

JavaScript Type Coercion & Object Conversion

These notes explain **how JavaScript converts objects and values** during operations like `+`, `-`, `==`, `!`, etc.

1 Object Conversion Methods

JavaScript objects have two important methods:

- ◊ **toString()**

- Used when JS needs a **string**
- Default output:

```
"[object Object]"
```

- ◊ **valueOf()**

- Used when JS needs a **number**
 - Default behavior: returns the object itself
-

2 Custom **toString()** and **valueOf()**

```
let obj = {  
  toString() {  
    return "90";  
  },  
  valueOf() {}  
};  
  
console.log(obj.toString()); // "90"
```

If **valueOf()** returns nothing, JS falls back to **toString()**.

3 **valueOf()** Returning a Number

```
let obj2 = {  
  x: 10,  
  valueOf() {  
    return 10;  
  }  
}
```

```
};

console.log(obj2.valueOf()); // 10
```

4 Default valueOf() Type

```
let obj3 = { x: 10 };

console.log(typeof obj3.valueOf()); // object
console.log(10 - obj3); // NaN
```

Reason: default `valueOf()` returns an object → `toString()` → "[object Object]" → NaN

5 Object → Number Conversion

```
let obj4 = {
  x: 7,
  valueOf() {
    return 90;
  }
};

console.log(100 - obj4); // 10
```

Conversion Flow:

```
object → ToPrimitive (number)
→ valueOf() → 90
→ 100 - 90 = 10
```

6 Invalid `toString()` Return

```
let obj5 = {
  toString() {
    return {};
  }
};

// 100 - obj5 ✗ TypeError
```

Reason: `toString()` must return a **primitive**

7 `toString()` Returning Number String

```
let obj6 = {
  toString() {
    return "88";
  }
};

console.log(100 - obj6); // 12
```

8 String + Object Behavior

```
let newObj = {};

console.log("18" + newObj); // "18[object Object]"
console.log(18 + newObj); // "18[object Object]"
```

Reason: + prefers **string conversion**

9 ToBoolean Conversion

```
console.log(!10); // false
```

Truthy values include:

- numbers except 0
 - objects
 - arrays
-

10 `Nan` Comparisons

```
NaN == NaN      // false
NaN === NaN    // false
'NaN' == NaN   // false
```

Rule: **NaN is never equal to anything (even itself)**

1 | 1 Object Comparison (`==`)

```
let t = {
  valueOf() {
    return 100;
  }
};

99 == t // false
100 == t // true
```

Because object → number → 100

1 | 2 Object Reference Comparison

```
let z = { x: 10 };
let m = { x: 10 };

z == m // false
z == z // true
```

Objects compare by **reference**, not value

1 | 3 String Conversion Tricks

```
"" + 0      // "0"
"" + -0     // "0"
"" + []     // ""
"" + {}     // "[object Object]"
"" + [1,2]   // "1,2"
"" + [null]  // ""
"" + [1,2,null,4] // "1,2,,4"
```

Arrays call `join('')`

1 | 4 ToNumber Conversion

```
0 - "010"    // -10 (string → decimal)
0 - "010"    // NaN
0 - 010      // -8 (octal literal)
0 - "0xb"    // -11 (hexadecimal)
```

1 | 5 Array to Number

```
[] - 1      // -1  
[""] - 1    // -1  
["0"] - 1   // -1
```

Reason:

```
[] → "" → 0  
["0"] → "0" → 0
```

🍪 Final Rule Summary

Operator	Conversion
+	String preferred
- * /	Number
==	Type coercion
====	No coercion
!	Boolean

Perfect for **Notion / Markdown / Interview prep**