DWA_04.3 Knowledge Check_DWA4

---

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

Destructuring: is a technique used to extract values from arrays or properties from objects and assign them to new, distinct variables, making the code more readable. It employs a shorthand syntax that simplifies working with complex data structures

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

Comments: are there for documentation, explanation or if you just want to comment out code without deleting it. You get 2 different comments, single-line & multi-line comments.

Use /** ... */ for multiline comments.

```
// bad
// make() returns a new element
// based on the passed in tag name
//
// @param {String} tag
// @return {Element} element
function make(tag) {
```

```
  // ...

  return element;
}

// good
/**
 * make() returns a new element
 * based on the passed-in tag name
 */
function make(tag) {

  // ...

  return element;
}
```

Use // for single line comments. Place single line comments on a newline above the subject of the comment. Put an empty line before the comment unless it’s on the first line of a block

```
// bad
const active = true;  // is current tab

// good
// is current tab
const active = true;

// bad
function getType() {
  console.log('fetching type...');
  // set the default type to 'no type'
  const type = this.type || 'no type';

  return type;
}

// good
function getType() {
  console.log('fetching type...');
```

```
  // set the default type to 'no type'
  const type = this.type || 'no type';

  return type;
}

// also good
function getType() {
  // set the default type to 'no type'
  const type = this.type || 'no type';

  return type;
}
```

Naming conventions: refer to a set of rules or guidelines for naming variables, functions, classes, and other identifiers within your code. Consistently following naming conventions helps improve code readability, maintainability, and collaboration among developers.

- Snake_case
- Kebab_case
- camelCase
- PascalCase
- SCREAMING_SNAKE_CASE

```
// bad
const OBJEcttsssss = {};
const this_is_my_object = {};
function c() {}

// good
const thisIsMyObject = {};
function thisIsMyFunction() {}
```

---

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

Classes & Constructors: 9.3 Methods can return this to help with method chaining. Using “this” makes me confused most of the time. Not always sure how or when to use it.

```
// bad
Jedi.prototype.jump = function () {
  this.jumping = true;
  return true;
};

Jedi.prototype.setHeight = function (height) {
  this.height = height;
};

const luke = new Jedi();
luke.jump(); // => true
luke.setHeight(20); // => undefined

// good
class Jedi {
  jump() {
    this.jumping = true;
    return this;
  }

  setHeight(height) {
    this.height = height;
    return this;
  }
}

const luke = new Jedi();

luke.jump()
  .setHeight(20);
```

Iterators and Generators: why is it bad to use for…of or for…in loops? Because it's more difficult to read and understand especially when it comes to other developers.

```
const numbers = [1, 2, 3, 4, 5];

// bad
let sum = 0;
for (let num of numbers) {
  sum += num;
}
sum === 15;

// good
let sum = 0;
numbers.forEach((num) => {
  sum += num;
});
sum === 15;

// best (use the functional force)
const sum = numbers.reduce((total, num) => total + num, 0);
sum === 15;

// bad
const increasedByOne = [];
for (let i = 0; i < numbers.length; i++) {
  increasedByOne.push(numbers[i] + 1);
}

// good
const increasedByOne = [];
numbers.forEach((num) => {
  increasedByOne.push(num + 1);
});

// best (keeping it functional)
const increasedByOne = numbers.map((num) => num + 1);
```

Blocks: 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
I find it easier to read the wrong version than the right version.

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

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