LEARN CSS GRID BY BUILDING A MAGAZINE

Introduction:

CSS Grid gives you control over the rows and columns of your web page design.

In this course, you'll build a magazine article. You'll learn how to use CSS Grid, including concepts like grid rows and grid columns.

Step 1:

Begin with your standard HTML boilerplate. Add a `DOCTYPE` declaration, an `html` element specifying this page is in English, a `head` element, and a `body` element.

Add a `<meta>` tag with the appropriate `charset` and a `<meta>` tag for mobile responsiveness within the `head` element.

Step 2:

Add a `title` element with the text `Magazine`, a `link` element for the `https://fonts.googleapis.com/css?family=Anton%7CBaskervville%7CRaleway&display=swap` font stylesheet, a `link` for the `https://use.fontawesome.com/releases/v5.8.2/css/all.css` FontAwesome stylesheet, and a `link` for your `./styles.css` stylesheet.

Your font stylesheet will load three separate fonts: `Anton`, `Baskervville`, and `Raleway`.

Step 3:

Within your `body`, create a `main` element. Then in that element, create a `section` with a `class` set to `heading`.

Step 4:

Within your `.heading` element, create a `header` element with the `class` set to `hero`.

In that element, create an `img` element with the `src` set to `https://cdn.freecodecamp.org/platform/universal/fcc_meta_1920X1080-indigo.png`, the `alt` set to `freecodecamp logo`, and the `class` set to `hero-img`.

The `loading` attribute on an `img` element can be set to `lazy` to tell the browser not to fetch the image resource until it is needed (as in, when the user scrolls the image into view). As an additional benefit, lazy loaded elements will not load until the non-lazy elements are loaded - this means users with slow internet connections can view the content of your page without having to wait for the images to load.

Give your new `img` element a `loading` attribute set to `lazy`.

Step 5:

After your `img` element, add an `h1` element with the `class` set to `hero-title` and the text set to `OUR NEW CURRICULUM`, followed by a `p` element with the `class` set to `hero-subtitle` and the text set to `Our efforts to restructure our curriculum with a more project-based focus`.

Step 6:

Your image currently takes up a lot of space. To better see what you are working on, add a `width` attribute to the `img` element, with a value of `400`.

You will remove this later on when you have worked on the CSS.

Step 7:

After your `header` element, create a `div` with the `class` set to `author`.

Within that `div`, create a `p` element with the `class` set to `author-name` and give it the text `By freeCodeCamp`. Wrap the `freeCodeCamp` portion in an `a` element with the `href` set to `https://freecodecamp.org`, and the `target` set to `_blank`.

Below that, add a second `p` element with the class `publish-date` and the text `March 7, 2019`.

Step 8:

The `Referer` HTTP header contains information about the address or URL of a page that a user might be visiting from. This information can be used in analytics to track how many users from your page visit freecodecamp.org, for example. Setting the `rel` attribute to `noreferrer` omits this information from the HTTP request. Give your `a` element a `rel` attribute set to `noreferrer`.

Step 9:

Below your `.author` element, create a new `div` element with the class `social-icons`.

Add five `a` elements within that new `div`, and give them the following `href` attributes.

- The first `a` element should have an `href` set to `https://www.facebook.com/freecodecamp`.
- The second `a` element should have an `href` set to `https://twitter.com/freecodecamp`.
- The third `a` element should have an `href` set to `https://instagram.com/freecodecamp`.

- The fourth `a` element should have an `href` set to `https://www.linkedin.com/school/free-code-camp`.
- The fifth `a` element should have an `href` set to `https://www.youtube.com/freecodecamp`.

Step 10:

Within each of your new `a` elements, add an `i` element and give them the following classes:

- Your first `i` element should have the class `fab fa-facebook-f`
- Your second `i` element should have the class `fab fa-twitter`
- Your third `i` element should have the class `fab fa-instagram`
- Your fourth `i` element should have the class `fab fa-linkedin-in`
- Your fifth `i` element should have the class `fab fa-youtube`

Step 11:

Below your `.heading` element, create a new `section` element with the `class` set to `text`. Within that, create a `p` element with the `class` set to `first-paragraph` and the following text:

Example Code:

Soon the freeCodeCamp curriculum will be 100% project-driven learning. Instead of a series of coding challenges, you'll learn through building projects - step by step. Before we get into the details, let me emphasize: we are not changing the certifications. All 6 certifications will still have the same 5 required projects. We are only changing the optional coding challenges.

Step 12:

Create another `p` element below your `.first-paragraph` element, and give it the following text:

Example Code

After years - years - of pondering these two problems and how to solve them, I slipped, hit my head on the sink, and when I came to I had a revelation! A vision! A picture in my head! A picture of this! This is what makes time travel possible: the flux capacitor!

Step 13:

Add a third `p` element at the end of your `.text` element, and give it the following text:

Example Code:

It wasn't as dramatic as Doc's revelation in Back to the Future. It just occurred to me while I was going for a run. The revelation: the entire curriculum should be a series of projects. Instead of individual coding challenges, we'll just have projects, each with their own seamless series of tests. Each test gives you just enough information to figure out how to get it to pass. (And you can view hints if that isn't enough.)

Step 14:

After the three `p` elements within your `.text` element, create a `blockquote` element. Within that, add an `hr` element, a `p` element with the `class` set to `quote`, and a second `hr` element.

Give the `.quote` element the text `The entire curriculum should be a series of projects`.

Step 15:

Below your `blockquote` element, add another `p` element with the following text:

Example Code:

No more walls of explanatory text. No more walls of tests. Just one test at a time, as you build up a working project. Over the course of passing thousands of tests, you build up projects and your own understanding of coding fundamentals. There is no transition between lessons and projects, because the lessons themselves are baked into projects. And there's plenty of repetition to help you retain everything because - hey - building projects in real life has plenty of repetition.

Step 16:

Create a fifth `p` element at the end of your `.text` element, and give it the following text:

Example Code:

The main design challenge is taking what is currently paragraphs of explanation and instructions and packing them into a single test description text. Each project will involve dozens of tests like this. People will be coding the entire time, rather than switching back and forth from "reading mode" to "coding mode".

Step 17:

Create one final `p` element at the end of your `.text` element and give it the following text:

Example Code:

Instead of a series of coding challenges, people will be in their code editor passing one test after another, quickly building up a project. People will get into a real flow state, similar to what they experience when they build the required projects at the end of each certification. They'll get that sense of forward progress right from the beginning. And freeCodeCamp will be a much smoother experience.

Step 18:

Below your `.text` element, create a new `section` element and give it a `class` of `text text-with-images`. Within that, create an `article` element with a `class` set to `brief-history`, and an `aside` element with the `class` set to `image-wrapper`.

Step 19:

Within your `article` element, create an `h3` element with the `class` set to `list-title` and the text of `A Brief History`. Below that, create a `p` element with the text `Of the Curriculum`. Then create a `ul` element with the class `lists`.

Step 20:

Within your `ul` element, create six `li` elements. Add an `h4` element with a `class` set to `list-subtitle` and a `p` element to each of your `li` elements.

Then give the `h4` and `p` elements the following text content, in order, with each `h4` using what's on the left side of the colon, and each `p` using what's on the right:

- `V1 - 2014`: `We launched freeCodeCamp with a simple list of 15 resources, including Harvard's CS50 and Stanford's Database Class.`
- `V2 - 2015`: `We added interactive algorithm challenges.`
- `V3 - 2015`: `We added our own HTML+CSS challenges (before we'd been relying on General Assembly's Dash course for these).`
- `V4 - 2016`: `We expanded the curriculum to 3 certifications, including Front End, Back End, and Data Visualization. They each had 10 required projects, but only the Front End section had its own challenges. For the other certs, we were still using external resources like Node School.`

- V5 - 2017: We added the back end and data visualization challenges.
- V6 - 2018: We launched 6 new certifications to replace our old ones. This was the biggest curriculum improvement to date.

Step 21:

Within your aside element, create two img elements, a blockquote element, and a third img element. Give the blockquote element a class set to image-quote.

Step 22:

Within the .image-wrapper element, give your first img element a src of https://cdn.freecodecamp.org/testable-projects-fcc/images/random-quote-machine.png, an alt of image of the quote machine project, a class of image-1, a loading attribute set to lazy, a width attribute of 600, and a height attribute of 400.

Step 23:

Within your .image-wrapper element, give the second img element a src of https://cdn.freecodecamp.org/testable-projects-fcc/images/calc.png, an alt of image of a calculator project, a loading attribute set to lazy, a class set to image-2, a width attribute set to 400, and a height attribute set to 400.

Step 24:

Within your .image-wrapper element, give your third img element a src of https://cdn.freecodecamp.org/testable-projects-fcc/images/survey-form-background.jpeg, an alt of four people working on code, a loading attribute of lazy, a class set to image-3, a width attribute set to 600, and a height attribute set to 400.

Step 25:

Within your .image-quote element, nest an hr element, a p element and a second hr element. Give the p element a class set to quote and the text The millions of people who are learning to code through freeCodeCamp will have an even better resource to help them learn these fundamentals.

Step 26:

To start your CSS, normalize the CSS rules by targeting all elements with *, including the ::before and ::after pseudo-selectors.

Set the padding and margin properties both to 0 and set the box-sizing property to border-box.

Step 27:

Create an html selector and give it a font-size property set to 62.5%. This will set the default font size for your web page to 10px (the browser default is 16px).

This will make it easier for you to work with rem units later, as 2rem would be 20px.

Step 28:

Create a body selector. Set the font-family property to Baskervville, with a fallback of serif. Set the color property to linen and the background-color property to rgb(20, 30, 40).

Step 29:

Create an `h1` selector, and set the `font-family` property to `Anton` with the fallback of `sans-serif`.

Step 30:

Create an `h2, h3, h4, h5, h6` selector. Give it a `font-family` property set to `Raleway` with a fallback of `sans-serif`.

Step 31:

Create an `a` selector, and give it a `text-decoration` property set to `none` and a `color` property set to `linen`.

Step 32:

Now you are ready to start putting together the grid layout. CSS Grid offers a two-dimensional grid-based layout, allowing you to center items horizontally and vertically while still retaining control to do things like overlap elements.

Begin by creating a `main` selector and giving it a `display` property set to `grid`.

Step 33:

Now you can style the layout of your grid. CSS Grid is similar to Flexbox in that it has a special property for both the parent and child elements.

In this case, your parent element is the `main` element. Set the content to have a three-column layout by adding a `grid-template-columns` property with a value of `1fr 94rem 1fr`. This will create three columns where the middle column is `94rem` wide, and the first and last columns are both 1 fraction of the remaining space in the grid container.

Step 34:

Use the `minmax` function to make your columns responsive on any device. The `minmax` function takes two arguments, the first being the minimum value and the second being the maximum. These values could be a length, percentage, `fr`, or even a keyword like `max-content`.

Wrap each of your already defined values of the `grid-template-columns` property in a `minmax` function, using each value as the second argument. The first argument should be `2rem`, `min-content`, and `2rem` respectively.

Step 35:

To add space between rows in the grid layout, you can use the `row-gap` property. Give the `main` selector a `row-gap` property of `3rem`.

Step 36:

Your magazine will have three primary sections. You already set the overall layout in the `main` rule, but you can adjust the placement in the child rules.

One option is the `grid-column` property, which is shorthand for `grid-column-start` and `grid-column-end`. The `grid-column` property tells the grid item which grid line to start and end at.

Create a `.heading` rule and set the `grid-column` property to `2 / 3`. This will tell the `.heading` element to start at grid line 2 and end at grid line 3.

Step 37:

Create a `.text` selector and give it a `grid-column` property set to `2 / 3`.

Step 38:

For additional control over the layout of your content, you can have a CSS Grid within a CSS Grid.

Set the `display` property of your `.heading` selector to `grid`.

Step 39:

Now you can style the content of the `.heading` element with CSS Grid.

The CSS `repeat()` function is used to repeat a value, rather than writing it out manually, and is helpful for grid layouts. For example, setting the `grid-template-columns` property to `repeat(20, 200px)` would create 20 columns each `200px` wide.

Give your `.heading` element a `grid-template-columns` property set to `repeat(2, 1fr)` to create two columns of equal width.

Step 40:

Give your `.heading` selector a `row-gap` property set to `1.5rem`.

Step 41:

Remember that the `grid-column` property determines which columns an element starts and ends at. There may be times where you are unsure of how many columns your grid will have, but you want an element to stop at the last column. To do this, you can use `-1` for the end column.

Create a `.hero` selector and give it a `grid-column` property set to `1 / -1`. This will tell the element to span the full width of the grid.

Step 42:

Give the `.hero` selector a `position` property set to `relative`.

Step 43:

You should remove the temporary `width` attribute before writing the CSS for your `.hero-img`.

Step 44:

Create an `img` selector and give it a `width` property set to `100%`, and an `object-fit` property set to `cover`.

The `object-fit` property tells the browser how to position the element within its container. In this case, `cover` will set the image to fill the container, cropping as needed to avoid changing the aspect ratio.

Step 45:

Create a `.hero-title` selector and give it a `text-align` property set to `center`, a `color` property set to `orangered` and a `font-size` property set to `8rem`.

Step 46:

The subtitle also needs to be styled. Create a `.hero-subtitle` selector and give it a `font-size` property set to `2.4rem`, a `color` property set to `orangered`, and a `text-align` property set to `center`.

Step 47:

Create an `.author` selector and give it a `font-size` property set to `2rem` and a `font-family` property set to `Raleway` with a fallback of `sans-serif`.

Step 48:

Create a `.author-name a:hover` selector and give it a `background-color` property set to `#306203`.

This will create a hover effect only for the `a` element within the `.author-name`, showing the original freeCodeCamp green in the background.

Step 49:

Create a `.publish-date` selector and give it a `color` property of `rgba(255, 255, 255, 0.5)`.

Step 50:

Create a `.social-icons` selector. Give it a `display` property set to `grid`, and a `font-size` property set to `3rem`.

Step 51:

The default settings for CSS Grid will create additional rows as needed, unlike Flexbox. Give the `.social-icons` selector a `grid-template-columns` property set to `repeat(5, 1fr)` to arrange the icons in a single row.

Step 52:

If you wanted to add more social icons, but keep them on the same row, you would need to update `grid-template-columns` to create additional columns. As an alternative, you can use the `grid-auto-flow` property.

This property takes either `row` or `column` as the first value, with an optional second value of `dense`. `grid-auto-flow` uses an auto-placement algorithm to adjust the grid layout. Setting it to `column` will tell the algorithm to create new columns for content as needed. The `dense` value allows the algorithm to backtrack and fill holes in the grid with smaller items, which can result in items appearing out of order.

For your `.social-icons` selector, set the `grid-auto-flow` property to `column`.

Step 53:

Now the auto-placement algorithm will kick in when you add a new icon element. However, the algorithm defaults the new column width to be `auto`, which will not match your current columns.

You can override this with the `grid-auto-columns` property. Give the `.social-icons` selector a `grid-auto-columns` property set to `1fr`.

Step 54:

Much like Flexbox, with CSS Grid you can align the content of grid items with `align-items` and `justify-items`. `align-items` will align child elements along the column axis, and `justify-items` will align child elements along the row axis.

Give the `.social-icons` selector an `align-items` property set to `center`.

Step 55:

Give the `.text` selector a `font-size` property set to `1.8rem` and a `letter-spacing` property set to `0.6px`.

Step 56:

Your `.text` element is not a CSS Grid, but you can create columns within an element without using Grid by using the `column-width` property.

Give your `.text` selector a `column-width` property set to `25rem`.

Step 57:

Magazines often use justified text in their printed content to structure their layout and control the flow of their content. While this works in printed form, justified text on websites can be an accessibility concern, for example presenting challenges for folks with dyslexia.

To make your project look like a printed magazine, give the `.text` selector a `text-align` property set to `justify`.

Step 58:

The `::first-letter` pseudo-selector allows you to target the first letter in the text content of an element.

Create a `.first-paragraph::first-letter` selector and set the `font-size` property to `6rem`. Also give it a `color` property set to `orangered` to make it stand out.

Step 59:

The other text has been shifted out of place. Move it into position by giving the `.first-paragraph::first-letter` selector a `float` property set to `left` and a `margin-right` property set to `1rem`.

Step 60:

Create an `hr` selector, and give it a `margin` property set to `1.5rem 0`.

Step 61:

To give the `hr` a color, you need to adjust the `border` property. Give the `hr` selector a `border` property set to `1px solid rgba(120, 120, 120, 0.6)`.

Step 62:

Create a `.quote` selector. Give it a `color` property set to `#00beef`, a `font-size` property set to `2.4rem`, and a `text-align` property set to `center`.

Step 63:

To make the quote text stand out more, give the `.quote` selector a `font-family` property set to `Raleway` with a fallback of `sans-serif`.

Step 64:

A quote is not really a quote without proper quotation marks. You can add these with CSS pseudo selectors.

Create a `.quote::before` selector and set the `content` property to `"` with a space following it.

Also, create a `.quote::after` selector and set the `content` property to `"` with a space preceding it.

Step 65:

Now it's time to style your third `section`. Note that it has the `text` and `text-with-images` values for the `class` attribute, which means it is already inheriting the styles from your `.text` rule.

Create a `.text-with-images` selector and set the `display` property to `grid`.

Step 66:

You will need to have a column for text and a column for images. Give the `.text-with-images` selector a `grid-template-columns` property set to `1fr 2fr`. Also set the `column-gap` property to `3rem` to provide more spacing between the columns.

Step 67:

Give the `.text-with-images` selector a `margin-bottom` property set to `3rem`.

Step 68:

Create a `.lists` selector and set the `list-style-type` property to `none`. This will get rid of the bullet points on the list items.

Step 69:

Give the `.lists` selector a `margin-top` property set to `2rem`.

Step 70:

Create a `.lists li` rule to target the list items within your `.lists` element. Give it a `margin-bottom` property set to `1.5rem`.

Step 71:

Create a `.list-title, .list-subtitle` selector and set the `color` property to `#00beef`.

Step 72:

Time to style the last section of the magazine - the images.

The images are wrapped with an `aside` element using the `image-wrapper` class, so create an `.image-wrapper` selector. Set the `display` property to `grid`.

Step 73:

The images should be within a two column, three row layout.

Give the `.image-wrapper` selector a `grid-template-columns` property set to `2fr 1fr` and a `grid-template-rows` property set to `repeat(3, min-content)`. This will give our grid rows that adjust in height based on the content, but columns that remain a fixed width based on the container.

Step 74:

The `gap` property is a shorthand way to set the value of `column-gap` and `row-gap` at the same time. If given one value, it sets the `column-gap` and `row-gap` both to that value. If given two values, it sets the `row-gap` to the first value and the `column-gap` to the second.

Give the `.image-wrapper` selector a `gap` property set to `2rem`.

Step 75:

The `place-items` property can be used to set the `align-items` and `justify-items` values at the same time. The `place-items` property takes one or two values. If one value is provided, it is used for both the `align-items` and `justify-items` properties. If two values are provided, the first value is used for the `align-items` property and the second value is used for the `justify-items` property.

Give the `.image-wrapper` selector a `place-items` property set to `center`.

Step 76:

Create an `.image-1, .image-3` rule and give it a `grid-column` property set to `1 / -1`. This will allow the first and third images to span the full width of the grid.

Step 77:

Now that the magazine layout is finished, you need to make it responsive.

Start with a `@media` query for `only screen` with a `max-width` of `720px`. Inside, create an `.image-wrapper` selector and give it a `grid-template-columns` property of `1fr`.

This will collapse the three images into one column on smaller screens.

Step 78:

Create another `@media` query for `only screen` with a `max-width` of `600px`. Within, create a `.text-with-images` rule and give it a `grid-template-columns` property of `1fr`.

This will collapse your bottom text area into a single column on smaller screens.

Step 79:

Create a third `@media` query for `only screen` with a `max-width` of `550px`. Within, create a `.hero-title` selector with a `font-size` set to `6rem`, a `.hero-subtitle, .author, .quote, .list-title` selector with a `font-size` set to `1.8rem`, a `.social-icons` selector with a `font-size` set to `2rem`, and a `.text` selector with a `font-size` set to `1.6rem`.

Step 80:

Create one final `@media` query for `only screen` with a `max-width` of `420px`. Within, create a `.hero-title` selector with a `font-size` property set to `4.5rem`.

Congratulations! Your magazine is now complete.