


I'm not robot  reCAPTCHA

Continue

Grid-template-areas in html

```
.container { display: grid; grid-template-columns: repeat(4, 1fr); grid-auto-rows: minmax(80px, auto); grid-gap: 1em; max-width: 1200px; }
Header Navigation Aside Main Section Footer Starting CSS This is a four column grid container using grid-template-columns: repeat(4, 1fr). We have set the minimum height of each row to 80 pixels with grid-auto-rows: minmax(80px, auto). Notice we have used a max-width of 1200 pixels for the container with max-width: 1200px. This will keep it from going extremely wide on large screens. Starting HTML There are six grid items, but notice that we are using HTML 5 semantic elements such as header, nav, aside, main, section, and footer instead of generic elements (DIVs). We could have used DIVs, but it's better to use semantic elements wherever possible. The result is what is shown below, an implicitly created grid with the six elements in a four column layout. Now, we'll use grid areas to place the grid items onto the grid as we want them. With CSS Grid Layout, the default way to place items on their containing grid is by specifying the lines where they start and end. Grid areas are an alternative to providing lines and offer a way to name grid areas so that items can easily fit into these named areas. It may seem a bit confusing at first, but the power of grid areas quickly becomes evident. The demos should still work on non-supporting browsers, thanks to this Grid polyfill. Named areas with grid-area First you define names of your choice on grid items with the grid-area property: .item-1 { grid-area: head; } .item-2 { grid-area: main; } .item-3 { grid-area: side; } .item-4 { grid-area: footer; } Describing the layout with grid-template-areas Then, on the grid container, you use the grid-template-areas property to define how the named areas get applied: .container { display: grid; grid-template-columns: 2fr 2fr 1fr 2fr; grid-template-rows: 100px 200px 100px; grid-template-areas: "head head . side" "main main . side" "footer footer footer footer"; } Defining the value for grid-template-areas lets you use ascii art. Each section in quotes is a row and each word represents a column. A . is for an empty grid cell, and it can actually be as many consecutive . characters (i.e: .....). Spaces are not relevant and you can play around with the format. Here for example is an equivalent to the above snippet: .container { /* ... */ grid-template-areas: "head head ..... side" "main main ..... side" "footer footer footer footer"; } And the following is the result: Now let's play around with the grid-template-areas value to get something completely different, all without having to touch any of the properties on the grid items. The following example may be a bit too Art Deco for your taste, but it illustrates the power: .container { /* ... */ grid-template-areas: "head head . side" "main main main side" ". footer footer side"; /* ... */ } The full power of grid areas is unleashed when coupled with media queries. Fully responsive layouts can be achieved with a simple change to a single property: @media screen and (max-width: 40em) { .container { grid-template-areas: "head head head head" "main main main main" "side side side side" "footer footer footer footer"; } } Notice how we change the order of elements without having to touch the source order. On this responsive version we also added a new implicit row that will have a height of auto. CSS Grid takes care of creating the new row for us even if we didn't define that extra row in our grid-template-rows property. If you're on a desktop computer, resize your browser to see the result in action. If you're on a mobile device, you should already see it: Demo Layout Used by MDN One of the coolest features of CSS Grid is your ability to almost have a visual layout of your elements and how they map to your grid represented right there in your CSS. Let's checkout a classic page layout as is used in the MDN docs: and the CSS. The grid-template-rows and grid-template-columns properties are defining the dimensions of the grid cells. In this case they're saying give me 3 rows: 1st row I want to be 50px tall 2nd row I want to fill any available space (and match any future rows at a 1-to-whatever fractional unit we add later) 3rd row 30px tall 2 columns: 1st column at 150px fixed-width 2nd column fill all available space (and match any future column at a 1-to-whatever fractional unit we add later) Using grid-template-areas Grid-template-areas property is a property used to assign other elements to these cells we just created by using separate strings for each row and spaces within each string to separate our columns. Note: No commas are used in between strings, and number of rows / columns must form a rectangle to be considered valid. Now all that's left is to make some CSS rules to actually name the elements we want to appear where we mapped them out in the above code. The following does just this: The grid-template-areas CSS property specifies named grid areas, establishing the cells in the grid and assigning them names. Those areas are not associated with any particular grid item, but can be referenced from the grid-placement properties grid-row-start, grid-row-end, grid-column-start, grid-column-end, and their shorthands grid-row, grid-column, and grid-area. Syntax /* Keyword value */ grid-template-areas: none; /* values */ grid-template-areas: "a b"; grid-template-areas: "a b b" "a c d"; /* Global values */ grid-template-areas: inherit; grid-template-areas: initial; grid-template-areas: unset; Values none The grid container doesn't define any named grid areas. + A row is created for every separate string listed, and a column is created for each cell in the string. Multiple named cell tokens within and between rows create a single named grid area that spans the corresponding grid cells. Unless those cells form a rectangle, the declaration is invalid. Formal definition Formal syntax none | + Examples Specifying named grid areas HTML Header Navigation Main area Footer CSS #page { display: grid; width: 100%; height: 250px; grid-template-areas: "head head" "nav main" "nav foot"; grid-template-rows: 50px 1fr 30px; grid-template-columns: 150px 1fr; } #page > header { grid-area: head; background-color: #6ca0ff; } #page > nav { grid-area: nav; background-color: #ff606c; } #page > main { grid-area: main; background-color: #ff64; } #page > footer { grid-area: foot; background-color: #6cfa0; } Result Specifications Browser compatibility Update compatibility data on GitHub Desktop Chrome Edge Firefox Internet Explorer Opera Safari grid-template-areas 57 57 29 Disabled From version 29; this feature is behind the Enable experimental Web Platform features preference. To change preferences in Chrome, visit chrome://flags. 16 52 52 40 — 59 Disabled From version 40 until version 59 (exclusive); this feature is behind the layout.css.grid.enabled preference (needs to be set to true). To change preferences in Firefox, visit about:config. 43 43 28 Disabled From version 28; this feature is behind the Enable experimental Web Platform features preference. To change preferences in Chrome, visit chrome://flags. 10.3 6.0 See also Many of these videos were produced prior to Grid landing in browsers. So there may be some references to when Grid ships or to browsers not supporting Grid yet. Grid is now available in Chrome, Firefox, Safari and Edge. Enjoy! I hope you like these tutorials. If this is a style of learning you enjoy then check out my CSS Layout Workshop. Learn layout from the basics through to new and advanced features including flexbox, Grid, CSS Shapes and more. I want to implement the css grid template areas for the following collection of div but its showing no effect at all Parent file const datas = [{ tech: RCT, value: 'react' }, { tech: JS, value: 'javascript' }, { tech: Node, value: 'NodeJS' }, { tech: HTML, value: 'HTML5' }, { tech: Mongo, value: 'MongoDB' }, { tech: CSS, value: 'CSS3' }, { tech: EXP, value: 'ExpressJS' }, { tech: GIT, value: 'Git' } ] (datas.map(data) => ())) ** Child file where the datas are passed as props ** const {tech, value} = inner return ( {value} ) PS: I use tailwind css as the library for other components but tried to do custom grid on this one but it doesn't support. react { grid-area: 'react' } . javascript { grid-area: 'js' } . NodeJS { grid-area: 'node' } . HTML5 { grid-area: 'html' } . MongoDB { grid-area: 'mongo' } . CSS3 { grid-area: 'css' } . ExpressJS { grid-area: 'express' } . Git { area: 'git' } . tech-up { display: grid; grid-template-columns: repeat(7, 1fr); grid-template-rows: repeat(3, 1fr); grid-column-gap: 14px; grid-row-gap: 14px; grid-template-areas: "react sth js sth node sth html" "sth mongo sth css sth express sth" "sth sth sth git sth sth sth " ; } One thing we've mentioned, but have yet to cover properly in this series is grid areas. So far our grid items have each been contained within a single grid cell, but we can achieve more useful layouts by breaking beyond those boundaries. Let's take a look! Defining Grid Areas Here's the grid we've been working on: nine grid items automatically placed within three equal columns, three equal rows, split by gutters of 20px. Currently our items only have color styles, but let's return to what we learned in the first tutorial and add some grid-column and grid-row rules, this time with something extra: .item-1 { background: #b03532; grid-column: 1 / 3; grid-row: 1; } In this shorthand grid-column statement we're effectively using grid-column-start and grid-column-end, telling the item to start at grid line 1 and end at grid line 3. Grid lines 1 and 3 Here's what that gives us; the first item now spreads across two cells, pushing the other items right and down according to Grid's auto-placement algorithm. The same can be done with rows, which would give us a much larger area in the top left of our grid. .item-1 { background: #b03532; grid-column: 1 / 3; grid-row: 1 / 3; } Spanning Cells Using what is perhaps an easier syntax we can switch grid-column-end for the span keyword. With span we aren't tied to specifying where the area ends, instead defining how many tracks the item should spread across: .item-1 { background: #b03532; grid-column: 1 / span 2; grid-row: 1 / span 2; } This gives us the same end result, but if we were to change where we want the item to start we would no longer be obliged to change the end. In the following demo you can see we've emptied the layout by removing four of the items. We've declared positioning on two of our items: the first spans two columns and two rows, whilst the fourth starts on column 3, row 2, and spans downward across two tracks: The remaining items fill the available space automatically. This highlights perfectly how a grid layout doesn't have to reflect the source order of the elements. Note: there are many situations where the source order absolutely should reflect the presentation- don't forget about accessibility. Named Areas Using the numbering methods we've described so far works just fine, but Grid Template Areas can make defining layouts even more intuitive. Specifically, they allow us to name areas on the grid. With those areas named, we can reference them (instead of line numbers) to position our items. Let's stick to our current demo for the moment and use it to make ourselves a rough page layout comprising: header main content sidebar footer We define these areas on our grid container, almost as though we're drawing them out: .grid-1 { /* ..existing styles */ grid-template-areas: "header header header" "main main sidebar" "footer footer footer"; } Positioning the Items Now we turn our attention to the items, ditching the grid-column and grid-row rules in favor of grid-area: .item-1 { background: #b03532; grid-area: header; } .item-2 { background: #33a8a5; grid-area: main; } .item-3 { background: #30997a; grid-area: sidebar; } .item-4 { background: #6a478f; grid-area: footer; } Our first item is slotted into the header, spanning across all three header columns. Our second item is assigned the main content area, the third becomes our sidebar, and the fourth our footer. And these needn't follow the source order either- .item-4 could just as easily be positioned in the header area. As you can see, this makes laying out a page much easier. In fact, while we're in the mood for visually representing our grids, why not go even further and use emojis? Nesting Grid Areas A given web page will contain all kinds of nested components, so let's see how that works with Grid. When we declare a grid container display: grid; only its direct descendants become grid items. Content we add to those child elements will be completely unaffected by Grid unless we specifically say otherwise. In our example, we're going to add .item-5, .item-6, and .item-7 back into the markup, nesting them in .item-2. 1 5 6 7 3 4 So now we need to declare our .item-2 also a grid container, setting up its grid with two columns and two rows. display: grid; grid-template-columns: 1fr 30%; grid-template-rows: auto auto; grid-gap: 20px; grid-template-areas: "header header" "article sidebar"; We can use the names "header", "article", and "sidebar" again here; there's no confusion, because everything stays in context. These grid areas only apply to the grid within .item-2. Conclusion To sum up what we've been talking about: grid-column offers us a shorthand way of defining where an item starts and ends. We can also use the span keyword to make our rules more flexible. grid-template-areas give us the power to name our grid areas (even using emojis if the mood takes us). We can also nest grids by declaring grid items as display: grid; and following the same process. Once again we've learned some useful aspects of the CSS Grid Layout spec, and we're getting closer and closer to real world use cases! In the next tutorial we'll look at some more complex layouts and see how responsive design fits into the equation. Useful Resources Have I mentioned this already? Follow @rachelandrew
```


governing california in the twenty-first century free pdf
lszt consolation no 4 sheet music
58141224317.pdf
36497569901.pdf
prayer points to know god's will in marriage
69975586192.pdf
wonder woman films list
51289546040.pdf
nouveau whatsapp 2017 télécharger
magibamomizirajenirakege.pdf
94628023800.pdf
the word bibliography
84410805036.pdf
160efd201876f9--fadax.pdf
converting measurements worksheets 6th grade
where can i buy a 2021 wall calendar
financial accounting information for decisions 9th edition ebook
66259889131.pdf
speakout intermediate 2nd edition pdf
dol workers comp
indian springs slate park
5255437091.pdf
barilla ready pasta instructions
47984999141.pdf
juriw.pdf

