DWA_04.3 Knowledge Check_DWA4

---

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

Objects 3.5
Group your shorthand properties at the beginning of you object declaration.

- ***Why it is easier to tell which properties are using shorthand:***

```
const anakinSkywalker = 'Anakin Skywalker';
const lukeSkywalker = 'Luke Skywalker';

// bad
const obj = {
  episodeOne: 1,
  twoJediWalkIntoACantina: 2,
  lukeSkywalker,
  episodeThree: 3,
  mayTheFourth: 4,
  anakinSkywalker,
};

// good
const obj = {
  lukeSkywalker,
  anakinSkywalker,
  episodeOne: 1,
  twoJediWalkIntoACantina: 2,
  episodeThree: 3,
  mayTheFourth: 4,
};
```

I find the above rule useful because using it will ensure code readability.

2. ***Functions 7.5***

- ***Never name a parameter ''arguments''. This will take precedence over the arguments object that is given to every function scope***.

```
// bad
function foo(name, options, arguments) {
  // ...
}

// good
function foo(name, options, args) {
  // ...
}
```

Explanation:

In MDN the arguments object is an array-like object accessible inside **functions,** that contains the values of the arguments passed to that function.
I find this very helpful to know. This will help avoid unexpected behavior that would occur when the word ‘arguments’ is used as a parameter to a function. For example, you have function with a parameter (arguments) and later you want to use the arguments object to check for an argument that was passed to the function, but it doesn’t work, because the arguments parameter took precedence over the arguments object.

3. **Comments 18.4**
Prefixing your comments with FIXME or TODO helps other developers quickly understand if you’re pointing out a problem that needs to be revisited, or if you’re suggesting a solution to the problem that needs to be implemented. These are different than regular comments because they are actionable. The actions are FIXME: -- need to figure this out or TODO: -- need to implement.

FIXME comment example:

```
class Calculator extends Abacus {
  constructor() {
    super();

    // FIXME: shouldn’t use a global here
    total = 0;
  }
}
```

TODO comment example:

```
class Calculator extends Abacus {
  constructor() {
    super();

    // TODO: total should be configurable by an options param
    this.total = 0;
  }
}
```

I think such comments are helpful in that they help you or another developer when going through the code know what needs to be fixed or done. This helps in code maintainability.

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

1. ***Modules 10.10***
   Do not include JavaScript file name extensions:
- ***Why? Including extensions inhibits refactoring, and inappropriately hardcodes implementation details of the module you're importing in every consumer.***

```
// bad
import foo from './foo.js';
import bar from './bar.jsx';
import baz from './baz/index.jsx';

// good
import foo from './foo';
import bar from './bar';
import baz from './baz';
```

What mainly confuses me about this style guide rule is when I was importing exports from other modules, the import wouldn't work if I didn't include .js extension.

2. ***Don't use iterators. Prefer JavaScript's higher-order functions instead of loops like for-in or for-of.***
- ***Why? This enforces our immutable rule. Dealing with pure functions that return values is easier to reason about than side effects.***
- ***Use map() / every() / filter() / find() / findIndex() / reduce() / some() / ... to iterate over arrays, and Object.keys() / Object.values() / Object.entries() to produce arrays so you can iterate over objects.***

```
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);
```

So, there is no need to use for-of and for-in to iterate over arrays and objects?

3. ***Prefer the use of the spread syntax (…) to call variadic functions.***
- ***Why? It's cleaner, you don't need to supply a context, and you cannot easily compose new with apply.***

```
// bad
const x = [1, 2, 3, 4, 5];
console.log.apply(console, x);

// good
const x = [1, 2, 3, 4, 5];
console.log(...x);

// bad
new (Function.prototype.bind.apply(Date, [null, 2016, 8, 5]));

// good
new Date(...[2016, 8, 5]);
```

I did some research and found that variadic functions are functions that can accept any number of arguments. But I don't understand how using the spread syntax to call these functions is better.