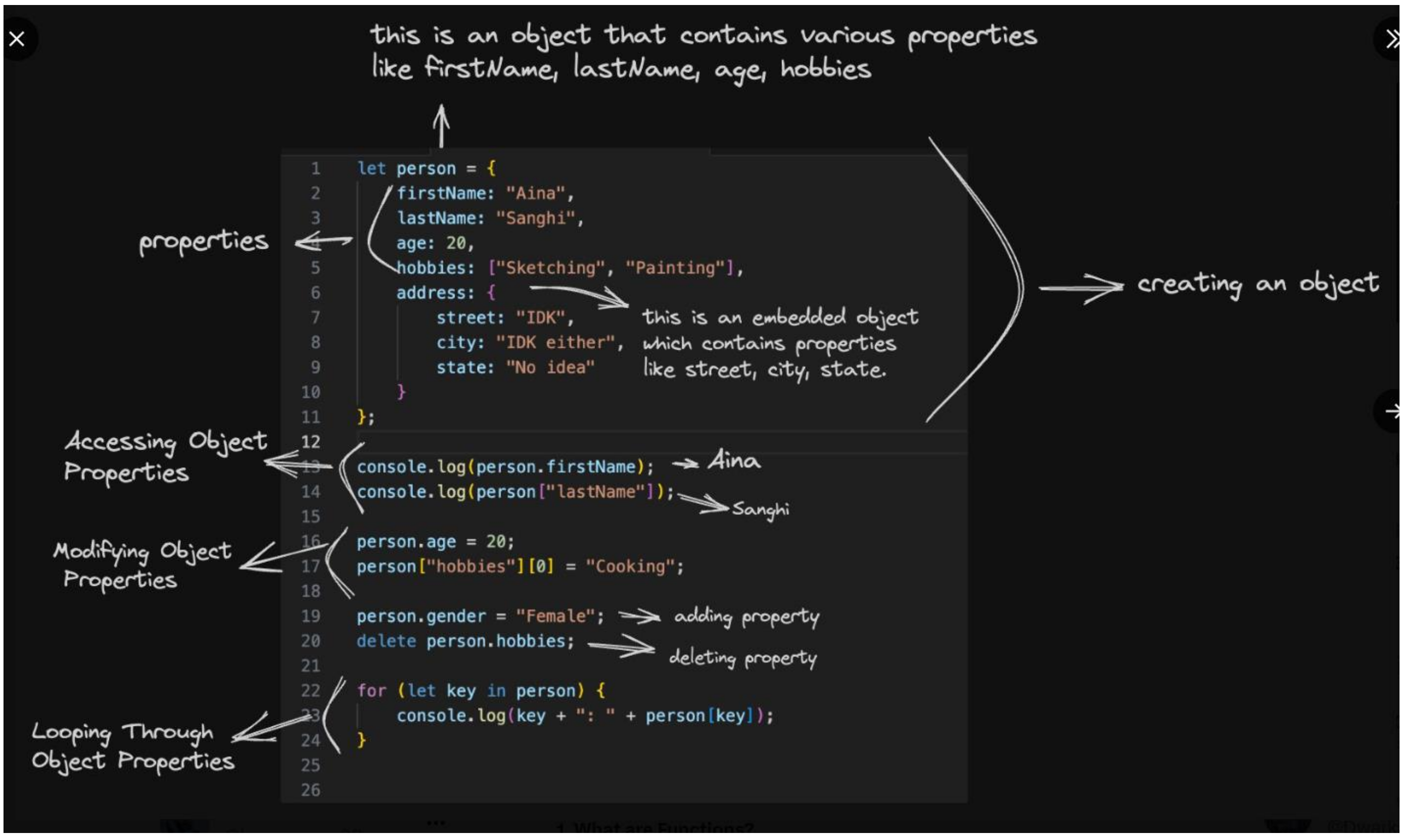
Day 6 of JS30Xplore:

1. **What are JavaScript Objects:**
   Think of objects as containers that hold information. These containers can store data in pairs: a name (key) and a value. Objects are like real-world things you want to describe or model in your code.
   2. Creating Objects To create an object, you use curly braces {}. Inside these braces, you put key-value pairs separated by colons. For example, you can create an object to describe a person
   3. Accessing Object Properties When you create an object in JavaScript, you can access its properties using dot notation or square bracket notation. Dot notation is commonly used and involves specifying the object's name, followed by a dot, and then the property's name. You can also use square bracket notation to access object properties. It involves enclosing the property name in square brackets

   4. Modifying Object Properties Objects are not static; you can modify their properties after creation. To change a property's value, you simply assign a new value to it.
   5. Adding and Deleting Properties
   You can add new properties to an object or remove existing ones at any time. To add a new property, assign a value to a non-existing property name. To delete a property, you can use the delete keyword.

   6. Looping Through Object Properties You can iterate through an object's properties using a for...in loop. This loop helps you access all properties of an object one by one.
   7. Object Methods
   In JavaScript, objects can contain functions as properties. These are known as methods. Methods allow objects to perform actions related to their purpose.
   8. Object Constructor Function In JavaScript, you can create objects using constructor functions. These functions act as templates for creating objects with specific structures.
   9. Object Prototypes Objects in JavaScript can share common properties and methods through prototypes. A prototype is like a blueprint for
   objects of a particular type. By defining properties and methods in a prototype, you make them available to all objects created from that constructor function.
   . IMAGE


properties
Accessing Object Properties
Modifying Object Properties
Looping Through Object Properties
creating an object
adding property
deleting property

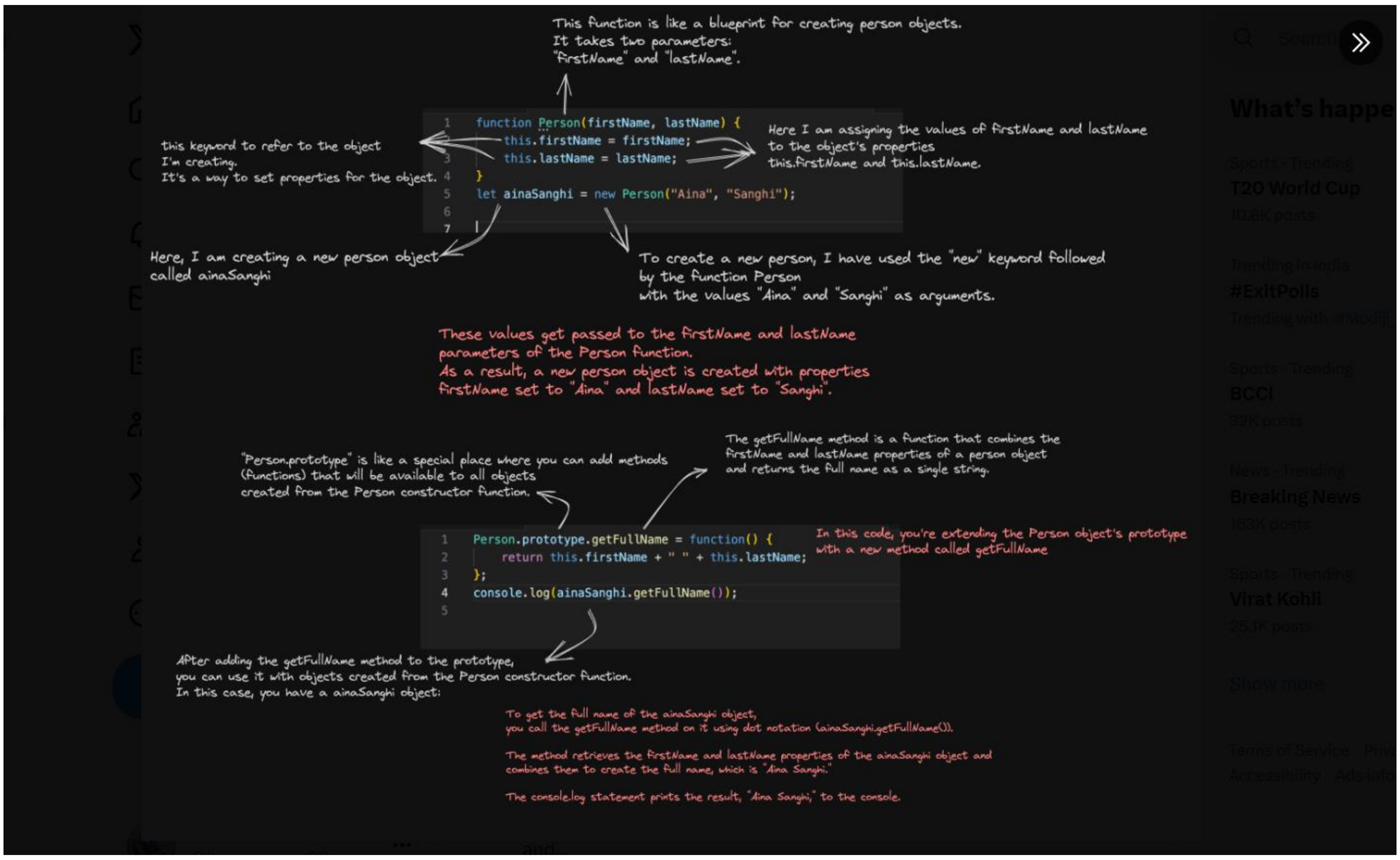
this is an object that contains various properties like firstName, lastName, age, hobbies

```javascript
let person = {
    firstName: "Aina",
    lastName: "Sanghi",
    age: 20,
    hobbies: ["Sketching", "Painting"],
    address: {
        street: "IDK",
        city: "IDK either",
        state: "No idea"
    }
};

console.log(person.firstName);
console.log(person["lastName"]);

person.age = 20;
person["hobbies"][0] = "Cooking";

person.gender = "Female";
delete person.hobbies;

for (let key in person) {
    console.log(key + ": " + person[key]);
}
```

this is an embedded object which contains properties like street, city, state.

Aina

Sanghi



This function is like a blueprint for creating person objects.
It takes two parameters:
"firstName" and "lastName".

```javascript
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}
let ainaSanghi = new Person("Aina", "Sanghi");

|
```

this keyword to refer to the object
I'm creating.
It's a way to set properties for the object.

Here I am assigning the values of firstName and lastName
to the object's properties
this.firstName and this.lastName.

Here, I am creating a new person object
called ainaSanghi

To create a new person, I have used the "new" keyword followed
by the function Person
with the values "Aina" and "Sanghi" as arguments.

These values get passed to the firstName and lastName
parameters of the Person function.
As a result, a new person object is created with properties
firstName set to "Aina" and lastName set to "Sanghi".

"Person.prototype" is like a special place where you can add methods
(functions) that will be available to all objects
created from the Person constructor function.

The getFullName method is a function that combines the
firstName and lastName properties of a person object
and returns the full name as a single string.

```javascript
Person.prototype.getFullName = function() {
    return this.firstName + " " + this.lastName;
};
console.log(ainaSanghi.getFullName());
```

In this code, you're extending the Person object's prototype
with a new method called getFullName

After adding the getFullName method to the prototype,
you can use it with objects created from the Person constructor function.
In this case, you have a ainaSanghi object:

To get the full name of the ainaSanghi object,
you call the getFullName method on it using dot notation (ainaSanghi.getFullName()).

The method retrieves the firstName and lastName properties of the ainaSanghi object and
combines them to create the full name, which is "Aina Sanghi."

The console.log statement prints the result, "Aina Sanghi," to the console.

Day 7 of JS30Xplore:

1. **What are Arrays?**
   An array is a structured collection of values, where each value is assigned a unique index. These values can be of any data type, including numbers, strings, objects, and

even other arrays. JavaScript arrays use zero-based indexing, meaning the first element is at index 0, the second at index 1, and so on. This indexing system is common in many programming languages.

2. **How to create arrays:**
   Arrays can be created using square brackets []. You can initialize them with values, or they can be empty.
3. **Accessing Array Elements:** Arrays use a zero-based index as mentioned earlier, meaning the first element is at index 0, the second at index 1, and so on. You access array elements by specifying their index inside square brackets.

4. Modifying Array Elements: You can change the value of an array element by assigning a new value to a specific index.

5. Adding and Removing Array Elements JavaScript provides methods to add and remove elements from arrays. The "push" method adds an element to the end, while the "pop" method removes the last element You can also use "unshift" to add an element to the beginning and "shift" to remove the first element.

6. Array Length: You can find the number of elements in an array using the ''length'' property.

7. Looping Through Arrays: Loops like for and modern methods like forEach help you iterate through array elements efficiently.

8. Array Destructuring: This feature allows you to extract multiple elements from an array and assign them to separate variables.

9. Array Spread Operator: The spread operator (...) allows you to copy the contents of one array into another or spread elements in function arguments.

10. Array Manipulation: Methods like map, filter, and reduce are essential for transforming and processing array data.

11. Iterating with for...of: The for...of loop simplifies array iteration by directly providing the elements, rather than indices.

12. Types of Arrays: - Static Arrays: These are the most basic types of arrays. They have a fixed length determined when the array is created. You can't add or remove elements beyond the initial length.

- Dynamic Arrays: Dynamic arrays are more flexible. They can grow or shrink in size dynamically by adding or removing elements. JavaScript arrays are typically dynamic.

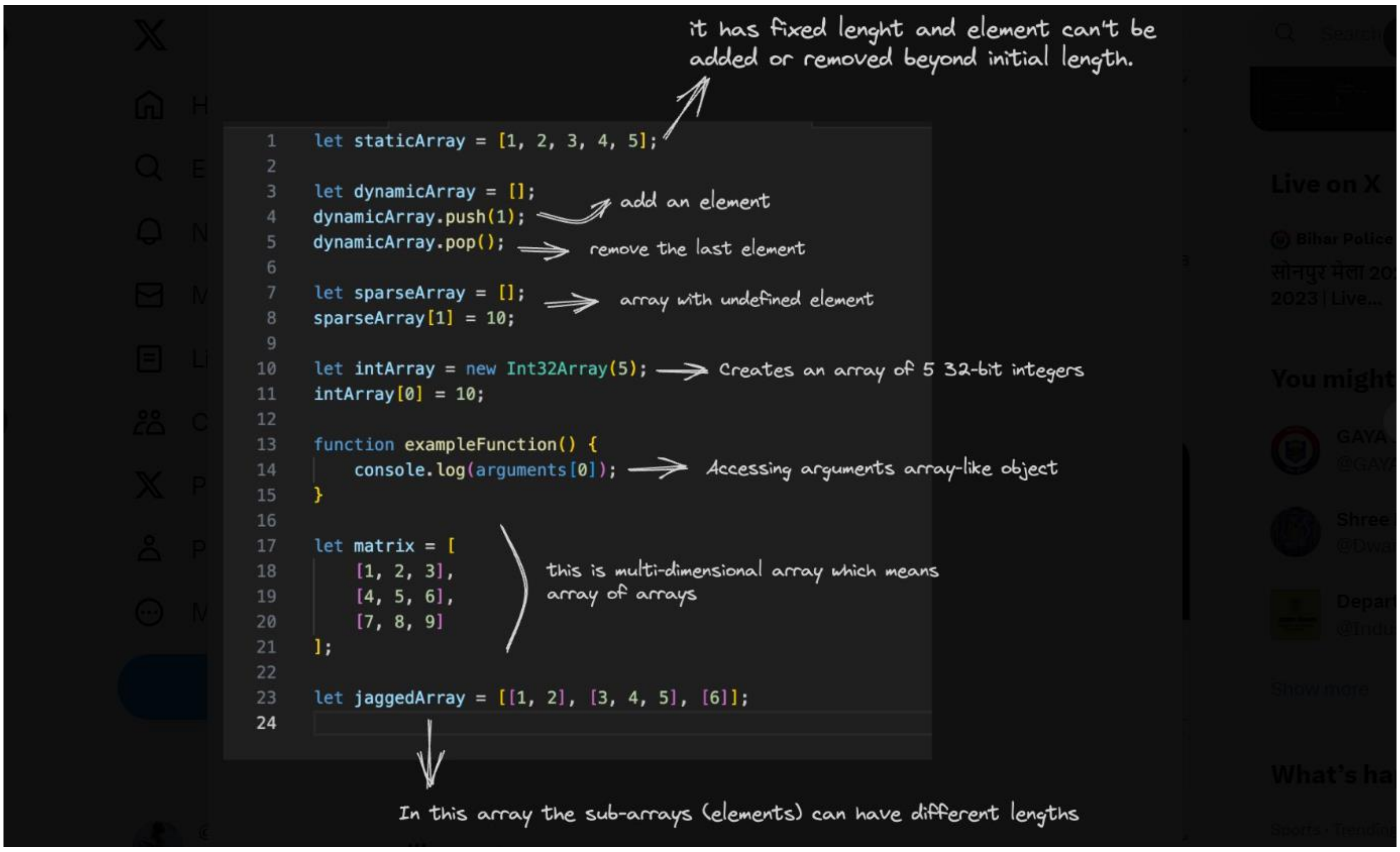
- Sparse Arrays: Sparse arrays are those with "holes" or undefined elements. They don't have consecutive index values. They are not commonly used but can be created intentionally.

- Typed Arrays: Typed arrays are designed for working with binary data and have a fixed data type for their elements. They are often used in scenarios like dealing with audio, video, or network data.

- Array-Like Objects: Some objects in JavaScript, like the ''arguments'' object or the ''NodeList'' returned by ''querySelectorAll'' , resemble arrays but are not true arrays. They have indexed properties, but they lack many array methods.

- 2D Arrays (Multidimensional Arrays): These arrays are arrays of arrays, allowing you to represent data in a grid or table format. They are useful for working with matrices or other two-dimensional data structures.

- Jagged Arrays: Jagged arrays are arrays in which the sub-arrays (elements) can have different lengths. They are often used in scenarios where data is irregular or doesn't conform to a strict grid structure.



```
let staticArray = [1, 2, 3, 4, 5];

let dynamicArray = [];
dynamicArray.push(1);
dynamicArray.pop();

let sparseArray = [];
sparseArray[1] = 10;

let intArray = new Int32Array(5);
intArray[0] = 10;

function exampleFunction() {
    console.log(arguments[0]);
}

let matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

let jaggedArray = [[1, 2], [3, 4, 5], [6]];
```

it has fixed lenght and element can't be added or removed beyond initial length.
add an element
remove the last element
array with undefined element
Creates an array of 5 32-bit integers
Accessing arguments array-like object
this is multi-dimensional array which means array of arrays
In this array the sub-arrays (elements) can have different lengths

Day 8 of JS30Xplore

1. **What is the DOM?**
   DOM stands for The Document Object Model (DOM). The DOM is like a map of a webpage that computers use to understand and change web pages. It's a way for

computer programs, like JavaScript, to talk to and change what you see on a website. The DOM is not a part of JavaScript itself but is provided by web browsers as an API (Application Programming Interface) that JavaScript can use to interact with web pages.

2. **DOM Elements and Tree Structure:** Think of a web page like a tree with branches. Each part of the page, like text, pictures, and buttons, is a branch (or node) on the tree. The top of the tree is the "Document," representing the whole page.

3. Node Types: Nodes in the DOM tree can be of various types, including elements, text nodes, attribute nodes, etc. The most commonly used node types include: - Element nodes: Represent HTML elements, like <div>, <p>, etc. - Text nodes: Represent the text within elements. - Attribute nodes: Represent attributes of elements.

4. Accessing DOM Elements: You can access DOM elements in JavaScript using various methods like "getElementById", "getElementsByClassName", "getElementsByTagName", or "querySelector".

5. Modifying DOM Elements: You can change the content, attributes, or structure of DOM elements using JavaScript. Common methods for modifying the DOM include:

- innerHTML: Changes the HTML content of an element.

- textContent: Changes the text content of an element.

- setAttribute: Sets or changes attributes of an element

6. Creating and Appending Elements: You can create new elements and append them to the DOM using functions like "createElement'' and "appendChild

7. Event Handling: You can use the DOM to set up event listeners to respond to user actions, like clicks, mouse movements, and keyboard inputs. Event listeners can be attached to DOM elements to trigger JavaScript functions when specific events occur.

8. Traversal and Selecting Elements: You can navigate and select elements within the DOM tree using properties like

''parentNode'', ''childNodes'', ''nextSibling'', ''previousSibling'', etc. The ''querySelector'' and ''querySelectorAll'' methods allow you to select elements using CSS-like selectors.

9. CSS Manipulation: You can change the styles of DOM elements by modifying their CSS properties. This can be done through the ''style'' property or by adding/removing CSS classes.

10. DOM Performance: - When manipulating the DOM, it's important to be mindful of performance, as excessive DOM manipulation can slow down web applications. - Techniques like batch updates and minimizing reflows and repaints are important for optimizing performance.

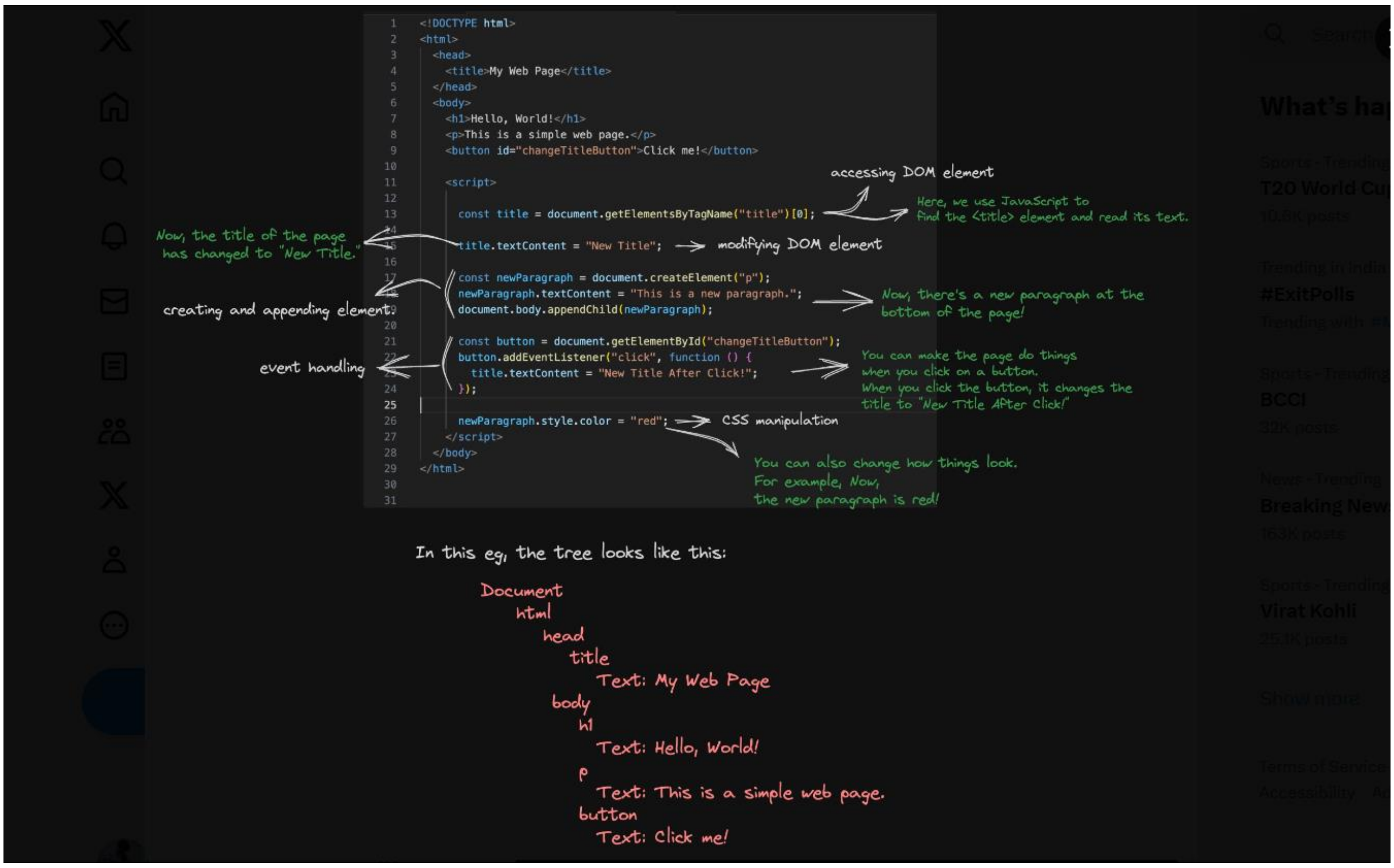
11. Cross-Browser Compatibility:

Different web browsers may have slight differences in how they implement the DOM.

Libraries like jQuery and modern frameworks often provide a layer of abstraction to handle cross-browser compatibility.

In summary, the DOM is a crucial part of web development, allowing you to interact with web page content and make it dynamic and interactive.

It is a core concept for client-side scripting with JavaScript and is essential for building modern web applications.



```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a simple web page.</p>
    <button id="changeTitleButton">Click me!</button>

    <script>

      const title = document.getElementsByTagName("title")[0];

      title.textContent = "New Title";

      const newParagraph = document.createElement("p");
      newParagraph.textContent = "This is a new paragraph.";
      document.body.appendChild(newParagraph);

      const button = document.getElementById("changeTitleButton");
      button.addEventListener("click", function () {
        title.textContent = "New Title After Click!";
      });

      newParagraph.style.color = "red";
    </script>
  </body>
</html>
```

accessing DOM element

Here, we use JavaScript to find the <title> element and read its text.

Now, the title of the page has changed to "New Title."

modifying DOM element

creating and appending element

Now, there's a new paragraph at the bottom of the page!

event handling

You can make the page do things when you click on a button. When you click the button, it changes the title to "New Title After Click!"

CSS manipulation

You can also change how things look. For example, Now, the new paragraph is red!

In this eg, the tree looks like this:

```
Document
  html
    head
      title
        Text: My Web Page
    body
      h1
        Text: Hello, World!
      p
        Text: This is a simple web page.
      button
        Text: Click me!
```

Day 9 of JS30Xplore

Selecting and Manipulating HTML elements in DOM:

1. **What does selecting elements mean?**
   Selecting an element in the DOM means using JavaScript to find a specific element in a web page. The DOM (Document Object Model) is a tree-like representation of

an HTML document, and each node in the DOM represents an HTML element attribute.
You can select elements in the DOM by their ID, class name, tag name, or CSS selector.
In order to manipulate elements in the DOM, you first need to select them.

JavaScript provides several methods to select HTML elements:
- getElementById:
This method allows you to select an element by its unique id attribute.

- getElementsByClassName:
Selects elements by their class name. It returns a collection of elements.

- getElementsByTagName:
Select elements by their HTML tag name. It also returns a collection.

- querySelector and querySelectorAll:
These methods use CSS selectors to select elements.
''querySelector'' returns the first matching element,
and ''querySelectorAll'' returns a collection of all matching elements.

2. **What does modifying elements in DOM mean?**
Modifying an element in the DOM means using JavaScript to change the properties or attributes of an HTML element.
You can modify elements in the DOM by setting their properties or attributes. For example, to change the text content of an element, you can set the "textContent" property.
To add a new class to an element, you can add the class name to the "classList" property. You can also remove elements from the DOM by calling the "remove()" method.
Modifying elements in the DOM is essential for creating dynamic and interactive web pages.
It allows you to change the content and appearance of a web page in response to user interactions or other events. Once you've selected elements, you can manipulate their content, attributes, and styles.

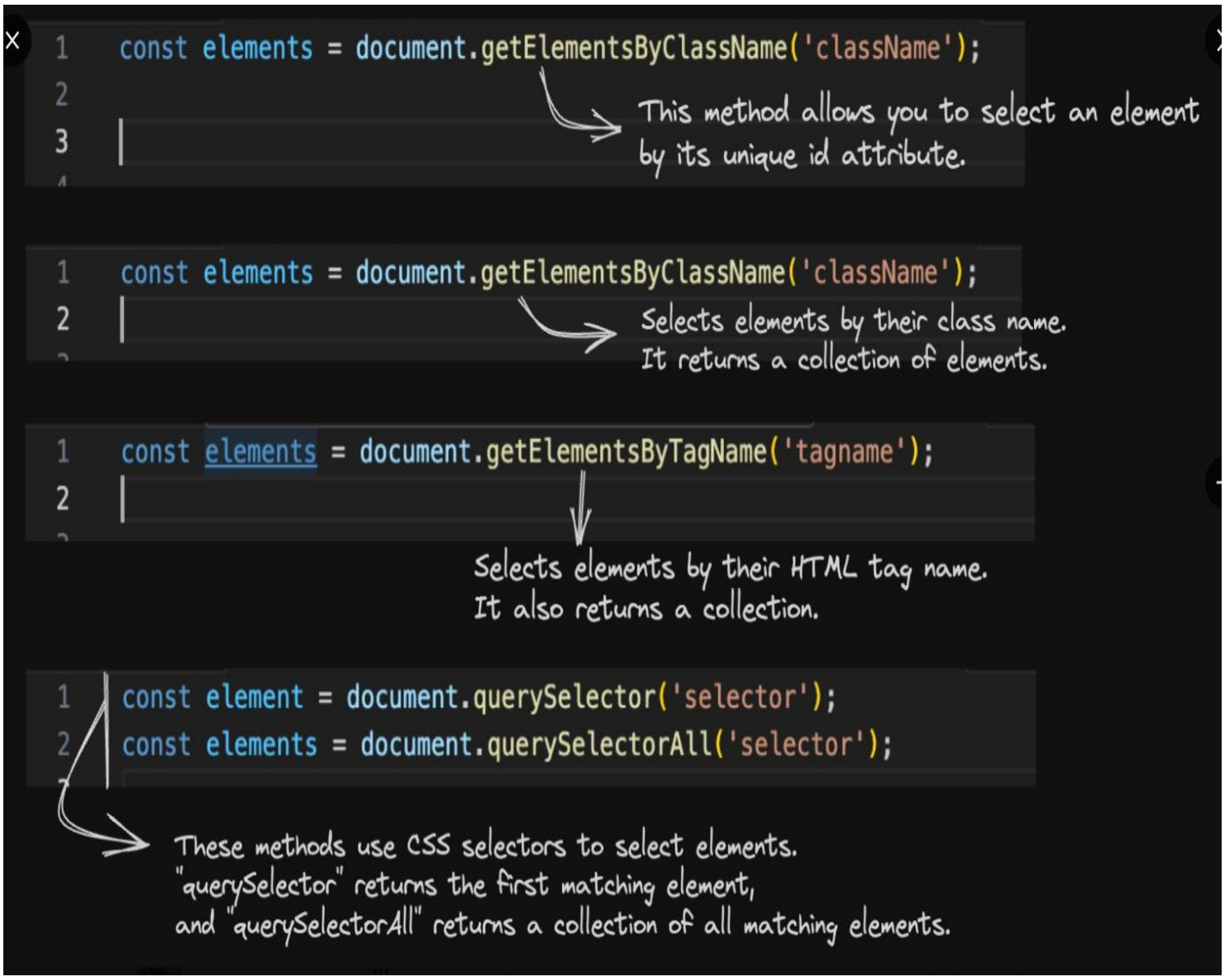
- Accessing and Modifying Element Content:

You can access and modify the content of an element using the "textContent" and "innerHTML" properties.

- Accessing and Modifying Element Attributes:
You can access and modify element attributes using the "getAttribute", "setAttribute", and "removeAttribute" methods.

- Modifying Element Styles: You can change the styles (CSS properties) of elements using the style property.
- Creating and Appending Elements: You can create new elements and append them to the DOM. This is useful for dynamically adding content to your web page.

- Removing Elements: To remove elements from the DOM, you can use the remove() method.



```
const elements = document.getElementsByClassName('className');
```

This method allows you to select an element by its unique id attribute.

```
const elements = document.getElementsByClassName('className');
```

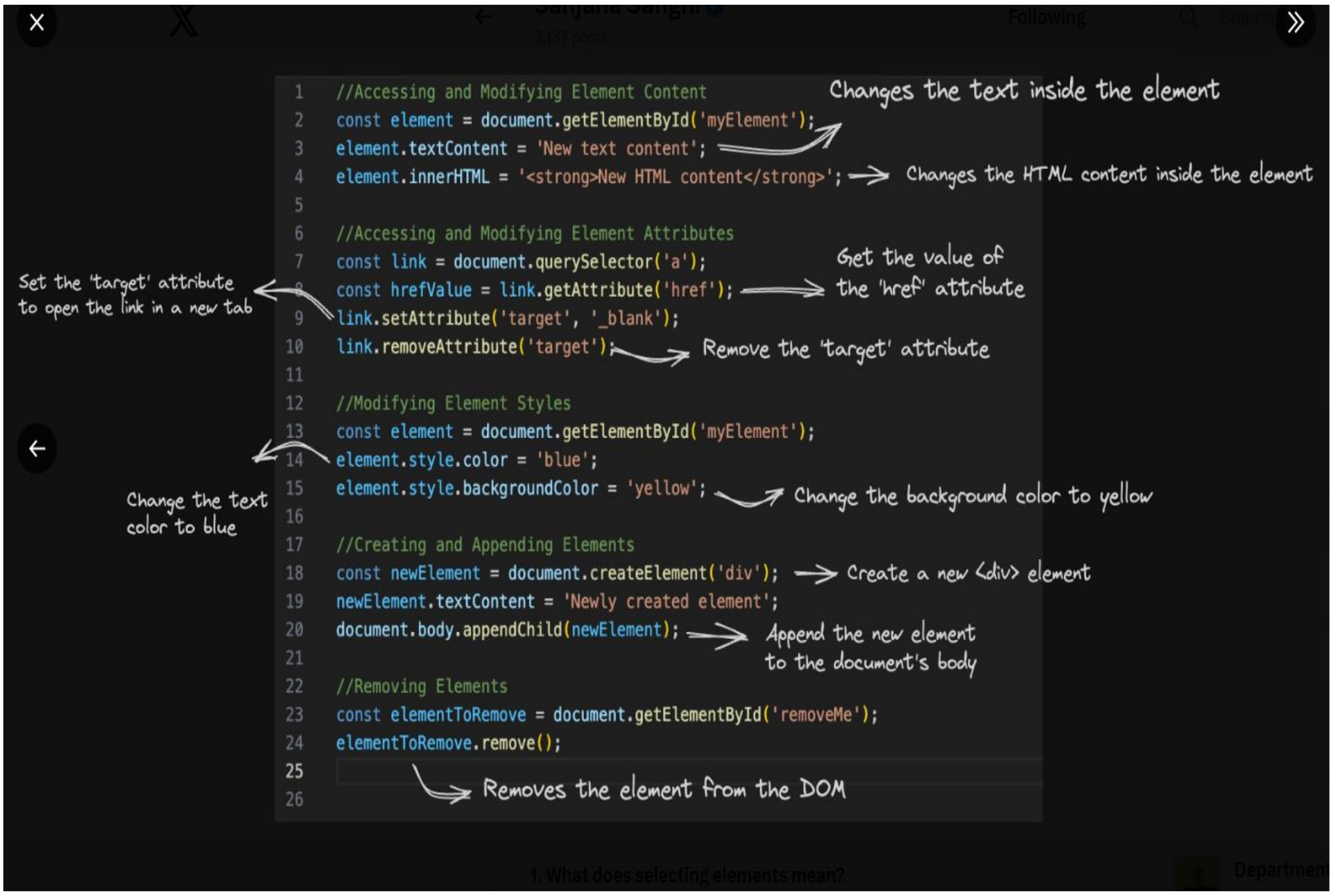
Selects elements by their class name. It returns a collection of elements.

```
const elements = document.getElementsByTagName('tagname');
```

Selects elements by their HTML tag name. It also returns a collection.

```
const element = document.querySelector('selector');
const elements = document.querySelectorAll('selector');
```

These methods use CSS selectors to select elements. "querySelector" returns the first matching element, and "querySelectorAll" returns a collection of all matching elements.


Changes the text inside the element
Changes the HTML content inside the element
Get the value of the 'href' attribute
Set the 'target' attribute to open the link in a new tab
Remove the 'target' attribute
Change the text color to blue
Change the background color to yellow
Create a new <div> element
Append the new element to the document's body
Removes the element from the DOM

```javascript
//Accessing and Modifying Element Content
const element = document.getElementById('myElement');
element.textContent = 'New text content';
element.innerHTML = '<strong>New HTML content</strong>';

//Accessing and Modifying Element Attributes
const link = document.querySelector('a');
const hrefValue = link.getAttribute('href');
link.setAttribute('target', '_blank');
link.removeAttribute('target');

//Modifying Element Styles
const element = document.getElementById('myElement');
element.style.color = 'blue';
element.style.backgroundColor = 'yellow';

//Creating and Appending Elements
const newElement = document.createElement('div');
newElement.textContent = 'Newly created element';
document.body.appendChild(newElement);

//Removing Elements
const elementToRemove = document.getElementById('removeMe');
elementToRemove.remove();
```

Day 10 of JS30Xplore:

What is DOM Tree Structure:

The Document Object Model (DOM) is a programming interface for web documents.

It represents the page so that programs can change the document structure, style, and content dynamically. The DOM represents the document as a tree of objects, where each object corresponds to part of the page.

This tree structure is called the DOM tree, and it's a fundamental concept in web development. The DOM tree structure in detail:

1. Document Object: At the top of the DOM tree is the document object, which represents the entire HTML document. It's the entry point to access and manipulate the entire document.

2. Element Nodes: HTML documents are composed of HTML elements, such as <html>, <head>, <body>, <div>, and so on. Each of these elements becomes a node in the DOM tree. These are referred to as element nodes.

3. Text Nodes: Text content within HTML elements also becomes nodes in the DOM tree. For example, the text inside a paragraph element becomes a text node. Text nodes are the leaves of the tree.

4. Attribute Nodes: Attributes of HTML elements are also represented as nodes in the DOM tree. For instance, the id and class attributes of elements are attribute nodes. They are properties of their respective element nodes.

5. Parent-Child Relationships: Elements are organized in a hierarchical structure. Each element node can have child nodes, such as other element nodes, text nodes, or attribute nodes. The element that contains another element is its parent, and the contained element is its child. This relationship creates a tree-like structure.

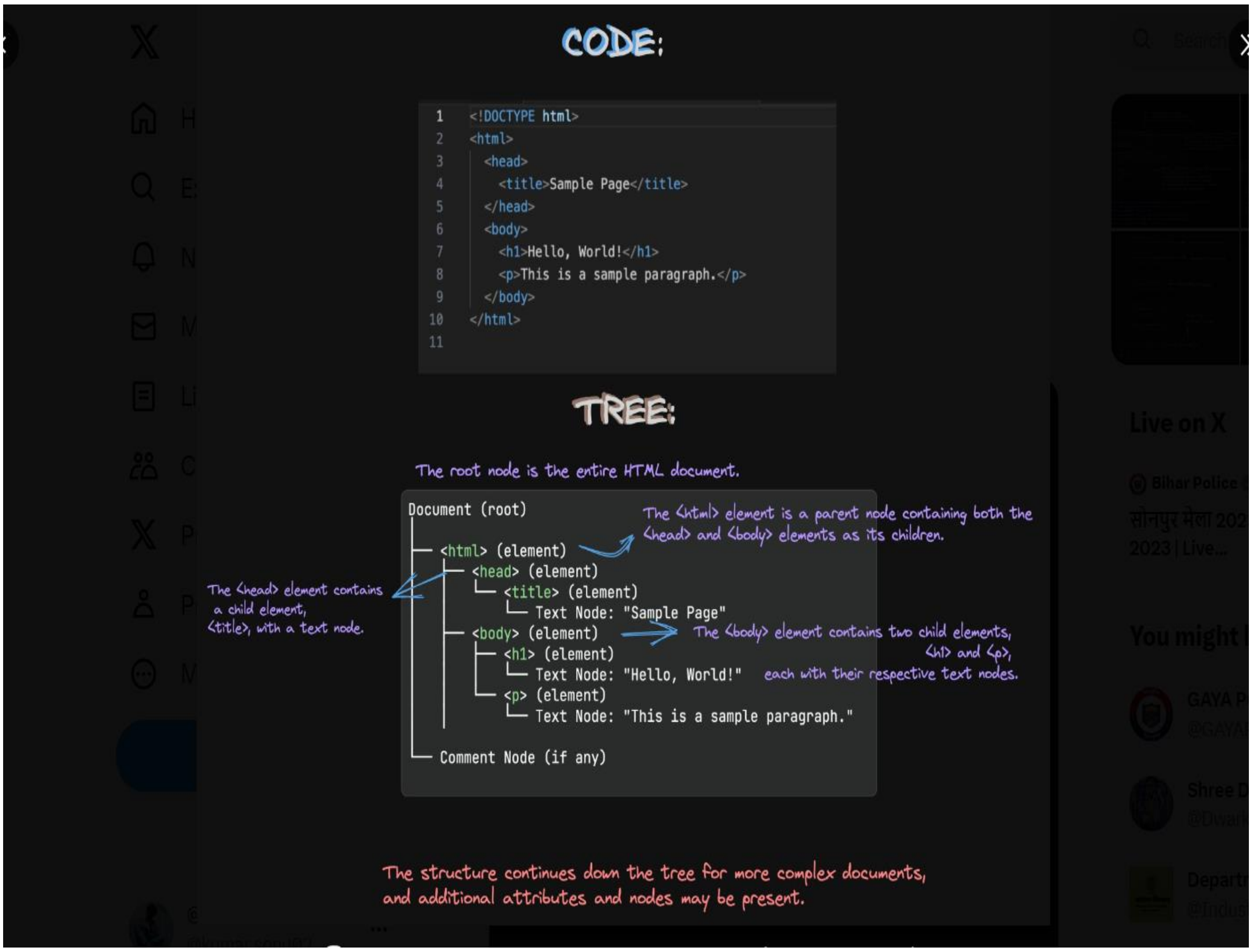
6. Siblings: Elements at the same level in the hierarchy are called siblings. They share the same parent node. For example, if you have multiple <div> elements inside the <body>, they are siblings.

7. Root Node: The root node is the top-level node in the DOM tree, representing the entire document. In an HTML document, the <html> element is the root node.

8. Traversal:

Developers can traverse the DOM tree using various properties and methods.

For example, you can access a child node from a parent node, move up to the parent from a child, or navigate to sibling nodes. In conclusion, the DOM tree structure is a hierarchical representation of an HTML document, consisting of element nodes, text nodes, and attribute nodes. It forms a tree-like structure with parent-child and sibling relationships, with the root node representing the entire document. This structure is essential for web development, enabling dynamic manipulation and interaction with web content using scripting languages like JavaScript.



CODE:

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Sample Page</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a sample paragraph.</p>
  </body>
</html>
```

TREE:

The root node is the entire HTML document.

```
Document (root)
│
├── <html> (element)
│   ├── <head> (element)
│   │   └── <title> (element)
│   │       └── Text Node: "Sample Page"
│   ├── <body> (element)
│   │   ├── <h1> (element)
│   │   │   └── Text Node: "Hello, World!"
│   │   └── <p> (element)
│   │       └── Text Node: "This is a sample paragraph."
│
└── Comment Node (if any)
```

The <html> element is a parent node containing both the <head> and <body> elements as its children.

The <head> element contains a child element, <title>, with a text node.

The <body> element contains two child elements, <h1> and <p>, each with their respective text nodes.

The structure continues down the tree for more complex documents, and additional attributes and nodes may be present.