**Question 1: What are variables in JavaScript? How do you declare a variable using var, let and const?**

**Variables in JavaScript**

Variables in JavaScript are used to store data values that can be referenced and manipulated in a program. JavaScript variables can hold different types of values such as numbers, strings, objects, arrays, and more.

**Declaring Variables in JavaScript**

JavaScript provides three ways to declare variables: var, let, and const.

**1. Using var**

* var was traditionally used in JavaScript but has some issues related to scoping.
* It has **function scope** (i.e., accessible throughout the function in which it is declared).
* It can be **redeclared** within the same scope.

**Example:**

**var name = "Dharmik";**

**console.log(name); // Output: Dharmik**

**var name = "Modi"; // Redeclaring is allowed**

**console.log(name); // Output: Modi**

**2. Using let**

* Introduced in ES6, let is **block-scoped** (i.e., accessible only within the block {} where it is declared).
* It **cannot be redeclared** in the same scope.
* However, it **can be updated** (i.e., reassigned a new value).

**Example:**

let age = 25;

console.log(age); // Output: 25

age = 30; // Allowed (Reassigning)

console.log(age); // Output: 30

// let age = 35; // Error! Cannot redeclare let in the same scope

**3. Using const**

* const is also **block-scoped** like let.
* It **must be initialized at the time of declaration** (i.e., it requires a value).
* It **cannot be reassigned** (i.e., immutable reference).

**Example:**

const country = "India";

console.log(country); // Output: India

// country = "USA"; // Error! Assignment to constant variable

**Question 2: Explain the different data types in JavaScript. Provide examples for each.**

**1. Primitive Data Types**

Primitive types are immutable and stored directly in memory.

**a) Number**

* Represents both integers and floating-point numbers.
* Special values: Infinity, -Infinity, and NaN (Not a Number).

**Example:**

let age = 25;

let price = 99.99;

let infinityValue = Infinity;

console.log(typeof age); // Output: number

**b) String**

* Represents a sequence of characters.
* Can be written using single ('), double ("), or backticks (`).

**Example:**

let name = "John Doe";

let greeting = `Hello, ${name}!`; // Template literals

console.log(typeof name); // Output: string

**c) Boolean**

* Represents true or false.

**Example:**

let isJavaScriptFun = true;

console.log(typeof isJavaScriptFun); // Output: boolean

**d) Undefined**

* A variable declared but not assigned a value.

**Example:**

let x;

console.log(x); // Output: undefined

console.log(typeof x); // Output: undefined

**e) Null**

* Represents an intentional absence of value.
* Unlike undefined, it is explicitly assigned.

**Example:**

let y = null;

console.log(y); // Output: null

console.log(typeof y); // Output: object (JavaScript quirk)

**f) Symbol (ES6)**

* Represents unique identifiers.
* Useful in objects to create unique keys.

**Example:**

let sym1 = Symbol("id");

let sym2 = Symbol("id");

console.log(sym1 === sym2); // Output: false (Symbols are unique)

**g) BigInt (ES11)**

* Used for large numbers beyond Number.MAX\_SAFE\_INTEGER.

**Example:**

let bigNumber = 9007199254740991n; // 'n' denotes BigInt

console.log(typeof bigNumber); // Output: bigint

**2. Non-Primitive (Reference) Data Types**

Reference types store memory addresses instead of actual values.

**a) Object**

* A collection of key-value pairs.

**Example:**

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

console.log(person.name); // Output: Alice

console.log(typeof person); // Output: object

**b) Array (Subtype of Object)**

* Ordered collection of values.

**Example:**

let numbers = [1, 2, 3, 4, 5];

console.log(numbers[0]); // Output: 1

console.log(typeof numbers); // Output: object

**c) Function (Subtype of Object)**

* A reusable block of code.

**Example:**

function greet() {

return "Hello, World!";

}

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

console.log(typeof greet); // Output: function

**Question 3: What is the difference between undefined and null in JavaScript?**

In JavaScript, undefined and null are both used to represent the absence of a value, but they have different meanings and uses:

1. **undefined:**
   * It means a variable has been declared but has not been assigned a value.
   * Functions return undefined when no explicit return value is specified.
   * Accessing an object property or an array element that does not exist returns undefined.
   * It is the default value of uninitialized variables.

**Example**:

let x;

console.log(x); // undefined

function test() {}

console.log(test()); // undefined

let obj = {};

console.log(obj.prop); // undefined

1. **null**:

* It is an intentional absence of any object value.
* It is a primitive value that represents "nothing" or "empty".
* Unlike undefined, null is explicitly assigned to a variable.

**Example**:

**let y = null;**

**console.log(y); // null**

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

**console.log(person.age); // null**