1. **var, let, and const:**

var declarations are globally scoped or function scoped.

while let and const are block 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.

Vars are hoisted

While var and let can be declared without being

initialized, const must be initialized during declaration

1. **Undefined vs. Not Defined vs. NaN:**

- Undefined: Variable declared but not assigned a value or a function that doesn't return a value explicitly.

- Not Defined: Variable that hasn't been declared or is out of scope.

- NaN: Represents a value that is "Not a Number" when performing arithmetic operations.

1. **Null vs. Undefined**

- Null: Explicitly assigned to a variable to indicate an empty or non-existent value.

- Undefined: Represents an uninitialized or unassigned value.

1. **Execution Context:**

In JavaScript, an execution context is an environment in which a piece of code is executed. It includes variables, functions, and the scope chain, which defines the accessibility of variables. Understanding execution contexts is crucial for grasping how JavaScript code is executed.

There are three types of execution contexts in JavaScript:

**Global Execution Context:**

The outermost context that represents the global scope.

Variables and functions defined in the global context are accessible throughout the entire program.

Created when the script is run and destroyed when the script finishes.

**Function Execution Context:**

Created whenever a function is called.

Each function has its own execution context, which includes parameters, local variables, and a reference to its outer (enclosing) context (the context in which the function was defined).

The function context is pushed onto the call stack when the function is invoked and popped off the stack when the function completes.

1. **Use of `prompt` in JavaScript:**

The prompt function in JavaScript is used to interact with the user by displaying a dialog box that prompts them to enter input. It's often used to collect information or responses from users within a web page.

Example:

var userInput = prompt("Please enter your name:");

alert("Hello, " + userInput + "!");

1. **JavaScript's Dynamic Nature:**

JavaScript is often described as a dynamically-typed and loosely-typed language, and its dynamic nature refers to several key characteristics that contribute to its flexibility and versatility. Here are some aspects of JavaScript's dynamic nature:

**Dynamic Typing:**

JavaScript is dynamically typed, meaning that you don't have to explicitly declare the data type of a variable. The data type of a variable is determined at runtime.

Example:

let x = 5; // x is a number

x = "Hello"; // x is now a string

x = [1, 2, 3]; // x is now an array

**Variable Mutability:**

Variables in JavaScript can change their type during the execution of a program. This allows for more flexibility but also requires careful attention to avoid unexpected behavior.

Example:

let message = "Hello, World!";

message = 42; // Valid, but the type has changed

**Late Binding:**

JavaScript performs late binding, meaning that the resolution of the value of a variable or the implementation of a function occurs at runtime. This is in contrast to languages with early binding, where these resolutions happen at compile-time.

Example:

function greet() {

console.log("Hello, World!");

}

let myFunction = greet; // Assigning the function to another variable

myFunction(); // Calling the function using the new variable

**Object Properties:**

Objects in JavaScript can have dynamic properties. Properties can be added or removed from an object even after the object is created.

Example:

let person = { name: "John", age: 30 };

person.gender = "Male"; // Adding a new property

delete person.age; // Removing a property

**Dynamic Functionality:**

JavaScript allows the dynamic creation and modification of functions during runtime. Functions can be assigned to variables, passed as arguments, and returned from other functions.

Example:

function add(a, b) {

return a + b;

}

let myFunction = add; // Assigning a function to a variable

console.log(myFunction(2, 3)); // Calling the function using the variable