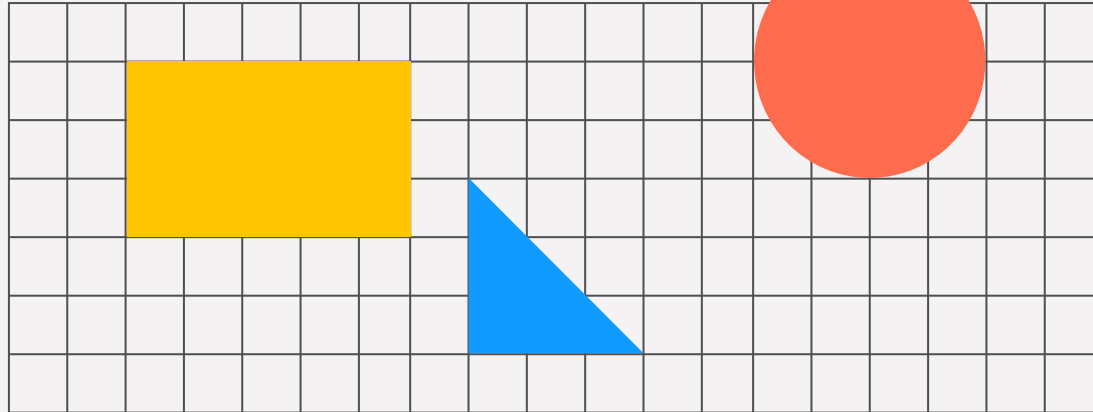
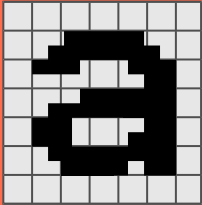


Pages That Look Elegant

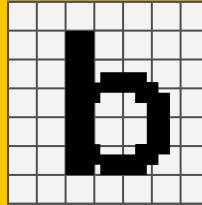
Ali Abrishami



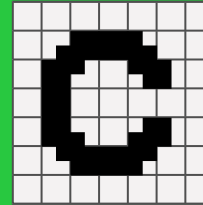
In This Lecture You Will...



Learn What CSS is



**Use CSS to style
your HTML
documents**



**Make beautiful
web pages!**

REMINDER: The Three Layers of a Website



Structure

What the **content** is

Style

How it **looks**

Behaviour

How it **reacts**

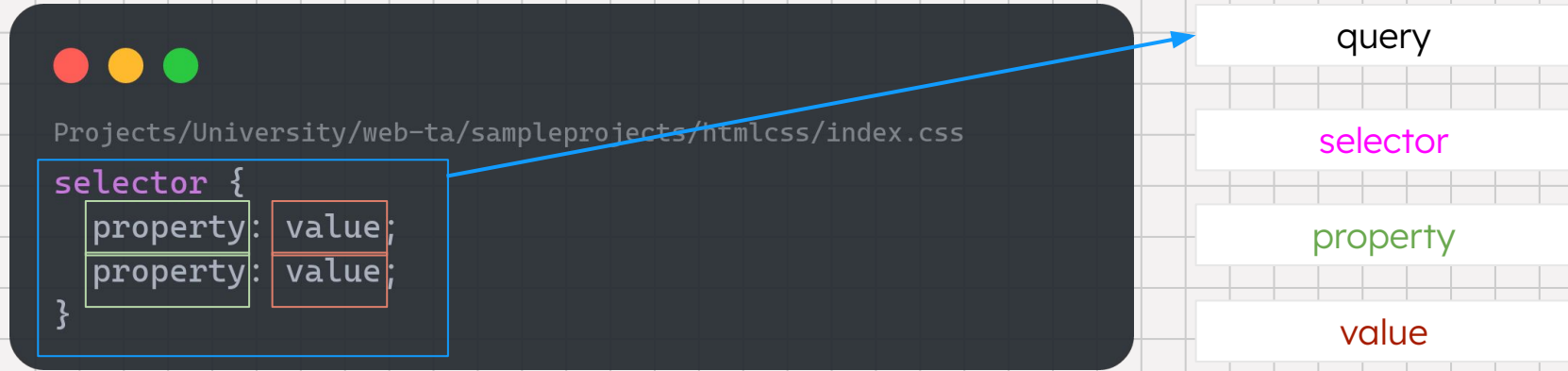
What is CSS?

Introduction

- CSS stands for **Cascading Style Sheets**.
- It is used to style HTML elements.
- CSS separates content (HTML) from presentation (design).
- Styles can be added inline, within a `<style>` tag, or via external files.
- It helps create responsive, modern, and user-friendly designs.



How CSS Looks Like



- CSS is written as rules with a **selector** and a **declaration block**.
- Each rule tells the browser how to style specific HTML elements.
- A declaration block contains one or more property-value pairs.

CSS Selectors



- `element` → Targets all elements of a type (p, div, etc.)
- `.class` → Targets elements with a specific class
- `#id` → Targets a specific element with an ID
- `*` → Targets all elements
- `A, B` → Targets both A and B elements
- `A B` → Targets B inside A (descendant)
- `A > B` → Targets B directly inside A (child)

Pseudo Selectors



- `:hover` → When the mouse is over the element. e.g. `button:hover`
- `:focus` → When the element is focused. e.g. `input:focus`
- `:first-child` → First child of its parent. e.g. `li:first-child`
- `:nth-child(2n)` → Every even element. e.g. `tr:nth-child(2n)`
- `:not(.active)` → All except those with `.active` class

Attribute Selectors



- `[type]` → Elements with the `type` attribute. e.g. `input[type]`
- `[type="text"]` → Elements with `type="text"`. e.g. `input[type="text"]`
- `[href^="https"]` → `href` starting with "https"
- `[href$=".pdf"]` → `href` ending with ".pdf".
- `[data-role*="btn"]` → `data-role` containing "btn"

CSS Properties



- **Color & Background:** e.g. `color`, `background-color`
- **Text & Fonts:** e.g. `font-size`, `font-weight`, `text-align`
- **Box Model:** e.g. `margin`, `padding`, `border`
- **Size & Positioning:** e.g. `width`, `height`, `position`
- **Display & Layout:** e.g. `display`, `flex`, `grid`

Include CSS in HTML



- CSS can be applied to HTML in three main ways:
 - **Inline** CSS: styling directly inside an element
 - **Internal** CSS: styling inside `<style>` in the HTML `<head>`
 - **External** CSS: linking a separate `.css` file with `<link>`
- Each method has different use cases and priority.

Method I: Inline CSS



- CSS is added directly to an element's style attribute
- Example: `<p style="color:red;">Hello</p>`
- Affects only that element
- Highest priority in the cascade (more on that later!)
- Useful for quick tests, **not** for large projects

Method II: Internal `<style>`

- CSS is written inside a `<style>` block in the `<head>`
- Styles apply to the whole page
- Easier to maintain than inline styles
- Best for **single-page** documents

```
Projects/University/web-ta/sampleprojects/htmlcss/index.html
<!Doctype HTML>
<html>

  <head>
    <style>
      h1 {
        color: red;
      }
    </style>
  </head>

  <body>
    <h1>Hello! </h1>
  </body>

</html>
```

Method III: External `<link>`

- CSS stored in a separate `.css` file
- Linked using
`<link rel="stylesheet" href="style.css">`
- Styles apply to multiple pages at once
- Encourages **clean** structure and **reusability**
- Best practice for larger projects

```
Projects/University/web-ta/sampleprojects/htmlcss/index.html
<!Doctype HTML>
<html>
  <head>
    <link rel="stylesheet" href="index.css">
  </head>


  <body>
    <h1>Hello!</h1>
  </body>
</html>
```

```
Projects/University/web-ta/sampleprojects/htmlcss/index.css
h1 {
  color: red;
}
```

Text-related Properties



- **font-family** sets the typeface (e.g., Arial, Times).
- **font-size** controls text size (e.g., 16px, 1.2em).
- **font-weight** sets thickness (normal, bold, 100–900).
- **text-align** aligns text (left, center, right, justify).
- **line-height** controls spacing between lines of text.

<div>CSS Font Property </div>		
Font	Syntax	Example
font color	font-color: Blue	A
font size	font-size: 48px;	A
font weight	font-weight: lighter;	A
font family	font-family: Noto Sans	A
font style	font_style: italic	A
font spacing	letter-spacing: 5px	A B

Source: TutorialBrain

Color-related Properties



- **color**: sets text color.
- **background-color**: sets element's background fill.
- **Background-image**: sets elements' background image
- Both accept named colors, HEX, RGB/RGBA, HSL/HSLA.
- color is usually inherited; background-color defaults to transparent.
- Use high contrast for readability and accessibility.

Ready to apply?

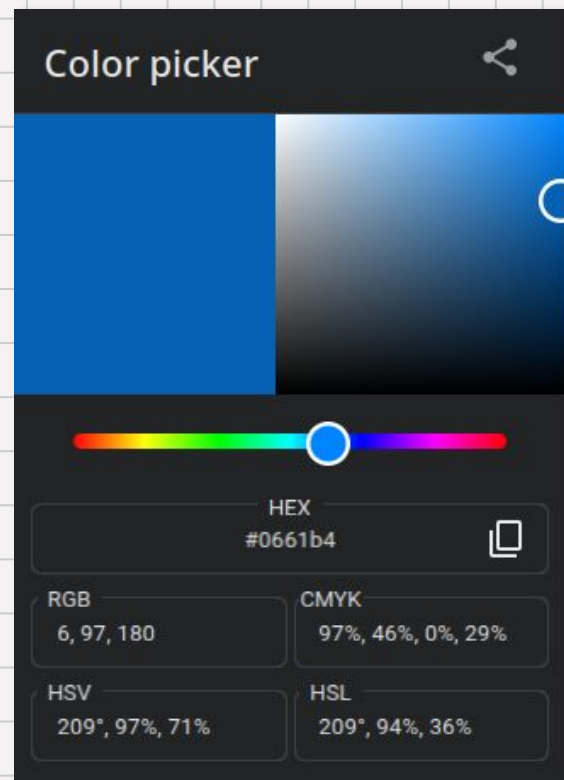
We are looking for leaders eager to share their knowledge with the design community.

```
div {  
  background-color: #F8DB46;  
}
```

Source: Uxcel

Colors in CSS

- CSS supports many different color systems
- **RGB**: Red, Green, Blue. e.g. (6, 97, 180)
- **HEX**: Same as RGB, but in hexadecimal. e.g. #0661b4
- **HSV/HSB**: Hue, Saturation, Value/Brightness. e.g. 209, 97%, 71%
- **HSL**: Hue, Saturation, Lightness. e.g. 209, 94% 36%



CSS Measurement Units



- Many CSS properties take "length" values. Hence, CSS supports many measurement units.
- Absolute Units:
 - `cm`, `mm`, `in`: real-life measurement units. (absolute units)
 - `px` (pixels, 1/96 in), `pt` (points, 1/72 inch), `pc` (picas, 1/6 pt). (absolute units)
- Relative Units:
 - `em` (relative to the font-size of the element), `rem` (relative to font-size of the root element)
 - `vw` (relative to 1% of viewport width), `vh` (relative to 1% of viewport height)
 - `%` (relative to the parent element)

CSS Selector Priority



- Inline styles (e.g. `style="color:red"`) have the highest priority
- ID selectors (`#id`) are stronger than class or element selectors
- Class, pseudo-class, attribute selectors (`.class`, `:hover`, `[type=text]`) come next
- Element and pseudo-element selectors (`div`, `p`, `::before`) have the lowest priority
- **NOTE:** When specificity is equal, the last rule declared wins
- The `!important` rule is used to give the value of a specific property the highest priority.

Class Activity Time

In each example, what color will the text in the body be?



Projects/University/web-ta/sampleprojects/htmlcss/index.html

```

<!DOCTYPE html>
<html>
<head>
  <style>
    p { color: blue; }
    .highlight { color: green; }
    #special { color: orange; }
  </style>
</head>
<body>
  <p id="special" class="highlight" style="color:red;">
    CSS Priority Example
  </p>
</body>
</html>

```



Projects/University/web-ta/sampleprojects/htmlcss/index.html

```

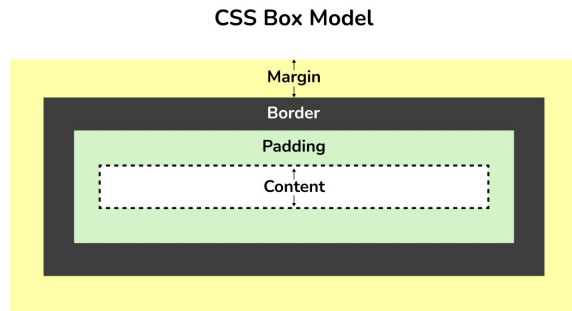
<!DOCTYPE html>
<html>
<head>
  <style>
    h1 { color: black; }
    .title { color: purple; }
    #mainTitle { color: blue; }
  </style>
</head>
<body>
  <h1 id="mainTitle" class="title" style="color: red;">
    Selector Priority Demo
  </h1>
</body>
</html>

```

CSS Box Model



- **margin**: space outside the element's border.
- **border**: line surrounding the padding and content.
- **padding**: space inside the border, around content.
- **width / height**: size of the content area (excluding padding/border unless box-sizing: border-box).
- Box Model order
→ content → padding → border → margin.

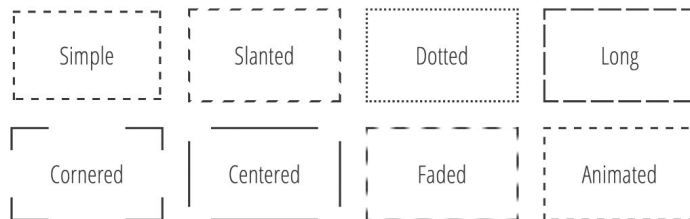


Source: LambdaTest

Outlines and Borders



- `border`: shorthand to set `width`, `style`, and `color`.
- Border styles: `solid`, `dashed`, `dotted`, `double`, etc.
- `border-radius`: rounds corners (50% for circles).
- `outline`: line drawn outside the border, not affecting layout.
- Outline vs Border outline doesn't take up space, border does.



Source: CSS Tricks

Class Activity Time

What does each one of these CSS properties do?

`padding-left`

`margin-right`

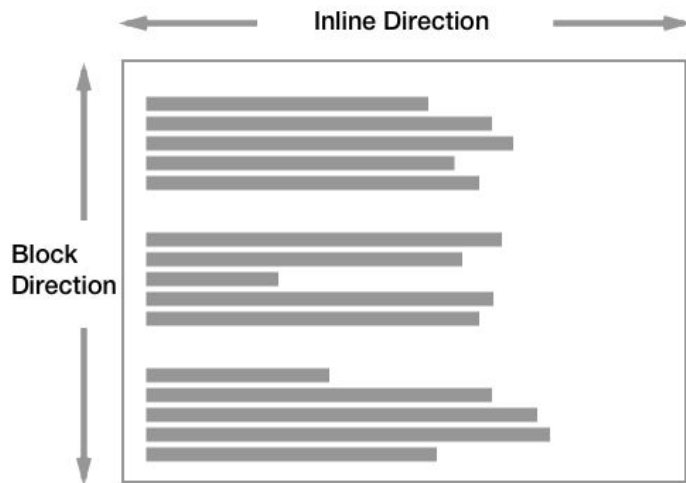
`padding-top`

`margin-bottom`

CSS Display & Layout



- **display**: defines element type (**block**, **inline**, **flex**, **grid**, etc.).
- **inline** vs **block**: **inline** flows with text, **block** starts on a new line.
- **flex**: 1D layout system for arranging items in rows or columns.
- **grid**: 2D layout system for rows and columns.
- **visibility** & **overflow**: control element visibility and content clipping.



Source: MDN Docs

CSS Layout - Normal Flow



- Elements on a webpage lay out in normal flow **by default**.
- The process begins as the boxes of individual elements are laid out in such a way that any padding, border, or margin they happen to have is added to their content.
- By default, a block-level element's content fills the available inline space of the parent element.
 - `p.ex1 {display: none;}`
 - `p.ex2 {display: inline;}`
 - `p.ex3 {display: block;}`
 - `p.ex4 {display: inline-block;}`

Basic document flow

I am a basic block level element. My adjacent block level elements sit on new lines below me.

By default we span 100% of the width of our parent element, and we are as tall as our child content. Our total width and height is our content + padding + border width/height.

We are separated by our margins. Because of margin collapsing, we are separated by the size of one of our margins, not both.

Inline elements like this one and this one sit on the same line along with adjacent text nodes, if there is space on the same line. Overflowing inline elements will wrap onto a new line if possible (like this one containing text), or just go on to a new line if not, much like this image will do:

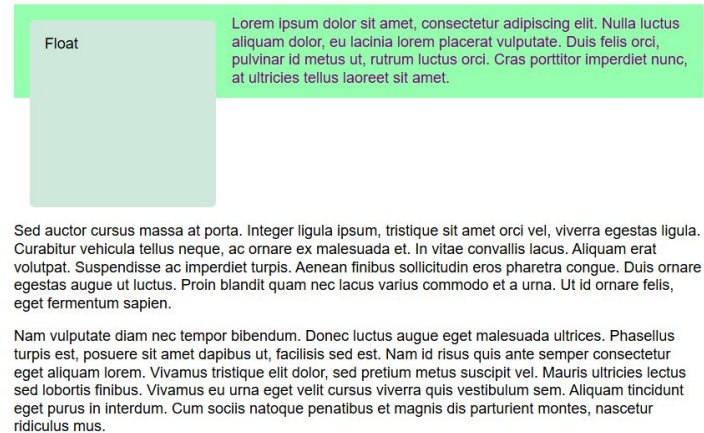


CSS Layout - Floats

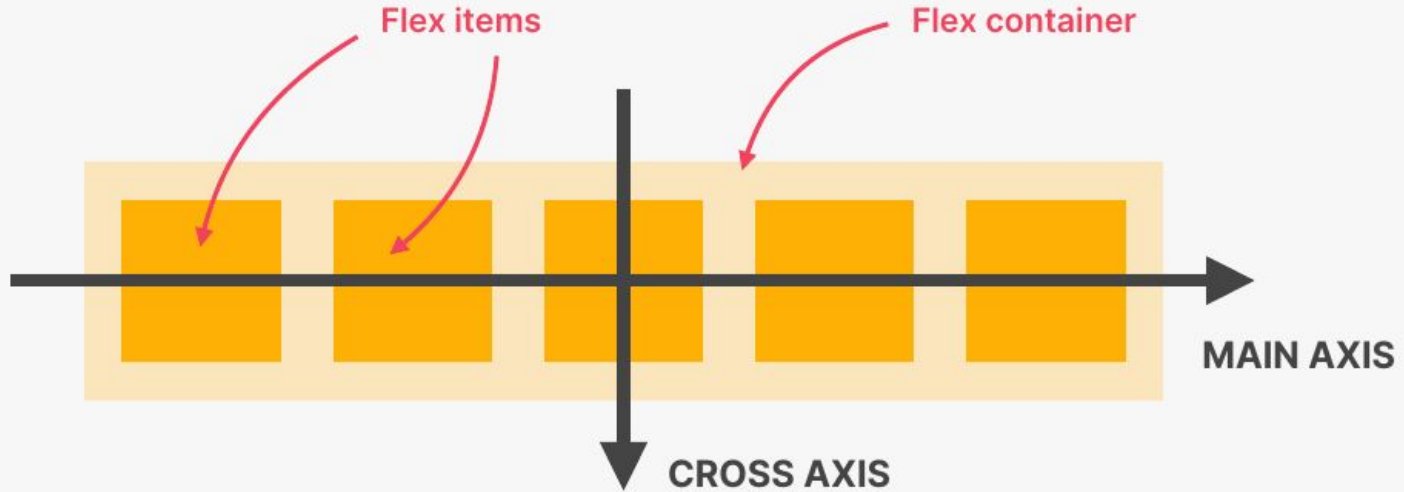


- Implementing layouts that involve an image floating inside a column of text, with the text wrapping around the left or right of it.
- ```
.box { float: left; margin-right: 15px; width: 150px; height: 100px; }
```
- If we want to stop the following element from moving up, we need to use the `clear` property:
  - left: Clear items floated to the left.
  - right: Clear items floated to the right.
  - both: Clear any floated items, left or right.

## Float example



# CSS Layout - Flexbox



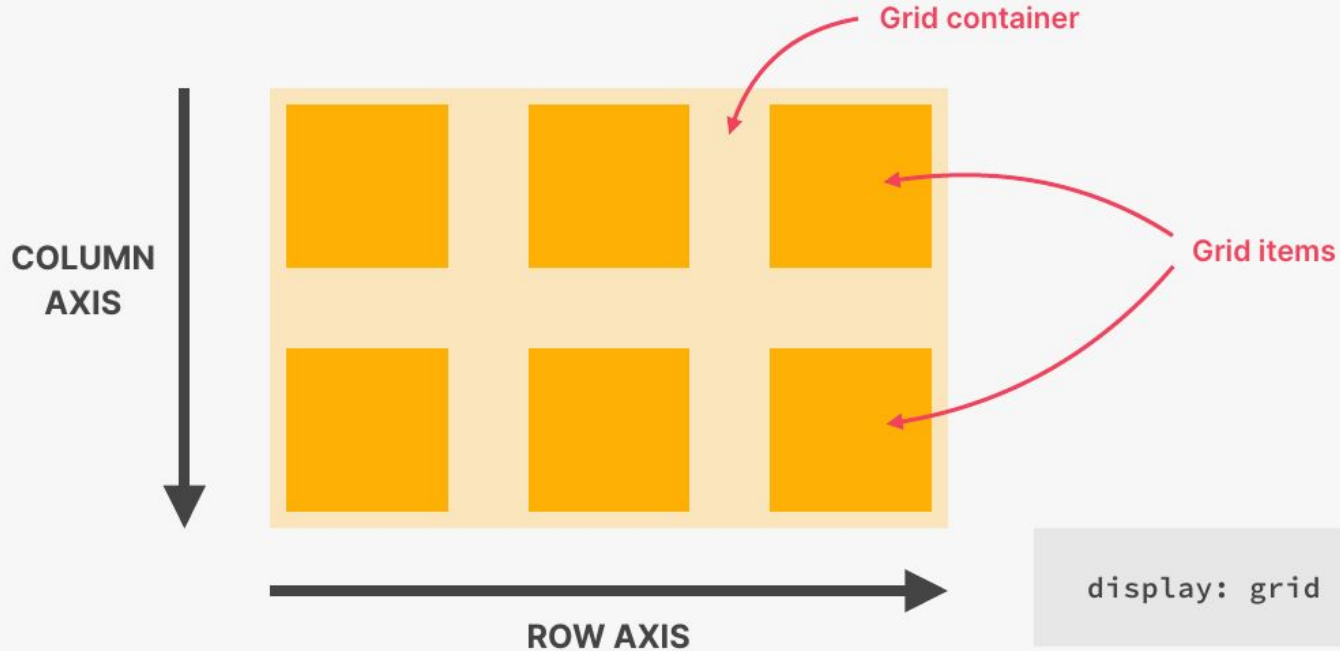
```
display: flex
```

# CSS Layout - Flexbox



- **One-dimensional layout** method for arranging items in rows or columns.
- `display: flex` Element becomes a flex container and its children become flex items.
- `flex-direction`: The direction for arranging flex items.
- `align-items`: Aligning flex items in cross axis.
- `justify-content`: Aligning flex items in main axis.
- `flex`: Sets how a flex item will grow or shrink to fit the space available in its flex container.
- `align-self`: Aligns the item on the cross axis.

# CSS Layout - Grid



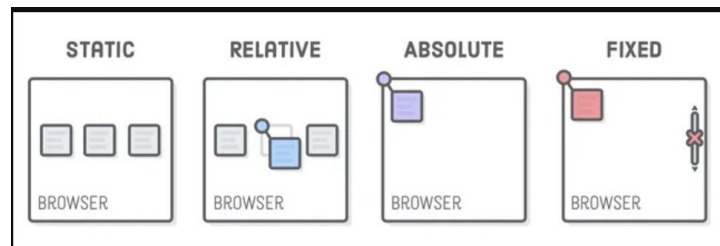
# CSS Layout - Grid

- Two-dimensional layout method for arranging items in rows and columns.
- `display: grid` Transforms all the direct children of the container into grid items.
- `grid-template-columns`: Defines the line names and track sizing functions.
- `gap`: Sets the gaps (also called gutters) between rows and columns.
- `grid-column`: Specifies a grid item's size and location within a grid column.
- `grid-row`: Specifies a grid item's size and location within a grid row.
- `grid-template-areas`: Specifies named grid areas, establishing the cells in the grid and assigning them names.
- `grid-area`: Specifies a grid item's size and location within a grid.

# CSS Positions



- **static**: default, elements flow normally in the page.
- **relative**: positioned relative to its normal location.
- **absolute**: positioned relative to the nearest positioned ancestor.
- **fixed**: positioned relative to the viewport, stays in place when scrolling.
- **sticky**: toggles between relative and fixed based on scroll position.



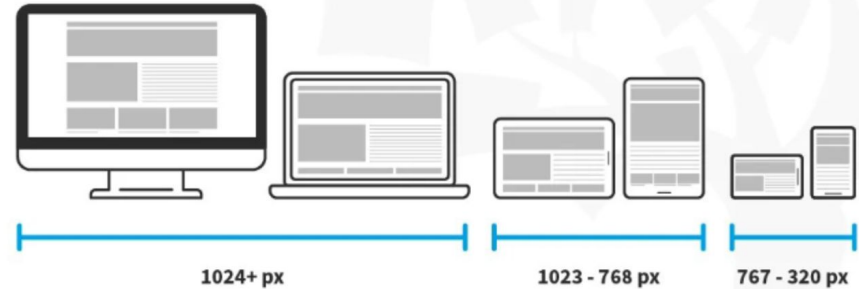
Source: *Linkedin*

# 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.
- This may seem complex (sometimes it is), but most of this can be achieved through pure CSS.

## Responsive Design



Source: LinkedIn

# Media Queries

- Allow responsive design by applying styles based on device properties
- Use conditions like width, height, orientation, resolution
- Syntax:  
`@media (condition) { CSS rules }`
- Commonly used for mobile, tablet, desktop **breakpoints**

Projects/University/web-ta/sampleprojects/htmlcss/index.css

```
@media (max-width: 768px) {
 body { font-size: 14px; }
}

@media (min-width: 768px) {
 .bigtext { color: #232322; }
}
```

# Media Query Conditions



- Viewport/Device Dimensions: `width`, `height`, `min/max-width`, `min/max-height`
- Device Display: `orientation`, `aspect-ratio`, `resolution` (DPI), `scan`, `grid`
- Color/Output: `color`, `color-index`, `monochrome`
- Interaction: `hover`, `pointer`, `any-hover`, `any-pointer`
- Environment: `light-level`, `prefers-color-scheme`, `prefers-reduced-motion`

# Class Activity Time

How does `max-width` differ from `min-width` in media queries?

What is the purpose of using media queries in responsive design?

Give an example of a common **breakpoint** for mobile devices.

Why is a mobile-first approach recommended in CSS?

# Root Element (:root)

- Targets the top-level element
- Used to define CSS variables such as `--primary-color` or `--font-size`
- Example variable: in `:root` define `--primary-color: blue;`
- To use that variable in a query, use `background: var(--primary-color);`
- Makes styles **consistent** and **easy to update** globally



Projects/University/web-ta/sampleprojects/htmlcss/index.css

```
:root {
 --normal-size: 16pt;

 --primary-color: #1733b9;
 --secondary-color: #b93317;
}

body {
 background-color: var(--primary-color);
 color: var(--secondary-color);
 font-size: var(--normal-size);
}
```

# Resources



- <https://www.w3schools.com/css/>
- <https://flexboxfroggy.com/> (Useful css layout practicing site)