Unit 3. Overview of Java Script

3.1 Overview of Client & Server-Side Scripting
3.2 Structure of Java Script
3.3 Data types and Variables
3.4 Operators (Arithmetic, Assignment, Comparison, Logical and Conditional Operator)
3.5 Control Structure
- 3.5.1 If…Else, switch. Case
- 3.5.2 While, Do…While, For Loop
- 3.5.3 Break, continue

3.6 Java Script String and Events
- 3.6.1 JavaScript Strings types
- 3.6.2 String functions:
  - concat(), split(), indexOf(), lastIndexOf(),substring(), trim(), slice(), replace(), charAt()
- 3.6.3 JavaScript Events :
  - 3.6.3.1 Mouse Events : (click, mouseover, mouseremove, mouse out, mouse up)
  - 3.6.3.2 KeyboardEvents:(keyup,keydown)
  - 3.6.3.3 Form Event: (focus, submit, blur, and change)

3.1 Overview of Client & Server-Side Scripting

Client side scripting:

Web browsers execute client side scripting. It is use when browser has all code. Source code used to transfer from web server to user's computer over internet and run directly on browsers. It is also used for validations and functionality for user events.

It allows for more interactivity. It usually performs several actions without going to user. It cannot be basically used to connect to databases on web server. These scripts cannot access file system that resides at web browser. Pages are altered on basis of user's choice. It can also use to create "cookies" that store data on user's computer.

Server side scripting:

Web servers are used to execute server side scripting. They are basically used to create dynamic pages. It can also access the file system residing at web server. Server-side environment that runs on a scripting language is a web-server.

Scripts can be written in any of a number of server-side scripting language available. It is used to retrieve and generate content for dynamic pages. It is used to require to download plugins. In this load times are generally faster than client-side scripting. When you need to store and retrieve information a database will be used to contain data. It can use huge resources of server. It reduces client-side computation overhead. Server sends pages to request of user/client.

BASIS FOR COMPARISON	SERVER-SIDE SCRIPTING	CLIENT-SIDE SCRIPTING
Basic	Works in the back end which could not be visible at the client end.	Works at the front end and script are visible among the users.
Processing	Requires server interaction.	Does not need interaction with the server.
Languages involved	PHP, ASP.net, Ruby on Rails, ColdFusion, Python, etcetera.	HTML, CSS, JavaScript, etc.

BASIS FOR COMPARISON	SERVER-SIDE SCRIPTING	CLIENT-SIDE SCRIPTING
Affect	Could effectively customize the web pages and provide dynamic websites.	Can reduce the load to the server.
Security	Relatively secure.	Insecure

3.2 Structure of Java Script

<html>

<head>

<meta charset="utf-8">

<title>JavaScript Code Structure</title>

</head>

<body>

<h1>This is a web page.</h1>

This webpage is about as simple as it gets. It will make it easy for you to understand the JavaScript Embedded within it.

JavaScript is:

<up>

deep

dark

delicious

```
</up>
<script>
alert('I am the JavaScript Victim');
</script>
</body>
</html>
```

3.3 Data types and Variables

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.

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.

The Concept of Data Types

In programming, data types are an important concept.

To be able to operate on variables, it is important to know something about the type.

Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

```
let carName1 = "Volvo XC60";   // Using double quotes
let carName2 = 'Volvo XC60';   // Using single quotes
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
let answer1 = "It's alright";      // Single quote inside double quotes
let answer2 = "He is called 'Johnny'";// Single quotes inside double quotes
let answer3 = 'He is called "Johnny"';// Double quotes inside single quotes
```

Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

```
let x1 = 34.00;     // Written with decimals
let x2 = 34;        // Written without decimals
```

Booleans

Booleans can only have two values: true or false.

```
let x = 5;
let y = 5;
let z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

```
const cars = ["Saab", "Volvo", "BMW"];
```

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

Objects

JavaScript objects are written with curly braces {}.

Object properties are written as name:value pairs, separated by commas.

Example

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

The typeof Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable.

The typeof operator returns the type of a variable or an expression:

```
typeof ""             // Returns "string"
typeof "John"         // Returns "string"
typeof "John Doe"     // Returns "string"


typeof 0              // Returns "number"
typeof 314            // Returns "number"
```

```
typeof 3.14          // Returns "number"
typeof (3)           // Returns "number"
typeof (3 + 4)       // Returns "number"
```

Undefined

In JavaScript, a variable without a value, has the value undefined. The type is also undefined.

Example

```
let car;    // Value is undefined, type is undefined
```

Empty Values

An empty value has nothing to do with undefined.

An empty string has both a legal value and a type.

Example

```
let car = "";    // The value is "", the typeof is "string"
```

Variables

There are 3 ways to declare a JavaScript variable:

Using var

Using let

Using const

Variables

Variables are containers for storing data (values).

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

Example

```
var x = 5;
var y = 6;
var z = x + y;
```

```
<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>
```

3.4 Operators (Arithmetic, Assignment, Comparison, Logical and Conditional Operator)

Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation

/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

```
<html>
<body>
<h2>JavaScript Operators</h2>
<p>x = 5, y = 2, calculate z = x + y, and display z:</p>
<p id="demo"></p>
<script>
let x = 5;
let y = 2;
let z = x + y;
document.getElementById("demo").innerHTML = z;
</script>
</body>
</html>
```

Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

The **addition assignment** operator (+=) adds a value to a variable.

```
<html>
<body>
<h2>The += Operator</h2>
```

```
<p id="demo"></p>
<script>
var x = 10;
x += 5;
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than

>=	greater than or equal to
<=	less than or equal to
?	ternary operator

```
<html>
<body>
<h2>JavaScript Comparison</h2>
<p>Assign 5 to x, and display the value of the comparison (x == 8) :
< /p>
<p id="demo"></p>
<script>
let x = 5;
document.getElementById("demo").innerHTML = (x == 8);
</script>
</body>
</html>
```

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that x = 6 and y = 3, the table below explains the logical operators:

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

```
<html>
<body>
<h2>JavaScript Comparison</h2>
<p>The AND operator (&&) returns true if both expressions are true, otherwise it returns false.</p>
<p id="demo"></p>
<script>
let x = 6;
let y = 3;
document.getElementById("demo").innerHTML =
(x< 10 && y > 1) + "<br>" +
(x< 10 && y < 1);
</script>
</body>
</html>
```

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

variablename = (*condition*) ? *value1*:*value2*

Example

```
<html>
<body>
<h2>JavaScript Comparison</h2>
<p>Input your age and click the button:</p>
<input id="age" value="18" />
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
Function myFunction () {
let age = document.getElementById("age").value;
let voteable = (age < 18) ? "Too young":"Old enough";
document.getElementById ("demo").innerHTML = voteable + " to vote.";
}
</script>
</body>
</html>
```

3.5 Control Structure

3.5.1 If…Else, switch..case

- 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

Syntax

```
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
}
```

Example :

```
<html>
<body>
<h2>JavaScript if ..else</h2>
<p>A time-based greeting:</p>
<p id="demo"></p>
<script>
const hour = new Date().getHours();
let greeting;
if (hour < 18) {
greeting = "Good day";
} else {
greeting = "Good evening";
}
document.getElementById("demo").innerHTML = greeting;
</script>
```

</body></html>

Switch… Case

Use the switch statement to select one of many code blocks to be executed.

Syntax

```
Switch (expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Example:

```
<html>
<body>
<h2>JavaScript switch</h2>
<p id="demo"></p>
<script>
let day;
switch (new Date().getDay()) {
case 0:
day = "Sunday";
break;
case 1:
day = "Monday";
break;
```

```
case 2:
day = "Tuesday";
break;
case 3:
day = "Wednesday";
break;
case 4:
day = "Thursday";
break;
case 5:
day = "Friday";
break;
case  6:
day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>
</body>
</html>
```

The break Keyword

When JavaScript reaches a break keyword, it breaks out of the switch block.

This will stop the execution inside the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

The default Keyword

The default keyword specifies the code to run if there is no case match:

3.5.2 While, Do…While, For Loop

While Loop

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

Syntax

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

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

```
<html>

<body>

<h2>JavaScript While Loop</h2>

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

<script>

let text = "";

leti = 0;

while (i< 10) {

text += "<br>The number is " + i;

i++;

}
```

```
document.getElementById("demo").innerHTML = text;

</script>

</body>

</html>
```

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.

Syntax

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

```
<html>
<body>
<h2>JavaScript Do While Loop</h2>
<p id="demo"></p>
<script>
let text = ""
leti = 0;
do {
text += "<br>The number is " + i;
i++;
}
while (i< 10);
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

For Loop

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

```
<html>
<body>
<h2>JavaScript For Loop</h2>
<p id="demo"></p>
<script>
const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];
let text = "";
for (let i = 0; i<cars.length; i++) {
text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

3.5.3 break, continue

The Break Statement

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.

The break statement can also be used to jump out of a loop:

```
<html>
<body>
```

```
<h2>JavaScript Loops</h2>

<p>A loop with a <b>break</b> statement.</p>

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

<script>

let text = "";

for (let i = 0; i< 10; i++) {

if (i === 3) { break; }

text += "The number is " + i + "<br>";

}

document.getElementById("demo").innerHTML = text;

</script>

</body>

</html>
```

The Continue Statement

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

```
<html>

<body>

<h2>JavaScript Loops</h2>

<p>A loop with a<b>continue</b> statement.</p>

<p>A loop which will skip the step where i = 3.</p>

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

```
<script>
let text = "";
for (let i = 0; i< 10; i++) {
if (i === 3) { continue; }
text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

3.6 Java Script String and Events

3.6.1 JavaScript Strings types

Back-Tics Syntax

Template Literals use back-ticks (``) rather than the quotes ("") to define a string:

Example

```
let text = `Hello World!`;
```

Quotes inside Strings

With **template literals**, you can use both single and double quotes inside a string:

Example

```
let text = `He's often called "Johnny"`;
```

Multiline Strings

Template literals allows multiline strings:

Example

```
let text =
`The quick
brown fox
jumps over
the lazy dog`;
```

Interpolation

Template literals provide an easy way to interpolate variables and expressions into strings.

The method is called string interpolation.

The syntax is:

${...}

Variable Substitutions

Template literals allows variables in strings:

Example

```
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName,}, ${firstName,}`;
```

3.6.2 String functions: concat(), split(), indexOf(), lastIndexOf(),substring(), trim(), slice(), replace(), charAt()

1. concat()

concat() joins two or more strings:

Example

```
<html>
<body>
<h2>JavaScript String Methods</h2>
<p id="demo"></p>
<script>
let text1 = "Hello";
let text2 = "World!";
let text3 = text1.concat(" ",text2);
document.getElementById("demo").innerHTML = text3;
</script>
</body>
</html>
```

OUTPUT:
Hello World!

2. split()

The split() method splits a string into an array of substrings, and returns the new array.

If an empty string ("") is used as the separator, the string is split between each character.

The split() method does not change the original string.

```
<html>

<body>

<h2>JavaScript Strings</h2>

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

<script>

letstr = "How are you doing today? I am a Student ";

constmyArr = str.split(" ");

document.getElementById("demo").innerHTML = myArr;
```

```
</script>

</body>

</html>
```

3. indexOf()

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:

Example

```
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The indexOf() method returns the position of the first occurrence of a specified text:</p>

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

<script>
letstr = "Please locate where 'locate' occurs!";
document.getElementById("demo").innerHTML = str.indexOf("locate");
</script>

</body>
</html>
```

OUTPUT: 7

4. lastIndexOf()

The lastIndexOf() method returns the index of the **last** occurrence of a specified text in a string:

Example

```
<html>
```

```
<body>

<h2>JavaScript String Methods</h2>

<p>The lastIndexOf() method returns the position of the last occurrence of a specified text:</p>

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

<script>
letstr = "Please locate where 'locate' occurs!";
document.getElementById("demo").innerHTML = str.lastIndexOf("locate");
</script>

</body>
</html>
```

OUTPUT:21

5. substring()

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes.

Example

```
<html>
<body>
<h2>JavaScript String Methods</h2>
<p id="demo"></p>
<script>
letstr = "Apple, Banana, Kiwi";
document.getElementById("demo").innerHTML = str.substring(14,19);
</script>

</body>
</html>
```

OUTPUT:

6. trim()

The trim() method removes whitespace from both sides of a string:

Example

```
<html>
<body>
<h2>JavaScript String.trim()</h2>
<p>Click the button to alert the string with removed whitespace.</p>
<button onclick="myFunction()">Try it</button>
<script>
functionmyFunction() {
let text = "     Hello World!     ";
alert(text.trim());
}
</script>
</body>
</html>
```

7. slice()

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

Example

```
<html>

<body>

<h2>JavaScript String Methods</h2>
```

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

<script>

letstr = "Apple, Banana, Kiwi";

document.getElementById("demo").innerHTML = str.slice(7,13);

</script>

</body>

</html>
```

Output:

Banana

8. replace()

The replace() method replaces a specified value with another value in a string:

Example

```
<html>
<body>
<h2>JavaScript String Methods</h2>
<button onclick="myFunction()">Try it</button>
<p id="demo">Please visit Microsoft!</p>
<script>
Function myFunction() {
let text = document.getElementById("demo").innerHTML;
document.getElementById("demo").innerHTML =
text.replace("Microsoft","W3Schools");
}
</script>
</body>
</html>
```

9. charAt()

The charAt() method returns the character at a specified index in a string.

The index of the first character is 0, the second character is 1, and so on.

The index of the last character in a string is *string*.length-1, the second last character is *string*.length-2, and so on

```
<html>
<body>
<h2>JavaScript Strings</h2>
<p id="demo"></p>
<script>
letstr = "HELLO WORLD";
document.getElementById("demo").innerHTML = str.charAt(6);
</script>
</body>
</html>
```

OUTPUT:

W

3.6.3 JavaScript Events:

3.6.3.1 Mouse Events: (click, mouseover, mouseremove, mouse out, mouse up)

1. **click**

The onclick event occurs when the user clicks on an element.

```
<html>
```

```
<body>

<p id="demo">Click me. </p>

<script>

document.getElementById("demo").onclick = function() {myFunction()};

function myFunction() {

document.getElementById("demo").innerHTML = "YOU CLICKED ME!";

}

</script>

</body>

</html>
```

2. **mouseover**

The onmouseover event occurs when the mouse pointer is moved onto an element, or onto one of its children.

```
<html>
<body>
<h1 id="demo">Mouse over me</h1>
<script>
document.getElementById("demo").onmouseover = function() {mouseOver()};
document.getElementById("demo").onmouseout = function() {mouseOut()};

functionmouseOver() {
document.getElementById("demo").style.color = "red";
}

functionmouseOut() {
document.getElementById("demo").style.color = "black";
}
</script>
```

```
</body>
</html>
```

3. **mouseremove**

Syntax

document.removeEventListener(*event*, *function*, *useCapture*)

```
<html>
<body>
<button onclick="removeHandler()">Try it</button>
<p id="demo"></p>
<script>
document.addEventListener("mousemove", myFunction);

function myFunction() {
document.getElementById("demo").innerHTML = Math.random();
}

functionremoveHandler() {
document.removeEventListener("mousemove", myFunction);
}
</script>

</body>
</html>
```

4. **mouse out**

The onmouseout event occurs when the mouse pointer is moved out of an element, or out of one of its children.

```
<html>
<body>

<h1 id="demo">Mouse over me</h1>

<script>
```

```
document.getElementById("demo").onmouseover = function()
{mouseOver()};
document.getElementById("demo").onmouseout = function() {mouseOut()};

functionmouseOver() {
document.getElementById("demo").style.color = "red";
}

functionmouseOut() {
document.getElementById("demo").style.color = "black";
}
</script>

</body>
</html>
```

5. **mouse up**

The onmouseup event occurs when a user releases a mouse button over an element.

```
<html>
<body>
<p id="demo">Click me.</p>
<script>
document.getElementById("demo").onmousedown = function()
{mouseDown()};
document.getElementById("demo").onmouseup = function() {mouseUp()};

functionmouseDown() {
document.getElementById("demo").innerHTML = "The mouse button is held
down.";
}

functionmouseUp() {
document.getElementById("demo").innerHTML = "You released the mouse
button.";
}
</script>

</body>
```

</html>

3.6.3.2 Keyboard Events: (key up, key down)

1. **Keyup**

The onkeyup event occurs when the user releases a key (on the keyboard).

```
<html>
<body>

<p>This example uses the HTML DOM to assign an "onkeyup" event to an input element.</p>

<p>Press a key inside the text field and release it to set the text to uppercase.</p>

Enter your name: <input type="text" id="fname">

<script>
document.getElementById("fname").onkeyup = function() {myFunction()};

functionmyFunction() {
var x = document.getElementById("fname");
x.value = x.value.toUpperCase();
}
</script>

</body>
</html>
```

2. **Keydown**

The onkeydown event occurs when the user is pressing a key (on the keyboard).

```
<html>
<body>
```

```
<p>This example uses the HTML DOM to assign an "onkeydown" event to an input element.</p>

<p>Press a key inside the text field to set a red background color.</p>

<input type="text" id="demo">

<script>
document.getElementById("demo").onkeydown = function() {myFunction()};

functionmyFunction() {
document.getElementById("demo").style.backgroundColor = "red";
}
</script>

</body>
</html>
```

3.6.3.3 Form Event: (focus, submit, blur, and change)

1. Focus

The onfocus event occurs when an element gets focus.

The onfocus event is most often used with <input>, <select>, and <a>.

```
<html>
<body>
Enter your name:<input type="text" id="fname">
<script>
document.getElementById("fname").onfocus = function() {myFunction()};
functionmyFunction() {
document.getElementById("fname").style.backgroundColor = "red";
}
```

```
</script>
</body>
</html>
```

2. Submit

The onsubmit event occurs when a form is submitted.

```
<html>
<body>
<form id="myForm" action="/action_page.php">
  Enter name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
<script>
document.getElementById("myForm").onsubmit = function() {myFunction()};
functionmyFunction() {
alert("The form was submitted");
}
</script>
</body>
</html>
```

3. Blur

The onblur event occurs when an object loses focus.

The onblur event is most often used with form validation code (e.g. when the user leaves a form field).

```
<html>
<body>
<input type="text" id="fname">
```

```
<script>
document.getElementById("fname").onblur = function() {myFunction()};
functionmyFunction() {
alert("Input field lost focus.");
}
</script>
</body>
</html>
```

4. change

The onchange event occurs when the value of an element has been changed.

For radiobuttons and checkboxes, the onchange event occurs when the checked state has been changed.

```
<html>
<body>
Enter your name:<input type="text" id="fname">
<script>
document.getElementById("fname").onchange = function() {myFunction()};
functionmyFunction() {
var x = document.getElementById("fname");
x.value = x.value.toUpperCase();
}
</script>
</body>
</html>
```