DWA_04.3 Knowledge Check_DWA4

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

Naming Conventions:

```
// bad
const OBJEcttsssss = {};
const this_is_my_object = {};
function c() {}

// good
const thisIsMyObject = {};
function thisIsMyFunction() {}
```

- Clear and consistent naming conventions make code more readable and understandable. When variable and function names accurately reflect their purpose or functionality, developers can quickly grasp what the code does without having to decipher cryptic or ambiguous names.

Destructuring:

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

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

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

- 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.

Classes & Constructors:

```
// bad
const inherits = require('inherits');
function PeekableQueue(contents) {
  Queue.apply(this, contents);
}
inherits(PeekableQueue, Queue);
PeekableQueue.prototype.peek = function () {
  return this.queue[0];
};

// good
class PeekableQueue extends Queue {
  peek() {
    return this.queue[0];
  }
}
```

- Classes provide a designated constructor method, constructor(), which is used to initialize object instances. This separation of initialization logic from the object's prototype helps in maintaining a clear distinction between object creation and object behavior.
- Classes provide a clear structure that makes the code more readable and understandable. When you use a class, it's easier to see the relationships between properties and methods, leading to code that's easier to follow and maintain.

---

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

Hoisting:

```
function example() {
  console.log(anonymous); // => undefined

  anonymous(); // => TypeError anonymous is not a function

  var anonymous = function () {
    console.log('anonymous function expression');
  };
}
```

- Code that heavily relies on hoisting can be difficult to read and understand. When variable declarations or function definitions are scattered throughout a scope, it becomes challenging to follow the logical flow of the code. This can make maintenance, debugging, and collaboration more complicated.

Using unary increments and decrements (++, --).:

```
// bad

const array = [1, 2, 3];
let num = 1;
num++;
--num;

let sum = 0;
let truthyCount = 0;
for (let i = 0; i < array.length; i++) {
  let value = array[i];
  sum += value;
  if (value) {
    truthyCount++;
  }
}

// good

const array = [1, 2, 3];
let num = 1;
num += 1;
num -= 1;

const sum = array.reduce((a, b) => a + b, 0);
const truthyCount = array.filter(Boolean).length;
```

- Unary increment and decrement statements are subject to automatic semicolon insertion and can cause silent errors with incrementing or decrementing values within an application. It is also more expressive to mutate your values with statements like num += 1 instead of num++ or num ++. Disallowing unary increment and decrement statements also prevents you from pre-incrementing/pre-decrementing values unintentionally which can also cause unexpected behavior in your programs

Iterators & Generators:

```
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;
```

- Familiarity with Loops: As a new developer we are more familiar with traditional for loops because they are a common looping construct used in many programming languages. As a result, we find the for loop easier to understand and more intuitive initially.