JavaScript Scope
Function and Block Scope

Learning Objective: By the end of this lesson, students will be able to distinguish between function and block scope in JavaScript, identify how variables behave within these scopes, and understand the benefits and limitations of each.

Similarities of function and block scope

Function scope and block scope behave similarly, so it makes sense to bundle these types of scope when we're thinking about them. Both function and block scope in JavaScript deal with where you can access a variable. Let's break it down:

- **Function scope**: When you declare a variable inside a function, it can only be accessed within that function. This includes the function's parameters as well.
- **Block scope**: Similarly, if you declare a variable inside a code block such as within an `if` statement or a `for` loop (including any variables defined inside the `( )`), you can only use that variable within that specific block.

In simpler terms, if you declare a variable inside curly braces `{ }`, whether it's part of a function or a code block, that variable can only be used within those curly braces. This helps keep your code organized and avoids conflicts between variables.

Here are a couple of examples of this:

```
const addNums = (numA) => {
  // Inside the addNums function
  const numB = 10;

  console.log(numA + numB);
  // Output: 15
};

addNums(5);

// Outside the addNums function:
console.log(numB);
// ERROR: ReferenceError: numB is not defined
// numB is out of scope!
```

Above, you can see that `numB` isn't available outside the `addNums` function.

> ❓ Is `numA` available outside of the `addNums` function?

Let's look at another example:

```
let isLoggedIn = true;

if (isLoggedIn) {
  // Defining username inside the if block
  const username = 'Frisco';

  console.log(username);
  // Output: 'Frisco'
}

// Outside the if block:
console.log(username);
// ERROR: ReferenceError: username is not defined
```

Here, you can see that `username` isn't available outside the block scope created by the `if` statement.

Let's take a look at an example that combines these two ideas:

```
const chooseDinner = () => {
  let isHungry = true;
  let mainDish;

  if (isHungry) {
    mainDish = 'meatloaf';
  } else {
    mainDish = 'corn';
  }
  // Note how variables that are part of the outer function scope
  // (created by the function) are available to blocks inside its scope!

  console.log(`Dinner tonight is ${mainDish}`);
  // Prints: 'Dinner tonight is Meatloaf'
}

chooseDinner();

// Outside of the function, the variables declared within are unavailable.
console.log(mainDish);
// ERROR: ReferenceError: mainDish is not defined
```

Differences between function and block scope

In JavaScript, both block and function scope are mechanisms that limit the visibility and accessibility of variables. However, they do so in different contexts.

So, if these two types of scope are similar, why do we refer to them separately?

1. Variables declared with the `var` keyword do not respect block scope. This means that `var` is generally less safe and can complicate scope. Therefore, its usage is frowned upon, so we don't use it in our examples, and neither should you when you write code.
2. Variables in a function's scope only exist while the function executes unless the function creates a closure. Closures are a complex topic not covered here. For more info check out MDN Closures.