DWA_04.3 Knowledge Check_DWA4

---

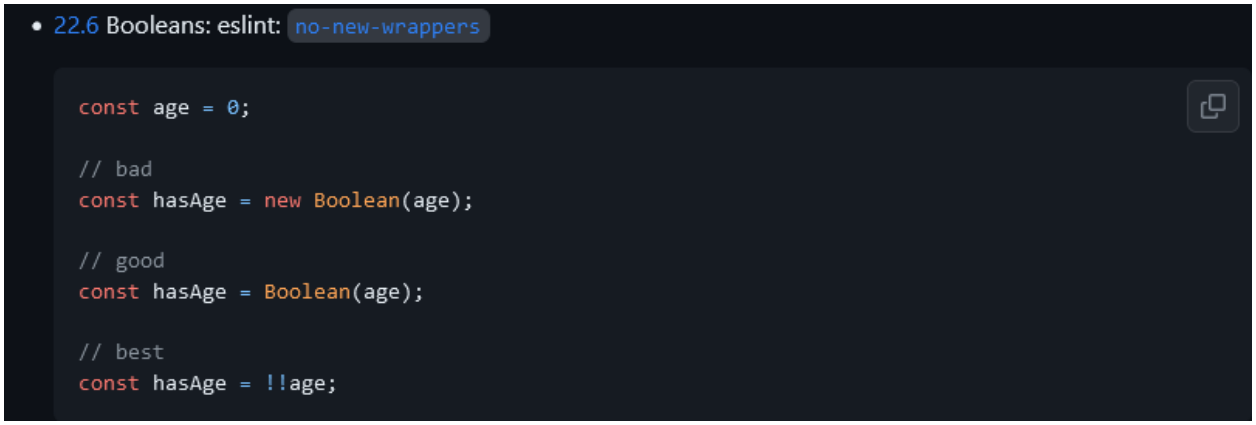
1. Select three rules from the Airbnb Style Guide that you find **useful** and explain why.

Because there are so many ways to convert to a number this limits my choices in a good way.

Why? The `parseInt` function produces an integer value dictated by interpretation of the contents of the string argument according to the specified radix. Leading whitespace in string is ignored. If radix is `undefined` or `0`, it is assumed to be `10` except when the number begins with the character pairs `0x` or `0X`, in which case a radix of 16 is assumed. This differs from ECMAScript 3, which merely discouraged (but allowed) octal interpretation. Many implementations have not adopted this behavior as of 2013. And, because older browsers must be supported, always specify a radix.

```
const inputValue = '4';

// bad
const val = new Number(inputValue);

// bad
const val = +inputValue;

// bad
const val = inputValue >> 0;

// bad
const val = parseInt(inputValue);

// good
const val = Number(inputValue);

// good
const val = parseInt(inputValue, 10);
```

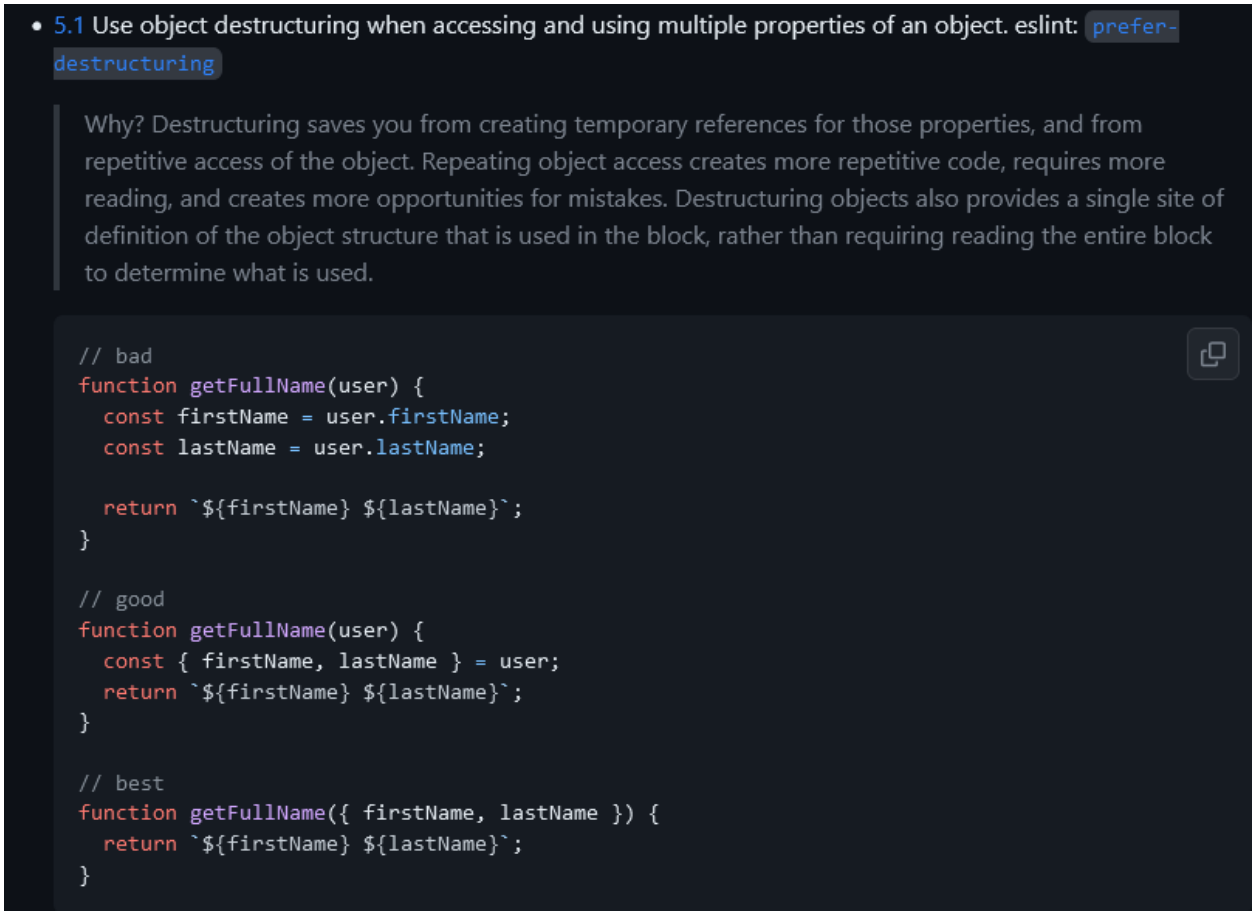
Another case where I didn't know which one to follow and now I know the best ways.



- 22.6 Booleans: eslint: `no-new-wrappers`

```
const age = 0;

// bad
const hasAge = new Boolean(age);

// good
const hasAge = Boolean(age);

// best
const hasAge = !!age;
```

Same as before to focus on one way of doing things.



- 5.1 Use object destructuring when accessing and using multiple properties of an object. eslint: `prefer-destructuring`

> Why? Destructuring saves you from creating temporary references for those properties, and from repetitive access of the object. Repeating object access creates more repetitive code, requires more reading, and creates more opportunities for mistakes. Destructuring objects also provides a single site of definition of the object structure that is used in the block, rather than requiring reading the entire block to determine what is used.

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

---

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

Not sure why this would be better than the above-spread syntax

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);
```

Put this here to ask about how often would we use .assign()

3.8 Prefer the object spread syntax over `Object.assign` to shallow-copy objects. Use the object rest parameter syntax to get a new object with certain properties omitted. eslint: `prefer-object-spread`

```
// very bad
const original = { a: 1, b: 2 };
const copy = Object.assign(original, { c: 3 }); // this mutates `original` ಠ_ಠ
delete copy.a; // so does this

// bad
const original = { a: 1, b: 2 };
const copy = Object.assign({}, original, { c: 3 }); // copy => { a: 1, b: 2, c: 3 }

// good
const original = { a: 1, b: 2 };
const copy = { ...original, c: 3 }; // copy => { a: 1, b: 2, c: 3 }

const { a, ...noA } = copy; // noA => { b: 2, c: 3 }
```

Dont know when I would need to use this and if I would ever need to use it.

22.5 **Note:** Be careful when using bitshift operations. Numbers are represented as 64-bit values, but bitshift operations always return a 32-bit integer (source). Bitshift can lead to unexpected behavior for integer values larger than 32 bits. Discussion. Largest signed 32-bit Int is 2,147,483,647:

```
2147483647 >> 0; // => 2147483647
2147483648 >> 0; // => -2147483648
2147483649 >> 0; // => -2147483647
```
