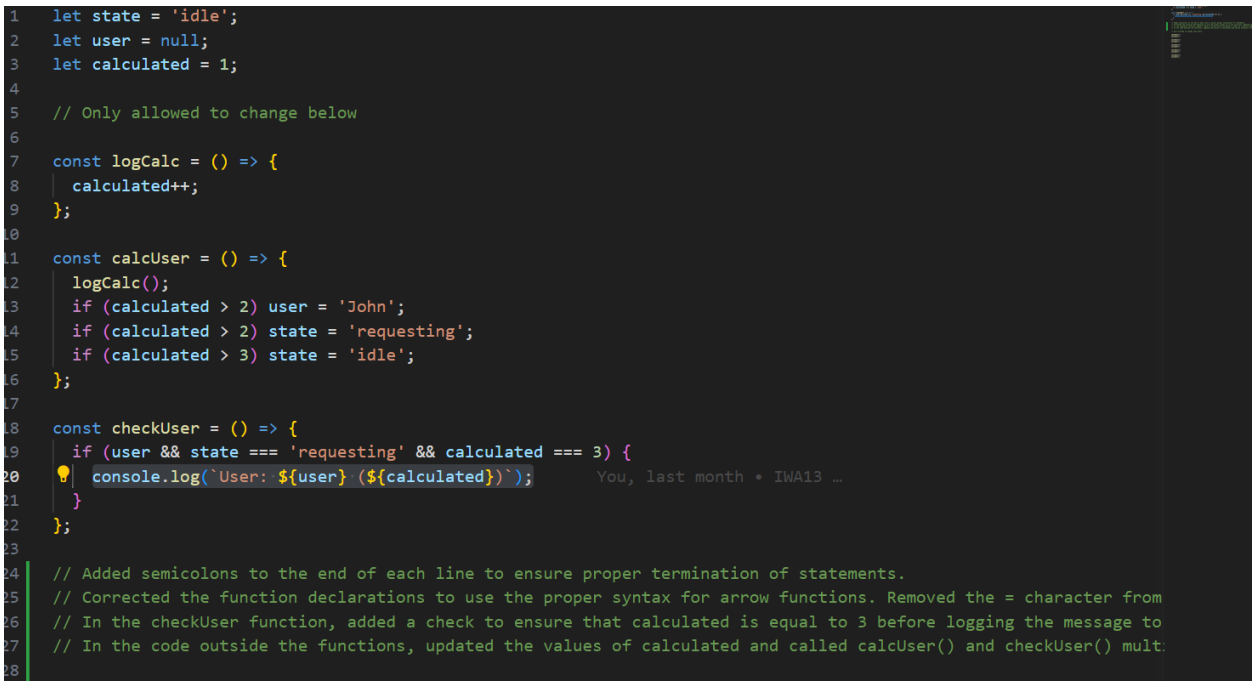
DWA_03.4 Knowledge Check_DWA3.1

---

1. Please show how you applied a Markdown File to a piece of your code.



```
let state = 'idle';
let user = null;
let calculated = 1;

// Only allowed to change below

const logCalc = () => {
  calculated++;
};

const calcUser = () => {
  logCalc();
  if (calculated > 2) user = 'John';
  if (calculated > 2) state = 'requesting';
  if (calculated > 3) state = 'idle';
};

const checkUser = () => {
  if (user && state === 'requesting' && calculated === 3) {
    console.log(`User: ${user} (${calculated})`);      You, last month • IWA13 …
  }
};

// Added semicolons to the end of each line to ensure proper termination of statements.
// Corrected the function declarations to use the proper syntax for arrow functions. Removed the = character from
// In the checkUser function, added a check to ensure that calculated is equal to 3 before logging the message to
// In the code outside the functions, updated the values of calculated and called calcUser() and checkUser() mult
```

When rendered as Markdown, the code block will be formatted with appropriate syntax highlighting, making it more readable and distinguishable from the surrounding text.

---

2. Please show how you applied JSDoc Comments to a piece of your code.

```
/**
 * Calculates the sum of two numbers.
 *
 * @param {number} a - The first number.
 * @param {number} b - The second number.
```

```
 * @returns {number} The sum of the two numbers.
 */
function calculateSum(a, b) {
  return a + b;
}
```

In the above example, JSDoc comments are used to document the `calculateSum` function. Here's a breakdown of the different parts:

- `/** ... */`: The opening and closing comment block for JSDoc comments.
- `@param`: Specifies the parameter(s) of the function and provides a description for each parameter.
- `{number}`: Indicates the type of the parameter. In this case, both `a` and `b` are expected to be numbers.
- `-`: The hyphen is used to separate the parameter name from its description.
- `@returns`: Indicates the return value of the function and provides a description for it.
- `{number}`: Specifies the type of the return value. In this case, the sum of `a` and `b` will be a number.

JSDoc comments help in generating documentation using tools like JSDoc itself or IDEs that support JSDoc annotations. They provide information about function signatures, parameter types, and return types, helping developers understand how to use the code correctly.

---

3. Please show how you applied the @ts-check annotation to a piece of your code.

```
// @ts-check

/**
 * Adds two numbers.
 * @param {number} a - The first number.
```

```
 * @param {number} b - The second number.
 * @returns {number} The sum of the two numbers.
 */
function addNumbers(a, b) {
  return a + b;
}

const result = addNumbers(4, "5"); // Intentional type mismatch

console.log(result);
```

The `@ts-check` annotation enables TypeScript checking within JavaScript files by allowing the TypeScript compiler to perform static type checking on the annotated code.

In the code above, we've added the `@ts-check` annotation at the beginning of the file. This tells TypeScript to perform type checking within this JavaScript file.

Next, we have a function `addNumbers` that takes two parameters, `a` and `b`, both of type `number`. The `@param` annotations provide type information for the function parameters, and the `@returns` annotation specifies the return type.

In the last line, we intentionally pass a string `"5"` instead of a number as the second argument to `addNumbers`. This causes a type mismatch, which will be caught by the TypeScript compiler when the code is checked.

When you run this code with TypeScript checking enabled, you will see an error reported by the compiler indicating the type mismatch between `number` and `string`.

---

4. As a BONUS, please show how you applied any other concept covered in the 'Documentation' module.