DWA_04.3 Knowledge Check_DWA4

---

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

1. **Named function expressions instead of function declarations.**

    a. Encapsulate and Scope Control - Name function expressions allow you to encapsulate functions within a specific scope. This helps avoid polluting the global namespace with function names that might conflict with other variables or functions.

    b. Improved Debugging - having meaningful names for functions can greatly assist in identifying the source of errors or analyzing stack traces.

    c. Code Readability and Maintainability - Giving functions meaningful names using named function expressions improves the readability and maintainability of your code

2. **Object destructuring when accessing and using multiple properties of an object.**

    a. Concise Syntax - Instead of repeatedly accessing properties using dot notation or bracket notation, you can destructure the once and access the desired properties directly.

    b. Simplified Assignment - This simplifies the assignment process and reduces the need for repetitive code.

    c. Renaming Variables - This can be helpful when you want to give the variable more meaningful or descriptive names, or when you need to avoid naming conflicts with existing variables in code.

3. **Using '/**..*/' for multiline comments.**

    a. Documentation - These comments can contain detailed information about purpose, usage, parameters, return values, and any other relevant details of a documented entity.

    b. Commenting Out Blocks of Code - Use for multiline comments allows you to quickly and easily comment out large chunks of code without needing to add '//' in front of each line.

c. Explanatory Comments - They can help describe complex algorithms, provide context for certain code blocks , or explain the reasoning behind specific implementing choices.

---

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

1. **Perform type coercion at the beginning of the statement.**

    a. Implicit coercion - if the coercion is not explicitly visible in the code, it becomes harder to understand what transformations are happening behind the scenes.

    b. Loss of precision - if this coercion happens at the beginning of a statement without explicit documentation or explanation, it may not be apparent to other developers or even the original coder.

    c. Maintenance and readability - It might require additional mental effort to understand how different types are being manipulated, especially if multiple coercions are chained together.

2. **Prefer higher-order functions instead of loops like for-in or for-of.**
    a. Paradigm shift - Developers who are accustomed to imperative programming may find it challenging to adopt this new way of thinking and writing code

    b. Complexity - Understanding how these concepts fit together and interact with each other can be confusing, especially for developers who are new to functional programming.

    c. Debugging - With higher-order functions, the control flow may be distributed across different functions, making it harder to pinpoint the exact location of errors.

3. **Using standard modules (import/ export) in programming language**

    a. Syntax and Compatibility - If developers are accustomed to a different module system or have experience with an older version of language, they may initially find the new module system syntax and rules confusing.

b. Learning Curve - Concepts like importing and exporting modules, understanding how module resolution works, and managing dependencies through package managers can be unfamiliar and take time to grasp fully.

c. Interoperability - Understanding how to handle imports and exports between different module systems and ensuring interoperability can be challenging.
