placing them in the head section (i.e. `<head>` element) of the document without any valid reason will significantly impact your website performance.

JavaScript in <head>

In this example, 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>

In this example, 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>
```

Difference Between Client-side and Server-side Scripting

Client-side scripting languages such as JavaScript, VBScript, etc. are interpreted and executed by the web browser, while server-side scripting languages such as PHP, ASP, Java, Python, Ruby, etc. runs on the web server and the output sent back to the web browser in HTML format.

Client-side scripting has many advantages over traditional server-side scripting approach.

For example, you can use JavaScript to check if the user has entered invalid data in form fields and show notifications for input errors accordingly in real-time before submitting the form to the web-server for final data validation and further processing in order to prevent unnecessary network bandwidth usages and the exploitation of server system resources.

Also, response from a server-side script is slower as compared to a client-side script, because server-side scripts are processed on the remote computer not on the user's local computer.

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using `innerHTML`.
- Writing into the HTML output using `document.write()`.
- Writing into an alert box, using `window.alert()`.
- Writing into the browser console, using `console.log()`

Create a simple slide show

1. We will need a container for the slides and the slides themselves. This is how it will look like:

```
1 <ul id="slides">
2   <li class="slide showing">Slide 1</li>
3   <li class="slide">Slide 2</li>
4   <li class="slide">Slide 3</li>
5   <li class="slide">Slide 4</li>
6   <li class="slide">Slide 5</li>
7 </ul>
```

Task

Base styles should:

- Set container for slides
- Position slides one above the other inside a container
- Define how slides appear and disappear
- Smoothly change transparency for fading and fading effect

Before looking into the CSS, don't forget to change the classes and IDs so that there are no conflicts with your sites.

CSS

Here, the names of classes and identifiers will be short.

```
#slides {  
  position: relative;  
  height: 300px;  
  padding: 0px;  
  margin: 0px;  
  list-style-type: none;  
}
```

```
.slide {  
  position: absolute;  
  left: 0px;  
  top: 0px;  
  width: 100%;  
  height: 100%;  
  opacity: 0;  
  z-index: 1;  
  
  -webkit-transition: opacity 1s;  
  -moz-transition: opacity 1s;  
  -o-transition: opacity 1s;  
  transition: opacity 1s;  
}
```

```
.showing {  
  opacity: 1;  
  z-index: 2;  
}
```

Now you can add styles to the appearance of your slideshow. For the demo it has been used the following styles:

```
1  /*
2  You can change the style
3  */
4
5  .slide {
6    font-size: 40px;
7    padding: 40px;
8    box-sizing: border-box;
9    background: #333;
10   color: #fff;
11 }
12
13 .slide:nth-of-type(1) {
14   background: red;
15 }
16 .slide:nth-of-type(2) {
17   background: orange;
18 }
19 .slide:nth-of-type(3) {
20   background: green;
21 }
22 .slide:nth-of-type(4) {
23   background: blue;
24 }
25 .slide:nth-of-type(5) {
26   background: purple;
27 }
28
```

JavaScript part

JS does one thing: it hides the current slide and shows the next one. To do this, we need to change the names of the slide classes. Here is our JS code:

```
3 var slides = document.querySelectorAll('#slides .slide');
4 var currentSlide = 0;
5 var slideInterval = setInterval(nextSlide, 2000);
6
7 function nextSlide() {
8     slides[currentSlide].className = 'slide';
9     currentSlide = (currentSlide + 1) % slides.length;
10    slides[currentSlide].className = 'slide showing';
11 }
```

Details

1. First, we use **querySelectorAll** to get all the slides from the container (The `querySelectorAll()` Document method returns a static (not dynamic) `NodeList` containing all found document elements that match the specified selector).
2. We then create a variable to get the current slide.
3. At the end, we set an interval of two seconds for the next slide (2000ms).

Let's take a closer look at the **nextSlide** function:

1. We are changing the class for the current slide to hide it. The transition property automatically handles the fade.
2. Then we add a class to the current slide.
3. We use the `%` operator in case it was the last slide to go back to the first one. This operator is great for cases where you need to perform mathematical operations on cycles such as clocks or calendars.
4. In our case, 5 slides. Let's count all the numbers: $1\%5=1$, $2\%5=2$, $3\%5=3$, $4\%5=4$, and $5\%5=0$.
5. After getting the index of the slide, we change the class and show the new one. Again, transparency is handled by the *transition* property.

Adding controls to the slider

Time to add a pause button, next slide and previous slide.

Pause button. First, add the button to the HTML:

```
1 <button class="controls"
  id="pause">Pause</button>
```

Then add this JS code ----->

What happens in the script:

1. The **playing** variable is stored when the slider is active.
2. We put the pause button in the **pauseButton** variable so that we don't have to search for it later in the document.
3. The **pauseSlideshow** function stops the slider and writes "Play" to the pause button.
4. The **playSlideshow** function starts the slider and sets Pause to the Play button.
5. At the end, we hang up a click handler so that the play/pause button can pause the slider and start it.

```
var playing = true;
var pauseButton = document.getElementById('pause');

function pauseSlideshow() {
  pauseButton.innerHTML = 'Play';
  playing = false;
  clearInterval(slideInterval);
}

function playSlideshow() {
  pauseButton.innerHTML = 'Pause';
  playing = true;
  slideInterval = setInterval(nextSlide,2000);
}

pauseButton.onclick = function() {
  if(playing) {
    pauseSlideshow();
  } else {
    playSlideshow();
  }
};
```

slide 1

Pause

Next and Previous buttons

First, add the Next and Previous buttons in the HTML:

```
1 <button class="controls"
2 id="previous">Previous</butt
on>
<button class="controls"
id="next">Next</button>
```

In JavaScript, change the function for the other one:

```
1 function nextSlide() {
2   slides[currentSlide].className =
3   'slide';
4   currentSlide =
5   (currentSlide+1)%slides.length;
   slides[currentSlide].className =
   'slide showing';
}
```

```
function nextSlide() {
4   goToSlide(currentSlide+1);
5 }
6
7 function previousSlide() {
8   goToSlide(currentSlide-1);
9 }
1
10 function goToSlide(n) {
1   slides[currentSlide].className = 'slide';
1   currentSlide =
1   (n+slides.length)%slides.length;
2   slides[currentSlide].className = 'slide
1 showing';
3 }
```

For more flexibility in the script above, was added a generic **goToSlide** function. Also, in order not to get a negative value, it was slightly changed the way the **currentSlide** variable is calculated.

For testing purposes, you can replace **slides.length** with your own number and see what gets into **currentSlide** as n changes.

Finish the code below to the script to make the buttons work

```
2 var next = document.getElementById('next');
3 var previous = document.getElementById('previous');
4
5
6 next.onclick = function() {
7     pauseSlideshow();
8     nextSlide();
9 };
10
11 previous.onclick = function() {
12     pauseSlideshow();
13     previousSlide();
14 };
15
```

Result:



Understanding the Document Object Model

In order for JavaScript to have access to the elements in your web page, it needs to know how to find them. The Document Object Model (DOM) provides a standard way of accessing objects placed within a web page. It creates a tree structure that contains every element, attribute, content text, and even CSS3 style contained within the web page.

It treats each of these items as objects that the browser (or your program code) can manipulate. Finding any of these items is just a matter of walking through the tree with your JavaScript code.

The browser defines every web page as a set of DOM objects that the web page contains.

Just as your family has a family tree that you can trace back to find relatives, every web page has its own DOM tree of the objects contained within the web page.

With JavaScript, you can peruse through the DOM tree and make modifications along the way.

The Document Object Model tree

Every family tree has a head, and for the DOM tree, the head is the html element that starts out the web page. Just as parents have children, the html object in the DOM tree has two child objects: the head object and the body object.

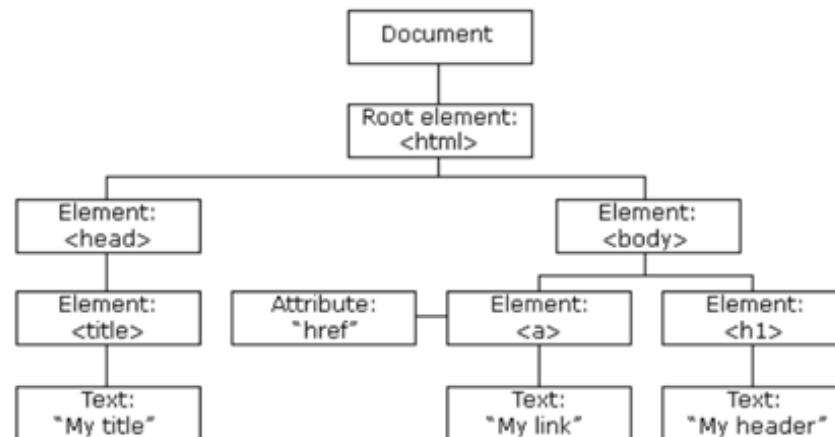
The head and body elements in the HTML code are called *child objects* of the html object in the DOM tree. Because it comes first in the code, the head object is called the “first child object,” while the body object is the “last child object.” This terminology is important when working with DOM objects. In DOM terminology these individual parts of the document are known as *nodes*.

With the HTML DOM, you can use JavaScript to build HTML documents, navigate their hierarchical structure, and add, modify, or delete elements and attributes or their content, and so on.

Almost anything found in an HTML document can be accessed, changed, deleted, or added using the JavaScript with the help of HTML DOM.

JavaScript DOM Nodes

With the HTML DOM, JavaScript can access and change all the elements of an HTML document. When a web page is loaded, the browser creates a **Document Object Model** of the page. 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

JavaScript Interactivity.

The Document Object Model (DOM)

When a web browser loads an HTML document, it displays (renders) the contents of that document on the screen, possibly styled according to CSS styling rules.

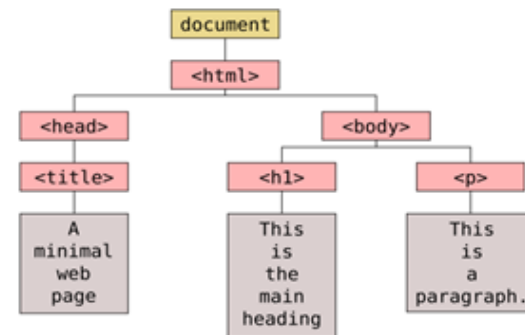
But that's not all the web browser does with the tags, attributes, and text contents of the HTML document. The browser also creates and memorizes a “model” of that page's HTML.

In other words, the browser remembers the HTML tags, their attributes, and the order in which they appear in the file.

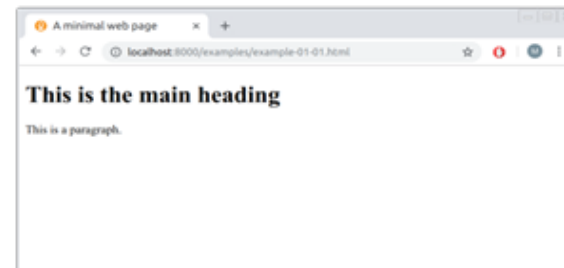
This representation of the page is called the **Document Object Model (DOM)**.

```
<!DOCTYPE html>
<html>
  <head>
    <title>A minimal web page</title>
  </head>
  <body>
    <h1>This is the main heading</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

The browser receives an HTML document



The browser creates a model of the page



The browser displays the page on screen

The principal standardization of the DOM was handled by the World Wide Web Consortium (W3C), which last developed a recommendation in 2004.

The W3C now publishes stable snapshots of the WHATWG standard.

In HTML DOM (Document Object Model), every element is a node:

- A document is a document node.
- All HTML elements are element nodes.
- All HTML attributes are attribute nodes.
- Text inserted into HTML elements are text nodes.
- Comments are comment nodes.

The DOM is sometimes referred to as an **Application Programming Interface (API)** for accessing HTML documents with JavaScript.

An API is a general term for methods of communication between software components.

The DOM can be considered an API, since it bridges between JavaScript code and page contents displayed in the browser. The DOM basically provides the information and tools necessary to navigate through, or make changes or additions, to the HTML on the page.

The browser creates the DOM when the page loads, stores it in the document variable, and reports that the DOM has been created with the **DOMContentLoaded** event. The document variable is where all work with HTML markup in JavaScript begins.

JavaScript and the Document Object Model

The browser uses the DOM tree to keep track of all the HTML5 elements, their content, and the styles that appear on the web page. However, because JavaScript programs run in the browser, they have full access to the DOM tree created by the browser.

That means your JavaScript programs can interact directly with the DOM tree that the browser follows to create the web page. And not only that, but your JavaScript programs can add, change, and even remove objects in the DOM tree.

As your JavaScript program modifies the DOM tree, the browser automatically updates the web page window with the new information. This is the key to client-side dynamic web programming. Just like the DOM tree, JavaScript treats each element contained in a web page as an object.

In JavaScript, objects have two features:

- **Properties:** Properties define information about the object.
- **Methods:** Methods are actions to take with the objects.

JavaScript assigns a special object named `document` to represent the entire web page DOM tree. You can reference many of the DOM objects directly from the `document` object, as well as add or remove objects.

TABLE 2-1

JavaScript Document Properties

Property	Description
<code>activeElement</code>	Returns the element that currently has the focus of the web page window
<code>anchors</code>	Returns a list of all the anchor elements on the web page
<code>body</code>	Sets or retrieves the body element of the web page
<code>cookie</code>	Returns all cookie names and values set in the web page
<code>characterSet</code>	Returns the character set defined for the web page
<code>documentElement</code>	Returns the DOM object for the html element of the web page
<code>documentMode</code>	Returns the mode used by the browser to display the web page
<code>domain</code>	Returns the domain name of the server used to send the document
<code>embeds</code>	Returns a list of all the embed elements in the web page
<code>forms</code>	Returns a list of all the form elements in the web page
<code>head</code>	Returns the head element for the web page
<code>images</code>	Returns a list of all the img elements in the web page
<code>lastModified</code>	Returns the time and date the web page was last modified
<code>links</code>	Returns a list of all the anchor and area elements in the web page
<code>title</code>	Sets or retrieves the title of the web page
<code>URL</code>	Returns the full URL for the web page

TABLE 2-2

JavaScript Document Methods

Method	Description
<code>createElement()</code>	Adds a new element object
<code>createTextNode()</code>	Adds a new text object
<code>getElementById(<i>id</i>)</code>	Returns an element object with the specified id value
<code>getElementsByName(<i>class</i>)</code>	Returns a list of elements with the specified class name
<code>getElementsByTagName(<i>tag</i>)</code>	Returns a list of elements of the specified element type
<code>hasFocus()</code>	Returns a true value if the web page has the window focus
<code>write(<i>text</i>)</code>	Sends the specified text to the web page
<code>writeln(<i>text</i>)</code>	Sends the specified text to the web page, followed by a new line character

Example

Take these steps to test using the **write() method** for a web page document:

- 1. Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
- 2. Enter the following code:**

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Test</title>
    <script>
      document.write("<h1>This is a test of the DOM</h1>");
    </script>
  </head>
<body>
</body>
</html>
```

- 3. Save the file as domtest.html in the DocumentRoot folder for your web server.**

If you're using XAMPP in Windows, that's the c:\xampp\htdocs folder; for XAMPP in macOS, it's /Applications/XAMPP/htdocs.

- 4. Open the XAMPP Control Panel and then start the Apache Web server.**

- 5. Open your browser and enter the following URL:**

<http://localhost:8080/domtest.html>

You may need to change the TCP port in the URL to match your Apache web server.

- 6. Close the browser.**

When you examine the code in the **domtest.html file**, you'll notice that there's nothing in the body element, so you may not expect to see anything on the resulting web page. However, when you run the program, you should see the output.

Explanation

The **document.write()** function runs the **write() method** from the document object to dynamically place the **h1** element in the web page for us.

The **write() method** is an easy way to dynamically place text in the web page.

The write() method overwrites everything that was originally in the web page.

In this example, it has been ran from the head element, so it placed the output at the top of the web page, before any elements defined in the body element.

However, if you use the **write() function** from within the body element, it'll remove any elements that were previously on the web page.

Besides the document properties and methods, JavaScript also has properties and methods that apply to each element object in the document.

JavaScript DOM object properties

Now that you have access to the objects contained in the web page, you can use JavaScript to manipulate them. Each DOM object contains one or more properties that define the actual object.

There are lots of object properties JavaScript uses with objects.

Property	Description
attributes	Returns a list of the object's attributes
childElementCount	Returns a list of the number of child objects the object has
childNodes	Returns a list of the object's child nodes, including text and comments
children	Returns a list of only the object's child element object nodes
classList	Returns a list of the class name attributes of an object
className	Sets or returns the value of a class attribute of an object
firstChild	Returns the first child object for the object
id	Sets or returns the id value of the object
innerHTML	Sets or returns the HTML content of the object
lastChild	Returns the last child object for the object
nodeName	Returns the name of the object
nodeType	Returns the element type of the object
nodeValue	Sets or returns the value for the object
nextSibling	Returns the next object at the same level in the tree as the object
parentNode	Returns the parent object for the object
previousSibling	Returns the previous object at the same level in the tree as the object
style	Sets or returns the value of the style property for the object

Try it

```
<!DOCTYPE html>
<html>
  <head>
    <title>Testing DOM properties</title>
  </head>
  <body>
    <body>
      <h1>This is the heading of the web page</h1>
      <p>This is sample text</p>
      <br>
      <button type="button" onclick="changeme('red')">Change background to red</button>
      <button type="button" onclick="changeme('white')">Change background to white</button>
    </body>
  </body>
  <script>
function changeme(color) {document.body.style.backgroundColor = color;}
  </script>
</body>
</html>
```

Run the same way as it was described previously, but you may need to change the TCP port to match your Apache web server.

The **domproperties.html** code uses two buttons to trigger the **changeme()** function. The **changeme()** function uses the **document.body** object to reference the body element in the web page.

It then uses the style object property to reference the CSS3 styles applied to the body element.

The **backgroundColor** style property isn't background-color, because that's how CSS3 defines that property. The DOM standard doesn't like using dashes in property names. So, instead, wherever there's a dash in a CSS3 property name (such as in background-color), it removes the dash and capitalizes the first letter of the next word. That's how we get **backgroundColor** as the DOM property to change the **background-color CSS3 property** on the web page.

JavaScript DOM object methods

Besides properties, JavaScript objects also contain methods. The methods provide actions to interact with the object. There are plenty more object methods for you to use in your JavaScript programs to help you retrieve information about the DOM objects, modify existing DOM objects, or even add new DOM objects to your web page.

Method	Description
<code>appendChild(object)</code>	Adds a new child object to an existing object
<code>blur()</code>	Removes the page focus from an object
<code>click()</code>	Simulates a mouse click on the object
<code>cloneNode</code>	Duplicates an object in the DOM
<code>contains(object)</code>	Returns a true value if the object contains the specified object
<code>focus()</code>	Places the window focus on the object
<code>getAttribute(attr)</code>	Returns the value for the specified object attribute
<code>getElementsByClassName(class)</code>	Returns a list of objects with the specified class name
<code>getElementsByTagName(tag)</code>	Returns a list of objects with the specified tag name
<code>hasAttribute(attr)</code>	Returns true if the object contains the specified attribute
<code>hasAttributes()</code>	Returns true if the object contains any attributes
<code>hasChildNodes()</code>	Returns true if the object contains any child objects
<code>insertBefore(object)</code>	Inserts the specified object before the object
<code>removeAttribute(attr)</code>	Removes the specified attribute from the object
<code>removeChild(object)</code>	Removes the specified child object from the parent object
<code>replaceChild(object)</code>	Replaces the child object with the specified object
<code>setAttribute(attr)</code>	Sets the specified attribute of the object to the specified value
<code>toString()</code>	Converts the object to a string value

How it works

The browser creates a DOM based on the HTML code of the page.

Then, based on the DOM and other structures, the browser renders the page to the user.

When the DOM changes, the browser parses it and updates the page.

The DOM and the page the user sees are linked.

If you change one, the other will also change:

- the user fills out the form - you can read the entered values in the DOM;
- when choosing a country, we fill in the list of cities in the DOM - the user sees the list.

Questions:

1. What is JavaScript?

Answer:

JavaScript is a **lightweight, interpreted** programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages.

The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

2. What are the data types supported by JavaScript?

The data types supported by JavaScript are:

- Undefined
- Null
- Boolean
- String
- Symbol
- Number
- Object

3. What are the advantages of JavaScript?

Following are the advantages of using JavaScript :

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.

4. What is the difference between Attributes and Property?

Answer:

- **Attributes-** provide more details on an element like id, type, value etc.
- **Property-** is the value assigned to the attribute like type="text", value='Name' etc.



5. List out the different ways an HTML element can be accessed in a JavaScript code.

Here are the list of ways an HTML element can be accessed in a Javascript code:

- **getElementById('idname')**: Gets an element by its ID name
- **getElementsByClass('classname')**: Gets all the elements that have the given classname.
- **getElementsByTagName('tagname')**: Gets all the elements that have the given tag name.
- **querySelector()**: This function takes css style selector and returns the first selected element.

6. In how many ways a JavaScript code can be involved in an HTML file?

Answer :

There are 3 different ways in which a JavaScript code can be involved in an HTML file:

- **Inline**
- **Internal**
- **External**

An **inline** function is a JavaScript function, which is assigned to a variable created at runtime. You can differentiate between Inline Functions and Anonymous since an inline function is assigned to a variable and can be easily reused.

When you need a JavaScript for a function, you can either have the script **integrated** in the page you are working on, or you can have it placed in a **separate** file that you call, when needed.

This is the difference between an **internal** script and an **external** script.



7. What is the difference between null & undefined?

Answer

Undefined means a variable has been **declared** but has not yet been **assigned** a value.

On the other hand, null is an assignment value. It can be assigned to a variable as a representation of no value.

Also, undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.



Thank you !