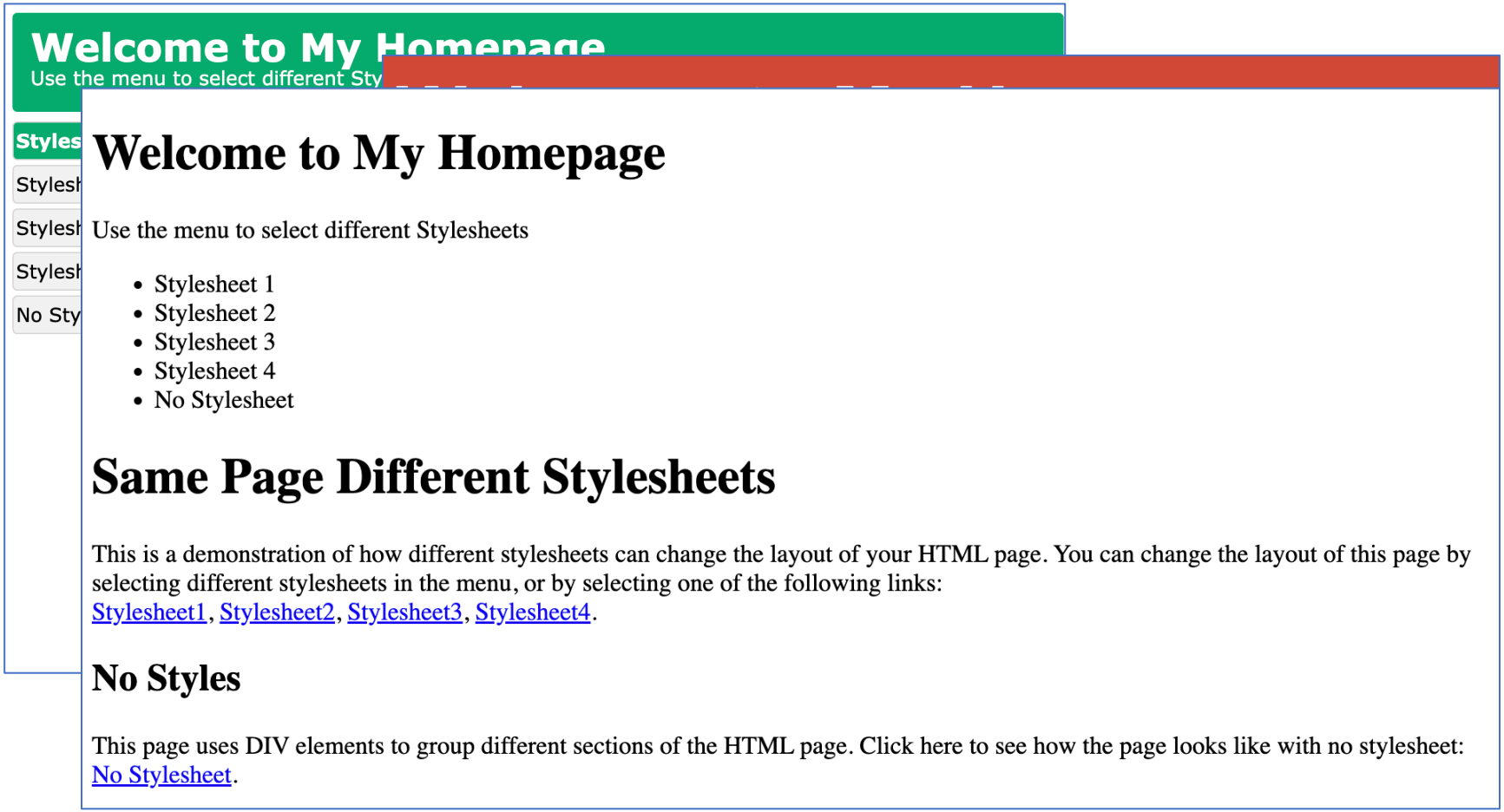
Cascading Style Sheets (CSS)

Motivation



https://www.w3schools.com/css/demo_default.htm

Content

1. Introduction
2. Responsive Design
3. CSS Framework

HUST

hust.edu.vn fb.com/dhbkhn

1. Introduction

1.1. Overview

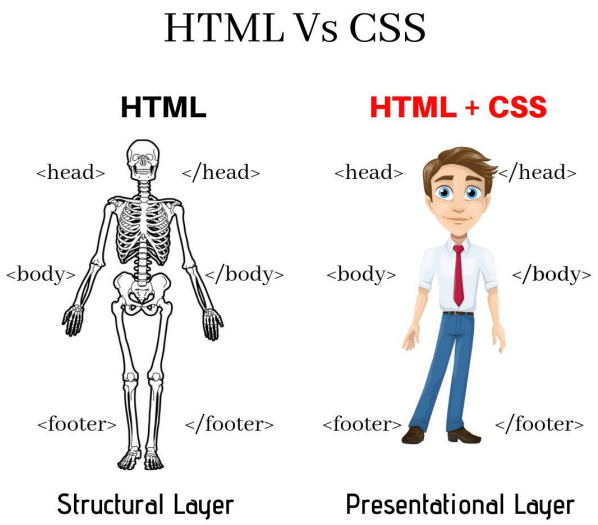
1.2. Syntax

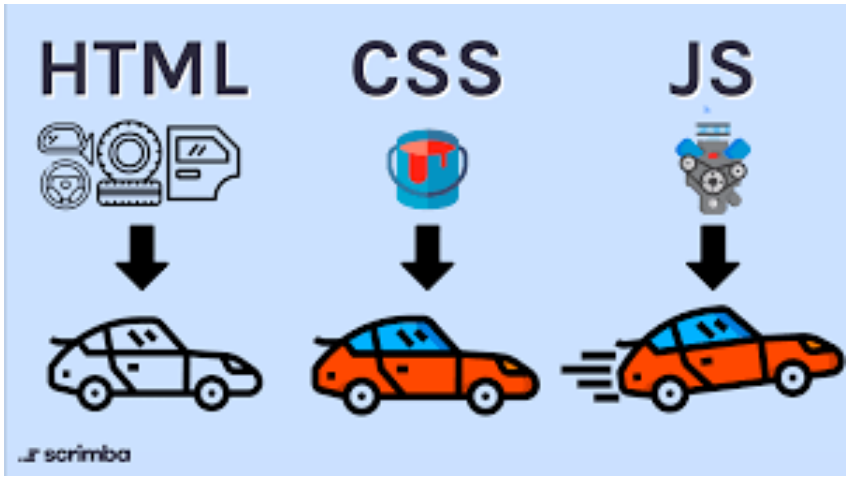
1.3. Selectors

1.4. Units and Important Properties

What is CSS?

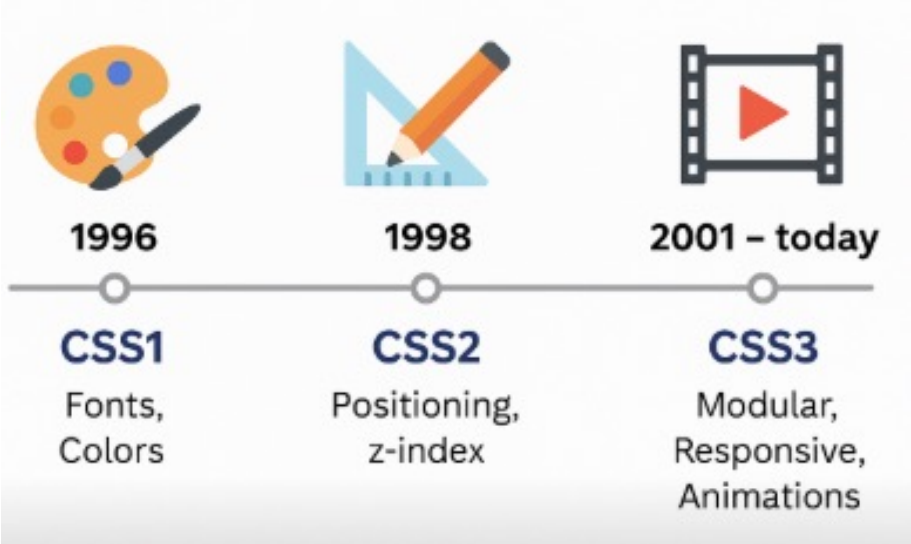
- CSS: Cascading Style Sheets
  - Used to **separate** the **content** of a document from its **presentation** aspects
  - Presentation: **colors, fonts, layout, etc.**
  - Works together with HTML and JavaScript:





History of CSS

- Håkon Wium Lie (1994). Idea of CSS
- CSS1 (1996)
  - Text formating and colors
  - Properties: font-family, font-size, color, backgroud
- CSS2 (1998)
  - Advanced layout
  - Properties: position, z-index, media types
- CSS3 (2001-now)
  - Modular structure
  - Responsive design support
  - Transition, animations



Advantages of CSS

- Separation of content & presentation
  - HTML handles content
  - CSS handles presentation
- Consistency across pages
  - Apply one presentation to multiple pages
- Easier maintenance
  - Update one CSS file → changes presentation everywhere
- Better SEO and device-friendliness
  - Clean HTML, faster loading
  - Responsive design for multiple device

1.2. Syntax

- Three ways to apply CSS

1. Inline CSS

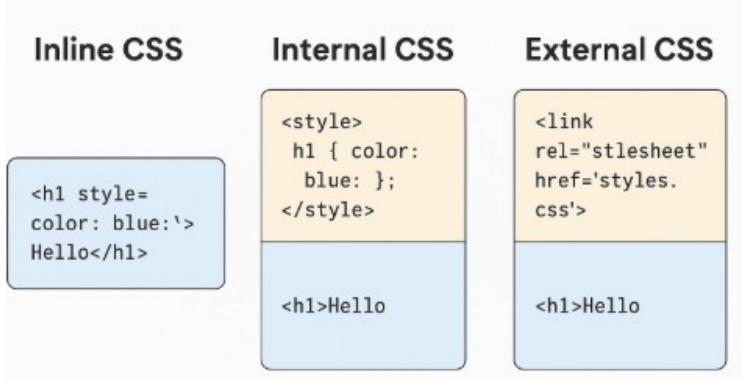
- CSS written directly inside an HTML tag
- Affects only to that HTML element

2. Internal CSS

- CSS defined inside <head>
- Affects only that HTML page

3. External CSS

- CSS Stored in a .css file and linked with <link>
- Can be reused across multiple pages



Inline CSS

- CSS rules are written in an element's style atribute
- Scope: affects that element only
- Syntax: style="property$_1$: value$_1$; property$_2$: value$_2$;"
- Features
  - Quick, apply to a single element
  - Hard to maintain

```
<!-- Inline CSS -->
<h2 style="color: red; font-family: Arial;">
  This is styled with Inline CSS
</h2>
```

This is styled with Inline CSS

Internal CSS

- Rules are written inside a <style> element within the <head> section of an HTML page
- Scope: affects that HTML page only
- Features:
  - Easier to manage than inline CSS
  - Cannot be reused across multiple pages

```
<head>
  <!-- Internal CSS -->
  <style>
    h2 {
      color: green;
      text-align: center;
    }
  </style>
</head>
```

This is styled with Internal CSS

External CSS

- Rules are written in a **separate .css file**
- Scope: can be reused across multiple pages
- Advantages:
  - Best practice for large projects
  - Promotes consistency across the whole site
  - Easier maintenance (update one file → changes everywhere)
  - Reduces HTML file size
- Limitations: Requires loading an additional file

style.css > h2

```css
h2 {
  color: darkblue;
  text-decoration: underline;
}
```

```html
<!-- External CSS -->
<link rel="stylesheet" href="style.css">
```

Basic Syntax of Internal & External CSS

- Syntax

```
selector{
    property: value;
}
```

- Selector → chooses the element(s) to style (e.g., h1, #id)
- Property → the aspect you want to style (e.g., color, font-size)
- A group of declarations is enclosed in curly braces { }

```
h1 {
  color: blue;
  font-size: 24px;
  text-align: center;
}
```

Selector: `h1` → all `<h1>` headings

Properties: `color`, `font-size`, `text-align`

Values: `blue`, `24px`, `center`

1.3. Selectors

Selector	Example	Description
*	*	Selects all elements
element	p	Selects all <p> elements
.class	.intro	Selects all elements with class="intro"
#id	#firstname	Selects the element with id="firstname"

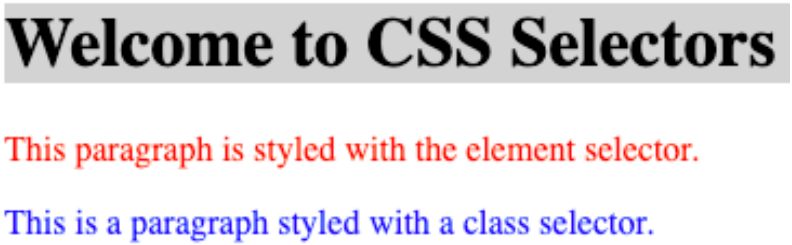
All elements

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  text-align: center;
  color: red;
}
</style>
</head>
<body>
<h1>Hello world!</h1>
<p>Me too!</p>
<p>And me!</p>
</body>
</html>
```

Element, Class, ID

basic-selectors.html > ...

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Basic Selectors Example</title>
6     <style>
7       p {
8         color: red;
9       }
10      .intro {
11        color: blue;
12      }
13      #header {
14        background-color: lightgray;
15      }
16    </style>
17  </head>
18  <body>
19    <header id="header">
20      <h1>Welcome to CSS Selectors</h1>
21    </header>
22
23    <p>This paragraph is styled with the element selector.</p>
24
25    <p class="intro">This is a paragraph styled with a class selector.</p>
26  </body>
27  </html>
28
```



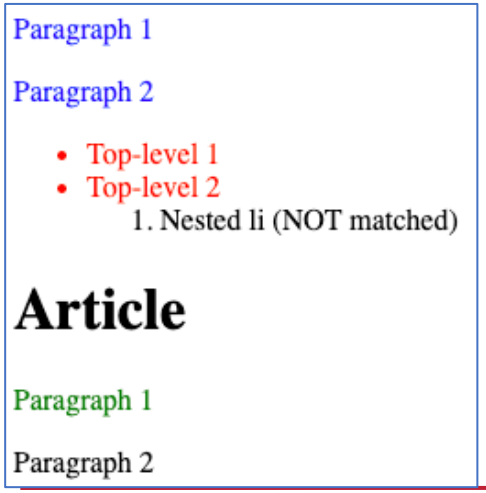
Descendant, Child, Sibling

- **Descendant A B:** selects **all B** that are anywhere inside **A**
- **Child A > B:** selects only **B that are direct children** of A
- **Adjacent sibling A + B:** selects **the first B immediately after A** (same parent).

Descendant Selector div p	 Paragraph 1 <section> Paragraph 2 </section>
Child Selector ul > li	 - Top-level 1 - Top-level 2 - Nested
Adjacent Silbing Selector h1+p	Article Paragraph 1 Paragraph 2

Example

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <style>
      /* Descendant */
      div p {
        color: blue;
      }
      /* Child (only direct LI under the outer UL) */
      ul > li {
        color: red;
      }
      /* Adjacent sibling (only the first P after H1) */
      h1 + p {
        color: green
      }
    </style>
  </head>
  <body>
    <div>
      <p>Paragraph 1</p>
      <section>
        <p>Paragraph 2</p>
      </section>
    </div>

    <ul>
      <li>Top-level 1</li>
      <li>Top-level 2 </li>
       <ol>
          <li>Nested li (NOT matched)</li>
       </ol>
    </ul>

    <h1>Article</h1>
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
  </body>
</html>
```



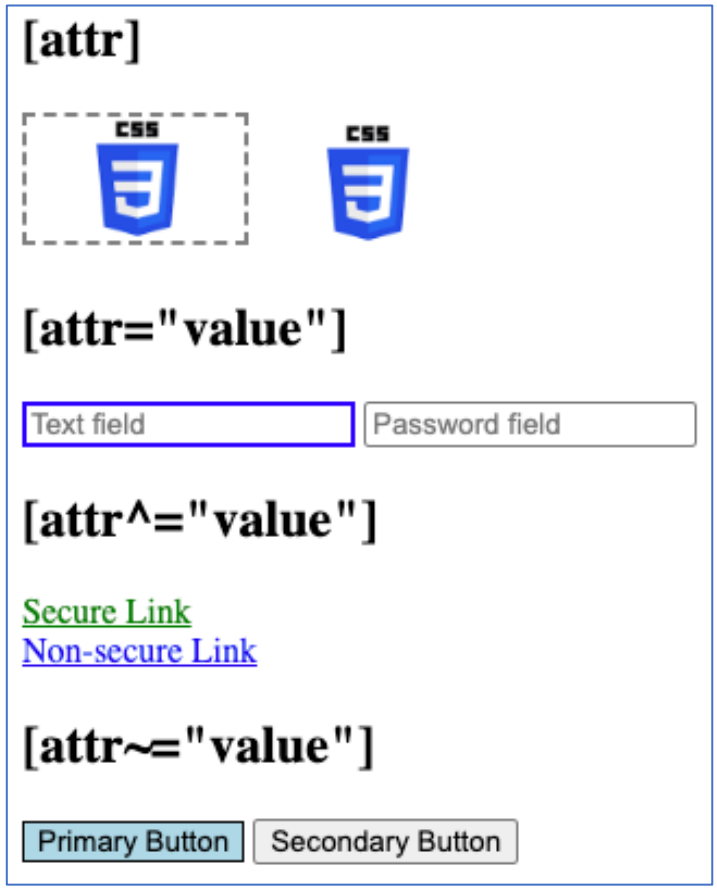
Attribute Selector

- Select elements by attributes/values

Selector	Description	Example
[attr]	Elements that **have the attribute**, regardless of its value.	img[alt]{...}
[attr=“value”]	Attribute value is **exactly equal** to "value"	input[type=“text”]
[attr^="value"]	Attribute value **starts with** “value”	a[href^="https"]
[attr$="value"]	Attribute value **ends with** “value”	img[src$=".png"]
[attr*="value"]	Attribute value **contains** "value" as substring	[class*="btn"] → .btn, .btn-primary, .btn primary
[attr~=”value"]	Attribute value contains "value" as a **whole word** (separated by spaces)	[class~="btn"] → .btn, .btn primary ❌: btn-primary

```css
<style>
  /* [attr] - has attribute */
  img[alt] {
    border: 2px dashed gray;
  }
  /* [attr="value"] - exact match */
  input[type="text"] {
    border: 2px solid blue;
  }
  /* [attr^="value"] - starts with */
  a[href^="https"] {
    color: green;
  }

  /* [attr~="value"] - contains whole word */
  [class~="btn"] {
    border: 1px solid black;
    background-color: lightblue;
  }
</style>
```

```html
<h2>[attr]</h2>
<img src="css3.png" alt="sample image" width="100">
<img src="css3.png" width="100">

<h2>[attr="value"]</h2>
<form>
  <input type="text" placeholder="Text field">
  <input type="password" placeholder="Password field">
</form>

<h2>[attr^="value"]</h2>
<a href="https://example.com">Secure Link</a><br>
<a href="http://example.com">Non-secure Link</a>

<h2>[attr~="value"]</h2>
<button class="btn primary">Primary Button</button>
<button class="btn-secondary">Secondary Button</button>
```



Pseudo-classes

- A **pseudo-class** defines the **special state** of an element
- Syntax: *selector:pseudo-class { property: value; }*
- Common Pseudo-class

  :hover → when mouse is over an element
  :active → when element is being clicked
  :focus → when input is focused
  :first-child → the first child of its parent
  :nth-child(n) → selects the nth child

- Examples

```
a:hover { color: red; }
input:focus { border: 2px solid blue; }
p:first-child { font-weight: bold; }
```

Pseudo-classes Example

```css
<style>
  /* :hover */
  a:hover {
    color: red;
  }

  /* :active */
  a:active {
    color: orange;
  }

  /* :focus */
  input:focus {
    border: 2px solid blue;
  }

  /* :first-child */
  p:first-child {
    color: darkgreen;
  }

  /* :nth-child */
  li:nth-child(2) {
    background-color: #f0f0f0;
  }
```

```html
<body>
  <h1>Pseudo-class Demonstration</h1>
  <p>Move the mouse</p>
  <a href="#">Hover or click me</a>

  <p>Click inside the input field:</p>
  <input type="text" placeholder="Focus me">

  <h2>:first-child</h2>
  <div>
    <p>This is the first paragraph</p>
    <p>This is the second paragraph</p>
  </div>

  <h2>:nth-child</h2>
  <ul>
    <li>Item 1</li>
    <li>Item 2 (highlighted)</li>
    <li>Item 3</li>
  </ul>
</body>
```

Move the mouse

Hover or click me

Click inside the input field:

Focus me

:first-child

This is the first paragraph

This is the second paragraph

:nth-child

- Item 1
- Item 2 (highlighted)
- Item 3

Pseudo-elements

- A **pseudo-element** allows you to **style specific parts** of an element.
- Syntax:

  selector::pseudo-element { property: value; }
- Common Pseudo-element

  ::before → insert content before element

  ::after → insert content after element

  ::first-line → style the first line of text

  ::first-letter → style the first letter of text

Pseudo-elements Example

```
<style>
  /* ::first-letter */
  p::first-letter {
    color: red;
  }

  /* ::first-line */
  p::first-line {
    color: green;
  }

  /* ::after */
  p::after {
    content: " ✔";
  }
</style>
```

```
<p>
  This is the first line<br>
  This is the second line
</p>

<p>
  This is a paragraph
</p>
```

This is the first line
This is the second line ✔

This is a paragraph ✔

CSS Specificity (Priority)

- An algorithm determines which style declaration is applied to an element

```
<html>
<head>
  <style>
    #demo {color: blue;}
    .test {color: green;}
    p {color: red;}
  </style>
</head>
<body>

<p id="demo" class="test" style="color: pink;">Hello World!</p>

</body>
</html>
```

```
<html>
<head>
  <style>
    .test {color: green;}
    p {color: red;}
  </style>
</head>
<body>

<p class="test">Hello World!</p>

</body>
</html>
```

```
<html>
<head>
  <style>
    #demo {color: blue;}
    .test {color: green;}
    p {color: red;}
  </style>
</head>
<body>

<p id="demo" class="test">Hello World!</p>

</body>
</html>
```

Specificity Hierarchy

Selector	Example	Description
Inline styles	<h1 style="color: pink;">	Highest priority
Id selectors	#navbar	Second highest priority
Classes, attribute selectors and pseudo-classes	.test, [type="text"], :hover	Third highest priority
Elements and pseudo-elements	h1, ::before, ::after	Low priority
Universal selector and :where()	*, where()	No priority

!important keyword: overrides ALL styling rules

```
selector {
  property: value !important;
}
```

Example

```
<style>
#demo {color: blue;}
.test {color: green;}
p {color: red;}
</style>
</head>
<body>
<p id="demo" class="test" style="color: pink;">
Hello World!
</p>
```

Hello World!

```
<style>
p {
  background-color: yellow !important;
}
#myid {
  background-color: blue;
}
.myclass {
  background-color: gray;
}
</style>
</head>
<body>

<p style="background-color:orange;">This is a paragraph.</p>
<p class="myclass">This is a paragraph.</p>
<p id="myid">This is a paragraph.</p>
```

This is a paragraph.

This is a paragraph.

This is a paragraph.

1.4. Units & Important Properties

Absolute Units

- px (pixels)
  - Fixed relative to the screen's resolution.
  - Most commonly used in web design.
- pt (points)
  - 1 pt = 1/72 of an inch.
  - More common in print rather than web.
- cm (centimeters)
  - Physical measurement unit.
  - Rarely used on screens due to device variability.
- Features
  - Always fixed
  - Good for print, not ideal for responsive web

1.4. Units & Important Properties

Relative Units

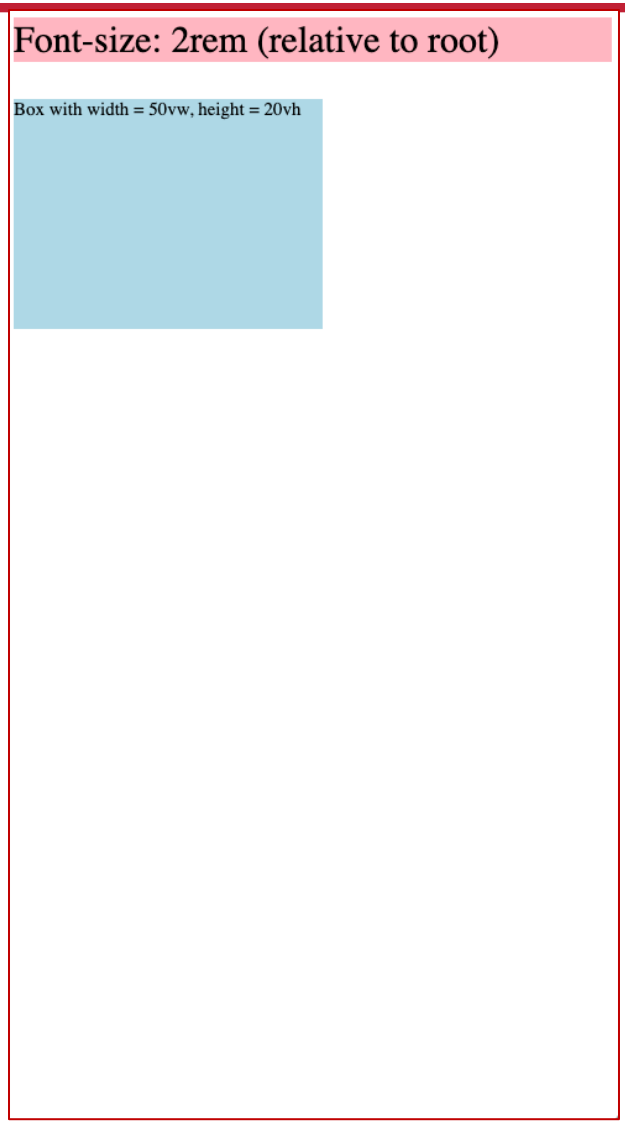
- % (percentage)
  - Relative to the **parent element.**
  - Common for width, height, margin, padding.
- em
  - Relative to the **font-size** of the **parent element.**
  - Example: if parent = 16px, then 2em = 32px.
- rem (root em)

  Relative to the **font-size** of the **root element (html).**
- vw / vh (viewport width / height)
  - 1vw = 1% of viewport's width.
  - 1vh = 1% of viewport's height.
  - Useful for full-screen layouts and responsive design.

Units Example

```
<style>
  html {
    font-size: 15px;
  }
  .rem-box {
    background: lightpink;
    font-size: 2rem;
  }
  .viewport-box {
    background: lightblue;
    width: 50vw;  /* 50% of viewport width */
    height: 20vh; /* 20% of viewport height */
  }
</style>
```

```
<body>
    <p class="rem-box">
        Font-size: 2rem (relative to root)
    </p>

    <p class="viewport-box">
        Box with width = 50vw, height = 20vh
    </p>
</body>
```

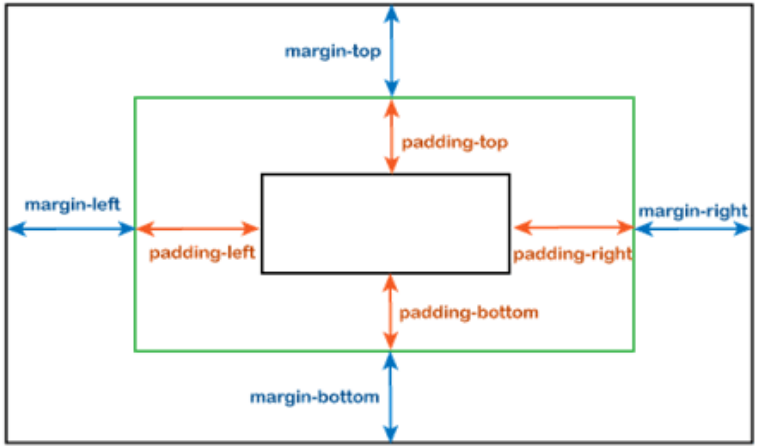


Color & Backgroud

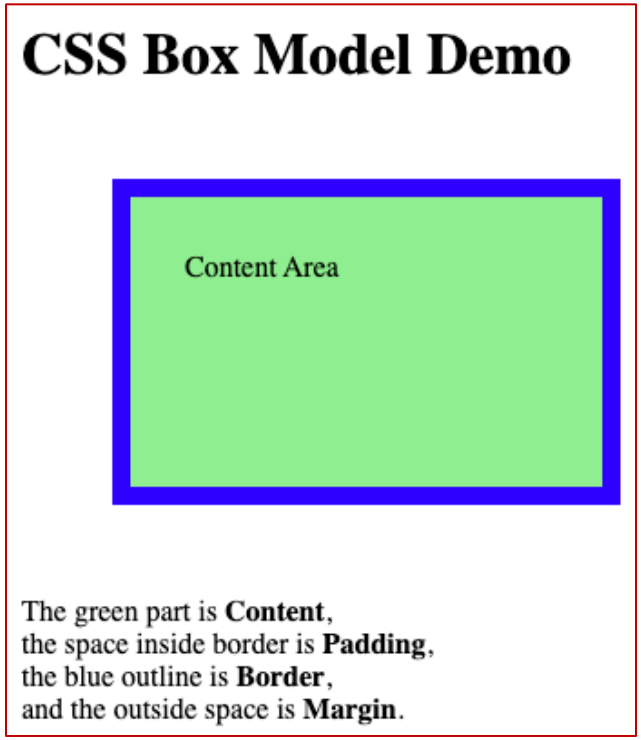
- Text color
  - color: red; → sets text color.
  - Supports: name (red), HEX (#ff0000), RGB (rgb(255,0,0)), HSL (hsl(0,100%,50%)).
- Background color
  - background-color: #ff0000;
  - Sets solid background behind an element.
- Background image
  - background-image: url("bg.png");
  - Allows adding image backgrounds.

Box Model

- Every element in CSS is a rectangular box.
- Box model consists of 4 layers:
  - Content → the actual text, image, or element inside.
  - Padding → space between content and border.
  - Border → surrounds the padding and content.
  - Margin → outermost space, separates element from neighbors



```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSS Box Model Demo</title>
  <style>
    .box {
      width: 200px;
      height: 100px;
      background-color: lightgreen; /* Content */
      padding: 30px;                /* Padding */
      border: 10px solid blue;      /* Border */
      margin: 50px;                 /* Margin */
    }
  </style>
</head>
<body>
  <h1>CSS Box Model Demo</h1>
  <div class="box">
    Content Area
  </div>
  <p>
    The green part is <strong>Content</strong>,<br>
    the space inside border is <strong>Padding</strong>,<br>
    the blue outline is <strong>Border</strong>,<br>
    and the outside space is <strong>Margin</strong>.
  </p>
</body>
</html>
```



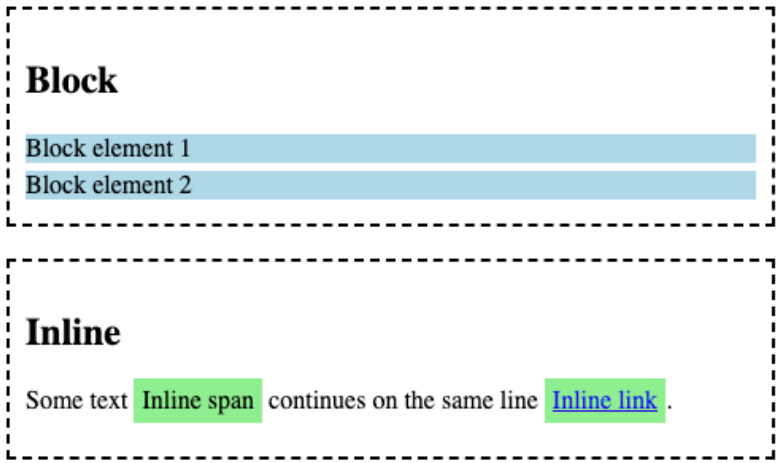
Display Property

- block
  - Element takes the full width of its parent.
  - Starts on a new line.
  - Examples: , <h1>, <p>.
- inline
  - Flows in the same line with text/content.
  - Width/height cannot be set.
  - Examples: , <a>, .
- inline-block
  - Behaves like inline (stays on the same line).
  - But allows width and height settings.
- none
  - Element is not displayed (removed from layout flow).
  - Different from visibility: hidden (hidden but space preserved).

Display Property - Example

```css
<style>

  .block {
    display: block;
    background: lightblue;
    margin: 5px 0;

  }

  .inline {
    display: inline;
    background: lightgreen;
    padding: 5px;
    /* width/height ignored for inline */
    width: 200px;
    height: 50px;
  }

  .wrapper {
    border: 2px dashed black;
    padding: 10px;
    margin-top: 20px;
  }
</style>
```

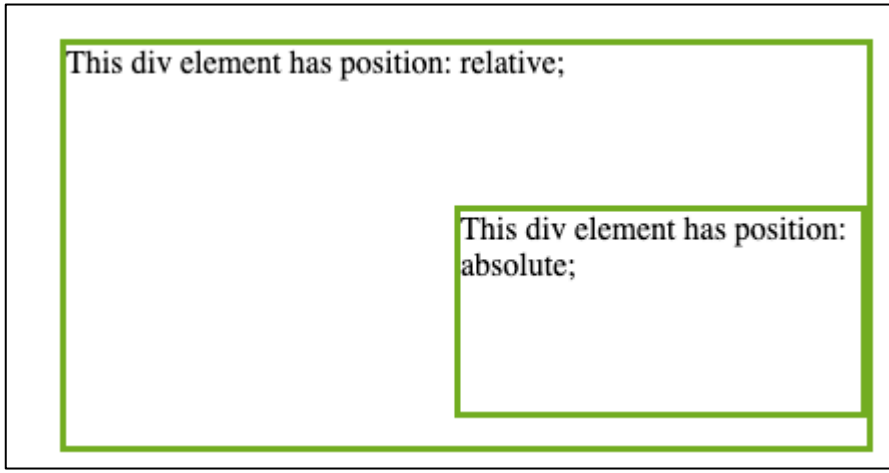
```html
<div class="wrapper">
  <h2>Block</h2>
  <div class="block">Block element 1</div>
  <div class="block">Block element 2</div>
</div>

<div class="wrapper">
  <h2>Inline</h2>
  <p>
    Some text
    <span class="inline">Inline span</span>
    continues on the same line
    <a href="#" class="inline">Inline link</a>.
  </p>
</div>
```



Position Property

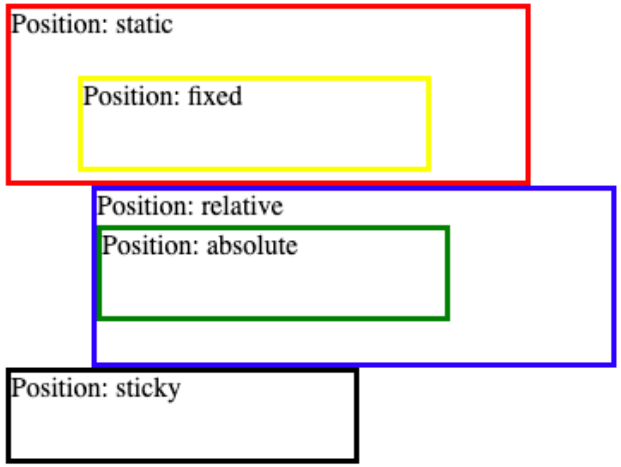
- static (default): is not affected by top, bottom, left, right.
- relative
  - Positioned relative to its normal position.
  - Can use top, left, etc. to shift slightly.
- absolute
  - Positioned relative to the nearest positioned ancestor
  - If no ancestor, positioned relative to the document <html>.
- fixed: stays in place when scrolling.
- sticky: is positioned based on the user's scroll position.

```
div.relative {
  position: relative;
  left: 20px;
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}

div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
  border: 3px solid #73AD21;
}
```



Position

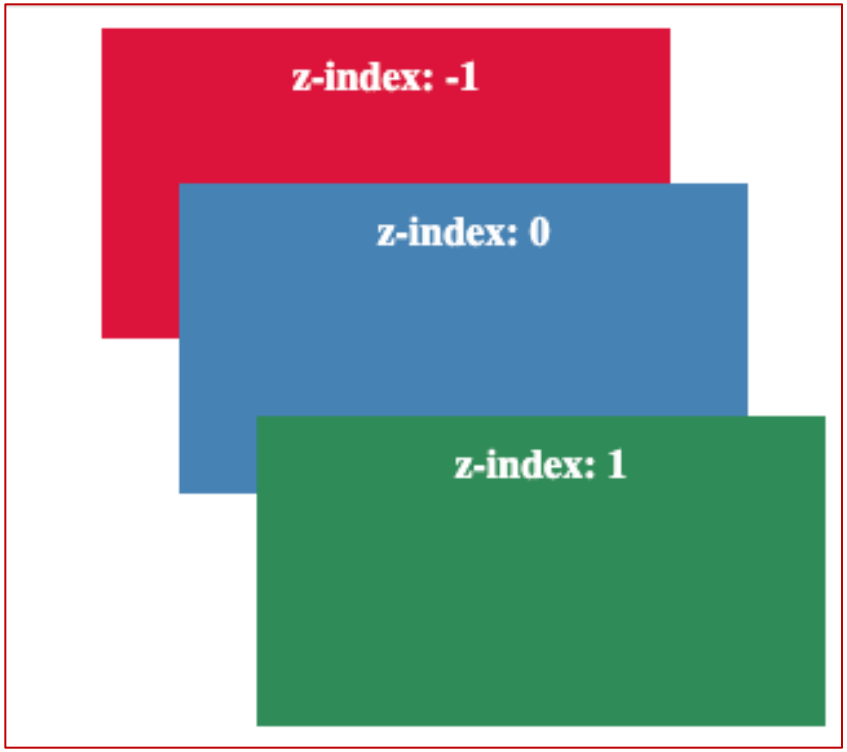
```css
.static {
  position: static;
  width: 300px;
  height: 100px;
  left: 50px;
  border: 3px solid red;
}

.relative {
  position: relative;
  width: 300px;
  height: 100px;
  left: 50px;
  border: 3px solid blue;
}

.absolute {
  position: absolute;
  top: 20px;
  width: 200px;
  height: 50px;
  border: 3px solid green;
}

.fixed {
  position: fixed;
  top: 50px;
  left: 50px;
  width: 200px;
  height: 50px;
  border: 3px solid yellow;
}

.sticky {
  position: sticky;
  top: 20px;
  width: 200px;
  height: 50px;
  border: 3px solid black;
}
</style>
</head>
<body>
<div class="static">Position: static</div>
<div class="relative">
Position: relative
<div class="absolute">Position: absolute</div>
</div>
<div class="fixed">Position: fixed</div>
<div class="sticky">Position: sticky</div>
</body>
```



Overlap

```css
<style>
  .box {
    position: relative;
    width: 200px;
    height: 100px;
    padding: 10px;
    color: white;
    font-weight: bold;
    text-align: center;
  }

  .z-neg {
    background: crimson;
    left: 30px;
    z-index: -1; /* nằm dưới cùng */
  }

  .z-zero {
    background: steelblue;
    top: -60px;
    left: 60px;
    z-index: 0; /* mặc định */
  }

  .z-pos {
    background: seagreen;
    top: -90px;
    left: 90px;
    z-index: 1; /* trên cùng */
  }
</style>
```

```html
<body>
  <div class="box z-neg">z-index: -1</div>
  <div class="box z-zero">z-index: 0</div>
  <div class="box z-pos">z-index: 1</div>
</body>
```



HUST

hust.edu.vn fb.com/dhbkhn

2. Responsive Web Design

2.1. Concepts & Principles

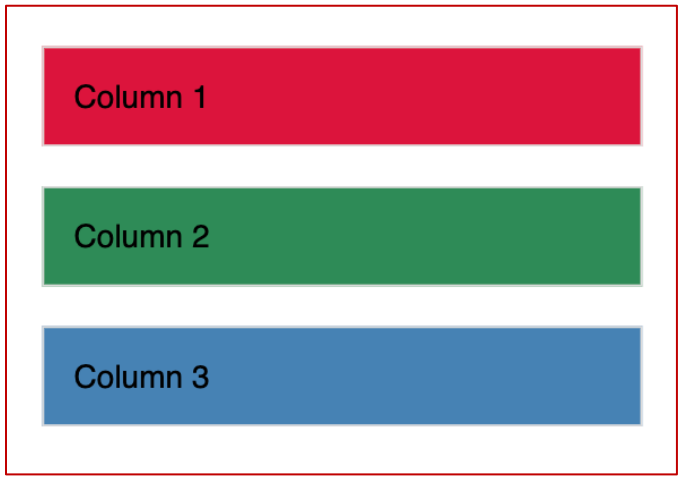
2.2. Media Query

2.3. Flexbox

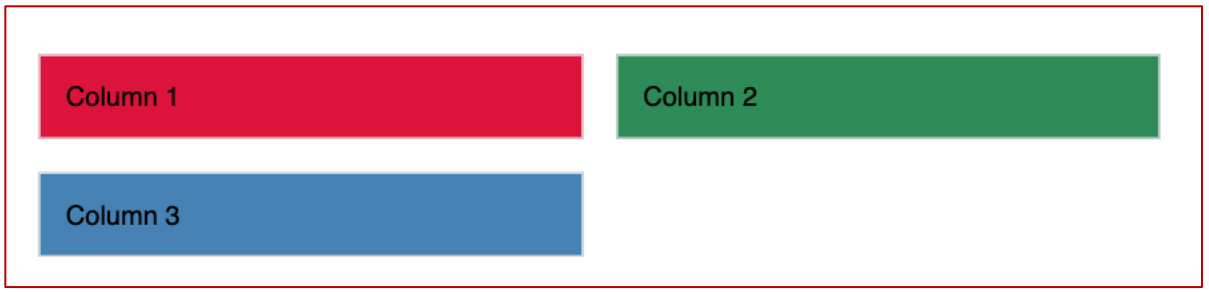
2.1. Concepts & Principles

- Design for multi-devices with different screen sizes (mobile, tablet, desktop).
- The layout adapts automatically to the device being used.

Example



Mobile



Tablet

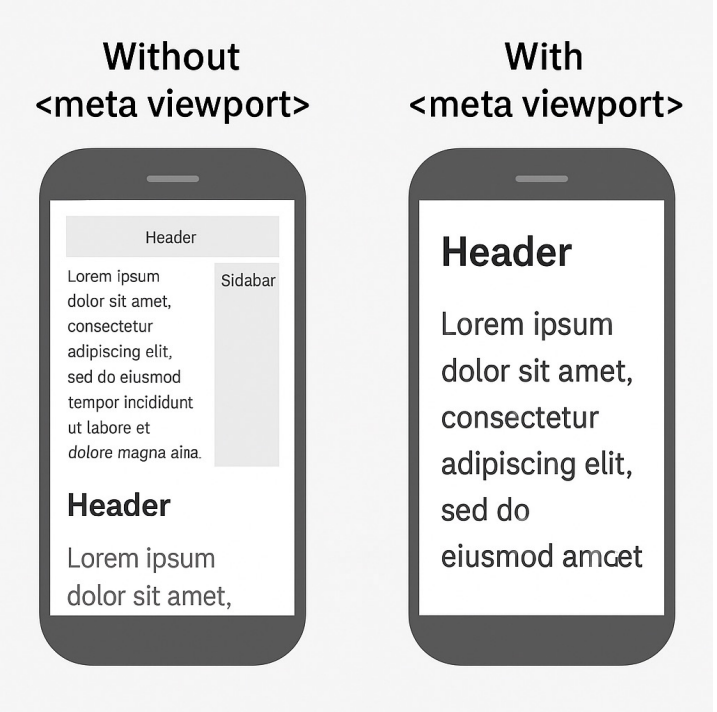
Laptop/Desktop

Meta Viewport

- The <meta viewport> tag tells the browser how to control the page's dimensions and scaling.
- Essential for **responsive design** on mobile and tablet devices.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

- `viewport` → visiable area of a web page (mobile, table, desktop)
- `width=device-width` → Page width matches the device screen width.
- `initial-scale=1.0` → Sets the default zoom level when the page loads.



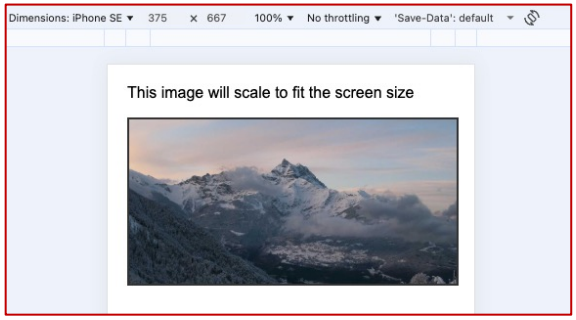
Flexible Image

- Images should adapt to the size of their container.
- Use max-width: 100% to make images scale down when needed.
- Prevents images from overflowing outside their parent element.

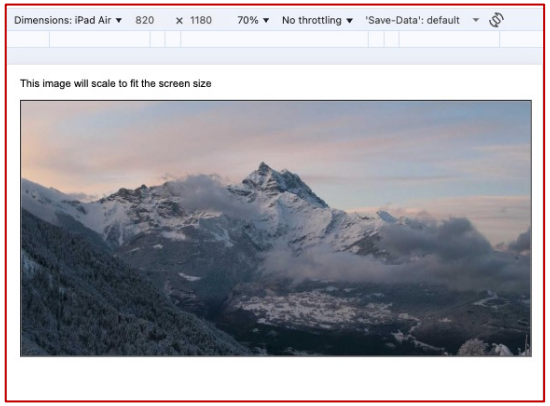
```
img {
  max-width: 100%;
  height: auto;
}
```

- max-width: 100% → image never exceeds container width.
- height: auto → keeps aspect ratio when scaling.
- Result: image automatically resizes on smaller screens

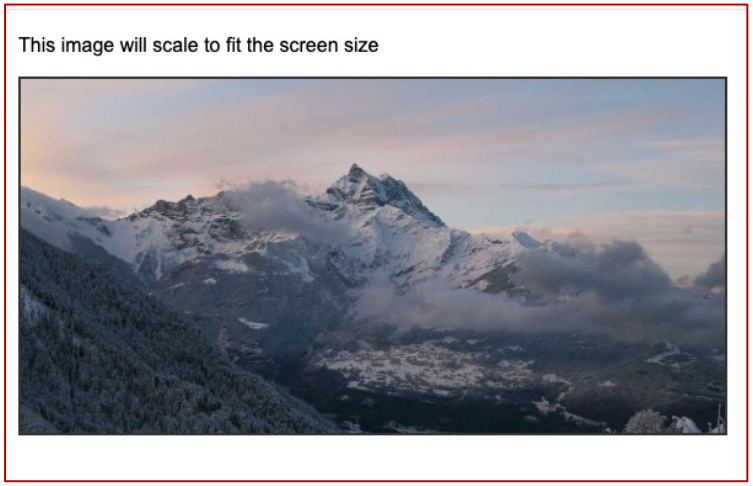
```html
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Responsive Image Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    img {
      max-width: 100%;
      height: auto;
      border: 2px solid #333;
      display: block;
      margin: auto;
    }
  </style>
</head>
<body>
  <p>This image will scale to fit the screen size</p>
  <img src="https://picsum.photos/1200/600" alt="Sample image">
</body>
```



Mobile



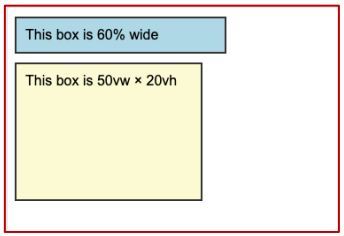
Tablet



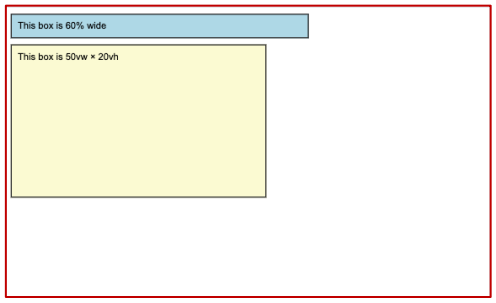
Laptop/Desktop

Relative Units

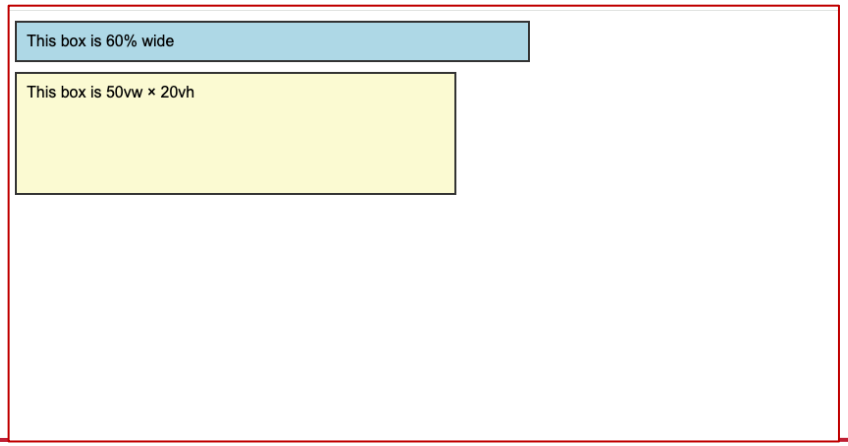
```html
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Relative Units Simple Demo</title>
  <style>
    body { font-family: Arial, sans-serif; }
    .box {
      border: 2px solid #333;
      margin: 10px 0;
      padding: 10px;
    }
    .percent { width: 60%; background: lightblue; }
    .vwvh    { width: 50vw; height: 20vh; background: lightgoldenrodyellow; }
  </style>
</head>
<body>
  <div class="box percent">This box is 60% wide</div>
  <div class="box vwvh">This box is 50vw × 20vh</div>
</body>
```



Mobile



Tablet



Laptop/Desktop

2.2. Media Query

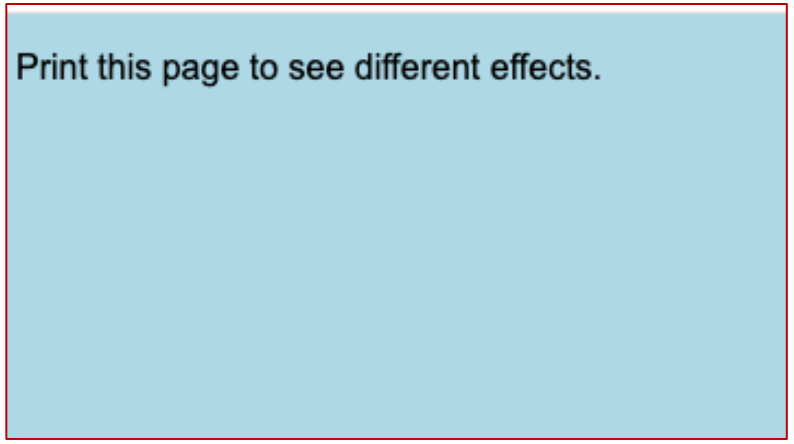
- Allow applying CSS conditionally based on device characteristics.
- Width/Height of viewport (max-width, min-width), e.g.,
  - 0px - 576px: mobile
  - 577px - 992px : tablet
  - 993px+ : laptop, desktop
- Media type (screen, print, all)

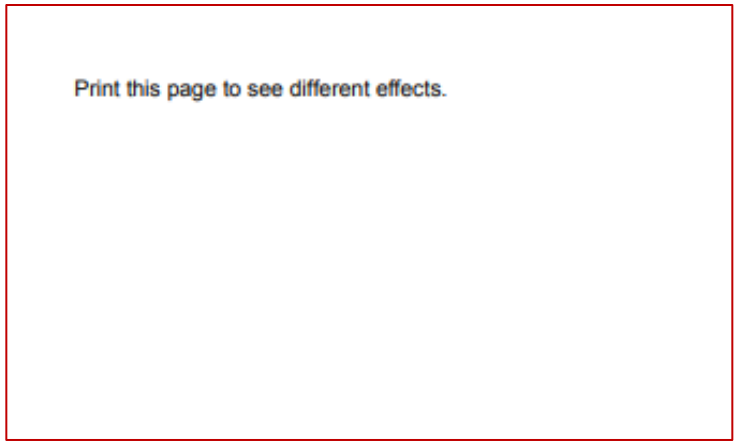
```
@media (max-width: 576px)
{
  body {
    font-size: 14px;
  }
}
```

```
@media print {
  body {
    font-size: 12pt;
    color: black;
    background: none;/*bỏ nền*/
  }
}
```

Media Query - Example

```
<style>
  /* Default */
  body {
      font-family: Arial;
      background: lightblue;
  }

  /* Print */
  @media print {
    body { background: none; color: black;}
  }
</style>
```





2.3. Flexbox

- flex: flex-grow flex-shrink flex-basis;
  - flex-grow → how much the item will grow relative to others
  - flex-shrink → how much the item will shrink relative to others
  - flex-basis → the initial size of the item

```
.item1 { flex: 1 1 200px; }
/* flex-grow = 1, flex-shrink = 1, flex-basis = 200px */

.item2 { flex: 2; }
/* shorthand for: flex-grow = 2, flex-shrink = 1, flex-basis = 0 */
```

- `flex: 1 1 200px;`

  → The item starts with 200px width, can grow to fill extra space, and can shrink if needed.
- `flex: 2;`

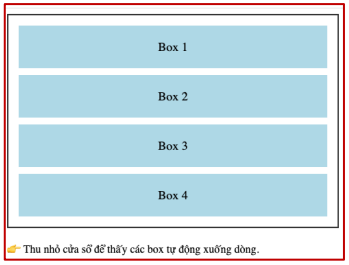
  → The item will grow at twice the rate of items with `flex: 1`.

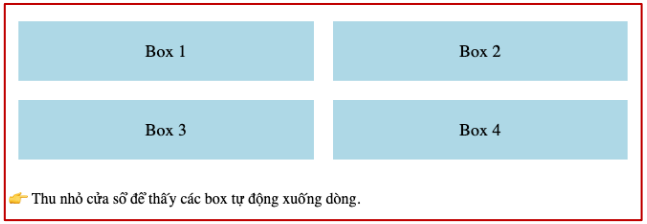
2.3. Flexbox

- Flexbox is a one-dimensional layout system (rows or columns).
- Works well for **responsive layouts** because elements can wrap and align easily.
- Main properties
  - flex-wrap: wrap; → Items automatically move to a new line if space is not enough.
  - justify-content → Align items horizontally
  - align-items → Align items vertically within the flex container

Example

```
  <style>
    .container {
      display: flex;
      flex-wrap: wrap;              /* Cho phép xuống dòng khi thiếu chỗ */
      justify-content: space-between; /* Căn đều ngang, có khoảng cách */
    }
    .box {
      flex: 1 1 200px;  /* Grow, shrink, base width */
      margin: 10px;
      background-color: lightblue;
      text-align: center;
      padding: 20px;
      font-size: 18px;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="box">Box 1</div>
    <div class="box">Box 2</div>
    <div class="box">Box 3</div>
    <div class="box">Box 4</div>
  </div>
  <p>👉 Thu nhỏ cửa sổ để thấy các box tự động xuống dòng.</p>
</body>
```





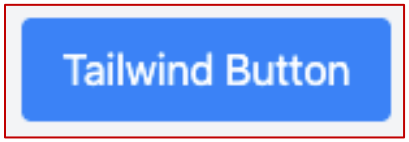
HUST

3. CSS Framework

What is Tailwind CSS?

- A utility-first CSS framework
- Focuses on building UI directly in HTML without writing new CSS files.
- Benefits
  - Fast development: uses predefined utility classes
  - Consistency: avoid messy custom CSS
  - Maintainability: easy to read and modify

```
<button class="bg-blue-500 text-white py-2 px-4 rounded">
  Tailwind Button
</button>
```



- background color: Tailwind's blue at level 500
- text: while color
- py-2: vertical padding (top, bottom) to 0.5rem
- px-4: horizontal paddign (left, right) to 1 rem

How to install Tailwind CSS

1. Using CDN (Quick Demo)

- Easiest way to start without setup.
- Add this line inside <head>:

```
<script src="https://cdn.tailwindcss.com"></script>
```

2. Other ways

https://tailwindcss.com/docs/installation/tailwind-cli

Typography

- Text Size
  - text-lg → Large text (≈18px).
  - text-2xl → Extra large text (≈24px).
- Font Weight
  - font-bold → Bold text.
- Text Alignment
  - text-center → Centers the text horizontally.
  - Other options: text-left, text-right, text-justify
- Text Style
  - italic → Italic text.

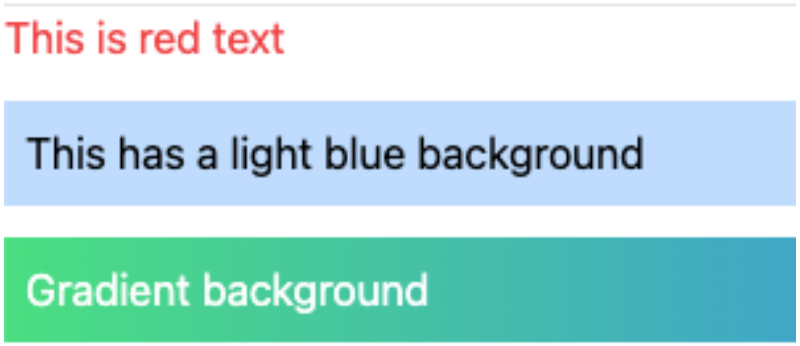
Typography Example

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Tailwind Typography Simple</title>
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="p-6 font-sans">

  <p class="text-lg">Large text (text-lg)</p>
  <p class="text-2xl font-bold">Extra large & bold (text-2xl font-bold)</p>
  <p class="text-center">This text is centered (text-center)</p>
  <p class="italic">Italic text (italic)</p>
  <p class="underline">Underlined text (underline)</p>

</body>
</html>
```

Color

- text-red-500 → Red text.
- bg-blue-200 → Light blue background.
- bg-gradient-to-r → Gradient background from left to right.

```
<body class="space-y-3">

  <p class="text-red-500">This is red text</p>
  <p class="bg-blue-200 p-2">This has a light blue background</p>
  <div class="bg-gradient-to-r from-green-400 to-blue-500 text-white p-2">
    Gradient background
  </div>

</body>
```



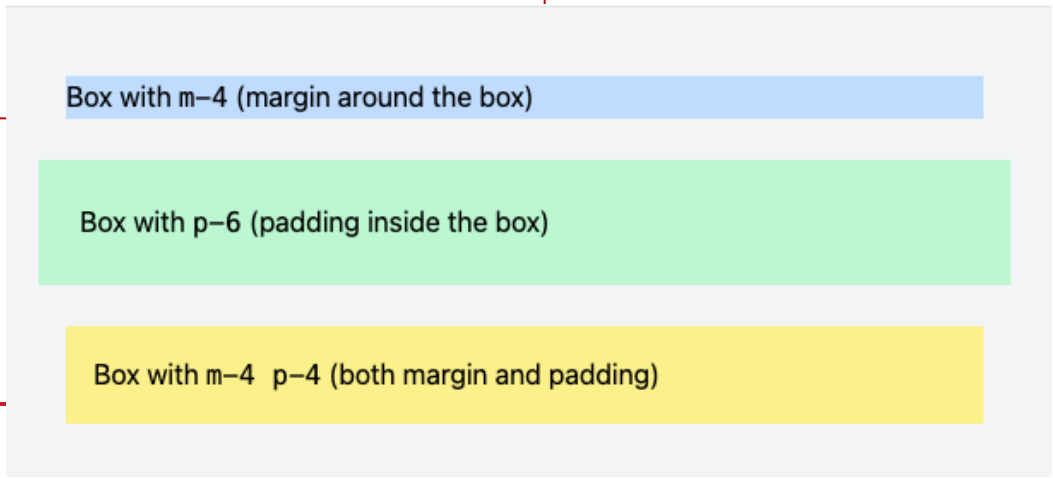
Spacing

- **Margin (m) Utilities**
  - m-4 → Applies margin on **all sides** (1rem ≈ 16px).
  - mt-2 → Margin on **top only** (0.5rem ≈ 8px).
  - mx-auto → Horizontal margin set to **auto**, often used for centering block elements.

- **Padding (p) Utilities**
  - p-4 → Padding on **all sides** (1rem ≈ 16px).
  - px-6 → Horizontal padding (left & right = 1.5rem).
  - py-2 → Vertical padding (top & bottom = 0.5rem).

Spacing Example

```
<body class="p-6 bg-gray-100 font-sans">

  <!-- Margin Example -->
  <div class="bg-blue-200 m-4">
    <p>Box with <code>m-4</code> (margin around the box)</p>
  </div>

  <!-- Padding Example -->
  <div class="bg-green-200 p-6 mt-6">
    <p>Box with <code>p-6</code> (padding inside the box)</p>
  </div>

  <!-- Combined Example -->
  <div class="bg-yellow-200 m-4 p-4 mt-6">
    <p>Box with <code>m-4 p-4</code> (both margin and padding)</p>
  </div>

</body>
```

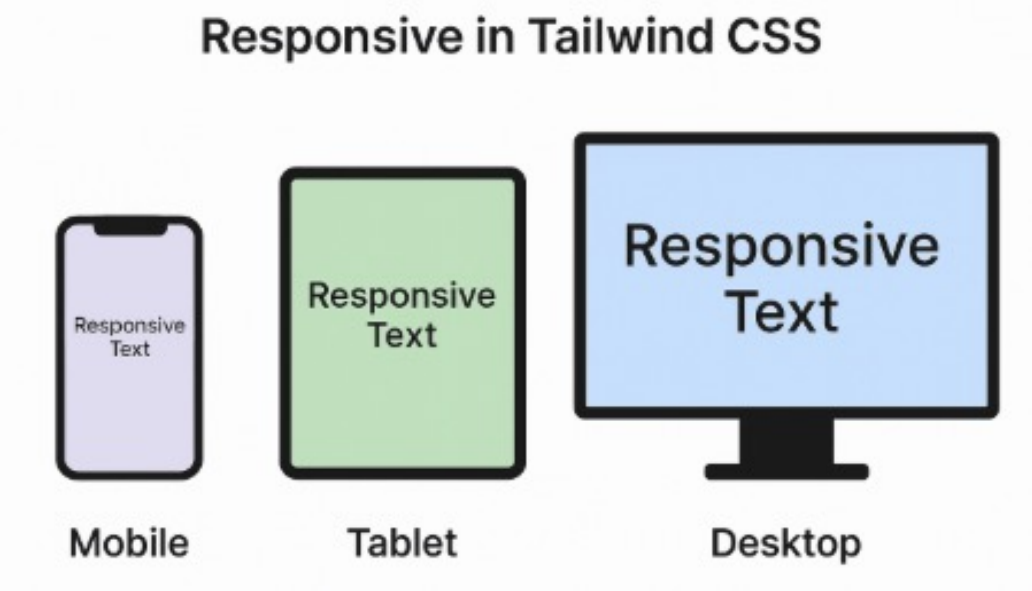


Responsive in Tailwind CSS

- Tailwind provides responsive utility classes using prefixes.
- Prefixes apply styles only when the screen width is greater than or equal to a certain breakpoint.
- Common Breakpoints
  - sm: → ≥ 640px (small devices, tablets in portrait)
  - md: → ≥ 768px (medium devices, tablets in landscape)
  - lg: → ≥ 1024px (large devices, laptops)

Responsive in Tailwind CSS - Example

```
<!-- Responsive Box -->
<div class="w-32 h-32 bg-purple-400 sm:bg-green-400 lg:bg-blue-400">
</div>
```



Exercises

- Ex1. Create a simple page based on HTML and CSS
- Ex2. Create a page following responsive web design

Exercises

- https://www.freecodecamp.org/learn/2022/responsive-web-design/
- Exercises 3, 4:
  - Learn HTML by Building a Cat Photo App
  - Build a Personal Portfolio Webpage