



MEGA ESTUDO – JAVASCRIPT NUMBERS



1. Tipo Number no JavaScript

- **JavaScript tem apenas UM tipo de número:** `Number`
- Não existe `int`, `float`, `double` como em outras linguagens.
- Todo número é **64-bit Floating Point (IEEE 754)**.

```
let a = 10;      // inteiro
let b = 3.14;    // decimal
```

✓ Ambos são do tipo `number`.



2. Notação Científica

Usada para números muito grandes ou muito pequenos:

```
let x = 123e5;    // 123000000
let y = 123e-5;   // 0.00123
```



3. Como os números são armazenados

- 64 bits no total:
 - 52 bits → fração (mantissa)
 - 11 bits → expoente
 - 1 bit → sinal (+ ou -)

✦ **Consequência:** alguns cálculos decimais não são exatos.



4. Precisão dos Números

◆ Inteiros

- Precisos até **15 dígitos**

```
999999999999999 // OK
999999999999999 // ERRO (perde precisão)
```

◆ Decimais (Problema clássico)

```
0.2 + 0.1 // 0.30000000000000004 🤖
```

✓ Solução:

```
(0.2 * 10 + 0.1 * 10) / 10
```

+ 5. Soma de Números e Strings (MUITO IMPORTANTE)

+ Regra do operador +

- Número + Número → soma
- String + qualquer coisa → concatenação

```
10 + 20          // 30
"10" + "20"      // "1020"
10 + "20"        // "1020"
```

✗ Erro comum:

```
"The result is: " + 10 + 20
// "The result is: 1020"
```

✓ Correto:

```
"The result is: " + (10 + 20)
```

6. Strings Numéricas

Strings com números dentro:

```
"100"
"10.5"
```

Funcionam com:

```
"100" / "10"    // 10
"100" * "10"    // 1000
"100" - "10"    // 90
```

✗ Não funciona:

```
"100" + "10"    // "10010"
```

? 7. NaN (Not a Number)

- Representa um valor inválido numérico
- NaN ainda é do tipo number

```
typeof NaN // "number"
100 / "Apple" // NaN
```

Verificar NaN:

```
isNaN(x)
Number.isNaN(x) // forma correta
```

✖ NaN contamina cálculos:

8. Infinity

- Valor muito grande ou divisão por zero

```
1 / 0    // Infinity
-1 / 0   // -Infinity
typeof Infinity // "number"
```

9. Bases Numéricas

Hexadecimal:

```
0xFF // 255
```

⚠ Nunca use número com zero à esquerda (07).

Converter bases:

```
let n = 32;

n.toString(2); // binário
n.toString(8); // octal
n.toString(16); // hexadecimal
```

10. Number como Objeto (NÃO USAR)

```
let x = new Number(5);
```

✗ Problemas:

```
x === 5 // false
```

✓ Use sempre:

```
let x = 5;
```

11. Métodos de Números

◆ Métodos básicos

```
toString()
toFixed()
toPrecision()
toExponential()
valueOf()
```

Exemplos:

```
let x = 9.656;

x.toFixed(2);          // "9.66"
x.toPrecision(3);      // "9.66"
x.toExponential(2);    // "9.66e+0"
```

📌 toFixed(2) → ideal para **dinheiro** 💰

12. Converter para Número

Métodos globais:

```
Number()
parseInt()
parseFloat()
```

Exemplos:

```
Number("10")          // 10
parseInt("10.9")       // 10
parseFloat("10.9")    // 10.9
```

✗ Se não converter:

```
Number("John") // NaN
```

13. Safe Integers

- Intervalo seguro:

```
-(2^53 - 1) até (2^53 - 1)
Number.isSafeInteger(10); // true
```

✗ Fora do limite:

```
Number.isSafeInteger(9007199254740992); // false
```

14. Operadores Bitwise

⚠️ Trabalham com **32 bits**

Operador	Nome
&	AND
	OR
^	XOR
~	NOT
<<	Shift Left
>>	Shift Right
>>>	Shift Right sem sinal

Exemplo:

```
5 & 1    // 1
5 | 1    // 5
5 ^ 1    // 4
~5       // -6
```



15. Binário e Dois Complementos

- Números negativos usam **two's complement**
 - ~5 não é 10 → é **-6**
-



16. Converter Decimal ↔ Binário

```
function dec2bin(dec) {
  return (dec >>> 0).toString(2);
}
```

```
function bin2dec(bin) {
  return parseInt(bin, 2);
}
```



RESUMÃO FINAL (PARA PROVA)

- ✓ JS só tem **Number**
- ✓ Problemas com decimais são normais
- ✓ + concatena strings
- ✓ NaN é number
- ✓ Não use `new Number()`
- ✓ Use `Number.isNaN()`
- ✓ Bitwise usa 32 bits