8. CSS Units & Measurements

What are CSS Units?

CSS units define the **size** of elements (width, height, font-size, margin, padding, etc.).

They are mainly of **two types**:

1. **Absolute Units** → fixed size (do not change with screen).
2. **Relative Units** → size depends on another value (like parent element or viewport).

Absolute Units

These units are fixed and do not change with screen size.

Unit	Description	Example
`px`	Pixels (most common, screen-based)	`font-size: 16px;`
`cm`	Centimeters	`width: 5cm;`
`mm`	Millimeters	`height: 30mm;`
`in`	Inches (1in = 96px)	`width: 2in;`
`pt`	Points (1pt = 1/72 inch)	`font-size: 12pt;`
`pc`	Picas (1pc = 12pt)	`width: 3pc;`

Note: Absolute units are rarely used in web design (except `px`), as they are not responsive.

Relative Units

These units change depending on the context (viewport, parent, or font).

Unit	Description	Example
`%`	Percentage of parent element	`width: 80%;`
`em`	Relative to element's own font-size	`padding: 2em;`
`rem`	Relative to root (`html`) font-size	`font-size: 1.5rem;`
`vw`	% of viewport width (1vw = 1% width)	`width: 50vw;`
`vh`	% of viewport height (1vh = 1% height)	`height: 100vh;`
`vmin`	% of smaller viewport side	`font-size: 5vmin;`
`vmax`	% of larger viewport side	`font-size: 5vmax;`
`ch`	Width of "0" character	`width: 40ch;`
`ex`	Height of lowercase "x"	`line-height: 2ex;`

Key Differences

- **em** → relative to **parent element** font size.
- **rem** → relative to **root element (html)** font size.
- **vw/vh** → relative to **viewport size** (responsive).

- **%** → relative to **parent container**.

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .absolute {
      font-size: 20px;  /* fixed size */
    }
    .relative-em {
      font-size: 2em;   /* twice the parent font-size */
    }
    .relative-rem {
      font-size: 2rem;  /* twice the root (html) font-size */
    }
    .viewport {
      width: 50vw;      /* 50% of viewport width */
      height: 20vh;     /* 20% of viewport height */
      background: lightblue;
    }
  </style>
</head>
<body>
  <p class="absolute">This is 20px text.</p>
  <div style="font-size: 16px;">
    <p class="relative-em">This is 2em (32px) text.</p>
  </div>
  <p class="relative-rem">This is 2rem (based on root font-size).</p>
  <div class="viewport">This box is responsive with viewport size.</div>
</body>
</html>
```

Float & Clear (Old Layout Method)

What is Float?

- The **float** property is used to move elements **to the left or right** of their container.
- Other elements will wrap around it.
- Originally created for text wrapping around images, later used for layouts (before Flexbox/Grid).

Syntax

```
selector {
  float: left | right | none | inherit;
}
```

- `left` → floats element to the left.
- `right` → floats element to the right.
- `none` → default (no floating).

Example (Image with Text)

```
<!DOCTYPE html>
<html>
<head>
  <style>
    img {
      float: left;
      margin: 10px;
      width: 150px;
    }
  </style>
</head>
<body>
  <img src="image.jpg" alt="Sample">
  <p>This text will wrap around the image because the image is floated to
the left.</p>
</body>
</html>
```

Problem with Float

- Floated elements are **removed from normal document flow**.
- Parent container may collapse if it only contains floated children.

The Clear Property

- **clear** is used to prevent elements from flowing around floated elements.
- Syntax:

```
selector {
  clear: left | right | both | none;
}
```

Example with Clear

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .box1 {
      float: left;
      width: 100px;
      height: 100px;
      background: lightblue;
    }
    .box2 {
      float: right;
      width: 100px;
      height: 100px;
      background: lightgreen;
    }
    .clear {
      clear: both;
    }
  </style>
</head>
<body>
  <div class="box1">Left</div>
  <div class="box2">Right</div>
  <div class="clear"></div>
  <p>This paragraph is below both floated boxes.</p>
</body>
</html>
```

Clearing Techniques

1. **Using `clear` element** (like `<div class="clear">`).
2. **CSS clearfix hack** (most common):

```
.container::after {
  content: "";
  display: table;
  clear: both;
}
```

When to Use Float Today?

- For **text wrapping around images**.
- Not recommended for page layouts (use **Flexbox or Grid** instead).

10. CSS Flexbox (Flexible Box Layout)

What is Flexbox?

- **Flexbox** is a CSS layout model that makes it easy to design **responsive layouts**.
- It arranges items in a **row** or **column**, and distributes space automatically.
- Useful for alignment, spacing, and ordering of elements.

Enable Flexbox

```
.container {
  display: flex;
}
```

All direct child elements of `.container` become **flex items**.

Flex Container Properties

These properties are applied on the **parent (container)**.

1. **flex-direction**
   Defines main axis (row or column).

   ```
   flex-direction: row;         /* default, left → right */
   flex-direction: row-reverse; /* right → left */
   flex-direction: column;      /* top → bottom */
   flex-direction: column-reverse; /* bottom → top */
   ```

2. **justify-content**
   Aligns items along the **main axis**.

```
justify-content: flex-start;   /* default */
justify-content: flex-end;     /* items at end */
justify-content: center;       /* center items */
justify-content: space-between; /* equal space between */
justify-content: space-around; /* space around each */
justify-content: space-evenly; /* equal space around */
```

3. **align-items**
   Aligns items along the **cross axis**.

```
align-items: stretch;   /* default */
align-items: flex-start;
align-items: flex-end;
align-items: center;
align-items: baseline;  /* aligns text baselines */
```

4. **align-content**
   Used when there are **multiple rows** of flex items.

```
align-content: flex-start;
align-content: flex-end;
align-content: center;
align-content: space-between;
align-content: space-around;
align-content: stretch; /* default */
```

5. **flex-wrap**
   Controls whether items stay in one line or wrap.

```
flex-wrap: nowrap;   /* default */
flex-wrap: wrap;     /* items move to next line */
flex-wrap: wrap-reverse;
```

6. **gap**
   Sets space between flex items.

```
gap: 20px;
```

Flex Item Properties

These are applied on the **children (items)**.

1. **order** → changes order of items.

```
.item1 { order: 2; }
.item2 { order: 1; }
```

2. **flex-grow** → how much item grows (relative).

```
.item1 { flex-grow: 1; } /* grow twice as much as item2 */
.item2 { flex-grow: 2; }
```

3. **flex-shrink** → how much item shrinks.

```
.item1 { flex-shrink: 1; }
```

4. **flex-basis** → initial size of item before space distribution.

```
.item1 { flex-basis: 200px; }
```

5. **align-self** → overrides `align-items` for a single item.

```
.item1 { align-self: flex-start; }
```

6. **Shorthand: flex**

```
.item {
  flex: 1 1 200px; /* grow | shrink | basis */
}
```

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .container {
      display: flex;
      flex-direction: row;
      justify-content: space-around;
      align-items: center;
      gap: 15px;
      border: 2px solid black;
      height: 200px;
    }
    .item {
      background: lightblue;
      padding: 20px;
      border: 1px solid blue;
    }
    .grow {
      flex-grow: 2;
    }
  </style>
</head>
<body>
  <h2>Flexbox Example</h2>
  <div class="container">
    <div class="item">Item 1</div>
    <div class="item grow">Item 2 (grows more)</div>
    <div class="item">Item 3</div>
  </div>
</body>
</html>
```

Key Points

- Flexbox works in **one dimension** (row OR column).
- Use **container properties** (`flex-direction`, `justify-content`, `align-items`) to control layout.

- Use **item properties** (`flex-grow`, `order`, `align-self`) to control individual items.
- For **2D layouts (rows + columns) →** use **CSS Grid**.