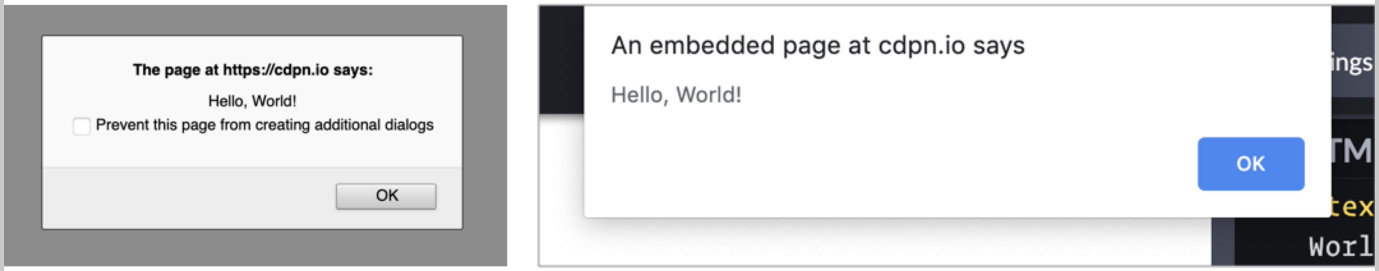
Comparing the Different Types of Native JavaScript Popups

Elliot Goldman on Oct 31, 2019 (Updated on Dec 16, 2019)

JavaScript has a variety of built-in popup APIs that display special UI for user interaction. Famously:

```
JavaScript
alert("Hello, World!");
```

The UI for this varies from browser to browser, but generally you'll see a little window pop up front and center in a very show-stopping way that contains the message you just passed. Here's Firefox and Chrome:



The page at https://cdpn.io says:

Hello, World!

Prevent this page from creating additional dialogs

OK

An embedded page at cdpn.io says

Hello, World!

OK

Native popups in Firefox (left) and Chrome (right). Note the additional UI preventing additional dialogs in Firefox from triggering it more than once. You can also see how Chrome is pinned to the top of the window.

(#aa-there-is-one-big-problem-you-should-know-about-up-front) There is one big problem you should know about up front

JavaScript popups are *blocking*.

The entire page essentially *stops* when a popup is open. You can't interact with anything on the page while one is open — that's kind of the point of a "modal" but it's still a UX consideration you should be keenly aware of. And crucially, no other main-thread JavaScript is running while the popup is open, which could (and probably is) unnecessarily preventing your site from doing things it needs to do.

Nine times out of ten, you'd be better off architecting things so that you don't have to use such heavy-handed stop-everything behavior. Native JavaScript alerts are also implemented by browsers in such a way that you have zero design control. You can't control *where* they appear on the page or *what* they look like when they get there. **Unless you absolutely need the complete blocking nature of them, it's almost always better to use a custom user interface** that you can design to tailor the experience for the user.

With that out of the way, let's look at each one of the native popups.

(#aa-window-alert) window.alert();

JavaScript

```
window.alert("Hello World");

<button onclick="alert('Hello, World!');">Show Message</button>

const button = document.querySelectorAll("button");
button.addEventListener("click", () => {
  alert("Text of button: " + button.innerText);
});
```

Embedded Pen Here

What it's for: Displaying a simple message or debugging the value of a variable.

How it works: This function takes a string and presents it to the user in a popup with a button with an "OK" label. You can only change the message and not any other aspect, like what the button says.

The Alternative: Like the other alerts, if you have to present a message to the user, it's probably better to do it in a way that's tailor-made for what you're trying to do.

If you're trying to debug the value of a variable, consider `console.log("Value of variable: ", variable);` and looking in the console.

(#aa-window-confirm) window.confirm();

JavaScript

```
window.confirm("Are you sure?");

<button onclick="confirm('Would you like to play a game?');">Ask Question</button>

let answer = window.confirm("Do you like cats?");
if (answer) {
  // User clicked OK
} else {
  // User clicked Cancel
}
```

Embedded Pen Here

What it's for: "Are you sure?"-style messages to see if the user really wants to complete the action they've initiated.

How it works: You can provide a custom message and popup will give you the option of "OK" or "Cancel," a value you can then use to see what was returned.

The Alternative: This is a very intrusive way to prompt the user. As Aza Raskin puts it (https://alistapart.com/article/neveruseawarning/) :

> *...maybe you don't want to use a warning at all."*

There are any number of ways to ask a user to confirm something. Probably a clear UI with a `<button>Confirm</button>` wired up to do what you need it to do.

(#aa-window-prompt) window.prompt();

JavaScript

```
window.prompt("What's your name?");

let answer = window.prompt("What is your favorite color?");
// answer is what the user typed in, if anything
```

Embedded Pen Here

What it's for: Prompting the user for an input. You provide a string (probably formatted like a question) and the user sees a popup with that string, an input they can type into, and "OK" and "Cancel" buttons.

How it works: If the user clicks OK, you'll get what they entered into the input. If they enter nothing and click OK, you'll get an empty string. If they choose Cancel, the return value will be `null`.

The Alternative: Like all of the other native JavaScript alerts, this doesn't allow you to style or position the alert box. It's probably better to use a `<form>` to get information from the user. That way you can provide more context and purposeful design.

(#aa-window-onbeforeunload) window.onbeforeunload();

JavaScript

```
window.addEventListener("beforeunload", () => {
  // Standard requires the default to be cancelled.
  event.preventDefault();
  // Chrome requires returnValue to be set (via MDN)
  event.returnValue = '';
});
```

Embedded Pen Here

What it's for: Warn the user before they leave the page. That sounds like it could be very obnoxious, but it isn't often used obnoxiously. It's used on sites where you can be doing work and need to explicitly save it. If the user hasn't saved their work and is about to navigate away, you can use this to warn them. If they *have* saved their work, you should remove it.

How it works: If you've attached the `beforeunload` event to the window (and done the extra things as shown in the snippet above), users will see a popup asking them to confirm if they would like to "Leave" or "Cancel" when attempting to leave the page. Leaving the site may be because the user clicked a link, but it could also be the result of clicking the browser's refresh or back buttons. You cannot customize the message.

MDN warns (https://developer.mozilla.org/en-US/docs/Web/API/WindowEventHandlers

/onbeforeunload) that some browsers require the page to be interacted with for it to work at all:

> *To combat unwanted pop-ups, some browsers don't display prompts created in beforeunload event handlers unless the page has been interacted with. Moreover, some don't display them at all.*

The Alternative: Nothing that comes to mind. If this is a matter of a user losing work or not, you kinda have to use this. And if they choose to stay, you should be clear about what they should to to make sure it's safe to leave.

(#aa-accessibility) Accessibility

Native JavaScript alerts used to be frowned upon in the accessibility world, but it seems that screen readers have since become smarter in how they deal with them. According to Penn State Accessibility (https://accessibility.psu.edu/scripts/alertboxes/) :

> *The use of an alert box was once discouraged, but they are actually accessible in modern screen readers.*

It's important to take accessibility into account when making your own modals, but there are some great resources like this post by Ire Aderinokun (https://bitsofco.de/accessible-modal-dialog/) to point you in the right direction.

(#aa-general-alternatives) General alternatives

There are a number of alternatives to native JavaScript popups such as writing your own, using modal window libraries, and using alert libraries. Keep in mind that nothing we've covered can fully block JavaScript execution and user interaction, but some can come close by greying out the background and forcing the user to interact with the modal before moving forward.

You may want to look at HTML's native `<dialog>` element. Chris recently took a hands-on look (https://css-tricks.com/some-hands-on-with-the-html-dialog-element/) at it. It's compelling, but apparently suffers (https://www.scottohara.me/blog/2019/03/05/open-dialog.html) from some significant accessibility issues. I'm not entirely sure if building your own would end up better or worse, since handling modals is an extremely non-trivial interactive element to dabble in. Some UI libraries, like Bootstrap, offer modals (https://getbootstrap.com/docs/4.0/components/modal/) but the accessibility is still largely in your hands. You might to peek at projects like a11y-dialog (https://github.com/edenspiekermann/a11y-dialog).

(#aa-wrapping-up) Wrapping up

Using built-in APIs of the web platform can seem like you're doing the right thing — instead of shipping buckets of JavaScript to replicate things, you're using what we already have built-in. But there are serious limitations, UX concerns, and performance considerations at play here, none of which land particularly in favor of using the native JavaScript popups. It's important to know what they are and how they can be used, but you probably won't need them a heck of a lot in production web sites.