INDEX

Advance Concepts

Session Storage & Local Storage

Session Storage in JavaScript is a client-side web storage mechanism that allows you to store key-value pairs during a single browsing session. Data stored in Session Storage is available only for the duration of that session and is cleared when the user closes the browser tab or window.

Here's how you can use Session Storage in JavaScript:

1. Storing Data:
```
// Store data in Session Storage
sessionStorage.setItem('key', 'value');
```

2. Retrieving Data:
```
// Retrieve data from Session Storage
const data = sessionStorage.getItem('key');
```

3. Removing Data:
```
// Remove a specific item from Session Storage
sessionStorage.removeItem('key');
```

4. Clearing Session Storage:
```
// Clear all data from Session Storage
sessionStorage.clear();
```

5. Checking for Item Existence:
```
// Check if a key exists in Session Storage
const exists = sessionStorage.getItem('key') !== null;
```

Remember that Session Storage is limited to the current browser session, so the data you store will be available only as long as the user's session is active. It's a useful tool for temporarily storing data between page transitions or interactions without the need for server-side storage.

Local Storage in JavaScript is a client-side web storage mechanism that allows you to store key-value pairs persistently in a user's web browser. Unlike Session Storage, which stores data for the duration of a single browsing session, data stored in Local Storage persists even after the browser is closed and can be used across multiple browser sessions. Here's how you can use Local Storage in JavaScript:

1. Storing Data:

```
// Store data in Local Storage
localStorage.setItem('key', 'value');
```

2. Retrieving Data:

```
// Retrieve data from Local Storage
const data = localStorage.getItem('key');
```

3. Removing Data:

```
// Remove a specific item from Local Storage
localStorage.removeItem('key');
```

4. Clearing Local Storage:

```
// Clear all data from Local Storage
localStorage.clear();
```

5. Checking for Item Existence:

```
// Check if a key exists in Local Storage
const exists = localStorage.getItem('key') !== null;
```

Local Storage is a powerful tool for persistently storing data in a user's browser, making it available even after they close the browser or navigate away from your website. However, it's important to be mindful of data security and privacy when using Local Storage, as data stored there is accessible to JavaScript on the same domain and can potentially be accessed by other scripts on the page.

Cookies:

Cookies are data, stored in small text files, on your computer.

Cookies are saved in name-value pairs like:
username = Jeet

JavaScript can create, read, and delete cookies with the document.cookie property.

cookie can be created like this:
document.cookie = "username=Jeet";

You can **also add an expiry date (in UTC time)**. By default, the cookie is deleted when the browser is closed:
document.cookie = "username=Jeet; expires=Wed, 13 Sep 2023 3:59:00 UTC";

With a **path parameter**, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.
document.cookie = "username=Jeet; expires=Wed, 13 Sep 2023 3:59:00 UTC; path=/";

cookies can be read like this:
let x = document.cookie;

document.cookie will return all cookies in one string much like: cookie1=value; cookie2=value; cookie3=value;

change a cookie the same way as you create it:
document.cookie = "username=jeet sorathiya";

Delete a Cookie
You don't have to specify a cookie value when you delete a cookie.
Just set the expires parameter to a past date:
document.cookie = "username=; expires=Wed, 13 Sep 2023 3:59:00 UTC; path=/;";

Some browsers will not let you delete a cookie if you don't specify the path.

Browser Debugging

Browser developer tools, commonly known as DevTools, are a set of integrated utilities available in modern web browsers that empower web developers and designers to inspect, debug, and optimize web pages and applications. These tools offer a wide range of features and functionalities, including:

1. Elements:

The Elements section of Chrome DevTools allows you to inspect individual elements on a web page. You can see the HTML, CSS, and JavaScript for each element, as well as its properties and attributes. You can also select elements and modify their properties and attributes.

To open the Elements section, open Chrome DevTools and click on the "Elements" tab. The Elements panel will be displayed, showing the DOM tree for the current web page.

The Elements section is a powerful tool for inspecting and debugging web pages. You can also use it to modify the appearance or behavior of a web page.

Here are some of the features of the Elements section:

- The DOM tree is a hierarchical representation of the HTML elements on the page. You can use the DOM tree to navigate to and select specific elements.
- Shows the HTML code for the selected element. You can use the HTML tab to edit the HTML code for the element.
- Shows the CSS rules that apply to the selected element. You can use the CSS tab to edit the CSS rules for the element.
- Shows the JavaScript code that is associated with the selected element. You can use the JavaScript tab to edit the JavaScript code for the element.
- Properties and attributes: The Properties and attributes pane shows the properties and attributes of the selected element. You can use the Properties and attributes pane to modify the properties and attributes of the element.

2. Console:

The "Console" tab is where you can log messages and run JavaScript code for debugging and testing in your web browser's developer tools.

Logging Messages:
- The primary function of the Console tab is to display messages generated by JavaScript code. Developers use various console methods to log messages:

Interactive Console:
- You can interactively execute JavaScript code directly within the console. Type expressions and statements, then press Enter to execute.
- The console displays the result of the executed code, such as values, objects, or error messages.

Grouping Messages:
- Group related log messages together for better organization using.

The "Console" tab is a fundamental tool for JavaScript debugging and diagnosing issues in web applications. It offers real-time feedback, allows for interaction with the runtime environment, and helps developers track the execution flow of their code. This makes it an essential part of a developer's toolkit when working on JavaScript projects.

3. Sources:

The "Sources" tab in browser developer tools is primarily used for debugging JavaScript code in web applications. It offers a range of features and tools to help developers inspect, navigate, and analyze the JavaScript source code of a webpage. Here's an overview of the key functionalities and components of the "Sources" tab:

File Navigation:
- The "Sources" tab displays the project files, including HTML, CSS, and JavaScript files, in a tree structure.
- You can expand and collapse folders to navigate through the file hierarchy.
- Clicking on a file opens it in the code editor.

Code Editor:

- The code editor provides syntax highlighting and code formatting for JavaScript files.
- It allows you to view and edit JavaScript code in a convenient, integrated environment.

Breakpoints:
- You can set breakpoints in your JavaScript code by clicking on line numbers in the code editor.
- Breakpoints pause the execution of the code when reached, allowing you to inspect variables and step through the code.

Call Stack:
- The "Call Stack" pane shows the sequence of function calls that led to the current execution point.
- It helps you understand the flow of code execution and the context of the current function.

Watch Expressions:
- You can add watch expressions to monitor specific variables or values while debugging.
- Watched expressions are evaluated and displayed in the "Watch" pane.

Scope Variables:
- The "Scope" pane allows you to inspect local and global variables within the current function's scope.
- It provides a detailed view of variables and their values.

Step Through Code:
- Use controls like "Step Over," "Step Into," and "Step Out" to navigate through the code line by line.
- These controls are helpful for debugging complex code execution paths.

Event Listener Breakpoints:
- Set breakpoints on specific JavaScript code.
- This helps you capture specific events that trigger JavaScript code.

The "Sources" tab is an essential tool for JavaScript debugging, helping developers identify and resolve issues in their code efficiently. It provides a comprehensive environment for inspecting, debugging, and understanding JavaScript behavior in web applications.

4. Network:

The "Network" tab in browser developer tools (DevTools) is a powerful feature for monitoring, analyzing, and debugging network activity within web applications. It provides insights into HTTP requests and responses, helping developers optimize performance, diagnose issues, and ensure that web pages load efficiently. Here's an overview of the key functionalities and components of the "Network" tab:

Network Requests:
- The "Network" tab displays a log of all network requests made by the web page, including HTTP requests for resources like HTML, CSS, JavaScript, images, and XHR/Fetch requests.
- Each request is listed with details such as the request method, URL, status code, and timing information.

Request Headers:
- You can inspect the headers of each network request to see details like user-agent, cookies, cache-control, and more.
- Request headers are essential for understanding how a request is configured.

Response Data:
- The tab also provides access to the response data for each request.
- You can view the response body, which includes HTML, JSON, text, or binary data.
- Response headers provide information about the server's response, including content type, content length, and cache control directives.

Timing Information:
- The "Timing" section of each request provides a breakdown of the various stages of request processing,connection setup, request, and response times.

Filtering and Sorting:
- You can filter network requests based on various criteria, such as request type (XHR, Fetch, JS, CSS, etc.) or text search.
- Sorting options allow you to order requests by time, size, or other attributes.

Request Details:
- Clicking on a specific network request opens a detailed view that provides additional information about the request and response, including headers, cookies, query parameters, and more.

Performance Analysis:

- The "Network" tab can help identify performance bottlenecks by showing a waterfall chart that visualizes the loading sequence of resources and their dependencies.

Throttling and Offline Mode:

- DevTools allows you to simulate various network conditions, such as slow 3G, fast 3G, or offline mode, to test how your application behaves under different network conditions.

Preserve Log Upon Navigation:

- By enabling the "Preserve log upon navigation" option, you can keep the network request log intact when navigating between pages, which can be useful for tracking network activity across multiple page loads.

The "Network" tab is an invaluable tool for web developers, allowing them to inspect and optimize the performance and behavior of web applications by monitoring network activity. It helps in diagnosing issues related to slow-loading resources, network errors, and inefficient resource utilization.

5. Performance:

The "Performance" tab in browser developer tools is a robust tool for analyzing and optimizing the performance of web applications. It provides insights into various aspects of page loading, rendering, and JavaScript execution. Here's an overview of the key functionalities and components of the "Performance" tab:

Recording Timeline:

- The "Performance" tab allows you to record a timeline of events that occur during a specified time frame, including page load and interactions.
- Recording captures a variety of performance data and events for analysis.

Overview and Summary:

- After recording, the "Performance" tab presents an overview summary that includes key metrics like load time, CPU usage, and frames per second (FPS).
- This summary provides a quick assessment of the web page's performance.

Frames View:

- In the "Frames" section, you can inspect how the page renders individual frames.
- Frame details include painting, rendering, and scripting activities.

The "Performance" tab is an essential tool for web developers aiming to optimize the speed and responsiveness of their web applications. It helps diagnose performance bottlenecks, monitor resource utilization, and identify areas for improvement, ultimately enhancing the user experience.

6. Memory:

The "Memory" tab in browser developer tools is a feature that allows developers to analyze and optimize the memory usage of web applications. It provides insights into JavaScript memory allocation, object retention, and potential memory leaks.

The "Memory" tab is a valuable tool for developers to diagnose and resolve memory-related issues in web applications. It aids in identifying memory leaks, optimizing memory usage, and improving the overall performance and stability of web pages.

7. Application:

The "Application" tab in browser developer tools (DevTools) is a feature-rich tool that provides insights into various aspects of a web application's storage and more. It is particularly useful for web developers working on web applications that rely on various client-side technologies. Here's an overview of the key functionalities and components of the "Application" tab:

Storage Management:

- The "Application" tab allows you to inspect and manage different client-side storage mechanisms, including:
- Cookies: View and manage HTTP cookies, including their values, domains, and expiration dates.
- Local Storage: Examine and edit data stored in the browser's local storage.
- Session Storage: Inspect and modify data stored in session storage.
- Cache Storage: Examine and manage data stored in the browser's cache, often used for service worker caching.

8. Security:

The "Security" tab in browser developer tools (DevTools) is a vital tool for understanding and securing web applications. It provides information and insights related to the security of a web page, including potential security vulnerabilities and issues.

9. Lighthouse:

The "Lighthouse" panel allows you to run Lighthouse audits for web page performance, accessibility, SEO, and best practices. Please note that there may have been updates or changes since then,

The Lighthouse tool, available within browser developer tools, is a valuable resource for web developers and site owners to assess and improve web page performance, accessibility, SEO, and adherence to best practices. It helps ensure that your web applications provide a better user experience and perform optimally.

What is OOJS?

OOJS, or Object-Oriented JavaScript, is an approach to programming in JavaScript that leverages object-oriented principles and concepts. JavaScript is a versatile programming language that supports various programming paradigms, including object-oriented programming (OOP). OOJS emphasizes the use of objects, classes, and inheritance to create modular, reusable, and maintainable code.

Class:

ECMAScript 2015 (ES6) introduced a built-in class keyword to define classes, making object-oriented programming more structured and readable in JavaScript.

- Use the keyword **class** to create a class.
- Always add a method named **constructor()**:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
  }
}

const person1 = new Person('Alice', 30);
person1.greet(); // Outputs: "Hello, my name is Alice and I am 30 years old."
```

The constructor method is a special method:

- It has to have the exact name "constructor"
- It is executed automatically when a new object is created
- It is used to initialize object properties

If you do not define a constructor method, JavaScript will add an empty constructor method.

Static:

Static class methods are defined on the class itself.

You cannot call a static method on an object, only on an object class.

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  static greet() {
    return "Hello from the Animal class!!";
  }
}

const myAnimal = new Animal("Lion");

// You can call 'greet()' on the Animal Class:
document.getElementById("demo").innerHTML = Animal.greet();

// But NOT on an Animal Object:
// document.getElementById("demo").innerHTML = myAnimal.greet();
// Attempting to call it on an object will raise an error.
```

ECMAScript6:

Difference between let, var & const:

var	**let**	**const**
The scope of a *var* variable is functional scope.	The scope of a *let* variable is block scope.	The scope of a *const* variable is block scope.
It can be updated and re-declared into the scope.	It can be updated but cannot be re-declared into the scope.	It cannot be updated or re-declared into the scope.
It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.
It can be accessed without initialization as its default value is “undefined”.	It cannot be accessed without initialization otherwise it will give ‘referenceError’.	It cannot be accessed without initialization, as it cannot be declared without initialization.

Arrow functions:

Arrow functions were introduced in ES6.

Arrow functions allow us to write shorter function syntax:

```
let myFunction = (a, b) => a * b;
```

Before Arrow:

```
hello = function() {
  return "Hello World!";
}
```

With Arrow Function:

```
hello = () => {
  return "Hello World!";
}
```

Import, Export, async, await Functions:

- **Import** : is used to bring in code from another file.
- **Export** : is used to make code available to other files.
- **Async** : is used to declare a function that can run asynchronous code.
- **Await** : is used to wait for the result of an asynchronous operation.

Here is an example of how to use these keywords:

```
// file1.js
export const getUsers = async () => {
  // This is an asynchronous function that returns a promise.
  const users = await fetch('/api/users');
  return users.json();
};

// file2.js
import { getUsers } from './file1';

const users = await getUsers();
console.log(users);
```

Difference between == & ===, != & !===

- == and != are loose equality operators. They compare the values of the operands, but they also perform type conversion. This means that they can return true even if the operands are of different types, as long as their values are equal. For example, the expression 1 == "1" will return true, because the values of the operands are equal, even though they are of different types.

- === and !== are strict equality operators. They compare the values of the operands without performing any type conversion. This means that they will only return true if the operands are of the same type and have the same value. For example, the expression 1 === "1" will return false, because the operands are of different types.