JavaScript Types & Coercion Cheatsheet

Day 1 — Values, Types, Variables

---

1. Primitive Types

Type	Example	`typeof`	Notes
String	`'Hello'`, `""`	`"string"`	Text data, immutable. Template literals: `` `Hello ${name}` ``
Number	`42`, `3.14`, `NaN`, `Infinity`	`"number"`	Single numeric type for int/float. Beware `NaN` and `-0`.
Boolean	`true`, `false`	`"boolean"`	Used in conditions.
Undefined	`let x;` → `undefined`	`"undefined"`	Declared but not assigned.
Null	`let x = null;`	`"object"`	Historical quirk. Represents intentional emptiness.
Symbol	`Symbol('id')`	`"symbol"`	Unique and immutable identifiers.
BigInt	`123n`, `BigInt(123)`	`"bigint"`	For large integers beyond `Number.MAX_SAFE_INTEGER`.

Quick Checks

```
typeof null; // 'object'
Number.isNaN(NaN); // true
isNaN('foo'); // true (converted to NaN)
Number.isNaN('foo'); // false
```

---

2. Truthy & Falsy Values

Falsy Values:

```
false, 0, -0, 0n, "", null, undefined, NaN
```

Everything else is **Truthy**, even:

```
Boolean('0'); // true
Boolean(' '); // true
Boolean([]);  // true
Boolean({});  // true
```

---

3. Type Conversion (Coercion)

3.1 Explicit Conversion

```
Number('10');   // 10
String(10);     // '10'
Boolean(0);     // false
BigInt('123');  // 123n
```

3.2 Implicit Conversion

Addition (`+`) → string concatenation if one operand is a string. **Other operators** (`-` , `*` , `/` , `**` , `%`) → numeric conversion.

```
'5' + 3;        // '53'
'5' - 3;        // 2
true + 1;       // 2 (true → 1)
false + 1;      // 1 (false → 0)
'10' / 2;       // 5
null + 1;       // 1 (null → 0)
undefined + 1;  // NaN
```

---

4. Equality Operators

Strict Equality (`===`)

No type coercion — safest.

```
0 === false; // false
'5' === 5;   // false
null === undefined; // false
```

Loose Equality (`==`)

Performs type coercion (avoid when possible).

```
0 == false; // true
'' == false; // true
'0' == false; // true
null == undefined; // true
[1] == '1'; // true
[1,2] == '1,2'; // true
```

✅**Best Practice:** Always use `===` and `!==`.

---

5. Variable Declarations

Keyword	Scope	Reassignment	Redeclaration	Notes
`var`	Function	✅	✅	Hoisted, avoid.
`let`	Block	✅	❌	Modern, safe for mutable variables.
`const`	Block	❌	❌	Default choice. Immutable binding.

Example

```
if (true) {
  var a = 1;
  let b = 2;
  const c = 3;
}
console.log(a); // 1
console.log(b); // ReferenceError
console.log(c); // ReferenceError
```

---

6. Hoisting & TDZ (Temporal Dead Zone)

Variable Hoisting

```
console.log(x); // undefined
var x = 5;
```

```
console.log(y); // ReferenceError (TDZ)
let y = 5;
```

Function Hoisting

```
foo(); // works
function foo() { console.log('Hello'); }

bar(); // TypeError
var bar = function() { console.log('Hi'); };
```

---

7. Common Gotchas

```
Number(''); // 0
+true; // 1
+false; // 0
+undefined; // NaN
[] + []; // ''
[] + {}; // '[object Object]'
{} + []; // 0 (depends on context)
0 === -0; // true
Object.is(0, -0); // false
Object.is(NaN, NaN); // true
```

---

8. Best Practices

- Always use `===` and `!==`.
- Prefer `const`, fallback to `let`.
- Avoid `var`.
- Use **explicit** conversions: `Number()`, `String()`, `Boolean()`.
- Use `Number.isNaN()` instead of `isNaN()`.
- Avoid mixing `BigInt` with `Number` in expressions.
- Comment reasoning for type conversions in complex logic.

---

9. Quick Reference Table

Concept	Example	Result
`'5' + 1`	`'51'`	String concatenation

Concept	Example	Result
`'5' - 1`	`4`	Numeric coercion
`true + true`	`2`	Boolean → Number
`null == undefined`	`true`	Loose equality special rule
`[1] == 1`	`true`	Array → String → Number
`[] == 0`	`true`	Array → Number("") → 0
`Boolean('false')`	`true`	Non-empty string is truthy
