4-10. Functions

Q What is a function? *4-10-1*

A Functions are **essential building blocks** in JavaScript. They are also objects. *4-10-2*

A function is a wrapped block of code that performs a task or calculation.

```
function logMessage() {
    console.log("This is my message");
}
```

The above is a named function.

We declare a function with *4-10-3*

- the `function` keyword
- a name of the function
- parameters *(optional)*
- code block defining the function, wrapped within `{ }`.

Declaring a function can be called **function definition**, **function declaration** or **function statement**. *4-10-4*

A function can contain calculations for example *4-10-5*

```
function echoer(word,times) {
  var echo = "";
  for(var i = 0; i < times; i = i + 1) {
    echo = echo + word;
  }
  return echo;
}
```

Invoke a function

4-10-6

We invoke a function by **calling it using `()`**.

```
function logMessage() {
    console.log("This is my message");
}

logMessage(); // This is my message
```

When we invoke the function the **code block defined within our function will be executed**.

4-10-7

When calling `logMessage()` the function will log *This is my message* to the console.

Functions makes it possible for us to run the same code several times without having to write duplicated code.

4-10-8

parameters

4-10-9

Parameters are values we **pass into our functions**.

We can use the value of the parameters within our function

4-10-10

```
function logMessage(message) {
    console.log(message);
}
```

`message` is a parameter to the `logMessage` function.

Parameters are **listed within two parentheses** after the function name.

4-10-11

We can have **zero, one or more** parameters into a function

4-10-12

```
function logMessage(message, secondMessage, thirdMessage) {
    console.log(message, secondMessage, thirdMessage);
}
```

A function does not necessarily need parameters, but we still want to have the **parentheses after our function name**. *4-10-13*

```
function logMessage() {
    console.log("This is my message");
}
```

Parameters are passed into the function when the function is being **invoked** *4-10-14*

```
function logMessage(message) {
    console.log(message);
}

logMessage("My message"); // My message
```

return value *4-10-15*

Functions can **return a value**, using the `return` keyword.

```
function createUsername(firstName, lastName) {
    return "user_" + firstName + lastName;
}

let userName = createUsername("Kalle", "Persson");
console.log(userName); // user_KallePersson
```

When we call a function that return a value we want to **store this in a variable**. *4-10-16*

```
let userName = createUsername("Kalle", "Persson");
```

The return value from a function can be anything, a string, a number, object, etc. *4-10-17*

```
function stringFunction() {
    return "Hello World!"
}

function numberFunction() {
    return 34;
}

function objectFunction() {
    return {
        name: "Kalle"
    }
}
```

If you use `return` in a function without defining what to be returned it will be `undefined`. *4-10-18*

```
function someFunction() {
    return;
}

var value = someFunction();
console.log(value); // undefined
```

If you'd store the execution of a function not using `return` you would also get `undefined`. *4-10-19*

```
function someFunction() {
    console.log("I'm not returning something");
}

var value = someFunction();
console.log(value); // undefined
```

Anonymous functions *4-10-20*

Previously we looked at named functions, we also have something called **anonymous function**.

Q What is an anonymous function? *4-10-21*

A An anonymous function is a function without a name. *4-10-22*

By **omitting the name** of a function is becomes an anonymous function. *4-10-23*

```
var myAnonymousFunction = function() {
    console.log("My anonymous function");
}

(function() {
    console.log("My anonymous function which is immediately invoked");
})();
```

We can store our anonymous function in a variable or choose to invoke it immediately. *4-10-24*

The latter is common if we want a function to run once in the code.

Both named and anonymous functions stored in a variable are invoked the same way. *4-10-25*

```
function logMessage() {
    console.log("My function");
}

var myAnonymousFunction = function() {
    console.log("My anonymous function");
}

logMessage(); // My function

myAnonymousFunction(); // My anonymous function
```

What is the **difference** between this: *4-10-26*

```
function func() {
    console.log("My Function");
}

var x = func;
```

and **this**:

```
function func() {
    console.log("My Function");
}

var x = func();
```

A

One **stores the function** and the other one **stores the value returned** from the function. *4-10-27*

The use of parentheses makes the difference.

4-10-28

```
function func() {
    console.log("My Function");
}

var x = func;
```

By assigning a function to a variable we get a **reference to the function**, in this case `x` has a reference to `func`.

This means that after assigning the function to a variable, *without using parentheses*, doing `x()` and `func()` would yield the same thing. *4-10-29*

```
function func() {
    console.log("My Function");
}

var x = func;
x(); // My Function
func(); // My Function
```

4-10-30

```
function func() {
    console.log("My Function");
}

var x = func();
```

In this second example, `x` is whatever `func` returns, in this case `undefined`.

If we would explicitly return something `x` would get the given value. *4-10-31*

```
function func() {
    return "Hello World!"
}

var x = func(); // Hello World!
```

This is because we invoke a function with the parentheses, `()` *4-10-32*

Functions as objects *4-10-33*

Functions are objects, which means there is nothing stopping us from **having functions on objects**:

```
myObject.beingAnnoying = function() {
  console.log("SPAM!");
};
```

It is created as an anonymous function.

When a function is a property on an object it is called a **method** of that object. *4-10-34*

```
myObject.beingAnnoying = function() {
  console.log("SPAM!");
};
myObject.beingAnnoying(); // SPAM!
```

We invoke the method just the same with parentheses.

And to end on a meta note; there is also nothing stopping us from having objects (or anything else) on functions, since **functions are objects**. *4-10-35*

```
var func = function(){};
func.prop = {
    hello: "world!"
};
console.log(func.prop.hello); // "world!"
```

First class citizens *4-10-36*

Functions are **first class citizens** in JavaScript.

Q What is a **first class citizen**? *4-10-37*

A

In programming, a **first class citizen** is a type that supports operations generally available for other entities. *4-10-38*

Such operation can be return from a function, modification, assign to a variable, pass as an argument to a function.

Functions can like strings, objects or other types be the entity in such operation. *4-10-39*

We have seen how we assign a function to a variable *4-10-40*

```
var x = function() {
    console.log("Function");
}
```

We can also pass functions as arguments to other functions. *4-10-41*

Such as callback functions for **on click events**: *4-10-42*

```
let button = document.getElementById('myButton');
button.addEventListener('click', function() {
    console.log("Clicked the button");
});
```

We pass a function into the `addEventListener` function.

IIFE *4-10-43*

IIFE stands for **I**mmediately **I**nvoked **F**unction **E**xpression.

Can also be called **Self-Executing Anonymous Function**

We have seen how to create a function to be invoked later in the code. *4-10-44*

```
function logMessage() {
    console.log("This is my message");
}

logMessage(); // This is my message
```

We can also create functions that get invoked as soon as it is defined. *4-10-45*

```
(function() {
    console.log(23*4);
})();
```

Immediately invoked functions are good when we want a function to run once, when we do not have to name it or store it for later use. *4-10-46*

Arrow functions *4-10-47*

With ES6 we get introduced to **arrow function**.

```
let myFunction = () => {
    console.log("Arrow function");
}
```

With arrow functions we get a shorter syntax for writing functions. *4-10-48*

We exclude the `function` keyword.