Curso Superior de Tecnologia em Sistemas para Internet
Desenvolvimento front-end I (front 1)
Rafael Cunha Cardoso
rafaelcardoso@ifsul.edu.br

Operadores em JavaScript

INSTITUTO FEDERAL
Sul-rio-grandense
Câmpus Pelotas

Operadores em JavaScript

- Aritméticos
  - Atribuição
    - Relacionais
      - Lógicos

Operadores Aritméticos

Operadores Aritméticos

Operador	Operação	Precedência
**	Potência	1ª
*	Multiplicação	2ª
/	Divisão	2ª
%	Resto da divisão	2ª
+	Adição	3ª
-	Subtração	3ª

Expressões	Resultado?
`a = 5 + 2 * 3;`	?

Operadores Aritméticos

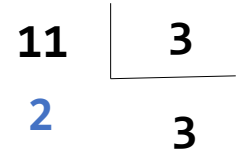
Operador	Operação	Precedência
**	Potência	1ª
*	Multiplicação	2ª
/	Divisão	2ª
%	Resto da divisão	2ª
+	Adição	3ª
-	Subtração	3ª

Expressões	Resultado?
`a = 5 + 2 * 3;`	11
`b = (5 + 2) * 3;`	21

Operadores Aritméticos

Operador	Operação	Precedência
**	Potência	1ª
*	Multiplicação	2ª
/	Divisão	2ª
%	Resto da divisão	2ª
+	Adição	3ª
-	Subtração	3ª

Expressões	Resultado?
`a = 5 + 2 * 3;`	11
`b = (5 + 2) * 3;`	21
`c = 11 % 3;`	2



OBS: O operador % deve ser utilizado apenas com operandos inteiros (int)

Operadores Aritméticos

Operador	Operação	Precedência
**	Potência	1ª
*	Multiplicação	2ª
/	Divisão	2ª
%	Resto da divisão	2ª
+	Adição	3ª
-	Subtração	3ª

Expressões	Resultado?
`a = 5 + 2 * 3;`	11
`b = (5 + 2) * 3;`	21
`c = 11 % 3;`	2
`d =((5-3)*4+6)/2;`	?

```
((5-3)*4 + 6)/2;
(2*4 + 6)/2;
14/2;
```

Operadores Aritméticos

Operador	Operação	Precedência
**	Potência	1ª
*	Multiplicação	2ª
/	Divisão	2ª
%	Resto da divisão	2ª
+	Adição	3ª
-	Subtração	3ª

Expressões	Resultado?
`a = 5 + 2 * 3;`	11
`b = (5 + 2) * 3;`	21
`c = 11 % 3;`	2
`d =((5-3)*4+6)/2;`	?

```
((5-3)*4 + 6)/2;

(2*4 + 6)/2;

 14/2;

 7
```

Operadores de Atribuição

Auto atribuições

```
var a = 5;
a = a + 4;
a = a - 3;
a = a * 5;
a = a / 2;
a = a ** 2;
a = a % 3;
```

Abreviando...

```
var a = 5;
a+=4;
a-=5;
a*=5;
a/=2;
a**= 2;
a%= 3;
```

Incremento - Decremento

```
var x = 5;
```

	Reduzindo	Pós-in/decremento	Pré-in/decremento
`x = x + 1;`	`x+=1;`	`x++;`	`++x;`
`x = x - 1;`	`x-=1;`	`x--;`	`--x;`

Pré-in/decremento

- **Descrição**: O operador ++ é precedido pelo nome da variável (pré-incremento). Significa que o valor da variável é incrementado antes de ser usado na expressão.
- **Comportamento**: O valor da variável é aumentado em 1 primeiro e, em seguida, o novo valor é retornado ou utilizado na expressão.

```
<script type="JavaScript">
    let x = 5;
    let y = ++x; // x é incrementado para 6 antes de ser atribuído a y

    console.log(x); // 6
    console.log(y); // 6
</script>
```

Pós-in/decremento

- **Descrição**: O operador ++ é seguido pelo nome da variável (pós-incremento). Significa que o valor da variável é usado na expressão primeiro e, em seguida, o valor da variável é incrementado.
- **Comportamento**: O valor da variável é retornado ou utilizado na expressão antes de ser aumentado em 1.

```
<script type="JavaScript">
  let x = 5;
  let y = x++; // o valor de x (5) é atribuído a y antes de x ser incrementado

  console.log(x); // 6
  console.log(y); // 5
</script>
```

Operadores Relacionais

Operadores Relacionais

Operador	Operação
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
==	Igual
!=	Diferente

O cálculo de expressões relacionais resultam sempre em valores **booleanos**.

`true` `false`

Igualdade x Igualdade estrita

```
3 == 3
```
`true`

No JS o operador de igualdade **não testa o TIPO**.

```
3 == '3'
```
`true`

Apesar de serem valores de tipos diferentes, eles possuem a mesma **grandeza**. Assim, a expressão é verdadeira (`true`)

A coerção de tipo é o processo de converter o valor de um tipo em outro (como uma string em um número, um objeto para um booleano...).

JS faz isso implicitamente.
Pode gerar efeitos colaterais no código.

Operador de IDENTIDADE (Igualdade Restrita)

```
3 === '3'
```
`false`

```
3 === 3
```
`true`

Testa se dois valores são idênticos, ou seja, tem o **mesmo valor** e o **mesmo tipo**.

Diferença x Diferença estrita

```
3 != 3
```

`false`

Assim como o operador igualdade, o diferente **não testa o TIPO**.

```
3 != '3'
```

`false`

Apesar de serem valores de tipos diferentes, eles possuem a mesma **grandeza**.

Operador de Diferença Estrita

```
3 !== '3'
```

`true`

Testa se dois valores são **estritamente diferentes**.

```
3 !== 3
```

`false`

Operadores Lógicos

Operadores Lógicos

Operador	Nome
!	Negação
&&	Conjunção
||	Disjunção

Ordem de resolução dos operadores lógicos em uma expressão (sem parênteses):

Negação → Conjunção → Disjunção

Operadores Lógicos

Tabela verdade do operador **E**

A	B	A E B
V	V	V
V	F	F
F	V	F
F	F	F

É verdade **APENAS** quando os dois operandos tiverem valor **verdadeiro**.

Tabela verdade do operador **OU**

A	B	A OU B
V	V	V
V	F	V
F	V	V
F	F	F

Para ser **verdade**, basta que **um dos operandos seja verdadeiro**.

Tabela verdade do operador **NÃO**

A	NÃO A
V	F
F	V

Falsy values

Em JavaScript, "*falsy values*" são valores que são tratados como "falsos" quando avaliados em contextos booleanos.

Isso significa que, quando você usa um valor falsy em uma expressão booleana, ele é considerado como false.

São úteis em construções condicionais, como declarações if, while, e em operadores lógicos.

- `false` >> O valor booleano falso.
- `0` >> O número zero (0).
- `-0` >> O valor negativo zero.
- `0n` >> O zero BigInt.
- `''` >> Uma string vazia.
- `null` >> O valor nulo.
- `undefined` >> O valor indefinido.
- `NaN:` >> frequentemente o resultado de operações matemáticas inválidas.

Qualquer outro valor em JavaScript é considerado "*truthy*," ou seja, é avaliado como true em um contexto booleano.

Valores "truthy" incluem números diferentes de zero, strings não vazias, objetos, arrays, funções e assim por diante.

Avaliação de curto circuito

Podem ser usadas para tornar o código mais performático.

false && true -> false

false || true -> true

JavaScript define FALSY VALUES, ou seja, valores que resultam em falso em uma expressão lógica.

FALSY VALUES

- **false**
- **0**
- **'' "" ``**
- **null /undefined**
- **Nan**

Qualquer coisa diferente destes valores, resulta em true, em JS.

```
// JS retorna sempre o último valor avaliado.

console.log('TSI' && true && 'IFSul’);
>> IFSul

console.log('TSI' && 0 && 'IFSul’);
>> 0
```

```
// Permite construir instruções lógicas, de forma mais simples.

const corUsuario = null;
const corPadrao = corUsuario || 'preto';

console.log(corPadrao)
>> preto
```

Precedência expressões

1) Operadores **Aritméticos**
   - Obedecendo as suas regras de precedência.

2) Operadores **Relacionais**
   - Não tem ordem de precedência, são resolvidos conforme aparecem, da esquerda para a direita.

3) Operadores **Lógicos**
   - Na ordem ! (não), && (conjunção), || (disjunção).

Curso Superior de Tecnologia em Sistemas para Internet
Desenvolvimento front-end I (front 1)
Rafael Cunha Cardoso
rafaelcardoso@ifsul.edu.br

Operadores em JavaScript

INSTITUTO FEDERAL
Sul-rio-grandense
Câmpus Pelotas