Batch: FSD61WD-T

Name: Duraimurugan H

Document and Window Objects in JavaScript

In the realm of web development, JavaScript interacts extensively with two fundamental objects: **Document** and **Window**. These objects are crucial for manipulating the content and behavior of web pages dynamically. Let's delve into the differences between these two objects to grasp their roles and functionalities.

1. Window Object: The `window` object represents the browser window that contains the web document. It is the top-level object in the hierarchy of the browser's JavaScript objects, and it acts as the global scope for any JavaScript code running within that window.

- **Global Scope:** All global JavaScript variables, functions, and objects are members of the `window` object. For example, a globally declared variable `var x = 10;` is implicitly `window.x`.

- **Properties and Methods:** The `window` object provides access to various properties and methods that control the browser window. These include `window.document` (which refers to the DOM document object), `window.location` (information about the current URL), `window.alert()` (displays an alert box), and many more.

- **Lifecycle:** The `window` object persists throughout the lifetime of the browser window/tab. It encapsulates the entire browsing context and its properties can be accessed and modified across different frames and iframes within the same window.

2. Document Object: The `document` object represents the HTML document that is currently loaded in the browser window. It is a child object of the `window` object and provides methods and properties to access and manipulate the content of the document.

- **DOM Manipulation:** The `document` object is primarily used to access and manipulate the elements and content of the HTML document. You can select elements, create new ones, modify attributes, and respond to events using methods like `getElementById()`, `getElementsByTagName()`, `createElement()`, etc.

- **Structure:** It represents the entire HTML structure of the web page, including the `<html>`, `<head>`, and `<body>` elements. The `document` object serves as an interface to interact with these elements and their content.

- **Events:** The `document` object also handles events related to the document, such as `DOMContentLoaded` (fired when the initial HTML document has been completely loaded and parsed), `click` events, form submissions, etc.

Key Differences:

Scope: While the `window` object encompasses the entire browser window and its global context, the `document` object is specific to the currently loaded HTML document within that window.

Purpose: The `window` object manages the browser window itself and provides global functionality, whereas the `document` object manages the content and structure of the HTML document within that window.

Hierarchy: The `window` object is the top-level object, whereas the `document` object is nested within it, reflecting the relationship between the browser window and its content.

In conclusion, understanding the distinctions between the `window` and `document` objects is crucial for effective web development. Utilizing their properties and methods correctly empowers developers to create dynamic and interactive web pages that respond to user actions and events seamlessly.

[Day-1 Q] Objects and its internal representation in Javascript

Certainly! Objects in JavaScript are fundamental to its structure and are widely used for organizing and manipulating data. Understanding how objects are internally represented and how they work is crucial for effective JavaScript programming. Let's dive into the details:

Objects in JavaScript

In JavaScript, objects are collections of key-value pairs where keys are strings (or Symbols) and values can be of any data type, including other objects. Objects can be created using either object literals `{}', the `new Object()' syntax, or constructor functions.

Example:

```
const person = {
                    name: 'Alice',
                    age: 30,
                    address: {
                                city: 'New York',
                                country: 'USA'
                              }
                    };

// Using new Object()

const car = new Object();
car.make = 'Toyota';
car.model = 'Camry';
```

Internal Representation

1. Properties and Methods

JavaScript objects have properties (data members) and methods (functions). Properties can be accessed using dot notation (`object.property`) or bracket notation (`object['property']`).

```
var person={name: “Alice”, age:30, address: { city: “New York”, country: “USA” } };

console.log(person.name);  // Output: 'Alice'

console.log(person.address.city);  // Output: 'New York'


const propName = 'age';

console.log(person[propName]);  // Output: 30
```

2. Prototype Chain

Every JavaScript object has a prototype (except for the base object, where `Object.prototype` is null). The prototype is another object where the object inherits properties and methods from.

```
// Creating an object with a prototype

const student = Object.create(person);

student.name = 'Bob';

student.grade = 'A';

console.log(student.age);  // Inherited from person object
```

3. Object Creation

When an object is created in JavaScript, memory is allocated both for the object itself and for its properties and methods. Properties are stored as key-value pairs, and methods (functions) are stored similarly but with executable code.

4. Property Descriptors

Each property in a JavaScript object has a corresponding property descriptor that defines its characteristics, such as `writable`, `enumerable`, and `configurable`. These descriptors can be accessed and modified using `Object.getOwnPropertyDescriptor()` and related methods.

```
const descriptor = Object.getOwnPropertyDescriptor(person, 'name');

console.log(descriptor);
```

5. Memory Management

JavaScript engines manage memory allocation and deallocation for objects dynamically using techniques like garbage collection. When an object is no longer referenced, it becomes eligible for garbage collection to free up memory.

Best Practices:

Use Object Literals: For simple objects, use object literals (`{}`) as they are concise and easy to understand.

**Avoid `new Object()':** Prefer object literals or constructor functions over `new Object()` for object creation to maintain consistency and readability.

Prototype Inheritance: Understand prototype chaining and use it judiciously for sharing methods and properties between objects.

Conclusion:

Objects are central to JavaScript programming, offering a powerful way to structure and manage data. Understanding their internal representation, including properties, methods, prototype chains, and memory management, is crucial for efficient and effective JavaScript development. By mastering these concepts, developers can leverage JavaScript's flexibility to create robust and scalable applications.