Lógica computacional

JavaScript

Operadores de atribuição

Um operador de atribuição atribui um valor ao operando à sua esquerda baseado no valor do operando à direita. O operador de atribuição básica é o igual (=), que atribui o valor do operando à direita ao operando à esquerda. Isto é, x = y atribui o valor de y a x.

```
var x = 3; // x contém o valor 3
var y = 4; // y contém o valor 4
x = y; // x agora contém o mesmo valor de y, 4
```

Mas existem alguns tipos mais complexos, que fornecem atalhos úteis para manter seu código mais puro e mais eficiente. Os mais comuns estão listados abaixo:

Operator	Name	Purpose	Example	Shortcut for
`+=`	Atribuição de adição	Adiciona o valor à direita ao valor da variável à esquerda e, em seguida, retorna o novo valor da variável	`x = 3;` `x += 4;`	`x = 3;` `x = x + 4;`
`-=`	Atribuição de subtração	Subtrai o valor à direita do valor da variável à esquerda e retorna o novo valor da variável	`x = 6;` `x -= 3;`	`x = 6;` `x = x - 3;`
`*=`	Atribuição de multiplicação	Multiplica o valor da variável à esquerda pelo valor à direita e retorna o novo valor da variável	`x = 2;` `x *= 3;`	`x = 2;` `x = x * 3;`
`/=`	Atribuição de divisão	Divide o valor da variável à esquerda pelo valor à direita e retorna o novo valor da variável	`x = 10;` `x /= 5;`	`x = 10;` `x = x / 5;`

Operadores de aritméticos

São os operadores que utilizamos para fazer as operações aritiméticas básicas:

Operador	Nome	Propósito	Exemplo
`+`	Adição	Adiciona um número a outro.	`6 + 9`
`-`	Subtração	Subtrai o número da direita do número da esquerda.	`20 - 15`
`*`	Multiplicação	Multiplica um número pelo outro.	`3 * 7`
`/`	Divisão	Divide o número da esquerda pelo número da direita.	`10 / 5`
`%`	Restante (*Remainder* - as vezes chamado de modulo)	Retorna o resto da divisão em números inteiros do número da esquerda pelo número da direita.	`8 % 3` (retorna 2; como três cabe duas vezes em 8, deixando 2 como resto.)

Operadores de comparação

Às vezes, queremos executar testes verdadeiro / falso e, em seguida, agir de acordo, dependendo do resultado desse teste, para fazer isso, usamos **operadores de comparação**.

Operator	Name	Purpose	Example
`===`	Igualdade estrita	Verifica se o valor da esquerda e o da direita são idênticos entre si.	`5 === 2 + 4`
`!==`	Não-igualdade-estrita	Verifica se o valor da esquerda e o da direita **não** são idênticos entre si.	`5 !== 2 + 3`
`<`	Menor que	Verifica se o valor da esquerda é menor que o valor da direita.	`10 < 6`
`>`	Maior que	Verifica se o valor da esquerda é maior que o valor da direita.	`10 > 20`
`<=`	Menor ou igual que	Verifica se o valor da esquerda é menor que ou igual ao valor da direita.	`3 <= 2`
`>=`	Maior ou igual que	Verifica se o valor da esquerda é maior que ou igual ao valor da direita.	`5 >= 4`

Nota

Talvez você já tenha visto alguém usando == e!= a fim de testar igualdade ou desigualdade em JavaScript. Estes são operadores válidos em JavaScript, mas que diferem de ===/!==. As versões anteriores testam se os valores são os mesmos, mas não se os tipos de dados dos valores são os mesmos. As últimas versões estritas testam a igualdade de ambos os valores e seus tipos de dados. Elas tendem a resultar em menos erros, por isso recomendamos que você as use.

Operadores lógicos

Operadores lógicos são utilizados tipicamente com valores booleanos (lógicos); neste caso, retornam um valor booleano. Entretanto, os operadores && e || na verdade retornam o valor de um dos operandos especificados, de forma que se esses operadores forem utilizados com valores não-booleanos, eles possam retornar um valor não-booleano. Os operadores lógicos são descritos na seguinte tabela.

Operadores lógico

Operador	Utilização	Descrição
`AND lógico (&&)`	`expr1 && expr2`	(E lógico) - Retorna `expr1` caso possa ser convertido para falso; senão, retorna `expr2`. Assim, quando utilizado com valores booleanos, `&&` retorna verdadeiro caso ambos operandos sejam verdadeiros; caso contrário, retorna falso.
`OU lógico (||)`	`expr1 || expr2`	(OU lógico) - Retorna `expr1` caso possa ser convertido para verdadeiro; senão, retorna `expr2`. Assim, quando utilizado com valores booleanos, `||` retorna verdadeiro caso ambos os operandos sejam verdadeiro; se ambos forem falsos, retorna falso.
`NOT lógico (!)`	`!expr`	(Negação lógica) Retorna falso caso o único operando possa ser convertido para verdadeiro; senão, retorna verdadeiro.

O código a seguir mostra exemplos do operador && (E lógico).

```
var a1 =  true && true;        // t && t retorna true
var a2 =  true && false;       // t && f retorna false
var a3 = false && true;        // f && t retorna false
var a4 = false && (3 == 4); // f && f retorna false
var a5 = "Gato" && "Cão";   // t && t retorna Cão
var a6 = false && "Gato";   // f && t retorna false
var a7 = "Gato" && false;   // t && f retorna false
```

O código a seguir mostra exemplos do operador || (OU lógico).

```
var o1 =  true || true;    // t || t retorna true
var o2 = false || true;    // f || t retorna true
var o3 =  true || false;   // t || f retorna true
var o4 = false || (3 == 4); // f || f retorna false
var o5 = "Gato" || "Cão";    // t || t retorna Gato
var o6 = false || "Gato";    // f || t retorna Gato
var o7 = "Gato" || false;    // t || f retorna Gato
```

O código a seguir mostra exemplos do operador ! (negação lógica).

```
var n1 = !true;   // !t retorna false
var n2 = !false;  // !f retorna true
var n3 = !"Gato"; // !t retorna false
```

Avaliação de curto-circuito

Como expressões lógicas são avaliadas da esquerda para a direita, elas são testadas como possíveis avaliações de "curto-circuito" utilizando as seguintes regras:

- false && qualquer coisa é avaliada em curto-circuito como falso.
- true || qualquer coisa é avaliada em curto-circuito como verdadeiro.

As regras de lógica garantem que estas avaliações estejam sempre corretas. Repare que a parte qualquer das expressões acima não é avaliada, de forma que qualquer efeito colateral de fazê-lo não produz efeito algum.

Operadores de incremento e decremento

Às vezes você deseja adicionar ou subtrair, repetidamente, um valor de uma variável numérica. Convenientemente isto pode ser feito usando os operadores incremento ++ e decremento --

contagem++;

Nota

Eles são mais comumente usados em Laços e interações, que será visto no curso mais tarde. Por exemplo, digamos que você queira passar por uma lista de preços e adicionar imposto sobre vendas a cada um deles. Você usaria um loop para passar por cada valor e fazer o cálculo necessário para adicionar o imposto sobre vendas em cada caso. O incrementador é usado para mover para o próximo valor quando necessário.

Vamos tentar brincar com eles no seu console. Para começar, note que você não pode aplicá-las diretamente a um número, o que pode parecer estranho, mas estamos atribuindo à variável um novo valor atualizado, não operando no próprio valor. O seguinte retornará um erro:

```
3++;
```

Então, você só pode incrementar uma variável existente. Tente isto:

```
var num1 = 4;
num1++;
```

Acontece a mesma coisa com -- : tente o seguinte

```
var num2 = 6;
num2--;
num2;
```

Nota

Você pode fazer o navegador fazer o contrário - incrementar/decrementar a variável e depois retornar o valor, colocando o operador no início da variável ao invés do final. Tente os exemplos acima novamente, mas desta vez use ++num1 e --num2.

Condicionais

Em qualquer linguagem de programação, o código precisa tomar decisões e realizar ações de acordo, dependendo de diferentes entradas. Por exemplo, em um jogo, se o número de vidas do jogador é 0, então o jogo acaba. Em um aplicativo de clima, se estiver sendo observado pela manhã, ele mostra um gráfico do nascer do sol; mostra estrelas e uma lua se for noite. Neste artigo, exploraremos como as chamadas declarações condicionais funcionam em JavaScript.

Você pode tê-lo em uma condição ...!

Seres humanos (e outros animais) tomam decisões o tempo todo que afetam suas vidas, desde pequenas ("devo comer um biscoito ou dois?") até grandes ("devo ficar no meu país de origem e trabalhar na fazenda do meu pai ou devo mudar para a América e estudar astrofísica?").

As declarações condicionais nos permitem representar tomadas de decisão como estas em JavaScript, a partir da escolha que deve ser feita (por exemplo, "um biscoito ou dois"), ao resultado obtido dessas escolhas (talvez o resultado de "comer um biscoito" possa ser "ainda sentido fome ", e o resultado de "comer dois biscoitos" pode ser "ter se sentido cheio”.

Declarações if ... else

```
if (condição) {
  código para executar caso a condição seja verdadeira
} else {
  senão, executar este código
}
```

Aqui nós temos:

1. A palavra reservada if seguida de um par de parênteses.
2. Um teste condicional, localizado dentro dos parênteses (normalmente "este valor é maior que esse", ou "este valor existe"). Esta condição pode fazer uso dos operadores de comparação que discutimos no último módulo, e podem retornar true ou false.
3. Um par de chaves, e dentro dele temos código — pode ser qualquer código que queiramos, e só vai ser executado se o teste condicional retornar true.
4. A palavra reservada else.
5. Outro par de chaves, dentro dele, temos mais um pouco de código — pode ser qualquer código que queiramos, e só vai executar se o teste condicional retornar um valor diferente de true, neste caso not true, ou false.

Este tipo de código é bem legível por seres humanos — ele diz: "if a condição for true, execute o bloco de código A, else execute o bloco de código B" (se a condição for verdadeira, execute o bloco de código A, senão execute o bloco de código B).

Você precisa saber que não é obrigado a colocar a palavra reservada else e o segundo bloco de par de chaves. O código apresentado a seguir é perfeitamente válido e não produz erros:

```
if (condição) {
  código para executar se a condição for verdadeira
}

código a ser executado
```

Entretanto, você precisa ser cauteloso aqui — neste caso, repare que o segundo bloco de código não é controlado pela declaração condicional, então ele vai executar sempre, independente do teste condicional retornar true ou false. É claro, isto não é necessariamente uma coisa ruim, mas isso pode não ser o que você quer — com muita frequência você vai querer executar ou um bloco de código ou outro, não os dois juntos.

Fontes:

https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/Math
https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/conditionals
https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators