JAVASCRIPT CONCEPTS

1. **getElementbyId()**
    - method is used to retrieve the HTML Element from **id** attribute
2. **Functions**
    - Block of reusable codes that perform specific tasks
3. **Events**
    - Actions / Occurrences that happen in system or document, triggered by users
    - **Example :** When the button is clicked an alert saying Button clicked is displayed
4. **Event handlers**
    - Functions that respond to events triggered by user interaction
    - **Example** : **"onclick"** property is the event handler
5. **Javascript display possibilities**
    - **innerHTML** (write content directly in JS File)
    - **document.write()** (directly displays the content in output (Similar to console.log))
    - **window.alert()** (Alert Message)
    - **console.log()** (Print)
6. **For**
    - Executes a block of code multiple times based on a specific condition or increment expression.
    - Example: A repeat button which lets you do something over and over again until a condition is met.
7. **Return**
    - Exits the current function and specifies a value to be returned
    - Example : sending a message back, when a function finishes its job. It can send back a result to whoever asked for it.
8. **Try**
    - Implements error handling by attempting to execute a block of code.
    - If an error occurs within try block it can be handled in catch block

9. **Variables**
    - Containers for storing data
    - Can be declared in **4 ways**
        - **Automatically**

- **Var**
  - Variable that can be reassigned throughout a program (i.e., you can change whatever inside the room)
- **Let**
  - Declared only in specific block of code (i.e., Similar to var, but this box only exists within the specific area , you can't take it outside the room)
- **Const**
- Variable that cannot be reassigned/redeclared
- Use const when you declare - Array, Object, Function & RegExp

10. Difference between var, let and const

	Scope	Redeclare	Reassign	Hoisted	Binds this
var	No	Yes	Yes	Yes	Yes
let	Yes	No	Yes	No	No
const	Yes	No	No	No	No

11. Operators

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

- JavaScript 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

JavaScript Logical Operators

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

JavaScript Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

- **Bitwise operators**

Operator	Description
&	AND
|	OR
~	NOT
^	XOR
<<	left shift
>>	right shift
>>>	unsigned right shift

12. Data Types

- 8 data Types
  - **String** - Represents text
  - **Number** - Numeric Value
  - **BigInt** - Whole Numbers
  - **Boolean** - True/False
  - **Undefined** - Declared a Variable without any value
  - **Null** - Absence of any object value
  - **Symbol** - Unique Identifier (‘Key’)
  - **Object** - Collection of key pair values (let person = {name:”nivi”, age:25})

13. Events

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

14. Strings

- Sequences of characters, like letters , numbers and symbols
    - Example : “Hello World!”
- **String Methods**
    - **indexOf()** - Finds the position of substring within a string
    - **String length**: Gives the number of characters in a string.
    - **String charAt():** Returns the character at a specified position in a string.
    - String charCodeAt(): Returns the Unicode value of the character at a specified index in a string.
    - String at(): Returns the character at a specified index in a string (similar to charAt()).
    - String [ ]: Accesses the character at a specified index in a string.
    - **String slice():** Extracts a section of a string and returns it as a new string *(Example: let myString = "Hello, world!";*
    - *let substring = myString.slice(7, 12);*
    - *console.log(substring); so it extracts the characters "world" from "Hello, world!".*
    - **String substr():** Gets a part of a word starting from a position. (similar to slice)
    - **String toUpperCase():** Converts a string to uppercase letters.
    - **String toLowerCase()**: Converts a string to lowercase letters.
    - **String concat()**: Merges two or more strings and returns a new string.
    - **String trim():** Removes whitespace from both ends of a string.
    - String trimStart(): Removes whitespace from the beginning of a string.

- String trimEnd(): Removes whitespace from the end of a string.
- **String padStart():** Adds characters at the beginning of the string
- **String padEnd():** Adds characters at the end of the string
- **String repeat():** Returns a new string with a specified number of copies of the string it was called on.
- **String replace():** Changes part of a word
- **String replaceAll()**: Replaces all occurrences of a specified value or pattern in a string.
- **String split():** Divides a sentence using comma (Ex: Hello, World)

- **String Search -** Finding specific substring within a string
  - String lastIndexOf() - Searches from the last occurrences
  - String search() - It searches for a word in sentence and shows where it starts
  - String match() - Searches for the first occurrence and returns only the first match found.
  - String matchAll() - Searches for all occurrences and returns only the all matches found.
  - String includes() - It checks for a specific word, if the word is found it is TRUE / if not it will be FALSE
  - String startsWith()/String endsWith() - It checks a sentence with a specific word or letter (TRUE / FALSE)

- **String Templates**
  - Instead of single or double quotes **(` `)** this is used
  - **`${ }`** - Inside this variables and expressions can be given
  - **`\n** - New Line

15. JS Numbers

- JavaScript has only one type of number. Numbers can be written with or without decimals.
- Basic operations can be performed
- Special Numbers - Infinity & Not an Number (NaN)
- Using Numbers directly in code

Number Methods

- **toFixed() -** Formats a number to specified number of decimal places

(let num = 10.56789;

console.log(num.toFixed(2)); // Output: "10.57")

- **toPrecision() -** Formats a number to a specified length including decimal places

  (let num = 123.456;

  console.log(num.toPrecision(4)); // Output: "123.5")
- **toString()** - Converts a number to a string
- **parseInt()** - Parses a string and returns an integer
- **parseFloat() -** Parses a string and returns an floating point number
- **MAX_VALUE -** Maximum number of value
- **MIN_VALUE -** Minimum number of value
- **Positive_Infinity**
- **Negative_Infinity**
- **toExponential() -** Represent numbers in a shorter format
- **valueOf()** - Gives the actual value of the number

16. Arrays

- Array is a special variable which can hold more than one value. [i.e., are like lists, each piece of data in array is called element]
- Const cars = [“Saab”, “Volvo”, “Audi”]

17. Array Methods

- Built in functions that allows to perform various operations on arrays
- Array length() - Returns the no.of.elements
- Array toString() - Converts array to a string
- Array at() - Returns the element at specified index in the array
- Array join() - Joins all elements of an array into a string, with an separator (‘,’)
- Array pop() - Removes and returns the last element from an array
- Array Push() - Add one or more elements to the end of an array & returns the new length
- Array Shift() - Removes & Returns the first element from an array *(Opposite of pop())*
- Array Unshift() - Add one or more elements at the beginning of an array *(Opposite of Push())*
- Array delete() - Removes an element from an array but leaves a hole(“undefined”) in the array
- Array Concat() - Joins two or more arrays and returns new array
- Array Copywithin() - Copies a sequence of elements within an array to another position and returns the modified array.

- Array flat() - Merges array with sub-array = output
- **Array Splice()** - changes the original array by adding, removing, or replacing elements.

  Example :

  let arr = [1, 2, 3, 4, 5];

  // We want to add 6 at index 2 without removing any elements

  arr.splice(2, 3, 6);

  console.log(arr); // Output: [1, 2, 6]

  - In the example, `splice(2, 0, 6)` means:
    - `2`: Start at index 2.
    - `3`:Remove three elements starting from index two
    - `6`: Add the number 6 at index 2.

  So, the output `[1, 2, 6]`

  - **Array slice() Method:**

  The `slice()` method creates a new array by extracting elements from a portion of an existing array. [i.e cut the start and end index]

  Example:

  let arr = [1, 2, 3, 4, 5];

  // We want to slice elements from index 1 to index 4

  let slicedArray = arr.slice(1, 4);

  console.log(slicedArray); // Output: [2, 3, 4]

  console.log(arr); // Output: [1, 2, 3, 4, 5]

[Extracts from Start index mentioned & Slices the last index in the above example]

18. **Array Search** *[i.e., similar to string search only]*
    - Array indexOf() : Finds the position of the first occurrence of a specific value in the array.
    - Array lastIndexOf() : Finds the position of the last occurrence of a specific value in the array.
    - Array includes() : Checks if the array contains a specific value, returning true or false.

- Array find() : Finds the first element in the array that meets a given condition. *(Position)*
- Array findIndex() : Finds the index of the first element in the array that meets a given condition. *(Now, let's analyze the condition element > 4. It searches for the first element in the numbers array that is greater than 4. The first element that satisfies this condition is 5, which is at index 4. Therefore, the output will be 4.) (Element)*
- Array findLast() : Returns the first **element** in the array that satisfies the provided testing function
- Array findLastIndex() : Returns the **index** of the first element in the array that satisfies the provided testing function, or -1 if not found.

19. Array Sort

- **Array sort() -** Puts array elements in order *(alphabetical order)*
- **Array Reverse()** - Flips the order of elements in the array

Sorting Objects

- **Numeric Sort** - Sorts number in ascending order
- **Random Sort** - Sorts number randomly
- **Math.min()** - Finds the smaller number in a set
- **Math.max()** - Finds the Largest number in a set

20. Array Iteration

Array forEach() - Runs a function for each element in a array

Example: let numbers = [1,2,3];

numbers.forEach(num => console.log(num*2));

Output : 2,4,6

Array map() - Creates a **new array** by applying function to each element

Example: let numbers = [1,2,3];

Let doubled = numbers.map(num => num*2);

Output : [2,4,6]

Array flatMap() - The flatMap() method first maps all elements of an array and then creates a new array by flattening the array.

```
let arr = [1, 2, 3];
let result = arr.flatMap(num => [num, num * 2]);
```

For num = 1, the function returns [1, 2].

For num = 2, the function returns [2, 4].

For num = 3, the function returns [3, 6]

Output: [1, 2, 2, 4, 3, 6]

Array filter() - It creates a new array with only the elements that pass a test.

Example : let numbers = [1,2,3,4,5,6,7,8,9,10]

Let even = numbers.filter (num => num % 2 == 0);

console.log(even);

Output : [2,4,6,8,10]

Array reduce() - It reduces the array to a single value by executing a function for each element.

Array every() - It check if all the elements in an array pass a certain condition & gives output as TRUE/FALSE

Array some() - It check if atleast one element in an array pass a certain condition

Array from() - It creates a new array from an array-like or iterable object

Array Keys() - Helps us to find the index of the elements in an array

Array entries() - It returns an iterator containing key/value for each index in the array.

Array Spread() - (`...`) lets to combine arrays, make copies, or pass elements as an individual arguments

21. **Date Get Methods -** They don't change the original date/hour/min

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

22. **Date Set Methods -** Original data can be altered

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)

23. **Math Object**

Math.abs(): Returns the distance of a number from zero.

Math.round(): Rounds a number to the nearest whole number.

Math.floor(): Rounds a number downwards to the nearest whole number.

Math.ceil(): Rounds a number upwards to the nearest whole number.

Math.min(): Finds the smallest of two or more numbers.

Math.max(): Finds the largest of two or more numbers.

Math.sqrt(): Returns the square root of a number.

Math.pow(): Raises a number to the power of another number.

Math.sin(): Returns the sine of an angle.

Math.cos(): Returns the cosine of an angle.

Math.tan(): Returns the tangent of an angle.

Math.exp(): Returns Euler's number raised to a power.

Math.log(): Returns the natural logarithm of a number.

Math.random(): Generates a random number between 0 (inclusive) and 1 (exclusive).

24. If Statement

- Executes a block of code if the condition is true
- If it is not true, it will do something else

Else

- Executes a block of code if the condition is false inside if statement

Else if

- Multiple times it can be used between if and else

25. Switch

- Executes different block of code based on different cases
- I.e., It helps your program make choices
- It's an alternate of if statement to check the same value against multiple conditions

26. For Loop

- **for -** The for loop repeats a block of code as long as a specified condition is true. It typically consists of three parts: initialization, condition, and increment/decrement.
- **for/in -** the loop iterates over each property (key) of the person object. For each iteration, it prints the key and its corresponding value using string interpolation.
- **for/of -** loops through the values of an iterable object
- **while -** loop evaluates the condition before the loop execution
- **do/while -** loop executes the loop body at least once before checking the condition.

Break and continue

- The **break statement** is like an emergency exit for loops in JavaScript. When encountered, it immediately stops the loop, no matter what, and the program continues with the next line of code after the loop.
- The **continue statement** is like a 'skip' button for loops. When it hits, it jumps to the next iteration of the loop, skipping the remaining code for the current round, and starts fresh with the next loop iteration.

27. JavaScript Label

- To label JavaScript statements you precede the statements with a label name and a colon:

28. JavaScript Iterables

An iterable in JavaScript is like a collection that you can easily loop through, grabbing each item as you go along. It's like flipping through pages in a book – you can go from one to the next until you've seen them all.

- Iterating over a String, Array, Set, Map

29. Javascript set

- a Set is like a special container that stores unique values, allowing you to keep a list of items without any duplicates.

Essential Set Methods

Method	Description
new Set()	Creates a new Set
add()	Adds a new element to the Set
delete()	Removes an element from a Set
has()	Returns true if a value exists in the Set
forEach()	Invokes a callback for each element in the Set
values()	Returns an iterator with all the values in a Set
Property	**Description**
size	Returns the number of elements in a Set

-

30. JavaScript Map

- Map is a data structure that allows you to store key-value pairs, where each key can be of any data type, and values are associated with those keys. (i.e.A Map in JavaScript is like a supercharged dictionary. It helps you store and organize information using key-value pairs, where keys can be any data type, providing a versatile way to manage and retrieve data.)

Essential Map Methods

Method	Description
new Map()	Creates a new Map
set()	Sets the value for a key in a Map
get()	Gets the value for a key in a Map
delete()	Removes a Map element specified by the key
has()	Returns true if a key exists in a Map
forEach()	Calls a function for each key/value pair in a Map
entries()	Returns an iterator with the [key, value] pairs in a Map
Property	**Description**
size	Returns the number of elements in a Map

-

31. JavaScript typeof

typeof is like a detective tool in JavaScript. It helps you figure out the type of a value or variable, telling you whether it's a number, string, boolean, or something else. It's handy when you need to know more about the data you're working with.

- **Primitive Data** (typeof "John" // Returns "string")
- **Complex Data** (typeof {name:'John', age:34} // Returns "object")
- **Constructor Property**

- Think of the constructor as a little nametag that objects wear in JavaScript. It tells you which function originally created the object. So, if you're ever curious about an object's "parent," check its constructor property
- **Difference between undefined and null**
    - **Undefined (JavaScript):**
    - Think of "undefined" like an empty backpack. It means a variable exists, but it hasn't been given a value yet. It's like having a bag, but it's still waiting to be filled with stuff.
    - **Null (JavaScript):**
    - Now, "null" is more intentional. It's like saying, "I have a bag, and I'm telling you there's nothing in it." It's a deliberate emptiness. So, while undefined is like having an empty backpack, null is like having a backpack and explicitly saying, "There's nothing inside."
    - **instanceof operator** in JavaScript is like a professional ID checker.
    - The **void operator** in JavaScript is like a silencer for expressions. When you use it, you're saying, "I want to run this, but I don't really care about the result." It's a way to perform an action without expecting anything back. It's like doing a task in the background without bothering about what it returns.

32. Type conversion

- It's like turning a number into a string or vice versa, depending on what you need. It's the language's way of being flexible, allowing you to seamlessly work with different types of data without causing too much trouble.

33. JavaScript Bitwise Operations

- 32 Bitwise Operands
- JavaScript Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>>	Zero fill right shift	Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

34. JavaScript Regular Expressions

- They're powerful tools for finding, matching, or replacing text based on specific rules. It's like having a super-smart search function for your strings.

35. Operator Precedence in JavaScript:

- Operator precedence describes the order in which operations are performed in an arithmetic expression.
- Multiplication (*) and division (/) have higher precedence than addition (+) and subtraction (-).
- **For example**, in the expression 2 + 3 * 4, the multiplication (*) has higher precedence than addition (+), so it's evaluated first, resulting in 2 + (3 * 4).

36. Javascript Errors

- **Try**- Implements error handling by attempting to execute a block of code.
- If an error occurs within try block it can handled, it can be handled in **catch** block
- The **finally** statement defines a code block to run regardless of the result.
- The **throw** statement defines a custom error.

Error Name Values

Six different values can be returned by the error name property:

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred

-

37. JavaScript Scope

Block Scope:

- If you declare a variable inside a set of curly braces (like {}), it stays inside those walls. It's like a little secret only known to the code within those curly braces.

Function Scope:

- If you declare a variable inside a function, it's accessible only within that function. It's like a VIP area – limited to the function's exclusive use.

Global Scope:

- If you declare a variable outside of any function or block, it becomes a party-goer that everyone can see and access. It's available everywhere in your script, making it the life of the programming party.

38. JavaScript Hoisting

- Hoisting is like lifting declarations to the top of the script, making variables and functions available even before they are written in the code.

39. JavaScript this Keyword

- **this**" in JavaScript is a contextual placeholder representing the current object or context, dynamically adapting its meaning based on where it's used.

40. JavaScript Arrow Function

- Arrow functions allow us to write **shorter function syntax**:
- Example: let myFunction = (a, b) => a * b;

41. JavaScript Modules

- JavaScript modules allow you to break up your code into separate files.
- **Example**

```
<script type="module">
import message from "./message.js";
</script>
```

JAVASCRIPT OBJECTS

1. Object Definitions

In JavaScript, almost "everything" is an object.

- Booleans , Numbers,Strings, Dates, Math, Regular expressions , Arrays, Functions
- All JavaScript values, except primitives, are objects.
- ***I.e., Objects can do more, but sometimes we treat primitives like objects with the new keyword. Yet, in reality, almost everything, except these basics, is treated like an object in JavaScript.***
- Create an object using Object.create().

2. Object Properties

Example:

```
let person = {
firstName: "John",
lastName: "Doe",
```

```
age: 30,
isStudent: false,
sayHello: function () {
console.log("Hello!");
  }
};
```

The person object has properties such as firstName, lastName, age, isStudent, and sayHello.

properties represent data or attributes of an object, while methods represent actions or behaviors that the object can perform.

3. **Object Methods**

```
let car = {
  brand: "Toyota",
  model: "Camry",
  year: 2022,

  // Method to display car information
  displayInfo: function() {
    console.log(`This ${this.year} ${this.brand} ${this.model} is a great car!`);
  }
};
```

When car.displayInfo() is called, it executes the method and prints: "This 2022 Toyota Camry is a great car!"

So, in this case, displayInfo is a JavaScript method that performs the action of displaying information about the car.

4. **Object Display**
   - Displaying JavaScript objects can be done using console.log() or alert() functions.
5. **Object Accessors**
   - **get Accessor**: It defines how to retrieve the value of a property.
   - **set Accessor**: It defines how to set or modify the value of a property.
   - These assessors provide a way to implement custom behavior when getting or setting the values of object properties.
6. **Object Constructors**
   - Object constructors are like blueprints for creating multiple objects with similar properties and methods.

7. **Object Prototypes**
    - Object prototypes are like recipe books for objects, letting them inherit traits and actions from a common source, so you don't have to explain the same things to each object individually. It's like a magical shortcut for creating and organizing similar objects.

JAVASCRIPT FUNCTIONS

1. **Function Definitions**
    - Block of reusable codes that perform specific tasks
    - Example: function addNumbers(a, b)
    - **Function Expressions**: Instead of defining a function with a name in advance, you create it as part of an expression.
2. **Function Parameters**
    - They are variables listed in the function definition, waiting to receive values when the function is called.
    - **Example:**

```
function greetPerson(name) {
  console.log(`Hello, ${name}!`);
}
// Calling the function with a specific value for the parameter
greetPerson("Alice"); // Output: Hello, Alice!
```

3. **Function Invocation**
    - It's the act of calling or executing a function to perform its defined task.
4. **Function Call**
    - It's the action of requesting a function to perform its defined task. When you call a function, you're asking it to execute its instructions and potentially return a result.
5. **Function Apply**
    - we use apply() to call the function with a specific context (this) and an array of arguments.
6. **Function Bind**
    - With the bind() method, an object can borrow a method from another object.

JAVASCRIPT CLASSES

1. **Class Intro**
    - JavaScript Classes are templates for JavaScript Objects.
2. **Class Inheritance**
    - Class inheritance in JavaScript allows one class to inherit properties and methods from another class.
    - To create a class inheritance, use the extends keyword.
3. **Class Static**
    - Static class methods are defined on the class itself.

JAVASCRIPT ASYNC

1. **JavaScript Callbacks**:

- They're functions passed as arguments to other functions, ready to be called when a certain task completes. It's like telling JavaScript, "Hey, when that thing is done, do this!"

2. **JS Asynchronous**

- JavaScript asynchronous operations are like multitasking. It allows tasks to be executed independently, avoiding delays and keeping the code responsive.
- **setTimeout()** is like setting an alarm clock for a function. It schedules the execution of a function once, after a specified delay (in milliseconds).
- **setInterval()** is like a repeating timer. It executes a function repeatedly at a specified interval (in milliseconds).

3. **JS Promises**

They represent success or failure and provide a cleaner syntax for handling async code.
Promises have three states:

Pending: The initial state when a promise is created, and the asynchronous operation is still in progress.

Fulfilled: The state when the asynchronous operation is successfully completed, and the promise returns a result.

Rejected: The state when the asynchronous operation encounters an error or fails, and the promise returns an error message.

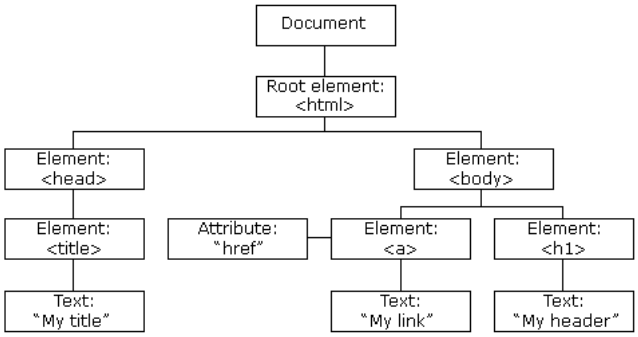
4. **JS Async/Await**
    - async and await make promises easier to write
    - async makes a function return a Promise
    - await makes a function wait for a Promise

JS HTML DOM

1. **DOM Intro**

When a web page is loaded, the browser creates a **Document Object Model** of the page.

The HTML DOM model is constructed as a tree of Objects:



With the object model, JavaScript gets all the power it needs to create dynamic HTML.

2. **DOM Methods & DOM Documents**
    - HTML DOM methods are actions you can perform (on HTML Elements).
    - HTML DOM properties are values (of HTML Elements) that you can set or change.

Finding HTML Elements

- **getElementById(id):**
- Finds an HTML element with a specific id.
- **getElementsByClassName(className):**
- Retrieves a collection of HTML elements with a specific class name.
- **getElementsByTagName(tagName):**
- Returns a collection of HTML elements with a specified tag name.
- **querySelector(selector):**
- querySelector to select the first element with ID
- **querySelectorAll(selector):**

- querySelectorAll to select all list items with class

Adding and Deleting Elements

createElement(tagName):

- Creates a new HTML element with the specified tag name.
- **appendChild(node):**
- Appends a child node to an element.
- **removeChild(node):**
- Removes a child node from an element.
- **addEventListener(event, function):**
- Attaches an event handler function to an HTML element.
- **textContent:**
- Gets or sets the text content of an element.

Changing HTML Elements

- **innerHTML:**
- Gets or sets the HTML content within an element.
- **setAttribute(attribute, value):**
- Sets the value of an attribute on an element.

3. **DOM Forms**
    - DOM forms refer to the interactive elements in a web page that allow users to input data and submit it to a server for processing.
    - **Data Validation :** Data validation is the process of ensuring that user input is clean, correct, and useful.
    - **Server side validation** is performed by a web server, after input has been sent to the server.
    - **Client side validation** is performed by a web browser, before input is sent to a web server.
4. **DOM CSS**
    - Changing CSS in the HTML Document Object Model (DOM) refers to using JavaScript to dynamically modify the styling of HTML elements, allowing for real-time adjustments to appearance and layout.
5. **DOM Events**
    - JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

6. **DOM Nodes**
    - Allowing developers to manipulate, access, and dynamically modify content, attributes, and style for interactive and responsive web pages.
7. **DOM Collections**
    - Allowing developers to perform actions or changes on multiple elements at once, such as styling all paragraphs or handling events for a group of buttons.

JS Browser BOM

1. **BOM (Browser Object Model):**

- It's like having a toolkit for interacting with the browser, enabling tasks such as opening new windows, navigating history, and accessing information about the user's environment.

2. **JavaScript Window:**

   The window is like the control center of the browser, managing the entire browser environment, including tabs, frames, and the document, offering methods to control behavior.

3. **JavaScript Screen:**

   Definition: It's like the window's assistant providing details about the user's screen, like width and height, assisting in creating responsive designs.

4. **JavaScript Location:**

   Definition: The location is like a map showing where the browser currently is, allowing developers to read or change the URL, leading to dynamic navigation.

5. **JavaScript History:**

   Definition: It's like a time-travel log, letting developers manipulate the browser's history, enabling actions like going back or forward through visited pages.

6. **JavaScript Navigator:**

   Definition: The navigator is like a helpful guide revealing details about the user's browser, operating system, and device, aiding in creating compatible web experiences.

7. **JavaScript Popup Alert:**

   Definition: It's like a friendly messenger popping up on the screen, allowing developers to convey important messages or gather user input.

8. **JavaScript Timing:**

   Definition: Timing is like having a stopwatch, providing methods to schedule actions, creating delays or repeating tasks for a more dynamic and responsive user experience.

9. **JavaScript Cookies:**

Definition: Cookies are like small notes stored by the browser, helping websites remember user preferences or login information for a personalized experience.

JavaScript Web APIs

- API stands for **Application Programming Interface.**
- **Web APIs in JavaScript act like messengers connecting your website to external services, allowing you to grab information or perform actions beyond your site. It's like reaching out to a friend for data or services on the internet.**
- **Example:** Imagine you want to show weather information on your website. You can use a weather API to fetch real-time data.

JS AJAX (Asynchronous JavaScript and XML):

AJAX in JavaScript is like having a superhero assistant for your website, allowing data to be fetched or sent to the server in the background without reloading the entire page. It's like having a seamless conversation with the server while your users enjoy a smooth and uninterrupted experience.

Remember, while the term "XML" is in AJAX, nowadays JSON is more commonly used due to its simplicity and flexibility. The example above uses JSON for data exchange.

JS JSON

- It's a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate.
- **JSON is often used to transmit data between a server and a web application, as well as between different parts of an application.**
- It's a text format that is completely language-independent but uses conventions familiar to programmers of the C family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.
- In JavaScript, you can easily work with JSON data using the
    - **JSON.parse()** method to convert a **JSON string into a JavaScript object**,
    - **JSON.stringify()** to convert a **JavaScript object into a JSON string**.
- **JSONP stands for JSON with Padding.** JSONP is a method for sending JSON data without worrying about cross-domain issues.

JQUERY

jQuery is a concise JavaScript library designed to simplify web development by providing an efficient and user-friendly approach for common tasks like DOM manipulation, event handling, and AJAX requests.

Difference between JS & JQuery

JavaScript: Preferred for complex projects, modern web development, and beyond.

jQuery: Useful for rapid development, simpler projects, or maintaining legacy codebases.

JS Graphics

- JavaScript provides a variety of tools and APIs for working with graphics on the web, allowing developers to create dynamic and visually appealing content.

Canvas API:

- The <canvas> element and associated JavaScript Canvas API enable developers to draw graphics, shapes, and images on a web page dynamically.

SVG (Scalable Vector Graphics):

- SVG is an XML-based language for describing two-dimensional vector graphics. JavaScript can manipulate SVG elements for dynamic and responsive graphics.

WebGL:

- WebGL is a JavaScript API for rendering interactive 3D and 2D graphics within a browser, providing access to the GPU for high-performance graphics.

Animation with requestAnimationFrame:

- The requestAnimationFrame function is commonly used to create smooth and efficient animations by updating the graphics on the screen at the optimal time.

D3.js (Data-Driven Documents):

- D3.js is a powerful JavaScript library for creating dynamic, data-driven visualizations in the browser. **It is commonly used for data visualization and manipulating the DOM based on data.**

THE END…..