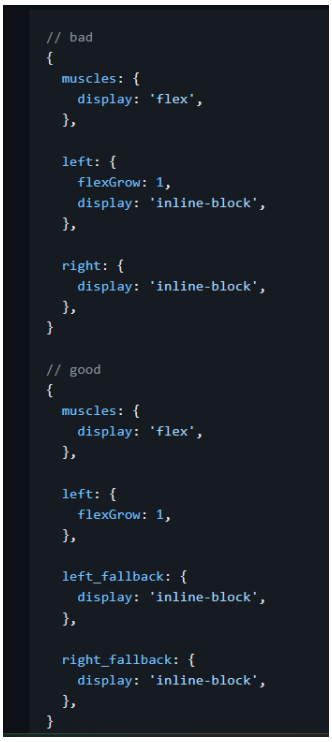
DWA_04.3 Knowledge Check_DWA4

---

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

Rule 1:

Use “selectorName_fallback” for sets of fallback styles. This rule makes it easier to understand the relation between the css classes, and gives us a clear understanding of what fallback for a class belongs to another.



```
// bad
{
  muscles: {
    display: 'flex',
  },

  left: {
    flexGrow: 1,
    display: 'inline-block',
  },

  right: {
    display: 'inline-block',
  },
}

// good
{
  muscles: {
    display: 'flex',
  },

  left: {
    flexGrow: 1,
  },

  left_fallback: {
    display: 'inline-block',
  },

  right_fallback: {
    display: 'inline-block',
  },
}
```

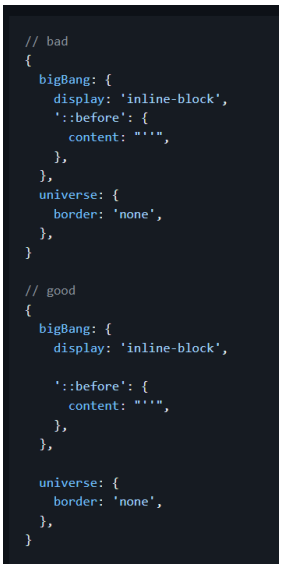
Rule 2:

Use device-agnostic names (e.g. "small", "medium", and "large") to name media query breakpoints. This rule makes it easier for the next person to understand the different screen sizes the media queries belong to.

```
// bad
const breakpoints = {
  mobile: '@media (max-width: 639px)',
  tablet: '@media (max-width: 1047px)',
  desktop: '@media (min-width: 1048px)',
};

// good
const breakpoints = {
  small: '@media (max-width: 639px)',
  medium: '@media (max-width: 1047px)',
  large: '@media (min-width: 1048px)',
};
```

Rule 3:

Leave a blank line between adjacent blocks at the same indentation level. This rule allows for better readability and allows us to also distinguish the differences between two sections of code.



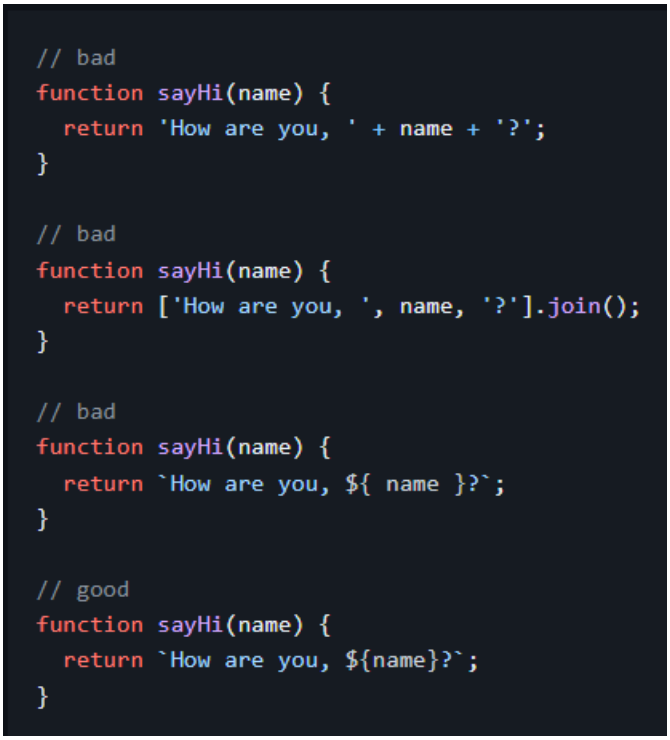
```
// bad
{
  bigBang: {
    display: 'inline-block',
    '::before': {
      content: "''",
    },
  },
  universe: {
    border: 'none',
  },
}

// good
{
  bigBang: {
    display: 'inline-block',

    '::before': {
      content: "''",
    },
  },

  universe: {
    border: 'none',
  },
}
```

---

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

Rule 1:

When programmatically building up strings, use template strings instead of concatenation. This rule is a bit confusing because the code is still readable when using "+" to join strings and variables.



```
// bad
function sayHi(name) {
  return 'How are you, ' + name + '?';
}

// bad
function sayHi(name) {
  return ['How are you, ', name, '?'].join();
}

// bad
function sayHi(name) {
  return `How are you, ${ name }?`;
}

// good
function sayHi(name) {
  return `How are you, ${name}?`;
}
```

Rule 2:

Use named function expressions instead of function declarations. This rule is confusing because a function can be explained with comments instead of it having a long descriptive name as the function.

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

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

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

Rule 3:

Never use the Function constructor to create a new function. This rule is confusing because no suggestion was provided as to what we should be using instead of this.

```
// bad
const add = new Function('a', 'b', 'return a + b');

// still bad
const subtract = Function('a', 'b', 'return a - b');
```
