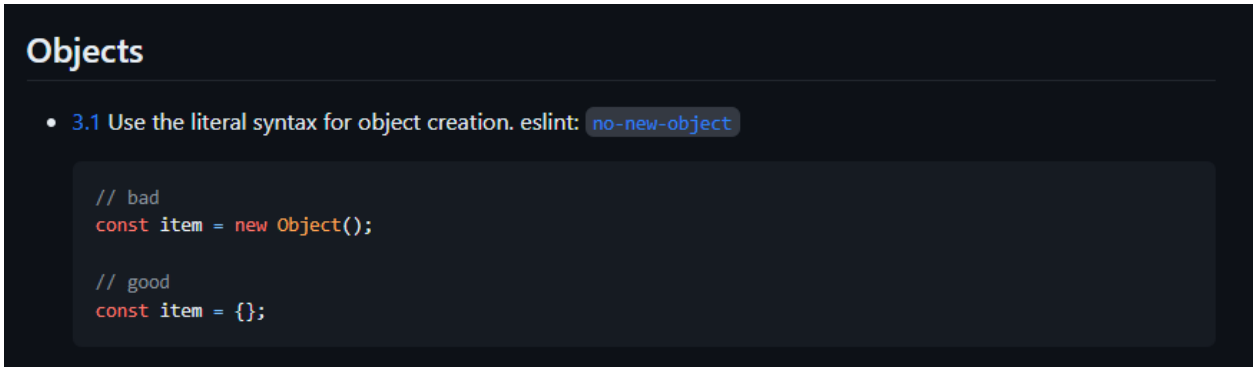
DWA_04.3 Knowledge Check_DWA4

---

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



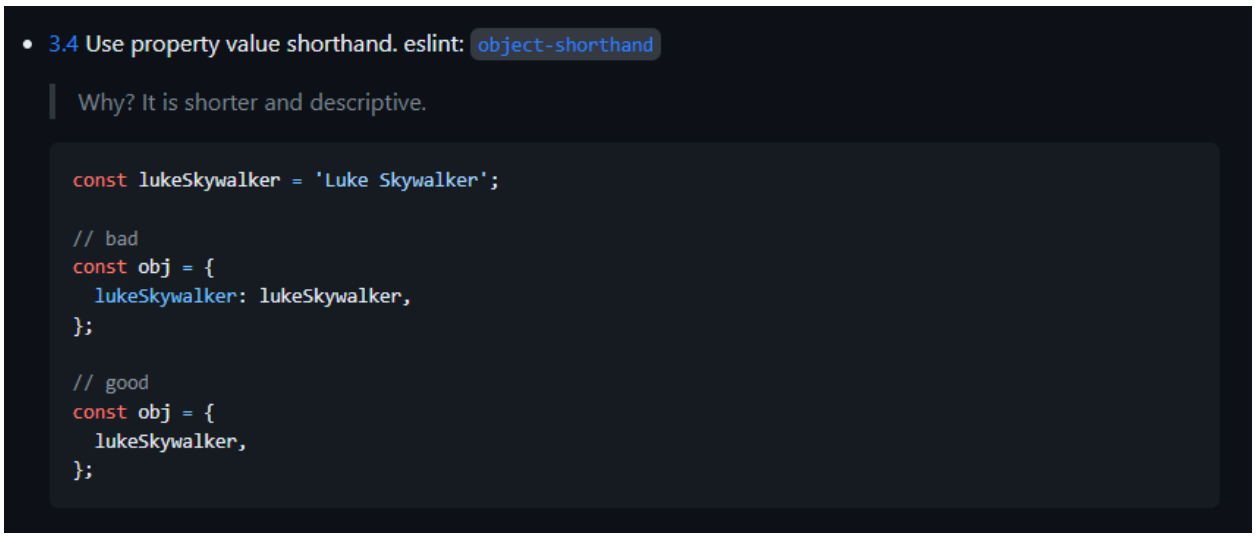
Objects

- 3.1 Use the literal syntax for object creation. eslint: `no-new-object`

```
// bad
const item = new Object();

// good
const item = {};
```

WHY: This is a simpler way to do it that takes less code to write.



- 3.4 Use property value shorthand. eslint: `object-shorthand`

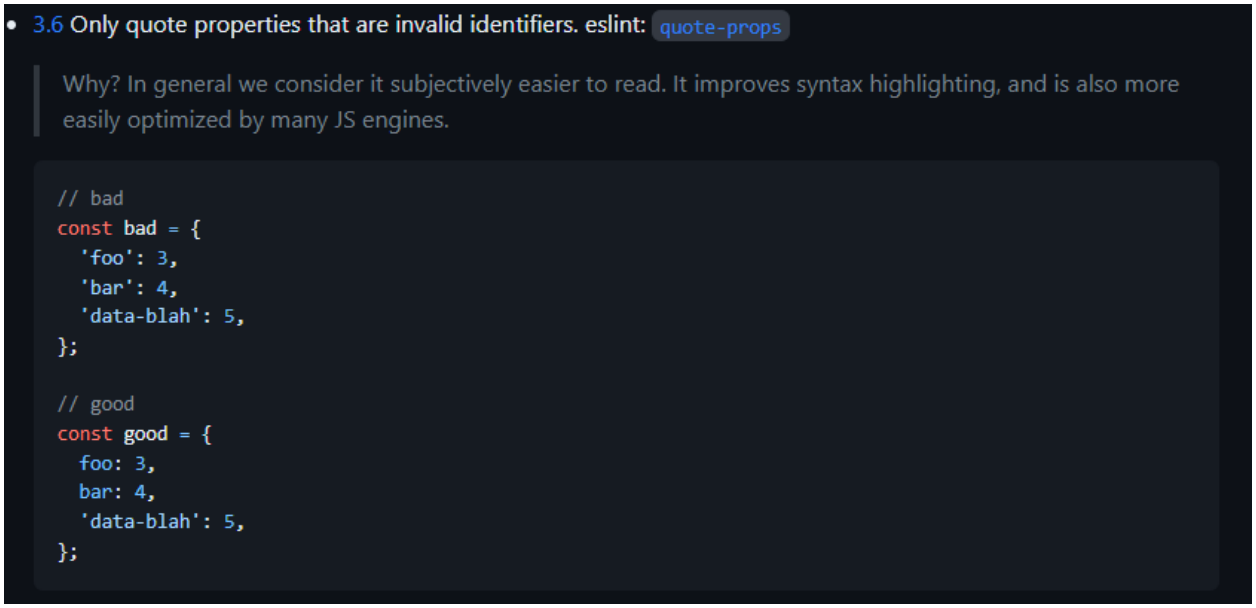
> Why? It is shorter and descriptive.

```
const lukeSkywalker = 'Luke Skywalker';

// bad
const obj = {
  lukeSkywalker: lukeSkywalker,
};

// good
const obj = {
  lukeSkywalker,
};
```

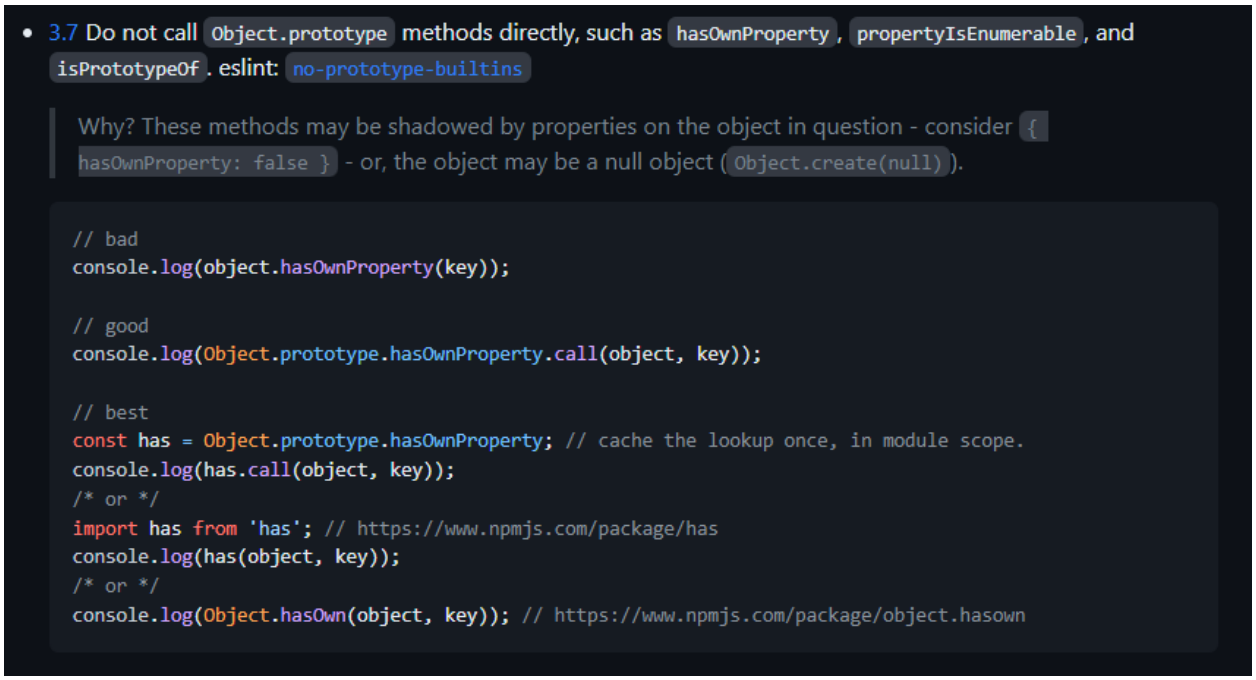
WHY: Simplifies object initialisation. But it only works for objects when the key and value pairs are the same and the properties are stored in a variable.



WHY: I agree with saying it is easier to read. It looks “neater” without the quotes that are distracting. Additionally, you can convey the unusual case at a glance without extra effort.

---

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



- 3.7 Do not call `Object.prototype` methods directly, such as `hasOwnProperty`, `propertyIsEnumerable`, and `isPrototypeOf`. eslint: `no-prototype-builtins`

  > Why? These methods may be shadowed by properties on the object in question - consider `{ hasOwnProperty: false }` - or, the object may be a null object (`Object.create(null)`).

```
// bad
console.log(object.hasOwnProperty(key));

// good
console.log(Object.prototype.hasOwnProperty.call(object, key));

// best
const has = Object.prototype.hasOwnProperty; // cache the lookup once, in module scope.
console.log(has.call(object, key));
/* or */
import has from 'has'; // https://www.npmjs.com/package/has
console.log(has(object, key));
/* or */
console.log(Object.hasOwn(object, key)); // https://www.npmjs.com/package/object.hasown
```

WHY: What is happening here? It overcomplicates what appears to be an obvious process. The bad version checks to see if an ‘object’ has a ‘key’ property.

- 5.3 Use object destructuring for multiple return values, not array destructuring.

  > Why? You can add new properties over time or change the order of things without breaking call sites.

```
// bad
function processInput(input) {
  // then a miracle occurs
  return [left, right, top, bottom];
}

// the caller needs to think about the order of return data
const [left, __, top] = processInput(input);

// good
function processInput(input) {
  // then a miracle occurs
  return { left, right, top, bottom };
}

// the caller selects only the data they need
const { left, top } = processInput(input);
```

WHY: In which situation would this be used? I don't understand the use of the dash as well.

- 7.1 Use named function expressions instead of function declarations. eslint: `func-style`, `func-names`

  > Why? Function declarations are hoisted, which means that it's easy - too easy - to reference the function before it is defined in the file. This harms readability and maintainability. If you find that a function's definition is large or complex enough that it is interfering with understanding the rest of the file, then perhaps it's time to extract it to its own module! Don't forget to explicitly name the expression, regardless of whether or not the name is inferred from the containing variable (which is often the case in modern browsers or when using compilers such as Babel). This eliminates any assumptions made about the Error's call stack. (Discussion)

```
// bad
function foo() {
  // ...
}

// bad
const foo = function () {
  // ...
};

// good
// lexical name distinguished from the variable-referenced invocation(s)
const short = function longUniqueMoreDescriptiveLexicalFoo() {
  // ...
};
```

WHY: Isn't it better to use arrow functions instead?
