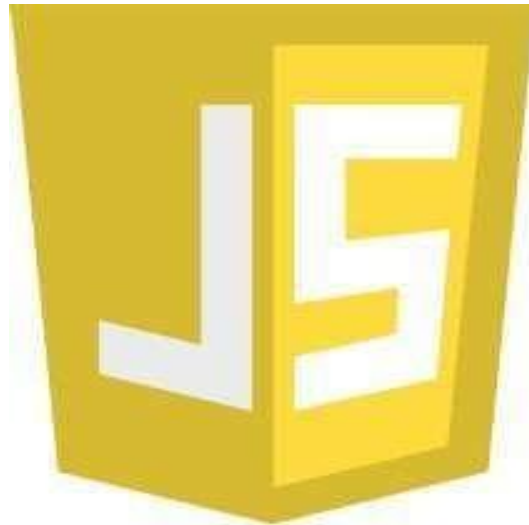


JAVASCRIPT



A study material for the students of GLS University

What is Script?

A Script is a program or sequence of instructions that is interpreted or carried out by another program rather than by computer processor.

Scripting Language

The scripting language is basically a language where instructions are written for a run time environment. They do not require the compilation step and are rather interpreted.

What is ECMAScript?

ECMAScript (European Computer Manufacturers Association) is a scripting language specification standardized by **ECMA International**.

It serves as the foundation for JavaScript, JScript, and ActionScript.

ECMAScript defines the rules, details, and guidelines for scripting languages, ensuring compatibility and consistency across different platforms and browsers.

What is JavaScript?

JavaScript is a versatile, dynamically typed programming language used for interactive web applications, supporting both client-side and server-side development, and integrating seamlessly with HTML, CSS, and a rich standard library.

- JavaScript is a **single-threaded** language that executes one task at a time.
- It is an **Interpreted** language which means it executes the code line by line.
- The data type of the variable is decided at run-time in JavaScript that's why it is called **dynamic**.

JavaScript was created by Brendan Eich, a Netscape Communications Corporation programmer, in September 1995.



- **HTML**
 - Hypertext Markup Language
 - Structure of Page
- **JavaScript**
 - Interactivity with User
 - Dynamic Updates in a Web Page
- **CSS**
 - Cascading Style Sheets
 - Presentation/Styling



JavaScript Features

Feature	Description
1. Lightweight	JavaScript is designed to be fast and efficient, with a minimal set of syntax and commands.
2. Interpreted Language	It doesn't need to be compiled; the browser interprets it line-by-line at runtime.
3. Dynamic Typing	You don't have to declare variable types explicitly (e.g., let x = 10;).
4. Object-Oriented	Supports object-based programming, allowing use of objects, classes, inheritance, etc.
5. Event-Driven	JavaScript can react to user actions such

Embedding JavaScript in HTML

1. Anywhere in the html file between `<script></script>` tags.

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

<h2>JavaScript Demo</h2>

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

<script>
document.write("My First JavaScript");
</script>
|
</body>
</html>
```

2. In an external file and refer to it using the SRC attribute.

```
<!DOCTYPE html>
<html>
<head>
<title>External Javascript</title>
<script src="myScript.js"></script>
</head>
<body>
<p id="demo">A Paragraph.</p>
</body>
</html>
```



```
<html>  
<body>  
<script type="text/javascript">  
document.write("Hello World!");  
</script>  
</body>  
</html>
```

Tells where the JavaScript starts

Commands for writing output to a page

Tells where the JavaScript ends

This code produce the output on an HTML page:
Hello World!

Console Object

The Console object provides access to the browser's debugging console.

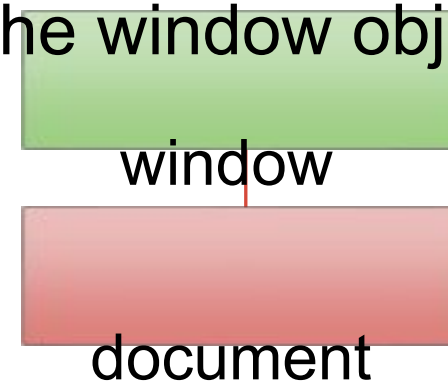
log() method: writes a message to the console.

Syntax
`console.log(message)`

```
<!DOCTYPE html>
<html>
<body>
<script>
console.log("Hello world!");
</script>
</body>
</html>
```

Window Object

- Window object represents the browser's window or potentially frame, that a document is displayed in.
- As long as a browser window is open, even if no document is loaded in the window, the window object is defined in the current model in memory.
- All global JavaScript variables, functions and objects automatically become members of the window object.



alert () Method

This Window object's method is used to display data in alert dialog box. alert really should be used only when you truly want to stop everything and let the user know something.

Syntax:- window.alert() or alert ()

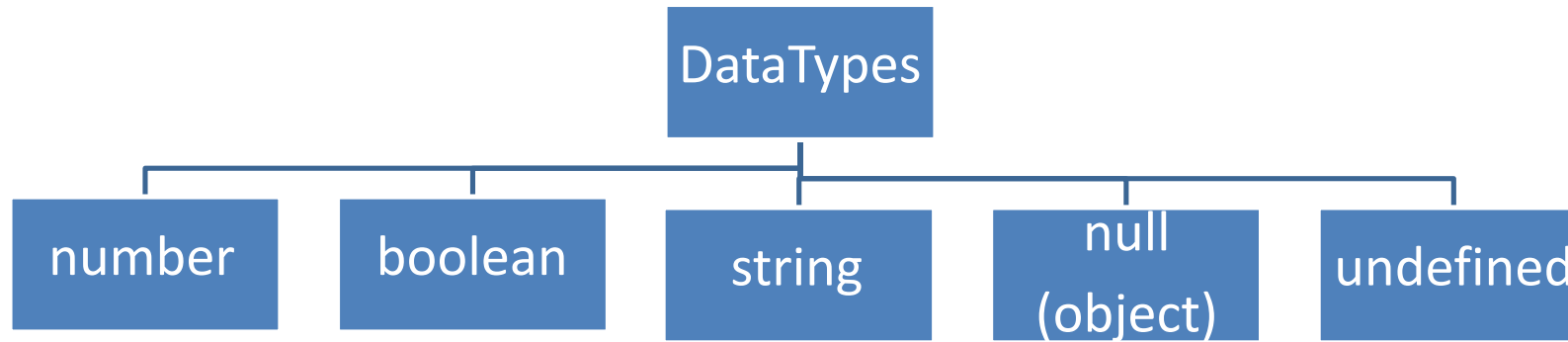
Ex: - window.alert("Hello World");

window.alert(variable);

window.alert(4+2);

window.alert("Hello World" + variable);

Data Types



```
var x=10;  
alert(typeof(x));
```

undefined value and null value

undefined value means the variable is declared but not assigned.

Ex.

```
var x;
```

```
console.log(x);
```

null value means the variable is declared and assigned as null means noting neighter number nor string or boolean.

```
var x=null;
```

```
console.log(x);
```

Global Properties & methods

Infinity: A numeric value that represents positive/negative infinity

NaN: "Not-a-Number" value

eval(): Evaluates a string and executes it as if it was script code

isFinite(): Determines whether a value is a finite, legal number

isNaN(): Determines whether a value is an illegal number

Number(): Converts an object's value to a number

parseFloat(): Parses a string and returns a floating point number

parseInt(): Parses a string and returns an integer

Javascript Operators

Arithmetic Operators: Perform basic mathematical operations.

Operator	Description	Example	Result
<code>+</code>	Addition	<code>5 + 3</code>	<code>8</code>
<code>-</code>	Subtraction	<code>5 - 3</code>	<code>2</code>
<code>*</code>	Multiplication	<code>5 * 3</code>	<code>15</code>
<code>/</code>	Division	<code>6 / 3</code>	<code>2</code>
<code>%</code>	Modulus (Remainder)	<code>5 % 3</code>	<code>2</code>
<code>**</code>	Exponentiation	<code>2 ** 3</code>	<code>8</code>
<code>++</code>	Increment	<code>let a = 1; a++</code>	<code>2</code>
<code>--</code>	Decrement	<code>let a = 1; a--</code>	<code>0</code>

Javascript

Assignment Operators: Assign values to variables.

Operator	Description	Example	Equivalent
=	Assignment	<code>x = 10</code>	-
+=	Add and assign	<code>x += 5</code>	<code>x = x + 5</code>
-=	Subtract and assign	<code>x -= 5</code>	<code>x = x - 5</code>
*=	Multiply and assign	<code>x *= 5</code>	<code>x = x * 5</code>
/=	Divide and assign	<code>x /= 5</code>	<code>x = x / 5</code>
%=	Modulus and assign	<code>x %= 5</code>	<code>x = x % 5</code>
**=	Exponentiation and assign	<code>x **= 2</code>	<code>x = x ** 2</code>

Javascript

Operators

Comparison Operators: Compare two values and return a boolean result (true or false).

Operator	Description	Example	Result
<code>==</code>	Equal to	<code>5 == '5'</code>	<code>true</code>
<code>===</code>	Strict equal to	<code>5 === '5'</code>	<code>false</code>
<code>!=</code>	Not equal to	<code>5 != '5'</code>	<code>false</code>
<code>!==</code>	Strict not equal to	<code>5 !== '5'</code>	<code>true</code>
<code>></code>	Greater than	<code>5 > 3</code>	<code>true</code>
<code><</code>	Less than	<code>5 < 3</code>	<code>false</code>
<code>>=</code>	Greater than or equal to	<code>5 >= 5</code>	<code>true</code>
<code><=</code>	Less than or equal to	<code>5 <= 3</code>	<code>false</code>

Equality (==) operators

The **equality (==)** operator converts the operands if they are not of the same type, then applies strict comparison.

Syntax:

A==B

```
1 == 1 // true
'1' == 1 // true
1 == '1' // true
0 == false // true
0 == null // false
```

Identity / strict equality (===) operators

The **identity (===)** operator returns true if the operands are strictly equal with no type conversion.

Syntax:

A===B

```
1 === 1 // true
1 === '1' // false
```

Operators

Logical Operators: Used to perform logical operations.

Operator	Description	Example	Result
<code>&&</code>	Logical AND	<code>true && false</code>	<code>false</code>
<code> </code>		<code> </code>	Logical OR
<code>!</code>	Logical NOT	<code>!true</code>	<code>false</code>

Falsy Values : false, 0 (zero), -0 (negative zero), "" (empty string), null, undefined, NaN (Not a Number)

Javascript Operators

Type Operators: Determine or manipulate the type of a value.

Operator	Description	Example	Result
<code>typeof</code>	Returns the type of a value	<code>typeof 42</code>	<code>'number'</code>
<code>instanceof</code>	Checks if an object is an instance of a class	<code>x instanceof Array</code>	<code>true</code>

Javascript Operators

Ternary Operator: A shorthand for if-else.

Syntax –

Condition ? True : false

```
let result = (5 > 3) ? 'Yes' : 'No'; // Result: 'Yes'
```

Javascript Operators

Nullish Coalescing Operator: Returns the right-hand operand if the left-hand operand is null or undefined.

```
let value = null ?? 'Default'; // Result: 'Default'
```


Decision-Making Statements

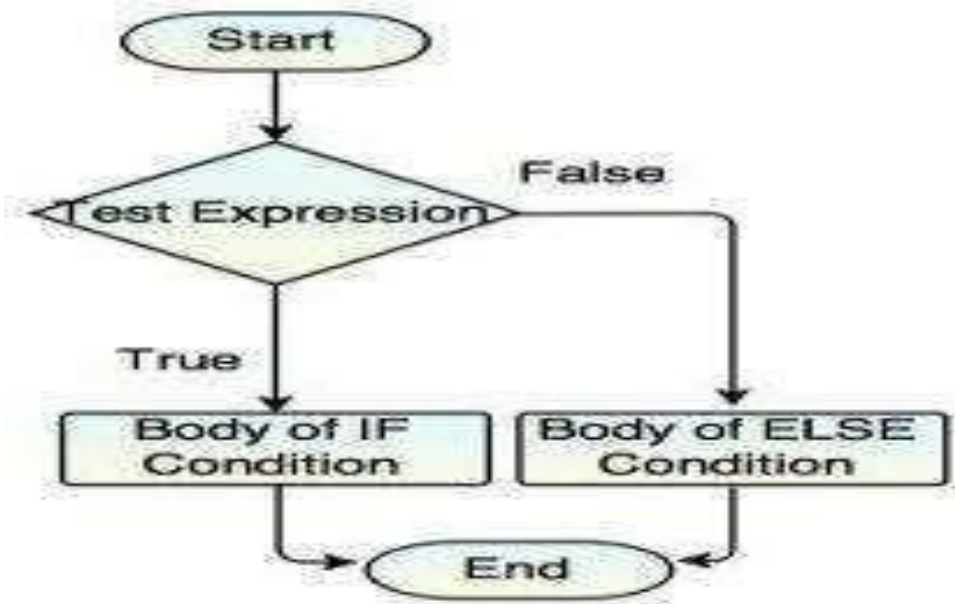
Decision-making statements control the flow of execution based on certain conditions. These conditions evaluate to true or false, and based on the result, different blocks of code are executed.

if Statement: Executes a block of code if the condition is true.

```
if (condition) {  
    // Code to execute if condition is true  
}
```

if...else Statement: Executes one block of code if the condition is true, and another if it is false.

```
if (condition) {  
    // Code if condition is true  
} else {  
    // Code if condition is false  
}
```



Decision-Making Statements

if...else if...else Statement:

Checks multiple conditions and executes the first block of code that evaluates to true.

```
if (condition1) {  
    // Code if condition1 is true  
} else if (condition2) {  
    // Code if condition2 is true  
} else {  
    // Code if none of the conditions are true  
}
```

Nested if Statements :

Checks multiple conditions and executes the first block of code that evaluates to true.

```
if (condition1) {  
    // Executes if condition1 is true  
    if (condition2) {  
        // Executes if condition1 and condition2 are true  
    } else {  
        // Executes if condition1 is true but condition2 is false  
    }  
} else {  
    // Executes if condition1 is false  
}
```

switch statement

Evaluates an expression and executes the matching case block. If no match is found, the default block is executed.

```
switch (expression) {  
  case value1:  
    // Code for value1  
    break;  
  case value2:  
    // Code for value2  
    break;  
  default:  
    // Code if no case matches  
}
```

```
let day = 3;  
switch (day) {  
  case 1:  
    console.log("Monday");  
    break;  
  case 2:  
    console.log("Tuesday");  
    break;  
  case 3:  
    console.log("Wednesday");  
    break;  
  default:  
    console.log("Invalid day");  
}
```

break and continue

break Statement:

The break statement is used to exit a loop or a switch statement immediately, regardless of whether the loop has completed all iterations or not.

continue Statement:

The continue statement is used to skip the current iteration of the loop and proceed to the next iteration.

Loops

A loop in programming is a control structure that allows you to execute a block of code repeatedly, based on a condition or a defined number of times. Loops help in automating repetitive tasks, reducing redundancy, and improving code efficiency.

Key Components of a Loop

Initialization: Sets the starting point of the loop.

Condition: Determines whether the loop should continue running. If the condition is true, the loop executes; if false, it stops.

Body: The block of code to execute repeatedly.

Update: Updates a variable to eventually meet or fail the condition, avoiding infinite loops.

Types of Loops

1. Entry-Controlled / Pre-tested Loops:

The condition is checked before executing the loop body. Examples: for and while loops.

2. Exit-Controlled / post-tested Loops:

The condition is checked after executing the loop body. Example: do-while loop.

Why Use Loops?

Automates repetitive tasks.

Reduces the length of the code.

Enhances code readability and maintainability.

Enables dynamic programming for varying input sizes.

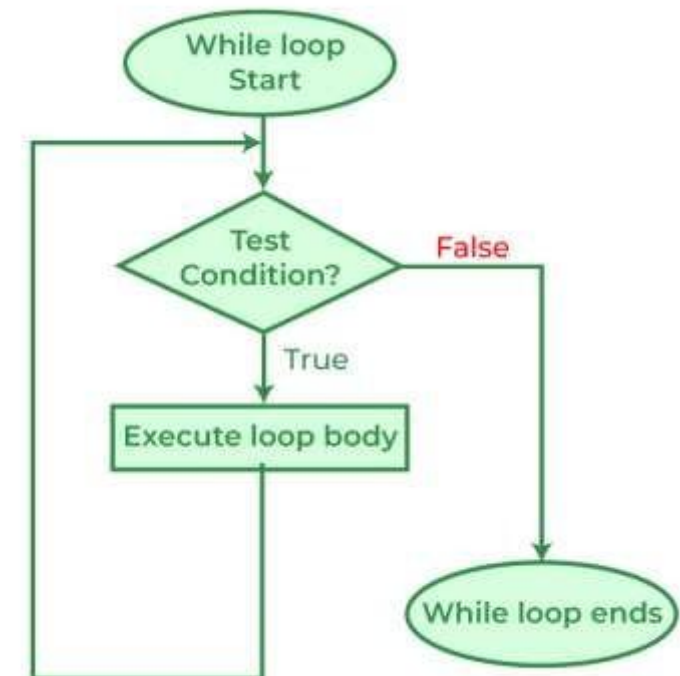
while loop

A while loop executes a block of code as long as a specified condition is true.

Syntax -

```
while (condition) {  
    // Code to execute  
}
```

```
let count = 1;  
  
while (count <= 5) {  
    console.log("Count is:", count); // Output the value of count  
    count++; // Increment count  
}
```



for loop

A for loop is used to execute a block of code a specific number of times. It is particularly useful when the number of iterations is known in advance.

Syntax –

```
for (initialization; condition; increment/decrement) {  
    // Code to execute  
}
```

```
for (let i = 1; i <= 5; i++) {  
    console.log("Number:", i);  
}
```

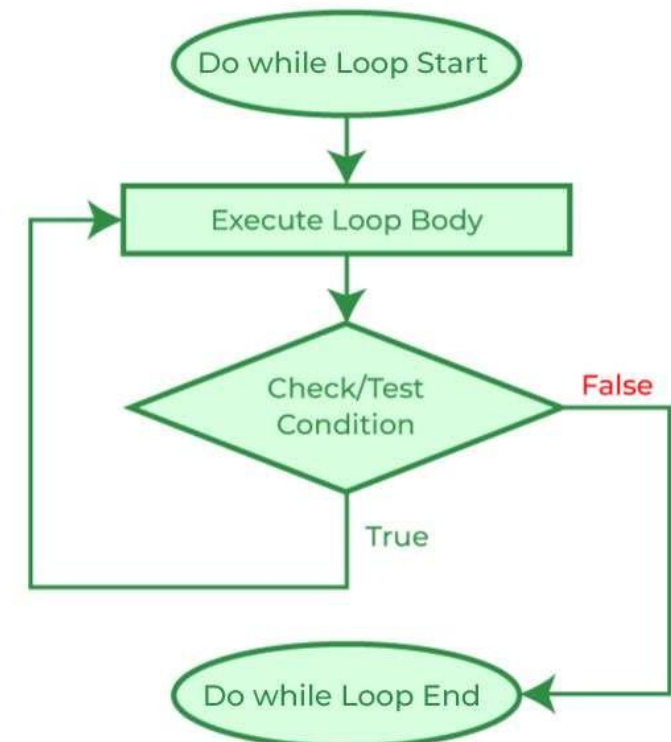
do while loop

A do-while loop ensures the block of code executes at least once, regardless of whether the condition is true or false, because the condition is evaluated after the loop body is executed.

Syntax -

```
do {  
    // Code to execute  
} while (condition);
```

```
let count = 1;  
  
do {  
    console.log("Count is:", count);  
    count++;  
} while (count <= 5);
```



nested loop

Nested loops are loops inside another loop. They are commonly used when working with multidimensional data structures like matrices or tables.

How Nested Loops Work :

- The outer loop executes first.
- Inside the outer loop, the inner loop runs completely for each iteration of the outer loop.
- This process repeats until the outer loop condition becomes false.

```
for (initialization; condition; increment) {  
  for (initialization; condition; increment) {  
    // Code to execute  
  }  
}
```

```
for (let i = 1; i <= 5; i++) {  
  for (let j = 1; j <= 5; j++) {  
    console.log(`${i} x ${j} = ${i * j}`);  
  }  
  console.log("-----");  
}
```

```
let matrix = [  
  [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9]  
];  
  
for (let i = 0; i < matrix.length; i++) {  
  for (let j = 0; j < matrix[i].length; j++) {  
    console.log(`Element at [${i}][${j}] is ${matrix[i][j]}`);  
  }  
}
```

What is Event

- The actions to which JavaScript can respond are called Events.
- An Event is some notable action to which a script can respond.

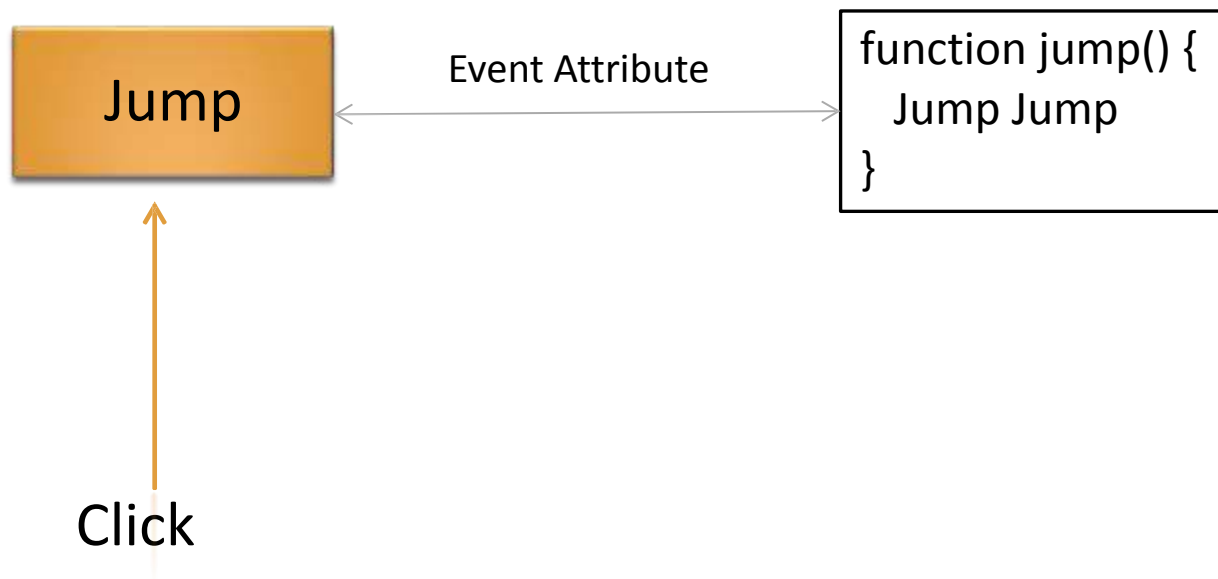


Click

- Clicking an element
- Submitting a form
- Scrolling page
- Hovering an element

Event Handler

- An Event handler is JavaScript code associated with a particular part of the document and a particular event. A handler is executed if and when the given event occurs at the part of the document to which it is associated.



- onclick
- ondblclick
- onchange
- onblur

Event Binding with HTML Attribute

- These bindings are element attributes, such as onclick and onchange, which can be set equal to JavaScript that is to be executed when the given event occurs at that object.
- Ex: -
- `<button onclick= "alert('Button Clicked');">Click Me</button>`

Mouse Event

- mousedown – It fires when mouse button is pressed down.
- mouseup – It fires when the mouse button is released.
- click – It fires when something is clicked. mousedown, mouseup and click events fires in sequence.
- dblclick – It fires when something is clicked twice in rapid succession. mousedown, mouseup, click, mousedown, mouseup, click, and dblclick events fires in sequence.
- mouseenter – It fires when a mouse starts to hover over some element. NO bubble.
- mouseleave – It fires when a mouse exits while hovering over some element. No bubble.

Focus Event

- focus – It fires when an element gains focus, such as selecting a form field.
No Bubble
- blur – It fires when element loses focus, such as moving away from a form field.
No Bubble
- focusin – It fires just as an element is about to gain focus.
- focusout – It fires just as an element loses focus and just before the blur event.

Key Event

- keydown – It fires as a key is pressed down.
- keypress – It fires after a key is pressed down (after keydown). It only works with printable characters.
- keyup – It fires as the key is released.

Array Object

An Array in JavaScript is a special type of object used to store multiple values in a single variable. Unlike regular objects, arrays use numeric indices to access elements.

Key Characteristics of an Array Object:

1. **Ordered Data:** Arrays store data in an ordered collection.
2. **Index-based Access:** Each item in an array is accessed via an index (starting from 0).
3. **Dynamic Size:** Arrays in JavaScript are dynamic, meaning their size can grow or shrink during runtime.
4. **Array Methods:** Arrays come with built-in methods to manipulate and interact with the elements.

Using Square Brackets (Recommended):

```
var arrayname=["value1","value2"....."valueN"];
```

By using an Array constructor (using new keyword):

```
Prepared by : Prof. Harshita Maheshwari var arrayname=new Array("value1","value2","value3");
```

```
let fruits = ["apple", "banana",  
"cherry"]; console.log(fruits[0]);  
console.log(fruits[2]);
```

Output:

apple
cherry

```
let emp=new Array("one","two","three");  
for (i=0;i<emp.length;i++){  
    document.write(emp[i] + "<br>");  
  
}
```

Output:

one
two
three

for of loop

The for...of loop is used to iterate over iterable objects such as arrays, strings, or sets.

Syntax -

```
for (variable of iterable) {  
    // Code to execute  
}
```

```
const fruits = ["apple", "banana", "cherry"];  
for (const fruit of fruits) {  
    console.log(fruit);  
}
```

Array Properties

Length Property: Returns the number of elements in an array.

```
let fruits = ["apple", "banana", "cherry"];  
console.log(fruits.length); // Output: 3
```

Array-methods

[join\(\)](#): Joins all elements of an array into a string.

It behaves just like `toString()`, but in addition we can specify the separator.

Syntax: `array.join(separator);`

[pop\(\)](#):

Removes the last element from an array and returns that element.

Syntax: `array.pop();`

[push\(\)](#):

Adds one or more elements to the end of an array and returns the new length of the array.

Syntax: `array.push(element1, ..., elementN);`

[shift\(\)](#): Removes the first element from an array and returns that element.

Syntax: `array.shift();`

unshift(): Adds one or more elements to the front of an array and returns the new length of the array.

Syntax: array.unshift(element1, ..., elementN);

splice(): Adds and/or removes elements from an array.

Syntax: array.splice(index, howMany, [element1][, ..., elementN]);

index – Index at which to start changing the array.

howMany – An integer indicating the number of old array elements to remove.

If **howMany** is 0, no elements are removed.

element1, ..., elementN – The elements to add to the array. If you don't specify any elements, splice simply removes the elements from the array.

slice(): Extracts a section of an array and returns a new array. Syntax:

Syntax: array.slice(begin [,end]);

It does not remove any elements from the source array.

sort() Sorts the elements of an array Syntax:

```
array.sort( );
```

```
array.sort(compareFunction);
```

When the `sort()` function compares two values, it sends the values to the compare function, and sorts the values according to the returned (*negative, zero, positive*) value

```
<script>
```

```
let num = [10,20,20,10];
```

```
console.log(num.sort((a , b)=>a-b);
```

```
</script>
```

reverse(): Reverses the order of the elements of an array – the first becomes the last, and the last becomes the first.

Syntax: `array.reverse()`;

Array.from(): Converts array-like or iterable objects to arrays.

Array forEach() method

The forEach method is also used to loop through arrays, but it uses a function differently than the classic "for loop".

The forEach method passes a callback function for each element of an array together with the following parameters:

Current Value (required) - The value of the current array

element Index (optional) - The current element's index number

Array (optional) - The array object to which the current element belongs

```
const numbers = [1, 2, 3, 4, 5];  
numbers.forEach((number, index) => {  
  console.log('Index: ' + index + ' Value: ' + number);  
});
```

Index: 0	Value: 1
Index: 1	Value: 2
Index: 2	Value: 3
Index: 3	Value: 4
Index: 4	Value: 5

Array map()

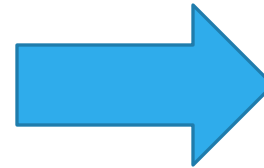
method

The `Array.map()` method allows you to iterate over an array and modify its elements using a callback function. The callback function will then be executed on each of the array's elements.

```
let arr = [3, 4, 5, 6];

for (let i = 0; i < arr.length; i++){
  arr[i] = arr[i] * 3;
}

console.log(arr); // [9, 12, 15, 18]
```



```
let arr = [3, 4, 5, 6];

let modifiedArr = arr.map(function(element){
  return element * 3;
});

console.log(modifiedArr); // [9, 12, 15, 18]
```

Array filter() method

It entails filtering out one or more items (a subset) from a larger collection of items (a superset) based on some condition/preference.

The filter() method takes in a callback function and calls that function for every item it iterates over inside the target array. The callback function can take in the following parameters:

- currentItem: This is the element in the array which is currently being iterated over.
- index: This is the index position of the currentItem inside the array.
- array: This represents the target array along with all its items.

The filter method creates a new array and returns all of the items which pass the condition specified in the callback.

Array filter() method Example

```
let people = [  
  {name: "aaron", age: 65},  
  {name: "beth", age: 2},  
  {name: "cara", age: 13},  
  {name: "daniel", age: 3},  
  {name: "ella", age: 25},  
  {name: "fin", age: 1},  
  {name: "george", age: 43},  
]  
  
let toddlers = people.filter(person => person.age <= 3)  
  
console.log(toddlers)
```

Array find() method

Returns the first element in the array that satisfies the provided test (callback function).

Returns: The found element, or undefined if no element satisfies the condition.

```
const numbers = [5, 12, 8, 130, 44];  
  
// Find the first number greater than 10  
const found = numbers.find(num => num > 10);  
console.log(found); // 12
```

Array `findIndex()` method

Returns the index of the first element in the array that satisfies the provided test (callback function).

Returns: The index of the found element, or -1 if no element satisfies the condition.

```
const numbers = [5, 12, 8, 130, 44];  
  
// Find the index of the first number greater than 10  
const index = numbers.findIndex(num => num > 10);  
console.log(index); // 1
```

Array reduce() method

The reduce() method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value.

```
<p id="demo"></p>
<script>
const numbers = [175, 50, 25];
document.getElementById("demo").innerHTML =
numbers.reduce((prev, curr) =>prev+curr);
</script>
```


Array reduceRight() method

The reduceRight() method applies a function against an accumulator and each value of the array (from right-to-left) to reduce it to a single value.

```
<p id="demo"></p>
<script>
const numbers = [175, 50, 25];
document.getElementById("demo").innerHTML =
numbers.reduceRight((prev, curr) =>prev+curr);
</script>
```

String Object

The String object is a constructor for strings, or a sequence of characters.

Syntax:

```
var s = new String(string);
```

Length: Returns the length of the string.

Special Characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

String Methods

[charAt\(\)](#): Returns the character at the specified index Syntax: `string.charAt(index);`

index – An integer between 0 and 1 less than the length of the string.
Return Value Returns the character from the specified index.

[split\(\)](#): Splits a String object into an array of strings by separating the string into substrings.

Syntax: `string.split([separator][, limit]);`

```
let x1 = ("Welcome to GLS".split());  
let x2 = (" Welcome to GLS ".split(" "));  
let x3= (" Welcome to GLS".split(" ", 2));
```

Math object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

- `Math.round(x)` returns the value of `x` rounded to its nearest integer
- `Math.pow(x, y)` returns the value of `x` to the power of `y`
- `Math.sqrt(x)` returns the square root of `x`
- `Math.abs(x)` returns the absolute (positive) value of `x`
- `Math.ceil(x)` returns the value of `x` rounded **up** to its nearest integer
- `Math.floor(x)` returns the value of `x` rounded **down** to its nearest integer
- `Math.min()` and `Math.max()` can be used to find the lowest or highest value in a list of arguments

Math.random() : returns a random number between 0 and 1.

`Math.floor(Math.random() * 10);` // returns a number between 0 and 9

`Math.floor(Math.random() * 11);` // returns a number between 0 and 10

`Math.floor(Math.random() * 100);` // returns a number between 0 and 99

`Math.floor(Math.random() * 10) + 1;` // returns a number between 1 and 10

`Math.floor(Math.random() * 100) + 1;` // returns a number between 1 and 100

Date Object

A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.
Date objects are created with the **new Date()** constructor.

new Date()

new Date(milliseconds) **new**

Date(dateString)

new Date(year, month, day, hours, minutes, seconds, milliseconds)

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

Document Object Model

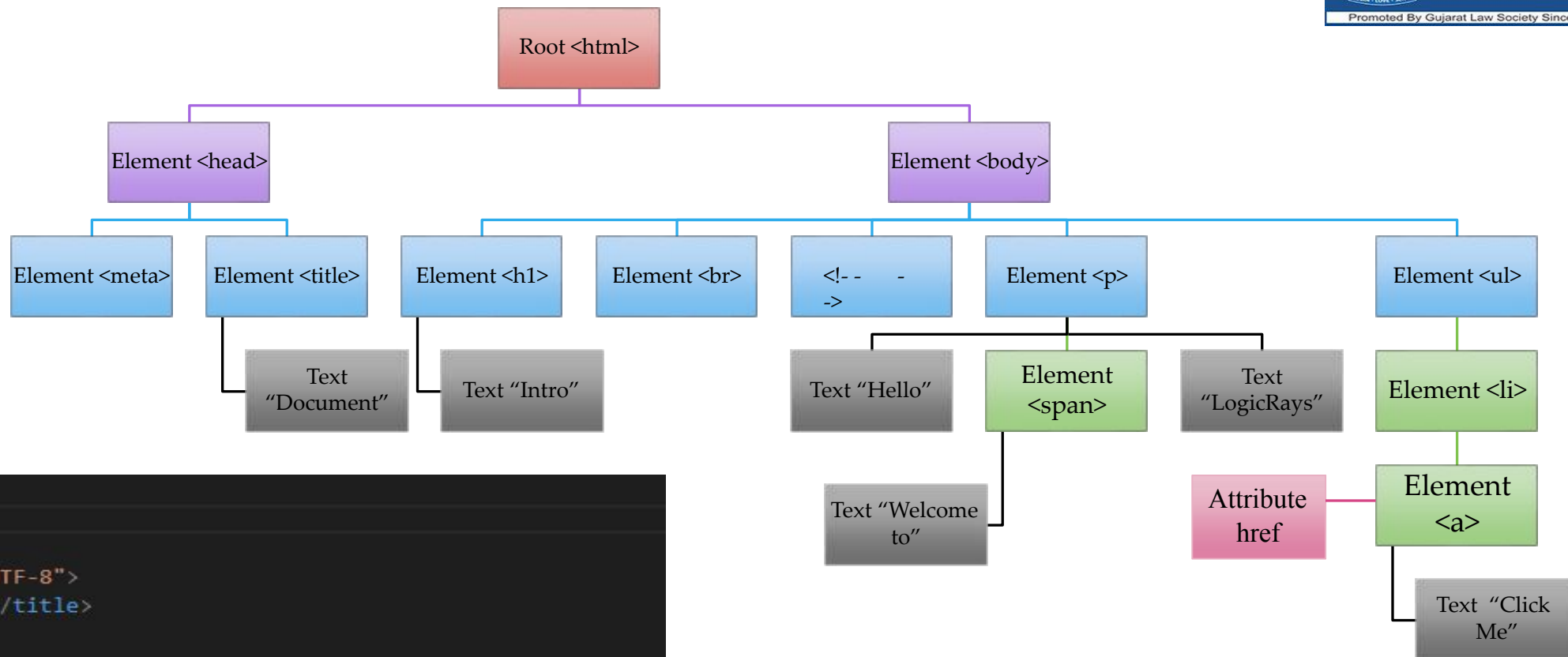
- The DOM is a Web API that allows developers to use programming logic to make changes to their HTML code. It's a reliable way to make changes that turn static websites into dynamic ones.
- With the Document Object Model, programmers can create and build documents, navigate their structure, and add, modify, or delete elements and content.
- The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.
- The DOM model represents a document with a logical tree.

Document:- Html Page

Object:- Elements and attributes

Model:- Tree Structure of HTML elements

Document Tree



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1>Intro</h1>
  <br>
  <!-- This is a paragraph -->
  <p>Hello<span>Welcome to</span>LogicRays</p>
  <ul>
    <li>
      <a href="https://www.logicraysacademy.com/">Click Me</a>
    </li>
  </ul>
</body>
</html>
```

getElementById("ID_Name")

The method `getElementById("ID_Name")` returns an Element object representing the element whose id property matches the specified string. Since element IDs are required to be unique if specified, they're a useful way to get access to a specific element quickly.

Ex: - `getElementById("uname")`

Where uname is id of that element.

```
<p id = "para">Welcome to GLS</p>
```

```
document.getElementById("para")
```


```
var result = document.getElementById("para")
```

innerHTML

- The innerHTML property defines the HTML content.
- `<p id = "para">This is a paragraph.</p>`
- `document.getElementById("para").innerHTML`

JavaScript User Defined Object

A JavaScript object is an entity having state and behavior (properties and method).

Object	Properties	Methods
	<code>car.name = Fiat</code> <code>car.model = 500</code> <code>car.weight = 850kg</code> <code>car.color = white</code>	<code>car.start()</code> <code>car.drive()</code> <code>car.brake()</code> <code>car.stop()</code>

Syntax:

object={property1:value1, property2:value2.....propertyN:valueN}

```
let employee= { name:"Ram Kumar", salary:40000 }  
document.write(`Name = ${employee.name} and Salary =  
${employee.salary}`);
```

for in loop

The for...in loop iterates over the properties of an object (keys) or the indexes of an array.

Syntax –

```
for (key in object) {  
    // Code to execute  
}
```

```
const person = { name: "John", age: 30, city: "New York" };  
for (const key in person) {  
    console.log(`${key}: ${person[key]}`);  
}
```

method in JavaScript Object literal

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

```
<script>
```

```
let person = {firstName:"Sohan", lastName : "Lal"};
```

```
person.name = function() {
```

```
    return this.firstName + " " + this.lastName;
```

```
};
```

```
document.getElementById("demo").innerHTML ="Name is " +
```

```
person.name();
```

```
</script>
```

Understanding this

this keyword in JavaScript refers to the object that is currently executing the code. Its value depends on the execution context in which it is used.

- In a browser: this refers to the window object.
- Inside an Object Method: this refers to the object that owns the method.
- Inside a Regular Function: this refers to the global object (like window in browsers).
- Arrow functions do not have their own this; they inherit it from their surrounding lexical scope.
- In a Constructor Function or Class: this refers to the new object being created.
- When Using call, apply, or bind: You can explicitly set the value of this.
- In event handlers, this refers to the element that fired the event.

Object Properties

An **object** is a collection of properties, and a property is an association between a **name** (or **key**) and a **value**. A property's value can be a **function**, in which case the property is known as a **method**.

syntax

- `objectName.property` *// person.age*
- `objectName["property"]` *//*
 person["age"]

Function Expressions

A JavaScript function can also be defined using an **expression**.

Unlike a function declaration, a function expression does not require a function name and is often used when functions are assigned to variables or passed as arguments.

Syntax -

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

```
const greet = function() {  
    console.log("Hello!");  
};
```

```
greet(); // Output: Hello!
```

```
const factorial = function fact(n) {  
    return n <= 1 ? 1 : n * fact(n - 1);  
};
```

```
console.log(factorial(5)); // Output: 120
```

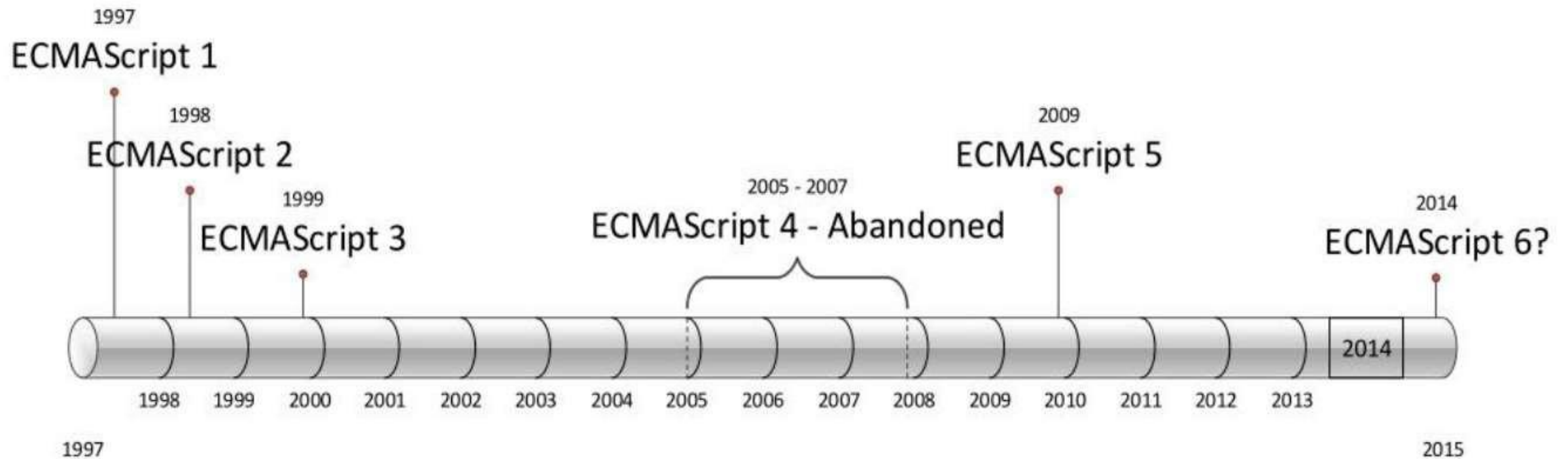
ECMA Script6

JavaScript ES6 (also known as ECMAScript 2015 or ECMAScript 6) is the newer version of JavaScript that was introduced in 2015.

ECMAScript is the standard that JavaScript programming language uses. ECMAScript provides the specification on how JavaScript programming language should work.

JavaScript ES6 brings new syntax and new awesome features to make your code more modern and more readable. It allows you to write less code and do more.

ECMA Script History



let and const

var	let	const
Function scope	Block scope	Block scope
Hoisted	Not Hoisted	Not Hoisted
Can declare and initialization can be done later	Can declare and initialization can be done later	Must be initialized at the time of declaration
Can change	Can change	Read-only
Use for top level variables	Use for localized variable in smaller scope	Use for read-only/fixed values

let

```
1 for (let i = 0; i < 10; ++i) {  
2   console.log(i); // 0, 1, 2, 3, 4 ... 9  
3 }  
4  
5 console.log(i); // i is not defined
```

const

```
1 const PI = 3.14159265359;  
2 PI = 0; // => 0  
3 console.log(PI); // => 3.14159265359
```

```
function f() {  
  {  
    let x;  
    {  
      // okay, block scoped name  
      const x = "sneaky";  
      // error, const  
      x = "foo";  
    }  
    // error, already declared in block  
    let x = "inner";  
  }  
}
```

Arrow Functions

Shorthand for Function Expression.

Makes your code more readable, more structured, and look like modern code.

```
// ES5

function myFunc(name) {
  return 'Hello' + name;
}

console.log(myFunc('said'));

// output
// Hello said
```

```
// ES6 Arrow function

const myFunc= name =>{
  return `Hi ${name}`;
}

console.log(myFunc('Said'))// output Hi Said

// or even without using arrow or implement `return` keyword
const myFunc= name => `Hi ${name}`;

console.log(myFunc('Said')) // output Hi Said
```


We can use Arrow function with map, filter, and reduce built-in functions.

```
// ES5

const myArray=['tony','Sara','Said',5];

let Arr1= myArray.map(function(item){
    return item;
});
console.log(Arr1);//output (4) ["tony", "Sara", "Said", 5]

//ES6 Syntax

let Arr2 = myArray.map(item => item);
console.log(Arr2); //output (4) ["tony", "Sara", "Said", 5]
```


Multi-line Strings

We can create multi-line strings by using back-ticks(`).

It can be done as shown below :

```
let greeting = `Hello World,  
                Greetings to all,  
                Keep Learning and  
                Practicing!`
```

Template Literals

Syntactic sugar for string construction.

We don't have to use the plus (+) operator to concatenate strings, or when we want to use a variable inside a string.

```
//ES5  
  
function myFunc1(name,age){  
  return 'Hi' + name + ' Your age is' + age + 'year old!';  
}  
  
console.log(myFunc1('Said',22))  
//output -->Hi Said,Your age is 22year old!
```

```
//ES6  
  
const myFunc= (name,age)=>{  
  return `Hi ${name},Your age is ${age}year old!`;  
}  
  
console.log(myFunc('Said',22))  
//output--> Hi Said,Your age is 22year old!
```

Default Parameter

Allows us to give default values to function parameters.

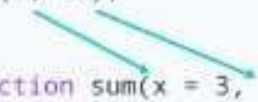
We can provide the default values right in the signature of the functions.

```
function sum(x = 3, y = 5) {  
    // return sum  
    return x + y;  
}  
  
console.log(sum(5, 15)); // 20  
console.log(sum(7));      // 12  
console.log(sum());       // 8
```

Case 1: Both Argument are Passed

sum(5, 15);


function sum(x = 3, y = 5) {
 return x + y;
}



Case 2: One Argument is Passed

sum(7);

function sum(x = 3, y = 5) {
 return x + y;
}



Case 3: No Argument is Passed

sum();

function sum(x = 3, y = 5) {
 return x + y;
}

Rest parameter

The rest parameters allows a function to accept an indefinite number of arguments as an array.

```
function myFun(a, b, ...manyMoreArgs) {  
  console.log("a", a)  
  console.log("b", b)  
  console.log("manyMoreArgs", manyMoreArgs)  
}  
  
myFun("one", "two", "three", "four", "five", "six")  
  
// Console Output:  
// a, one  
// b, two  
// manyMoreArgs, ["three", "four", "five", "six"]
```

Spread Operator

The spread operator ... is used to expand or spread an iterable or an array.

```
function sum(x, y, z) {  
  return x + y + z;  
}  
  
const numbers = [1, 2, 3];  
  
console.log(sum(...numbers));  
// expected output: 6
```

```
const arr1 = ['one', 'two'];  
const arr2 = [...arr1, 'three', 'four', 'five'];  
  
console.log(arr2);  
// Output:  
// ["one", "two", "three", "four", "five"]
```

```
const obj1 = { x : 1, y : 2 };  
const obj2 = { z : 3 };  
  
// add members obj1 and obj2 to obj3  
const obj3 = {...obj1, ...obj2};  
  
console.log(obj3); // {x: 1, y: 2, z: 3}
```

Destructuring Assignment

The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

Array Destructuring :-

```
const arrValue = ['one', 'two', 'three'];  
  
// destructuring assignment in arrays  
const [x, y, z] = arrValue;  
  
console.log(x); // one  
console.log(y); // two  
console.log(z); // three
```

```
let arrValue = [10];  
  
// assigning default value 5 and 7  
let [x = 5, y = 7] = arrValue;  
  
console.log(x); // 10  
console.log(y); // 7
```


Object Destructuring -

```
// assigning object attributes to variables
const person = {
  name: 'Sara',
  age: 25,
  gender: 'female'
}

// destructuring assignment
let { name, age, gender } = person;

console.log(name); // Sara
console.log(age); // 25
console.log(gender); // female
```

```
const person = {
  name: 'Sara',
  age: 25,
  gender: 'female'
}

// destructuring assignment
// using different variable names
let { name: name1, age: age1, gender: gender1 } = person;

console.log(name1); // Sara
console.log(age1); // 25
console.log(gender1); // female
```

for of loop

Allows to iterate over iterable objects (arrays, strings etc).

Syntax -

```
for (element of iterable) {  
    // body of for...of  
}
```

iterable - an iterable object (array, set, strings, etc).

element - items in the iterable

```
// array  
const students = ['John', 'Sara', 'Jack'];  
  
// using for...of  
for ( let element of students ) {  
  
    // display the values  
    console.log(element);  
}
```


class

A Class is template/blueprint for creating objects.

Ex- A sketch (prototype) of a house is a class . It contains all the details about the floors, doors, windows, etc. Based on these descriptions, you build the house. House is the object.

Syntax -

```
class myClass{  
    constructor(){  
  
    }  
}
```

```
class Employee{
    constructor(id,fname,lname,salary){
        this.id=id
        this.fname=fname
        this.lname=lname
        this.salary=salary
    }
    set setid(x){
        this.id=x;
    }
    get getid(){
        return this.id
    }
    getFullName(){
        return `${this.fname} ${this.lname}`
    }
    getSalary(){
        document.write(this.salary)
    }
}

const e1=new Employee(101,"Smith","Boss",20000)
document.write(`${e1.getFullName()} <BR>`)
e1.setid=110
document.write(e1.getid)
```

Inheritance

Inheritance enables you to define a class that takes all the functionality from a parent class and allows you to add more.

Reusability + Extensibility

```
class Manager extends Employee{
  constructor(id,fname,lname,salary,target){
    super(id,fname,lname,salary);
    this.target=target }
  getSalary(){ document.write(this.salary*1.15) }
}
const m1=new Manager(110,"dffdf","fdfnkdh",2000,1000)
document.write(`${m1.getFullName()} <BR>`)
document.write(`${m1.target} <BR>`)
m1.getSalary()
```

modules

A module is nothing more than a chunk of JavaScript code written in a file. By default, variables and functions of a module are not available for use. Variables and functions within a module should be exported so that they can be accessed from within other files. Modules in ES6 work only in strict mode.

Exporting and importing a Module :-

Named Exports and imports

Default Exports and imports

Named export and import

Named exports are distinguished by their names. There can be several named exports in a module.

```
//using multiple export keyword  
export component1  
export component2  
...  
...  
export componentN
```

```
//using single export keyword  
  
export {component1,component2,...,componentN}
```

```
import {component1,component2..componentN} from module_name
```

```
import {original_component_name as new_component_name }
```

```
import * as variable_name from module_name
```

Default export and import

Modules that need to export only a single value can use default exports. There can be only one default export per module.

```
export default component_name
```

Unlike named exports, a default export can be imported with any name.

```
import any_variable_name from module_name
```

To execute both the modules we need to make a html file as shown below and run this in live server. Note that we should use the attribute `type="module"` in the script tag.

JSON

What is JSON

- **JavaScriptObject Notation(JSON):**

- Is an open standard light-weight format that is used to store and exchange data.
- Is an easier and faster alternative to XML.
 - Is a language independent format that uses human readable text to transmit data objects.
- Consists of objects of name/value pairs.
- Files have the extension .json.

Syntactically, JSON is similar to the code for creating JavaScript objects. Due to this similarity, standard JavaScript methods can be used to convert JSON data into JavaScript objects.

Why use JSON

- Straightforward syntax
- Easy to create and manipulate
- Supported by all major JavaScript frameworks Supported by most backend technologies

When use JSON

Transfer data to and from a server

Perform asynchronous data calls without requiring a page refresh

Working with data stores

Compile and save form or user data for local storage

JSON Syntax

JSON uses name/value pairs to store data.

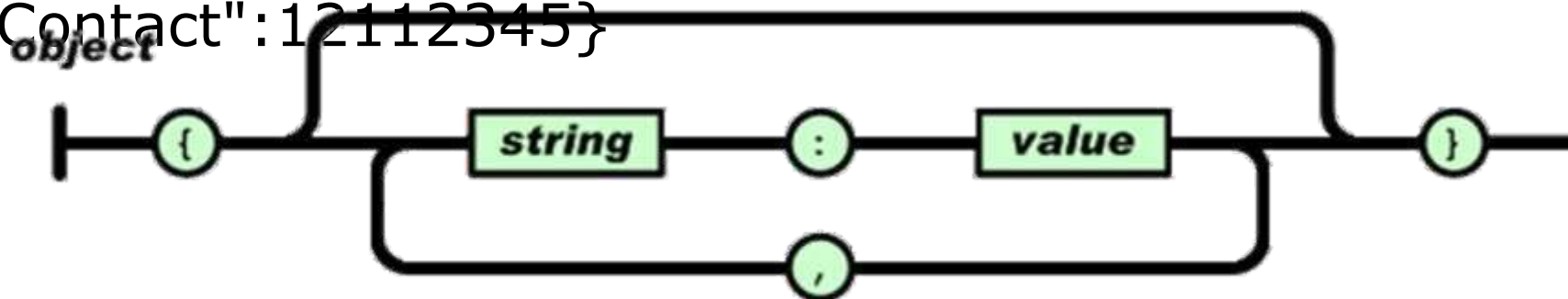
Commas are used to separate multiple data values. Objects are enclosed within curly braces.

Square brackets are used to store arrays.

JSON keys must be enclosed within double quotes.

Syntax:

```
{"fName":"Ronald", "lName":"Smith",  
"Contact":12112345}
```



JSON Values

String

Boolean

Object

Number

Null

Array

```
{
  "actor": {
    "name": "Tom Cruise",
    "age": 56,
    "Born At": "Syracuse, NY",
    "Birthdate": "July 3 1962",
    "photo": "https://jsonformatter.org/img/tom-cruise.jpg"
  }
}
```

```
{
  "Actors": [
    {
      "name": "Tom Cruise",
      "age": 56,
      "Born At": "Syracuse, NY",
      "Birthdate": "July 3, 1962",
      "photo": "https://jsonformatter.org/img/tom-cruise.jpg"
    },
    {
      "name": "Robert Downey Jr.",
      "age": 53,
      "Born At": "New York City, NY",
      "Birthdate": "April 4, 1965",
      "photo": "https://jsonformatter.org/img/Robert-Downey-Jr.jpg"
    }
  ]
}
```

JSON vs XML

Similarities

- Both are human-readable, that is, self-describing.
- Both represent hierarchical structure, that is, values within values.
- Both can be accessed and parsed by almost every programming language.
- Both can be accessed and fetched with an XMLHttpRequest object.

Dissimilarities

- XML needs an XML parser, whereas, a standard JavaScript method can be used to parse JSON.
- There is no need of end tag in JSON.
- JSON is much shorter as compared to XML.
- It is easy to read and write JSON.
- JSON can be used with arrays.

JSON vs XML

XML

```
<students>
  <student>
    <fName>Jenny</fName>
    <lName>Watson</lName>
  </student>
  <student>
    <fName>Dean</fName>
    <lName>Smith</lName>
  </student>
</students>
```

JSON

```
{"students": [
  {"fName": "Jenny", "lName": "Watson"},
  {"fName": "Dean", "lName": "Smith"}
]}
```


JSON string to JavaScript Object

To read data from a JSON object, you can use the `JSON.parse()` method provided by JavaScript.

Syntax: `var obj =JSON.parse(text);`

Example:

```
<script>
var x = '[
    { "code": "1001", "name": "ram" },
    { "code": "1002", "name": "shyam" },
    { "code": "1003", "name": "seeta" }
]';
var r = JSON.parse(x);
for (var i in r)
document.write(r[i].code+" "+r[i].name+"<br/>");
</script>
```

JSON String from JavaScript Object

JavaScript provides you the `JSON.stringify()` method that allows you to

convert JavaScript value to a JSON string.

Syntax: `var obj =JSON.stringify(value);`

```
<script>
  var x = [
    { code: "1001", name: "ram" },
    { code: "1002", name: "shyam" },
    { code: "1003", name: "seeta" }
  ];
  var r = JSON.stringify(x);
  document.write(r);
</script>
```


END

Thank you