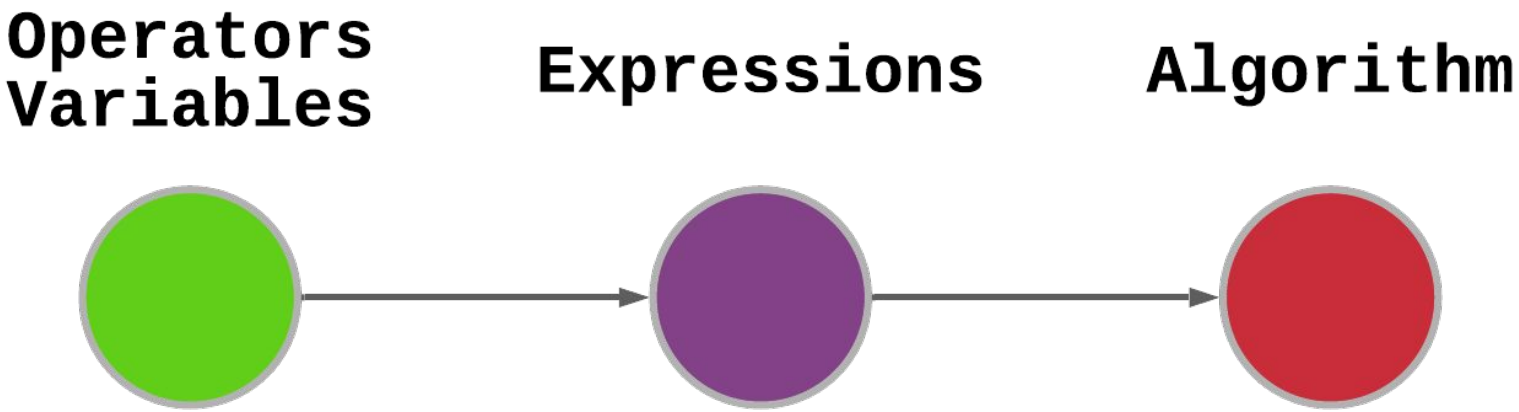
Operators

Ba Nguyễn

Operators



Operators

Toán tử trong JavaScript được chia thành 5 loại:

- Arithmetic
- Comparision
- Assignment
- Logical
- Bitwise

Quy tắc thực hiện biểu thức theo thứ tự *từ trái qua phải, dựa theo độ ưu tiên toán tử*. Các toán tử có độ ưu tiên khác nhau, quyết định phép tính nào sẽ được thực hiện trước

JavaScript cũng ***tự động chuyển đổi*** các kiểu dữ liệu của các toán hạng về kiểu phù hợp khi thực hiện các biểu thức tính toán hay so sánh

Type Conversions

JavaScript ***tự động chuyển đổi kiểu dữ liệu*** khi thực hiện tính toán các giá trị, đồng thời cung cấp thêm một số phương thức để chuyển đổi thủ công từ kiểu này sang kiểu khác

```
let result = 1 + 2; // 3
let answer = "1 + 2 = " + result;
// result từ "number" -> "string"
// "1 + 2 = 3"

let other = 1 - "1"; // 0
// "1" từ "string" -> number
```

Type Conversions

```
// Chuyển đổi các kiểu khác về kiểu "string"
String(123); // "123"
String(true); // "true"

// Chuyển đổi các kiểu khác về kiểu "boolean"
Boolean(12); // true

// Một số giá trị đặc biệt
"", 0, null, undefined, NaN; // false
// Còn lại tất cả giá trị khác là true
```

Type Conversions

```
// Chuyển đổi các kiểu khác về kiểu "number"
Number("12"); // 12

// Một số giá trị đặc biệt
Number(null); // 0
Number(undefined); // NaN
Number(true); // 1
Number(false); // 0
Number("   12    "); // 12
Number("12ab"); // NaN
Number(""); // 0
```

Arithmetic Operators

Các toán tử số học cơ bản: **+ - * / % ****

```
// Chia lấy phần dư
5 % 3; // 2
6 % 2; // 0

// Lũy thừa (mũ)
2 ** 2; // 4
2 ** 3; // 8

// Mọi phép tính (trừ nối chuỗi) với NaN đều cho kết
quả là NaN
5 - NaN; // NaN
```

Arithmetic Operators

Đối với kiểu **string** phép **+** chuyển đổi kiểu của toán hạng về kiểu **string** và thực hiện nối chuỗi

```
1 + "2"; // "12"
1 + 2 + "3"; // "33"
"1" + true + 0; // "1true0"
```

Phép nối chuỗi ***chỉ hoạt động duy nhất với toán tử +*** , với những toán tử số học khác, mọi kiểu dữ liệu được chuyển về **number**

```
6 - "2" + 3; // 7
5 % "3" + 2; // 4
```

Assignment Operators

```
// Toán tử gán chỉ hoạt động với biến
let x = 1;
let y = 1 + 2 + 3; // 6

// Gán kết hợp với toán tử số học
x += 1; // x = x + 1 -> 2
y /= 2; // y = y / 2 -> 3
```

Assignment Operators

Increment, Decrement (tự tăng/giảm)

++ và -- là hai toán tử đặc biệt, nó thực hiện phép tính ***tăng/giảm*** giá trị của biến đi **1**, hai toán tử này có thể đặt ở trước biến - ***prefix*** hoặc sau biến - ***postfix***. Khi sử dụng trong một câu lệnh riêng biệt thì không có sự khác nhau

```
let a = 1;
a++; // 2
++a; // 3
a--; // 2
--a; // 1
```

Assignment Operators

Tuy nhiên, khi dùng trong một biểu thức, ***postfix*** - tăng/giảm giá trị đi **1** và trả về ***giá trị trước đó (cũ)***, ***prefix*** - cũng tăng/giảm giá trị đi **1** nhưng trả về ***giá trị sau (mới)***

```
let a = 1;
let b = a++ + 1; // a = 2, b = 2
let c = ++a + 3; // a = 3, c = 6

let d = a++ + ++a - a-- - --a;
// d = 3 + 5 - 5 - 3 = 0
// a = 3
```

Exercises

Tính giá trị các biểu thức

```
let a = 1,
    b = (a % 2) * 2,
    c = a++ - b--,
    d = "0";

a + b + c + d;
a - b + c - d;
a-- + (b-- * c) / d;
++a - +b * c + d;
d + ++a + (--b % c);
```

```
let a = 1,
    b = (a * 2) / 2,
    c = a-- + b++,
    d = "-0";

a - b - c - d;
a + b - c + d;
a++ - (b++ / c) * d;
--a + -b / c - d;
d - --a - ++b * c;
```

Comparision Operators

Toán tử so sánh == != > >= < <= === !==

Kết quả của các phép so sánh **luôn luôn là một giá trị boolean**

== != > >= < <= tự động chuyển đổi kiểu của 2 toán hạng về cùng một kiểu và thực hiện so sánh

```
2 < 2; // false
2 <= 2; // true


2 == "2"; // true
2 != "2"; // false
```

Comparision Operators

=== và !== (*strict comparison*) so sánh cả kiểu giá trị của dữ liệu

```
2 === 2; // true
2 === "2"; // false
2 !== "2"; // true
"2" === "2"; // true
```

Comparision Operators

So sánh chuỗi

JavaScript sử dụng bảng mã Unicode, khi so sánh 2 chuỗi, nó thực hiện so sánh từng ký tự dựa theo thứ tự trong bảng mã (Unicode table)

```
"a" > "A"; // true
"A" > "Z"; // false
"Ba" == "Ba"; // true
"Ba" < "Ba Nguyễn"; // true
```

Tham khảo bảng mã Unicode: wiki/unicode_character

Comparision Operators

null, **undefined**, **NaN**

null và **undefined** là 2 trường hợp đặc biệt

```
null == 0; // false
null <= 0; // true
null == undefined; // true
null === undefined; // false
```

Mọi biểu thức so sánh với **NaN** đều cho kết quả là **false**

Logical Operators

Toán tử logic || - or, && - and, ! - not

|| - or tìm **giá trị đúng** đầu tiên trong biểu thức và trả về giá trị đó, nếu không có thì trả về giá trị cuối cùng trong biểu thức

```
true || false; // true
0 || 1; // 1
0 || false || ""; // ""
"abc" || "xyz"; // "abc"
```

Logical Operators

&& - and tìm **giá trị sai** đầu tiên trong biểu thức và trả về giá trị đó, nếu không có thì trả về giá trị cuối cùng trong biểu thức

```
true && false; // false
0 && 1; // 0
0 && false && ""; // 0
"abc" && "xyz"; // "xyz"
```

Logical Operators

! - **not** chuyển giá trị về kiểu **boolean** và phủ định nó

```
!""; // true
!"123"; // false
!!false; // false
!!"xyz"; // true
!!!!!!!!!!false; // false
```

Exercises

Tính giá trị các biểu thức

```
let a = 1,
    b = !a;
let c = (!a && !!b) || 0;

a && b && c;
a || b || c;
(a && !b) || !!c;
!(a || !b) && c;
!!(a && !!b) || !c;
```

```
let a = 0,
    b = !!a;
let c = a || (!b && 0);

a && b && c;
a || b || c;
!a || (b && !c);
!!(!a && b) || c;
!(!a || !b) || c;
```