Null vs Undefined

null	has no value, on purpose
undefined	declared, but not defined (is not assign a vaalue)
null type	object
undefined type	undefiend (it s a data type itself)

Most languages have data type for variables. JS also has, but at the time of declaration, there is no type decided.
loose equality(==) - performs type conversion (converts the operands to the same type before making the comparison) ('5' == 5)
strict equality(===) - compares the value and also checks the data type

Function vs Block Scope

function scope (ES5)	hoisting - **var**
block scope (ES6)	creates separate scope - **let + const**
local scope	limited to a function
global scope	accessible for all functions

HOISITNG - A process which is happening behind the scene, internally it is bringing the declarations on top (not the assignment)
const - cannot be reassigned
let + const - cannot be redeclarable, **var** can be
The concept of **"block scope"** refers to the visibility domain of variables declared within such a code block. In JS, a variable declared within a code block is **visible only within that block** and not outside of it.

ASI (Automatic Semicolon Insertion)

It's a good practice to have a ; (semicolon) at the end of a line, but it is **optional** because JS compiler inserts a semicolon

"use strict" doesn't change the behaviours, it doesn't force you to put a semicolon on all lines

The purpose of "use strict" in JavaScript is to enable a stricter set of rules for code execution, helping to catch common coding errors and promoting safer and more maintainable cod

function rest() { return ;- undefined because js compiler adds a semicolon
{ a : 5
}

Rest & Spread operator (ES6)

Rest operator	is used in function parameters to deal with an arbitrary number of arguments
Spread operator	spreads elements of **an array** or **object** into individual elements

function restex(...elems){ console.log(-elems)}
let max = Math.max(...arr)

Infinity & -Infinity

Number.NEGATIVE_INFINITY

Number.POSITIVE_INFINITY

isFinite()	to check finite or infinite value.

You got -Infinity or Infinity when a numeric value exceeds the range of 64-bit format.

NaN (Not a Number)

You get this error when there is a **non-numeric value or operation** performed

isNaN()	is going to check whether the value is a number or not
isFinite()	checks for NaN as well as for Infinite values

console.log(NaN === NaN) - **false** - strict equality or non-equality, it is not going to match with NaN because there is always a unique value for NaN.

Arrow functions

this	object does not work with arrow function
arguments	object does not work with arrow function (we can use spread operator ...arg)
new	cannot use it to call arrow function

Arrow functions offer a concise syntax for writing anonymous functions, leading to shorter and more readable code.
if you have one parameter you can avoid writing paranthesis
IIFE – It is a function which gets called automatically (function())()

Currying

sum(5)(6) - sum is calling the parent function, and the second parentheses calls the inner function

Unique way to call **inner functions** where you can pass arguments partially or pass multiple arguments in a function but 1 arument at a time

Solves various purposes like passing partial parameters or avoiding unwanted repetitions (functional programming technique)

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Closure

CLOSURE is useful when you want to make private members available globally when needed.

When a function comes under another function a **"closure"** is created. Closure pattern remembers outer variable & also helps to access outer scope members

```
function outer() {
function inner() {
console.log("inner called..") }}
```

Iterables & Iterators (ES6)

Symbol.iterator	convert an object literal into an interable object
Arrays, Strings, Maps, Sets	iterable
next() method	itr obj will automatically have this method

New mechanism to iterate or traverse through data structures.
console.log(itr.next()) = {value: 4, status: false}

Generators

function*	define a generator function
yeld	pauses the generator + receive input & send outpur
yeld*	recursive function
next()	returns an obj with 2 keys (value + next status - boolean)
return()	terminates generator execution

Generators (cont)

throw()	triggers custom exception.

Generators help you to pause & resume the code. Normally when you write a function, it returns a single value -> generators are kind of function which returns multiple values in phases
next() - it moves the function pointer to the next line from last suspended yield.

Errors

try ... catch()	err.name / err.message
finally()	code is executed at the end
throw new (Error, TypeError, SyntaxError)	generate your own custom error

When the program faces errors, even after validation, it should handle it & notify the user with proper error details.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com