
✅ 6. Type Coercion & Type Conversion

👨‍💻 Interview Q&A:

Q: What's the difference between type coercion and type conversion?

> **Type coercion** is when JS **automatically** converts types (e.g. string + number = string).
> **Type conversion** is when **you explicitly** convert types using `String()`, `Number()`, `Boolean()`.

🧪 Example:

```
console.log('5' + 2);    // '52' (coercion to string)
console.log('5' - 2);    // 3   (coercion to number)
console.log(Number('5')) // 5   (conversion)
```

🎯 Analogy:

Type coercion is like autocorrect — JS tries to "guess" what you meant.
Type conversion is you correcting the text manually.

✅ 7. `==` vs `===`

👨‍💻 Interview Q&A:

Q: What's the difference between `==` and `===`?

> `==` checks for **value equality with type coercion**.
> `===` checks for **value and type equality** (strict equality). Always prefer `===` in modern JS.

🧪 Example:

```
console.log(5 == '5');  // true
console.log(5 === '5'); // false
```

🎯 Analogy:

`==` is like saying “we look the same”.
`===` is like checking both ID and appearance — type and value must match.

✅ 8. Truthy & Falsy Values

👨‍💻 Interview Q&A:

Q: What values are falsy in JavaScript?

> JS treats the following as falsy:
> `false`, `0`, `""`, `null`, `undefined`, and `NaN`.
> Everything else is truthy.

🧪 Example:

```
if ("") console.log("This won't run");
if ("hello") console.log("This will run");
```

🎯 Analogy:

Truthy/falsy is like a light switch:
Falsy = switch is OFF (no light)
Truthy = switch is ON (light on)

✅ 9. Function Declarations vs Expressions

👨‍💻 Interview Q&A:

Q: What’s the difference between function declarations and expressions?

> Function **declarations** are hoisted and available before they appear in code.
> Function **expressions** are not hoisted — they exist only after their definition.

🧪 Example:

```
sayHi(); // works
function sayHi() {
  console.log("Hi!");
}

greet(); // Error
const greet = function() {
  console.log("Hello");
};
```

🎯 Analogy:

Function declaration is like someone arriving early at a party — they're already there.
Function expression is like someone showing up only after being invited.

✅ 10. Arrow Functions

👨‍💻 Interview Q&A:

Q: What are arrow functions and how are they different?

> Arrow functions are a shorter syntax for functions and **do not have their own `this`**.
> They inherit `this` from the surrounding lexical context.

🧪 Example:

```
const add = (a, b) => a + b;
console.log(add(2, 3)); // 5
```

🎯 Analogy:

Arrow functions are like remote workers — they don't bring their own desk (`this`), they just use whatever room they're in.

✅ 11. IIFE (Immediately Invoked Function Expression)

👨‍💻 Interview Q&A:

Q: What is an IIFE and why is it used?

> IIFE is a function that runs immediately after it's defined.
> It's often used to **create private scope** or avoid polluting the global scope.

🧪 Example:

```
(function () {
  console.log("I'm an IIFE!");
})();
```

🎯 Analogy:

An IIFE is like a to-go coffee — made and consumed immediately, without storing it for later.

✅ 12. Template Literals

👨‍💻 Interview Q&A:

Q: What are template literals and how do they differ from regular strings?

> Template literals use backticks (`` ` ``) and allow **multi-line strings** and **expression interpolation** using `${}`.

🧪 Example:

```
const name = "Alex";
console.log(`Hello, ${name}!`);
```

🎯 Analogy:

Template literals are like Mad Libs — you insert your own values into blanks dynamically.

✅ 13. Default Parameters

👨‍💻 Interview Q&A:

Q: What are default parameters in functions?

> They allow parameters to have default values if no value or `undefined` is passed.

🧪 Example:

```
function greet(name = "Guest") {
  console.log(`Hello, ${name}`);
}
greet(); // Hello, Guest
```

🎯 Analogy:

Default parameters are like ordering a meal combo — if you don't specify a drink, you get water by default.

✅ 14. typeof Operator

👨‍💻 Interview Q&A:

Q: What does the `typeof` operator do?

> It returns the data type of a value as a string.
> Edge case: `typeof null` returns `"object"` (a long-standing quirk).

🧪 Example:

```
console.log(typeof 42);     // "number"
console.log(typeof "hi");   // "string"
console.log(typeof null);   // "object" ❗
```

🎯 Analogy:

`typeof` is like a label detector — it tells you what kind of item something is. But sometimes it mislabels things (like `null`).

✅ 15. Strict Mode

👨‍💻 Interview Q&A:

Q: What is `"use strict"` and why is it used?

> `"use strict"` enables strict mode in JS.
> It helps catch common bugs like undeclared variables and restricts bad practices.

🧪 Example:

```
"use strict";
x = 5; // ReferenceError: x is not defined
```

🎯 Analogy:

Strict mode is like grammar-check in an editor — it forces you to write cleaner, more correct code.
