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

Answer:- In JavaScript, a **variable** is a container that holds data that can be referenced and manipulated in your program. Variables are used to store values such as numbers, strings, objects, arrays, etc.

There are three primary ways to declare a variable in JavaScript: using var, let, and const. Each one has different behavior and scoping rules. Here’s a breakdown of how you use them:

**1. var**

var is the oldest way to declare a variable in JavaScript, but it has some limitations related to scoping. Variables declared with var are **function-scoped** or **globally-scoped** (if not in a function), and they can be re-assigned and redeclared within their scope.

**Key Characteristics of var:**

* **Function-scoped**: A variable declared with var inside a function is accessible within the entire function, even before its declaration (hoisting).
* **Global scope**: If declared outside a function, it’s globally accessible.
* **Redeclaring**: You can redeclare a variable using var within the same scope.

**2. let**

let was introduced in ECMAScript 6 (ES6) to provide block-scoping to variables. Unlike var, let is **block-scoped**, meaning it’s only accessible within the block (like loops or conditionals) where it's declared.

**Key Characteristics of let:**

* **Block-scoped**: The variable is only accessible within the block (e.g., inside {} in a loop or a conditional statement).
* **No redeclaration**: You cannot redeclare a let variable within the same scope, but you can reassign it.
* **Hoisting**: Like var, let is hoisted, but it’s not accessible before its declaration (i.e., accessing it before declaration will result in a ReferenceError).
* **3 const**
* const is also introduced in ES6 and is used to declare variables that should not be reassigned after initialization. However, note that **objects and arrays** declared with const are **mutable**, meaning their contents can be changed, but the variable itself cannot be reassigned.

**Key Characteristics of const:**

* **Block-scoped**: Like let, const is block-scoped.
* **No reassignment**: You cannot reassign a const variable after it’s been initialized. This helps avoid accidental value changes.
* **Objects/Arrays can be modified**: If you declare an object or array with const, you can modify its properties or elements, but you cannot reassign the entire object or array to something else.
* **Question 2: Explain the different data types in JavaScript. Provide examples for each.**

**1. Primitive Data Types**

Primitive data types are simple, immutable types that hold a single value. These types are not objects and are stored directly in memory.

**a) Number**

* Represents both integer and floating-point numbers.
* Example of numbers: 1, 42, 3.14, -99

**b) String**

* Represents a sequence of characters enclosed in either single quotes ('), double quotes ("), or backticks (`).
* Strings are immutable.

**c) Boolean**

* Represents a logical value: either true or false.
* Typically used for conditional checks.

**d) Undefined**

* Represents a variable that has been declared but has not yet been assigned a value.
* undefined is also the default value of function arguments that are not passed.

**e) Null**

* Represents the intentional absence of any object value.
* null is an object, though it is used to represent "no value".

**f) Symbol (ES6)**

* A unique and immutable value used primarily as object property identifiers.
* Symbols are often used to avoid name clashes in object properties.

**g) BigInt**

* Represents integers with arbitrary precision. It is used to work with large integers that exceed the size limits of the Number type.
* Can be created by appending n to the number.

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

In JavaScript, both **undefined** and **null** represent the absence of a value, but they are used in different contexts and have distinct meanings. Here's a breakdown of the differences.

**1. undefined**

* **Meaning**: undefined indicates that a variable has been **declared** but has **not yet been assigned** a value.
* **Type**: The type of undefined is its own type, which is also called undefined.
* **Usage**:
  + A variable that is declared but not assigned a value is automatically assigned undefined.
  + A function that does not return a value implicitly returns undefined.
  + When accessing an object property or array element that doesn't exist, undefined is returned.

let x;

console.log(x); // Outputs: undefined (because x is declared but not assigned)

* + function greet() {
  + // No return statement
  + }
  + console.log(greet()); // Outputs: undefined (because the function does not return anything)
  + let person = {};
  + console.log(person.age); // Outputs: undefined (property 'age' does not exist)

**2. null**

* **Meaning**: null represents an **intentional** absence of a value or **no value**. It's explicitly set by the developer to indicate "nothing" or "empty".
* **Type**: The type of null is an **object**, which is actually considered a historical bug in JavaScript.
* **Usage**:
  + null is typically used when you want to indicate that a variable or object property should not contain any value, or it's being cleared out (i.e., explicitly set to "no value").
  + It is often used for variables that will later be assigned an object or other value.

let y = null;

console.log(y); // Outputs: null (explicitly assigned a null value)

let person = {

name: "Alice",

age: null // 'age' is intentionally set to null

};

console.log(person.age); // Outputs: null (because 'age' was explicitly set to null)