JavaScript

Airbnb JavaScript () {

JavaScript A mostly reasonable approach to JavaScript

> *: Babel babel-preset-airbnb airbnb-browser-shims*
> *`shims/polyfills`*

Types

<a name="1.1"></a>
<a name="types--primitives"></a>

- 1.1 :
- * `string`
- * `number`
- * `boolean`
- * `null`
- * `undefined`
- * `symbol`

```
•
  * const foo = 1;
  * let bar = foo;
  * bar = 9;
  * console.log(foo, bar); // => 1, 9
  *
```

- * Symbols polyfill symbol symbol
- <a name="1.2"></a>
- <a name="types--complex"></a>
- 1.2 :
- * `object`
- * `array`
- * `function`

```
•
  * const foo = [1, 2];
  * const bar = foo;
  * bar[0] = 9;
  * console.log(foo[0], bar[0]); // => 9, 9
  *
```

back to top

References

<a name="2.1"></a>
<a name="references--prefer-const"></a>

- 2.1 {{const}}{{var}}. eslint: `prefer-const`, `no-const-assign`

> - *Why? bug*

```
  * // bad
  * var a = 1;
  * var b = 2;
  * // good
  * const a = 1;
  * const b = 2;
  *
```

- <a name="2.2"></a>
- <a name="references--disallow-var"></a>
- 2.2 {{let}}{{var}}. eslint: `no-var`

> - *Why? {{let}}{{var}}*

```
* // bad
* var count = 1;
* if (true) {
*   count += 1;
* }
* // good, use the let.
* let count = 1;
* if (true) {
*   count += 1;
* }
*
```

- <a name="2.3"></a>
- <a name="references--block-scope"></a>
- 2.3 `let`{{const}}

```
•
  * // const  let
  * {
  *   let a = 1;
  *   const b = 1;
  * }
  * console.log(a); // ReferenceError
  * console.log(b); // ReferenceError
  *
```

back to top

Objects

<a name="3.1"></a>
<a name="objects--no-new"></a>

- 3.1 . eslint: `no-new-object`

```
•
  * // bad
  * const item = new Object();
  * // good
  * const item = {};
  *
```

- <a name="3.2"></a>
- <a name="es6-computed-properties"></a>
- 3.2

  > - *Why? .*

```javascript
- function getKey(k) {
- return {[a key named $
  Unknown macro: {k}
  }},
- }
- // bad
- const obj =
  Unknown macro: {* id}
  ,
```

- objgetKey('enabled') = true;
- // good getKey('enabled')
- const obj = {
- id: 5,
- name: 'San Francisco',

: 'lang'

```
  [getKey('enabled')]: true,
};
```

<a name="3.3"></a>
<a name="es6-object-shorthand"></a>

- 3.3 . eslint: `object-shorthand`

```
•
  * // bad
  * const atom = {
  *   value: 1,
  *   addValue: function (value) {
  *     return atom.value + value;
  *   },
  * };
  * // good
  * const atom = {
  *   value: 1,
  *   //
  *   addValue(value) {
  *     return atom.value + value;
  *   },
  * };
  *
```

- <a name="3.4"></a>
- <a name="es6-object-concise"></a>
- 3.4 . eslint: `object-shorthand`

> - *Why?*

```
  * const lukeSkywalker = 'Luke Skywalker';
  * // bad
  * const obj = {
  *   lukeSkywalker: lukeSkywalker,
  * };
  * // good
  * const obj = {
  *   lukeSkywalker,
  * };
  *
```

- <a name="3.5"></a>
- <a name="objects--grouped-shorthand"></a>
- 3.5 .

> - *Why?* .

```
* const anakinSkywalker = 'Anakin Skywalker';
* const lukeSkywalker = 'Luke Skywalker';
* // bad
* const obj = {
*   episodeOne: 1,
*   twoJediWalkIntoACantina: 2,
*   lukeSkywalker,
*   episodeThree: 3,
*   mayTheFourth: 4,
*   anakinSkywalker,
* };
* // good
* const obj = {
*   lukeSkywalker,
*   anakinSkywalker,
*   episodeOne: 1,
*   twoJediWalkIntoACantina: 2,
*   episodeThree: 3,
*   mayTheFourth: 4,
* };
*
```

- <a name="3.6"></a>
- <a name="objects--quoted-props"></a>
- 3.6 `''`. eslint: `quote-props`

> - *Why? JS*

```
* // bad
* const bad = {
*   'foo': 3,
*   'bar': 4,
*   'data-blah': 5,
* };
* // good
* const good = {
*   foo: 3,
*   bar: 4,
*   'data-blah': 5,
* };
*
```

- <a name="3.7"></a>
- <a name="objects--prototype-builtins"></a>
- 3.7 {{Object.prototype}}{{hasOwnProperty}}, `propertyIsEnumerable`, `isPrototypeOf`

> - *Why? - {{*
>
>   *Unknown macro: { hasOwnProperty}*
>
>   *}} - {{Object.create(null)}}*

```
* // bad
* console.log(object.hasOwnProperty(key));
* // good
* console.log(Object.prototype.hasOwnProperty.call(object, key));
* // best
* const has = Object.prototype.hasOwnProperty; //
* /* or */
* import has from 'has'; // https://www.npmjs.com/package/has
* // ...
* console.log(has.call(object, key));
*
```

- <a name="3.8"></a>
- <a name="objects--rest-spread"></a>
- 3.8 {{...}}`Object.assign`rest{{...}}
- * ·

```
•
  * // very bad
  * const original = { a: 1, b: 2 };
  * const copy = Object.assign(original, { c: 3 }); // this mutates
  `original` _
  * delete copy.a; // so does this
  * // bad
  * const original = { a: 1, b: 2 };
  * const copy = Object.assign({}, original, { c: 3 }); // copy => {
  a: 1, b: 2, c: 3 }
  * // good es6 ...
  * const original = { a: 1, b: 2 };
  * //
  * const copy = { ...original, c: 3 }; // copy => { a: 1, b: 2, c:
  3 }
  * // rest
  * const { a, ...noA } = copy; // noA => { b: 2, c: 3 }
  *
```

back to top

Arrays

<a name="4.1"></a>
<a name="arrays--literals"></a>

- 4.1 eslint: `no-array-constructor`

```
•
  * // bad
  * const items = new Array();
  * // good
  * const items = [];
  *
```

- <a name="4.2"></a>
- <a name="arrays--push"></a>
- 4.2 Array#push

-
```
  * const someStack = [];
  * // bad
  * someStack[someStack.length] = 'abracadabra';
  * // good
  * someStack.push('abracadabra');
  *
```

- <a name="4.3"></a>
- <a name="es6-array-spreads"></a>
- 4.3
-
```
  * // bad
  * const len = items.length;
  * const itemsCopy = [];
  * let i;
  * for (i = 0; i < len; i += 1) {
  *   itemsCopy[i] = items[i];
  * }
  * // good
  * const itemsCopy = [...items];
  *
```

- <a name="4.4"></a>
- <a name="arrays--from-iterable"></a>
- 4.4 `...` `Array.from`
-
```
  * const foo = document.querySelectorAll('.foo');
  * // good
  * const nodes = Array.from(foo);
  * // best
  * const nodes = [...foo];
  *
```

- <a name="4.5"></a>
- <a name="arrays--from-array-like"></a>
- 4.5 `Array.from`
-
```
  * const arrLike = { 0: 'foo', 1: 'bar', 2: 'baz', length: 3 };
  * // bad
  * const arr = Array.prototype.slice.call(arrLike);
  * // good
  * const arr = Array.from(arrLike);
  *
```

- <a name="4.6"></a>
- <a name="arrays--mapping"></a>
- 4.6 `Array.from` `...` map

•

```
* // bad
* const baz = [...foo].map(bar);
* // good
* const baz = Array.from(foo, bar);
*
```

- <a name="4.7"></a>
- <a name="arrays--callback-return"></a>
- 4.7 return   return 8.2. eslint: `array-callback-return`

•

```
* // good
* [1, 2, 3].map((x) => {
*   const y = x + 1;
*   return x * y;
* });
* // good
* [1, 2, 3].map(x => x + 1);
* // bad -  acc undefined
* [[0, 1], [2, 3], [4, 5]].reduce((acc, item, index) => {
*   const flatten = acc.concat(item);
*   acc[index] = flatten;
* });
* // good
* [[0, 1], [2, 3], [4, 5]].reduce((acc, item, index) => {
*   const flatten = acc.concat(item);
*   acc[index] = flatten;
*   return flatten;
* });
* // bad
* inbox.filter((msg) => {
*   const { subject, author } = msg;
*   if (subject === 'Mockingbird') {
*     return author === 'Harper Lee';
*   } else {
*     return false;
*   }
* });
* // good
* inbox.filter((msg) => {
*   const { subject, author } = msg;
*   if (subject === 'Mockingbird') {
*     return author === 'Harper Lee';
*   }
*   return false;
* });
*
```

- <a name="4.8"></a>
- <a name="arrays--bracket-newline"></a>
- 4.8 `}}` `{{`

-
```
  * // bad
  * const arr = [
  *   [0, 1], [2, 3], [4, 5],
  * ];
  * const objectInArray = [{
  *   id: 1,
  * }, {
  *   id: 2,
  * }];
  * const numberInArray = [
  *   1, 2,
  * ];
  * // good
  * const arr = [[0, 1], [2, 3], [4, 5]];
  * const objectInArray = [
  *   {
  *     id: 1,
  *   },
  *   {
  *     id: 2,
  *   },
  * ];
  * const numberInArray = [
  *   1,
  *   2,
  * ];
  *
```

back to top

Destructuring

<a name="5.1"></a>
<a name="destructuring--object"></a>

- 5.1 eslint: `prefer-destructuring`

  > - *Why? /*

```
* // bad
* function getFullName(user) {
*   const firstName = user.firstName;
*   const lastName = user.lastName;
*   return `${firstName} ${lastName}`;
* }
* // good
* function getFullName(user) {
*   const { firstName, lastName } = user;
*   return `${firstName} ${lastName}`;
* }
* // best
* function getFullName({ firstName, lastName }) {
*   return `${firstName} ${lastName}`;
* }
*
```

- <a name="5.2"></a>
- <a name="destructuring--array"></a>
- 5.2 .

•

```
* const arr = [1, 2, 3, 4];
* // bad
* const first = arr[0];
* const second = arr[1];
* // good
* const [first, second] = arr;
*
```

- <a name="5.3"></a>
- <a name="destructuring--object-over-array"></a>
- 5.3

> - *Why?*

```
* // bad
* function processInput(input) {
*   //
*   return [left, right, top, bottom];
* }
* //
* const [left, __, top] = processInput(input);
* // good
* function processInput(input) {
*   // oops
*   return { left, right, top, bottom };
* }
* //
* const { left, top } = processInput(input);
*
```

back to top

Strings

<a name="6.1"></a>
<a name="strings--quotes"></a>

- 6.1 string `''` eslint: `quotes`

  ```
  •
    * // bad
    * const name = "Capt. Janeway";
    * // bad -
    * const name = `Capt. Janeway`;
    * // good
    * const name = 'Capt. Janeway';
    *
  ```

- <a name="6.2"></a>
- <a name="strings--line-length"></a>
- 6.2 100string

  > - *Why?*

  ```
    * // bad
    * 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.';
    * // bad
    * 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.';
    * // good
    * 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.';
    *
  ```

- <a name="6.3"></a>
- <a name="es6-template-literals"></a>
- 6.3 eslint: `prefer-template` `template-curly-spacing`

  > - *Why?*

```
* // bad
* function sayHi(name) {
*   return 'How are you, ' + name + '?';
* }
* // bad
* function sayHi(name) {
*   return ['How are you, ', name, '?'].join();
* }
* // bad
* function sayHi(name) {
*   return `How are you, ${ name }?`;
* }
* // good
* function sayHi(name) {
*   return `How are you, ${name}?`;
* }
*
```

- <a name="6.4"></a>
- <a name="strings--eval"></a>
- 6.4 {{eval()}} eslint: `no-eval`
- <a name="6.5"></a>
- <a name="strings--escaping"></a>
- 6.5 eslint: `no-useless-escape`

> - *Why?*

```
* // bad
* const foo = '\'this\' \i\s \"quoted\"';
* // good
* const foo = '\'this\' is "quoted"';
* //best
* const foo = `my name is '${name}'`;
*
```

back to top

Functions

<a name="7.1"></a>
<a name="functions--declarations"></a>

- 7.1 eslint: `func-style`

> - *const func = function () {}*
>
>   - *function func() {}*
>   - *Why? babel (Discussion)*
>   - *Why?*
>   - *Why? Function declarations are hoisted, which means that it's easy - too easy - to reference the function before it is defined in the file. This harms readability and maintainability. If you find that a function's definition is large or complex enough that it is interfering with understanding the rest of the file, then perhaps it's time to extract it to its own module! Don't forget to explicitly name the expression, regardless of whether or not the name is inferred from the containing variable (which is often the case in modern browsers or when using compilers such as Babel). This eliminates any assumptions made about the Error's call stack. (Discussion)*

```
* // bad
* function foo() {
*   // ...
* }
* // bad
* const foo = function () {
*   // ...
* };
* // good
* // lexical name distinguished from the variable-referenced
invocation(s)
* //
* const short = function longUniqueMoreDescriptiveLexicalFoo() {
*   // ...
* };
*
```

- <a name="7.2"></a>
- <a name="functions--iife"></a>
- 7.2 eslint: `wrap-iife`

> - *Why? immediately invoked function expression = IIFE*
>
>   - *Why? -*
>   - *Why? IIFE*

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

- <a name="7.3"></a>
- <a name="functions--in-blocks"></a>
- 7.3 ifwhile{{no-loop-func}} eslint: `no-loop-func`
- <a name="7.4"></a>
- <a name="functions--note-on-blocks"></a>
- 7.4 **Note:** ECMA-262 `block`

-

```
* // bad
* if (currentUser) {
*   function test() {
*     console.log('Nope.');
*   }
* }
* // good
* let test;
* if (currentUser) {
*   test = () => {
*     console.log('Yup.');
*   };
* }
*
```

- <a name="7.5"></a>
- <a name="functions--arguments-shadow"></a>
- 7.5 {{arguments}} `arguments` `arguments`

  ```
  •
    * // bad
    * function foo(name, options, arguments) {
    *   // ...
    * }
    * // good
    * function foo(name, options, args) {
    *   // ...
    * }
    *
  ```

- <a name="7.6"></a>
- <a name="es6-rest"></a>
- 7.6 {{arguments}}rest{{...}} eslint: `prefer-rest-params`

  > - *Why? `...}}rest{{arguments`*

  ```
    * // bad
    * function concatenateAll() {
    *   const args = Array.prototype.slice.call(arguments);
    *   return args.join('');
    * }
    * // good
    * function concatenateAll(...args) {
    *   return args.join('');
    * }
    *
  ```

- <a name="7.7"></a>
- <a name="es6-default-parameters"></a>
- 7.7

-

```
  * // really bad
  * function handleThings(opts) {
  *   //  arguments
  *   //   opts  false,  {}
  *   //  bug
  *   opts = opts || {};
  *   // ...
  * }
  * // still bad
  * function handleThings(opts) {
  *   if (opts === void 0) {
  *     opts = {};
  *   }
  *   // ...
  * }
  * // good
  * function handleThings(opts = {}) {
  *   // ...
  * }
  *
```

- <a name="7.8"></a>
- <a name="functions--default-side-effects"></a>
- 7.8

> - *Why? a*

```
  * var b = 1;
  * // bad
  * function count(a = b++) {
  *   console.log(a);
  * }
  * count();  // 1
  * count();  // 2
  * count(3); // 3
  * count();  // 3
  *
```

- <a name="7.9"></a>
- <a name="functions--defaults-last"></a>
- 7.9

-

```
  * // bad
  * function handleThings(opts = {}, name) {
  *   // ...
  * }
  * // good
  * function handleThings(name, opts = {}) {
  *   // ...
  * }
  *
```

- <a name="7.10"></a>
- <a name="functions--constructor"></a>
- 7.10 eslint: `no-new-func`

> - *Why? eval()*

```
* // bad
* var add = new Function('a', 'b', 'return a + b');
* // still bad
* var subtract = Function('a', 'b', 'return a - b');
*
```

- <a name="7.11"></a>
- <a name="functions--signature-spacing"></a>
- 7.11 eslint: `space-before-function-paren` `space-before-blocks`

> - *Why? //*

```
* // bad
* const f = function(){};
* const g = function (){};
* const h = function() {};
* // good
* const x = function () {};
* const y = function a() {};
*
```

- <a name="7.12"></a>
- <a name="functions--mutate-params"></a>
- 7.12 . eslint: `no-param-reassign`

> - *Why?*

```
* // bad
* function f1(obj) {
*   obj.key = 1;
* };
* // good
* function f2(obj) {
*   const key = Object.prototype.hasOwnProperty.call(obj, 'key') ?
obj.key : 1;
* };
*
```

- <a name="7.13"></a>
- <a name="functions--reassign-params"></a>
- 7.13 eslint: `no-param-reassign`

> - *Why? `arguments`V8*

```
* // bad
* function f1(a) {
*   a = 1;
*   // ...
* }
* function f2(a) {
*   if (!a) { a = 1; }
*   // ...
* }
* // good
* function f3(a) {
*   const b = a || 1;
*   // ...
* }
* function f4(a = 1) {
*   // ...
* }
*
```

- <a name="7.14"></a>
- <a name="functions--spread-vs-apply"></a>
- 7.14 {{spread}}{{...}} eslint: `prefer-spread`

> - *Why? {{apply}}{{new}}*

```
* // bad
* const x = [1, 2, 3, 4, 5];
* console.log.apply(console, x);
* // good
* const x = [1, 2, 3, 4, 5];
* console.log(...x);
* // bad
* new (Function.prototype.bind.apply(Date, [null, 2016, 8, 5]));
* // good
* new Date(...[2016, 8, 5]);
*
```

- <a name="7.15"></a>
- <a name="functions--signature-invocation-indentation"></a>
- 7.15

-

```
* // bad
* function foo(bar,
*              baz,
*              quux) {
*   // ...
* }
* // good
* function foo(
*   bar,
*   baz,
*   quux,
* ) {
*   // ...
* }
* // bad
* console.log(foo,
*   bar,
*   baz);
* // good
* console.log(
*   foo,
*   bar,
*   baz,
* );
*
```

back to top

Arrow Functions

<a name="8.1"></a>
<a name="arrows--use-them"></a>

- 8.1 eslint: `prefer-arrow-callback`, `arrow-spacing`

> - *Why? {{this}}*
>
>   - *Why?*

```
* // 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;
* });
*
```

- <a name="8.2"></a>
- <a name="arrows--implicit-return"></a>
- 8.2 return `return` eslint: `arrow-parens`, `arrow-body-style`

> - *Why?*

{{``javascript
- // bad
- 1, 2, 3.map(number => {
- const nextNumber = number + 1;}}A string containing the $

Unknown macro: {nextNumber}

.`;

  - });
  - // good
  - 1, 2, 3.map(number => {{A string containing the $

Unknown macro: {number}

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

.}};
- });
- // good
- 1, 2, 3.map((number, index) => ({

: 'lang'

```
  [index]: number
}));

// return
function foo(callback) {
  const val = callback();
  if (val === true) {
    // Do something if callback returns true
  }
}

let bool = false;

// bad
// return bool = true,
foo(() => bool = true);

// good
foo(() => {
  bool = true;
});
```

<a name="8.3"></a>
<a name="arrows--paren-wrap"></a>

- 8.3

> - *Why?*

: 'lang'

```
* // bad
* ['get', 'post', 'put'].map(httpMethod => Object.prototype.hasOwnProperty.call(
*     httpMagicObjectWithAVeryLongName,
*     httpMethod
*   )
* );
* // good
* ['get', 'post', 'put'].map(httpMethod => (
*   Object.prototype.hasOwnProperty.call(
*     httpMagicObjectWithAVeryLongName,
*     httpMethod
*   )
* ));
*
```

- <a name="8.4"></a>
- <a name="arrows--one-arg-parens"></a>
- 8.4 "always" option for eslint. eslint: `arrow-parens`

> - *Why?*

: 'lang'

```
* // bad
* [1, 2, 3].map((x) => x * x);
* // good
* [1, 2, 3].map(x => x * x);
* // good
* [1, 2, 3].map(number => (
*   `A long string with the ${number}. It's so long that we don't want it to take up space on
the .map line!`
* ));
* // bad
* [1, 2, 3].map(x => {
*   const y = x + 1;
*   return x * y;
* });
* // good
* [1, 2, 3].map((x) => {
*   const y = x + 1;
*   return x * y;
* });
*
```

- <a name="8.5"></a>
- <a name="arrows--confusing"></a>
- 8.5 (=>)<=, >=. eslint: `no-confusing-arrow`
- : 'lang'

```
* // 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;
* };
*
```

- <a name="8.6"></a>
- <a name="whitespace--implicit-arrow-linebreak"></a>
- 8.6 return eslint: `implicit-arrow-linebreak`

  -
    ```
    * // bad
    * (foo) =>
    *   bar;
    * (foo) =>
    *   (bar);
    * // good
    * (foo) => bar;
    * (foo) => (bar);
    * (foo) => (
    *    bar
    * )
    *
    ```

back to top

Classes & Constructors

<a name="9.1"></a>
<a name="constructors--use-class"></a>

- 9.1 {{class}}{{prototype}}

  > - *Why? {{class}}*

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

: 'lang'

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

<a name="9.2"></a>
<a name="constructors--extends"></a>

- 9.2 {{extends}}

> - *Why? {{instanceof}}*

```
* // bad
* const inherits = require('inherits');
* function PeekableQueue(contents) {
*   Queue.apply(this, contents);
* }
* inherits(PeekableQueue, Queue);
* PeekableQueue.prototype.peek = function () {
*   return this._queue[0];
* }
* // good
* class PeekableQueue extends Queue {
*   peek() {
*     return this._queue[0];
*   }
* }
*
```

- <a name="9.3"></a>
- <a name="constructors--chaining"></a>
- 9.3 {{this}}

-

```
* // bad
* 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
* // good
* class Jedi {
*   jump() {
*     this.jumping = true;
*     return this;
*   }
*   setHeight(height) {
*     this.height = height;
*     return this;
*   }
* }
* const luke = new Jedi();
* luke.jump()
*   .setHeight(20);
*
```

<a name="9.4"></a>
<a name="constructors--tostring"></a>

- 9.4 toString()

-

```
* class Jedi {
*   constructor(options = {}) {
*     this.name = options.name || 'no name';
*   }
*   getName() {
*     return this.name;
*   }
*   toString() {
*     return `Jedi - ${this.getName()}`;
*   }
* }
*
```

- <a name="9.5"></a>
- <a name="constructors--no-useless"></a>
- 9.5 eslint: `no-useless-constructor`

-

```
* // 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';
*   }
* }
*
```

- <a name="9.6"></a>
- <a name="classes--no-duplicate-members"></a>
- 9.6 eslint: `no-dupe-class-members`

> - *Why? —— bug*

```
* // bad
* class Foo {
*   bar() { return 1; }
*   bar() { return 2; }
* }
* // good
* class Foo {
*   bar() { return 1; }
* }
* // good
* class Foo {
*   bar() { return 2; }
* }
*
```

back to top

Modules

<a name="10.1"></a>
<a name="modules--use-them"></a>

- 10.1 (`import`/`export`)

> - *Why?*

```
* // 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;
*
```

- <a name="10.2"></a>
- <a name="modules--no-wildcard"></a>
- 10.2 import *

> - *Why?*

```
* // bad
* import * as AirbnbStyleGuide from './AirbnbStyleGuide';
* // good
* import AirbnbStyleGuide from './AirbnbStyleGuide';
*
```

- <a name="10.3"></a>
- <a name="modules--no-export-from-import"></a>
- 10.3 importexport

> - *Why?*

```
* // bad
* // filename es6.js
* export { es6 as default } from './AirbnbStyleGuide';
* // good
* // filename es6.js
* import { es6 } from './AirbnbStyleGuide';
* export default es6;
*
```

- <a name="10.4"></a>
- <a name="modules--no-duplicate-imports"></a>
- 10.4 import
- eslint: `no-duplicate-imports`

> - *Why? import*

```
* // 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';
*
```

- <a name="10.5"></a>
- <a name="modules--no-mutable-exports"></a>
- 10.5
- eslint: `import/no-mutable-exports`

> - *Why?*

```
* // bad
* let foo = 3;
* export { foo }
* // good
* const foo = 3;
* export { foo }
*
```

- <a name="10.6"></a>
- <a name="modules--prefer-default-export"></a>
- 10.6 `export default`
- eslint: `import/prefer-default-export`

> - *Why?*

```
* // bad
* export function foo() {}
* // good
* export default function foo() {}
*
```

- <a name="10.7"></a>
- <a name="modules--imports-first"></a>
- 10.7 `import`
- eslint: `import/first`

> - *Why? {{import}}*

```
* // bad
* import foo from 'foo';
* foo.init();
* import bar from 'bar';
* // good
* import foo from 'foo';
* import bar from 'bar';
* foo.init();
*
```

- <a name="10.8"></a>
- <a name="modules--multiline-imports-over-newlines"></a>
- 10.8 import

> - *Why?*

```
* // bad
* import {longNameA, longNameB, longNameC, longNameD, longNameE}
from 'path';
* // good
* import {
*   longNameA,
*   longNameB,
*   longNameC,
*   longNameD,
*   longNameE,
* } from 'path';
*
```

- <a name="10.9"></a>
- <a name="modules--no-webpack-loader-syntax"></a>
- 10.9 importWebpack loader
- eslint: `import/no-webpack-loader-syntax`

> - *Why? Webpackimport{{webpack.config.js}}webpack loader*

```
* // bad
* import fooSass from 'css!sass!foo.scss';
* import barCss from 'style!css!bar.css';
* // good
* import fooSass from 'foo.scss';
* import barCss from 'bar.css';
*
```

back to top

Iterators and Generators

<a name="11.1"></a>
<a name="iterators--nope"></a>

- 11.1 JavaScript{{for-in}} `for-of` eslint: `no-iterator` `no-restricted-syntax`

> - *Why?*
>
>   - *Why? `map()` / `every()` / `filter()` / `find()` / `findIndex()` / `reduce()` / `some()` / ... , `Object.keys()` / `Object.values()` / `Object.entries()`*

```
* const numbers = [1, 2, 3, 4, 5];
* // bad
* let sum = 0;
* for (let num of numbers) {
*   sum += num;
* }
* sum === 15;
* // good
* let sum = 0;
* numbers.forEach(num => sum += num);
* sum === 15;
* // best (use the functional force)
* const sum = numbers.reduce((total, num) => total + num, 0);
* sum === 15;
* // bad
* const increasedByOne = [];
* for (let i = 0; i < numbers.length; i++) {
*   increasedByOne.push(numbers[i] + 1);
* }
* // good
* const increasedByOne = [];
* numbers.forEach(num => increasedByOne.push(num + 1));
* // best (keeping it functional)
* const increasedByOne = numbers.map(num => num + 1);
*
```

- <a name="11.2"></a>
- <a name="generators--nope"></a>
- 11.2 generator

> - *Why? es5*

<a name="11.3"></a>
- <a name="generators--spacing"></a>
- 11.3 , eslint: `generator-star-spacing`

> - *Why? `function` {} - {}{{function}}{{function*}}{{function}}*

: 'lang'

```
* // bad
* function * foo() {
*   // ...
* }
* // bad
* const bar = function * () {
*   // ...
* }
* // bad
* const baz = function *() {
*   // ...
* }
* // bad
* const quux = function*() {
*   // ...
* }
* // bad
* function*foo() {
*   // ...
* }
* // bad
* function *foo() {
*   // ...
* }
* // very bad
* function
* *
* foo() {
*   // ...
* }
* // very bad
* const wat = function
* *
* () {
*   // ...
* }
* // good
* function* foo() {
*   // ...
* }
* // good
* const foo = function* () {
*   // ...
* }
*
```

back to top

Properties

<a name="12.1"></a>
<a name="properties--dot"></a>

- 12.1 . eslint: `dot-notation`

```
•
  * const luke = {
  *   jedi: true,
  *   age: 28,
  * };
  * // bad
  * const isJedi = luke['jedi'];
  * // good
  * const isJedi = luke.jedi;
  *
```

- <a name="12.2"></a>
- <a name="properties--bracket"></a>
- 12.2 {{[]}}

```
•
  * const luke = {
  *   jedi: true,
  *   age: 28,
  * };
  * function getProp(prop) {
  *   return luke[prop];
  * }
  * const isJedi = getProp('jedi');
  *
```

- <a name="12.3"></a>
- <a name="es2016-properties--exponentiation-operator"></a>
- 12.3 `**` eslint: `no-restricted-properties`.

```
•
  * // bad
  * const binary = Math.pow(2, 10);
  * // good
  * const binary = 2 ** 10;
  *
```

back to top

Variables

<a name="13.1"></a>
<a name="variables--const"></a>

- 13.1 {{const}} eslint: `no-undef` `prefer-const`

```
•
  * // bad
  * superPower = new SuperPower();
  * // good
  * const superPower = new SuperPower();
  *
```

- <a name="13.2"></a>
- <a name="variables--one-const"></a>
- 13.2 `const` `let` eslint: `one-var`

> - *Why? {{;}}{{,}}*

```
* // bad
* const items = getItems(),
*     goSportsTeam = true,
*     dragonball = 'z';
* // bad
* // (compare to above, and try to spot the mistake)
* const items = getItems(),
*     goSportsTeam = true;
*     dragonball = 'z';
* // good
* const items = getItems();
* const goSportsTeam = true;
* const dragonball = 'z';
*
```

- <a name="13.3"></a>
- <a name="variables--const-let-group"></a>
- 13.3 {{const}}{{let}}

> - *Why?*

```
* // bad
* let i, len, dragonball,
*     items = getItems(),
*     goSportsTeam = true;
* // bad
* let i;
* const items = getItems();
* let dragonball;
* const goSportsTeam = true;
* let len;
* // good
* const goSportsTeam = true;
* const items = getItems();
* let dragonball;
* let i;
* let length;
*
```

- <a name="13.4"></a>
- <a name="variables--define-where-used"></a>
- 13.4

> - *Why? `let` `const`*

```
* // bad - unnecessary function call
* function checkName(hasName) {
*   const name = getName();
*   if (hasName === 'test') {
*     return false;
*   }
*   if (name === 'test') {
*     this.setName('');
*     return false;
*   }
*   return name;
* }
* // good
* function checkName(hasName) {
*   if (hasName === 'test') {
*     return false;
*   }
*   //
*   const name = getName();
*   if (name === 'test') {
*     this.setName('');
*     return false;
*   }
*   return name;
* }
*
```

- <a name="13.5"></a>
- <a name="variables--no-chain-assignment"></a>
- 13.5 eslint: `no-multi-assign`

> - *Why?*

```
* // bad
* (function example() {
*   // JavaScript
*   // let a = ( b = ( c = 1 ) );
*   // let  a ;  b  c
*   let a = b = c = 1;
* }());
* console.log(a); // undefined
* console.log(b); // 1
* console.log(c); // 1
* // good
* (function example() {
*   let a = 1;
*   let b = a;
*   let c = a;
* }());
* console.log(a); // undefined
* console.log(b); // undefined
* console.log(c); // undefined
* // `const`
*
```

- <a name="13.6"></a>
- <a name="variables--unary-increment-decrement"></a>
- 13.6 ++ --. eslint `no-plusplus`

> - *Why? eslint {{num + = 1}}{{num ++}}{{num ++}} /*

```
*   // bad
*   let array = [1, 2, 3];
*   let num = 1;
*   num++;
*   --num;
*   let sum = 0;
*   let truthyCount = 0;
*   for(let i = 0; i < array.length; i++){
*     let value = array[i];
*     sum += value;
*     if (value) {
*       truthyCount++;
*     }
*   }
*   // good
*   let array = [1, 2, 3];
*   let num = 1;
*   num += 1;
*   num -= 1;
*   const sum = array.reduce((a, b) => a + b, 0);
*   const truthyCount = array.filter(Boolean).length;
*
```

- <a name="13.7"></a>

- <a name="variables--linebreak"></a>
- 13.7 = / `max-len` eslint `operator-linebreak`.

> - *Why? =*

```
* // bad
* const foo =
*   superLongLongLongLongLongLongLongLongFunctionName();
* // bad
* const foo
*   = 'superLongLongLongLongLongLongLongLongString';
* // good
* const foo = (
*   superLongLongLongLongLongLongLongLongFunctionName()
* );
* // good
* const foo = 'superLongLongLongLongLongLongLongLongString';
*
```

- <a name="13.8"></a>
- <a name="variables--no-unused-vars"></a>
- 13.8 eslint: `no-unused-vars`

> - *Why?*

```
* // bad
* var some_unused_var = 42;
* //
* var y = 10;
* y = 5;
* //
* var z = 0;
* z = z + 1;
* //
* function getX(x, y) {
*     return x;
* }
* // good
* function getXPlusY(x, y) {
*   return x + y;
* }
* var x = 1;
* var y = a + 2;
* alert(getXPlusY(x, y));
* // 'type'   rest
* //
* var { type, ...coords } = data;
* // 'coords'  'type'  'data'
*
```

back to top

Hoisting

<a name="14.1"></a>
<a name="hoisting--about"></a>

- 14.1 `var}}{{const` {{let}} —— Temporal Dead Zones (TDZ) typeof.

```
•
  * // notDefined
  * function example() {
  *   console.log(notDefined); // => throws a ReferenceError
  * }
  * //
  * //  declaredButNotAssigned
  * function example() {
  *   console.log(declaredButNotAssigned); // => undefined
  *   var declaredButNotAssigned = true;
  * }
  * //
  * //
  * function example() {
  *   let declaredButNotAssigned;
  *   console.log(declaredButNotAssigned); // => undefined
  *   declaredButNotAssigned = true;
  * }
  * //  const let
  * function example() {
  *   console.log(declaredButNotAssigned); // => throws a
  ReferenceError
  *   console.log(typeof declaredButNotAssigned); // => throws a
  ReferenceError
  *   const declaredButNotAssigned = true;
  * }
  *
```

- <a name="14.2"></a>
- <a name="hoisting--anon-expressions"></a>
- 14.2 `var`

```
•
  * function example() {
  *   console.log(anonymous); // => undefined
  *   anonymous(); // => TypeError anonymous is not a function
  *   var anonymous = function () {
  *     console.log('anonymous function expression');
  *   };
  * }
  *
```

- <a name="14.3"></a>
- <a name="hoisting--named-expresions"></a>
- 14.3

-

```
* 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');
*   };
* }
* //
* function example() {
*   console.log(named); // => undefined
*   named(); // => TypeError named is not a function
*   var named = function named() {
*     console.log('named');
*   };
* }
*
```

- <a name="14.4"></a>
- <a name="hoisting--declarations"></a>
- 14.4

-

```
* function example() {
*   superPower(); // => Flying
*   function superPower() {
*     console.log('Flying');
*   }
* }
*
```

- JavaScript Scoping & Hoisting by Ben Cherry.

back to top

Comparison Operators & Equality

<a name="15.1"></a>
<a name="comparison--eqeqeq"></a>

- 15.1 `=== Unable to render embedded object: File (==}}  {{==}}  {{) not found.=`. eslint: `eqeqeq`
- <a name="15.2"></a>
- <a name="comparison--if"></a>
- 15.2 'if'`ToBoolean'
- * **Objects true**
- * **Undefined false**
- * **Null false**
- * **Booleans the value of the boolean**
- * **Numbers*** **+0, -0, or NaN false**
- * * **true**
- * **Strings*** ' ' **false**
- * * **true**

-
  ```
  * if ([0] && []) {
  *   // true
  *   // true
  * }
  *
  ```

- <a name="15.3"></a>
- <a name="comparison--shortcuts"></a>
- 15.3

  -
  ```
  * // bad
  * if (isValid === true) {
  *   // ...
  * }
  * // good
  * if (isValid) {
  *   // ...
  * }
  * // bad
  * if (name) {
  *   // ...
  * }
  * // good
  * if (name !== '') {
  *   // ...
  * }
  * // bad
  * if (collection.length) {
  *   // ...
  * }
  * // good
  * if (collection.length > 0) {
  *   // ...
  * }
  *
  ```

- <a name="15.4"></a>
- <a name="comparison--moreinfo"></a>
- 15.4 Angus CrollJavaScript —— Truth Equality and JavaScript
- <a name="15.5"></a>
- <a name="comparison--switch-blocks"></a>
- 15.5 {{case}}{{default}}(e.g. `let`, `const`, `function`, and `class`). eslint rules: `no-case-declarations`.

  > - *Why? {{switch}}{{case}} {{case}}*

```
* // bad
* switch (foo) {
*   case 1:
*     let x = 1;
*     break;
*   case 2:
*     const y = 2;
*     break;
*   case 3:
*     function f() {
*       // ...
*     }
*     break;
*   default:
*     class C {}
* }
* // good
* switch (foo) {
*   case 1: {
*     let x = 1;
*     break;
*   }
*   case 2: {
*     const y = 2;
*     break;
*   }
*   case 3: {
*     function f() {
*       // ...
*     }
*     break;
*   }
*   case 4:
*     bar();
*     break;
*   default: {
*     class C {}
*   }
* }
*
```

- <a name="15.6"></a>
- <a name="comparison--nested-ternaries"></a>
- 15.6
- eslint rules: `no-nested-ternary`.

-
  ```
  * // bad
  * const foo = maybe1 > maybe2
  *   ? "bar"
  *   : value1 > value2 ? "baz" : null;
  * // better
  * const maybeNull = value1 > value2 ? 'baz' : null;
  * const foo = maybe1 > maybe2
  *   ? 'bar'
  *   : maybeNull;
  * // best
  * const maybeNull = value1 > value2 ? 'baz' : null;
  * const foo = maybe1 > maybe2 ? 'bar' : maybeNull;
  *
  ```

- <a name="15.7"></a>
- <a name="comparison--unneeded-ternary"></a>
- 15.7
- eslint rules: `no-unneeded-ternary`.
  -
    ```
    * // 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;
    *
    ```

- <a name="15.8"></a>
- <a name="comparison--no-mixed-operators"></a>
- 15.8 (+, -, *, & /) eslint: `no-mixed-operators`

  > - *Why?*

```
* // bad
* const foo = a && b < 0 || c > 0 || d + 1 === 0;
* // bad
* const bar = a ** b - 5 % d;
* // bad
* // (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;
*
```

back to top

Blocks

<a name="16.1"></a>
<a name="blocks--braces"></a>

- 16.1 eslint: `nonblock-statement-body-position`

```
•
  * // bad
  * if (test)
  *   return false;
  * // good
  * if (test) return false;
  * // good
  * if (test) {
  *   return false;
  * }
  * // bad
  * function foo() { return false; }
  * // good
  * function bar() {
  *   return false;
  * }
  *
```

- <a name="16.2"></a>
- <a name="blocks--cuddled-elses"></a>
- 16.2 {{if}}{{else}}{{if}} eslint: `brace-style`

-

```
* // bad
* if (test) {
*   thing1();
*   thing2();
* }
* else {
*   thing3();
* }
* // good
* if (test) {
*   thing1();
*   thing2();
* } else {
*   thing3();
* }
*
```

- <a name="16.3"></a>
- <a name="blocks--no-else-return"></a>
- 16.3 `if` `return` `else if` `return` `else` `if` `return` `return` `if` eslint: `no-else-return`

•

```
 * // 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;
 *   }
 * }
 *
```

back to top

Control Statements

<a name="17.1"></a>
<a name="control-statements"></a>

- 17.1 (`if`, `while`) ()

  > - *Why?*

```
* // bad
* if ((foo === 123 || bar === 'abc') &&
doesItLookGoodWhenItBecomesThatLong() && isThisReallyHappening()) {
*   thing1();
* }
* // bad
* if (foo === 123 &&
*   bar === 'abc') {
*   thing1();
* }
* // bad
* if (foo === 123
*   && bar === 'abc') {
*   thing1();
* }
* // bad
* if (
*   foo === 123 &&
*   bar === 'abc'
* ) {
*   thing1();
* }
* // good
* if (
*   foo === 123
*   && bar === 'abc'
* ) {
*   thing1();
* }
* // good
* if (
*   (foo === 123 || bar === 'abc')
*   && doesItLookGoodWhenItBecomesThatLong()
*   && isThisReallyHappening()
* ) {
*   thing1();
* }
* // good
* if (foo === 123 && bar === 'abc') {
*   thing1();
* }
*
```

- <a name="17.2"></a>
- <a name="control-statements--value-selection"></a>

- 17.2

  -
    ```
    * // bad
    * !isRunning && startRunning();
    * // good
    * if (!isRunning) {
    *   startRunning();
    * }
    *
    ```

back to top

Comments

<a name="18.1"></a>
<a name="comments--multiline"></a>

- 18.1 `/** ... */`

  -
    ```
    * // bad
    * // make() returns a new element
    * // based on the passed in tag name
    * //
    * // @param {String} tag
    * // @return {Element} element
    * function make(tag) {
    *   // ...
    *   return element;
    * }
    * // good
    * /**
    *  * make() returns a new element
    *  * based on the passed-in tag name
    *  */
    * function make(tag) {
    *   // ...
    *   return element;
    * }
    *
    ```

- <a name="18.2"></a>
- <a name="comments--singleline"></a>
- 18.2 {{//}}

-

```
* // bad
* const active = true;  // is current tab
* // good
* // is current tab
* const active = true;
* // bad
* function getType() {
*   console.log('fetching type...');
*   // set the default type to 'no type'
*   const type = this._type || 'no type';
*   return type;
* }
* // good
* function getType() {
*   console.log('fetching type...');
*   // set the default type to 'no type'
*   const type = this._type || 'no type';
*   return type;
* }
* // also good
* function getType() {
*   // set the default type to 'no type'
*   const type = this._type || 'no type';
*   return type;
* }
*
```

- <a name="18.3"></a>
- <a name="comments--spaces"></a>
- 18.3 eslint: `spaced-comment`

-

```
* // bad
* //is current tab
* const active = true;
* // good
* // is current tab
* const active = true;
* // bad
* /**
*   *make() returns a new element
*   *based on the passed-in tag name
*   */
* function make(tag) {
*   // ...
*   return element;
* }
* // good
* /**
*   * make() returns a new element
*   * based on the passed-in tag name
*   */
* function make(tag) {
*   // ...
*   return element;
* }
*
```

- <a name="18.4"></a>
- <a name="comments--actionitems"></a>
- 18.4 {{FIXME'}}TODO'   {{FIXME - }}{{TODO - }}
- <a name="18.5"></a>
- <a name="comments--fixme"></a>
- 18.5 {{// FIXME:}}

-

```
* class Calculator extends Abacus {
*   constructor() {
*     super();
*     // FIXME: shouldn't use a global here
*     total = 0;
*   }
* }
*
```

- <a name="18.6"></a>
- <a name="comments--todo"></a>
- 18.6 {{// TODO:}}

-
```
* class Calculator extends Abacus {
*   constructor() {
*     super();
*     // TODO: total should be configurable by an options param
*     this.total = 0;
*   }
* }
*
```

back to top

Whitespace

<a name="19.1"></a>
<a name="whitespace--spaces"></a>

- 19.1 tab. eslint: `indent`

-
```
* // bad
* function foo() {
* const name;
* }
* // bad
* function bar() {
* const name;
* }
* // good
* function baz() {
* const name;
* }
*
```

- <a name="19.2"></a>
- <a name="whitespace--before-blocks"></a>
- 19.2 eslint: `space-before-blocks`

-
  ```
  * // bad
  * function test(){
  *   console.log('test');
  * }
  * // good
  * function test() {
  *   console.log('test');
  * }
  * // bad
  * dog.set('attr',{
  *   age: '1 year',
  *   breed: 'Bernese Mountain Dog',
  * });
  * // good
  * dog.set('attr', {
  *   age: '1 year',
  *   breed: 'Bernese Mountain Dog',
  * });
  *
  ```

- <a name="19.3"></a>
- <a name="whitespace--around-keywords"></a>
- 19.3 (`if`, `while`) eslint: `keyword-spacing`

-
  ```
  * // bad
  * if(isJedi) {
  *   fight ();
  * }
  * // good
  * if (isJedi) {
  *   fight();
  * }
  * // bad
  * function fight () {
  *   console.log ('Swooosh!');
  * }
  * // good
  * function fight() {
  *   console.log('Swooosh!');
  * }
  *
  ```

- <a name="19.4"></a>
- <a name="whitespace--infix-ops"></a>
- 19.4 eslint: `space-infix-ops`

-
  ```
  * // bad
  * const x=y+5;
  * // good
  * const x = y + 5;
  *
  ```

- <a name="19.5"></a>
- <a name="whitespace--newline-at-end"></a>
- 19.5 . eslint: `eol-last`

```
•
  * // bad
  * import { es6 } from './AirbnbStyleGuide';
  *   // ...
  * export default es6;
  *
```

```
•
  * // bad
  * import { es6 } from './AirbnbStyleGuide';
  *   // ...
  * export default es6;
  *
  *
```

```
•
  * // good
  * import { es6 } from './AirbnbStyleGuide';
  *   // ...
  * export default es6;
  *
```

- <a name="19.6"></a>
- <a name="whitespace--chains"></a>
- 19.6 >2eslint: `newline-per-chained-call` `no-whitespace-before-property`

-

```
* // bad
* $('#items').find('.selected').highlight().end().find('.open').
updateCount();
* // bad
* $('#items').
*   find('.selected').
*     highlight().
*     end().
*   find('.open').
*     updateCount();
* // good
* $('#items')
*   .find('.selected')
*     .highlight()
*     .end()
*   .find('.open')
*     .updateCount();
* // bad
* const leds = stage.selectAll('.led').data(data).enter().append
('svg:svg').classed('led', true)
*     .attr('width', (radius + margin) * 2).append('svg:g')
*     .attr('transform', `translate(${radius + margin},${radius +
margin})`)
*     .call(tron.led);
* // good
* const leds = stage.selectAll('.led')
*     .data(data)
*   .enter().append('svg:svg')
*     .classed('led', true)
*     .attr('width', (radius + margin) * 2)
*   .append('svg:g')
*     .attr('transform', `translate(${radius + margin},${radius +
margin})`)
*     .call(tron.led);
* // good
* const leds = stage.selectAll('.led').data(data);
*
```

- <a name="19.7"></a>
- <a name="whitespace--after-blocks"></a>
- 19.7

-
```
* // bad
* if (foo) {
*   return bar;
* }
* return baz;
* // good
* if (foo) {
*   return bar;
* }
* return baz;
* // bad
* const obj = {
*   foo() {
*   },
*   bar() {
*   },
* };
* return obj;
* // good
* const obj = {
*   foo() {
*   },
*   bar() {
*   },
* };
* return obj;
* // bad
* const arr = [
*   function foo() {
*   },
*   function bar() {
*   },
* ];
* return arr;
* // good
* const arr = [
*   function foo() {
*   },
*   function bar() {
*   },
* ];
* return arr;
*
```

- <a name="19.8"></a>
- <a name="whitespace--padded-blocks"></a>
- 19.8 eslint: `padded-blocks`

-
  ```
  * // bad
  * function bar() {
  *   console.log(foo);
  * }
  * // also bad
  * if (baz) {
  *   console.log(qux);
  * } else {
  *   console.log(foo);
  * }
  * // good
  * function bar() {
  *   console.log(foo);
  * }
  * // good
  * if (baz) {
  *   console.log(qux);
  * } else {
  *   console.log(foo);
  * }
  *
  ```

- <a name="19.9"></a>
- <a name="whitespace--in-parens"></a>
- 19.9 eslint: `space-in-parens`

-
  ```
  * // bad
  * function bar( foo ) {
  *   return foo;
  * }
  * // good
  * function bar(foo) {
  *   return foo;
  * }
  * // bad
  * if ( foo ) {
  *   console.log(foo);
  * }
  * // good
  * if (foo) {
  *   console.log(foo);
  * }
  *
  ```

- <a name="19.10"></a>
- <a name="whitespace--in-brackets"></a>
- 19.10 eslint: `array-bracket-spacing`

-

```
* // bad
* const foo = [ 1, 2, 3 ];
* console.log(foo[ 0 ]);
* // good
* const foo = [1, 2, 3];
* console.log(foo[0]);
*
```

- <a name="19.11"></a>
- <a name="whitespace--in-braces"></a>
- 19.11 eslint: `object-curly-spacing`

-

```
* // bad
* const foo = {clark: 'kent'};
* // good
* const foo = { clark: 'kent' };
*
```

- <a name="19.12"></a>
- <a name="whitespace--max-len"></a>
- 19.12 100
- ——strings--line-length eslint: `max-len`

> - *Why?*

```
* // bad
* const foo = jsonData && jsonData.foo && jsonData.foo.bar &&
jsonData.foo.bar.baz && jsonData.foo.bar.baz.quux && jsonData.foo.
bar.baz.quux.xyzzy;
* // bad
* $.ajax({ method: 'POST', url: 'https://airbnb.com/', data: {
name: 'John' } }).done(() => console.log('Congratulations!')).fail
(() => console.log('You have failed this city.'));
* // good
* const foo = jsonData
*   && jsonData.foo
*   && jsonData.foo.bar
*   && jsonData.foo.bar.baz
*   && jsonData.foo.bar.baz.quux
*   && jsonData.foo.bar.baz.quux.xyzzy;
* // good
* $.ajax({
*   method: 'POST',
*   url: 'https://airbnb.com/',
*   data: { name: 'John' },
* })
*   .done(() => console.log('Congratulations!'))
*   .fail(() => console.log('You have failed this city.'));
*
```

- <a name="19.13"></a>
- <a name="whitespace--block-spacing"></a>

- 19.13 —— } { eslint: `block-spacing`

  ```
  •
    * // bad
    * function foo() {return true;}
    * if (foo) { bar = 0;}
    * // good
    * function foo() { return true; }
    * if (foo) { bar = 0; }
    *
  ```

- <a name="19.14"></a>
- <a name="whitespace--comma-spacing"></a>
- 19.14 `,` `,` eslint: `comma-spacing`

  ```
  •
    * // bad
    * var foo = 1,bar = 2;
    * var arr = [1 , 2];
    * // good
    * var foo = 1, bar = 2;
    * var arr = [1, 2];
    *
  ```

- <a name="19.15"></a>
- <a name="whitespace--computed-property-spacing"></a>
- 19.15 eslint: `computed-property-spacing`

  ```
  •
    * // bad
    * obj[foo ]
    * obj[ 'foo']
    * var x = {[ b ]: a}
    * obj[foo[ bar ]]
    * // good
    * obj[foo]
    * obj['foo']
    * var x = { [b]: a }
    * obj[foo[bar]]
    *
  ```

- <a name="19.16"></a>
- <a name="whitespace--func-call-spacing"></a>
- 19.16 eslint: `func-call-spacing`

  ```
  •
    * // bad
    * func ();
    * func
    * ();
    * // good
    * func();
    *
  ```

- <a name="19.17"></a>
- <a name="whitespace--key-spacing"></a>
- 19.17 `key` `value` eslint: `key-spacing`

-
  ```
  * // bad
  * var obj = { "foo" : 42 };
  * var obj2 = { "foo":42 };
  * // good
  * var obj = { "foo": 42 };
  *
  ```

- <a name="19.18"></a>
- <a name="whitespace--no-trailing-spaces"></a>
- 19.18 eslint: `no-trailing-spaces`
- <a name="19.19"></a>
- <a name="whitespace--no-multiple-empty-lines"></a>
- 19.19 eslint: `no-multiple-empty-lines`
- <!-- markdownlint-disable MD012 -->

-
  ```
  * // bad
  * var x = 1;
  *
  ```

: 'lang'

```
var y = 2;

// good
var x = 1;

var y = 2;
```
```

<!-- markdownlint-enable MD012 -->
```

back to top

Commas

<a name="20.1"></a>
<a name="commas--leading-trailing"></a>

- 20.1 eslint: `comma-style`

•

```
* // bad
* const story = [
*     once
*   , upon
*   , aTime
* ];
* // good
* const story = [
*   once,
*   upon,
*   aTime,
* ];
* // bad
* const hero = {
*     firstName: 'Ada'
*   , lastName: 'Lovelace'
*   , birthYear: 1815
*   , superPower: 'computers'
* };
* // good
* const hero = {
*   firstName: 'Ada',
*   lastName: 'Lovelace',
*   birthYear: 1815,
*   superPower: 'computers',
* };
*
```

- <a name="20.2"></a>
- <a name="commas--dangling"></a>
- 20.2 : eslint: `comma-dangle`

> - *Why? git diffs Babel*

```
* // bad -  git diff
* const hero = {
*      firstName: 'Florence',
* -    lastName: 'Nightingale'
* +    lastName: 'Nightingale',
* +    inventorOf: ['coxcomb chart', 'modern nursing']
* };
* // good -  git diff
* const hero = {
*      firstName: 'Florence',
*      lastName: 'Nightingale',
* +    inventorOf: ['coxcomb chart', 'modern nursing'],
* };
*
```

•

```
* // bad
* const hero = {
*   firstName: 'Dana',
*   lastName: 'Scully'
* };
* const heroes = [
*   'Batman',
*   'Superman'
* ];
* // good
* const hero = {
*   firstName: 'Dana',
*   lastName: 'Scully',
* };
* const heroes = [
*   'Batman',
*   'Superman',
* ];
* // bad
* function createHero(
*   firstName,
*   lastName,
*   inventorOf
* ) {
*   // does nothing
* }
* // good
* function createHero(
*   firstName,
*   lastName,
*   inventorOf,
* ) {
*   // does nothing
* }
* // good (note that a comma must not appear after a "rest"
element)
* function createHero(
*   firstName,
*   lastName,
*   inventorOf,
*   ...heroArgs
* ) {
*   // does nothing
* }
* // bad
* createHero(
*   firstName,
*   lastName,
*   inventorOf
* );
* // good
* createHero(
*   firstName,
*   lastName,
*   inventorOf,
* );
* // good (note that a comma must not appear after a "rest"
element)
```

```
 * createHero(
 *   firstName,
 *   lastName,
 *   inventorOf,
 *   ...heroArgs
 * )
 *
```

back to top

Semicolons

<a name="21.1"></a>

- 21.1 **Yup.** eslint: `semi`

> - *Why? JavaScript `Automatic Semicolon Insertion`JavaScriptJavaScript*

```
 * // bad
 * (function () {
 *   const name = 'Skywalker'
 *   return name
 * })()
 * // good
 * (function () {
 *   const name = 'Skywalker';
 *   return name;
 * }());
 * // good,
 * ;(() => {
 *   const name = 'Skywalker';
 *   return name;
 * }());
 *
```

- Read more.

back to top

Type Casting & Coercion

<a name="22.1"></a>
<a name="coercion--explicit"></a>

- 22.1
- <a name="22.2"></a>
- <a name="coercion--strings"></a>
- 22.2 Strings: eslint: `no-new-wrappers`

```
•
  * // => this.reviewScore = 9;
  * // bad
  * const totalScore = new String(this.reviewScore); // typeof
  totalScore is "object" not "string"
  * // bad
  * const totalScore = this.reviewScore + ''; // invokes this.
  reviewScore.valueOf()
  * // bad
  * const totalScore = this.reviewScore.toString(); // string
  * // good
  * const totalScore = String(this.reviewScore);
  *
```

- <a name="22.3"></a>
- <a name="coercion--numbers"></a>
- 22.3 Numbers: `Number` {{parseInt}}string eslint: `radix`

```
•
  * const inputValue = '4';
  * // bad
  * const val = new Number(inputValue);
  * // bad
  * const val = +inputValue;
  * // bad
  * const val = inputValue >> 0;
  * // bad
  * const val = parseInt(inputValue);
  * // good
  * const val = Number(inputValue);
  * // good
  * const val = parseInt(inputValue, 10);
  *
```

- <a name="22.4"></a>
- <a name="coercion--comment-deviations"></a>
- 22.4 `parseInt` ,

```
•
  * // good
  * /**
  *   * parseInt
  *   * Bitshifting
  *   */
  * const val = inputValue >> 0;
  *
```

- <a name="22.5"></a>
- <a name="coercion--bitwise"></a>
- 22.5 : . 64-32source)32Discussion. 32 2,147,483,647:

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

- <a name="22.6"></a>
- <a name="coercion--booleans"></a>
- 22.6 :
  -
    ```
    * const age = 0;
    * // bad
    * const hasAge = new Boolean(age);
    * // good
    * const hasAge = Boolean(age);
    * // best
    * const hasAge = !!age;
    *
    ```

back to top

Naming Conventions

<a name="23.1"></a>
<a name="naming--descriptive"></a>

- 23.1 eslint: `id-length`
  -
    ```
    * // bad
    * function q() {
    *   // ...
    * }
    * // good
    * function query() {
    *   // ...
    * }
    *
    ```

- <a name="23.2"></a>
- <a name="naming--camelCase"></a>
- 23.2 eslint: `camelcase`
  -
    ```
    * // bad
    * const OBJEcttsssss = {};
    * const this_is_my_object = {};
    * function c() {}
    * // good
    * const thisIsMyObject = {};
    * function thisIsMyFunction() {}
    *
    ```

- <a name="23.3"></a>
- <a name="naming--PascalCase"></a>

- 23.3 eslint: `new-cap`

```
•

  * // bad
  * function user(options) {
  *   this.name = options.name;
  * }
  * const bad = new user({
  *   name: 'nope',
  * });
  * // good
  * class User {
  *   constructor(options) {
  *     this.name = options.name;
  *   }
  * }
  * const good = new User({
  *   name: 'yup',
  * });
  *
```

- <a name="23.4"></a>
- <a name="naming--leading-underscore"></a>
- 23.4 eslint: `no-underscore-dangle`

> - *Why? JavaScript "private"API "private"*

```
  * // bad
  * this.__firstName__ = 'Panda';
  * this.firstName_ = 'Panda';
  * this._firstName = 'Panda';
  * // good
  * this.firstName = 'Panda';
  *
```

- <a name="23.5"></a>
- <a name="naming--self-this"></a>
- 23.5 {{this}} ——Function#bind.

-

```
* // bad
* function foo() {
*   const self = this;
*   return function () {
*     console.log(self);
*   };
* }
* // bad
* function foo() {
*   const that = this;
*   return function () {
*     console.log(that);
*   };
* }
* // good
* function foo() {
*   return () => {
*     console.log(this);
*   };
* }
*
```

- <a name="23.6"></a>
- <a name="naming--filename-matches-export"></a>
- 23.6 export defaultAA.* import A

-

```
* // file 1 contents
* class CheckBox {
*   // ...
* }
* export default CheckBox;
* // file 2 contents
* export default function fortyTwo() { return 42; }
* // file 3 contents
* export default function insideDirectory() {}
* // in some other file
* // bad
* import CheckBox from './checkBox'; // PascalCase import/export,
camelCase filename
* import FortyTwo from './FortyTwo'; // PascalCase import
/filename, camelCase export
* import InsideDirectory from './InsideDirectory'; // PascalCase
import/filename, camelCase export
* // bad
* import CheckBox from './check_box'; // PascalCase import/export,
snake_case filename
* import forty_two from './forty_two'; // snake_case import
/filename, camelCase export
* import inside_directory from './inside_directory'; // snake_case
import, camelCase export
* import index from './inside_directory/index'; // requiring the
index file explicitly
* import insideDirectory from './insideDirectory/index'; //
requiring the index file explicitly
* // good
* import CheckBox from './CheckBox'; // PascalCase export/import
/filename
* import fortyTwo from './fortyTwo'; // camelCase export/import
/filename
* import insideDirectory from './insideDirectory'; // camelCase
export/import/directory name/implicit "index"
* // ^ supports both insideDirectory.js and insideDirectory/index.
js
*
```

- <a name="23.7"></a>
- <a name="naming--camelCase-default-export"></a>
- 23.7 export-default

-

```
* function makeStyleGuide() {
*   // ...
* }
* export default makeStyleGuide;
*
```

- <a name="23.8"></a>
- <a name="naming--PascalCase-singleton"></a>
- 23.8 export////

-

```
* const AirbnbStyleGuide = {
*   es6: {
*   }
* };
* export default AirbnbStyleGuide;
*
```

- <a name="22.9"></a>
- <a name="naming--Acronyms-and-Initialisms"></a>
- 22.9

> - *Why?*

```
* // bad
* import SmsContainer from './containers/SmsContainer';
* // bad
* const HttpRequests = [
*   // ...
* ];
* // good
* import SMSContainer from './containers/SMSContainer';
* // good
* const HTTPRequests = [
*   // ...
* ];
* // best
* import TextMessageContainer from './containers
/TextMessageContainer';
* // best
* const Requests = [
*   // ...
* ];
*
```

- <a name="23.10"></a>
- <a name="naming--uppercase"></a>
- 23.10
- #
- # const
- #

> - *Why?*
>   - ** const ——*
>   - ** —— export(e.g. EXPORTED_OBJECT.key)*

```
* // bad
* const PRIVATE_VARIABLE = 'should not be unnecessarily uppercased
within a file';
* // bad
* export const THING_TO_BE_CHANGED = 'should obviously not be
uppercased';
* // bad
* export let REASSIGNABLE_VARIABLE = 'do not use let with
uppercase variables';
* // ---
* // allowed but does not supply semantic value
* export const apiKey = 'SOMEKEY';
* // better in most cases
* export const API_KEY = 'SOMEKEY';
* // ---
* // bad - unnecessarily uppercases key while adding no semantic
value
* export const MAPPING = {
*   KEY: 'value'
* };
* // good
* export const MAPPING = {
*   key: 'value'
* };
*
```

back to top

Accessors

<a name="24.1"></a>
<a name="accessors--not-required"></a>

- 24.1
- <a name="24.2"></a>
- <a name="accessors--no-getters-setters"></a>
- 24.2 JavaScriptgetters/setters getVal()setVal('hello')accessor

-
```
 * // bad
 * class Dragon {
 *   get age() {
 *     // ...
 *   }
 *   set age(value) {
 *     // ...
 *   }
 * }
 * // good
 * class Dragon {
 *   getAge() {
 *     // ...
 *   }
 *   setAge(value) {
 *     // ...
 *   }
 * }
 *
```

- <a name="24.3"></a>
- <a name="accessors--boolean-prefix"></a>
- 24.3 /{{boolean}} `isVal()` `hasVal()`

-
```
 * // bad
 * if (!dragon.age()) {
 *   return false;
 * }
 * // good
 * if (!dragon.hasAge()) {
 *   return false;
 * }
 *
```

- <a name="24.4"></a>
- <a name="accessors--consistent"></a>
- 24.4 get()set()

-
```
 * class Jedi {
 *   constructor(options = {}) {
 *     const lightsaber = options.lightsaber || 'blue';
 *     this.set('lightsaber', lightsaber);
 *   }
 *   set(key, val) {
 *     this[key] = val;
 *   }
 *   get(key) {
 *     return this[key];
 *   }
 * }
 *
```

back to top

Events

<a name="25.1"></a>
<a name="events--hash"></a>

- 25.1 (DOMBackbone)

```
•
  * // bad
  * $(this).trigger('listingUpdated', listing.id);
  * ...
  * $(this).on('listingUpdated', (e, listingId) => {
  *   // do something with listingId
  * });
  *
```

- prefer:

```
•
  * // good
  * $(this).trigger('listingUpdated', { listingId: listing.id });
  * ...
  * $(this).on('listingUpdated', (e, data) => {
  *   // do something with data.listingId
  * });
  *
```

- **back to top**

jQuery

<a name="26.1"></a>
<a name="jquery--dollar-prefix"></a>

- 26.1 jQuery{{$}}

```
•
  * // bad
  * const sidebar = $('.sidebar');
  * // good
  * const $sidebar = $('.sidebar');
  * // good
  * const $sidebarBtn = $('.sidebar-btn');
  *
```

- <a name="26.2"></a>
- <a name="jquery--cache"></a>
- 26.2 jQuery

-
  ```
  * // bad
  * function setSidebar() {
  *   $('.sidebar').hide();
  *   // ...
  *   $('.sidebar').css({
  *     'background-color': 'pink'
  *   });
  * }
  * // good
  * function setSidebar() {
  *   const $sidebar = $('.sidebar');
  *   $sidebar.hide();
  *   // ...
  *   $sidebar.css({
  *     'background-color': 'pink'
  *   });
  * }
  *
  ```

- <a name="26.3"></a>
- <a name="jquery--queries"></a>
- 26.3 DOM{{$('.sidebar ul')}} `> $('.sidebar > ul')`. jsPerf
- <a name="26.4"></a>
- <a name="jquery--find"></a>
- 26.4 jQuery{{find}}

-
  ```
  * // bad
  * $('ul', '.sidebar').hide();
  * // bad
  * $('.sidebar').find('ul').hide();
  * // good
  * $('.sidebar ul').hide();
  * // good
  * $('.sidebar > ul').hide();
  * // good
  * $sidebar.find('ul').hide();
  *
  ```

back to top

ES5

<a name="27.1"></a>
<a name="es5-compat--kangax"></a>

- 27.1 KangaxES5.

back to top

ECMAScript 6+ (ES 2015+) Styles

<a name="28.1"></a>
<a name="es6-styles"></a>

- 28.1 ES6

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. ——Exponentiation Operator
13. ——Iterators and Generators
14. ——Modules
15. <a name="28.2"></a>
16. <a name="tc39-proposals"></a>
17. * 28.2 TC39 proposals TC39 stage 3
18. *

> *Why? , JavaScript JavaScript*
>
> 1. *

back to top

Standard Library

<a name="29.1"></a>
<a name="standard-library--isnan"></a>

- 29.1 `Number.isNaN` `isNaN`.
- eslint: `no-restricted-globals`

> - *Why?* `isNaN` `NaN` `true`
>
>   -

```
* // bad
* isNaN('1.2'); // false
* isNaN('1.2.3'); // true
* // good
* Number.isNaN('1.2.3'); // false
* Number.isNaN(Number('1.2.3')); // true
*
```

- <a name="29.2"></a>
- <a name="standard-library--isfinite"></a>
- 29.2 `Number.isFinite` `isFinite`.
- eslint: `no-restricted-globals`

> - *Why?*

```
* // bad
* isFinite('2e3'); // true
* // good
* Number.isFinite('2e3'); // false
* Number.isFinite(parseInt('2e3', 10)); // true
*
```

Testing

<a name="30.1"></a>
<a name="testing--yup"></a>

- 30.1 **Yup.**

```
•
  * function foo() {
  *   return true;
  * }
  *
```

- <a name="30.2"></a>
- <a name="testing--for-real"></a>
- 30.2 **No, but seriously**:*
- *
- * stub mock ——
- * Airbnb `mocha` `tape`
- * 100%
- * bug bug

back to top

The JavaScript Style Guide Guide

- Reference

};

Airbnb JavaScript () {

JavaScript A mostly reasonable approach to JavaScript

Types

- 1.1 :
    - `string`
    - `number`
    - `boolean`
    - `null`
    - `undefined`
    - `symbol`

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

  bar = 9;

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

    - Symbols polyfill symbol[] symbol

- 1.2 :
    - `object`
    - `array`
    - `function`

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

  bar[0] = 9;

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

back to top

References

- 2.1 `const` `var`. eslint: `prefer-const`, `no-const-assign`

  > *Why? bug*

  ```
  // bad
  var a = 1;
  var b = 2;

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

- 2.2 `let` `var`. eslint: `no-var`

  > *Why? `let` `var`*

  ```
  // bad
  var count = 1;
  if (true) {
    count += 1;
  }

  // good, use the let.
  let count = 1;
  if (true) {
    count += 1;
  }
  ```

- 2.3 `let` `const`

  ```
  // const  let
  {
    let a = 1;
    const b = 1;
  }
  console.log(a); // ReferenceError
  console.log(b); // ReferenceError
  ```

back to top

Objects

- 3.1 . eslint: `no-new-object`

  ```
  // bad
  const item = new Object();

  // good
  const item = {};
  ```

- 3.2

  > *Why? .*

  ```

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

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

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

- 3.3 . eslint: `object-shorthand`

```
// bad
const atom = {
  value: 1,

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

// good
const atom = {
  value: 1,

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

- 3.4 . eslint: `object-shorthand`

  > *Why?*

  ```
  const lukeSkywalker = 'Luke Skywalker';

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

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

- 3.5 .

  > *Why? .*

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

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

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

- 3.6 `''`. eslint: `quote-props`

  > *Why? JS*

  ```
  // bad
  const bad = {
    'foo': 3,
    'bar': 4,
    'data-blah': 5,
  };

  // good
  const good = {
    foo: 3,
    bar: 4,
    'data-blah': 5,
  };
  ```

- 3.7 `Object.prototypehasOwnProperty`, `propertyIsEnumerable`, `isPrototypeOf`

  > *Why? - { hasOwnProperty: false } - Object.create(null)*

  ```
  // bad
  console.log(object.hasOwnProperty(key));

  // good
  console.log(Object.prototype.hasOwnProperty.call(object, key));

  // best
  const has = Object.prototype.hasOwnProperty; //
  /* or */
  import has from 'has'; // https://www.npmjs.com/package/has
  // ...
  console.log(has.call(object, key));
  ```

- 3.8 `...`][`Object.assign`rest[`...`]

  - ^.^

  ```
  // very bad
  const original = { a: 1, b: 2 };
  const copy = Object.assign(original, { c: 3 }); // this mutates `original` _
  delete copy.a; // so does this

  // bad
  const original = { a: 1, b: 2 };
  const copy = Object.assign({}, original, { c: 3 }); // copy => { a: 1, b: 2, c: 3 }

  // good es6 ...
  const original = { a: 1, b: 2 };
  //
  const copy = { ...original, c: 3 }; // copy => { a: 1, b: 2, c: 3 }

  // rest
  const { a, ...noA } = copy; // noA => { b: 2, c: 3 }
  ```

back to top

Arrays

- 4.1 eslint: `no-array-constructor`

  ```
  // bad
  const items = new Array();

  // good
  const items = [];
  ```

- 4.2 Array#push

  ```
  const someStack = [];

  // bad
  someStack[someStack.length] = 'abracadabra';

  // good
  someStack.push('abracadabra');
  ```

- 4.3

  ```
  // bad
  const len = items.length;
  const itemsCopy = [];
  let i;

  for (i = 0; i < len; i += 1) {
    itemsCopy[i] = items[i];
  }

  // good
  const itemsCopy = [...items];
  ```

- 4.4 `...` `Array.from`

```
const foo = document.querySelectorAll('.foo');

// good
const nodes = Array.from(foo);

// best
const nodes = [...foo];
```

- 4.5 `Array.from`

```
const arrLike = { 0: 'foo', 1: 'bar', 2: 'baz', length: 3 };

// bad
const arr = Array.prototype.slice.call(arrLike);

// good
const arr = Array.from(arrLike);
```

- 4.6 `Array.from` ... map

```
// bad
const baz = [...foo].map(bar);

// good
const baz = Array.from(foo, bar);
```

- 4.7 return   return 8.2. eslint: `array-callback-return`

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

// good
[1, 2, 3].map(x => x + 1);

// bad -  acc undefined
[[0, 1], [2, 3], [4, 5]].reduce((acc, item, index) => {
  const flatten = acc.concat(item);
  acc[index] = flatten;
});

// good
[[0, 1], [2, 3], [4, 5]].reduce((acc, item, index) => {
  const flatten = acc.concat(item);
  acc[index] = flatten;
  return flatten;
});

// bad
inbox.filter((msg) => {
  const { subject, author } = msg;
  if (subject === 'Mockingbird') {
    return author === 'Harper Lee';
  } else {
    return false;
  }
});

// good
inbox.filter((msg) => {
  const { subject, author } = msg;
  if (subject === 'Mockingbird') {
    return author === 'Harper Lee';
  }

  return false;
});
```

- 4.8 [ ]

```
// bad
const arr = [
  [0, 1], [2, 3], [4, 5],
];

const objectInArray = [{
  id: 1,
}, {
  id: 2,
}];

const numberInArray = [
  1, 2,
];

// good
const arr = [[0, 1], [2, 3], [4, 5]];

const objectInArray = [
  {
    id: 1,
  },
  {
    id: 2,
  },
];

const numberInArray = [
  1,
  2,
];
```

back to top

Destructuring

- 5.1 eslint: `prefer-destructuring`

  > *Why? /*

  ```
  // bad
  function getFullName(user) {
    const firstName = user.firstName;
    const lastName = user.lastName;

    return `${firstName} ${lastName}`;
  }

  // good
  function getFullName(user) {
    const { firstName, lastName } = user;
    return `${firstName} ${lastName}`;
  }

  // best
  function getFullName({ firstName, lastName }) {
    return `${firstName} ${lastName}`;
  }
  ```

- 5.2 .

  ```
  const arr = [1, 2, 3, 4];

  // bad
  const first = arr[0];
  const second = arr[1];

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

- 5.3

  > *Why?*

  ```
  // bad
  function processInput(input) {
    //
    return [left, right, top, bottom];
  }

  //
  const [left, __, top] = processInput(input);

  // good
  function processInput(input) {
    // oops
  ```

```
  return { left, right, top, bottom };
}

//
const { left, top } = processInput(input);
```

back to top

Strings

- 6.1 string `' '` eslint: `quotes`

  ```
  // bad
  const name = "Capt. Janeway";

  // bad -
  const name = `Capt. Janeway`;

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

- 6.2 100string

  > *Why?*

  ```
  // bad
  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.';

  // bad
  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.';

  // good
  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.';
  ```

- 6.3 eslint: `prefer-template` `template-curly-spacing`

  > *Why?*

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

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

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

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

- 6.4 `eval()` eslint: `no-eval`

- 6.5 eslint: `no-useless-escape`

  > *Why?*

  ```
  // bad
  const foo = '\'this\' \i\s \"quoted\"';

  // good
  const foo = '\'this\' is "quoted"';

  //best
  const foo = `my name is '${name}'`;
  ```

back to top

Functions

- 7.1 eslint: `func-style`

  > *const func = function () {}*

  > *function func() {}*

  > *Why? babel (Discussion)*

  > *Why?   Why? Function declarations are hoisted, which means that it's easy - too easy - to reference the function before it is defined in the file. This harms readability and maintainability. If you find that a function's definition is large or complex enough that it is interfering with understanding the rest of the file, then perhaps it's time to extract it to its own module! Don't forget to explicitly name the expression, regardless of whether or not the name is inferred from the containing variable (which is often the case in modern browsers or when using compilers such as Babel). This eliminates any assumptions made about the Error's call stack. (Discussion)*

```
// bad
function foo() {
  // ...
}

// bad
const foo = function () {
  // ...
};

// good
// lexical name distinguished from the variable-referenced invocation(s)
//
const short = function longUniqueMoreDescriptiveLexicalFoo() {
  // ...
};
```

- 7.2 eslint: `wrap-iife`

  > *Why? immediately invoked function expression = IIFE Why? - Why? IIFE*

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

- 7.3 ifwhileno-`loop-func` eslint: `no-loop-func`

- 7.4 Note: ECMA-262 [`block`]

```
// bad
if (currentUser) {
  function test() {
    console.log('Nope.');
  }
}

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

- 7.5 `arguments` `arguments` `arguments`

```
// bad
function foo(name, options, arguments) {
  // ...
}

// good
function foo(name, options, args) {
  // ...
}
```

- 7.6 `arguments`rest`...` eslint: `prefer-rest-params`

  > *Why? `...`restarguments*

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

  // good
  function concatenateAll(...args) {
    return args.join('');
  }
  ```

- 7.7

  ```
  // really bad
  function handleThings(opts) {
    //  arguments
    //   opts  false,  {}
    //  bug
    opts = opts || {};
    // ...
  }

  // still bad
  function handleThings(opts) {
    if (opts === void 0) {
      opts = {};
    }
    // ...
  }

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

- 7.8

  > *Why? a*

  ```
  var b = 1;
  // bad
  function count(a = b++) {
    console.log(a);
  }
  count();  // 1
  count();  // 2
  count(3); // 3
  count();  // 3
  ```

- 7.9

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

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

- 7.10 eslint: `no-new-func`

  > *Why? eval()*

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

  // still bad
  var subtract = Function('a', 'b', 'return a - b');
  ```

- 7.11 eslint: `space-before-function-paren` `space-before-blocks`

  >

> *Why? //*

```
// bad
const f = function(){};
const g = function (){};
const h = function() {};

// good
const x = function () {};
const y = function a() {};
```

- 7.12 . eslint: `no-param-reassign`

  > *Why?*

  ```
  // bad
  function f1(obj) {
    obj.key = 1;
  };

  // good
  function f2(obj) {
    const key = Object.prototype.hasOwnProperty.call(obj, 'key') ? obj.key : 1;
  };
  ```

- 7.13 eslint: `no-param-reassign`

  > *Why? `arguments`V8*

  ```
  // bad
  function f1(a) {
    a = 1;
    // ...
  }

  function f2(a) {
    if (!a) { a = 1; }
    // ...
  }

  // good
  function f3(a) {
    const b = a || 1;
    // ...
  }

  function f4(a = 1) {
    // ...
  }
  ```

- 7.14 spread`...` eslint: `prefer-spread`

  > *Why? `apply`new*

  ```
  // bad
  const x = [1, 2, 3, 4, 5];
  console.log.apply(console, x);

  // good
  const x = [1, 2, 3, 4, 5];
  console.log(...x);

  // bad
  new (Function.prototype.bind.apply(Date, [null, 2016, 8, 5]));

  // good
  new Date(...[2016, 8, 5]);
  ```

- 7.15

  ```
  // bad
  function foo(bar,
               baz,
               quux) {
    // ...
  }

  // good
  function foo(
    bar,
  ```

```
  baz,
  quux,
) {
  // ...
}

// bad
console.log(foo,
  bar,
  baz);

// good
console.log(
  foo,
  bar,
  baz,
);
```

back to top

Arrow Functions

- 8.1 eslint: `prefer-arrow-callback`, `arrow-spacing`

  > *Why? `this`*

  > *Why?*

  ```
  // 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;
  });
  ```

- 8.2 return `return` eslint: `arrow-parens`, `arrow-body-style`

  > *Why?*

  ```
  // 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
  }));

  // return
  function foo(callback) {
    const val = callback();
    if (val === true) {
      // Do something if callback returns true
    }
  }

  let bool = false;

  // bad
  // return bool = true,
  foo(() => bool = true);

  // good
  ```

```
foo(() => {
  bool = true;
});
```

- 8.3

  > *Why?*

  ```
  // bad
  ['get', 'post', 'put'].map(httpMethod => Object.prototype.hasOwnProperty.call(
      httpMagicObjectWithAVeryLongName,
      httpMethod
    )
  );

  // good
  ['get', 'post', 'put'].map(httpMethod => (
    Object.prototype.hasOwnProperty.call(
      httpMagicObjectWithAVeryLongName,
      httpMethod
    )
  ));
  ```

- 8.4 “always” option for eslint. eslint: `arrow-parens`

  > *Why?*

  ```
  // bad
  [1, 2, 3].map((x) => x * x);

  // good
  [1, 2, 3].map(x => x * x);

  // good
  [1, 2, 3].map(number => (
    `A long string with the ${number}. It’s so long that we don’t want it to take up space on the .
  map line!`
  ));

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

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

- 8.5 (=>)<=, >=. eslint: `no-confusing-arrow`

  ```
  // 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;
  };
  ```

- 8.6 return eslint: `implicit-arrow-linebreak`

  ```
  // bad
  (foo) =>
    bar;

  (foo) =>
    (bar);

  // good
  (foo) => bar;
  (foo) => (bar);
  ```

```
(foo) => (
   bar
)
```

back to top

Classes & Constructors

- 9.1 classprototype

  > *Why? class*

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

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

- 9.2 extends

  > *Why? instanceof*

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

  // good
  class PeekableQueue extends Queue {
    peek() {
      return this._queue[0];
    }
  }
  ```

- 9.3 this

  ```
  // bad
  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

  // good
  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 toString()

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

    getName() {
      return this.name;
    }

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

- 9.5 eslint: `no-useless-constructor`

  ```
  // 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';
    }
  }
  ```

- 9.6 eslint: `no-dupe-class-members`

  > *Why? —— bug*

  ```
  // bad
  class Foo {
    bar() { return 1; }
    bar() { return 2; }
  }

  // good
  class Foo {
    bar() { return 1; }
  }

  // good
  class Foo {
    bar() { return 2; }
  }
  ```

back to top

Modules

- 10.1 (`import`/`export`)

  > *Why?*

```
// 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;
```

- 10.2 import *

  > *Why?*

```
// bad
import * as AirbnbStyleGuide from './AirbnbStyleGuide';

// good
import AirbnbStyleGuide from './AirbnbStyleGuide';
```

- 10.3 importexport

  > *Why?*

```
// bad
// filename es6.js
export { es6 as default } from './AirbnbStyleGuide';

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

- 10.4 import eslint: `no-duplicate-imports`

  > *Why? import*

```
// 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';
```

- 10.5 eslint: `import/no-mutable-exports`

  > *Why?*

```
// bad
let foo = 3;
export { foo }

// good
const foo = 3;
export { foo }
```

- 10.6 `export default` eslint: `import/prefer-default-export`

  > *Why?*

```
// bad
export function foo() {}

// good
export default function foo() {}
```

- 10.7 `import` eslint: `import/first`

  > *Why? `import`*

  ```
  // bad
  import foo from 'foo';
  foo.init();

  import bar from 'bar';

  // good
  import foo from 'foo';
  import bar from 'bar';

  foo.init();
  ```

- 10.8 import

  > *Why?*

  ```
  // bad
  import {longNameA, longNameB, longNameC, longNameD, longNameE} from 'path';

  // good
  import {
    longNameA,
    longNameB,
    longNameC,
    longNameD,
    longNameE,
  } from 'path';
  ```

- 10.9 importWebpack loader eslint: `import/no-webpack-loader-syntax`

  > *Why? Webpackimportwebpack.config.jswebpack loader*

  ```
  // bad
  import fooSass from 'css!sass!foo.scss';
  import barCss from 'style!css!bar.css';

  // good
  import fooSass from 'foo.scss';
  import barCss from 'bar.css';
  ```

back to top

Iterators and Generators

- 11.1 JavaScript`for-in` `for-of` eslint: `no-iterator` `no-restricted-syntax`

  > *Why?*

  > *Why? `map()` / `every()` / `filter()` / `find()` / `findIndex()` / `reduce()` / `some()` / ... , `Object.keys()` / `Object.values()` / `Object.entries()`*

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

  // bad
  let sum = 0;
  for (let num of numbers) {
    sum += num;
  }
  sum === 15;

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

  // best (use the functional force)
  ```

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

// bad
const increasedByOne = [];
for (let i = 0; i < numbers.length; i++) {
  increasedByOne.push(numbers[i] + 1);
}

// good
const increasedByOne = [];
numbers.forEach(num => increasedByOne.push(num + 1));

// best (keeping it functional)
const increasedByOne = numbers.map(num => num + 1);
```

- 11.2 generator

  > *Why? es5*

- 11.3 , eslint: `generator-star-spacing`

  > *Why? `function` `*` - `*functionfunction*function`*

```
// bad
function * foo() {
  // ...
}

// bad
const bar = function * () {
  // ...
}

// bad
const baz = function *() {
  // ...
}

// bad
const quux = function*() {
  // ...
}

// bad
function*foo() {
  // ...
}

// bad
function *foo() {
  // ...
}

// very bad
function
*
foo() {
  // ...
}

// very bad
const wat = function
*
() {
  // ...
}

// good
function* foo() {
  // ...
}

// good
const foo = function* () {
  // ...
}
```

back to top

Properties

- 12.1 . eslint: dot-notation

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

  // bad
  const isJedi = luke['jedi'];

  // good
  const isJedi = luke.jedi;
  ```

- 12.2 []

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

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

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

- 12.3 ** eslint: no-restricted-properties.

  ```
  // bad
  const binary = Math.pow(2, 10);

  // good
  const binary = 2 ** 10;
  ```

back to top

Variables

- 13.1 const eslint: no-undef prefer-const

  ```
  // bad
  superPower = new SuperPower();

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

- 13.2 const let eslint: one-var

  > *Why? ; ,*

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

  // bad
  // (compare to above, and try to spot the mistake)
  const items = getItems(),
      goSportsTeam = true;
      dragonball = 'z';

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

- 13.3 constlet

  > *Why?*

```javascript
// bad
let i, len, dragonball,
    items = getItems(),
    goSportsTeam = true;

// bad
let i;
const items = getItems();
let dragonball;
const goSportsTeam = true;
let len;

// good
const goSportsTeam = true;
const items = getItems();
let dragonball;
let i;
let length;
```

- 13.4

  > *Why? let const*

  ```javascript
  // bad - unnecessary function call
  function checkName(hasName) {
    const name = getName();

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

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

    return name;
  }

  // good
  function checkName(hasName) {
    if (hasName === 'test') {
      return false;
    }

    //
    const name = getName();

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

    return name;
  }
  ```

- 13.5 eslint: `no-multi-assign`

  > *Why?*

  ```javascript
  // bad
  (function example() {
    // JavaScript
    // let a = ( b = ( c = 1 ) );
    // let  a ;  b  c
    let a = b = c = 1;
  }());

  console.log(a); // undefined
  console.log(b); // 1
  console.log(c); // 1

  // good
  (function example() {
    let a = 1;
    let b = a;
    let c = a;
  }());

  console.log(a); // undefined
  console.log(b); // undefined
  console.log(c); // undefined
  ```

```
// `const`
```

- 13.6 ++ --. eslint `no-plusplus`

  > *Why? eslint num + = 1num ++num ++/*

  ```
  // bad

  let array = [1, 2, 3];
  let num = 1;
  num++;
  --num;

  let sum = 0;
  let truthyCount = 0;
  for(let i = 0; i < array.length; i++){
    let value = array[i];
    sum += value;
    if (value) {
      truthyCount++;
    }
  }

  // good

  let array = [1, 2, 3];
  let num = 1;
  num += 1;
  num -= 1;

  const sum = array.reduce((a, b) => a + b, 0);
  const truthyCount = array.filter(Boolean).length;
  ```

- 13.7 = / `max-len` eslint `operator-linebreak`.

  > *Why? =*

  ```
  // bad
  const foo =
    superLongLongLongLongLongLongLongLongFunctionName();

  // bad
  const foo
    = 'superLongLongLongLongLongLongLongLongString';

  // good
  const foo = (
    superLongLongLongLongLongLongLongLongFunctionName()
  );

  // good
  const foo = 'superLongLongLongLongLongLongLongLongString';
  ```

- 13.8 eslint: `no-unused-vars`

  > *Why?*

  ```
  // bad

  var some_unused_var = 42;

  //
  var y = 10;
  y = 5;

  //
  var z = 0;
  z = z + 1;

  //
  function getX(x, y) {
      return x;
  }

  // good
  function getXPlusY(x, y) {
    return x + y;
  }

  var x = 1;
  ```

```
var y = a + 2;

alert(getXPlusY(x, y));

// 'type'   rest
//
var { type, ...coords } = data;
// 'coords'  'type'  'data'
```

back to top

Hoisting

- 14.1 `varconst let` —— Temporal Dead Zones (TDZ) typeof.

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

//
//  declaredButNotAssigned
function example() {
  console.log(declaredButNotAssigned); // => undefined
  var declaredButNotAssigned = true;
}

//
//
function example() {
  let declaredButNotAssigned;
  console.log(declaredButNotAssigned); // => undefined
  declaredButNotAssigned = true;
}

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

- 14.2 `var`

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

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

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

- 14.3

```
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');
  };
}

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

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

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

- 14.4

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

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

- JavaScript Scoping & Hoisting by Ben Cherry.

back to top

Comparison Operators & Equality

- 15.1 `===` `!==` `==` `!=`. eslint: `eqeqeq`

- 15.2 'if'`ToBoolean'

  - Objects true
  - Undefined false
  - Null false
  - Booleans the value of the boolean
  - Numbers
    - +0, -0, or NaN false
    - true
  - Strings
    - `''` false
    - true

  ```
  if ([0] && []) {
    // true
    // true
  }
  ```

- 15.3

  ```
  // bad
  if (isValid === true) {
    // ...
  }

  // good
  if (isValid) {
    // ...
  }

  // bad
  if (name) {
    // ...
  }

  // good
  if (name !== '') {
    // ...
  }

  // bad
  if (collection.length) {
    // ...
  }

  // good
  if (collection.length > 0) {
    // ...
  }
  ```

- 15.4 Angus CrollJavaScript —— Truth Equality and JavaScript

- 15.5 `casedefault`(e.g. `let`, `const`, `function`, and `class`). eslint rules: `no-case-declarations`.

  > *Why?* `switchcase case`

```javascript
// bad
switch (foo) {
  case 1:
    let x = 1;
    break;
  case 2:
    const y = 2;
    break;
  case 3:
    function f() {
      // ...
    }
    break;
  default:
    class C {}
}

// good
switch (foo) {
  case 1: {
    let x = 1;
    break;
  }
  case 2: {
    const y = 2;
    break;
  }
  case 3: {
    function f() {
      // ...
    }
    break;
  }
  case 4:
    bar();
    break;
  default: {
    class C {}
  }
}
```

- 15.6

  eslint rules: `no-nested-ternary`.

  ```javascript
  // bad
  const foo = maybe1 > maybe2
    ? "bar"
    : value1 > value2 ? "baz" : null;

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

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

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

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

- 15.7

  eslint rules: `no-unneeded-ternary`.

  ```javascript
  // 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;
  ```

- 15.8 (+, -, *, & /) eslint: `no-mixed-operators`

  > *Why?*

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

// bad
const bar = a ** b - 5 % d;

// bad
// (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;
```

back to top

Blocks

- 16.1 eslint: `nonblock-statement-body-position`

```
// bad
if (test)
  return false;

// good
if (test) return false;

// good
if (test) {
  return false;
}

// bad
function foo() { return false; }

// good
function bar() {
  return false;
}
```

- 16.2 `ifelseif` eslint: `brace-style`

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

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

- 16.3 `if return  else if return else if return return if` eslint: `no-else-return`

```
// 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;
  }
}
```

back to top

Control Statements

- 17.1 (`if`, `while`) ()

  > *Why?*

  ```
  // bad
  if ((foo === 123 || bar === 'abc') && doesItLookGoodWhenItBecomesThatLong() && isThisReallyHappen
  ing()) {
    thing1();
  }

  // bad
  if (foo === 123 &&
    bar === 'abc') {
    thing1();
  }

  // bad
  if (foo === 123
    && bar === 'abc') {
  ```

```
  thing1();
}

// bad
if (
  foo === 123 &&
  bar === 'abc'
) {
  thing1();
}

// good
if (
  foo === 123
  && bar === 'abc'
) {
  thing1();
}

// good
if (
  (foo === 123 || bar === 'abc')
  && doesItLookGoodWhenItBecomesThatLong()
  && isThisReallyHappening()
) {
  thing1();
}

// good
if (foo === 123 && bar === 'abc') {
  thing1();
}
```

- 17.2

  ```
  // bad
  !isRunning && startRunning();

  // good
  if (!isRunning) {
    startRunning();
  }
  ```

back to top

Comments

- 18.1 `/** ... */`

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

    // ...

    return element;
  }

  // good
  /** * make() returns a new element * based on the passed-in tag name */
  function make(tag) {

    // ...

    return element;
  }
  ```

- 18.2 //

  ```
  // bad
  const active = true;  // is current tab

  // good
  // is current tab
  ```

```
const active = true;

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

  return type;
}

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

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

  return type;
}

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

  return type;
}
```

- 18.3 eslint: `spaced-comment`

```
// bad
//is current tab
const active = true;

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

// bad
/** *make() returns a new element *based on the passed-in tag name */
function make(tag) {

  // ...

  return element;
}

// good
/** * make() returns a new element * based on the passed-in tag name */
function make(tag) {

  // ...

  return element;
}
```

- 18.4 FIXME'TODO' `FIXME - TODO -`

- 18.5 // `FIXME:`

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

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

- 18.6 // `TODO:`

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

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

back to top

Whitespace

- 19.1 tab. eslint: `indent`

  ```
  // bad
  function foo() {
  const name;
  }

  // bad
  function bar() {
  const name;
  }

  // good
  function baz() {
  const name;
  }
  ```

- 19.2 eslint: `space-before-blocks`

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

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

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

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

- 19.3 (`if`, `while`) eslint: `keyword-spacing`

  ```
  // bad
  if(isJedi) {
    fight ();
  }

  // good
  if (isJedi) {
    fight();
  }

  // bad
  function fight () {
    console.log ('Swooosh!');
  }

  // good
  function fight() {
    console.log('Swooosh!');
  }
  ```

- 19.4 eslint: `space-infix-ops`

  ```
  // bad
  const x=y+5;

  // good
  const x = y + 5;
  ```

- 19.5 . eslint: `eol-last`

  ```
  // bad
  import { es6 } from './AirbnbStyleGuide';
    // ...
  export default es6;

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


  // good
  import { es6 } from './AirbnbStyleGuide';
    // ...
  export default es6;
  ```

- 19.6 >2eslint: `newline-per-chained-call` `no-whitespace-before-property`

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

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

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

  // bad
  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);

  // good
  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);

  // good
  const leds = stage.selectAll('.led').data(data);
  ```

- 19.7

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

  // good
  if (foo) {
    return bar;
  }

  return baz;

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

```
// good
const obj = {
  foo() {
  },

  bar() {
  },
};

return obj;

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

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

  function bar() {
  },
];

return arr;
```

- 19.8 eslint: `padded-blocks`

```
// bad
function bar() {

  console.log(foo);

}

// also bad
if (baz) {

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

}

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

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

- 19.9 eslint: `space-in-parens`

```
// bad
function bar( foo ) {
  return foo;
}

// good
function bar(foo) {
  return foo;
}

// bad
if ( foo ) {
  console.log(foo);
}

// good
if (foo) {
  console.log(foo);
}
```

- 19.10 eslint: `array-bracket-spacing`

```
// bad
const foo = [ 1, 2, 3 ];
console.log(foo[ 0 ]);

// good
const foo = [1, 2, 3];
console.log(foo[0]);
```

- 19.11 eslint: `object-curly-spacing`

```
// bad
const foo = {clark: 'kent'};

// good
const foo = { clark: 'kent' };
```

- 19.12 100
- ——strings--line-length eslint: `max-len`

> *Why?*

```
// bad
const foo = jsonData && jsonData.foo && jsonData.foo.bar && jsonData.foo.bar.baz && jsonData.foo.
bar.baz.quux && jsonData.foo.bar.baz.quux.xyzzy;

// bad
$.ajax({ method: 'POST', url: 'https://airbnb.com/', data: { name: 'John' } }).done(() => console.
log('Congratulations!')).fail(() => console.log('You have failed this city.'));

// good
const foo = jsonData
  && jsonData.foo
  && jsonData.foo.bar
  && jsonData.foo.bar.baz
  && jsonData.foo.bar.baz.quux
  && jsonData.foo.bar.baz.quux.xyzzy;

// good
$.ajax({
  method: 'POST',
  url: 'https://airbnb.com/',
  data: { name: 'John' },
})
  .done(() => console.log('Congratulations!'))
  .fail(() => console.log('You have failed this city.'));
```

- 19.13 —— `{ }` eslint: `block-spacing`

```
// bad
function foo() {return true;}
if (foo) { bar = 0;}

// good
function foo() { return true; }
if (foo) { bar = 0; }
```

- 19.14 `,` `,` eslint: `comma-spacing`

```
// bad
var foo = 1,bar = 2;
var arr = [1 , 2];

// good
var foo = 1, bar = 2;
var arr = [1, 2];
```

- 19.15 eslint: `computed-property-spacing`

```
// bad
obj[foo ]
obj[ 'foo']
var x = {[ b ]: a}
obj[foo[ bar ]]

// good
```

```
obj[foo]
obj['foo']
var x = { [b]: a }
obj[foo[bar]]
```

- 19.16 eslint: `func-call-spacing`

  ```
  // bad
  func ();

  func
  ();

  // good
  func();
  ```

- 19.17 `key` `value` eslint: `key-spacing`

  ```
  // bad
  var obj = { "foo" : 42 };
  var obj2 = { "foo":42 };

  // good
  var obj = { "foo": 42 };
  ```

- 19.18 eslint: `no-trailing-spaces`

- 19.19 eslint: `no-multiple-empty-lines`

  ```
  // bad
  var x = 1;
  ```

```
var y = 2;

// good
var x = 1;

var y = 2;
```

```
```
<!-- markdownlint-enable MD012 -->
```

back to top

## Commas

- 20.1 eslint: `comma-style`

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

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

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

  // good
  const hero = {
    firstName: 'Ada',
    lastName: 'Lovelace',
  ```
  ```

```
  birthYear: 1815,
  superPower: 'computers',
};
```

- 20.2 : eslint: `comma-dangle`

  > *Why? git diffs Babel*

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

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

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

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

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

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

// bad
function createHero(
  firstName,
  lastName,
  inventorOf
) {
  // does nothing
}

// good
function createHero(
  firstName,
  lastName,
  inventorOf,
) {
  // does nothing
}

// good (note that a comma must not appear after a "rest" element)
function createHero(
  firstName,
  lastName,
  inventorOf,
  ...heroArgs
) {
  // does nothing
}

// bad
createHero(
  firstName,
  lastName,
  inventorOf
);

// good
createHero(
  firstName,
  lastName,
  inventorOf,
);

// good (note that a comma must not appear after a "rest" element)
createHero(
  firstName,
  lastName,
  inventorOf,
  ...heroArgs
)
```

back to top

Semicolons

- 21.1 Yup. eslint: `semi`

  > *Why? JavaScript `Automatic Semicolon Insertion`JavaScriptJavaScript*

  ```
  // bad
  (function () {
    const name = 'Skywalker'
    return name
  })()

  // good
  (function () {
    const name = 'Skywalker';
    return name;
  }());

  // good,
  ;(() => {
    const name = 'Skywalker';
    return name;
  }());
  ```

  Read more.

back to top

Type Casting & Coercion

- 22.1

- 22.2 Strings: eslint: `no-new-wrappers`

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

  // bad
  const totalScore = new String(this.reviewScore); // typeof totalScore is "object" not "string"

  // bad
  const totalScore = this.reviewScore + ''; // invokes this.reviewScore.valueOf()

  // bad
  const totalScore = this.reviewScore.toString(); // string

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

- 22.3 Numbers: `Number` `parseInt`string eslint: `radix`

  ```
  const inputValue = '4';

  // bad
  const val = new Number(inputValue);

  // bad
  const val = +inputValue;

  // bad
  const val = inputValue >> 0;
  ```

```
// bad
const val = parseInt(inputValue);

// good
const val = Number(inputValue);

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

- 22.4 `parseInt` ,

  ```
  // good
  /** * parseInt * Bitshifting */
  const val = inputValue >> 0;
  ```

- 22.5 : . 64-32source)32Discussion. 32 2,147,483,647:

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

- 22.6 :

  ```
  const age = 0;

  // bad
  const hasAge = new Boolean(age);

  // good
  const hasAge = Boolean(age);

  // best
  const hasAge = !!age;
  ```

back to top

Naming Conventions

- 23.1 eslint: `id-length`

  ```
  // bad
  function q() {
    // ...
  }

  // good
  function query() {
    // ...
  }
  ```

- 23.2 eslint: `camelcase`

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

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

- 23.3 eslint: `new-cap`

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

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

  // good
  class User {
  ```

```
  constructor(options) {
    this.name = options.name;
  }
}

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

- 23.4 eslint: `no-underscore-dangle`

> *Why? JavaScript “private”API “private”*

```
// bad
this.__firstName__ = 'Panda';
this.firstName_ = 'Panda';
this._firstName = 'Panda';

// good
this.firstName = 'Panda';
```

- 23.5 `this` ——Function#bind.

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

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

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

- 23.6 export defaultAA.* import A

```
// file 1 contents
class CheckBox {
  // ...
}
export default CheckBox;

// file 2 contents
export default function fortyTwo() { return 42; }

// file 3 contents
export default function insideDirectory() {}

// in some other file
// bad
import CheckBox from './checkBox'; // PascalCase import/export, camelCase filename
import FortyTwo from './FortyTwo'; // PascalCase import/filename, camelCase export
import InsideDirectory from './InsideDirectory'; // PascalCase import/filename, camelCase export

// bad
import CheckBox from './check_box'; // PascalCase import/export, snake_case filename
import forty_two from './forty_two'; // snake_case import/filename, camelCase export
import inside_directory from './inside_directory'; // snake_case import, camelCase export
import index from './inside_directory/index'; // requiring the index file explicitly
import insideDirectory from './insideDirectory/index'; // requiring the index file explicitly

// good
import CheckBox from './CheckBox'; // PascalCase export/import/filename
import fortyTwo from './fortyTwo'; // camelCase export/import/filename
import insideDirectory from './insideDirectory'; // camelCase export/import/directory name
/implicit "index"
// ^ supports both insideDirectory.js and insideDirectory/index.js
```

- 23.7 export-default

  ```
  function makeStyleGuide() {
    // ...
  }

  export default makeStyleGuide;
  ```

- 23.8 export////

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

  export default AirbnbStyleGuide;
  ```

- 22.9

  > *Why?*

  ```
  // bad
  import SmsContainer from './containers/SmsContainer';

  // bad
  const HttpRequests = [
    // ...
  ];

  // good
  import SMSContainer from './containers/SMSContainer';

  // good
  const HTTPRequests = [
    // ...
  ];

  // best
  import TextMessageContainer from './containers/TextMessageContainer';

  // best
  const Requests = [
    // ...
  ];
  ```

- 23.10

  1.
  2. `const`
  3.

  > *Why?*
  >
  > - *`const` ——*
  > - *—— export(e.g. `EXPORTED_OBJECT.key`)*

  ```
  // bad
  const PRIVATE_VARIABLE = 'should not be unnecessarily uppercased within a file';

  // bad
  export const THING_TO_BE_CHANGED = 'should obviously not be uppercased';

  // bad
  export let REASSIGNABLE_VARIABLE = 'do not use let with uppercase variables';

  // ---

  // allowed but does not supply semantic value
  export const apiKey = 'SOMEKEY';

  // better in most cases
  export const API_KEY = 'SOMEKEY';

  // ---

  // bad - unnecessarily uppercases key while adding no semantic value
  export const MAPPING = {
    KEY: 'value'
  };

  // good
  ```

```
export const MAPPING = {
  key: 'value'
};
```

back to top

Accessors

- 24.1

- 24.2 JavaScriptgetters/setters getVal()setVal('hello')accessor

  ```
  // bad
  class Dragon {
    get age() {
      // ...
    }

    set age(value) {
      // ...
    }
  }

  // good
  class Dragon {
    getAge() {
      // ...
    }

    setAge(value) {
      // ...
    }
  }
  ```

- 24.3 /boolean `isVal()` `hasVal()`

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

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

- 24.4 get()set()

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

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

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

back to top

Events

- 25.1 (DOMBackbone)

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

  ...
  ```

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

prefer:

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

...

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

back to top

jQuery

- 26.1 jQuery$

  ```
  // bad
  const sidebar = $('.sidebar');

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

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

- 26.2 jQuery

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

    // ...

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

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

    // ...

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

- 26.3 DOM`$('.sidebar ul')` > `$('.sidebar > ul')`. jsPerf

- 26.4 jQuery`find`

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

  // bad
  $('.sidebar').find('ul').hide();

  // good
  $('.sidebar ul').hide();

  // good
  $('.sidebar > ul').hide();

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

back to top

ES5

- 27.1 KangaxES5.

back to top

ECMAScript 6+ (ES 2015+) Styles

- 28.1 ES6
- 28.2 TC39 proposals TC39 stage 3

> *Why? , JavaScript JavaScript*

back to top

Standard Library

- 29.1 `Number.isNaN` `isNaN`. eslint: `no-restricted-globals`

> *Why? `isNaN` `NaN` `true`*

```
// bad
isNaN('1.2'); // false
isNaN('1.2.3'); // true

// good
Number.isNaN('1.2.3'); // false
Number.isNaN(Number('1.2.3')); // true
```

- 29.2 `Number.isFinite` `isFinite`. eslint: `no-restricted-globals`

> *Why?*

```
// bad
isFinite('2e3'); // true

// good
Number.isFinite('2e3'); // false
Number.isFinite(parseInt('2e3', 10)); // true
```

Testing

- 30.1 Yup.

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

- 30.2 No, but seriously:
    -
    -
    - stub mock ——
    - Airbnb `mocha` `tape`
    - 100%
    - bug bug

back to top

The JavaScript Style Guide Guide

- Reference

};