**MỤC LỤC**

[CHƯƠNG 1. CÚ PHÁP 4](#_Toc200965230)

[1.1. GIỚI THIỆU 5](#_Toc200965231)

[1.1.1. Quy tắc đặt tên BEM (Block Element Modifier) 5](#_Toc200965232)

[1.2. Selectors 7](#_Toc200965233)

[1.2.1. Selector cơ bản 7](#_Toc200965234)

[1.2.2. Selector kết hợp 7](#_Toc200965235)

[1.2.3. Pseudo-class (lớp giả) 8](#_Toc200965236)

[1.2.4. Pseudo-element (phần tử giả) 9](#_Toc200965237)

[1.2.5. Attribute selector (theo thuộc tính) 9](#_Toc200965238)

[1.3. Sepecificity 10](#_Toc200965239)

[1.3.1. Lưu ý 10](#_Toc200965240)

[1.4. Box Model 11](#_Toc200965241)

[1.4.1. Box-sizing 12](#_Toc200965242)

[1.4.2. Margin collapse (Hiện tượng "sập lề" trong CSS) 13](#_Toc200965243)

[CHƯƠNG 2. Layout 16](#_Toc200965244)

[2.1. Flex box 16](#_Toc200965245)

[2.2. GRID 16](#_Toc200965246)

[CHƯƠNG 3. Responsive Design 17](#_Toc200965247)

[3.1. Media queries 17](#_Toc200965248)

[3.2. Mobile first 17](#_Toc200965249)

[CHƯƠNG 4. Animation 18](#_Toc200965250)

[4.1. Keyframs 18](#_Toc200965251)

[4.2. Trasitions 18](#_Toc200965252)

[CHƯƠNG 5. Tìm hiểu về css framework 19](#_Toc200965253)

[5.1. Cơ bản về tailwindCSS. 19](#_Toc200965254)

**MỤC LỤC HÌNH ẢNH**

[Hình 1.1 Quy tắc đặt biến định kiểu CSS. 4](#_Toc200869078)

[Hình 1.2 Sơ đồ minh họa box model 10](#_Toc200869079)

**MỤC LỤC BẢNG BIỂU**

[Bảng 1.1 Quy tắc đặt tên BEM 5](#_Toc200869068)

[Bảng 1.2 Ví dụ cho bem 5](#_Toc200869069)

[Bảng 1.3 Các slector cơ bản 6](#_Toc200869070)

[Bảng 1.4 Các slector kết hợp 6](#_Toc200869071)

[Bảng 1.5 Các secletor cho các lớp giả 7](#_Toc200869072)

[Bảng 1.6 Các selector cho phần tử giả 8](#_Toc200869073)

[Bảng 1.7 Các selector theo thuộc tính 8](#_Toc200869074)

[Bảng 1.8 Thứ tự ưu tiên của các thành phần css 9](#_Toc200869075)

# CÚ PHÁP

Lưu ý: Các kiến thức cần nắm

**BEM**: block element và modifier (khối thành phần và đặc điểm của khối hoặc thành phần)

**SELECTORS:** Cần học đầy đủ cả các loại:

* Basic (element, class, id)
* Combinators (descendant div p, child div > p)
* Attribute selectors [type="text"]
* Pseudo-classes (:hover, :nth-child(), v.v.)
* Pseudo-elements (::before, ::after)

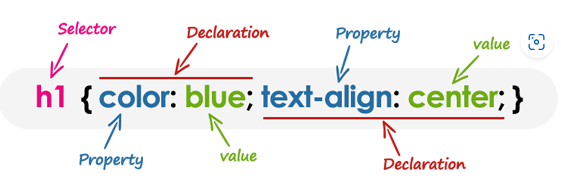
**SPECIFICITY:** Quan trọng, cần làm bài tập để hiểu rõ mức độ ưu tiên.

**BOX MODEL:** Rất cần thiết, nên bổ sung thêm các kiến thức như:

* margin collapse
* box-sizing: border-box

## GIỚI THIỆU

Cú pháp của biến định kiểu css là một tập hợp các quy tắc được web diễn giải và áp dụng cho các thẻ html.



Hình 1.1 Quy tắc đặt biến định kiểu CSS.

* CSS đều phân biệt chữ hoa chữ thường.
* Có thể lồng nhau giữa các biến định kiểu.

### Quy tắc đặt tên BEM (Block Element Modifier)

Tại sao phải dùng BEM:

Khi sử dụng css thường xuyên gặp các vấn đề như là:  
⇨ Cấu trúc html lồng nhau khi thay đổi một cấu trúc html thì css sẽ không còn hoạt động đúng như yêu cầu dẫn đến code khó bào trì và mở rộng.

.header ul li a span {

color: red;

}.

* Style bị ghi đè ngoài ý muốn. Code không có tổ chức. Phụ thuộc quá nhiều vào HTML DOM.
* Từ đó Quy tắc BEM ra đời. ngoài quy tắc BEM vẫn còn nhiều quy tắc đặt tên cho class.

Cú pháp chung của BEM là: .block\_\_element—modifier

Trong đó:

|  |  |  |
| --- | --- | --- |
| **Thành phần** | **Ý nghĩa** | **Ví dụ** |
| Block | Thành phần độc lập, có thể tái sử dụng | menu, button, form |
| Element | Thành phần con phụ thuộc vào block | menu\_\_item, form\_\_input |
| Modifier | Trạng thái, phiên bản của block hoặc element | button--active, menu\_\_item--highlighted |

Bảng 1.1 Quy tắc đặt tên BEM

Ví dụ:



Bảng 1.2 Ví dụ cho bem

* Dấu cách trong thuộc tính class là biểu thị cho nhiều class như ví dụ trên thì li đầu tiên có 2 class menu\_item và menu\_item—active.

#### Lưu ý khi sử dụng BEM

👉 Đặt tên theo **ngữ nghĩa**, **chức năng**, **vai trò**, không theo màu sắc hay kích thước.

👉 Không dùng **BEM** cho những thứ nhỏ nhặt ví dụ các thẻ hr br.

👉 Chỉ sử dụng **class selectors** trong CSS.

## Selectors

**Selector** (bộ chọn) là phần dùng để **chọn ra các phần tử HTML mà bạn muốn áp dụng kiểu dáng (style)**.

### Selector cơ bản

|  |  |  |  |
| --- | --- | --- | --- |
| **Loại Selector** | **Cú pháp** | **Ý nghĩa** | **Ví dụ** |
| Phần tử | element | Chọn tất cả thẻ có tên tương ứng | p, h1, div |
| Class | .class | Chọn phần tử có class cụ thể | .menu, .title |
| ID | #id | Chọn phần tử có id cụ thể | #header, #main |
| Tất cả phần tử | \* | Chọn mọi phần tử trong tài liệu | \* { margin: 0; } |

Bảng 1.3 Các slector cơ bản

#### Lưu ý khi sử dụng selector

* Không nên viết selector lồng nhiều cấp dễ bị phụ thuộc cấu trúc DOM thay vì vậy ta nên sử dụng selector class với quy tắc BEM để style cho tags.

### Selector kết hợp

|  |  |  |
| --- | --- | --- |
| **Loại kết hợp** | **Cú pháp** | **Ý nghĩa** |
| **Chọn trong nhau** | div p | Chọn tất cả p nằm trong div |
| **Chọn trực tiếp** | div > p | Chọn p là con **trực tiếp** của div |
| **Chọn anh em liền kề** | h2 + p | Chọn p đứng **ngay sau** h2 , và cả hai đều có cùng một phần tử cha. |
| **Chọn tất cả anh em sau** | h2 ~ p | Chọn tất cả p đứng **sau** h2 cùng cấp, và cả hai đều có cùng một phần tử cha. |

Bảng 1.4 Các slector kết hợp

### Pseudo-class (lớp giả)

Lớp giả giúp bạn định kiểu cho một phần tử khi nó ở một trạng thái đặc biệt hoặc mối quan hệ nhất định với các phần tử khác.

|  |  |
| --- | --- |
| **Selector** | **Ý nghĩa** |
| :active | Chọn một phần tử khi nó đang được kích hoạt (ví dụ: khi nhấp chuột và giữ trên một nút). |
| :hover | Khi trỏ chuột vào phần tử |
| :focus | Chọn một phần tử khi nó nhận được tiêu điểm (ví dụ: khi nhấp vào một trường nhập liệu hoặc điều hướng đến nó bằng phím Tab). |
| :visited | Chọn một liên kết đã được truy cập. |
| :first-child | Phần tử đầu tiên trong cha |
| :last-child | Phần tử cuối cùng trong cha |
| :nth-child(n) | Phần tử thứ n trong cha |
| :not(selector) | Chọn tất cả trừ những cái khớp với selector |

Bảng 1.5 Các secletor cho các lớp giả

### Pseudo-element (phần tử giả)

Phần tử giả giúp bạn định kiểu cho **các phần cụ thể bên trong một phần tử** hoặc để **thêm nội dung trang trí** mà không cần sửa đổi HTML.

|  |  |
| --- | --- |
| **Selector** | **Ý nghĩa** |
| ::before | Thêm nội dung **trước** phần tử |
| ::after | Thêm nội dung **sau** phần tử |
| ::first-line | Áp dụng cho dòng đầu tiên của đoạn văn |
| ::first-letter | Áp dụng cho chữ cái đầu của đoạn văn |

Bảng 1.6 Các selector cho phần tử giả

#### Lưu ý sử dụng Phần tử giả

* Bắt buộc phải có thuộc tính content khi sử dụng phần tử giả.

::before và ::after chỉ hoạt động trên các phần tử có thể chứa nội dung (ví dụ: div, p, span, a, button). Chúng không hoạt động trên các phần tử rỗng (<img>, <input>, <br>).

* Không thêm vào DOM thực. Vì chỉ tạo phần tử giả nên nó không thêm thẻ html thật nên ko tạo ra DOM.

### Attribute selector (theo thuộc tính)

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| [type] | Chọn phần tử có thuộc tính type |
| [type="text"] | Chọn phần tử có type=text |
| [href^="https"] | Bắt đầu bằng |
| [href$=".pdf"] | Kết thúc bằng |
| [href\*="google"] | Chứa từ "google" trong giá trị thuộc tính |

Bảng 1.7 Các selector theo thuộc tính

## Sepecificity

**Specificity** (độ ưu tiên của bộ chọn) là cơ chế mà **trình duyệt dùng để quyết định style nào sẽ được áp dụng**, khi có **nhiều rule CSS cùng áp dụng cho một phần tử HTML**.

|  |  |  |
| --- | --- | --- |
| **Nhóm** | **Loại selector** | **Giá trị tăng thêm** |
| a | Inline style | +1000 |
| b | ID selector (#id) | +100 |
| c | Class, attribute, pseudo-class (.class, [type], :hover) | +10 |
| d | Phần tử, pseudo-element (div, p, ::before) | +1 |

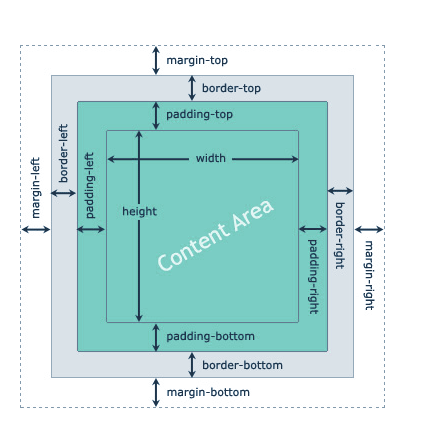
Bảng 1.8 Thứ tự ưu tiên của các thành phần css

### Lưu ý

**Tránh lạm dụng ID**: Bộ chọn ID có specificity cao, có thể gây khó khăn khi cần ghi đè (override) quy tắc.

**Sử dụng class thay vì ID**: Class có specificity thấp hơn, dễ bảo trì và tái sử dụng hơn.

## Box Model



Hình 1.2 Sơ đồ minh họa box model

Mô hình hộp CSS (CSS Box Model) là một khái niệm mô tả cách các phần tử được cấu trúc và hiển thị trên một trang web.

Mỗi phần tử HTML có thể được hình dung như **một hộp chữ nhật**, và mô hình hộp này định nghĩa các phần khác nhau tạo nên tổng không gian mà một phần tử chiếm giữ.

Gồm 4 phần:

* Content: nội dung kiểm soát ⇨ width và height.
* Padding (đệm):
* Boder (viền)
* Margin (lề)

### Box-sizing

Thuộc tính box-sing giúp kiểm soát box model hay kiểm soát kích thước của một khối.

Mặc định : **box-sizing: content-box** ⇨ Chỉ bao gồm phần content.

Mà thực tế cái khối ta có cả phần : lề, đệm. ⇨ nếu không quản lý thì khi mà thay kích thước ta phải điều chỉnh cả 3 thông số trên.

Tổng kích thước thực tế = content width + padding + border

Vây có cách nào? ⇨ **box-sizing: border-box**.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 1.3 Ví dụ về box-sizing

A screenshot of a computer

AI-generated content may be incorrect.

Hình 1.4 Ví dụ box\_sizing

### Margin collapse (Hiện tượng "sập lề" trong CSS)

**Margin collapse** xảy ra khi **hai (hoặc nhiều) margin theo chiều dọc** tiếp xúc nhau — **trình duyệt sẽ không cộng dồn mà lấy giá trị lớn hơn**.

* 3 trường hợp phổ biến:
* TH1: Margin giữa 2 phần tử liền kề (siblings)
* TH2: Margin của phần tử con và cha
* TH3: Phần tử rỗng không có border, padding, hoặc nội dung

TH1: Ví dụ

A screenshot of a computer

AI-generated content may be incorrect.

Hình 1.5 Hiện tượng sập lề TH1

TH2: Ví dụ

A screenshot of a computer program

AI-generated content may be incorrect.

Hình 1.6 Hiện tượng sập lề CSS TH2

TH3:

A screenshot of a chat

AI-generated content may be incorrect.

Hình 1.7 Hiện tượng sập lề CSS TH3

Cách xử lý: Sử dụng boder và padding thay cho margin.

Dùng overflow: hidden hoặc overflow: auto ⇨ Tạo block formatting context mới.

Dùng display: flex hoặc grid ⇨ Các layout hiện đại không bị collapse.

Tại sao dùng display: flex hoặc grid ko còn hiện tượng gộp lề:

|  |  |
| --- | --- |
| **Cơ chế Layout** | **Gọi là gì** |
| Mặc định (block/inline) | **Normal Flow** |
| Flexbox | **Flex Formatting Context** |
| Grid | **Grid Formatting Context** |
| Overflow, float, etc. | Tạo **BFC** (Block Formatting Context) |

Bảng 1.9 Cơ chế Layout

* Mỗi cơ chế thì có quy tắc riêng. Vì vậy khi gộp lề ta đang ở chế độ mặc định ⇨ dùng display: flex ta đang ở mặc định.

# Layout

Lưu ý: Các kiến thức cần nắm

**FLEXBOX**: Rất quan trọng trong thiết kế hiện đại. Nên thực hành tạo layout đơn giản như navbar, card...

**GRID:** Khuyến khích học sau Flexbox, dùng cho layout phức tạp hơn như gallery, dashboard...

## Flex box

Flexbox (viết tắt của Flexible Box Layout Module) là một mô hình bố cục một chiều (one-dimensional layout model) trong CSS.

"Một chiều" có nghĩa là nó quản lý việc phân phối không gian và căn chỉnh các mục (items) dọc theo **một hàng** hoặc **một cột** tại một thời điểm.

* **Flexbox** là cung cấp một cách hiệu quả để sắp xếp, phân phối và căn chỉnh các phần tử **trong một container**, ngay cả khi kích thước của chúng không rõ hoặc thay đổi động.

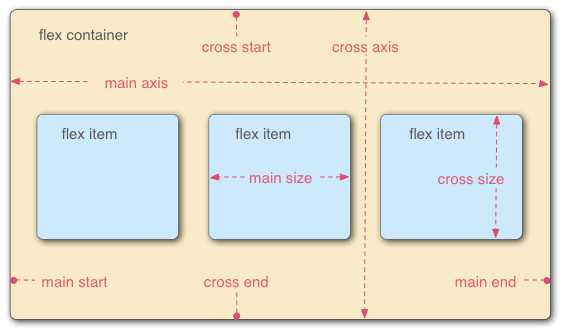
### Thành phần và cơ chế chính.

Thành phần chính:

* Flex Container (Cha): Một phần tử **có display: flex** hoặc **display: inline-flex.**
* Flex Items (Con) : Là các phần tử con trực tiếp bên trong Flex Container.

Cơ chế chính: Dựa trên hai trục từ đó ta có thao tác bằng thuộc tính để căn chỉnh layout theo bố cục.

* Trục Chính (Main Axis) và Trục Phụ (Cross Axis)



Hình 2.1 Truch trong flex box.

**Trục Chính (Main Axis):** Là trục mà các flex item được sắp xếp theo. Hướng của trục chính được xác định bởi thuộc tính **flex-direction** trên container.

**Trục Phụ (Cross Axis):** Là trục vuông góc với trục chính.

### Các thuộc tính thường sử dụng.

#### Thuộc tính CSS flex-box container.

|  |  |  |  |
| --- | --- | --- | --- |
| Tên Thuộc tính | Mô tả chi tiết | Giá trị phổ biến & Ý nghĩa | Ghi chú quan trọng |
| display | Biến phần tử thành một Flex Container, định nghĩa một ngữ cảnh định dạng flex cho các phần tử con trực tiếp của nó. | flex: Container là block-level (chiếm toàn bộ chiều rộng).  inline-flex: Container là inline-level (chỉ chiếm đủ chiều rộng). | Đây là thuộc tính bắt buộc để kích hoạt Flexbox. Và chỉ dùng cho container. |
| flex-direction | Xác định hướng của trục chính mà các flex item sẽ được sắp xếp. | row (mặc định): Hàng ngang, từ trái sang phải.  row-reverse: Hàng ngang, từ phải sang trái.  column: Cột dọc, từ trên xuống dưới.  column-reverse: Cột dọc, từ dưới lên trên. | Rất quan trọng để thiết lập hướng bố cục cơ bản của bạn. |
| flex-wrap | Kiểm soát việc các flex item có được phép xuống dòng (wrap) khi không còn đủ không gian trên một hàng/cột hay không. | nowrap (mặc định): Các item cố gắng nằm trên một dòng duy nhất.  wrap: Các item sẽ xuống dòng khi tràn.  wrap-reverse: Các item xuống dòng theo hướng ngược lại. | Thường dùng với các bố cục responsive để tránh tràn. |
| justify-content | Căn chỉnh các flex item dọc theo trục chính của container. | flex-start (mặc định): Về đầu trục chính.  flex-end: Về cuối trục chính.  center: Căn giữa.  space-between: Phân phối đều, item đầu ở đầu, item cuối ở cuối.  space-around: Phân phối đều, khoảng trống bằng nhau ở hai bên mỗi item.  space-evenly: Khoảng trống giữa các item và giữa item với cạnh container là như nhau. | Kiểm soát vị trí ngang (nếu flex-direction: row) hoặc dọc (nếu flex-direction: column). |
| align-items | Căn chỉnh các flex item dọc theo trục phụ (vuông góc với trục chính) trong một dòng duy nhất. | stretch (mặc định): Kéo dài để lấp đầy container (nếu không có kích thước cố định).  flex-start: Về đầu trục phụ.  flex-end: Về cuối trục phụ.  center: Căn giữa.  baseline: Căn chỉnh theo đường cơ sở văn bản. | Kiểm soát vị trí dọc (nếu flex-direction: row) hoặc ngang (nếu flex-direction: column). |
| align-content | Căn chỉnh các dòng flex (flex lines) dọc theo trục phụ khi có nhiều hơn một dòng (chỉ có tác dụng khi flex-wrap: wrap;). | stretch (mặc định): Các dòng kéo dài  flex-start: Về đầu trục phụ.  flex-end: Về cuối trục phụ.  center: Căn giữa.  space-between: Phân phối đều, dòng đầu ở đầu, dòng cuối ở cuối.  space-around: Phân phối đều, khoảng trống bằng nhau ở hai bên mỗi dòng. | Không có tác dụng nếu chỉ có một dòng flex (tức là flex-wrap: nowrap; hoặc chỉ có đủ không gian cho một dòng). |
| gap | Thiết lập khoảng cách (gutter) giữa các hàng và cột của flex items. | <length> (ví dụ: 10px)  <row-gap>  <column-gap> (ví dụ: 10px 20px) | Là thuộc tính hiện đại, được ưu tiên hơn việc dùng margin để tạo khoảng cách giữa các item để tránh các vấn đề gộp lề. |
| row-gap | Thiết lập khoảng cách giữa các hàng của flex items (một phần của gap). | <length> (ví dụ: 10px) | Hữu ích khi bạn muốn khoảng cách hàng và cột khác nhau. |
| column-gap | Thiết lập khoảng cách giữa các cột của flex items (một phần của gap). | <length> (ví dụ: 20px) | Hữu ích khi bạn muốn khoảng cách hàng và cột khác nhau. |

Bảng 2.1 Thuộc tính flex-box Container

#### Thuộc tính CSS flex-box item.

|  |  |  |  |
| --- | --- | --- | --- |
| Tên Thuộc Tính | Mô Tả Chi Tiết | Giá Trị Phổ Biến & Ý Nghĩa | LƯU Ý |
| order | Xác định thứ tự hiển thị của một flex item, độc lập với thứ tự của nó trong mã HTML. | <integer> (mặc định: 0)  Các số nguyên âm (ví dụ: -1, -2) sẽ làm item xuất hiện sớm hơn.  Các số nguyên dương (ví dụ: 1, 2) sẽ làm item xuất hiện muộn hơn. | Hữu ích khi bạn muốn thay đổi thứ tự trực quan của các phần tử mà không cần sửa đổi cấu trúc HTML. Các item có giá trị order thấp hơn sẽ luôn xuất hiện trước. |
| flex-grow | Xác định khả năng của một flex item để "lớn lên" và chiếm thêm không gian trống còn lại trong flex container. | <number> (mặc định: 0).  0: Item sẽ không giãn ra, chỉ giữ kích thước cơ sở.  1: Item sẽ giãn ra theo tỷ lệ.  2: Item sẽ giãn ra gấp đôi so với một item có flex-grow: 1. | Thuộc tính này chỉ có tác dụng khi có không gian trống còn lại trên trục chính của container. Nó kiểm soát cách không gian thừa được phân phối giữa các item. |
| flex-shrink | Xác định khả năng của một flex item để "co lại" khi không có đủ không gian trong flex container (ví dụ: khi kích thước màn hình nhỏ hơn tổng kích thước các item). | <number> (mặc định: 1).  0: Item sẽ không bao giờ co lại dưới kích thước cơ sở của nó (có thể gây tràn nếu không gian quá hẹp)  .1: Item sẽ co lại theo tỷ lệ. | Thuộc tính này chỉ có tác dụng khi không có đủ không gian trên trục chính. Nó kiểm soát cách không gian cần thiết để thu hẹp được phân bổ giữa các item. |
| flex-basis | Xác định kích thước mặc định của một flex item trước khi không gian trống được phân phối (bởi flex-grow) hoặc bị thu hẹp (bởi flex-shrink). Nó hoạt động tương tự width hoặc height, nhưng dành riêng cho flex items. | <length> (ví dụ: 100px, 50%, 10rem)  .auto (mặc định): Kích thước cơ sở được lấy từ kích thước nội dung của item hoặc các thuộc tính width/height của nó.  content: Kích thước dựa trên nội dung. | Thuộc tính này ưu tiên hơn width hoặc height cho flex items trên trục chính. Nó là điểm khởi đầu cho việc tính toán kích thước của item trong bố cục flex. |
| flex | Là thuộc tính viết tắt của flex-grow, flex-shrink, và flex-basis. Cú pháp: flex: <flex-grow> <flex-shrink> <flex-basis>; | flex: initial; (tương đương 0 1 auto): Item sẽ co lại nếu cần nhưng không giãn ra.  flex: auto: (tương đương 1 1 auto): Item có thể giãn ra hoặc co lại nếu cần.  flex: none; (tương đương 0 0 auto): Item không co lại cũng không giãn ra, giữ kích thước gốc  .flex: 1; (tương đương 1 1 0%): Item sẽ giãn ra và co lại, lấy 0% làm kích thước cơ sở. | Đây là thuộc tính rất khuyến khích sử dụng vì nó ngắn gọn và xử lý tốt các trường hợp phổ biến.  flex: 1; là giá trị rất phổ biến để các item chia đều không gian còn lại một cách linh hoạt. |
| align-self | Cho phép ghi đè thuộc tính align-items của flex container cho một flex item cụ thể. | auto (mặc định): Kế thừa giá trị từ align-items của container  .flex-start: Căn về đầu trục phụ  .flex-end: Căn về cuối trục phụ  .center: Căn giữa trên trục phụ.baseline: Căn chỉnh theo đường cơ sở văn bản.  stretch: Kéo dài để lấp đầy container (nếu không có kích thước cố định). | Hữu ích khi bạn muốn một item cụ thể có cách căn chỉnh khác biệt so với các item còn lại trong cùng một dòng flex. |

Bảng 2.2 Thuộc tính flex-box Item

A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 2.2 Thuộc tính flexbox

## GRID

CSS Grid Layout là một hệ thống bố cục hai chiều (two-dimensional layout system) cho web.

Điều này có nghĩa là nó cho phép bạn sắp xếp các phần tử không chỉ theo hàng (ngang) mà còn theo cột (dọc) cùng một lúc.

* Mục đích chính của Grid là giải quyết các bài toán bố cục tổng thể của trang web (page layout) hoặc các thành phần lớn, phức tạp có cấu trúc dạng lưới.
* Được sử dụng: Dashboard, gallery, hoặc bố cục trang chính (header, sidebar, content, footer)

### So sánh Grid và Flex-box

* Flexbox: Là mô hình một chiều (hàng HOẶC cột). Nó tối ưu cho việc phân phối và căn chỉnh các nhóm phần tử trong một dòng/cột.
* Grid: Là mô hình hai chiều (hàng VÀ cột). Nó tối ưu cho việc sắp xếp các phần tử vào một cấu trúc lưới tổng thể, kiểm soát cả vị trí ngang và dọc.

Thực tế, bạn sẽ thường **kết hợp cả Flexbox và Grid** để xây dựng các giao diện hiện đại. Grid tạo ra khung sườn chính, Flexbox quản lý các chi tiết bên trong từng ô.

### Các thành phần chính.

### Các thuộc tính dành cho Grid Container.

### Các thuộc tính dành cho Grid Item.

# Responsive Design

Lưu ý: Các kiến thức cần nắm

**MEDIA QUERIES:** Cần học rõ cú pháp, breakpoints phổ biến.

**MOBILE FIRST:** Nên giải thích rõ triết lý "mobile-first" là gì (viết CSS cho màn hình nhỏ trước, sau đó mở rộng dần).

## Media queries

## Mobile first

# Animation

Lưu ý kiến thức cần nắm:

**KEYFRAMES:** Cần minh họa với ví dụ sinh động như loading spinner, fade in/out.

**TRANSITIONS:** Nên học rõ các thuộc tính có thể transition và cách kiểm soát timing (ease-in, ease-out, cubic-bezier, v.v.)

## Keyframs

## Trasitions

# Tìm hiểu về css framework

Lưu ý kiến thức cần nắm:

**TAILWINDCSS:** Rất phù hợp, nhưng cần đảm bảo bạn đã hiểu CSS thuần trước.

* Nên học: cấu trúc class tiện ích, cách cấu hình, responsive trong Tailwind
* Có thể mở rộng thêm Bootstrap để so sánh cách tiếp cận giữa "utility-first" và "component-based"

## Cơ bản về tailwindCSS.