Operatoren

Wat zijn operatoren?

JavaScript operatoren worden gebruikt om **waarden toe te kennen** *(Eng. assign values),* **waarden met elkaar te vergelijken** *(Eng. compare values),* **wiskundige operaties uit te voeren** *(Eng. perform arithmetic operations)* …

JavaScript heeft zowel **tweeledige** *(Eng. binary),* **enkele** *(Eng. unary)* en **drieledige** *(Eng. ternary)* of conditionele operatoren. Een binarie operator vereist twee **operanden** *(Eng. operands),* namelijk één voor de operator en één na de operator:

```
operand1 operator operand2
```

Bijvoorbeeld: `6 + 2` of `x * y`.

Een **unaire operator** vereist een enkele operand, hetzij voor of na de operator:

```
operator operand
```

of

```
operand operator
```

Bijvoorbeeld: `i++` of `++i`.

Wiskundige operatoren

Een **wiskundige operator** *(Eng. arithmetic operator)* bevat numerieke waarden als operanden en geeft een enkele numerieke waarde **terug** *(Eng. return).* De standaard wiskundige operatoren zijn:

Operator	Betekenis	Voorbeeld

Operator	Betekenis	Voorbeeld
`… + …`	Som of **optellen** *(Eng. addition)*	`3 + 6` geeft `9` terug
`… - …`	Verschil of **aftrekken** *(Eng. subtraction)*	`3 - 6` geeft `-3` terug
`… * …`	Product of **vermenigvuldigen** *(Eng. multiplication)*	`3 * 6` geeft `18` terug
`… / …`	Quotiënt of **deling** *(Eng. division)*	`3 / 6` geeft `0.5` terug

> ⚠ **Opgelet**
>
> `3 / 0` resulteert in de waarde `Infinity`

In JavaScript zijn er nog bijkomende wiskundige operatoren:

Operator	Betekenis	Voorbeeld
`… ** …`	Exponent of tot een **macht verheffen** *(Eng. exponentiation)*	`3 ** 2` geeft `9` terug, `10 ** -1` geeft `0.1` terug
`… % …`	Modulo of de **rest van een deling** *(Eng. division remainder)*	`3 % 2` geeft `1` terug, `6 % 2` geeft `0` terug
`…++`	Ophogen of **incrementeren** *(Eng. post increment)*	`3++` geeft `3` terug en zet dan de waarde op `4`
`++…`	Ophogen of **incrementeren** *(Eng. pre increment)*	`++3` geeft `4` terug
`…--`	Verlagen of **decrementeren** *(Eng. post decrement)*	`3--` geeft `3` terug en zet dan de waarde op `2`
`--…`	Verlagen of **decrementeren** *(Eng. pre decrement)*	`--3` geeft `2` terug

Operator	Betekenis	Voorbeeld
`-...`	Unaire **negatie** *(Eng. negation)*	Als `i` de waarde `3` bevat, door `-i` bevat het de waarde `-3`
`+...`	Unaire **plus** *(Eng. plus)*	Probeert een operand te converteren naar een numerieke waarde. Bijvoorbeeld: `+"3"` geeft `3` terug, `+true` geeft `1` terug

Met de `+` operator kunnen we ook twee strings (die fungeren als operanden) met elkaar verbinden.

```js
console.log("My name is " + "Philippe"); // Output: My name is Philippe
let data = "Micro";
data += " Macro";
console.log(data); // Output: Micro Macro
```

Toewijzings operatoren

Toewijzingsoperator *(Eng. assignment operator)* kent een waarde toe aan de **linker operand** *(Eng. left operand)* gebaseerd op de waarde van de **rechter operand** *(Eng. right operand).* De eenvoudigste toewijzingsoperator is **"gelijk aan"** *(Eng. equal)* (`=`).

Bijvoorbeeld: `x = y` , de waarde van `y` wordt toegekend als waarde voor `x` .

In JavaScript zijn er heel wat verschillende soorten **samengestelde** *(Eng. compound)* toewijzingsoperatoren:

Name	Korte notatie	Betekenis
Assignment	`x = y`	`x = y`
Addition assignment	`x += y`	`x = x + y`
Subtraction assignment	`x -= y`	`x = x - y`
Multiplication assignment	`x *= y`	`x = x * y`
Division assignment	`x /= y`	`x = x / y`

Name	Korte notatie	Betekenis
Remainder assignment	`x %= y`	`x = x % y`
Exponentiation assignment	`x **= y`	`x = x ** y`
Left shift assignment	`x <<= y`	`x = x << y`
Right shift assignment	`x >>= y`	`x = x >> y`
Unsigned right shift assignment	`x >>>= y`	`x = x >>> y`
Bitwise AND assignment	`x &= y`	`x = x & y`
Bitwise XOR assignment	`x ^= y`	`x = x ^ y`
Bitwise OR assignment	`x | = y`	`x = x | y`

Toewijzen van een waarde aan een variabele. We kunnen via “chaining” ook een waarde toekennen aan meerdere variabelen.

Tip

- Sameness comparision
- Equality Table

```js
let x = 4;
const y = 6;
x = y; // The value of x is 6

let z = 8;
x = z = y; //x, y and z contains the value 6
```

Met de addition assignment operator kunnen we de waarde van het rechter operand toevoegen aan de waarde van het linker operand.

```js
let a = 5;
a += 3; // a contains the value 8

let b = 7;
```

```
b += true; // b contains 8

let c = false;
c += 7; // c contains 7

let d = 'John';
d += false; // d contains 'Johnfalse'

let e = 'Hello;
e += ' World'; // e contains 'Hello World'
```

Met de subtraction assignment operator kunnen we de waarde van het rechter operand aftrekken van de waarde van het linker operand.

```js
let a = 4;
a -= 6; // a contains -2
a -= "b"; // a contains NaN

let b = "Hello";
b -= 4; // b contains NaN
```

Met de multiplication assignment operator vermenigvuldigen we de waarde van de variabele met de waarde van het rechter operand.

```js
let a = 4;
a *= -6; // a contains -24
a *= "b"; // a contains NaN
```

Met de division assignment operator delen we de waarde van de variabele met de waarde van het rechter operand.

```js
let a = 4;
a /= -2; // a contains -2
a /= "b"; // a contains NaN

let b = 6;
b /= -0; // a contains -Infinity
```

De remainder assignment operator deelt de variabele door de waarde van de rechter operand en kent vervolgens de rest van de deling toe aan deze variabele.

```js
let a = 4;
a %= -3; // a contains 1
a %= "b"; // a contains NaN

let b = 6;
b %= 0; // b contains NaN
```

De exponentiation assignment operator verheft de waarde van een variabele tot de macht vermeld in de rechter operand.

```js
let a = 4;
a **= 3; // a contains 64
a **= "b"; // a contains NaN

let b = 1000;
b **= -1; // b contains 0.001 --> 1 / (1000 ^ 1)
```

De left shift assignment operator verplaatst de hoeveelheid bits (rechter operand) naar links en kent deze waarde toe aan de variabele.

```js
let a = 5; // 00000000000000000000000000000101
a <<= 2; // 00000000000000000000000000010100 --> 20
```

De right shift assignment operator verplaatst de hoeveelheid bits (rechter operand) naar rechts en kent deze waarde toe aan de variabele.

```js
let a = 5; // 00000000000000000000000000000101
a >>= 2; // 00000000000000000000000000000001 --> 1
```

Logische operatoren

Logische operatoren *(Eng. logical operators)* worden gebruikt bij **Boolean**-operanden en geeft een Boolean-waarde terug. JavaScript bevat de volgende logische operatoren:

Operator	Voorbeeld

Operator	Voorbeeld
AND (`&&`)	`expr1 && expr2` Wanneer beide expressies de waarde `true` bevatten resulteert dit in `true` . Bevat beiden `false` dan resulteert dit in `false` . Bevat de expressie 1 de waarde `true` en expressie 2 de waarde `false` , dan geeft dit `false` terug.
OR (`||`)	`expr1 || expr2` Wanneer ten minste één van beide expressies de waarde `true` bevat zal dit resulteren in de waarde `true` . Als beide `false` waarde bevatten geeft dit de waarde `false` terug.
NOT (`!`)	`!expr1` Geeft de inverse Boolean waarde terug van de Boolean waarde van de expressie 1. Bevat de expressie de waarde `true` , dan resulteert dit in de waarde `false` .

De volgende waarden geven na conversie, via `Boolean(waarde)` , de waarde `false` terug: `null` , `0` , `NaN` , `""` en `undefined` .

Voorbeelden van de `&&` operator:

```js
true && true; // returns true
false && true; // returns false
false && "PGM"; // returns false
"PGM" && false; // returns false
true && "PGM"; // returns 'PGM'
"PGM" && true; // returns true
true && 3 == 6; // returns false
```

Voorbeelden van de `||` operator:

```js
true || true; // returns true
false || true; // returns true
false || "PGM"; // returns 'PGM'
"PGM" || false; // returns false
true || "PGM"; // returns true
"PGM" || true; // returns 'PGM'
true || 3 == 6; // returns true
"PGM" || "NMD"; // returns 'PGM'
```

Voorbeelden van de `!` operator:

```js
!true; // returns false
!false; // returns true
!"PGM"; // returns false
```

Bitwise operatoren

Een bitwise operator behandelt de operanden als een verzameling van 32 bits resulterend in een binair getal. Bijvoorbeeld het decimaal getal `9` komt overeen met het binair getal `1001`. De bitwise operator voer operaties uit op de binaire representatie van een getal, maar het geeft wel een numerieke waarde terug.

Operator	Gebruik	Omschijving
AND	`x & y`	Geeft een `1` terug op elke bit positie waarvan de corresponderende bits van beide operanden een `1` bevatten. Bijv.: `15 & 9` komt overeen met `1111 & 1001` is gelijk aan `1001` wat resulteert in `9`.
OR	`x | y`	Geeft een `0` terug op elke bit positie waarvan de corresponderende bits van beide operanden een `0` bevatten. Bijv.: `15 | 9` komt overeen met `1111 | 1001` is gelijk aan `1111` wat resulteert in `15`.
XOR	`x ^ y`	Geeft een `0` terug op elke bit positie waarvan de corresponderende bits van beide operanden gelijk zijn. Bijv.: `15 ^ 9` komt overeen met `1111 | 1001` is gelijk aan `0110` wat resulteert in `6`.
NOT	`~x`	Iverteert de bits van het operand. Bijv.: `15` komt overeen met `00000000000000000000000000001111` zodat `~15` resulteert in `11111111111111111111111111110000`
Left shift	`x << y`	Shift `x` in binaire representatie `y` bits naar links. De binaire representatie wordt rechts aangevuld met `0` per shiftpositie. Bijv.: `9 << 2` komt overeen met `1001 << 0010` is gelijk aan `100100` wat resulteert in `36`.
Zero-fill right shift	`x >>> y`	Shift `x` in binaire representatie `y` bits naar rechts. De binaire representatie wordt links aangevuld met `0`. Bijv.: `9 >>> 2` komt overeen met `1001 >> 0010` is gelijk aan `0010` wat resulteert in `2`. `-9 >> 2` komt overeen met `11111111111111111111111111110111 >> 0010` is gelijk aan `00111111111111111111111111111101` wat resulteert in `1073741821`

```js
const permissionRead = 4;
const permissionWrite = 2;
const permissionExecute = 1;

let myPermission = 0 | permissionWrite;
const message =
  myPermission & permissionRead
    ? "You have got read permission"
    : "You can't read files and documents!";
console.log(message); // Output: "You can't read files and documents!'
```

```js
const hex = "ffaadd";
const rgb = parseInt(hex, 16); // rgb value is 16755421 in binary form 111111111010101

// 1. rgb value:                    111111111010101011011101
// 2. shift operation (rgb >> 16):  000000000000000011111111
// 3. 255:                          000000000000000011111111
// 4. Result (255)                  000000000000000011111111
const red = (rgb >> 16) & 0xff;

// 1. rgb value:                    111111111010101011011101
// 2. shift operation (rgb >> 8):   000000001111111110101010
// 3. 255:                          000000000000000011111111
// 4. Result (170)                  000000000000000010101010
const green = (rgb >> 8) & 0xff;

// 1. rgb value:                    111111111010101011011101
// 2. shift operation (rgb >> 8):   111111111010101011011101
// 3. 255:                          000000000000000011111111
// 4. Result (221)                  000000000000000011011101
const blue = rgb & 0xff;
```

Ternaire operator

Een **ternaire operator** *(Eng. ternary operator)* of conditionele operator bestaat uit drie operanden. De operator kan een van de twee opgegeven waarden bevatten afhankelijk van de conditie. De syntax ziet er als volgt uit:

```js
condition ? val1 : val2;
```

Als de `condition` de waarde `true` bevat, dan zal de conditionele operator de waarde `val1` bevatten. Is de conditie `false`, dan bevat te ternary operator de waarde `val2`.

Voorbeeld:

```js
let isPlaying = false;
const gameState = isPlaying ? "You are playing." : "The game is finished.";
console.log(gameState); // Output: "The game is finished."
```

Unaire operatoren

Een **unaire operator** *(Eng. unary operator)* operator met slechts één operand.

delete

De `delete` operator verwijdert een object, een eigenschap van een object of een element uit een array op een specifieke index. De syntax is als volgt:

```js
delete objectName;
delete objectName.property;
delete objectName[index];

with (objectName) {
  delete property;
}
```

`objectName` is de naam van het object, `property` is de naam van een bestaande eigenschap binnen een object en `index` is een positief geheel getal die de locatie van een element in de array aanduidt. Variabelen gedeclareerd met `var`, `let` en `const` kunnen niet verwijderd worden, enkel impliciet gedeclareerde variabelen kunnen verwijderd worden. Door gebruik te maken van een `with` statement kunnen binnen in het blok rechtstreeks de eigenschap aanspreken. Voorgedefinieerde eigenschappen van built-in objecten kunnen niet verwijderd worden.

```js
x = 67;
delete x; // returns true
person = {
```

```
    firstName: 'Philippe',
    surName: 'De Pauw',
    age = '999'
}
delete person.age; // returns true
delete person; // returns true
delete Math.PI; // returns false
let y = 89;
delete y; // retrurns false
```

typeof

Met de `typeof` -operator kunnen we het type opvragen van een operand. Het type is een stringwaarde die het type identificeert. De `typeof` operator zorgt als volgt gebruikt:

```js
typeof operand;
typeof operand;
```

Voorbeeld:

```js
const size = 1;
typeof size; // returns "number"
let now = new Date();
typeof now; // returns "object"
let firstName = "Philippe";
typeof firstName; // returns "string"
typeof alien; // returns "undefined"
typeof true; // returns "boolean"
typeof null; // returns "object"
typeof Math.PI; // returns "number"
typeof document.lastModified; // returns "string"
typeof Date; // returns "function"
typeof Math; // returns "object"
```

void

De `void` operator evalueert een expressie zonder dat een waarde **teruggegeven** ***(Eng. return)*** zal worden. “Zonder een waarde teruggeven” betekent dat deze waarde `undefined` is.

```js
void 0 === undefined; // true
```

```js
void (function IIFE() {
  // do something
})();
```

```html
<a href="javascript: void 0; window.alert('Clicked the link');">Click me</a>
```

Comma operator

De comma (`,`) seperator evalueert alle operanden en geeft de waarde van het laatste operand terug.

```js
let a, b, c;
(a = b = 6), (c = 4);
console.log(a); // Output: 6

let d, e, f;
d = ((e = 5), (f = 8));
console.log(d); // Output: 8

let x = 0;
x = (x++, x);
console.log(x); // Output: 1
```

```js
for (let i = 0, j = 9; i <= j; i++, j--) {
  console.log(`i: ${i}, j: ${j}`);
}
/*
"i: 0, j: 9"
"i: 1, j: 8"
"i: 2, j: 7"
"i: 3, j: 6"
"i: 4, j: 5"
*/
```

Relationele operator

De **relationele operator** *(Eng. relational operator)* vergelijkt de operanden en geeft een boolean waarde terug.

De `in` operator geeft de waarde `true` terug indien de operand 1 als eigenschap voor komt in operand 2.

```js
const daysOfWeek = [
  "sunday",
  "monday",
  "tuesday",
  "wednesday",
  "thursday",
  "friday",
  "saturday",
];
0 in daysOfWeek; // returns true
"monday" in daysOfWeek; // returns false
"length" in daysOfWeek; // returns true because length is a property of a string
const person = { firstName: "Philippe" };
"firstName" in person; // returns true;
```

De `instanceof` operator geeft `true` terug wanneer het gespecifieerd object van een welbepaald object type is.

```js
const now = new Date();
if (now instanceof Date) {
  // do something
}
console.log(new String("pol") instanceof String); // Output: true
```

Operator voorrang

De voorrang van operatoren bepalen de volgorde waarin deze worden toegepast tijdens het evalueren van een expressie. De volgende tabel beschrijft de voorrang van operatoren van de hoogste naar de laagste voorrang:

#	Operator type	Individual operators

#	Operator type	Individual operators
1	member	`.` `[]`
2	call / create instance	`()` `new`
3	negation/increment	`!` `~` `-` `+` `++` `--` `typeof` `void` `delete`
4	multiply/divide	`*` `/` `%`
5	addition/subtraction	`+` `-`
6	bitwise shift	`<<` `>>` `>>>`
7	relational	`<` `<=` `>` `>=` `in` `instanceof`
8	equality	`==` `!=` `===` `!==`
9	bitwise AND	`&`
10	bitwise XOR	`^`
11	bitwise OR	`|`
12	logical AND	`&&`
13	logical OR	`| |`
14	conditional	`?:`
15	assignment	`=` `+=` `-=` `*=` `/=` `%=` `<<=` `>>=` `>>>=` `&=` `^=` `| =`
16	comma	`,`

← Console en dialogen

Control Flow →

Laatst bijgewerkt: 21/9/2022 15:53:57