# DWA_04.3 Knowledge Check_DWA4

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

- Always use double quotes ( `"` ) for JSX attributes, but single quotes ( `'` ) for all other JS. eslint: `jsx-quotes`

> Why? Regular HTML attributes also typically use double quotes instead of single, so JSX attributes mirror this convention.

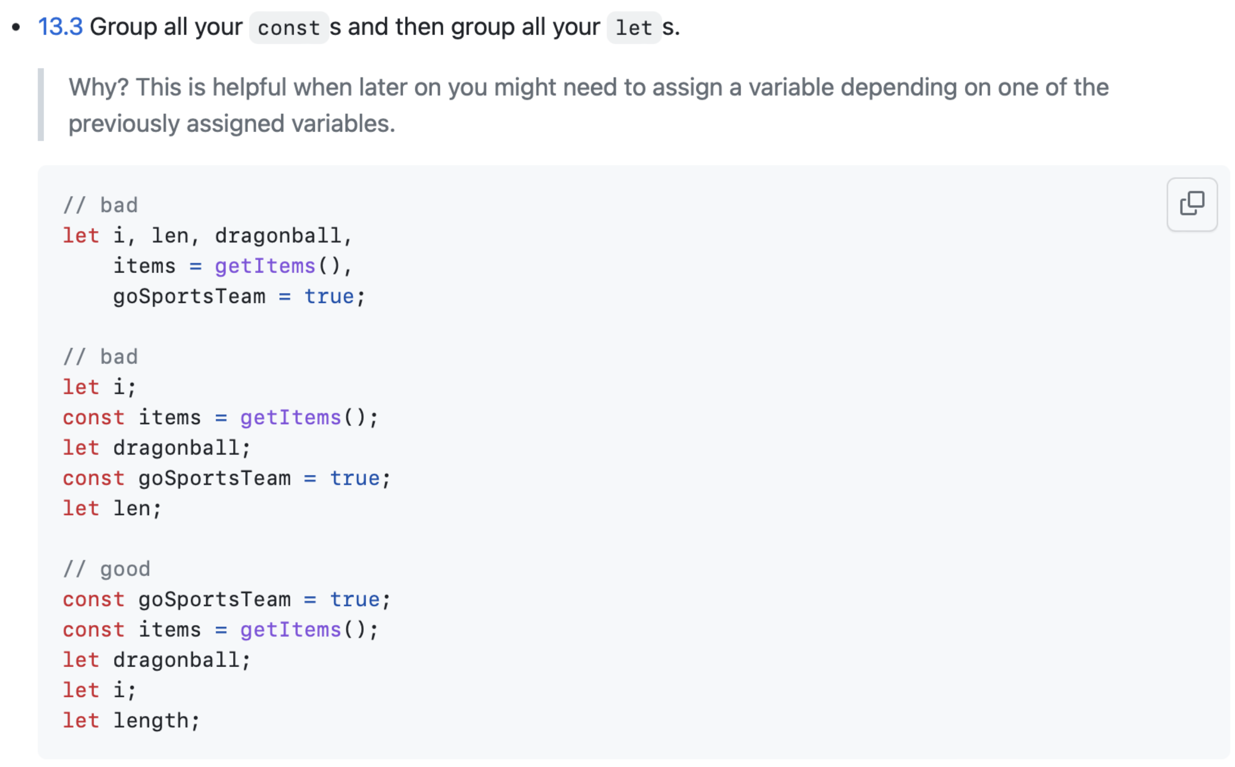
```
// bad
<Foo bar='bar' />

// good
<Foo bar="bar" />

// bad
<Foo style= />

// good
<Foo style= />
```

This is useful because it gives us consistency throughout our documents. Personally, I prefer single quotes, but with this, I have a clear outline of when to use which quotation character.

- Always self-close tags that have no children. eslint: `react/self-closing-comp`

```
// bad
<Foo variant="stuff"></Foo>

// good
<Foo variant="stuff" />
```

This is useful because it might feel like we forgot something if we have a closing tag without any children nested inside. By self-closing the tag we know that we are done with it and that we didn't leave anything out.



- 13.3 Group all your `const`s and then group all your `let`s.

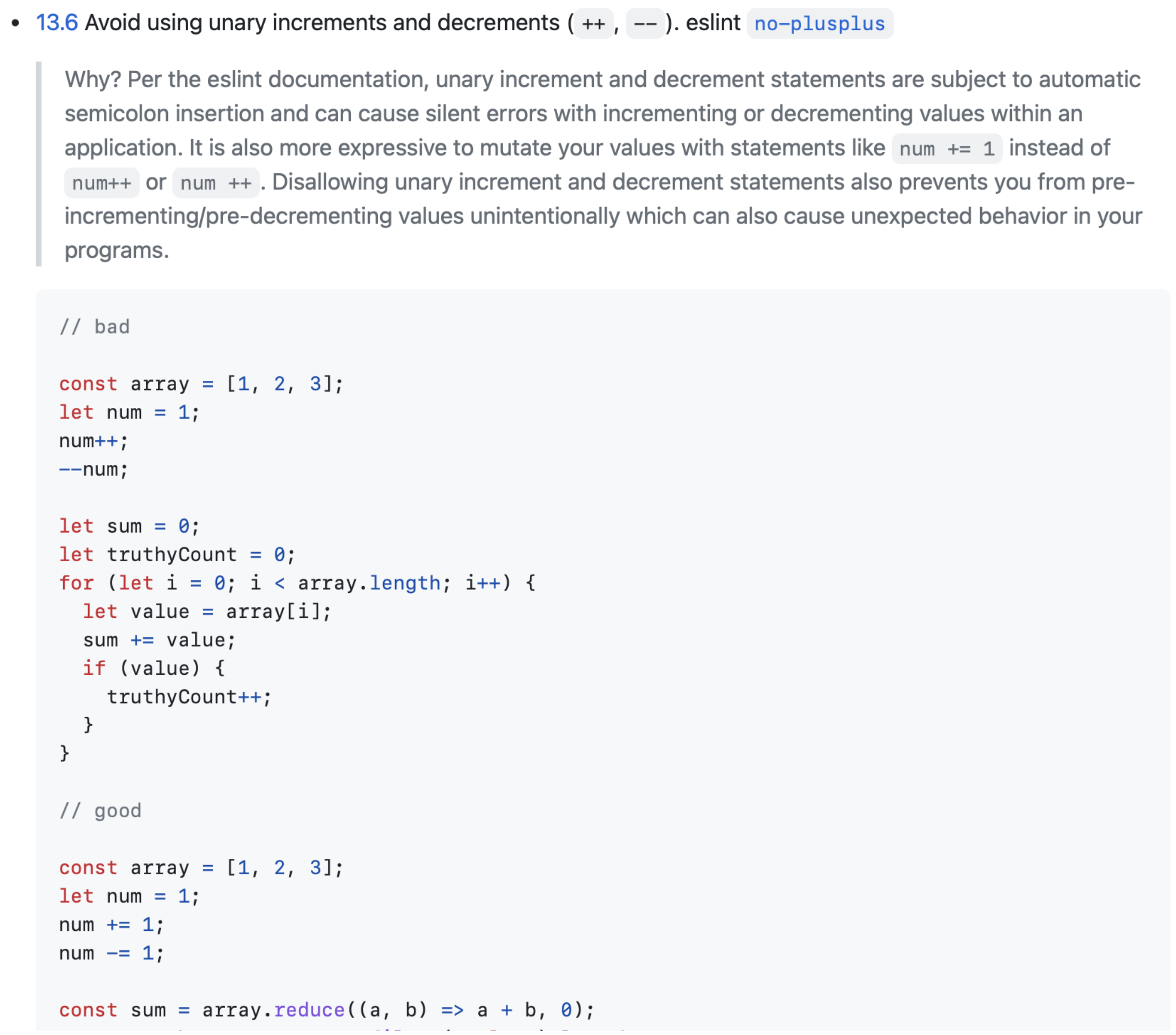
  > Why? This is helpful when later on you might need to assign a variable depending on one of the previously assigned variables.

  ```
  // 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;
  ```

This is useful because all variable types are grouped together. This makes it easier to find the variable for when we need to change something later along the lines

---

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



- 13.6 Avoid using unary increments and decrements (`++`, `--`). eslint `no-plusplus`

  > Why? Per the eslint documentation, unary increment and decrement statements are subject to automatic semicolon insertion and can cause silent errors with incrementing or decrementing values within an application. It is also more expressive to mutate your values with statements like `num += 1` instead of `num++` or `num ++`. Disallowing unary increment and decrement statements also prevents you from pre-incrementing/pre-decrementing values unintentionally which can also cause unexpected behavior in your programs.

```
// bad

const 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

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

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

What is a silent error? Is there any way to pick them up?

- 21.1 **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 `ret
function foo() {
  return
    'search your feelings, you know it to be foo'
}

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

If JS has automatic semicolon insertion why not use it? Should we be using semicolons?

- 15.3 Use shortcuts for booleans, but explicit comparisons for strings and numbers.

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

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

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

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

What is the reason for AirBnB to require us to do it in this way?