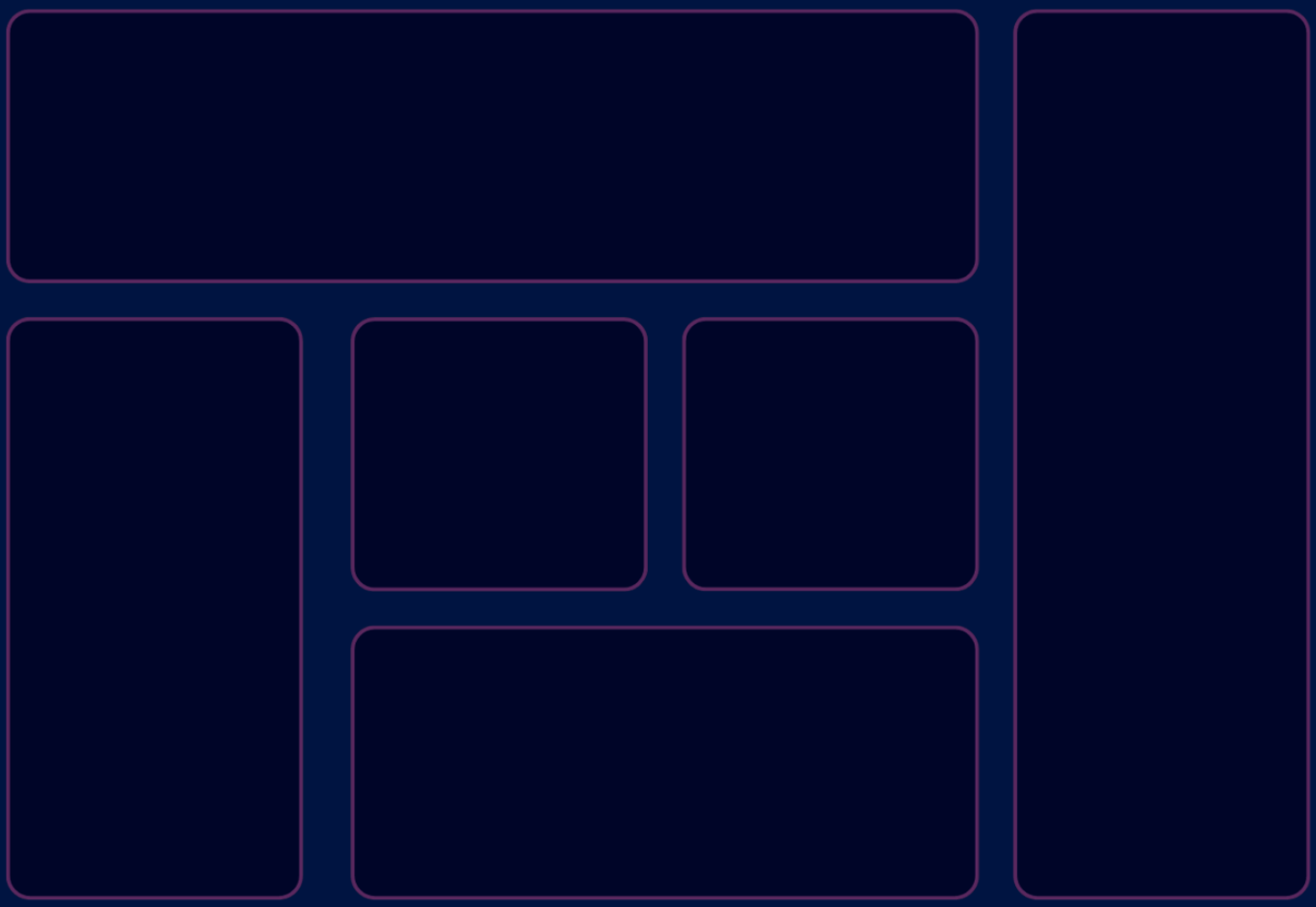
FLEX & GRID TRICKS

45 Tricks to Build Any Layout in Minutes



UTSAV MEENA

Table of Contents

Table of Contents **1**
1. Power Moves **3**
1. Stretching Buttons 3
2. Go Negative for Full Span 4
3. The Space Separator 5
4. A Visual Grid 6
5. Ultimate Flex Shorthand 7
6. Ultimate Grid Shorthand 8
7. Span with Ease 10
8. Centering with Flexbox 11
9. Centering with Grid 12
10. One Liner Center 12
11. Effortless Repeat 13
2. Mastering Control and Precision **14**
12. Gap Property: Simplify Your Spacing 14
13. Baseline Alignment: Polished Layouts 15
14. Z-Index: Layering Made Simple 16
15. Compact Grids with Inline-Grid 17
16. Reorder Items Seamlessly 17
17. Control Grid Item Flow 18
18. Dynamic Sizing with Minmax 19
19. Shrink-Proof Shapes 20
3. Common Layouts—The Modern Way **21**
20. The Holy Grail Layout 22
21. Elevate Your Card Game 23
22. Streamline Your Layouts 24
23. Corner Caption 26
24. Dynamic Thumbnail Magic 27
25. Navbar with a Twist 28
26. Footers that Stick 29
4. Make Layouts Responsive **30**
27. Responsive Form Style 30
28. Make the Masonry 31
29. Mosaic Gallery Flair 32
30. Intelligent Sidebars 34
31. Auto-Fit Layouts Simplified 34

32. Perfectly Fitting Grids 35
33. The Magic Line 36
34. Alternate Image-Caption Cards 37
5. Advanced Tricks and Layouts 38
35. Honeycomb Structure 39
36. Calendar in 3 Lines of CSS 41
37. Patterns on Repeat 42
38. Reorder Cart Items 42
39. Keyword-Based Track Size 43
40. Layered Profile Design 44
41. Expandable Description 47
42. Named Grid Lines 48
43. Formed Grid Areas 49
44. Dense Packing Algorithm 51
45. The Subgrid Trick 53

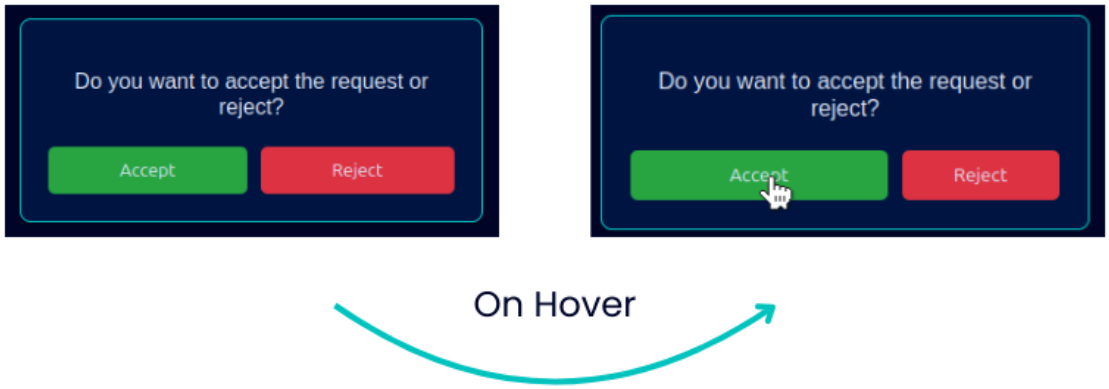
1. Power Moves

In this section, we'll dive into the basics of Flexbox and Grid. I'll walk you through the essential techniques for creating flexible, organized layouts. You'll see how to make the most of space to keep everything in place.

By the end, you'll have a strong grasp of these tools. This solid foundation will set you up for the more advanced tricks we'll cover later.

Ready to get started? Let's build!

1. Stretching Buttons



Usually, we change the color or scale of the buttons for a hover indication. But here...

We'll stretch the button. To achieve this:

1. Wrap both the buttons in a flex wrapper:

```
<div class="button-container">
    <button class="accept-button">Accept</button>
    <button class="reject-button">Reject</button>
</div>
```

2. Set both buttons' `flex-grow` to 1.
3. Finally, increase the `flex-grow` of the hovered button.

```
button{
    flex-grow: 1;
}
button:hover{
    flex-grow: 3;
}
```

Simple!

It's just one example, but you can use this concept in many other scenarios. But I'd leave it up to your creativity.

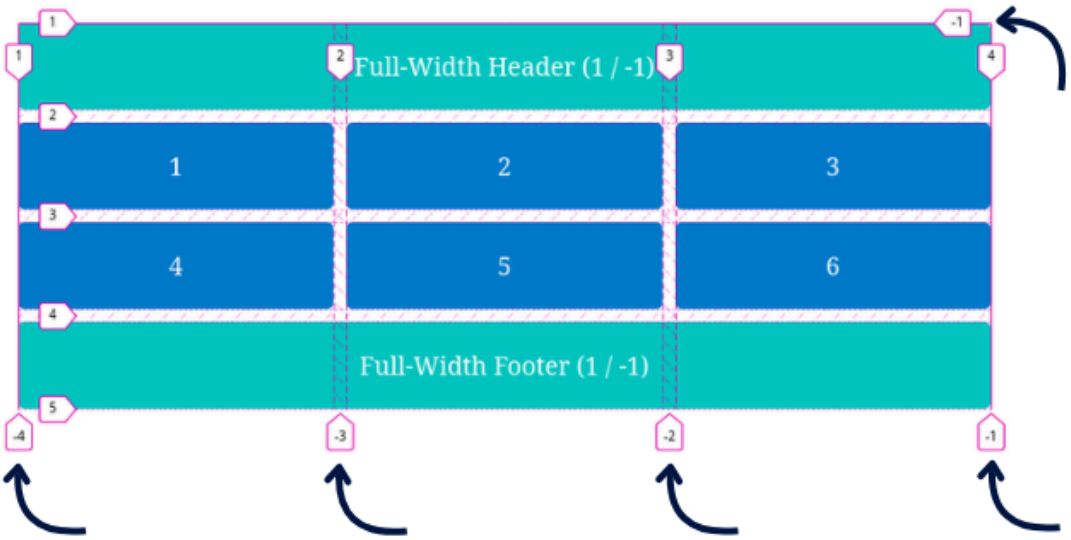
Play around with it on CodePen.

2. Go Negative for Full Span

This is one of my favorite tricks! So, how do you make a grid item span up to a certain grid line?

You might use something like `grid-column: 1 / 4`, right?

Well, here's another way: you can use negative line numbers. They start from the end, with -1 as the last line.



This trick is super handy when you want an element to span across all columns or rows.

Here's how:

```
.full-span {
      grid-column: 1 / -1;
}
```

This will span across all the columns in the grid.

Play around with it on CodePen.

3. The Space Separator

In this Navbar, there are 5 elements.

Four nav links and one search bar. You can easily achieve it with Flexbox, except for one problem.

All the extra space is between the nav links and the search bar—all in one place.

So, you'd need to wrap the nav links in an extra wrapper.

But you can achieve it simply by adding an extra empty element between the links and the search bar:

```
...
<a href="#" class="nav-link">Contact</a>
<div class="space-separator"></div>
<div class="search-bar"></div>
```

Then growing it to take up all the space:

```
.space-separator{
    flex-grow: 1;
}
```

That's it. You'll get this layout. To see the complete code:

Play around with it on CodePen.

4. A Visual Grid

Usually, when creating a grid, you have to picture it in your mind.

But with the `grid-template-areas` property, you can design the grid right in your code, like this:

```
.grid-container{
grid-template: 100px 1fr 100px / 20% 1fr 1fr;
grid-template-areas: "nav nav nav"
                     "aside main main"
                     "footer footer footer";
}
nav{grid-area: nav;}
aside{grid-area: aside;}
main{grid-area: main;}
footer{grid-area: footer;}
```

Play around with it on CodePen.

5. Ultimate Flex Shorthand

It's a shorthand, rather than a trick, but makes your code much cleaner.

With the `flex` shorthand, you can define three flex item properties at once:

1. `Flex-grow`
2. `Flex-shrink`
3. `Flex-basis`

But it has a catch.

It behaves differently based on the number (and type) of values.

See it yourself:

```
.item {
  /* One value, unitless */
  flex: 2; /* = flex-grow: 1 */

  /* One value, with unit */
  flex: 10em; /* = flex-basis: 10em */

  /* Two values, one unitless & one with unit */
  flex: 1 30px; /* = flex-grow: 1; flex-basis: 30px */

  /* Two values: both unitless */
  flex: 2 2; /* = flex-grow: 2; flex-shrink: 2 */

  /* Three values */
  flex: 2 2 10%; /* flex-grow: 2; flex-shrink: 2;
  flex-basis: 10% */

  /* Keyword Based Value */
  flex: initial; /* 0 1 auto */
}
```

Play around with it on CodePen.

6. Ultimate Grid Shorthand

Just like Flex, the grid also has a shorthand. But it can take much more values than flex.

It also behaves differently based on the values provided.

You can give one value:

```
.grid-container{
grid: 200px 1fr;
/* Equivalent to
grid-template-rows:  200px 1fr; */
}
```

You can give two sets of values, separated by a / sign. They'd act as `grid-template-rows` and `grid-template-columns`, respectively:

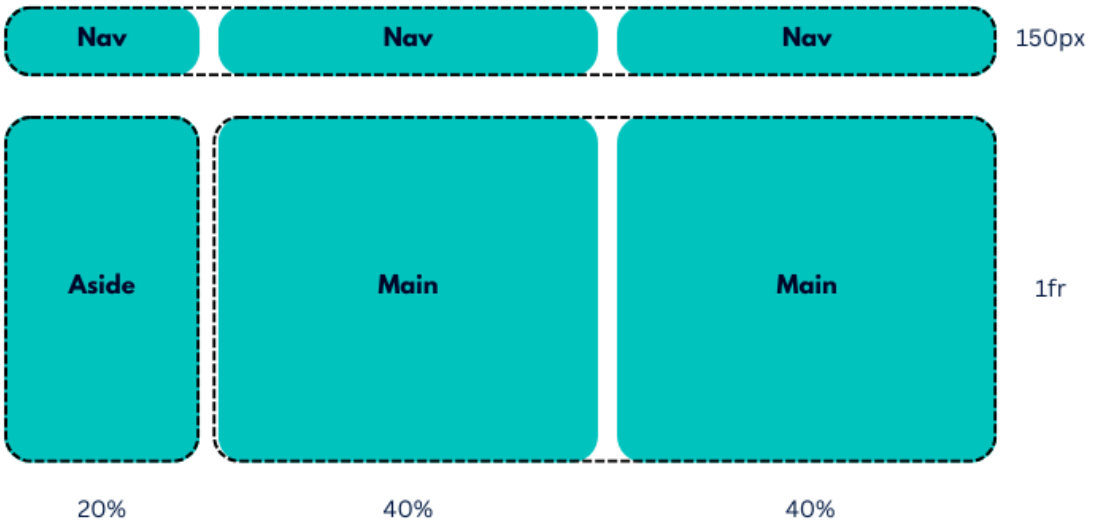
```
grid: 200px 1fr / 40% 60%;
```

You can also combine `grid-template-areas` with these properties.

It's like the grid-template-areas we talked about earlier, but now you can define the template in the same property.

This makes it easier to visualize (sort of)!

```
grid: "nav nav nav" 150px
      "aside main main" 1fr
      /20% 40% 40%;
```



You can also add implicit properties like `grid-auto-flow`, `grid-auto-columns`, and `grid-auto-rows`.

Just keep in mind—they don't work with `grid-template-areas` as used above.

Here's the correct way to use them:

```
grid: 150px 1fr auto-flow / 20% 40% 40%;
```

This will set the `grid-auto-flow` to row, as it's defined on the row side.

You can do many more things using this, so:

Play around with it on CodePen.

7. Span with Ease

As we discussed, we use grid line numbers to span a grid element.

But did you know you can also set how many columns to span from the starting position? Just use the `span` keyword, like this:

```
.span-two {
      grid-column: span 2;
}
```

This will make the item cover two columns from where it is.

Play around with it on CodePen.

8. Centering with Flexbox

The classic CSS question: How do I center a div?

It's simple with Flexbox! Just do it like this:

```
.container{
    display: flex;
    justify-content: center;
    align-items: center;
}
```

Play around with it on CodePen.

9. Centering with Grid

If you're using grid, then you can center items like this:

```
.grid-container{
    display: grid;
    justify-items: center;
    align-items: center;
}
```

Play around with it on CodePen.

10. One Liner Center

Earlier, we set `justify-items` and `align-items` individually.

If they have the same value, you can make it easier by using `place-items`.

Here's how:

```
.grid-container {
    display: grid;
    place-items: center;
}
```

Play around with it on CodePen.

11. Effortless Repeat

Let's say you want to create a layout with 5 equal columns.

How would you do that?

Like this?

```
.grid {
display: grid;
grid-template-columns: 1fr 1fr 1fr 1fr 1fr;
}
```

No! Instead, you could use the `repeat()` function, like this:

```
grid-template-columns: repeat(4, 1fr);
```

Much cleaner, right?

And if you ever need to change the number of columns, it's just a quick edit—no more counting or extra typing!

Play around with it on CodePen.

Feeling lost with CSS properties? If Flexbox or Grid still leaves you confused and frustrated…

My Pro Pack can help. With in-depth e-books and cheat sheets, it'll take you from "Why won't my CSS behave?" to confidently building layouts—no guesswork needed.

Dive in to master Flex & Grid, from basics to advanced!

2. Mastering Control and Precision

This section is all about getting things exactly where you want them.

We'll cover tricks to align items, space things out just right, and control the flow of elements.

With each tip, you'll see how to make quick, precise tweaks that instantly improve your layout. It's like fine-tuning your layout with just a few lines of CSS.

Let's jump in and take control!

12. Gap Property: Simplify Your Spacing

In CSS, you usually create gaps with margins or padding. But in Flexbox or Grid, you can use the `gap` property.

You can use `gap` in four ways:

1. `Gap`: This sets the space between both rows and columns. Like this:

```
.container {
    display: grid;
    gap: 20px;
}
```

1. `row-gap`: This specifically sets the space between rows:

```
row-gap: 15px;
```

2. `column-gap`: This controls the space between columns only:

```
column-gap: 30px;
```

3. `Gap` (with two values): As you might have guessed, it defines different row and column gaps, but with a single property. Like this:

```
gap: 15px 30px;
```

Play around with it on CodePen.

13. Baseline Alignment: Polished Layouts

First, what's a baseline in CSS?

It's the invisible line along the bottom of most text (excluding tails, like in "y" or "g"). Aligning to it keeps text bottoms even across the layout.

See this illustration from w3c.org:

To do this, just use *baseline* with `align-items`, like this:

```
.flex-container {
    display: flex;
    align-items: baseline;
}
```

You can do the same thing in Grid using the exact same method.

Baseline alignment is perfect for text-heavy designs, aligning icons with labels, or layouts with mixed media (like images and text). It keeps everything looking neat and well-aligned across the layout.

Play around with it on CodePen.

14. Z-Index: Layering Made Simple

Usually, `z-index` doesn't work with `position: static` (the default position). To use it, you first have to change the position.

But there's one exception that can be useful.

`z-index` will work on static elements if they're flex children.

Here's how:

```
.flex-container {
    display: flex;
}
.item1 {
    z-index: 1;
}
.item2 {
    z-index: 2;
}
```

In this example, `.item2` will appear on top of `.item1` due to its higher `z-index` value, even though both items have the default positioning.

Play around with it on CodePen.

15. Compact Grids with Inline-Grid

Did you know you can make Grid or Flexbox inline? Here's how:

Just add the inline prefix to `display: flex` or `display: grid`, and it becomes inline.

For example, if you're building a compact tag display for a blog or portfolio:

```
.tag-list {
    display: inline-grid;
    grid-template-columns: auto auto;
}
```

Now, the grid will be inline, so it won't interrupt the flow of surrounding text. This is great for small, embedded grids!

You can do the same thing with Flexbox, too.

Check out its working demo on CodePen.

16. Reorder Items Seamlessly

With Flexbox, you can visually reorder flex items without changing their markup order. Here's how:

Use the `order` property, which takes a numerical value, to arrange items from lowest to highest.

For example, in the code below, the second item will show before the first, regardless of its position in the markup:

```css
.flex-container {
    display: flex;
}
.item1 {
    order: 1; /* Default Value */
}
.item2 {
    order: -1; /* Moves item2 to the start */
}
```

No HTML rearrangement is needed—just adjust the order!

Play around with it on CodePen.

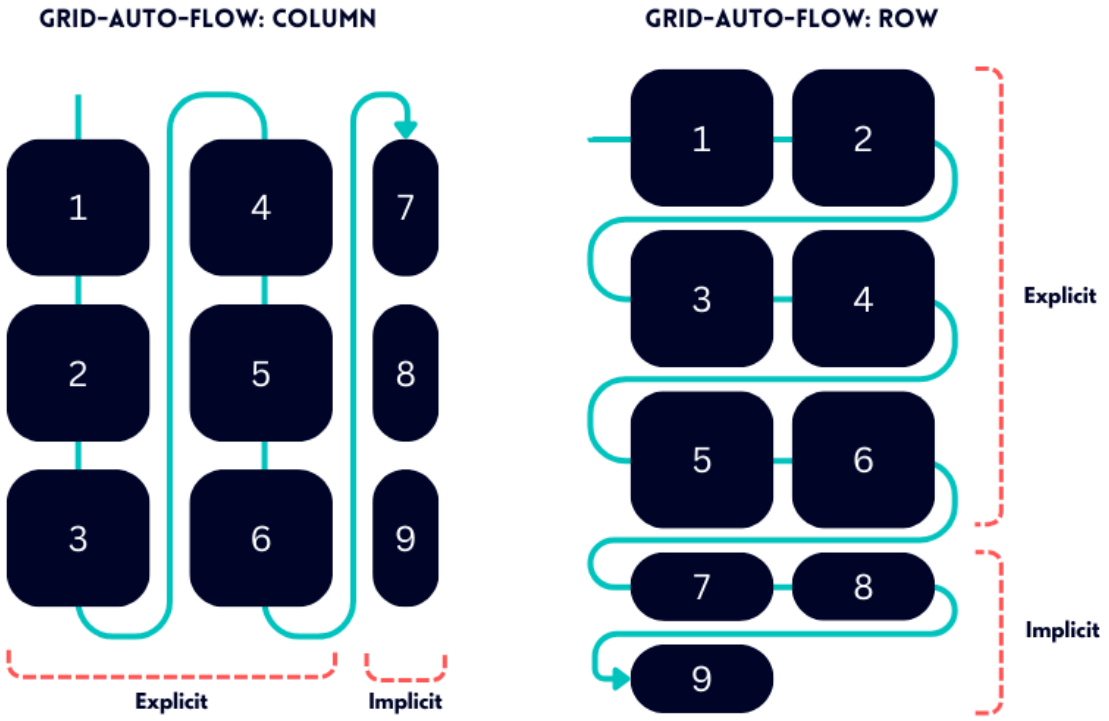
17. Control Grid Item Flow

By default, a grid fills rows first (left to right) and then moves to the next row. If it runs out of implicit rows, it creates new ones for extra elements.

Setting `grid-auto-flow` to column changes this behavior.

The grid will fill columns first (top to bottom) and create new columns for any additional elements when there's no more implicit space.

This allows you to control item placement easily!



Here's how it works in code:

```
.grid{
    display: grid;
    grid-template: repeat(3, 100px) / repeat(2, 100px);
    grid-auto-flow: column;
}
```

This will stack down the column before moving to the next, instead of rows.

Play around with it on CodePen.

18. Dynamic Sizing with Minmax

The `minmax()` function is one of the best features of Grid.

It allows you to define grid track sizes that vary within a range without exceeding it.

For example, if you set a track size like this:

```
.cont {
    display: grid;
    grid-template-columns: minmax(1rem, 1fr);
}
```

This creates a column that can grow up to 1fr, but will not go below 1rem.

Play around with it on CodePen.

19. Shrink-Proof Shapes

This tip is more of a helpful reminder than a trick, but it can save you a lot of time.

In a horizontal flex layout, you might have some elements with a fixed width, like the product image above.

However, they may get squeezed and not respect the defined width.

This happens because Flexbox is running out of space, and by default, it shrinks elements.

You can prevent this by setting the `flex-shrink` value to zero, like this:

```css
.product-img{
    flex-shrink: 0;
}
```

And you're good to go.

Play around with it on CodePen.

Feeling lost with CSS properties? If Flexbox or Grid still leaves you confused and frustrated…

My Pro Pack can help. With in-depth e-books and cheat sheets, it'll take you from "Why won't my CSS behave?" to confidently building layouts—no guesswork needed.

Dive in to master Flex & Grid, from basics to advanced!

3. Common Layouts—The Modern Way

In this section, we explore essential layouts for web development.

These tricks will simplify your design process. You'll learn to create responsive and engaging designs using Flexbox and Grid.

Each trick offers practical solutions for showcasing content effectively. You'll discover techniques for organizing images and ensuring key elements stay visible.

By the end, you'll have a set of layouts ready to use in your projects.

20. The Holy Grail Layout

The "Holy Grail" layout is a classic web design that features a header, footer, main content area, and sidebars. It's popular for providing a clear and flexible structure.

Here's how to create it using grid-template-areas and custom area names:

```
.holy-grail {
display: grid;
grid-template-areas:
    "header header header"
    "sidebar main aside"
    "footer footer footer";
grid-template-columns: 1fr 3fr 1fr;
grid-template-rows: auto 1fr auto;
}
.header {grid-area: header;}
.sidebar {grid-area: sidebar;}
.main {grid-area: main;}
.aside {grid-area: aside;}
.footer {grid-area: footer;}
```

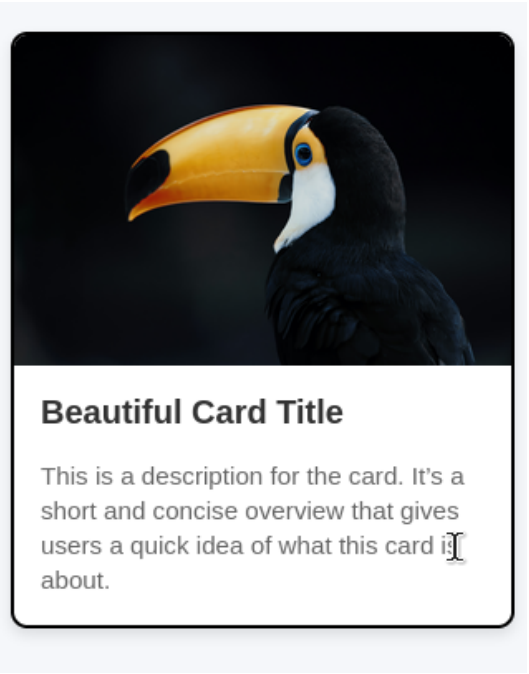
In this setup:

1. `grid-template-areas` defines the layout by naming each section.

2. Each class (.header, .main, etc.) is assigned to its corresponding grid area, keeping the structure organized.

Play around with it on CodePen.

21. Elevate Your Card Game



A card layout is one of the most common designs developers need to create.

Here's a quick way to set it up using Grid:

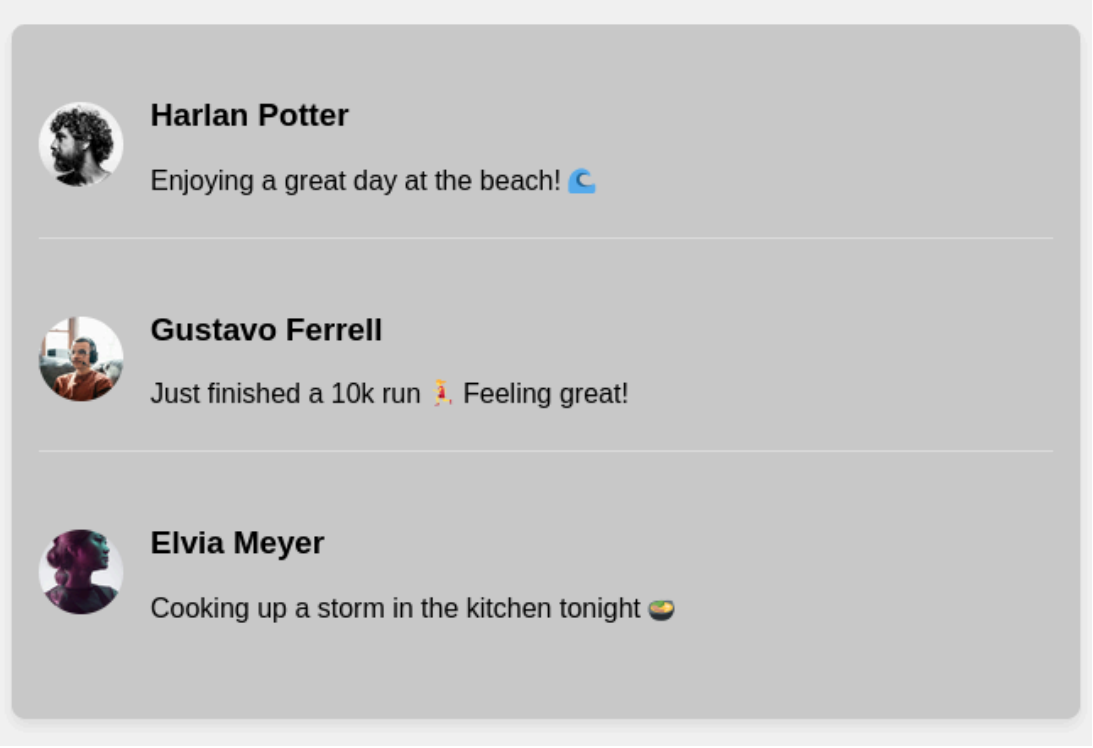
```
.card {
    display: grid;
    grid-template-rows: auto 1fr auto;
    gap: 0.5rem;
}

.card img {
    width: 100%;
}
```

Now, if you add this markup, you'll create a simple card featuring an image, a title, and a brief description:

```
<div class="card">
    <img src="img.png">
    <div class="card-title">Beautiful Card Title</div>
    <div class="card-description">
        Description
    </div>
    </div>
```

Play around with it on CodePen.

22. Streamline Your Layouts



Want to create a layout with a circular profile image on the left and text on the right, like a social media feed? Flexbox makes it easy!

Start by creating items with an image (using the `<img>` tag), a name, and a description.

Then, simply add this code:

```
.stream-item {
    display: flex;
    align-items: center;
    gap: 1rem;
}
```

As simple as that.

Play around with it on CodePen.

23. Corner Caption

Adding a caption in the corner of an image should be easy unless you can't use `position: absolute`.

In that case, you can use Flexbox or Grid instead.

Here's how:

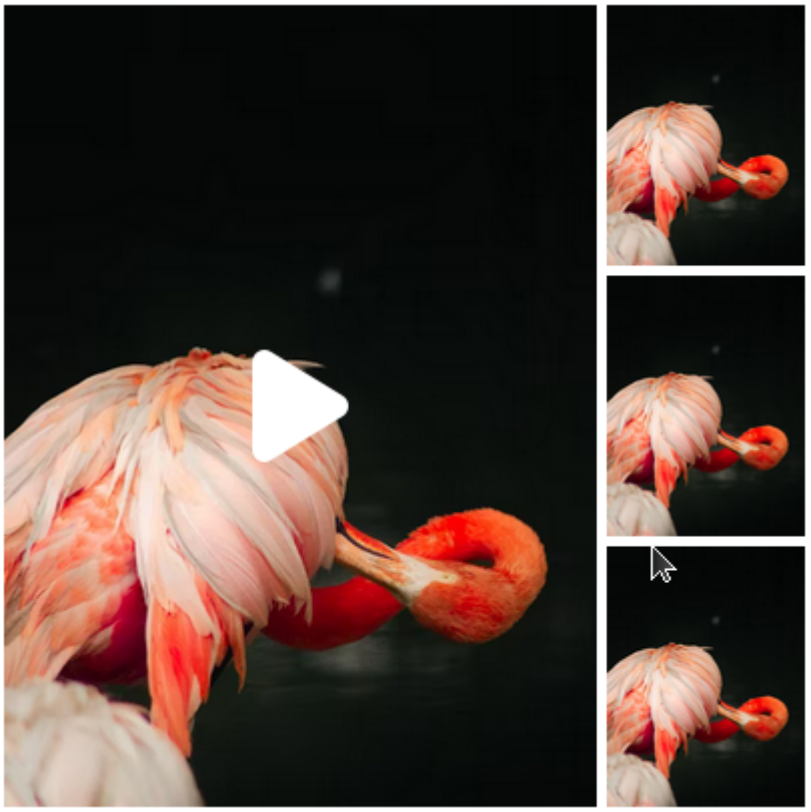
```
.photo-div {
    display: flex;
    align-items: flex-end;
    justify-content: flex-end;
}
```

That's it, now the caption will be placed in the bottom right corner.

Play around with it on CodePen.

24. Dynamic Thumbnail Magic

Let's say you need to create a layout featuring a video and three preview images. Like this:



It's fairly simple using grid. Just follow these steps:

1. Place the video and then all three preview images after it.
2. Put them in a grid container.
3. Now, create a grid structure like this:

```
.wrapper {
    grid-template: repeat(3, 1fr) / 3fr 1fr;
}
```

Then span the video in all three rows:

```
.video {
    grid-row-end: span 3;
}
```

That's it. You'll get the above layout.

Play around with it on CodePen.

25. Navbar with a Twist

We previously created a navbar with a space separator. But what if you want the search bar and links to be evenly spaced instead?

That's where `space-evenly` comes in.

Here's how to do it:

```
nav {
    display: flex;
    justify-content: space-evenly;
    align-items: center;
}
```

As simple as that. Now you'll get this new navbar version.

Play around with it on CodePen.

26. Footers that Stick

Keeping the footer at the bottom can be tricky, but Flexbox makes it super easy.

Just make the body a vertical flex container with a `min-height` of 100vh. Then, stretch the main content to fill the space.

Here's how:

```
body {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
}
main {
    flex: 1 0 0;
}
```

This approach keeps the footer at the bottom, no matter the content height—perfect for sticky footer layouts.

Play around with it on CodePen.

Feeling lost with CSS properties? If Flexbox or Grid still leaves you confused and frustrated…

My Pro Pack can help. With in-depth e-books and cheat sheets, it'll take you from "Why won't my CSS behave?" to confidently building layouts—no guesswork needed.

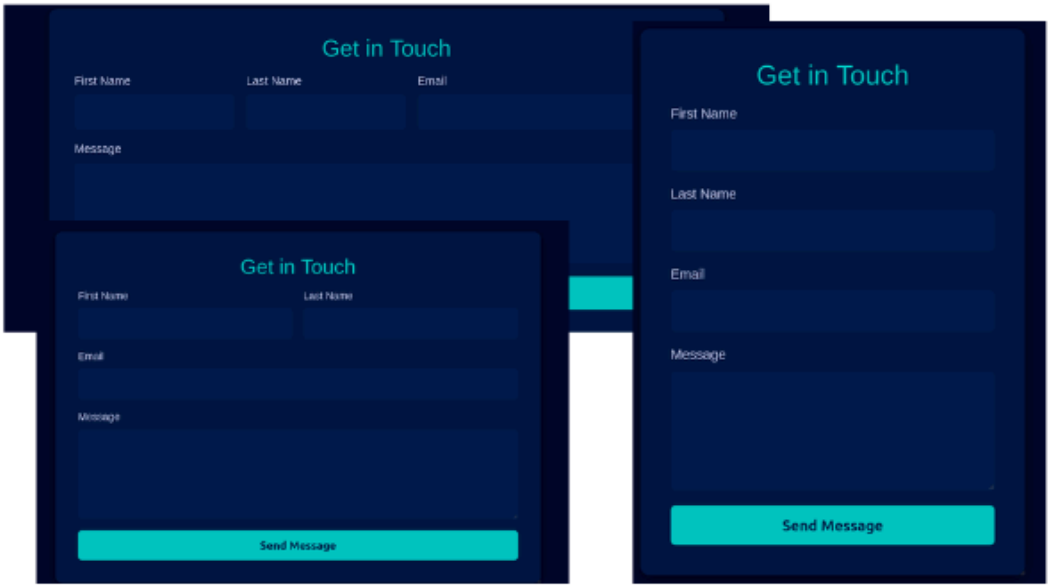
Dive in to master Flex & Grid, from basics to advanced!

4. Make Layouts Responsive

In this section, you'll dive into tricks for creating layouts that adapt smoothly across devices. These techniques make your designs flexible, adjusting beautifully on any screen size.

From forms to dynamic grids, each trick will give you new ways to handle changing layouts without extra effort. By the end, you'll have a toolkit of responsive designs ready to make your projects look sharp on any device.

27. Responsive Form Style



Forms are essential in web design, and here's how to make a responsive one without media queries.

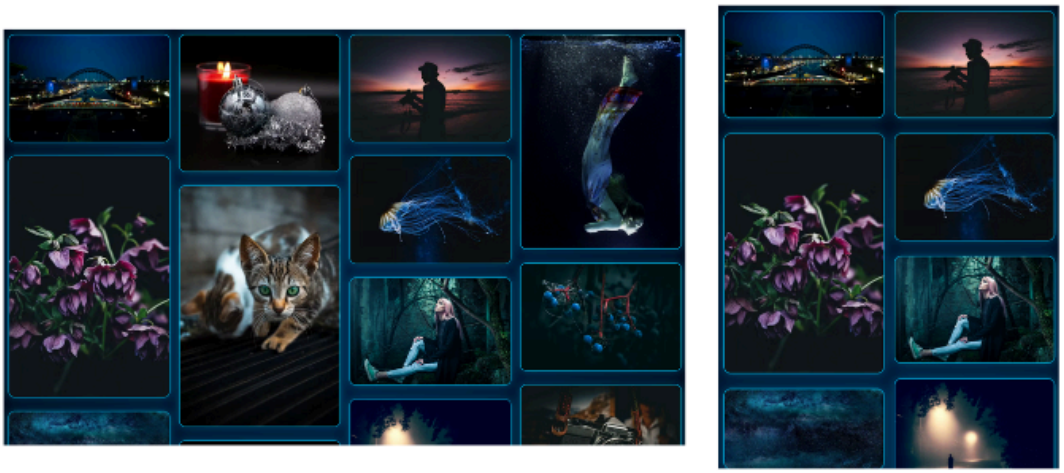
Start by adding the basic fields. Then, wrap smaller fields like first name, last name, and email in a flex container with `flex-wrap: wrap`.

Now, just set these properties as needed:

```
.fName, .lName {
    flex: 1;
    min-width: min(200px, 100%);
}

.email {
    flex: 3;
    min-width: min(350px, 100%);
}
```

Play around with it on CodePen.

28. Make the Masonry



I know this book focuses on Flexbox and Grid tricks, but I found a simple, non-Flex/Grid way to create a masonry layout that's too useful not to share.

All you need to do is place your images (`<img>` tags) in a container, then add these two lines:

```
.masonry-cont{
    column-width: 200px; /* Controls the column size */
}
img{
    width: 100%;
}
```

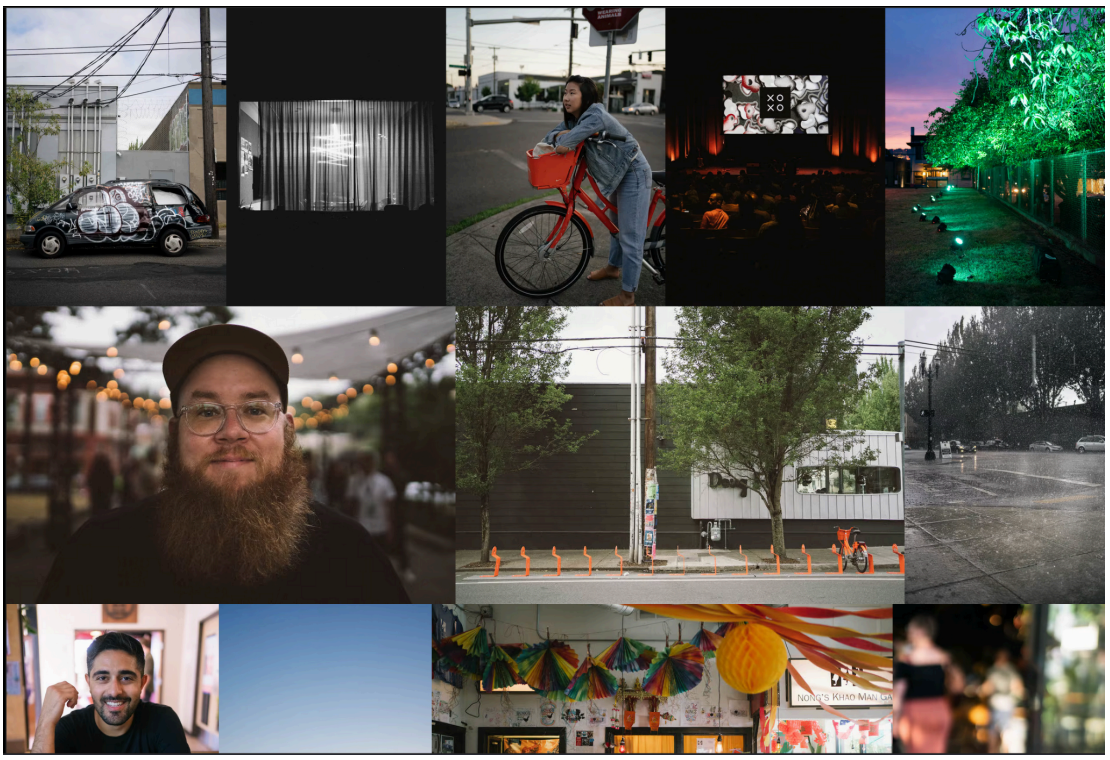
Yeah, it's that simple.

See it yourself on CodePen.

29. Mosaic Gallery Flair

This trick is one of the most interesting ones I've come across, inspired by an article from Tim Van.

The goal: create a responsive image gallery like this:



Notice how the images don't follow any specific aspect ratio or width, so a grid layout wouldn't work well here.

While it might seem tricky with Flexbox, it's actually simple:

1. Wrap all the images in a flex container with `flex-wrap: wrap`.
2. Set each image's `flex-grow` to 1.
3. Set the images' `min-width` and height to 100% and `object-fit` to cover.

Here's how it looks:

```
.img_container img {
    max-height: 100%;
    min-width: 100%;
    object-fit: cover;
}
```

And that's how you'll get a perfectly responsive mosaic layout.

Play around with it on CodePen.

30. Intelligent Sidebars

While responsiveness for narrow and wide devices is common, ultra-wide screens are often overlooked.

This trick prevents text from becoming too wide by adding flexible sidebars on either side of the main container.

These sidebars shrink on smaller screens and expand on wider ones, keeping your main content within a comfortable range. Here's how:

```
body {
    display: grid;
    grid-template-columns: minmax(1rem, 1fr) minmax(auto,
                           100ch) minmax(1rem, 1fr);
}
```

Play around with it on CodePen.

31. Auto-Fit Layouts Simplified

One of the coolest features of CSS Grid is the `auto-fit` keyword.

It allows you to dynamically create columns or rows based on the available space.

For example, check out this grid:

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px,
                           1fr));
}
```

It will create as many 200px columns as can fit. If there's extra space that can't fit another full column (less than 200px), it will grow the existing columns equally until there's no more room.

See it in action:

Play around with it on CodePen.

32. Perfectly Fitting Grids

With one line of code, you can control how your grid items resize based on their content.

If you want each card to fit closely around its content and limit its maximum size, do this:

Wrap your cards in a grid container and use this code:

```
.cards-wrapper {
    display: grid;
    grid-template-columns: repeat(auto-fill,
              minmax(200px, fit-content(300px)));
}
```

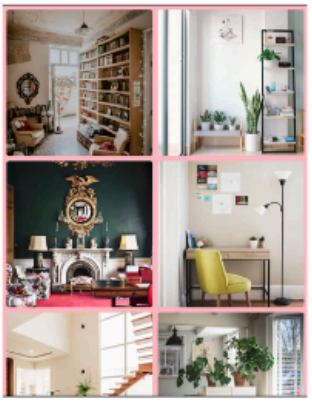
This setup allows each column to grow to fit its content, but not beyond 300px. It also adjusts nicely on smaller screens.

Play around with it on CodePen.

33. The Magic Line

Here's another one-liner trick. With just this line, you can create a responsive layout that fits any screen size.

For example, imagine a squared image gallery like this:



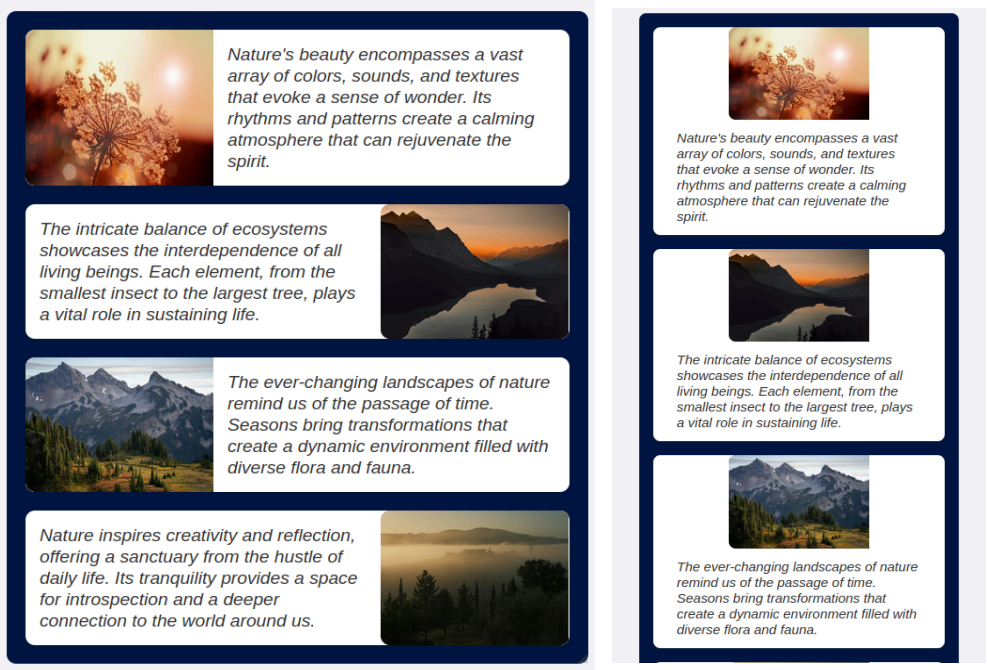
To achieve this, simply wrap all the images in a grid container and define the structure like this:

```
.grid {
  width: 100%;
  display: grid;
  grid-template-columns: repeat(auto-fill,
                         minmax(min(90vw, 250px), 1fr));
  gap: 10px;
}
```

Now, no matter what the screen size is, it'll always be responsive.

[Play around with it](#) on CodePen.

34. Alternate Image-Caption Cards



To create this layout, start by making the cards. Each card should have an image and a description, arranged in alternating orders like this:

```
<div class="card">
    <img src="img.jpg">
    <div class="text">Description
    </div>
</div>
<div class="card">
    <div class="text">Description</div>
    <img src="img.jpg">
</div>
<!-- And so on... -->
```

To set up the cards, make them flex containers with `flex-wrap: wrap`. Also, set a maximum width for the text. For cards that have the opposite order, use `wrap-reverse`. Here's the code:

```
.card{
    display: flex;
    flex-wrap: wrap;
}
.text{
    max-width: max(60%, 348px);
}
.card:nth-child(2n) {
    flex-wrap: wrap-reverse;
}
```

That's it, it's done. You'll have a responsive card layout like the one above.

Play around with it on CodePen.

---

5. Advanced Tricks and Layouts

This section explores advanced techniques to enhance your layouts. You'll discover innovative methods that elevate your designs.

From building unique structures to precise sizing control, these tricks will expand your toolkit. By the end, you'll have powerful strategies for creating intricate and dynamic layouts that impress your users.

35. Honeycomb Structure

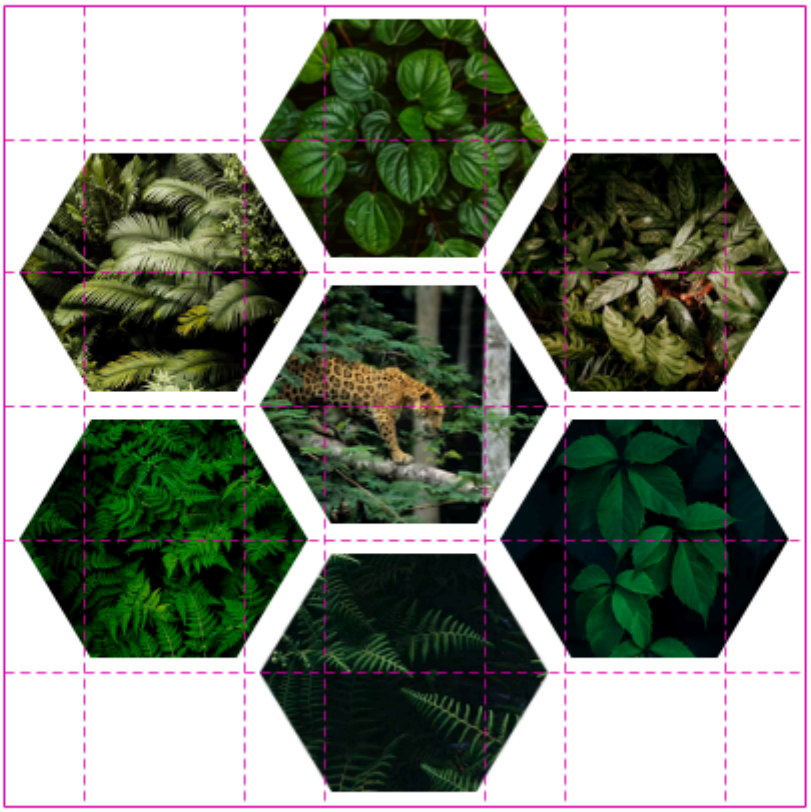
Creating this layout may seem tough, but it's simple with grid. Just follow these three steps:

1. Create seven hexagons using the `clip-path` property.
2. Wrap them in a container and set up a grid structure like this:

```
.wrapper {
  grid-template: repeat(6, 1fr) / 1fr repeat(3, 2fr 1fr);
```

```
}
```

Why this structure? Well, we want to use it like this:



3. Place and span these hexagons like this:

```
#hexagon1{grid-area: 1 / 3 / span 2 / span 3;}
#hexagon2{grid-area: 2 / 5 / span 2 / span 3;}
#hexagon3{grid-area: 4 / 5 / span 2 / span 3;}
/* And so on... */
```

And that's how you'll get a nice looking honeycomb structure.

Play around with it on CodePen.

36. Calendar in 3 Lines of CSS

For a full breakdown of this trick, check out the article on CSS-Tricks.

Here's the goal: creating a calendar layout like this:

First, set up a grid with seven equal columns to represent each day of the week, like this:

```
.calendar-container {
    display: grid;
    grid-template-columns: repeat(7, 1fr);
}
```

Next, place the first date in the third column to start from Tuesday, like this:

```
.first-day {
    grid-column-start: 3;
}
```

That's how you'll get this calendar layout in just three lines of CSS.

Play around with it on CodePen.

37. Patterns on Repeat

As we saw before, the `repeat()` function helps create multiple similar tracks with cleaner code.

But did you know you can also use it to `repeat()` a specific pattern? Here's how:

In the code below, the pattern `1fr 2fr` is repeated three times:

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr 2fr);
}
```

Using multiple tracks in the `repeat()` function enables you to design dynamic and organized layouts efficiently, giving your design flexibility and depth!

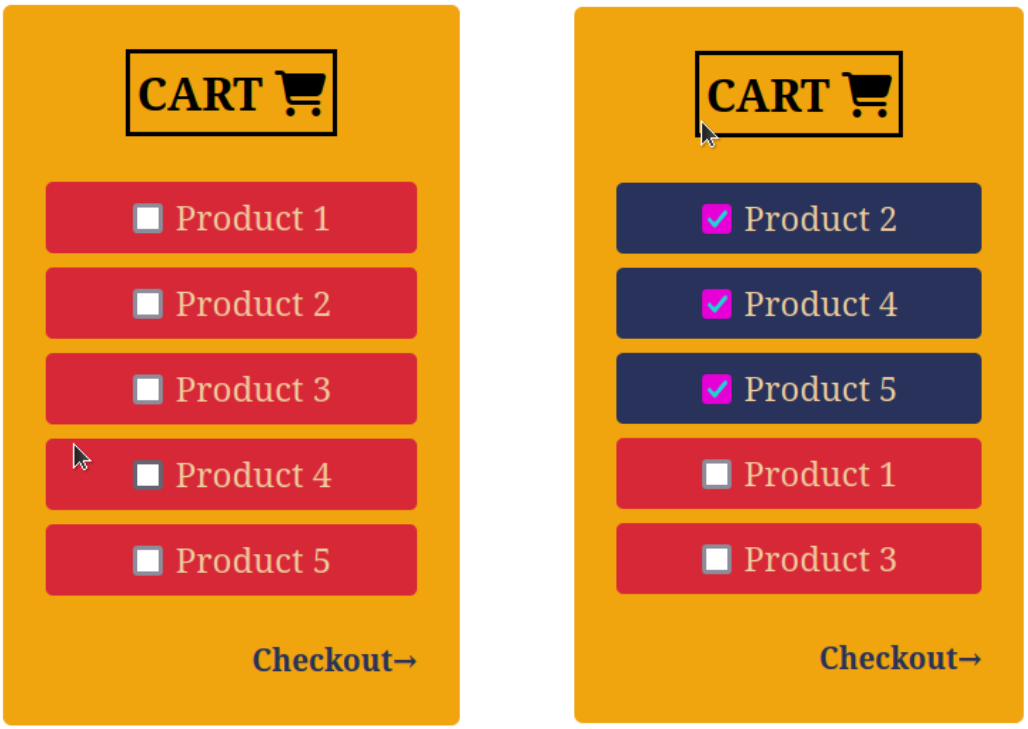
Play around with it on CodePen.

38. Reorder Cart Items

This trick is quite interesting.

Using Flexbox's order property, you can move selected items to the top of a product cart, making them stand out.

Like this:



Here's how to achieve this:

1. Make the cart wrapper a vertical flex.
2. Set the `order` value of the checked items to -1 to move them to the top.

But there's a catch: by default, the order value is 0, so checked items may appear above the cart heading. To fix this, set the `order` of the heading to -2.

Play around with it on CodePen.

39. Keyword-Based Track Size

So far, we've used numerical values to define track sizes, but you can also use keyword-based values like `min-content`, `max-content`, and `fit-content`.

Here's an example:

```
.grid-container {
    display: grid;
    grid-template-columns: min-content 1fr max-content;
}
```

This gives you more flexibility in controlling how your layout responds to content size, without just relying on fixed or relative units.

Play around with it on CodePen.

40. Layered Profile Design

How'd you create a layout like this in grid?

It's pretty simple to achieve.

First, wrap all three cards in a grid wrapper with this grid structure:

```
.wrapper {
    display: grid;
    grid-template-columns: 3fr 1fr 3fr 1fr 3fr;
}
```

Then create three classes like this:

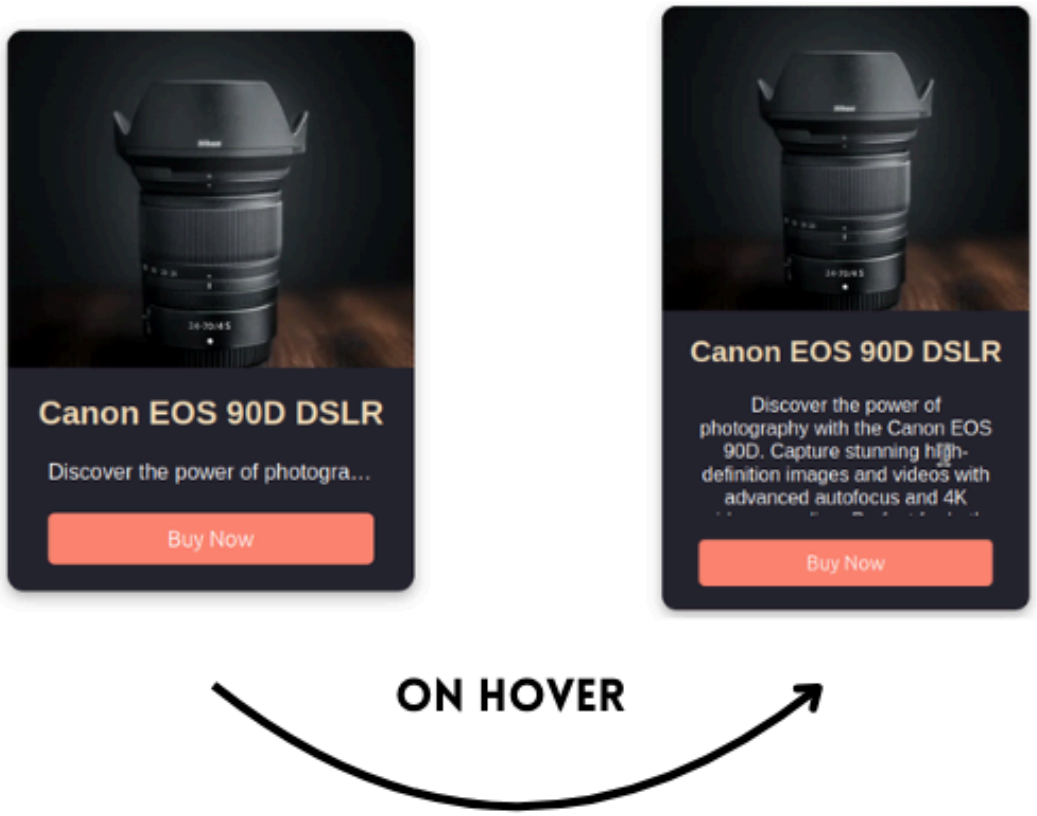
```
.leftCard {
    grid-area: 1 / 1 / span 1 / span 2;
}
.middleCard {
    grid-area: 1 / 2 / span 1 / span 3;
    z-index: 1;
}
.rightCard {
    grid-area: 1 / 4 / span 1 / span 2;
}
```

This will place the cards like this

You can also use JavaScript to change the card positions as you scroll by adding or removing the classes on different cards.

You can see it with added JavaScript on CodePen.

41. Expandable Description



Here's a card with an expandable description, created entirely with grid—no JavaScript or opacity tricks needed.

To start, make the card a grid container. This will make things easier than using flex.

Next, set a fixed height for the description row like this:

```
.card {
    display: grid;
    grid-template-rows: auto auto 20px auto;
}
```

For the description, set **overflow: hidden** initially to hide the content. Now, on hover, we'll change the height and overflow like this:

```
.card:hover {
    grid-template-rows: auto auto 100px auto;
}
.card:hover .desc {
    overflow: scroll;
}
```

And that's it! These are the main steps to achieve the effect. For more details and minor tweaks, you can explore the live demo on CodePen.

42. Named Grid Lines

When working with CSS Grids, you don't have to rely solely on numbered lines for placement. Instead, named grid lines offer a more intuitive approach, making your layout more readable and easier to maintain.

Here's how it works:

```
.grid-container {
    display: grid;
    grid-template-columns: [left-start] 1fr
                           [middle-start] 2fr [right-end];
}
.header {
    grid-column: left-start / right-end;
}
.main-content {
    grid-column: middle-start / right-end;
}
```

In this example, grid lines are named, like `left-start` for columns and top for rows.

This way, elements can be positioned based on grid areas, no need for numerical lines.

Play around with it on CodePen.

43. Formed Grid Areas

The layout above is simple to make with grid, but here's the real trick:

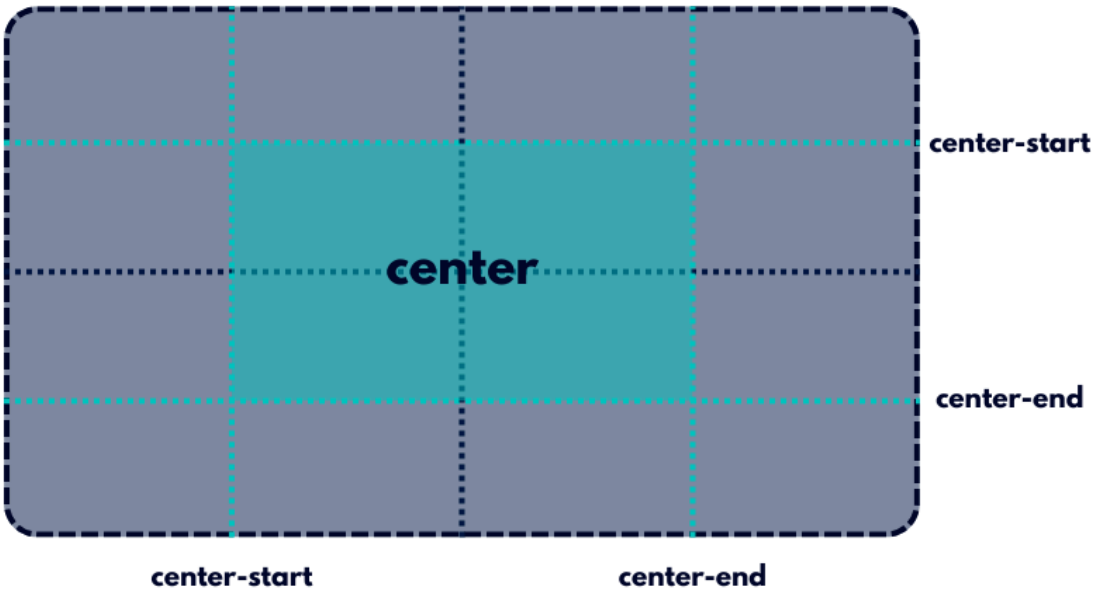
When you define named grid lines, you can enclose a specific grid area and use that area name to place items within it.

For example, let's say you name the grid lines like this:

```
.container{
    display: grid;
    grid-template: 1fr [center-start] 1fr 1fr
                   [center-end] 1fr / 1fr [center-start]
                   1fr 1fr [center-end] 1fr;
}
```

Then, you can simply refer to that area using the shared prefix (center) from those four lines.

Like this:



Now, you can place the heading in that area with just this single line:

```
.heading{
    grid-area: center;
}
```

One thing to note: to make this work, you must define all four lines and follow the pattern of using `[name]-start` for the starting lines and `[name]-end` for the ending lines.

Play around with it on CodePen.

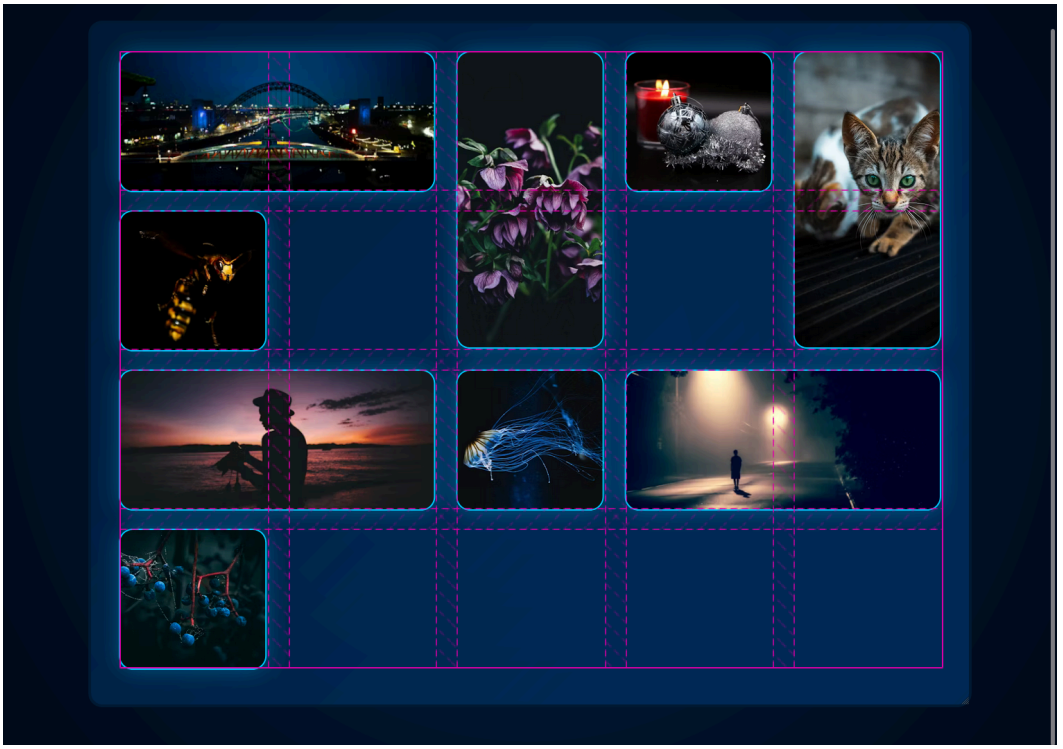
44. Dense Packing Algorithm

This trick is my favorite of all.

Imagine you have three types of images: portrait, landscape, and square. And you need to create a gallery out of them.

So, you span portrait images in two columns, landscapes in two rows, and square ones in just one row and column

Then you put them in a responsive grid container, so they become like this:



Notice those two gaps?

They appear because the wide image (guy holding a camera) was meant for the second column, second row. However, with the tall image already taking up space, it skipped it.

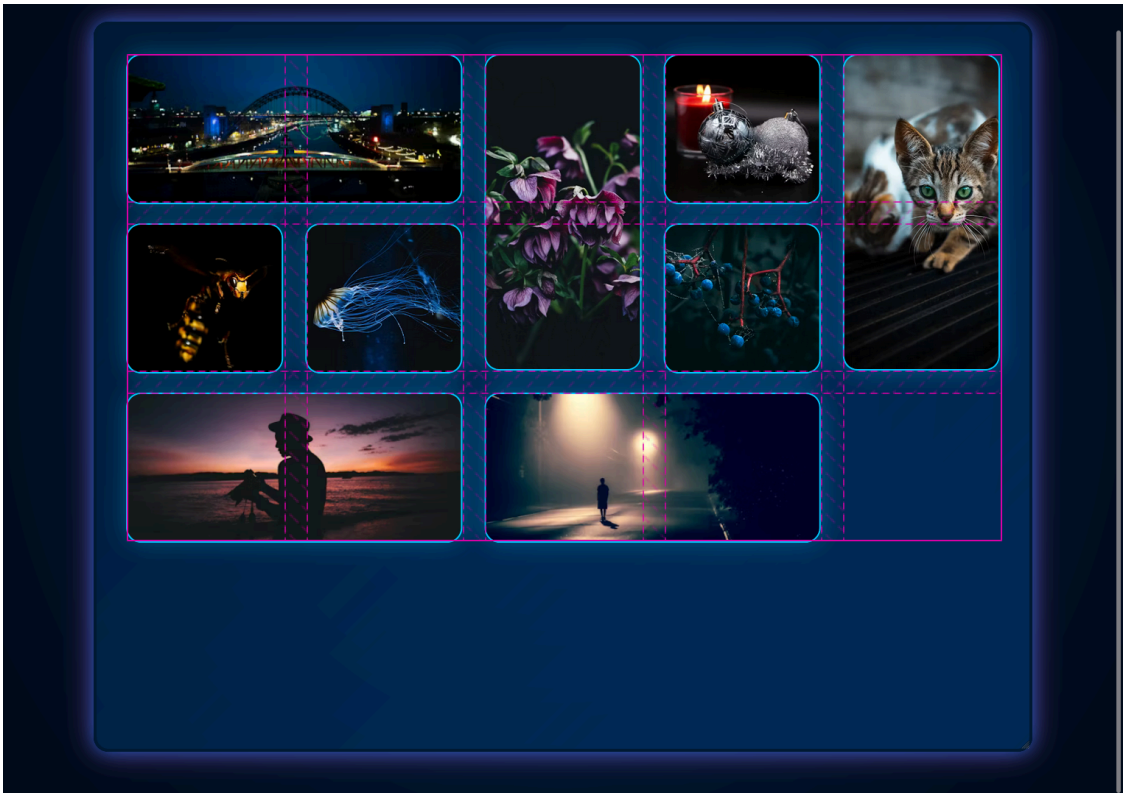
The same thing in the next available block.

Finally, it's placed in the first column, third row, creating those gaps.

You can fix this with one line:

```
.img-grid {
    grid-auto-flow: dense;
}
```

This line finds any images after the wide image that can fit in the empty spaces (like square images) and places them accordingly, fixing the layout like this:



Cool, right?

You can play around with it on CodePen.

45. The Subgrid Trick

Normally, when you have multiple similar items (like cards), they aren't aware of each other's content.

As a result, their content can't align properly.

Here's an example to show what I mean:

It's insanely simple with a subgrid. Here's what to do:

1. Wrap both cards in a grid container.
2. Create a grid structure with two columns (for the cards) and four rows (for each card's four elements).

Like this:

```
.wrapper {
    display: grid;
    grid-template: repeat(4, 1fr) / repeat(2, 1fr);
}
```

3. Span each card across all four rows in its column.
4. Make each card a grid container.
5. Set their `grid-template-rows` value to subgrid.

Like this:

```
.plan_cards {
    grid-row: 1 / -1;
    display: grid;
    grid-template-rows: subgrid;
}
```

After this, the layout will turn into this:

Subgrid is one of the most powerful features of CSS Grid, but it's still gaining browser support, with around 90% compatibility at the time of writing. So, check before using it.

Play around with it on CodePen.
