JavaScript Essentials

DOM

Table of Contents

Lesson Objectives

- Understand the fundamentals concepts behind DOM
- Able to access DOM in JavaScript
- Understand the fundamentals data type of DOM
- Able to location DOM elements using selectors

Section 1

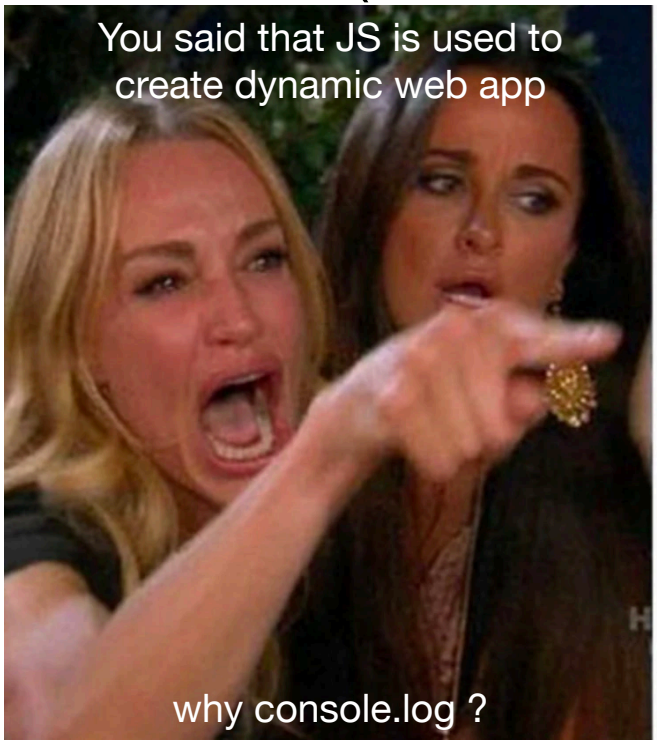
Overview

Overview

- What we have learned so far (about JavaScript) ?
  1. Variables
  2. Array/Object
  3. Conditionals
  4. Loops
  5. Function

Overview

- What we have learned so far (about JavaScript) ?



Overview

- What we have learned so far (about JavaScript) ?

Overview

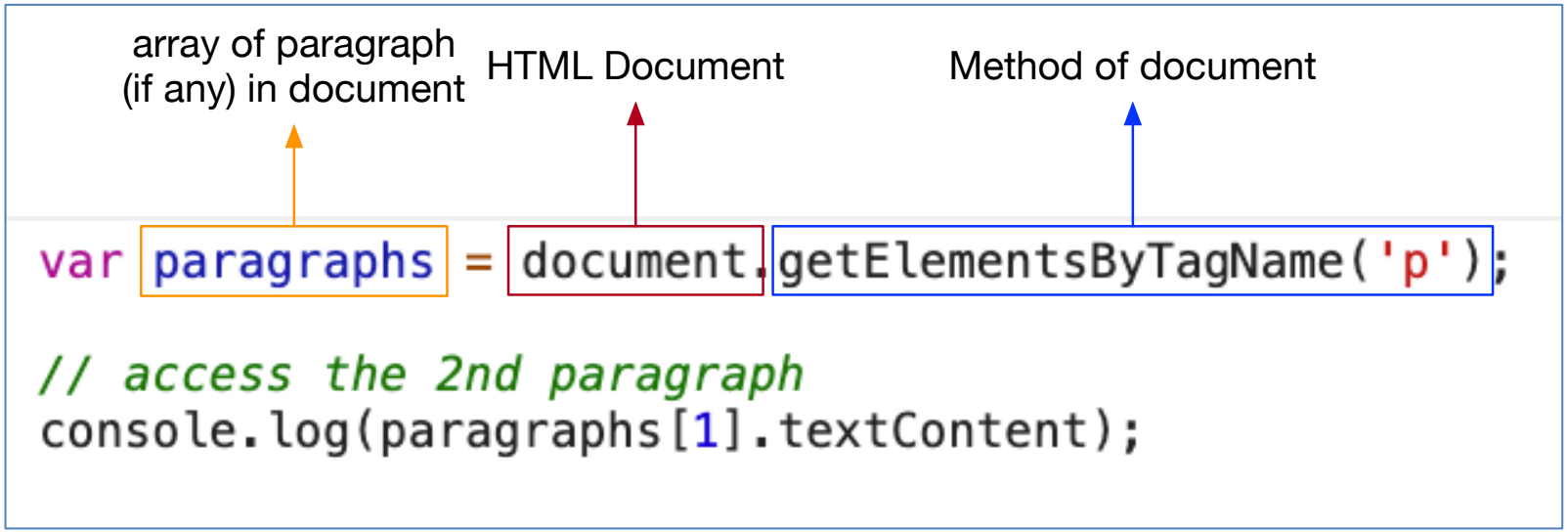
- The Document Object Model (DOM) is a programming interface for HTML and XML documents.
- It **represents** the page so that programs can change the document structure, style, and content.
- The DOM represents the document as nodes and objects. That way, JavaScript can connect to the page.

Overview

- The Document Object Model (DOM) represents that same document so it can be manipulated.
- The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.
- The W3C DOM and WHATWG DOM standards are implemented in most modern browsers.
- Many browsers extend the standard, so care must be exercised when using them on the web where documents may be accessed by various browsers with different DOMs.

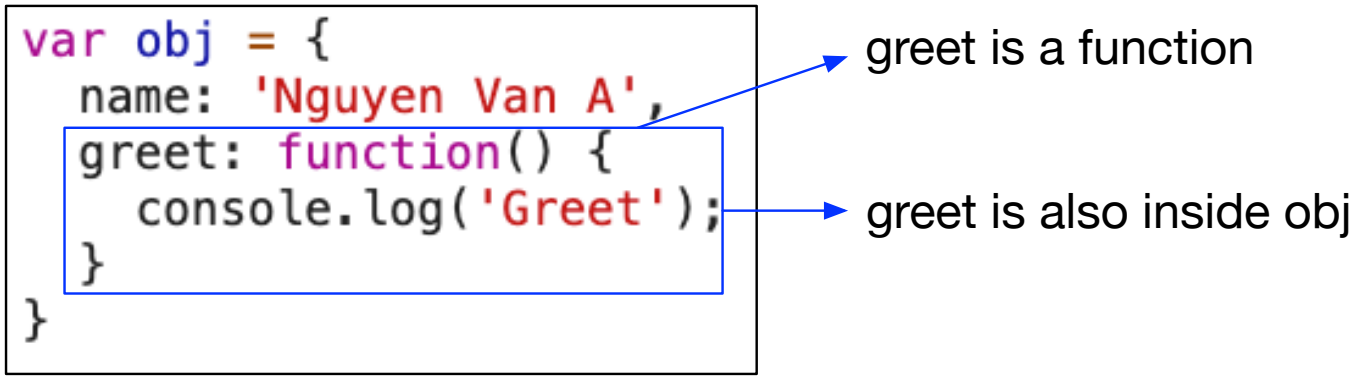
Overview

- For example, the standard DOM specifies that the getElementsByTagName method in the code below must return a list of all the <p> elements in the document:



Overview

- What is a method ?
  - It's another name of a function which **live** inside an object



Overview

- All of the properties, methods, and events available for manipulating and creating web pages are organized into objects
- This documentation provides an object-by-object reference to the DOM.
- The modern DOM is built using multiple APIs that work together.
- The core DOM defines the objects that fundamentally describe a document and the objects within it.

Overview – Summary

- The Document Object Model (DOM) is a programming interface for HTML and XML documents.
- It represents the page so that programs can change the document structure, style, and content.
- The DOM represents the document as nodes and objects.
- That way, programming languages (like JavaScript) can connect to the page.

Section 2

DOM and JavaScript

DOM and JavaScript

- In previous Demo, the code is ***written*** in JavaScript, but it ***uses*** the DOM to access the document and its elements.
- The DOM is not a programming language, but without it, the JavaScript language wouldn't have any **model** or **notion** of web pages
- HTML documents, XML documents, and their component parts (e.g. elements).
- Every element in a document can be accessed and manipulated using the DOM and a scripting language like JavaScript.

DOM and JavaScript

- In the beginning, JavaScript and the DOM were tightly intertwined, but eventually, they evolved into separate entities.
- The page content is stored in the DOM and may be accessed and manipulated via JavaScript, so that we may write this approximative equation:

Your Code = DOM + JavaScript

DOM and Programminng Language

- The DOM was designed to be **independent** of any particular programming language, making the structural representation of the document available from a single, consistent API.
- DOM can be **built** for any language, as this Python example demonstrates:

```
# Python DOM example
import xml.dom.minidom as m
doc = m.parse(r"C:\Projects\Py\chap1.xml")
doc.nodeName # DOM property of document object
p_list = doc.getElementsByTagName("para")
```

DOM and JavaScript - Overview

- The DOM is not a programming language, but without it, the JavaScript language wouldn't have any model or notion of web pages
- Every **element** in a document is part of the **document** object model for that document, so they can all be accessed and manipulated using the DOM and a scripting language like JavaScript.
- The **DOM** was designed to be independent of any particular programming language, making the structural representation of the document available from a single, consistent API

Section 3

Accessing the DOM

Accessing the DOM

- You don't have to do anything special to begin using the DOM.
- Different browsers have different implementations of the DOM, and these implementations exhibit varying degrees of conformance to the actual DOM standard
- But every web browser uses some document object model to make web pages accessible via JavaScript.

Accessing the DOM

- When you create a script, you can immediately begin using the API for the document or window elements to manipulate the document itself or to get at the children of that document
- Your DOM programming may be something as simple as the following, which displays an alert message by using the alert() function from the window object, or it may use more sophisticated DOM methods to actually create new content, as in the longer example below.

Accessing the DOM

- This following JavaScript will display an alert when the document is loaded (and when the whole DOM is available for use):

```
<body onload="window.alert('Welcome to my home page!');">
```

Accessing the DOM in <script>

- Beware when access DOM in script, you may have the following error:

```
▶Uncaught TypeError: Cannot read property 'textContent' of undefined
    at main.js:4
```

Accessing the DOM in <script>

- Because in HTML we load the <script> tag before HTML body

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="utf-8" />
5       <title>Unit 10 - Demo 1</title>
6       <script src="main.js"></script>
7     </head>
8     <body>
9       <p>First paragraph</p>
10      <p>Second paragraph</p>
11    </body>
12  </html>
```

At this line, browser load and run main.js

But at that time, the browser haven't loaded the body yet

Accessing the DOM in <script>

- To fix this problem:
  1. Make sure the script tag is before closing </body> tag
  2. Or use **defer** attribute on <script> tag
  3. Or wrap the code inside window.onload = function () {}

Accessing the DOM in <script>

- Fix 1: by add <script> tag before closing </body> tag

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="utf-8" />
5       <title>Unit 10 - Demo 1</title>
6     </head>
7     <body>
8       <p>First paragraph</p>
9       <p>Second paragraph</p>
10      <script src="main.js"></script>
11    </body>
12  </html>
```

Accessing the DOM in <script>

- Fix 2: by add **defer** attribute to <script> tag (**Recommended**)

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="utf-8" />
5       <title>Unit 10 - Demo 1</title>
6       <script src="main.js" defer></script>
7     </head>
8     <body>
9       <p>First paragraph</p>
10      <p>Second paragraph</p>
11    </body>
12  </html>
```

Accessing the DOM in <script>

- Fix 3: wrap the code inside window.load = function () {}

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Unit 10 - Demo 1</title>
    <script src="main3.js"></script>
  </head>
  <body>
    <p>First paragraph</p>
    <p>Second paragraph</p>
  </body>
</html>
```

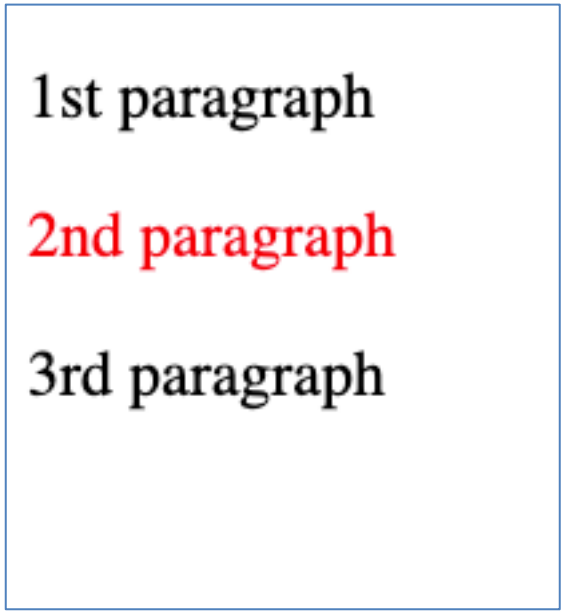
main3.js

```js
window.onload = function () {
  // code only run when document is loadded
  var paragraphs = document.getElementsByTagName('p');

  // access the 2nd paragraph
  console.log(paragraphs[1].textContent);
};
```

Demo 3

- Add paragraph to document using DOM and JavaScript:



Accessing DOM – Practice 1

Output

Ouput HTML (added with DOM)

```
▼<body>
  ▼<main>
      <h1 class="text-red">Hello from FA</h1>
    </main>
```

Accessing window

- window represents the currently opened window (or browser Tab)
- Useful properties/method:
  1. Location: window.location
  2. Scroll: window.scrollTo
  3. Browser: window.navigatior

Accessing window - Demo

- Demo accessing window object
  1. Log out current URL
  2. Navigate to another URL
  3. Scroll using JS
  4. Scroll using JS but smoth scroll

Accessing the DOM - Overview

- You don’t have to declare anything, the DOM is provided to you by the Browser
- Inside <script> tag or JavaScript file, you have access to document or window object
- document represents the HTML document. Use this when you want to manipulate the HTML page
- window represents the currently opened window (or browser Tab). Use this when you want to control the Browser behavior such as go to another url

Section 4

Fundamental data types

Fundamental data types

- This reference tries to describe the various objects and types in simple terms.
- But there are a number of different data types being passed around the API that you should be aware of.
- **Note:** tt's common to refer to the nodes in the DOM as **Elements**, although strictly speaking not every node is an element.

Fundamental data types

Data type (Interface)	Description
Document	When a member returns an object of type document (e.g., the ownerDocument property of an element returns the document to which it belongs), this object is the root document object itself. The DOM document Reference chapter describes the document object.
Node	Every object located within a document is a node of some kind. In an HTML document, an object can be an element node but also a text node or attribute node.
Element	The element type is based on node. It refers to an element or a node of type element returned by a member of the DOM API. Rather than saying, for example, that the document.createElement() method returns an object reference to a node

Fundamental data types

Data type (Interface)	Description
NodeList	•A nodeList is an array of elements, like the kind that is returned by the method document.getElementsByTagName(). Items in a nodeList are accessed by index in either of two ways: •list.item(1) •list[1] These two are equivalent. In the first, item() is the single method on the nodeList object. The latter uses the typical array syntax to fetch the second item in the list.

Fundamental data types - Summary

- Documen, Node and Element are three most important Data Type
- There are also some common terminology considerations to keep in mind.
- It's common to refer to any Attribute node as simply an attribute, for example, and to refer to an array of DOM nodes as a nodeList.
- You'll find these terms and others to be introduced and used throughout the documentation

Section 5

Locating DOM elements using selectors

Locating DOM elements using selectors

- The Selectors API provides methods that make it quick and easy to retrieve Element nodes from the DOM by matching against a set of selectors.
- This is much faster than past techniques, wherein it was necessary to, for example, use a loop in JavaScript code to locate the specific items you needed to find.

Locating DOM elements using selectors

- querySelector(): Returns the first matching Element node within the node's subtree. If no matching node is found, null is returned.
- querySelectorAll(): Returns a NodeList containing all matching Element nodes within the node's subtree, or an empty NodeList if no matches are found.
- **Note:** The NodeList returned by querySelectorAll() is not live, which means that changes in the DOM are not reflected in the collection. This is different from other DOM querying methods that return live node lists.

Locating DOM elements using selectors

- The selector methods accept one or more comma-separated selectors to determine what element or elements should be returned.
- For example, to select all paragraph (p) elements in a document whose CSS class is **either** warning or note, you can do the following:

```
var special = document.querySelectorAll( "p.warning, p.note" );
```

Locating DOM elements using selectors

- You can also query by ID. For example:

```
var el = document.querySelector( "#main, #basic, #exclamation" );
```

- After executing the above code, el contains the first element in the document whose ID is one of main, basic, or exclamation.
- You may use any CSS selectors with the **querySelector**() and **querySelectorAll**() methods.

Locating DOM elements using selectors

- Once you have selected an Element, you can do the following to it:
  1. Change its content with (textContent or innerHTML)
  2. Change its style with (el.style)
  3. Change its class with (el.classList.add or el.classList.remove)
  4. Add Event listener (Unit 11)
- Check Demo 5

Practice Selectors API

1. Use querySelector() and universal selector to selects the first element in the document.body
2. Selects all h2 elements and add background-color: #ccc to them
3. Selects first element with class .menu-item and add class .active to it
4. Selects the item with id #logo and change its src to another image URL (use Google image to get another image)
5. Finds all li elements that are directly inside a <ul> element and change the 1st li text to 'Test'

Selectors API - Overview

- Use any CSS selectors with the querySelector() and querySelectorAll() methods to locating the desired Element
- Once you selected an Element, you can modify its text content, change its appearance or even append new Element

Thank you

Q&A