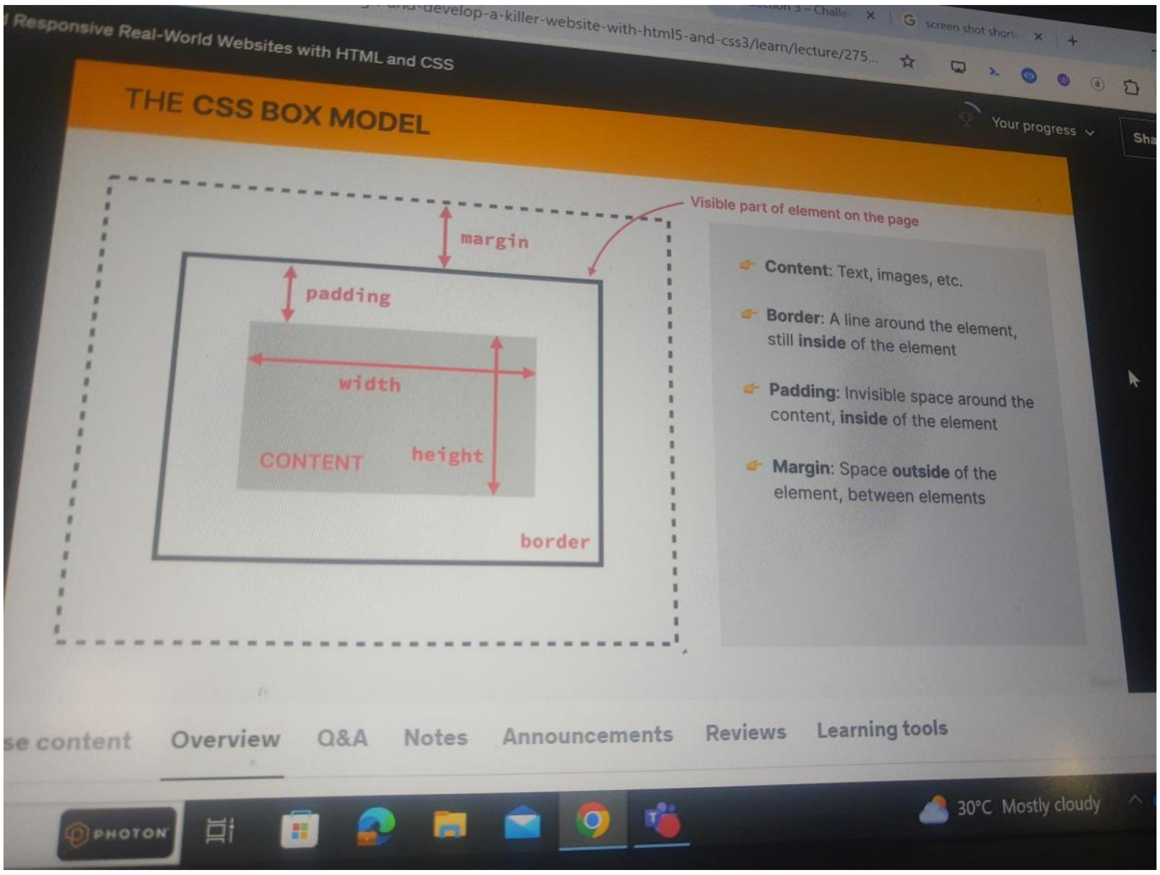
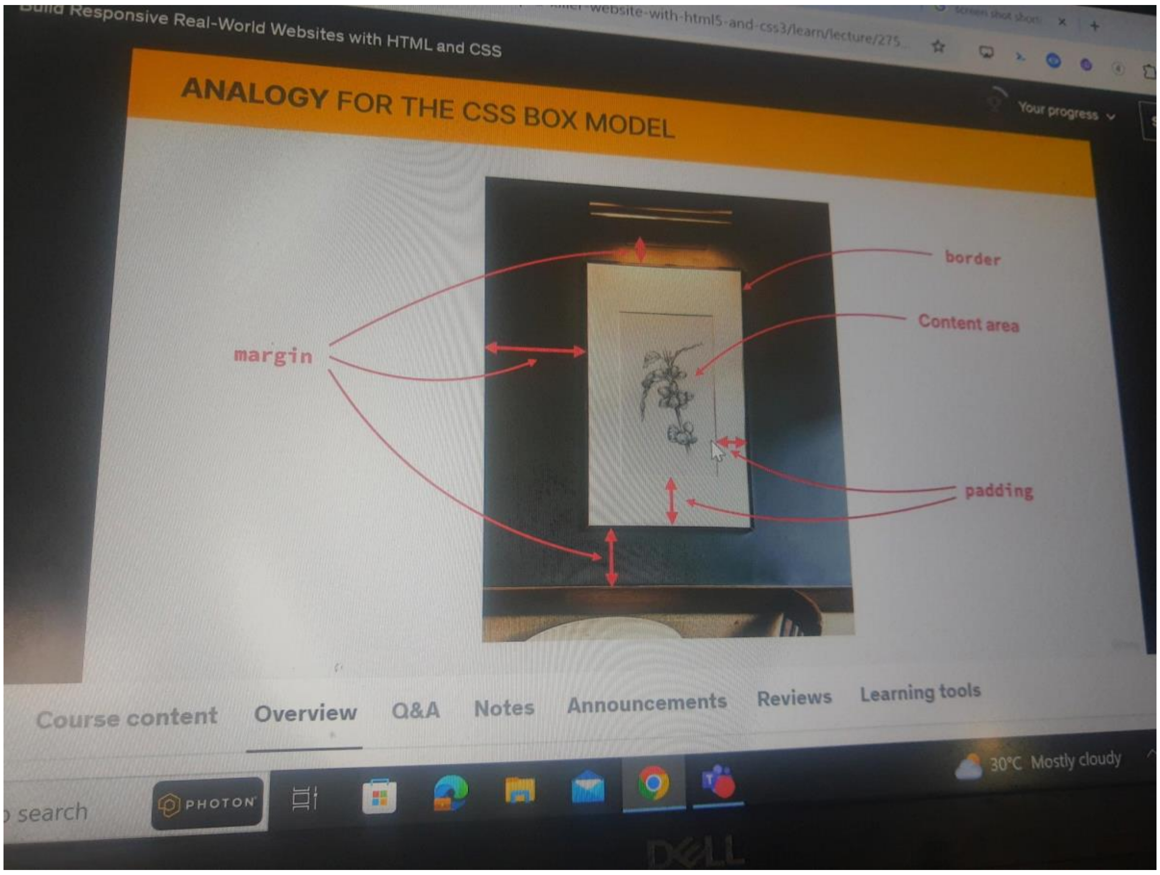
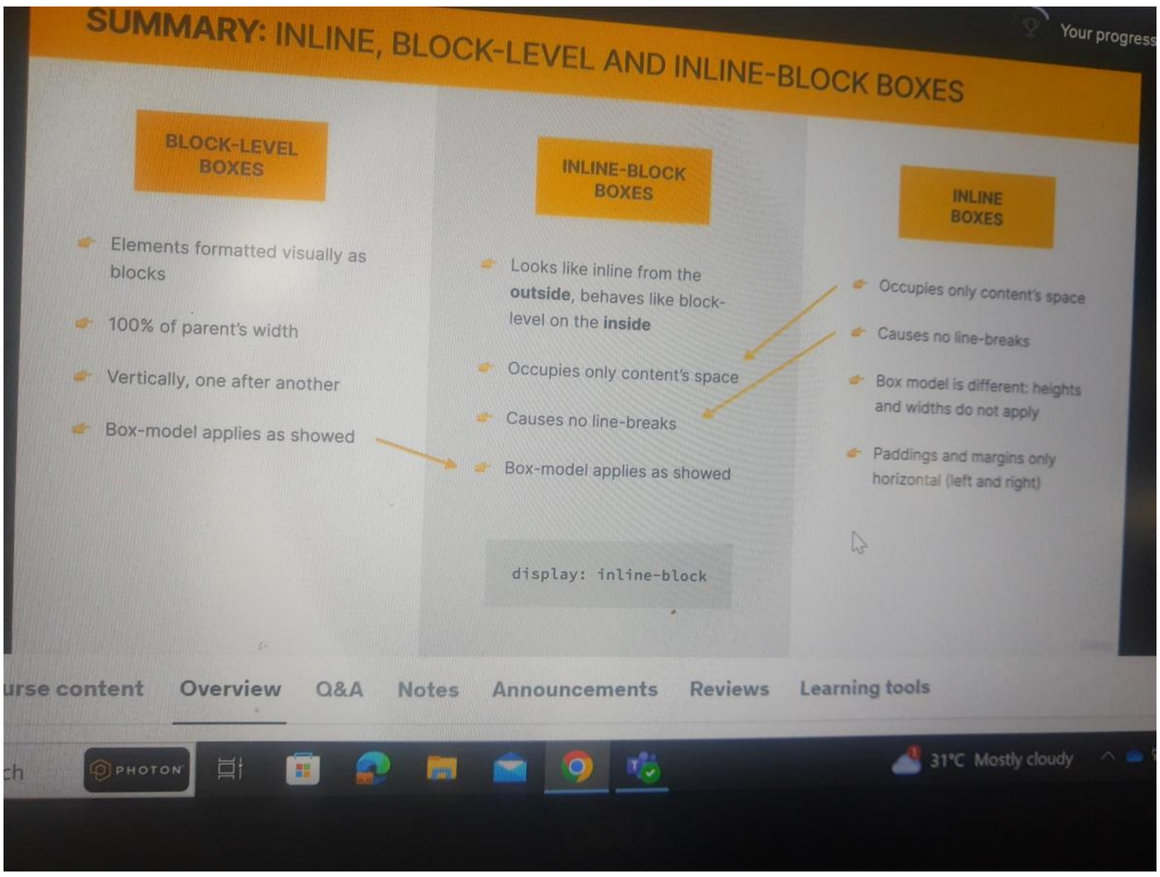


THE CSS BOX MODEL

- **Content**: Text, images, etc.
- **Border**: A line around the element, still **inside** of the element
- **Padding**: Invisible space around the content, **inside** of the element
- **Margin**: Space **outside** of the element, between elements



ANALOGY FOR THE CSS BOX MODEL



SUMMARY: INLINE, BLOCK-LEVEL AND INLINE-BLOCK BOXES

BLOCK-LEVEL BOXES

- Elements formatted visually as blocks
- 100% of parent's width
- Vertically, one after another
- Box-model applies as showed

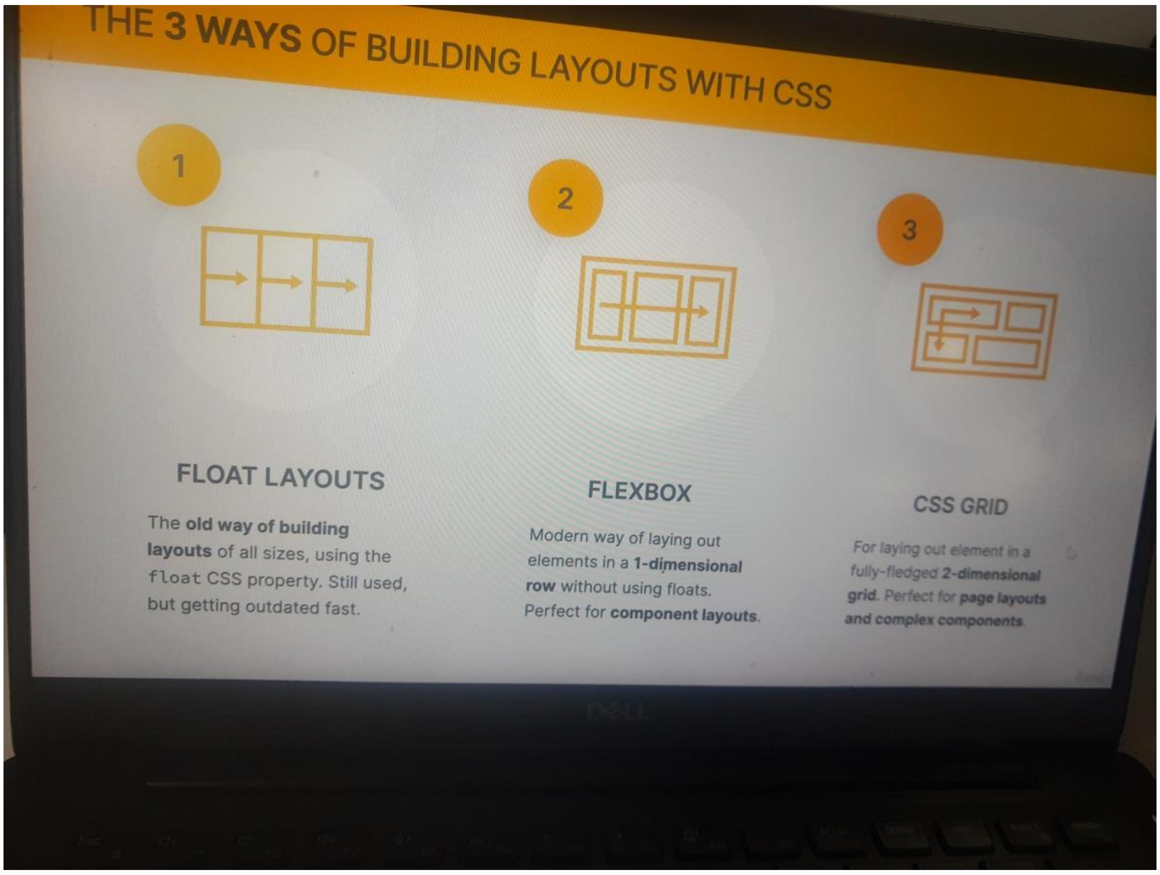
INLINE-BLOCK BOXES

- Looks like inline from the **outside**, behaves like block-level on the **inside**
- Occupies only content's space
- Causes no line-breaks
- Box-model applies as showed

```
display: inline-block
```

INLINE BOXES

- Occupies only content's space
- Causes no line-breaks
- Box model is different: heights and widths do not apply
- Paddings and margins only horizontal (left and right)



THE 3 WAYS OF BUILDING LAYOUTS WITH CSS

1 FLOAT LAYOUTS

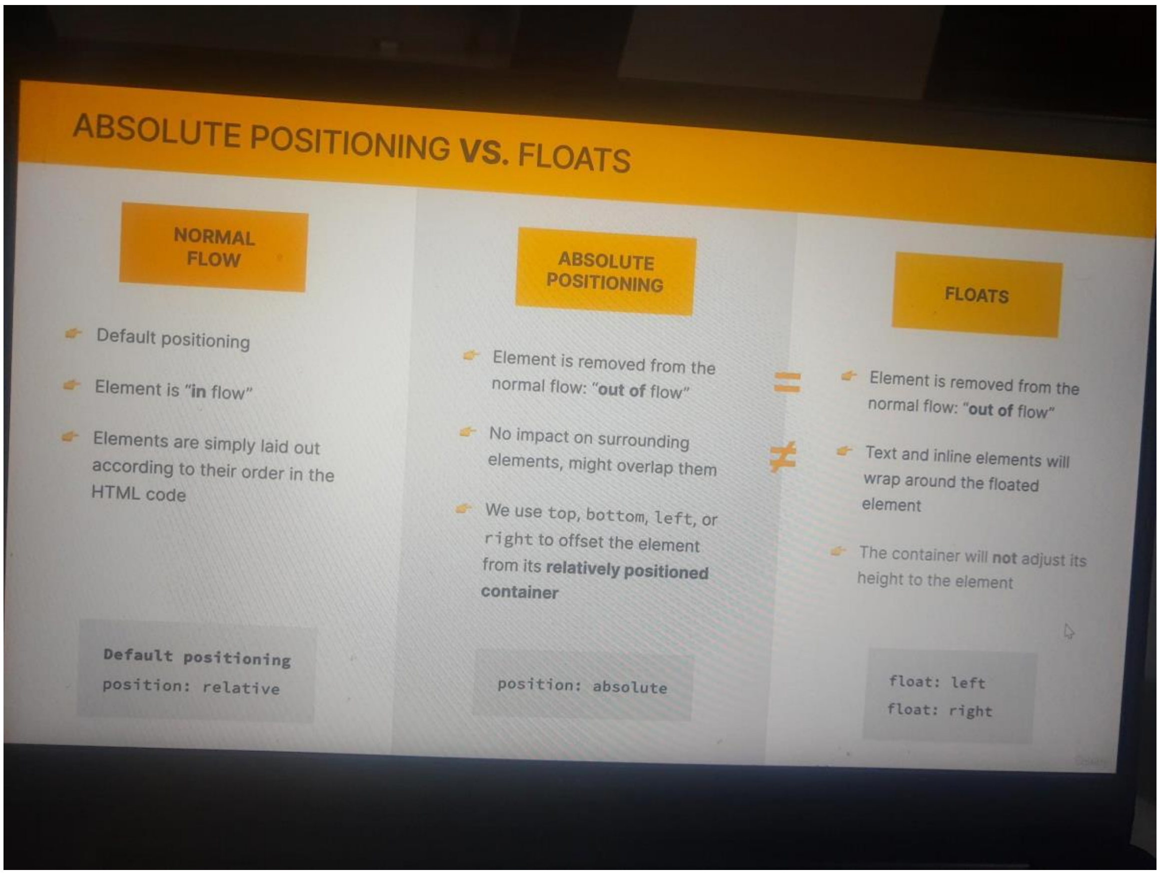
The **old way of building layouts** of all sizes, using the `float` CSS property. Still used, but getting outdated fast.

2 FLEXBOX

Modern way of laying out elements in a **1-dimensional row** without using floats. Perfect for **component layouts**.

3 CSS GRID

For laying out element in a fully-fledged **2-dimensional grid**. Perfect for **page layouts and complex components**.



ABSOLUTE POSITIONING **VS.** FLOATS

NORMAL FLOW

- Default positioning
- Element is "**in** flow"
- Elements are simply laid out according to their order in the HTML code

```
Default positioning
position: relative
```

ABSOLUTE POSITIONING

- Element is removed from the normal flow: "**out of** flow"
- No impact on surrounding elements, might overlap them
- We use `top`, `bottom`, `left`, or `right` to offset the element from its **relatively positioned container**

```
position: absolute
```

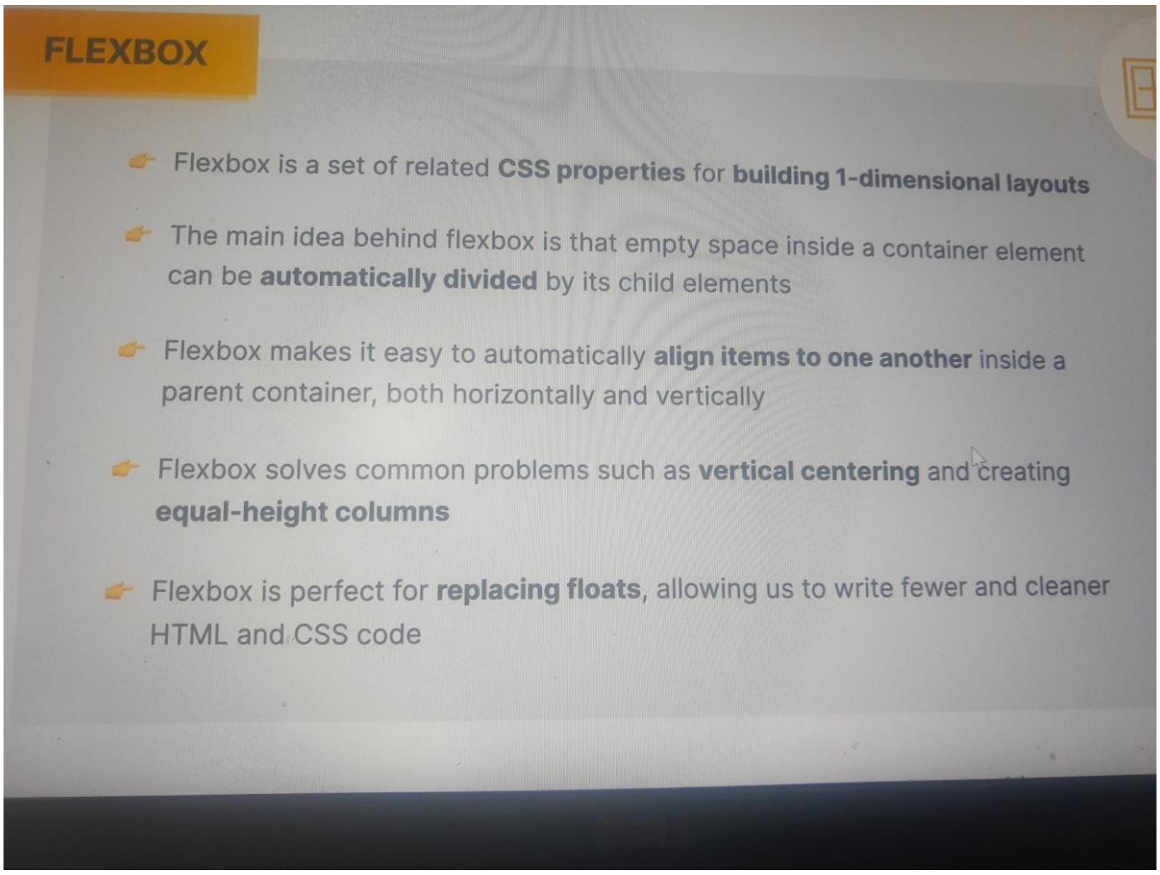
=

≠

FLOATS

- Element is removed from the normal flow: "**out of** flow"
- Text and inline elements will wrap around the floated element
- The container will **not** adjust its height to the element

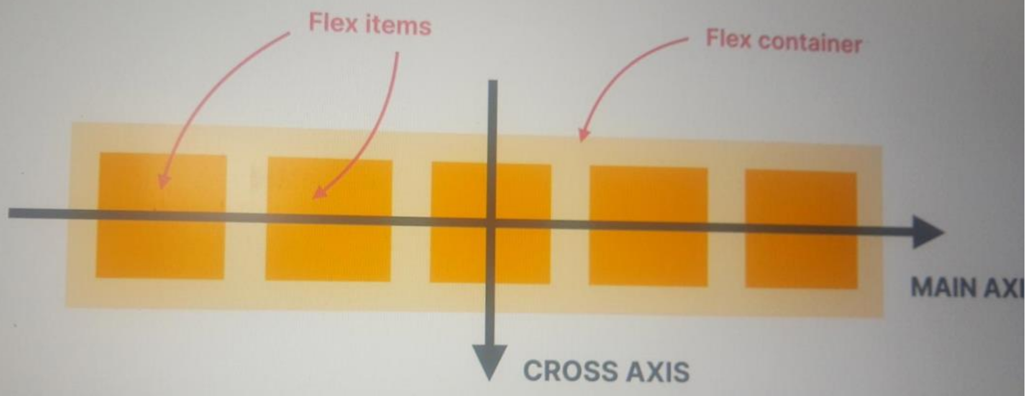
```
float: left
float: right
```



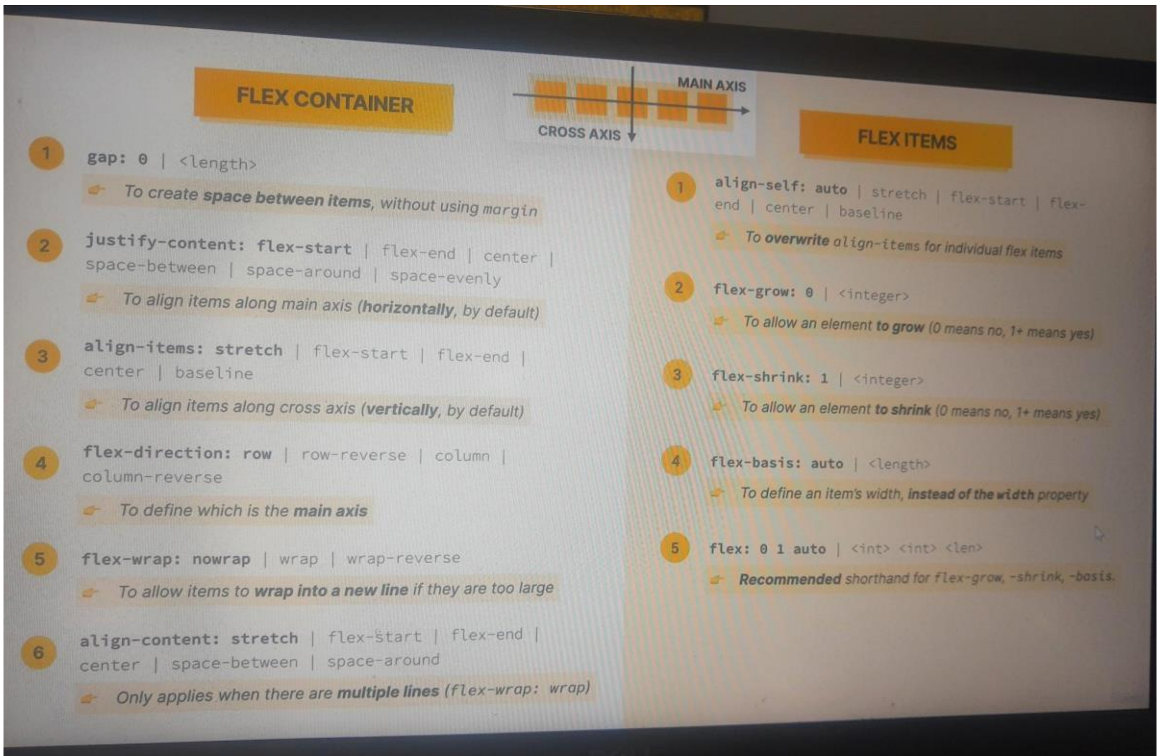
FLEXBOX

- Flexbox is a set of related **CSS properties** for **building 1-dimensional layouts**
- The main idea behind flexbox is that empty space inside a container element can be **automatically divided** by its child elements
- Flexbox makes it easy to automatically **align items to one another** inside a parent container, both horizontally and vertically
- Flexbox solves common problems such as **vertical centering** and creating **equal-height columns**
- Flexbox is perfect for **replacing floats**, allowing us to write fewer and cleaner HTML and CSS code

FLEXBOX TERMINOLOGY



```
display: flex
```



FLEX CONTAINER

1 `gap: 0 | <length>`

To create **space between items**, without using `margin`

2 `justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly`

To align items along main axis (**horizontally**, by default)

3 `align-items: stretch | flex-start | flex-end | center | baseline`

To align items along cross axis (**vertically**, by default)

4 `flex-direction: row | row-reverse | column | column-reverse`

To define which is the **main axis**

5 `flex-wrap: nowrap | wrap | wrap-reverse`

To allow items to **wrap into a new line** if they are too large

6 `align-content: stretch | flex-start | flex-end | center | space-between | space-around`

Only applies when there are **multiple lines** (`flex-wrap: wrap`)

FLEX ITEMS

1 `align-self: auto | stretch | flex-start | flex-end | center | baseline`

To **overwrite** `align-items` for individual flex items

2 `flex-grow: 0 | <integer>`

To allow an element **to grow** (0 means no, 1+ means yes)

3 `flex-shrink: 1 | <integer>`

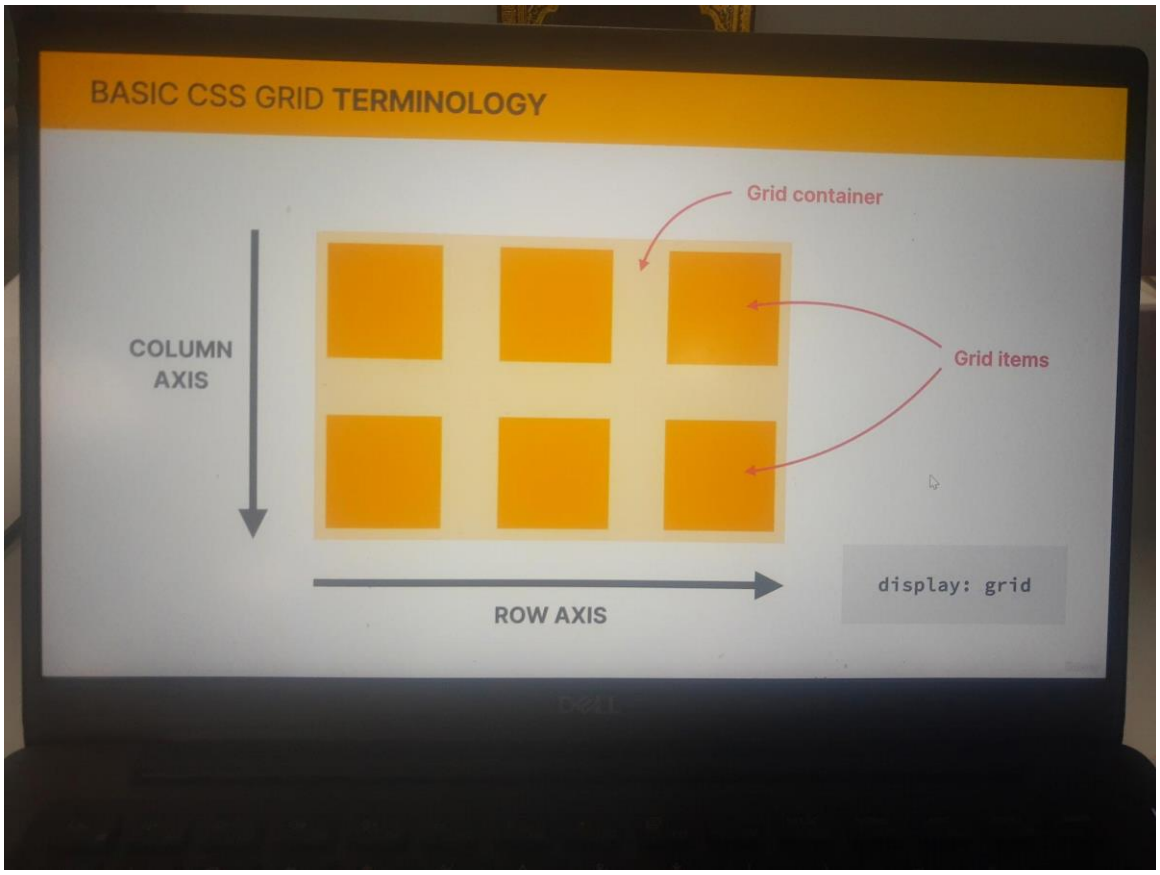
To allow an element **to shrink** (0 means no, 1+ means yes)

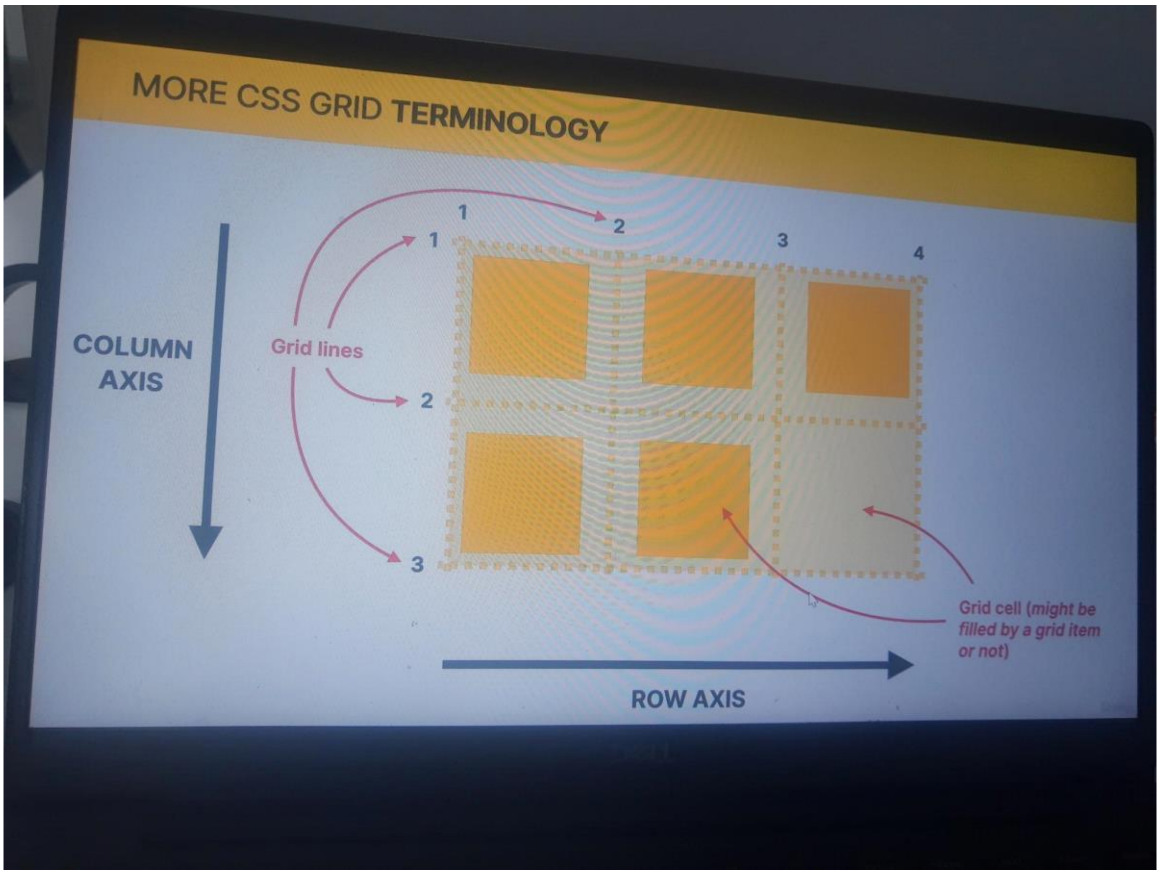
4 `flex-basis: auto | <length>`

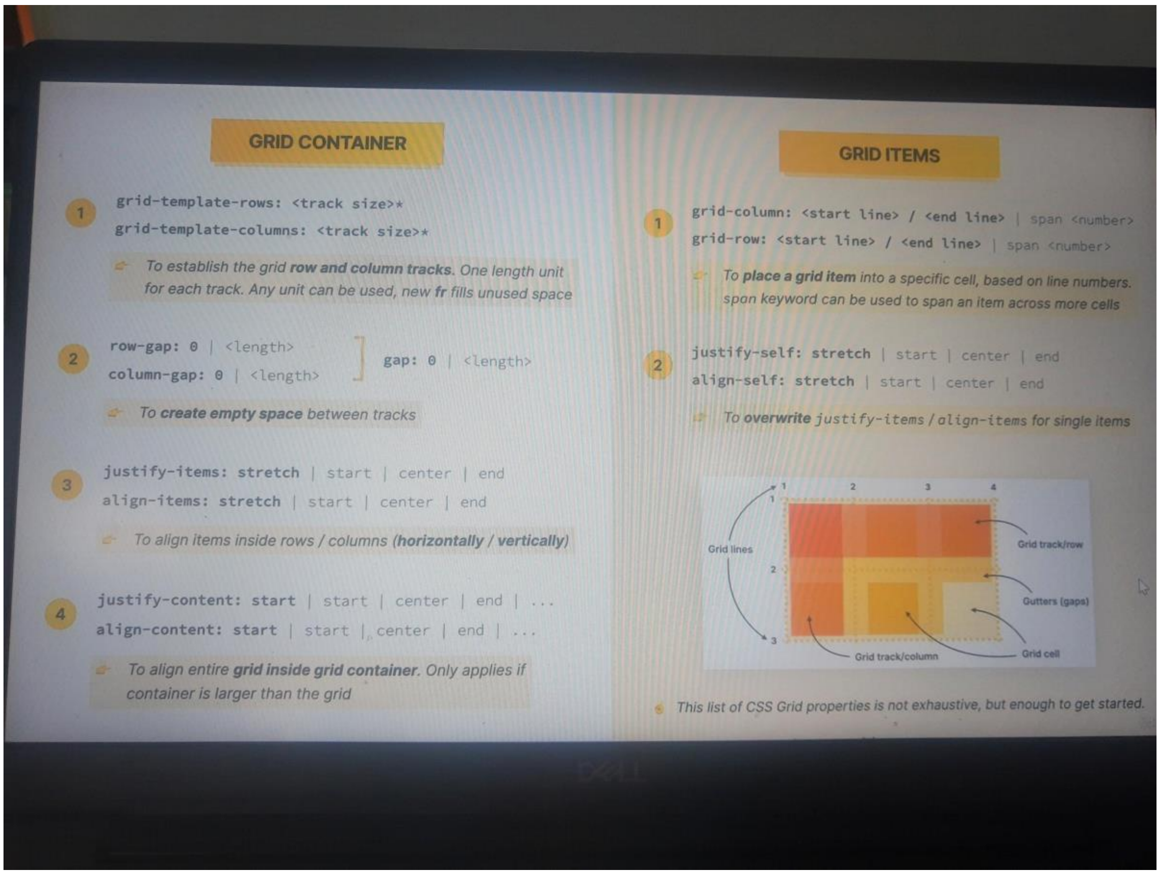
To define an item's width, **instead of the `width`** property

5 `flex: 0 1 auto | <int> <int> <len>`

Recommended shorthand for `flex-grow`, `-shrink`, `-basis`.


BASIC CSS GRID TERMINOLOGY
Grid container
Grid items
COLUMN AXIS
ROW AXIS
display: grid


MORE CSS GRID TERMINOLOGY
COLUMN AXIS
Grid lines
1
2
3
4
1
2
3
Grid cell (might be filled by a grid item or not)
ROW AXIS



GRID CONTAINER

1.
```
grid-template-rows: <track size>*
grid-template-columns: <track size>*
```
To establish the grid **row and column tracks**. One length unit for each track. Any unit can be used, new **fr** fills unused space

2.
```
row-gap: 0 | <length>
column-gap: 0 | <length>      ] gap: 0 | <length>
```
To **create empty space** between tracks

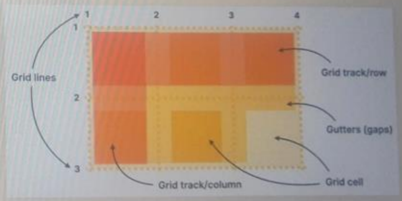
3.
```
justify-items: stretch | start | center | end
align-items: stretch | start | center | end
```
To align items inside rows / columns (**horizontally** / **vertically**)

4.
```
justify-content: start | start | center | end | ...
align-content: start | start | center | end | ...
```
To align entire **grid inside grid container**. Only applies if container is larger than the grid

GRID ITEMS

1.
```
grid-column: <start line> / <end line> | span <number>
grid-row: <start line> / <end line> | span <number>
```
To **place a grid item** into a specific cell, based on line numbers. span keyword can be used to span an item across more cells

2.
```
justify-self: stretch | start | center | end
align-self: stretch | start | center | end
```
To **overwrite** `justify-items` / `align-items` for single items



This list of CSS Grid properties is not exhaustive, but enough to get started.