HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

CSB4301 - WEB TECHNOLOGY

B.Tech – V Semester

UNIT II

Dr. Muthukumaran M
Associate Professor
School of Computing Sciences,
Department of Computer Science and Engineering

INTRODUCTION TO JAVASCRIPT & AJAX

- Java Script: Variables
- Control Statements
- Functions
- Arrays
- Objects, Strings & Manipulations
- Handling Events
- Ajax: The Basics, XMLHttpRequest
- JavaScript and XML

Practical Component:

1. Create a web page with JS to validate age (onblur event) greater than 18 and count the no of words in the description field (onclick event)
2. Create a web page to display the contents of XML in a tabular format

Java Script: Variables

There are 3 ways to declare a JavaScript variable:

Using var

Using let

Using const

JavaScript variables are containers for storing data values.

Variables

Variables are containers for storing data (values).

In this example, x, y, and z, are variables, declared with the var keyword:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>In this example, x, y, and z are variables.</p>

<p id="demo"></p>

<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>

</body>
</html>
```

.getElementById()

The getElementById() is a DOM method used to return the element that has the ID attribute with the specified value. This is one of the most common methods in the HTML DOM and is used almost every time we want to manipulate an element on our document. This method returns null if no elements with the specified ID exists.

.innerHTML

The innerHTML property is part of the Document Object Model (DOM) that allows Javascript code to manipulate a website being displayed. Specifically, it allows reading and replacing everything within a given DOM element (HTML tag)

In this example, price1, price2, and total, are variables:

- In programming, just like in algebra, we use variables (like price1) to hold values.
- In programming, just like in algebra, we use variables in expressions (total = price1 + price2).
- From the example above, you can calculate the total to be 11.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p id="demo"></p>

<script>
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
document.getElementById("demo").innerHTML =
"The total is: " + total;
</script>

</body>
</html>
```

JavaScript Identifiers

All JavaScript variables must be identified with unique names.

These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript identifiers are case-sensitive.

The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

JavaScript Data Types

- JavaScript variables can hold numbers like 100 and text values like "John Doe".
- In programming, text values are called text strings.
- JavaScript can handle many types of data, but for now, just think of numbers and strings.
- Strings are written inside double or single quotes. Numbers are written without quotes.
- If you put a number in quotes, it will be treated as a text string.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>Strings are written with quotes.</p>
<p>Numbers are written without quotes.</p>

<p id="demo"></p>

<script>
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';

document.getElementById("demo").innerHTML =
pi + "<br>" + person + "<br>" + answer;
</script>

</body>
</html>
```

JavaScript Variables
Strings are written with quotes.

Numbers are written without quotes.

3.14
John Doe
Yes I am!

Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the var keyword:

```
var carName;
```

After the declaration, the variable has no value (technically it has the value of undefined).

To assign a value to the variable, use the equal sign:

```
carName = "Volvo";
```

You can also assign a value to the variable when you declare it:

```
var carName = "Volvo";
```

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>Create a variable, assign a value to it, and display it:</p>

<p id="demo"></p>

<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

JavaScript Variables

Create a variable, assign a value to it, and display it:

Volvo

One Statement, Many Variables

You can declare many variables in one statement.
Start the statement with var and separate the variables by comma:

```
var person = "John Doe", carName = "Volvo", price = 200;
```

OR

```
var person = "John Doe",
carName = "Volvo",
price = 200;
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>You can declare many variables in one statement.</p>

<p id="demo"></p>

<script>
var person = "John Doe", carName = "Volvo", price = 200;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value undefined.

The variable carName will have the value undefined after the execution of this statement:

```
var carName;
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>A variable declared without a value will have the value undefined.</p>

<p id="demo"></p>

<script>
var carName;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

```
var carName = "Volvo";
var carName;
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>If you re-declare a JavaScript variable, it will not lose its value.</p>

<p id="demo"></p>

<script>
var carName = "Volvo";
var carName;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

JavaScript Dollar Sign $

Remember that JavaScript identifiers (names) must begin with:

A letter (A-Z or a-z)
A dollar sign ($)
Or an underscore (_)
Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:

```
var $$$ = "Hello World";
var $ = 2;
var $myMoney = 5;
```

JavaScript Underscore (_)

Since JavaScript treats underscore as a letter, identifiers containing _ are valid variable names:

```
var _lastName = "Johnson";
var _x = 2;
var _100 = 5;
```

JavaScript Let

The let keyword was introduced in ES6 (2015).

- Variables defined with let cannot be Redeclared.
- Variables defined with let must be Declared before use.
- Variables defined with let have Block Scope.

Cannot be Redeclared

Variables defined with let cannot be redeclared.

You cannot accidentally redeclare a variable.

With let you can not do this:

```
let x = "John Doe";

let x = 0;

// SyntaxError: 'x' has already been declared
```

Block Scope

Before ES6 (2015), JavaScript had only Global Scope and Function Scope.

ES6 introduced two important new JavaScript keywords: let and const.

These two keywords provide Block Scope in JavaScript.

Variables declared inside a { } block cannot be accessed from outside the block:

```
{
  let x = 2;
}
// x can NOT be used here
```

Variables declared with the var keyword can NOT have block scope.

Variables declared inside a { } block can be accessed from outside the block.

```
{
  var x = 2;
}
// x CAN be used here
```

Redeclaring Variables

Redeclaring a variable using the var keyword can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

```
var x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
```

```
<!DOCTYPE html>
<html>
<body>

<h2>Redeclaring a Variable Using var</h2>

<p id="demo"></p>

<script>
var  x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Redeclaring a variable using the let keyword can solve this problem.

Redeclaring a variable inside a block will not redeclare the variable outside the block:

```
let x = 10;
// Here x is 10

{
let x = 2;
// Here x is 2
}

// Here x is 10
```

```
<!DOCTYPE html>
<html>
<body>

<h2>Redeclaring a Variable Using let</h2>

<p id="demo"></p>

<script>
let  x = 10;
// Here x is 10

{
  let x = 2;
  // Here x is 2
}

// Here x is 10
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Const

- The const keyword was introduced in ES6 (2015).
- Variables defined with const cannot be Redeclared.
- Variables defined with const cannot be Reassigned.
- Variables defined with const have Block Scope.

Cannot be Reassigned

A const variable cannot be reassigned:

```
const PI = 3.141592653589793;
PI = 3.14;      // This will give an error
PI = PI + 10;   // This will also give an error
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript const</h2>

<p id="demo"></p>

<script>
try {
  const PI = 3.141592653589793;
  PI = 3.14;
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
</script>

</body>
</html>
```

Must be Assigned

JavaScript const variables must be assigned a value when they are declared:

Correct
```
const PI = 3.14159265359;
```

Incorrect
```
const PI;
PI = 3.14159265359;
```

When to use JavaScript const?

As a general rule, always declare a variables with const unless you know that the value will change.

Always use const when you declare:

A new Array
A new Object
A new Function
A new RegExp

Control Statements

JavaScript Statement Identifiers

JavaScript statements often start with a statement identifier to identify the JavaScript action to be performed.

Statement identifiers are reserved words and cannot be used as variable names (or any other things).

The following table lists all JavaScript statement identifiers:

Statement	Description
break	Exits a switch or a loop
const	Declares a variable with a constant value
class	Declares a class
continue	Breaks one iteration (in the loop) if a specified condition occurs, and continues with the next iteration in the loop
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements and repeats the block while a condition is true
for	Loops through a block of code a number of times
for ... in	Loops through the properties of an object
for ... of	Loops through the values of an iterable object

function	Declares a function
if ... else ... else if	Marks a block of statements to be executed depending on a condition
let	Declares a variable inside brackets {} scope
return	Stops the execution of a function and returns a value from that function
switch	Marks a block of statements to be executed depending on different cases
throw	Throws (generates) an error
try ... catch ... finally	Marks the block of statements to be executed when an error occurs in a try block, and implements error handling
var	Declares a variable
while	Marks a block of statements to be executed while a condition is true

JavaScript break Statement

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to do a loop with a break. The loop is supposed to output the numbers 0 to 4, but the break statement exits the loop when the variable i is equal to "3".</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>
```

```
<script>
function myFunction() {
  var text = "";
  var i;
  for (i = 0; i < 5; i++) {
    if (i === 3) {
      break;
    }
    text += "The number is " + i + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

In this example we use a for loop together with the break statement.
Loop through a block of code, but exit the loop when the variable i is equal to "3":

Definition and Usage

The break statement exits a switch statement or a loop (for, for ... in, while, do ... while).

When the break statement is used with a switch statement, it breaks out of the switch block. This will stop the execution of more execution of code and/or case testing inside the block.

When the break statement is used in a loop, it breaks the loop and continues executing the code after the loop (if any).

The break statement can also be used with an optional label reference, to "jump out" of any JavaScript code block (see "More Examples" below).

Note: Without a label reference, the break statement can only be used inside a loop or a switch.

Syntax

```
break;
```

Using the optional label reference:

```
break labelname;
```

JavaScript if/else Statement

Definition and Usage
The if/else statement executes a block of code if a specified condition is true. If the condition is false, another block of code can be executed.

The if/else statement is a part of JavaScript's "Conditional" Statements, which are used to perform different actions based on different conditions.

In JavaScript we have the following conditional statements:

Use if to specify a block of code to be executed, if a specified condition is true
Use else to specify a block of code to be executed, if the same condition is false
Use else if to specify a new condition to test, if the first condition is false
Use switch to select one of many blocks of code to be executed

Syntax

The if statement specifies a block of code to be executed if a condition is true:

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

The else statement specifies a block of code to be executed if the condition is false:

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

The else if statement specifies a new condition if the first condition is false:

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

```
<!DOCTYPE html>
<html>
<body>

<p>Clck the button to display "Good day", only if the time is less than
20:00.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var time = new Date().getHours();
  if (time < 20) {
    document.getElementById("demo").innerHTML = "Good day";
  }
}
</script>

</body>
</html>
```

Example

If the time is less than 20:00, create a "Good day" greeting, otherwise "Good evening":

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to display a time-based
greeting.</p>

<button onclick="myFunction()">Try
it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var time = new Date().getHours();
  var greeting;
  if (time < 20) {
    greeting = "Good day";
  } else {
    greeting = "Good evening";
  }
  document.getElementById("demo").innerHTML = greeting;
}
</script>

</body>
</html>
```

JavaScript for Statement

Definition and Usage
The for statement creates a loop that is executed as long as a condition is true.

The loop will continue to run as long as the condition is true. It will only stop when the condition becomes false.

JavaScript supports different kinds of loops:

for - loops through a block of code a number of times
for/in - loops through the properties of an object
for/of - loops through the values of an iterable object
while - loops through a block of code while a specified condition is true
do/while - loops through a block of code once, and then repeats the loop while a specified condition is true

Syntax

```
for (statement 1; statement 2; statement 3) {
  code block to be executed
}
```

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to loop through a block of code five times.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var text = "";
  var i;
  for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to loop through the indices of an array, in ascending order.</p>
<button onclick="myFunction()">Try it</button>

<p id="demo"></p>
<script>
function myFunction() {
  var cars = ["BMW", "Volvo", "Saab", "Ford"];
  var text = "";
  var i;
  for (i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

The For In Loop

The JavaScript for in statement loops through the properties of an Object:

```
for (key in object) {
  // code block to be executed
}
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For In Loop</h2>
<p>The for in statement loops through the properties of an object:</p>

<p id="demo"></p>

<script>
const person = {fname:"John", lname:"Doe", age:25};

let txt = "";
for (let x in person) {
  txt += person[x] + " ";
}

document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```

JavaScript While Loop

he while loop loops through a block of code as long as a specified condition is true.

```
while (condition) {
  // code block to be executed
}
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript While Loop</h2>

<p id="demo"></p>

<script>
let text = "";
let i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

The Do While Loop

The do while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
do {
  // code block to be executed
}
while (condition);
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Do While Loop</h2>

<p id="demo"></p>

<script>
let text = ""
let i = 0;

do {
  text += "<br>The number is " + i;
  i++;
}
while (i < 10);

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Comparing For and While

The loop in this example uses a for loop to collect the car names from the cars array:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
const cars = ["BMW", "Volvo", "Saab", "Ford"];

let i = 0;
let text = "";
for (;cars[i];) {
  text += cars[i] + "<br>";
  i++;
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

The loop in this example uses a while loop to collect the car names from the cars array:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
const cars = ["BMW", "Volvo", "Saab", "Ford"];

let i = 0;
let text = "";
while (cars[i]) {
  text += cars[i] + "<br>";
  i++;
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```