# Advance JS Training Document

**session Storage**:

maintains a separate storage area for each given origin that's available for the duration of the page session (as long as the browser is open, including page reloads and restores)

**local Storage**:

 does the same thing, but persists even when the browser is closed and reopened?

[Reference Link: https://developer.mozilla.org/en-US/docs/Web/API/Web\_Storage\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API)

**Cookie**:

* The Document property cookie lets you read and write cookies associated with the document. It serves as a getter and setter for the actual values of the cookies.
* Cookie Attributes:
  + **path**=path (e.g., '/', '/midrib') If not specified, defaults to the current path of the current document location.
  + **domain**=domain (e.g., 'example.com' or 'subdomain.example.com'). If not specified, this defaults to the host portion of the current document location.
  + **expires**=date-in-GMTString-format If neither expires nor max-age specified it will expire at the end of the session.
  + **secure** Cookie to only be transmitted over a secure protocol like HTTPS. Before Chrome 52, this flag could appear with cookies from HTTP domains.
  + **samesite** SameSite prevents the browser from sending this cookie along with cross-site requests. Possible values are lax, strict or none.

**Difference Between Local Storage, Session Storage**,**And Cookies**

|  |  |  |
| --- | --- | --- |
| Local Storage | Session Storage | Cookies |
| The storage capacity of local storage is 5MB/10MB | The storage capacity of session storage is 5MB | The storage capacity of Cookies is 4KB |
| As it is not session-based, it must be deleted via JavaScript or manually | It’s session-based and works per window or tab. This means that data is stored only for the duration of a session, i.e., until the browser (or tab) is closed | Cookies expire based on the setting and working per tab and window |
| The client can only read-write local storage | The client can only read-write local storage | Both clients and servers can read and write the cookies |
| There is no transfer of data to the server | There is no transfer of data to the server | Data transfer to the server is exist |
| There are fewer old browsers that support it | There are fewer old browsers that support it | It is supported by all the browsers including older browser |

# **Browser Debugging using Chrome-Dev Tools**

* This is a very powerful and useful tool for debugging our website.
* Every browser has its debugging tools.
* We can open this tool in chrome via right-clicking anywhere on the web page and selecting inspect elements.
* After clicking inspect element A new window opens on the right site.
* There are different tabs available to use in this window.
* Every tab has its functionality.

**First is the Elements window:**

* We can inspect a particular element of the whole DOM Model and change its value.
* We can select a particular element of the website and manipulate its value or add or delete that element by clicking on inspect button.
* We can Add Edit find, reorder, or filter any node element or attribute via this tab.
* We can see add edit or delete all the CSS applied on a particular CSS attribute at bottom of this tab.
* We can also inspect the CSS grid, and debug the CSS flex value here also.

**Console:**

* Web developers often log messages to the Console to make sure that their JavaScript is working as expected.
* To log a message, you insert an expression like console.log('Hello, Console!') into your JavaScript.
* When the browser executes your JavaScript and sees an expression like that, it knows that it's supposed to log the message to the Console.
* We can log, Run javascript, watch javascript value and debug js code here.

**Sources:**

* We can use this panel to [View files](https://developer.chrome.com/docs/devtools/javascript/sources/#files).
* [Edit CSS and JavaScript](https://developer.chrome.com/docs/devtools/javascript/sources/#edit).
* [Create and save **Snippets** of JavaScript](https://developer.chrome.com/docs/devtools/javascript/sources/#snippets), which you can run on any page. **Snippets** are similar to bookmarklets.
* [Debug JavaScript](https://developer.chrome.com/docs/devtools/javascript/sources/#debug).
* [Set up a Workspace](https://developer.chrome.com/docs/devtools/javascript/sources/#workspace), so that changes you make in DevTools get saved to the code on your file system.

**Network:**

* This is also the most useful panel.
* It is used when you need to make sure that resources are being downloaded or uploaded as expected. The most common use cases for the Network panel are:
  + Make sure that resources are being uploaded or downloaded at all.
  + Inspecting the properties of an individual resource, such as its HTTP headers, content, size, and so on.

**Performance:**

* It is used to check and analyse the performance of the website.
* We can record runtime performance on this panel.
* We can also analyse the resulting frame by frame here.
* By using this feature we can find all the bottleneck code and remove it.

**Memory:**

* This panel is useful for analysing memory.
* We can fix memory problems, record heap snapshots and manage allocation via the allocation profilers tool.
* The allocation profiler tool is useful for finding objects that aren't being properly garbage collected, and continue to retain memory.
* Application:
* This panel is useful for debugging background services, debugging progressive web apps, viewing, editing or deleting the cookie, local storage data, session storage data and cache data here.

**Sensors:**

* This panel is useful for override geolocation and simulating device orientation.

**Remote debugging:**

* This panel is used for remotely debugging android devices, Accessing local servers and remotely debugging Webviews.

[Reference link: https://developer.chrome.com/docs/devtools/](https://developer.chrome.com/docs/devtools/)

# **Object-Oriented JavaScript**

* JavaScript **IS** an object-oriented language. Object literals are objects, arrays are objects, functions are objects, etc.
* JavaScript **IS NOT** a class-based language. You could say: "Hey, I've seen *class* and *extends* keywords in ES6!". Yes, but this is just syntactic sugar. JavaScript is not based on classes like Java, it is based on **prototypes**.
* With ES5 syntax, there are two ways to create what you call a "class":
  + An object literal
  + A constructor functions
* With the object literal, you have a sort of Singleton where properties are all static to some extent.
* With the constructor function, you can produce instances whose properties will be introduced by this keyword.
* We access object property using Dot [object.property or method] and Bracket[object[‘property’]] notation.

[Reference Link: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics)

<https://stackoverflow.com/questions/43474967/call-es5-class-method-from-static-method>

**Object Prototype:**

* Prototypes are the mechanism by which JavaScript objects inherit features from one another.
* The property of an object that points to its prototype is **not** called prototype. Its name is not standard, but in practice, all browsers use [\_\_proto\_\_](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/proto). The standard way to access an object's prototype is the [Object.getPrototypeOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getPrototypeOf) method.
* In JavaScript, all functions have a property named prototype. When you call a function as a constructor, this property is set as the prototype of the newly constructed object (by convention, in the property named \_\_proto\_\_).
* So, if we set the prototype of a constructor, we can ensure that all objects created with that constructor are given that prototype
* Properties that are defined directly in the object, like the name here, are called **own properties**, and you can check whether a property is an own property using the static [Object.hasOwn ()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwn) method.

# **Classes in JavaScript**

* Classes in JS are introduced in the ES6 version.
* Syntax of class and implementation of js same as in other languages like JAVA and C++.
* For creating a constructor in a class function name must be the constructor.
* For implementing inheritance, we must extend another class.
* With an access property or method of superclass via super keyword.
* For performing encapsulation, we can make property private by adding ‘#’ before the property or method name.
* For making static property or method in object literal or class we can write property name or method name with the class name.

# **Difference let, var, const:**

* var declarations are globally scoped or function scoped while let and const are blocks scoped.
* var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.
* They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.
* While var and let can be declared without being initialized, const must be initialized during declaration.

[Reference Link: https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/](https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/)

# **Difference between =,==,===**

* = is used for assigning values to a variable, == is used for comparing two variables, but it ignores the datatype of the variable whereas === is used for comparing two variables, but this operator also checks datatype and compares two values.
* = is called an assignment operator, == is called a comparison operator whereas It is also called a comparison operator.
* = does not return true or false, == Return true only if the two operands are equal while === returns true only if both values and data types are the same for the two variables.

[Reference Link: https://www.guru99.com/difference-equality-strict-operator-javascript.html#:~:text=KEY%20DIFFERENCES%3A,datatype%20and%20compares%20two%20values](https://www.guru99.com/difference-equality-strict-operator-javascript.html#:~:text=KEY%20DIFFERENCES%3A,datatype%20and%20compares%20two%20values).

# **Arrow Function Limitation:**

There are differences between arrow functions and traditional functions, as well as some limitations:

* Arrow functions don't have their bindings to [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this) or [super](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super), and should not be used as [methods](https://developer.mozilla.org/en-US/docs/Glossary/Method).
* Arrow functions don't have access to the [new. target](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new.target) keyword.
* Arrow functions aren't suitable for the [call](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call), [apply](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply) and [bind](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind) methods, which generally rely on establishing a [scope](https://developer.mozilla.org/en-US/docs/Glossary/Scope).
* Arrow functions cannot be used as [constructors](https://developer.mozilla.org/en-US/docs/Glossary/Constructor).
* Arrow functions cannot use [yield](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield), within its body.

[Reference Link: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow\_functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions)