3. Overview of JavaScript

Overview of client & server side scripting

Client side scripting:

It is the program that runs on the client machine (browser) and deals with the user interface/display and any other processing that can happen on client machine like reading/writing cookies.

1) Interact with temporary storage
2) Make interactive web pages
3) Interact with local storage
4) Sending request for data to server
5) Send request to server
6) work as an interface between server and user

Example: JavaScript, HTML, Ajax, CSS etc.

Server side scripting:

It is the program that runs on server dealing with the generation of content of web page.

1) Querying the database
2) Operations over databases
3) Access/Write a file on server.
4) Interact with other servers.
5) Structure web applications.
6) Process user input. For example if user input is a text in search box, run a search algorithm on data stored on server and send the results.

Example: PHP, C++, Python, JSP etc.

Structure of JavaScript

We can write the JavaScript on the different location of the webpage:

1. Inside <script> tag:

In HTML, JavaScript code is inserted between <script> and </script> tags.

Example:

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```


Prepared By:
Akansha Srivastav – SDJ International College, Surat.

2. Inside<head>:

In this example, a JavaScript function is placed in the <head> section of an HTML page. The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction()
{
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

3. Inside<body>:

In this example, a JavaScript function is placed in the <body> section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

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

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

External JavaScript

Scripts can also be placed in external files:

External file: myScript.js

```
function myFunction()
 {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension **.js**.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

Example

```
<script src="myScript.js"></script>
```

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To display data in javascript:

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.
- Example : document.getElementById("demo").innerHTML = 5 + 6;
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

Data types and Variables

Variables:

There are 3 ways to declare a JavaScript variable:

1. Using var
2. Using let
3. Using const

Variables are containers for storing data (values).

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 _ .
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

1. Using var:

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

Example:

```
var carname;
carname = “maruti”;
var newcar = “volvo”;
var car1 = “Toyota” , car2 = “Audi”, car3 = “tavera”;
```

2. Using let:

Variables defined with let cannot be Redeclared. Variables defined with let must be Declared before use.

Example:

```
let x = "John Doe";
let x = 0;

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

3. Using const:

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

Example:

```
const PI = 3.14 ; //Correct
const pi;
pi = 3.14 // Incorrect
```

Data Types:

In javascript, there is no need to specify any type while creating variables. We can use “let” keyword while creating variable.

Example:

let x=5;

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Example:

```
let x;           // Now x is undefined
x = 5;           // Now x is a Number
x = "John";      // Now x is a String
```

Types:

1. Numbers:

JavaScript has only one type of numbers. Numbers can be written with, or without decimals:

Example:

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

2. Booleans:

Booleans can only have two values: true or false.

Example:

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

3. 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:

Example:

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

4. Arrays:

JavaScript arrays are written with square brackets. Array items are separated by commas. Example: The following code declares (creates) an array called cars, containing three items (car names):

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

5. 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"};
```

6. Undefined:

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

Example:
let car;

Any variable can be emptied, by setting the value to undefined. The type will also be undefined.

Example:
car = undefined;

Operators

1. Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

2. 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

3. 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

4. Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

5. Conditional Operator (?:) :-

The conditional (ternary) operator is the only JavaScript operator that takes three operands: a condition followed by a question mark (?), then an expression to execute if the condition is truth followed by a colon (:), and finally the expression to execute if the condition is false.

Example:

```
x = a >b ? a : b;
```

Control Structure

Conditional Statement:

1. If Statement:

if statement to specify a block of JavaScript code to be executed if a condition is true.

Example:
if (hour < 18) {
greeting = "Good day";
}

2.If..Else Statement:

The If..else statement contains two blocks, one as if statement and second is else statement where if statement specify a block of JavaScript code to be executed if a condition is true and else specify a block of code to be executed if the condition is false.

Example:

```
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

3. If..ElseIf.. Statement:

Use the else if statement to specify a new condition if the first condition is false.
Example:

```
if (time < 10) {
 greeting = "Good morning";
} else if (time < 20) {
 greeting = "Good day";
} else {
 greeting = "Good evening";
}
```

4. Switch Statement

In Switch statement, switch block is used 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
}
```

Looping Structure

1. 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:

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

2. 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);
```

Example:

The example below uses a do while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

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

3. For Loop:

The for loop has the following syntax:

Syntax:

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

Statement 1 is executed (one time) before the execution of the code block.
Statement 2 defines the condition for executing the code block.
Statement 3 is executed (every time) after the code block has been executed.

Example:

```
for (let i = 0; i< 5; i++) {
  text += "The number is " + i + "<br>";
}
```

4. For In Loop:

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

Syntax:

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

Example:

```
const numbers = [45, 4, 9, 16, 25];

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

Break and Continue Statement

Break Statement

The break statement is used to "jumps out" of a loop and switch statement.

Example:

```
for (let i = 0; i< 10; i++) {
if (i === 3) { break; }
text += "The number is " + i + "<br>";
}
```

In the example above, the break statement ends the loop ("breaks" the loop) when the loop counter (i) is 3.

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.

Example:

```
for (let i = 0; i< 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

In the above example it skips the value of 3.

JavaScript String functions

1. charAt(x): Returns the character at the “x” position within the string.

```
//charAt(x)
Var myString = 'jQuery FTW!!!';
console.log(myString.charAt(7));
//output: F
```

2. charCodeAt(x): Returns the Unicode value of the character at position “x” within the string.

```
//charAt(position)
var message="jquery4u"
//alerts "q"
alert(message.charAt(1))
```

3. concat(v1, v2,…): Combines one or more strings (arguments v1, v2 etc) into the existing one and returns the combined string. Original string is not modified.

```
//concat(v1, v2,..)
var message="Sam"
var final=message.concat(" is a"," hopeless romantic.")
//alerts "Sam is a hopeless romantic."
alert(final)
```

4. fromCharCode(c1, c2,…): Returns a string created by using the specified sequence of Unicode values (arguments c1, c2 etc). Method of String object, not String instance. For example: String.fromCharCode().

```
//fromCharCode(c1, c2,...)
console.log(String.fromCharCode(97,98,99,120,121,122))
//output: abcxyz
console.log(String.fromCharCode(72,69,76,76,79))
//output: HELLO
//(PS - I have no idea why you would use this? any ideas?)
```

Also see: Full List of JavaScript Character Codes

5. indexOf(substr, [start]): Searches and (if found) returns the index number of the searched character or substring within the string. If not found, -1 is returned. “Start” is an optional argument specifying the position within string to begin the search. Default is 0.

```
//indexOf(char/substring)
var sentence="Hi, my name is Sam!"
if (sentence.indexOf("Sam")!=-1)
alert("Sam is in there!")
```

6. lastIndexOf(substr, [start]) : Searches and (if found) returns the index number of the searched character or substring within the string. Searches the string from end to beginning. If not found, -1 is returned. “Start” is an optional argument specifying the position within string to begin the search. Default is string.length-1.

```
//lastIndexOf(substr, [start])
varmyString = 'javascriptrox';
console.log(myString.lastIndexOf('r'));
//output: 11
```

7. match(regexp) : Executes a search for a match within a string based on a regular expression. It returns an array of information or null if no match is found.

```
//match(regexp) //select integers only
varintRegex = /[0-9 -()+]+$/;

varmyNumber = '999';
varmyInt = myNumber.match(intRegex);
console.log(isInt);
//output: 999

varmyString = '999 JS Coders';
varmyInt = myString.match(intRegex);
console.log(isInt);
//output: null
```

Also see: jQueryRegEx Examples to use with .match()

8. replace(regexp/substr, replacetext): Searches and replaces the regular expression (or sub string) portion (match) with the replaced text instead.

```
//replace(substr, replacetext)
varmyString = '999 JavaScript Coders';
console.log(myString.replace(/JavaScript/i, "jQuery"));
//output: 999 jQuery Coders

//replace(regexp, replacetext)
varmyString = '999 JavaScript Coders';
console.log(myString.replace(new RegExp( "999", "gi" ), "The"));
//output: The JavaScript Coders
```

9. search(regexp): Tests for a match in a string. It returns the index of the match, or -1 if not found.

```
//search(regexp)
varintRegex = /[0-9 -()+]+$/;

varmyNumber = '999';
varisInt = myNumber.search(intRegex);
console.log(isInt);
//output: 0

varmyString = '999 JS Coders';
varisInt = myString.search(intRegex);
console.log(isInt);
//output: -1
```

10. slice(start, [end]): Returns a substring of the string based on the "start" and "end" index arguments, NOT including the "end" index itself. "End" is optional, and if none is specified, the slice includes all characters from "start" to end of string.

```
//slice(start, end)
var text="excellent"
text.slice(0,4) //returns "exce"
text.slice(2,4) //returns "ce"
```

11. split(delimiter, [limit]): Splits a string into many according to the specified delimiter, and returns an array containing each element. The optional "limit" is an integer that lets you specify the maximum number of elements to return.

```
//split(delimiter)
var message="Welcome to jQuery4u"
//word[0] contains "We"
//word[1] contains "lcome to jQuery4u"
var word=message.split("l")
```

12. substr(start, [length]): Returns the characters in a string beginning at "start" and through the specified number of characters, "length". "Length" is optional, and if omitted, up to the end of the string is assumed.

```
//substring(from, to)
var text="excellent"
text.substring(0,4) //returns "exce"
text.substring(2,4) //returns "ce"
```

13. substring(from, [to]): Returns the characters in a string between "from" and "to" indexes, NOT including "to" inself. "To" is optional, and if omitted, up to the end of the string is assumed.

```
//substring(from, [to])
varmyString = 'javascriptrox';
myString = myString.substring(0,10);
```

```
console.log(myString)
//output: javascript
```

14. toLowerCase(): Returns the string with all of its characters converted to lowercase.

```
//toLowerCase()
varmyString = 'JAVASCRIPT ROX';
myString = myString.toLowerCase();
console.log(myString)
//output: javascriptrox
```

15. toUpperCase(): Returns the string with all of its characters converted to uppercase.

```
//toUpperCase()
varmyString = 'javascriptrox';
myString = myString.toUpperCase();
console.log(myString)
//output: JAVASCRIPT ROX
```

JavaScript Events

An HTML event can be something the browser does, or something a user does. Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

1. Mouse Events

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

2. Keyboard events:

Event Performed	Event Handler	Description
Keydown&Keyup	onkeydown&onkeyup	When the user press and then release the key

3. Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element