Keotshepile Maje

DWA_04.3 Knowledge Check_DWA4

---

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

- 4.3 Use array spreads `...` to copy arrays.

```
// bad
const len = items.length;
const itemsCopy = [];
let i;

for (i = 0; i < len; i += 1) {
  itemsCopy[i] = items[i];
}

// good
const itemsCopy = [...items];
```

This rule makes the code shorter and much easier to understand and read. And instead of using a loop you just use a single line which makes sense.

- 8.5 Avoid confusing arrow function syntax (`=>`) with comparison operators (`<=` , `>=`). eslint: `no-confusing-arrow`

```
// bad
const itemHeight = (item) => item.height <= 256 ? item.largeSize : item.smallSi

// bad
const itemHeight = (item) => item.height >= 256 ? item.largeSize : item.smallSi

// good
const itemHeight = (item) => (item.height <= 256 ? item.largeSize : item.smallS

// good
const itemHeight = (item) => {
  const { height, largeSize, smallSize } = item;
  return height <= 256 ? largeSize : smallSize;
};
```

Putting your code in brackets() when using ternary and a single line function make sense as it improves readability and makes it easier to distinguish the arrow from the function and the arrow from the comparison in the ternary code.

- 12.1 Use dot notation when accessing properties. eslint: `dot-notation`

```
const luke = {
  jedi: true,
  age: 28,
};

// bad
const isJedi = luke['jedi'];

// good
const isJedi = luke.jedi;
```

The dot makes the code neat and easier to read especially if you have to access the properties that have more than one nesting

---

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

- 6.1 Use single quotes `''` for strings. eslint: `quotes`

```
// bad
const name = "Capt. Janeway";

// bad - template literals should contain interpolation or newlines
const name = `Capt. Janeway`;

// good
const name = 'Capt. Janeway';
```

For me, I don't think it should be a strict rule to use only single quotes for strings as long as you are consistent with the type of quotes you decide to use.

- 4.4 To convert an iterable object to an array, use spreads `...` instead of `Array.from`

```
const foo = document.querySelectorAll('.foo');

// good
const nodes = Array.from(foo);

// best
const nodes = [...foo];
```

- 4.6 Use `Array.from` instead of spread `...` for mapping over iterables, because it avoids creating an intermediate array.

```
// bad
const baz = [...foo].map(bar);

// good
const baz = Array.from(foo, bar);
```

From the two above screenshots of the rules. The first picture shows that it's the best practice to use the spread … method instead of the Array.from method to convert an iterable object to an array. But the second rule that follows discourages us from using the spread method and using the Array.from method. For me look more readable to use the spread method.

- 11.1 Don't use iterators. Prefer JavaScript's higher-order functions instead of loops like `for-in` or `for-of`. eslint: `no-iterator` `no-restricted-syntax`

```
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;
```

11.1
I find it more easier to interpret the 'for of loop' which is classified as bad than it is for the method said to be best. The 'for of loop' is much more simple and readable.
