COURSEWARE

Professional Skills

Agile Fundamentals

Jira

Git

Databases Introduction

Java Beginner

Maven

Testing (Foundation)

Java Intermediate

HTML

CSS

Javascript

- What is JavaScript
- Getting started with JS
- Variables
- Data types
- ASI
- Strict mode
- Iteration
- Conditionals with Truthy / Falsey
- Objects, Arrays + JSON
- Structuring JS Code
- Destructuring
- Scope
- Functions, function expressions and arrow functions
- The ECMAScript 6 Specification
- OOP in JavaScript
- Best Practices
- Closures
- Callbacks and Promises
- Cookies
- Hoisting
- Prototypes
- Query Parameters
- Higher Order Functions

Functions, function expressions and arrow functions

Contents

Overview

In this module we will be exploring how to write and use functions, function expressions and arrow functions.

What are functions

A JavaScript function is a block of code designed to perform a particular task(s). A JavaScript function is executed when "something" invokes it (calls it). We use functions in order to stop the duplication of code, we are able to use the same function with multiple times with the different arguments and as a result produce different outcomes. Functions have a global scope and therefore allow use throughout our code.

What are function expressions

Function expressions are remarkably like functions, however they are syntactically different. The major difference between functions and function expressions are that functions are used globally making it available throughout our code. whereas function expressions are limited to where the function is available keeping the global scope light and maintain clean syntax.

what are arrow functions

An arrow function expression is a syntactically compact alternative to a regular function expression. Arrow function expressions are ill suited as methods, and they cannot be used as constructors.

Tutorial

How to write and use javascript functions

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.

```
function name(parameter1, parameter2, parameter3) {
  // Some 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.

Function 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)

Function 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 = myFunc(4, 3);
// myFunc is called with two parameters, the return value will end up in x

function myFunc(a, b) {
  return a + b;
// myFunc returns the sum of a and b
}
console.log(x);
//Will give an output of 12.
```

Using the **()** invokes the function, accessing a function without () will return the function object instead of the function result.
e.g.

```
function add(a,b) {
    return a+b;
}
add(10,12);
// Will give out of 22

add
// Will return function add (a,b){return a+b;}
```

Functions can also be used as variable values.

```
function add(a,b) {
    return a+b;
}
let myStr = "The sum of 10 + 12 is " + add(10,12);
console.log(myStr);
//Output is "The sum of 10 + 12 is 22"
```

Variables within functions become LOCAL to the function and can only be accessed from within the function.

How to write and use javascript function expressions

The way a function expression is written is as follows.

```
let welcome = function(){
    alert("Hello");
};
```

Another example of a function expression with parameters

```
const getRectArea = function(width, height) {
  return width * height;
};

console.log(getRectArea(3, 4));
// Output is 12
```

Another difference between functions and function expressions is that functions (as they have a global scope) are initialised first due to the internal algorithms of JavaScript.
Whereas function expressions, on the other hand, won't be executed until the execution reaches them.
Below is an example.

Function - this would work

```
sayHi("John"); // Hello, John
function sayHi(name) {
  alert( `Hello, ${name}` );
}
```

Function expression - this would not work.

```
sayHi("John"); // error!

let sayHi = function(name) {  // (*) no magic any more
  alert( `Hello, ${name}` );
};
```

As mentioned before function expressions are created when the execution reaches them.
In this case that would happen on the line marked `(*)`.

You may wonder why at the end of the function expression we have a `;`, the reason for this is because a Function Expression is used inside the statement: `let sayHi = ...;`, as a value.
It's not a code block, but rather an assignment. The semicolon ; is recommended at the end of statements, no matter what the value is.
So the semicolon here is not related to the Function Expression itself, it just terminates the statement.

How to write and use arrow functions

When writing arrow functions we omit the `function` keyword.

e.g.

```
hello = () => {
    return "Hello World!";
}
```

We can make them even shorter.
If the function has only one statement, and the statement returns a value, we can remove the brackets and the `return` keyword.
This only works if the function has only one statement.

```
hello = () => "Hello World!";
```

Parsing parameters with arrow functions.

```
hello = (name) => "Hello " + name;
```

We can also pass in multiple parameters.

```
hello = (name, age) => "Hello " + name +"! you are "+ age +" years old!";
```

If you have only one parameter, you can skip the parentheses as well:

```
hello = name => "Hello " + name;
```

Exercises

1. Create a **function** that takes in 2 parameters `num1` and `num2` and subtracts the two numbers.

   ▶ Solution
2. Create a **function expression** called `welcome` that take in `name`, `age`,`gender` as a parameters. The outcome should be like so:
   e.g. `"My name is Felix Cited, i am 20 years old and of gender Male"`

   ▶ Solution
3. Create an **arrow function** called `powerUp` that takes in two values `n1` and `n2`. The arrow function will return the power of the two numbers.

   e.g.

   - n1=2, n2=3 will return 8 (2 x 2 x 2);
   - n1=3, n2=3 will return 27 (3 x 3 x 3);

     hint use `Math.pow()`

     ▶ Solution