Summary Notes -> webpage layouts in CSS

- **Part #1 - Apply basic CSS positioning techniques**
- **Part #2 - Create two-dimensional layouts with CSS Grid (creating grids with CSS Grid)**
- **Part #3 - Implement one-dimensional layouts with Flexbox (Flexbox)**
- **Part #4 - Explore legacy layouts (understanding legacy layouts)**

Part #1 - Apply basic CSS positioning techniques

- understanding elements on a webpage as boxes
- these are marked up / defined in HTML
- formatted in CSS (two documents for the same webpage)
- you can change the borders of them and their widths with different units in CSS
- you can also have multiple elements next to each other on a webpage and control the distance between them when they stack
- you can also control how they stack for different screen sizes with media queries

Part #2 - Create two-dimensional layouts with CSS Grid (creating grids with CSS Grid)

- we have pieces of content defined in HTML and formatted in CSS
- they are surrounded by content wrappers in HTML -> and in CSS shown as a box around that specific element
- this chapter is how to lay out those boxes by defining a grid on the page in CSS
- i.e if you have little boxes of content -> you can stack them and arrange them in a CSS grid
- you can change the grid depending on the screen size, and how many cells each of the little boxes / containers take up in the grid
- a CSS grid is 2D (compared to flex box which is in the next chapter and is 1D stacking of the little boxes)

Part #3 - Implement one-dimensional layouts with Flexbox (Flexbox)

- this is containing the little boxes in a flexbox
- this is a 1D vector (a big box), aligned in the x or the y
- you define it in HTML by wrapping the content you want to go in the big box -> and then id'ing each of the elements inside it (these are what form the little boxes)
- then because it's in a vector, you have two directions you can align the little boxes along (aligning and justifying the content)
- you can also wrap the little boxes to fit on the next line or force them all onto one line, and reorder the little boxes (by formatting the big box in CSS)

Part #4 - Explore legacy layouts (understanding legacy layouts)

- for legacy browsers -> transforming the position of an element on a webpage without flexbox or the CSS grid (and instead looking at different coordinate systems for doing it)
- relative to - where the element would normally be, its absolute position on the webapage or the nearest other element on the webpage
- having text move around the image (rather than being blocks stacked on each other, floating the text around the image)
- legacy layouts: the page isn't little boxes on a grid, it's little boxes on a three dimensional axis
- you can bring elements to the front of the page (the z axis is out of the page)

Part #1 - Apply basic CSS positioning techniques

> understanding elements on a webpage as boxes

> these are marked up / defined in HTML

> formatted in CSS (two documents for the same webpage)

> you can change the borders of them and their widths with different units in CSS

> you can also have multiple elements next to each other on a webpage and control the distance between them when they stack

> you can also control how they stack for different screen sizes with media queries

Contents

Part #2 - Create two-dimensional layouts with CSS Grid (creating grids with CSS Grid)

> we have pieces of content defined in HTML and formatted in CSS

> they are surrounded by content wrappers in HTML -> and in CSS shown as a box around that specific element

> this chapter is how to lay out those boxes by defining a grid on the page in CSS

> i.e if you have little boxes of content -> you can stack them and arrange them in a CSS grid

> you can change the grid depending on the screen size, and how many cells each of the little boxes / containers take up in the grid

> a CSS grid is 2D (compared to flex box which is in the next chapter and is 1D stacking of the little boxes)

1. Introduction to CSS Grid (P7-8)
- -> how the different elements which are contained in containers defined in HTML can be stacked in different ways (like boxes) on a grid in CSS

2. Set up a basic grid (P8)
- -> how to define a CSS grid
- -> if we have a few elements -> e.g in a list defined in HTML and want to stack them in a different way
- -> defining a CSS grid for a certain set of HTML elements

3. Set columns and rows in shorthand (P8-9)
- -> you are telling it which elements to stack in CSS
- -> and then what their widths are
- -> this is about defining and controlling the CSS grid

4. Define grid element height and width (P9-11)
- -> so, elements in HTML are marked up and these are surrounded by boxes in CSS
- -> if you have a collection of those boxes, you can stack them in a CSS grid
- -> this video is how to make one of those elements (the little boxes) take up two spaces on the larger box, for instance

5. Create a grid template area (P11-13)
- -> this is a similar idea to 4.
- -> instead, we are using tags to describe which little box goes in which part of the larger box (the CSS grid)

6. Set columns depending on screen size (P13-14)
- -> this is shrinking the screen size -> getting the elements on the CSS grid to stack, rather than be next to each other e.g -> below a certain screen size
- -> **this is for screen resizing with multiple little boxes (rather than just one) -> for one, see #1 6. (media queries, vs the CSS grid which is a 2D grid for stacking the little boxes containing the content defined in HTML)**

Part #3 - Implement one-dimensional layouts with Flexbox (Flexbox)

> this is containing the little boxes in a flexbox

> this is a 1D vector (a big box), aligned in the x or the y

> you define it in HTML by wrapping the content you want to go in the big box -> and then id'ing each of the elements inside it (these are what form the little boxes)

> then because it's in a vector, you have two directions you can align the little boxes along (aligning and justifying the content)

> you can also wrap the little boxes to fit on the next line or force them all onto one line, and reorder the little boxes (by formatting the big box in CSS)

1. What is Flexbox? (P15)

- -> we have little boxes which contain the content
  - those are defined in HTML in <> tags and formatted in CSS {}'s
- -> there are different ways of stacking the little boxes
- -> a new way (for more moden not legacy browsers) is flexbox
- **-> instead of the CSS grid (2D) flexbox aligns the little boxes on a vector (either in the x or the y)**
- -> so we have a big box which contains all of the little boxes (a flex container) - in the x or the y (this is called the flex direction), and the little boxes inside it are the flex items

2. Set flex direction and wrapping (P15-20)

- -> **the flexbox aligns the little boxes in the x or the y**
  - **-> you have a main axis and a cross axis (depending on which one it is)**
  - **-> the little boxes align in there**
  - -> this video is how to set if the big box aligns in the x or the y
- -> the second half of it is how to wrap elements (you either force all of the little boxes to fit onto one line, or you can wrap them onto the next)
  - -> either their size is determined by forcing them all onto the same line
  - -> or by setting their size and letting the boxes with extra space fit onto the next one

3. Align items and justify content (P20-24)

- -> so we have a big box, which is either in the x or the y and contains little boxes (it's like a vector, a 1D vector - rather than a 2D CSS grid to align the little boxes on)
- -> we have a main and a cross axis
- **-> we align the little boxes along the cross axis and justify them along the main axis <- these are the terms we use in CSS to move the little boxes**
- -> you can also target one little box

4. Align multiple lines of content (P25-28)

- -> if you have little boxes in a big box (a vectorial box) and haven't forced the little boxes all onto the next line of the page, then they will wrap onto the next row
- -> this is video is how to align them (align is for the cross axis of the flex box vector the little boxes are in)

5. Adjust element dimensions (P28-31)

- -> we have HTML content marked up (each element is marked and then there is another wrapper around all of the elements)
- -> and then it's formatted in CSS so that the elements are surrounded by the little boxes -> and **the big box that surrounds all of them (a vector in the x or the y) is called the flexbox**
- -> this video is how to
  - change the sizes of the little boxes relative to each other in the big box (**flex-basis**)
  - force the little boxes to expand to fill the big boxes (**flex grow**)
  - control the shrink rate of those boxes relative to each other when the webpage shrinks (**flex shrink**)

6. Reorder elements (P31-32)

- -> changing the order of the little boxes in the big box
  - **wrap-reverse, order**
  - -> id'ing the little boxes in HTML <>, then formatting them in CSS # {} (for id'd elements, one id per element) or .{} (for classes) to change their orders in the big box

Part #4 - Explore legacy layouts (understanding legacy layouts)

> for legacy browsers -> transforming the position of an element on a webpage without flexbox or the CSS grid (and instead looking at different coordinate systems for doing it)

> relative to - where the element would normally be, its absolute position on the webapage or the nearest other element on the webpage

> having text move around the image (rather than being blocks stacked on each other, floating the text around the image)

-> legacy layouts: the page isn't little boxes on a grid, it's little boxes on a three dimensional axis

1. Manually adjust elements' positions (P33-34)

- -> this is the **legacy** way of rearranging the little boxes on the page
- -> not on a CSS grid (2D) or a flexbox (1D)
- **-> transforming the position of an element on a webpage without a grid or flexbox**
  - **-> relative to where the element would normally be in the page**
  - **-> or relative to the absolute position of the webpage (think of the entire webpage like a grid system) - not the position of the webpage --- the position of the nearest positioned element on the webpage**
  - **-> or fixed in comparison to the webpage (scrolling through the webpage and the image being fixed on the screen as you scroll)**

2. Float elements (P34-35)

- **-> having text move around images on webpages**
- **-> the images are elements <img src ="..."> etc in HTML -> having text float around those images on the webpage**
  - -> the image is a little box
  - -> the text is a little box
  - -> the default is for them to stack on top of each other -> not for the text to float around the image

3. Clear floated elements (P35-36)

- -> if we have an image and two paragraphs
- -> they are all tagged in HTML -> the two paragraphs and the image
- -> in CSS, each of the elements has a container (box) around it -> and the default is for these boxes to stack we are formatting these boxes / containers in CSS
- -> if we float the paragraphs around the text, the text then flows around the image
- -> we can make one of the paragraphs do that and the other one go under the image -> by clearing the float
- **-> you want the paragraph to flow around the image (to float it) and if you want it to stop half way through and fit under the image again (e.g for the second paragraph) -> you can clear the float -> this is what this video is <- "cleared float"**

> you can bring elements to the front of the page (the z axis is out of the page)

4. Stack elements in an order (P37)

- **-> think about the webpage (in a legacy system) as having an x, y and z axis -> rather than the modern CSS grid or 1D flexbox**
- -> **in HTML, we have elements which are marked up with a div -> and in CSS, they have different containers / boxes around them**
- -> these stack to create parts of the webpage
- -> in the non-static case
  - **if we move those boxes in CSS to overlap with each other on the webpage (the default is that they stack as they are block elements) -> we can change the other they stack in on the z axis (e.g bring image to front in powerpoint)**
- -> this is done in CSS
- -> **since we're stacking on the z axis (it's called z-index) -> the positive z is out of the page (the bigger the z index, the closer that little box is to the top of the pile -> and they are set separately by id'ing them in HTML them formatting them in CSS, and we use an ID because we're targeting that one element)**

5. Discover third-party solutions (P37-38)

- **-> there are third party solutions to arranging elements on pages in CSS (bootstrap is one of these)**
- -> the CSS grid is 2D and flex box is 1D -> ways of arranging the little boxes on the page
  - -> but using the native version of CSS because it's being updated all the time

Notes from doing the quizzes (P39-41)

Part #1 - Apply basic CSS positioning techniques

entire course is CSS

1. Discover elements as boxes
2. Add custom borders to elements
3. Cushion elements with padding
4. Add breathing room with margins
5. Control an element's width and height
6. Set media queries for different devices

the course is four of these

Quiz: Understand elements as boxes

1. Discover elements as boxes

HTML = content

```
1  <h1>This is a heading.</h1>
2  <h3>This is another heading.</h3>
3  <p id="one">This is a paragraph.</p>
4  <ul>
5      <li>This</li>
6      <li>Is</li>
7      <li>A</li>
8      <li>List.</li>
9  </ul>
10 <p id="two">This is another paragraph with a
   <a href="#">link</a> in it.</p>
```

THIS IS A HEADING
This is another heading
This is a paragraph

- This
- is
- a
- list.

This is another paragraph with a link in it.

"CODEPEN" ~ overleaf but for HTML / CSS

CSS

```
1  body {
2      color: #fff;
3      font-family: Futura;
4      padding: 20px;
5  }
6
7  h1 {
8      background-color: #A4036F;
9  }
10
11 h3 {
12     background-color: #048BA8;
13 }
14
15 #one {
16     background-color: #16DB93;
17 }
18
19 ul {
20     background-color: #EFEA5A;
21 }
22
23 #two {
24     background-color: #FE5E41;
25 }
26
27 a {
28     background-color: #E6F6C2;
29     color: #000;
30 }
```

each el. is a box

box takes up whole page

Block-level

This is a heading.
This is another heading.
This is a paragraph.
- This
- Is
- A
- List.

This is another paragraph with link in it.

Inline

box takes up not whole page

DIY CONTAINERS

Containers
Wrappers

DIV

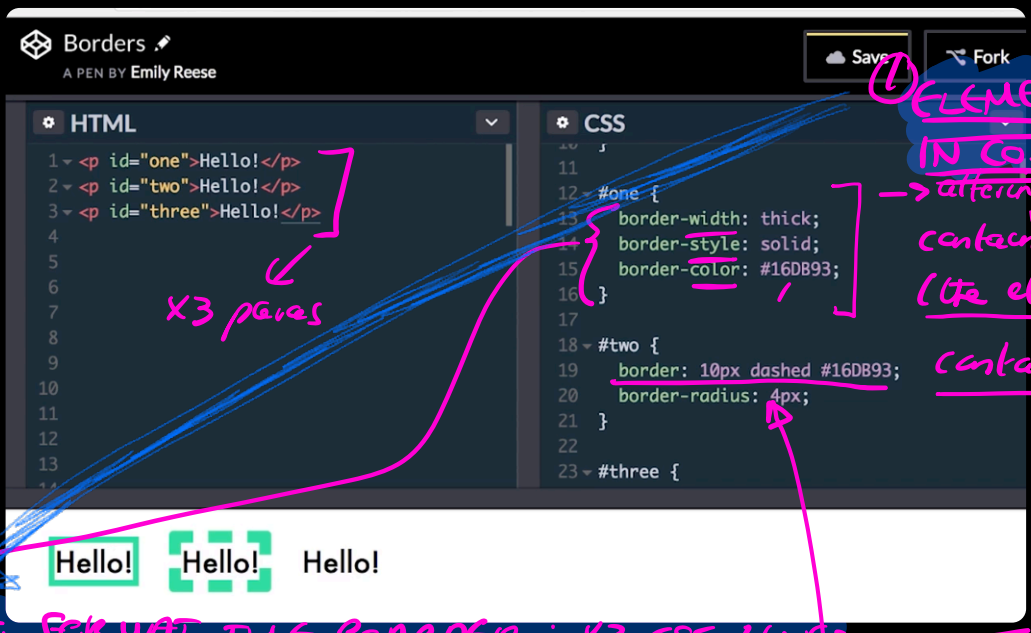
Paragraph

Paragraph

divides up page into sections of different elements (in boxes)

2. Add custom borders to elements

→ adding borders around the boxes which sep. el's appt.



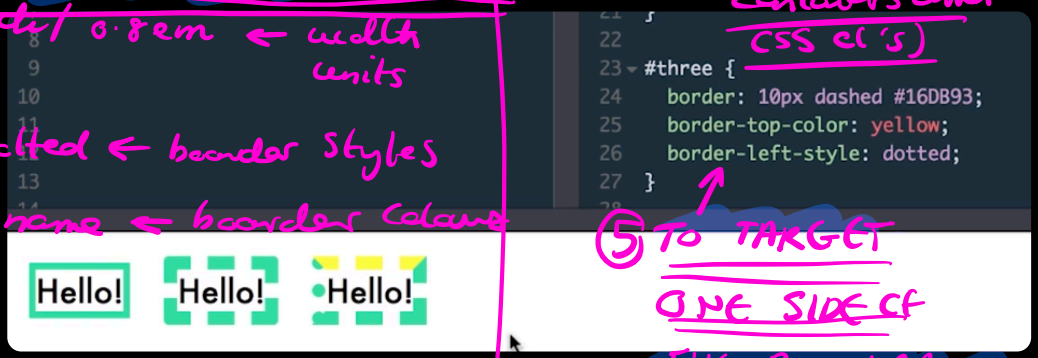
① ELEMENTS ARE CONTAINED IN CONTAINERS
→ affecting the boarder of the container and. para (the el has it's own container (box)

X3 paras

formatting it in CSS

② TO FORMAT THE BORDER : X3 CSS LINES
→ thin / medium / thick / 0.8em ← width units
→ solid / dashed / dotted ← boarder styles
→ hex / rgb / colour name ← boarder colour

(the boarder colours and CSS el's)



⑤ TO TARGET ONE SIDE OF THE BOARDER IN CSS

③ TO FORMAT BEARDERS IN ONE CSS LINE
Boarder: width style colour

④ TO ROUND BORDER CORNERS: USE BORDER-RADIUS: 4px; IN CSS

3. Cushion elements with padding

Padding
A PEN BY Emily Reese

Save | Fork | Settings | Change View | User Avatar

Desserts are good! ← if padding = 0

Cupcake ipsum dolor sit amet caramels marshmallow powder chocolate bar. Jujubes halvah chocolate sweet roll croissant muffin muffin. Apple pie jelly beans caramels apple pie pudding sugar plum candy icing. Soufflé marshmallow icing jelly brownie donut icing muffin halvah. Bear claw powder chocolate topping chupa chups. Cotton candy pie halvah. Gummies cake fruitcake cotton candy candy pudding cupcake brownie. Chupa chups danish brownie gummi bears dragée.

space bet. el + border = padding

```
HTML
<h1>Desserts are good!</h1>

<p>Cupcake ipsum dolor sit amet caramels marshmallow powder chocolate bar. Jujubes halvah chocolate sweet roll croissant muffin muffin. Apple pie jelly beans caramels apple pie pudding sugar plum candy icing. Soufflé marshmallow icing jelly brownie donut icing muffin halvah. Bear claw powder chocolate topping chupa chups. Cotton candy pie halvah. Gummies cake fruitcake cotton candy candy pudding cupcake brownie. Chupa chups danish brownie gummi bears dragée.</p>

CSS
  color: #B33C86;
  border: 10px solid #B33C86;
  padding: 20px 5px;
}

p {
  color: #190E4F;
  border: 5px solid #190E4F;
  padding: 10px;
}
```

top and bottom padding
left, right padding

→ %, px = [T's

VARIABLE PADDING

[t r b l top right bottom left

← x4 no's

padding: 100px ← same padding everywhere (1 no)

padding: 100px 10px ← top and bottom, then left and right (2 no's)

4. Add breathing room with margins

→ margins are spaces bet. el's
→ the space bet. different borders

Margins
A PEN BY Emily Reese

Save | Fork | Settings | Change View

Best practices; greenwashing parse collaborative cities revolutionary think tank social impact agile mobilize. Relief replicable citizen-centered; the resistance inspire.

Mobilize strategize academic thought leadership collaborative consumption social return on investment shine. Parse problem-solvers changemaker, systems thinking empathetic society.

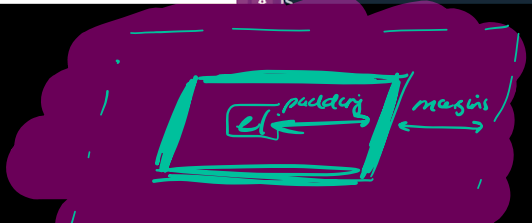
```
HTML
<p id="one">Best practices; greenwashing parse collaborative cities revolutionary think tank social impact agile mobilize. Relief replicable citizen-centered; the resistance inspire.</p>

<p id="two">
  Mobilize strategize academic thought leadership collaborative consumption social return on investment shine. Parse problem-solvers changemaker, systems thinking empathetic society.</p>

CSS
p {
  font-family: 'Minion Pro';
  background-color: black;
  color: white;
  padding: 10px;
}

#one {
  margin: 20px;
}
```

```
#one {
  margin: 20px;
}
```



→ margins are for space around els (outside border) → padding = space bet. el. content + its own border walls

Margins
A PEN BY Emily Reese

Save | Fork | Settings | Change View

Best practices; greenwashing parse collaborative cities revolutionary think tank social impact agile mobilize. Relief replicable citizen-centered; the resistance inspire.

Mobilize strategize academic thought leadership collaborative consumption social return on investment shine. Parse problem-solvers changemaker, systems thinking empathetic society.



HTML

```
<p id="one">Best practices; greenwashing parse collaborative cities revolutionary think tank social impact agile mobilize. Relief replicable citizen-centered; the resistance inspire.</p>

<p id="two">
  Mobilize strategize academic thought leadership collaborative consumption social return on investment shine. Parse problem-solvers changemaker, systems thinking empathetic society.</p>
```

CSS

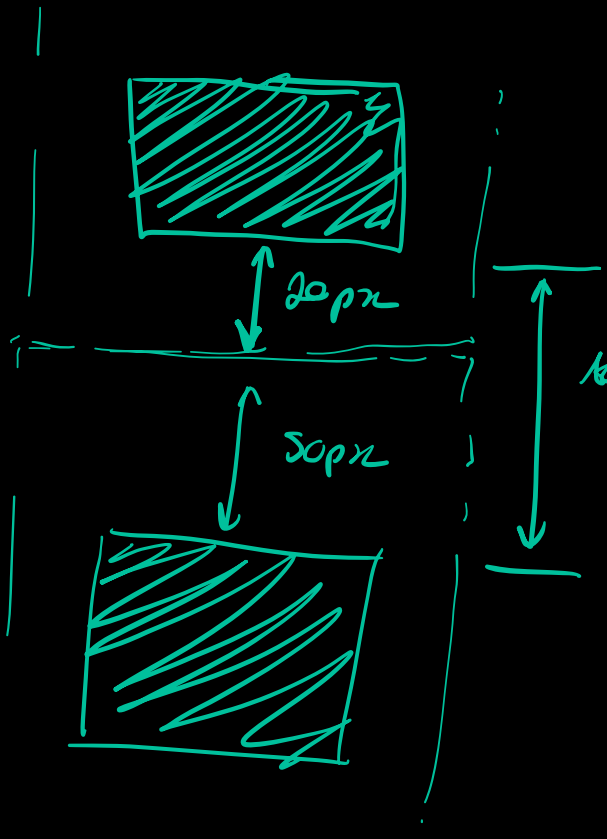
```
  margin: 20px;
}

#two {
  margin: 50px;
}

img {
  margin-left: 20px;
  margin-right: 20px;
}
```

JS

→ vertical margins collapse



total margin ≠ 20 + 50 = 70px
→ total margin = 50px → it uses the largest margin val. of the x2 (≠ add)

→ put two els next to each other → the margin which is used is largest of the x2 (not the Σ of them both)

→ but horizontal margins add

5. Control an element's width and height

→ el's have default heights + widths in CSS ← you can fix

Secure | https://codepen.io/eclairereese/pen/YLRzdO

Width and height
A PEN BY Emily Reese

Save | Fork | Settings | Change View

Buy our products

We are the leading worldwide manufacturer of paintbrushes and pride ourselves on the highest bristle quality possible. Check out what our customers have to say:

"Amazing brushes! My painting career wouldn't be the same without them."
- John

"An artist is only as good as her tools. Thanks, ABC, for a great product."
- Sarah

"I've never used such strong brushes!"
- Linda

HTML

```
<div class="main">
  <h1>Buy our products</h1>
  <p>We are the leading worldwide manufacturer of paintbrushes and pride ourselves on the highest bristle quality possible. Check out what our customers have to say:</p>
  <blockquote>"Amazing brushes! My painting career
```

CSS

```
blockquote {
  padding: 20px;
  background-color: lightgreen;
  border-radius: 10px;
  margin: 10px auto;
  text-align: center;
}
```

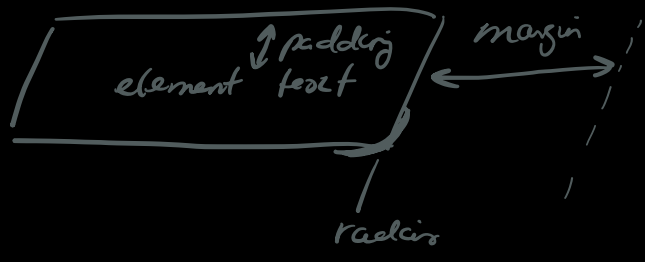
JS

width : 50% ← the el takes up 50% of the width of the containing el
: 300px → this can withstand page resizes (is proportional)

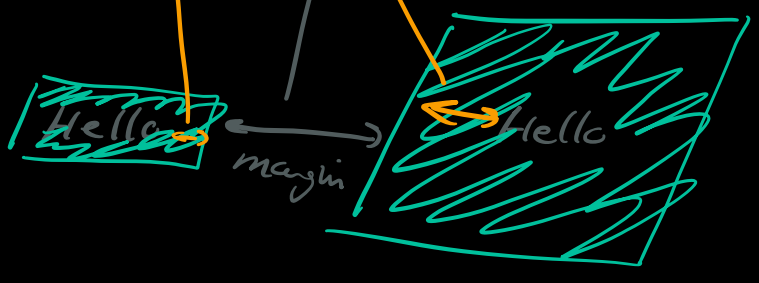
height : 50%

spans entire width of page

→ padding



→ border radius – how curved the boarders are
→ padding – dist. bet content & edge of highlighted section
→ margin – dist bet rectangles containing content –



→ this video is how to Δ content width (not box around content) → if the content is in a textbox, we are resizing the textbox containing the content – vs the box which contains the highlights behind it

MIN WIDTH / MAX WIDTH

min-width: 300px ← the el will expand to fill the page, but doesn't get smaller than 300px

→ "visual responsiveness" across different screen sizes / devices

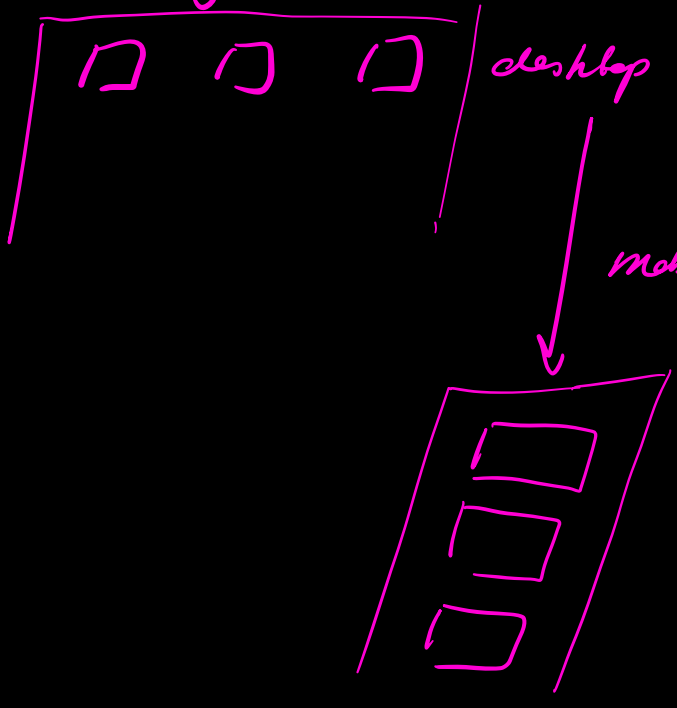
- make the screen bigger, the box containing the element expands to fill it → but if you expand the screen size - past max width (set in CSS), the content will stop expanding as the screen expands (and vice versa with the screen shrinking)

"MAX WIDTH"

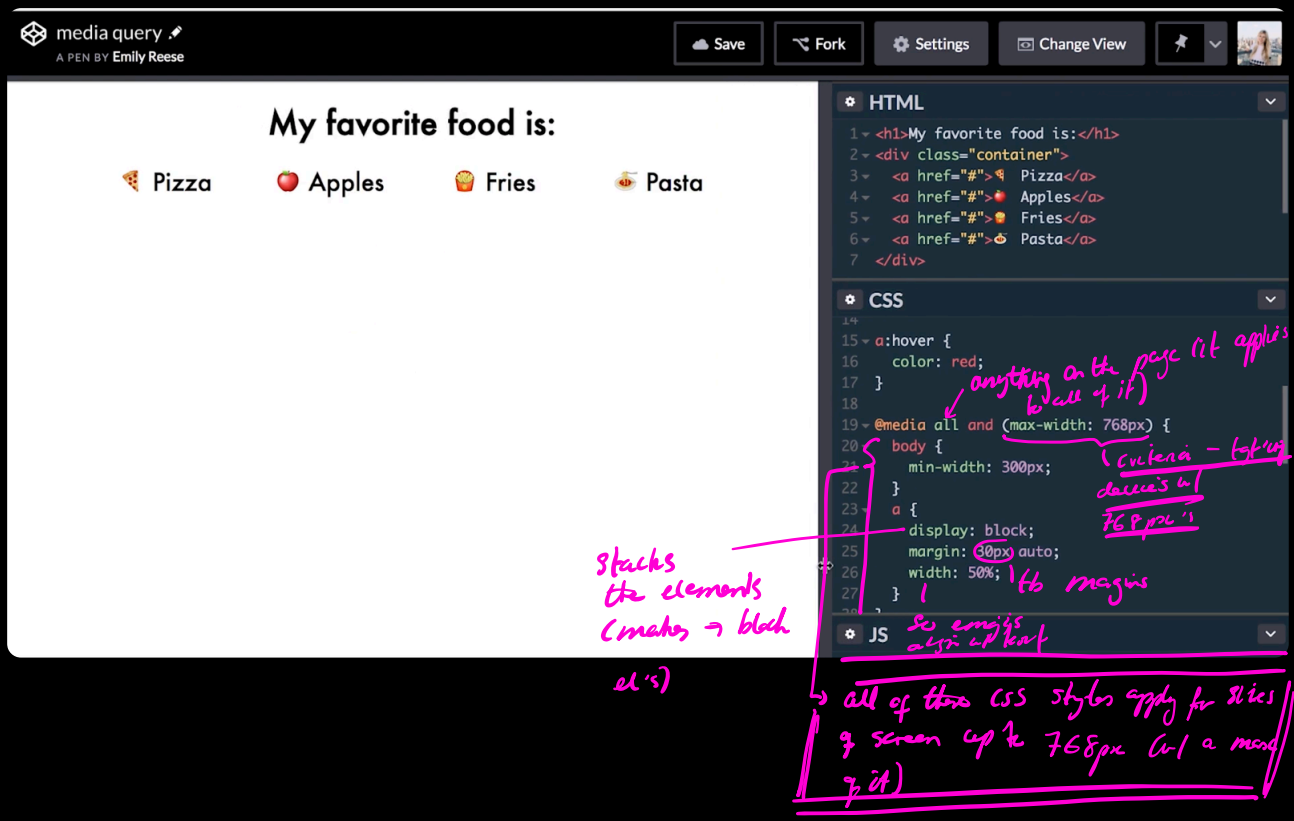
6. Set media queries for different devices

→ pages viewed from different sizes ← media queries in CSS

→ stacking menu items on top of eachother



in CSS you use a "media query" → there are standard screen sizes, e.g. 768px



MEDIA QUERIES: FOR EXPANDING SCREEN SIZES

IN CSS → STACKING CONTENT

FOR DIFFERENT SCREEN SIZES

Part #2 - Create two-dimensional layouts with CSS Grid

1. Introduction to CSS Grid
2. Set up a basic grid
3. Set columns and rows in shorthand
4. Define grid element height and width
5. Create a grid template area
6. Set columns depending on screen size

Quiz: Create grids with CSS Grid

1. Introduction to CSS Grid

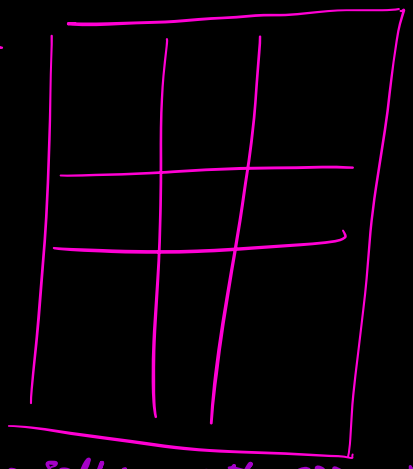
→ Arranging els on a webpage → CSS box model
→ each el = surrounded by a box

→ this course content is → CSS grid, flexbox + legacy ways of setting layouts

→ so, all el's on a page are surrounded by boxes - prev chapter was how to format / ctrl each box → this one is about styling different boxes on an entire webpage

2. Set up a basic grid

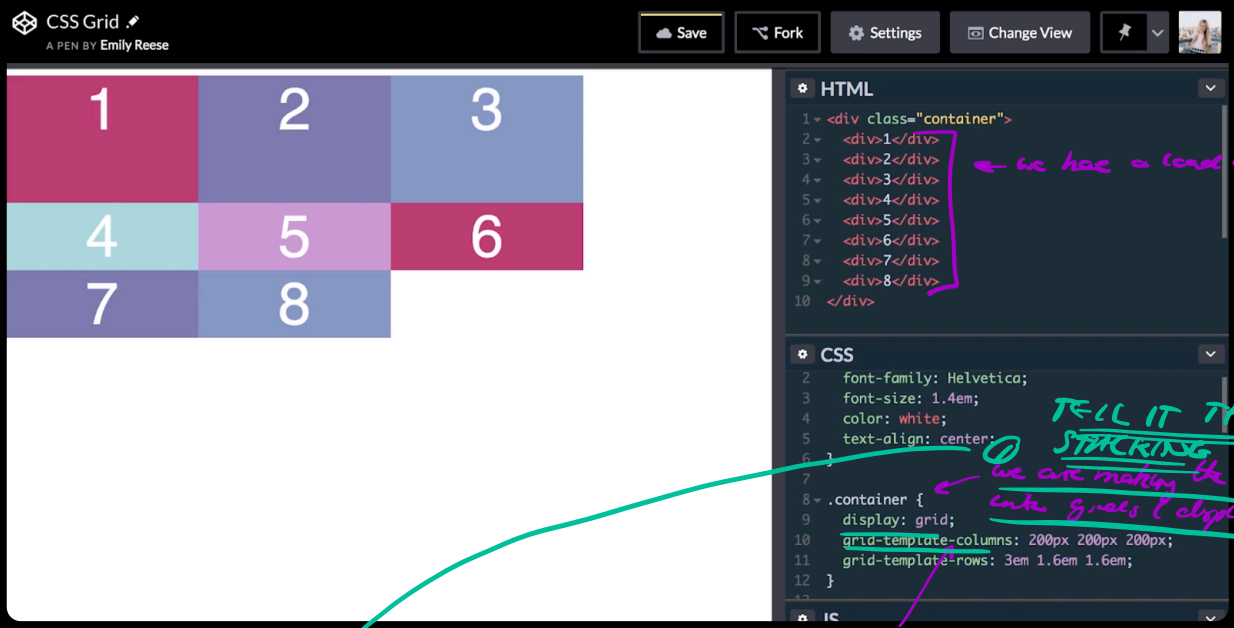
→ the CSS grid — prop's + values



split webpage into grids (each el = part of the grid)

→ setting up the CSS grid in CSS

3. Set columns and rows in shorthand



← we have a load of containers

TELL IT THE EL's YOU HAVE STACKING

① we are making the containers into grids (display: grid)

TELL IT THEIR WIDTHS ETC.

② if you don't specify that, then nothing happens

the widths of each column you want the content stacked into

200px | 200px | 200px

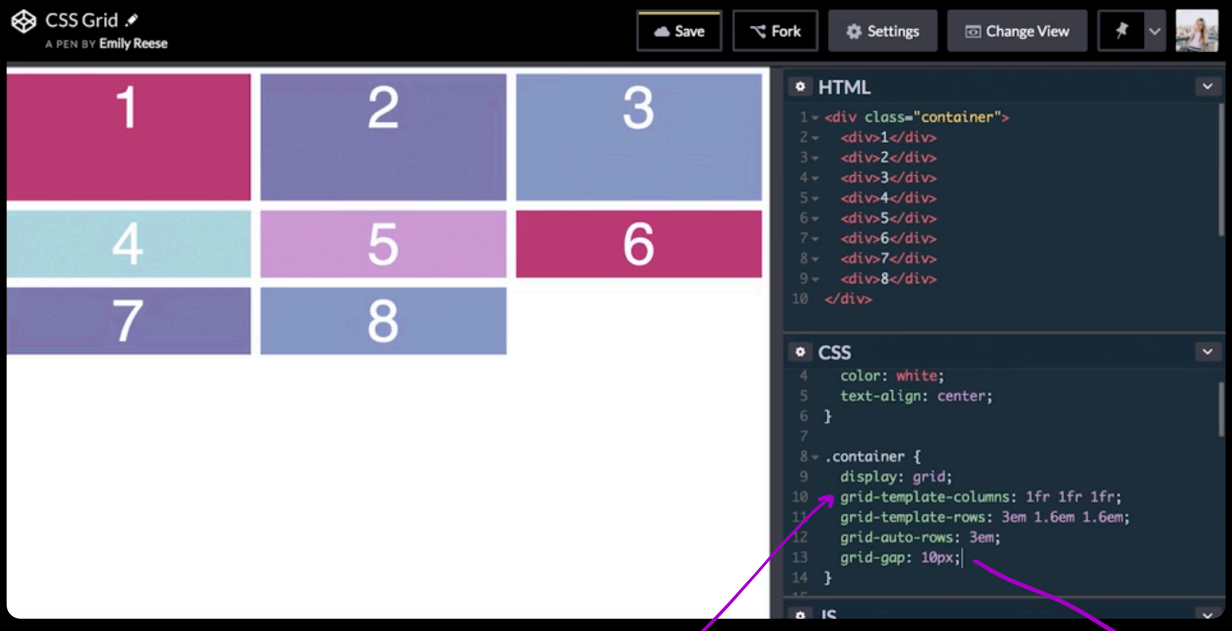
→ the boxes are stacked in it

→ you can also set row heights

→ alt. grid-auto-rows: 3em;

→ if you add more content, it adds onto rows of height 3em

"grid-auto-rows"

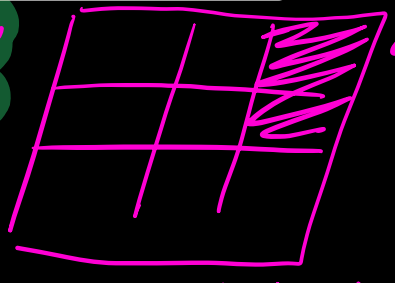


the fraction unit = fr
→ the width of the three columns to be ratio 1:1:1
→ you can also eg 1:2:3

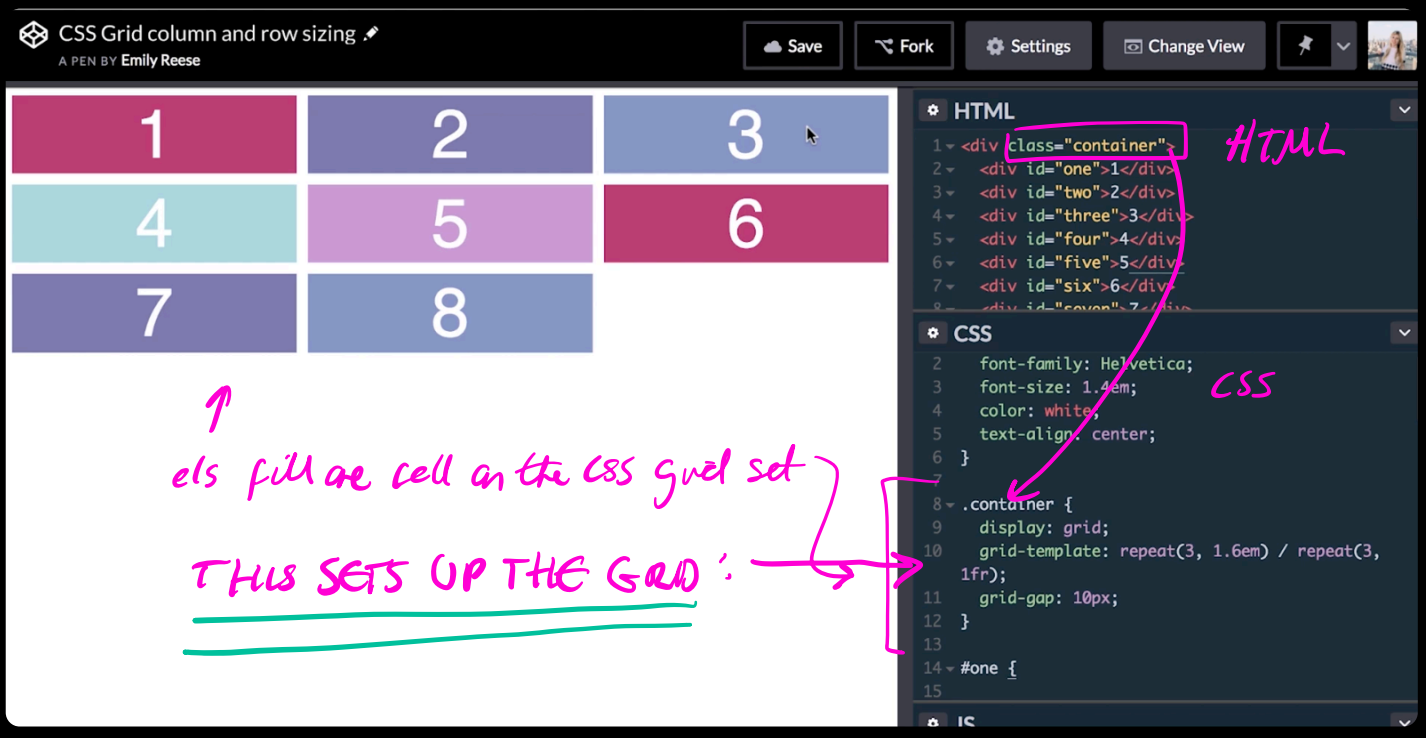
TO ADD GAPS BET ROWS
→ grid-gap → spaces out the rows, columns

4. Define grid element height and width

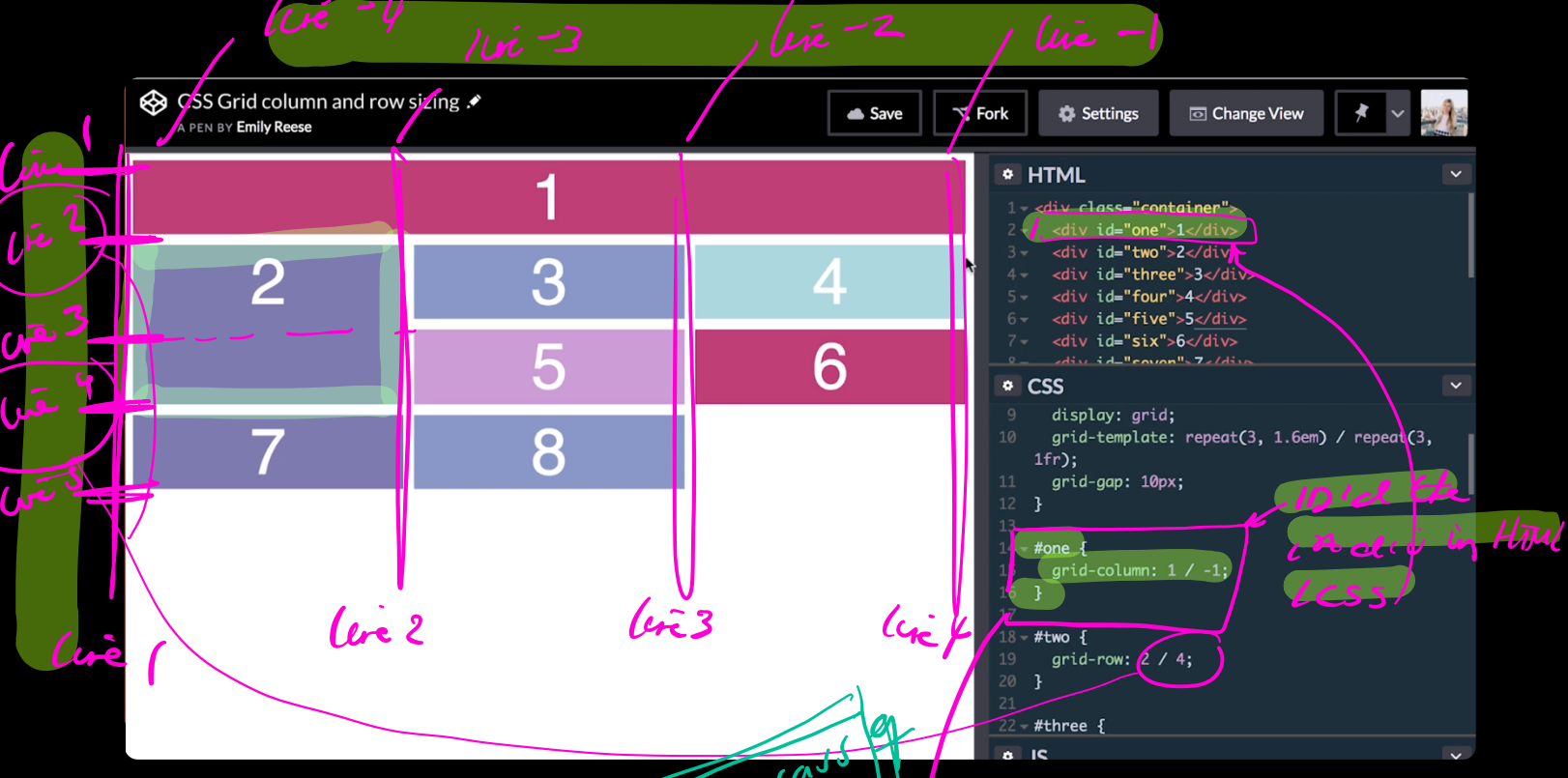
→ if you have a css grid and want one el to take up two boxes e.g on the grid



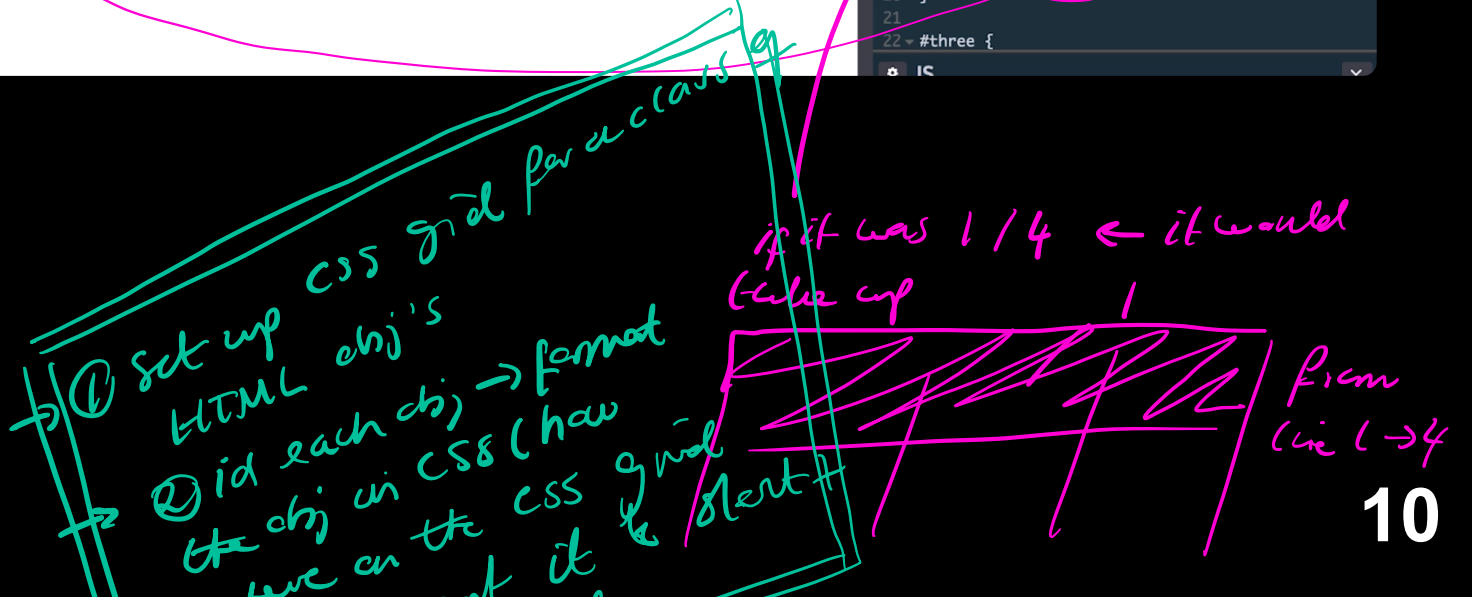
→ prev. was setting grid size, this is ctrl'ing the boxes each el takes on grid

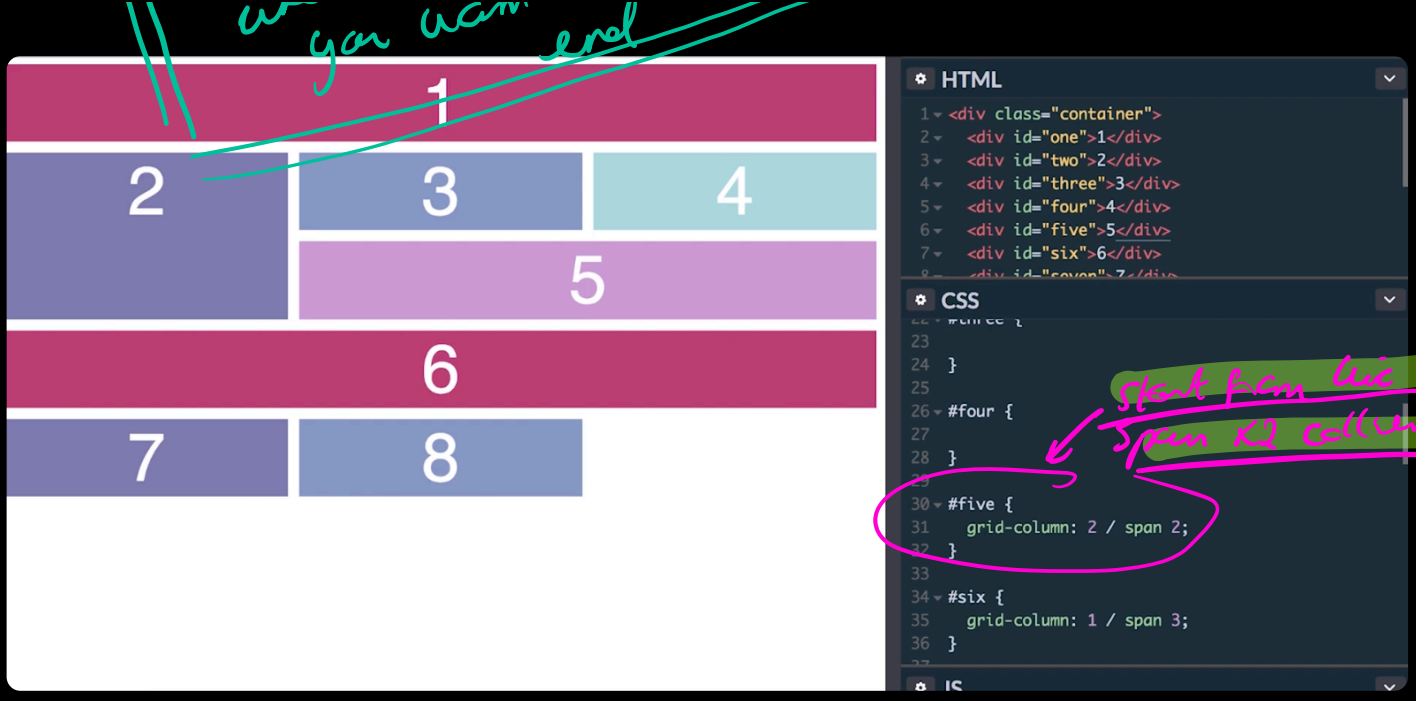

CSS Grid column and row sizing
A PEN BY Emily Reese
Save
Fork
Settings
Change View
1
2
3
4
5
6
7
8
HTML

1
2
3
4
5
6
CSS
font-family: Helvetica;
font-size: 1.4em;
color: white;
text-align: center;
}
.container {
display: grid;
grid-template: repeat(3, 1.6em) / repeat(3, 1fr);
grid-gap: 10px;
}
#one {
HTML
CSS
els fill ae cell an the CSS grid set
THIS SETS UP THE GRID :

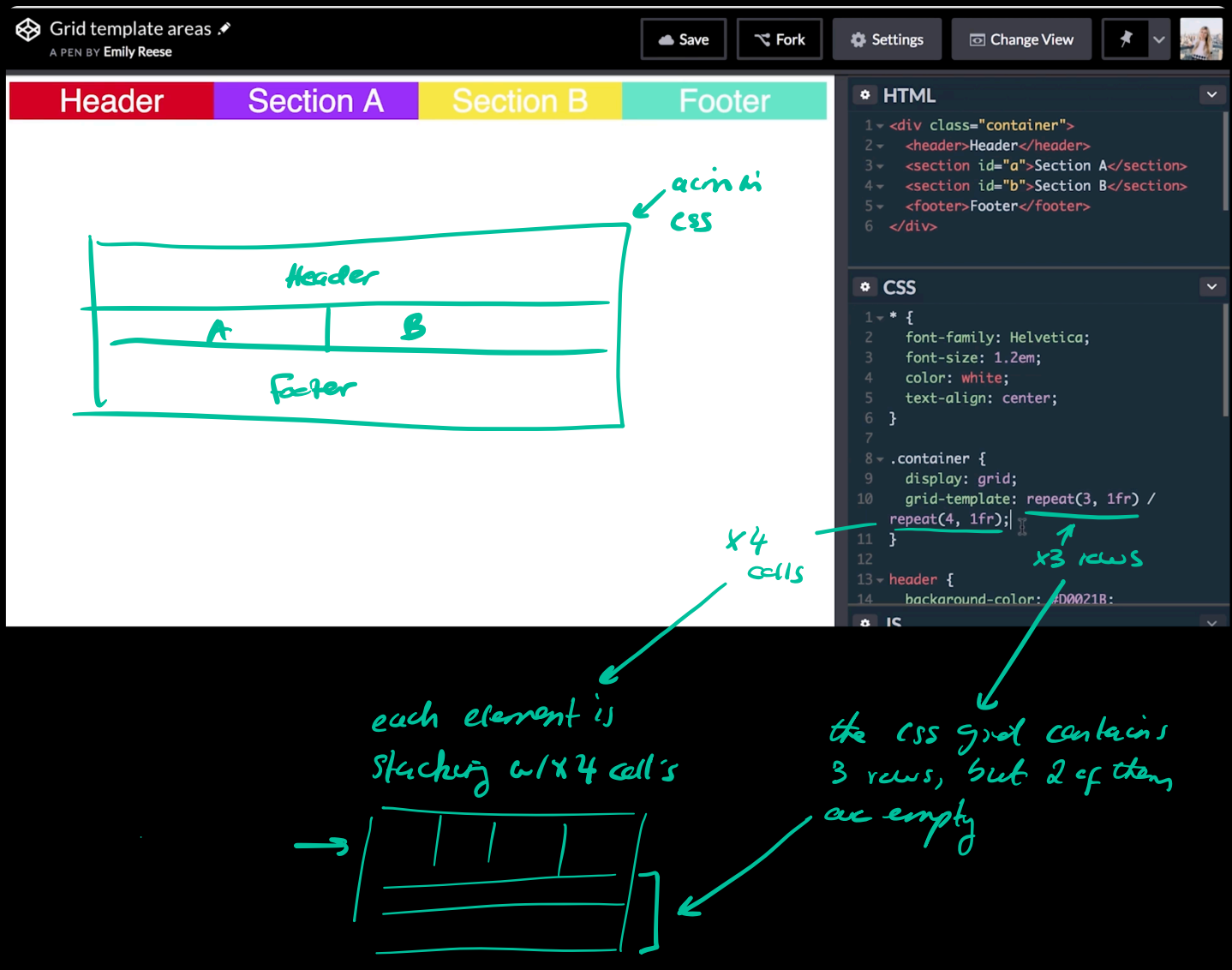

line -4
line -3
line -2
line -1
CSS Grid column and row sizing
A PEN BY Emily Reese
Save
Fork
Settings
Change View
1
2
3
4
5
6
7
8
HTML

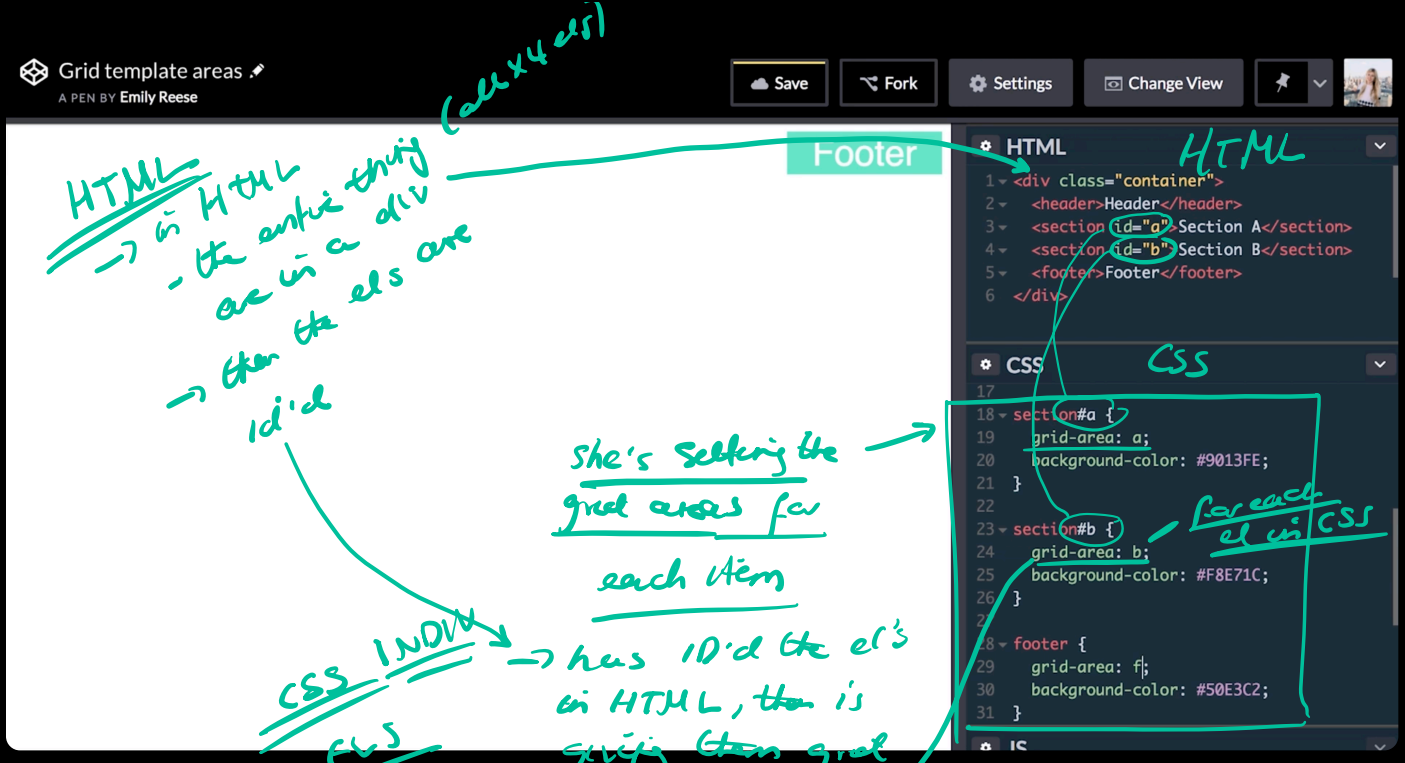
1
2
3
4
5
6
CSS
display: grid;
grid-template: repeat(3, 1.6em) / repeat(3, 1fr);
grid-gap: 10px;
}
#one {
grid-column: 1 / -1;
}
#two {
grid-row: 2 / 4;
}
#three {
id the
in HTML
CSS
line 1
line 2
line 3
line 4

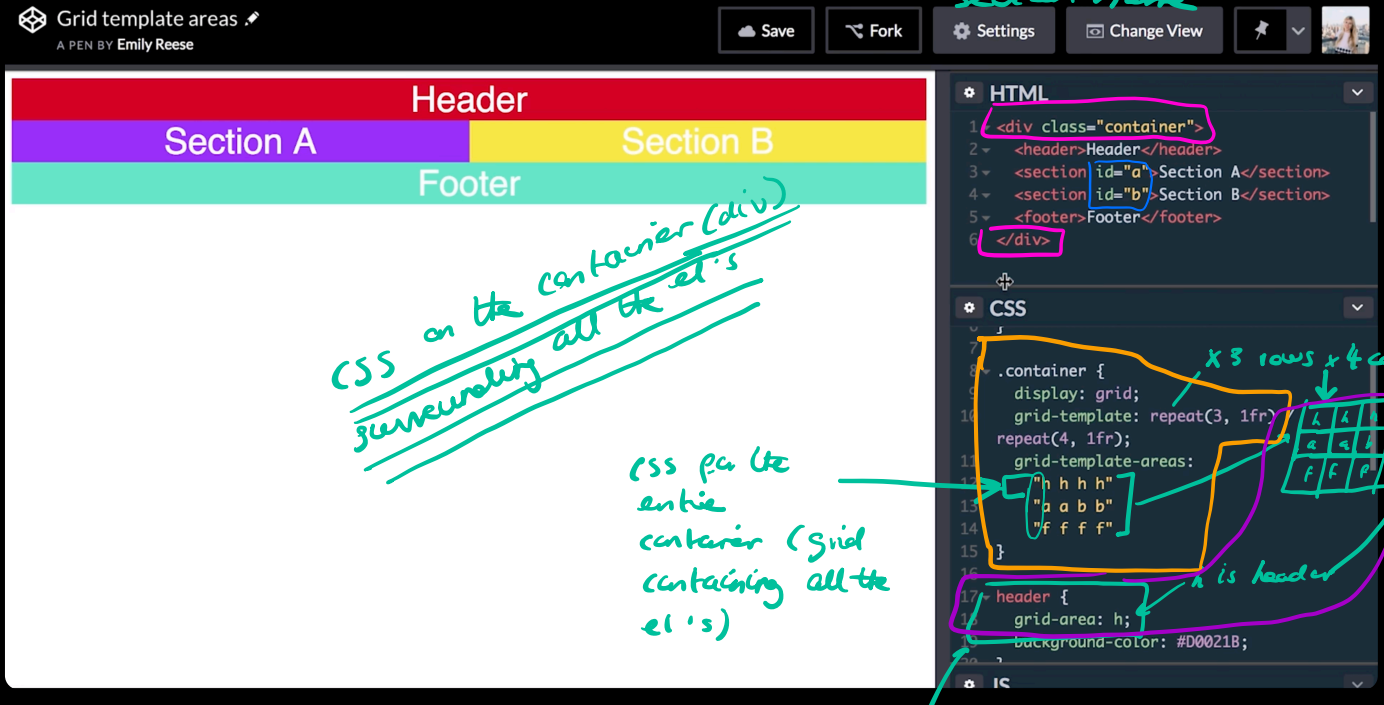

① set up CSS grid for classes
HTML obj's
② id each obj → format
the obj in CSS (how
we on the CSS grid
if it was 1 / 4 ← it would
Color up
1
from
line 1 → 4



5. Create a grid template area







IN HTML (CONTENT)

→ div around all el's in a class ← entire thing

→ then id's on the indiv el's ← indiv el's

IN CSS

→ entire thing → . container {

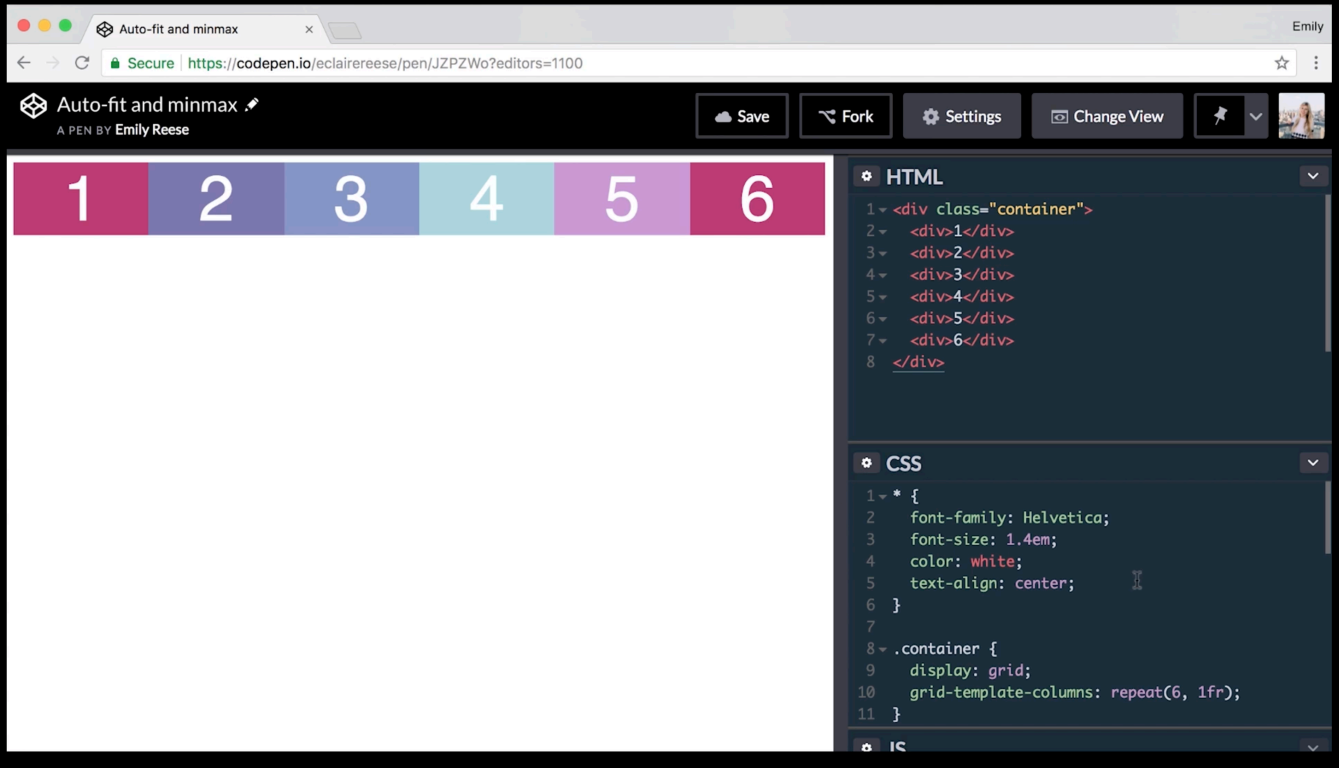
↳ and the ones which have

→ tells it to have a CSS grid (CSS, CSS) and what part of the grid belongs to what el

(e.g. header) are first

→ formats grid items els according to when should go on the CSS grid

6. Set columns depending on screen size



→ screen sizes and the CSS grid

→ shrink the screen and els on the CSS grid start overlapping → e.g. for iPhones - you need the els to stack as be next to each other

→ to stack els on the CSS grid below a certain screen size → autofit, minmax

THIS IS FOR THE DIV AROUND ALL THE CONTENT:

BEFORE: repeat(6, 1fr); ← 6 cells, each in a ratio of 1:1:1:etc.

AFTER: repeat(auto-fit, minmax); ← (an auto no of cells, the ratio of which adapts on the screen size)

```
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
```

min width of the cells

max width of the cells

→ as the screen size Δ's

Points from CSS Quiz 2 -> create 2D layouts with the CSS grid

- the CSS grid is 2D -> there are probably 3D versions
- you set grids on individual CSS elements -> by putting divs around them with classes which you call containers, then formatting those containers in CSS -> display: grid
    - so you can have an element in HTML on the webpage and make it into a grid
- the syntax
    - -> grid-template: repeat(3, 80px) / repeat(2, 80px);
    - -> the rows / columns
    - -> and in each of them, the units are repeated last
- older browsers can't use the CSS grid -> and it needs formatting for different screen sizes

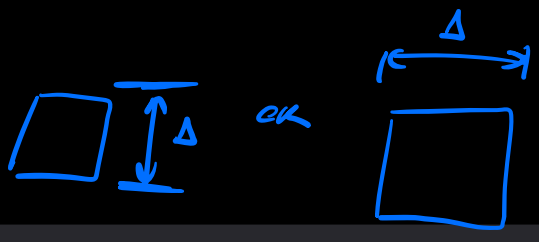
Part #3 - Implement one-dimensional layouts with Flexbox

1. What is Flexbox?
2. Set flex direction and wrapping
3. Align items and justify content
4. Align multiple lines of content
5. Adjust element dimensions
6. Reorder elements

Quiz: Flexbox basics

1. What is Flexbox?

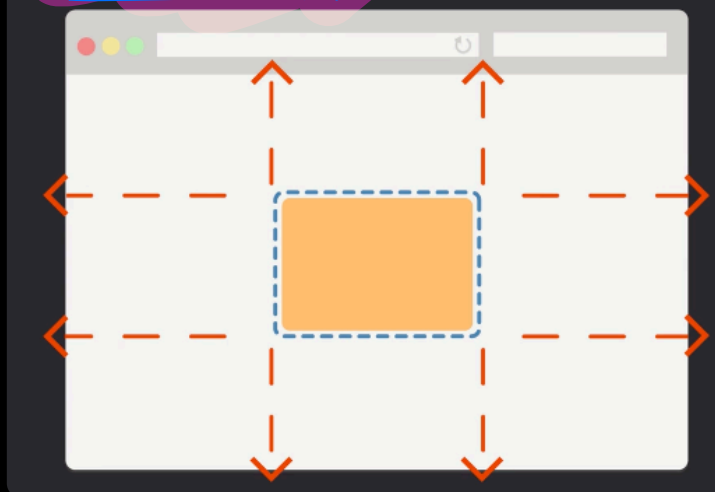
→ CSS → 1D, either rows or cols for an el
→ not an el in a grid
→ flexbox is



2. Set flex direction and wrapping

LANGUAGE:

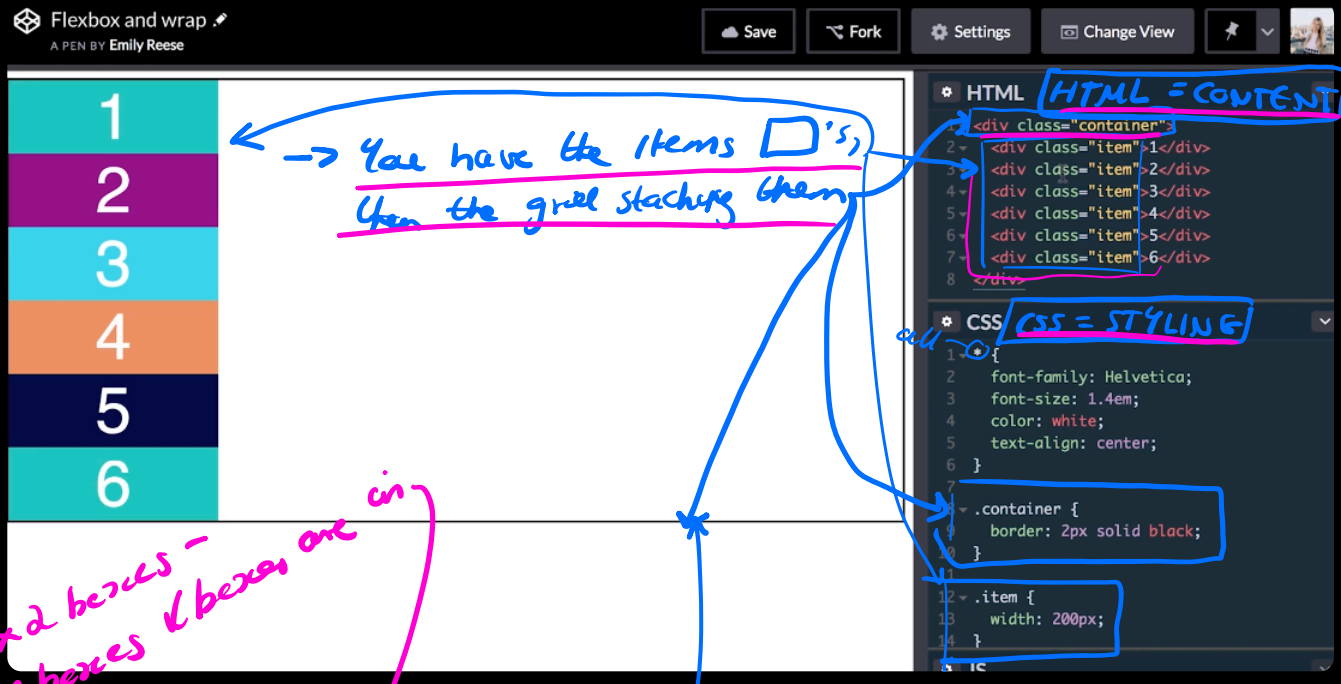
BIG BOXES:



Flex container
- the container where the el lives

Flex item
- the item in the container

Flex direction
- the direction the item flows



→ You have the items □'s, then the grid stacking them

You have 2 boxes - 1 boxes (boxes are in)

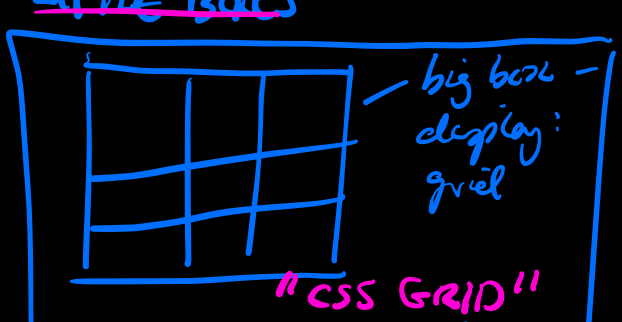
→ 2D

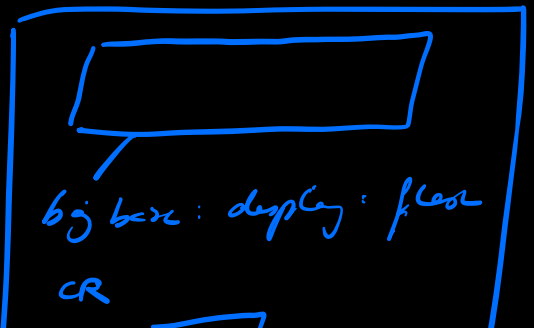
→ flexbox is on the big box for all the el's

→ A FLEX CONTAINER VS GRID CONTAINER FOR BIG BOX CONTAINING LITTLE BOXES → this is as a regular one w/ h, C's (display: grid)

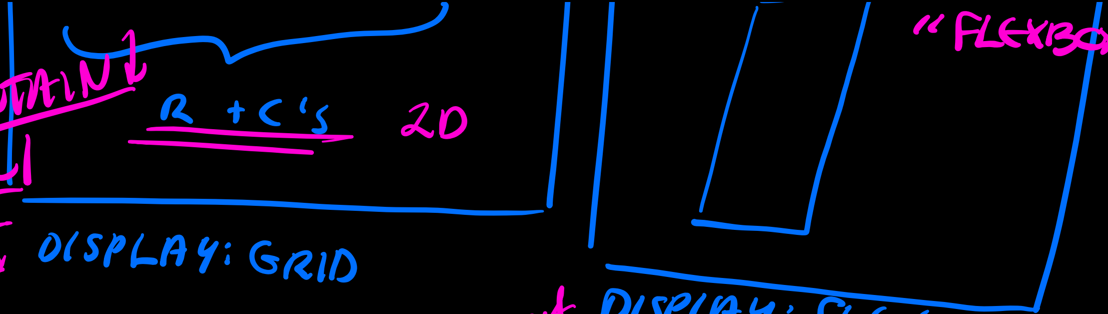
→ display: flex → is for styling the container in one d. ↔ or ↕

BIG BOXES ARE ID CR





2D AND CONTAINT BOXES w/ CONTENT



R + C's 2D

DISPLAY: GRID = R's + C's

"FLEXBOX"

DISPLAY: FLEX = R or C's

1D

→ ↑boxes are defined in divs w/ classes in HTML and ↓box els, are sep'd out in divs wrapped inside the ↑boxes

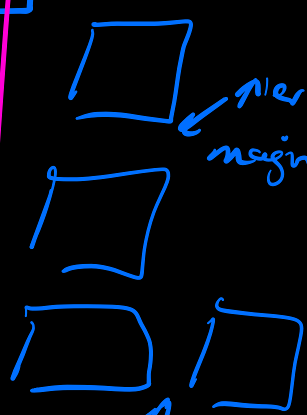
all ↓box els, of BIG BOX CONTAINING LITTLE BOXES

LITTLE BOXES CONTAINING CONTENT:

This is the terminology for the ↓ boxes containing the content

nothing touches things outside these margins —

margins impact how el's stack (gaps bet them)



↑ler 4 x2 margins

Σ 4 x2 margins

→ then when resizing = min/max

→ for iPhone version e.g. = for ↑er box on CSS grid

then you have the biggest box which is the screen containing the webpage

BIG BOXES → big boxes contain little boxes w/ content

a) flexbox → r or c's ← this video (1D), ( ) or ( )

b) CSS grid → r's + c's (2D grid) → (2D matrix)