**var:**

* Scope: Variables declared with var are function-scoped. They are accessible within the function in which they are declared, or globally if declared outside any function.
* Hoisting: Variables declared with var are hoisted to the top of their scope. This means they can be used before they are declared in the code.
* Reassignment: Variables declared with var can be reassigned and redeclared within the same scope.

Example:

function example() {

if (true) {

var x = 10;

}

console.log(x); // Outputs: 10 (accessible outside the if block)}

**let:**

* Scope: Variables declared with let are block-scoped. They are accessible only within the block (e.g., inside a loop or conditional statement) in which they are declared.
* Hoisting: Variables declared with let are hoisted to the top of their block but are not initialized. This means they cannot be used before they are declared in the code.
* Reassignment: Variables declared with let can be reassigned but not redeclared within the same block scope.

Example:

function example() {

if (true) {

let y = 20;

console.log(y); // Outputs: 20 (accessible within the if block)

}

console.log(y); // Error: y is not defined (outside the if block)}

**const:**

* Scope: Variables declared with const are block-scoped like let.
* Hoisting: Variables declared with const are hoisted to the top of their block but are not initialized. They cannot be used before they are declared.
* Reassignment: Variables declared with const cannot be reassigned or redeclared. However, for objects and arrays, the content (properties or elements) can be modified, but the variable itself cannot be reassigned.

Example:

function example() {

const z = 30;

console.log(z); // Outputs: 30 (accessible within the function)

z = 40; // Error: Assignment to constant variable

}

const person = { name: 'Alice', age: 25 };

person.age = 30; // Valid: Modifying object properties is allowed

person = { name: 'Bob', age: 30 }; // Error: Assignment to constant variable

Choosing Between let and const:

Use let when the variable's value might change over time.

Use const for variables that should not be reassigned, especially for constants like configuration values, mathematical constants, or references to immutable objects.

It's a best practice to use const by default and only use let when you know the variable will change.

**Implicit, Explicit, Nominal, Structuring, and Duck Typing:**

* Implicit Typing: When a variable’s type is determined by the value assigned to it. For example:

const x = 42; // Implicitly assigns the type "number" to x

* Explicit Typing: When you explicitly declare the type of a variable. In TypeScript, this is common:

const y: string = "hello";

* Nominal Typing: Type checking based on the name or label of a type. TypeScript uses nominal typing to distinguish between different types with the same structure.
* Structural Typing: Type checking based on the structure or shape of an object. JavaScript and TypeScript use structural typing.
* Duck Typing: If an object behaves like a certain type (e.g., quacks like a duck), it’s treated as that type. Duck typing is common in dynamically typed languages like JavaScript.

**== vs === vs typeof:**

* == (loose equality): Compares values after type coercion. For example:

console.log(1 == "1"); // true (string is coerced to a number)

* === (strict equality): Compares values without type coercion. For example:

console.log(1 === "1"); // false

* typeof: A unary operator that returns a string representing the type of an operand. For example:

console.log(typeof 42); // "number"

**Function Scope, Block Scope, and Lexical Scope:**

* Function Scope: Variables declared inside a function are only accessible within that function.

function greet() {

const message = "Hello!";

console.log(message);

}

greet(); // Outputs: "Hello!"

// console.log(message); // Error: message is not defined outside the function

* Block Scope: Introduced by let and const. Variables declared within a block (e.g., inside an if statement) are only accessible within that block.

if (true) {

let count = 10;

console.log(count); // Outputs: 10

}

// console.log(count); // Error: count is not defined outside the block

* Lexical Scope (Closures): Functions remember the scope where they were created. Inner functions have access to variables from outer functions.

function outer() {

const outerVar = "I'm from outer!";

return function inner() {

console.log(outerVar);

};

}

const innerFunc = outer();

innerFunc(); // Outputs: "I'm from outer!"

**IIFE, Modules, and Namespaces:**

* IIFE (Immediately Invoked Function Expression): A function that runs immediately after being defined. Used for encapsulation and avoiding global scope pollution.

(function () {

const secret = "I'm hidden!";

console.log(secret);

})(); // Outputs: "I'm hidden!"

// console.log(secret); // Error: secret is not defined outside the IIFE

* Modules: A way to organize code into separate files. ES6 introduced native support for modules using import and export.
* Namespaces: A way to organize code by grouping related functionality under a single name. Commonly used in TypeScript.

**Manipulating using DOM:**

* Accessing Elements: To work with elements, you first need to access them using methods like getElementById, getElementsByClassName, or querySelector. For example:

const myElement = document.getElementById("myId");

* Modifying Content and Styles: You can change an element’s content using properties like innerHTML or textContent. For styles, use properties like style.backgroundColor or classList. Example:

myElement.innerHTML = "New content";

myElement.style.backgroundColor = "blue";

* Adding or Removing Elements: Creating, adding, or removing elements are key aspects of DOM manipulation. You can use methods like createElement, appendChild, or removeChild. Example:

const newElement = document.createElement("div");

newElement.textContent = "New element!";

document.body.appendChild(newElement);

* Changing Attributes:

We can modify attributes like src, href, or data-\* using properties like setAttribute or directly accessing the attribute. Example:

myElement.setAttribute("src", "new-image.jpg");

* Event Handling: Add event listeners to elements to make them interactive. Use methods like addEventListener to respond to events like clicks or form submissions. Example:

myElement.addEventListener("click", () => { console.log("Element clicked!");});

**JSON:**

JSON stands for JavaScript Object Notation.

It is a lightweight, text-based format used for storing and transporting data.

JSON is self-describing, meaning it is easy for humans to read and write, as well as parse and generate programmatically.

It is language-independent, making it widely adopted in various programming languages beyond JavaScript.

JSON Syntax:

JSON data consists of name/value pairs (similar to JavaScript object properties).

Each name/value pair is separated by a comma.

Objects are enclosed in curly braces {}, and arrays are enclosed in square brackets [].

Property names (keys) are always enclosed in double quotes.

Example of a simple JSON object:

{

"name": "John",

"age": 30,

"car": null

}

**Why Use JSON?**

Similar to JavaScript Objects: The JSON format is syntactically similar to the code for creating JavaScript objects. JavaScript can easily convert JSON data into native JavaScript objects.

Text-Based: JSON data can be sent between computers as plain text, making it versatile and usable by any programming language.

Parsing and Serialization: JavaScript provides built-in functions for converting JSON strings to objects (JSON.parse()) and objects to JSON strings (JSON.stringify()).

Data Interchange: JSON is commonly used for APIs, configuration files, and structured data.

Example Usage:

Suppose we receive the following JSON string from a server:

{ "employees": [

{ "firstName": "John", "lastName": "Doe" },

{ "firstName": "Anna", "lastName": "Smith" },

{ "firstName": "Peter", "lastName": "Jones" }

]

}

We can parse it into a JavaScript object:

const jsonString = '{ ... }'; // The received JSON string

const obj = JSON.parse(jsonString);

console.log(obj.employees[1].firstName); // Outputs: "Anna"