**Chapter 5**

**Advance Javascript**

|  |  |
| --- | --- |
| **5.1** | **Advance Concept** |
| **5.1.1** | **Session storage & localstorage** |
|  | Scope:  sessionStorage: Available only within the same tab or window.  localStorage: Available across tabs and windows of the same domain.  Duration:  sessionStorage: Data is cleared when the tab or window is closed.  localStorage: Data persists even after the browser is closed.  Capacity:  sessionStorage and localStorage: Typically around 5-10 MB per domain, depending on the browser.  Use Cases:  sessionStorage: Storing temporary data needed during a single session.  localStorage: Storing user preferences, cached data, or semi-permanent data. |
| **5.1.2** | **Basic of cookies** |
|  | Cookies: Available across tabs and windows of the same domain.  Cookies: Data persists until the cookie's expiration date or until cleared.  Cookies: Limited to about 4KB per domain.  Cookies: Maintaining sessions, storing user-specific data, and passing data to the server. |
| **5.1.3** | **Browser Debugging** |
| **5.1.3.1** | **Inspect element Window** |
| **5.1.3.2** | **Detail knowledge of diffrent tabs in inspect element window** |
|  | Elements Tab:  This tab displays the HTML structure of the webpage, showing the DOM (Document Object Model) hierarchy.  You can inspect and manipulate HTML elements, their attributes, and content.  It often shows CSS rules applied to elements, allowing you to see and edit styles.  Console Tab:  The console is an interactive JavaScript console where you can execute JavaScript code and see the results.  It's used for debugging, testing code snippets, logging information, and interacting with the page using JavaScript.  Sources Tab:  The sources tab is used for debugging and inspecting JavaScript code.  It allows you to see and navigate through JavaScript files, set breakpoints, step through code, and inspect variables.  Network Tab:  The network tab displays all network requests made by the webpage, including HTTP requests for resources like images, stylesheets, and scripts.  You can analyze request and response headers, status codes, and timings.  Performance Tab:  The performance tab helps you analyze the performance of your webpage by recording and visualizing metrics related to loading, rendering, scripting, and more.  Application (or Storage) Tab:  This tab provides access to various types of storage used by websites, including cookies, local storage, session storage, You can view, modify, add, or remove data stored in these storage mechanisms.  Security Tab:  The security tab provides information about the security of the page, including SSL certificates and mixed content issues.  It warns about insecure resources and potential security vulnerabilities.  Top of Form |
| **5.1.3.3** | **Caching** |
|  | Browsers can cache static resources like images, stylesheets, and scripts. When a user visits a website, the browser stores these resources locally. On subsequent visits, the browser checks if the resource has changed on the server before fetching it again.   1. Checking cache API 2. Opening cache 3. Adding Resources 4. Retrive Resources 5. Deleting Resources |
| **5.2** | **OOJS study** |
| **5.2.1** | **What is OOJS** |
|  | OOJS stands for "Object-Oriented JavaScript," which is a programming paradigm and approach that uses objects to organize and structure code.  Key concepts of Object-Oriented JavaScript include:   1. Object 2. Class 3. Inheritance 4. Encapsulation 5. Polymorphism 6. Abstraction |
| **5.2.2** | **Possible ways to implement class** |
|  | ES6 Classes: - Keyword class  Constructor Functions: - keyword function |
| **5.2.3** | **Static class, Properties declaration** |
|  | Static properties can be used to initialize the class or configure its behavior globally.  Static properties are properties that are associated with the class itself, not with its instances. They are accessed using the class name, not an instance of the class.  Static methods are methods that are defined on the class itself, not n its instances. They can't access instance-specific data. |
| **5.3** | **ECMAScript6** |
| **5.3.1** | **Difference between let, var & const** |
|  | **Var**: was the original way to declare variables in JavaScript, and it has function scope or global scope, depending on where it's declared.  **Let:** was introduced in ES6 (ECMAScript 2015) and has block scope.  **Const**: is also introduced in ES6 and has block scope like let.  Variables declared with const must be assigned a value when declared, and their value cannot be reassigned after that initial assignment. |
| **5.3.2** | **JavaScript Classes** |
|  | **Class Declaration:** To define a class, you use the class keyword followed by the class name. Inside the class block, you can define the constructor and methods.  **Creating Instances:** To create an instance (object) of a class, you use the new keyword followed by the class name, passing any required constructor arguments.  **Methods and Properties:** Class methods and properties are defined within the class block. They can be accessed and used on instances of the class.  **Inheritance:** You can create a subclass (derived class) that inherits properties and methods from a parent class using the extends keyword. |
| **5.3.4** | **Arrow functions** |
|  | An arrow function in JavaScript is a concise way to write function expressions.  const functionName = (param1, param2, ...) => {  // function body  }; |
| **5.3.4** | **Import, Export, async, await Functions** |
|  | **Import:**  In another file, you can import the exported entities using the import statement.  **Export:**  In a module, you can export functions, variables, classes, or other entities to make them available for use in other files.  **Async:**  An async function returns a Promise. Within an async function, you can use the await keyword to pause the execution until a Promise is resolved  **Await:**  The await keyword is used to pause the execution of the async function until the Promise is resolved. |
| **5.4** | **Extra** |
| **5.4.1** | **Difference between == & ===, != & !===** |
|  | ==: Checks for equality after performing type coercion if necessary.  ===: Checks for strict equality (both value and data type).  !=: Checks for inequality after performing type coercion if necessary.  !==: Checks for strict inequality (both value and data type). |