**CSS Columns** is a straightforward way to style content so it is gathered together into 2 or more columns instead of just flowing horizontally to the edge of the parent element. This mimics a traditional printed layout and is especially intuitive for displaying text, since it avoids the long and narrow paragraph look when displaying text on wide screens.

Columns are defined primarily through two properties: column-width and column-count, whose effects are easy to understand. Width defines the minimum width of any column, while count defines the maximum number of columns to render. This helps the column content handle a variety of displays.

Advanced options for CSS Columns include ways to style the room between columns, and directives to force breaks in column flow, say making all h2s start in a new column. Unfortunately, these options differ slightly between Firefox and most other browsers, so doing things that depend on them might pose issues. But overall, if you’re fine letting the browser adjust column count and width on its own this is an easy way to make simple content responsive, but if you want stricter control over layout it may be worth considering a different method.

**Flexbox** is a simple way to create page layouts that adjust to a variety of screen sizes and layouts and can handle diverse child elements. A web developer can individually define how to handle sizing, spacing, alignment, and wrapping of elements, and they can control positioning on both the horizontal and vertical axes to easily handle various screen orientations.

Defining a container as “display: flex” will give handle a wide variety of screens and content types out-of-the-box, but a designer can customize the properties to control how items are spaced (on either axis) and whether or not the flexbox will allow them to wrap if their collective width exceeds that of the flexbox.

Helpfully, children of a flexbox are also able to flex by default, meaning the browser will not only adjust their position in the box, but can change their height and width to better use the space, shrinking or expanding them to ensure that the content best matches the flexbox that contains it. If the content contains a diversity of elements, the designer can even apply special properties that set relative scale or starting size of exceptional elements. Flexbox is an excellent way to style a wide variety of content so that the browser can sanely arrange it without requiring the designer to get bogged down in the specifics; this of course means that content which needs very precise arrangement is probably not a good fit for flexbox.

**CSS Grid** is a powerful and flexible way to define layout that allows the designer to mix statically positioned and sized content with dynamic spaces to allow for careful control over the display without sacrificing responsive principles. A grid container is defined with a set of rows and/or columns, which can be arbitrarily sized and need not be uniform; the grid cells are rectangular but not necessarily square. Sizes for these spaces can be absolute (px or em) or relative (%), and they can also be proportional to the other flexible dimensions in the grid template. By mixing and matching, a developer can define a fixed-with component, say a navbar, and leave the rest of the space responsive.

Once the grid template is defined, elements can be placed within it, either within a single cell or spanning multiple cells along either axis. Like flexbox, this makes it easy to accommodate elements of many sizes within a single layout. While powerful, grid requires more direct definition and a clear vision of the final layout than other methods because it is based on row and column template definitions that are made up front. But if you know what elements will be present and you want to have precise control over their position on the screen, grid gives you the tools to provide the specifics where necessary while leaving the rest to the browser.

**Combining Grid and Flexbox** is a particularly useful option, because it allows you to get “the best of both worlds” in many cases. The strength of Grid is the ability to carefully define absolutely positioned elements and to mix that with fluid regions, while flexbox focuses on letting the browser figure out the best way to fit an arbitrary collection of items into a defined space. Thus, you can use Grid to frame out the page’s overall structure, allocating space for things like navbars and other major elements. Then, within the regions you’ve defined via the Grid you can rely on flexbox to simplify handling elements within the region.

For instance, if you have a navbar along the top of the site’s main content area, you could defined a grid column for each menu item, but if nothing else on the page needs those individual rows you’ve just made your grid much more complicated than needed. Instead, you could define your grid so there’s a single wide cell for the navbar, then have that cell be a flexbox and let the browser handle item arrangement. This would also simplify making additions or subtractions to the bar: instead of having to redefine your grid to add or remove columns, you’d just change the elements in the navbar flexbox and let the browser work its magic.

**The “Best” Layout:**

While there’s obviously not a single “right” way to handle layout for all pages, CSS Grid offers a compelling model that is powerful and flexible, and web developers would do well to seriously consider it for modern responsive projects. While it requires a bit more effort to set up the grid template than methods like flexbox, the ability to seamlessly weave hand-tuned elements alongside proportional content areas is useful in a wide range of scenarios. Additionally, using Grid does not preclude leveraging the strengths of other strategies, particularly Flexbox, within specific regions of the page where appropriate.