JS Functions 2

Learning Objectives

- What a return statement of a function is and how to use it in your JavaScript functions
- What an `early return` is
- How to write functions with the `fat arrow notation`

Return Statements

Functions are an incredible versatile and central tool in most programming languages. We already learned how to pass values into a function with input parameters. But a function can also return a value back to the place where it was called. This is done via a `return statement`.

```
function add3Numbers(first, second, third) {
  const sum = first + second + third;
  return sum;
}
```

The `return statement` begins with the keyword `return` followed by an expression. This this case, the expression is the variable sum. Its value is returned by the function and can be stored when the function is called:

```
const firstSum = add3Numbers(1, 2, 3);
// the return value is stored in "firstSum", namely 6

const secondSum = add3Numbers(4, 123, 33);
// the return value is now stored in "secondSum", namely 160
```

> 💡 An expression is anything that produces a value: a variable, a hardcoded value like `true` or `6`, a math operation like `2 + 3` or even another function call! This article explains this in greater depth.

By this, we can outsource computations and / or decision processes and continue using the returned value in the program.

A function can return only one expression value, but can have multiple return statements, in combination with if else statements for example:

```
function checkInputLength(inputString) {
  if (inputString.length > 3) {
    return true;
  } else {
    return false;
    }
  }
```

Early Return Statements

As soon as a return statement is reached in a function call, the function execution is ended. The following `console.log()` is therefore never reached:

```
function testFunction() {
  return "a returned string";

  console.log("I am never logged in the console.");
}
```

This behavior can be used to our advantage as early return statements. Sometimes we want to execute certain parts of our code only if a condition applies. We can check this with an if else statement. When multiple conditions are in place, the code becomes harder to read and to understand:

```
function setBackgroundColor(color) {
  if (typeof color === "String") {
    if (color.startsWith("#")) {
      if (color.length >= 7) {
        document.body.style.backgroundColor = color;
      }
    }
  }
}
```

An alternative approach is to terminate the function with early return statements:

```
function setBackgroundColor(color) {
    // first condition
    if(typeOf color !== 'String') {
        return;
    }

    // second condition
    if(!color.startsWith('#')) {
        return;
    }

    // third condition
    if(color.length < 7) {
        return;
    }
    // only if all 3 conditions are passed the final line of code is
executed.
    body.style.backgroundColor = color;
}
```

This way of writing the code is more readable

💡 Hint: A return statement can be left empty, the returned value is then `undefined`.

Arrow Function Expressions

Next to the classic function declaration, JavaScript has a second way to write functions as `arrow function expressions`:

```
const addNumbers = (first, second) => {
  return first + second;
};
```

The function is saved like a variable with the keyword `const`. The parameters are written normally in round brackets followed by an fat arrow `=>`. Then the function body is written in curly brackets.

Implicit Return Statements

The advantage of arrow functions are possible shorter notations when certain criteria apply:

1. Omit the round brackets around the parameters: This is possible, if there is only one input:

   ```
   const addOne = (number) => {
     return number + 1;
   };
   ```

2. Implicit return statements: If the function consists only of a return statement, the curly brackets and the return keyword can be omitted:

   ```
   const addNumbers = (first, second) => {
     return first + second;
   };
   ```

   can be rewritten as:

   ```
   const addNumbers = (first, second) => first + second;
   ```

> 💡 This shorthand notation comes in handy as soon as we work with callback functions in a few days. So try to remember this feature.

> 💡 Maybe you remember the syntax of the `addEventListener` method. We encountered these arrow functions there already!
>
> ```
> button.addEventListener('click',() => {
>     ...
> })
> ```

Resources

- Statements vs Expressions by Josh Comeau