**ES6 Javascript**

1. Biến: Ta sử dụng let hoặc const thay vì dùng var. let và const đều là phạm vi khối, tuy nhiên let là kiểu có thể thay đổi được, còn const thì không.

Vd: let a = 5;

const b = 4;

1. Đối tượng:

* sử dụng cú pháp chữ thông thương để khai báo Oject không dùng new.

Vd: let a = {};

* Sử dụng cú pháp tắt :

Vd: // bad

const atom = {

value: 1,

addValue: function (value) {

return atom.value + value;

},

};

// good

const atom = {

value: 1,

addValue(value) {

return atom.value + value;

},

};

* Sử dụng kí hiệu thuộc tính:

Vd:

const lukeSkywalker = ' Luke Skywalker ' ;

// bad const obj = {

lukeSkywalker : lukeSkywalker,

};

// tốt const obj = {

lukeSkywalker,

};

* Nhóm các thuộc tính viết tắt vào đầu khai báo đối tượng:

Vd:

const anakinSkywalker = ' Anakin Skywalker ' ;

const lukeSkywalker = ' Luke Skywalker ' ; // bad const obj = {

episodeOne : 1 ,

twoJediWalkIntoACantina : 2 ,

lukeSkywalker,

episodeThree : 3 ,

mayTheFourth : 4 ,

anakinSkywalker,

}; // tốt const obj = {

lukeSkywalker,

anakinSkywalker,

episodeOne : 1 ,

twoJediWalkIntoACantina : 2 ,

episodeThree : 3 ,

mayTheFourth : 4 ,

};

* Không cần khai báo thuộc tính của đối tượng trong ngoặc:

Vd:

// bad

const bad = {

' foo ' : 3 ,

' bar ' : 4 ,

' data-blah ' : 5 ,

}; // tốt const tốt = {

foo : 3 ,

bar : 4 ,

' data-blah ' : 5 ,

};

1. Mảng:

* Sử dụng .push để gán trực tiếp vào 1 mảng:

Vd:

const someStack = [];

// bad

someStack[someStack.length] = 'abracadabra';

// good

someStack.push('abracadabra');

* Sử dụng mảng lan truyền để sao chép mảng:

Vd:

// bad

const len = items.length;

const itemsCopy = [];

let i;

for (i = 0; i < len; i += 1) {

itemsCopy[i] = items[i];

}

// good

const itemsCopy = [...items];

1. String:

* Sử dung ‘’ cho chuoi
* Các chuỗi từ 100 kí tự trở lên k nên viết thành nhiều dòng
* Không bao giờ sử dụng eval() trên một chuỗi

1. Function

* Sử dụng các biểu thức hàm được đặt tên thay vì khai báo hàm
* Luôn để tham số mặc định ở cuối cùng:

Vd:

// bad

function handleThings ( opts = {}, name ) {

// ...

} // hàm handleThings tốt ( name , opts = {}) {

// ...

}

* Dùng => thay vì khai báo function

Vd:

// bad

[1, 2, 3].map(function (x) {

const y = x + 1;

return x \* y;

});

// good

[1, 2, 3].map((x) => {

const y = x + 1;

return x \* y;

});

* Nếu thân hàm bao gồm một câu lệnh đơn trả về một biểu thức không cso hiệu ứng phụ , bỏ qua dấu ngoặc và sử dung => . nếu không thfi hãy giữ {} và sử dụng return một câu lệnh

Vd:

// bad

[1, 2, 3].map(number => {

const nextNumber = number + 1;

`A string containing the ${nextNumber}.`;

});

// good

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

// good

[1, 2, 3].map((number) => {

const nextNumber = number + 1;

return `A string containing the ${nextNumber}.`;

});

// good

[1, 2, 3].map((number, index) => ({

[index]: number,

}));

// No implicit return with side effects

function foo(callback) {

const val = callback();

if (val === true) {

// Do something if callback returns true

}

}

let bool = false;

// bad

foo(() => bool = true);

// good

foo(() => {

bool = true;

});

* Tránh cú pháp khó hiểu (=>) toán tửu so sánh (<= , >=)

Vd:

// bad

const itemHeight = item => item.height > 256 ? item.largeSize : item.smallSize;

// bad

const itemHeight = (item) => item.height > 256 ? item.largeSize : item.smallSize;

// good

const itemHeight = item => (item.height > 256 ? item.largeSize : item.smallSize);

// good

const itemHeight = (item) => {

const { height, largeSize, smallSize } = item;

return height > 256 ? largeSize : smallSize;

};

1. Class and Contructors

* Luôn sử dụng class tránh nguyên mẫu trực tiếp
* Tránh các thành viên lớp trùng lặp:

Vd:

// bad

class Foo {

bar() { return 1; }

bar() { return 2; }

}

// good

class Foo {

bar() { return 1; }

}

// good

class Foo {

bar() { return 2; }

}

* Các lớp có một hàm khởi tạo mặc định nếu lướp không được chỉ định . không cần thiết với một hàm rỗng hoặc một hàm đại diện cho một lớp cha

Vd:

// bad

class Jedi {

constructor() {}

getName() {

return this.name;

}

}

// bad

class Rey extends Jedi {

constructor(...args) {

super(...args);

}

}

// good

class Rey extends Jedi {

constructor(...args) {

super(...args);

this.name = 'Rey';

}

}

1. Modules

* Luôn sử đụng cacs mô đun (import/ export) trên hệ thông module không chuẩn, bạn có thể ưu tiên chuyển đổi sang hệ thống module của mình

Vd:

// bad

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

module.exports = AirbnbStyleGuide.es6;

// ok

import AirbnbStyleGuide from './AirbnbStyleGuide';

export default AirbnbStyleGuide.es6;

// best

import { es6 } from './AirbnbStyleGuide';

export default es6;

* Không sử dụng nhập ki tự đại diện:

Vd:

// bad

import \* as AirbnbStyleGuide from './AirbnbStyleGuide';

// good

import AirbnbStyleGuide from './AirbnbStyleGuide';

* Không xuát trực tiếp từ một lần nhập:

Vd:

// bad

// filename es6.js

export { es6 as default } from './AirbnbStyleGuide';

// good

// filename es6.js

import { es6 } from './AirbnbStyleGuide';

export default es6;

* Chỉ nhập một đường dẫn ở một nơi:

Vd: / bad

import foo from 'foo';

// … some other imports … //

import { named1, named2 } from 'foo';

// good

import foo, { named1, named2 } from 'foo';

// good

import foo, {

named1,

named2,

} from 'foo';

1. Properties

* Sử dụng “ .” để truy cập thuộc tính
* Sử dụng ksi hiệu khung khi truy cập các giá trị của biến

Vd: const luke = {

jedi: true,

age: 28,

};

function getProp(prop) {

return luke[prop];

}

const isJedi = getProp('jedi');

* Sử dụng toán tử \*\* để tính lũy thừa

1. Các toán tử so sánh

* Sử dụng === và !== hơn == và !=
* Không được lồng ghép các ternaries vầ thường là các biểu thức dòng đơn

Vd: // bad

const foo = maybe1 > maybe2

? "bar"

: value1 > value2 ? "baz" : null;

// split into 2 separated ternary expressions

const maybeNull = value1 > value2 ? 'baz' : null;

// better

const foo = maybe1 > maybe2

? 'bar'

: maybeNull;

// best

const foo = maybe1 > maybe2 ? 'bar' : maybeNull;

* Tránh các cấu lệnh ternary không cần thiết:

Vd: // bad

const foo = a ? a : b;

const bar = c ? true : false;

const baz = c ? false : true;

// good

const foo = a || b;

const bar = !!c;

const baz = !c;

* Khi trộn toán tử hãy đặt chúng vào dấu ngoặc đơn. Ngoại lệ là các toán tử đại số chuẩn ( +, - , \*, /) vì ưu tiên của chúng được hiểu một cách rộng rãi

Vd: // bad

const foo = a && b < 0 || c > 0 || d + 1 === 0;

// bad

const bar = a \*\* b - 5 % d;

// bad

// one may be confused into thinking (a || b) && c

if (a || b && c) {

return d;

}

// good

const foo = (a && b < 0) || c > 0 || (d + 1 === 0);

// good

const bar = (a \*\* b) - (5 % d);

// good

if (a || (b && c)) {

return d;

}

// good

const bar = a + b / c \* d;

1. Block

* Sử dụng {} với tất cả các khối nhiều dòng
* Nếu một khối if luôn return một câu lệnh , else khối tiếp theo là không cần thiết, . một return trong một else if khối sau một if khối có chứa một return có thể tách rời thành nhiều if khối

Vd:

// bad

function foo() {

if (x) {

return x;

} else {

return y;

}

}

// bad

function cats() {

if (x) {

return x;

} else if (y) {

return y;

}

}

// bad

function dogs() {

if (x) {

return x;

} else {

if (y) {

return y;

}

}

}

// good

function foo() {

if (x) {

return x;

}

return y;

}

// good

function cats() {

if (x) {

return x;

}

if (y) {

return y;

}

}

// good

function dogs(x) {

if (x) {

if (z) {

return y;

}

} else {

return z;

}

}

1. Comment

* Sử dụng /\*\* \*/ để comment nhiều dòng
* Sư dụng // cho các chú thích một dòng
* Sử dụng // FIXME: để chsu thcihs các vấn đề:

Vd: class Calculator extends Abacus {

constructor() {

super();

// TODO: total should be configurable by an options param

this.total = 0;

}

}

1. Khoảng trắng

* Sử dụng khoảng trắng để ngăn cách các toán tử, ngăn cách tên hàm, hay if, else, while, for, …
* Sử dụng thụt đầu dòng khi tạo các chuỗi phương pháp dài
* Để lại một dòng trống trước câu lệnh hoặc một khối tiếp theo
* Khi coppy không để lại dòng trống

Vd: // bad

function bar() {

console.log(foo);

}

// bad

if (baz) {

console.log(qux);

} else {

console.log(foo);

}

// bad

class Foo {

constructor(bar) {

this.bar = bar;

}

}

// good

function bar() {

console.log(foo);

}

// good

if (baz) {

console.log(qux);

} else {

console.log(foo);

}

* Không thêm dấu cách trong ngoặc đơn [], ngoặc nhọn {}, tránh dấu cách trước dấu “ ,” thêm dấu cách đằng sau đấu phẩy

1. Quy ước đặt tên

* Tránh đặt tên biết là một kí tự duy nhất
* Sử dụng camelCase để đặt tên đối tượng, hàm, haowcj một cá thể,…
* Sử dựng PascalCase khi dặt tên các hàm hoặc các lớp
* Không sử dụng gạch duwosi hoặc dấu gạch dưới hang đầu
* Không lưu các tham chiếu đến this
* Các từ viết tắt phải luôn được viết hoa