DWA_04.3 Knowledge Check_DWA4

---

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

First Rule: Use `const` for references; avoid `var`.

This rule suggests using `const` for variables that won't be reassigned, as opposed to `var`. This practice clarifies that the variable's value is constant, enhancing code readability and preventing accidental changes.

Second Rule: Always use braces for block statements in control structures.

Including braces in control structures like `if`, `else`, `for`, and `while` adds clarity and prevents bugs caused by unintended scoping issues. Even though JavaScript permits single-line statements without braces, it's a good practice to include them for better readability and maintenance.

Third Rule: Use template literals for string concatenation and formatting.

Template literals, enclosed in backticks (`), offer a more readable and organized way to concatenate strings and embed expressions. They support multi-line strings without the need for concatenation or escaping characters and allow for easy interpolation of variables directly into the string.

---

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

First Rule: Avoid using `++` or `--` operators.

The guide suggests avoiding the increment (`++`) and decrement (`--`) operators in favor of `+= 1` or `-= 1`. Although this aims to promote clarity and avoid side effects, experienced developers might find `++` and `--` concise and appropriate for simple numeric changes.

Second Rule: Disallow the use of `alert`, `confirm`, and `prompt`.

This rule advises against using `alert`, `confirm`, and `prompt` for user interactions, recommending more modern and customizable alternatives like modals. While these methods may be outdated and limited, they remain functional and simple for basic tasks, especially in small projects or quick prototypes.

Third Rule: Disallow the use of `Object.prototype` built-ins.

The rule advises against directly using methods from `Object.prototype` like `hasOwnProperty` and `isPrototypeOf`, recommending safer alternatives such as `Object.hasOwnProperty`. While this is intended to avoid conflicts if the prototype is modified, some developers may prefer the convenience of using `Object.prototype` methods directly in certain situations.
