JavaScript

What is JavaScript?

- JavaScript is a dynamic computer programming language.
- It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.
- It is an interpreted programming language with object-oriented capabilities.

What is JavaScript?

- JavaScript is a single-threaded programming language that we can use for client-side or server-side development.
- It is a dynamically typed programming language, which means that we don’t care about variable data types while writing the JavaScript code.
- Also, it contains the control statements, operators, and objects like Array, Math, Data, etc.

JavaScript Features

- **Easy Setup**
  - We don’t need a particular editor to start writing the JavaScript code. Even anyone can write JavaScript code in Notepad. Also, JavaScript can be executed in the browser without any interpreter or compiler setup.
  - You can use the <script > tag to add JavaScript in the HTML file. However, it also allows you to add JavaScript to the web page from the external JavaScript file, having '.js' extension.

JavaScript Features

- **Browser Support**
  - All browsers support JavaScript, as all modern browser comes with the built-in JavaScript execution environment.
  - However, you can also use the ‘window’ object to check whether the browser supports JavaScript or its particular feature.

JavaScript Features

- **Dom Manipulation**
  - JavaScript allows developers to manipulate the webpage elements. Also, you can control the browser.
  - It contains the various methods to access the DOM elements using different attributes and allows to customize the HTML elements.

JavaScript Features

- **Event Handling**
  - JavaScript allows you to handle the **events** used to interact with the web page.
  - For example, you can detect the mouse click on a particular HTML element using JavaScript and interact with the HTML element.
  - Some other events also exist, like detecting the scrolling behavior of web page, etc.

JavaScript Features

- **Dynamic Typing**
  - JavaScript decides the type of variables at runtime. So, we don’t need to care about variable data type while writing the code, providing more flexibility to write code.
  - Also, you can assign the values of the different data types to a single variable. For example, if you have stored the number value of a particular variable, you can update the variable’s value with the string.

JavaScript Features

- **Functional Programming**
  - JavaScript supports the **functional** programming. In JavaScript, you can define the first-class function, pure functions, closures, higher-order functions, arrow functions, function expressions, etc.
  - It mostly uses the functions as a primary building blocks to solve the problem.

JavaScript Features

- **Cross-platform Support**
  - Each operating system and browser support JavaScript. So, it is widely used for developing websites, mobile applications, games, desktop applications, etc.

- **Object-oriented Programming**
  - JavaScript contains the classes, and we can implement all **object-oriented** programming concepts using its functionality.
  - It also supports inheritance, abstraction, polymorphism, encapsulation, etc, concepts of Object-oriented programming.

JavaScript Features

- **Built-in Objects**
  - JavaScript contains built-in objects like Math and Date. We can use a Math object to perform mathematical operations and a Date object to manipulate the date easily.
  - However, you can also manipulate the functionality of the built-in object.
- **Object Prototypes**
  - In JavaScript, everything is an **object**. For example, array, function, number, string, boolean, set, map, etc. are objects.
  - Each object contains the prototype property, which is hidden. You can use the prototype property to achieve inheritance or extend the functionality of class or object, by other object's functionality.

JavaScript Features

- **Built-in Methods**
  - JavaScript also contains the built-in methods for each object. Developers can use the built-in methods to write efficient and shorter codes.
  - For example, the Array object contains the **filter**() method to filter array elements and the **sort**() method to sort the array. The String object contains the **replace**() method to replace text in the string, the **trim**() method to remove whitespaces from the string, etc.

JavaScript Syntax

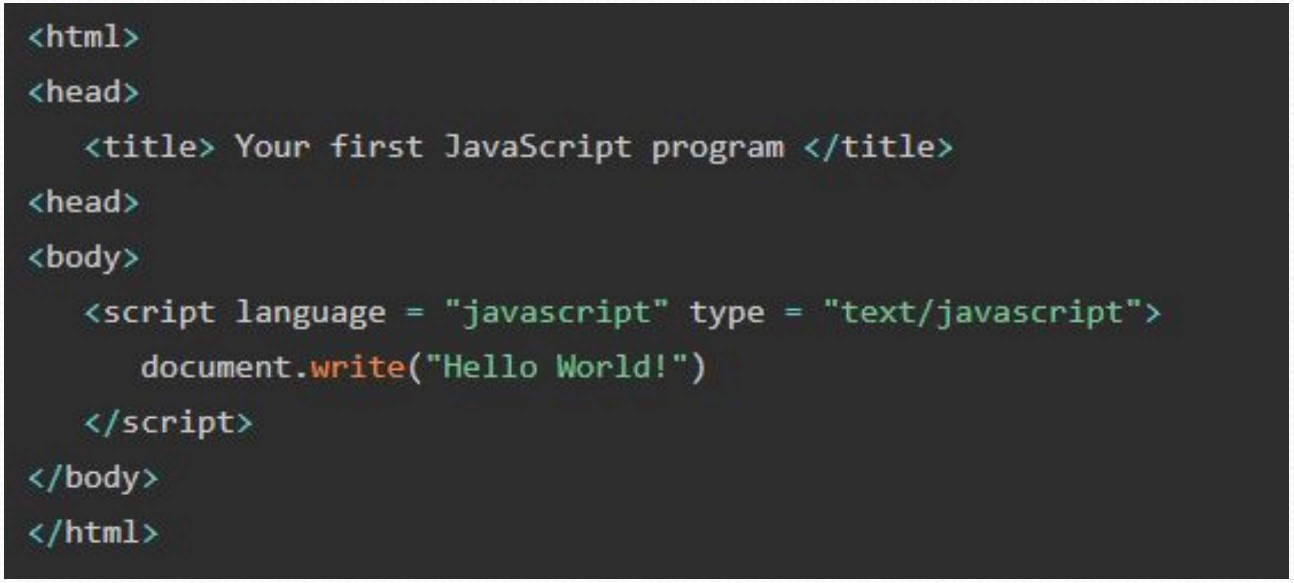
JavaScript Syntax

- JavaScript syntax comprises a set of rules that define how to construct a JavaScript code. JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.
- You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.
- The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
    JavaScript code
</script>
```

First JavaScript Example

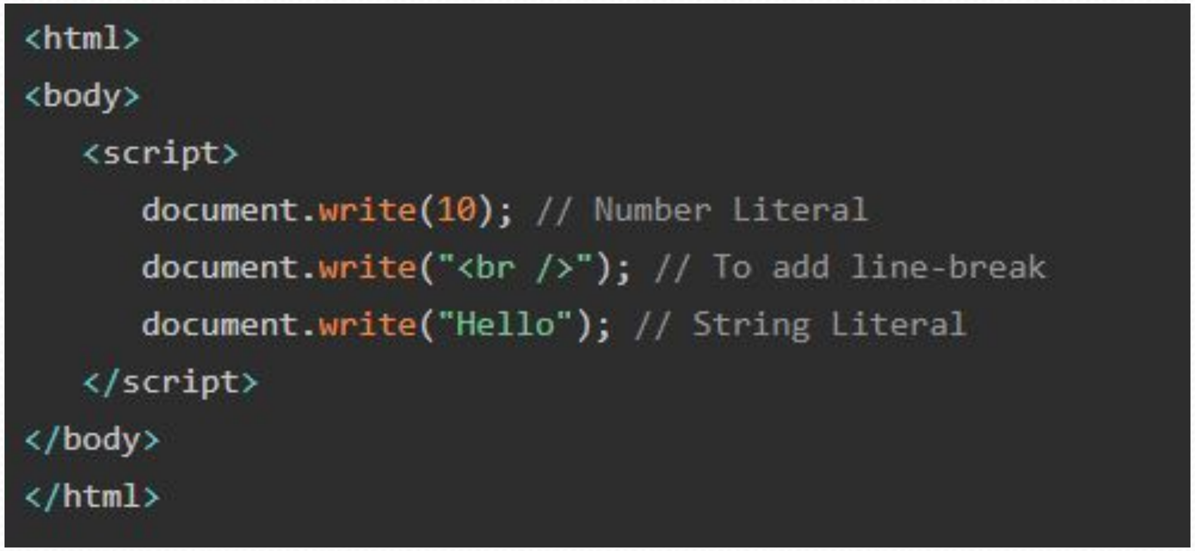
- Let us take a sample example to print out "Hello World". We call **document.write** method which writes a string into our HTML document. This method can be used to write text, HTML, or both.



```
<html>
<head>
   <title> Your first JavaScript program </title>
<head>
<body>
   <script language = "javascript" type = "text/javascript">
      document.write("Hello World!")
   </script>
</body>
</html>
```

JavaScript Values

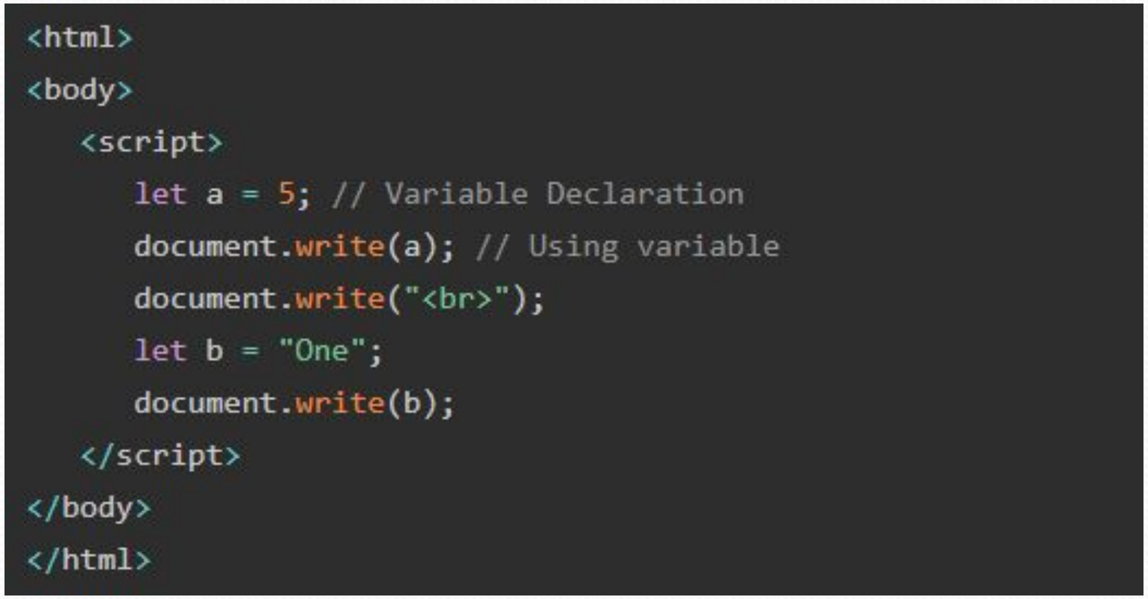
- In JavaScript, you can have two types of values.
  - Fixed values (Literals)
  - Variables (Dynamic values)



```
<html>
<body>
   <script>
      document.write(10); // Number Literal
      document.write("<br />"); // To add line-break
      document.write("Hello"); // String Literal
   </script>
</body>
</html>
```

JavaScript Variables

- In JavaScript, variables are used to store the dynamic data. You can use the below keyword to define variables in JavaScript.
  - var
  - let
  - const



```
<html>
<body>
   <script>
      let a = 5; // Variable Declaration
      document.write(a); // Using variable
      document.write("<br>");
      let b = "One";
      document.write(b);
   </script>
</body>
</html>
```

JavaScript Data Types

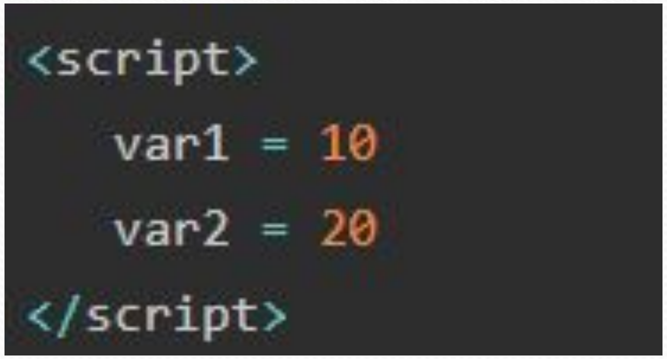
- Data types in JavaScript refers to the types of the values that we are storing or working with.

- One of the most fundamental characteristics of a programming language is the set of **data types** it supports. These are the type of values that can be represented and manipulated in a programming language.

- JavaScript data types can be categorized as primitive and non-primitive (object)

Semicolons are Optional

- Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line.
- For example, the following code could be written without semicolons.



```
<script>
   var1 = 10
   var2 = 20
</script>
```

Case Sensitivity

- JavaScript is a case-sensitive language.
- This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.
- So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

Java Script Operators

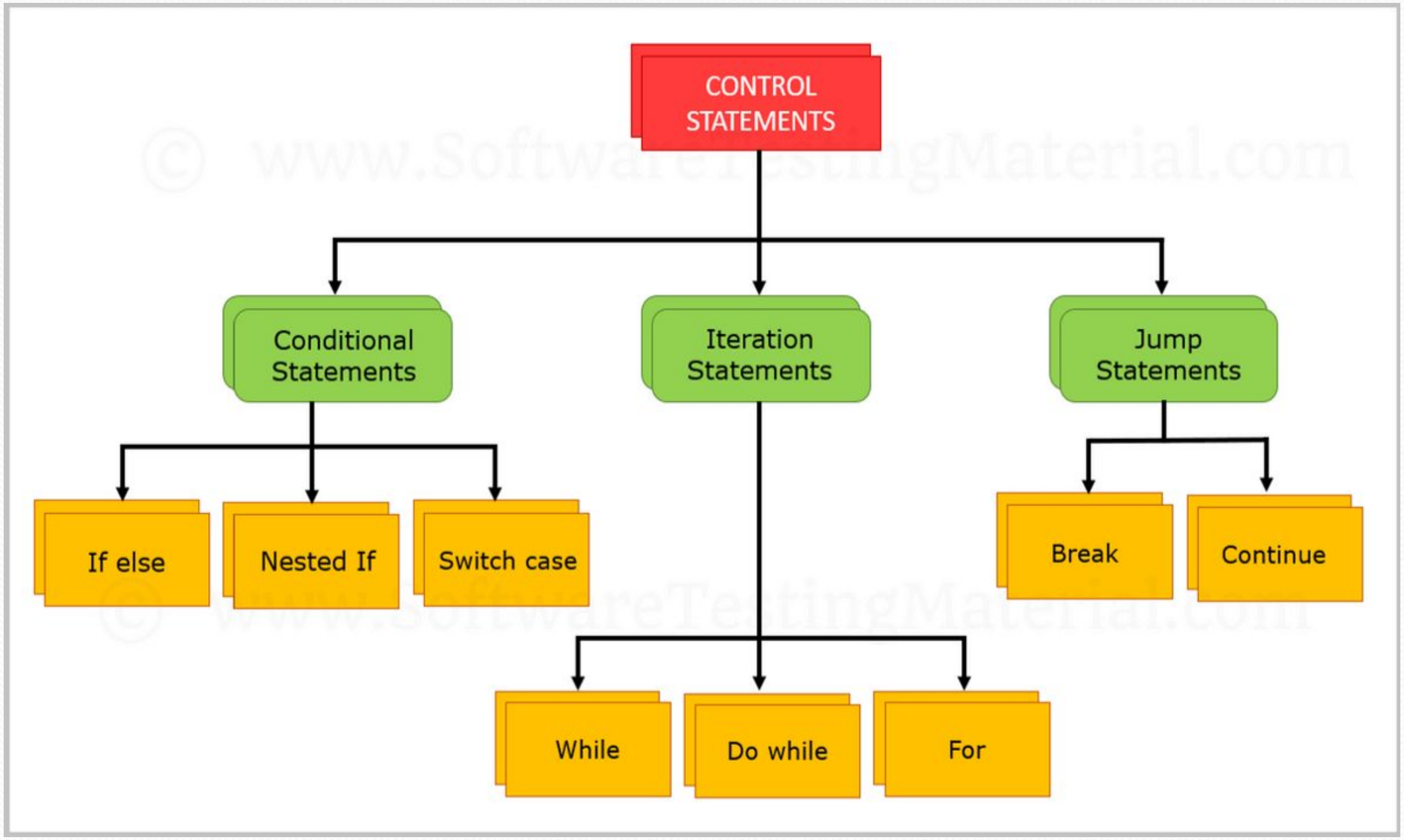
- JavaScript supports the following types of operators.
  - Arithmetic Operators
  - Comparison Operators
  - Logical (or Relational) Operators
  - Bitwise Operators
  - Assignment Operators

Java Script Control Statements

- **Types of Control Statements in JavaScript**
  - **Conditional Statement:** These statements are used for decision-making, a decision is made by the conditional statement based on an expression that is passed. Either YES or NO.

  - **Iterative Statement:** This is a statement that iterates repeatedly until a condition is met. Simply said, if we have an expression, the statement will keep repeating itself until and unless it is satisfied.

Java Script Control Statements



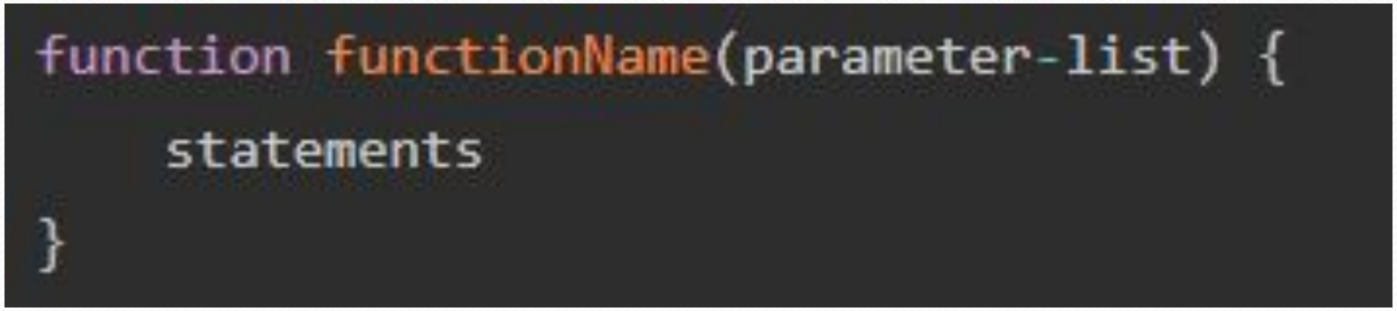
JavaScript Functions

JavaScript Functions

- A **function** in JavaScript is a group of reusable code that can be called anywhere in your program.
- It eliminates the need of writing the same code again and again.
- It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

JavaScript Function Definition

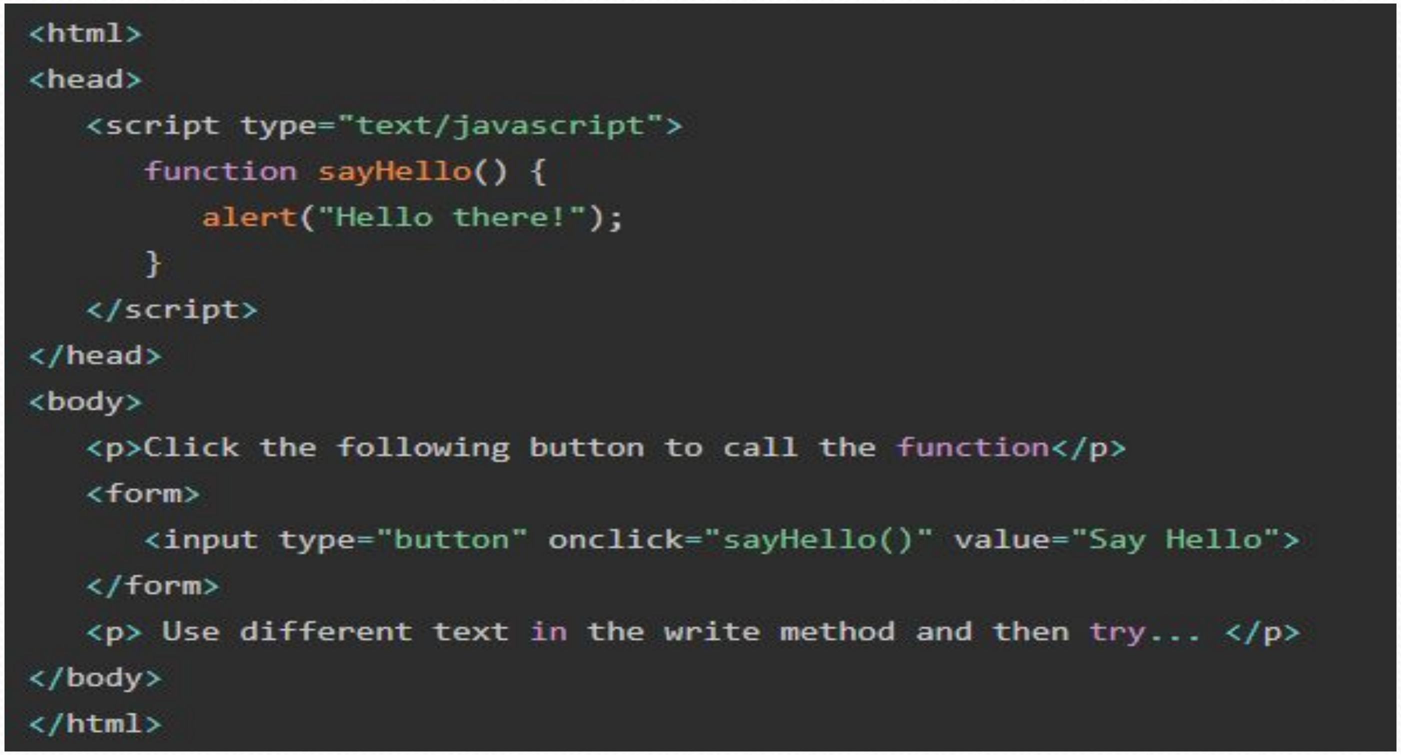
- Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.
- All statements you need to execute on the function call must be written inside the curly braces.



```
function functionName(parameter-list) {
    statements
}
```

JavaScript Function Calling

- To invoke a function somewhere later in the script, you would simply need to write the name of that function with the parentheses ().

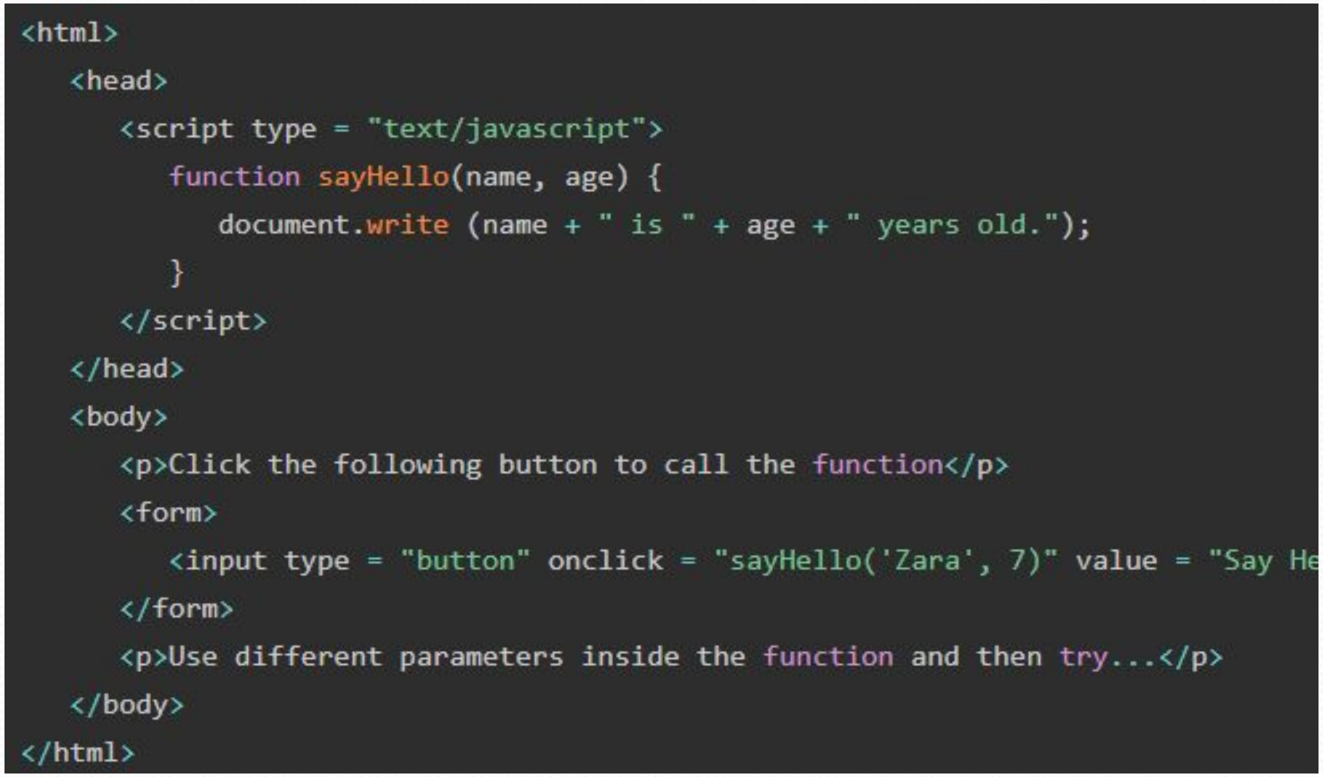


```
<html>
<head>
   <script type="text/javascript">
      function sayHello() {
         alert("Hello there!");
      }
   </script>
</head>
<body>
   <p>Click the following button to call the function</p>
   <form>
      <input type="button" onclick="sayHello()" value="Say Hello">
   </form>
   <p> Use different text in the write method and then try... </p>
</body>
</html>
```

JavaScript Function Parameters

- Till now, we have seen functions without parameters.
- But there is a facility to pass different parameters while calling a function.
- These passed parameters can be captured inside the function and any manipulation can be done over those parameters.
- A function can take multiple parameters separated by comma.

JavaScript Function Parameters



```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello(name, age) {
            document.write (name + " is " + age + " years old.");
         }
      </script>
   </head>
   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "sayHello('Zara', 7)" value = "Say He
      </form>
      <p>Use different parameters inside the function and then try...</p>
   </body>
</html>
```

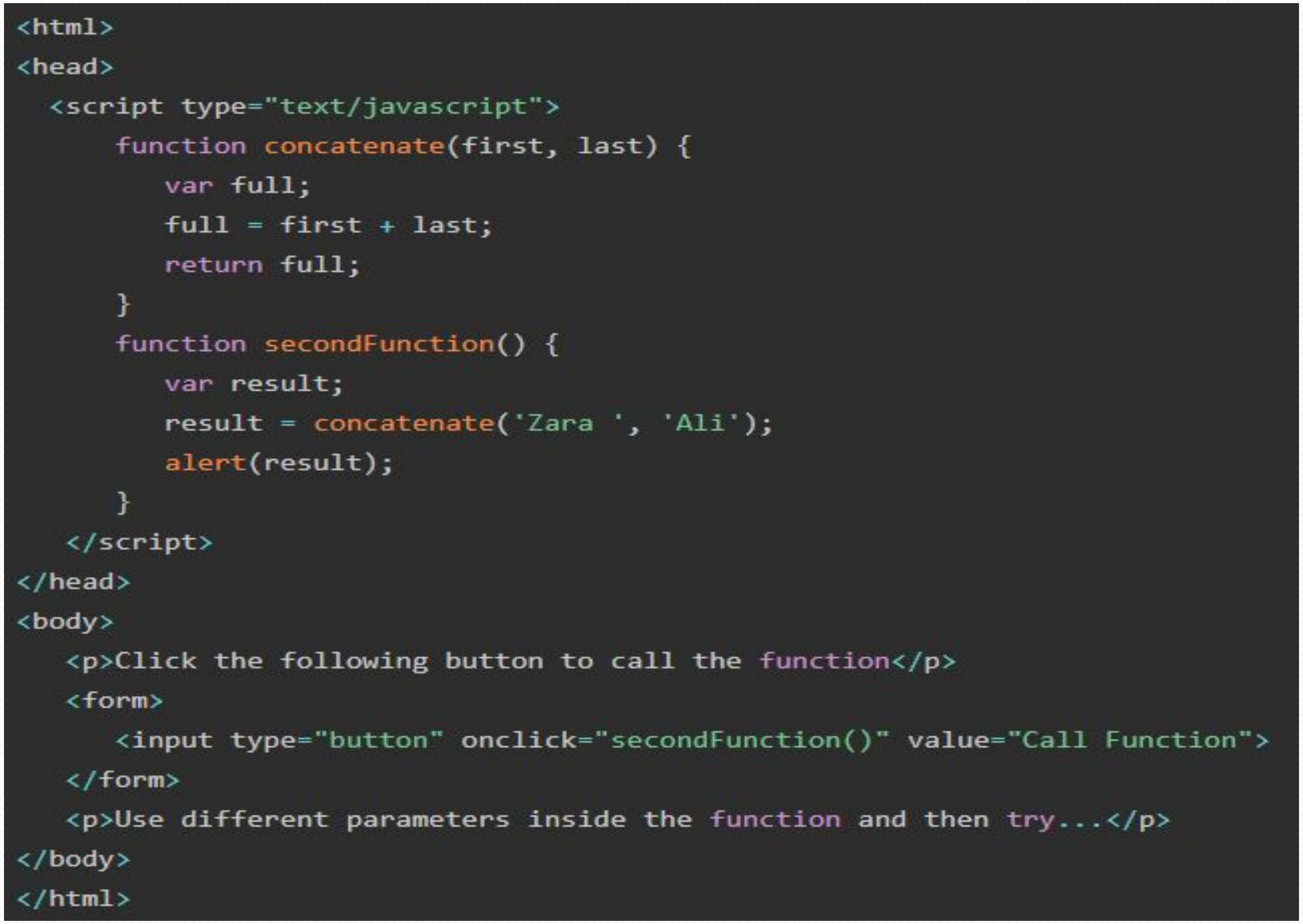
The return Statement

- A JavaScript function can have an optional **return** statement.

- This is required if you want to return a value from a function. This statement should be the last statement in a function.

- For example, you can pass two numbers in a function, and then you can expect the function to return their multiplication in your calling program.

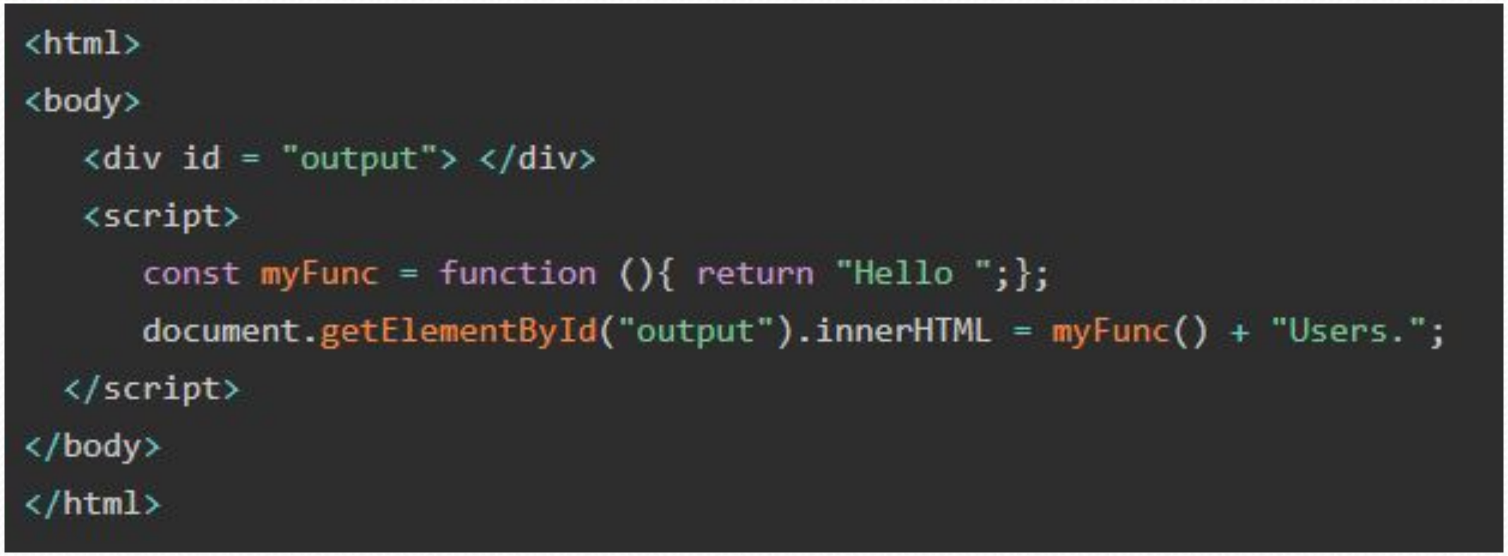
The return Statement



```
<html>
<head>
  <script type="text/javascript">
      function concatenate(first, last) {
         var full;
         full = first + last;
         return full;
      }
      function secondFunction() {
         var result;
         result = concatenate('Zara ', 'Ali');
         alert(result);
      }
   </script>
</head>
<body>
   <p>Click the following button to call the function</p>
   <form>
      <input type="button" onclick="secondFunction()" value="Call Function">
   </form>
   <p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Function as variable values

In JavaScript, functions can be used same as other variables.



```
<html>
<body>
   <div id = "output"> </div>
   <script>
      const myFunc = function (){ return "Hello ";};
      document.getElementById("output").innerHTML = myFunc() + "Users.";
  </script>
</body>
</html>
```

JavaScript Arrow Functions

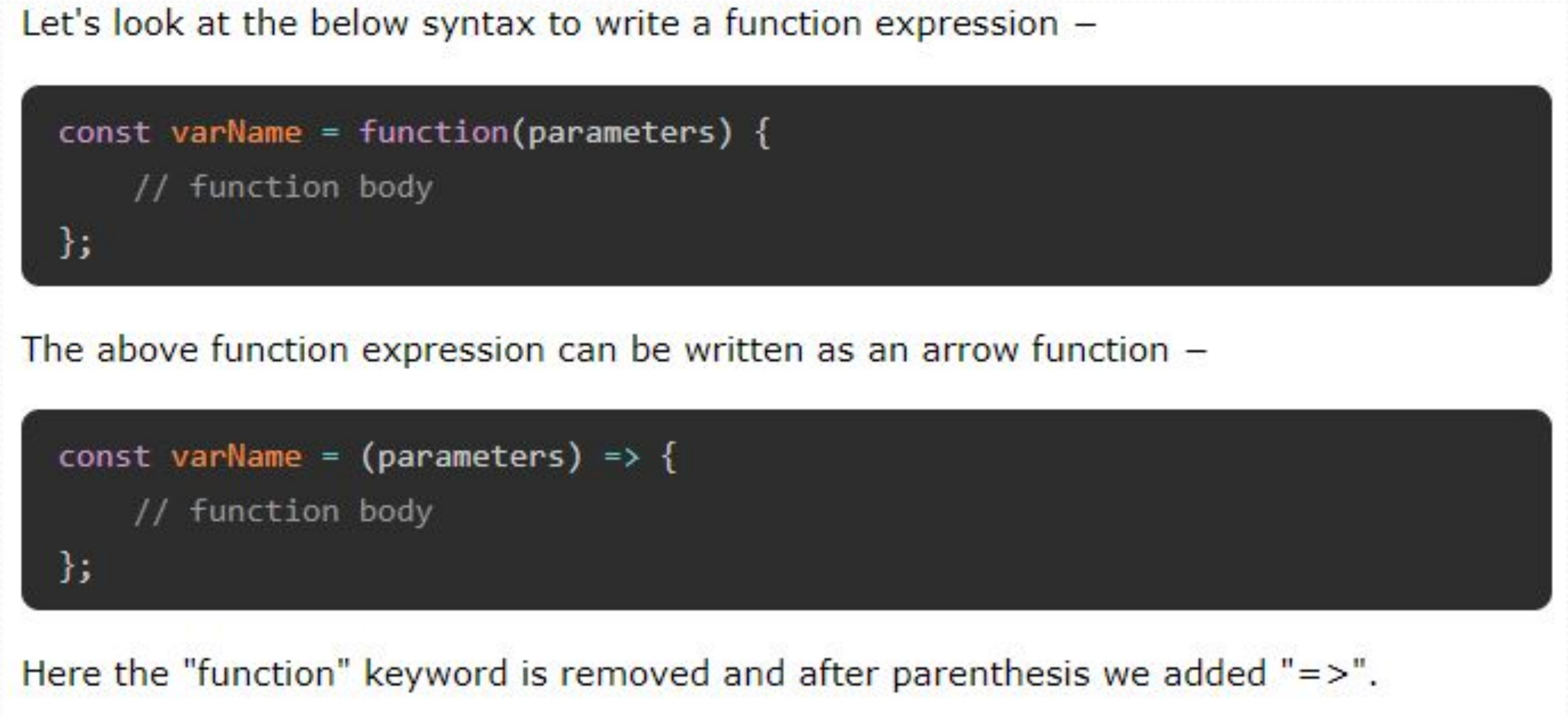
Arrow Functions

- The **arrow functions** in JavaScript allow us to create a shorter and **anonymous** function.

- Arrow functions are written without "function" keyword.

- The JavaScript *arrow functions* are introduced in ES6.

Arrow Functions



Let's look at the below syntax to write a function expression –

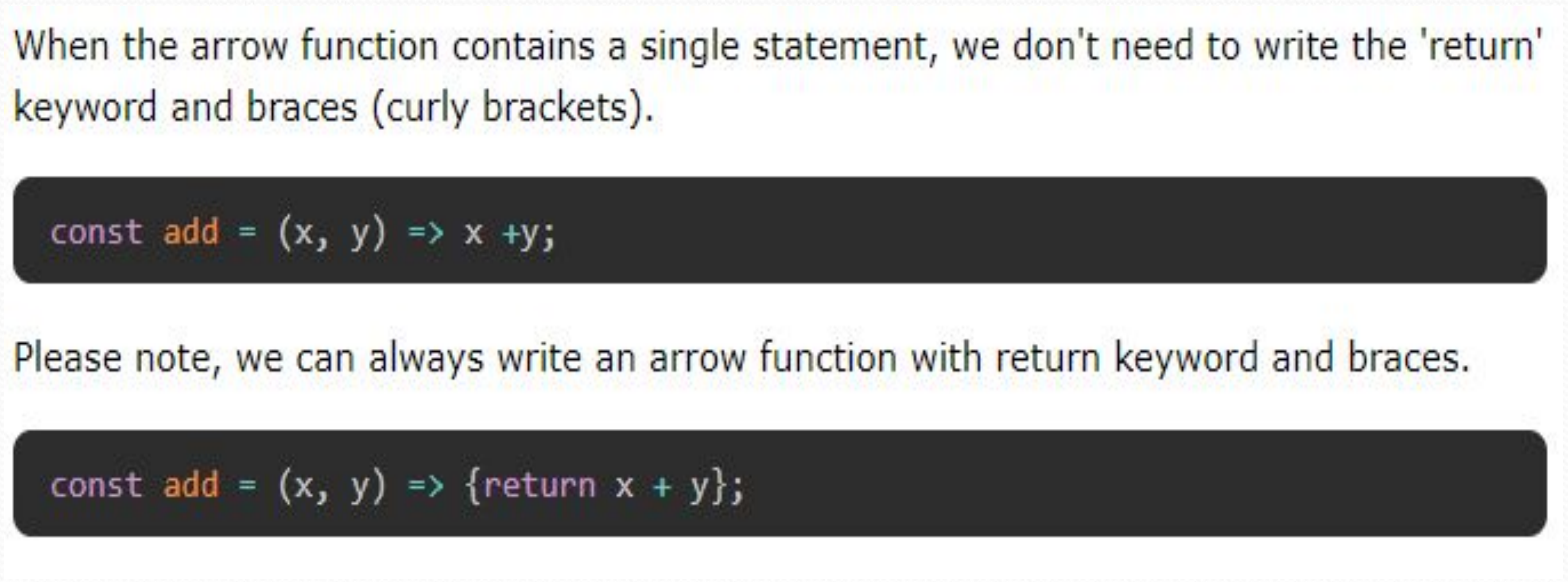
```
const varName = function(parameters) {
    // function body
};
```

The above function expression can be written as an arrow function –

```
const varName = (parameters) => {
    // function body
};
```

Here the "function" keyword is removed and after parenthesis we added "=>".

Arrow Functions with Single Statement



When the arrow function contains a single statement, we don't need to write the 'return' keyword and braces (curly brackets).

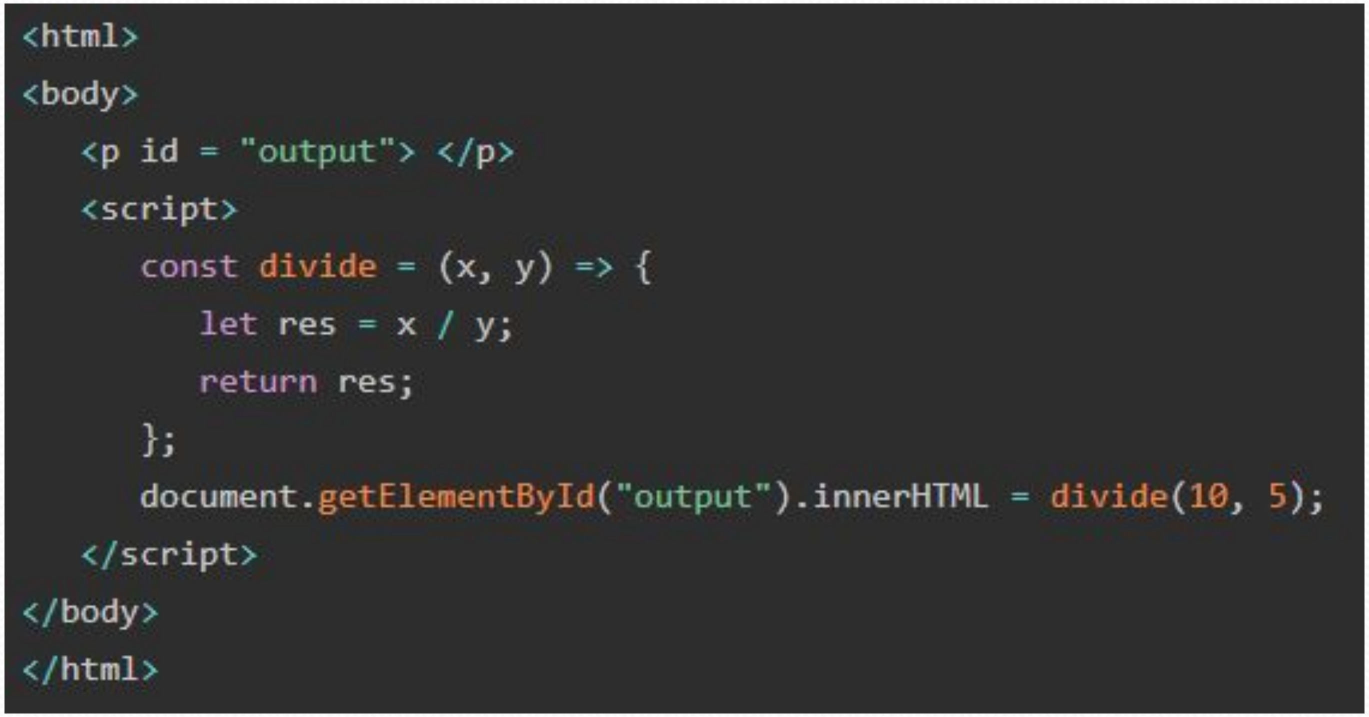
```
const add = (x, y) => x +y;
```

Please note, we can always write an arrow function with return keyword and braces.

```
const add = (x, y) => {return x + y};
```

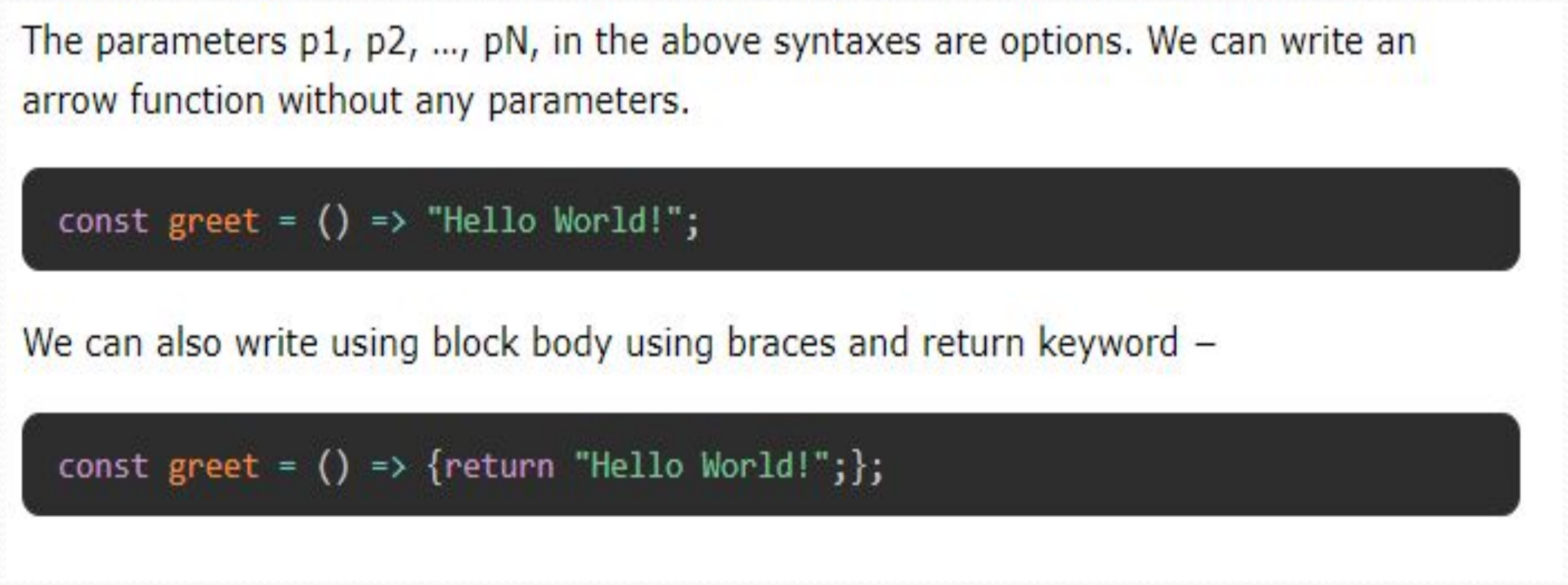
Arrow Functions with Multiple Statement

When the function body contains *multiple statements*, we should always use the 'return' statement to return a value. Also we should use the curly brackets.



```
<html>
<body>
   <p id = "output"> </p>
   <script>
      const divide = (x, y) => {
         let res = x / y;
         return res;
      };
      document.getElementById("output").innerHTML = divide(10, 5);
   </script>
</body>
</html>
```

Arrow Functions without Parameters



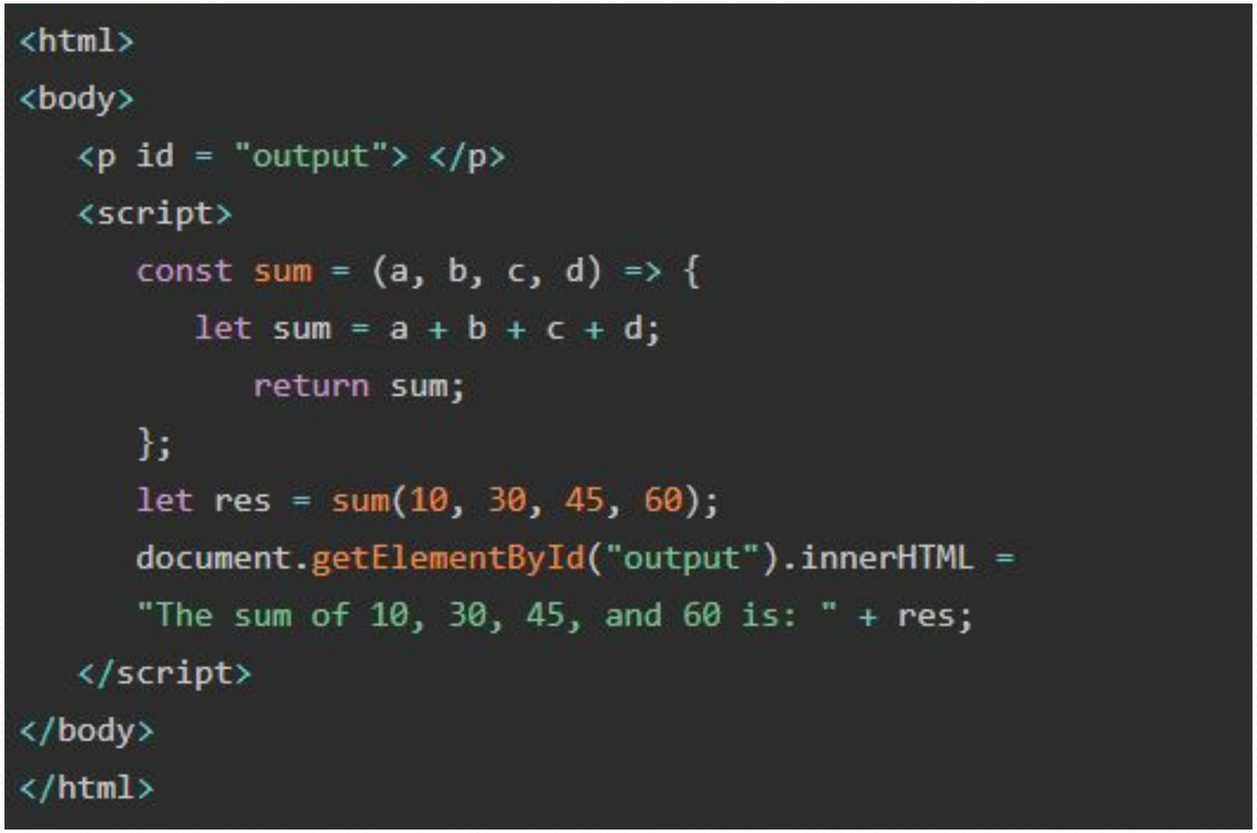
The parameters p1, p2, …, pN, in the above syntaxes are options. We can write an arrow function without any parameters.

```
const greet = () => "Hello World!";
```

We can also write using block body using braces and return keyword –

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

Arrow Functions with Multiple Parameters



```
<html>
<body>
   <p id = "output"> </p>
   <script>
      const sum = (a, b, c, d) => {
         let sum = a + b + c + d;
            return sum;
      };
      let res = sum(10, 30, 45, 60);
      document.getElementById("output").innerHTML =
      "The sum of 10, 30, 45, and 60 is: " + res;
   </script>
</body>
</html>
```

JavaScript Events

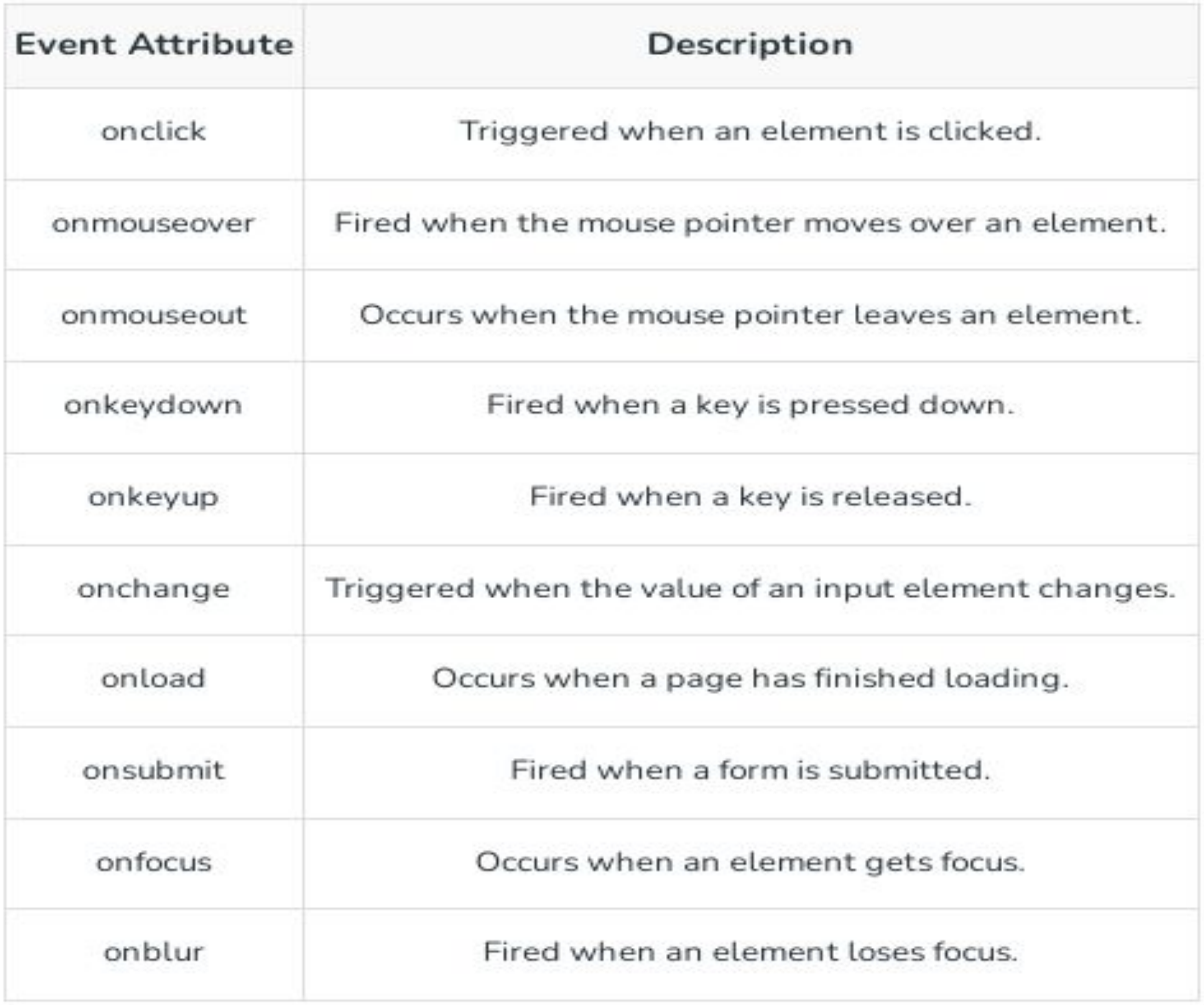
JavaScript Events

- **JavaScript Events** are **actions or occurrences** that happen in the browser. They can be triggered by various user interactions or by the browser itself.

- Common events include mouse clicks, keyboard presses, page loads, and form submissions. Event handlers are JavaScript functions that respond to these events, allowing developers to create interactive web applications.

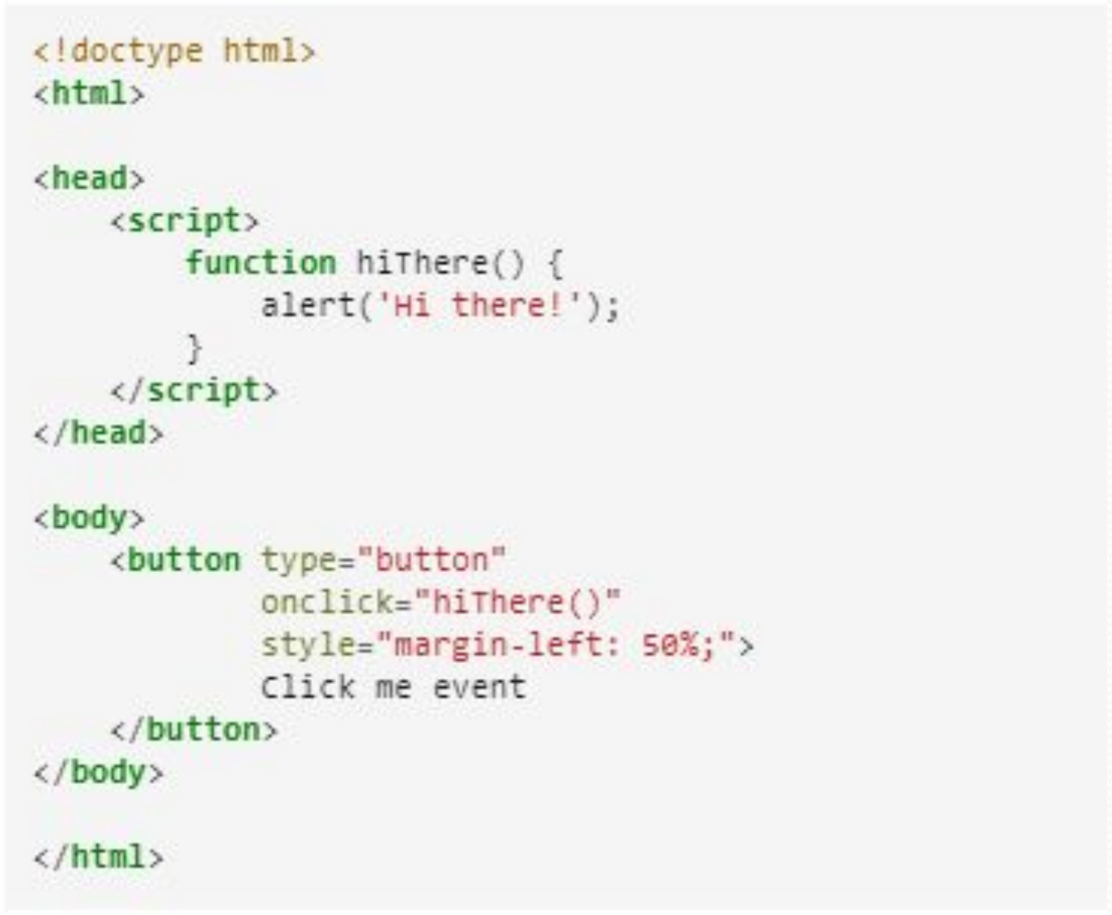
- **Syntax:**
  - **<HTML-element Event-Type = "Action to be performed">**

Common JavaScript Events



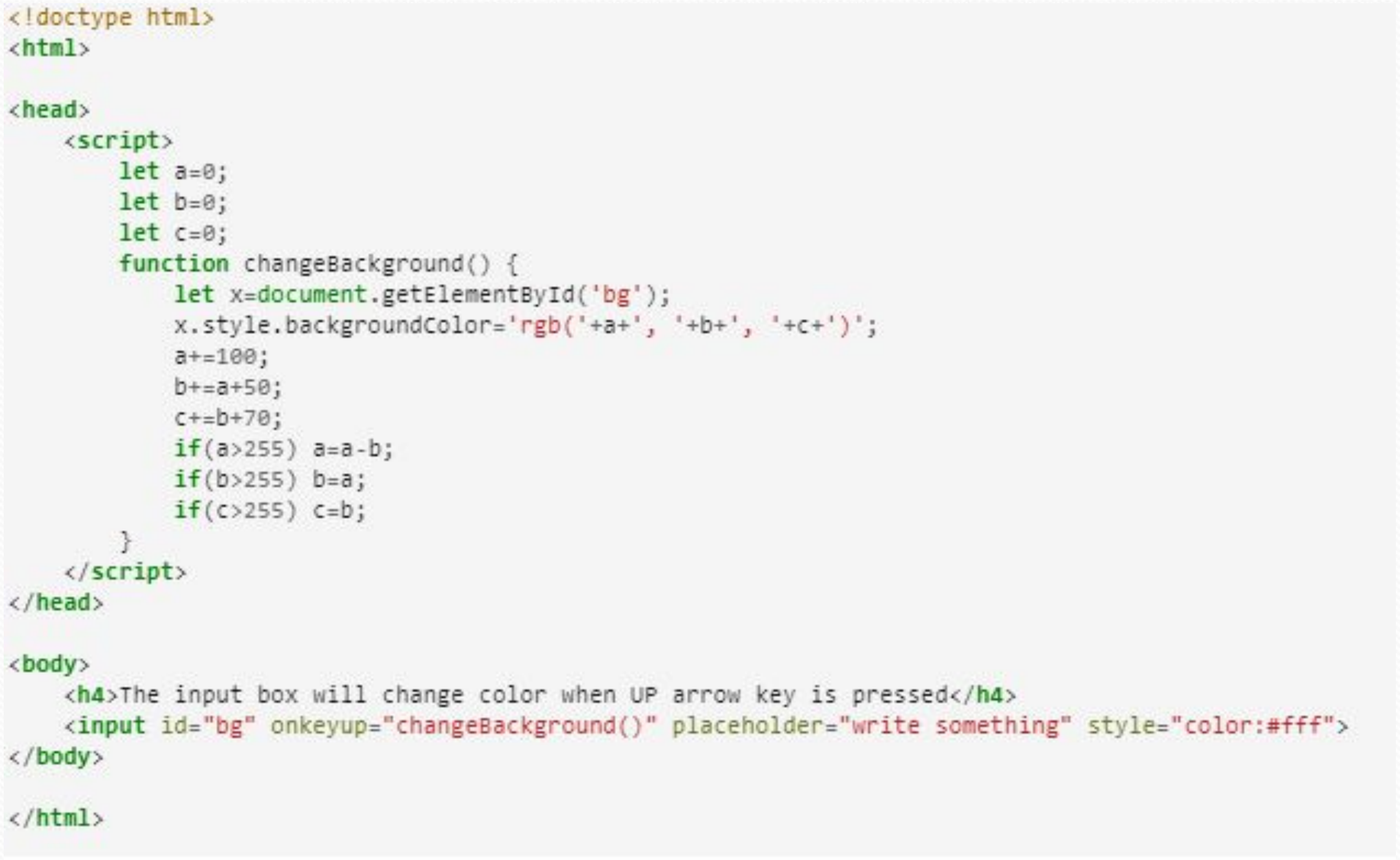
Event Attribute	Description
onclick	Triggered when an element is clicked.
onmouseover	Fired when the mouse pointer moves over an element.
onmouseout	Occurs when the mouse pointer leaves an element.
onkeydown	Fired when a key is pressed down.
onkeyup	Fired when a key is released.
onchange	Triggered when the value of an input element changes.
onload	Occurs when a page has finished loading.
onsubmit	Fired when a form is submitted.
onfocus	Occurs when an element gets focus.
onblur	Fired when an element loses focus.

JavaScript Event - Example



```
<!doctype html>
<html>

<head>
    <script>
        function hiThere() {
            alert('Hi there!');
        }
    </script>
</head>

<body>
    <button type="button"
            onclick="hiThere()"
            style="margin-left: 50%;">
            Click me event
    </button>
</body>

</html>
```

JavaScript Event - Example

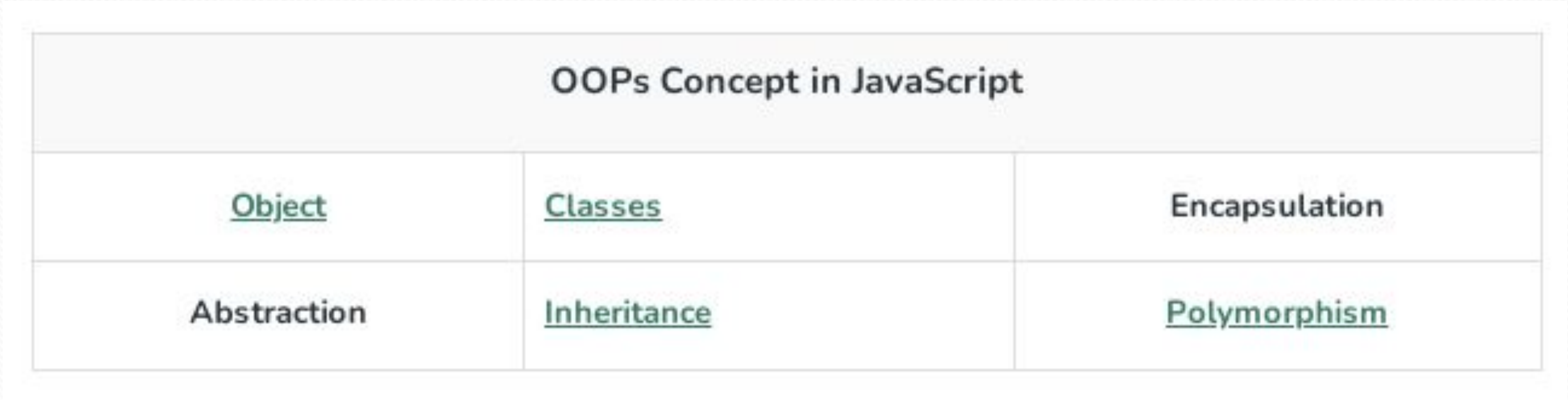


```
<!doctype html>
<html>

<head>
    <script>
        let a=0;
        let b=0;
        let c=0;
        function changeBackground() {
            let x=document.getElementById('bg');
            x.style.backgroundColor='rgb('+a+', '+b+', '+c+')';
            a+=100;
            b+=a+50;
            c+=b+70;
            if(a>255) a=a-b;
            if(b>255) b=a;
            if(c>255) c=b;
        }
    </script>
</head>

<body>
    <h4>The input box will change color when UP arrow key is pressed</h4>
    <input id="bg" onkeyup="changeBackground()" placeholder="write something" style="color:#fff">
</body>

</html>
```

Object Based JavaScript

Object Based JavaScript

As JavaScript is widely used in Web Development, in this topic we will explore some of the **Object Based** mechanisms supported by **JavaScript**.



OOPs Concept in JavaScript		
Object	Classes	Encapsulation
Abstraction	Inheritance	Polymorphism

Object

- An Object is a **unique** entity that contains **properties** and **methods**.
- For example "a car" is a real-life Object, which has some characteristics like color, type, model, and horsepower and performs certain actions like driving.
- The characteristics of an Object are called Properties in Object-Oriented Programming and the actions are called methods.
- An Object is an **instance** of a class.
- Objects are everywhere in JavaScript, almost every element is an Object whether it is a function, array, or string.

Object

- The object can be created in two ways in JavaScript:
  - **Object Literal**
  - **Object Constructor**

Creating Objects in JavaScript

Using Literals

```
var student = { name: "Chris", age: 21, studies: "Computer Science", };
document.getElementById("demo").innerHTML = student.name + " of
   the age " + student.age + " studies " + student.studies;
```

Using Object

```
var student = new Object();
  student.name = "Chris", student.age=21, student.studies = "Computer
   Science";
 document.getElementById("demo").innerHTML = student.name + " of
   the age " + student.age + " studies " + student.studies;
```

Class

- We know that objects hold the data and the functions to manipulate the data. However, the two can be bound together in a user-defined data type with the help of classes. Any number of objects can be created in a class. Each object is associated with the data of type class. A class is therefore a collection of objects of similar types.

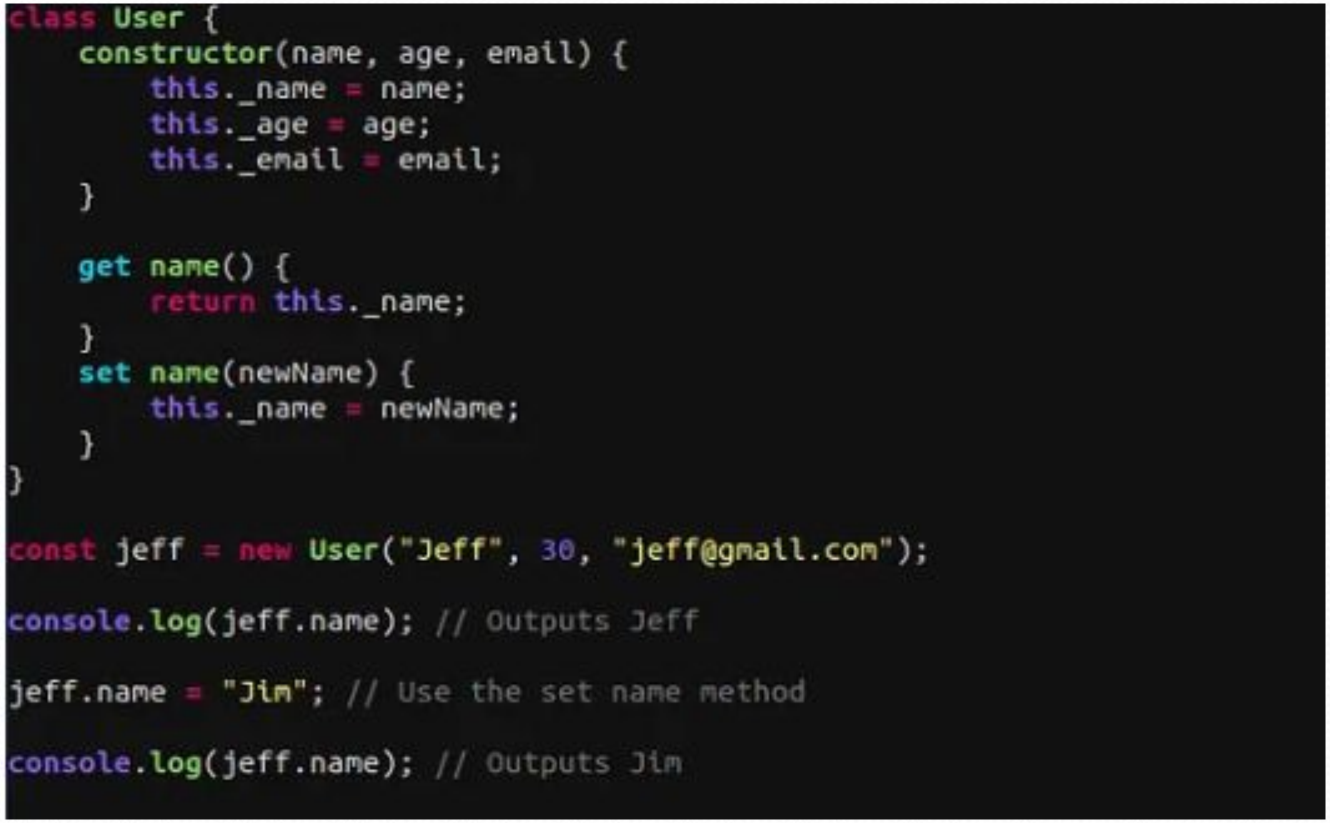
- For example, consider the class "Fruits". We can create multiple objects for this class -
  - Fruit Mango;

- This will create an object mango belonging to the class fruit.

Encapsulation

- One of the core concepts of OOP is **encapsulation.**
- An important part of **encapsulation** is that data (object properties) should not be directly accessed or modified from outside the object.
- To access or modify a property we would use a **getter** (access) or a **setter** (modify), which are specific methods we define in our class.

Encapsulation implementation in JavaScript

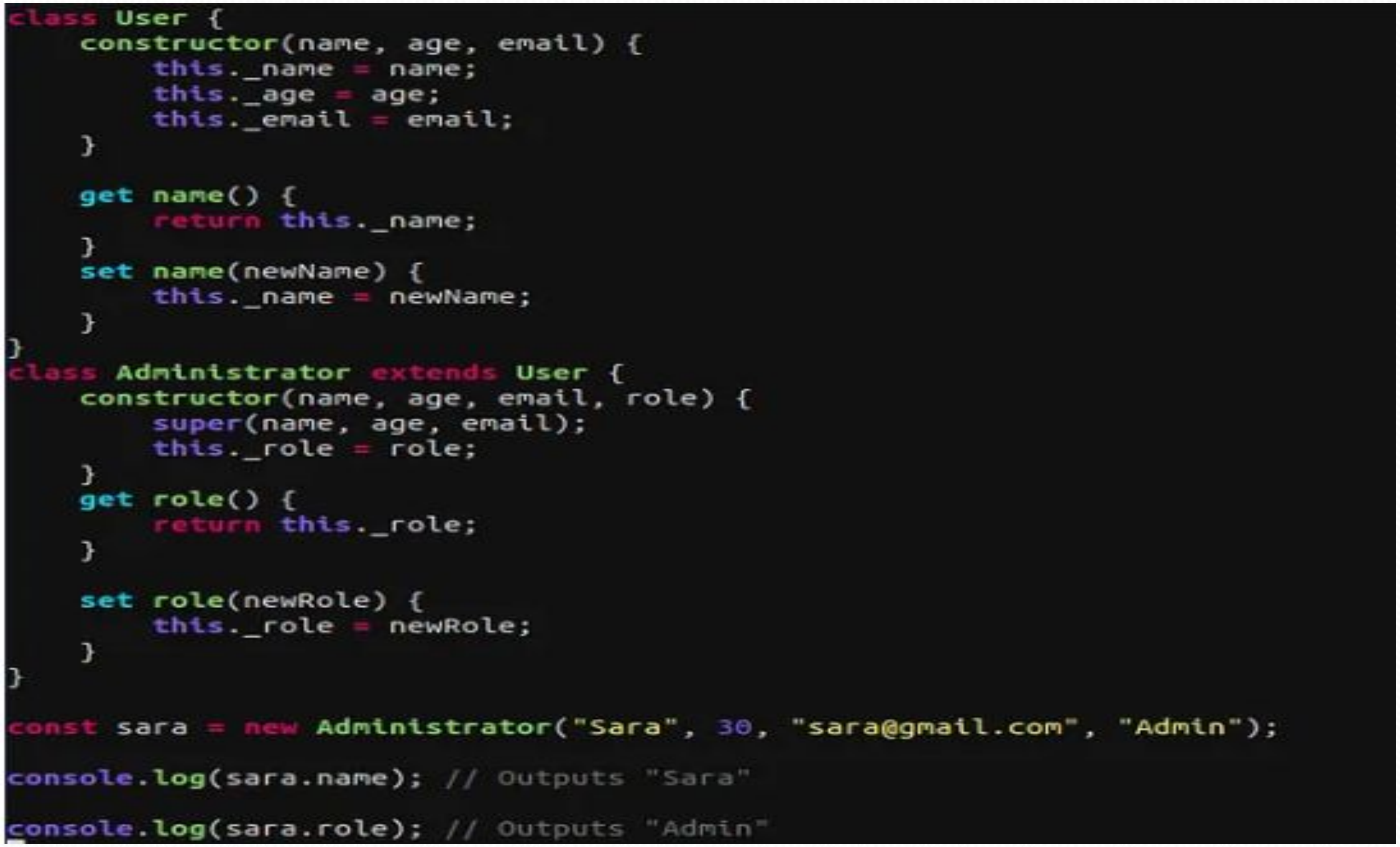


```
class User {
    constructor(name, age, email) {
        this._name = name;
        this._age = age;
        this._email = email;
    }

    get name() {
        return this._name;
    }
    set name(newName) {
        this._name = newName;
    }
}

const jeff = new User("Jeff", 30, "jeff@gmail.com");

console.log(jeff.name); // Outputs Jeff

jeff.name = "Jim"; // Use the set name method

console.log(jeff.name); // Outputs Jim
```

Inheritance

- Classes can also inherit from other classes.
- The class being inherited from is called the **parent**, and the class inheriting from the parent is called the **child**.
- In our example, another class, let’s say *Administrator*, can inherit the properties and methods of the *User* class:

Inheritance in JavaScript



```
class User {
    constructor(name, age, email) {
        this._name = name;
        this._age = age;
        this._email = email;
    }

    get name() {
        return this._name;
    }
    set name(newName) {
        this._name = newName;
    }
}
class Administrator extends User {
    constructor(name, age, email, role) {
        super(name, age, email);
        this._role = role;
    }
    get role() {
        return this._role;
    }

    set role(newRole) {
        this._role = newRole;
    }
}

const sara = new Administrator("Sara", 30, "sara@gmail.com", "Admin");

console.log(sara.name); // Outputs "Sara"

console.log(sara.role); // Outputs "Admin"
```

Thank You

Valan Arasu M