

Tooltip & Popover Positioning Engine

  

\*\*Positioning tooltips and popovers is difficult. Popper is here to help!\*\* Given an element, such as a button, and a tooltip element describing it, Popper will automatically put the tooltip in the right place near the button. It will position \_any\_ UI element that "pops out" from the flow of your document and floats near a target element. The most common example is a tooltip, but it also includes popovers, drop-downs, and more. All of these can be generically described as a "popper" element. ## Demo [![Popper visualized](https://i.imgur.com/F7qWsmV.jpg)](https://popper.js.org) ## Docs - [v2.x (latest)](https://popper.js.org/docs/v2/) - [v1.x](https://popper.js.org/docs/v1/) We've created a [Migration Guide](https://popper.js.org/docs/v2/migration-guide/) to help you migrate from Popper 1 to Popper 2. To contribute to the Popper website and documentation, please visit the [dedicated repository](https://github.com/popperjs/website). ## Why not use pure CSS? CSS tooltips have accessibility and usability problems: - \*\*Clipping and overflow issues\*\*: CSS tooltips will not be prevented from overflowing clipping boundaries, such as the viewport. The tooltip gets partially cut off or overflows if it's near the edge since there is no dynamic positioning logic. When using Popper, your tooltip will always be positioned in the right place. - \*\*No flipping\*\*: CSS tooltips will not flip to a different placement to fit better in view if necessary. Popper automatically flips the tooltip to make it fit in view as best as possible for the user. - \*\*Using HTML\*\*: Popovers containing interactive HTML are difficult or not possible to create without UX issues using pure CSS. Popper positions any HTML element – no pseudo-elements are used. - \*\*No virtual positioning\*\*: CSS tooltips cannot follow the mouse cursor or be used as a context menu. Popper allows you to position your tooltip relative to any coordinates you desire. - \*\*Lack of extensibility\*\*: CSS tooltips cannot be easily extended to fit any arbitrary use case you may need to adjust for. Popper is built with extensibility in mind. ## Why Popper? With the CSS drawbacks out of the way, we now move on to Popper in the JavaScript space itself. Naive JavaScript tooltip implementations usually have the following problems: - \*\*Scrolling containers\*\*: They don't ensure the tooltip stays with the reference element while scrolling when inside any number of scrolling containers. - \*\*DOM context\*\*: They often require the tooltip move outside of its original DOM context because they don't handle `offsetParent` contexts. - \*\*Configurability\*\*: They often lack advanced configurability to suit any possible use case. - \*\*Size\*\*: They are usually relatively large in size, or require an ancient jQuery dependency. - \*\*Performance\*\*: They often have runtime performance issues and update the tooltip position too slowly. \*\*Popper solves all of these key problems in an elegant, performant manner.\*\* It is a lightweight ~3 kB library that aims to provide a reliable and extensible positioning engine you can use to ensure all your popper elements are positioned in the right place. When you start writing your own popper implementation, you'll quickly run into all of the problems mentioned above. These widgets are incredibly common in our UIs; we've done the hard work figuring this out so you don't need to spend hours fixing and handling numerous edge cases that we already ran into while building the library! Popper is used in popular libraries like Bootstrap, Foundation, Material UI, and more. It's likely you've already used popper elements on the web positioned by Popper at some point in the past few years. Since we write UIs using powerful abstraction libraries such as React or Angular nowadays, you'll also be glad to know Popper can fully integrate with them and be a good citizen together with your other components. Check out `react-popper` for the official Popper wrapper for React. ## Installation ### 1. Package Manager ```bash # With npm npm i @popperjs/core # With Yarn yarn add @popperjs/core ``` ### 2. CDN ```html ``` ### 3. Direct Download? Managing dependencies by "directly downloading" them and placing them into your source code is not recommended for a variety of reasons, including missing out on feat/fix updates easily. Please use a versioning management system like a CDN or npm/Yarn. ## Usage The most straightforward way to get started is to import Popper from the `unpkg` CDN, which includes all of its features. You can call the `Popper.createPopper` constructor to create new popper instances. Here is a complete example: ```html I'm a button

I'm a tooltip

``` Visit the [tutorial](https://popper.js.org/docs/v2/tutorial/) for an example of how to build your own tooltip from scratch using Popper. ### Module bundlers You can import the `createPopper` constructor from the fully-featured file: ```js import { createPopper } from '@popperjs/core'; const button = document.querySelector('#button'); const tooltip = document.querySelector('#tooltip'); // Pass the button, the tooltip, and some options, and Popper will do the // magic positioning for you: createPopper(button, tooltip, { placement: 'right', }); ``` All the modifiers listed in the docs menu will be enabled and "just work", so you don't need to think about setting Popper up. The size of Popper including all of its features is about 5 kB minzipped, but it may grow a bit in the future. #### Popper Lite (tree-shaking) If bundle size is important, you'll want to take advantage of tree-shaking. The library is built in a modular way to allow to import only the parts you really need. ```js import { createPopperLite as createPopper } from "@popperjs/core"; ``` The Lite version includes the most necessary modifiers that will compute the offsets of the popper, compute and add the positioning styles, and add event listeners. This is close in bundle size to pure CSS tooltip libraries, and behaves somewhat similarly. However, this does not include the features that makes Popper truly useful. The two most useful modifiers not included in Lite are `preventOverflow` and `flip`: ```js import { createPopperLite as createPopper, preventOverflow, flip } from "@popperjs/core"; const button = document.querySelector('#button'); const tooltip = document.querySelector('#tooltip'); createPopper(button, tooltip, { modifiers: [preventOverflow, flip], }); ``` As you make more poppers, you may be finding yourself needing other modifiers provided by the library. See [tree-shaking](https://popper.js.org/docs/v2/tree-shaking/) for more information. ## Distribution targets Popper is distributed in 3 different versions, in 3 different file formats. The 3 file formats are: - `esm` (works with `import` syntax — \*\*recommended\*\*) - `umd` (works with `