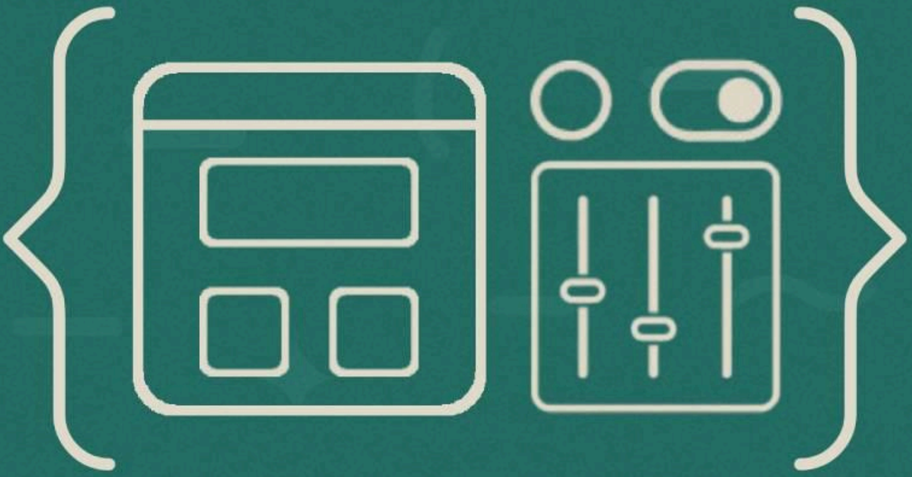
YOU DON'T NEED JAVASCRIPT

PREVIEW



A practical guide to creating modern websites and interfaces using only CSS

by

THEO SOTI

Preview Edition - You Don’t Need JavaScript

Thank you for downloading this **preview edition** of the book.
This sample includes the introduction and chapter `II.3` only. It is designed to give you a sense of the style, the approach, and the kind of practical knowledge you can expect in the full version.

If you find this preview useful and want to continue learning, the complete book includes over 120 pages of advanced techniques, practical examples, and future-facing CSS features.

To access the **full edition**, please visit: https://theosoti.com/you-dont-need-js/

Introduction

JavaScript is powerful. No argument there. But what if much of what you're solving with JS could be handled just as well, or even better, using modern CSS and semantic HTML? This book is based on a simple principle: use the least powerful tool that gets the job done.

The Rule of Least Power

This principle originates from the philosophy behind web standards: always choose the simplest tool that effectively accomplishes the job. Start with semantic HTML to structure your content, enhance presentation and interactions with CSS, and reach for JavaScript only when absolutely necessary.

HTML and CSS are declarative languages. You describe your intent (the final look and behavior) and the browser determines the most efficient way to execute it. This approach inherently leads to simpler, more maintainable, and more accessible websites. JavaScript, however, is imperative. It demands that you explicitly define every step the browser must take, increasing complexity, performance overhead, and potential points of failure.

Why Do We Default to JavaScript?

It's easy and tempting to reach for JavaScript. Popular frameworks such as React, Vue, and Angular have made rapid prototyping and rich interactivity extremely convenient. While this convenience has significant advantages, it also often leads to bloated bundles, slower load times, and unnecessary complexity.

On the other hand, browsers have significantly evolved. Modern CSS and HTML have caught up remarkably, offering powerful capabilities once reserved solely for JavaScript. You can now implement sophisticated UI patterns such as modals, accordions, carousels, toggles, and animations purely through native browser technologies.

Why it Matters

Using minimal JavaScript translates to leaner codebases, improved accessibility, better performance, and reduced reliance on third-party libraries. Moreover, it results in decreased energy consumption, both for users and servers, significantly lowering your site's environmental footprint.

However, the aim isn't to entirely remove JavaScript. Certain tasks like real-time data fetching, complex business logic, advanced interactivity, will always benefit from JS. The goal is balance: employ JavaScript exactly where it provides genuine value, not simply because it's familiar or convenient.

By embracing a CSS-first approach, you enhance accessibility inherently. Semantic HTML and native browser behaviors are optimized for assistive technologies, supporting a broader audience by default.

Performance and Sustainability

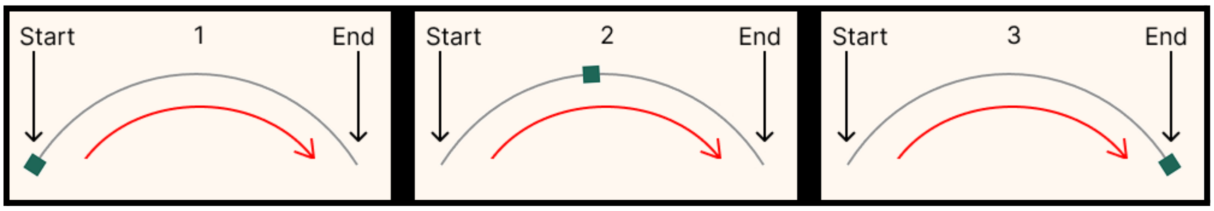
A lighter reliance on JavaScript also means faster initial load times, reduced CPU usage, and less battery drain. For instance, substituting a JavaScript slider library (which typically weighs 50-100KB of minified JS) with a pure CSS implementation reduces overhead and significantly boosts performance.

This book invites you to pause before reaching for JavaScript. Always ask yourself first: Can CSS handle this? You'll find that the answer is "yes" more often than you'd expect, empowering you to build faster, cleaner, and more accessible websites from the start.

II.3 - CSS Motion Path

Motion Path is the name of the specification that introduces `offset-path`, `offset-distance`, and related properties. Together, these let you describe a shape (the path) and move an element along it, in pure CSS.

We'll keep this as small as possible: one container, one SVG path, one square. No extra styling, no helper wrappers.



Final result: going from left to right following the curve.

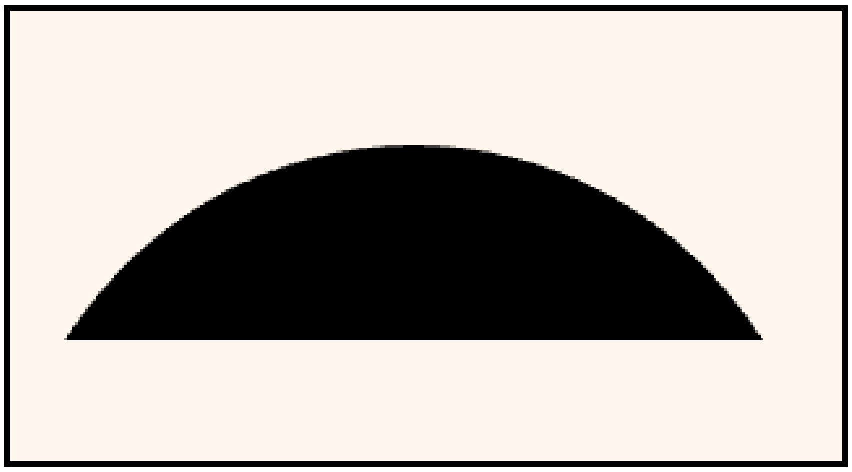
The Basic HTML Structure

We'll start by creating an SVG path for the element to follow. In this example we're using a curve, but the path could be any shape as long as it's defined in the SVG.

```html
<div class="container">
  <svg viewBox="0 0 300 140" aria-hidden="true">
    <path d="M16,120 C80,20 220,20 284,120" />
  </svg>

  <div class="box"></div>
</div>
```

- `<svg>` is the "map" area, with a size of 300 x 140 units.
- `<path>` draws a curve (a C-shaped line) inside that map (you can use a tool like this to create your own path).
- `<div class="box">` is the square we'll animate later. In practice, this can be any element: an icon, an image, a button, or even a more complex component.


The raw svg we drew.

Draw the route

At this point the path exists, but by default it shows up as a filled black shape (because SVG paths have a black fill unless you override it). Here we just need the curve to be visible as a line, like a track.

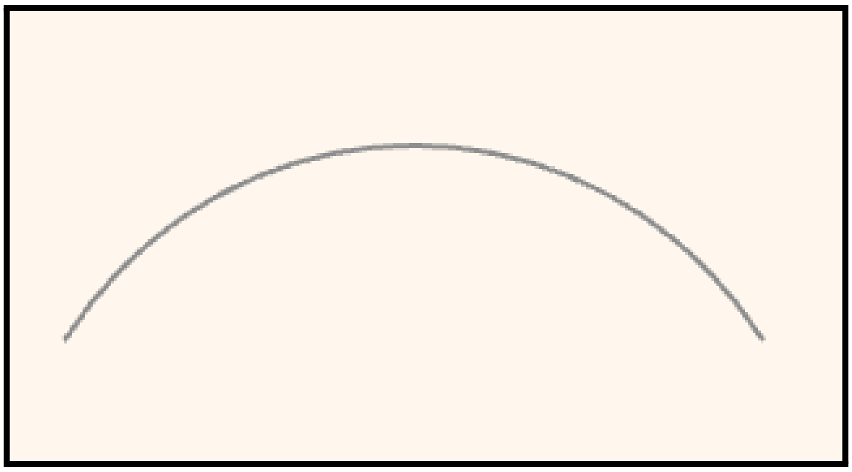
First, we make sure the SVG itself stretches exactly over the container:

```css
svg  {
  position: absolute;
  inset: 0;
  width: 100%;
  height: 100%
}
```

Then we style the path itself:

```css
path {
  fill: none;
  stroke: #999;
  stroke-width: 2;
}
```

You should now see a thin gray curve across the container. This curve is the track for the motion path.


Only the route displayed.

Make the Container Match the SVG dimensions

Here's a subtle but critical detail: the container (here `<div class="container">`) must match the SVG's dimensions. Why? Because the Motion Path positions the moving element relative to the container. If your SVG defines its drawing area with `viewBox="0 0 300 140"`, that means the path is drawn on a grid that is 300px wide and 140px tall.

To make the path and the moving element line up, the container should be set to the same size:

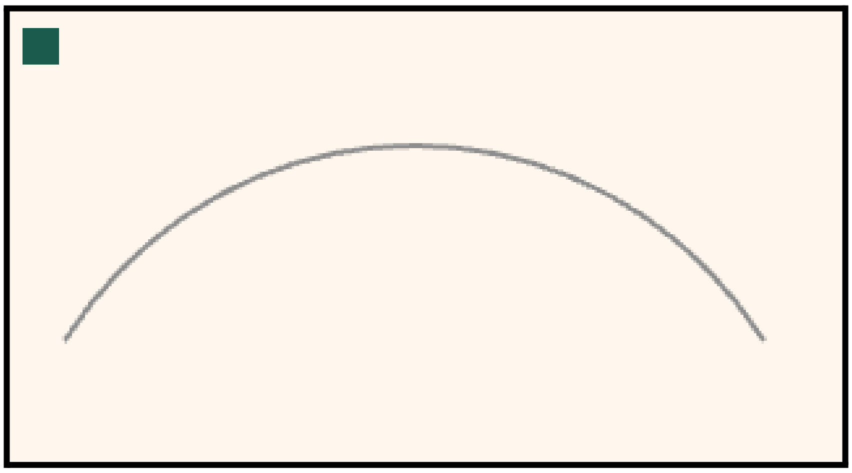
```
.container {
  position: relative;
  width: 300px;
  height: 140px;
}
```

If the sizes don't match, the square will still move, but it won't appear to follow the visible curve.

Draw the square

Let's create a square 14px wide. By default, the motion path positions the element's center on the curve, so the square will sit neatly on the path without extra alignment.

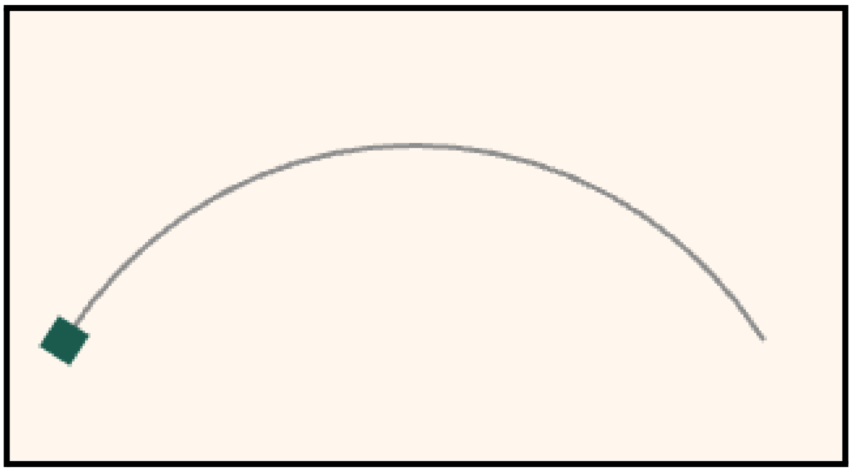
```css
.box {
  position: absolute;
  top: 0;
  left: 0;
  width: 14px;
  height: 14px;
  background: #e6007a;
}
```


The square is here, but we have to place it correctly.

Give it a path to follow

Paste the same `d` string from the SVG into `offset-path: path('...')`. Reusing the identical numbers keeps animation and drawing perfectly aligned. Also set `offset-distance: 0%;` so that the square starts at the beginning of the path.

```css
.box {
  offset-path: path('M16,120 C80,20 220,20 284,120');
  offset-distance: 0%;
}
```

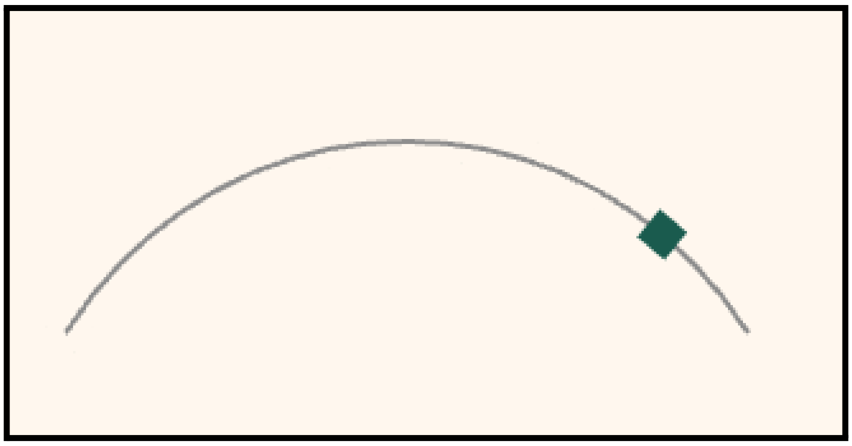

Everything is now in place.

Animate along the curve

Drive the motion by animating `offset-distance` from `0%` to `100%`. The `alternate` direction sends the square back without further work. If you later replace the square with an arrow and want it to face the direction of travel, add `offset-rotate: auto`.

```
@keyframes move {
  to {
    offset-distance: 100%;
  }
}

.box {
  animation: move 3s ease-in-out infinite alternate;
}
```



The square is now moving.

Optional Path Visibility

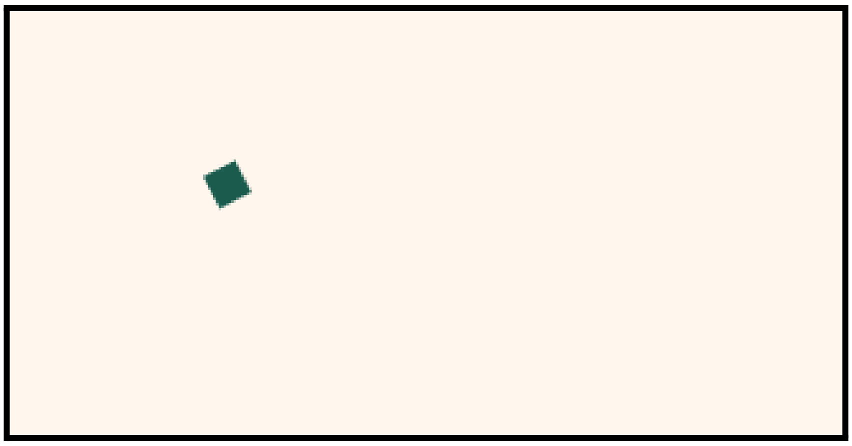
The visible SVG line we drew is just a guide. It helps during development to see the curve your element is following, but it isn't required for the animation to work.

The only critical piece is the `d` string that you paste into `offset-path`. Once the animation is in place, you can safely remove the stroke, or even the `<svg>` entirely, if you don't need a visible trace.

For example, you can hide the outline simply by removing the `stroke` properties:

```
path {
  fill: none;
  /* stroke removed -- path is now invisible */
}
```

The square (or whatever element you animate) will still follow the curve perfectly.


The square is moving without the visual line.

Use the outline when you want users to see a track, skip it when you want a "floating" effect.

Accessibility

Not everyone enjoys animation. Some people experience motion sensitivity or simply prefer a calmer experience. Modern browsers respect this by exposing a system-level preference called `prefers-reduced-motion`.

You can target that preference with a media query in CSS. In our case, we'll pause the motion path animation entirely if the user has asked for reduced motion:

```css
@media (prefers-reduced-motion: reduce) {
  .box {
    animation: none; /* stop the movement */
  }
}
```

In some cases, the moving element doesn't add any real value on its own. If it's purely decorative, it might be better to hide it entirely when motion is disabled:

```css
@media (prefers-reduced-motion: reduce) {
  .box {
    display: none; /* remove it if animation isn't meaningful */
  }
}
```

That way you respect the user's preferences and keep the page free of distractions.

Troubleshooting

If the square travels near the line rather than on it, check these three things in order:

- Make sure the container's CSS width and height match the dimensions defined in the SVG's `viewBox` (300 × 140 in this example).
- Verify that the CSS `path('...')` string and the SVG `d="..."` value are character-for-character identical.
- Ensure the square starts at the origin with `top: 0; left: 0;`.

When you need the curve to resize fluidly with the layout, the simplest approach is to scale the entire stage. If the curve must actually reshape, you'll need to update the path data, which is one of the few cases where a tiny bit of JavaScript to generate coordinates can be justified.

Here's a Codepen to see the demo.

ℹ️ Browser Compatibility

- `offset-path`, `offset-distance`, and `offset-rotate`: ~93,5% global support.
- In older engines that ignore these properties, the square remains static while the rest of the page renders normally. Ship a non-animated fallback if your audience includes those browsers.

✅ Key Takeaways

- Use `offset-path: path('...')` and animate `offset-distance` to move an element along any curve.
- Keep the container size equal to the SVG `viewBox` and reuse the same `d` string in CSS for perfect alignment.
- Start with a simple shape (a square or dot) so the center rides the path naturally. Add `offset-rotate` (and, if needed, `offset-anchor`) only for directional artwork.
- Honor `prefers-reduced-motion` to provide a no-animation experience.
- Treat responsiveness by scaling the whole component, regenerate path data only if the curve itself must reshape.

Motion Path gives you the ability to choreograph precise movement, bringing individuality and playfulness to your designs. It shows how CSS animation has matured beyond simple transforms.

Preview Edition - You Don’t Need JavaScript

Thank you for downloading this **preview edition** of the book.
This sample includes the introduction and chapter **`II.3`** only. It is designed to give you a sense of the style, the approach, and the kind of practical knowledge you can expect in the full version.

If you find this preview useful and want to continue learning, the complete book includes over 120 pages of advanced techniques, practical examples, and future-facing CSS features.

To access the **full edition**, please visit: https://theosoti.com/you-dont-need-js/