# CSS

LET’S MAKE IT LIKE AN ARTIST

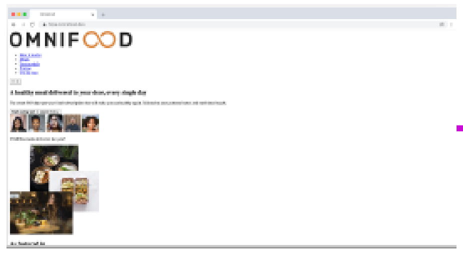
## INTRODUCTION TO CSS

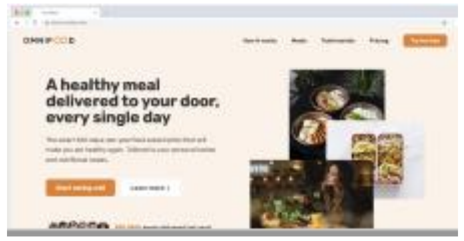
**Cascading Style Sheets**

CSS describes the visual style and presentation of the content written in HTML

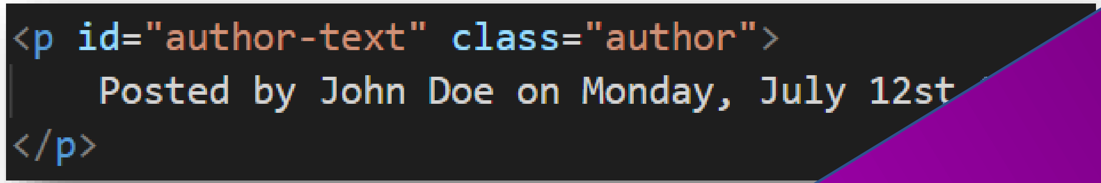
CSS consists of countless properties that developers use to format the content: properties about font, text, spacing, layout, etc.

Web browsers understand HTML and render HTML code as websites



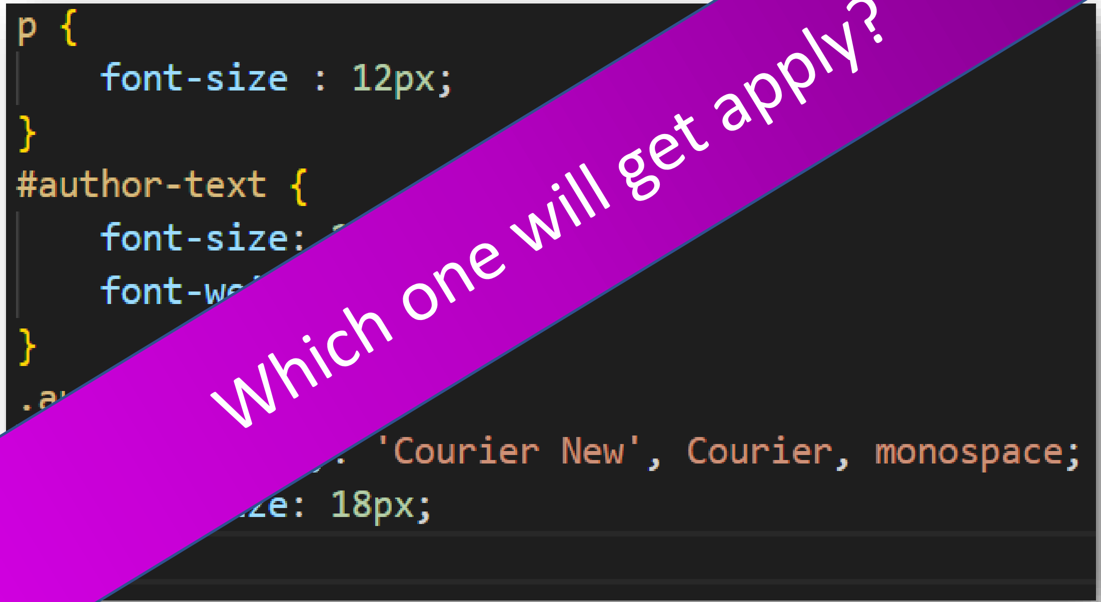

A healthy meal delivered to your door, every single day

# CONFLICTING SELECTORS AND DECLARATIONS



```
<p id="author-text" class="author">
    Posted by John Doe on Monday, July 12st
</p>
```

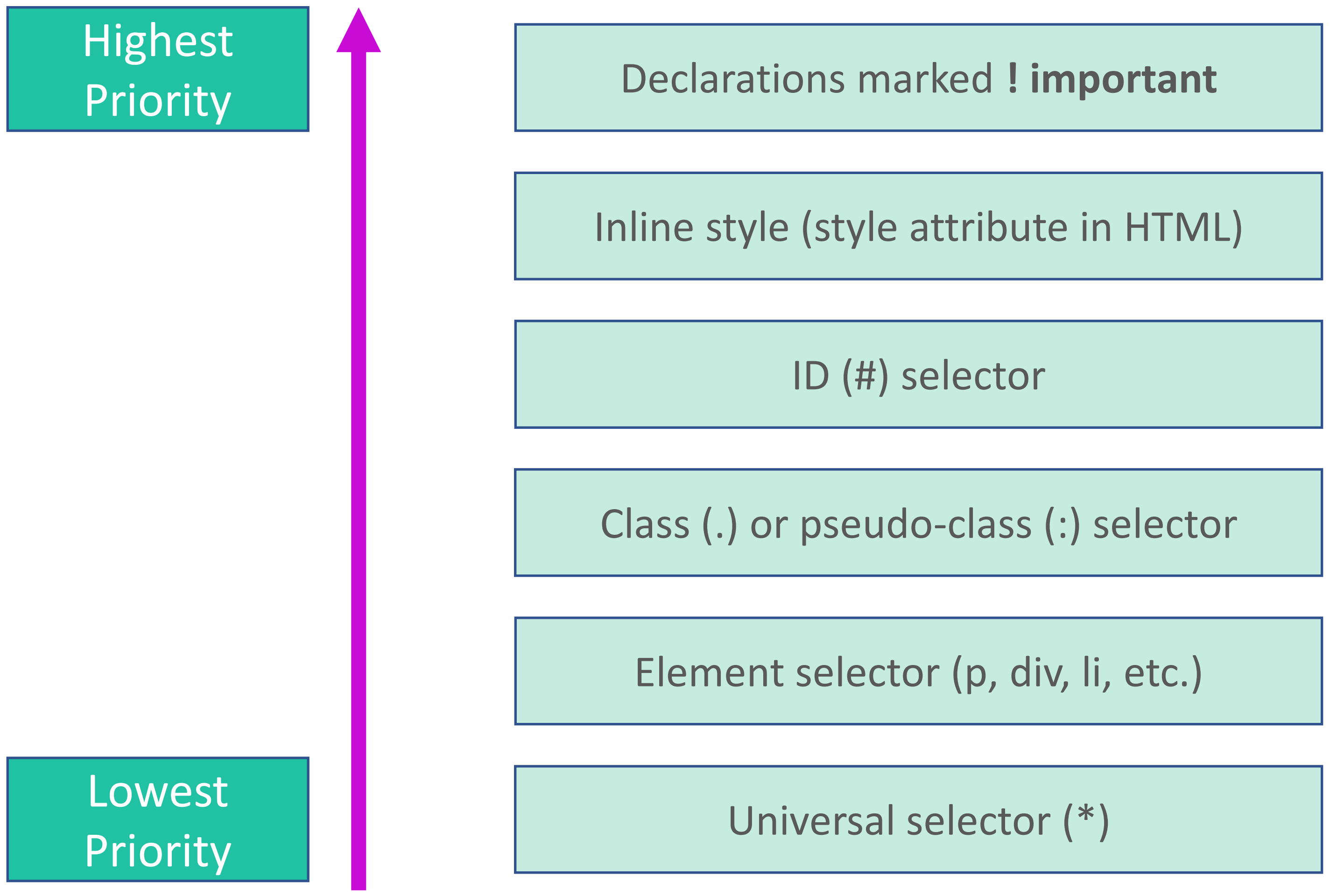
Multiple Selectors



```
p {
    font-size : 12px;
}
#author-text {
    font-size:
    font-we
}
.a
          : 'Courier New', Courier, monospace;
       ze: 18px;
```

Which one will get apply?

# RESOLVING CONFLICTING DECLARATIONS


Highest Priority
Lowest Priority
Declarations marked **! important**
Inline style (style attribute in HTML)
ID (#) selector
Class (.) or pseudo-class (:) selector
Element selector (p, div, li, etc.)
Universal selector (*)

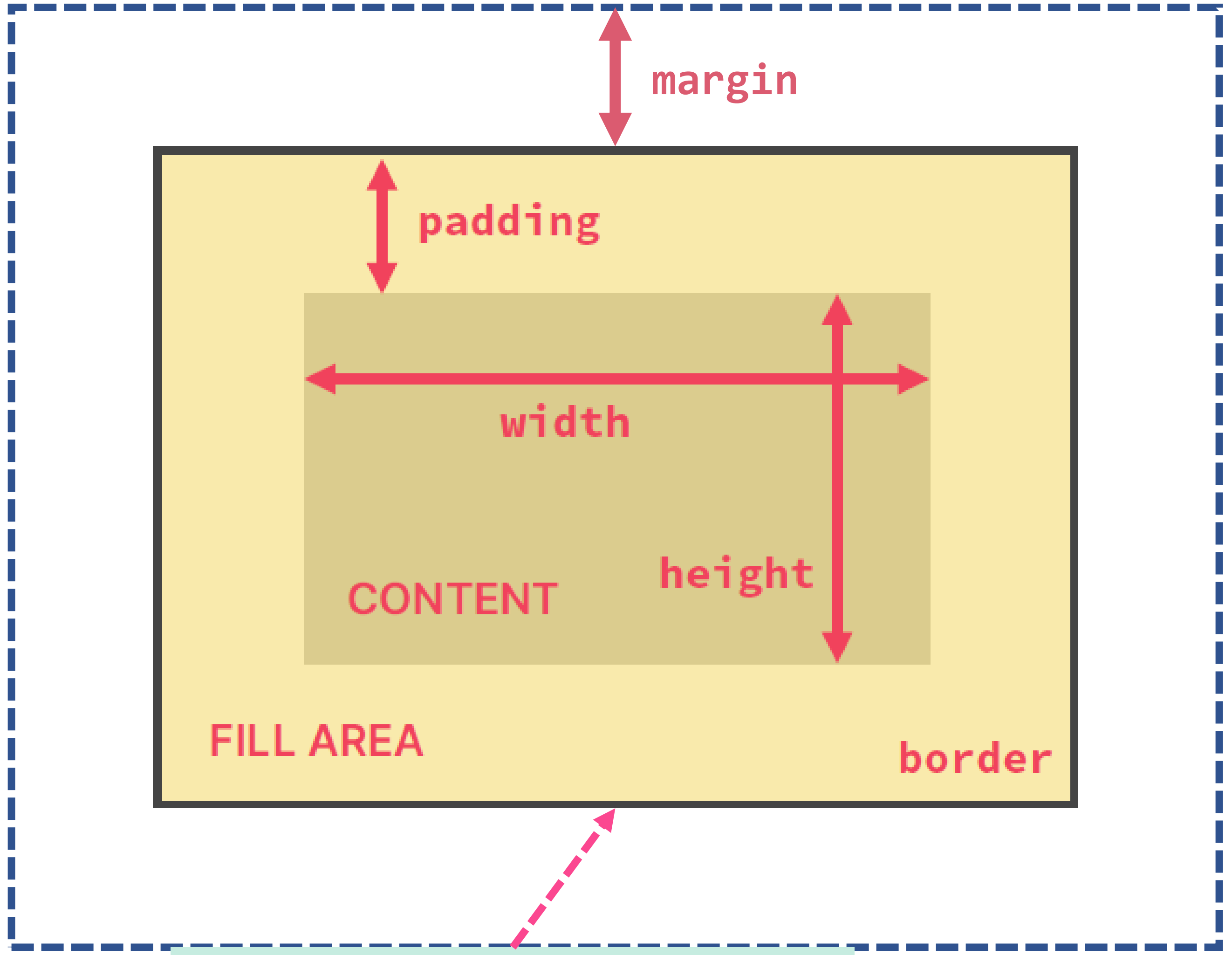
# CSS : Inheritance

**In CSS, inheritance** controls what happens when no value is specified for a property on an element.

CSS properties can be categorized in two types:

| Inherited properties | Non-inherited properties |
|---|---|
| By default, are set to the computed value of the parent element | By default, are set to initial value of the property |

# THE CSS BOX MODEL



Visible part of element on the page

**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

**Fill area**: Area that gets filled with **background color** or **background image**

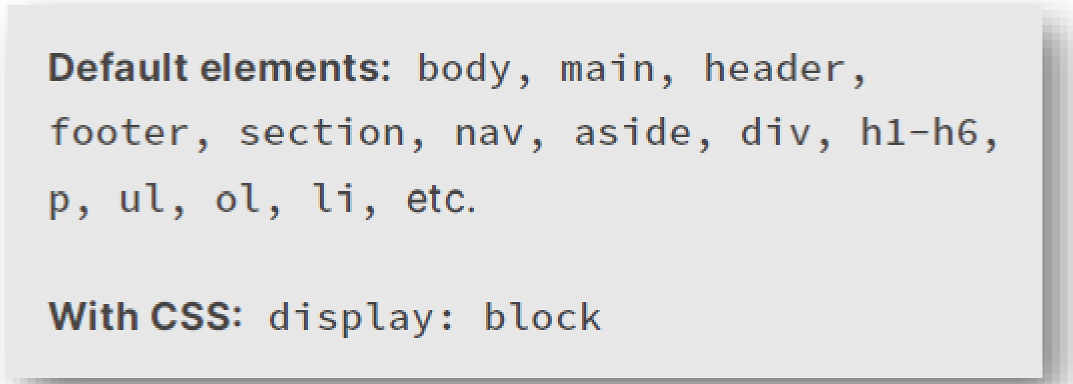
## BLOCK-LEVEL ELEMENTS

Elements are formatted visually as blocks

Elements occupy 100% of parent element's width, no matter the content

Elements are stacked vertically by default, one after another

The box-model applies as showed earlier



**Default elements:** `body, main, header, footer, section, nav, aside, div, h1-h6, p, ul, ol, li,` etc.

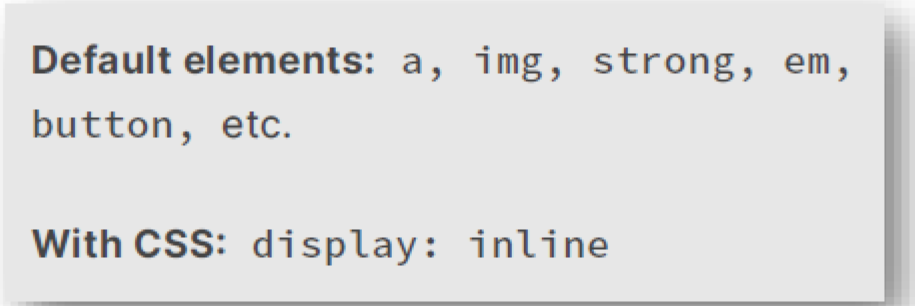
**With CSS:** `display: block`

# INLINE ELEMENTS

Occupies only the space necessary for its content

Causes no line-breaks after or before the element

Paddings and margins are applied only horizontally (left and right)



**Default elements:** `a, img, strong, em, button,` etc.

**With CSS:** `display: inline`

# ANIMATION

Animate transitions from one CSS style configuration to another

Animations consist of two components, a style describing the CSS animation and a set of keyframes that indicate the start and end states of the animation's style, as well as possible intermediate waypoints

@keyframes at-rule defines the appearance of the animation

keyframes use a <percentage> to indicate the time during the animation sequence at which they take place. 0% indicates the first moment of the animation sequence, while 100% indicates the final state of the animation

## ANIMATION PROPERTIES

| Property Name | Description |
|---|---|
| **animation-name** | Specifies the name of the @keyframes at-rule describing the animation's keyframes |
| **animation-duration** | Configures the length of time that an animation should take to complete one cycle |
| **animation-delay** | Configures the delay between the time the element is loaded and the beginning of the animation sequence |
| **animation-iteration-count** | Configures the number of times the animation should repeat; you can specify infinite to repeat the animation indefinitely |
| **animation-direction** | Configures whether or not the animation should alternate direction on each run through the sequence or reset to the start point and repeat itself |
| **animation-fill-mode** | Configures what values are applied by the animation before and after it is executing |

## NORMAL FLOW & ABSOLUTE POSITIONING

| NORMAL FLOW | ABSOLUTE POSITIONING |
|---|---|
| Default positioning | Element is removed from the normal flow: “out of flow” |
| Element is “in flow” | No impact on surrounding elements, might overlap them |
| Elements are simply laid out according to their order in the HTML code | We use top, bottom, left, or right to offset the element from its relatively positioned container |

position: relative;

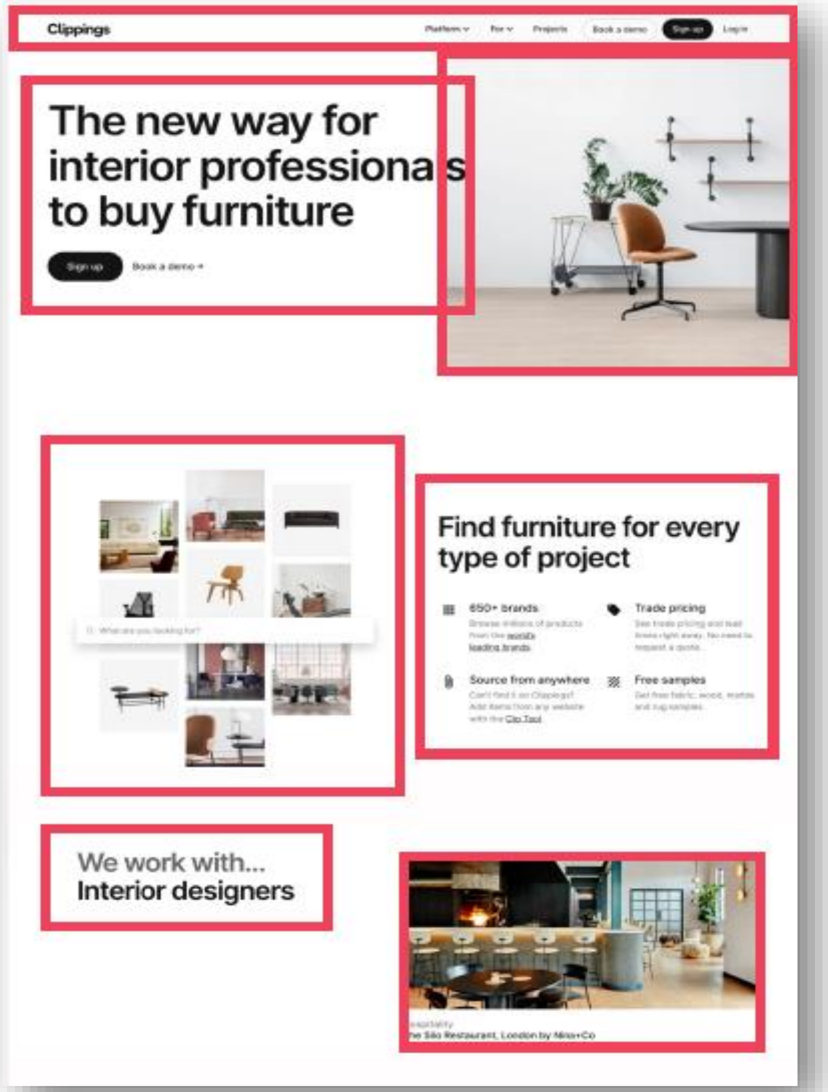
position: absolute;

# WHAT DOES “LAYOUT” MEAN?

Layout is the way text, images and other content is placed and arranged on a webpage

Layout gives the page a visual structure, into which we place our content

**Building a layout:** arranging page elements into a visual structure, instead of simply having them placed one after another (normal flow)



The new way for interior professionals to buy furniture

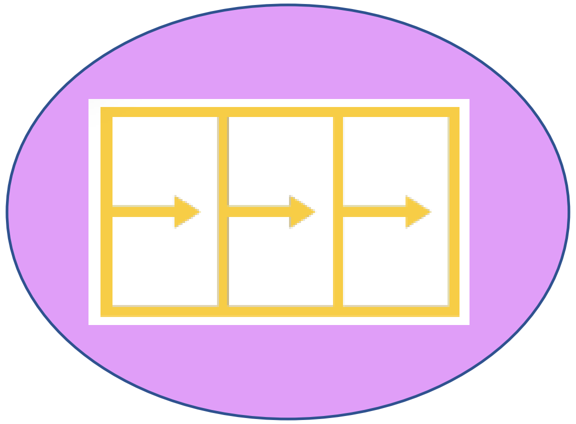
Find furniture for every type of project

We work with...
Interior designers

## THE 3 WAYS OF BUILDING LAYOUTS WITH CSS

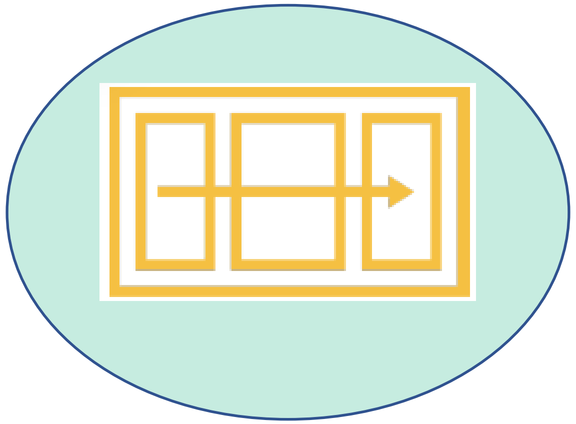
**FLOAT LAYOUTS**

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



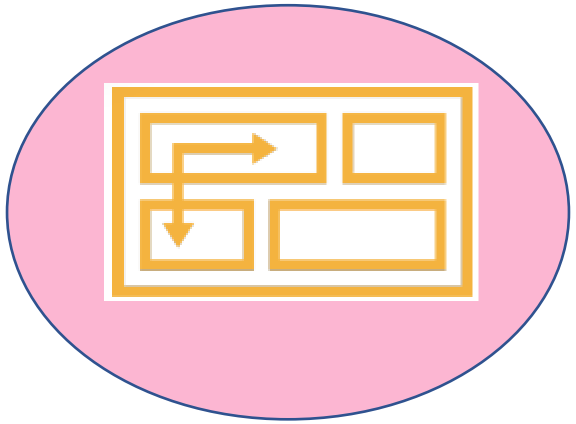
**FLEXBOX**

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



**CSS GRID**

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



## WHAT IS FLEXBOX?

Flexbox is a set of related CSS properties for building 1-dimensional layouts

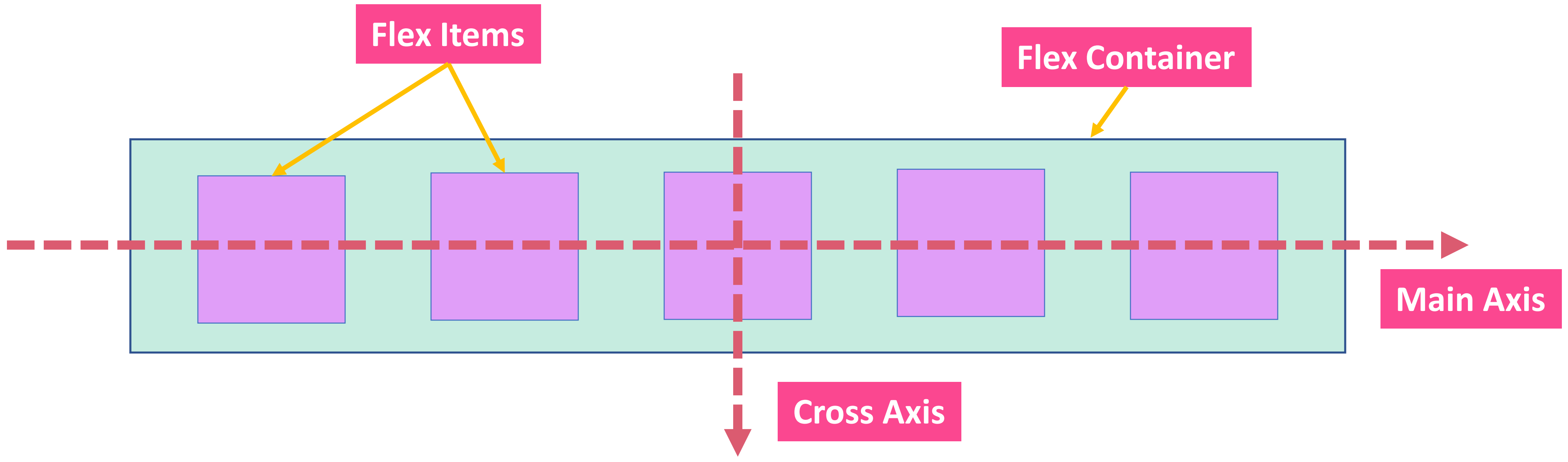
The main idea behind flexbox is that empty space inside a container element can be automatically divided by its child elements

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 PROPERTIES

| Property Name | Description | Possible Values |
|---|---|---|
| **gap** | To create **space between items**, without using margin | **0** \| <length> |
| **justify-content** | To align items along main axis (**horizontally**, by default) | flex-start \| flex-end \| center \| space-between \| space-around \| space-evenly |
| **align-items** | To align items along cross axis (**vertically**, by default) | stretch \| flex-start \| flex-end \| center \| baseline |
| **flex-direction** | To define which is the **main axis** | row \| row-reverse \| column \| column-reverse |
| **flex-wrap** | To allow items to **wrap into a new line** if they are too large | nowrap \| wrap \| wrap-reverse |
| **align-content** | Only applies when there are **multiple lines** (flex-wrap: wrap) | stretch \| flex-start \| flex-end \| center \| space-between \| space-around |

# FLEX ITEM PROPERTIES

| Property Name | Description | Possible Values |
|---|---|---|
| **align-self** | To **overwrite** align-items for individual flex items | **auto** \| stretch \| flex-start \| flex-end \| center \| baseline |
| **flex-grow** | To allow an element **to grow** (0 means no, 1+ means yes) | **0** \| <integer> |
| **flex-shrink** | To allow an element **to shrink** (0 means no, 1+ means yes) | **1** \| <integer> |
| **flex-basis** | To define an item's width, **instead of the width** property | **auto** \| <length> |
| **flex** | **Recommended** shorthand for flex-grow, -shrink, -basis. | **0 1 auto** \| <int> <int> <len> |
| **order** | Controls order of items. -1 makes item **first**, 1 makes it **last** | **0** \| <integer> |

## WHAT IS CSS GRID?

CSS Grid 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

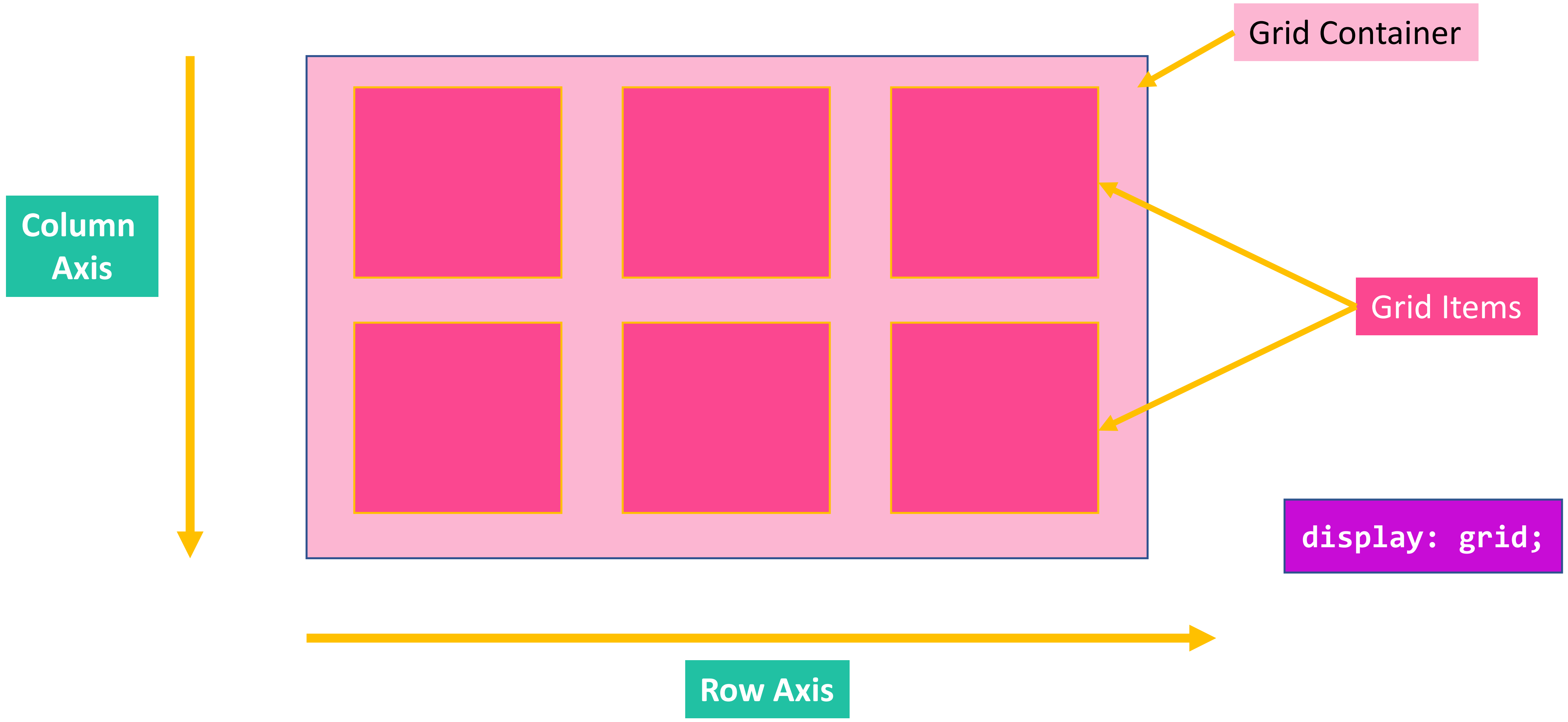
In two-dimensional contexts, CSS Grid allows us to write less nested HTML and easier-to-read CSS

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*.

# BASIC CSS GRID TERMINOLOGY



# GRID CONTAINER PROPERTIES

| Property Name | Description | Possible Values |
|---|---|---|
| **grid-template-rows**<br>**grid-template-columns** | To establish the grid row and column tracks. One length unit for each track. Any unit can be used, new **fr** fills unused space | <track size>* |
| **row-gap**<br>**column-gap** | To create empty space between tracks | **0** \| <length> |
| **justify-items**<br>**align-items** | To align items inside rows / columns (horizontally / vertically) | stretch \| start \| center \| end |
| **justify-content**<br>**align-content** | To align entire grid inside grid container. Only applies if container is larger than the grid | start \| center \| end |

## GRID ITEM PROPERTIES

| Property Name | Description | Possible Values |
|---|---|---|
| **grid-column**<br>**grid-row** | 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 | <start line> / <end line> \| span <number> |
| **justify-self**<br>**align-self** | To overwrite justify-items / align-items for single items | stretch \| start \| center \| end |

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

## WHAT IS RESPONSIVE DESIGN?

Design technique to make a webpage adjust its layout and visual style to **any possible screen size** (window or viewport size)

In practice, this means that responsive design makes websites usable on all devices, such as **desktop computers, tablets, and mobile phones**.

It’s a set of practices, **not a separate technology**. It’s all just CSS!

# RESPONSIVE DESIGN INGREDIENTS

## FLUID LAYOUTS

To allow webpage to adapt to the current viewport width (or even height)

Use **%** (or vh / vw) unit instead of px for elements that should adapt to viewport (usually layout)

Use **max-width** instead of width

## RESPONSIVE UNITS

Use **rem** unit instead of px for most lengths to make it easy to scale the entire layout down (or up) automatically

**Helpful trick:** setting 1rem to 10px for easy calculations

## FLEXIBLE IMAGES

By default, images don't scale automatically as we change the viewport, so we need to fix that

Always use % for image dimensions, together with the **max-width** property

Use max-width for responsiveness

## MEDIA QUERIES

Bring responsive sites to life!

To change CSS styles on certain viewport widths (called breakpoints)

Use media queries and select breakpoints

# DESKTOP-FIRST VS. MOBILE-FIRST DEVELOPMENT

## DESKTOP-FIRST

Start writing CSS for the desktop: large screen

Then, media queries shrink design to smaller screens.

## MOBILE-FIRST

Start writing CSS for mobile devices: small screen

Then, media queries expand design to a large screen

Forces us to reduce websites and apps to the absolute essentials.

## BLUEPRINT

A lightweight layout library for building great responsive mobile first UIs that work everywhere. Open Source, built with CSS Grid and Flexbox.

# FOUNDATION

The most advanced responsive front-end framework

# MICRODATA

Microdata is used to nest metadata within existing content on web pages.

Search engines and web crawlers can extract and process microdata from a web page and use it to provide a richer browsing experience for users.

Microdata allows search engines to understand the information on web pages and provide more relevant results to users.

Microdata consists of a group of name-value pairs -

- To create an item, the Itemscope attribute is used.

- To add a property to an item, the itemprop attribute is used on one of the item's descendants.

Google and other major search engines support the *schema.org* vocabulary for structured data

## REFERENCES

**READING MATERIAL**

- https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps

**VIDEO LINKS**

- https://www.youtube.com/playlist?list=PLu0W_9lII9agiCUZYRsvtGTXdxkzPyItg
- https://www.youtube.com/watch?v=1Rs2ND1ryYc&t=2s