🧾 Variables in JavaScript

🔤 Variable Declaration in JavaScript

JavaScript allows you to declare variables in **three ways**:

Keyword	Scope	Reassignment	Redeclaration	Hoisted	Initialized as
`var`	Function / Global	✅ Yes	✅ Yes	✅ Yes	`undefined`
`let`	Block	✅ Yes	❌ No	✅ Yes	`undefined`
`const`	Block	❌ No	❌ No	✅ Yes	❌ Not assigned

```
var dp;   // Function-scoped
let ap;   // Block-scoped
const hp; // ❌ Error: Missing initializer in const declaration
```

💡 Tips:

- Use `let` for variables that might change.
- Use `const` by default for constants and objects you don't reassign.
- Avoid using `var` in modern code (legacy).

📌 Declaration + Assignment

```
var myVariable = 5;
let anotherVariable = "JS";
const PI = 3.14159;
```

✅ You can declare and assign in one line.

🔠 JavaScript is Case-Sensitive

```
let myValue = 10;
let MyValue = 20;
console.log(myValue);  // 10
console.log(MyValue);  // 20
```

📊 Data Types in JavaScript

Primitive Data Types:

`Number`, `String`, `Boolean`, `BigInt`, `Symbol`, `null`, `undefined`

```
let y = BigInt("265");
let x = Symbol("I Am Symbol");
let s = null;
console.log(typeof x); // Output: symbol
```

Non-Primitive Data Type:

- **Object** – like dictionaries in Python

```
const item = {
  name: "CryptoMinds",
  age: 12
};

console.log(item["age"]); // Output: 12
```

Scope Examples

`var` is **function/global scoped**

```
var b = 11;
var b = 13;

{
  var b = 15;
  console.log(b); // 15
}
console.log(b);     // 15
```

`let` is **block scoped**

```
let b = 11;

{
  let b = 15;
  console.log(b); // 15
}
console.log(b);     // 11
```

`const` cannot be reassigned or redeclared

```
let c = 16;
c = 17; // ✅ Allowed

let d = 16;
let d = 17; // ❌ Error: Identifier 'd' has already been declared
```

🧠 Variable Naming Rules

✅ Allowed: Letters, digits, `_`, `$` ❌ Not allowed: Starting with a digit ✔️ JavaScript is case-sensitive

```
var firstName;
let age;
const PI = 3.14;
```

🧪 Hoisting in JavaScript

```
console.log(x); // Output: undefined (not ReferenceError)
var x = 10;
```

📌 `var` is hoisted and initialized as `undefined`. ❌ `let` and `const` are hoisted but not initialized — accessing them before declaration throws an error.

🔒 Const Behavior

```
const PI = 3.14;
PI = 3.14159; // ❌ Error: Assignment to constant variable

const myArray = [1, 2, 3];
myArray.push(4); // ✅ Valid
```

📌 `const` protects the *binding*, not the *value inside* the object/array.

✅ Best Practice

🟢 Use `const` unless reassignment is needed. 🟡 Use `let` for values that change. 🔴 Avoid `var` unless working with older codebases.

🧪 Function + Scope Example

```
function myFunction() {
  var x = 10;
  if (x > 5) {
    let y = 20;
    console.log(x + y); // 30
  }
  console.log(x); // 10
  console.log(y); // ❌ ReferenceError
}
```

🧠 Practice Set

🧩 Q1) Create a variable of type `string` and add a number to it.

```
let a = "Darshan";
let b = 10;

console.log(a + b); // Output: Darshan10
```

🧩 Q2) Use `typeof` to find the result type of `a + b`.

```
console.log(typeof (a + b)); // Output: string
```

🧩 Q3) Create a const object and try changing it to a number.

```
const c = {
  name: "Darshan",
  author: "CryptoMinds",
  isPrincipal: false
};

c = 1; // ❌ Error: Assignment to constant variable
```

🧩 Q4) Add a new key to the object above.

```
const c1 = {
  name: "Darshan",
  author: "CryptoMinds",
  isPrincipal: false
};

c1["friend"] = "Krupali";
console.log(c1);
```

✅ Output:

```
{
  name: 'Darshan',
  author: 'CryptoMinds',
  isPrincipal: false,
  friend: 'Krupali'
}
```

📝 `const` allows modifying internal properties, but not reassigning the object reference.

🧩 Q5) Create a word-meaning dictionary of 5 words.

```
const dict = {
  appreciate: "recognize the full worth of",
  ataraxia: "a state of freedom from emotional disturbance",
  yakka: "Work, especially hard work",
  serendipity: "the occurrence of events by chance in a happy way",
  ephemeral: "lasting for a very short time"
};

console.log(dict.yakka);          // Output: Work, especially hard work
console.log(dict["ephemeral"]);  // Output: lasting for a very short time
```

🏁 Conclusion

🧩 Understanding the difference between `var`, `let`, and `const` helps you write **cleaner**, **safer**, and **modern JavaScript**. Proper variable usage is a key pillar in building robust applications. 🎯
