

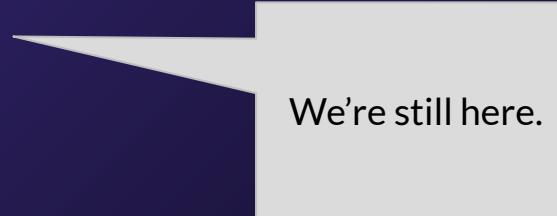


Week 5

CSS Layouts

The website development process

- Discovery and planning
 - What is the problem?
 - What are we doing to address it?
- Development
 - Actually doing the things
- Deployment
 - Shipping
- Maintenance
 - Keeping up and measuring success



We're still here.

In this lesson:

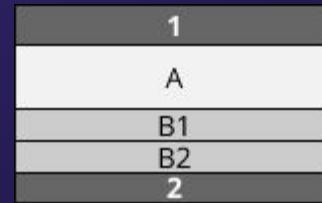
- Flex layouts
- Grid layouts
- Grid systems
- Making a responsive layout
- CSS frameworks

CSS, continued

In the previous CSS lesson we learned about its generalities. Now let's talk more about how to create complete **page layouts** - that is, the general structure of a page.

Common page layouts

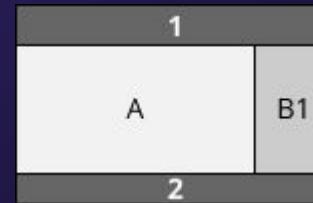
One column: Suitable for even the smallest screens (mobile devices, etc.)



Images from [Mozilla](#).

Common page layouts

Two columns: Okay for bigger screens.



Images from [Mozilla](#).

Common page layouts

Three columns: Only really suitable for big screens, such as PC monitors.



Images from [Mozilla](#).

Common page layouts

Doing responsive page design often means changing layouts depending on the screen width.



While this is possible with layout techniques used with `display: block` and other CSS attributes such as `float` and `clear`, the advent of new display values made these techniques **obsolete**.

Google Flights - Find Cheap Flig

google.com/...

Google Keep The Sunday Mass J...

All Bookmarks

Google

Flights

Round trip 1 Economy

San Francisco Where to?

Departure Return

Explore

Tracked prices

View all

- San Francisco to San Diego ↑ \$344
Cheapest flight · Round trip
\$236
- Oakland to San Diego ↓ \$137
Cheapest flight · Round trip
\$196
- San Diego to Oakland or San Francisco ↓ \$49
Cheapest flight · One way
\$78

Suggested trips from San Francisco

①

Mapt data ©2023 Google, INEGI

New York Jan 8 – 15, 2024 1 stop · 8 hr 59 min \$188

Google Flights - Find Cheap Flig

google.com/travel/flights

Google Keep The Sunday Mass J...

Read JavaScript Allo... ファイル - SkyDrive Learn to Code by D...

Collective Health

All Bookmarks

Google

Travel Explore Flights Hotels Vacation rentals

Flights

Round trip 1 Economy

San Francisco Where to?

Departure Return

Explore

Tracked prices

View all

- San Francisco to San... ↑ \$344
Cheapest flight - Round trip
\$236
- Oakland to San Diego ↓ \$137
Cheapest flight - Round trip
\$196
- San Diego to Oakland or San Francisco ↓ \$49
Cheapest flight - One way
\$78

Suggested trips from San Francisco

①

Explore destinations

New York Jan 8 – 15, 2024 1 stop · 8 hr 59 min \$188

Los Angeles Feb 8 – 17, 2024 Nonstop · 1 hr 38 min

Google Flights - Find Cheap Flig

google.com/...

Google Keep The Sunday Mass ... All Bookmarks

Google

Flights

Round trip 1 Economy

San Francisco Where to?

Departure Return

Explore

Tracked prices

View all

- San Francisco to San Diego ↑ \$344
Cheapest flight · Round trip
\$236
- Oakland to San Diego ↓ \$137
Cheapest flight · Round trip
\$196
- San Diego to Oakland or San Francisco ↓ \$49
Cheapest flight · One way
\$78

Suggested trips from San Francisco

①

New York Jan 8 – 15, 2024 1 stop · 8 hr 59 min \$188

Los Angeles Feb 8 – 17, 2024 Nonstop · 1 hr 38 min

Mapt data ©2023 Google, INEGI

Google Flights - Find Cheap Flig

google.com/travel/flights

Google Keep The Sunday Mass ... Read JavaScript Allo... ファイル - SkyDrive Learn to Code by D... Collective Health All Bookmarks

Google

Travel Explore Flights Hotels Vacation rentals

Flights

This section goes from one column to three columns

Departure Return

Explore

Tracked prices

View all

- San Francisco to San Diego ↑ \$344
Cheapest flight · Round trip
\$236
- Oakland to San Diego ↓ \$137
Cheapest flight · Round trip
\$196
- San Diego to Oakland or San Francisco ↓ \$49
Cheapest flight · One way
\$78

Suggested trips from San Francisco

①

Explore destinations

New York Jan 8 – 15, 2024 1 stop · 8 hr 59 min \$188

Los Angeles Feb 8 – 17, 2024 Nonstop · 1 hr 38 min

This screenshot shows the original Google Flights search interface. It features a dark-themed header with the Google logo and a "Flights" section. Below the header, there are search fields for "From" (San Francisco), "To" (Where to?), and date selection ("Departure" and "Return"). A large "Explore" button is centered below the search fields. To the right, a "Tracked prices" section displays flight price changes for routes like San Francisco to San Diego, Oakland to San Diego, and San Diego to Oakland or San Francisco. At the bottom, a "Suggested trips from San Francisco" section includes a map of North America with a blue dot over California and a thumbnail for a New York trip.



This screenshot shows the updated Google Flights search interface. The main difference is that the search fields for "From" (San Francisco) and "To" (Where to?) are now stacked vertically in a single column, reducing the overall height of the search bar. The date selection fields ("Departure" and "Return") remain separate. The "Explore" button is still present. The "Tracked prices" and "Suggested trips from San Francisco" sections are identical to the left screenshot. A prominent callout box in the center states: "Search inputs go from one col, two rows to one col, one row".

Search inputs go from one col, two rows to one col, one row

Google Flights - Find Cheap Flig... +

google.com/travel/flights

Google Keep The Sunday Mass J... Read JavaScript Allo... ファイル - SkyDrive Learn to Code by D... Collective Health All Bookmarks

Google Flights

Round trip 1 Economy

San Franc... Where to? Departure Return

Explore

Tracked prices

San Fran... ↑ \$344
Oakland t... ↓ \$137
San Diego... ↓ \$49

Suggested trips from San Francisco

New York Jan 8 - 15, 2024 1 stop · 8 hr 59 min \$188

Los Angeles Feb 8 - 17, 2024 Nonstop · 1 hr 38 min \$88

Explore destinations

This is a single column!

Elements Console Recorder Performance insights Sources Network

Styles Computed Layout Event Listeners

Filter

```
ot;United States&quot;], [[&quot;/m/d61p&quot;, 4], &quot;San Fr  
ancisco&quot;,[&quot;/m/d61p&quot;],&quot;San Francisco&quot;,  
[[&quot;https://encrypted-tbn1.gstatic.com/images?q=tbo03dtn:AN  
d9GerbCxCKTUbj5vz8PfReUUBMcm83fxKMrJpnz4UFNsPeL0kP1EZK9se1DX  
-y-VLxpPAaVb05zoihzXliHe13l0VFnWVz7UCQpkzCY&quot;],[&quot;ht  
tps://encrypted-tbn1.gstatic.com/images?q=tbo03dtn:ANd9GerbCxCK  
shv1R080CmVyc32HyosYzF6CnctDNW8T1xSxq81Xicn01cM6-rm7Kc  
u2QD53kOy&quot;],&quot;Golden Gate Bridge, cable cars u0026  
fog&quot;,[&quot;Hilly bayside city known for the Golden Gate Br  
idge, year-round fog, cable cars u0026 Victorian homes.&quot;]  
,[37.7749295,-122.419455],&quot;US&quot;,[&quot;United Sta  
tes&quot;],[&null;,&null;,&quot;/travel/flights&quot;]]]]  
jscontroller="eyhbk" jsaction="rcu0b:npT2md;zTgw:pjtQI;6vneH  
b:logIfDd;NIeule:vp9Hf;" data-node-index="1" jsmodel="hc60bd  
c-wiz">
```

Inherited from body#y0M0d.tQ5Y.ghyPEc.I..

.CQYF { /mss/bq-Ze,Aiq70d:1
font-family: Roboto, "Helvetica
Neue", Arial, sans-serif;
font-size: 14px;
font-weight: 400;
letter-spacing: 0.2px;
line-height: 20px;
color: #var(--ta884a309a24f44f4);
}
media screen and (prefers-color-scheme: dark)
body.where(:not(.S7wQg) / mss/bq-Ze,Aiq70d:1
):not([data-theme=dark]):not([data-
theme=light)) {
background-color: #rgb(32,33,36);
color: #rgb(232,234,237);
--t8cd825be1054f6b: #171717;
--t372c7ec7f19a9632: #3c0483;

--tf/148cf5018d50/4: #d2e3fc;
--t0e57d8ba9f73fa9: #d2e3fc;
--t9ab0d92230fd428d: #fff;
--t404202342c23b828: #var(--tf7148cf5018d5074);
--ta884a309a24f44f4: #9aa0a6;
--t67325fc9a5d6d4: #bdc1c6;
--tb31ee2eaa34f424: #rgba(232,234,237,0.38);
--t8f1ea4d4928e48: #e8ead;
--tfecd90dc541b96b: #bdc1c6;
--tf77282071a9ab: #e8ead;
--t342c96645d1fe13: #9aa0a6;
--t913f80c965bc42: #202124;
--t55e1fd139c688: #8a04f8;
--t6608c2157760b7: #f28862;
--t8ab0fb84d8e97495: #f6e699;
--ta071ba2f6f54-57: #ffd6d3;
--t49aa0f2f6a3b0718: #81c995;
--t2f4d90aab019535: #8abdab5;
--teaf0492bf6daa47c: #8abf4f8;
--tc54bd00368c12e0c: #f8f9f9a;
--t184c7395a93e1d18: #var(--t6c7325fc9a3d36d4);
--t9bcb1e15a274-cc: #d1-1-c;

The “modern” display modes

We've learned `display: block`, `display: inline` (the basic modes) and `display: inline-block`. While possible to build layouts with these, there are two modes more suited for building layouts: `flexbox` and `grid`.

Flexbox

A flex layout is done with `display: flex` on the parent element.

Its children will now behave like “flex children”. They will fall all in a single line at time (a column or a row; this is the “main axis”).

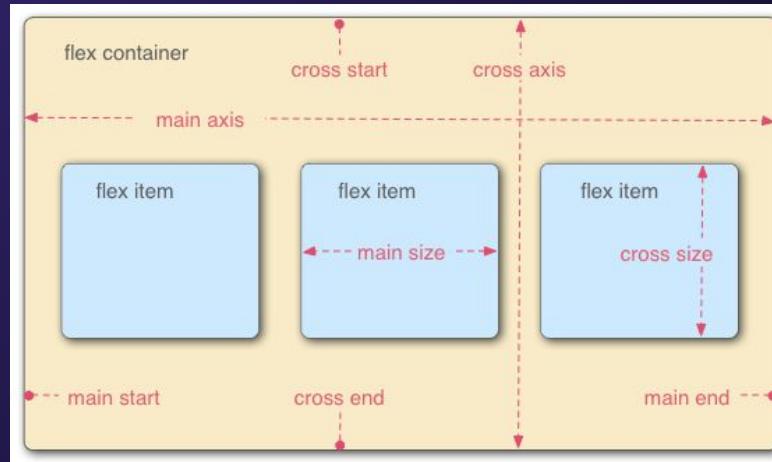


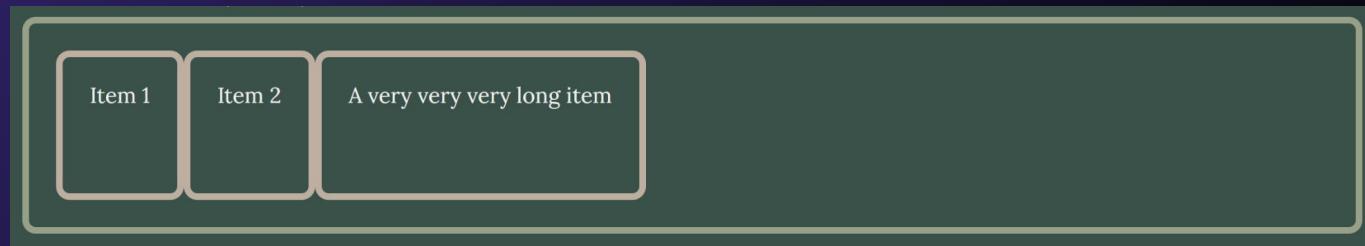
Image from [Mozilla](#).

Flexbox

A flex layout is done with `display: flex` on the parent element.

Its children will now behave like “flex children”. They will fall all in a single line at time (a column or a row; this is the “main axis”).

```
#parent {  
    display: flex;  
}
```



Flexbox: attributes in the parent

There are many ways
to configure the layout.

`flex-direction`
makes the main axis
horizontal or vertical.

```
#parent {  
  display: flex;  
  flex-direction:  
  row;  
}
```

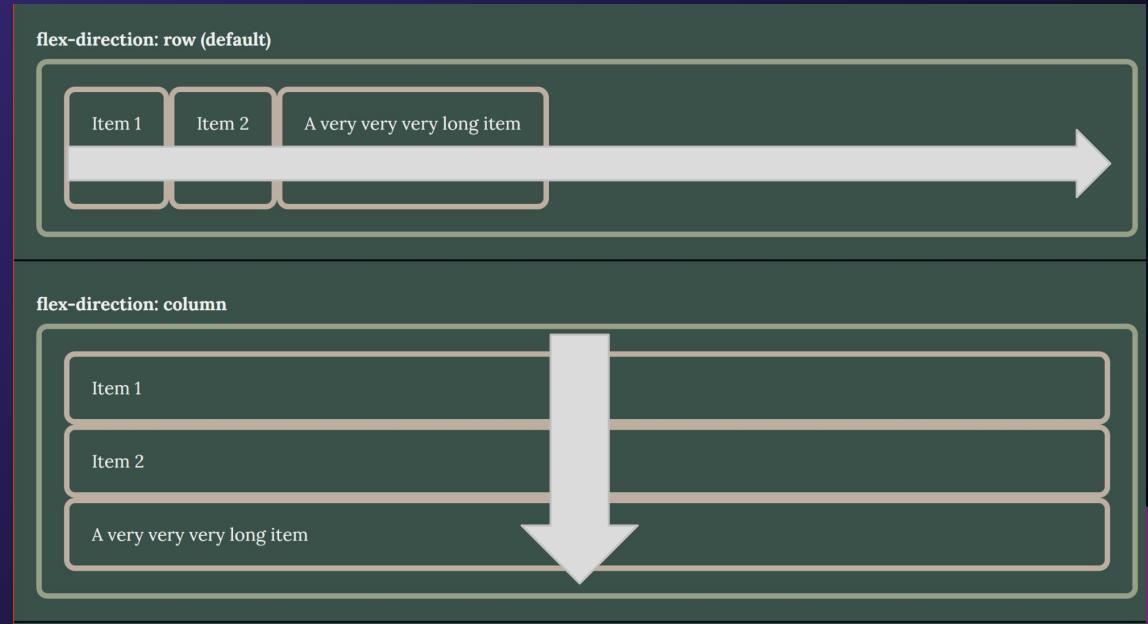


Flexbox: attributes in the parent

There are many ways
to configure the layout.

flex-direction
makes the main axis
horizontal or vertical.

```
#parent {  
  display: flex;  
  flex-direction:  
  row;  
}
```



Flexbox: attributes in the parent

There are many ways to configure the layout.

`flex-direction` makes the main axis horizontal or vertical.

It's also possible to reverse this line (eg. down-up instead of up-down).



Flexbox: attributes in the parent

justify-content regulates where, in the main line, the items are put - and also the spacing between them.

(In all cases, **flex-direction** here is row).

`justify-content: flex-start (default)`



`justify-content: flex-end`



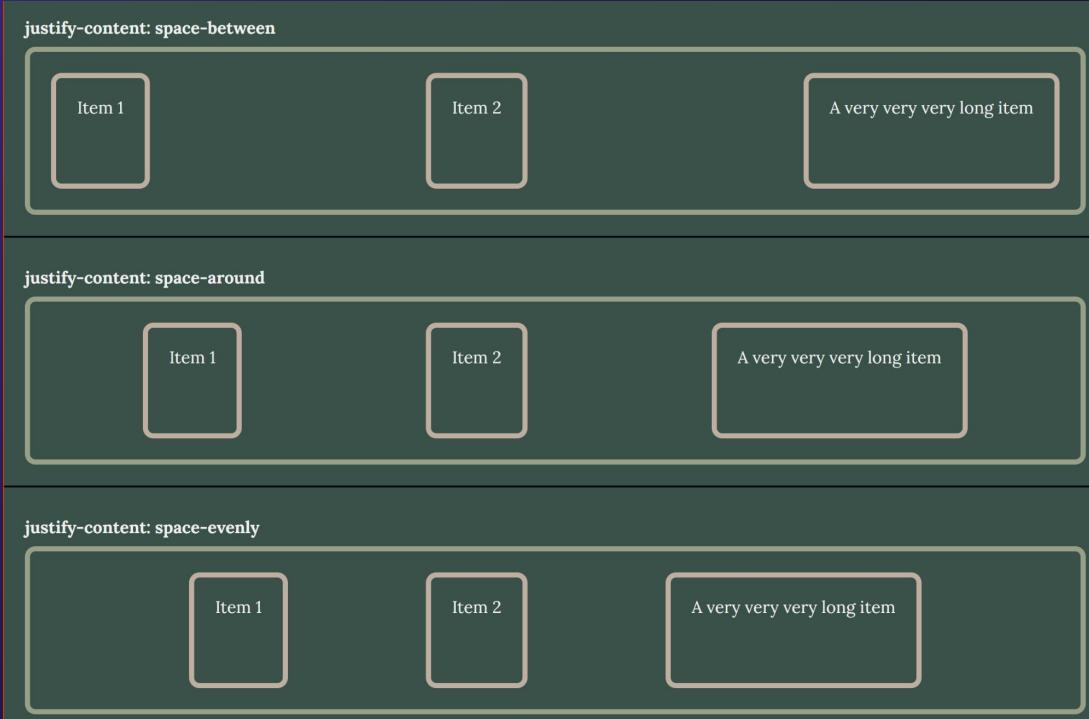
`justify-content: center`



Flexbox: attributes in the parent

justify-content regulates where, in the main line, the items are put - and also the spacing between them.

(In all cases, **flex-direction** here is row).



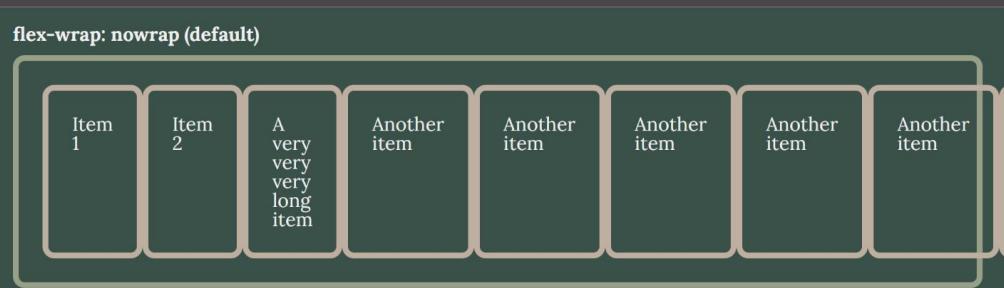
Flexbox: attributes in the parent

`gap` sets a **minimum** spacing between items. Can be used instead of adding margin on the children.



Flexbox: attributes in the parent

flex-wrap
determines if the
line can be broken
in many lines and
if so, how. By
default, it won't.



Flexbox: attributes in the parent

`align-items` sets
the children on the
secondary axis.

In these examples,
the height of the
parent has been set
to be higher using
`min-height:`
`16rem`.

The image displays three separate flexbox containers, each containing three items: "Item 1", "Item 2", and "A very very very long item". The containers are stacked vertically. The top container is labeled "align-items: flex-start (default)". The middle container is labeled "align-items: flex-end". The bottom container is labeled "align-items: center". In all three cases, "Item 1" and "Item 2" are small rounded rectangles, while the long item is a larger rectangle. In the "flex-start" case, the long item is at the top. In the "flex-end" case, the long item is at the bottom. In the "center" case, the long item is centered vertically between the two smaller items.

Flexbox: attributes in the parent

`align-items` sets
the children on the
secondary axis.

We can also
stretch the
items to occupy
the whole space.



Flexbox: attributes in the parent

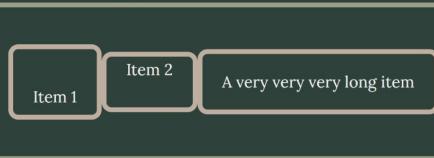
`align-items` sets the children on the secondary axis.

`align-items: baseline` means items are aligned so that the first line of **text** in all of them falls in the same line, regardless of paddings and margins.

`align-items: center`

These elements have been arranged with different padding-top and padding-bottom values.

Since we're arranging the elements and not the text, now the text is misaligned.



`align-items: baseline`

These elements have been arranged with different padding-top and padding-bottom values.

Note how all of the text still falls on the same line.



Flexbox: the flex attribute in children

There are also attributes that are set directly in the children. The most important of us is `flex`.

`flex` tells the browser how much space each element takes on the main axis, and how it should distribute the space available between the elements. It is a combination of three values: `flex-grow`, `flex-shrink` and `flex-basis`.

```
.children {  
    flex: 0 1 auto;  
}
```

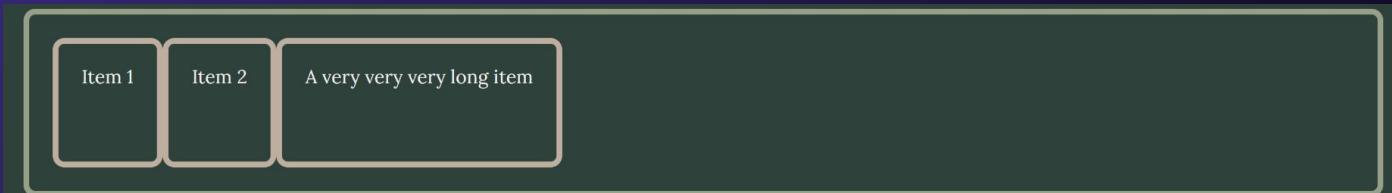
The same

```
.children {  
    flex-grow: 0;  
    flex-shrink: 1;  
    flex-basis: auto;  
}
```

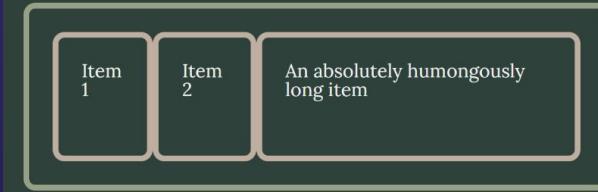
Flexbox: the flex attribute in children

By default (`flex: 0 1 auto`):

- Children will only take the space needed for their content;
- If there is not enough space, children will be made smaller as much as possible to try to fit.



If there is not enough space, children will be made smaller as much as possible to try to fit.



Flexbox: the flex attribute in children

Often we will want the children to use all of the space of the parent.

The concept of “available space”:

- When there is more space in the parent than what the children naturally occupy, there is “positive free space”. This is space that flex can use to make the children grow.

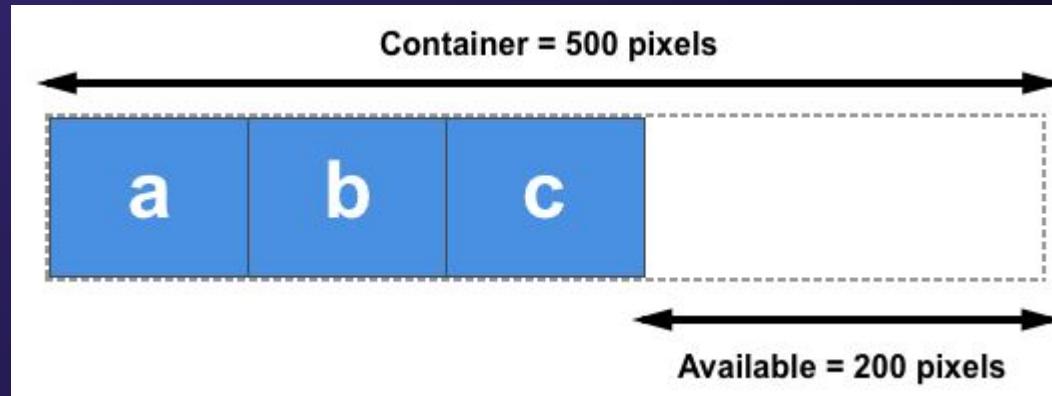


Image from [Mozilla](#).

Flexbox: the flex attribute in children

The concept of “available space”:

- When there is **less** space in the parent than what the children naturally occupy, there is “negative free space”. **flex** will try to reclaim this space from the children.

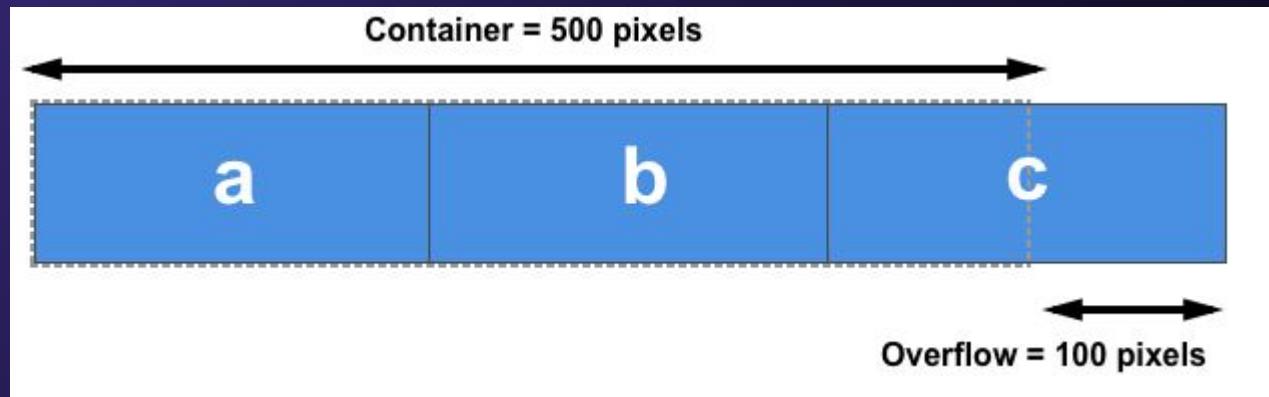
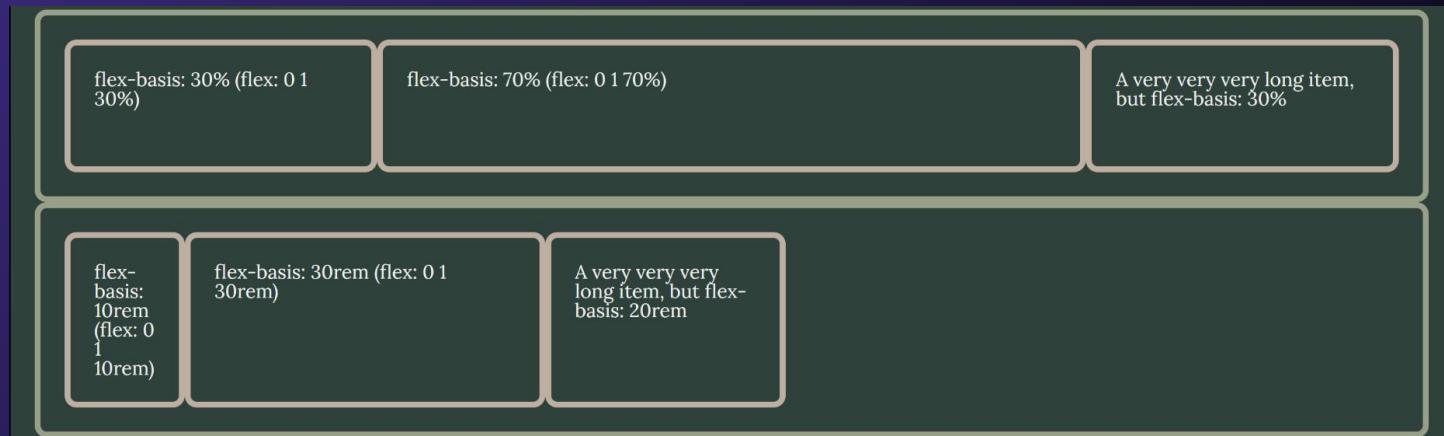


Image from [Mozilla](#)

Flexbox: the flex attribute in children

flex-basis:

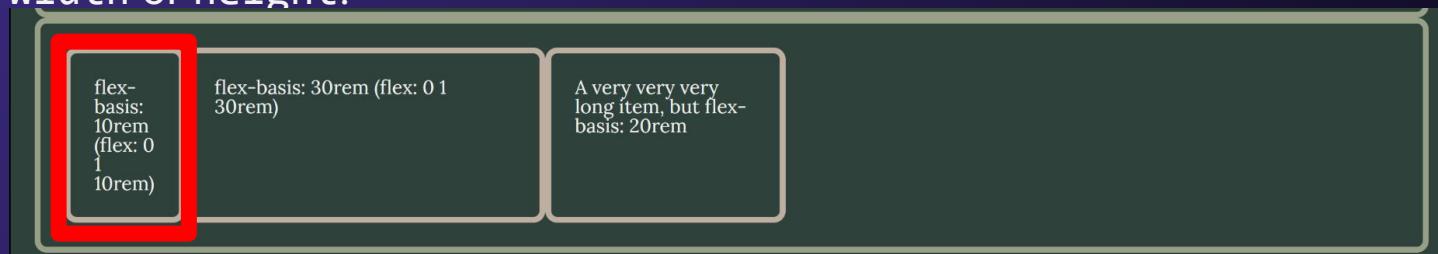
When provided, it tells the browser how big is the children (instead of the size needed by its content).



Flexbox: the flex attribute in children

flex-basis:

If its value is auto, we can still set the size with other attributes such as width or height.



```
.first-children {  
  flex: 0 1 10rem;  
}
```

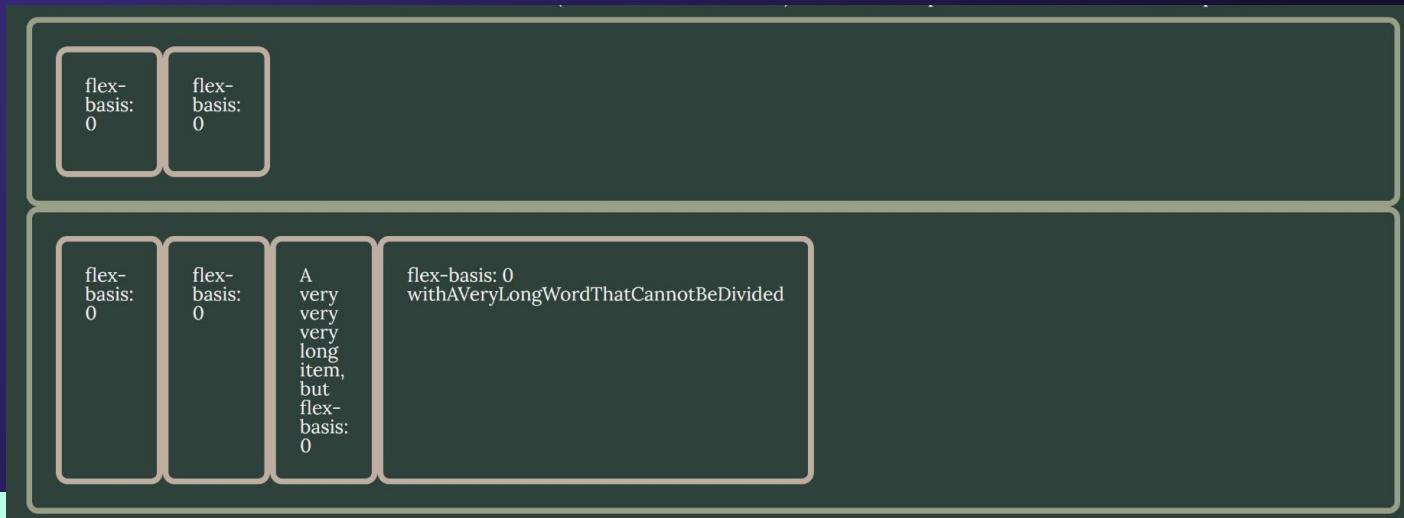
The same

```
.first-children {  
  flex: 0 1 auto;  
  width: 10rem;  
}
```

Flexbox: the flex attribute in children

flex-basis:

If `flex-basis: 0`, flex assumes the child's size is actually 0, and will compress its contents as much as possible - for example, breaking lines where there are spaces.



Flexbox: the flex attribute in children

flex-grow:

This tells the browser to give this child some of the available space in the parent.

How much? It's a proportion of the space requested by all children.

For example, if all children have `flex-grow: 1` (or any number, but the same number in all of them), the browser will break the available space in equal parts and give each to each children.



Flexbox: the flex attribute in children

flex-grow:

If children have different flex-grow values, the ones with higher value get more space added:

- First, the values of all children are summed. In this case, $1+2+1 = 4$.
- Then, space is distributed according to the proportion (this child's value / total of values).

In this case, the center item ends up with $2/4 = 50\%$ and the others end up with $1/4 = 25\%$ of the available space.

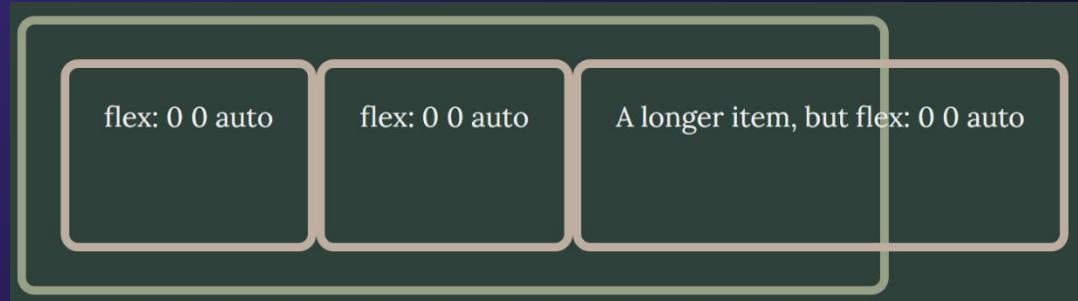


Flexbox: the flex attribute in children

flex-shrink:

By default (`flex-shrink: 1`), the elements will shrink to fit the available space in the parent.

With `flex-shrink: 0`, they won't.

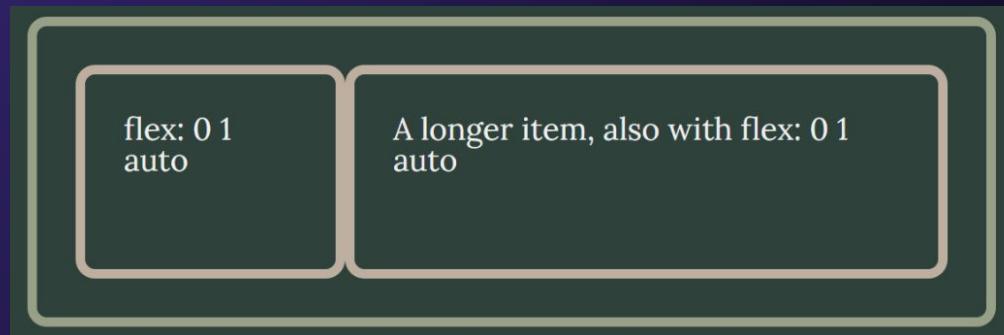


Flexbox: the flex attribute in children

flex-shrink:

With the `flex-shrink: 1` default, all children won't shrink exactly the same. Instead, the browser calculates a proportion adequate to each child's content.

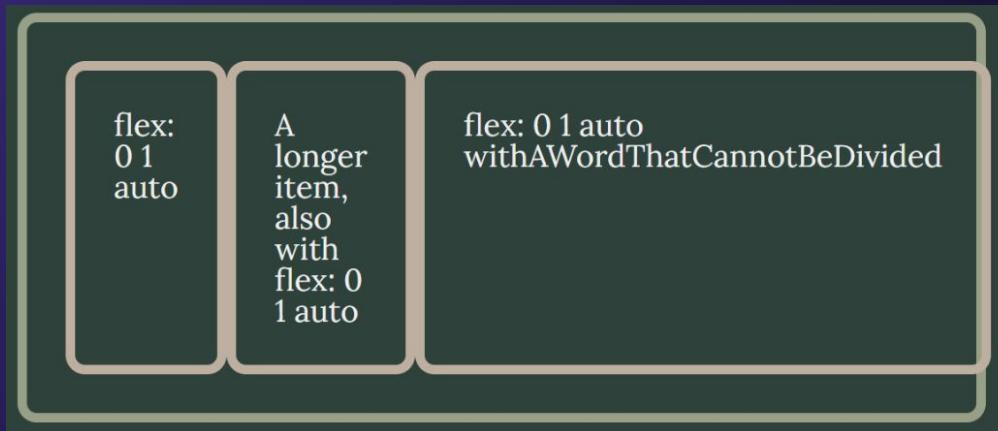
- First, each child element tries to take as little space for their contents - for example, making text lines shorter.
- Then, space gets removed. Elements with more content get more space removed, even if `flex-shrink` is the same.



Flexbox: the flex attribute in children

flex-shrink:

If after all of that the contents are still too big, they will overflow the parent.

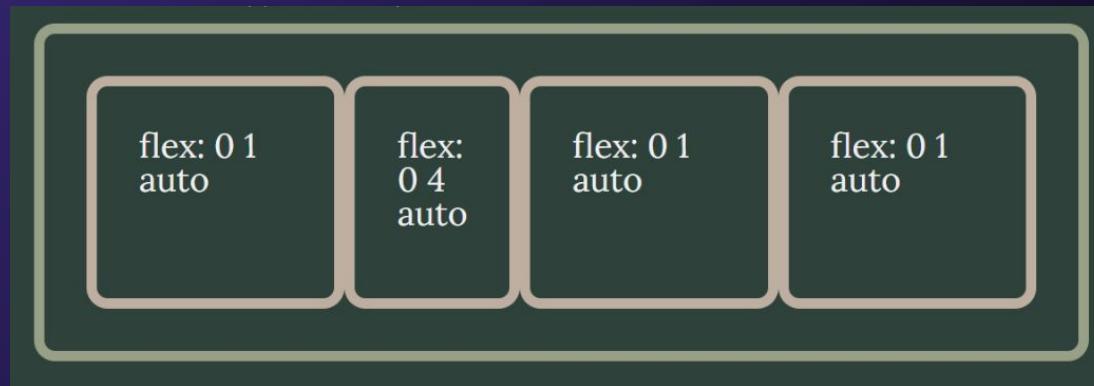


Flexbox: the flex attribute in children

flex-shrink:

With different flex-shrink values, the higher the value, the more space gets removed from the element.

This is similar to how flex-grow works, but it's not an exact proportion: elements with more contents still get more space removed.



Flexbox layouts

Using the attributes above, it's possible to easily build layouts. For example:

```
/* parent element */
article {
  display: flex;
}

/* center column */
.three-column .content {
  flex: 5 1 0;
}

/* sidebars */
.three-column .sidebar {
  flex: 2.5 1 0;
}
```



Responsive flexbox

Mobile-first layouts are one-column only, and get more columns as the screen grows wider.

To achieve this effect using flexbox:

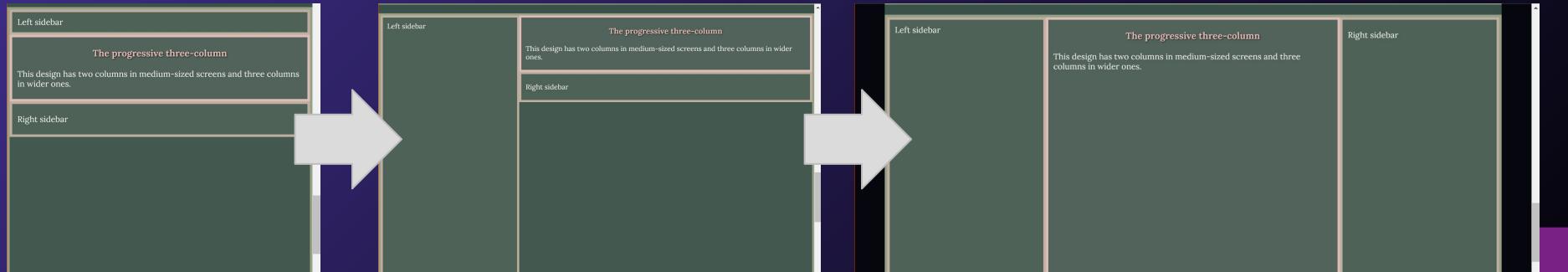
- Start with a base one-column layout.
- Next, set up breakpoints for wider screens. Inside them, write CSS that overwrites the base layout to create the columns.

Define a breakpoint with the CSS instruction `@media (...) {}`.



Responsive flexbox

An example: a layout that is one-column for smaller screens, two-column for medium ones, and three-column for wider ones.



Responsive flexbox

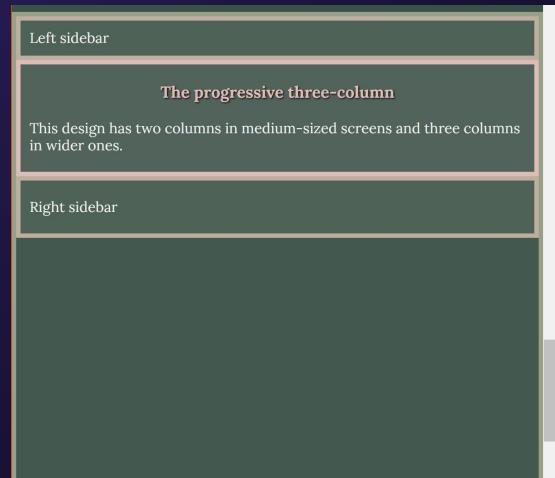
To achieve this, the HTML structure looks as follows. Note there's a **right-wrapper** div around the **content** and **sidebar-right** divs.

```
<article class="three-column-progressive">
  <div class="sidebar sidebar-left">
    Left sidebar
  </div>
  <div class="right-wrapper">
    <div class="content">
      <h2>The progressive three-column</h2>
    </div>
    <div class="sidebar sidebar-right">
      <p>Right sidebar</p>
    </div>
  </div>
</article>
```

Responsive flexbox

The base styles are the ones for the smaller screens.

```
/* parent element */  
article {  
    display: flex;  
    flex-direction: column;  
}  
  
/* no particular styles for  
children at this point */
```

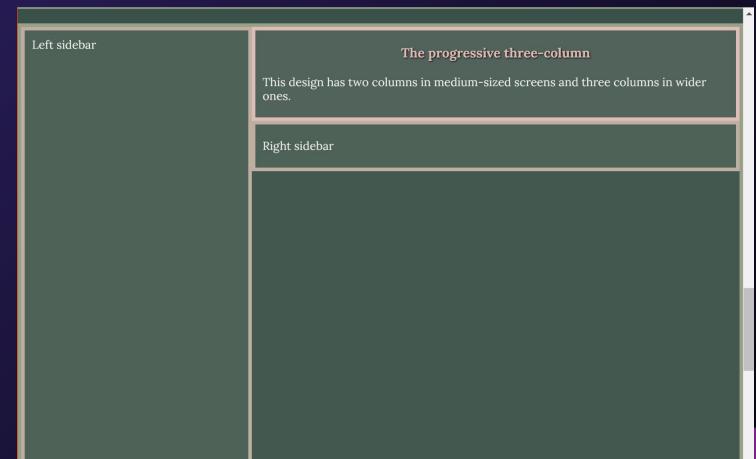


Responsive flexbox

Next, styles for screens that are at least 640px wide. Let's create a first breakpoint.

Only the wrapper will get a `flex` value for now. Also, we flip the `flex-direction` of the parent to be `row`.

```
@media (min-width: 640px) {  
  .three-column-progressive {  
    flex-direction: row;  
  }  
  
  .three-column-progressive .sidebar-left {  
    flex: 3 1 0;  
  }  
  
  .three-column-progressive .right-wrapper {  
    flex: 7 1 0;  
  }  
}
```



Responsive flexbox

Lastly, a third breakpoint for the wider screens. We adjust widths again. Also, .right-wrapper becomes a flex parent itself (`display: flex`) which allows us to define another flexbox for its children.

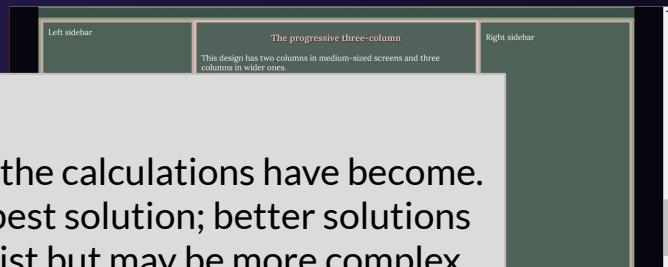
```
@media (min-width: 1024px) {  
    .three-column-progressive .right-wrapper {  
        flex: 7.6 1 0; /* it's a children! */  
        display: flex; /* it's also a parent! */  
    }  
    .three-column-progressive .right-wrapper .content {  
        flex: 6.6 1 0;  
    }  
    .three-column-progressive .right-wrapper .sidebar {  
        flex: 3.4 1 0;  
    }  
    /* the left sidebar also needs to become thinner */  
    .three-column-progressive .sidebar-left {  
        flex: 2.4 1 0;  
    }  
}
```



Responsive flexbox

Lastly, a third breakpoint for the wider screens. We adjust widths again. Also, .right-wrapper becomes a flex parent itself (`display: flex`) which allows us to define another flexbox for its children.

```
@media (min-width: 1024px) {  
    .three-column-progressive .right-wrapper {  
        flex: 7.6 1 0; /* it's a child! */  
        display: flex; /* it's also a parent! */  
    }  
    .three-column-progressive .right-wrapper {  
        flex: 6.6 1 0;  
    }  
    .three-column-progressive .right-wrapper {  
        flex: 3.4 1 0;  
    }  
    /* the left sidebar also needs to become thinner */  
    .three-column-progressive .sidebar-left {  
        flex: 2.4 1 0;  
    }  
}
```



Note how fussy the calculations have become.
This is not the best solution; better solutions
with flexbox exist but may be more complex.



Flexbox layouts

Created by HideMaru
from Noun Project

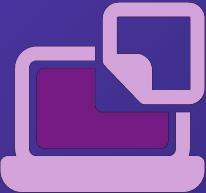
Files with sample layouts have been provided as part of the course contents.

Review how these layouts were constructed.

Then, **create your own example:**

- In a new set of files, create your own HTML and CSS.
- Create a two-column layout with a vertical navigation in the sidebar. The navigation itself should use `display: flex`.

Tip: if you run into trouble, use the Developer Tools to debug the CSS that applies to the parent and the children.



Practice: Flexbox

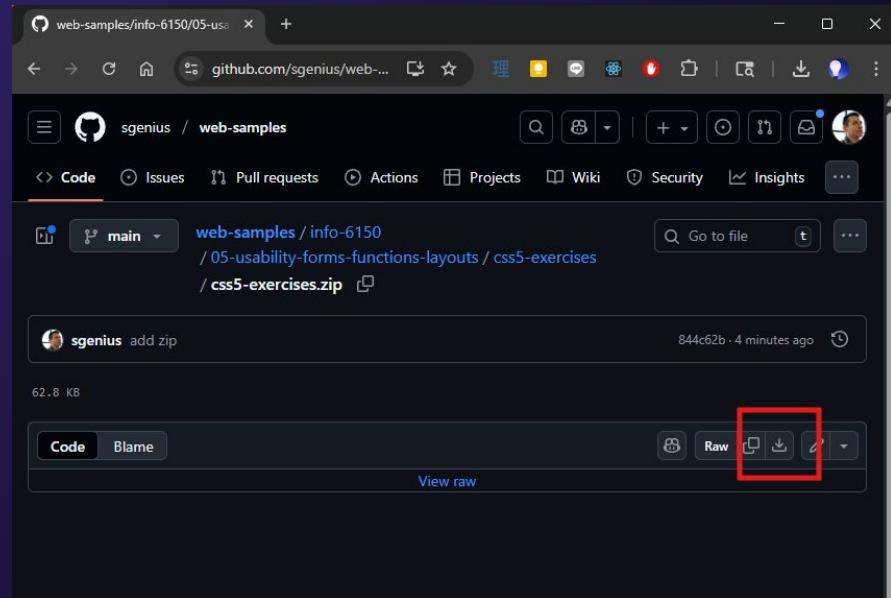
Created by HideMaru
from Noun Project

Download the following files and save them locally: [zip file](#)

(You need to click on the “Download raw file” button)

Then, open the “flex.html” file. It will give you instructions for three exercises. Do them locally, in your computer.

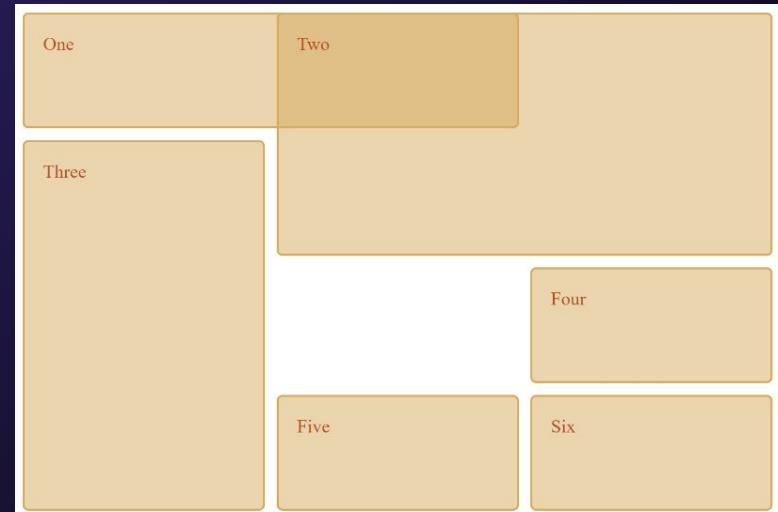
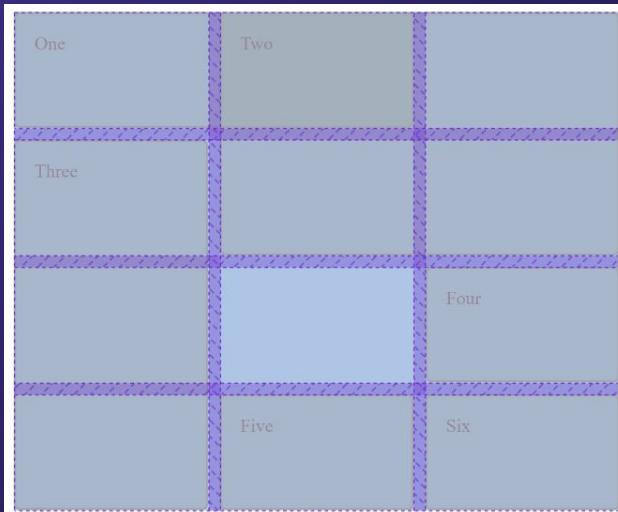
First, try to solve them by yourself (don’t ask help to the AI).



Grid

`display: grid` is the other “modern” display mode.

It allows you to create an invisible grid first, then specify where each element falls within this grid.



Grid: attributes in the parent

display: grid also has attributes both in the parent and in its immediate children.

- In the parent we'll define the grid itself.
- Then, we'll position each child in the grid independently.

Grid: attributes in the parent

display: grid also has attributes both in the parent and in its immediate children.

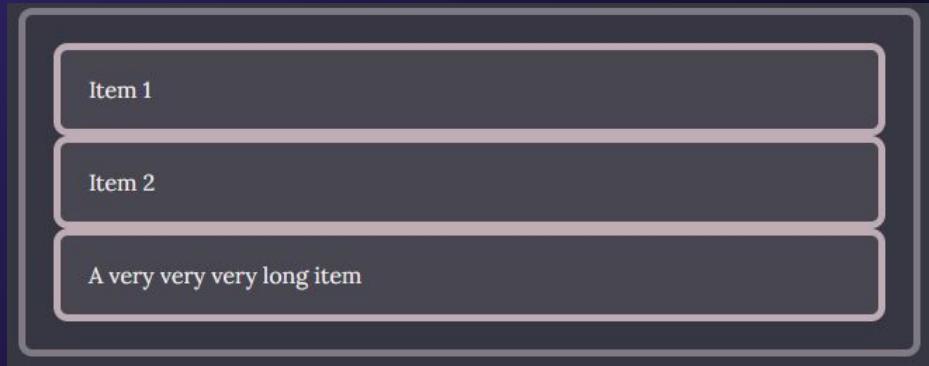
- In the parent we'll define the grid itself.
- Then, we'll position each child in the grid independently.

A default grid

Let's start with a default grid. At this point, it's the same as a display: inline: it creates a single column and a row per element.

```
<div class="grid">  
  <div>Item 1</div>  
  <div>Item 2</div>  
  <div>A very very very long item</div>  
</div>
```

```
.grid {  
  display: grid;  
}
```



A default grid

The children will change if the parent changes. Here we force the parent to be higher.

```
<div class="grid" style="height: 600px">  
  <div>Item 1</div>  
  <div>Item 2</div>  
  <div>A very very very long item</div>  
</div>
```

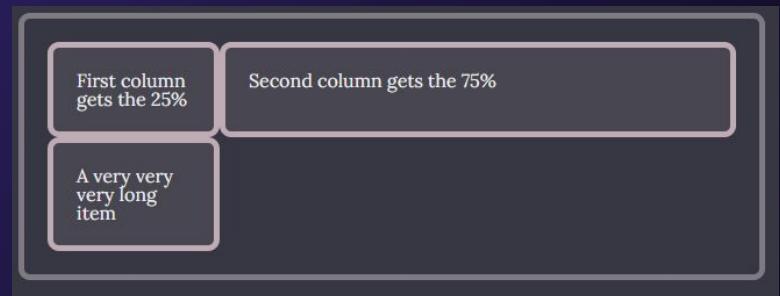
```
.grid {  
  display: grid;  
}
```



Defining columns

With grid-template-columns we express what columns exist and how big they are.

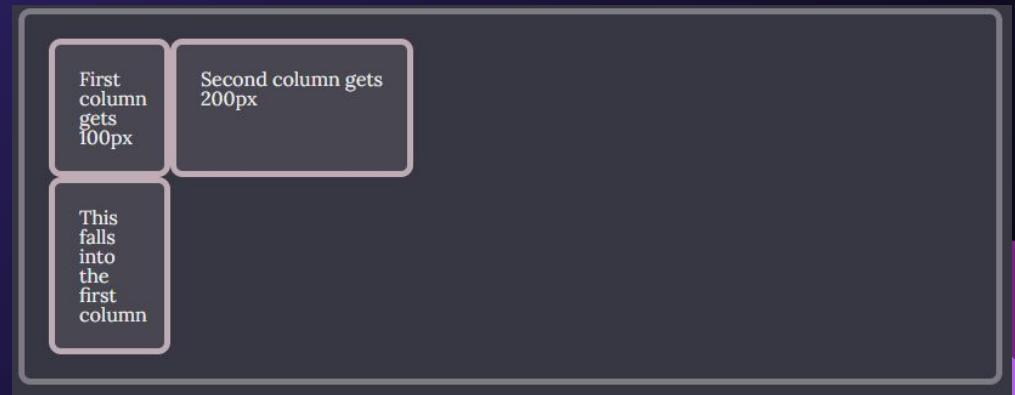
```
<div class="grid"  
style="grid-template-columns: 25% 75%;  
">  
    <div>Item 1</div>  
    <div>Item 2</div>  
    <div>A very very very long item</div>  
</div>  
  
.grid {  
    display: grid;  
}
```



Defining columns

Widths can be expressed with any measurement, such as px or rem.
If our columns are smaller than the available parent width, there will be a gap.

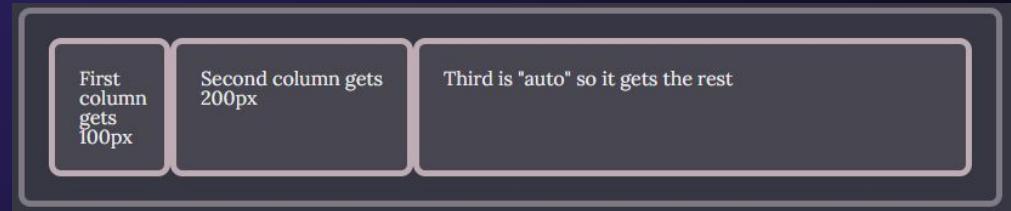
```
<div class="grid" style="grid-template-columns: 100px 200px">  
    ...  
</div>
```



Defining columns

We can use the keyword “auto” to create a column that fills that gap.

```
<div class="grid" style="grid-template-columns: 100px 200px auto">  
    ...  
</div>
```



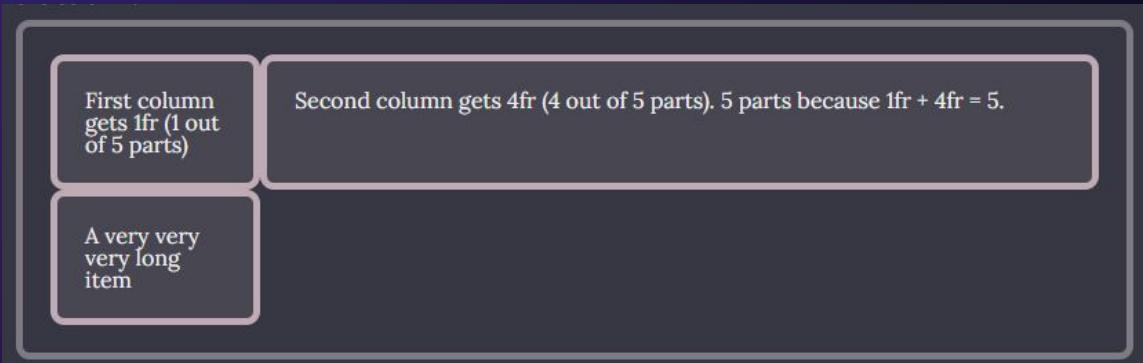
Defining columns with “fr”

Another way to define columns that extend through the available width is using the special measurement “fr” (“fraction”).

This breaks the parent width in “parts”. The sum of all fr values is the width of the parent.

The higher the fr value is for a column, the wider it is.

```
<div class="grid" style="grid-template-columns: 1fr 4fr">  
  ...  
</div>
```



Defining rows

Even though rows are automatically created, we can choose to use `grid-template-rows` to control, for example, the size of the rows:

```
<div class="grid" style="grid-template-columns: 1fr 4fr; grid-template-rows: 2fr 1fr">  
    ...  
</div>
```



Defining rows

If the grid needs more rows than we have defined, new “default” rows are created automatically. Here we only defined two rows, but need more.

```
<div class="grid" style="grid-template-columns: 1fr 4fr; grid-template-rows: 1fr 3fr">  
    /* lots of children */  
</div>
```



The repeat keyword

Let's say you have this, which is very long to write:

```
<div class="grid" style="grid-template-columns: 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr">  
  /* nine elements go here */  
</div>
```



The repeat keyword

Since all of the columns are the same, you can do this instead:

```
<div class="grid" style="grid-template-columns: repeat(9, 1fr)">  
  /* nine elements go here */  
</div>
```

Use `repeat` for values that are repeated. This works for columns and rows.



Align content, justify content

This works the same as in flexbox.

```
<div class="grid" style="height: 600px; align-content: center">  
  ...  
</div>
```



Align content, justify content

This works the same as in flexbox.

```
<div class="grid" style="height: 600px; align-content: space-evenly">  
  ...  
</div>
```



Align content, justify content

This works the same as in flexbox.

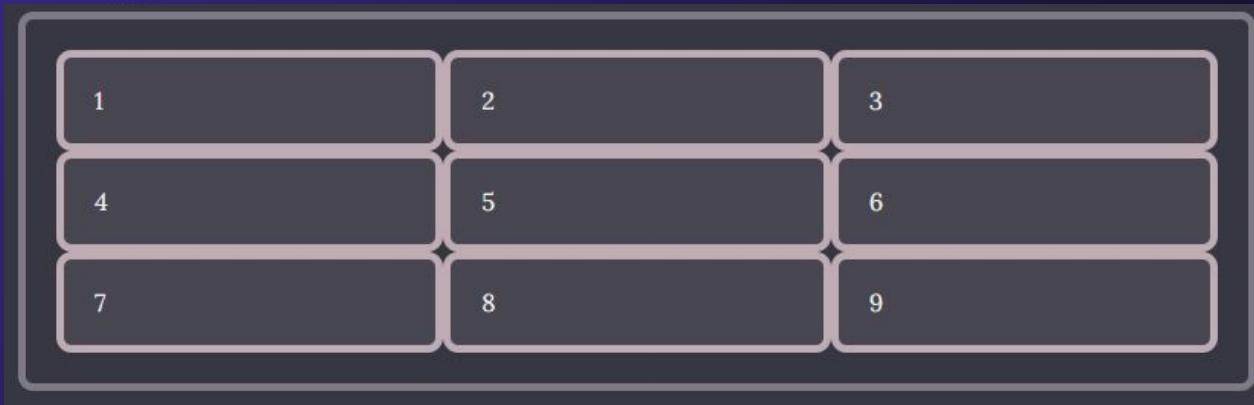
```
<div class="grid" style="height: 600px; justify-content: end">  
  ...  
</div>
```



Gaps

This also works the same as in flexbox.
Here, our parent has three columns of equal size.

No gap:

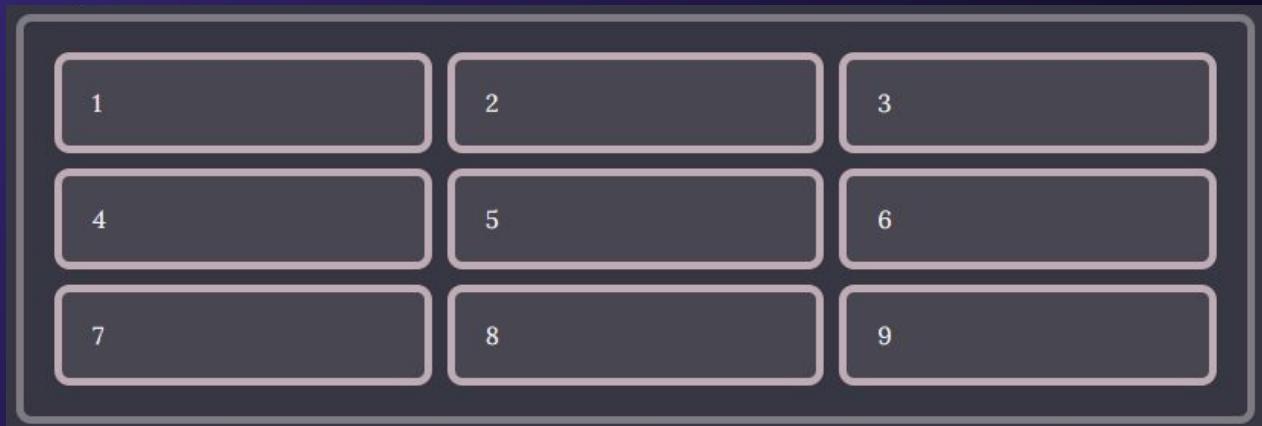


Gaps

This also works the same as in flexbox.
Here, our parent has three columns of equal size.

With gap:

```
<div class="grid" style="gap: 1rem">  
  ...  
</div>
```



Line-based placement

So far, all of the children occupy only one “cell” of the grid, in order.

We will now learn how to change this. We do this with attributes that go in the children.

We will study two methods:

- Line-based placement
- Named areas

Line-based placement

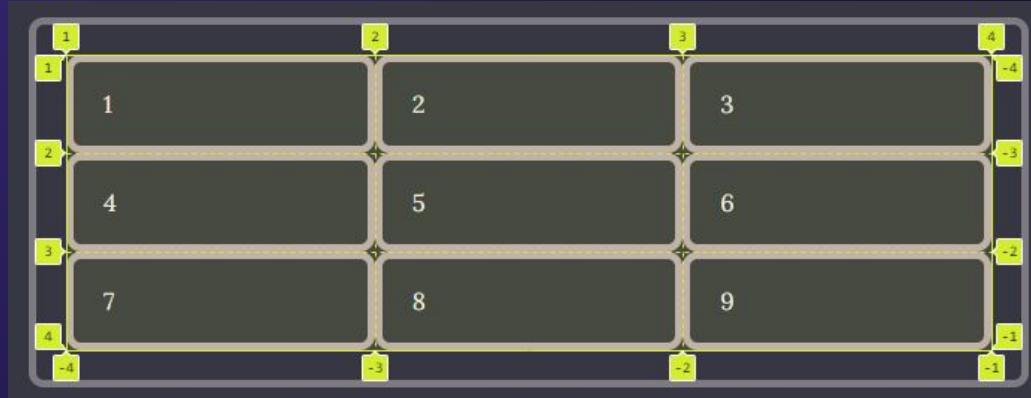
Observe this grid:



Line-based placement

Imagine the lines that divide the grid: four horizontal and four vertical lines.

This is how they appear with the grid analyzer of Chrome's developer console.
Note they are numbered.

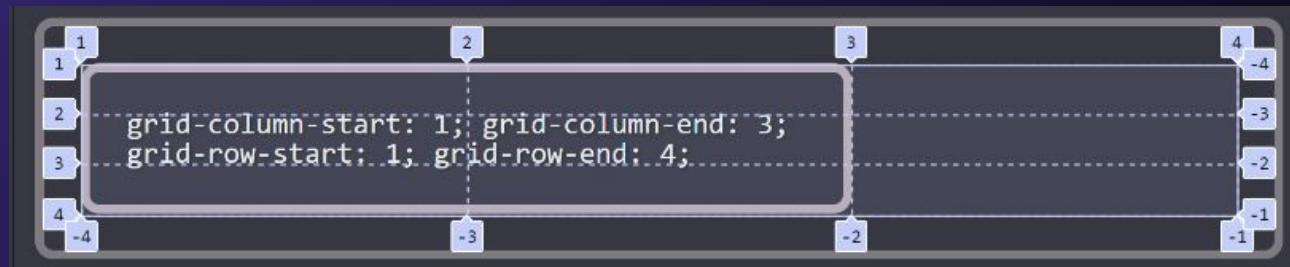


Line-based placement

We can use these numbers to tell each cell, “go from this line to this line”.

We do it like this:

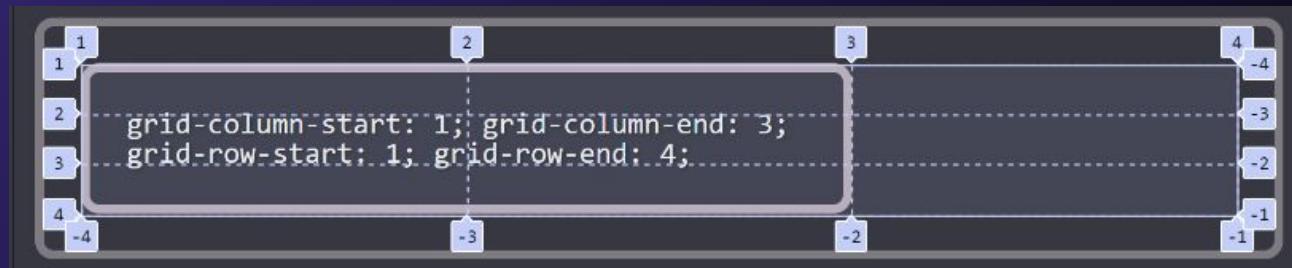
```
<div class="grid">  
  <div style="grid-column-start: 1; grid-column-end: 3; grid-row-start: 1; grid-row-end: 4;">  
    ...  
  </div>  
</div>
```



Line-based placement

grid-column-start and grid-column-end refer to vertical lines.
grid-row-start and grid-row-end refer to horizontal lines.

```
<div class="grid">  
  <div style="grid-column-start: 1; grid-column-end: 3; grid-row-start: 1; grid-row-end: 4;">  
    ...  
  </div>  
</div>
```



Line-based placement

We can play with these values for all cells.

Nothing specified. By default: one by one

```
grid-column-start: 2;  
grid-column-end: 3;  
grid-row-start: 2;  
grid-row-end: 3;
```

```
grid-column-start: 3;  
grid-column-end: 4;  
grid-row-start: 3;  
grid-row-end: 4;
```

Line-based placement

We can also make overlapping cells!

```
grid-column-start: 1;  
grid-column-end: 2;  
grid-row-start: 2;  
grid-row-end: 4;  
grid-column-start: 1; grid-column-end: 4; grid-row-start: 3; grid-row-end: 4;
```

```
grid-column-start: 3;  
grid-column-end: 4;  
grid-row-start: 1;  
grid-row-end: 4;
```

Line-based placement

Now, these things are long to write, but there are shorthands.

We can write:

`grid-column: <start line> / <end line>`

`grid-row: <start line> / <end line>`

`grid-column: 1; grid-row: 1 (same as default)`

`grid-column: 2 / 3;
grid-row: 2 / 3`

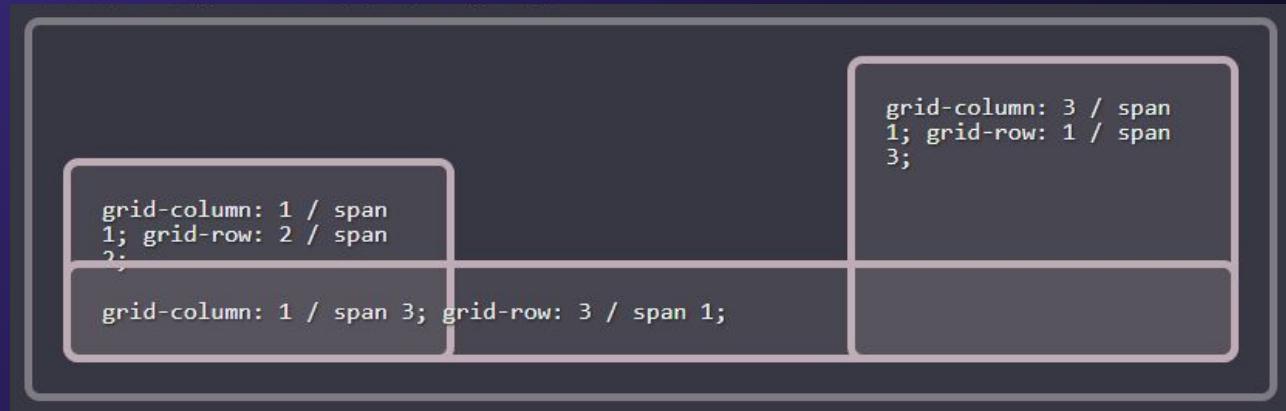
`grid-column: 3 / 4;
grid-row: 3 / 4`

Line-based placement

Instead of writing in what line the cell ends, we can say how many spaces it occupies. For this, use the “span” keyword:

grid-column: <start line> / span <size>

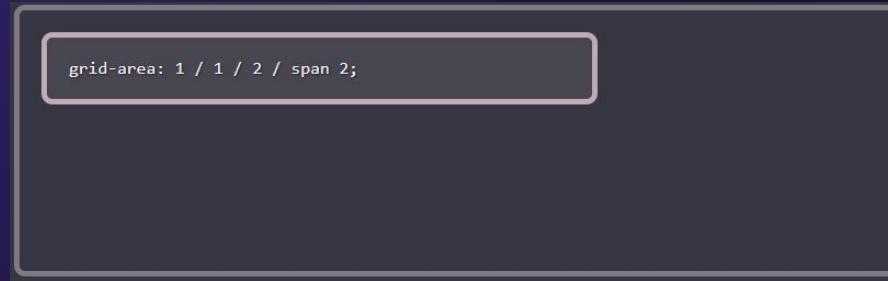
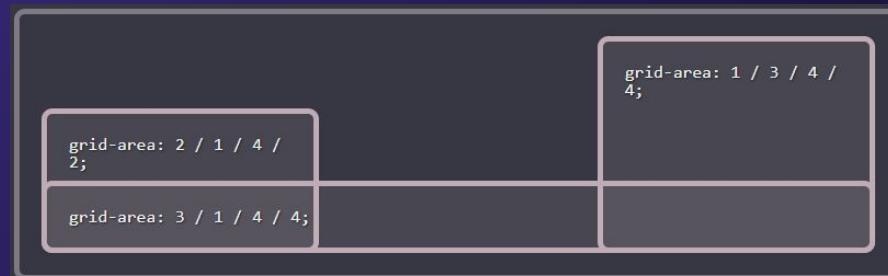
grid-row: <start line> / span <size>



Line-based placement

Want this to be even shorter? Use the grid-area keyword instead:

`grid-area: <grid-row-start> / <grid-column-start> / <grid-row-end> / <grid-column-end>`



Named areas

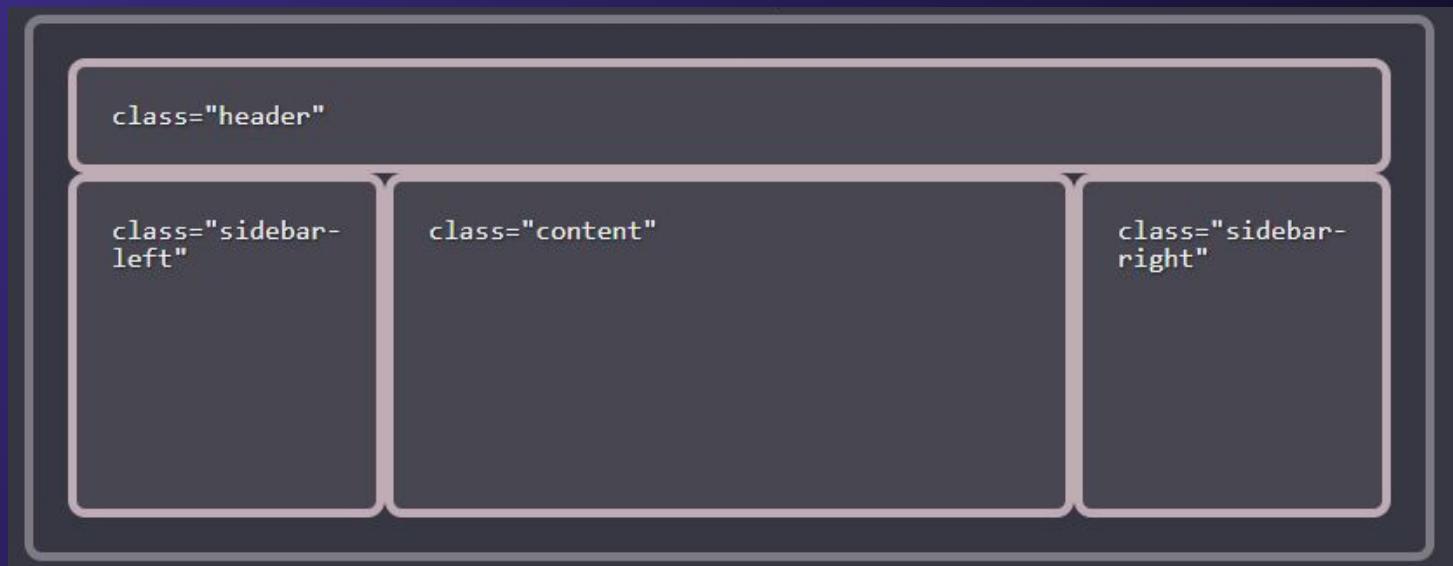
The other cell positioning method allows us to “paint” the areas where the cells will fall. This makes it easier to visualize how the page will look like, without having to calculate numbers.

This method has two steps:

1. Defining grid-area values
2. Using grid-template-areas to “paint” where each grid-area goes.

Named areas

We will create something like this. Our grid will have six columns and four rows.



Named areas

For reference, our HTML looks like this. Note we've given classes to the children:

```
<div class="grid">
    <div class="header">
        <code>class="header"</code>
    </div>
    <div class="sidebar-left">
        <code>class="sidebar-left"</code>
    </div>
    <div class="content">
        <code>class="content"</code>
    </div>
    <div class="sidebar-right">
        <code>class="sidebar-right"</code>
    </div>
</div>
```

Named areas

First, in CSS we give “grid-area” values to each class, like this:

```
.header { grid-area: head }
.sidebar-left { grid-area: left }
.sidebar-right { grid-area: right }
.content { grid-area: cont }
```

The values here don't have to be “head”, “left”, etc.

They can be any string you want: you get to invent them.

Named areas

Next, we tell CSS where each grid-area goes. We use grid-template-areas for this:

```
.gtareas {  
    grid-template-columns: repeat(6, 1fr);  
    grid-template-rows: repeat(4, 1fr);  
    grid-template-areas:  
        'head head head head head head'  
        'left cont cont cont cont right'  
        'left cont cont cont cont right'  
        'left cont cont cont cont right';  
}
```

You'll notice I just attached this to a ".gtareas" class, so I also need to do:

```
<div class="grid gtareas">...</div>
```

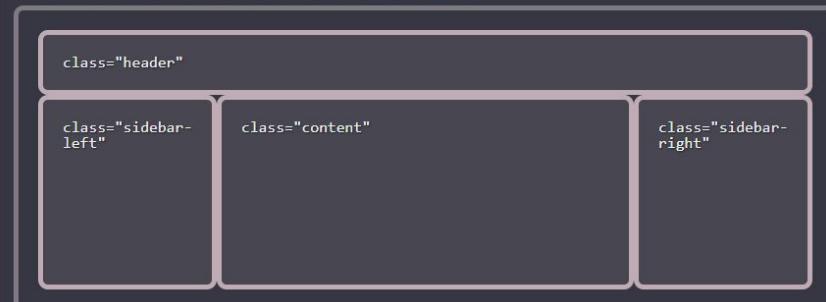
Named areas

Named areas

We can also use `grid-area` in a very cool way: by "painting" our grid. Suppose we want to create a layout with header and two columns. First, we will define names for each area, and attach those to CSS rules. We will do the following:

```
.gtareas .header { grid-area: head }
.gtareas .sidebar-left { grid-area: left }
.gtareas .sidebar-right { grid-area: right }
.gtareas .content { grid-area: cont }
```

Now, all of those names are used to "paint" the layout inside an instruction called `grid-template-areas`. In this parent with class `gtareas`, we're using this technique with 6 columns and 4 rows. Use the developer tools to take a closer look to the CSS styles.



Elements Console Recorder > □ 1 | ⚙ X

```
<pre> ...</pre>
<pre> .gtareas .header { grid-area: head } .gtareas .sidebar-left { grid-area: left } .gtareas .sidebar-right { grid-area: right } .gtareas .content { grid-area: cont } </pre>
<p> ...</p>
<p> ...</p>
<p> Use the developer tools to take a closer look to the CSS styles.</p>
...
<div class="parent grid gtareas"> grid == $0
  <div class="header"> ...</div>
  <div class="sidebar-left"> ...</div>
  <div class="content"> ...</div>
  <div class="sidebar-right"> ...</div>
</div>
```

html body main article div.parent.grid.gtareas

Styles Computed Layout Event Listeners DOM Breakpoints >

Filter element.style {

```
element.style {
```

gtareas {

```
gtareas {
```

grid-template-columns: repeat(6, 1fr);
grid-template-rows: repeat(4, 1fr);
grid-template-areas:
 'head head head head head head'
 'left cont cont cont cont right'
 'left cont cont cont cont right'
 'left cont cont cont cont right';

}

grid {

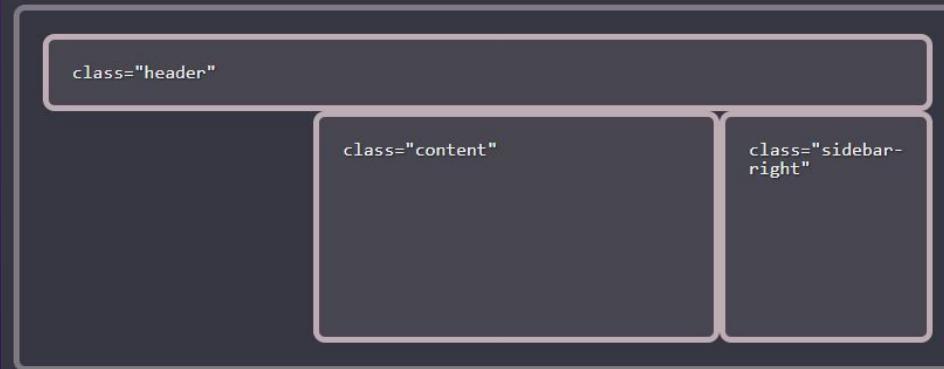
```
grid {
```

display: grid;

Named areas

If you need an empty space, define it with periods (".") inside grid-template-areas:

```
.gtareas-2 {  
    grid-template-columns: repeat(6, 1fr);  
    grid-template-rows: repeat(4, 1fr);  
    grid-template-areas:  
        'head head head head head head'  
        '. : cont cont cont right'  
        '. : cont cont cont right'  
        '. : cont cont cont right';  
}
```



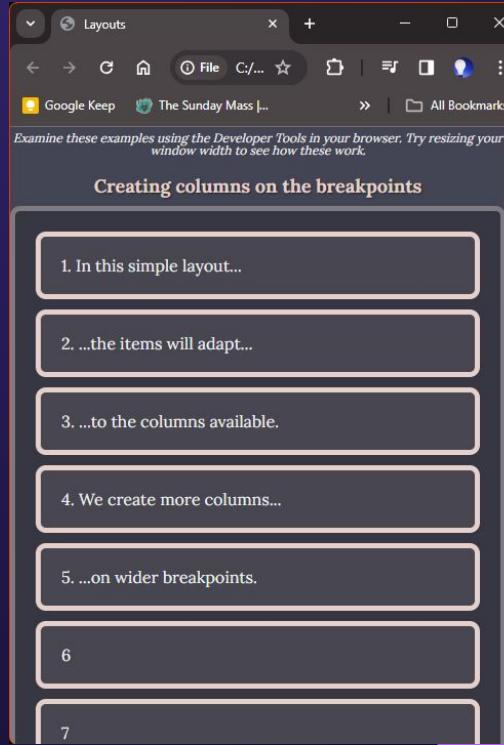
Responsive grids

There are many techniques to create responsive grids. We will study three.

Responsive grids

Creating columns on the breakpoints:

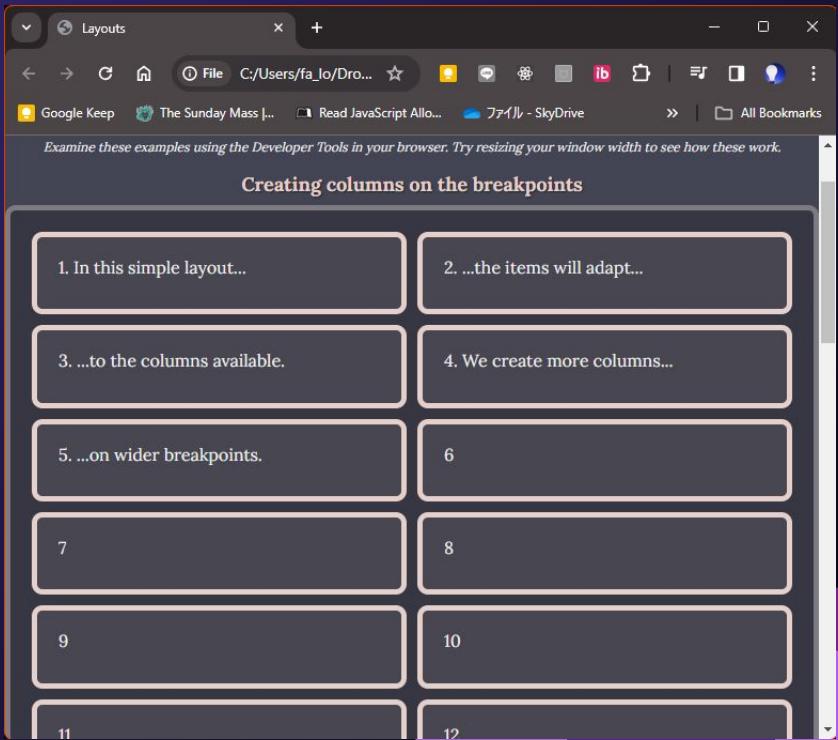
In this technique, we start with no columns and create them as the page grows.



Responsive grids

Creating columns on the breakpoints:

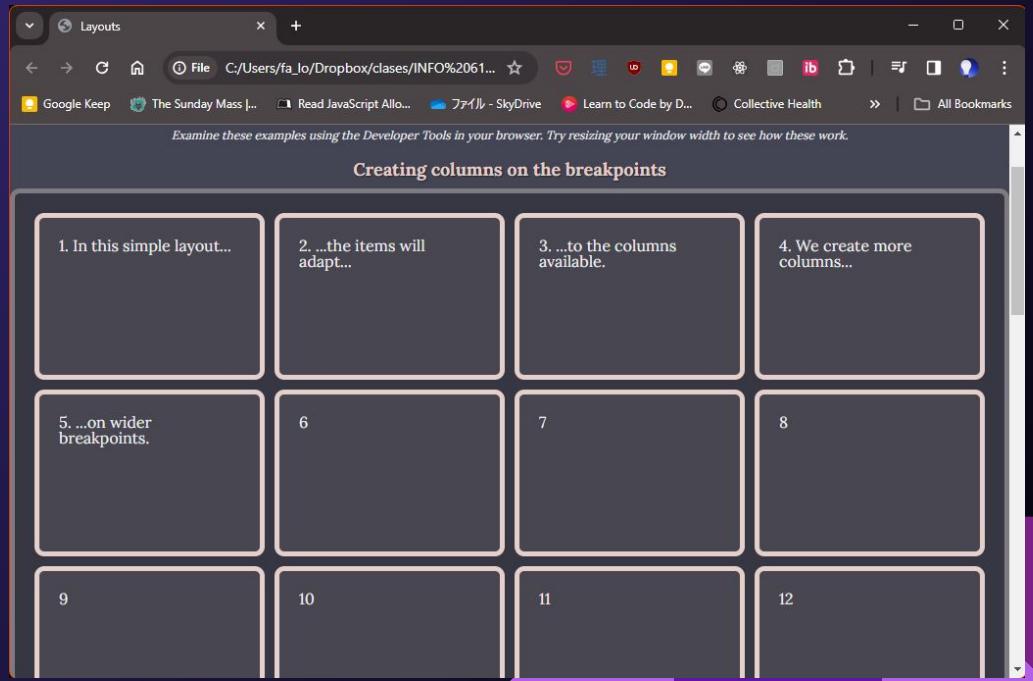
In this technique, we start with no columns and create them as the page grows.



Responsive grids

Creating columns on the breakpoints:

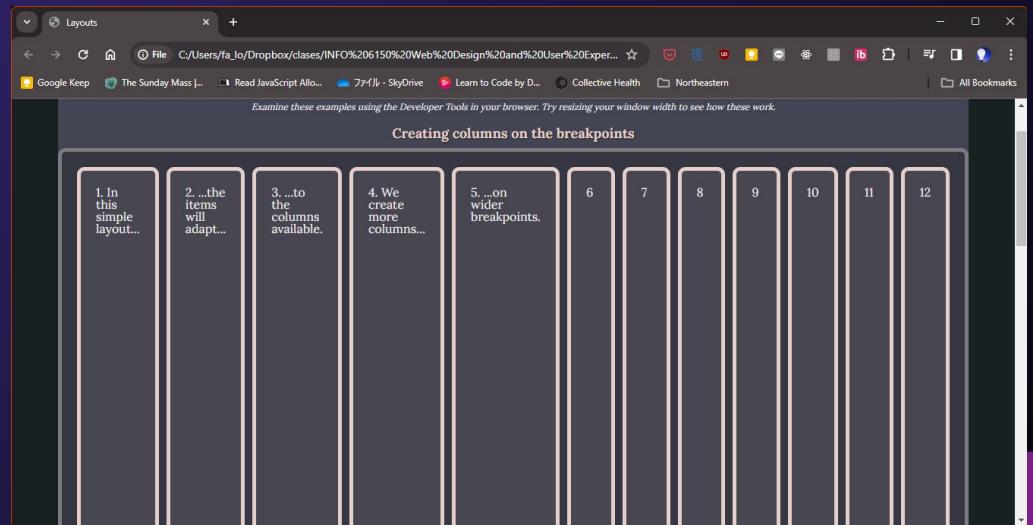
In this technique, we start with no columns and create them as the page grows.



Responsive grids

Creating columns on the breakpoints:

In this technique, we start with no columns and create them as the page grows.



Responsive grids

```
/* Creating columns on breakpoints */
.layout-1 {
    gap: 1rem;
    /* nothing else; this way we get one column by default */
}

@media (min-width: 600px) {
    .layout-1 {
        grid-template-columns: 1fr 1fr;
    }
}

@media (min-width: 800px) {
    .layout-1 {
        grid-template-columns: repeat(4, 1fr);
    }
}

@media (min-width: 1200px) {
    .layout-1 {
        grid-template-columns: repeat(12, 1fr);
    }
}
```

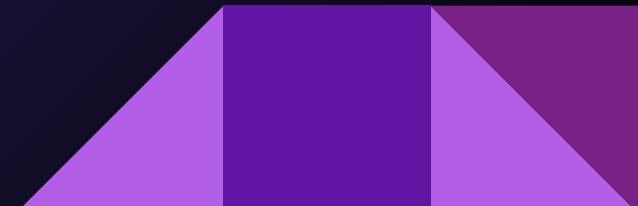
Responsive grids

Always 12 columns, change where cells start on each breakpoint:

This is very close to how grid design systems work.

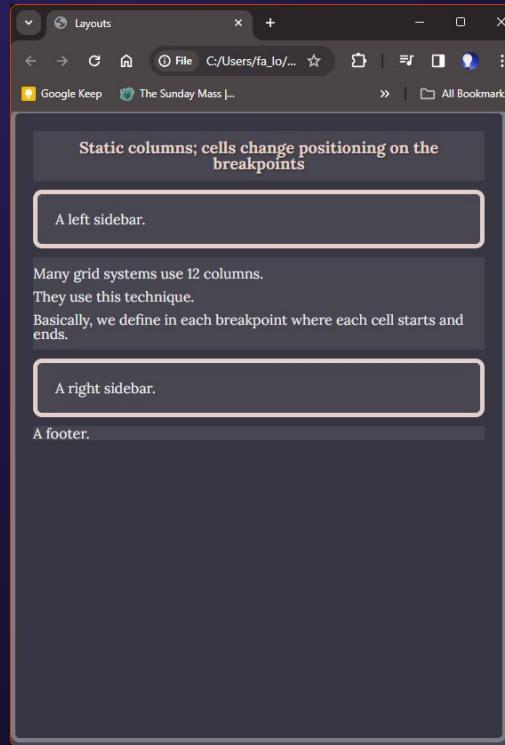
There is always the same number of columns in the layout, and what keeps changing is the cells.

This can make things much more flexible.



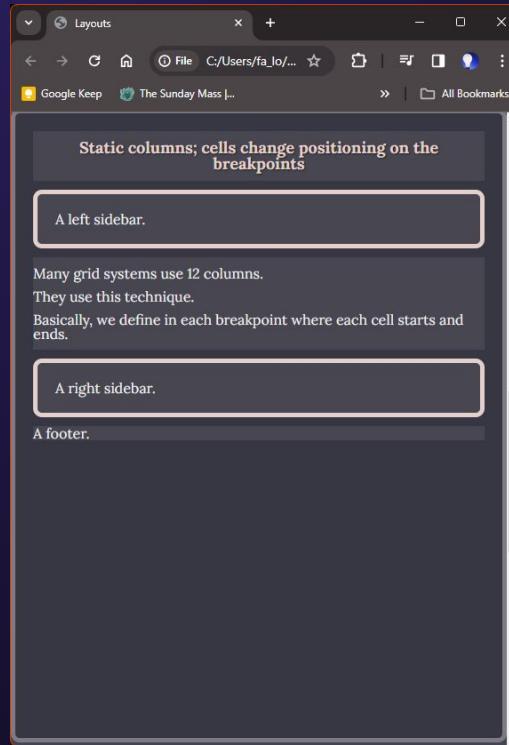
Responsive grids

```
.layout-2 {  
    grid-template-columns: repeat(12, 1fr);  
    gap: 1rem;  
  
    /* Make the heights flexible. */  
    /* max-content means: the maximum size of  
the content. */  
    /* Note: we can only do this if we know how  
many rows we need. */  
    grid-template-rows: repeat(5, max-content);  
}
```



Responsive grids

```
.layout-2 {  
    grid-template-columns: repeat(12, 1fr);  
    gap: 1rem;  
  
    /* Make the heights flexible. */  
    /* max-content means: the maximum size of  
the content. */  
    /* Note: we can only do this if we know how  
many rows we need. */  
    grid-template-rows: repeat(5, max-content);  
}  
  
/* initially, all elements use the 12 columns.  
This is the same as having a single column.  
*/  
  
.layout-2 > * {  
    grid-column: 1 / span 12;  
}
```

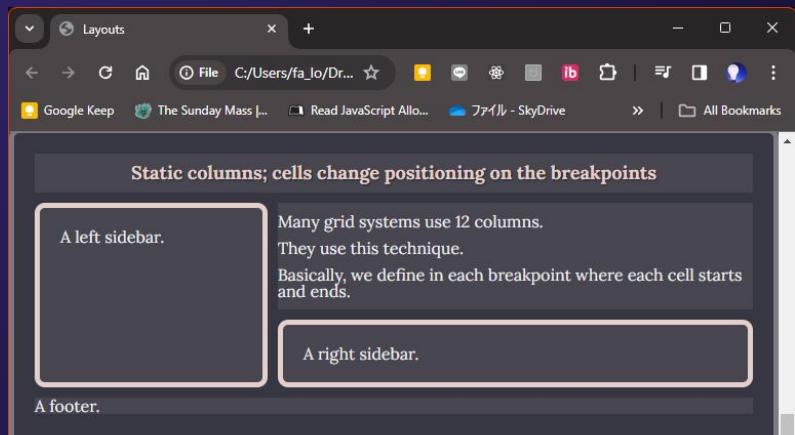


Responsive grids

```
/* two-column layout here */  
@media (min-width: 600px) {  
    /* header uses all the width already */  
    .layout-2 header {  
        grid-column: 1 / span 12;  
        grid-row: 1 / span 1;  
    }  
}
```

```
.layout-2 .left-sidebar {  
    grid-column: 1 / span 4;  
    grid-row: 2 / span 2;  
}  
}
```

```
.layout-2 main {  
    grid-column: 5 / span 8;  
    grid-row: 2 / span 1;  
}  
...  
.
```



```
.layout-2 .right-sidebar {  
    grid-column: 5 / span 8;  
    grid-row: 3 / span 1;  
}  
}
```

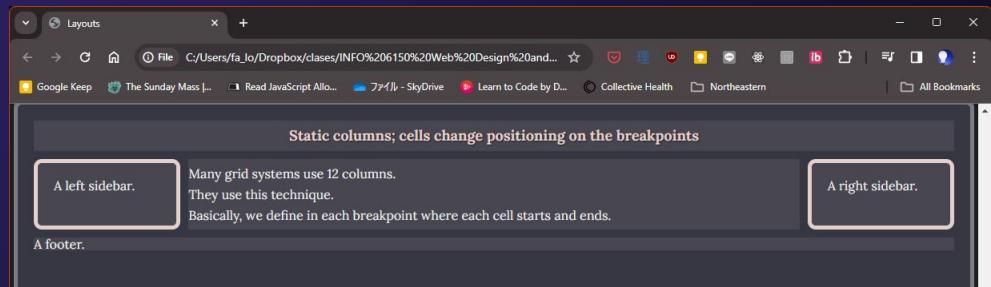
```
.layout-2 footer {  
    grid-column: 1 / span 12;  
    grid-row: 4 / span 1;  
}  
}
```

Responsive grids

```
/* three-column layout here */  
  
@media (min-width: 1200px) {  
    .layout-2 .left-sidebar {  
        grid-column: 1 / span 2;  
    }  
}
```

```
.layout-2 main {  
    grid-column: 3 / span 8;  
    grid-row: 2 / span 2;  
}  
}
```

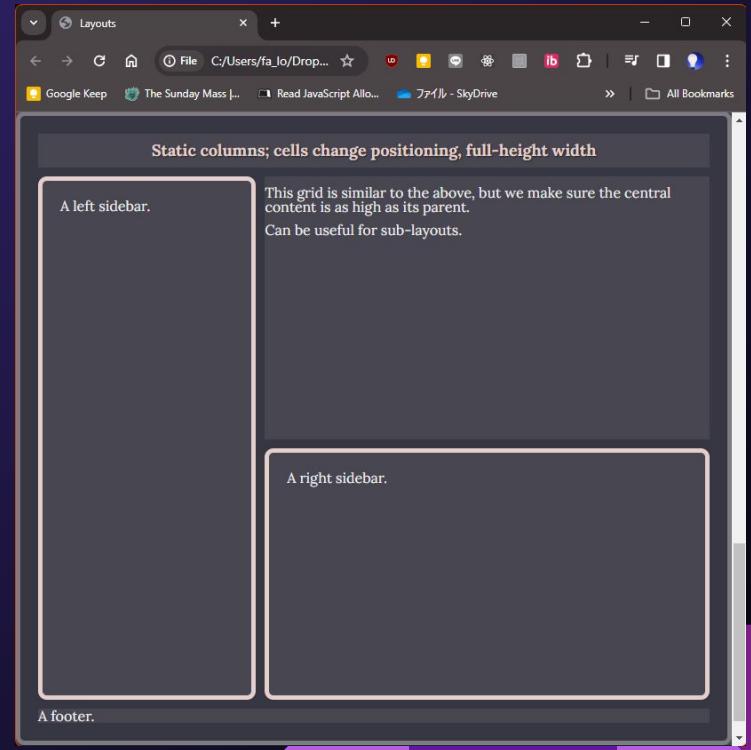
```
.layout-2 .right-sidebar {  
    grid-column: 11 / span 2;  
    grid-row: 2 / span 2;  
}  
}
```



Responsive grids

The layout row heights can be changed to `auto` to make them fit the parent's size.

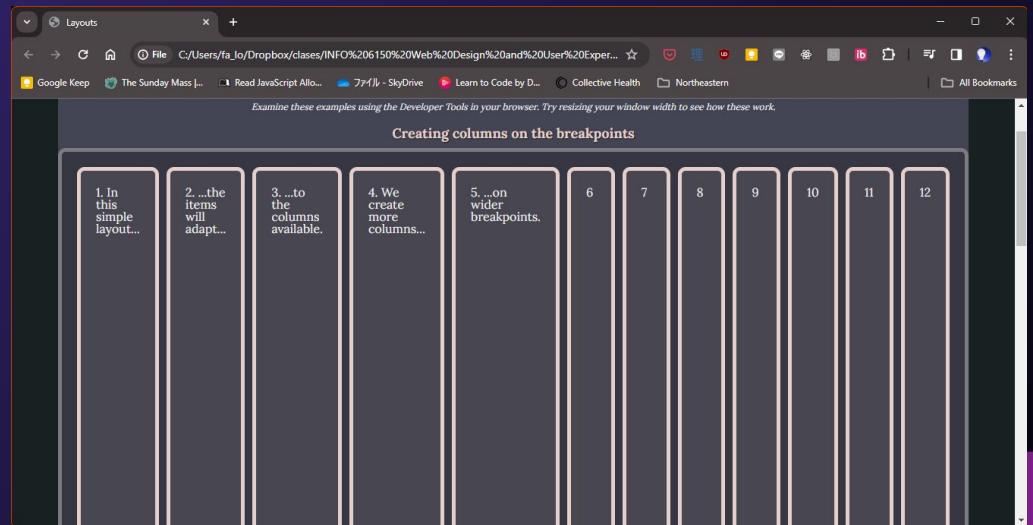
```
/* two-column layout here */
@media (min-width: 600px) {
    .layout-3 {
        grid-template-rows: max-content auto
        auto max-content;
    }
}
```



Responsive grids

Creating columns on the breakpoints:

In this technique, we start with no columns and create them as the page grows.

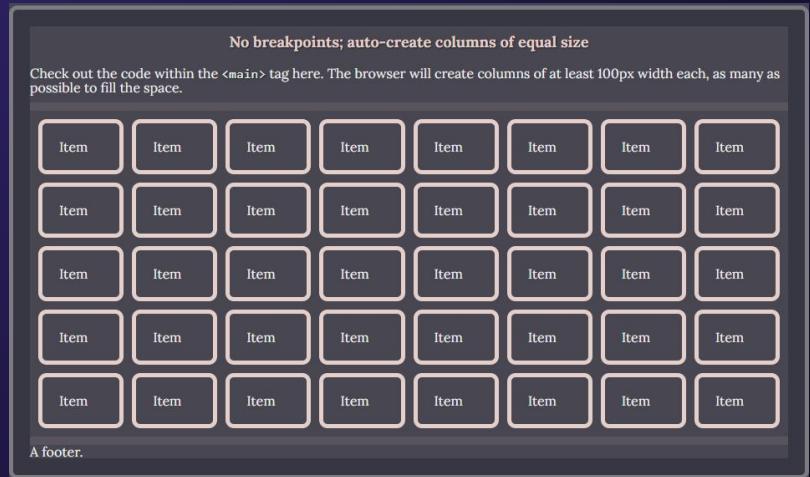


Responsive grids

The third technique is useful for when you have many items with a regular size, for example, product galleries.

Instead of relying on breakpoints, we tell the browser a **minimum size** for our columns and let it fill the space with as many columns as it can:

```
.layout-4 main {  
    display: grid;  
    gap: 1rem;  
    /* auto-fit means: as many as will fit in the container */  
    grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));  
}
```

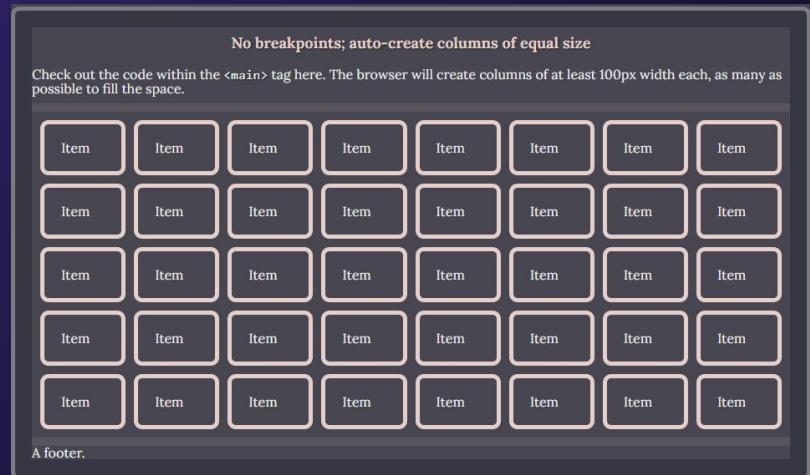


Responsive grids

The third technique is useful for when you have many items with a regular size, for example, product galleries.

Instead of relying on a fixed number of columns or a minimum size for each item, you can use auto-fit: “as many as you can fit in the space”

```
.layout-4 main {  
  display: grid;  
  gap: 1rem;  
  /* auto-fit means: as many as will fit in the container */  
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));  
}
```



Responsive grids

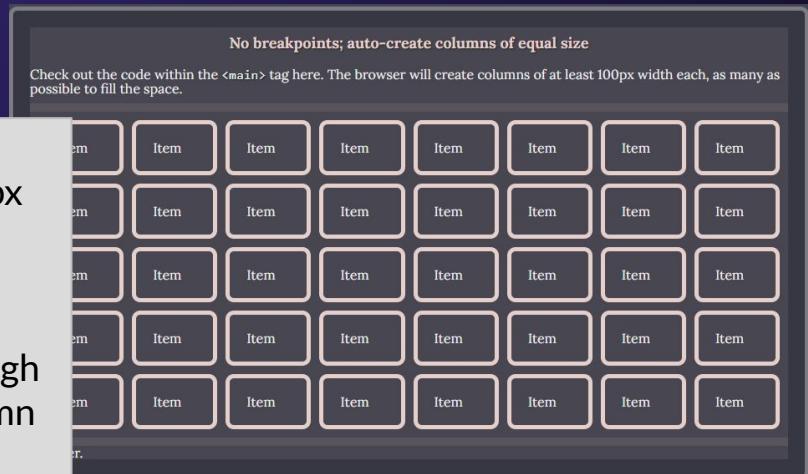
The third technique is using items with a regular size, f...

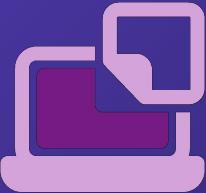
Instead of relying on break
minimum size for our colu...
with as many columns as

```
.layout-4 main {  
    display: grid;  
    gap: 1rem;  
    /* auto-fit means: as many as will fit in the container */  
    grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));  
}
```

minmax: “at least 100px
and at most 1fr”

At most 1fr means:
as soon as there is enough
space for another column
(1fr), insert it





Practice: Grid

Created by HideMaru
from Noun Project

Go back to the files you downloaded ([zip file](#)).

Now, open the “grid.html” file. It will give you instructions for two exercises. Do them locally, in your computer.

First, try to solve them by yourself (don’t ask help to the AI).

The screenshot shows a web browser window titled "CSS exercises 2". The address bar indicates the file is located at "C:/Users/fa_lo/Projects/web-sa...". The main content area has a green header bar with the title "CSS exercises 2: Grid". Below this, there is a section titled "Instructions" containing three bullet points: "Copy the HTML code in a new file of your own.", "Link that HTML to a CSS file.", and "Add styles so that your HTML looks like the screenshot.". A large yellow box highlights the "HTML code" section, which contains the following code:

```
<div class="container">
  <ul>
    <!-- Repeat this line 19 times: -->
```

At the bottom right of the page, there is a note: "This page has no copyright."

CSS frameworks

In modern web development, it's increasingly popular to not create (most or any) CSS from scratch, and instead use CSS frameworks.

CSS frameworks are software packages that can be included into your code. They already provide solutions to most situations. Note that you should be familiar with how CSS works to take full advantage of them.

Two popular frameworks are Bootstrap and Tailwind.

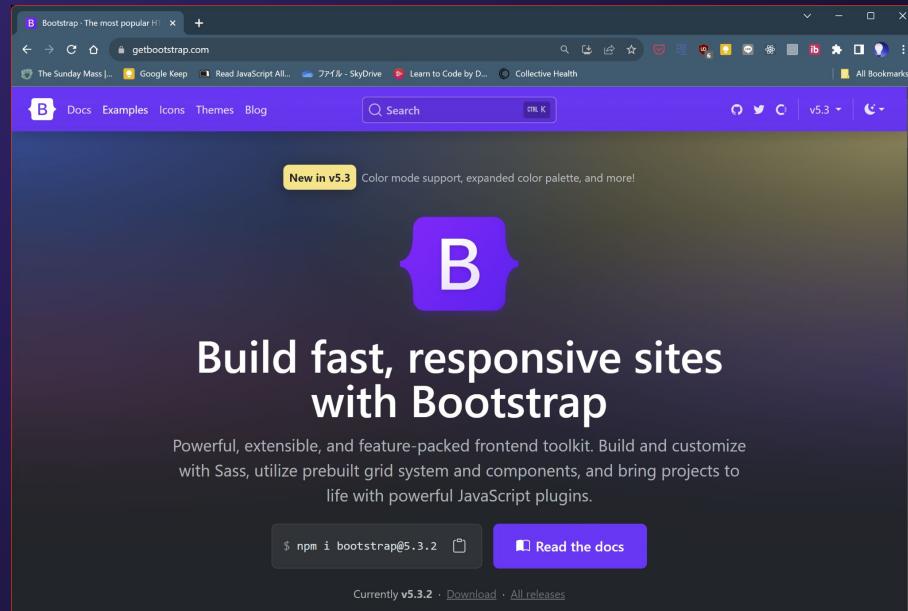
Bootstrap

Bootstrap uses Sass. Sass is CSS with more features, basically.

You can play with the example [here](#) and read the docs [here](#).

The easiest way to get started is [here](#): add a <script> and a <style> tag pointing to Bootstrap's files. You will automatically get some useful base styles.

Bootstrap also contains a full library of JS components - you're welcome to explore it!

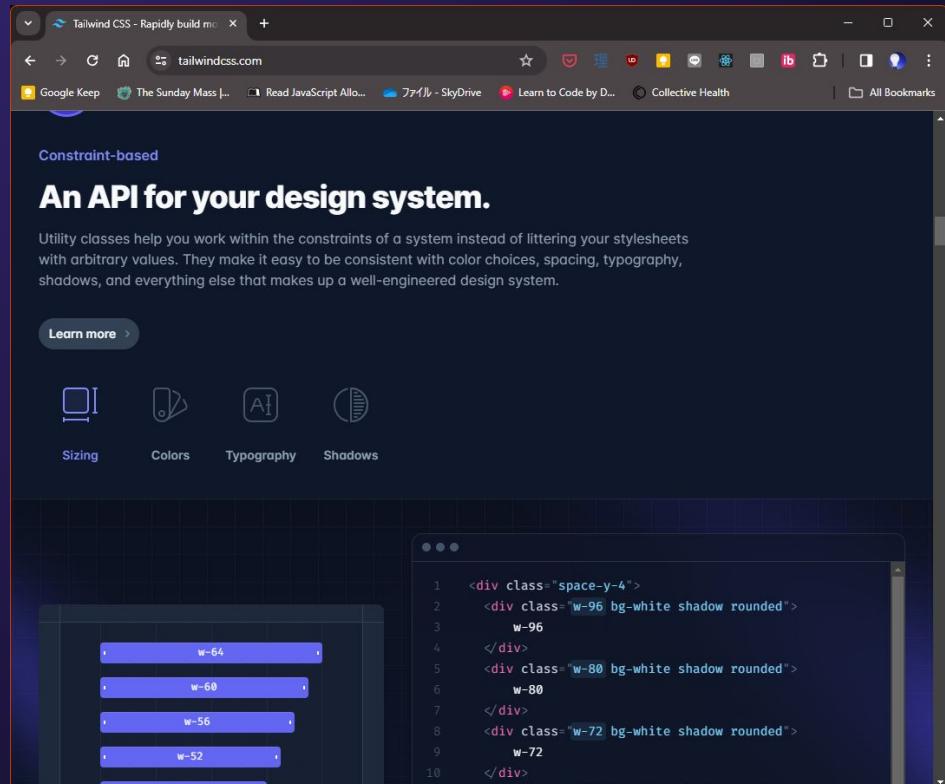


Tailwind

In Tailwind you get a collection of pre-styled classes, which you can add to your HTML tags.

The website is [here](#); documentation is [here](#). For a quick start, check their [playground](#).

Note: the installation documentation mentions npm and npx; these are covered in a later section of the course. I recommend starting with the “Play CDN” option for now.



In the next lesson...

Now you have the tools to do your own CSS styles and layouts.

Next time, we'll learn the essentials of the JavaScript language.