Functions

Functions are blocks of code designed to perform specific tasks. Once a function is declared, the code within it only runs when the function is called. You can reuse the code in functions every time they are called.

```
function doSomething( ) {
  /* function code */
}
```

The syntax for a function begins with the keyword function, followed by the name you choose for the function. The name is your choice, but it should describe what the function does. It's common to use camelCase for function names with more than one word. Also, you can't start a function name with a number. Following the function name, you have parentheses that may or may not contain parameters (more on this later). This function doesn't have anything within the parentheses, so it has no parameters. Then, within the curly braces, is all the code that will run when the function is called. This is called a declared function.

```
// A simple function to greet someone
function greetUser() {
  console.log('Hello');
}
// Calling the function
greetUser();
```

It simply logs the message 'Hello' onto the console. Notice you must call it by name for it to run.

This function does the same thing, but this time it's called greetName and it has a parameter called name. The value that comes into the function via the name parameter is used after the word 'Hello' in the console message. The function is called twice, and this time an argument is sent with each call. So, the first time the function runs, name holds the value 'Ruby', and the second time, name holds the value 'Bob'. The message shows up on the console for each function call.

```
// A simple function to greet someone
function greetName(name) {
  console.log('Hello ' + name);
}
// Calling the function
greetName('Ruby');
```

```
greetName('Bob');
```

Here is another function named addIt.

```
let num1 = 2;
let num2 = 5;

function addIt(n1, n2) {
  return n1 + n2;
}

document.querySelector('h1').textContent = addIt(num1,
num2);
```

We have two variables, num1 and num2, which have been assigned the numbers 2 and 5. The function will add the two numbers coming in and return the sum of the numbers back to where the function was called. It is being called here, and once the value is returned back where it was called, it is assigned to the text content of an <h1> element on our HTML page.

Notice when the function was called, the two arguments, 2 and 5, were sent to the function by referencing the variable names num1 and num2, but when they came in as parameters to the addIt function, the parameter or value placeholder names changed to n1 and n2. You don't have to use a different name here, but in this case, we did, and it's fine as long as you keep using the parameter names of n1 and n2 within the function's code.

Something interesting with declared functions like we've seen here is that you can call them before or after they are actually declared. This is called 'hoisting'. For example, in all our functions so far, we called them after the declaration of the function, but we can actually move the call above or before the function is declared, and it will find it later in the code just fine.

7

```
let h1Content = document.querySelector('h1');
let num1 = 2;
let num2 = 5;

h1Content.textContent = addIt(num1, num2);

function addIt(n1, n2) {
    return(n1 + n2)
}
```

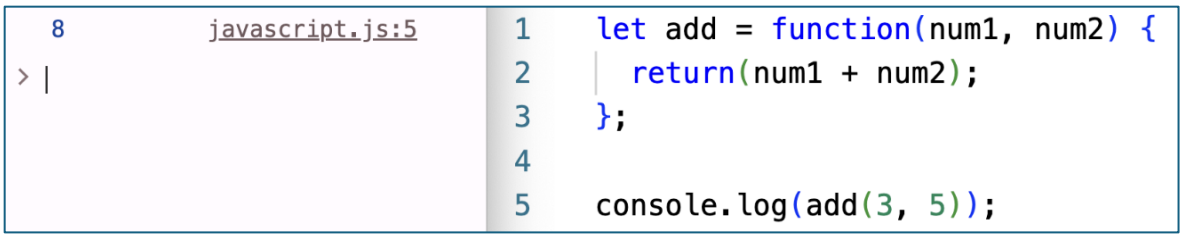
```
let h1Content = document.querySelector(‘h1’);
let num1 = 2;
let num2 = 5;
h1Content.textContent = addIt(num1, num2);
function addIt(n1, n2) {
```

```
        return(n1 + n2)
}
```

Other types of functions cannot be hoisted, only declaration functions like we have seen so far.

Let's look at other types of functions.

An expression function is a way to define a function as part of an expression. Here, we have a function with no name, or in other words, an anonymous function is defined. It is being assigned to a variable called add.



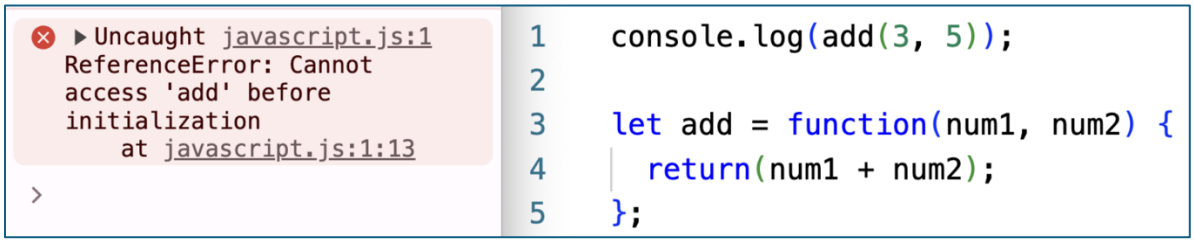

```
let add = function(num1, num2) {
     return(num1 + num2);
};
console.log(add(3, 5);
```

With function expressions, you can pass a function as an argument or store a function in a data structure easily.

How do I know if I should use an anonymous function? Ask yourself, will the code in the function be used again in any other context or need to be called from somewhere else in the code? If the answer is yes, then name it. If no, then you can use an anonymous function.

The function ends in a semicolon here because the function is part of an expression.

Also, functions used in an expression cannot be hoisted.




```
console.log(add(3, 5));
let add = function(num1, num2) {
```

```
        return(num1 + num2)
    }
```

You can call the function by using the variable name add, the same way you might call a declared function.

You might also see anonymous functions in other ways. Here is an event listener using an anonymous function. Instead of calling a declared function name as the second parameter of the .addEventListener() method, the entire anonymous function is the second parameter. It doesn't need a name because it doesn't have to be called. It just runs when the event happens.

```
<button id="myButton">Click Me!</button>
<p id="message"></p>

<script>
  // Get the button element
  const button = document.getElementById('myButton');

  // Add an event listener for the 'click' event
  button.addEventListener('click', function () {
    // Change the text of the paragraph
    document.getElementById('message').textContent = 'Button was clicked!';
  });
</script>
```

Another type of function is the arrow function. It is useful for short, simple functions. The syntax is short and concise. Again, because arrow functions are part of an expression, they cannot be hoisted, and you should end them in a semicolon.

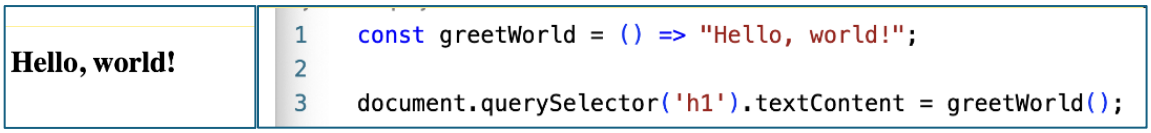
```
11          javascript.js:3
```

```
1   let sum = (no1, no2) => no1 + no2;
2
3   console.log(sum(5,6));
```

```
let sum = (no1, no2) => no1 + no2;
console.log(sum(5,6));
```

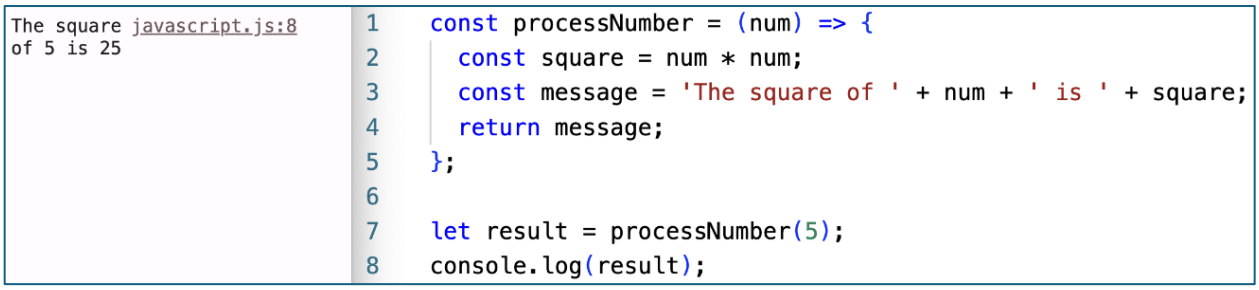
This arrow function is being assigned to a variable called sum. Notice there is no keyword function. It begins with the two parameters followed by an arrow, which is the equal sign and greater than symbol. The code that runs when the function is called follows the arrow, with no need for curly braces if the code that runs is a single expression.

Here's an example with no parameters:




```
const greetWorld = ( ) => “Hello, world!”;
document.querySelector(‘h1’).textContent = greetWorld( );
```

If there is more than one expression to run in the arrow function, you will need curly braces.




```
const processNumber = (num) => {
     const square = num * num;
     const message = ‘The square of ‘ + num + ‘ is ‘ +
square;
     return message;
};
let result = processNumber(5);
console.log(result);
```

So, there we have some functions. You will use functions often in code, and they are key to writing organized, maintainable, and reusable code.