Transcript

Module 5: Introduction to JavaScript

Video 1: Overview

JavaScript is widely recognised as one of the most commonly used programming languages of all time. It is a highly versatile programming language that can be used to build modern, full-stack software applications. Indeed, many popular web application development frameworks, such as React and Angular, are based on JavaScript. With the arrival of JavaScript runtime environments, such as Node.js, JavaScript can also be used to build highly scalable and dependable backend.

This module introduces you to everything that you need to get started writing your first JavaScript. You will learn basic programming methodology with JavaScript and how to manipulate the Document Object Model or DOM with JavaScript. These essential knowledge and skills will provide a strong foundation for you to build high-quality web applications.

Video 2: Introduction to JavaScript

JavaScript is a programming language and a core technology of the web alongside Hypertext Markup Language, or HTML, and Cascading Style Sheets, or CSS. JavaScript is most famously known for building dynamic content in websites. JavaScript is an interpreted language that does not require the compilation of codes into an executable. Instructions written in JavaScript are directly interpreted by a piece of special software known as an interpreter.

Beyond this, JavaScript exhibits various other important technical characteristics. JavaScript uses dynamic typing and the interpreter assigns each variable a data type during runtime based on its initial or current value. JavaScript is a multi-paradigm programming language that supports the imperative, object-oriented and functional programming paradigms. As an object-oriented language, it supports the use of objects without the need to explicitly declare classes. As a functional programming language, JavaScript allows programmers to work with first-class functions. Last but not least, JavaScript is single-threaded.

In the past, JavaScript can only run within a web browser, and this era is sometime known as classic JavaScript. Web browser also includes micro browsers in mobile devices and web view control. A web view control allows interactivity via a web interface to be added to a native mobile application. Classic JavaScript is typically used for developing frontend applications. With the arrival of non-browser runtime environments such as Node. js, JavaScript can now run outside a web browser. This era is sometimes known as Modern JavaScript. Modern JavaScript can be used to develop backend applications in addition to frontend applications.

In other words, JavaScript can now be used for full-stack software engineering. Modern JavaScript also provides support for multiple threads through the use of web workers to deliver enhanced performance. As a high-level programming language, JavaScript supports the use of variables, which are symbolic names to reference data, symbolic instructions that are highly readable to programmers, as well as control flow statements.

JavaScript is typically written in a file with the .js extension that contains a list of instructions to be executed. These instructions are also known as statements. JavaScript statements are terminated by semicolons and may compose of values, operators, expressions, keywords and comments.

As in most programming languages, JavaScript supports two types of values. Fixed values are known as literals and variable values are called variables. Operators are used to manipulate these values and to perform computation. You can think of a JavaScript expression as consisting of a combination of literal values, variables and operators that lead to the evaluation of a result. JavaScript keywords or reserve words are used to identify actions to be performed, such as the declaration of variables and implementation of control flows.

Comments are used to annotate the code and helps you to recollect your thought process during initial coding. Comments are ignored and will not be executed. JavaScript supports both single line commenting with the double forward slash notation and multiline commenting with the forward slash asterisk and asterisk slash notation.

When a piece of JavaScript is running within a non-browser environment, such as Node.js, you can print out diagnostic messages to the console and utilise the debugging tool provided by an Integrated Development Environment, or IDE, to troubleshoot errors. In contrast, when running a piece of JavaScript in a web browser, you will need to utilise the developer tools provided by the browser. Most modern web browsers provide tools for you to view the source code, the DOM, as well as the console. You can also use the developer tools to emulate the running of JavaScript in mobile devices. This is especially useful when you are creating a responsive web application or a cross-platform hybrid mobile application.

In the subsequent videos of this module, you will gradually learn more about programming methodology with JavaScript and manipulating the DOM with JavaScript.

Video 3: Variables Data Types and Operators in JavaScript

Variables are symbolic names that we use to reference data in JavaScript. These data included those input by users, intermediate processing values and final results to be output to user. In JavaScript, the official term for variable name is identifier. An identifier must satisfy four rules.

First, an identifier can only contain letters, digits, underscores and dollar signs. No other special character is allowed. Second, an identifier must begin with a letter, underscore or dollar sign. This means that you cannot start an identifier with a digit. Next, an identifier is case sensitive. Thus, you should avoid the use of casing to differentiate variables. It can be very confusing. Fourth and lastly, reserved words cannot be used as identifiers. To declare a variable in JavaScript, you use the var reserved word followed by an identifier. The identifier must conform to the rules mentioned earlier.

Observe that unlike some other programming languages, such as C or Java, it is not necessary to declare the data type of the variable. This is because variables in JavaScript use dynamic typing as discussed previously. However, JavaScript can handle many types of data, including numbers, strings and Boolean values. Thus, you should be cognisant of a variable's data type based on its initial or current value as the data type will affect the evaluation of operators.

For example, the plus character represents the arithmetic addition operator if both operands are numbers but becomes a string concatenation operator if either one or both operands are strings. The data type of a variable will also change as new values are assigned to the variables. Do also take note that JavaScript does not differentiate between integer and floating-point values, and both are simply considered as numbers. Finally, strings in JavaScript can be quoted with either single or double quotes, but you must be consistent between the opening and closing quotes. Finally, you can change the data type of a value by performing an explicit typecasting operation using the required data type function.

For example, Number('8') will return the numerical value of 8 instead of the string '8'. The typeof operator can be used to view the current data type of a variable. Observe that in this sample code, the data type of variable x changes throughout execution from undefined to number and finally string. In JavaScript, the assignment operator is denoted by a single equal character. It is used to assign a literal value, or a value of another previously declared variable, to a variable. In other words, the left-hand side of the assignment operator must be a variable. The right-hand side can be any JavaScript expression consisting of literal value, variable and operator.

A variable can be assigned special values such as empty string, null or undefined as required. Although not compulsory, it is generally a good practice to initialise a variable to a default value during declaration with the assignment operator, for example, var num = 0. JavaScript provides a set of seven arithmetic operators to perform basic operations such as addition, multiplication and modulo. In particular, the exponentiation operator returns the result of raising the first operand to the power of the second operand without the need to use a math library function.

JavaScript also supports the increment and decrement operators, both of which may be used in either pre mode or post mode. For example, the pre-increment and pre-decrement operators affect both the value of the operand and the returned value. But the post-increment and post-decrement operators will not affect the return value, that is, the returned value remains unchanged as the original operand value. You can also use a set of assignment with arithmetic operators as shorthand notations to use the arithmetic operators concurrently as assignment operators. For example, x += 8 will add 8 to the value of variable x and assign the result back to x.

As in all other programming languages, JavaScript offers a set of binary relational operators that allow you to compare the left operand with the right operand and returns the result as a boolean value. Observe that there is a triple equals operator that compares whether the two operands have equal value and equal type. For example, 8 === 8. 0 will return true since both operands are considered as numbers.

Finally, you can also use the logical operators 'and', 'or' and 'not' to construct more complex Boolean expressions. These operators follow the usual logic rules as depicted in the Boolean truth table. By combining relational and logical operators, you can write more complex Boolean expressions in your JavaScript. For example, the Boolean expression num1 greater equals num2 and num1 greater equals num3 can be used to determine whether num1 is the largest of the three numbers.

To wrap up this video, let's briefly discuss operator precedence in JavaScript. Operator precedence determines the order of evaluation of operators in a JavaScript expression. Operators with higher precedence are evaluated first, in a left-to-right order. For example, the multiplication and division operators have higher precedence than the addition and subtraction operators. The expression 100 plus 50 multiplied by 3 will thus evaluate to 250. To override the default operator precedence, you can use parentheses. The expression parenthesis 100 plus 50 multiplied by 3 will evaluate to 450. Now that you have a good understanding of variables and operators, let's move on to learn how to write control flow statements in JavaScript starting with conditional control flow.

Video 4: Conditional Control Flow with JavaScript

In this video, you will learn how to write conditional control flow statements to make decisions in JavaScript. JavaScript supports two types of conditional control flow statements, the if statement and the switch statement. The if statement is used to determine which subsequent statements to execute using a general Boolean expression. In comparison, the switch statement uses a variable containing a discrete value, such as integer and string to make decisions.

In JavaScript, control flow statements are written using the bracket convention as in most other programming languages. The if statement starts with the if reserved word and is followed by the boolean expression enclosed within a pair of parentheses. Next comes the body of the if statement, which is enclosed within a pair of curly brackets.

The boolean expression may be written with three different approaches. First, a Boolean expression may simply be a Boolean literal or variable value. Note that in JavaScript, only the numerical value of 0 and the Boolean value of false evaluate to false. All other values are evaluated to true. Second, a relational operator can be used to write a simple comparison expression, which evaluates to a Boolean value. Third and lastly, relational operators may be combined with logical operators to write a more complex Boolean expression that ultimately evaluates to either true or false.

Returning back to the if statement, observe that it performs an optional evaluation to determine whether to execute the statements enclosed within its body. The if statement can also be written with an else clause if you need alternative evaluation. In this case, the body of the else clause will be executed instead when the Boolean expression evaluates to false.

Suppose the decision-making process involves more than two alternatives, you can write an if-else-if statement that contains as many else if clauses as required. If a default action is required, when all boolean expressions evaluate to false, you can end the statement with an else clause. In this if-else-if example, the JavaScript will print out grade B if the user enters a score of 75 marks. If statements can be nested within each other to handle more complex decision-making scenarios as required.

The JavaScript switch statement can be used to evaluate the value of a discrete variable such as integer or string, and then execute the corresponding body of the matching case clause. It is important to end the body of each case clause with a break reserve word. Otherwise, execution will fall through to the body of the next case clause.

In this example, you should observe exactly one statement being printed out. That is, if user enters 'A', 'B' or 'C', the respective option will be printed out. Otherwise, an error message will be printed instead. You would probably have noticed that the switch statement is not essential and can be replaced by an equivalent if-else-if statement. This wraps up our discussion on conditional control flow.

Video 5: Iterative Control Flow with JavaScript

In this video, you will learn how to write iterative control flow statements to perform certain actions repeatedly. More specifically, iterative control flows are used to execute a block of JavaScript statements repeatedly. JavaScript supports three main types of iterative control flow statements. The while loop and do-while loop are used in conjunction with a Boolean expression to repeatedly execute a block of statements.

In contrast, a for-loop iterates through a block of codes multiple times according to the value of a counting variable. As in all other programming languages, the while loop is the most generic type of iterative control flow in JavaScript. Whatever that can be performed using other iterative control flow statements can be performed with the while loop.

Similar to the conditional control flow statements, iterative control flow statements in JavaScript are also written using the bracket convention. A typical loop starts with the corresponding reserved word and is followed by the loop expression enclosed within a pair of parentheses. Next comes the body of the loop, which is enclosed within a pair of curly brackets.

Let's examine the while loop for a start. A while loop repeatedly executes the statements in its body as long as the Boolean expression evaluates to true. There are some important implications that you should usefully note. First, if the Boolean expression evaluates to false initially, the loop body will be skipped over altogether. Second, if the Boolean expression always evaluates to true, the loop body will be repeatedly executed infinitely. This is also known as an infinite loop. An infinite loop will cause your JavaScript to become unresponsive.

In this example, the while loop will continue to prompt the user whether to exit until the user has entered an uppercase character 'Y'. Recall that in JavaScript, string values and their comparisons are case sensitive. In this second example, the JavaScript will print out a series of integer numbers starting from the number entered by the user until 10, with each iteration incrementing by 1. If the user enters a number that is greater than 10, the while loop body will be skipped over altogether.

Let's move on to examine a do-while loop. Recall that a while loop may be executed zero, one or multiple times. If the Boolean expression evaluates to false initially, the loop body executes zero time. This is because the loop expression is evaluated first before the loop body. In the case of a do-while loop, the loop body is executed one time initially before the loop expression is evaluated. Thus, the body of a do-while loop is guaranteed to execute at least once. Of course, the do-while loop is not exactly essential since we can always rewrite the Boolean expression of a while loop such that it evaluates to true initially.

Now, let's explore how to write a for-loop in JavaScript, which is quite different from the while loop and do-while loop. The loop expression of a for-loop consists of three different statements. The first statement is executed before the loop body and is used to initialise the value of an integer counting variable. The second statement defines the condition for iterating the loop's body. It is essentially a Boolean expression to compare the value of the integer counting variable. This statement is executed once at the end of each iteration. The third statement is also executed once after each iteration but before the second statement. It is typically used to increment the value of the integer counting variable before the second statement determines whether to continue iterating. Typically, the for-loop body will continue to execute as long as the value of the integer counting variable is less than or equal to a reference value indicated in the second statement.

This example of for-loop resembles the earlier example on while loop. The JavaScript will print out a series of integer numbers starting from the number entered by the user until 10, with each iteration incrementing by 1. Observe the use of the post-increment operator in the third statement of the for-loop expression. Iterating control flow statements can be nested within each other to handle more complex looping scenarios as required. Indeed, a mix of while loop, do-while loop and for-loop can be nested within each other. A typical scenario involves the use of nested for loops involving multiple integer counting variables.

One distinct counting variable is declared inside each for-loop. An inner for-loop will need to use the counting variables of an outer for-loop to perform some computation. This example illustrates the generation of a times table based on an integer number entered by the user.

In the last segment of this video, let's talk about the premature termination of iterative control flows. By default, an iterative control flow terminates once the pertinent loop expression evaluates to false. But in some cases, it might be necessary to terminate the loop prematurely. In JavaScript, the break and continue statements can be used for this purpose. The break statement causes an immediate termination of the current loop iteration and then exits from the enclosing loop. The continue

statement also causes an intermediate termination of the current loop but does not exit from the enclosing loop.

Let's compare an example each of the break and continue statements to illustrate the salient difference. In the case of break, you will only see the output when j is 1 but not 2 and 3 since the inner for loop will exit when j reaches 2. And in comparison, the continue example will print out the output when j is 1 and 3 but just not 2. The next example demonstrates the use of the break statement inside an infinite while loop to force the JavaScript to exit when user enters an upper character 'Y'. This concludes our discussion on iterative control flows in JavaScript.

Video 6: JavaScript Functions and Scope

In this video, I'll explain how to modularise your JavaScript code through the use of functions. A JavaScript function is a block of code that is designed to perform a specific task. By modularising your JavaScript code into functions, it enhances the maintainability of the code and promotes reusability. More specifically, JavaScript functions can be reused at multiple locations within the same JavaScript source file or across multiple different files.

Let's begin with the anatomy of JavaScript functions. A function is literally defined with the function reserve word using the same bracket convention as how we have been writing control flow statements. The function reserve word is followed by the name of the function, which must conform to the same identifier naming rules of variables. The list of input parameters enclosed within a pair of parentheses comes after the name of the function. If a function does not accept any parameter, the content inside the parentheses will be left blank. Finally, the body of the function containing the block of statements to be executed when the function is invoked will be enclosed within a pair of curly brackets.

Observe that you do not need to declare the data type of parameters nor the data type of the return value. This is because JavaScript uses dynamic typing. The advantage is that you can overload a function by passing arguments, the actual values received by the parameters, of different data types and the function can be expected to return a value of the corresponding data type if it is written appropriately. A function is invoked using its name and followed by a pair of parentheses. A pair of empty parentheses is compulsory even if no argument is being passed into the function. In this example, we invoke or call the addition function separately using two integer arguments and two floating-point arguments. Observe that the respective return value is in the correct numerical format albeit with the same data type of number.

Let's focus on function return in greater details. When execution within a JavaScript function reaches a return statement, the function will stop execution. Execution will return to the code immediately after the invoking statement. Functions typically compute a return value, and this return value is then returned to the invoking statement or 'caller'. In the preceding example, the return value of the addition function is assigned to the respective variable on the left-hand side of the assignment operators. But do take note that return value is not mandatory, that is, a function stops execution too when it reaches the end of the function's body even if there is no return statement. In this case, the return value will be the special value of undefined.

Next, let's explore the concept of scope in JavaScript. Scope generally refers to the set of variables that your code has access to. JavaScript uses function scope by default. In other words, the scope of variables changes when execution passes over to a function. Variables that are declared within a function are considered as local variables with a local scope and they can only be accessed within the function. Note that the parameters of functions are considered as local variables.

In this example, carName is a local variable declared inside myFunction. Codes that are outside the function will not be able to access this local variable due to its local scope. Executing this JavaScript will result in a runtime error occurring on the second line of code that carName is not defined. Since local variables are only recognised inside their respective functions, it is possible to declare variables with the same name across different functions. In addition, local variables are automatically created when the function starts execution. They are also automatically deleted when the execution of the function completes.

Variables that are declared outside a function are known as global variables. Global variables have global scope and all JavaScript codes and functions in the same source file can access them. In this example, carName is a global variable declared outside myFunction and thus has global scope. All the three console. log statements will be able to print out the string value of 'Volvo' correctly.

Finally, JavaScript automatically designates a variable as global if you assign a value to a variable that has not been previously declared. In this example, observe that carName has not been declared with the var reserved word inside myFunction. Thus, it becomes a global variable and is accessible outside of myFunction.

This concludes our discussion on JavaScript functions. I encourage you to explore more about functions to modularise your JavaScript code for better maintainability and reusability.

Video 7: Basic Data Structures in JavaScript

In this video, you will learn how to work with some basic data structures in JavaScript such as arrays and objects. Data structures plays an important role in any programming languages as they allow you to store multiple data values simultaneously referenced by a single variable.

In comparison, a variable can only contain a single value. Data structures are thus useful for solving complex computational problems such as sorting an infinite number of integer values in ascending order. Solving such problems with variables is not feasible as we cannot declare enough variables during coding time and writing the control flow statement will be a huge challenge. But with a JavaScript array, these problems can be easily solved.

In JavaScript, an array can be used to store multiple values or elements in a single variable and with each element being easily referenced by a zero-based index number. An array can be created using an array literal enclosed within a pair of square brackets or using the new reserved word.

In this example, nums is declared using the array literal syntax and contains five integer values. The first element is referenced using the index number 0, and thus, the last element is referenced using the index number 4, or n minus 1 where n is the size of the array. Observe that the index number is enclosed within a pair of square brackets after the array variable. Since JavaScript uses dynamic typing, elements stored in an array can be of any data type. More importantly, an array in JavaScript is dynamic and resizable. You can continue to add and remove elements to or from an array after the initial declaration.

A JavaScript array has several properties and methods that make it easy to manipulate its elements. These include the length property, which returns the number of elements in the array, the push method, which adds a new element to the end of the array, the port method, which removes us an element from the end of the array and many others. Here is an example to illustrate the manipulation of an array using these properties and methods. You can iterate through all elements in an array using either a for loop or a for-of loop. The for-of loop is a special control flow that can be used with an iterable object such as an array or string.

Let's move on to examine the second type of data structure in JavaScript, which is known as an object. An object can also contain multiple values but more specifically multiple name-value pairs. Each element in an object consists of a value referenced by a name or text label. An object thus resembles a container for named values enclosed within a pair of curly brackets. To reference a value in an object, you can use the square bracket notation or dotted notation in conjunction with its name.

In this example, the object represents a car with elements describing the characteristics of the car such as type, model and colour. You can iterate through all properties of an object using a for-in loop. Within the for-in loop, each property can then be used to retrieve the corresponding value. Thus far, the arrays and objects that we have created are known as one-dimensional data structures in which each element represents a single value or name-value pair. Data structures in JavaScript can also be of a higher dimension. The most common ones are two-dimensional in nature.

For example, we can create an array of objects to represent multiple cars. A for-of loop can be used to iterate over the array with each element being a car object. Another nested for-in loop is then used to iterate over a car's properties. Data structures such as arrays and objects will come in handy when you learn how to create software applications for your AI solutions in future modules of this course. So don't forget them.

Video 8: Overview of DOM Programming Interface

In an earlier module of this course, you have learned about the Document Object Model or DOM. Recall that DOM is created by the web browser when a webpage or HTML document is loaded. Each element or tag in the HTML document is added to the DOM, which resembles a tree of objects. In this simple example of a DOM, the head element contains the title of the HTML document, and the body element contains a hyperlink and a level 1 heading. You will also recall that the DOM can be modified using the browser developer tool.

In this module, you will progress to learn how to access and change elements of a HTML document using JavaScript to create dynamic HTML content within the browser. This video kickstarts the process by explaining to you more about the DOM programming interface. Other than representing the elements in a HTML document, the DOM also provides a standard object model and programming interface.

Each HTML element is represented as an object with its own properties, methods and events. A property is a value of a HTML element that can be get and set, or equivalently read and modified. A method is an action that can be performed on a HTML element, for example, add or delete. An event allows an HTML element to respond to user's interaction with the element, for example, clicking on an element. The programming interface also provides a collection of methods that can be used to get, change, add or delete objects in the DOM. In summary, with the standard object model and programming interface, JavaScript can be used to create dynamic HTML content within the browser.

Let's examine a simple example of an input tag that is used to render a button. The input tag itself is represented as an object in the DOM. Beyond this, the object also has two properties, namely type and value. The value property specifies the text label of the button. The object also has a click event handler defined that pops up an alert box and then invokes the object's add EventListener method. By invoking the addEventListener method, another event handler for the on mouseover event is added to the object. After clicking the button, the next time you mouseover the button, the button's text label will change from 'Hello World' to 'Mouse Over'.

With this essential knowledge of the DOM's standard object model and programming interface, you are now ready to learn how to use JavaScript to create dynamic HTML content!

Video 9: Basic DOM Manipulation with JavaScript

In this video, you will learn how to work with the Document Object Model, or DOM, using JavaScript. More specifically, you will be using JavaScript to interact with the DOM's Standard Object Model and Programming Interface to create dynamic HTML content within a web browser.

The general programming technique involves just two simple steps. First, you need to obtain a reference to the DOM object that you want to work with. After obtaining a reference to the required DOM object, you can perform various tasks on it. These tasks mainly involves changing properties, adding and deleting other objects to or from the main object, and working with its event handlers. The DOM document object is the owner of all other objects in a webpage or HTML document. In other words, the document object represents the HTML document. Thus, to access the objects of any other HTML elements, you need to start from the document object.

Indeed, the document object provides various methods that you can use to obtain a reference to other objects. The most commonly used method is document. getElementById, which returns a single object associated with the required Id. This is because the ID of objects in a HTML document must be unique. Beyond getElementById, there are various other methods that you can use such as getElementsByTagName and getElementByClassName. These two methods return you multiple objects that match the required tag name or CSS class name.

In this example, we mark up three paragraph tags each with its own unique Id. The second and third paragraph also has their CSS cross property set to 'blueText'. The number of elements returned by getElementById, getElementsByTagName and getElementsByClassName is respectively one, three and two.

Can you explain why? After you have obtained a reference to the required object, you can get or modify its properties. In the previous example, observe that we are getting the innerHTML property of each paragraph object. The innerHTML property contains the textual content placed between the opening and closing paragraph tags. Changing the CSS properties of an object is relatively easy since the CSS style is just another property of an object albeit with sub-properties. Each sub-property corresponds to a particular CSS property, for example, foreground colour, that you can get or modify. In this example, observe how JavaScript is being used to modify the content of the tree paragraphs and to change the foreground colour of the textual content to red colour.

Let's continue to find out how you can add or delete DOM objects. To add a DOM object, you need to create the new object first before appending it to the DOM tree. To delete a DOM object is marginally easier. You simply obtain a reference to the required object and remove it from the DOM tree. In this example, we mark up a HTML table with an empty table body. The table body is then dynamically populated with table rows using JavaScript with data stored in a two-dimensional array of car objects. The insertRow method creates a new table row object and appends it to the table body simultaneously.

If you examine the browser developer tool closely, you will observe that the source view does not contain any table row in the table body, whereas the elements view does show tree table rows. When the source view and the elements view are different, this will indicate the presence of dynamic HTML content created using JavaScript. The document object's addEventListener and removeEventListener methods can be used to attach or remove event handlers to a DOM object. It is important to note that

the addEventListener method can be used to attach multiple handlers to A DOM object for the same event.

If an object and its parent both define an event handler for the same event, the default event propagation is event bubbling, that is, the innermost element's event handler will execute first. The other alternative is event capturing with the outermost element's event handler executing first.

In this example, we use JavaScript to attach two click event handlers to the button object and one click event handler to the parent paragraph object. Observe that neither the paragraph tag nor the input tag contains an onclick attribute. When the button is clicked, its two click event handlers are executed first before the paragraph's click event handler. To reiterate, this is known as event bubbling. Other than obtaining a direct reference to the required DOM object for manipulation, it is possible to navigate the DOM tree to access whichever object you need.

All objects in the DOM tree follow a hierarchical relationship to each other, and this forms the basis to navigate the DOM tree. An object in the DOM tree is also known as a node. As shown in this infographic, the top node is called the root node. Every node has exactly one parent, except the root (which has no parent). A node can have several children. Siblings are nodes with the same parent. Based on the hierarchical relationship of the DOM tree, each node provides several properties that can be used to navigate between other node with JavaScript.

For example, the parentNode property will reference the parent of the current node. The firstChild property will reference the first child node of the current node. There are multiple properties that can return the same relationship node. For example, firstChild is equivalent to childNodes[0]. This example illustrates how to use the childNodes property to navigate around the table body, that is, the table rows. Observe that the DOM's standard object model and programming interface provide a powerful capability to create dynamic HTML content using JavaScript within the web browser. This capability lies at the foundation of all modern client-side web application development frameworks such as React, Angular and Vue.

Video 10: Module Summary

Congratulations on completing this challenging but useful module on JavaScript. You have acquired important knowledge and skills on JavaScript, from programming methodology and data structures to dynamic HTML content creation. These knowledge and skills will come in handy as you progress to learn how to develop software applications as part of the bigger full-stack AI solutioning process. Please continue to expand your horizon on JavaScript, which is easily the most popular programming language of our time.