Section 6 - Functions, Style and Best Practices

Learning Outcomes

Functions isolate sections of code that perform specific tasks so that they can be reused. The use of common coding styles and best practices within teams and organizations improve the quality, consistency, and maintainability of application code.

Content Links

- Functions - https://www.w3schools.com/js/js_functions.asp
- Style Guide - https://www.w3schools.com/js/js_conventions.asp
- Best Practices - https://www.w3schools.com/js/js_best_practices.asp

Functions

A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

Syntax Overview

- A JavaScript function is defined with the `function` keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)
- The code to be executed by the function is placed inside curly brackets: {}

```
Syntax:
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

- Function parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Invocation

The code inside the `function` will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code

- Automatically (self invoked)

Return

When JavaScript reaches a `return` statement, the function will stop executing. If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a `return` value. The return value is "returned" back to the "caller":

```
let x = myFunction(4, 3);   // Function is called, return value will end up in x (12)
function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

Style Guide

Coding Conventions

Coding conventions are style guidelines for programming. They typically cover:

- Naming and declaration rules for variables and functions.
- Rules for the use of white space, indentation, and comments.
- Programming practices and principles

Coding conventions secure quality:

- Improves code readability
- Make code maintenance easier

Coding conventions can be documented rules for teams to follow, or just be your individual coding practice.

Variable Names

Use camelCase for identifier names (variables and functions). All names start with a letter.

```
let firstName = "John";
let lastName = "Doe";
let price = 19.90;
let tax = 0.20;
fullPrice = price + (price * tax);
```

Spaces Around Operators

Always put spaces around operators (= + - * /), and after commas:

```
let x = y + z;
const myArray = ["Volvo", "Saab", "Fiat"];
```

Code Indentation

Always use 2 spaces for indentation of code blocks:

```
function toCelsius(fahrenheit) {
  return (5 / 9) * (fahrenheit - 32);
}
```

Statement Rules

General rules for simple statements:

- Always end a simple statement with a semicolon.

```
const cars = ["Volvo", "Saab", "Fiat"];
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

General rules for complex (compound) statements:

- Put the opening bracket at the end of the first line.
- Use one space before the opening bracket.
- Put the closing bracket on a new line, without leading spaces.
- Do not end a complex statement with a semicolon.

```
Functions:
function toCelsius(fahrenheit) {
  return (5 / 9) * (fahrenheit - 32);
}
Loops:
for (let i = 0; i < 5; i++) {
  x += i;
}
Conditionals:
if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

Naming Conventions

Always use the same naming convention for all your code. For example:

- Variable and function names written as camelCase
- Global variables written in UPPERCASE (We don't, but it's quite common)
- Constants (like PI) written in UPPERCASE

Best Practices

Avoid Global Variables

Minimize the use of global variables. Global variables and functions can be overwritten by other scripts. Use local variables instead.

Always Declare Local Variables

All variables used in a `function` should be declared as local variables. Local variables must be declared with the `var` keyword or the `let` keyword, or the `const` keyword, otherwise they will become global variables.

Declarations on Top

It is a good coding practice to put all declarations at the top of each script or function.

This will:

- Give cleaner code
- Provide a single place to look for local variables
- Make it easier to avoid unwanted (implied) global variables
- Reduce the possibility of unwanted re-declarations

```
// Declare at the beginning
let firstName, lastName, price, discount, fullPrice;
// Use later
firstName = "John";
lastName = "Doe";
price = 19.90;
discount = 0.10;
fullPrice = price - discount;
```

Initialize Variables

It is a good coding practice to initialize variables when you declare them.

This will:

- Give cleaner code
- Provide a single place to initialize variables
- Avoid undefined values

Declare Arrays and Other Objects with `const`.

Declaring arrays and objects with `const` will prevent any accidental change of type:

```
let car = {type:"Fiat", model:"500", color:"white"};
car = "Fiat";      // Changes object to string
const car = {type:"Fiat", model:"500", color:"white"};
car = "Fiat";      // Not possible
```

```
let cars = ["Saab", "Volvo", "BMW"];
cars = 3;    // Changes array to number
const cars = ["Saab", "Volvo", "BMW"];
cars = 3;    // Not possible
```

Beware of Automatic Type Conversions

Recall that JavaScript is loosely typed. A variable can contain all data types. A variable can change its data type:

```
let x = "Hello";     // typeof x is a string
x = 5;               // changes typeof x to a number
```

Beware that numbers can accidentally be converted to strings or NaN (Not a Number). When doing mathematical operations, JavaScript can convert numbers to strings:

```
let x = 5 + 7;       // x.valueOf() is 12,  typeof x is a number
let x = 5 + "7";     // x.valueOf() is 57,  typeof x is a string
let x = "5" + 7;     // x.valueOf() is 57,  typeof x is a string
let x = 5 - 7;       // x.valueOf() is -2,  typeof x is a number
let x = 5 - "7";     // x.valueOf() is -2,  typeof x is a number
let x = "5" - 7;     // x.valueOf() is -2,  typeof x is a number
let x = 5 - "x";     // x.valueOf() is NaN, typeof x is a number
```