

# Boolean

---

1. Boolean overview.
2. Type conversion & Type coercion.
3. Truthy vs Falsy.
4. Logical operators.
5. Comparison.
6. if...else.
7. switch...case.
8. Bài tập thực hành.

## 1. Boolean overview:

Trong JavaScript, kiểu dữ liệu boolean chỉ có hai giá trị: true và false. Boolean thường được sử dụng để biểu diễn trạng thái logic, như đúng (true) hoặc sai (false). Đây là một số ví dụ cơ bản về cách sử dụng boolean trong JavaScript:

```
var isTrue = true;
var isFalse = false;

// Sử dụng boolean trong các biểu thức logic
var x = 10;
var y = 5;
var result = (x > y); // Kết quả sẽ là true

// Boolean có thể được trả về từ các biểu thức so sánh
var isEqual = (x === y); // Kết quả sẽ là false

// Boolean có thể được sử dụng trong các câu lệnh điều kiện
if (isTrue) {
    console.log("Đây là true");
} else {
    console.log("Đây là false");
}

// Boolean có thể được sử dụng để kiểm tra trạng thái của một biến
var isNotNull = (x !== null); // Nếu x không null, thì kết quả sẽ là true

// Boolean cũng có thể được sử dụng trong các phép gán
var isEnabled = true;
var isDisabled = !isEnabled; // Kết quả sẽ là false
```

Tham khảo: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>

## 2. Type conversion & Type coercion:

Type conversion và type coercion là hai khái niệm liên quan đến việc chuyển đổi kiểu dữ liệu trong JavaScript, nhưng chúng có một số điểm khác biệt nhất định:

### 1. Type Conversion (Chuyển đổi kiểu dữ liệu):

- Type conversion (chuyển đổi kiểu dữ liệu) là quá trình rõ ràng mà bạn tự định nghĩa và thực hiện để chuyển đổi một giá trị từ một kiểu dữ liệu sang một kiểu dữ liệu khác.
- Trong type conversion, bạn thực hiện chuyển đổi kiểu dữ liệu bằng cách sử dụng các hàm hoặc phương thức như `parseInt()`, `toString()`, `Number()`, `String()`, v.v.

Ví dụ của type conversion:

```
var x = "5"; // x là một chuỗi
var y = parseInt(x); // Chuyển đổi chuỗi thành số nguyên
console.log(y); // Kết quả sẽ là số nguyên 5
```

### 2. Type Coercion (Ép kiểu dữ liệu):

- Type coercion (ép kiểu dữ liệu) là quá trình ngầm định mà JavaScript tự động thực hiện để chuyển đổi một giá trị từ một kiểu dữ liệu sang một kiểu dữ liệu khác trong quá trình thực thi phép toán.
- Trong type coercion, JavaScript tự động chuyển đổi kiểu dữ liệu một cách ngầm định để thực hiện phép toán.

Ví dụ của type coercion:

```
var x = 5; // x là một số nguyên
var y = "10"; // y là một chuỗi

var result = x + y; // Khi thực hiện phép cộng, JavaScript chuyển đổi x thành chuỗi và sau đó thực hiện phép cộng chuỗi
console.log(result); // Kết quả sẽ là chuỗi "510"
```

Tóm lại:

- Điểm tương đồng là đều chuyển đổi dữ liệu từ kiểu này sang kiểu khác.
- Type coercion là ép kiểu tự động (ép kiểu ngầm).
- Type conversion mang ý nghĩa là có thể là ép kiểu tự động như type coercion hoặc ép kiểu tường minh (explicit, tức do mình chỉ định nó ép kiểu).

Ví dụ thêm:

```
// Type coercion
const a = 1 + '2'; // JavaScript tự động chuyển đổi số 1 thành chuỗi '1' để thực hiện phép cộng với chuỗi '2', dẫn đến kết quả là chuỗi '12'.
const b = 1 - '2' // JavaScript tự động chuyển đổi chuỗi '2' thành số 2 để thực hiện phép trừ, dẫn đến kết quả là số -1.
```

```
const message = 'JS is easy'
if (message) { // message here will be converted into boolean
  automatically
  console.log('Really!? 🤖 ')
}
// Explicit conversion
const error = 'Something wrong!';
const hasError = Boolean(error); // we ask to convert string error to
boolean
const mark = 9;
const markString = mark.toString(); // or String(mark)
const type = '1';
const typeId = Number.parseInt(type); // or Number(type)
```

### 3. Truthy vs Falsy:

- **Truthy:** là những giá trị khi chuyển đổi về **boolean** thì sẽ được giá trị true.
- **Falsy:** là những giá trị khi chuyển đổi về **boolean** thì sẽ được giá trị false.

Làm sao xác định được đâu là **truthy** hay **falsy**?

Với **falsy** thì mình có một danh sách như bên dưới, ngoài danh sách này, tất cả đều là **truthy**.

- **false:** false được coi là falsy.
- **0, -0:** Số nguyên 0 được coi là falsy.
- **' ' "" ``**(Chuỗi rỗng): Chuỗi rỗng không chứa bất kỳ ký tự nào cũng là falsy.
- **null:** Giá trị null là falsy. Null đại diện cho giá trị không tồn tại hoặc không có giá trị.
- **undefined:** Giá trị undefined là falsy. Undefined đại diện cho một biến chưa được gán giá trị hoặc một thuộc tính không tồn tại.
- **NaN (Not-a-Number):** Giá trị NaN là falsy. NaN là kết quả của một phép toán số học không hợp lệ.
- **document.all:** Trong một số trình duyệt cũ, document.all được sử dụng để xác định tất cả các phần tử trong trang. Tuy nhiên, nó cũng là một giá trị falsy và không nên được sử dụng.

Tham khảo:

- <https://developer.mozilla.org/en-US/docs/Glossary/Truthy>
- <https://developer.mozilla.org/en-US/docs/Glossary/Falsy>

Bài tập:

Xác định truthy hay falsy cho các giá trị dưới đây:

1. {}
2. []
3. ''
4. '0'
5. 'null'
6. 1000
7. 'false'
8. true

9. -10

### Giải bài tập

1. {} (Đối tượng rỗng): Đối tượng rỗng {} được coi là truthy. Trong JavaScript, một đối tượng (bao gồm cả đối tượng rỗng) được coi là truthy khi được xem xét trong ngữ cảnh của một biểu thức logic.
2. [] (Mảng rỗng): Mảng rỗng [] cũng được coi là truthy. Tương tự như đối tượng rỗng, một mảng rỗng cũng được coi là truthy.
3. "" (Chuỗi rỗng): Chuỗi rỗng "" là một giá trị falsy. Trong JavaScript, chuỗi rỗng không chứa bất kỳ ký tự nào, do đó được coi là falsy.
4. '0': Chuỗi '0' không rỗng, do đó được coi là truthy. Trong JavaScript, chuỗi có chứa ký tự thì được coi là truthy, ngay cả khi chứa ký tự số 0.
5. 'null': Chuỗi 'null' không rỗng, do đó được coi là truthy. Trong JavaScript, chuỗi có chứa ký tự thì được coi là truthy, ngay cả khi nó có giống với một từ khóa như null.
6. 1000: Số nguyên 1000 là một giá trị truthy. Trong JavaScript, một số khác 0 được coi là truthy.
7. 'false': Chuỗi 'false' không rỗng, do đó được coi là truthy. Trong JavaScript, chuỗi có chứa ký tự thì được coi là truthy, ngay cả khi nó có giống với một từ khóa như false.
8. true: Giá trị boolean true là truthy.
9. -10: Số nguyên âm -10 là một giá trị truthy. Trong JavaScript, một số khác 0 được coi là truthy.

## 4. Logical operators (Toán tử luận lý):

Trong JavaScript, các toán tử logic được sử dụng để thực hiện các phép toán logic trên các biểu thức có giá trị boolean và trả về kết quả boolean. Các toán tử logic cơ bản trong JavaScript bao gồm:

1. Toán tử AND (&&): Trả về true nếu cả hai biểu thức là true, ngược lại trả về false.

```
true && true;    // true
true && false;   // false
false && true;   // false
false && false;  // false
```

2. Toán tử OR (||): Trả về true nếu ít nhất một trong hai biểu thức là true, ngược lại trả về false.

```
true || true;    // true
true || false;   // true
false || true;   // true
false || false;  // false
```

3. Toán tử NOT

!: Đảo ngược giá trị của biểu thức, trả về true nếu biểu thức là false, và ngược lại.

!!: thường được sử dụng để chuyển đổi một giá trị sang kiểu boolean. Khi sử dụng toán tử !!, nó sẽ đảo ngược giá trị của biểu thức trước đó hai lần, kết quả sẽ là một giá trị boolean.

```
!true; // false
!false; // true
!!true; // true
!!false; // false

var x = "Hello";
var y = ""; // chuỗi rỗng

console.log(!x); // true
console.log(!y); // false
```

Các toán tử logic có thể được kết hợp với nhau để tạo ra các biểu thức logic phức tạp hơn:

```
var a = 5;
var b = 10;
var c = 15;

// Kiểm tra xem a là số chẵn và b là số lẻ
if (a % 2 === 0 && b % 2 !== 0) {
  console.log("a là số chẵn và b là số lẻ");
} else {
  console.log("Không thỏa điều kiện");
}

// Kiểm tra xem c có phải là số chẵn hoặc số là 15 không
if (c % 2 === 0 || c === 15) {
  console.log("c là số chẵn hoặc là số 15");
} else {
  console.log("Không thỏa điều kiện");
}
```

#### 4. Nullish Coalescing (??)

Nullish coalescing operator (??) trong JavaScript được sử dụng để đánh giá và trả về giá trị bên trái nếu nó là null hoặc undefined; nếu không, nó trả về giá trị bên phải.

```
console.log(null ?? 'hello'); // hello
console.log(undefined ?? 'hello'); // hello
console.log('' ?? 'hello'); // ''
console.log(0 ?? 'hello'); // 0
```

Bài tập:

- Nếu là số dương thì ...
- Nếu là số dương chẵn thì ...
- Nếu là số dương chẵn và lớn hơn 10 thì ...
- Nếu là số dương chẵn chia hết cho 5 và lớn hơn 10 thì ...
- Nếu là số dương chẵn hoặc số âm lẻ thì ...

Giải bài tập:

```
function checkNumber1(number){
    if(n > 0){
    }
}

function checkNumber2(number){
    if(n > 0 && n % 2 === 0){
    }
}

function checkNumber3(number){
    if(n > 0 && n % 2 === 0 && n > 10){
    }
}

function checkNumber4(number){
    if(n > 0 && n % 2 === 0 && n > 10 && n % 5 === 0){
    }
}

function checkNumber5(number){
    const isEvenPositive = n > 0 && n % 2 === 0
    const isOddNegative = n < 0 && n % 2 !== 0

    if(isEvenPositive || isOddNegative){
    }

    if((n > 0 && n % 2 === 0) || (n < 0 && n % 2 !== 0)){
    }
}
```

## 5. Comparison operators:

1. So sánh cùng kiểu dữ liệu
2. So sánh khác kiểu dữ liệu
3. So sánh với null/undefined

Danh sách các phép so sánh trong Javascript:



1. Toán tử bằng (==) và toán tử bằng cùng kiểu dữ liệu (===):

- Toán tử `==` so sánh giá trị của hai biểu thức và trả về `true` nếu chúng bằng nhau. Nó sẽ tự động chuyển đổi kiểu dữ liệu nếu cần.
- Toán tử `===` (gọi là toán tử so sánh nghiêm ngặt hoặc toán tử so sánh tuyệt đối) so sánh giá trị của hai biểu thức và kiểu dữ liệu của chúng, trả về `true` nếu cả giá trị và kiểu dữ liệu đều giống nhau.

## 2. Toán tử khác (`!=`) và toán tử khác cùng kiểu dữ liệu (`!==`):

- Toán tử `!=` so sánh giá trị của hai biểu thức và trả về `true` nếu chúng khác nhau. Nó cũng tự động chuyển đổi kiểu dữ liệu nếu cần.
- Toán tử `!==` so sánh giá trị của hai biểu thức và kiểu dữ liệu của chúng, trả về `true` nếu giá trị hoặc kiểu dữ liệu khác nhau.

## 3. Toán tử lớn hơn (`>`), nhỏ hơn (`<`), lớn hơn hoặc bằng (`>=`), và nhỏ hơn hoặc bằng (`<=`):

- Toán tử `>` so sánh xem một giá trị có lớn hơn giá trị khác hay không và trả về `true` nếu điều này đúng.
- Toán tử `<` so sánh xem một giá trị có nhỏ hơn giá trị khác hay không và trả về `true` nếu điều này đúng.
- Toán tử `>=` so sánh xem một giá trị có lớn hơn hoặc bằng giá trị khác hay không và trả về `true` nếu điều này đúng.
- Toán tử `<=` so sánh xem một giá trị có nhỏ hơn hoặc bằng giá trị khác hay không và trả về `true` nếu điều này đúng.

ví dụ:

```
1 < 2 //true
1 >= 2 //false
'2' == '1' //false
'2' > '1'
3 == '3'
4 === '4'
null == undefined
null == 0
null >= 0
'' == 0
```

Giải thích:

- `1 < 2` là một phép so sánh đúng, vì 1 nhỏ hơn 2. Kết quả: `true`
- `1 >= 2` là một phép so sánh sai, vì 1 không lớn hơn hoặc bằng 2. Kết quả: `false`
- `'2' == '1'` là một phép so sánh giữa hai chuỗi, nên không xác định kiểu dữ liệu. Kết quả: `false`
- `'2' > '1'` cũng là một phép so sánh giữa hai chuỗi, nhưng trong trường hợp này, nó so sánh theo thứ tự từ điển của các ký tự. Vì ký tự '2' trong bảng mã Unicode lớn hơn ký tự '1', nên kết quả là `true`.
- `3 == '3'` là một phép so sánh giữa một số và một chuỗi. Trong trường hợp này, JavaScript thực hiện chuyển đổi kiểu dữ liệu tự động và so sánh giá trị. Kết quả: `true`
- `4 === '4'` là một phép so sánh nghiêm ngặt giữa một số và một chuỗi. Vì kiểu dữ liệu của chúng khác nhau, nên kết quả sẽ là `false`.
- `null == undefined` là một phép so sánh đặc biệt. Trong JavaScript, `null` và `undefined` được coi là bằng nhau (trong ngữ cảnh so sánh đường dẫn). Kết quả: `true`

- `null == 0` là một phép so sánh giữa `null` và một số. Trong quá trình so sánh, `null` sẽ được chuyển đổi thành `0`. Kết quả: `false`
- `null >= 0` cũng là một phép so sánh giữa `null` và một số. Trong quá trình so sánh, `null` sẽ được chuyển đổi thành `0`. Kết quả: `true`
- `" == 0` là một phép so sánh giữa một chuỗi rỗng và số `0`. Trong quá trình so sánh, chuỗi rỗng sẽ được chuyển đổi thành số `0`. Kết quả: `true`

### So sánh cùng kiểu dữ liệu

- So sánh số (number)
- So sánh chuỗi (string)
- So sánh boolean
- So sánh object / array

### So sánh số (number)

```
// Greater than / Less than
1 > 2; // false
1 < 2; // true
// Greater than or equal / Less than or equal
1 >= 2; // false
2 <= 2; // true
// Equal / Not equal
1 == 2; // false
1 != 2; // true
```

### So sánh chuỗi (string)


- Chuỗi cũng có tất cả các operators so sánh như số, tuy nhiên cách so sánh chuỗi hơi khác biệt. So sánh theo từng kí tự một, từ trái sang phải
- Nếu kí tự đầu tiên lớn hơn, thì string đó lớn hơn.
- Nếu kí tự đầu tiên nhỏ hơn, thì string đó nhỏ hơn.
- Nếu kí tự đầu tiên giống nhau thì đi kiểm tra tiếp kí tự tiếp theo.
- Tiếp tục cho đến khi hết chuỗi.

Làm sao biết kí tự nào lớn hơn? --> Tra trong bảng [Unicode UTF-16](#)

```
'a' > 'b' // false
'aa' <= 'b' // true (dài hơn không có nghĩa là lớn hơn)
'abc' > 'abd' // false (because c is less than d)
'a' > 'A' // true
'A' > '9' // true
'easy' == 'easy' // true
'dev frontend' != 'dev' // true
'dev frontend' > 'dev' // true
```

### So sánh boolean



- Boolean values: true / false
- true > false 

```
true > false // true
true == true // true
true == false // false
true != false // true
```

## So sánh object / array

```
const student1 = { name: 'Frontend' };
const student2 = { name: 'Frontend' };
// student1 == student2?
```

- Không như kiểu dữ liệu primitive(nguyên thủy), object không thể so sánh lớn hơn, nhỏ hơn hay bằng được.
- Việc so sánh object thường được định nghĩa một cái hàm riêng đặc biệt để xử lý việc so sánh.
- Còn về so sánh bằng thì mình cần biết thêm về tham trị và tham chiếu (nói sau trong chương object).

## So sánh khác kiểu dữ liệu

Khi so sánh khác kiểu dữ liệu, js sẽ tự động convert giá trị về dạng number để so sánh.

Nên để hiểu được phần này, bạn cần nắm được khi convert một giá trị nào đó về number thì nó sẽ thành giá trị bao nhiêu?

```
123 == '123' // true because '1' will be converted to 1
123 == '0123' // true because '01' will be converted to 1
false == 0 // true because false will be converted to 0
true == 1 // true because true will be converted to 1
'0' == false // true
```

Tham khảo: <https://getify.github.io/coercions-grid/>

## Strict equality (=== and !==)

Giữ nguyên giá trị (không tự động convert kiểu dữ liệu) và so sánh với nhau.

- Nếu khác kiểu dữ liệu, lập tức return false
- Nếu cùng kiểu dữ liệu, thì so sánh như cách thức đã đề cập ở phần so sánh cùng kiểu dữ liệu.

```
111 === '111' // false because they are different data types
0 === false // false
'' === 0 // false
```

```
1 === 1 // true
'abc' === 'abc' // true
```

NOTE: Nên sử dụng === thay vì == để hạn chế rủi ro không đáng có nhé. Nếu muốn dùng == thì phải hiểu thực sự mình đang làm gì nhé!

## So sánh với null/undefined

```
// non-strict ==, null và undefined bằng nhau nhưng không có giá trị nào khác.
null == undefined; // true
// strict check ===
null === undefined; // sai vì chúng là loại khác nhau
// other comparisons
// null sẽ chuyển thành 0
// không xác định sẽ chuyển đổi thành NaN
null > 0 // false
null == 0 // false
null >= 0 // true WHAT???
```

### Tóm tắt:

- Các nhóm so sánh: lớn bé, bằng và không bằng
- Chỉ quan tâm tới kiểu primitive types (kiểu nguyên thủy), còn object/array thì sẽ có cách so sánh riêng.
- So sánh cùng kiểu dữ liệu, number thì như toán học, string thì so sánh từng ký tự dựa theo bảng UTF16.
- So sánh khác kiểu dữ liệu, values sẽ được tự động convert về number để so sánh (type coercion)
- Nên dùng === cho đến khi gặp trường hợp đặc biệt cần dùng ==
- Chỉ có cặp null == undefined cho true, còn so sánh với các giá trị khác đều cho false

### Tham khảo:

- <https://javascript.info/comparison>
- <https://dorey.github.io/JavaScript-Equality-Table/unified/>
- <https://getify.github.io/coercions-grid/>

## 6. if...else:

Trong JavaScript, câu lệnh if...else được sử dụng để thực hiện một hành động nếu điều kiện được đưa ra là đúng (true), và thực hiện một hành động khác nếu điều kiện là sai (false). Cú pháp tổng quát của câu lệnh if...else như sau:

- condition là một giá trị boolean
- Nếu condition là kiểu giá trị khác, nó sẽ tự động chuyển đổi về boolean.

```
if (condition) {
    // Thực thi các câu lệnh khi điều kiện là đúng
} else {
```

```
// Thực thi các câu lệnh khi điều kiện là sai  
}
```

ví dụ:

```
if (condition) doSomething();  
  
if (condition) {  
    doSomething();  
    doSomethingElse();  
}
```

Ngoài ra, câu lệnh if...else cũng có thể được sử dụng với nhiều điều kiện bằng cách sử dụng các câu lệnh else if. Ví dụ:

```
var num = 0;  
  
if (num > 0) {  
    console.log("Số " + num + " là số dương.");  
} else if (num < 0) {  
    console.log("Số " + num + " là số âm.");  
} else {  
    console.log("Số " + num + " là số không.");  
}
```

Hạn chế sử dụng else, bỏ được hãy bỏ với if...return\

```
// bad  
if (condition) {  
    doSomething();  
} else if (anotherCondition) {  
    doSomethingElse();  
} else {  
    doSomethingFinally();  
}  
  
//good  
function yourFunction() {  
    if (condition) {  
        doSomething();  
        return;  
    }  
  
    if (anotherCondition) {  
        doSomethingElse();  
        return;  
    }  
}
```

```
    doSomethingFinally();  
}
```

## Nested if...else

Nested if...else là một cấu trúc lồng nhau của câu lệnh if...else trong JavaScript, có nghĩa là một câu lệnh if...else được đặt trong một khối if hoặc else của một câu lệnh if...else khác. Điều này cho phép kiểm tra nhiều điều kiện phức tạp hơn.

```
var score = 75;  
  
if (score >= 90) {  
    console.log("Grade: A");  
} else {  
    if (score >= 80) {  
        console.log("Grade: B");  
    } else {  
        if (score >= 70) {  
            console.log("Grade: C");  
        } else {  
            console.log("Grade: D");  
        }  
    }  
}
```

Tuy nested if...else có thể giúp xử lý các tình huống phức tạp, nhưng nó cũng có nhược điểm:

- Khó đọc và hiểu: Khi sử dụng quá nhiều nested if...else, mã sẽ trở nên khó đọc và hiểu. Cấu trúc lồng nhau có thể làm cho mã trở nên phức tạp và khó duy trì.
- Dễ gây lỗi: Với mỗi cấp độ của nested if...else, có thêm cơ hội để mắc lỗi. Điều này làm cho việc debug trở nên khó khăn hơn.
- Khó mở rộng: Khi bạn cần thêm điều kiện hoặc logic, sử dụng nested if...else có thể làm cho việc mở rộng mã trở nên phức tạp hơn.

## 7. switch...case:

Cấu trúc switch...case trong JavaScript cung cấp một cách để kiểm tra một biến cho nhiều giá trị khác nhau và thực hiện các hành động tương ứng với mỗi giá trị.

Viết hàm nhận vào số nguyên dương từ 1 tới 12, trả về chuỗi là tên của tháng tương ứng.

Ví dụ: printMonth(1) --> Ja

```
// using if...else  
function printMonth(n) {  
    let month = '';  
    if (n === 1) month = 'Jan'
```

```
else if (n === 2) month = 'Feb'
else if (n === 3) month = 'Mar'
else if (n === 4) month = 'Apr'
else if (n === 5) month = 'May'
else if (n === 6) month = 'Jun'
else if (n === 7) month = 'Jul'
else if (n === 8) month = 'Aug'
else if (n === 9) month = 'Sep'
else if (n === 10) month = 'Oct'
else if (n === 11) month = 'Nov'
else if (n === 12) month = 'Dec'
else month = 'Invalid number'
return month;
}
```

```
// using switch...case
function printMonth(n) {
  let month = '';
  switch (n) {
    case 1: {
      month = 'Jan';
      break;
    }

    case 2: {
      month = 'Feb';
      break;
    }

    case 3: {
      month = 'Mar';
      break;
    }

    // ... similar for month 4 -> 11

    case 12: {
      month = 'Dec';
      break;
    }
    default: {
      month = 'Invalid number'
    }
  }
  return month;
}
```

Lưu ý:

- Nếu điều kiện là tập các giá trị xác định, hãy dùng switch...case thay vì if...else

- Nếu điều kiện là dạng khoảng (lớn/bé hơn) thì nên dùng if...else

## 8. Bài tập thực hành.

1. viết 1 function kiểm tra số dương chẵn, nếu đúng trả về true sai trả về false.

2. Viết 1 function kiểm tra điểm của học sinh:

- `mark > 8 -> 'Excellence'`
- `7 <= mark <= 8 -> 'Good'`
- `4 <= mark <= 6 -> 'Not Good'`
- `mark < 4 -> 'Bad'`