Practical positioning examples

- A tabbed info-box
  - General setup
  - Styling our tabs
  - Styling the panels
  - JavaScript
- A fixed position tabbed info-box
  - HTML additions
  - Changes to the existing CSS
  - Styling the main content
- A sliding hidden panel
  - Styling the button
  - Styling the panel
  - Setting the checked state
- Summary

This article shows how to build some real-world examples to illustrate what kinds of things you can do with positioning.

Prerequisites:	HTML basics (study Structuring content with HTML), and an idea of How CSS works (study CSS Styling basics.)
Objective:	To get an idea of the practicalities of positioning

A tabbed info-box

The first example we'll look at is a classic tabbed info box — a very common feature used when you want to pack a lot of information into a small area. This includes information-heavy apps like

strategy/war games, mobile versions of websites where the screen is narrow and space is limited, and compact information boxes where you might want to make lots of information available without having it fill the whole UI. Our simple example will look like this once we are finished:

Tab 1 is selected. 'Tab 2' and 'Tab 3' are the other two tabs. Only the contents of the selected tab ar

> **NOTE:** You can see the finished example running live at tabbed-info-box.html (source code). Check it out to get an idea of what you will be building in this section of the article.

You might be thinking "Why not just create the separate tabs as separate webpages, and just have the tabs clicking through to the separate pages to create the effect?" This code would be simpler, yes, but then each separate "page" view would actually be a newly-loaded webpage, which would make it harder to save information across views, and integrate this feature into a larger UI design.

To start with, we'd like you to make a local copy of the starting files — tabbed-info-box-start.html and tabs-manual.js. Save these somewhere sensible on your local computer, and open `tabbed-info-box-start.html` in your text editor. Let's look at the HTML contained within the body:

```
<section class="info-box">
  <div role="tablist" class="manual">
    <button
      id="tab-1"
      type="button"
      role="tab"
      aria-selected="true"
      aria-controls="tabpanel-1">
      <span>Tab 1</span>
    </button>

    <button
      id="tab-2"
      type="button"
      role="tab"
      aria-selected="false"
      aria-controls="tabpanel-2">
      <span>Tab 2</span>
    </button>
    <button
```

```
    id="tab-3"
    type="button"
    role="tab"
    aria-selected="false"
    aria-controls="tabpanel-3">
    <span>Tab 3</span>
  </button>
</div>

<div class="panels">
  <article id="tabpanel-1" role="tabpanel" aria-labelledby="tab-1">
    <h2>The first tab</h2>
    <p>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque
      turpis nibh, porttitor nec venenatis eu, pulvinar in augue. Vestibulum
      et orci scelerisque, vulputate tellus quis, lobortis dui. Vivamus varius
      libero at ipsum mattis efficitur ut nec nisl. Nullam eget tincidunt
      metus. Donec ultrices, urna maximus consequat aliquet, dui neque
      eleifend lorem, a auctor libero turpis at sem. Aliquam ut porttitor
      urna. Nulla facilisi.
    </p>
  </article>

  <article id="tabpanel-2" role="tabpanel" aria-labelledby="tab-2">
    <h2>The second tab</h2>
    <p>
      This tab hasn't got any Lorem Ipsum in it. But the content isn't very
      exciting all the same.
    </p>
  </article>

  <article id="tabpanel-3" role="tabpanel" aria-labelledby="tab-3">
    <h2>The third tab</h2>
    <p>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque
      turpis nibh, porttitor nec venenatis eu, pulvinar in augue. And now an
      ordered list: how exciting!
    </p>
    <ol>
      <li>dui neque eleifend lorem, a auctor libero turpis at sem.</li>
```

```
        <li>Aliquam ut porttitor urna.</li>
        <li>Nulla facilisi</li>
      </ol>
    </article>
  </div>
</section>
```

So here we've got a section element with a `class` of `info-box`, which contains two divs. The first div contains three buttons, which will become the actual tabs to click on for displaying our content panels. The second div contains three article elements, which will make up the content panels that correspond to each tab. Each panel contains some sample content.

The idea here is that we will style the tabs to look like a standard horizontal navigation menu and style the panels to sit on top of one another using absolute positioning. We'll also give you a bit of JavaScript to include on your page to display the corresponding panel when a tab is pressed, and style the tab itself. You won't need to understand the JavaScript code itself at this stage, but you should think about learning some basic JavaScript as soon as possible — the more complex your UI features become, the more likely it is that you'll need some JavaScript to implement your desired functionality.

General setup

To begin with, add the following between your opening and closing style tags:

```
html {
  font-family: sans-serif;
}

* {
  box-sizing: border-box;
}

body {
  margin: 0;
}
```

This is just some general setup to set a sans-serif font on our page, use the `border-box` box-sizing model, and get rid of the default body margin.

Next, add the following just below your previous CSS:

```
.info-box {
  width: 452px;
  height: 400px;
  margin: 1.25rem auto 0;
}
```

This sets a specific width and height on the content and centers it on the screen using the old `margin: 1.25rem auto 0`. Previously in the course, we advised against setting a fixed height on content containers if at all possible; it is OK in this circumstance because we have fixed content in our tabs.

Styling our tabs

Now we want to style tabs to look like tabs — basically, these are a horizontal navigation menu, but instead of loading different web pages when they are clicked on like we've seen previously in the course, they cause different panels to be displayed on the same page. First, add the following rule at the bottom of your CSS to make the `tablist` a flex container and have it span 100% width:

```
.info-box [role="tablist"] {
  min-width: 100%;
  display: flex;
}
```

> **NOTE:** We are using descendant selectors with `.info-box` at the start of the chain throughout this example — this is so that we can insert this feature into a page with other content already on it, without fear of interfering with the styles applied to other parts of the page.

Next, we'll style the buttons to look like tabs. Add the following CSS:

```
.info-box [role="tab"] {
  padding: 0 1rem;
  line-height: 3rem;
  background: white;
  color: #b60000;
  font-weight: bold;
  border: none;
  outline: none;
}
```

Next, we'll set the `:focus` and `:hover` states of the tabs to look different when they are focused/hovered, providing users with some visual feedback.

```
.info-box [role="tab"]:focus span,
.info-box [role="tab"]:hover span {
  outline: 1px solid blue;
  outline-offset: 6px;
  border-radius: 4px;
}
```

Then we'll set a rule that highlights one of the tabs when `aria-selected` property is set to `true` on it. We will set this using JavaScript when a tab is clicked on. Place the following CSS below your other styles:

```
.info-box [role="tab"][aria-selected="true"] {
  background-color: #b60000;
  color: white;
}
```

Styling the panels

The next job is to style our panels. Let's get going!

First, of all, add the following rule to style the `.panels` div container. Here we set a fixed height to make sure the panels fit snugly inside the info-box, position `relative` to set the div as the positioning context, so you can then place positioned child elements relative to it and not the initial viewport, and finally we clear the float set in the CSS above so that it doesn't interfere with the remainder of the layout.

```
.info-box .panels {
  height: 352px;
  clear: both;
  position: relative;
}
```

Finally, we will style the individual article elements that make up our panels. The first rule we'll add will absolutely position the panels, and make them all sit flush to the top and left of their div container — this part is key to this whole layout feature, as it makes the panels sit on top of one another. The rule also sets the panels to the same height as the container and adds padding, a text color, and a background-color.

```
.info-box [role="tabpanel"] {
  background-color: #b60000;
  color: white;
  position: absolute;
  padding: 0.8rem 1.2rem;
  height: 352px;
  top: 0;
  left: 0;
}
```

The second rule we'll add here makes it so that a panel with a class of `is-hidden` set on it will be hidden. Again, we'll add/remove this class using JavaScript at the appropriate time. When a tab is selected, the corresponding panel will have its `is-hidden` class removed, and all other panels will have `is-hidden` class set, thus only one panel will be visible at a time.

```
.info-box [role="tabpanel"].is-hidden {
  display: none;
}
```

JavaScript

The final part that makes this feature work is the JavaScript code. The `tabs-manual.js` file has been included using the `<script>` tag:

```
<script src="tabs-manual.js"></script>
```

This code does the following:

- On window load event it initializes `TabsManual` class for all the `tablist` elements.
- When a `TabsManual` object is created, in the constructor all the tab and panel references are collected in `tabs` and `tabpanels` variables, so we can easily do things to them later on.
- The constructor also registers `click` and `keydown` event handlers on all the tabs. The event handlers include logic about what should happen when a tab is selected using a click or keypress.
- In the `setSelectedTab(currentTab)` function, the following occurs:
  - A `for` loop is used to cycle through all the tabs and deselect them by setting `aria-selected` property to `false` and by setting `is-hidden` class on the corresponding panels.
  - On the selected tab (`currentTab`) the `aria-selected` is set to `true` and `is-hidden` class is removed from the corresponding panel.

- The code also has logic to support keyboard navigation using `Left arrow`, `Right arrow`, `Home`, and `End` keys.

A fixed position tabbed info-box

In our second example, we will take our first example — our info-box — and add it into the context of a full web page. But not only that — we'll give it a fixed position so that it stays in the same position in the browser window. When the main content scrolls, the info-box will stay in the same position on the screen. Our finished example will look like this:

Info-box is a container with 3 tabs with the first tab selected and only the contents of the first tab ar

> **NOTE:** You can see the finished example running live at fixed-info-box.html (source code). Check it out to get an idea of what you will be building in this section of the article.

As a starting point, you can use your completed example from the first section of the article, or make a local copy of tabbed-info-box.html from our GitHub repo.

HTML additions

First of all, we need some additional HTML to represent the webpage's main content. Add the following section just below your opening body tag, just before the existing section:

```
<section class="fake-content">
  <h1>Fake content</h1>
  <p>
    This is fake content. Your main web page contents would probably go here.
  </p>
  <p>
    This is fake content. Your main web page contents would probably go here.
  </p>
  <p>
    This is fake content. Your main web page contents would probably go here.
  </p>
  <p>
    This is fake content. Your main web page contents would probably go here.
```

```
  </p>
  <p>
    This is fake content. Your main web page contents would probably go here.
  </p>
  <p>
    This is fake content. Your main web page contents would probably go here.
  </p>
  <p>
    This is fake content. Your main web page contents would probably go here.
  </p>
  <p>
    This is fake content. Your main web page contents would probably go here.
  </p>
</section>
```

> **NOTE:** You can feel free to change the fake content for some real content if you like.

Changes to the existing CSS

Next, we need to change the existing CSS to position the info-box. Add position: fixed; to your `.info-box` rule to stick it to the top of the browser viewport. Once the info-box is fixed-positioned, `margin: 0 auto;` no longer centers it, so remove that declaration.

It should now look like this:

```
.info-box {
  width: 452px;
  height: 400px;
  position: fixed;
  top: 0;
}
```

Styling the main content

The only thing left for this example is to provide the main content with some styling. Add the following rule underneath the rest of your CSS:

```
.fake-content {
  background-color: #a60000;
  color: white;
  padding: 10px;
  height: 2000px;
  margin-left: 470px;
}

.fake-content p {
  margin-bottom: 200px;
}
```

To start with, we give the content the same background-color, color, and padding as the info-box panels. We then give it a large margin-left to move it over to the right, making space for the info-box to sit in, so it is not overlapping anything else.

This marks the end of the second example; we hope you'll find the third just as interesting.

A sliding hidden panel

The final example we'll present here is a panel that slides on and off the screen at the press of an icon — as mentioned earlier, this is popular for situations like mobile layouts, where the available screen space is small, so you don't want to use up most of it by showing a menu or info panel instead of the useful content.

Our finished example will look like this:

A blank screen on the left 60% of the screen with a 40% width panel displaying information on the r

> **NOTE:** You can see the finished example running live at hidden-info-panel.html (source code). Check it out to get an idea of what you will be building in this section of the article.

As a starting point, make a local copy of hidden-info-panel-start.html from our GitHub repo. This doesn't follow on from the previous example, so a fresh start file is required. Let's have a look at the HTML in the file:

```
<button
  type="button"
  id="menu-button"
  aria-haspopup="true"
  aria-controls="info-panel"
  aria-expanded="false">
      
</button>

<aside id="info-panel" aria-labelledby="menu-button">
  …
</aside>
```

To start with here we've got a button element with a special question mark character as the button text. The button will be pressed to show/hide the `aside` info panel. In the below sections we'll explain how this all works.

Styling the button

First let's deal with the button — add the following CSS in between your style tags:

```
#menu-button {
  position: absolute;
  top: 0.5rem;
  right: 0.5rem;
  z-index: 1;

  font-size: 3rem;
  cursor: pointer;
  border: none;
  background-color: transparent;
}
```

The first rule styles the `<button>`; here we've:

- Set a large font-size to make the icon nice and big.
- Remove the border and make the background transparent so that instead of the button only the `?` icon will show.

- Set position `absolute` on it, and used top and right to position it nicely in the top-right corner.
- Set a z-index of 1 on it — this is so that when the info panel is styled and shown, it doesn't cover up the icon; instead the icon will sit on top of it so it can be pressed again to hide the info pane.
- Used the cursor property to change the mouse cursor when it is hovering over the icon to a hand pointer (like the one you see when links are hovered over), as an extra visual clue to users that the icon does something interesting.

Styling the panel

Now it's time to style the actual sliding panel itself. Add the following rule to the bottom of your CSS:

```
#info-panel {
  background-color: #a60000;
  color: white;

  width: 340px;
  height: 100%;
  padding: 0 20px;

  position: fixed;
  top: 0;
  right: -370px;

  transition: 0.6s right ease-out;
}
```

A lot going on here — let's discuss it bit by bit:

- First, we set some simple background-color and color on the info box.
- Next, we set a fixed width on the panel, and make its height the entire height of the browser viewport.
- We also include some horizontal padding to space it out a bit.
- Next we set position: fixed; on the panel so it will always appear in the same place, even if the page has content to scroll. We glue it to the top of the viewport, and set it so that by default it is offscreen to the right.

- Finally, we set a transition on the element. Transition is an interesting feature that allows you to make changes between states happen smoothly, rather than just going "on" or "off" abruptly. In this case, we intend to make the panel slide smoothly onscreen when the checkbox is checked. (Or to put it another way, when the question mark icon is clicked.)

Setting the checked state

There is one final bit of CSS to add — put the following at the bottom of your CSS:

```
#info-panel.open {
  right: 0px;
}
```

The rule states that when the info-panel has `.open` class set on it, set the right property of the `<aside>` to `0px`, which causes the panel to appear on the screen again (smoothly due to the transition). Removing the `.open` class hides the panel again.

To add/remove the `.open` class from the info-panel with a click of the button we need to use some JavaScript. Add the following code in between script tags:

```
const button = document.querySelector("#menu-button");
const panel = document.querySelector("#info-panel");

button.addEventListener("click", () => {
  panel.classList.toggle("open");
  button.setAttribute("aria-expanded", panel.classList.contains("open"));
});
```

The code adds a click event handler to the button. The click handler toggles the `open` class on the info-box panel which slides the panel in or out of the view. The event handler also sets `aria-expanded` property on the button to improve accessibility.

So there you have it — the easiest way to create a toggling info panel effect.

Summary

So that rounds off our look at positioning — by now, you should have an idea of how the basic mechanics work, as well as understanding how to start applying these to build some interesting UI features. Don't worry if you didn't get this all immediately — positioning is a fairly advanced topic, and you can always work through the articles again to aid your understanding.

□□□ #3
□ □□□ □□□ 2026-10-31 11:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:32:35 UTC