CODING

Learning

▼ HTML - CSS

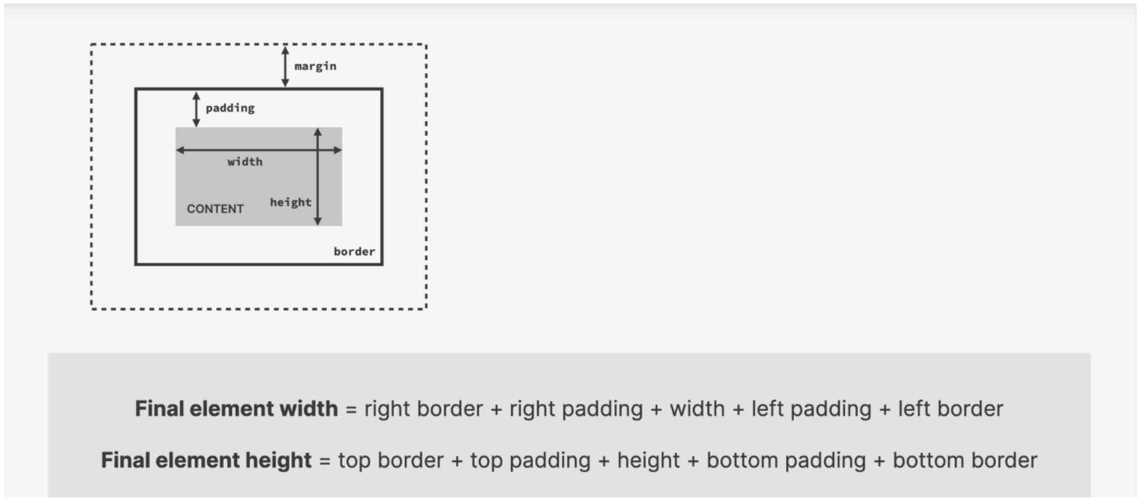
- Each ID name can be used only once
- Class names can be used multiple times
- ID is not used in daily life because to be ready for the future it might not be used more than once today but it is not guaranteed in the future

Priority Applying CSS (BASIC VERISION, normally more complicated than this)
5- Declaration marked !important
4- Inline Style(Style attributes in HTML)
3- ID(#) selectors, if there are multiple LAST SELECTOR in code applies
2- Classes/pseudo-classes, if there are multiple same as IDs
1- Element selector(p, div, li, etc.), if there are multiple same as above
0- Lowest priority is Universal selector(*)

Padding: Is the whitespace we create INSIDE the element
Margin : is the whitespace we create OUTSIDE the element

💡 It is more common to create bottom whitespaces than the top ones and do not mix them top or bottom

▼ Photo



Final element width = right border + right padding + width + left padding + left border

Final element height = top border + top padding + height + bottom padding + bottom border

---

Universal selector applies properties to all elements but those properties will not be inherited(changed)

INLINE ELEMENTS

1. Box model applies in a different wat: height and widths do not apply
2. Padding and margins are applied only horizontally (left and right).

Inline-Block Boxes:

1. Looks like inline from the outside, behaves like block-level on the inside
2. Occupies only content's space
3. Causes no line-breaks
4. Box-model applies as showed (Height, padding, margin)

Normal Flow (position: relative):

- Default positioning

- Element is "in flow"
- Elements are simply laid out according to their order in the html code

Absolute Positioning (position: absolute):

- 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

! You can put any element that you want wherever you want it to be in the page
! Do not use it for complex positioning.

Styling Links

1. Normally link and visited pseudo classes are very similar so most of the time it is better to select them together

   ▼ Example:

   ```
   .more-info:link,
   .more-info:visited {
       color: black;
       display: inline-block;
       margin-bottom: 30px;
   }
   .more-info:hover,
   .more-info:active {
       text-decoration: none;
   }
   ```

2. ALWAYS IN THIS EXACT ORDER:

   Link -> Visited -> Hover -> Active

   ▼ Example:

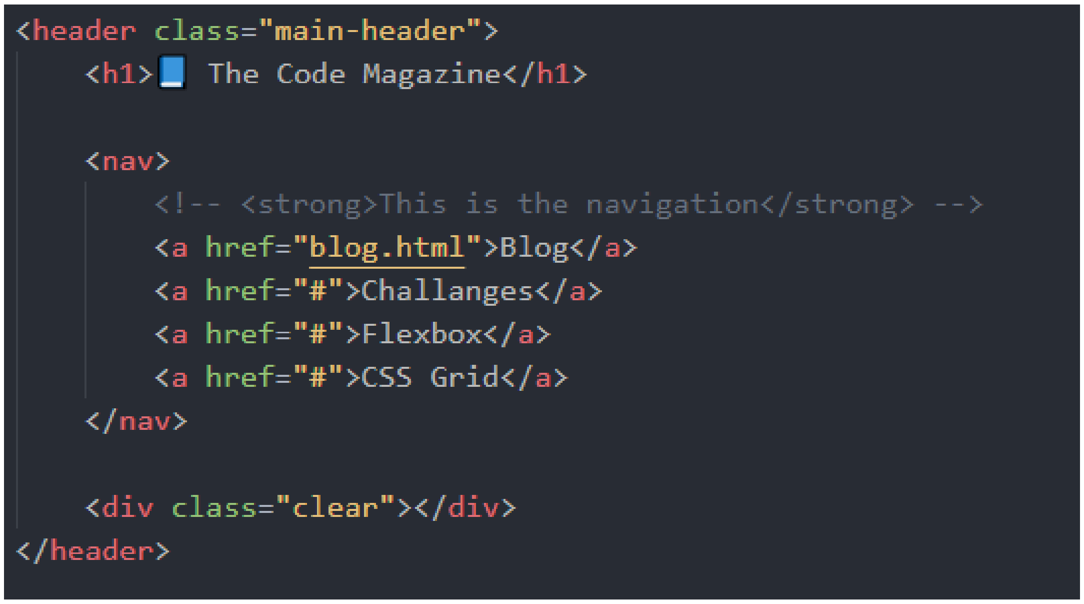
   ```
   a:link {
       color: #1098ad;
       text-decoration: none;
   }
   ```

```
a:visited {
    color: #1098ad;
}

a:hover {
    color: orangered;
    font-weight: bold;
    text-decoration: underline orangered;
}

a:active {
    background-color: black;
    font-style: italic;
}
```

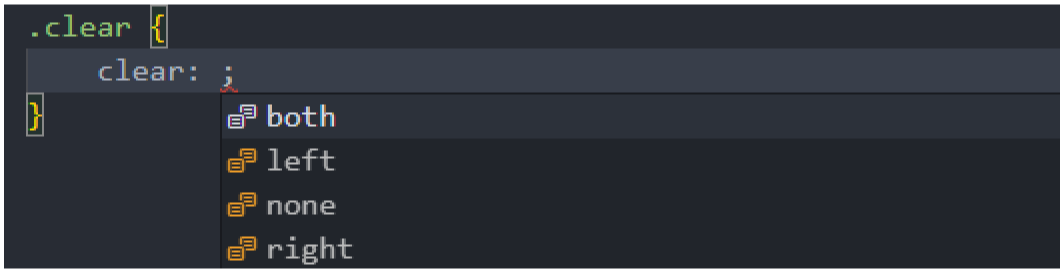
FLOATS

- When we add float attribute to an element, that element will be taken out off the pages flow.
- If you float all elements in a container, there might be problem with containers attributes. For example background color. Because when all elements have float attribute, container act like there is no element in there so it won’t have a height and width.

  ▼ To fix that problem we can put an empty container into the main container and style it with clear attribute, but this isn’t very effective because for each container we have to put an clear container element.



```html
<header class="main-header">
    <h1>📘 The Code Magazine</h1>

    <nav>
        <!-- <strong>This is the navigation</strong> -->
        <a href="blog.html">Blog</a>
        <a href="#">Challanges</a>
        <a href="#">Flexbox</a>
        <a href="#">CSS Grid</a>
    </nav>

    <div class="clear"></div>
</header>
```

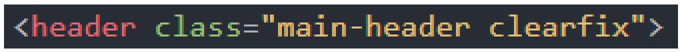


```css
.clear {
    clear: ;
}
```

▼ There is a clear hack for that.

1. We also give main container “clearfix” class

   ▼ Photo

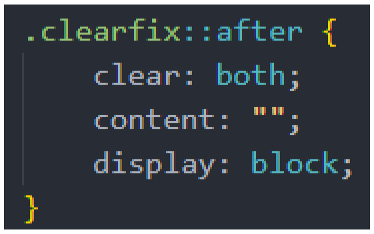
   

   ```html
   <header class="main-header clearfix">
   ```

2. In CSS, with **::after** pseudo element we can create a new element which will be the last child element of the container.

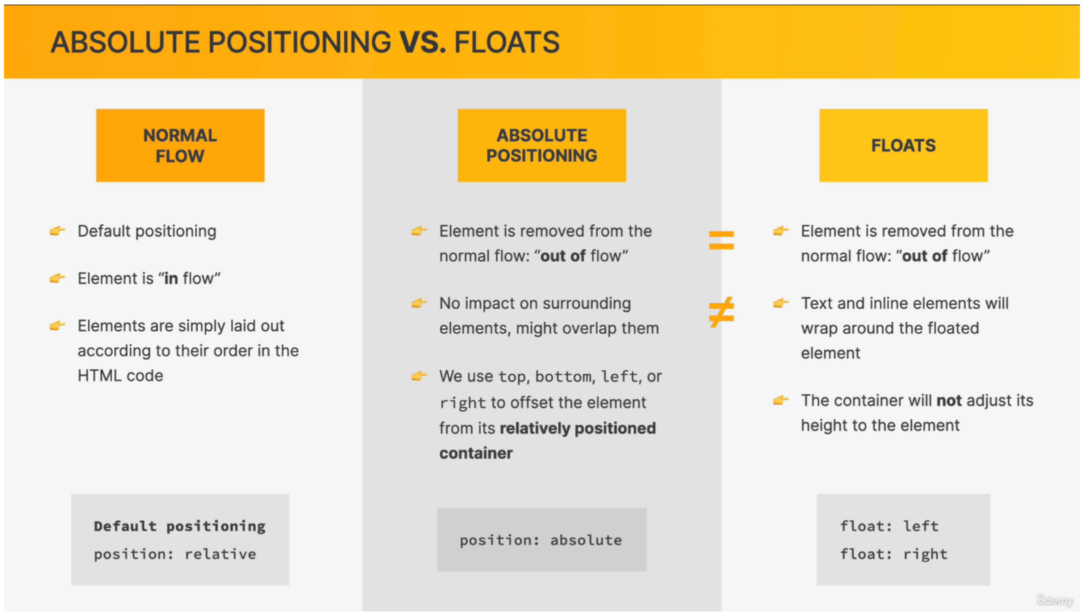
💡 To after pseudo element to **appear**, there must be something defined for content property even if its just an empty text. Pseudo elements like before and after are inline elements, however clearing floats like this only really work on **block** elements.

▼ Example



```
.clearfix::after {
    clear: both;
    content: "";
    display: block;
}
```

▼ Absolute Positioning vs Floats:



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

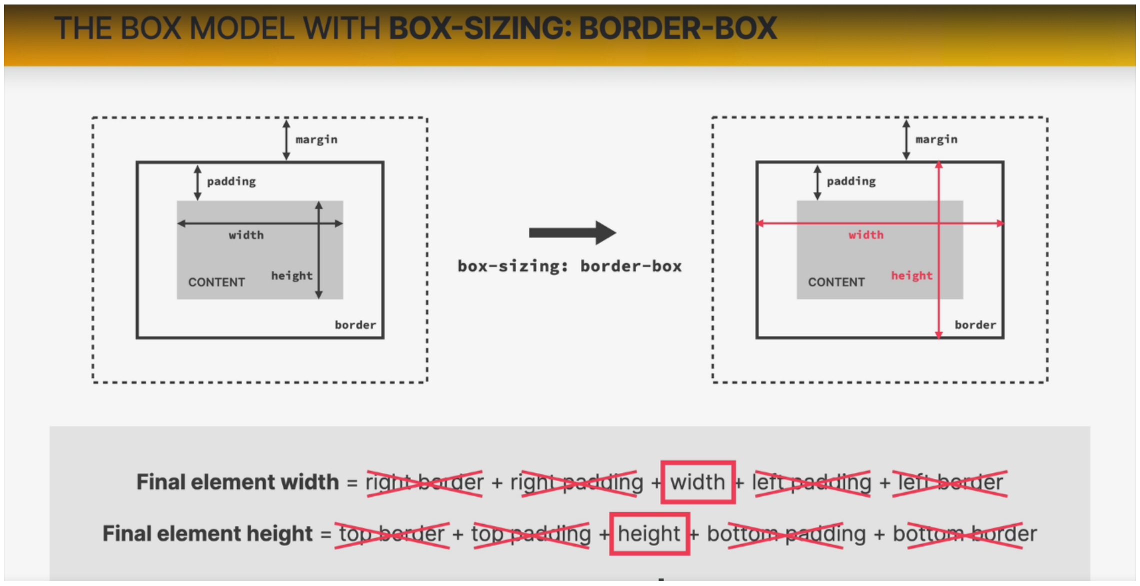
```
float: left
float: right
```

The Box Model With BOX-SIZING: BORDER-BOX

Normally when we add padding to an element, the value of padding will be added to the width and height of the element. But with **box-sizing: border-box** attribute padding will be included in width and height of the element itself.

💡 We always want the box model to work like this by default on every single element. So we put it in the universal selector.

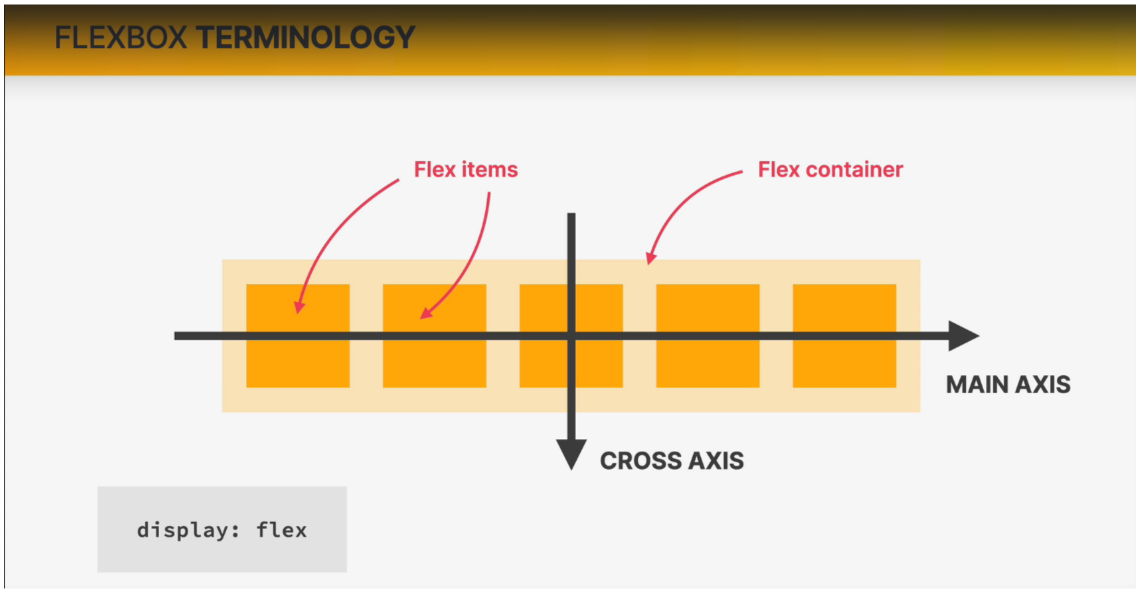
▼ Explanation

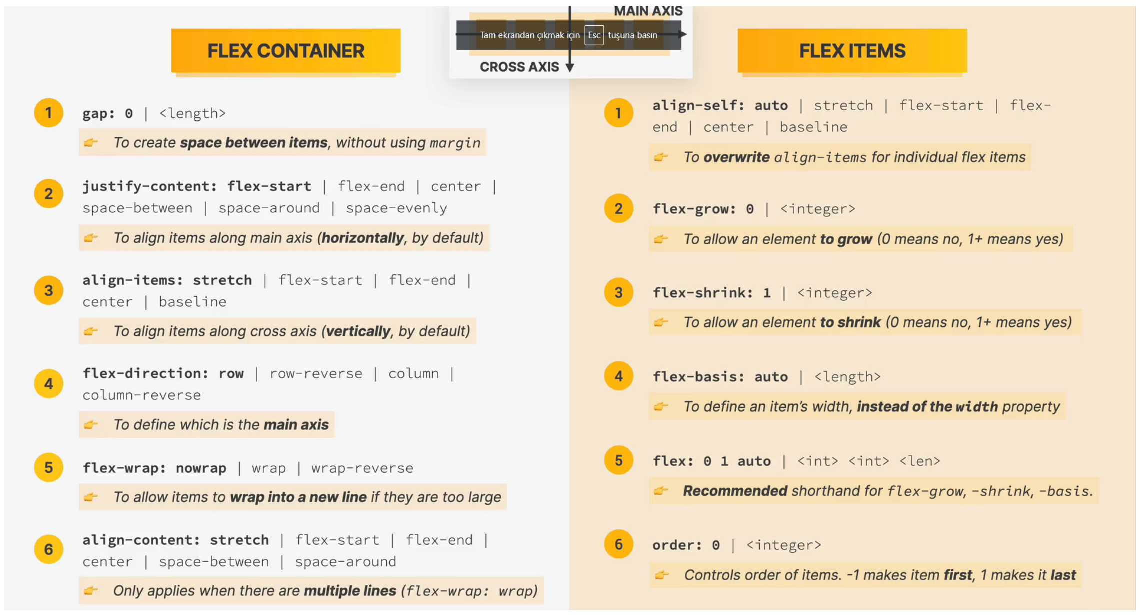

THE BOX MODEL WITH **BOX-SIZING: BORDER-BOX**
margin
padding
width
height
CONTENT
border
box-sizing: border-box
margin
padding
width
height
CONTENT
border
Final element width = ~~right border~~ + ~~right padding~~ + width + ~~left padding~~ + ~~left border~~
Final element height = ~~top border~~ + ~~top padding~~ + height + ~~bottom padding~~ + ~~bottom border~~

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 elements 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 HMTL and CSS code.

▼ Terminology




MAIN AXIS
Tam ekrandan çıkmak için Esc tuşuna basın
CROSS AXIS

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`.
6. `order: 0 | <integer>`
   - Controls order of items. -1 makes item **first**, 1 makes it **last**

- **flex-basis:** So when want to size flex items and in particular with a width, than we usually do not use the width property but instead we use **flex-basis**.
- **flex-shrink:** When we set flex-basis too much sometimes flex items' width won't fit in the container, so this attribute automatically shrinks them evenly.
- **flex-grow:** This attribute allows to fit all flex items evenly in to the container without any spaces. Basically stretches all of the flex items evenly.

💡 The number we input determines the size of the growth. If all flex items have the same input, they will stretch equally. If one of the flex item has 1 and another item has 2 as input, second one will stretch twice as much the first one. Basically it mean that it gets double of the available empty space than the other one.

💡 Never use flex-basis, flex-shrink, flex-grow separately. Flex attribute can do all at once. Example: flex: (flex-grow) (flex-shrink) (flex-basis);

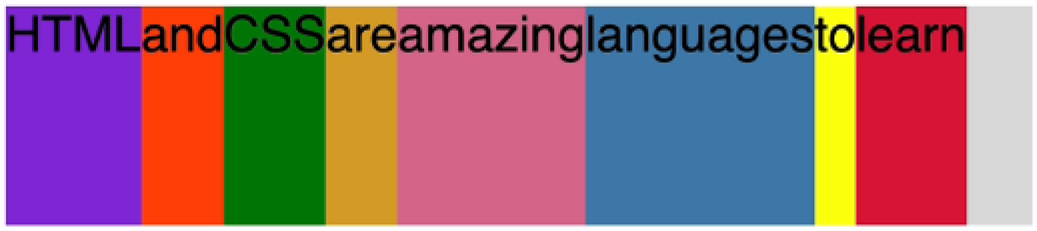
```
flex: 0 0 200px;
```

In order to get started with flexbox is to **use the display property and set it to flex** on some **CONTAINER** element (an element that has a couple child elements).

Elements of a flex container are called flex items.

- Horizontally each of the flex item takes up exactly the space that is necessary for its text content.
- However, vertically flex items are as tall as the tallest element.

▼ Example



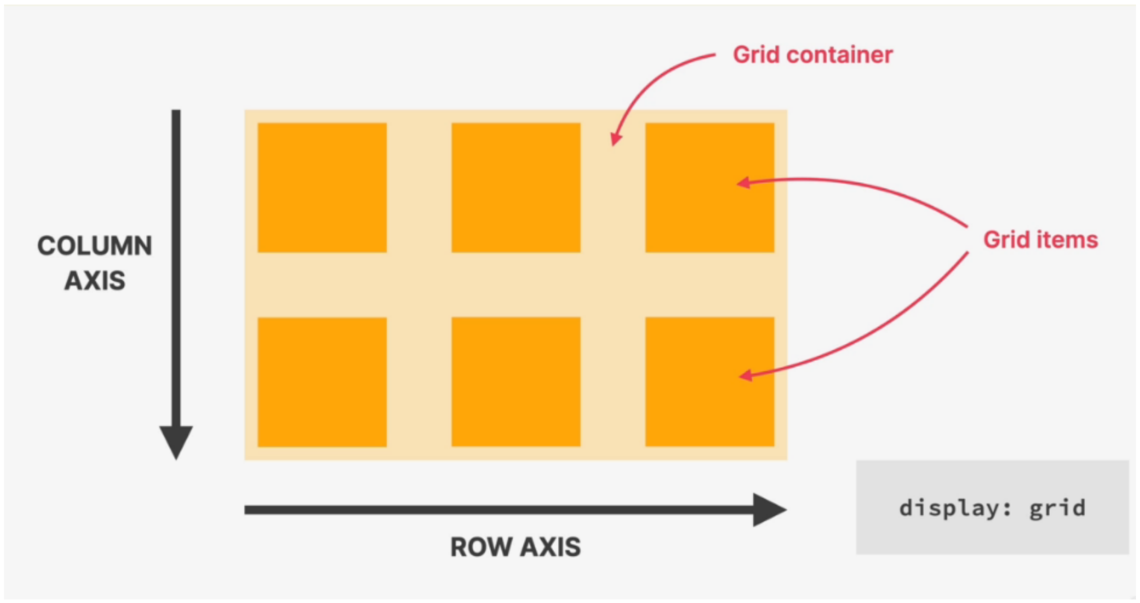
💡 Once we align the items in a flex container, they simply take the space that they need for the content and they also get aligned.

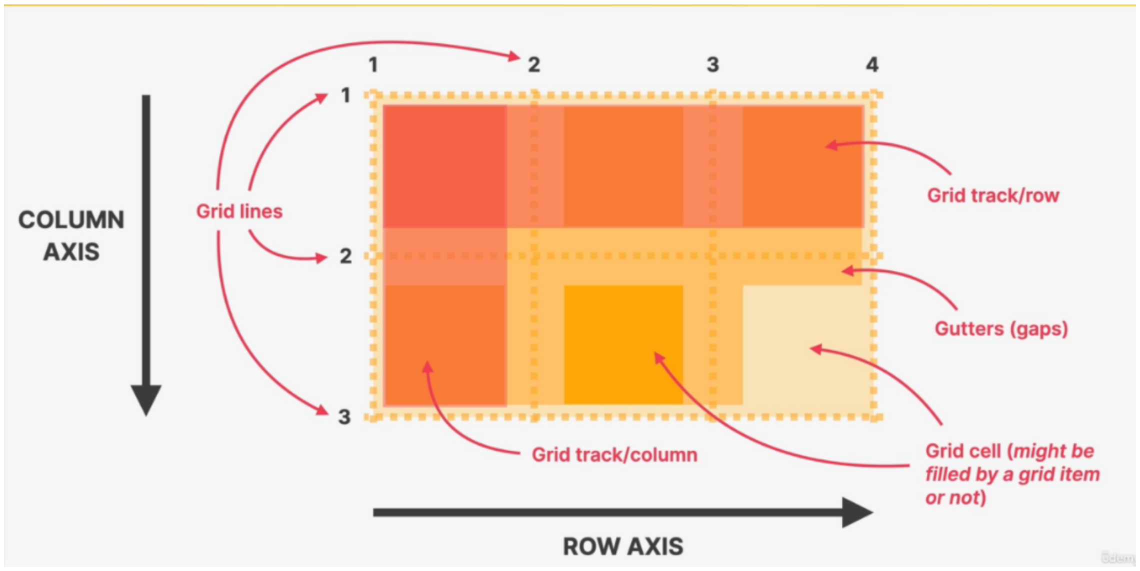
Default align setting on elements is **stretch,** it means that all the element will automatically stretch as tall as the tallest element.

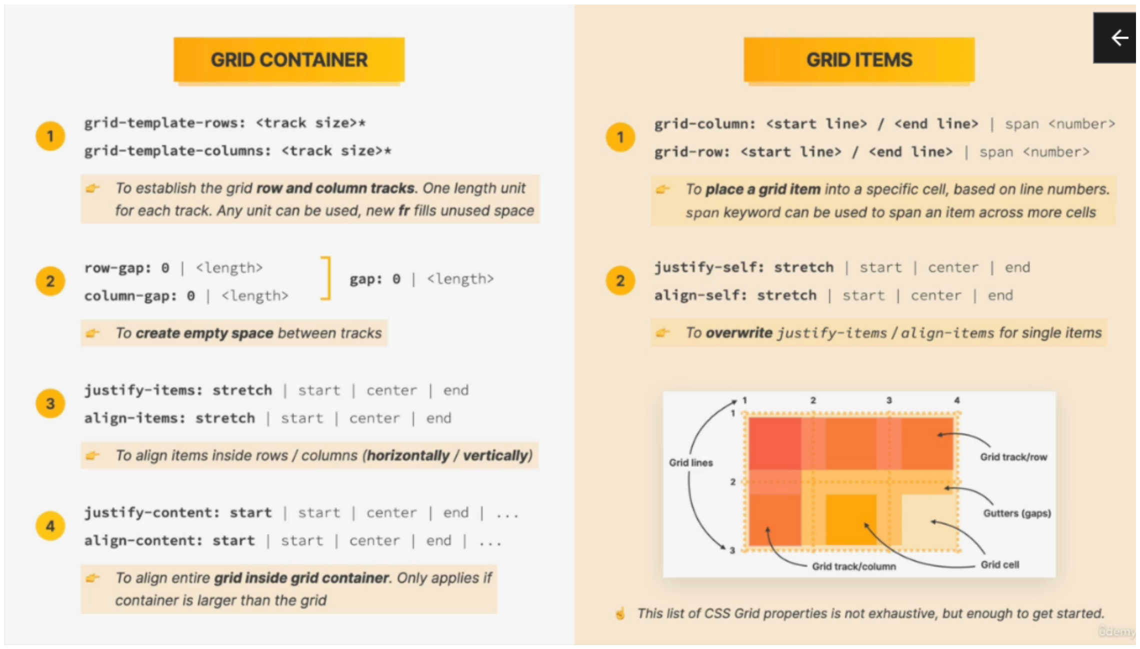
CSS GRID

- It is a set of **CSS properties** for **building 2-dimensional layouts.**
- The main idea behind CSS Grid is that we **divide a container element into rows and columns** that can be filled with its child elements.
- In 2-dimensional contexts, CSS Grid allows us to write less nested HTML and easier-to-read CSS.
- CSS Grid is **not meant to replace flexbox!** Instead, they work perfectly together. Need a **1D** layout? Use flexbox. Need a **2D** layout? Use CSS Grid.
- Like in flexbox we have a **Grid Container** and then **Grid Items**.
- Always use GAP and never use MARGINS

▼ Terminology


Grid container
COLUMN AXIS
Grid items
display: grid
ROW AXIS


1
2
3
4
1
2
3
COLUMN AXIS
Grid lines
Grid track/row
Gutters (gaps)
Grid track/column
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

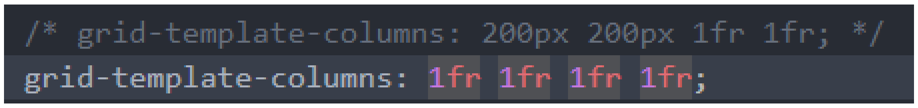
2

```
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.

- **xfr** (x an integer number): This value make grid item to fill the the grid container automatically while resizing the window. x value determines ratio of space, that grid item should take. For example if one item has 1fr and other has 2fr value 2fr one should have twice as much space compared to 1fr one.



```
/* grid-template-columns: 200px 200px 1fr 1fr; */
grid-template-columns: 1fr 1fr 1fr 1fr;
```

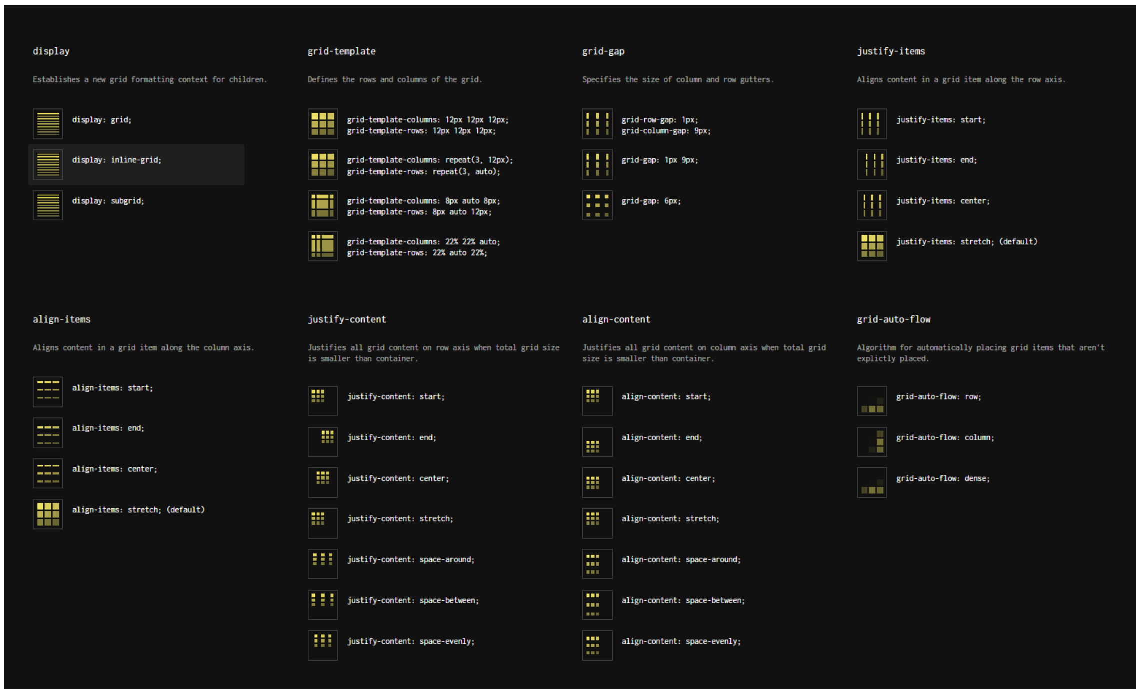
- **auto:** with this value it will only take exactly the size that is necessary to fill its content. And many times in practice, this is actually exactly what we need for one of the columns.
- For repetition there is an attribute that makes our life easier: **grid-template-columns: repeat(***amount of columns***,** *size of the columns***);**
- **Examples for spanning grid cells:**

  ▼ Examples:

  - **grid-column: 1 / 4**; : Grid cell's column starts from grid line 1 and ends at grid line 4

- **grid-column: 1 / span 3;** : Grid cell's column starts from grid line 1 and spans for 3 grid cells
- **grid-column: 1 / -1;** : Grid cell's column starts from grid line 1 and goes to the end of the column.

▼ CHEAT SHEET EVERTHING



```
display

Establishes a new grid formatting context for children.

display: grid;
display: inline-grid;
display: subgrid;
```

```
grid-template

Defines the rows and columns of the grid.

grid-template-columns: 12px 12px 12px;
grid-template-rows: 12px 12px 12px;

grid-template-columns: repeat(3, 12px);
grid-template-rows: repeat(3, auto);

grid-template-columns: 8px auto 8px;
grid-template-rows: 8px auto 12px;

grid-template-columns: 22% 22% auto;
grid-template-rows: 22% auto 22%;
```

```
grid-gap

Specifies the size of column and row gutters.

grid-row-gap: 1px;
grid-column-gap: 9px;

grid-gap: 1px 9px;

grid-gap: 6px;
```

```
justify-items

Aligns content in a grid item along the row axis.

justify-items: start;
justify-items: end;
justify-items: center;
justify-items: stretch; (default)
```

```
align-items

Aligns content in a grid item along the column axis.

align-items: start;
align-items: end;
align-items: center;
align-items: stretch; (default)
```

```
justify-content

Justifies all grid content on row axis when total grid size
is smaller than container.

justify-content: start;
justify-content: end;
justify-content: center;
justify-content: stretch;
justify-content: space-around;
justify-content: space-between;
justify-content: space-evenly;
```

```
align-content

Justifies all grid content on column axis when total grid
size is smaller than container.

align-content: start;
align-content: end;
align-content: center;
align-content: stretch;
align-content: space-around;
align-content: space-between;
align-content: space-evenly;
```

```
grid-auto-flow

Algorithm for automatically placing grid items that aren't
explictly placed.

grid-auto-flow: row;
grid-auto-flow: column;
grid-auto-flow: dense;
```

Instead of grid gap now we use only gap.