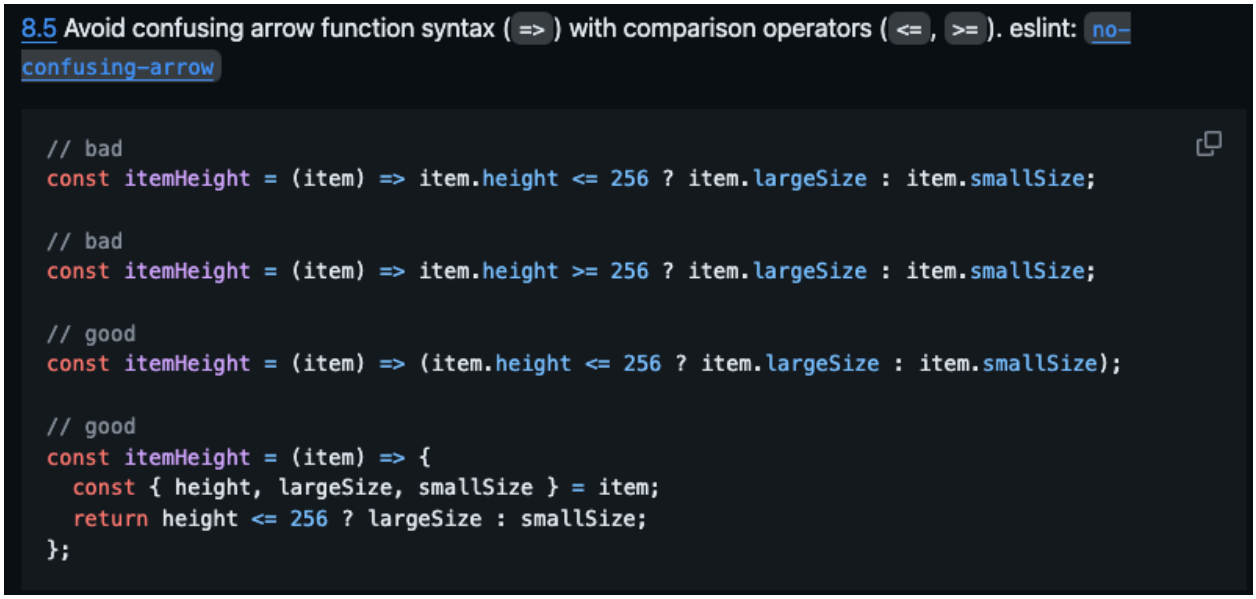
DWA_04.3 Knowledge Check_DWA4

---

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



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

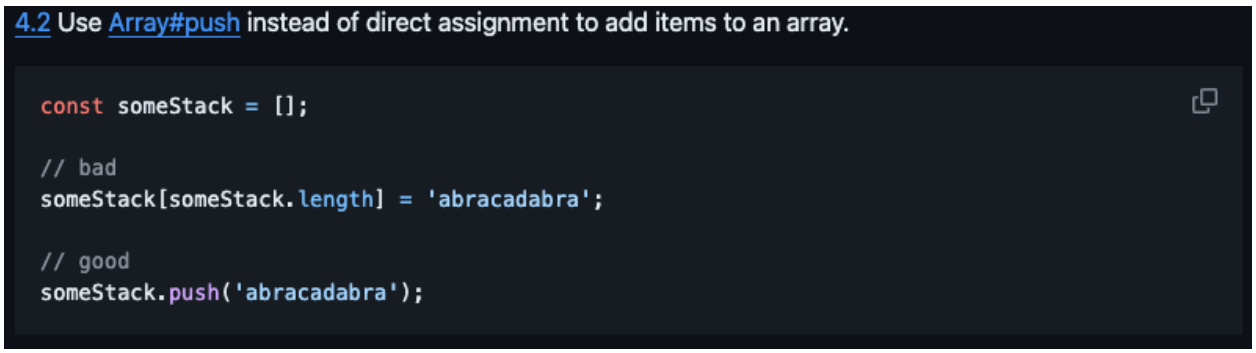
```
// bad
const itemHeight = (item) => item.height <= 256 ? item.largeSize : item.smallSize;

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

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

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

It is easy to see in the example on how this would be confusing or easy to overlook. Arrow functions and <= look so similar that a mistake would be easy to make



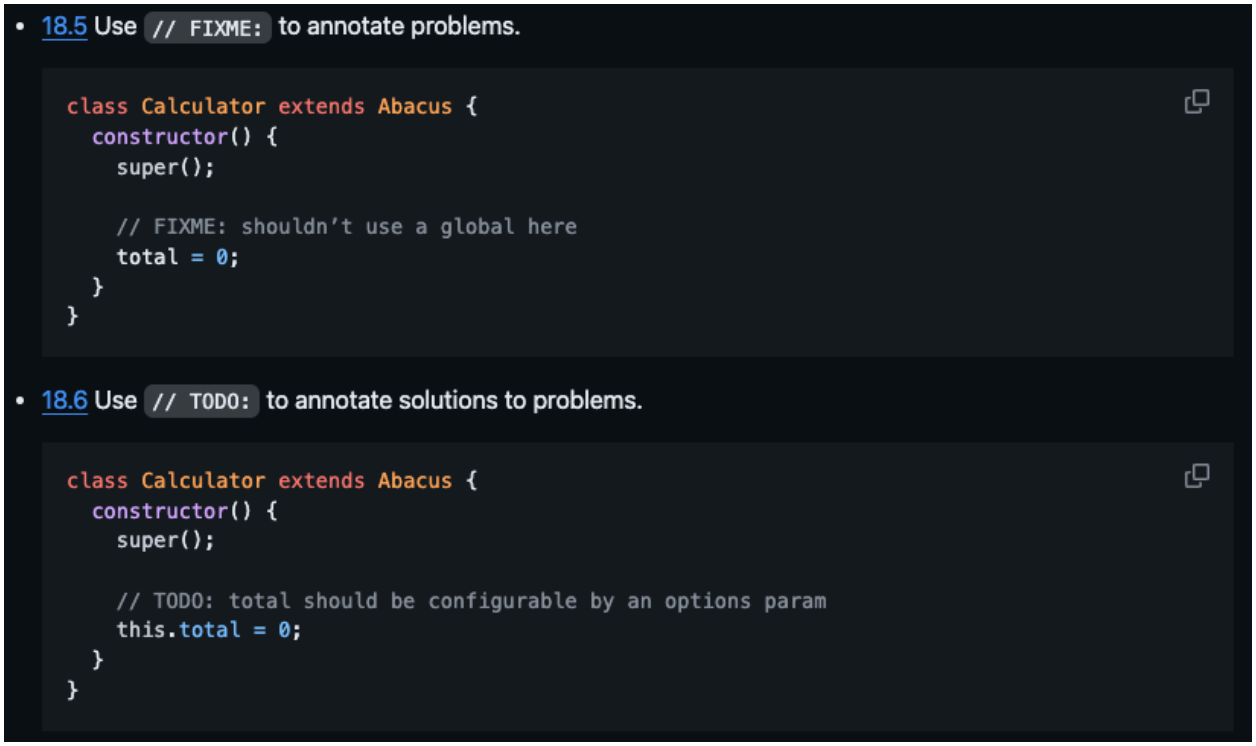
4.2 Use Array#push instead of direct assignment to add items to an array.

```
const someStack = [];

// bad
someStack[someStack.length] = 'abracadabra';

// good
someStack.push('abracadabra');
```

Directly assigning things to an array could lead to something being overwritten.



Using FIXME and TODO became invaluable when I was doing my capstone and trying to manage my time. Encountering a bug or problem while working on something else. It is important that you don't forget to come back to it

---

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

The rules I find confusing seem to be relating to concepts I am unfamiliar with. When it comes to things I have used in JS I understand it. But I will include a few I didn't understand here.

- 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];
```

Why don't we just say itemsCopy = items

5.2 Use array destructuring. eslint: `prefer-destructuring`

```
const arr = [1, 2, 3, 4];

// bad
const first = arr[0];
const second = arr[1];

// good
const [first, second] = arr;
```

I will need to go and see what this outputs, I don't understand how this array destructuring works

22.3 Numbers: Use `Number` for type casting and `parseInt` always with a radix for parsing strings. eslint: `radix` `no-new-wrappers`

> 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);
```

What should I use?
