JavaScript Functions

Summary: in this tutorial, you will learn about JavaScript functions and how to use them to structure the code into smaller and more reusable units.

Introduction to JavaScript functions

When developing an application, you often need to perform the same action in many places. For example, you may want to show a message whenever an error occurs.

To avoid repeating the same code all over places, you can use a function to wrap that code and reuse it.

JavaScript provides many built-in functions such as `parseInt()` and `parseFloat()`. In this tutorial, you will learn how to develop custom functions.

Declaring a function

To declare a function, you use the `function` keyword, followed by the function name, a list of parameters, and the function body as follows:

```
function functionName(parameters) {
    // function body
    // ...
}
```

The function name must be a valid JavaScript identifier. By convention, the function names are in `camelCase` and start with verbs like `getData()`, `fetchContents()`, and `isValid()`.

A function can accept zero, one, or multiple parameters. In the case of multiple parameters, you need to use a comma to separate two parameters.

The following declares a function `say()` that accepts no parameter:

```
function say() {
}
```

The following declares a function named `square()` that accepts one parameter:

```
function square(a) {
}
```

The following declares a function named `add()` that accepts two parameters:

```
function add(a, b) {
}
```

Inside the function body, you can write the code to implement an action.

For example, the following `say()` function shows a message to the console:

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

In the body of the `say()` function, we call the `console.log()` function to output a message to the console.

Calling a function

To use a function, you need to call it. Calling a function is also known as invoking a function.

To call a function, you use its name followed by arguments enclosed in parentheses like this:

```
functionName(arguments);
```

When calling a function, JavaScript executes the code inside the function body. For example, the following shows how to call the `say()` function:

```
say('Hello');
```

In this example, we call the `say()` function and pass a literal string `'Hello'` into it.

Parameters vs. Arguments

The terms parameters and arguments are often used interchangeably. However, they are essentially different.

When declaring a function, you specify the parameters. However, when calling a function, you pass the arguments corresponding to the parameters.

For example, in the `say()` function, the `message` is the parameter and the `'Hello'` string is an argument that corresponds to the `message` parameter.

Returning a value

Every function in JavaScript implicitly returns `undefined` unless you explicitly specify a return value. For example:

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

let result = say('Hello');
console.log('Result:', result);
```

Output:

```
Hello
Result: undefined
```

To specify a return value for a function, you use the `return` statement followed by an expression or a value, like this:

```
return expression;
```

For example, the following `add()` function returns the sum of the two arguments:

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

The following shows how to call the `add()` function:

```
let sum = add(10, 20);
console.log('Sum:', sum);
```

Output:

```
Sum: 30
```

The following example uses multiple `return` statements in a function to return different values based on conditions:

```
function compare(a, b) {
    if (a > b) {
        return -1;
    } else if (a < b) {
        return 1;
    }
```

```
    return 0;
}
```

The `compare()` function compares two values. It returns:

- -1 if the first argument is greater than the second one.
- 1 if the first argument is less than the second one.
- 0 if the first argument equals the second one.

The function immediately stops executing when it reaches the `return` statement. Therefore, you can use the `return` statement without a value to exit the function prematurely, like this:

```
function say(message) {
  // show nothing if the message is empty
  if (!message) {
    return;
  }
  console.log(message);
}
```

In this example, if the `message` is blank (or `undefined`), the `say()` function will show nothing.

The function can return a single value. If you want to return multiple values from a function, you need to pack these values in an array or an object.

The arguments object

Inside a function, you can access an object called `arguments` that represents the named arguments of the function.

The `arguments` object behaves like an array though it is not an instance of the Array type.

For example, you can use the square bracket `[]` to access the arguments: `arguments[0]` returns the first argument, `arguments[1]` returns the second one, and so on.

You can also use the `length` property of the `arguments` object to determine the number of arguments.

The following example implements a generic `add()` function that calculates the sum of any number of arguments.

```
function add() {
  let sum = 0;
  for (let i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
}
```

Hence, you can pass any number of arguments to the `add()` function, like this:

```
console.log(add(1, 2)); // 3
console.log(add(1, 2, 3, 4, 5)); // 15
```

Function hoisting

In JavaScript, you can use a function before declaring it. For example:

```
showMe(); // a hoisting example

function showMe() {
  console.log('an hoisting example');
}
```

This feature is called hoisting.

Function hoisting is a mechanism in which the JavaScript engine physically moves function declarations to the top of the code before executing them.

The following shows the version of the code before the JavaScript engine executes it:

```
function showMe() {
  console.log('a hoisting example');
}

showMe(); // a hoisting example
```

Function hoisting allows you to call a function before declaring it, making the development workflow more smoothly.

Summary

- Use the `function` keyword to declare a function.
- Use the `functionName()` to call a function.
- All functions implicitly return `undefined` if they don't explicitly return a value.
- Use the `return` statement to return a value from a function explicitly.
- The `arguments` variable is an array-like object inside a function, representing function arguments.
- The function hoisting allows you to call a function before declaring it.

Quiz