9

The Document Object Model

The **Document Object Model** (**DOM**) is a lot more exciting than it may sound at first. In this chapter, we will introduce you to the DOM. This is a fundamental concept you will need to understand before working with JavaScript on web pages. It grabs an HTML page and turns it into a logical tree. If you do not know any HTML, no worries. We start with an HTML crash course section that you can skip if you are familiar with HTML.

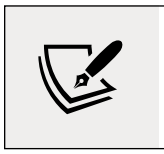
Once we are sure that we are on the same page with HTML knowledge, we will introduce you to the **Browser Object Model** (**BOM**). The BOM holds all the methods and properties for JavaScript to interact with the browser. This is information related to previous pages visited, the size of the window of the browser, and also the DOM.

The DOM contains the HTML elements on the web page. With JavaScript, we can select and manipulate parts of the DOM. This leads to interactive web pages instead of static ones. So, long story short, being able to work with the DOM means you're able to create interactive web pages!

We will cover the following topics:

- HTML crash course
- Introducing the BOM
- Introducing the DOM
- Types of DOM elements
- Selecting page elements

We can imagine you cannot wait to get started, so let's dive into it.



Note: exercise, project and self-check quiz answers can be found in the *Appendix*.

HTML crash course

Hyper-Text Markup Language (**HTML**) is the language that shapes the content of web pages. Web browsers understand HTML code and represent it in the format we are used to seeing: web pages. Here is a little very basic HTML example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Tab in the browser</title>
  </head>
  <body>
    <p>Hello web!</p>
  </body>
</html>
```

This is what this basic web page looks like:

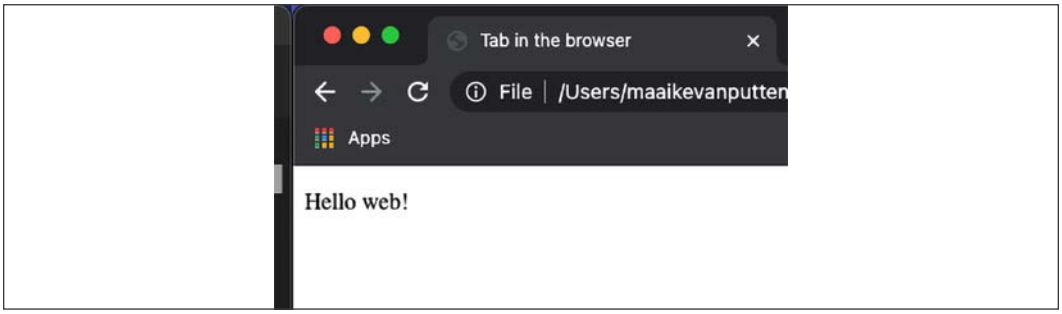


Figure 9.1: Basic website

HTML code consists of elements. These elements contain a tag and attributes. We will explain these fundamental concepts in the coming sections.

HTML elements

As you can see, HTML consists of words between `<angle brackets>`, or elements. Any element that gets opened needs to be closed. We open with `<elementname>` and we close with `</elementname>`.

Everything in between that is part of the element. There are a few exceptions with regards to the closing, but you will run into them at your own pace. In the previous example we had multiple elements, including these two. It is an element with the tag `body` and an inner element with the tag `p`:

```
  <body>
    <p>Hello web!</p>
  </body>
```

So elements can contain inner elements. Elements can only be closed if all inner elements have been closed. Here is an example to demonstrate that. Here is the right way:

```
<outer>
  <sub>
    <inner>
    </inner>
  </sub>
</outer>
```

And here is the wrong way:

```
<outer>
  <sub>
    <inner>
  </sub>
    </inner>
</outer>
```

Please note, these are just made-up element names. In the last example, we close `sub` before we have closed the `inner` element. This is wrong; you must always close the inner elements before closing the outer element. We call inner elements child elements, and outer elements parent elements. Here is some correct HTML:

```
<body>
  <div>
    <p>
    </p>
  </div>
</body>
```

This isn't correct HTML, because the `div` is closed before its inner element `p`:

```
<body>
  <div>
    <p>
  </div>
    </p>
</body>
```

The different elements represent different pieces of layout. The `p` we just saw represents paragraphs. And another common one is `h1`, which represents a big title. What is more important is to know the three major building elements of every HTML page. The HTML element, the head element, and the body element.

In the HTML element, all the HTML takes place. You can only have one of these in your HTML page. It is the outer element, and all other elements are housed in it. It contains the other two top-level elements: `head` and `body`. If you are ever confused about the order of `head` and `body`, just think of a human; the head is on top of the body.

In the `head` element, we arrange a lot of things that are meant for the browser and not for the user. You can think of certain metadata, such as which JavaScript scripts and which stylesheets need to be included, and what the searching engine should use as a description on the search result page. We will not really be doing a lot with the head element as JavaScript developers, other than including scripts.

Here's an example of a basic `head` element:

```
<head>
    <title>This is the title of the browser tab</title>
    <meta name="description" content="This is the preview in google">
<script src="included.js"></script>
</head>
```

The `body` element is mostly the content that will appear on the web page. There can only be one `body` element in the HTML element. Titles, paragraphs, images, lists, links, buttons, and many more are all elements that we can come across in the body. They have their own tag, so for example, `img` for image and `a` for a link. Here is a table including common tags for in the body. It is definitely not an exhaustive list.

Tag to open	Tag to end	Description
`<p>`	`</p>`	Used to create a paragraph.
`<h1>`	`</h1>`	Used to create a header; smaller headers are h2 to h6.
`<span>`	`</span>`	Generic inline container for content that needs to be separated, for example, for layout purposes.
`<a>`	`</a>`	Used for hyperlinks.
`<button>`	`</button>`	Used for buttons.
`<table>`	`</table>`	Creates a table.
`<tr>`	`</tr>`	Creates a table row, must be used inside a table.
`<td>`	`</td>`	Creates a table data cell inside a row.
`<ul>`	`</ul>`	Unordered lists, with bullet points, for example.
`<ol>`	`</ol>`	Ordered lists with numbers.
`<li>`	`</li>`	List items for inside ordered and unordered lists.
`<div>`	`</div>`	Section inside the HTML page. It is often used as a container for other styles or sections and can easily be used for special layouts.
`<form>`	`<form>`	Creates an HTML form.
`<input>`	`</input>`	Creates an input field in which the user can enter information. These can be textboxes, checkboxes, buttons, passwords, numbers, dropdowns, radio buttons, and much more.
`<input />`	`None`	Same as input, but written without a closing tag, the / at the end makes it self-closing. This is only possible for a few elements.
` `	`None`	Used to make a line break (go to a new line). It does not need an end tag and is therefore an exception.

Can you figure out what this HTML example does:

```
<html>

<head>
    <title>Awesome</title>
</head>

<body>
    <h1>Language and awesomeness</h1>
    <table>
```

```
        <tr>
            <th>Language</th>
            <th>Awesomeness</th>
        </tr>
        <tr>
            <td>JavaScript</td>
            <td>100</td>
        </tr>
        <tr>
            <td>HTML</td>
            <td>100</td>
        </tr>
    </table>
</body>

</html>
```

It creates a web page, with `Awesome` in the tab title. And on the page, it has a big header saying `Language and awesomeness`. Then there is a table with three rows and two columns. The first row contains the headers `Language` and `Awesomeness`. The second row holds the values `JavaScript` and `100`, and the third row holds the values `HTML` and `100`.

HTML attributes

The last part of HTML that we will discuss in this crash course is HTML attributes. Attributes influence the element they are specified on. They exist inside the element they are specified on and are assigned a value using an equal sign. For example, the attribute of `a` (which indicates a hyperlink) is the `href`. This specifies where the link is redirecting to:

```
<a href="https://google.com">Ask Google</a>
```

This displays a link with the text `Ask Google`. And when you click it, you will be sent to Google, which can be told by the value of the `href` attribute. This modifies the `a` element. There are many attributes out there, but for now you just need to know that they modify the element they are specified on.

Here is a table with an overview of the most important attributes to get started with HTML and JavaScript. Why these are important will unfold somewhere in the next chapter.

Attribute name	Description	Can be used on which element?
`id`	Gives an element a unique ID, such as age.	All of them
`name`	Used to give a custom name to an element.	`input`, `button`, `form`, and quite a few we haven't seen yet
`class`	Special metadata that can be added to an element. This can result in a certain layout or JavaScript manipulation.	Almost all of them inside body
`value`	Sets the initial value of the element it is added to.	`button`, `input`, `li`, and a few we haven't seen yet
`style`	Gives a specified layout to the HTML element it is added to.	All of them

We will introduce you to other attributes when you will need them for practicing your JavaScript magic.

Okay, this has been one of the more brief HTML crash courses out there. There are many great resources to find more information. If you need more information or explanation at this point, create and open an HTML file like the following and take it from there!

```
<!DOCTYPE html >
<html>

<body>
    <a href="https://google.com">Ask google</a>
</body>

</html>
```

We will now go ahead and have a look at the BOM and the different parts of the BOM.

The BOM

The BOM, sometimes also called the **window browser object**, is the amazing "magic" element that makes it possible for your JavaScript code to communicate with the browser.

The window object contains all the properties required to represent the window of the browser, so for example, the size of the window and the history of previously visited web pages. The window object has global variables and functions, and these can all be seen when we explore the window object. The exact implementation of the BOM depends on the browser and the version of the browser. This is important to keep in mind while working your way through these sections.

Some of the most important objects of the BOM we will look into in this chapter are:

- History
- Navigator
- Location

As well as the preceding useful objects, we will also consider the DOM in more detail. But first, we can explore the BOM and see the objects of it with the command `console.dir(window)`. We will enter this in the console of our browser. Let's discuss how to get there first.

We can access the HTML elements and the JavaScript if we go to the inspection panel of our browser. It differs a bit in how you get there, but often the *F12* button while in the browser will do the trick, or else a right-click on the website you want to see the console for and clicking on **Inspect element** or **Inspect** on a macOS device.

You should see a side panel (or if you have changed your settings, a separate window) pop up.

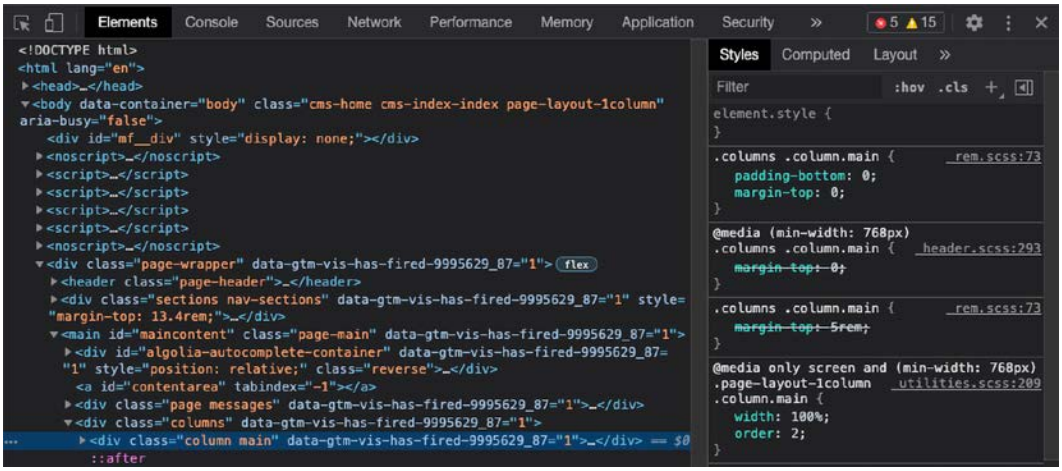


Figure 9.2: Inspecting a page in the browser

Navigate to the **Console** tab, which is next to the **Elements** tab in the image above. You can type the following command and press *Enter* to get information about the window object:

```
console.dir(window);
```

This command will produce a view like the following:

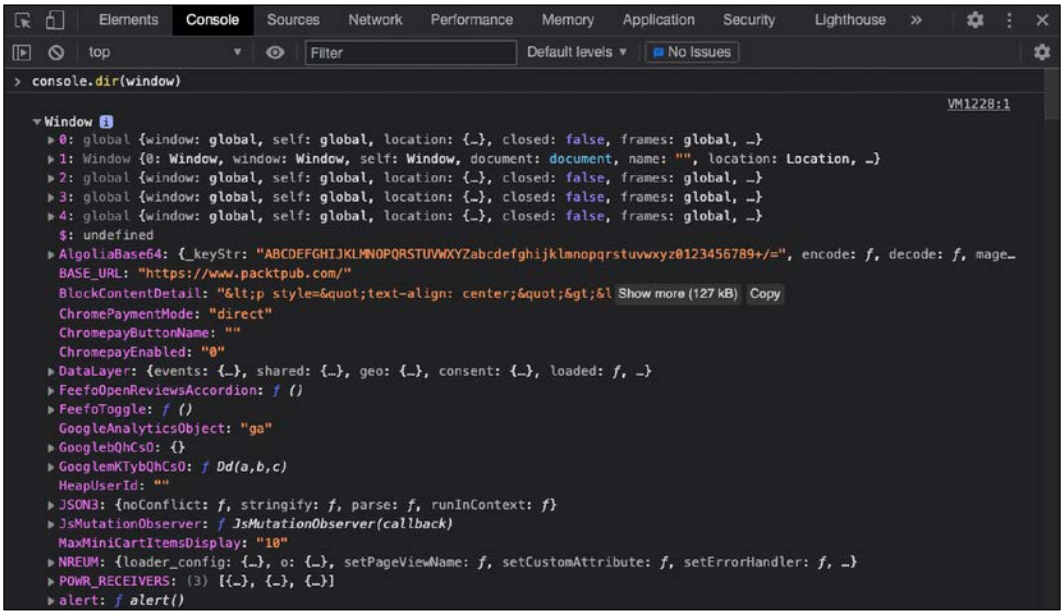


Figure 9.3: Part of the output of console.dir(window) showing the window browser object

The `console.dir()` method shows a list of all the properties of the specified object. You can click on the little triangles to open the objects and inspect them even more.

The BOM contains many other objects. We can access these like we saw when we dealt with objects, so for example, we can get the length of the history (in my browser) accessing the `history` object of the window and then the length of the `history` object, like this:

```
window.history.length;
```

After the exercise, we will learn more about the `history` object.

Practice exercise 9.1

1. Go back to the website you were just viewing and execute the command `console.dir(window)`.
2. Can you find the `document` object that is nested within the `window` object? Under the root of the `window` object in the console, you can navigate down to the object that is named `document`.
3. Can you find the height and width (in pixels) of your window? You can return the inner and outer window.

Window history object

The window browser object also contains a `history` object. This object can actually be written without the prefix of `window` because it has been made globally available, so we can get the exact same object by using the `console.dir(window.history)` or simply the `console.dir(history)` command in the console:

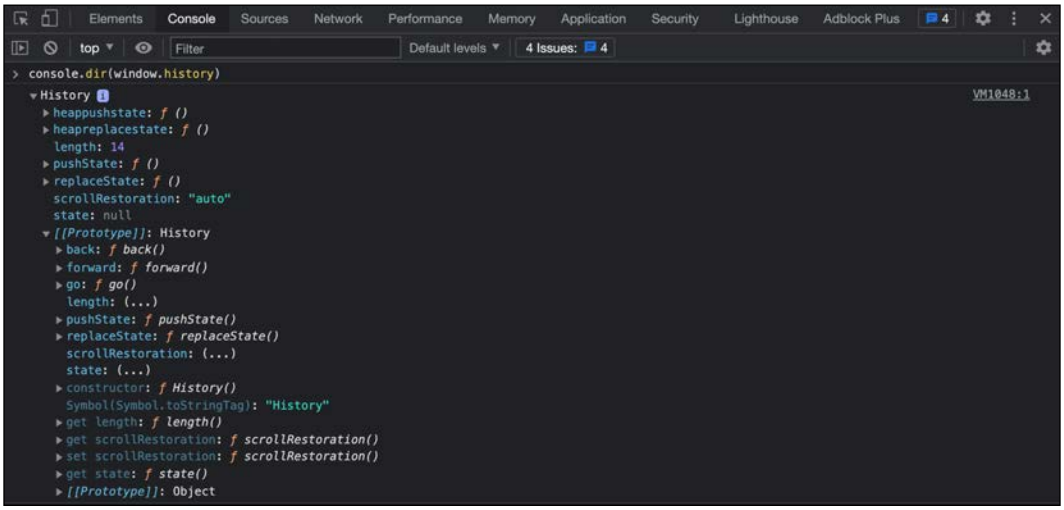


Figure 9.4: History object

This object is actually what you can use to go back to a previous page. It has a built-in function for that called `go`. What happens when you execute this command?

```
window.history.go(-1);
```

Go ahead and try it for yourself in the console of your browser (make sure that you did visit multiple pages in that tab).

Window navigator object

In the window object that we just saw, there is a `navigator` property. This property is particularly interesting because it contains information about the browser we are using, such as what browser it is and what version we are using, and what operating system the browser is running on.

This can be handy for customizing the website for certain operating systems. Imagine a download button that will be different for Windows, Linux, and macOS.

You can explore it using this command in the console:

```
console.dir(window.navigator);
```

As you can see, we start with accessing the window, because `navigator` is an object of the `window` object. So it is a property of the `window` object, which we specify with the dot in between. In other words, we access these `window` objects in the same way we do any other object. But, in this case, as `navigator` is also globally available, we can also access this without `window` in front of it with this command:

```
console.dir(navigator);
```

Here is what the `navigator` object might look like:

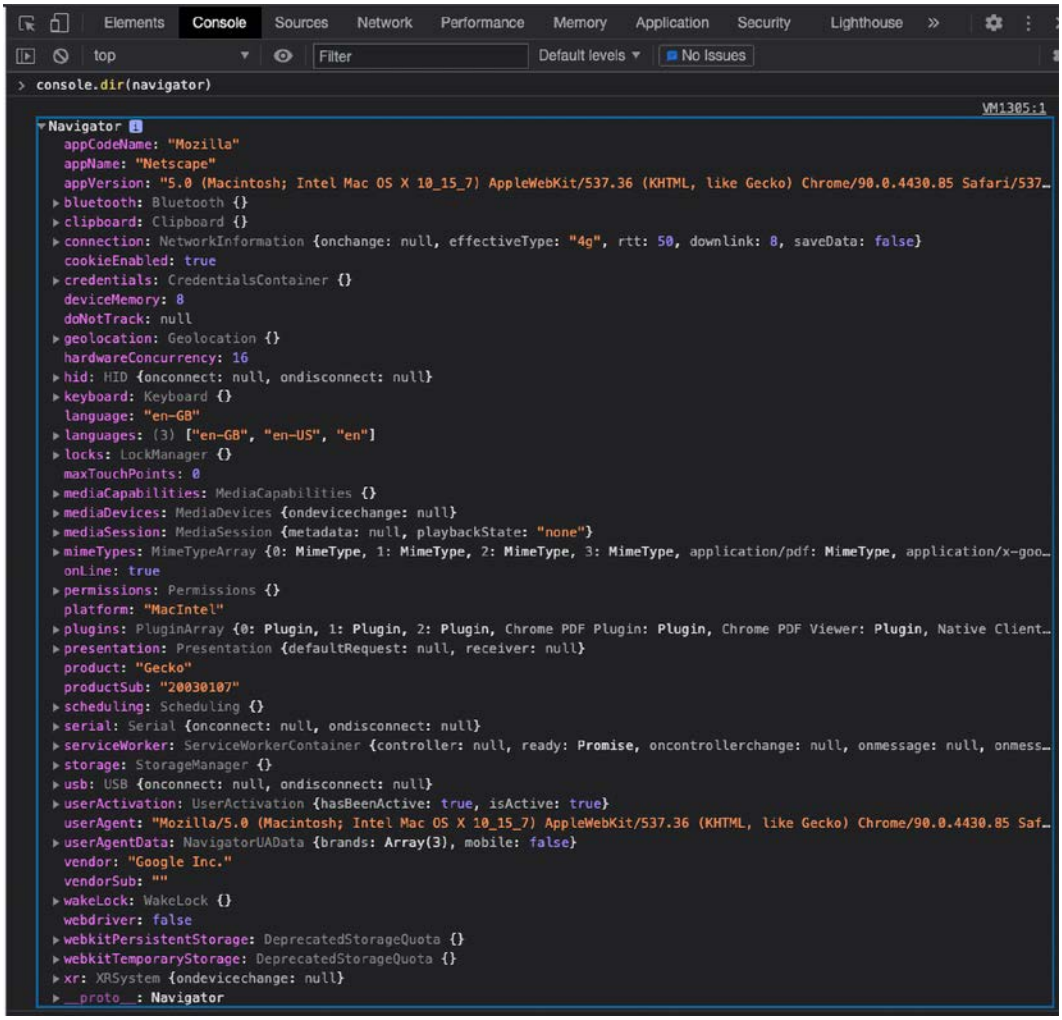


Figure 9.5: The navigator object

Window location object

Another rather interesting and unique property of `window` is the `location` object. This contains the URL of the current web page. If you override (parts of) that property, you force the browser to go to a new page! How to do this exactly differs per browser, but the next exercise will guide you through this.

The `location` object consists of a few properties. You can see them by using the command `console.dir(window.location)` or `console.dir(location)` in the console. Here is what the output will look like:

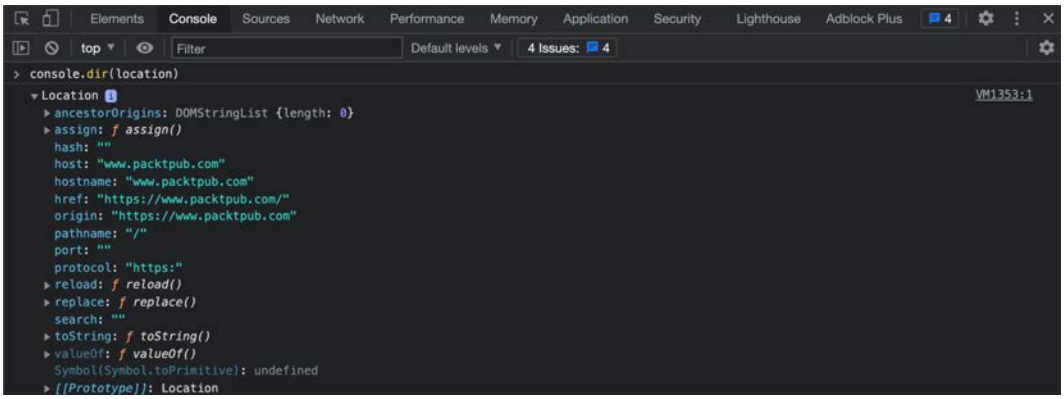


Figure 9.6: The location object

There are many objects on the `location` object, just as with the others we have seen. We can access the nested objects and properties using dot notation (like for other objects we have seen). So, for example, in this browser I can enter the following:

```
location.ancestorOrigins.length;
```

This will get the length of the `ancestorOrigins` object, which represents how many browsing contexts our page is associated with. This can be useful to determine whether the web page is framed in an unexpected context. Not all browsers have this object though; again, this BOM and all the elements of it vary per browser.

Follow the steps in the practice exercise to do such magic yourself.

Practice exercise 9.2

Travel through the `window` object to get to the `location` object, then output the values of the `protocol` and `href` properties of the current file, into the console.

The DOM

The DOM is actually not very complicated to understand. It is a way of displaying the structure of an HTML document as a logical tree. This is possible because of the very important rule that inner elements need to be closed before outer elements get closed.

Here is an HTML snippet:

```
<html>
  <head>
    <title>Tab in the browser</title>
  </head>
  <body>
    <h1>DOM</h1>
    <div>
      <p>Hello web!</p>
      <a href="https://google.com">Here's a link!</a>
    </div>
  </body>
</html>
```

And here is how we can translate it to a tree:

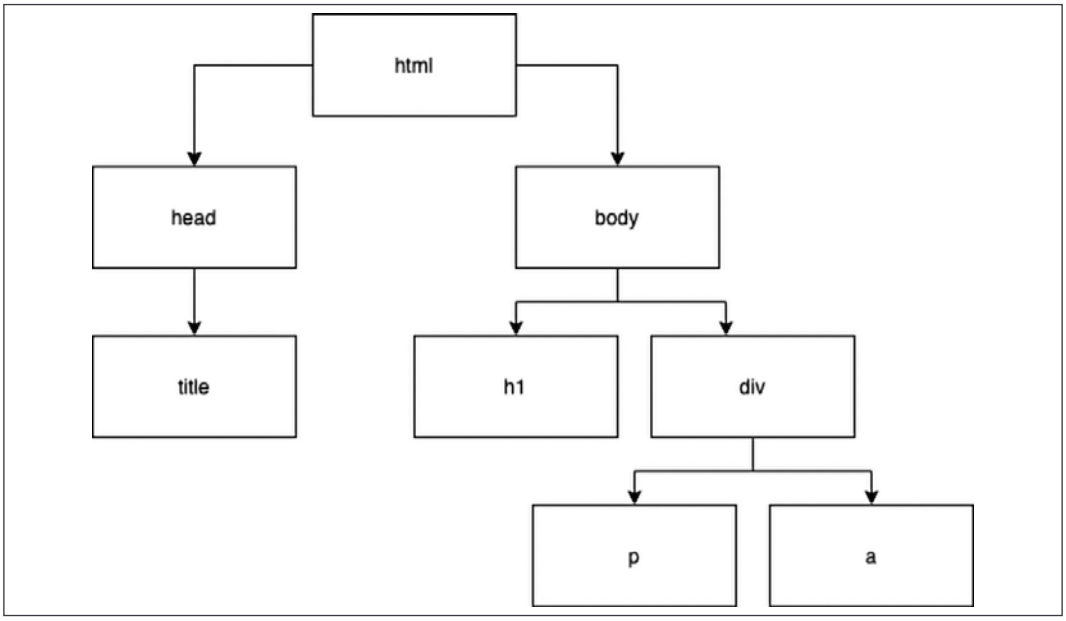


Figure 9.7: Tree structure of the DOM of a very basic web page

As you can see, the most outer element, **html**, is at the top of the tree. The next levels, **head** and **body**, are its children. **head** has only one child: **title**. **body** has two children: **h1** and **div**. And **div** has two children: **p** and **a**. These are typically used for paragraphs and links (or buttons). Clearly, complex web pages have complicated trees. This logical tree and a bunch of extra properties make up a web page's DOM.

The DOM of a real web page wouldn't fit on a page in this book. But if you can draw trees like these in your head, it will be of great help soon.

Additional DOM properties

We can inspect the DOM in a similar fashion as we did the others. We execute the following command in the console of our website (again, the `document` object is globally accessible, so accessing it through the `window` object is possible but not necessary):

```
console.dir(document);
```

In this case, we want to see the `document` object, which represents the DOM:

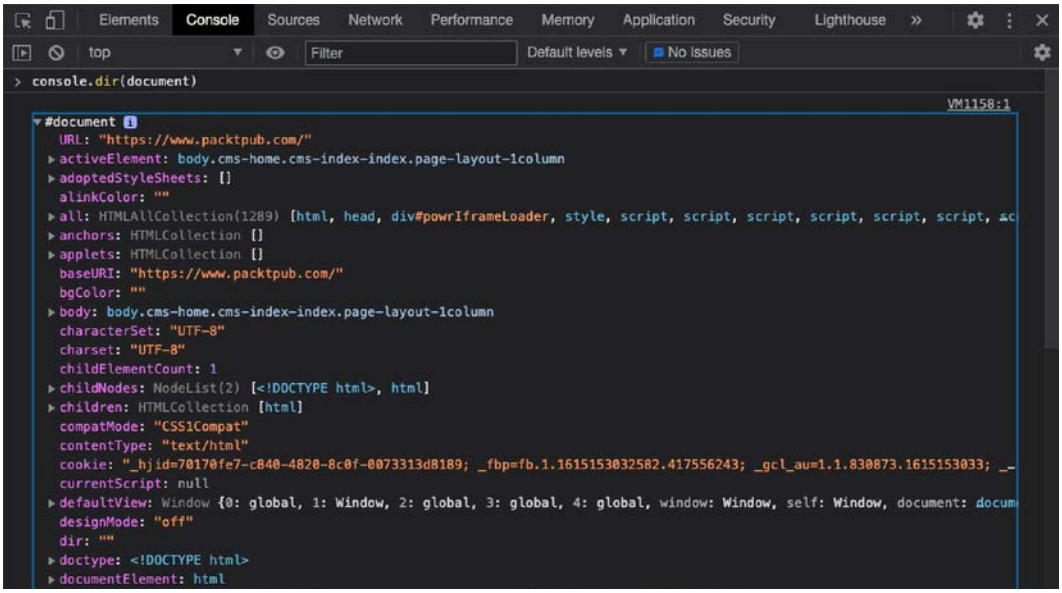


Figure 9.8: The DOM

You really do not need to understand everything you are seeing here, but it is showing many things, among which are the HTML elements and JavaScript code.

Great, right now you have got the basics of the BOM down, and its child object that is most relevant to us right now, the DOM. We saw many properties of the DOM earlier already. For us, it is most relevant to look at the HTML elements in the DOM. The DOM contains all the HTML elements of a web page.

These basics of DOM elements, combined with some knowledge of manipulating and exploring the DOM, will open up so many possibilities.

In the next chapter, we will focus on traversing the DOM, finding elements in the DOM, and manipulating the DOM. The code we will be writing there will really start to look like proper projects.

Selecting page elements

The `document` object contains many properties and methods. In order to work with elements on the page, you'll first have to find them. If you need to change the value of a certain paragraph, you'll have to grab this paragraph first. We call this selecting the paragraph. After selecting, we can start changing it.

To select page elements to use within your JavaScript code and in order to manipulate elements, you can use either the `querySelector()` or `querySelectorAll()` method. Both of these can be used to select page elements either by tag name, ID, or class.

The `document.querySelector()` method will return the first element within the document that matches the specified selectors. If no matching page elements are found, the result `null` is returned. To return multiple matching elements, you can use the method `document.querySelectorAll()`.

The `querySelectorAll()` method will return a static `NodeList`, which represents a list of the document's elements that match the specified group of selectors. We will demonstrate the usage of both `querySelector()` and `querySelectorAll()` with the following HTML snippet:

```
<!doctype html>
<html>
  <head>
    <title>JS Tester</title>
  </head>
  <body>
    <h1 class="output">Hello World</h1>
    <div class="output">Test</div>
  </body>
</html>
```

We are going to select the `h1` element with `querySelector()`. Therefore, if there is more than one, it will just grab the first:

```
const ele1 = document.querySelector("h1");
console.dir(ele1);
```

If you want to select multiple elements, you can use `querySelectorAll()`. This method is going to return all the elements that match the selector in an array. In this example, we are going to look for instances of the `output` class, which is done by prepending the class name with a dot.

```
const eles = document.querySelectorAll(".output");
console.log(eles);
```

After selecting, you can start using the dynamic features of the DOM: you can manipulate the elements using JavaScript. Content can be changed in the same way a variable's contents can be, elements can be removed or added, and styles can be adjusted. This can all be done with JavaScript and the way the user interacts with the page can affect this. We have seen the two most common methods to select in the DOM here, `querySelector()` and `querySelectorAll()`. You can actually select any element you might need with these. There are lots more, which you'll encounter in the next chapter, along with many of the ways the DOM can be manipulated.

Practice exercise 9.3

Select a page element and update the content, change the style, and add attributes. Create an HTML file containing a page element with a class of `output` using the following code template:

```
<!DOCTYPE html >
<html>

<div class="output"></div>
    <script>
    </script>

</html>
```

Within the `script` tags, make the following changes to the output element:

1. Select the page element as a JavaScript object.
2. Update the `textContent` property of the selected page element.
3. Using the `classList.add` object method, add a class of `red` to the element.
4. Update the `id` property of the element to `tester`.
5. Through the `style` object, add a `backgroundColor` property of `red` to the page element.

6. Get the document URL via `document.URL` and update the text of the output element to contain the value of the document URL. You can log it in the console first to ensure you have the correct value.

Chapter project

Manipulating HTML elements with JavaScript

Take the HTML code below:

```
<div class="output">
    <h1>Hello</h1>
    <div>Test</div>
    <ul>
        <li id="one">One</li>
        <li class="red">Two</li>
    </ul>
    <div>Test</div>
</div>
```

Take the following steps (and experiment further) to understand how HTML elements can be manipulated with JavaScript code.

1. Select the element with the class `output`.
2. Create another JavaScript object called `mainList` and select only the `ul` tag that is within the `output` element. Update the ID of that `ul` tag to `mainList`.
3. Search for the `tagName` of each `div`, and output them into the console as an array.
4. Using a `for` loop, set the ID of each of the `div` tags to `id` with a numeric value of the order they appear within output. Still within the loop, alternate the color of the contents of each element in `output` to be red or blue.

Self-check quiz

1. Go to your favorite website and open the browser console. Type `document.body`. What do you see in the console?
2. As we know, with objects, we can write to the property value and assign a new value with the assignment operator. Update the `textContent` property of the `document.body` object on a web page of your choosing to contain the string `Hello World`.

3. Use what we learned about objects to list out BOM object properties and values. Try it on the `document` object.
4. Now do the same for the `window` object.
5. Create an HTML file with an `h1` tag. Use JavaScript and select the page element with the `h1` tag and assign the element into a variable. Update the `textContent` property of the variable to `Hello World`.

Summary

We started this chapter with the basics of HTML. We learned that HTML consists of elements and that these elements can contain other elements. Elements have a tag that specifies the type of element they are and they can have attributes that alter the element or add some metadata to the element. These attributes can be used by JavaScript.

We then had a look at the BOM, which represents the window of the browser that is being used for the web page and contains other objects, such as the `history`, `location`, `navigator`, and `document` objects. The `document` object is referred to as the DOM, which is what you are most likely to be working with. The document contains the HTML elements of the web page.

We also started to consider how we can select document elements and use these to manipulate the web page. This is what we'll continue exploring in the next chapter!