Best Practices

Overview

Some practices are followed by most of the developers to maintain consistency in the codes and avoid common mistakes.

- It makes the program more readable.
- It makes the program faster and cleaner.

Best Practices :

- Avoid global variables
- Avoid new keyword
- Avoid == operator
- Avoid eval( )

Avoid Global variables

- Avoid declaring global variables, objects, and functions.
- Other scripts can overwrite global variables and functions.
- Use local variables instead, making use of closures.

NOTE: Local variables must be declared with the **var** keyword or the **let** keyword. Otherwise, they will become global variables or **use strict mode**.

Variables on top

Put all declarations at the top of the javascript program or function.

- It makes the code cleaner.
- Provide a single place to look for local variables
- Make it easier to avoid unwanted (implied) global variables
- Reduce the possibility of unwanted re-declarations
- Declare the variables at the top, and they can be used/initialised later

Example :	// Declaration at the start let name, salary , expenditure, savings ; name = "Sam" ; salary = 50000 ; expenditure = 30000 ; savings = salary - expenditure ;

Initialisation of variables

Initialise variables when you declare them.

- Give cleaner code
- Provide a single place to initialise variables
- **Avoid undefined values**

Initialising variables provides an idea of the intended use and intended data type as javascript is a dynamically typed language.

Example :	let name = "Sam" ; let salary = 50000 ; let expenditure = 30000 ;

Declaring using const

While creating an Array or Object, declare them using const to prevent accidental change of type.

Object :

Example :	**let** student = { name : “Sam” , age : “20” , course : “Btech” } ; student = “Yash” ; **// Changes object to string**
	const student = { name : “Sam” , age : “20” , course : “Btech” } ; student = “Yash” ; **//Re-Initialisation not possible**

Array :

Example :	**let** arr = { 1 , 2 , 3 } ; arr = “Array” ; **// Changes object to string**
	const arr = { 1 , 2 , 3 } ; arr = “Array” ; **//Re-Initialisation not possible**

Avoid new keyword

- Use **""** instead of **new String( )**
- Use **0** instead of **new Number( )**
- Use **false** instead of **new Boolean( )**
- Use **{ }** instead of **new Object( )**
- Use **[ ]** instead of **new Array( )**

NOTE: Using the new keyword makes your program slower

```
Example :   let str = "" ;          // new String
            let num = 0 ;           // new primitive number
            let bool = false ;      // new primitive boolean
            const obj = { } ;       // new object
            const arr = [ ] ;       // new array object
```

Use strict equality (===)

The **==** comparison operator always converts (to matching types) before comparison.
The **===** operator compares both values and data types

Example :	0 == "" ; // true
	2 == "2" ; // true
	0 == false ; // true

	0 === "" ; // false
	2 === "2" ; // false
	0 === false; // false

Avoid Using eval( )

The eval( ) function is used to **run text as code**. In almost all cases, it should not be necessary to use it.

Because it allows arbitrary code to be run, it also represents a security problem.