Airbnb JS Style Guide

Whitespace
Commas
Semicolons

Full Stack Web Development

Whitespace

Commas

Semicolons

Whitespace

Whitespace

- Use soft tabs (space character) set to 2 spaces. eslint: **`indent`**

```
// bad
function foo() {
····let name;
}

// bad
function bar() {
·let name;
}

// good
function baz() {
··let name;
}
```

Whitespace

- Place 1 space before the leading brace. 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",
});
```

Whitespace

- Place 1 space before the opening parenthesis in control statements (**`if`** , **`while`** etc.). Place no space between the argument list and the function name in function calls and declarations. eslint: **`keyword-spacing`**

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

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

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

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

Whitespace

- Set off operators with spaces. eslint: **space-infix-ops**

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

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

- End files with a single newline character. 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;↵
```

Whitespace

- Use indentation when making long method chains (more than 2 method chains). Use a leading dot, which emphasizes that the line is a method call, not a new statement. eslint: **`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);
const svg = leds.enter().append("svg:svg");
svg.classed("led", true).attr("width", (radius + margin) * 2);
const g = svg.append("svg:g");
g.attr("transform", `translate(${radius + margin},${radius + margin})`).call(tron.led);
```

Whitespace

- Leave a blank line after blocks and before the next statement.

```
// 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() {
  },
};
```

Whitespace

- Do not pad your blocks with blank lines. eslint: **padded-blocks**

```
// bad
function bar() {

  console.log(foo);

}

// bad
if (baz) {

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

}

// bad
class Foo {

  constructor(bar) {
    this.bar = bar;
  }
}

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

Whitespace

- Do not add spaces inside parentheses. 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);
}
```

Whitespace

- Do not add spaces inside brackets. eslint: **array-bracket-spacing**

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

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

- Add spaces inside curly braces. eslint: **object-curly-spacing**

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

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

Whitespace

- Avoid having lines of code that are longer than 100 characters (including whitespace). Note: per **above**, long strings are exempt from this rule, and should not be broken up. eslint: **`max-len`**

> *"Why? This ensures readability and maintainability."*

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

// bad
$.ajax({ method: "POST", url: "https://airbnb.com/", data: { name: "John" } }).done(() => console.log("Congratul

// 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."));
```

Commas

Whitespace

- Leading commas: **Nope.** 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",
};
```

Whitespace

- Additional trailing comma: **Yup.** eslint: **`comma-dangle`**

> *"Why? This leads to cleaner git diffs. Also, transpilers like Babel will remove the additional trailing comma in the transpiled code which means you don't have to worry about the trailing comma problem in legacy browsers."*

```
// bad - git diff without trailing comma
const hero = {
     firstName: "Florence",
-    lastName: "Nightingale"
+    lastName: "Nightingale",
+    inventorOf: ["coxcomb chart", "modern nursing"]
};

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

Semicolons

Semicolons

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

> *"Why? When JavaScript encounters a line break without a semicolon, it uses a set of rules called Automatic Semicolon Insertion to determine whether it should regard that line break as the end of a statement, and (as the name implies) place a semicolon into your code before the line break if it thinks so. ASI contains a few eccentric behaviors, though, and your code will break if JavaScript misinterprets your line break. These rules will become more complicated as new features become a part of JavaScript. Explicitly terminating your statements and configuring your linter to catch missing semicolons will help prevent you from encountering issues."*

```
// bad - raises exception
const luke = {}
const leia = {}
[luke, leia].forEach((jedi) => jedi.father = "vader")

// bad - raises exception
const reaction = "No! That's impossible!"
(async function meanwhileOnTheFalcon() {
  // handle `leia`, `lando`, `chewie`, `r2`, `c3p0`
  // ...
}())

// bad - returns `undefined` instead of the value on the next line - always happens when `return` is on a line b
function foo() {
  return
    "search your feelings, you know it to be foo"
}

// good
const luke = {};
const leia = {};
[luke, leia].forEach((jedi) => {
  jedi.father = "vader";
});

// good
const reaction = "No! That's impossible!";
(async function meanwhileOnTheFalcon() {
  // handle `leia`, `lando`, `chewie`, `r2`, `c3p0`
  // ...
}());

// good
function foo() {
  return "search your feelings, you know it to be foo";
}
```

Airbnb JS Style Guide

Whitespace
Commas
Semicolons

Full Stack Web Development