1. **Advance JavaScript**

**1 Advance Concepts**

* 1. **Session Storage & LocalStorage**

**Session Storage**:

 Stores data only for the duration of a page session. Data is lost when the page session ends (i.e., when the tab or window is closed).

 Accessible only within the current tab.

Ex:   
// Set data in session storage

sessionStorage.setItem("username", "Keyur");

// Retrieve data

const user = sessionStorage.getItem("username"); // "Keyur"

// Remove data

sessionStorage.removeItem("username");

**Local Storage**:

* Data persists even after the browser or tab is closed, remaining available indefinitely until deleted.
* Accessible across multiple tabs and windows in the same browser.

Ex :

// Set data in local storage

localStorage.setItem("username", "Keyur");

// Retrieve data

const user = localStorage.getItem("username"); // "Keyur"

// Remove data

localStorage.removeItem("username");

Difference:

Session Storage: Temporary data storage for each session.

Local Storage: Persistent storage across sessions.

* 1. **Basics of Cookies**
* What are Cookies?

Small pieces of data stored on the user's computer by the web browser, used to track, authenticate, and store information.

* Properties:

expires: Sets an expiration date.

domain: Specifies the domain the cookie is available to.

path: Restricts the cookie to a specific path.

secure: Makes the cookie accessible only through HTTPS.

Ex :

// Setting a cookie

document.cookie = "username=Keyur; expires=Fri, 31 Dec 2024 23:59:59 GMT; path=/";

// Retrieving cookies

console.log(document.cookie); // "username=Keyur"

**1.3 Browser Debugging**

Inspect Element Window: Access developer tools by right-clicking a page and selecting “Inspect” or pressing Ctrl+Shift+I.

Detailed Tabs:

* Elements: View and edit HTML and CSS. Inspect the DOM structure.
* Console: Execute JavaScript, log errors, and see debug information.
* Sources: View and debug JavaScript files, add breakpoints, and step through code.
* Network: Monitor network requests, see request/response headers, and analyze page load performance.
* Performance: Measure page performance, identify delays and bottlenecks.
* Application: Manage storage (cookies, local storage, etc.), and cache.

1. **Object-Oriented JavaScript (OOJS)**

**2.1 What is OOJS?**

OOJS is a programming paradigm based on objects rather than functions or procedures.

* Core principles:

Encapsulation: Wrapping code into a single unit.

Inheritance: Mechanism to create a new class using an existing class.

Polymorphism: Same method behaves differently in different contexts.

Abstraction: Hides complex implementation details.

**2.2 Possible ways to implement class**

* **ES6 Class Syntax**

Introduced in ECMAScript 6, this syntax provides a more readable and structured way to define classes.

Ex :

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

console.log(`Hello, my name is ${this.name}.`);

}

}

const person1 = new Person('Keyur', 20);

person1.greet();

* **Constructor Functions (Pre-ES6)**

Before ES6, classes were commonly created using functions.

Ex :

function Person(name, age) {

this.name = name;

this.age = age;

}

Person.prototype.greet = function() {

console.log(`Hello, my name is ${this.name}.`);

};

const person1 = new Person('Keyur', 20);

person1.greet();

**2.3 Static class, Properties declaration**

Static Methods

* Declared using the static keyword within a class.
* Can only be called on the class itself, not on instances of the class.

Ex :

class MathUtility {

// Static method to calculate square of a number

static square(number) {

return number \* number;

}

}

// Accessing static method directly from the class

console.log(MathUtility.square(5)); // Output: 25

Static Properties

* A static property is assigned directly to the class, outside the constructor.
* From JavaScript (ES2022), you can declare static properties directly within the class body using the static keyword.

Ex :

class Config {

// Declaring a static property

static appName = "My Application";

static version = "1.0";

static getAppInfo() {

return `${this.appName} - Version ${this.version}`;

}

}

// Accessing static properties directly from the class

console.log(Config.appName); // Output: "My Application"

console.log(Config.getAppInfo()); // Output: "My Application - Version 1.0"

1. **ECMAScript6**
   1. **Difference between let, var & const**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | var | let | const |
| Scope | Function-scoped | Block-scoped | Block-scoped |
| Re-declaration | Allowed | Not Allowed | Not Allowed |
| Mutability | Mutable | Mutable | Immutable |

Ex :

var name = “keyur”; // function-scoped

let age = 20; // block-scoped

const birthYear = 2004; // cannot be changed

* 1. **JavaScript Classes**
* Defining a Class

A class in JavaScript is defined using the class keyword, followed by the class name and a body enclosed in {} brackets.

Class Components

1. Constructor:
   * The constructor is a special method for creating and initializing an instance of a class.
   * Each class can have only one constructor.
   * It’s called automatically when a new instance of the class is created

Ex :

class Animal {

constructor(type) {

this.type = type;

}

}

const cat = new Animal("Cat");

console.log(cat.type); // Output: "Cat"

1. Methods :
   * Defined directly within the class body.
   * No need for the function keyword.

Ex :

class Car {

constructor(model) {

this.model = model;

}

start() {

return `${this.model} is starting...`;

}

}

const myCar = new Car("Tesla Model S");

console.log(myCar.start()); // Output: "Tesla Model S is starting..."

1. Inheritance

Classes can inherit from other classes using the extends keyword. The super keyword is used to call the constructor of the parent class.

Ex :

class Animal {

constructor(name) {

this.name = name;

}

speak() {

return `${this.name} makes a noise.`;

}

}

class Dog extends Animal {

constructor(name, breed) {

super(name); // Call parent constructor

this.breed = breed;

}

speak() {

return `${this.name}, the ${this.breed}, barks.`;

}

}

const myDog = new Dog("Buddy", "Golden Retriever");

console.log(myDog.speak()); // Output: "Buddy, the Golden Retriever, barks."

* 1. **Arrow functions**

Arrow functions allows a short syntax for writing function expressions.

You don't need the function keyword, the return keyword, and the curly brackets.

Ex :

// ES5  
var x = function(x, y) {  
   return x \* y;  
}  
  
// ES6  
const x = (x, y) => x \* y;

Arrow functions do not have their own “this”.

Arrow functions are not hoisted. They must be defined before they are used.

Using const is safer than using var, because a function expression is always a constant value.

You can only omit the return keyword and the curly brackets if the function is a single statement.

* 1. **Import, Export, async, await Functions**

**export**: Used to make functions, variables, or classes available to other files.

There are two types of exports:

* **Named exports**: Export multiple values by name.
* **Default exports**: Export a single value per file as the default export.

Named Ex :

// math.js

export const add = (a, b) => a + b;

export const subtract = (a, b) => a - b;

Default Ex :

// greet.js

const greeting = "Hello, World!";

export default greeting;

**import**: Used to bring in values from other files.

Importing Named Exports:

// main.js

import { add, subtract } from './math.js';

console.log(add(5, 3)); // Output: 8

Importing a Default Export:

// main.js

import greeting from './greet.js';

console.log(greeting); // Output: "Hello, World!"

**Async and Await :** async and await provide a way to work with asynchronous code more simply than using Promises alone. An async function always returns a Promise, and await pauses execution until the Promise resolves.

Ex :

async function getData() {

const response = await fetch('https://api.example.com/data');

const data = await response.json();

return data;

}

getData().then(console.log); // Outputs fetched data

1. **Extra Points**
   1. **Difference between == & ===, != & !===**

**Summary Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Allows Type conversion** | **Example** | **Result** |
| == | Loose equality | Yes | 5 == '5' | true |
| === | Strict equality | No | 5 === '5' | false |
| != | Loose inequality | Yes | 5 != '5' | false |
| !== | Strict inequality | No | 5 !== '5' | true |

**Spread Operator**

The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

Ex:

const myVehicle = {

brand: 'Ford',

model: 'Mustang',

color: 'red'

}

const updateMyVehicle = {

type: 'car',

color: 'yellow'

}

const myUpdatedVehicle = {...myVehicle, ...updateMyVehicle}

**Function Rest Parameter**

The rest parameter (...) allows a function to treat an indefinite number of arguments as an array:

Ex :

function sum(...args) {  
  let sum = 0;  
  for (let arg of args) sum += arg;  
  return sum;  
}  
  
let x = sum(4, 9, 16, 25, 29, 100, 66, 77);