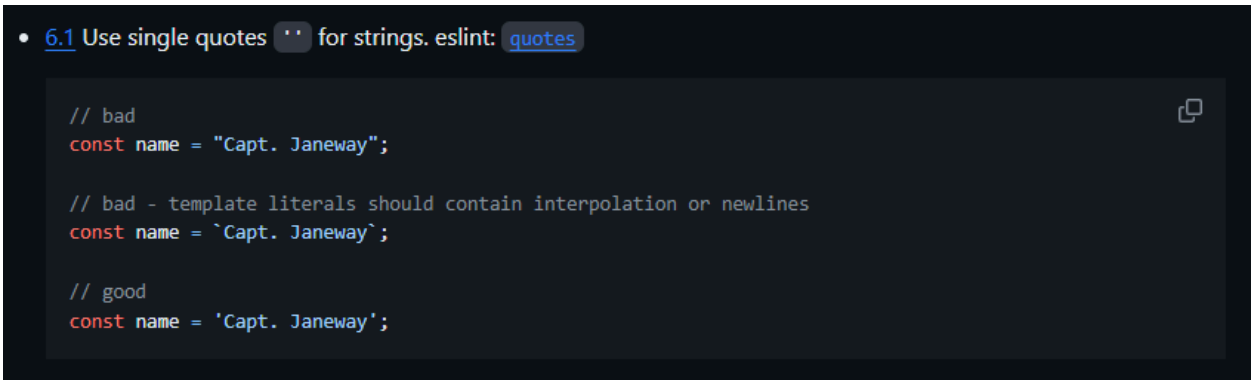
DWA_04.3 Knowledge Check_DWA4

---

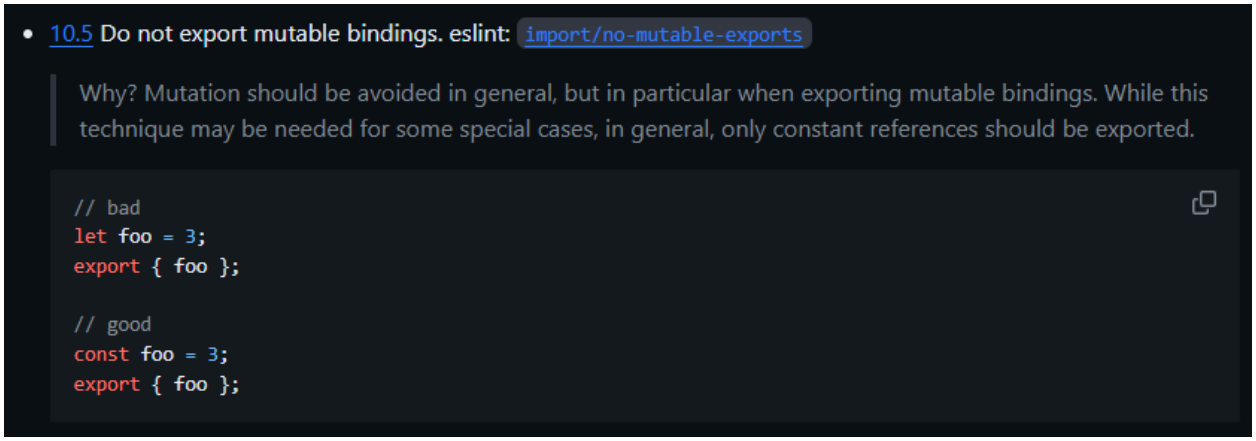
1. Select three rules from the Airbnb Style Guide that you find **useful** and explain why.



- 6.1 Use single quotes `''` for strings. eslint: `quotes`

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

// bad - template literals should contain interpolation or newlines
const name = `Capt. Janeway`;

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

I was never sure and it would never be consistent in my code, so this is very useful for me to sort of have it set out for me.



- 10.5 Do not export mutable bindings. eslint: `import/no-mutable-exports`

> Why? Mutation should be avoided in general, but in particular when exporting mutable bindings. While this technique may be needed for some special cases, in general, only constant references should be exported.

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

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

This definitely makes a lot of sense to me, I don't want to imagine the things that go wrong if you start importing/exporting mutable things

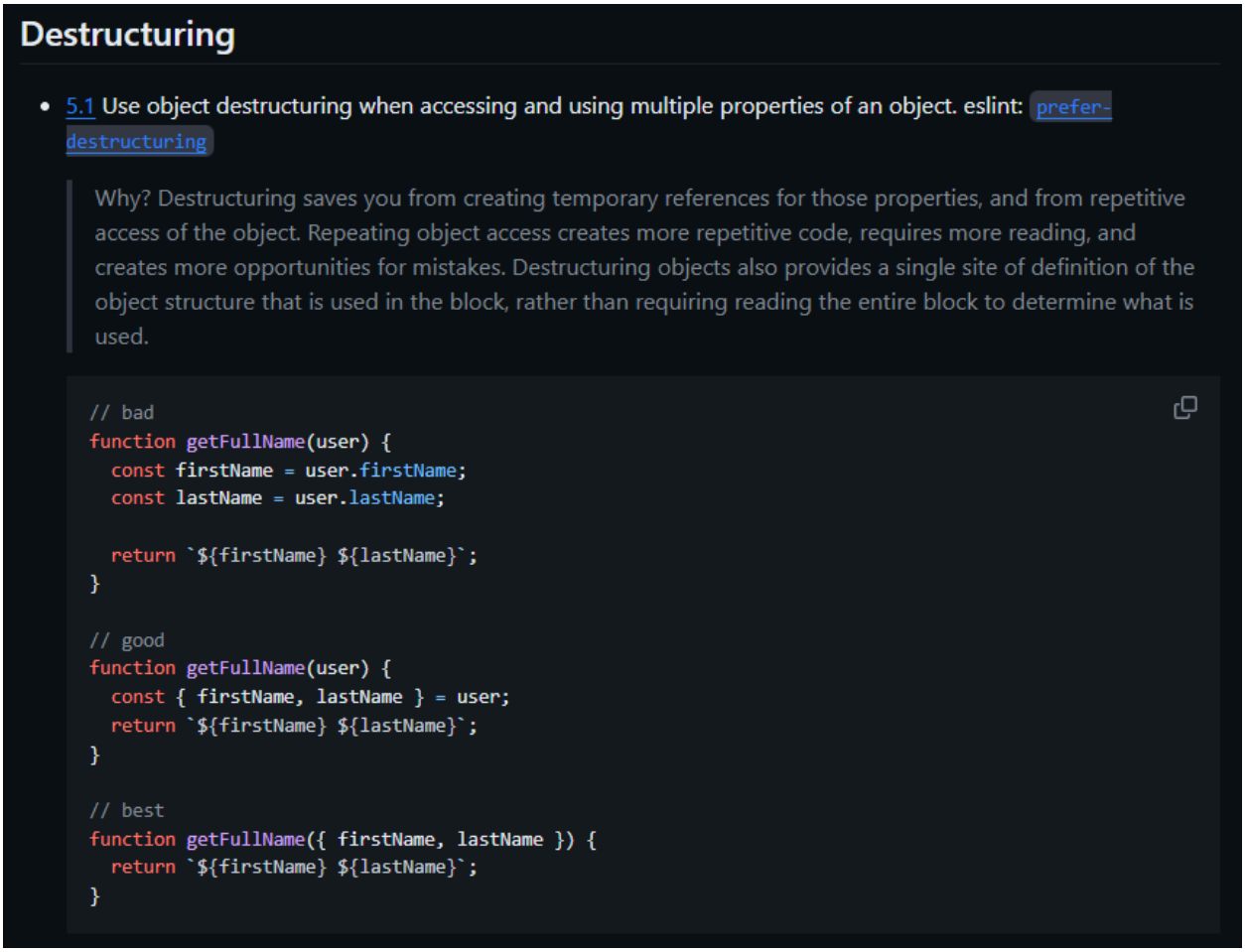
- 23.6 A base filename should exactly match the name of its default export.

```
// 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
// ^ supports both insideDirectory.js and insideDirectory/index.js
```

This is something I didn't know which also makes a lot of sense. It's like tools in a toolbox so when you're looking for a spanner you wanna grab the spanner.

---

2. Select three rules from the Airbnb Style Guide that you find **confusing** and explain why.



Destructuring

- 5.1 Use object destructuring when accessing and using multiple properties of an object. eslint: `prefer-destructuring`

> Why? Destructuring saves you from creating temporary references for those properties, and from repetitive access of the object. Repeating object access creates more repetitive code, requires more reading, and creates more opportunities for mistakes. Destructuring objects also provides a single site of definition of the object structure that is used in the block, rather than requiring reading the entire block to determine what is used.

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

I don’t quite understand their explanation.

- 10.2 Do not use wildcard imports.

> Why? This makes sure you have a single default export.

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

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

What do they mean?

- 16.2 If you're using multiline blocks with `if` and `else`, put `else` on the same line as your `if` block's closing brace. eslint: `brace-style`

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

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

Personally, the bad example looks more readable to me. Not sure why you would need to follow this rule.
