Hyperiondev

TASK

Using Built-In Functions and Defining your own Functions

Visit our website

Introduction

WELCOME TO THE BUILT-IN FUNCTIONS AND DEFINING YOUR OWN FUNCTIONS TASK!

This is an introduction to *functions* in JavaScript. A function is a reusable and organised block of code that is used to perform a single action or specific task(s). Functions can either be user-defined or built-in. In this task, you will be introduced to functions that are built into the JavaScript language itself and are readily available for us to use. We are going to be exploring how to create your own function to make your code more efficient.

INTRODUCTION TO FUNCTIONS IN JAVASCRIPT

A function is a unit/block of code that contains all the instructions needed to complete a specific task. Functions allow you to split a complex task into simpler tasks, which makes managing and maintaining scripts easier. Another benefit of using functions is that functions are the means by which we will restructure our code to minimise repetition and to reduce potential errors. Functions are little blocks of code we can call on repeatedly in our code which enables us to not repeat the same lines of code over and over again.

Functions can either be user-defined or built-in. Built-in functions are built into the JavaScript language itself and are readily available for us to use.

There are thousands of functions already implemented in JavaScript that you can use to get things done. Programmers have already written the logic for many common and even complex tasks and sometimes you can find the exact built-in function that you need to complete a task.

However, you are not limited to these built-in functions. You can also create your own functions to meet your own specific needs. These are what are known as “user-defined” functions.

CREATING YOUR OWN FUNCTIONS: FUNCTION DECLARATION

Before learning about some built-in JavaScript functions, you will first learn to create your own functions. There are a few ways in which a function can be created.

The easiest method of declaring a function is shown below.

```
function doubleNumber(number) {
    return number * 2;
}
```

The function that has been created in the example above is called **doubleNumber**. It takes as input the parameter **number**. A *parameter* is a variable that is declared in a function definition. Parameters store the data needed to perform the action that the function specifies. Parameters are filled when data is passed to the function as the function is called (which you will learn about soon). The code between the curly braces (**{}**), is the logic of the function. It defines what happens when the function is called. Simply put, the function in the example above takes a number and multiplies it by 2. It then 'returns' the resulting value.

The general syntax of a function in JavaScript is as follows:

```
function functionName(parameters){
       statements;
       return (expression);
}
```

The `function` and `return` Keywords

Note the **function** keyword. JavaScript knows that you're defining a function when you start a line with this keyword. After the keyword **function,** you will then put a function name, the function's input parameters in brackets, (**()**), and then curly braces, (**{}**), with the logic of the function indented underneath.

Note the **return** keyword. A JavaScript function can return a value but it doesn't have to. The value after the keyword **return** will be returned/passed back to whatever code called the function. As mentioned previously the **return** keyword returns a value, however it also ends the execution of the function; therefore all statements added after the **return** keyword will not be executed.

CALLING A FUNCTION

In order to execute a function, you need to 'call' it (a function is therefore not automatically 'called'). You call a function by using the function's name followed by the values you would like to pass to the parameters within parentheses. The values that you pass to the function are referred to as *arguments*.

In the example below, the function that was defined above (**doubleNumber**) is called. In this example, we pass the value 10 as an argument to the function **doubleNumber**. Since we created a parameter called **number** when we defined the function **doubleNumber**, passing the argument 10 to the function **doubleNumber** will result in

the parameter **number** being assigned the value 10.

```
let doubleTen = doubleNumber(10);
```

Think of a call to the function (e.g. **doubleNumber(10)**), as a 'placeholder' for some computation. The function will go off and run its code and return its result in that place. The instruction above, therefore, does the following:

1. Creates a variable called **doubleTen** (**let doubleTen**)
2. Calls the function called **doubleNumber** and passes the value 10 as an argument to that function. The function **doubleNumber** is then executed and the result of the statements in the function (**return number * 2;**) are returned.
3. The returned value is then stored in the variable. The result of the statement **let doubleTen = doubleNumber(10);** is therefore **doubleTen = 20;**

You can define a function, but it will not run unless called somewhere in the code. For example, although we have defined the function **doubleNumber** above, the code within the curly braces would never be executed unless there was another line that called **doubleNumber** with the command **doubleNumber(some_value)** somewhere in the main body of your code.

Try this:

1. Open Chrome's Developer Console. To do this, open Chrome and then press either Ctrl+Shift+J if you are using Windows / Linux or Cmd+Opt+J if you are using Mac.

2. Copy and paste the code below into the console.

```
function doubleNumber(number) {
    return number * 2;
}

let doubleTen = doubleNumber(10);
console.log(doubleTen);
```

3. Press enter. Take careful note of the output. Be sure you understand how the code you have entered resulted in the output displayed in the console.

4. Now clear the console and then copy and paste the code below into the console and press enter:

```
function doubleNumber(number) {
    return number * 2;
}
console.log(doubleNumber(10));
```

5. You should notice that this code does exactly the same as the code above but with less code. Do you understand why?

SCOPE

Scope is what we call a program's ability to find and use variables in a program. The rule of thumb is that a function is covered in one-way glass: it can see out, but no one can see in. This means that a function can call variables that are outside the function, but the rest of the code cannot call variables that are defined within the function. Let's look at an example:

```
function adding(a, b) {
    let total = a + b;
    return description + String(total);
}

let x = 2;
let y = 3;
let description = "Total: ";

let sum = adding(x, y);

console.log(sum);
```

Output:

```
Total: 5
```

In the code above, the function makes use of the *description* variable inside the function. This shows that the function can look outside and use variables from outside the function. Now let's see what happens if we put *description* inside the function:

```
function adding(a, b) {
    let total = a + b;
    let description = "Total: ";
    return String(total)
}

let x = 2;
let y = 3;

let sum = adding(x, y);
```

```
console.log(description + sum);
```

Output:

```
error: Uncaught ReferenceError: description is not defined
```

See how the program complains that it can't find the *description* variable? That's because of the 'one-way glass': the rest of the code can't see into the function and so does not know that a *description* variable exists.

BUILT-IN JAVASCRIPT METHODS

A method is a special type of function (more on this later). As you have learned, you can create your own functions, but there are also a lot of methods that have already been written by JavaScript developers that we can use. In a later section, you will learn about some very important built-in methods that you will use to get your JavaScript 'talking' to your HTML.

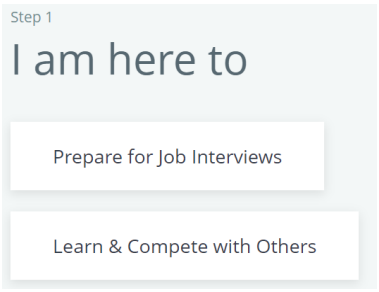
For now, we will look at some common built-in functions. You may not realise it, but you have already been using some built-in functions, especially with arrays and maps (have a look at the arrays and maps task if you need a refresher).

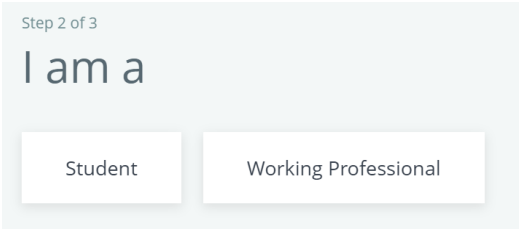
- *charAt()* - returns a character in a string at the given index
- *indexOf()* - returns the index of a the first occurrence of a character in a string
- *charCodeAt()* - returns the ascii value of the character at the given index in a string
- *fromCharCode()* - returns the character of the given ascii value
- *replace()* - replaces a matched substring with a new substring
- *split()* - splits a string into an array of substrings
- *toUpperCase()* - returns the given string all in upper case
- *toLowerCase()* - returns the given string in all lower case
- *join()* - opposite of *split()*. Joins all array elements into a string
- *pow()* - returns a base to the exponent power
- *min()* - returns the smallest valued element
- *max()* - returns the largest value element
- *round()* - returns a number rounded to the nearest integer

Extra resource

HackerRank is an excellent resource for newbies and professional coders alike. It is a hub of standardised coding exercises that you can use to practise your skills as you go through this bootcamp. Go to **https://www.hackerrank.com/** and sign

up. Once you have created an account, select “I am here to prepare for job interviews”, “I am a student” and fill in when you are expected to graduate from HyperionDev.





Once you’re on your dashboard, scroll down to “Skills Available For Practice” and select Algorithms — you will then be able to select JavaScript as your coding language when you open your first challenge. You will be taken to a whole bunch of algorithmic challenges to try. You should already be able to solve a couple, but keep checking back to your dashboard as you go through the bootcamp to see how you progress!

HackerRank is a very common platform for companies on which to interview and test potential developers, so make sure that you are comfortable with the way that problems are phrased and how the platform works.

Instructions

Open **example.js** in Visual Studio Code and read through the comments before attempting these tasks.

Getting to grips with JavaScript takes practice. You will make mistakes in this task. This is completely to be expected as you learn the keywords and syntax rules of this programming language. It is vital that you learn to debug your code. To help with this, remember that you can:

- Use either the JavaScript console or Visual Studio Code (or another editor of your choice) to execute and debug JavaScript.
- Remember that if you really get stuck, you can contact an expert code reviewer for help.

Compulsory Task 1

Follow these steps:

- *Note: For this task you will need to create an HTML file to get input from a user. If you need a refresher on how to do this, go back to the **example.js** and **index.html** files in your Task 1 folder for a refresher.*
- Create a new JavaScript file in this folder called **numberManipulation.js**
- Write a program that starts by asking the user to input 10 numbers (these can be a combination of whole numbers and decimals). Store these numbers in a list.
- Find the total of all the numbers and log the result to the console.
- Find the index of the maximum and log the result to the console.
- Find the index of the minimum and log the result to the console.
- Calculate the average of the numbers and round off to 2 decimal places. Log the result to the console.
- Find the median number and log the result to the console.

Compulsory Task 2

Follow these steps:

- *Note: For this task you will need to create an HTML file to get input from a user. If you need a refresher on how to do this, go back to the **example.js** and **index.html** files in your Task 1 folder for a refresher.*
- Create a JavaScript file called **wordManipulation.js** in this folder.
- Write a program that starts by asking the user to input a word.
- The program should log the following manipulations to the console:
    - The word where every second character is replaced with a '!'.
    - The word reversed.

- The word where every 6th letter is uppercase.

Rate us
Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.