JavaScript Fundamentals

1. Variables (var, let, const)

Definition: A variable is a container used to store data values in JavaScript. Think of it as a named storage where you can keep values (like numbers, strings, objects) and use them later.

Example:

```
let name = "Deepak";
```

Difference Between var, let, and const

- **var**: Function-scoped, allows re-declaration and reassignment. Hoisted and initialized with `undefined`.
- **let**: Block-scoped, allows reassignment but not re-declaration. Stays in Temporal Dead Zone (TDZ) until declared.
- **const**: Block-scoped, allows neither re-declaration nor reassignment (but objects/arrays can be modified internally).

Temporal Dead Zone (TDZ):

- **Definition**: The period between variable hoisting (memory allocated) and declaration (initialized with value).
- Accessing a variable in TDZ causes a `ReferenceError`.
- **var**: Hoisted and initialized with `undefined`, accessible before declaration.
- **let & const**: Hoisted but not initialized, stays in TDZ until declared.

Example:

```
console.log(a); // undefined (var hoisted)
var a = 10;
console.log(b); // ReferenceError (b in TDZ)
let b = 20;
console.log(c); // ReferenceError (c in TDZ)
const c = 30;
```

2. JavaScript Data Types

JavaScript has two main categories of data types: Primitive (immutable, stored by value) and Reference (mutable, stored by reference).

Primitive Types

- **Number**: `10`, `3.14`, `NaN`, `Infinity`
- **String**: `"Hello"`
- **Boolean**: `true`, `false`
- **Undefined**: Declared but not assigned
- **Null**: Intentional empty value
- **Symbol**: Unique identifier
- **BigInt**: Large integers, e.g., `123n`
- Copied by value (new copy is made).

Reference Types

- **Object**: `{ key: value }`
- **Array**: `[1, 2, 3]`
- **Function**: `function() {}`
- **Others**: `Date`, `RegExp`, `Map`, `Set`
- Copied by reference (points to same memory).

3. Type Conversion & Coercion

Type Conversion (Explicit / Type Casting)

Manually converting a value from one type to another.
Examples:

```
String(123); // "123"
Number("456"); // 456
Boolean(1); // true
```

Type Coercion (Implicit)

JavaScript automatically converts one type to another.
Examples:

```
"5" + 1 // "51" (number → string, concatenation)
"5" - 1 // 4 (string → number, subtraction)
true + 1 // 2 (boolean → number)
```

Quick Tricks:

- Explicit: You do it (manual).
- Implicit (Coercion): JS does it (automatic).
- `+`: Favors string concatenation.
- `-`, `*`, `/`: Favor number conversion.
- **Falsy Values**: `0`, `""` (empty string), `null`, `undefined`, `NaN`, `false`.

4. Difference Between == vs ===

== (Loose Equality)

Compares values only, performs type coercion.
Examples:

```
5 == "5" // true (string → number)
0 == false // true
null == undefined // true
```

=== (Strict Equality)

Compares values and types, no type coercion.
Examples:

```
5 === "5" // false (different types)
0 === false // false
null === undefined // false
10 === 10 // true
```

Shortcut:

- `==`: Loose (values only, auto conversion).
- `===`: Strict (values + types must match).

5. Truthy & Falsy Values

Definition:

- **Truthy Values**: Treated as `true` in a boolean context.
- **Falsy Values**: Treated as `false` in a boolean context.

Falsy Values (7 in JS)

- `false`
- `0`, `-0`, `0n` (BigInt zero)
- `""` (empty string)
- `null`
- `undefined`
- `NaN`

Note: Everything else is Truthy.
Examples:

```
if ("hello") { console.log("Truthy!"); } // runs
if (0) { console.log("Falsy!"); } // won't run
```

Shortcut:

- Empty, zero, nothing Falsy
- Anything else Truthy