JavaScript Types & Coercion Quick-Reference Cheat Sheet

Data types, typeof quirks, and == vs === behavior — for QA, BA, and IT professionals debugging front-end and integration defects.

1. JavaScript's Data Types

Type	Example	Where You'll See It
String	`"CLM-4471"`	Form inputs, IDs, API text fields
Number	`1499.00`	Calculations, quantities, amounts
Boolean	`true / false`	Flags, toggle states, validation results
Undefined	`undefined`	Declared but never assigned
Null	`null`	Intentionally empty, set by code
Object	`{ id: 1042 }`	API payloads, form state, config
BigInt	`9007199254740993n`	Values beyond safe integer range

2. Loose (==) vs. Strict (===) Comparison

Comparison	Result	Why
`"5" == 5`	true	String coerced to number before comparing
`"5" === 5`	false	No coercion — types differ
`0 == false`	true	Boolean coerced to number (false to 0)
`"" == 0`	true	Empty string coerces to 0
`null == undefined`	true	Special-cased loose equality, nothing else

3. typeof Quirks to Remember

Expression	Returns	Why It Trips People Up
`typeof null`	"object"	Language-level bug kept for backward compatibility
`typeof NaN`	"number"	NaN is technically a number type — use Number.isNaN()
`typeof []`	"object"	Arrays report as objects — use Array.isArray()
`typeof undefined`	"undefined"	Correct — but distinct from null, which means 'set to empty'

4. Five Defect Patterns to Recognize on Sight

Falsy-but-valid values skipping logic
0, empty string, and false are all falsy. A truthy check (if (value)) silently mistreats valid zero or empty input as missing.

String concatenation instead of addition
The + operator concatenates if either operand is a string. A 'sum' that looks like two numbers stuck together is a type bug, not a math bug.

Form inputs assumed to be numbers
Every HTML form field returns a string, regardless of its type attribute. Missing Number() conversion causes silent type errors downstream.

API type mismatches across services
One service returns a number, another returns the same field as a formatted string. Assertions on value alone won't catch this — assert type too.

Floating-point precision in money math
0.1 + 0.2 does not equal 0.3 in JavaScript. Use integer cents or a decimal library for currency calculations, never raw floats.

Edge case: a const-declared object can still have its properties changed. const only locks the variable binding, not the object's contents — user.name = 'Jones' is legal even when user was declared with const.