28/3/24

ASSIGNMENT 2

Unit - 4,5 & 6. The BOM and DOM, Event Handling and Javascript APIs & Introduction to framework and its components.

Q.1 Differentiate between Real DOM and virtual DOM.

—>

Real DOM	Virtual DOM
> Real DOM represents actual structure of the webpage.	Virtual DOM represents the virtual/memory representation of the webpage.
> Slow rendering due to direct updates	Fast rendering due to virtual representation
> Direct manipulation of DOM through APIs	Manipulates the DOM using libraries like React.
> It consumes more memory	It consumes less memory.
> Slower performance for large updates	~~Faster~~ Better performance for larger updates.
> used for traditional web development	used for modern web development, particularly with React.

Q.2 "In React, everything is a component." Explain.

—> "In React, everything is a component" is a fundamental principle that encapsulates

the way React.js structures and manages user interfaces.

i) Component-Based Architecture.

React.js is built on a component-based architecture, where the UI is broken down into small, reusable pieces called component. These components can be composed together to create complex user interface.

ii) Reusable and Composable.

Components in React are highly reusable and they encapsulates both the UI and its behaviour, making them easy to understand, maintain and modify. They are also composed together (nesting components within other components) to create more complex UI structures.

iii) Hierarchical Structure

Components are organized ~~a~~ in a tree-like fashion in React, where each component represents a part of UI ~~eter~~ and they can contain other components as children.

Q.3 Explain features of React.

—4 React is a popular Javascript library for building user interfaces, known for its efficiency, flexibility and component-based architecture.

i) Component - Based Architecture

React follows a component - based architecture where the UI is broken down into reusable and independent components. They can also be composed together.

ii) Virtual DOM

React uses a virtual DOM to optimize performance. It is a light weight copy of the Real DOM, maintained in memory. When the state changes, React creates a new virtual DOM and compares it to previous to identify the differences (diffing), then these differences are applied to the Real DOM, minimizing the number of updates and improving performance.

iii) JSX (JavaScript XML)

JSX is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. JSX is transpiled into plain JS before being interpreted by the browser.

iv) React Router

It is a popular routing library for React applications which enables developers to create single - page applications with multiple views, each ~~managed~~ mapped to a specific URL

v) Unidirectional Data Flow

React follows a unidirectional data flow pattern where data flows in a single direction, from parent to child components which helps maintain a predictable state and simplifies debugging.

Q.4 State the purpose of render() function.

—ᵛ In React, the 'render()' function servers the purpose of defining what the UI should look like based on the current state and props of a component.

i) Defining UI Components: ~~It defines~~ The 'render()' function is where you define the structure of your UI components using JSX.

ii) Reacting to state and props changes: Whenever the state or props of a component change, React will re-run the 'render()' function to update the UI accordingly.

iii) Pure Function: It should be a pure function, meaning it should not modify component state, produce side effects or interact with the browser's DOM directly.

iv) Return Value: It must return a single React element, which represents the root of the component's UI tree.

v) Determining UI Structure: Inside 'render()', you can use conditional logic, loops and other expressions to determine the structure and content of the UI.

Q.5 Enlist advantages and disadvantages of React.

→ Advantages of using React are:-

i) Large Ecosystem and Community Support.
React has a large and active community, along with a vast ecosystem of libraries, tools and resources, making it easier for developers to find solutions to their problems.

ii) Strong Developer Tools:
React provides a set of powerful developer tools that aid in debugging, inspecting component hierarchies and optimizing performance.

iii) React Native
React Native allows developers to build mobile applications using React principles for both iOS and Android platform.

iv) Server-Side Rendering (SSR)
React supports SSR, allowing developers to render React components on the server and send HTML to the client.

v) Testing Capabilities

React applications are easily testable due to the component-based architecture and availability of testing libraries like Jest.

-> Disadvantages of using React are:-

i) learning Curve

React has a steep learning curve, especially for beginners with limited experience in Javascript or frontend development.

ii) Toolchain Complexity

Setting up and configuring a React development environment, including tools like webpack, Babel and ESLint, can be complex and time consuming.

iii) limited Official Guidance.

React's official documentation is comprehensive but may lack detailed guidance on certain advanced topics.

iv) Backward Compatibility

React releases updates and new features regularly, which may require developers to update their codebase and dependencies.

v) Boilerplate Code.

React may require writing more boilerplate code for complex applications.

Q.6 Explain prop and state using an example.

—> Prop and state are used for managing data and passing information between components.

i) Prop (Properties)

> Props are read-only data passed from a parent component to child component.
> They are used to customize child components and provide them with necessary data.
> They are immutable and cannot be modified by the child component.

eg:

```
// ParentComponent.js
import React from 'react';
import ChildComponent from './ChildComponent';
const ParentComponent = () => {
    const greeting = 'Hello';
    const name = 'Tony';
    return (
      <div>
        <ChildComponent greeting={greeting}
          name={name} />
      </div>
    );
};
export default ParentComponent;
```

```
// ChildComponent.js
import React from 'react';
class ChildComponent = (props) => {
    return (
        <div>
            <p>{props.greeting}, {props.name}!</p>
        </div>
    );
}
export default ChildComponent;
```

ii) State

- State represents the internal state of a component, which can be changed over time to user actions, network responses or other events.
- State is managed internally within a component and can be modified using the 'setState()' method provided by React.
- State changes trigger re-renders of the component, updating the UI to reflect the new state

eg:

```
import React, {useState} from 'react';
const Counter = () => {
    const [count, setCount] = useState(0);
    const increment = () => {
        setCount(count + 1); }
    const decrement = () => {
        setCount(count - 1); }
```

```
    return (
      <div>
        <p>Count: {count}</p>
        <button onClick={increment}>Increment</button>
        <button onClick={decrement}>Decrement</button>
      </div>
    );
}
export default Counter;
```

Q.7 Define events. Show event creation in React.

—> Events are actions or occurrences that happen in the system or in the user's interaction with the web page. Examples of events include clicking a button, hovering, typing in an input field, etc.

—> Create and handle events in React :-

```
import React from 'react';
const MyComponent = () => {
  const handleClick = () => {
    alert('Button Clicked');
  }
  return (
    <div>
      <button onClick={handleClick}>Click Me</button>
    </div>
  );
}
export default MyComponent;
```

Q.8 Explain JSX using a counter example of Javascript

-> JSX is used to define the structure of the counter application. It provides a more concise and readable syntax for creating React elements.

eg: > Given in Question - 6 (with JSX)

> Difference between the counter application using JSX and not using JSX is in the Rendering part (displaying)

> In Plain Javascript (without JSX) the rendering of counter application is done as:

```
return React.createElement('div', null,
    React.createElement('p', null, 'count: ' + count),
    React.createElement('button', {onClick: increment}, 'Increment'),
    React.createElement('button', {onClick: decrement}, 'Decrement')
);
```

Q.9 Write a note on various phases of React component lifecycle.

-> The lifecycle of a React Component refers to the series of stages that a component goes through from its initialization to its removal from the DOM. As of React 16.3, the component lifecycle is divided into 3 main phases:

i) Mounting Phase

This phase occurs when a component is being initialized and inserted into the DOM. lifecycle methods in the phase are called in the following order:

1. 'constructor()': Initializes state and blinds event handlers.
2. 'static getDerivedStateFromProps()': updates state based on changes in props.
3. 'render()': Renders the component's UI.
4. 'componentDidMount()': Executes after the component is inserted into the DOM.

ii) Updating Phase.

This phase occurs when a component's props or state change, causing it to ~~re-order~~ re-render.

1. 'static getDerivedStateFromProps()': (same)
2. 'shouldComponentUpdate()': Controls if the component should re-render.
3. 'render()': (same)
4. 'getSnapshotBeforeUpdate()': Captures information before DOM changes.
5. 'componentDidUpdate()': Executes after component updates.

iii) Unmounting Phase.

This phase occurs when a component

is being removed from the DOM.

1. 'componentWillUnmount()': Executes before component is removed from DOM, used for cleanup tasks.

Q.10 Explain Event Flow with various phases of event with diagram.

→ Event Flow refers to the sequence of steps that occur when an event is triggered on an HTML element and how it propogates through the DOM hierarchy. There are two main phases of event flow:

i) Capturing Phase.

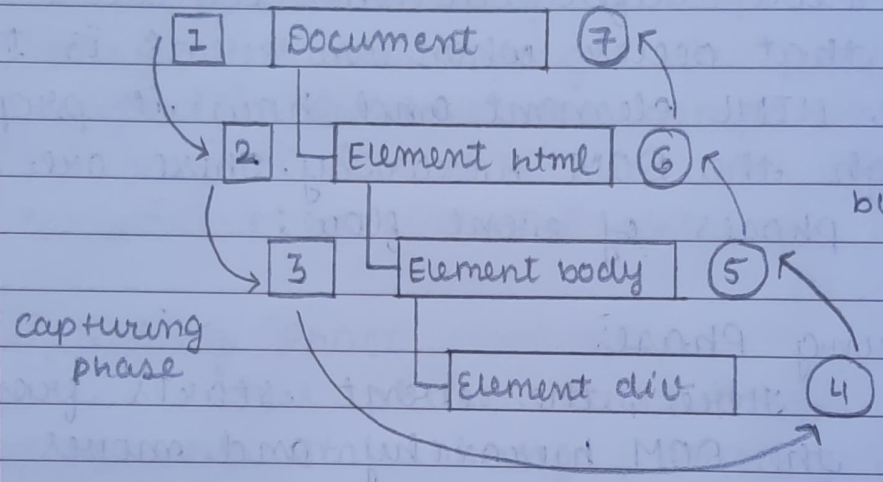
Here, the event starts from the top of the DOM hierarchy and moves to the target Element. During this phase, the browser capatures the event and triggers any event handlers registered on ancestor elementes of the target element. Event handlers on the target element itself are not triggered.

ii)* Target Phase.

Once the event reaches the target element, it enters the target phase. Event handlers registered directly on the target element are triggered during this phase.

ii) Bubbling Phase

Here, the events starts from the target element and moves upwards through the DOM hierarchy, bubbling up to the root. Event handlers on ancestor elements are triggered. Event handlers are executed from the target element towards the outermost ancestor.



Q.11 Describe various mouse and wheel events.

→ Mouse and wheel events provide ways to detect and respond to user interactions with a mouse or a scrolling wheel.

i) Click Events

> 'click()': Fired when the mouse is clicked on an element.

ii) Double click Events

> 'dblclick()': Fired when the mouse is double-clicked on an element.

iii) Mouse Button Press and Release Events.
> 'mousedown()': Fired when a mouse button is pressed down on an element
> 'mouseup()': Fired when a mouse button is released over an element.

iv) Mouse Move Event.
> 'mousemove()': ~~Fired~~ Fired when the mouse pointer is moved over an element

v) Mouse Enter and Leave Event.
> 'mouseenter()': Fired when the mouse pointer enters an element
> 'mouseleave()': Fired when the mouse pointer leaves an element.

vi) Mouse Over and Out Events
> 'mouseover()': Fired when the mouse pointer enters the boundaries of an element
> 'mouseout()': Fired when the mouse pointer leaves the boundaries of an element

vii) Mouse Wheel Events
> 'wheel()': Fired when the mouse wheel (or trackpad scroll gesture) is rotated.

Q.12 Enlist and explain properties and methods associated with HTML form element.
-> HTML form elements provide a range of properties and methods that allow developers

to interact with and manipute form elements programmatically.

- Properties :-

i) 'value': Represents the current value of the form field.
ii) 'name': Gets or sets the name attributes of the form element.
iii) 'type': Gets or sets the type of the form elements.
iv) 'disabled': Indicates whether the form element is disabled.
v) 'required': Indicates whether the form element is required.
vi) 'checked': Indicates whether a checkbox or radio button is checked.

- Methods :-

i) 'focus()': sets focus to the form element, making it ready to accept input.
ii) 'blur()': Removes focus from the form element.
iii) 'submit()': Submits the form.
iv) 'reset()': Resets the form to its initial state.

eg:
```
<form id="myForm">
  <input type="text" name="username"
  value="Tony" required>
```

<button type="submit">Submit</button>
</form>

Q.13 Describe various keyboard events.

→ Keyboard events in the provide ways to detect and respond to user keyboard interactions.

i) Key Press Event
- 'keydown': Fired when a key is pressed down.
- 'keyup': Fired when a key is released after being pressed.

i) Keydown Event
The 'keydown' event is fired when a key is pressed down. It's useful for detecting the initial press of a key and detecting held-down keys. This event provides information about which key was pressed. For example, detecting arrow key presses for navigation or game controls.

ii) Keyup Event
The 'keyup' event is fired when a key is released after being pressed. It's useful for detecting when a key is released after being pressed. This event provides information about which key was released. For example, triggering an action when a user releases the Enter key after typing in

an input field.

iii) KeyPress Event (Deprecated)

The 'keypress' event is fired when a key is pressed down and held. This event is deprecated and has limited browser support. It's recommended to use 'keydown' or 'keyup' events instead. For example, simulating typing effects in a text editor.

Q.14 Describe methods of DOM event handlers.

—> DOM event handlers are function that are executed to specific events occurring in the DOM. These event handlers can be attached to HTML elements and respond to various user interactions.

i) Inline Event Handlers

They are defined directly within the HTML markup using attributes such as 'onclick', 'onmouseover', 'onkeydown', etc.

eg: <button onclick="handleClick()">Click Me</button>

ii) DOM Element Properties

Event handlers can be assigned directly to specific properties of DOM elements.

eg: element.onclick = handleClick;

iii) 'addEventListener()' Method

The 'addEventListener()' method allows

attaching event listeners to DOM elements to handle specific events. It can be used to attach multiple event handlers for the same event type to the same element.

eg: element.addEventListener('click', handleClick);

ii) 'removeEventListener()' Method.

The 'removeEventListener()' method is used to remove event listeners that were previously added with 'addEventListener()'. It requires specifying the same event type and handler function that were used when adding the event listener.

eg: element.removeEventListener('click', handleClick);

Q.15 Explain event delegation using Javascript.

-> Event delegation is a technique where a single event listener is attached to a parent element to handle events that occur on its descendant elements. Instead of attaching event listeners to each individual child element, event delegation leverages the event bubbling mechanism to listen for events as they bubble up the DOM hierarchy.

Here's how event delegation works :-

* First, you attach an event listener to a

parent element that contains the child elements you want to target.

- When an event occurs on a ~~desere~~ descendant elements, it bubbles up through the DOM hierarchy, triggering the event listener attached to the parent element.

- Within the event handler function attached to the parent element, you can check the 'event.target' property to determine which descendant element triggered the event.

eg:

```
<html>
<head>
    <title> Event Delegation </title>
</head>
<body>
    <ul id="parentList">
        <li> Item 1
        <li> Item 2
        <li> Item 3
    </ul>
<script>
    document.getElementById('parentList').
    addEventListener('click', function(event) {
        if (event.target.tagName === 'LI') {
            const listItems = document.query
            SelectorAll('#parentList li');
            listItems.forEach(item => item.classList.
            remove('highlight');
```

```
        event.target.classList.add('highlight');
        console.log('clicked:', event.target.
        textContent);
    });
</script>
</body>
</Html>
```

Q.16 Explain bulk and stream encoding API using an example.

-> Bulk and Stream encoding are used in APIs for encoding or converting data from one format to another. They are relevant in scenarios where large amounts of data needs to be processed efficiently.

i) Bulk Encoding.

The entire input data is processed as a single unit. The entire input is loaded into memory, processed and then the entire output is generated and returned. This is applicable ~~where~~ when the processing logic requires access to the entire input dataset at once.

eg:

```
function encodeData(data) {
    const string = JSON.stringify(data);
    const encodedData = btoa(string);
    return encodedData;
}
```

```
const data = {"name": "Tony", "age": 25};
const string = encodeData(data);
console.log("Encoded Data:", string);
```

ii) Stream Encoding

The input data is processed incrementally, without loading the entire input into memory at once. The input data is sequentially processed and the output is generated incrementally. Used where the input ~~is~~ data size is large.

eg:

```
const dataStream = new writableStream({
    start(controller) {
        const data = [1, 2, 3, 4, 5];
        for(const point of data) {
            controller.enqueue(point);
        }
        controller.close();
    }
});
fetch('/api/endpoint', {
    method: 'POST',
    body: dataStream,
    headers: {
        'content-Type': 'application/octetStream'
    }
});
```

Q.17 Explain Blob API using an example.

—4 The Blob (Binary large Object) API provides a way to handle binary data, such as images, audio or other types of files. Blob represents immutable raw data in memory, and they can be used to create objects URLs, which can then be used as the src attribute of elements or as the href attribute of anchor elements.

eg:
```
const data = 'Hello, World!';
const blob = new Blob[data], {type: 'text/plain'});
const blobUrl = URL.createObjectURL(blob);
const downloadLink = document.createElement('a');
downloadLink.href = blobUrl;
downloadLink.download = 'sample.txt';
downloadLink.textContent = 'Download Sample
                            Text File';
document.body.appendChild(downloadLink);
downloadLink.addEventListener('click', () => {
    URL.revokeObjectURL(blobUrl);
});
```

Q.18 Enlist and explain all objects of BOM.

—4 The BOM (Browser Object Model) provides objects and interfaces for interacting with the browser environment. These objects allow developers to manipulate browser window, interact with the user and more.

i) Window Object

The 'window' object represents the browser window or tab. It serves as the global object in client-side Javascript, providing access to various properties and methods.

ii) Location Object

The 'location' object represents the current URL of the browser window. It provides information about the URL's components like hostname, pathname, etc.

iii) History Object

The 'history' object represents the browser's navigation history. It allows developers to navigate forward and backward in the browser's history, control the browsing session and manipulate the history stack.

iv) Navigator Object

The 'navigator' object provides information about the browser and its capabilities. It provides ~~provide~~ properties to access information about the browser environment.

v) Screen Object

The 'screen' object represents the

user's screen or monitor. It provides properties to access information about the screen's size and resolution.

Q.19 Explain Node Type and hierarchy of nodes.
Ans In the DOM, each element, attribute and piece of text in an HTML document is represented as a node. Nodes are organized in a hierarchical structure, forming a tree-like representation of the document's structure. Various node types are:-

i) Element Node: Represents an HTML element such as ' ', '<p>', '<a>', etc. Element nodes can have child nodes.

ii) Text Node: Represents textual content within an element like the text inside '<p>'.

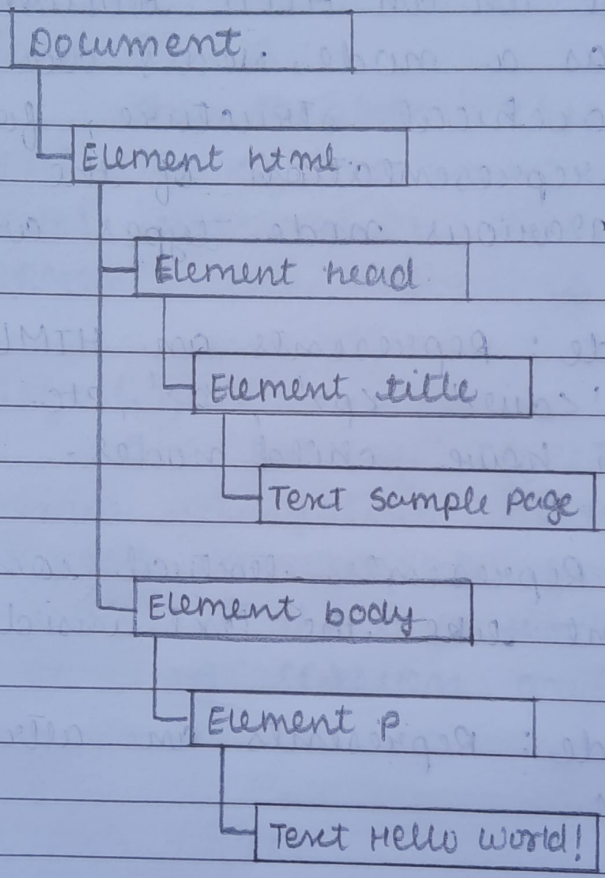
iii) Attribute Node: Represents an attribute of an element.

iv) Comment Node: Represents a comment in the HTML document, enclosed within '<!--' and '-->'.

v) Document Node: Represents the entire HTML document. It serves as the root node of the DOM Tree

vi) Document Type Node : Represents the document type declaration (`<!DOCTYPE>`).

→ The Node Hierarchy refers to the structure of the DOM, which represents the structure of an HTML document as a tree of nodes:



Q.20 How to detect user-agent using Javascript and what information it enlists ? Explain.

→ You can detect the user-agent string using the 'navigator.userAgent' property. The user-agent string provides information about the browser and operating system being used by the visitor. You can use this information to customize the behaviour

or appearance of your web application.

- The 'navigator.userAgent' property returns a string which contains details:

i) Browser Name and version.
It includes the name and version of the browser being used, eg: Safari/15.2.

ii) Operating System.
It includes information about the operating system on which the browser is running, eg: Macintosh.

iii) Device Type
It may indicate the type of device being used eg: iPhone, Android.

iv) Browser Engine
It include details about the browser's rendering engine, eg: "WebKit" for Safari.

eg:
```
const userAgent = navigator.userAgent;
if (userAgent.includes("Mobile")) {
    console.log("Mobile Detected");
}
```

Q.21 Enlist and explain various browsers and rendering engines.

-> Various web browsers use different rendering

engines to interpret and render HTML, CSS and Javascript code into a visual representation that users can interact with.

i) Safari
- Rendering Engine : WebKit
- Description : Safari is Apple's web browser, primarily used on MacOS and iOS devices. WebKit is the rendering engine developed by Apple, known for its performance, efficiency and support for web standards.

ii) Google Chrome
- Rendering Engine : Blink (a fork of WebKit)
- Description : Google Chrome is known for its speed, stability and support for modern web standards. Blink is developed by Google and used in Chrome since version 28.

iii) Mozilla Firefox.
- Rendering Engine : Gecko
- Description : Firefox is an open-source browser developed by Mozilla which emphasizes user privacy, security and customization options. Gecko used in Firefox, known for its adherence to web standards & robustness.

iv) Opera
- Rendering Engine : Blink (since version 15), Presto (pre-Blink versions)

> Description: Opera is a feature-rich web browser known for its speed, efficiency and built-in tools like ad blocker and VPN. Presto was the original rendering engine used by Opera, but since version 15, Opera has adopted Blink.

v) ~~Blink~~ Brave.

> Rendering Engine: Blink

> Description: Brave is a privacy-focused web browser known for its built-in ad and tracker blocking feature. It is based on the Chromium open-source project.

Q.22 write a Javascript to implement HTML element.

```
<table border="1" width="100%">
<tbody>
<tr>
<td> Cell 1,1 </td>
<td> Cell 2,1 </td>
</tr>
</tbody>
</table>
```

→ Javascript Code:

```
const table = document.createElement('table');
table.setAttribute('border', '1');
table.setAttribute('width', '100%');
const tbody = document.createElement('tbody');
const tr = document.createElement('tr');
```

```
const td1 = document.createElement('td');
td1.TextContent = 'Cell 1,1';
tr.appendChild(td1);
const td2 = document.createElement('td');
td2.textContent = 'Cell 2,1';
Ir.appendChild(td2);
tbody.appendChild(Ir);
table.appendChild(tbody);
document.body.appendChild(table);
```

Q.23 Parse the given a URL "https://fooUser:barPassword@www.wrox.com:80/Wiley CO/?q=javascript&num=10" and find out the parameter passed in the URL.

```
const urlString = "Above URL";
const url = new URL(urlString);
const searchParams = url.searchParams;
searchParams.forEach((value, key) => {
    console.log('${key}: ${value}');
});

// Output : q: javascript
            num: 10.
```

Q.24 Describe the requirement of mutation observer and show its implementation using Javascript.

→ Mutation Observer is a Javascript API that provides developers with a way to asynchronously observe changes to the DOM

and react to them in a performant manner.

- Requirements of Mutation Observer :-

i) Asynchronous Observation : It allows developers to observe changes to the DOM asynchronously, meaning that the browser won't be blocked while observing changes, ensuring smooth user experience.

ii) Fine - Grained Control : It provides fine-grained control over what changes to ~~ob~~ observe and how to respond to them.

iii) Efficiency : It is designed to be efficient and performant, minimizing the overhead of observing DOM mutations, even in complex web applications with large DOM trees.

iv) Cross - Browser Support : It is supported by modern web browsers, including Chrome, Firefox, Safari, Edge, and others, ensuring compatibility across different platforms.

eg: Implementation :-

```
const targetNode = document.getElementById('targetElement');
const config = { attributes: true, childList: true, subtree: true };
```

```
const callback = function (mutationList, Observer){
    for (const mutation of mutationsList) {
      if (mutation.type === 'attributes') {
        console.log ('Attributes mutated:',
                     mutation.attributeName);
      } else if (mutation.Type === 'childList') {
        console.log ('Child nodes mutated:',
                     mutation.addedNodes);
      }
    }
};
const observer = new MutationObserver (callback);
Observer.Observe (targetNode, config);
```

xxxxxxxxx