

# Dot vs Bracket Notation

## Question - 1



What is the difference between dot notation and bracket notation when accessing properties of an object? Give the difference in tabular format



Here's a comparison of dot notation and bracket notation when accessing properties of an object in JavaScript:

| Aspect                  | Dot Notation   | Bracket Notation  |
|-------------------------|--|---|
| Syntax                  | <code>`object.property`</code>   | <code>`object["property"]`</code>   |
| Usage                   | Used for accessing properties with known, valid identifiers.   | Used for accessing properties with variable names, special characters, or spaces.             |
| Property Naming         | Property names must be valid identifiers (e.g., no spaces or special characters except <code>`_`</code> ). | Property names can be any string, including spaces, special characters, or numbers.           |
| Dynamic Access          | Not suitable for dynamic property names.   | Suitable for dynamic property names using variables (e.g., <code>`object[variable]`</code> ). |
| Code Readability        | More readable and concise for static properties.   | More flexible for dynamic properties but less readable for static properties.                 |
| Performance             | Slightly faster as it's more straightforward for the interpreter.  | Slightly slower due to the need for string evaluation but generally negligible.               |
| Use with Reserved Words | Cannot use reserved words as property names.   | Can use reserved words as property names.   |

| Aspect                 | Dot Notation  | Bracket Notation  |
|------------------------|---|---|
| Example Usage          | <code>`person.name`</code>  | <code>`person["name"]`</code>   |
| Example with Variables | Not applicable (e.g., <code>`person.propName`</code> is treated literally). | <code>`person[propName]`</code> where <code>`propName`</code> is a variable containing the property name. |
| Special Characters     | Cannot use property names with spaces or special characters.                | Allows property names with spaces or special characters (e.g., <code>`object["first name"]`</code> ).     |

## Examples

- Dot Notation

```
javascript
```

```
let person = {
  name: "Alice",
  age: 30
};

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

- Bracket Notation

```
javascript
```

```
let person = {
  "first name": "Alice",
  age: 30
};

let propName = "age";
```

```
console.log(person["first name"]); // Output: Alice  
console.log(person[propName]); // Output: 30
```

## When to Use Each Notation

- **Dot Notation** is preferred when the property name is a valid identifier and known at coding time, as it is more readable and concise.
- **Bracket Notation** is useful when dealing with dynamic property names, property names stored in variables, or property names that include characters not allowed in dot notation, such as spaces or special characters.

ChatGPT can make mistakes. Check important info.

# Object Literal Overview

## Question - 2



What is an object literal in JavaScript? Explain in simple words.



An object literal in JavaScript is a way to create an object using a simple and concise syntax. It allows you to define an object and its properties in one step, without having to explicitly use a constructor or a separate object creation process.

### Basic Syntax

Here's how an object literal looks:

javascript

```
let person = {  
  name: "John",  
  age: 30,  
  isStudent: false  
};
```

### Explanation

- **Curly Braces `{}`**: The object literal is enclosed in curly braces.
- **Key-Value Pairs**: Inside the braces, you define properties using key-value pairs. Each key is a property name, and each value is the property value.
  - **Keys**: The keys are usually strings or identifiers (e.g., `name`, `age`).

- **Values:** The values can be any valid JavaScript expression, such as a string, number, boolean, array, function, or even another object.
- **Commas:** Each key-value pair is separated by a comma.

## Example

Here's a simple example of an object literal representing a car:

javascript

```
let car = {  
  brand: "Toyota",  
  model: "Camry",  
  year: 2020,  
  startEngine: function() {  
    console.log("Engine started");  
  }  
};
```

## Accessing Object Properties

You can access the properties of an object using dot notation or bracket notation:

- **Dot Notation:**

javascript

```
console.log(car.brand); // Outputs: Toyota
```

- **Bracket Notation:**

```
javascript
```

```
console.log(car["model"]); // Outputs: Camry
```

## Benefits of Using Object Literals

- **Easy to Read and Write:** Object literals provide a clear and concise way to create objects.
- **Flexible:** You can define complex structures, including nested objects and functions.
- **Dynamic:** You can add, remove, or update properties at any time.

## Example with Nested Object

```
javascript
```

```
let book = {  
  title: "The Great Gatsby",  
  author: "F. Scott Fitzgerald",  
  details: {  
    publisher: "Scribner",  
    year: 1925  
  }  
};  
  
console.log(book.details.publisher); // Outputs: Scribner
```

In summary, object literals are a straightforward and powerful way to define and manipulate objects in JavaScript, making them a fundamental part of the language for handling structured data.

ChatGPT can make mistakes. Check important info.

# Cloning Objects in JavaScript

## Question - 3



How would you clone an object in JavaScript? Explain in Simple Words.



Cloning an object in JavaScript means creating a new object with the same properties and values as an existing object. There are different ways to do this, depending on whether you need a **shallow copy** or a **deep copy**.

### Shallow Copy

A **shallow copy** creates a new object, but it only copies the references to nested objects, not the actual nested objects themselves. This means that if you change a nested object in the original, it will also change in the copy, because they are still pointing to the same place in memory.

#### Ways to Make a Shallow Copy:

##### 1. Using Object Spread Operator:

javascript

```
const original = { a: 1, b: 2 };  
const shallowCopy = { ...original };  
  
console.log(shallowCopy); // Output: { a: 1, b: 2 }
```

##### 2. Using Object.assign():

javascript

```
const original = { a: 1, b: 2 };
const shallowCopy = Object.assign({}, original);

console.log(shallowCopy); // Output: { a: 1, b: 2 }
```

## Deep Copy

A **deep copy** means copying all the properties and values, including nested objects and arrays, so that the new object is completely independent of the original. Changes to the original won't affect the copy and vice versa.

### Ways to Make a Deep Copy:

#### 1. Using `JSON.stringify()` and `JSON.parse()`:

This method works well for objects that only contain serializable data (i.e., no functions or special object types like Date).

javascript

```
const original = { a: 1, b: { c: 2 } };
const deepCopy = JSON.parse(JSON.stringify(original));

console.log(deepCopy); // Output: { a: 1, b: { c: 2 } }
```



### 3. Using Structured Clone (for modern browsers):

The `structuredClone` method is a modern, built-in way to make deep copies of objects, supporting more data types than `JSON.stringify/parse`.

javascript

```
const original = { a: 1, b: { c: 2 }, d: new Date() };
const deepCopy = structuredClone(original);

console.log(deepCopy); // Output: { a: 1, b: { c: 2 }, d: Date object }
```

## Choosing the Right Method

- **Shallow Copy:** Use it when the object doesn't contain nested objects or if you don't mind the nested objects being shared between the original and the copy.
- **Deep Copy:** Use it when you need a completely independent copy of an object, including all nested objects and arrays.

Knowing which method to use depends on the complexity of your object and the level of independence you need between the original and the copy.

# Understanding JSON Format

## Question - 4



What is a JSON? Explain in Simple Words.



JSON, which stands for **JavaScript Object Notation**, is a lightweight data format used to store and exchange information. It is easy for humans to read and write and easy for machines to parse and generate. JSON is commonly used in web applications to send data between a server and a client (like a web browser).

### Key Features of JSON:

1. **Human-Readable:** JSON is text-based, which makes it easy to read and understand. It looks a bit like how you might write data in a plain text file.
2. **Structured Data:** JSON represents data in a structured way using key-value pairs. It can also include lists (arrays) and more complex data structures.
3. **Language-Independent:** Although JSON was derived from JavaScript, it is language-independent. It is supported by most programming languages, including Python, Java, C#, and more.
4. **Simple Syntax:**
  - **Objects:** Curly braces `{}` are used to define objects. Objects contain key-value pairs.
  - **Arrays:** Square brackets `[]` are used to define arrays. Arrays contain ordered lists of values.
  - **Key-Value Pairs:** Keys are strings, and they are followed by a colon `:` and the value.

## JSON Example:

Here's a simple example of a JSON object:

```
json
```

```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "courses": ["Math", "Science", "History"],
  "address": {
    "street": "123 Main St",
    "city": "New York",
    "zip": "10001"
  }
}
```

## Explanation of the Example:

- **Object:** The entire data structure is a JSON object (enclosed in `{}`).
- **Key-Value Pairs:**
  - `"name": "John Doe"` - The key is `"name"`, and the value is `"John Doe"`.
  - `"age": 30` - The key is `"age"`, and the value is `30`.
  - `"isStudent": false` - The key is `"isStudent"`, and the value is `false` (a boolean).
- **Array:** `"courses": ["Math", "Science", "History"]` - The key is `"courses"`, and the value is an array of strings.
- **Nested Object:**
  - `"address": {...}` - The key is `"address"`, and the value is another JSON object containing more key-value pairs for the address details.

## Use Cases of JSON:

- **Data Exchange:** JSON is commonly used in APIs (Application Programming Interfaces) to exchange data between a client and a server.
- **Configuration Files:** JSON is often used for configuration files because of its readability and simplicity.
- **Data Storage:** JSON can be used for storing simple data structures and is often used in databases like MongoDB.

## Conclusion:

JSON is a straightforward and flexible way to represent data. Its simplicity and ease of use make it a popular choice for data interchange in web development and other programming tasks.

ChatGPT can make mistakes. Check important info.

# Understanding Window Location

## Question - 5



What is the window.location object in JavaScript? Explain in Detail in simple words.



The `window.location` object in JavaScript is a part of the Web APIs and represents the URL (Uniform Resource Locator) of the current page loaded in the browser window. It is an interface that allows you to interact with and manipulate the current page's URL. Here's a detailed explanation of its properties and methods, along with how you can use them:

### Properties of `window.location`

#### 1. `window.location.href`

- **Description:** This is the complete URL of the current page. You can read it to get the current page's URL or set it to change the page location.
- **Example:**

```
javascript
```

```
console.log(window.location.href); // Outputs: "https://www.example.com/page.html"  
window.location.href = 'https://www.google.com'; // Redirects to Google
```

#### 2. `window.location.protocol`

- **Description:** This gives you the protocol used by the URL (e.g., `http:`, `https:`, `ftp:`).
- **Example:**

```
javascript
```

```
console.log(window.location.protocol); // Outputs: "https:"
```

### 3. `window.location.host`

- **Description:** This returns the host name and port number (if specified) of the URL.
- **Example:**

```
javascript
```

```
console.log(window.location.host); // Outputs: "www.example.com"
```

### 4. `window.location.hostname`

- **Description:** This returns only the domain name of the URL.
- **Example:**

```
javascript
```

```
console.log(window.location.hostname); // Outputs: "www.example.com"
```

### 5. `window.location.port`

- **Description:** This returns the port number used in the URL. If no port is specified, it returns an empty string.
- **Example:**

```
javascript
```

```
console.log(window.location.port); // Outputs: ""
```

#### 6. `window.location.pathname`

- **Description:** This gives the path of the URL, which is everything after the host and port.
- **Example:**

```
javascript
```

```
console.log(window.location.pathname); // Outputs: "/page.html"
```

#### 7. `window.location.search`

- **Description:** This returns the query string part of the URL, starting with the question mark `?`.
- **Example:**

```
javascript
```

```
console.log(window.location.search); // Outputs: "?id=123&name=John"
```

#### 8. `window.location.hash`

- **Description:** This returns the fragment identifier of the URL, which is the part after the `#` symbol.
- **Example:**



```
javascript
```

```
console.log(window.location.hash); // Outputs: "#section1"
```

#### 9. `window.location.origin`

- **Description:** This returns the protocol, host, and port of the URL.
- **Example:**

```
javascript
```

```
console.log(window.location.origin); // Outputs: "https://www.example.com"
```

## Methods of `window.location`

#### 1. `window.location.assign()`

- **Description:** This method loads a new document at the specified URL, essentially redirecting the browser to a new page.
- **Example:**

```
javascript
```

```
window.location.assign('https://www.google.com');
```

#### 2. `window.location.replace()`

- **Description:** Similar to `assign()`, but it doesn't create a new entry in the browser's history. This means you can't use the back button to go back to the previous page.

- **Example:**

```
javascript
```

```
window.location.replace('https://www.google.com');
```

### 3. `window.location.reload()`

- **Description:** This method reloads the current page. You can force a reload from the server by passing `true` as an argument.
- **Example:**

```
javascript
```

```
window.location.reload(); // Reloads the page  
window.location.reload(true); // Reloads from the server
```

### 4. `window.location.toString()`

- **Description:** Returns the full URL as a string, similar to `window.location.href`.
- **Example:**

```
javascript
```

```
console.log(window.location.toString()); // Outputs: "https://www.example.com/page.html"
```

## How to Use `window.location`

- **Getting the Current URL**

javascript

```
const currentUrl = window.location.href;  
console.log(currentUrl);
```

javascript

```
window.location.pathname = '/newpage.html';
```

## Changing the URL Path

### Summary

The `window.location` object is a powerful tool for interacting with the URL of a webpage. It allows developers to read and change various parts of the URL, reload the page, and redirect users to different pages. This makes it essential for tasks like navigation, parsing query parameters, and handling browser history effectively.

ChatGPT can make mistakes. Check important info.