

WEEK 10

SPRINT

[HTTPS://ONTHEGO-RENTALS.NETLIFY.APP](https://onthego-rentals.netlify.app)



TEAM

01

SHAAKIER RAILOU

02

AQEEL HANSLO

03

AYAVUYA KOBE

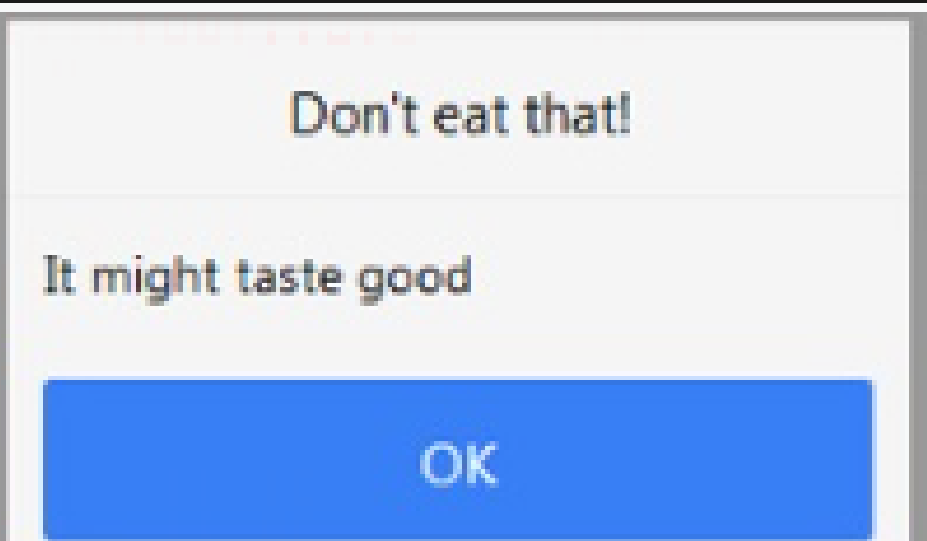
04

SAKHUMZI KWAZA

POPUPS AND WINDOW METHODS

A popup window is one of the oldest methods to show the additional documents to the user. Just run: `window.open('https://javascript.info/')` and it will open a new window with a given URL. The idea for popups was to show another content without closing the main window. Nowadays, you can do this in other ways, such as loading content dynamically with `fetch` and showing it in a dynamically generated `<div>`. So popups aren't something you use all the time.

Popup example



WINDOW.OPEN

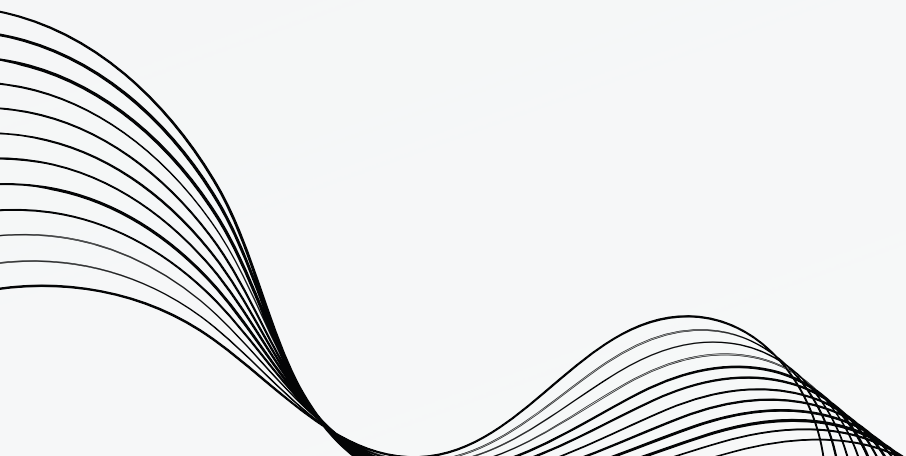
The syntax to open a window is: `window.open(url, name, params)`:

- `url` – A URL to load into the new window.
- `name` – The name of the new window. Each window has a `window.name`, and here you can specify which window to use for the popup. If there's already a window with such a name – the given URL opens in it, otherwise a new window is opened.
- `params` – The configuration string for the new window. It contains settings, delimited by a comma. There must be no spaces in `params`.

ACCESSING POPUP FROM WINDOW

The open call returns a reference to the new window. It can be used to manipulate it's properties, change location and even more.

Same origin policy – Windows may freely access content of each other only if they come from the same origin(the same protocol://domain:port). Otherwise, e.g. if the main window is from site.com, and the popup from gmail.com, that's impossible for user safety reasons.



ACCESSING WINDOW FROM POPUP

A popup may access the “opener” window as well using `window.opener` reference. It is null for all windows except popups.

So the connection between the windows is bidirectional: the main window and the popup have a reference to each other.

Closing a Popup

To close a window: `win.close()`. To check if a window is closed: `win.closed`.

Technically, the `close()` method is available for any window, but `window.close()` is ignored by most browsers if window is not created with `window.open()`. So it'll only work on a popup. The `closed` property is true if the window is closed. That's useful to check if the popup (or the main window) is still open or not. A user can close it anytime, and your code should take that possibility into account.

SAME ORIGIN

Two URLs are said to have the “same origin” if they have the same protocol, domain, and port. These URLs all share the same origin:

- `·http://site.com`
- `·http://site.com/`
- `http://site.com/my/page.html`

These ones do not:

- `·http://www.site.com` (another domain: `www.matters`)
- `·http://site.org` (another domain: `.orgmatters`)
- `https://site.com` (another protocol: `https`)

IN ACTION: IFRAME

An `<iframe>` tag hosts a separate embedded window, with its own separate document and window objects. You can access them using properties:

- `iframe.contentWindow` to get the window inside the `<iframe>`.
- `iframe.contentDocument` to get the document inside the `<iframe>`,
`iframe.contentWindow.document`.

When you access something inside the embedded window, the browser checks if the iframe has the same origin. If that's not so then the access is denied.



CROSS WINDOW MESSAGING

The `postMessage` interface allows windows to talk to each other no matter which origin they are from. So, it's a way around the "Same Origin" policy. It allows a window from `john-smith.com` to talk to `gmail.com` and exchange information, but only if they both agree and call corresponding JavaScript functions. That makes it safe for users. The interface has two parts.

`postMessage` – The window that wants to send a message calls `postMessage` method of the receiving window. In other words, if we want to send the message to `win`, we should call `win.postMessage(data, targetOrigin)`.

`onmessage` – To receive a message, the target window should have a handler on the message event. It triggers when `postMessage` is called (and `targetOrigin` check is successful).

CLICKJACKING

clickjacking is a technique used to perform actions on behalf of the user for malicious intentions.

Targeting clicks and taps,

A link is layered with an iFrame that redirects the user to another site

HOW IT OCCURS

This method tries to prevent a page from being displayed in a frame

FRAMEBUSTING

1. Blocking Top-
Navigation
2. Sandbox Attribute
3. X-Frame-Options
4. SameSite Cookie
Attribute

MORE DEFENSES

ARRAY

WHAT IS IT?



```
let buffer = new ArrayBuffer(8);  
let view = new Int32Array(buffer);  
console.log(view)
```

ArrayBuffer is used to represent a generic, fixed-length raw binary data buffer

You can't directly read or manipulate the contents of an ArrayBuffer. That's where Typed Arrays and DataViews come in.

WAYS TO ACCESS ARRAYBUFFER

- **Typed Arrays** provide a way to work with binary data in a structured manner. They views into the ArrayBuffer.
- **DataView** provides a more flexible way to read and write binary data from/to an ArrayBuffer.

Typed Arrays

- Uint8Array
- Uint16Array
- Uint32Array

FILE OBJECT

Properties:

- **name:** name of the file.
- **type:** MIME type of the file.
- **size:** Size of the file in bytes.
- **lastModified:** Timestamp of the file's last modification

Usage:

- Capturing user-selected files.
- Common in forms for file uploads.

Example:

```
document.getElementById('uploadButton').addEventListener('click', function() {
  const fileInput = document.getElementById('fileInput');
  const file = fileInput.files[0];

  if (file) {
    const formData = new FormData();
    formData.append('file', file);

    fetch('upload.php', {
      method: 'POST',
      body: formData
    })
    .then(response => {
      if (response.ok) {
        alert('File uploaded successfully.');
```

Upload a File

Choose File No file chosen

Upload File



FORM DATA



JavaScript Form Data:

- Capturing user input from HTML forms.

Fetch API:

- Modern API for making HTTP requests.
- Ideal for sending form data to the sever.

Download Progress:

- Monitoring progress when downloading data from the sever.
- Valuable for large file downloads.

Benefits:

- Simplifies handling form data.
- Allows real-time tracking of download progress.
- Enhances user experience for large file downloads.

Cross-Origin Resource

Cross-Origin Resource Sharing (CORS) is a protocol that enables scripts running on a browser client to interact with resources from a different origin.

Simple Request

- Uses these one of these methods, GET, POST, or HEAD.

```
let promise = fetch('https://www.aopa.org/training-and-s
method: "GET", //POST, PUT, DELETE, etc.
header: {
  //the content type header value is usually auto-
  "Content-Type": "text/plain;charset=UTF-8"
},

body: undefined, // string, formData, Blob, BufferSc
referrer: "about:client",
```

CORS - Why Is It Needed?

- JavaScript can't normally access resources on other origins is a good thing for security.
- "Other origins" refer to URLs that differ from the JavaScript location due to different schemes, domains, or ports.

Preflights

- Uses the option method to if the request fails to meet the simple criteria.
- If the option call fails the actual server request will not be executed.
- The preflight set mode and headers.



PATTERNS

Regular expressions are patterns that provide a powerful way to search and replace in text.

In JavaScript they are available via the RegExp object, as well as being integrated in methods of strings. There are two syntaxes that can be used to create a regular expression object.

- The “long” syntax

```
regexp = new RegExp("pattern", "flags");
```

- The "short" one

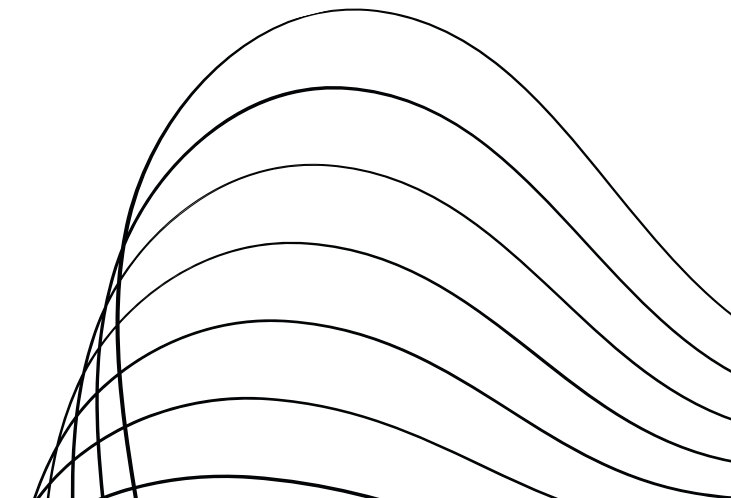
```
regexp = /pattern/; //no flags  
regexp = /pattern/gmi; //with flags g,m and i
```

FLAGS

JavaScript has six flags that affect search:

1. i- case-insensitive
2. g- searches full matches without it.
3. m- multiline mode
4. s- allows dotall matching
5. u- support unicode
6. y- searches exact position in the text.

The regexp search function is similar to a substring search without flags or special symbols.



THANK YOU

