

Chapter 9 The DOM Object Model

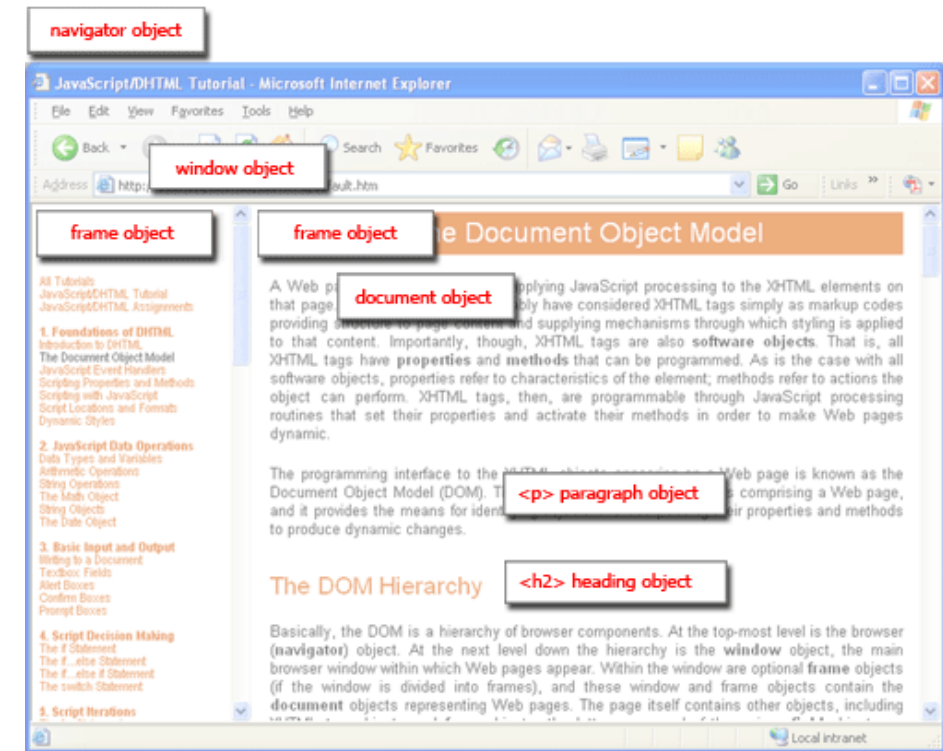
Agenda

- Browser Object Model (BOM)
 - window, history, location, navigator objects
- Document Object Model (DOM)
- Selecting page elements
- Navigating the DOM

Browser Object Model (BOM)

BOM is a hierarchy of objects that represent the browser window and its components.

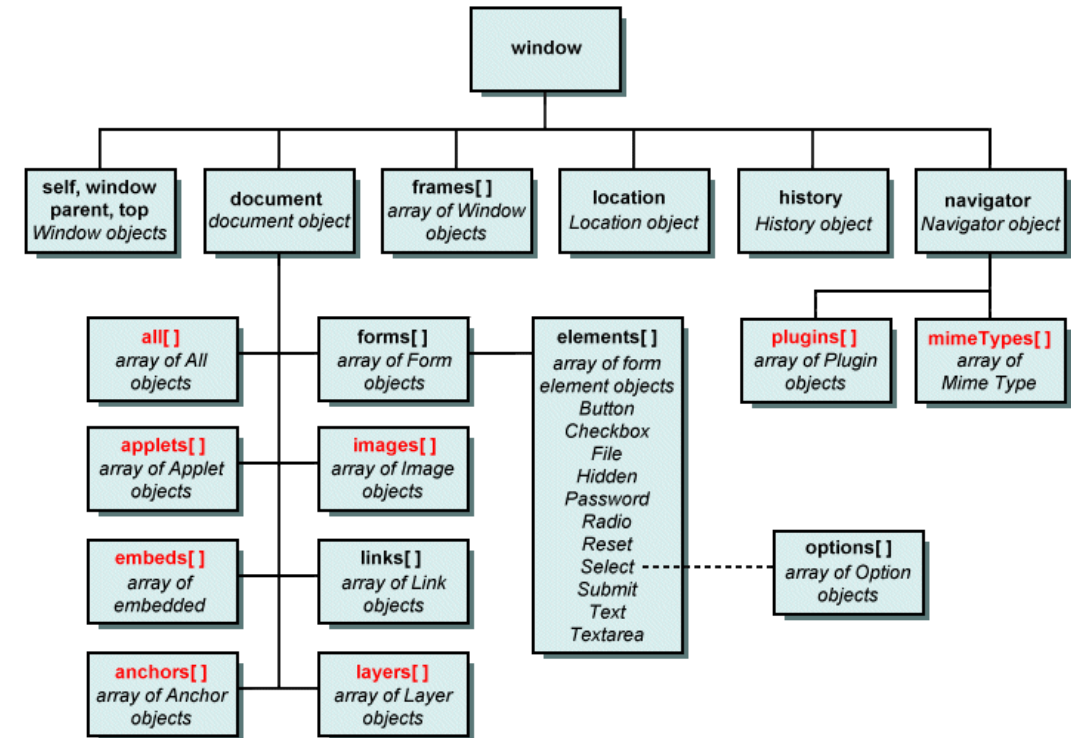
- provides the means to interact with the browser.
- The top level object in the BOM is the **window** object.



The hierarchy of the Browser Object Model (BOM)

Some of the important second-level objects in the BOM are:

- Window: the top-level object
- Document: the HTML document loaded in the browser window
- History: allowing to manipulate the browser session history
- Location: reading and manipulating the URL on the browser's address bar.
- Navigator: information about the browser



Window Object

The `window` object is the top-level object in the BOM.

- It represents the browser window or tab.

The `window` object is the global object and is the **default object** for JavaScript.

- It represents the window in which the script is running.
- All global variables declared with `var` and functions becomes properties and methods of the window object.
- However, this is not the case with `let` and `const` variables.

For example, the following two lines of code are equivalent:

```
alert('Hello, World!'); // same as window.alert('Hello, World!');  
window.alert('Hello, World!'); // same as alert('Hello, World!');
```

That is, if you do not specify an object to a property or a method, it is assumed that you use the one of the `window` object.

What can the `window` object do for us?

`window` object provide methods to interact with the browser window, including:

- opening, closing, and resizing the window
- scrolling viewport
- prompting, alerting, confirming with dialog boxes
- and others ...

See more about the `window` object at: [Window - Web APIs | MDN](#)

Example 9-1 : Write a script to open a new window with a specific size and location.

```
<script>
  // show a new window with the URL "https://example.com"
  let exampleWin = window.open("https://example.com", "_default", "resizable");
  // Resize the pop-up window to 500 pixels wide and 500 pixels high
  // and set the window title to "Example Window" and position it to the center of the screen.
  exampleWin.resizeTo(500, 500);
  // the window.screen object contains information about the user's screen
  exampleWin.moveTo((screen.width - 500) / 2, (screen.height - 500) / 2);
</script>
```

In the above code:

- The second argument of the `open()` method is the name of the browsing context (a tab, window, or `<iframe>`) the resource is being loaded into.

Syntax: `window.open()` method

```
open()  
open(url)  
open(url, target)  
open(url, target, windowFeatures)
```

See: [Window: open\(\) method - Web APIs | MDN](#)

Lab 09_01: Open a url to a iframe

with the following HTML code, enter the given code in the browser's console to see the result.

```
<body>  
  <iframe width="100%" height="500px" name="course_frame"></iframe>  
</body>
```

Command to enter in the console:

```
window.open("https://example.com", "course_frame");
```

History Object

- The `History` object provides an interface for manipulating the browser session history.
- Use `window.history` property to access the `History` object.
- Use the `back()` and `forward()` methods to navigate through the history programmatically.

Example 9-2: Navigate back to the previous page by calling the `back()` method in the console.

Open a page and navigate to another page. Then, open the browser console and type the following code to navigate back to the previous page.

```
window.history.back();  
// or simply  
history.back(); // the default object is window
```


Location Object

- The `Location` object represents the URL of the current page.
- Use `window.location` property or `location` to get the `Location` object.
- The `Location` object provides properties to get and set the parts of the URL.
 - e.g. `location.href`: the full URL of the current page.
 - e.g. `location.hostname`: the domain name of the server.

Location Object cheat sheet for properties and methods

Location object's methods:




- `assign(url)` : Navigate to the specified URL.
- `reload()` : Reloads the current page.
- `replace(url)` : Replaces the current document with a new one.
 - redirects to the provided URL; the replaced page is not saved in the session history.
- `toString()` : Returns the full URL of the current page.



 **window.location CHEATSHEET**

PROPERTIES

protocol	hostname	pathname	search	hash
https:	//	www.samanthaming.com	/tidbits/	?filter=JS #2

METHODS

assign  history	replace 	reload 	toString "url"
---	---	--	--------------------------

 samanthaming samanthaming.com  samantha_ming

Lab 09_02: Assign a new URL to the current page

Open the browser console and type the following code to assign a new URL to the current page.

```
location.assign("https://example.com");
```

What happens when you run the above code?

Navigator Object

- The `Navigator` object represents the state and the identity of the user agent.
- Use `window.navigator` property to access the `Navigator` object.

Typical scenarios to use the `Navigator` object include:

- Detecting the browser type and version by checking the `navigator.userAgent` property.
- Get the geographical location of the user by using the `navigator.geolocation` property.

Example 9-3: Display the user agent string in the console.

Open the browser console and type the following code to display the user agent string.

```
console.log(navigator.userAgent);
```

If you are interested, go to [Window: navigator property - Web APIs | MDN](#) to see how to detect the user's browser.

Lab 09_03: Print the keys and values of the `navigator` object

Open the browser console and type the following code to print the keys and values of the `navigator` object.

```
for (let key in window.navigator){  
    console.log(`${key}: ${window.navigator[key]}`)  
}
```

Next, we will discuss the Document Object Model (DOM).
The DOM allows us to interact with the HTML document by using JavaScript.

Document Object Model (DOM)

- The `Document` object represents the HTML document loaded in the browser window.
- Whenever we modify the DOM, the browser re-renders the web page.
 - e.g. adding a new element to the DOM will cause the browser to display the new element on the web page, or
 - applying a new CSS style to an element will cause the browser to update the element's appearance.
- Use `window.document` property to access the `Document` object.

Structure of the DOM:

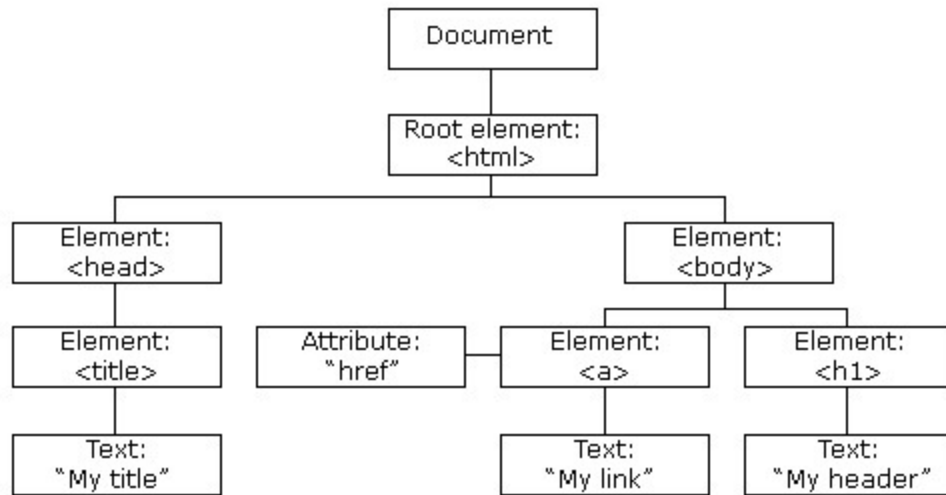
- The DOM is a tree-like structure that represents the HTML document.
- The DOM comprises NODE objects.
- The `node.nodeType` property indicates the type of the node [2].
- Common node types:
 - document node (nodeType = 9)
 - element node (nodeType = 1)
 - attribute node (nodeType = 2) of an element
 - text node (nodeType = 3) of an element
 - comment node (nodeType = 8)

Representing a HTML document via the DOM:

Consider the following HTML document:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My title</title>
</head>
<body>
  <a href="">My link</a>
  <h1>My Header</h1>
</body>
</html>
```

The DOM representation of the above HTML document is as follows:



If you are interested, go to [Live DOM viewer \[11\]](#) to generate a DOM tree for your HTML document.

Selecting page elements

We must get the references to the elements in the DOM before we can manipulate them.

The `document` object provides several `getElement...` methods to select elements in the DOM.

You can get one or more elements by `id` attribute, `name` attribute, `class name`, `tag name` or CSS `selector`.

The list of methods to select elements in the DOM:

Select **(one)** by id:

- `document.getElementById(your_id)` : Return the found HTML element (Element object) with the specified id.

Select **all** by name:

- `document.getElementsByName("your_name")` : Return the set of elements that have the specified name attribute in an array-like object (NodeList).

Select **all** by css class name:

- `document.getElementsByClassName("your_class_name")` : Return the set of elements that meet the specified class name in an array-like object (HTMLCollection).

Select **all** by tag name:

- `document.getElementsByTagName("your_tag_name")` : Return the set of elements that meet the specified tag name in an array-like object (HTMLCollection).

Select **first** by CSS selector:

- `document.querySelector("your_css_selector")` : Return the first element (Element object) that meets the specified CSS selector.

Select **all** by CSS selector:

- `document.querySelectorAll("your_css_selector")` : Return all elements that meet the specified CSS selector in an array-like object (NodeList).

Consider the following HTML document:

```
<body>
  <h1>Header 1</h1>
  <!-- Ice Cream flavors, multiple input boxes -->
  <input type="checkbox" name="flavor" id="flavor1" value="Vanilla">
  <label for="flavor1">Vanilla</label>
  <input type="checkbox" name="flavor" id="flavor2" value="Chocolate">
  <label for="flavor2">Chocolate</label>
  <input type="checkbox" name="flavor" id="flavor3" value="Strawberry">
  <label for="flavor3">Strawberry</label>
  <input type="checkbox" name="flavor" id="flavor4" value="Mint">
  <label for="flavor4">Mint</label>
  <h1>Header 2</h1>
  <h1>Header 3</h1>
</body>
```

1. Select all elements with the name "flavor"
2. Select the first element with the name "flavor"
3. Select all h1 elements

Lab 09_05: Numbering the h1 elements

Consider the following HTML document. We need to write a script to number the h1 elements from 1 to n in the order they appear in the document.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Numbering the h1 elements </title>
  </head>
  <body>
    <h1>Header 1</h1>
    <h1>Header 2</h1>
    <h1>Header 3</h1>
  </body>
  <script>
    //TBD
  </script>
</html>
```

1. Header 1

2. Header 2

3. Header 3

Steps to complete the lab:

1. Get all the h1 elements by using the `document.getElementsByTagName()` method.
2. Iterate through the HTMLCollection object to number the h1 elements.
 - use the for loop to iterate through the HTMLCollection object.
 - use the `innerHTML` property of the h1 element to set the content with the number.

Q: Element and Node objects?

- Element object inherits from Node object.
- Element object is the general base object for all HTML elements.
- Other specific html elements inherit from the Element object, such as HTMLDivElement, [HTMLAnchorElement](#), HTMLInputElement, etc.
- The inheritance hierarchy is as follows:
 - Node <- Element <- HTMLElement <- HTMLDivElement (or other specific html elements)

Example: 9-4: Inheritance hierarchy of the HTMLAnchorElement object.

HTMLAnchorElement



Baseline Widely available



The **HTMLAnchorElement** interface represents hyperlink elements and provides special properties and methods (beyond those of the regular **HTMLElement** object interface that they inherit from) for manipulating the layout and presentation of such elements. This interface corresponds to `<a>` element; not to be confused with `<link>`, which is represented by **HTMLLinkElement**.



Example 9-5: Show the favorite colors.

Complete the `showColors()` function to display the selected favorite colors in the paragraph with the id "display".

```
<!DOCTYPE html>
<html>
  <head>
    <title> Show the favorite colors </title>
  </head>
  <body>
    <!-- Checkbox group showing a list of colors -->
    <fieldset>
      <legend>Select one or more favorite colors</legend>
      <input type="checkbox" name="color" value="Red"/> <label for="Red">Red</label><br>
      <input type="checkbox" name="color" value="Yellow"/> <label for="Yellow">Yellow</label><br>
      <input type="checkbox" name="color" value="Green"/> <label for="Green">Green</label><br>
      <input type="checkbox" name="color" value="Blue"/> <label for="Blue">Blue</label><br>
    </fieldset>
    <button onclick="showColors()">Show the favorite colors</button>
    <p>Your favorite colors:</p>
    <p id="display" ></p>
  </body>
  <script>
    function showColors(){
      //TBD
    }
  </script>
</html>
```

Select one or more favorite colors

☒ Red

☒ Yellow

☐ Green

☐ Blue

Show the favorite colors

Your favorite colors:

Red

Yellow

The steps in the `showColors()` function are as follows:

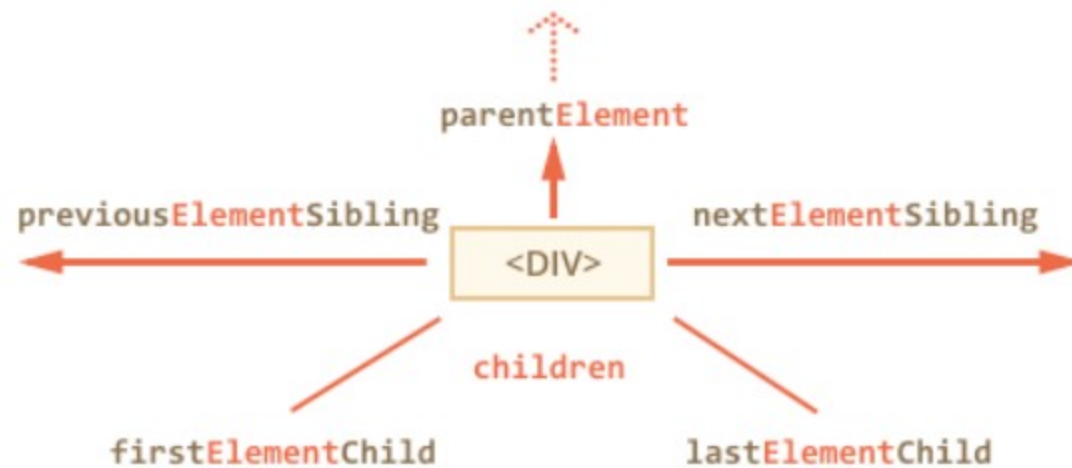
1. Select all the checkboxes with the name "color" by using the `document.getElementsByName()` method.
 - The method returns a `NodeList` object which is an array-like object.
 - The `NodeList` object contains all the checkboxes (`HtmlInputElement` objects) with the name "color".
2. Iterate through the `NodeList` object to get the values of the checked checkboxes.
 - the `checked` property of the `HtmlInputElement` object indicates whether the checkbox is checked.
 - the `value` property of the `HtmlInputElement` object returns the value of the checkbox.
3. Append the values of the checked checkboxes to the paragraph with the id "display".
 - Use the `innerHTML` property of the paragraph to set the content with html tags.

The resultant code is as follows:

```
function showColors(){
    // Get all the checkboxes
    let checkboxes = document.getElementsByName("color"); // NodeList<HtmlElement>
    // Get the display element and clear it
    const displayElement = document.getElementById("display");
    displayElement.innerHTML = "";
    // Iterate the checkboxes
    checkboxes.forEach(checkbox => {
        // If the checkbox is checked
        console.log(checkbox);
        if(checkbox.checked){
            // Display the value of the checkbox
            displayElement.innerHTML += checkbox.value + "<br>";
        }
    });
}
```

Navigating the DOM

Sometimes, we need to navigate the DOM to access the parent, child, or sibling elements of a specific element.



Use the following properties of the `Element` object to navigate the DOM:

Parents:

- `element.parentElement` : returns the parent element of the element.
- `element.children` : returns a `HTMLCollection` of child elements of the element.

Children:

- `element.firstElementChild` : returns the first child node of the element.
- `element.lastElementChild` : returns the last child node of the element.

Siblings:

- `element.previousElementSibling` : returns the previous sibling node of the element.
- `element.nextElementSibling` : returns the next sibling node of the element.



Example 9-6: Navigate the DOM and apply inline styles to elements

Implement the DOM Object Model

- Modify the above example.
- In addition to showing the selected favorite colors, apply the inline style to the texts of the selected favorite colors.
- Change the text of the selected checkbox's label to the color it represents.
 - For example, if the user selects the "Red" checkbox, the text of the label should be displayed in red color.

Select one or more favorite colors _____

☐ Red

☐ Yellow

☒ Green

☒ Blue

Show the favorite colors

Your favorite colors:

Green
Blue

The idea to come up with the solution is as follows:

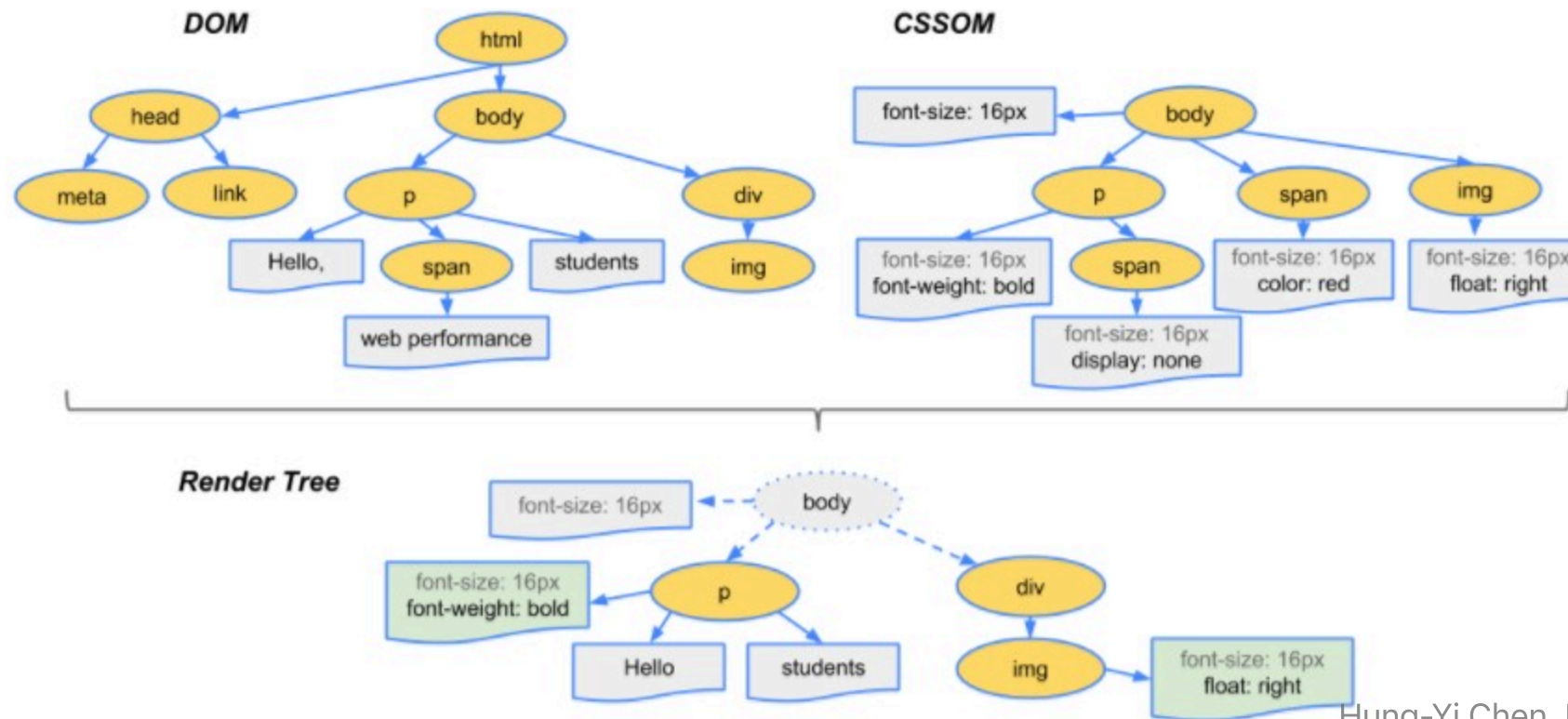
- While iterating through the checkboxes NodeList, we can access the label elements of the visited checkbox by using its `nextElementSibling` property.
- To change the text color of the label, we can set the `style.color` property of the label element to the value of the checkbox.
 - the `element.style` property returns the inline style of the element.
- For the non-selected checkboxes, we set the text color of its label to black.

The modified arrow function for the `forEach()` method is as follows:

```
checkboxes.forEach(checkbox => {  
    // If the checkbox is checked  
    console.log(checkbox);  
    if(checkbox.checked){  
        // Display the value of the checkbox  
        displayElement.innerHTML += checkbox.value + "<br>";  
        // Apply the inline style to the checkbox  
        const colorName = checkbox.value.toLowerCase();  
        const labelElement = checkbox.nextElementSibling;  
        labelElement.style.color = colorName;  
    } else {  
        // set the the text color of the label next to the checkbox to black  
        checkbox.nextElementSibling.style.color = "black";  
    }  
});
```

Behind the Scene: What happens when a web page is loaded?

1. The browser parses HTML content
2. then, build the DOM and CSS Object Model(CSSOM).
3. Finally, DOM and CSS Object Model(CSSOM) are merged to create the Render Tree.
4. The browser uses the Render Tree to render the web page [1]



S1. The browser requests and parses the HTML file from the server. Chapter 9 The DOM Object Model

S2. The browser requests resources, such as images, CSS files, and JavaScript files, from the server.

S3. The browser builds the following models by combining the HTML content and the resources:

- Document Object Model (DOM): A tree-like model based on the HTML content for the browser to render the web page.
- CSS Object Model (CSSOM): A tree-like model based on the CSS content for the browser to render the web page.
- Rendering Tree: A combination of the DOM and CSSOM. The browser uses the Rendering Tree to render the web page, including knowing the nodes to display in the DOM, the position and size of each node, and the computed style of each node.

S4. The browser renders the web page based on the Rendering Tree.

Refer to the following video to learn more: [The process to build the render tree \(Youtube\)](#)

Summary

- The Browser Object Model (BOM) provides objects to interact with the browser.
- The `window` object is the top-level object in the BOM.
- Other important objects in the BOM include the `History`, `Location`, and `Navigator` objects.
- The Document Object Model (DOM) represents the HTML document loaded in the browser window.
- The `document` object provides methods to select elements in the DOM.
- You can select elements by id, name, class name, tag name, or CSS selector.

References

- [1] <https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model?hl=zh-tw>
- [2] Node.nodeType, <https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeType>