**POPUP BLOCKING**

WHAT IS THE POPUP BLOCKING?

Popup blocking is a feature in web browsers or other software that stops some kinds of pop-up windows or pop-up adverts from showing on a user's screen when they visit a website. By minimizing annoying and unnecessary interruptions when using the internet, pop-up blocking aims to enhance the user experience.

Here are key points about popup blocking:

1. Windows That Pop Up - Pop-up windows are brief browser tabs that overlay the main browser window and frequently provide adverts, alerts, or more material. Even though some pop-ups are valid and beneficial (such as alerts or login prompts), the majority are used for advertising and can be dangerous.

3. Why Popup Blocking is Important

User Experience Intrusive pop-ups can disrupt the user's browsing experience, making it difficult to find information or interact with a website.

Security Some pop-ups can contain malicious content or links that may harm a user's device or compromise their online security.

Privacy Unwanted pop-ups can collect user data without consent or direct users to phishing sites, posing privacy risks.

**Window.Open**

A method in JavaScript which allows users to open a new tab or browser window using a specified URL or webpage.

In windows URL is a string that specifies the URL of a webpage that you want to open in windows or tab.

window.open() enables you to open a new browser tab or window with a given URL or website. It can be used to open links in new tabs or windows or pop-up windows.

window.open(URL, windowName, windowFeatures);

this is how you open a new windows, URL and window features

- windowName can be optional: A string that specifies the name of the new window.

- windowFeatures A string that specifies various window features, such as its dimensions, whether it has a toolbar, status bar, …

window.open() to open a new window with a specific URL:

const newWindow = window. open('https://www.open new window ', 'openWindow', 'width=100, height=100');

**A MINIMALISTIC WINDOW**

To create a minimalistic window using the `window.open() ` method in JavaScript, you can specify the desired dimensions and minimal features.

const minimalisticWindow = window.open('https://www.minimalisticWindow.com', 'minimalWindow', 'width=100, height=200, toolbar=no, location=no,status=no,menubar=no,scrollbars=no,resizable=no');

explaining minimalistic windows

- The URL 'https://www.minimalisticWindow.com', 'minimalWindow', is loaded into the new window.

- The window is given the name `'minimalWindow'`.

- The width is set to 100 pixels, and the height is set to 200 pixels.

- Features like the toolbar, location bar, status bar, and menu bar are set to no meaning they won't be displayed.

- Scrollbars and the ability to resize the window are disabled.

**Accessing popup from window**

To access and interact with a popup window from the parent window in JavaScript, you can follow these steps:

1.Open the Popup Window

Use the window.open() method to open a new popup window and store a reference to it in a variable. You can provide a name to the popup window, which will be used to reference it.

2. Access Popup Window Properties and Methods:

You can access various properties and methods of the popup window using the reference (`popupWindow`) you created.

- To access the document of the popup window:

const popupDocument = popupWindow.document;

- To modify the content of the popup window (e.g., changing the title):

popupDocument.title = 'New Title';

- To close the popup window programmatically:

popupWindow.close();

- To execute JavaScript code within the popup window:

popupWindow.executeFunctionInPopup();

window.addEventListener('message', (event) => {

if (event.origin === 'popup-origin') {

// Handle the message from the popup

console.log('Message received from popup:', event.data);

}

});

```

In the popup window, you can send messages to the parent window:

// Assuming 'window.open' is available and refers to the parent window

window.open.postMessage('Hello from the popup!', 'parent-origin');

Ensure that you specify the correct origins when sending and receiving messages to enhance security.

4. Accessing Popup Window Elements

You can access and manipulate DOM elements in the popup window just like you would in the parent window. For example, to change the content of an element in the popup.

JavaScript

const popupElement = popupDocument.getElementById('popup-element-id');

popupElement.textContent = 'New Content';

**closing a popup**

To close a popup window in JavaScript, you can use the window.close()

To close the popup window from within the popup itself, you can simply call window.close()

window.close();

This code, when executed within the popup window, will close the popup window.

const popupWindow = window.open('popup.html', 'PopupWindow', 'width=600,height=900');

popupWindow.close();

It's important to note that there are certain restrictions and considerations when using `window.close()`

**Scrolling and resizing**

In JavaScript, you can control scrolling and resizing of a browser window or a popup window using various methods and properties. Here's how you can achieve scrolling and resizing:

window.scrollTo(0, 200); // Scroll to vertical position 200 pixels

2. Scrolling by a Specific Amount:

You can scroll the window by a specific amount (in pixels) using the `window.scrollBy()` method:

// Scroll by a certain number of pixels

window.scrollBy(0, 90); // Scroll down by 90 pixels

3. Scrolling to an Element:

To scroll to a specific DOM element, you can use the `element.scrollIntoView()` method:

const element = document.getElementById('elementId');

element.scrollIntoView();

Resizing a Window:

You can resize a browser window or popup window using the `window.resizeTo()` method:

window.resizeTo(800, 600); // Resize the window to 800x600 pixels

2. Adjusting Window Size by a Specific Amount:

You can adjust the size of a window by a specific amount (in pixels) using the `window.resizeBy()` method:

window.resizeBy(10, 20); // Increase width by 100 pixels and height by 300 pixels

**window scroll()**

**scrol**l a window to a particular place of a document.

We have a to scroll method. Which allows you to scroll to a window or element to a specific position. It can a be both vertical and horizontal scroll positions.

window.scrollTo(0, 300); // Scrolls to vertical position 300 pixels

// Scroll an element with the ID 'elementId' into view

const element = document.getElementById('elementId');

element.scrollIntoView({ behavior: 'smooth', block: 'start' });

**WINDOW FOCUS/BLUR**

a browser window or tab's focus status can be changed using the focus() and blur() functions. These techniques let you programmatically give or take away a window's attention. To react to focus shifts, you can control the focus and blur events.

1.The `focus()` method is used to give focus to a window or tab, making it the active window for user input and interactions.

window.focus();

2. The blur() method is used to remove focus from the current window or tab, making it inactive for user input.

window.blur();

3. `focus` and `blur` Events:

You can react to focus changes by using the 'focus' and 'blur' events in addition to the 'focus()' and 'blur()' methods. A window or tab receives or loses focus when one of these events is fired..

window.addEventListener('focus', function() {

console.log('Window is in focus.');

});

window.addEventListener('blur', function() {

console.log('Window lost focus.');

});

**DAY 2 CROSS WINDOW MESSAGING**

Definition of Same Origin:

Two web pages or resources are said to have the same origin if they share the same

Protocol HTTP or HTTPS

Domain (search.com)

Port ( 30, 880)

The main goal of the same-origin policy is to shield users from rogue websites that can try to steal their data or carry out unlawful actions on other websites on their behalf.

It stops JavaScript from one website from sending requests to another and reading data from that website, including cookies, local storage, or response data.

**<iframe>**

The HTML element known as an "iframe" (short for inline frame) enables you to insert another HTML page inside the one you are now viewing. In web development, iframe> elements are frequently used to embed external material, such as web pages, films, maps, or interactive widgets. JavaScript allows you to interact and manipulate iframes.

**Windows on subdomains**

Same-Origin Policy and Subdomains

- By default, JavaScript cannot access resources or make requests across different subdomains due to the same-origin restriction. Examples of distinct beginnings include "subdomain.example.com" and "example.com."

document.domain

- You can set the JavaScript property 'document.domain' to the parent domain to allow communication between subdomains. For subdomains under the same parent domain, this "relaxes" the same-origin policy.

**Iframe: wrong document pitfall**

Iframe: wrong document" When using iframes (inline frames) for web development, a pitfall is an error that can happen. Iframes are HTML elements used to insert another HTML document inside the one that is now open. When an iframe's content document has problem, the error message "Iframe: wrong document" frequently displays in the browser's developer console.

**Collection: window.frames**

Window.frames is a property used in web development that gives access to a collection of all the iframe elements present in the active window or document. A different HTML document that is embedded within the parent document is represented by each iframe in the collection. You can programmatically interact with and control these iframes using the window.frames property.

**Sandbox iframe attribute**

The sandbox property enables the exclusion of specific operations inside an iframe> to stop it from running untrusted code. By treating the iframe as originating from a different source and/or imposing additional restrictions, it "sandboxes" the iframe.

Whenusing a sandbox There is a "default set" of restrictions that apply to iframe sandbox src="...">. The restriction can be relaxed if we provide a list of restrictions that shouldn't be utilized as the value of the property, like this: iframe sandbox="allow-forms allow-popups">.

**Cross window massaging**

Purpose: Cross-window messaging enables communication between different parts of a web application or between separate web applications running in different browser windows or iframes. It's commonly used for scenarios like sharing data, coordinating actions, or syncing state between these windows.

Security: The "same-origin policy," which browsers enforce, prevents JavaScript code from one origin (for example, domain), from accessing data or running code in another origin. Bypassing these limitations while retaining security is made possible through cross-window messaging, which enables windows or iframes from various origins to communicate.

Methods:

postMessage: The most used method for cross-window messaging is the postMessage API. It allows a window or iframe to send a message to another window or iframe by specifying the target window, the message, and the target origin (the domain to which the message is intended).

otherWindow.postMessage('Hello!', 'https://example.com');

window.opener: In cases where you have a popup window opened from a parent window, you can access the parent window using window.opener and communicate between them directly.

**Day3 Clickjacking**

Clickjacking, sometimes referred to as a "UI (User Interface) redress attack" or "UI redressing," is a form of security vulnerability and attack in web applications. In a clickjacking assault, the attacker successfully hijacks the user's clicks by deceiving them into clicking on something different than what they intended to.

Clickjacking typically involves placing a transparent or disguised element (e.g., a button, link, or video) over a legit webpage or application.

The user sees and interacts with the visible content, unaware that they are interacting with the hidden, malicious element underneath.

When the user clicks or interacts with the visible content, they unintentionally trigger actions on the hidden element, which could lead to unintended consequences.

Attackers may use clickjacking to trick users into revealing sensitive information or performing actions like transferring funds, changing account settings, or submitting forms without their knowledge or consent.

Clickjacking can be used to trick users into downloading malware or performing actions that can harm their devices or compromise their security.

X-Frame-Options Header Websites can set the "X-Frame-Options" HTTP header to specify whether a page can be displayed in an iframe and under what conditions.

**Old school defenses (weak)**

Old school, weak, or out-of-date JavaScript defenses are security procedures or methods that were once employed to guard against various vulnerabilities in online applications but are now regarded as inefficient or insufficient due to developments in security research and the changing threat landscape. Though they may have been often employed in the past, these defenses are no longer advised as the principal security measures.

**Samesite cookie attribute**

The `SameSite` cookie attribute is a security feature introduced in web browsers to help mitigate certain types of cross-site request forgery (CSRF) and cross-site scripting (XSS) attacks by controlling when cookies are sent with a request. It defines how cookies should be handled when a request is made from one site to another (cross-origin requests).

A fundamental and useful component of JavaScript is the ArrayBuffer, which offers a low-level representation of binary data that may be applied in a variety of circumstances.

Due of its ability to allocate memory and directly handle binary data, ArrayBuffer is effective for managing binary data. When handling large datasets or improving code that affects performance, this efficiency is especially helpful.

Binary Data Manipulation ArrayBuffer enables developers to work with binary data at a granular level. It allows for direct manipulation of bytes, making it suitable for tasks like parsing binary file formats, networking protocols, and encryption/decryption.

ArrayBuffer is often used in conjunction with other APIs and technologies, such as TypedArrays (e.g., Uint8Array, Int16Array) and DataView, to perform various operations on binary data. This interoperability makes it a powerful tool for data processing.

Web APIs and File Handling Many web APIs and libraries use ArrayBuffer for handling binary data, such as the File API for reading files from user input. It also plays a crucial role in working with features like WebGL, WebSockets, and the Fetch API.

Typed arrayWorks pretty much the same as normal arrays but this time around you are creating binary data objects and you can then store them in an arrayBuffer. You can use typed array constructors to create those binary data objects.

You can perform a typed array without mentioning any arrayBuffer but you cannot view it without the mention of an arrayBuffer. When you want to view the arrayBuffer, you would generally need to access

Uint8Array [ 8-bit unsigned integer] >> range of values = 0 - 255, anything outside the range triggers an out-of-bound behaviour

We use Uint8ClampedArray to solve this. With this new method, any value that exceeds 255 is treated as 255 and any value that is less than 0 is treated as 0.