
 [hngnaig](#) / [javascript-style-guide](#)

Watch2

Star17

Fork11


[Code](#) [Issues0](#) [Pull requests0](#) [Projects0](#) [Insights](#)



Join GitHub today

GitHub is home to over 20 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)



Dismiss

Airbnb JavaScript Style Guide

[javascript](#) [arrow-functions](#) [styleguide](#)

3 commits




1 branch

0 releases

1 contributor

MIT

Branch: master [New pull request](#) [Find file](#) [Clone or download](#)

 hngnaig committed on GitHub	Deprecated project.	Latest commit 102b8c4 on May 15
 LICENSE	Initial commit	8 months ago
 README.md	Deprecated project.	3 months ago

README.md

(Deprecated) Airbnb JavaScript Style Guide() {

Bản cập nhật mới nhất [Javascript style guide](#)

Table of Contents

- [Types](#)
- [References](#)
- [Objects](#)
- [Arrays](#)
- [Destructuring](#)
- [Strings](#)
- [Functions](#)
- [Arrow Functions](#)
- [Constructors](#)
- [Modules](#)
- [Iterators and Generators](#)
- [Properties](#)
- [Variables](#)
- [Hoisting](#)

15. [Comparison Operators & Equality](#)
16. [Blocks](#)
17. [Comments](#)
18. [Whitespace](#)
19. [Commas](#)
20. [Semicolons](#)
21. [Type Casting & Coercion](#)
22. [Naming Conventions](#)
23. [Accessors](#)
24. [Events](#)
25. [jQuery](#)
26. [ECMAScript 5 Compatibility](#)
27. [ECMAScript 6 Styles](#)
28. [Testing](#)
29. [Performance](#)
30. [Resources](#)
31. [In the Wild](#)
32. [Translation](#)
33. [The JavaScript Style Guide Guide](#)
34. [Chat With Us About JavaScript](#)
35. [Contributors](#)
36. [License](#)

Types

- **1.1 Primitives:** Khi bạn truy cập các kiểu dữ liệu nguyên thủy, bạn làm việc trực tiếp với giá trị của biến (tham trị).

- `string`
- `number`
- `boolean`
- `null`
- `undefined`

```
const foo = 1;
let bar = foo;

bar = 9;

console.log(foo, bar); // => 1, 9
```

- **1.2 Complex:** Khi bạn làm việc với các kiểu dữ liệu phức tạp, bạn làm việc gián tiếp với giá trị của biến thông qua địa chỉ vùng nhớ của nó (tham chiếu).

- `object`
- `array`
- `function`

```
const foo = [1, 2];
const bar = foo;

bar[0] = 9;

console.log(foo[0], bar[0]); // => 9, 9
```

References

- [2.1](#) Sử dụng `const` cho khai báo hằng; tránh sử dụng `var`.

Khi sử dụng `const` chúng ta không thể thay đổi giá trị của nó. Điều này giúp chúng ta ngăn chặn các lỗi có thể xảy ra

```
```javascript
// Không tốt
var a = 1;
var b = 2;

// Tốt
const a = 1;
const b = 2;
```
```

- [2.2](#) Để thay đổi các thông số, sử dụng `let` và tránh sử dụng `var`.

Vì `let` nó chỉ có nghĩa trong một phạm vi nhất định (Block-Scope), không giống như `var` có nghĩa trong một hàm (Function-Scope).

```
```javascript
// Không tốt
var count = 1;
if (true) {
 count += 1;
}

// Tốt.
let count = 1;
if (true) {
 count += 1;
}
```
```

- [2.3](#) `let` và `const` có giá trị trong một phạm vi nhất định (Block-Scope).

```
{
  let a = 1;
  const b = 1;
}
console.log(a); // ReferenceError Khi ngoài phạm vi của biến sẽ không thể sử dụng biến đó.
console.log(b); // ReferenceError Khi ngoài phạm vi của biến sẽ không thể sử dụng biến đó.
```

Objects

- [3.1](#) Dùng `{}` để tạo một đối tượng.

```
// Không tốt
const item = new Object();

// Tốt
const item = {};
```

- [3.2](#) Không sử dụng các từ khoá mặc định để làm thuộc tính.

```
// Không tốt
const superman = {
  default: { clark: 'kent' },
  private: true,
};

// Tốt
const superman = {
  defaults: { clark: 'kent' },
  hidden: true,
};
```

- 3.3 Dùng những từ ngữ đồng nghĩa với thuật ngữ dành riêng.

```
// Không tốt
const superman = {
  class: 'alien', // `class` từ khoá của hệ thống
};

// Không tốt
const superman = {
  klass: 'alien', `klass` không có ý nghĩa gì trong trường hợp này
};

// Tốt
const superman = {
  type: 'alien',
};
```

- 3.4 Sử dụng tên thuộc tính khi tạo đối tượng với tên thuộc tính động.

Việc này giúp chúng ta định nghĩa tất cả các thuộc tính của đối tượng một lần duy nhất.

```
```javascript

function getKey(k) {
 return `a key named ${k}`;
}

// Không tốt
const obj = {
 id: 5,
 name: 'San Francisco',
};
obj[getKey('enabled')] = true;

//
const obj = {
 id: 5,
 name: 'San Francisco',
 [getKey('enabled')]: true,
};
```
```

- 3.5 Sử dụng cách khai báo phương thức rút ngắn (Property Value Shorthand).

```
// Không tốt
const atom = {
  value: 1,

  addValue: function (value) {
    return atom.value + value;
  },
};
```

```
// Tốt
const atom = {
  value: 1,

  addValue(value) {
    return atom.value + value;
  },
};
```

- 3.6 Sử dụng cách khai báo rút ngắn cho thuộc tính (Object Method).

Cách viết ngắn gọn và dễ hiểu hơn.

```
```javascript
const lukeSkywalker = 'Luke Skywalker';

// bad
const obj = {
 lukeSkywalker: lukeSkywalker,
};

// good
const obj = {
 lukeSkywalker,
};
```
```

- 3.7 Gom nhóm các thuộc tính được khai báo rút ngắn đặt lên đầu của mỗi đối tượng.

```
const anakinSkywalker = 'Anakin Skywalker';
const lukeSkywalker = 'Luke Skywalker';

// Không tốt
const obj = {
  episodeOne: 1,
  twoJediWalkIntoACantina: 2,
  lukeSkywalker,
  episodeThree: 3,
  mayTheFourth: 4,
  anakinSkywalker,
};

// Tốt
const obj = {
  lukeSkywalker, // Thuộc tính được khai báo ngắn gọn
  anakinSkywalker, // Thuộc tính được khai báo ngắn gọn
  episodeOne: 1,
  twoJediWalkIntoACantina: 2,
  episodeThree: 3,
  mayTheFourth: 4,
};
```

[↑ Trở lại đầu trang](#)

Arrays

- 4.1 Sử dụng `[]` để khai báo một mảng.

```
// Không tốt
const items = new Array();

// Tốt
const items = [];
```

- 4.2 Dùng `Array#push` để thêm một phần tử vào mảng thay vì thêm trực tiếp.

```
const someStack = [];  
  
// Không tốt  
someStack[someStack.length] = 'abracadabra';  
  
// Tốt  
someStack.push('abracadabra');
```

- 4.3 Dùng `...` (Spreads) để sao chép mảng.

```
// Không tốt  
const len = items.length;  
const itemsCopy = [];  
let i;  
  
for (i = 0; i < len; i++) {  
  itemsCopy[i] = items[i];  
}  
  
// Tốt  
const itemsCopy = [...items];
```

- 4.4 Dùng `Array#from` để chuyển đổi từ đối tượng sang mảng.

```
const foo = document.querySelectorAll('.foo');  
const nodes = Array.from(foo);
```

[↑ Trở lại đầu trang](#)

Destructuring

- 5.1 Dùng `Destructuring` để chuyển giá trị từng thuộc tính của đối tượng vào một biến.

Điều này giúp giảm bớt việc dùng các biến tạm thời để lưu các thuộc tính trong đối tượng.

```
// Không tốt  
function getFullName(user) {  
  const firstName = user.firstName;  
  const lastName = user.lastName;  
  
  return `${firstName} ${lastName}`;  
}  
  
// Tốt  
function getFullName(obj) {  
  const { firstName, lastName } = obj;  
  return `${firstName} ${lastName}`;  
}  
  
// Tốt nhất  
function getFullName({ firstName, lastName }) {  
  return `${firstName} ${lastName}`;  
}
```

- 5.2 Dùng `destructuring` cho mảng.

```
const arr = [1, 2, 3, 4];  
  
// Không tốt  
const first = arr[0];
```

```
const second = arr[1];

// Tốt
const [first, second] = arr;
```

- 5.3 Dùng destructuring cho nhiều giá trị trả về, không dùng destructuring array.

Cách dùng này giúp chúng ta có thể thêm một thuộc tính mới hoặc sắp xếp thứ tự trả về mà không gây ảnh hưởng cho các hàm khác.

```
``javascript
// Không tốt
function processInput(input) {
  // then a miracle occurs
  return [left, right, top, bottom];
}

// the caller needs to think about the order of return data
const [left, __, top] = processInput(input);

// Tốt
function processInput(input) {
  // then a miracle occurs
  return { left, right, top, bottom };
}

// the caller selects only the data they need
const { left, right } = processInput(input);
````
```

[↑ Trở lại đầu trang](#)

## Strings

- 6.1 Sử dụng dấu nháy đơn '' đối với chuỗi.

```
// Không tốt
const name = "Capt. Janeway";

// Tốt
const name = 'Capt. Janeway';
```

- 6.2 Khi chuỗi dài hơn 100 kí tự nên chia nhiều dòng và nối chuỗi đó lại.
- 6.3 Ghi chú: Việc áp dụng nối chuỗi nhiều sẽ gây ảnh hưởng tới hiệu năng. [jsPerf](#) & [Thảo luận](#).

```
// Không tốt
const errorMessage = 'This is a super long error that was thrown because of Batman. When you stop to think about

// Không tốt
const errorMessage = 'This is a super long error that was thrown because \
of Batman. When you stop to think about how Batman had anything to do \
with this, you would get nowhere \
fast.';

// Tốt
const errorMessage = 'This is a super long error that was thrown because ' +
 'of Batman. When you stop to think about how Batman had anything to do ' +
 'with this, you would get nowhere fast.';
```

- 6.4 Dùng template thay vì dùng cách nối chuỗi.

Dùng `template` sẽ giúp chúng ta dễ đọc, cú pháp ngắn gọn.

```
```javascript
// Không tốt
function sayHi(name) {
  return 'How are you, ' + name + '?';
}

// Không tốt
function sayHi(name) {
  return ['How are you, ', name, '?'].join();
}

// Tốt
function sayHi(name) {
  return `How are you, ${name}?`;
}
```
```

- 6.5 Không bao giờ sử dụng `eval()` cho chuỗi.

† [Trở lại đầu trang](#)

## Functions

- 7.1 Sử dụng `Function declarations` thay vì `Function expressions`.

Function declarations được đặt tên rõ ràng, do đó có thể xác định nó ở `call stacks`. Luôn luôn dùng [Arrow Functions](#) với `Function expressions`.

```
```javascript
// không tốt
const foo = function () {
};

// Tốt
function foo() {
}
```
```

- 7.2 Function expressions:

```
// immediately-invoked function expression (IIFE)
(() => {
 console.log('Welcome to the Internet. Please follow me.');
```

```
})();
```

- 7.3 Không được khai báo một hàm khi sử dụng các câu điều kiện (if, while, ...). Thay vào đó lưu hàm vào một biến cụ thể.
- 7.4 Ghi chú: ECMA-262 định nghĩa `block` như là danh sách các câu lệnh. [Read ECMA-262's note on this issue.](#)

```
// Không tốt
if (currentUser) {
 function test() {
 console.log('Nope.');
```

```
 }
}
```

```
// Tốt
let test;
if (currentUser) {
 test = () => {
 console.log('Yup.');
```



```
};
}
```

- 7.5 Không được khai báo các tham số của hàm trùng với `arguments`.

```
// Không tốt
function nope(name, options, arguments) {
 // ...stuff...
}

// Tốt
function yup(name, options, args) {
 // ...stuff...
}
```

- 7.6 Không được dùng `arguments`, dùng `...` (Spreads) thay thế.

`...` sẽ chuyển các `arguments` thành một mảng các giá trị.

```
```javascript  
// Không tốt  
function concatenateAll() {  
  const args = Array.prototype.slice.call(arguments);  
  return args.join('');  
}  
  
// Tốt  
function concatenateAll(...args) {  
  return args.join('');  
}  
```
```

- 7.7 Sử dụng cách truyền tham số mặc định.

```
// Không tốt, không bao giờ áp dụng kiểu này
function handleThings(opts) {
 opts = opts || {};
 // ...
}

// Không tốt
function handleThings(opts) {
 if (opts === void 0) {
 opts = {};
 }
 // ...
}

// Tốt
function handleThings(opts = {}) {
 // ...
}
```

- 7.8 Tránh đặt nó là các thông số mặc định.

```
var b = 1;
// Không tốt
function count(a = b++) {
 console.log(a);
}
count(); // 1
count(); // 2
count(3); // Nó được định nghĩa như là tham số thứ ba, mặc định sẽ không được thực thi (= b++ không chạy)
count(); // 3
```

- 7.9 Luôn đặt tham số mặc định ở cuối.

```
// Không tốt
function handleThings(opts = {}, name) {
 // ...
}

// Tốt
function handleThings(name, opts = {}) {
 // ...
}
```

- 7.10 Không bao giờ dùng cách khởi tạo một hàm bằng cách dùng `new Function`.

Cách này đồng nghĩa với việc dùng `eval()`.

```
// Không tốt
var add = new Function('a', 'b', 'return a + b');

// Không tốt
var subtract = Function('a', 'b', 'return a - b');
```

[↑ Trở lại đầu trang](#)

## Arrow Functions

- 8.1 Một khi bạn sử dụng `Function expressions` (Anonymous function), thì nên dùng `Arrow Functions` hoặc `=>`.

```
// Không tốt
[1, 2, 3].map(function (x) {
 const y = x + 1;
 return x * y;
});

// Tốt
[1, 2, 3].map((x) => {
 const y = x + 1;
 return x * y;
});
```

- 8.2 Nếu chỉ một câu lệnh tính toán thì có thể không dùng `{}`, nhưng khi có nhiều câu lệnh thì nên dùng `{}` và dùng `return` để trả về kết quả cuối cùng.

```
// Tốt
[1, 2, 3].map(number => `A string containing the ${number}.`);

// Không tốt
[1, 2, 3].map(number => {
 const nextNumber = number + 1;
 `A string containing the ${nextNumber}.`;
});

// Tốt
[1, 2, 3].map(number => {
 const nextNumber = number + 1;
 return `A string containing the ${nextNumber}.`;
});
```

- 8.3 Khi có nhiều chuỗi có nhiều dòng nên bao chuỗi đó trong dấu `()`.

```
// Không tốt
[1, 2, 3].map(number => 'As time went by, the string containing the ' +
 `${number} became much longer. So we needed to break it over multiple ` +
 'lines.'
);

// Tốt
[1, 2, 3].map(number => (
 `As time went by, the string containing the ${number} became much ` +
 'longer. So we needed to break it over multiple lines.'
));
```

- 8.4 Nếu trong hàm chỉ có một câu lệnh duy nhất có thể không cần dùng `()`.

```
// Tốt
[1, 2, 3].map(x => x * x);

// Tốt
[1, 2, 3].reduce((y, x) => x + y);
```

[↑ Trở lại đầu trang](#)

## Constructors

- 9.1 Luôn luôn dùng `class`. Không nên dùng `prototype`.

```
// Không tốt
function Queue(contents = []) {
 this._queue = [...contents];
}
Queue.prototype.pop = function() {
 const value = this._queue[0];
 this._queue.splice(0, 1);
 return value;
}

// Tốt
class Queue {
 constructor(contents = []) {
 this._queue = [...contents];
 }
 pop() {
 const value = this._queue[0];
 this._queue.splice(0, 1);
 return value;
 }
}
```

- 9.2 Sử dụng `extends` để tạo một lớp kế thừa.

```
// Không tốt
const inherits = require('inherits');
function PeekableQueue(contents) {
 Queue.apply(this, contents);
}
inherits(PeekableQueue, Queue);
PeekableQueue.prototype.peek = function() {
 return this._queue[0];
}

// Tốt
class PeekableQueue extends Queue {
```

```

 peek() {
 return this._queue[0];
 }
 }
}

```

- 9.3 Phương thức có thể trả về `this` .

```

// Không tốt
Jedi.prototype.jump = function() {
 this.jumping = true;
 return true;
};

Jedi.prototype.setHeight = function(height) {
 this.height = height;
};

const luke = new Jedi();
luke.jump(); // => true
luke.setHeight(20); // => undefined

// Tốt
class Jedi {
 jump() {
 this.jumping = true;
 return this;
 }

 setHeight(height) {
 this.height = height;
 return this;
 }
}

const luke = new Jedi();

luke.jump()
 .setHeight(20);

```

- 9.4 Có thể mở rộng phương thức `toString()` .

```

class Jedi {
 constructor(options = {}) {
 this.name = options.name || 'no name';
 }

 getName() {
 return this.name;
 }

 toString() {
 return `Jedi - ${this.getName()}`;
 }
}

```

[↑ Trở lại đầu trang](#)

## Modules

- 10.1 Luôn luôn dùng ( `import` / `export` ) khi làm việc với `modules` thay thế kiểu `modules` truyền thống `require/module.exports` .

```

// Không tốt
const AirbnbStyleGuide = require('./AirbnbStyleGuide');

```

```

module.exports = AirbnbStyleGuide.es6;

// ok
import AirbnbStyleGuide from './AirbnbStyleGuide';
export default AirbnbStyleGuide.es6;

// Tốt nhất
import { es6 } from './AirbnbStyleGuide';
export default es6;

```

- 10.2 Không được dùng wildcard imports.

```

// Không tốt
import * as AirbnbStyleGuide from './AirbnbStyleGuide';

// Tốt
import AirbnbStyleGuide from './AirbnbStyleGuide';

```

- 10.3 Không được dùng export trực tiếp từ import.

Bởi vì tách import và export giúp cho việc đọc dễ, có ý nghĩa hơn.

```

```javascript
// Không tốt
// filename es6.js
export { es6 as default } from './airbnbStyleGuide';

// Tốt
// filename es6.js
import { es6 } from './AirbnbStyleGuide';
export default es6;
```

```

[↑ Trở lại đầu trang](#)

## Iterators and Generators

- 11.1 Không được dùng vòng lặp thay vào đó dùng map() hoặc reduce thay thế for-of.

```

const numbers = [1, 2, 3, 4, 5];

// Không tốt
let sum = 0;
for (let num of numbers) {
 sum += num;
}

sum === 15;

// Tốt
let sum = 0;
numbers.forEach((num) => sum += num);
sum === 15;

// Tốt nhất
const sum = numbers.reduce((total, num) => total + num, 0);
sum === 15;

```

- 11.2 Không được dùng generators ở thời điểm hiện tại.

Có lỗi khi chuyển sang ES5.

[↑ Trở lại đầu trang](#)

## Properties

- 12.1 Sử dụng `.` khi truy cập vào một biến.

```
const luke = {
 jedi: true,
 age: 28,
};

// Không tốt
const isJedi = luke['jedi'];

// Tốt
const isJedi = luke.jedi;
```

- 12.2 Sử dụng `[]` để truy cập thuộc tính đối với biến.

```
const luke = {
 jedi: true,
 age: 28,
};

function getProp(prop) {
 return luke[prop];
}

const isJedi = getProp('jedi');
```

[↑ Trở lại đầu trang](#)

## Variables

- 13.1 Luôn luôn sử dụng `const` để khai báo một biến.

```
// Không tốt
superPower = new SuperPower();

// Tốt
const superPower = new SuperPower();
```

- 13.2 Dùng mỗi `const` cho việc khai báo một biến.

```
// Không tốt
const items = getItems(),
 goSportsTeam = true,
 dragonball = 'z';

// Không tốt
const items = getItems(),
 goSportsTeam = true;
 dragonball = 'z';

// Tốt
const items = getItems();
const goSportsTeam = true;
const dragonball = 'z';
```

- 13.3 Gộp nhóm biến theo `const` s và `let` s.

```
// Không tốt
let i, len, dragonball,
```

```

 items = getItem(),
 goSportsTeam = true;

// Không tốt
let i;
const items = getItem();
let dragonball;
const goSportsTeam = true;
let len;

// Tốt
const goSportsTeam = true;
const items = getItem();
let dragonball;
let i;
let length;

```

- [13.4](#) Khai báo biến khi cần thiết và đặt chúng ở đúng nơi.

`let` và `const` là `block scoped` và không phải `function scoped`.

```

```javascript
// Tốt
function() {
  test();
  console.log('doing stuff..');

  //...other stuff..

  const name = getName();

  if (name === 'test') {
    return false;
  }

  return name;
}

// Không tốt - hàm không cần thiết
function(hasName) {
  const name = getName();

  if (!hasName) {
    return false;
  }

  this.setFirstName(name);

  return true;
}

// good
function(hasName) {
  if (!hasName) {
    return false;
  }

  const name = getName();
  this.setFirstName(name);

  return true;
}
```

```

[↑ Trở lại đầu trang](#)

## Hoisting

- 14.1 `var` được khai báo trước ở đầu trong phạm vi của biến hoặc hàm. `const` và `let` được dùng với một khái niệm mới Temporal Dead Zones (TDZ). `typeof` is no longer safe.

```
function example() {
 console.log(notDefined); // => ReferenceError
}

// Khai báo một biến sau khai biến đó đã được gọi,
// trong trường hợp này biến sẽ không `hoisted`
function example() {
 console.log(declaredButNotAssigned); // => undefined
 var declaredButNotAssigned = true;
}

// Biến được khai báo ở đầu
function example() {
 let declaredButNotAssigned;
 console.log(declaredButNotAssigned); // => undefined
 declaredButNotAssigned = true;
}

// sử dụng `const` and `let`
function example() {
 console.log(declaredButNotAssigned); // => throws a ReferenceError
 console.log(typeof declaredButNotAssigned); // => throws a ReferenceError
 const declaredButNotAssigned = true;
}
```

- 14.2 Anonymous function được khai báo bằng một biến.

```
function example() {
 console.log(anonymous); // => undefined

 anonymous(); // => TypeError anonymous is not a function

 var anonymous = function() {
 console.log('anonymous function expression');
 };
}
```

- 14.3 Named Function expressions - Việc thông báo này hoạt động bằng tên hàm. Kết quả như ví dụ trước.

```
function example() {
 console.log(named); // => undefined

 named(); // => TypeError named is not a function

 superPower(); // => ReferenceError superPower is not defined

 var named = function superPower() {
 console.log('Flying');
 };
}

// Kết quả giống như khi dùng tên hàm
// trùng với tên biến
function example() {
 console.log(named); // => undefined

 named(); // => TypeError named is not a function

 var named = function named() {
 console.log('named');
 }
}
```



- [14.4 Function declarations](#) - Đối với hàm mà không có các giá trị đầu vào cho biến.

```
function example() {
 superPower(); // => Flying

 function superPower() {
 console.log('Flying');
 }
}
```

- [JavaScript Scoping & Hoisting](#) by Ben Cherry.

[↑ Trở lại đầu trang](#)

## Comparison Operators & Equality

- [15.1](#) Sử dụng `===` và `!==` thay cho `==` và `!=`.
- [15.2](#) Khi dùng `if` các loại đối tượng sẽ được chuyển đổi sang kiểu `Boolean`:
  - **Objects** chuyển thành **true**
  - **Undefined** chuyển thành **false**
  - **Null** chuyển thành **false**
  - **Booleans** chuyển thành **the value of the boolean**
  - **Numbers** chuyển thành **false** Nếu `+0`, `-0`, or `NaN`, ngược lại **true**
  - **Strings** chuyển thành **false** Nếu là chuỗi rỗng `''`, ngược lại **true**

```
if ([0]) {
 // true
 // `array` là `object`, `objects` chuyển thành true
}
```

- [15.3](#) Sử dụng ngắn gọn.

```
// Không tốt
if (name !== '') {
 // ...stuff...
}

// Tốt
if (name) {
 // ...stuff...
}

// Không tốt
if (collection.length > 0) {
 // ...stuff...
}

// Tốt
if (collection.length) {
 // ...stuff...
}
```

- [15.4 Truth Equality and JavaScript](#) by Angus Croll.

[↑ Trở lại đầu trang](#)

## Blocks

- [16.1](#) Sử dụng `{}`.

```

// Không tốt
if (test)
 return false;

// Tốt
if (test) return false;

// Tốt
if (test) {
 return false;
}

// Không tốt
function() { return false; }

// Tốt
function() {
 return false;
}

```

- 16.2 Nếu dùng nhiều câu điều kiện `if` và `else`, đặt `else` cùng dòng với dấu `}` của `if`.

```

// Không tốt
if (test) {
 thing1();
 thing2();
}
else {
 thing3();
}

// Tốt
if (test) {
 thing1();
 thing2();
} else {
 thing3();
}

```

[↑ Trở lại đầu trang](#)

## Comments

- 17.1 Sử dụng `/** ... */` khi cần chú thích nhiều. Nên mô tả đầy đủ như `types`, `values` của nhiều tham số, giá trị trả về là gì.

```

// Không tốt
// make() returns a new element
// based on the passed in tag name
//
// @param {String} tag
// @return {Element} element
function make(tag) {

 // ...stuff...

 return element;
}

// Tốt
/**
 * make() returns a new element
 * based on the passed in tag name
 *
 * @param {String} tag

```

```

* @return {Element} element
*/
function make(tag) {

 // ...stuff...

 return element;
}

```

- 17.2 Sử dụng `//` khi chú thích một dòng duy nhất. Cách một dòng đối với câu lệnh phía trước khi sử dụng `//` để chú thích.

```

// Không tốt
const active = true; // is current tab

// Tốt
// is current tab
const active = true;

// Tốt
function getType() {
 console.log('fetching type...');
 // set the default type to 'no type'
 const type = this._type || 'no type';

 return type;
}

// Tốt
function getType() {
 console.log('fetching type...');

 // set the default type to 'no type'
 const type = this._type || 'no type';

 return type;
}

// Tốt
function getType() {
 // set the default type to 'no type'
 const type = this._type || 'no type';

 return type;
}

```

- 17.3 Thêm tiền tố phía trước `FIXME` hoặc `TODO` để những người trong cùng một team có thể dễ đọc hiểu code. `FIXME -- need to figure this out` hoặc `TODO -- need to implement`.
- 17.4 Use `// FIXME:` to annotate problems.

```

class Calculator extends Abacus {
 constructor() {
 super();

 // FIXME: shouldn't use a global here
 total = 0;
 }
}

```

- 17.5 Dùng `// TODO:` chú thích cách giải quyết vấn đề.

```

class Calculator extends Abacus {
 constructor() {
 super();

```

```
// TODO: total should be configurable by an options param
this.total = 0;
}
}
```

[↑ Trở lại đầu trang](#)

## Whitespace

- [18.1](#) Dùng 2 spaces thay vì 4.

```
// Không tốt
function() {
 ...const name;
}

// Tốt
function() {
 .const name;
}

// Tốt
function() {
 ..const name;
}
```

- [18.2](#) Thêm dấu cách trước { .

```
// Không tốt
function test(){
 console.log('test');
}

// Tốt
function test() {
 console.log('test');
}

// Không tốt
dog.set('attr',{
 age: '1 year',
 breed: 'Bernese Mountain Dog',
});

// Tốt
dog.set('attr', {
 age: '1 year',
 breed: 'Bernese Mountain Dog',
});
```

- [18.3](#) Thêm 1 khoảng cách sau các câu điều kiện ( if , while ...). Không nên có khoảng cách trong Function Arguments .

```
// Không tốt
if(isJedi) {
 fight ();
}

// Tốt
if (isJedi) {
 fight();
}

// Không tốt
```

```
function fight () {
 console.log ('Swoosh!');
}

// Tốt
function fight() {
 console.log('Swoosh!');
}
```

- 18.4 Đối với các phép tính (+, -, \*, / ...) thêm khoảng cách trước và sau các phép tính đó.

```
// Không tốt
const x=y+5;

// Tốt
const x = y + 5;
```

- 18.5 Thêm một dòng trống khi file đó kết thúc.

```
// Không tốt
(function(global) {
 // ...stuff...
})(this);
```

```
// Không tốt
(function(global) {
 // ...stuff...
})(this);
```

```
// Tốt
(function(global) {
 // ...stuff...
})(this);
```

- 18.6 Sử dụng indentation khi gọi nhiều methods cùng một lúc.

```
// Không tốt
$('#items').find('.selected').highlight().end().find('.open').updateCount();

// Không tốt
$('#items').
 find('.selected').
 highlight().
 end().
 find('.open').
 updateCount();

// Tốt
$('#items')
 .find('.selected')
 .highlight()
 .end()
 .find('.open')
 .updateCount();

// Không tốt
const leds = stage.selectAll('.led').data(data).enter().append('svg:svg').class('led', true)
 .attr('width', (radius + margin) * 2).append('svg:g')
 .attr('transform', 'translate(' + (radius + margin) + ',' + (radius + margin) + ')')
 .call(tron.led);

// Tốt
```

```
const leds = stage.selectAll('.led')
 .data(data)
 .enter().append('svg:svg')
 .classed('led', true)
 .attr('width', (radius + margin) * 2)
 .append('svg:g')
 .attr('transform', 'translate(' + (radius + margin) + ',' + (radius + margin) + ')')
 .call(tron.led);
```

- 18.7 Thêm một dòng trống sau `{}` và bắt đầu một câu lệnh tiếp theo.

```
// Không tốt
if (foo) {
 return bar;
}
return baz;
```

```
// Tốt
if (foo) {
 return bar;
}
```

```
return baz;
```

```
// Không tốt
const obj = {
 foo() {
 },
 bar() {
 },
};
return obj;
```

```
// Tốt
const obj = {
 foo() {
 },

 bar() {
 },
};
```

```
return obj;
```

```
// Không tốt
const arr = [
 function foo() {
 },
 function bar() {
 },
];
return arr;
```

```
// Tốt
const arr = [
 function foo() {
 },

 function bar() {
 },
];
```

```
return arr;
```

- 18.8 Trong một `blocks` không được thêm dòng trống.

```
// Không tốt
function bar() {

 console.log(foo);

}

// Không tốt
if (baz) {

 console.log(qux);
} else {
 console.log(foo);

}

// Tốt
function bar() {
 console.log(foo);
}

// Tốt
if (baz) {
 console.log(qux);
} else {
 console.log(foo);
}
```

[↑ Trở lại đầu trang](#)

## Commas

- [19.1](#) Dùng `,` ở đầu: Không

```
// Không tốt
const story = [
 once
, upon
, aTime
];

// Tốt
const story = [
 once,
 upon,
 aTime,
];

// Không tốt
const hero = {
 firstName: 'Ada'
, lastName: 'Lovelace'
, birthYear: 1815
, superPower: 'computers'
};

// Tốt
const hero = {
 firstName: 'Ada',
 lastName: 'Lovelace',
 birthYear: 1815,
 superPower: 'computers',
};
```

- [19.2](#) Thêm `trailing comma`: Yup.

```
// Không tốt - git diff without trailing comma
const hero = {
 firstName: 'Florence',
 - lastName: 'Nightingale'
 + lastName: 'Nightingale',
 + inventorOf: ['coxcomb graph', 'modern nursing']
};

// Tốt - git diff with trailing comma
const hero = {
 firstName: 'Florence',
 lastName: 'Nightingale',
 + inventorOf: ['coxcomb chart', 'modern nursing'],
};

// Không tốt
const hero = {
 firstName: 'Dana',
 lastName: 'Scully'
};

const heroes = [
 'Batman',
 'Superman'
];

// Tốt
const hero = {
 firstName: 'Dana',
 lastName: 'Scully',
};

const heroes = [
 'Batman',
 'Superman',
];
```

[↑ Trở lại đầu trang](#)

## Semicolons

- [20.1](#) Yup.

```
// Không tốt
(function() {
 const name = 'Skywalker'
 return name
})();

// Tốt
(() => {
 const name = 'Skywalker';
 return name;
})();

// Tốt (Trong trường hợp này `` để tránh xung đột giữa 2 `IIFEs` liên tiếp nhau.)
;(() => {
 const name = 'Skywalker';
 return name;
})();
```

[Read more.](#)

[↑ Trở lại đầu trang](#)



## Type Casting & Coercion

- [21.1](#) Áp dụng các kiểu chuyển đổi.
- [21.2](#) Strings:

```
// => this.reviewScore = 9;

// Không tốt
const totalScore = this.reviewScore + '';

// Tốt
const totalScore = String(this.reviewScore);
```

- [21.3](#) Numbers: Sử dụng `Number` cho `casting` và `parseInt` luôn luôn dùng với `radix` khi chuyển từ chuỗi sang số.

```
const inputValue = '4';

// Không tốt
const val = new Number(inputValue);

// Không tốt
const val = +inputValue;

// Không tốt
const val = inputValue >> 0;

// Không tốt
const val = parseInt(inputValue);

// Tốt
const val = Number(inputValue);

// Tốt
const val = parseInt(inputValue, 10);
```

- [21.4](#) Sử dụng `Bitshift` [Lý do](#), và chú thích đầy đủ khi dùng Bitshift

```
// Tốt
/**
 * parseInt was the reason my code was slow.
 * Bitshifting the String to coerce it to a
 * Number made it a lot faster.
 */
const val = inputValue >> 0;
```

- [21.5](#) Ghi chú: Cẩn thận khi dùng `Bitshift`. `Bitshift` luôn trả về 32-bit integer ([source](#)). Khi số lớn hơn 32 bits sẽ dẫn đến một số lỗi. [Thảo luận](#). Số lớn nhất 32-bit Int là 2,147,483,647:

```
2147483647 >> 0 //=> 2147483647
2147483648 >> 0 //=> -2147483648
2147483649 >> 0 //=> -2147483647
```

- [21.6](#) Booleans:

```
const age = 0;

// Không tốt
const hasAge = new Boolean(age);

// Tốt
const hasAge = Boolean(age);
```

```
// Tốt
const hasAge = !!age;
```

[↑ Trở lại đầu trang](#)

## Naming Conventions

- 22.1 Nên mô tả đầy đủ một tên hàm hay biến.

```
// Không tốt
function q() {
 // ...stuff...
}

// Tốt
function query() {
 // ..stuff..
}
```

- 22.2 Dùng cách đặt tên `camelCase` cho đối tượng, biến, hàm, kế thừa ... .

```
// Không tốt
const OBJEcttsssss = {};
const this_is_my_object = {};
function c() {}

// Tốt
const thisIsMyObject = {};
function thisIsMyFunction() {}
```

- 22.3 Dùng kiểu `PascalCase` để đặt tên cho `Class` hoặc `Constructor` .

```
// Không tốt
function user(options) {
 this.name = options.name;
}

const bad = new user({
 name: 'nope',
});

// Tốt
class User {
 constructor(options) {
 this.name = options.name;
 }
}

const good = new User({
 name: 'yup',
});
```

- 22.4 Dùng `_` ở đầu tên biến khi nó là loại `private` .

```
// Không tốt
this.__firstName__ = 'Panda';
this.firstName_ = 'Panda';

// Tốt
this._firstName = 'Panda';
```

- 22.5 Sử dụng Arrow function (`=>`) hoặc `Function#bind`.

```
// Không tốt
function foo() {
 const self = this;
 return function() {
 console.log(self);
 };
}

// Không tốt
function foo() {
 const that = this;
 return function() {
 console.log(that);
 };
}

// Tốt
function foo() {
 return () => {
 console.log(this);
 };
}
```

- 22.6 Khi `export` một `class` duy nhất, thì tên `file` nên trùng với tên `class`.

```
// file contents
class CheckBox {
 // ...
}
export default CheckBox;

// in some other file
// Không tốt
import CheckBox from './checkBox';

// Không tốt
import CheckBox from './check_box';

// Tốt
import CheckBox from './CheckBox';
```

- 22.7 Dùng kiểu `camelCase` khi `export` mặc định hàm. Tên `file` và tên hàm nên tương tự nhau.

```
function makeStyleGuide() {
}

export default makeStyleGuide;
```

- 22.8 Dùng kiểu `PascalCase` khi `export` một `singleton` / `function library` / `bare object`.

```
const AirbnbStyleGuide = {
 es6: {
 }
};

export default AirbnbStyleGuide;
```

[↑ Trở lại đầu trang](#)

## Accessors

- 23.1 Accessor functions cho các thuộc tính không cần thiết.
- 23.2 Khi tạo một accessor functions sử dụng cấu trúc `getVal()` và `setVal('hello')`.

```
// Không tốt
dragon.age();

// Tốt
dragon.getAge();

// Không tốt
dragon.age(25);

// Tốt
dragon.setAge(25);
```

- 23.3 Nếu thuộc tính là `boolean`, sử dụng `isVal()` và `hasVal()`.

```
// Không tốt
if (!dragon.age()) {
 return false;
}

// Tốt
if (!dragon.hasAge()) {
 return false;
}
```

- 23.4 Có thể ghi đè 2 hàm mặc định `get()` và `set()`, nhưng phải có tính nhất quán.

```
class Jedi {
 constructor(options = {}) {
 const lightsaber = options.lightsaber || 'blue';
 this.set('lightsaber', lightsaber);
 }

 set(key, val) {
 this[key] = val;
 }

 get(key) {
 return this[key];
 }
}
```

[↑ Trở lại đầu trang](#)

## Events

- 24.1 Dùng `hash value` thay vì `raw value` khi truyền các tham số vào `events`.

```
// Không tốt
$(this).trigger('listingUpdated', listing.id);

...

$(this).on('listingUpdated', function(e, listingId) {
 // do something with listingId
});
```

prefer:

```
// Tốt
$(this).trigger('listingUpdated', { listingId: listing.id });

...

$(this).on('listingUpdated', function(e, data) {
 // do something with data.listingId
});
```

[↑ Trở lại đầu trang](#)

## jQuery

- [25.1](#) Thêm tiền tố `$` khi biến đó được tạo ra từ `jQuery`.

```
// Không tốt
const sidebar = $('.sidebar');

// Tốt
const $sidebar = $('.sidebar');

// Tốt
const $sidebarBtn = $('.sidebar-btn');
```

- [25.2](#) Cache jQuery.

```
// Không tốt
function setSidebar() {
 $('.sidebar').hide();

 // ...stuff...

 $('.sidebar').css({
 'background-color': 'pink'
 });
}

// Tốt
function setSidebar() {
 const $sidebar = $('.sidebar');
 $sidebar.hide();

 // ...stuff...

 $sidebar.css({
 'background-color': 'pink'
 });
}
```

- [25.3](#) Sử dụng kiểu Cascading `$('.sidebar ul')` hoặc là kiểu parent > child `$('.sidebar > ul')`. [jsPerf](#)
- [25.4](#) Sử dụng `find`.

```
// Không tốt
$('ul', '.sidebar').hide();

// Không tốt
$('.sidebar').find('ul').hide();

// Tốt
$('.sidebar ul').hide();

// Tốt
```

```
$('.sidebar > ul').hide();

// Tốt
$sidebar.find('ul').hide();
```

[↑ Trở lại đầu trang](#)

## ECMAScript 5 Compatibility

---

- [26.1](#) Refer to [Kangax's ES5 compatibility table](#).

[↑ Trở lại đầu trang](#)

## ECMAScript 6 Styles

---

- [27.1](#) `ES6` Features.

1. [Arrow Functions](#)
2. [Classes](#)
3. [Object Shorthand](#)
4. [Object Concise](#)
5. [Object Computed Properties](#)
6. [Template Strings](#)
7. [Destructuring](#)
8. [Default Parameters](#)
9. [Rest](#)
10. [Array Spreads](#)
11. [Let and Const](#)
12. [Iterators and Generators](#)
13. [Modules](#)

[↑ Trở lại đầu trang](#)

## Testing

---

- [28.1](#) `Yup`.

```
function () {
 return true;
}
```

- [28.2](#) No, but seriously:
- Luôn luôn viết `test` !
- Chia nhỏ hàm.
- Cẩn thận khi dùng `stubs` và `mocks` .
- `mocha` và `tape` được sử dụng ở `Airbnb` .
- Bảo đảm các `test` đều chạy tốt (100% coverage).
- Khi sửa chữa một lỗi nào đó nên viết lại `test` .

[↑ Trở lại đầu trang](#)

## Performance

---

- [On Layout & Web Performance](#)
- [String vs Array Concat](#)
- [Try/Catch Cost In a Loop](#)
- [Bang Function](#)
- [jQuery Find vs Context, Selector](#)
- [innerHTML vs textContent for script text](#)
- [Long String Concatenation](#)
- [Loading...](#)

[↑ Trở lại đầu trang](#)

## Resources

---

### Learning ES6

- [Draft ECMA 2015 \(ES6\) Spec](#)
- [ExploringJS](#)
- [ES6 Compatibility Table](#)
- [Comprehensive Overview of ES6 Features](#)

### Nên đọc

- [Standard ECMA-262](#)

### Tools

- Code Style Linters
  - [ESLint](#) - Airbnb Style .eslintrc
  - [JSHint](#) - Airbnb Style .jshintrc
  - [JSCS](#) - Airbnb Style Preset

### Các `styles` Guide khác

- [Google JavaScript Style Guide](#)
- [jQuery Core Style Guidelines](#)
- [Principles of Writing Consistent, Idiomatic JavaScript](#)

### `styles` Khác

- [Naming this in nested functions](#) - Christian Johansen
- [Conditional Callbacks](#) - Ross Allen
- [Popular JavaScript Coding Conventions on Github](#) - JeongHoon Byun
- [Multiple var statements in JavaScript, not superfluous](#) - Ben Alman

### Further Reading

- [Understanding JavaScript Closures](#) - Angus Croll
- [Basic JavaScript for the impatient programmer](#) - Dr. Axel Rauschmayer
- [You Might Not Need jQuery](#) - Zack Bloom & Adam Schwartz
- [ES6 Features](#) - Luke Hoban
- [Frontend Guidelines](#) - Benjamin De Cock

### Sách

- [JavaScript: The Good Parts](#) - Douglas Crockford
- [JavaScript Patterns](#) - Stoyan Stefanov
- [Pro JavaScript Design Patterns](#) - Ross Harmes and Dustin Diaz
- [High Performance Web Sites: Essential Knowledge for Front-End Engineers](#) - Steve Souders
- [Maintainable JavaScript](#) - Nicholas C. Zakas
- [JavaScript Web Applications](#) - Alex MacCaw
- [Pro JavaScript Techniques](#) - John Resig
- [Smashing Node.js: JavaScript Everywhere](#) - Guillermo Rauch
- [Secrets of the JavaScript Ninja](#) - John Resig and Bear Bibeault
- [Human JavaScript](#) - Henrik Joreteg
- [Superhero.js](#) - Kim Joar Bekkelund, Mads Mobæk, & Olav Bjorkoy
- [JSBooks](#) - Julien Bouquillon
- [Third Party JavaScript](#) - Ben Vinegar and Anton Kovalyov
- [Effective JavaScript: 68 Specific Ways to Harness the Power of JavaScript](#) - David Herman
- [Eloquent JavaScript](#) - Marijn Haverbeke
- [You Don't Know JS: ES6 & Beyond](#) - Kyle Simpson

## Blogs

- [DailyJS](#)
- [JavaScript Weekly](#)
- [JavaScript, JavaScript...](#)
- [Bocoup Weblog](#)
- [Adequately Good](#)
- [NCZOnline](#)
- [Perfection Kills](#)
- [Ben Alman](#)
- [Dmitry Baranovskiy](#)
- [Dustin Diaz](#)
- [nettuts](#)

## Podcasts

- [JavaScript Jabber](#)

[↑ Trở lại đầu trang](#)

## In the Wild

---

Danh sách các trang web sử dụng `style guide` của Airbnb.

- **Aan Zee:** [AanZee/javascript](#)
- **Adult Swim:** [adult-swim/javascript](#)
- **Airbnb:** [airbnb/javascript](#)
- **Apartment:** [apartment/javascript](#)
- **Avalara:** [avalara/javascript](#)
- **Billabong:** [billabong/javascript](#)
- **Blendle:** [blendle/javascript](#)
- **ComparaOnline:** [comparaonline/javascript](#)
- **Compass Learning:** [compasslearning/javascript-style-guide](#)
- **DailyMotion:** [dailymotion/javascript](#)
- **Digitpaint** [digitpaint/javascript](#)



- Ecosia: [ecosia/javascript](#)
- Evernote: [evernote/javascript-style-guide](#)
- ExactTarget: [ExactTarget/javascript](#)
- Expensify [Expensify/Style-Guide](#)
- Flexberry: [Flexberry/javascript-style-guide](#)
- Gawker Media: [gawkermedia/javascript](#)
- General Electric: [GeneralElectric/javascript](#)
- GoodData: [gooddata/gdc-js-style](#)
- Grooveshark: [grooveshark/javascript](#)
- How About We: [howaboutwe/javascript](#)
- Huballin: [huballin/javascript](#)
- HubSpot: [HubSpot/javascript](#)
- Hyper: [hyperoslo/javascript-playbook](#)
- InfoJobs: [InfoJobs/JavaScript-Style-Guide](#)
- Intent Media: [intentmedia/javascript](#)
- Jam3: [Jam3/Javascript-Code-Conventions](#)
- JSSolutions: [JSSolutions/javascript](#)
- Kinetica Solutions: [kinetica/javascript](#)
- Mighty Spring: [mightyspring/javascript](#)
- MinnPost: [MinnPost/javascript](#)
- MitocGroup: [MitocGroup/javascript](#)
- ModCloth: [modcloth/javascript](#)
- Money Advice Service: [moneyadvice.service/javascript](#)
- Muber: [muber/javascript](#)
- National Geographic: [natgeo/javascript](#)
- National Park Service: [nationalpark.service/javascript](#)
- Nimbl3: [nimbl3/javascript](#)
- Orion Health: [orionhealth/javascript](#)
- Peerby: [Peerby/javascript](#)
- Razorfish: [razorfish/javascript-style-guide](#)
- reddit: [reddit/styleguide/javascript](#)
- REI: [reidev/js-style-guide](#)
- Ripple: [ripple/javascript-style-guide](#)
- SeekingAlpha: [seekingalpha/javascript-style-guide](#)
- Shutterfly: [shutterfly/javascript](#)
- Springload: [springload/javascript](#)
- StudentSphere: [studentsphere/javascript](#)
- Target: [target/javascript](#)
- TheLadders: [TheLadders/javascript](#)
- T4R Technology: [T4R-Technology/javascript](#)
- VoxFeed: [VoxFeed/javascript-style-guide](#)
- Weggo: [Weggo/javascript](#)
- Zillow: [zillow/javascript](#)
- ZocDoc: [ZocDoc/javascript](#)

[↑ Trở lại đầu trang](#)

## Translation

---

Style Guide đã được chuyển đổi sang một số ngôn ngữ khác:

-  Brazilian Portuguese: [armoucar/javascript-style-guide](#)
-  Bulgarian: [borislavvv/javascript](#)
-  Catalan: [fpmweb/javascript-style-guide](#)
-  Chinese (Simplified): [sivan/javascript-style-guide](#)
-  Chinese (Traditional): [jigsawye/javascript](#)
-  French: [nmussy/javascript-style-guide](#)
-  German: [timofurrer/javascript-style-guide](#)
-  Italian: [sinkswim/javascript-style-guide](#)
-  Japanese: [mitsuruog/javascript-style-guide](#)
-  Korean: [tipjs/javascript-style-guide](#)
-  Polish: [mjurczyk/javascript](#)
-  Russian: [uproch/javascript](#)
-  Spanish: [paolocarrasco/javascript-style-guide](#)
-  Thai: [lvarayut/javascript-style-guide](#)

## The JavaScript Style Guide Guide

---

- [Reference](#)

## Chat With Us About JavaScript

---

- Find us on [gitter](#).

## Contributors

---

- [View Contributors](#)

## License

---

(The MIT License)

Copyright (c) 2014 Airbnb

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

[↑ Trở lại đầu trang](#)

## Amendments

---

We encourage you to fork this guide and change the rules to fit your team's style guide. Below, you may list some amendments to the style guide. This allows you to periodically update your style guide without having to deal with merge conflicts.

`};`

---

