Hyperiondev

TASK

# Javascript III: Scripting and Event Handling

Visit our website

# Introduction

## WELCOME TO THE THIRD JAVASCRIPT TASK!

Now that you know a little JavaScript we can use it to make our web pages more interesting and useful! In this task, you will learn how to apply JavaScript to your HTML. To be able to do this you will need to understand two *very* important concepts including:

1. What *functions* are and how to create your own JavaScript functions.
2. How to write JavaScript functions that respond to DOM *events*.

Get in touch

**Connect for support**

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

## INTEGRATING JAVASCRIPT AND HTML

Up until now, you've learnt some basic JavaScript, but you haven't yet used it on a web page. Before you can learn more JavaScript, it's important to see how it's applied to web pages as a scripting language.

## THE SCRIPT ELEMENT

To add JavaScript to an HTML file we need to introduce an important new HTML element: the `<script>` element. The HTML `<script>` element is used to embed or reference executable code like JavaScript. All JavaScript, when placed in an HTML document, needs to be within a script element. Assuming the page is viewed in a browser that has JavaScript enabled, the browser will execute the JavaScript statements in the <script> element as the page is loading.

Where to place the `<script>` tags are important in the functioning of your code. Ideally, it should be placed above the closing `</body>` (outside the head elements). This way, the html markdown will load first and then the JavaScript code will load.

You can either insert JavaScript directly into the `<script>` element of the HTML page (see **example.html**) or you can put the JavaScript in an external JavaScript file and link to that file in the <script> element (see **example2.html**).

A script element that contains inline JavaScript looks like this:

```
<script type="text/javascript">
     console.log("Hello World");
</script>
```

An example of a `<script>` element that refers to an external JavaScript file is shown below:

```
<script type="text/javascript" src="script.js"></script>
```

It is generally best to use external JavaScript files instead of including all JavaScript in HTML files. One reason for this is that it is considered best practice to separate content (HTML) from behaviour (JavaScript). This generally makes code easier to maintain and can even improve the performance of your website.

**SPOT CHECK 1**

Let's see what you can remember from this section.

1. How do we add a JavaScript file called **script.js** to our HTML file?

## EVENTS

JavaScript is often used to handle *events* that occur on a website. An event is basically an action that occurs that your program responds to. You have encountered DOM events. DOM events (you will learn what DOM is soon) are things that either a user does, such as clicking a button or entering text into a textbox, or actions caused by the browser, such as when a web page has completely loaded.

The most common events you'll deal with when getting started are:

| Event | Description |
|---|---|
| `onchange` | Some HTML element has been modified |
| `onclick` | An HTML element has been clicked on |
| `onmouseover` | An HTML element was hovered over |
| `onmouseout` | Mouse cursor moves off HTML element |
| `onkeydown` | A keyboard button is pressed |
| `onload` | The HTML page has finished loading |

Other events are listed and explained **here**.

We can write JavaScript code that tells your browser what you want it to do when it realises that a certain event has occurred. This code could be written in a *JavaScript function*.

## 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 make 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. Run and read through the code and comments in 'Example 3' for this task to see the benefits of using functions.

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 logic 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. 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.

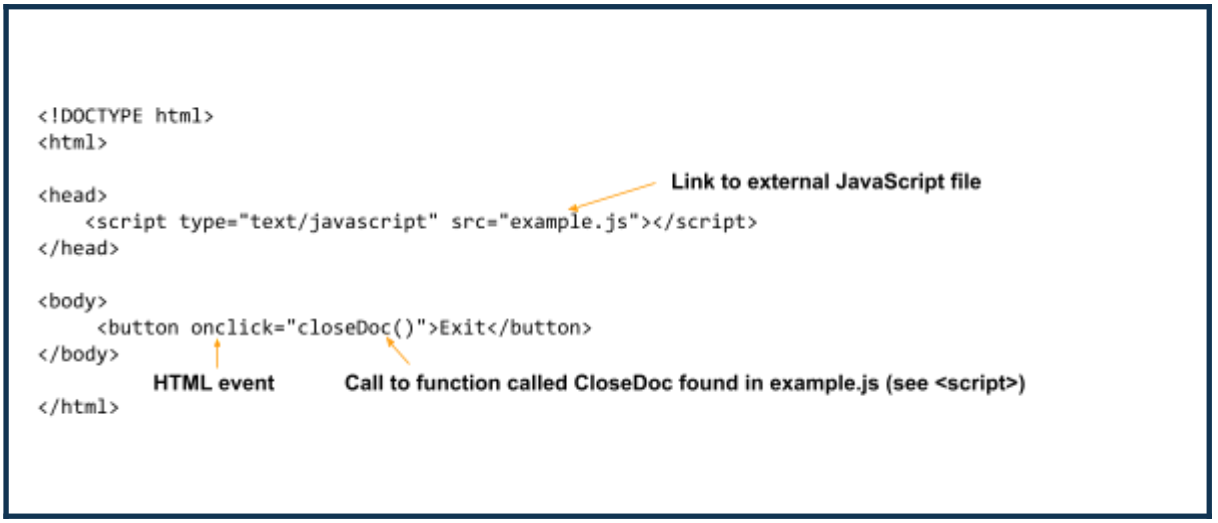
4. Now clear the console and then copy and paste the code below into the console and press enter:

```
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?

## JAVASCRIPT FUNCTIONS AS EVENT HANDLERS

As stated earlier, JavaScript is often used to handle events that occur on a website. To do this, simply identify the DOM event and specify the name of the JavaScript function that you want to call when the event is triggered. See the example below:




```
<!DOCTYPE html>
<html>

<head>
    <script type="text/javascript" src="example.js"></script>
</head>

<body>
    <button onclick="closeDoc()">Exit</button>
</body>
</html>
```

The above is an example of an HTML file where we want our browser to listen for a certain event. Notice that in the `<head>` element, there is a `<script>` element that refers to the external JavaScript file called **example.js**. Also, notice the button

element: here we instruct the browser to call the function named `closeDoc()` when the button is clicked. The `closeDoc()` function is declared in the external Javascript file, **example.js**.

The JavaScript file called **example.js** would contain the following function which will be called and executed when the button in the HTML page is clicked:

```
function closeDoc(){
      alert("You are closing this page!");
      window.close();
}
```

## 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.

**SPOT CHECK 2**

Let's see what you can remember from this section.

1. How would you create a function called `sentence` that received three strings (`str1`, `str2` and `str3`) as arguments, and returned the first, second and third strings put together with a space in between each?

2. **True or false:** the names of the arguments that the function is given when it's called must be the same names as the arguments that the function itself uses. I.e. Will this code run?

```
function minus (num1, num2) {
   answer = num1 - num2;
   return answer;
}

let firstNumber = 6;
let secondNumber = 2;
let total = minus(firstNumber,secondNumber);

console.log(total);
```

## BUILT-IN JAVASCRIPT METHODS

A method as 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. Here are some common examples:

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 function, 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 number of the character at the given index in a string
- *fromCharCode()* - returns the character of the given ascii number
- *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

In this section, you will learn about some very important built-in methods that you will use to get your JavaScript 'talking' to your HTML. Some of the most commonly used methods that are used to get information about your HTML page are listed below:

| Method | Description |
|---|---|
| `document.createElement('tag');` | Creates an element with the given tag name. E.g. document.createElement('p') would create a `<p>` element. |
| `document.getElementById('id');` | Returns an object reference to the identified element. For example if you had a `<div>` element for which you had set the id attribute of that element to 'myDiv' and you called `document.getElementById('myDiv')` that `<div>` element would be returned. |

| | |
|---|---|
| `element.appendChild(aChild);` | Nests an element to another element. See “Example 3” for more information. |

See “Example 4” for more details regarding how each of these methods is used.

# Instructions

Open *all* the examples (in the Examples directory for this task) in VSCode 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 VSCode (or another editor of your choice) to execute and debug JavaScript in the next few tasks.
- Remember that if you really get stuck you can contact your mentor for help.

# Compulsory Task 1

**Follow these steps:**

- Create a basic HTML file called **task1.html** with a few headings, paragraphs, forms, and images. Ensure that each item has an ID assigned to it. Create a JavaScript file called **task1.js**. Save both **task1.html** and **task1.js** in a directory called “Task 1” within the “Compulsory Tasks” directory.

- Create buttons to do the following:
    - Change the size of a heading
    - Change the font style of a paragraph
    - Highlight a paragraph
    - Hide an Image
    - Alternate between two images

- Change an image when the mouse moves onto it, and then back to the original image when the mouse moves off of it.

- Create a script which creates an alert on the page when a user right clicks on a specific image.

- Create an alert on the page when a user enters data into a form.

- Create an alert to let the user know when the page has loaded

- Highlight all paragraphs when the mouse moves over them (then revert back once it has moved past them).

## Compulsory Task 2

**Follow these steps:**

- Modify the basic HTML page called **math.html** in the "Compulsory Tasks" directory. Create a JavaScript file called **task2.js**. Modify the HTML and write the JavaScript necessary to create a web page that allows a user to do some basic mathematical computations. The user should be able to enter numbers and press on a button to show the results of the calculation.

## Compulsory Task 3

**Follow these steps:**

- Create a basic HTML file (called **task3.html**) with a few headings, paragraphs, forms and images. Ensure that each item has an ID assigned to it. You will use this HTML page to get input and display output for each task listed below. Write the JavaScript needed in a file called **task3.js**.
- Create a function which counts and displays the number of times a button has been clicked.
- Create a function to convert rands into dollars, euros and pounds.
- Create a function which creates a drop-down menu with 25 option elements and add it to the HTML page you have created. Each option element should display the number of the option. Use a loop.
- Create a calculator:
    - It should have the layout similar to the calculator on your computer
    - Use CSS to help you style the layout
    - Use JavaScript functions to implement its operation

## Completed the task(s)?

Ask your mentor to review your work!

**Review work**

### Things to look out for:

1. Make sure that you have installed and set up all programs correctly. You have set up **Dropbox** correctly if you are reading this, but **Visual Studio Code** may not be installed correctly.

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.

**SPOT CHECK 1 ANSWERS:**

1. You add the following within the head element of the HTML file: `<script type="text/javascript" src="script.js"></script>`

**SPOT CHECK 2 ANSWERS:**

1.

```
function sentence(str1,str2,str3) {
   let sentence = `${str1} ${str2} ${str3}`
   return sentence;
}
```

2. False. The names of the arguments can be different as long as the data types are the same. Therefore, the code will run.