 This page was translated from English by the community. Learn more and join the MDN Web Docs community.

Expressões e operadores

Este artigo descreve expressões e operadores de JavaScript, incluindo operadores de atribuição, comparação, aritméticos, bit a bit, lógicos, de strings e especiais.

Operadores

O JavaScript possui os tipos de operadores a seguir. Esta seção descreve os operadores e contém informações sobre precedência de operadores.

- Operadores de atribuição
- Operadores de comparação
- Operadores aritméticos
- Operadores bit a bit
- Operadores lógicos
- Operadores de string
- Operador condicional (ternário)
- Operador vírgula
- Operadores unário
- Operadores relacionais

O JavaScript possui tanto operadores *binários* quanto *unários* e um operador ternário, o operador condicional. Um operador binário exige dois operandos, um antes do operador e outro depois:

```
operando1 operador operando2
```

Por exemplo, `3+4` ou `x*y`.

Um operador unário exige um único operando, seja antes ou depois do operador:

`operador operando`

ou

`operando operador`

Por exemplo, `x++` ou `++x`.

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 básico 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`.

Os outros operadores de atribuição são encurtamentos de operadores padrão, como mostrado na tabela a seguir.

Operadores de atribuição composto

Nome	Operador encurtado	Significado
Atribuição	<code>x = y</code>	<code>x = y</code>
Atribuição de adição	<code>x += y</code>	<code>x = x + y</code>
Atribuição de subtração	<code>x -= y</code>	<code>x = x - y</code>
Atribuição de multiplicação	<code>x *= y</code>	<code>x = x * y</code>
Atribuição de divisão	<code>x /= y</code>	<code>x = x / y</code>
Atribuição de resto	<code>x %= y</code>	<code>x = x % y</code>

Nome	Operador encurtado	Significado
Atribuição exponencial		
Atribuição bit-a-bit por deslocamento á esquerda	x <<= y	x = x << y
Atribuição bit-a-bit por deslocamento á direita	x >>= y	x = x >> y

Atribuição de bit-a-bit deslocamento á direita não assinado	x >>>= y	x = x >>> y
Atribuição AND bit-a-bit	x &= y	x = x & y
Atribuição XOR bit-a-bit	x ^= y	x = x ^ y
Atribuição OR bit-a-bit	x = y	x = x y

Operadores de comparação

This seems to me kind of poorly explained, mostly the difference between "==" and "==="...

Um operador de comparação compara seus operandos e retorna um valor lógico baseado em se a comparação é verdadeira. Os operandos podem ser numéricos, strings, lógicos ou objetos. Strings são comparadas com base em ordenação lexográfica utilizando valores Unicode. Na maioria dos casos, se dois operandos não são do mesmo tipo, o JavaScript tenta convertê-los para um tipo apropriado. Isto geralmente resulta na realização de uma comparação numérica. As únicas exceções a esta regra são os operadores `===` e o `!==`, que realizam comparações de igualdade e desigualdade "estritas". Estes operadores não tentam converter os operandos em tipos compatíveis antes de verificar a igualdade. A tabela a seguir descreve os operadores de comparação levando em conta o seguinte código:

```
var var1 = 3;  
var var2 = 4;
```

Operadores de comparação

Operador	Descrição	Exemplos que retornam verdadeiro
----------	-----------	----------------------------------

Igual (==)	Retorna verdadeiro caso os operandos sejam iguais.	3 == var1 "3" == var1 3 == '3'
Não igual (!=)	Retorna verdadeiro caso os operandos não sejam iguais.	var1 != 4 var2 != "3"
Estritamente igual (===)	Retorna verdadeiro caso os operandos sejam iguais e do mesmo tipo. Veja também Object.is e igualdade em JS (en-US) .	3 === var1
Estritamente não igual (!==)	Retorna verdadeiro caso os operandos não sejam iguais e/ou não sejam do mesmo tipo.	var1 !== "3" 3 !== '3'
Maior que (>)	Retorna verdadeiro caso o operando da esquerda seja maior que o da direita.	var2 > var1 "12" > 2
Maior que ou igual (>=)	Retorna verdadeiro caso o operando da esquerda seja maior ou igual ao da direita.	var2 >= var1 var1 >= 3
Menor que (<)	Retorna verdadeiro caso o operando da esquerda seja menor que o da direita.	var1 < var2 "12" < "2"
Menor que ou igual (<=)	Retorna verdadeiro caso o operando da esquerda seja menor ou igual ao da direita.	var1 <= var2 var2 <= 5

Nota: (\Rightarrow) não é um operador, mas a notação para função de seta

Operadores aritméticos

Operadores aritméticos tomam valores numéricos (sejam literais ou variáveis) como seus operandos e retornam um único valor numérico. Os operadores aritméticos padrão são os de soma (+), subtração (-), multiplicação (*) e divisão (/). Estes operadores trabalham da mesma forma como na maioria das linguagens de programação quando utilizados com números de ponto flutuante (em particular, repare que divisão por zero produz um [NaN \(en-US\)](#)). Por exemplo:

```
console.log(1 / 2); /* imprime 0.5 */  
console.log(1 / 2 == 1.0 / 2.0); /* isto também é verdadeiro */
```




Em complemento às operações aritméticas padrões (+, -, *, /), o JavaScript disponibiliza os operadores aritméticos listados na tabela a seguir.

Operadores aritméticos

Operador	Descrição	Exemplo
Módulo (%)	Operador binário. Retorna o inteiro restante da divisão dos dois operandos.	12 % 5 retorna 2.
Incremento (++)	Operador unário. Adiciona um ao seu operando. Se usado como operador prefixado (++x), retorna o valor de seu operando após a adição. Se usado como operador pósfixado (x++), retorna o valor de seu operando antes da adição.	Se x é 3, então ++x define x como 4 e retorna 4, enquanto x++ retorna 3 e, somente então, define x como 4.

Operador	Descrição	Exemplo
----------	-----------	---------

Decremento (--)	Operador unário. Subtrai um de seu operando. O valor de retorno é análogo àquele do operador de incremento.	Se x é 3, então --x define x como 2 e retorna 2, enquanto x-- retorna 3 e, somente então, define x como 2.
Negação (-)	Operador unário. Retorna a negação de seu operando.	Se x é 3, então -x retorna -3.
Adição (+)	Operador unário. Tenta converter o operando em um número, sempre que possível.	+ "3" retorna 3. +true retorna 1.
Operador de exponenciação (**) 	Calcula a base elevada á potência do expoente, que é, base ^{expoente}	2 ** 3 retorna 8. 10 ** -1 retorna 0.1

Operadores bit a bit tratam seus operandos como um conjunto de 32 bits (zeros e uns), em vez de tratá-los como números decimais, hexadecimais ou octais. Por exemplo, o número decimal nove possui uma representação binária 1001. Operadores bit a bit realizam suas operações nestas representações, mas retornam valores numéricos padrões do JavaScript.

A tabela a seguir resume os operadores bit a bit do JavaScript.

Operadores bit a bit

Operador	Expressão	Descrição
AND	a & b	Retorna um 1 para cada posição em que os bits da posição correspondente de ambos operandos sejam uns.
OR	a b	Retorna um 0 para cada posição em que os bits da posição correspondente de ambos os operandos sejam zeros.
XOR	a ^ b	Retorna um 0 para cada posição em que os bits da posição correspondente são os mesmos. [Retorna um 1 para cada posição em que os bits da posição correspondente sejam diferentes.]
NOT	~ a	Inverte os bits do operando.
Deslocamento à esquerda	a << b	Desloca a em representação binária b bits à esquerda, preenchendo com zeros à direita.
Deslocamento à direita com propagação de sinal	a >> b	Desloca a em representação binária b bits à direita, descartando bits excedentes.
Deslocamento à direita com preenchimento zero	a >>> b	Desloca a em representação binária b bits à direita, descartando bits excedentes e preenchendo com zeros à esquerda.

Operadores bit a bit lógicos

Conceitualmente, os operadores bit a bit lógicos funcionam da seguinte maneira:

- Os operandos são convertidos em inteiros de 32 bits e expressos como uma série de bits (zeros e uns). Números com representação maior que 32 bits terão seus bits truncados. Por exemplo, o seguinte inteiro tem representação binária maior que 32 bits será convertido em um inteiro de 32 bits.

Antes: 111001101111101000000000000000110000000000001

Depois: 10100000000000000011000000000001

- Cada bit do primeiro operando é pareado com o bit correspondente do segundo operando: primeiro bit com primeiro bit, segundo bit com segundo bit e assim por diante.
- O operador é aplicado a cada par de bits e o resultado é construído bit a bit.

Por exemplo, a representação binária de nove é 1001 e a representação binária de quinze é 1111. Desta forma, quando operadores bit a bit são aplicados a estes valores, os resultados são como se segue:

Exemplo de operação bit a bit

Expressão	Resultado	Descrição binária
15 & 9	9	1111 & 1001 = 1001
15 9	15	1111 1001 = 1111
15 ^ 9	6	1111 ^ 1001 = 0110
~15	-16	~ 00000000... 00001111 = 1111 1111 ... 11110000
~9	-10	~ 00000000 ... 0000 1001 = 1111 1111 ... 1111 0110

Nota: No quadro acima perceba que todos os 32 bits são invertidos quando usa-se o operador bit a bit NOT, e que os bits mais significativos (extrema esquerda) são definidos com 1 que representam valores negativos (representação complemento de dois).

Operadores de deslocamento bit a bit

Os operadores de deslocamento bit a bit possui dois operandos: o primeiro é uma quantidade a ser deslocada e o segundo especifica o número de posições binárias as quais o primeiro operando deverá ser deslocado. A direção da operação de deslocamento é controlada pelo operador utilizado.

Operadores de deslocamento convertem seus operandos em inteiros de 32 bits e retornam um resultado do tipo do operando à esquerda.

Os operadores de deslocamento são listados na tabela a seguir.

Operadores bit a bit de deslocamento

Operador	Descrição	Exemplo
Deslocamento à esquerda (<<)	Este operador desloca o primeiro operando pelo número especificado de bits à esquerda. Bits excedentes deslocados para fora do limite à esquerda são descartados. Bits zero são inseridos à direita.	9<<2 produz 36 porque 1001 deslocado 2 bits à esquerda se torna 100100, que é 36.
Deslocamento à direita com propagação de sinal (>>)	Este operador desloca o primeiro operando pelo número especificado de bits à direita. Bits excedentes deslocados para fora do limite à	9>>2 produz 2 porque 1001 deslocado 2 bits à direita se torna 10, que é 2. De forma similar, -9>>2

Operador	direita são descartados. Descrição Cópia dos bits mais à esquerda são deslocadas a partir da esquerda.	produz -3 porque o sinal é preservado. Exemplo

Deslocamento à direita com preenchimento zero (>>>)	Este operador desloca o primeiro operando pelo número especificado de bits à direita. Bits excedentes deslocados para fora do limite à direita são descartados. Bits zero são inseridos à esquerda.	19>>>2 produz 4 porque 10011 deslocado 2 bits à direita se torna 100, que é 4. Para números não negativos o deslocamento à direita com propagação de sinal e o deslocamento à direita com preenchimento zero produzem o mesmo resultado.
--	---	--

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.

Operador	Utilização	Descrição
		&& 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.

Exemplos de expressões que podem ser convertidas para falso são aquelas que são avaliados como nulo, 0, string vazia ("") ou undefined .

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` && *qualquercoisa* é avaliado em curto-circuito como falso.
- `true` || *qualquercoisa* é avaliado em curto-circuito como verdadeiro.

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

Operadores de string

Além dos operadores de comparação, que podem ser utilizados em valores string, o operador de concatenação (+) concatena dois valores string, retornando outra string que é a união dos dois operandos.

Por exemplo,

```
console.log("minha " + "string"); // exibe a string "minha string".
```

O operador de atribuição encurtado += também pode ser utilizado para concatenar strings.

Por exemplo,

```
var minhaString = "alfa";
```

```
minhaString += "beto"; // É avaliada como "alfabeto" e atribui este valor
```

Operador condicional (ternário)

O operador condicional é o único operador JavaScript que utiliza três operandos. O operador pode ter um de dois valores baseados em uma condição. A sintaxe é:

```
condicao ? valor1 : valor2
```

Se `condicao` for verdadeira, o operador terá o valor de `valor1`. Caso contrário, terá o valor de `valor2`. Você pode utilizar o operador condicional em qualquer lugar onde utilizaria um operador padrão.

Por exemplo,

```
var status = (idade >= 18) ? "adulto" : "menor de idade";
```



Esta declaração atribui o valor "adulto" à variável `status` caso `idade` seja dezoito ou mais. Caso contrário, atribui o valor "menor de idade".

Operador vírgula

O operador vírgula (,) simplesmente avalia ambos de seus operandos e retorna o valor do segundo. Este operador é utilizado primariamente dentro de um laço `for` para permitir que multiplas variáveis sejam atualizadas cada vez através do laço.

Por exemplo, se `a` é uma matriz bidimensional com 10 elementos em um lado, o código a seguir utiliza o operador vírgula para incrementar duas variáveis de uma só vez. O código imprime os valores dos elementos diagonais da matriz:

```
for (var i = 0, j = 9; i <= 9; i++, j--)  
  console.log("a[" + i + "][" + j + "] = " + a[i][j]);
```



Operadores unário

Um operador unário é uma operação com apenas um operando.

delete

O operador `delete` apaga um objeto, uma propriedade de um objeto ou um elemento no índice especificado de uma matriz. A sintaxe é:

```
delete nomeObjeto;  
delete nomeObjeto.propriedade;  
delete nomeObjeto[indice];  
delete propriedade; // válido apenas dentro de uma declaração with
```



onde `nomeObjeto` é o nome de um objeto, `propriedade` é uma propriedade existente e `indice` é um inteiro que representa a localização de um elemento em uma matriz.

A quarta forma é permitida somente dentro de uma declaração `with` para apagar uma propriedade de um objeto.

Você pode utilizar o operador `delete` para apagar variáveis declaradas implicitamente mas não aquelas declaradas com `var`.

Se o operador `delete` for bem-sucedido, ele define a propriedade ou elemento para `undefined`. O operador `delete` retorna verdadeiro se a operação for possível; ele retorna falso se a operação não for possível.

```
x = 42;  
var y = 43;  
meuobj = new Number();  
meuobj.h = 4; // cria a propriedade h  
delete x; // retorna true (pode apagar se declarado implicitamente)  
delete y; // retorna false (não pode apagar se declarado com var)  
delete Math.PI; // retorna false (não pode apagar propriedades predefinidas)  
delete meuobj.h; // retorna true (pode apagar propriedades definidas pelo usuário)  
delete meuobj; // retorna true (pode apagar se declarado implicitamente)
```



Apagando elementos de array

Quando você apaga um elemento de um array, o tamanho do array não é afetado. Por exemplo, se você apaga `a[3]`, o valor de `a[4]` ainda estará em `a[4]` e `a[3]` passa a ser `undefined`.

Quando o operador `delete` remove um elemento do array, aquele elemento não pertence mais ao array. No exemplo a seguir, `arvores[3]` é removido com `delete`. Entretanto,

`arvores[3]` ainda é endereçável e retorna `undefined`.

```
var arvores = new Array("pau-brasil", "loureiro", "cedro", "carvalho", "s:
delete arvores[3];
if (3 in arvores) {
    // isto não é executado
}
```

Se você quer que um elemento do array exista, mas tenha um valor indefinido, utilize a palavra-chave `undefined` em vez do operador `delete`. No exemplo a seguir, o valor `undefined` é atribuído a `arvores[3]`, mas o elemento da matriz ainda existe:

```
var arvores = new Array("pau-brasil", "loureiro", "cedro", "carvalho", "s:
arvores[3] = undefined;
if (3 in arvores) {
    // isto será executado
}
```

typeof

O operador `typeof` é utilizado em qualquer uma das seguintes formas:

```
typeof operando
typeof (operando)
```

O operador `typeof` retorna uma string indicando o tipo do operando sem avaliação. `operando` é uma string, variável, palavra-chave ou objeto cujo tipo deve ser retornado. Os parênteses são opcionais.

Suponha que você defina as seguintes variáveis:

```
var meuLazer = new Function("5 + 2");
var forma = "redondo";
var tamanho = 1;
var hoje = new Date();
```

O operador `typeof` retornaria o seguinte resultado para aquelas variáveis:

```
typeof meuLazer; // retorna "function"
```

```
typeof forma; // retorna "string"  
typeof tamanho; // retorna "number"  
typeof hoje; // retorna "object"  
typeof naoExiste; // retorna "undefined"
```

Para as palavras-chave `true` e `null`, o `typeof` retorna os seguintes resultados:

```
typeof true; // retorna "boolean"  
typeof null; // retorna "object"
```

Para um número ou uma string, o `typeof` retorna os seguintes resultados:

```
typeof 62; // retorna "number"  
typeof 'Olá mundo'; // retorna "string"
```

Para valores de propriedades, o `typeof` retorna o tipo do valor que a propriedade possui:

```
typeof document.lastModified; // retorna "string"  
typeof window.length; // retorna "number"  
typeof Math.LN2; // retorna "number"
```

Para métodos e funções, o `typeof` retorna os seguintes resultados:

```
typeof blur; // retorna "function"  
typeof eval; // retorna "function"  
typeof parseInt; // retorna "function"  
typeof forma.split; // retorna "function"
```

Para objetos predefinidos, o `typeof` retorna os seguintes resultados:

```
typeof Date; // retorna "function"  
typeof Function; // retorna "function"  
typeof Math; // retorna "object"  
typeof Option; // retorna "function"  
typeof String; // retorna "function"
```

void

O operador `void` é utilizado de qualquer uma das seguintes formas:


```
void (expressao)
void expressao
```



O operador `void` especifica que uma expressão deve ser avaliada sem retorno de valor. `expressao` é uma expressão JavaScript que deve ser avaliada. Os parênteses em torno da expressão são opcionais, mas é uma boa prática utilizá-los.

Você pode utilizar o operador `void` para especificar uma expressão como um link de hipertexto. A expressão é avaliada mas não é carregada no lugar do documento atual.

O código a seguir cria um link de hipertexto que não faz coisa alguma quando clicado pelo usuário. Quando o usuário clica no link, `void(0)` é avaliado como indefinido, que não tem efeito em JavaScript.

```
<a href="javascript:void(0)">Clique aqui para fazer nada</a>
```



O código a seguir cria um link de hipertexto que submete um formulário quando clicado pelo usuário.

```
<a href="javascript:void(document.form.submit())">
Clique aqui para enviar</a>
```



Operadores relacionais

Um operador relacional compara seus operando e retorna um valor booleano baseado em se a comparação é verdadeira.

in

O operador `in` retorna verdadeiro se a propriedade especificada estiver no objeto especificado. A sintaxe é:

```
nomePropriedadeOuNumero in nomeObjeto
```



onde `nomePropriedadeOuNumero` é uma string ou uma expressão numérica que representa um nome de propriedade ou um índice de um array, e `nomeObjeto` é o nome de um objeto.

Os exemplos a seguir mostram alguns usos do operador `in`.

```
// Arrays
var arvores = new Array("pau-brasil", "loureiro", "cedro", "carvalho");
0 in arvores;           // retorna verdadeiro
3 in arvores;           // retorna verdadeiro
6 in arvores;           // retorna falso
"cedro" in arvores;     // retorna falso (você deve especificar o número do índice, não o valor naquele índice)
"length" in arvores;    // retorna verdadeiro (length é uma propriedade de Array)

// Objetos predefinidos
"PI" in Math;           // retorna verdadeiro
var minhaString = new String("coral");
"length" in minhaString; // retorna verdadeiro

// Objetos personalizados
var meucarro = {marca: "Honda", modelo: "Accord", ano: 1998};
"marca" in meucarro;    // retorna verdadeiro
"modelo" in meucarro;   // retorna verdadeiro
```

`instanceof`

O operador `instanceof` retorna verdadeiro se o objeto especificado for do tipo de objeto especificado. A sintaxe é:

```
nomeObjeto instanceof tipoObjeto
```

onde `nomeObjeto` é o nome do objeto a ser comparado com `tipoObjeto`, e `tipoObjeto` é um tipo de objeto como `Date` ou `Array`.

Utilize o `instanceof` quando você precisar confirmar o tipo de um objeto em tempo de execução. Por exemplo, ao capturar exceções você pode desviar para um código de manipulação de exceção diferente dependendo do tipo de exceção lançada.

Por exemplo, o código a seguir utiliza o `instanceof` para determinar se `dia` é um objeto `Date`. Como `dia` é um objeto `Date`, as declarações do `if` são executadas.

```
var dia = new Date(1995, 12, 17);
if (dia instanceof Date) {
```

```
// declarações a serem executadas
}
```

Precedência de operadores

A *precedência* de operadores determina a ordem em que eles são aplicados quando uma expressão é avaliada. Você pode substituir a precedência dos operadores utilizando parênteses.

A tabela a seguir descreve a precedência de operadores, da mais alta para a mais baixa.

Precedência de operadores	
Tipo de operador	Operadores individuais
membro	. []
chamada / criação de instância	() new
negação / incremento	! ~ - + ++ -- typeof void delete
multiplicação / divisão / resto ou módulo	* / %
adição / subtração	+ -
deslocamento bit a bit	<< >> >>>
relacional	< <= > >= in instanceof
igualdade	== != === !==
E bit a bit	&
OU exclusivo bit a bit	^
OU bit a bit	
E lógico	&&
OU lógico	

Tipo de operador condicional	Operadores individuais ?:
atribuição	= += -= *= /= %= <<= >>= >>>= &= ^= =
vírgula	,

Uma versão mais detalhada desta tabela, com links adicionais para detalhes de cada operador, pode ser vista em [Referência do JavaScript \(en-US\)](#).

Expressões

Uma *expressão* consiste em qualquer unidade válida de código que é resolvida como um valor.

Conceitualmente, existem dois tipos de expressões: aquelas que atribuem um valor a uma variável e aquelas que simplesmente possuem um valor.

A expressão `x = 7` é um exemplo do primeiro tipo. Esta expressão utiliza o *operador* `=` para atribuir o valor sete à variável `x`. A expressão em si é avaliada como sete.

O código `3 + 4` é um exemplo do segundo tipo de expressão. Esta expressão utiliza o operador `+` para somar três e quatro sem atribuir o resultado, sete, a uma variável.

O JavaScript possui as seguintes categorias de expressão:

- Aritmética: é avaliada como um número, por exemplo 3.14159. (Geralmente utiliza **operadores aritméticos**).
- String: é avaliada como uma string de caracteres, por exemplo, "Fred" ou "234". (Geralmente utiliza **operadores de string**).
- Lógica: é avaliada como verdadeira ou falsa. (Costuma envolver **operadores lógicos**).
- Expressões primárias: Palavras reservadas e expressões gerais do JavaScript.
- Expressão lado esquerdo: atribuição à esquerda de valores.

Expressões primárias

Palavras reservadas e expressões gerais do JavaScript.

this

Utilize a palavra reservada `this` para se referir ao objeto atual. Em geral, o `this` se refere ao objeto chamado em um método. Utilize o `this` das seguintes formas:

```
this["nomePropriedade"]  
this.nomePropriedade
```



Suponha uma função chamada `valide` que valida a propriedade `valor` de um objeto, dado o objeto e os valores máximo e mínimo:

```
function valide(obj, minimo, maximo){  
  if ((obj.valor < minimo) || (obj.valor > maximo))  
    alert("Valor inválido!");  
}
```



Você poderia chamar `valide` em cada manipulador de evento `onChange` de um formulário utilizando `this` para passar o elemento do formulário, como no exemplo a seguir:

```
<b>Informe um número entre 18 e 99:</b>  
<input type="text" name="idade" size=3  
  onChange="valide(this, 18, 99);">
```



Operador de agrupamento

O operador de agrupamento `()` controla a precedência de avaliação de expressões. Por exemplo, você pode substituir a precedência da divisão e multiplicação para que a adição e subtração sejam avaliadas primeiro.

```
var a = 1;  
var b = 2;  
var c = 3;  
  
// Precedência padrão  
a + b * c      // 7  
// a avaliação padrão pode ser assim  
a + (b * c)    // 7
```



```

a + (b * c) // 7

// Agora substitui a precedência
// soma antes de multiplicar
(a + b) * c // 9

// o que é equivalente a
a * c + b * c // 9

```

Comprehensions

Comprehensions são uma característica experimental de JavaScript, marcada para ser inclusa em uma versão futura do ECMAScript. Existem duas versões de Comprehensions:



[for (x of y) x]

Comprehensions de array.



(for (x of y) y)

gerador de comprehensions

Comprehensions existem em muitas linguagens de programação e permitem que você rapidamente monte um novo array com base em um existente, por exemplo:

```

[for (i of [ 1, 2, 3 ]) i*i ];
// [ 1, 4, 9 ]

var abc = [ "A", "B", "C" ];
[for (letras of abc) letras.toLowerCase()];
// [ "a", "b", "c" ]

```



Expressão lado esquerdo

Atribuição à esquerda de valores.

new

Você pode utilizar o [operador new](#) para criar uma instância de um tipo de objeto definido pelo usuário ou de um dos tipos de objeto predefinidos: `Array`, `Boolean`, `Date`, `Function`, `Image`, `Number`, `Object`, `Option`, `RegExp` ou `String`. No servidor, você pode também utilizar `DbPool`, `Lock`, `File` ou `SendMail`. Utilize o operador `new` da seguinte forma:

```
var nomeObjeto = new tipoObjeto([parametro1, parametro2, ..., parametroN]);
```

super

A palavra reservada super é utilizada para chamar a função pai de um objeto. É útil para nas classes para a chamada do construtor pai, por exemplo:

```
super([argumentos]); //chama o construtor pai.  
super.funcaoDoPai([argumentos]);
```

Operador spread

O operador `spread` permite que uma expressão seja expandido em locais onde são esperados vários argumentos (para chamadas de função) ou vários elementos (para arrays).

Exemplo: Se você tem um array e deseja criar um novo array com os elementos do array já existente sendo parte do novo array, a sintaxe do array não será suficiente e você terá de usar uma combinação de `push`, `splice`, `concat`, etc. Com a sintaxe `spread`, isto torna-se muito mais sucinto:

```
var partes = ['ombro', 'joelhos'];  
var musica = ['cabeca', ...partes, 'e', 'pés'];
```

Da mesma forma, o operador `spread` funciona com chamadas de função:

```
function f(x, y, z) { }  
var args = [0, 1, 2];  
f(...args);
```