

CHƯƠNG TRÌNH HỌC CSS TOÀN DIỆN

Mã sinh viên:

Họ và tên:

HỌC PHẦN 1: NỀN TẢNG VỮNG CHẮC (The Foundations)

Mục tiêu: Sinh viên hiểu được CSS là gì, cú pháp ra sao và cách áp dụng các quy tắc tạo kiểu cho những phần tử HTML đơn giản nhất.

- **Bài 1: Giới thiệu về CSS**
 - CSS là gì? Tại sao cần CSS? (Tách biệt nội dung HTML và trình bày)
CSS (Cascading Style Sheets) là ngôn ngữ chuyên dùng để định dạng, trình bày, và trang trí giao diện các phần tử HTML trong trang web
 - Lịch sử phát triển ngắn gọn (CSS1, CSS2, CSS3).

CSS trải qua các phiên bản:

CSS1 (1996): Đặt nền móng các quy tắc cơ bản.

CSS2 (1998): Bổ sung nhiều thuộc tính mới như bố cục trình bày, hỗ trợ định vị, tab, v.v.

CSS3 (2001 trở về sau): Chia thành nhiều module nhỏ, tăng cường hiệu ứng (transition, animation, responsive design).

- Minh họa một trang web "trước" và "sau" khi có CSS.

HỌC PHẦN 1: NỀN TẢNG VỮNG CHẮC (The Foundations)

Mục tiêu: Sinh viên hiểu được CSS là gì, cú pháp ra sao và cách áp dụng các quy tắc tạo kiểu cho những phần tử HTML đơn giản nhất.

- **Bài 1: Giới thiệu về CSS**
 - CSS là gì? Tại sao cần CSS? (Tách biệt nội dung HTML và trình bày)
CSS (Cascading Style Sheets) là ngôn ngữ chuyên dùng để định dạng, trình bày, và trang trí giao diện các phần tử HTML trong trang web
 - Lịch sử phát triển ngắn gọn (CSS1, CSS2, CSS3).

CSS trải qua các phiên bản:

CSS1 (1996): Đặt nền móng các quy tắc cơ bản.

CSS2 (1998): Bổ sung nhiều thuộc tính mới như bố cục trình bày, hỗ trợ định vị, tab, v.v.

CSS3 (2001 trở về sau): Chia thành nhiều module nhỏ, tăng cường hiệu ứng (transition, animation, responsive design).

- Minh họa một trang web "trước" và "sau" khi có CSS.

(không có CSS):

◦

Sau khi có CSS:

◦

○

- **Bài 2: 3 Cách Thêm CSS vào Trang Web**

- **Inline CSS:** style attribute (Dùng khi nào, và tại sao nên hạn chế).

Sử dụng thuộc tính style ngay trên thẻ HTML. Ví dụ: `<p style="color: red;">Text</p>`.

Cách này nên hạn chế vì khó bảo trì, khó chỉnh sửa toàn dự án và dễ bị lặp lại.

- **Internal CSS:** Thẻ `<style>` trong `<head>` (Dùng cho một trang duy nhất).

Viết quy tắc CSS trong thẻ `<style>` nằm ở phần `<head>` của tài liệu HTML. Cách này phù hợp với các trang đơn lẻ, hoặc những trường hợp muốn tùy chỉnh giao diện một trang cụ thể.

- **External CSS:** File .css riêng biệt và thẻ `<link>` (Phương pháp tốt nhất - Best Practice).

Tạo file .css riêng, liên kết bằng thẻ `<link rel="stylesheet" href="style.css">` trong `<head>`. Đây là phương pháp tốt nhất, giúp tái sử dụng quy tắc, quản lý giao diện cho toàn bộ website một cách linh hoạt, chuyên nghiệp.

- **Bài 3: Cú pháp và Quy tắc Cơ bản**

- Quy tắc CSS: `Selector { property: value; }`
- Giải thích từng thành phần: Bộ chọn (Selector), Thuộc tính (Property), Giá trị (Value).
- Cách viết comment trong CSS (`/* ... */`).

- **Bài 4: Bộ chọn (Selectors) - Phần 1: Các Bộ chọn Cốt lõi**

- Bộ chọn theo Thẻ (Type/Element Selector): `p`, `h1`, `div`.
- Bộ chọn theo Class: `.ten-class`.
- Bộ chọn theo ID: `#mot-id-duy-nhat`.
- Bộ chọn Universal: `*`.

- **Bài 5: Độ ưu tiên (Specificity) và Thác nước (Cascade)**

- "Thác nước" hoạt động như thế nào? (Quy tắc sau đè quy tắc trước).
- Độ ưu tiên: `ID > Class > Element`. Giải thích bằng hệ thống điểm số đơn giản (ví dụ: `ID=100`, `Class=10`, `Element=1`).
- `!important`: Khi nào nên và không nên dùng (nhấn mạnh là "vạn bất đắc dĩ").

- **Bài 6: Bộ chọn (Selectors) - Phần 2: Kết hợp và Nâng cao**

- Bộ chọn Nhóm (Grouping): `h1, h2, h3`.
- Bộ chọn Con cháu (Descendant): `div p`.
-

- **Bài 2: 3 Cách Thêm CSS vào Trang Web**

- **Inline CSS:** style attribute (Dùng khi nào, và tại sao nên hạn chế).
- **Internal CSS:** Thẻ `<style>` trong `<head>` (Dùng cho một trang duy nhất).

- **External CSS:** File .css riêng biệt và thẻ <link> (Phương pháp tốt nhất - Best Practice).

- **Bài 3: Cú pháp và Quy tắc Cơ bản**

- Quy tắc CSS: Selector { property: value; }
- Giải thích từng thành phần: Bộ chọn (Selector), Thuộc tính (Property), Giá trị (Value).
- Cách viết comment trong CSS (*/* ... */*).

- **Bài 4: Bộ chọn (Selectors) - Phần 1: Các Bộ chọn Cốt lõi**

- Bộ chọn theo Thẻ (Type/Element Selector): p, h1, div.
- Bộ chọn theo Class: .ten-class.
- Bộ chọn theo ID: #mot-id-duy-nhat.
- Bộ chọn Universal: *.

- **Bài 5: Độ ưu tiên (Specificity) và Thác nước (Cascade)**

- "Thác nước" hoạt động như thế nào? (Quy tắc sau đè quy tắc trước).
- Độ ưu tiên: ID > Class > Element. Giải thích bằng hệ thống điểm số đơn giản (ví dụ: ID=100, Class=10, Element=1).
- !important: Khi nào nên và không nên dùng (nhấn mạnh là "vạn bất đắc dĩ").

- **Bài 6: Bộ chọn (Selectors) - Phần 2: Kết hợp và Nâng cao**

- Bộ chọn Nhóm (Grouping): h1, h2, h3.
- Bộ chọn Con cháu (Descendant): div p.

- Bộ chọn Con trực tiếp (Child): `ul > li`.
 - Bộ chọn Giả-lớp (Pseudo-classes) tương tác: `:hover`, `:focus`, `:active`.
 - Bộ chọn Giả-lớp cấu trúc: `:first-child`, `:last-child`, `:nth-child()`.
-

HỌC PHẦN 2: "TRANG ĐIỂM" CHO PHẦN TỬ (Styling Elements)

Mục tiêu: Sinh viên có thể định dạng các yếu tố phổ biến nhất trên trang web như văn bản, màu sắc, nền và hiểu được khái niệm quan trọng nhất: Box Model.

- **Bài 7: Làm việc với Màu sắc**

- Các thuộc tính: `color` (màu chữ), `background-color` (màu nền).
- Các định dạng màu: Tên màu (`red`), Mã HEX (`#ff0000`), RGB (`rgb(255,0,0)`), RGBA (`rgba(255,0,0,0.5)`) - giải thích về kênh alpha/độ trong suốt).

- **Bài 8: Định dạng Văn bản và Font chữ**

- `font-family`: Web-safe fonts và cách nhúng Google Fonts.
- `font-size`: Các đơn vị `px`, `em`, `rem` (giới thiệu sơ bộ).
- `font-weight`: `normal`, `bold`, các giá trị số.
- `font-style`: `normal`, `italic`.
- `text-align`: `left`, `right`, `center`, `justify`.
- `line-height`: Giãn cách dòng.
- `text-decoration`, `text-transform`.

- **Bài 9: Mô hình Hộp (The Box Model) - Khái niệm Tối quan trọng**

- Giải thích 4 lớp: `Content` -> `Padding` -> `Border` -> `Margin`.
- `width` và `height`.
- `padding` (Vùng đệm): Khoảng trống bên trong viền.
- `border`: `border-width`, `border-style`, `border-color`.
- `margin` (Lề): Khoảng trống bên ngoài viền.
- Cách viết tắt cho `padding` và `margin` (1, 2, 3, 4 giá trị).
- `border-radius`: Bo tròn góc.

- **Bài 10: box-sizing: border-box - "Vị cứu tinh"**

- Giải thích vấn đề của `box-sizing`: `content-box` (mặc định).
- Minh họa `box-sizing`: `border-box` giúp việc tính toán kích thước dễ dàng hơn như thế nào.
- Khuyến khích sinh viên luôn sử dụng nó cho mọi dự án.

- **Bài 11: Thuộc tính display**

- block: Chiếm toàn bộ chiều rộng, bắt đầu trên dòng mới (div, p, h1).
- inline: Chỉ chiếm chiều rộng cần thiết, nằm trên cùng một dòng (span, a, img).
- inline-block: Kết hợp ưu điểm của cả hai.
- none: Ẩn hoàn toàn phần tử.

HỌC PHẦN 3: XÂY DỰNG BỐ CỤC TRANG WEB (Layout)

Mục tiêu: Sinh viên học các kỹ thuật sắp xếp, bố trí các phần tử trên trang, từ cổ điển đến hiện đại (Flexbox & Grid).

- **Bài 12: Định vị (Positioning)**

- position: static (Mặc định).
- position: relative: "Điểm tựa" cho absolute.
- position: absolute: Định vị theo "tổ tiên" relative gần nhất.
- position: fixed: Định vị theo cửa sổ trình duyệt (viewport).
- position: sticky: Hiệu ứng "dính" lại khi cuộn.
- Các thuộc tính top, right, bottom, left và z-index.

- **Bài 13: Flexbox - Công cụ Layout Một chiều**

- Giới thiệu về Flexbox và trục chính/trục phụ.
- **Thuộc tính cho Container (cha):** display: flex, flex-direction, justify-content, align-items, flex-wrap.
- **Thuộc tính cho Items (con):** flex-grow, flex-shrink, flex-basis, order.
- Các ví dụ thực tế: Căn giữa một phần tử, tạo thanh điều hướng (navigation bar).

- **Bài 14: Grid - Công cụ Layout Hai chiều**

- So sánh Grid và Flexbox.
- **Thuộc tính cho Container (cha):** display: grid, grid-template-columns, grid-template-rows, gap, grid-template-areas.
- Đơn vị fr.
- **Thuộc tính cho Items (con):** grid-column, grid-row.
- Ví dụ thực tế: Xây dựng bố cục layout chính của một trang web.

HỌC PHẦN 4: THIẾT KẾ ĐÁP ỨNG (Responsive Web Design)

Mục tiêu: Sinh viên có thể làm cho trang web hiển thị tốt trên mọi kích thước màn hình, từ điện thoại đến máy tính để bàn.

- **Bài 15: Giới thiệu Responsive Design**
 - Viewport là gì? Tầm quan trọng của thẻ `<meta name="viewport">`.
 - Khái niệm "Mobile First".
 - **Bài 16: Media Queries**
 - Cú pháp `@media`.
 - Các điểm ngắt (Breakpoints) phổ biến.
 - Ví dụ: Thay đổi bố cục từ 1 cột (mobile) sang 3 cột (desktop).
 - **Bài 17: Hình ảnh và Video Đáp ứng**
 - Kỹ thuật `max-width: 100%;` và `height: auto;`.
 - Thuộc tính `object-fit` cho ảnh.
 - **Bài 18: Các Đơn vị Tương đối**
 - Ôn lại và đi sâu vào: `%`, `em`, `rem`, `vw`, `vh`.
 - Tại sao nên dùng `rem` cho `font-size`.
-

HỌC PHẦN 5: HIỆU ỨNG VÀ KỸ THUẬT NÂNG CAO

Mục tiêu: Thêm các hiệu ứng chuyển động và các kỹ thuật CSS hiện đại để làm trang web sống động và chuyên nghiệp hơn.

- **Bài 19: Các Bộ chọn Giả-phần tử (Pseudo-elements)**
 - `::before` và `::after`: Cách sử dụng với thuộc tính `content`.
 - Ví dụ: Tạo icon trang trí, dấu ngoặc kép cách điệu.
 - **Bài 20: Biến đổi (Transforms)**
 - `transform: translate()` (di chuyển), `rotate()` (xoay), `scale()` (phóng to/thu nhỏ).
 - **Bài 21: Chuyển tiếp (Transitions)**
 - Tạo hiệu ứng chuyển động mượt mà khi thay đổi thuộc tính (ví dụ: khi `:hover`).
 - `transition-property`, `transition-duration`, `transition-timing-function`.
 - **Bài 22: Hoạt ảnh (Animations) với @keyframes**
 - Tạo các hoạt ảnh phức tạp hơn `transition`.
 - Cú pháp `@keyframes` và thuộc tính `animation`.
 - **Bài 23: Biến trong CSS (CSS Variables)**
 - Cú pháp: `--ten-bien: gia-tri;` và `var(--ten-bien)`.
 - Lợi ích trong việc quản lý màu sắc, font chữ (theming).
-

HỌC PHẦN 6: DỰ ÁN THỰC TẾ

Mục tiêu: Tổng hợp tất cả kiến thức đã học để xây dựng một dự án hoàn chỉnh từ đầu.

- **Bài 24: Dự án cuối kỳ - Xây dựng Trang Portfolio Cá nhân**

- Yêu cầu: Trang web phải có header, footer, phần giới thiệu, danh sách dự án, form liên hệ.
- Bắt buộc phải responsive.
- Khuyến khích sử dụng Flexbox/Grid cho layout, có hiệu ứng transition khi di chuột.
- Đây sẽ là bài kiểm tra tổng hợp tốt nhất cho sinh viên.

Lời khuyên cho bạn:

- **Thực quan là Vua:** Với CSS, hãy luôn dùng các ví dụ trực quan. Các công cụ như [CodePen](#) rất tuyệt vời để minh họa trực tiếp.
- **Tập trung vào "Tại sao":** Thay vì chỉ nói "dùng display: flex", hãy giải thích "tại sao flex lại giải quyết được vấn đề mà float hay position gặp khó khăn".
- **Thực hành, thực hành, thực hành:** Mỗi bài học nên có một "Phiếu Học Tập" đi kèm, bắt buộc sinh viên phải viết code và thấy kết quả. Dự án cuối kỳ là không thể thiếu.

Video đã xem (Link):

1. Kiến thức cốt lõi (Tóm tắt bằng lời của bạn)

Sau khi xem video, hãy *thi* diễn giải 2 khái niệm quan trọng nhất:

1. CSS là gì? (Nó là viết tắt của chữ gì và mục đích chính của nó là gì?)

CSS là viết tắt của "Cascading Style Sheets". Đây là ngôn ngữ dùng để định dạng và thiết kế giao diện trang web, kiểm soát màu sắc, bố cục, font chữ, khoảng cách và nhiều yếu tố hiển thị khác.

2. Tại sao chúng ta cần CSS? (Nó giải quyết vấn đề gì cho HTML?)

CSS được dùng để tách biệt phần nội dung (HTML) với phần trình bày. Nếu chỉ có HTML thì trang web chỉ hiển thị nội dung thô, thiếu màu sắc và hình thức. CSS giúp các trang web trở nên sinh động, trực quan, đẹp mắt và dễ sử dụng hơn, đồng thời giúp quản lý giao diện cho nhiều trang dễ dàng, không phải chỉnh từng dòng HTML khi muốn đổi kiểu hiển thị.

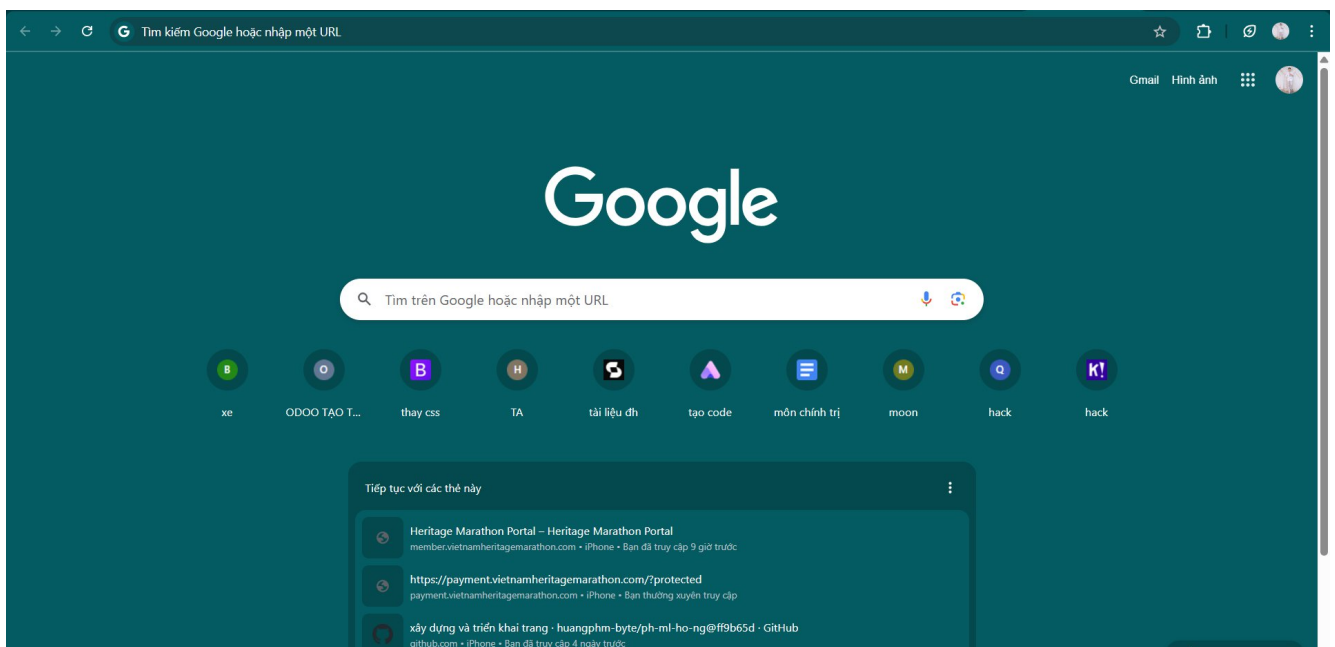
2. Ví dụ thực hành (Bắt buộc)

Bài tập này không cần viết code, chỉ cần quan sát để hiểu rõ vai trò của CSS.

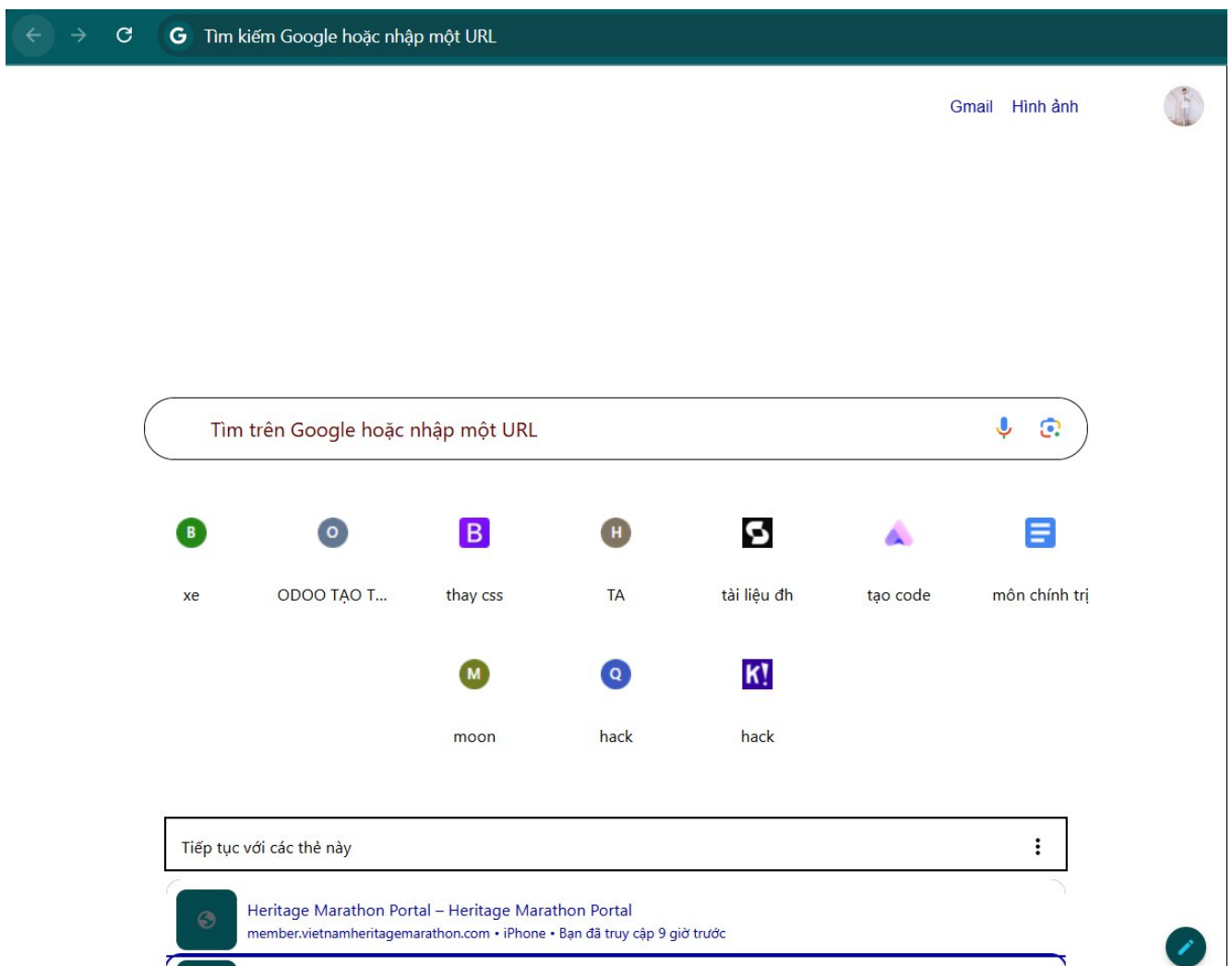
- Bước 1:** Mở một trang web bất kỳ mà bạn hay truy cập (ví dụ: vnexpress.net, dantri.com.vn, hoặc youtube.com).
- Bước 2:** Chụp ảnh màn hình toàn bộ trang web khi nó hiển thị bình thường.
- Bước 3:** Sử dụng công cụ của trình duyệt để **TẮT (Disable)** toàn bộ CSS của trang web đó.
 - Gợi ý: Dùng phím F12 (Developer Tools), tìm một extension (như "Web Developer" extension) và chọn "Disable All Styles".
- Bước 4:** Chụp ảnh màn hình trang web sau khi đã tắt CSS (bạn sẽ chỉ thấy nội dung HTML thô, đen trắng).

KẾT QUẢ QUAN SÁT:

Ảnh 1: Trang web KHI CÓ CSS (Hiển thị bình thường) (Dán ảnh chụp màn hình của bạn vào đây)



Ảnh 2: Trang web KHI ĐÃ TẮT CSS (Chỉ còn HTML) (Dán ảnh chụp màn hình của bạn vào đây)



Nhận xét (Bắt buộc): Hãy mô tả 3 điều khác biệt lớn nhất bạn thấy giữa 2 bức ảnh. Điều này cho bạn biết CSS thffic sffi làm gì?

1. trang có CSS hiển thị màu sắc, bố cục rõ ràng và các phần được căn chỉnh đẹp mắt; còn trang không có CSS chỉ là văn bản và hình ảnh xếp dọc đơn giản.
2. Khi tắt CSS, toàn bộ màu nền, font chữ, nút bấm và thanh tìm kiếm mất định dạng — trông thô, khó sử dụng.

3. CSS giúp trang web có giao diện thân thiện, dễ nhìn và chuyên nghiệp, chứ không chỉ là hiển thị nội dung HTML.

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em thấy trang web sau khi tắt CSS vẫn có một chút định dạng (chữ to, chữ nhỏ), tại sao vậy?" hoặc "HTML và CSS luôn phải đi chung với nhau hay có thể dùng riêng?")*

PHIẾU HỌC TẬP CSS [02] - 3 CÁCH THÊM CSS VÀO WEB

Video đã xem (Link):

1. Kiến thức cốt lõi (Tóm tắt bằng lời của bạn)

Sau khi xem video, hãy hoàn thành bảng so sánh 3 cách thêm CSS dưới đây:

Tên phương pháp	Cách viết (Cú pháp)	Ưu điểm (Nên dùng khi nào?)	Nhược điểm (Tại sao nên tránh?)
1. Inline CSS	(Viết cú pháp ví dụ vào đây)	Dễ thử nghiệm nhanh, áp dụng riêng lẻ cho một phần tử cụ thể.	Khó bảo trì, lặp lại mã nhiều, không tách biệt nội dung và trình bày.
2. Internal CSS	(Viết cú pháp ví dụ vào đây)	Dễ dùng cho một trang đơn lẻ, có thể định dạng cùng lúc nhiều phần tử.	Không tái sử dụng được cho nhiều trang khác, file HTML bị dài.
3. External CSS	(Viết cú pháp ví dụ vào đây)	Quản lý, tái sử dụng và bảo trì dễ dàng; áp dụng được cho nhiều trang.	Cần nhiều file, phải đảm bảo đường dẫn đúng.

Kết luận quan trọng nhất: Phương pháp nào là tốt nhất (best practice) và được khuyên dùng trong hầu hết các dự án? Tại sao?

pháp tốt nhất (best practice) là External CSS, vì tách biệt rõ nội dung (HTML) và trình bày (CSS), dễ bảo trì, dễ mở rộng cho các dự án lớn, phù hợp tiêu chuẩn chuyên nghiệp trong phát triển web.

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Tạo 2 file (*index.html* và *style.css*) và áp dụng cả 3 phương pháp CSS để tạo ra kết quả như mô tả.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 02</title>
  <style>

  </style>
</head>
<body>
  <p>Đoạn 1: Tôi được style bằng INLINE CSS.</p>
  <p class="internal-style">Đoạn 2: Tôi được style bằng INTERNAL CSS.</p>
  <p class="external-style">Đoạn 3: Tôi được style bằng EXTERNAL CSS.</p>
</body>
```

</html>

Mã CSS của tôi (style.css):

```
.external-style {  
    font-size: 24px;  
}
```

CSS

/* TODO 3: Viết quy tắc CSS cho class .external-style vào đây */

```
.external-style {  
  font-size: 24px;  
}
```

Kết quả (Chụp ảnh màn hình và dán vào đây):



(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:

- Dòng 1 có chữ màu đỏ.
- Dòng 2 có nền màu vàng.
- Dòng 3 có chữ kích thước 24px.

)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Nếu em dùng cả 3 cách để cùng style cho 1 phần tử (ví dụ cùng đổi màu chữ) thì máy tính sẽ nghe theo cách nào? Cách nào có quyền ưu tiên cao nhất?")*

1. Kiến thức cốt lõi (Giải phẫu một quy tắc CSS)

Sau khi xem video, hãy điền tên chính xác cho từng thành phần trong hình ảnh quy tắc CSS mẫu dưới đây:

```
h1 { ... } color blue color: blue; /* Đây là... */
```

1. **h1** được gọi là: Selector (Bộ chọn)

2. **{ ... }** (Toàn bộ phần trong ngoặc nhọn) được gọi là: Declaration Block (Khối khai báo)

3. **color** được gọi là: Property (Thuộc tính)

4. **blue** được gọi là: Value (Giá trị)

5. **color: blue;** (Cả cụm) được gọi là: Declaration (Khai báo)

6. **/* Đây là... */** được gọi là: Comment (Chú thích)

Diễn giải (Bắt buộc):

- **Selector (Bộ chọn):** Dùng để làm gì?

Dùng để chỉ định *phần tử HTML* nào sẽ được áp dụng kiểu CSS.

- **Property (Thuộc tính):** Dùng để làm gì?

Dùng để xác định *thuộc tính cần thay đổi* (màu, kích thước, nền, lề, v.v...).

- **Value (Giá trị):** Dùng để làm gì?

Dùng để chỉ *giá trị cụ thể* của thuộc tính.

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết các quy tắc CSS (viết trong thẻ `<style>` - Internal CSS) để định dạng file HTML bên dưới theo đúng 3 yêu cầu.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 03</title>

  <style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY
      (Không sửa file HTML)
    */

    /* Yêu cầu 1:
      Làm cho TOÀN BỘ trang web (thẻ <body>)
```

có MÀU NỀN (background-color) là #f0f0f0 (một màu xám rất nhạt).

```

    */
body {
    background-color: #f0f0f0;
}

/* Yêu cầu 2:
    Làm cho TIÊU ĐỀ (thẻ <h1>)
    có MÀU CHỮ (color) là #4a4a4a (một màu xám đậm).
    */
h1 {
    color: #4a4a4a;
}

/* Yêu cầu 3:
    Làm cho ĐOẠN VĂN BẢN (thẻ <p>)
    có CỠ CHỮ (font-size) là 18px.
    */
p {
    font-size: 18px;
}

</style>
</head>
<body>

    <h1>Đây là Tiêu đề chính</h1>

    <p>Đây là đoạn văn bản đầu tiên. Cỡ chữ của nó cần được làm to hơn.</p>

    <p>Đây là đoạn văn bản thứ hai. Nó cũng sẽ bị ảnh hưởng bởi quy tắc CSS bạn viết.</p>

</body>
</html>

```

Kết quả (Chụp ảnh màn hình và dán vào đây): *(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:*



- *Nền trang web màu xám nhạt.*
 - *Tiêu đề <h1> màu xám đậm.*
 - *Cả 2 đoạn văn <p> có chữ to 18px.)*
-

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em quên dấu chấm phẩy (;) sau font-size: 18px thì điều gì sẽ xảy ra?", hoặc "Viết h1 và H1 trong CSS có khác gì nhau không?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (So sánh các bộ chọn)

Sau khi xem video, hãy hoàn thành bảng so sánh 4 bộ chọn cơ bản dưới đây:

Tên Bộ chọn	Cú pháp (Cách viết)	Mục đích sử dụng (Khi nào dùng?)
1. Universal (Toàn cục)	*	Áp dụng cho tất cả các phần tử trên trang web (dùng để đặt quy tắc chung, ví dụ font, margin, padding...).
2. Element (Thẻ)	p, h1, div	(Tự điền mục đích: Áp dụng cho TẤT CẢ các thẻ cùng loại, vd: mọi thẻ <p>)
3. Class (Lớp)	.noi-bat	(Tự điền mục đích: Áp dụng cho NHIỀU thẻ khác nhau mà bạn muốn chúng có chung 1 kiểu)
4. ID	#tieu-de-chinh)	(Tự điền mục đích: Áp dụng cho 1 và CHỈ 1 thẻ DUY NHẤT trên trang)

Câu hỏi bắt buộc:

- Đây là điểm khác biệt quan trọng nhất giữa Class và ID?

lass có thể dùng cho nhiều phần tử, còn ID chỉ nên dùng cho một phần tử duy nhất.

- Bạn có thể dùng *nhiều* Class cho 1 thẻ HTML không? (Ví dụ: <p class="chu-dam chu-nghieng">)

Có thể. Ví dụ: <p class="chu-dam chu-nghieng"> (thẻ này có cả 2 class).

- Bạn có thể dùng *nhiều* ID cho 1 thẻ HTML không?

Không nên. Theo quy tắc HTML, mỗi thẻ chỉ có 1 ID duy nhất.

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS trong thẻ <style> để định dạng file HTML bên dưới. **Không được sửa đổi file HTML**, chỉ được viết thêm CSS.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 04</title>
```

<style>

/* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

```
/* Yêu cầu 1 (Dùng bộ chọn Universal): Thiết lập
font chữ chung cho TOÀN BỘ trang là font 'Arial',
'sans-serif'.
*/
```

```
{
font-family: Arial, sans-serif;
}
```

```
/* Yêu cầu 2 (Dùng bộ chọn Element/Thẻ): Làm
cho TẤT CẢ các thẻ <p>
có giãn cách dòng (line-height) là 1.5.
*/
```

```
p {
line-height: 1.5;
}
```

```
/* Yêu cầu 3 (Dùng bộ chọn ID):
Làm cho thẻ có ID là "tieu-de-chinh"
có MÀU CHỮ (color) là 'navy' (xanh navy)
và có một đường gạch chân (text-decoration: underline).
*/
```

```
#tieu-de-chinh {
color: navy;
text-decoration: underline;
}
```

```
/* Yêu cầu 4 (Dùng bộ chọn Class):
Làm cho TẤT CẢ các phần tử có class "noi-bat" có MÀU
NỀN (background-color) là 'yellow'
và IN ĐẬM (font-weight: bold).
*/
```

```
.noi-bat {
background-color: yellow;
font-weight: bold;
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h1 id="tieu-de-chinh">Bài viết quan trọng về CSS</h1>
```

```
<p>
```

Học CSS là một điều rất thú vị. Hôm nay chúng ta học về

bộ chọn Class và

bộ chọn ID.

</p>

<p

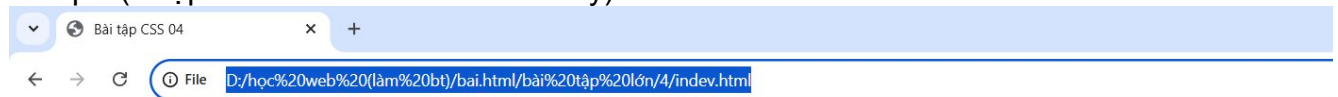
> Bạn phải nhớ rằng ID là duy nhất, trong khi Class có thể dùng lại nhiều lần.

</p>

</body>

</html>

Kết quả (Chụp ảnh màn hình và dán vào đây):



Bài viết quan trọng về CSS

Học CSS là một điều rất thú vị. Hôm nay chúng ta học về **bộ chọn Class** và **bộ chọn ID**.

Bạn phải nhớ rằng ID là **duy nhất**, trong khi Class có thể dùng lại nhiều lần.

(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:

- *Toàn bộ trang dùng font Arial.*
- *Cả 2 đoạn văn <p> đều dẫn dòng.*

- Tiêu đề <h1> màu xanh navy và có gạch chân.
 - Cả 3 từ/cụm từ có class "noi-bat" đều được bôi nền vàng và in đậm.)
-

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Nếu em lỡ đặt 2 ID giống nhau trên cùng 1 trang thì chuyện gì sẽ xảy ra? Trang web có bị lỗi không?", hoặc "Dùng Class hay ID tốt hơn cho hiệu suất trang web?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Ai là người "mạnh" nhất?)

Sau khi xem video, hãy điền vào chỗ trống:

1. Quy tắc Thác nước (Cascade):

- Khi hai quy tắc CSS có cùng độ ưu tiên (ví dụ 2 quy tắc p { ... } đều đổi màu chữ), thì quy tắc được định nghĩa ở đâu sẽ "thắng"?
 - Trả lời:** Quy tắc được định nghĩa _____ SAU (trước/sau) trong file CSS sẽ thắng.

2. Quy tắc Độ ưu tiên (Specificity):

- Khi các bộ chọn khác nhau cùng nhắm vào 1 phần tử, trình duyệt sẽ dùng một hệ thống "tính điểm" để xem ai "mạnh" hơn.
- Hãy sắp xếp các loại sau theo thứ tự ưu tiên từ CAO NHẤT đến THẤP NHẤT (1 là mạnh nhất, 5 là yếu nhất):**

Thứ hạng	Tên bộ chọn	Ví dụ cú pháp
1	Style Nội tuyến (Inline)	<p style="...">
(Điền 2)	ID _____	#my-id
(Điền 3)	Class, Pseudo-class, Attribute	.my-class, :hover
(Điền 4)	Element, Pseudo-element _____	p, div, ::before
(Điền 5)	Universal	*

3. "Kẻ hủy diệt" !important:

- !important dùng để làm gì?

Dùng để cưỡng chế một khai báo CSS, khiến khai báo đó ưu tiên hơn các khai báo cùng thuộc tính khác. tức là nó ghi đè hầu hết các quy tắc bình thường trong cascade.

- Tại sao chúng ta nên *hạn chế tối đa* việc sử dụng nó?

Vì:

Làm cho CSS khó bảo

Dẫn đến xung đột khi nhiều nơi dùng !important — gây phải dùng thêm !important khác để ghi đè → chu kỳ tòi tệ.

Làm giảm tính rõ ràng và lợi ích của hệ thống specificity/cascade.

→ Chỉ dùng !important khi vỡ trận (ví dụ muốn ghim tạm thời để override stylesheet từ thư viện bên ngoài) và sau đó tìm giải pháp sạch hơn.

2. Ví dụ thực hành (Bài tập Dự đoán s Kiểm chứng)

Yêu cầu: Tạo file HTML/CSS bên dưới, chạy và quan sát kết quả, sau đó giải thích tại sao kết quả lại như vậy.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

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

```
<title>Bài tập CSS 05</title>
```

```
<link rel="stylesheet" href="style.css">
```

```
</head>
```

```
<body>
```

```
<h1 id="tieu-de-chinh" class="tieu-de" style="color: purple;">
```

Tiêu đề này cuối cùng sẽ có màu gì?

```
</h1>
```

```
</body>
```

```
</html>
```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Bài tập CSS 05</title>
<link rel="stylesheet" href="style.css">
</head>
<body>

    <h1 id="tieu-de-chinh" class="tieu-de" style="color: purple;">
        Tiêu đề này cuối cùng sẽ có màu gì?
    </h1>

</body>
</html>

```

CSS

```
/* --- VIẾT 3 QUY TẮC NÀY VÀO FILE CSS --- */
```

```
/* Quy tắc 1 (Dùng Element) */
```

```
h1 {
    color: red;
}
```

```
/* Quy tắc 2 (Dùng Class) */
```

```
.tieu-de {
    color: blue;
}
```

```
/* Quy tắc 3 (Dùng ID) */
```

```
#tieu-de-chinh
{ color: green;
}
```

THỰC HÀNH VÀ TRẢ LỜI:

1. Kết quả quan sát (Chụp ảnh màn hình):



Tiêu đề này cuối cùng sẽ có màu gì?

(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Bạn thấy tiêu đề có màu gì?)

2. Giải thích kết quả:

- Tại sao tiêu đề lại có màu đỏ? (Ví dụ: Tại sao nó không phải màu red, blue, hay green?)

Trả lời:

Có 3 quy tắc trong `style.css` (`element` → `class` → `id`). Nếu chỉ xét specificity thì `#tieu-de-chinh` (ID) có specificity cao hơn `.tieu-de` và `h1`, nên nếu không có inline thì màu xanh lá (`green`) từ ID sẽ thắng.

Tuy nhiên, thẻ có `style="color: purple;"` — đó là inline style, và inline có độ ưu tiên rất cao (đứng trên quy tắc external thông thường). Vì vậy inline (`purple`) thắng mọi quy tắc trong file CSS bình thường → tiêu đề hiển thị `purple`.

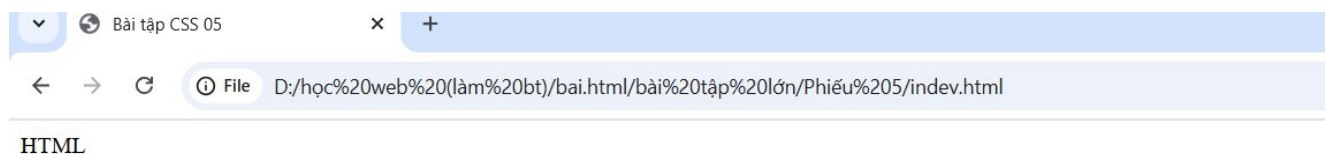
3. Thí nghiệm với `!important`:

- Yêu cầu:** Bây giờ, hãy quay lại file `style.css` và sửa **Quy tắc 1** thành:

CSS

```
h1 {  
  color: red !important;  
}
```

- Kết quả MỚI (Chụp ảnh màn hình):



Tiêu đề này cuối cùng sẽ có màu gì?

(Dán ảnh chụp màn hình kết quả MỞI của bạn vào đây)

- **Giải thích thí nghiệm:**
- Tại sao bây giờ màu red lại "thắng" tất cả các quy tắc khác, kể cả style inline?

Trả lời: Khi một khai báo được gắn !important, khai báo đó được tăng ưu tiên so với các khai báo không có !important, bất kể specificity thông thường (bao gồm inline style) — *trong cùng origin (author)*. Vì vậy `h1 { color: red !important; }` sẽ ghi đè `style="color: purple;"` và `#tieu-de-chinh { color: green; }` bình thường.

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Nếu em có 2 ID `#tieu-de-chinh {color: green;}` và `#tieu-de-chinh {color: orange;}` thì cái nào sẽ thắng? (Đây là câu hỏi về Cascade)", hoặc "Làm thế nào để tính điểm specificity chính xác khi bộ chọn phức tạp như `div#main p.content?`")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Các cách "phối hợp" bộ chọn)

Sau khi xem video, hãy điền vào bảng mô tả ý nghĩa của các bộ chọn kết hợp và trạng thái:

Tên Bộ chọn	Cú pháp (Ví dụ)	Ý nghĩa (Diễn giải bằng lời của bạn)
1. Grouping (Nhóm)	h1, h2, p	(Tự điền: Áp dụng style cho tất cả các thẻ h1 VÀ h2 VÀ p)
2. Descendant (Con cháu)	div p	(Tự điền: Chỉ chọn thẻ p nằm... div, bất kể là con hay cháu)
3. Child (Con trực tiếp)	ul > li	(Tự điền: Chỉ chọn thẻ li là... của ul)
4. Pseudo-class (Trạng thái)	a:hover	(Tự điền: Áp dụng style cho thẻ a chỉ khi...)

Câu hỏi bắt buộc:

- Đây là sự khác biệt lớn nhất giữa div p (Descendant) và div > p (Child)?

div p chọn mọi p bên trong div (kể cả p là cháu).
div > p chỉ chọn p là con trực tiếp của div.

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS trong thẻ <style> để định dạng file HTML bên dưới theo đúng 4 yêu cầu.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 06</title>
  <style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

    /* Yêu cầu 1 (Dùng bộ chọn Nhóm): Làm
       cho cả thẻ <h2> VÀ thẻ <h3>
       có chung MÀU CHỮ (color) là 'purple'.
    */
    h2, h3 {
      color: purple;
```

}

/* Yêu cầu 2 (Dùng bộ chọn Con cháu):

Làm cho TẤT CẢ các thẻ <p> NẴM BÊN TRONG

class "noi-dung-bai-viet" có chữ in nghiêng (font-style: italic).

(Phải ảnh hưởng cả 2 thẻ <p>)

```

        */
.noi-dung-bai-viet p {
    font-style: italic;
}

/* Yêu cầu 3 (Dùng bộ chọn Con trực tiếp):
    CHỈ làm cho các thẻ <li> LÀ CON TRỰC TIẾP
    của class "menu-chinh" có nền (background-color) là '#eee'.
    (Lưu ý: "Mục 2.1" không được có nền)
    */

.menu-chinh > li {
    background-color: #eee;
}

/* Yêu cầu 4 (Dùng bộ chọn Trạng thái):
    Làm cho cái nút (class "nut-bam")
    đổi MÀU NỀN thành 'black' và MÀU CHỮ thành 'white' KHI DI
    CHUỘT LÊN (hover).
    */

.nut-bam:hover {
    background-color: black;
    color: white;
}

</style>
</head>
<body>

<h2>Tiêu đề Bài viết</h2>
<div class="noi-dung-bai-viet">
    <p>Đây là đoạn văn 1 (con trực tiếp).</p>
    <section>
        <p>Đây là đoạn văn 2 (nằm sâu bên trong).</p>
    </section>
</div>

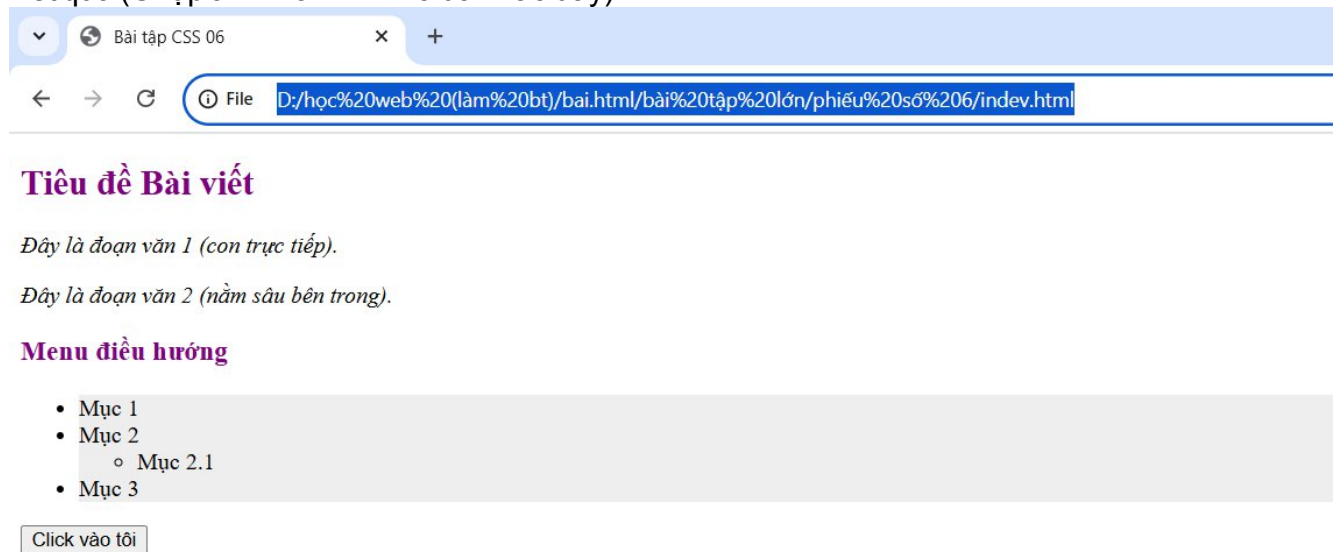
<h3>Menu điều hướng</h3>
<ul class="menu-chinh">
    <li>Mục 1</li>
    <li>Mục 2
        <ul>
            <li>Mục 2.1</li>
        </ul>
    </li>
    <li>Mục 3</li>
</ul>

<button class="nut-bam">Click vào tôi</button>

```

</body>
</html>

Kết quả (Chụp ảnh màn hình và dán vào đây):



(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:

- Tiêu đề <h2> và <h3> có màu tím.
- Cả 2 đoạn văn (Đoạn 1 và Đoạn 2) đều in nghiêng.
- Mục 1, Mục 2, Mục 3 có nền xám. "Mục 2.1" KHÔNG có nền.

- Khi di chuột vào nút, nút đổi thành nền đen chữ trắng.

)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Ngoài :hover ra thì còn trạng thái nào hay dùng không ạ? Em muốn style cho cái ô input khi mình click vào gõ chữ thì dùng gì?")*

PHIẾU HỌC TẬP CSS [07] - LÀM VIỆC VỚI MÀU SẮC

Video đã xem (Link):

1. Kiến thức cốt lõi (Các hệ màu trong CSS)

Sau khi xem video, hãy điền vào bảng mô tả các cách định nghĩa màu sắc trong CSS:

Tên hệ màu	Cú pháp (Ví dụ)	Ý nghĩa (Diễn giải)
1. Tên màu (Keywords)	color: red;	Dùng tên tiếng Anh định sẵn của màu sắc.
2. Mã HEX	color: #FF0000;	(Tự điền: Dùng mã 16... để biểu diễn 3 kênh màu R-G-B)
3. Hàm RGB	color: rgb(255, 0, 0);	(Tự điền: Dùng 3 giá trị từ 0-255 cho 3 kênh...)
4. Hàm RGBA	color: rgba(255, 0, 0, 0.5);	(Tự điền: Giống hệt RGB nhưng có thêm kênh thứ 4 là...)

Câu hỏi bắt buộc:

- Kênh "A" (Alpha) trong RGBA dùng để làm gì? Giá trị của nó chạy từ 0 đến 1, vậy 0.5 có nghĩa là gì?
- Hai thuộc tính cơ bản nhất để đổi màu trong CSS là gì?
 - color: Dùng để đổi màu chữ
 - background-color: Dùng để đổi màu nền

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS trong thẻ <style> để định dạng file HTML bên dưới. **Mỗi yêu cầu phải dùng một hệ màu khác nhau như chỉ định.**

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 07</title>
  <style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

    /* Yêu cầu 1 (Dùng Tên màu):
```

Làm cho tiêu đề <h1> có MÀU CHỮ (color) là 'darkblue'.

*/

```
h1 {  
  color: darkblue;  
  /* Yêu cầu 2 (Dùng mã HEX):  
    Làm cho đoạn văn có class "hop-hex"  
    có MÀU NỀN (background-color) là '#f0f8ff' (màu AliceBlue).  
    và MÀU CHỮ là '#333333' (màu xám đậm).
```

*/

```
.hop-hex {  
  background-color: #f0f8ff;  
  color: #333333;  
}
```

```
/* Yêu cầu 3 (Dùng hàm RGB):  
  Làm cho đoạn văn có class "hop-rgb"  
  có MÀU NỀN (background-color) là 'rgb(255, 228, 225)' (màu MistyRose).  
  và MÀU CHỮ là 'rgb(139, 0, 0)' (màu DarkRed).
```

*/

```
.hop-rgb {  
  background-color: rgb(255, 228, 225);  
  color: rgb(139, 0, 0);  
}
```

```
/* Yêu cầu 4 (Dùng hàm RGBA):  
  Làm cho đoạn văn có class "hop-rgba"  
  có MÀU NỀN (background-color) là 'rgba(0, 128, 0, 0.2)'.  
  (Đây là màu xanh lá cây (Green) nhưng CHỈ TRONG SUỐT 20%)
```

*/

```
.hop-rgba {  
  
  background-color: rgba(0, 128, 0, 0.2);  
  
}
```

</style>

</head>

<body>

<h1>Học về các hệ màu trong CSS</h1>

<div class="hop-hex">

<p>Hộp này dùng hệ màu HEX. Nền màu xám xanh nhạt (#f0f8ff) và chữ màu xám đậm (#333333).</p>

</div>

<div class="hop-rgb">

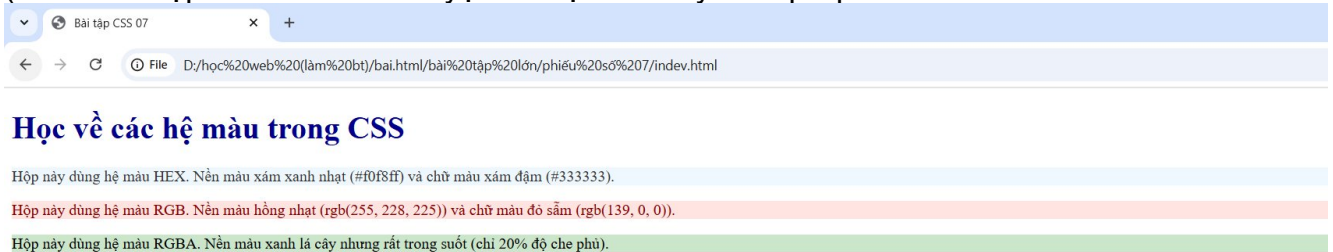
```
<p>Hộp này dùng hệ màu RGB. Nền màu hồng nhạt (rgb(255, 228, 225)) và chữ màu  
đỏ sẫm (rgb(139, 0, 0)).</p>  
</div>
```

```
<div class="hop-rgba">  
  <p>Hộp này dùng hệ màu RGBA. Nền màu xanh lá cây nhưng rất trong suốt (chỉ 20% độ  
che phủ).</p>  
</div>
```

```
</body>  
</html>
```

Kết quả (Chụp ảnh màn hình và dán vào đây):

(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:



- Tiêu đề <h1> có màu xanh đậm.
- Hộp 1 có nền xám xanh nhạt, chữ xám đậm.

- *Hộp 2 có nền hồng nhạt, chữ đỏ sẫm.*
- Hộp 3 có nền màu xanh lá cây rất nhạt/trong suốt.

)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em thấy mã HEX đôi khi có 3 chữ (ví dụ #FFF), đôi khi có 6 chữ (#FFFFFF). Chúng khác gì nhau?", hoặc "Làm thế nào để em biết được mã màu chính xác của một logo công ty?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Các thuộc tính "biên tập" văn bản)

Sau khi xem video, hãy điền vào cột "Mục đích sử dụng" bằng lời của bạn cho các thuộc tính sau:

Thuộc tính CSS	Mục đích sử dụng (Dùng để làm gì?)
font-family	(Tự điền: Dùng để thay đổi kiểu chữ (phông chữ))
font-size	(Tự điền: Dùng để thay đổi kích thước chữ.)
font-weight	(Tự điền: Dùng để thay đổi độ đậm/nhạt của chữ.)
font-style	(Tự điền: Dùng để làm chữ nghiêng)
text-align	(Tự điền: Dùng để căn lề văn bản.)
line-height	(Tự điền: Dùng để thay đổi khoảng cách giữa các dòng.)
text-decoration	(Tự điền: Dùng để thêm/bỏ gạch chân, gạch ngang....)
text-transform	(Tự điền: Dùng để biến đổi chữ hoa/thường biến đổi chữ hoa/thường.)

Câu hỏi bắt buộc:

- Để nhúng một font chữ đặc biệt từ Google Fonts (ví dụ: font 'Roboto'), bạn cần thực hiện 2 bước chính là gì?

1. Bước 1 (Trong file HTML):

Dán đoạn <link> do Google Fonts cung cấp vào trong thẻ <head>.

2. Bước 2 (Trong file CSS):

Dùng font-family: 'Roboto', sans-serif; để áp dụng font đó cho phần tử mong muốn.

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS trong thẻ <style> để định dạng file HTML bên dưới, tạo ra một bài viết được định dạng rõ ràng, dễ đọc.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 08</title>
```

<link rel,preconnect" href="https://fonts.googleapis.com">

```
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,400;0,700;1,400&dis
isplay=swap" rel="stylesheet">
```

```
<style>
```

```
/* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */
```

```
/* Yêu cầu 1 (Tiếp):
```

```
    Áp dụng font 'Roboto' cho toàn bộ trang (thẻ <body>) và đặt
    cỡ chữ cơ bản là 16px.
```

```
*/
```

```
body{
```

```
    font-family: 'Roboto', sans-serif; font-
    size: 16px;
```

```
}
```

```
/* Yêu cầu 2:
```

```
    Định dạng tiêu đề <h1>:
```

- Căn lề GIỮA (text-align)
- Chuyển thành CHỮ HOA (text-transform)
- In ĐẬM (font-weight: 700)

```
*/
```

```
h1 {
```

```
    text-align: center;
```

```
    text-transform: uppercase;
```

```
    font-weight: 700;
```

```
}
```

```
/* Yêu cầu 3:
```

```
    Định dạng tiêu đề <h2>:
```

- Chữ IN NGHIÊNG (font-style)
- Độ đậm 'normal' (font-weight: 400)
- Có một đường gạch chân (text-decoration)

```
*/
```

```
h2 {
```

```
    font-style: italic;
```

```
    font-weight: 400;
```

```
    text-decoration: underline;
```

```
}
```

```
/* Yêu cầu 4:
```

```
    Định dạng đoạn văn <p>:
```

- Giãn cách dòng (line-height) thành 1.6
 (gấp 1.6 lần cỡ chữ)

```
*/
```

```
p {
```

```
line-height: 1.6;
```

```
}
```

```
/* Yêu cầu 5:
```

```
    Định dạng thẻ <a> nằm trong <p>:
```

- Bỏ gạch chân MẶC ĐỊNH (text-decoration)
- Đổi màu chữ (color) thành 'darkgreen'

```
*/
```

```
p a {
```

```
    text-decoration: none;
```

```
    color: darkgreen;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h1>Tiêu đề chính của bài viết</h1>
```

```
<h2>Đây là tiêu đề phụ, mô tả ngắn</h2>
```

<p

>

Đây là nội dung của đoạn văn. Văn bản cần phải dễ đọc, vì vậy chúng ta sử dụng `line-height` để tăng khoảng cách giữa các dòng. Văn bản này cũng chứa một

liên kết (link).

Chúng ta cần bỏ gạch chân của liên kết này.

</p>

</body>

</html>

Kết quả (Chụp ảnh màn hình và dán vào đây):



TIÊU ĐỀ CHÍNH CỦA BÀI VIẾT

Đây là tiêu đề phụ, mô tả ngắn

Đây là nội dung của đoạn văn. Văn bản cần phải dễ đọc, vì vậy chúng ta sử dụng `line-height` để tăng khoảng cách giữa các dòng. Văn bản này cũng chứa một liên kết (link). Chúng ta cần bỏ gạch chân của liên kết này.

(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:

- Toàn bộ trang dùng font Roboto.
- Tiêu đề <h1> in hoa, đậm, căn giữa.
- Tiêu đề <h2> in nghiêng, không đậm, có gạch chân.
- Đoạn văn <p> có các dòng dẫn cách thoải mái.
- Chữ "liên kết (link)" có màu xanh lá đậm và KHÔNG bị gạch chân.

)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em thấy font-weight có lúc dùng chữ (bold), lúc dùng số (700). Chúng khác gì nhau?", hoặc "Các đơn vị px, em, rem cho font-size khác gì nhau và khi nào nên dùng loại nào?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Giải phẫu một "cái hộp")

Sau khi xem video, hãy điền tên 4 lớp của Mô hình Hộp vào đúng vị trí. Bắt đầu từ trong ra ngoài.

- **Lớp 1 (Trong cùng):** content (Là chữ, ảnh...)
- **Lớp 2 (Bao quanh Lớp 1):** padding (Vùng đệm - khoảng trống bên trong viền)
- **Lớp 3 (Bao quanh Lớp 2):** Border (Đường viền)
- **Lớp 4 (Ngoài cùng):** margin (Lề - khoảng trống bên ngoài viền)

Giải thích các thuộc tính:

- **width và height:** Dùng để thiết lập kích thước cho phần nội dung Content (Lớp 1 - Content).
 - **padding:** Dùng để tạo Khoảng trống (khoảng trống/không gian) giữa Content và Border.
 - **border:** Dùng để tạo đường viền. Ba thuộc tính con quan trọng là border-width (độ dày), border-style (kiểu viền), và border-color (màu viền).
 - **margin:** Dùng để tạo space (khoảng cách) giữa hộp này và các phần tử khác.
-

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS trong thẻ `<style>` để tạo ra một "tấm thẻ" (card) sản phẩm đơn giản, áp dụng đầy đủ các thuộc tính của Box Model.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 09</title>
  <style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

    /* Yêu cầu 1:
       Style cho thẻ .product-card (cái hộp chính)
    */
    .product-card {
      /* Lớp 1: Đặt chiều RỘNG (width) là 300px */
      width: 300px;
```

```
/* Lớp 2: Đặt VÙNG ĐỆM (padding) là 15px cho cả 4 cạnh */  
padding: 15px;
```

```

/* Lớp 3: Đặt VIỀN (border) dày 1px,
    kiểu 'solid' (nét liền), và màu '#ccc' (xám nhạt)
*/
border: 1px solid #ccc;

/* Lớp 4: Đặt LỀ (margin) là 20px cho cả 4 cạnh để nó
    cách xa mọi thứ xung quanh
*/
margin: 20px;
}

/* Yêu cầu 2:
    Style cho tiêu đề <h2> BÊN TRONG thẻ
    - Bỏ lề mặc định của trình duyệt (margin: 0;)
*/
.product-card h2
{ margin: 0;
  color: #333;
}

/* Yêu cầu 3:
    Style cho đoạn mô tả <p> BÊN TRONG thẻ
    - Đặt LỀ PHÍA TRÊN (margin-top) là 10px
    để nó cách tiêu đề <h2>
*/
.product-card p
{ margin-top: 10px;
  font-size: 14px;
  color: #666;
}

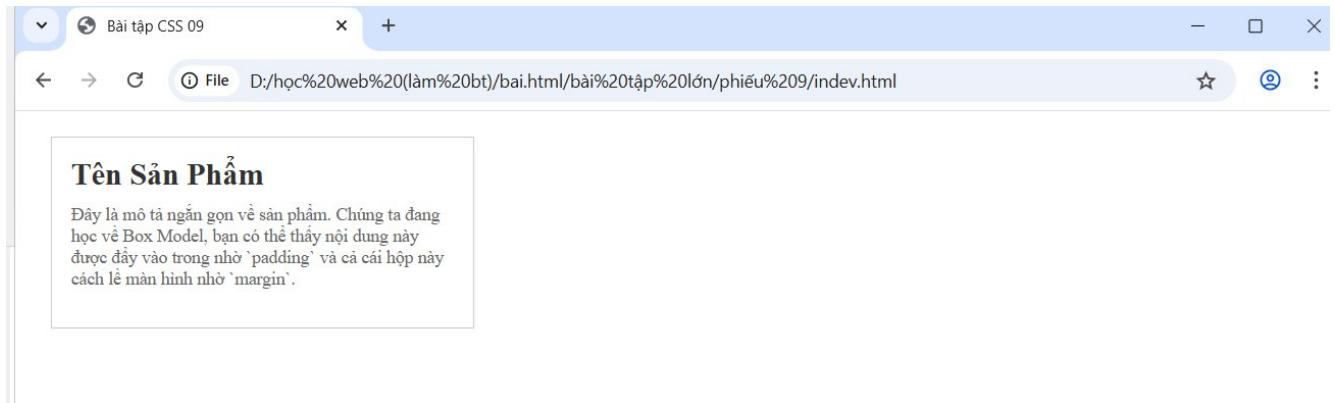
</style>
</head>
<body>

<div class="product-card">
  <h2>Tên Sản Phẩm</h2>
  <p>Đây là mô tả ngắn gọn về sản phẩm. Chúng ta đang học về Box Model, bạn có
    thể thấy nội dung này được đẩy vào trong nhờ `padding`
    và cả cái hộp này cách lề màn hình nhờ `margin`.
  </p>
</div>

</body>
</html>

```

Kết quả (Chụp ảnh màn hình và dán vào đây): (Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:



- Một cái hộp rộng 300px (không tính padding, border).
- Hộp có viền xám nhạt.

- Chữ "Tên Sản Phẩm" và "Đây là mô tả..." nằm cách viền 15px (do padding).
 - Cả cái hộp nằm cách lề trình duyệt 20px (do margin).)
-

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em vẫn chưa phân biệt được padding và margin! Khi nào em muốn đẩy 2 cái hộp ra xa nhau thì em dùng padding hay margin?", hoặc "Em đặt width: 300px và padding: 15px, vậy chiều rộng tổng cộng của cái hộp là bao nhiêu?")*

Video đã xem (Link):

1. Vấn đề của Box Model (Bài G)

Hãy nhớ lại bài S. Khi bạn đặt style:

CSS

```
.box {  
    width: 300px;  
    padding: 15px;  
    border: 1px solid black;  
}
```

Trình duyệt (ở chế độ mặc định) sẽ tính toán **chiều rộng TỔNG CỘNG** mà cái hộp chiếm trên màn hình là:

Tổng Chiều rộng = width + padding (trái + phải) + border (trái + phải)

Tổng Chiều rộng = 300px + (15px + 15px) + (1px + 1px) = 332px

- **Vấn đề:** Điều này rất phiền phức. Bạn muốn một cái hộp rộng 300px, nhưng nó lại thành 332px. Nếu bạn muốn nó *đúng* 300px, bạn phải tự làm toán trừ (width: 268px).

2. Kiến thức cốt lõi (Giải pháp box-sizing)

CSS cung cấp thuộc tính box-sizing để thay đổi cách trình duyệt tính toán kích thước.

Giá trị	Cách tính toán (Mặc định là content-box)
box-sizing: content-box;	(Mặc định) width và height CHỈ áp dụng cho Content . Tổng kích thước = width + padding + border.
box-sizing: border-box;	(Vị cứu tinh) width và height áp dụng cho tất cả (Content + Padding + Border).

Kết luận: Khi bạn dùng box-sizing: border-box; và đặt width: 300px;, cái hộp sẽ **luôn luôn rộng đúng 300px** trên màn hình, bất kể bạn thêm padding hay border bao nhiêu. Trình duyệt sẽ tự động co phần *content* bên trong lại để giữ nguyên tổng kích thước.

3. Ví dụ thực hành (Thấy tận mắt)

Yêu cầu: Tạo 2 cái hộp có CÙNG một style, nhưng KHÁC nhau về box-sizing để thấy sffi khác biệt.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 10</title>
  <style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */
    body {
      font-family: Arial, sans-serif;
      background-color: #fafafa;
      padding: 20px;
    }
    /* Style chung cho cả 2 hộp */
    .box {
      width: 300px; /* Cả 2 đều 300px */
      padding: 20px; /* Cả 2 đều 20px */
      border: 10px solid green; /* Cả 2 đều 10px */
      margin-bottom: 20px;
      background-color: #f0f0f0;
    }

    /* Yêu cầu 1: Hộp này dùng chế độ mặc định */
    .box-1 {
      box-sizing: content-box;
    }

    /* Yêu cầu 2: Hộp này dùng "vị cứu tinh" */
    .box-2 {
      box-sizing: border-box;
    }

  </style>
</head>
<body>

  <div class="box box-1">
    <h2>Hộp 1 (content-box)</h2>
    <p>Width: 300px, Padding: 20px, Border: 10px</p>
    <p>Tổng chiều rộng của tôi BỊ TO RA.</p>
  </div>

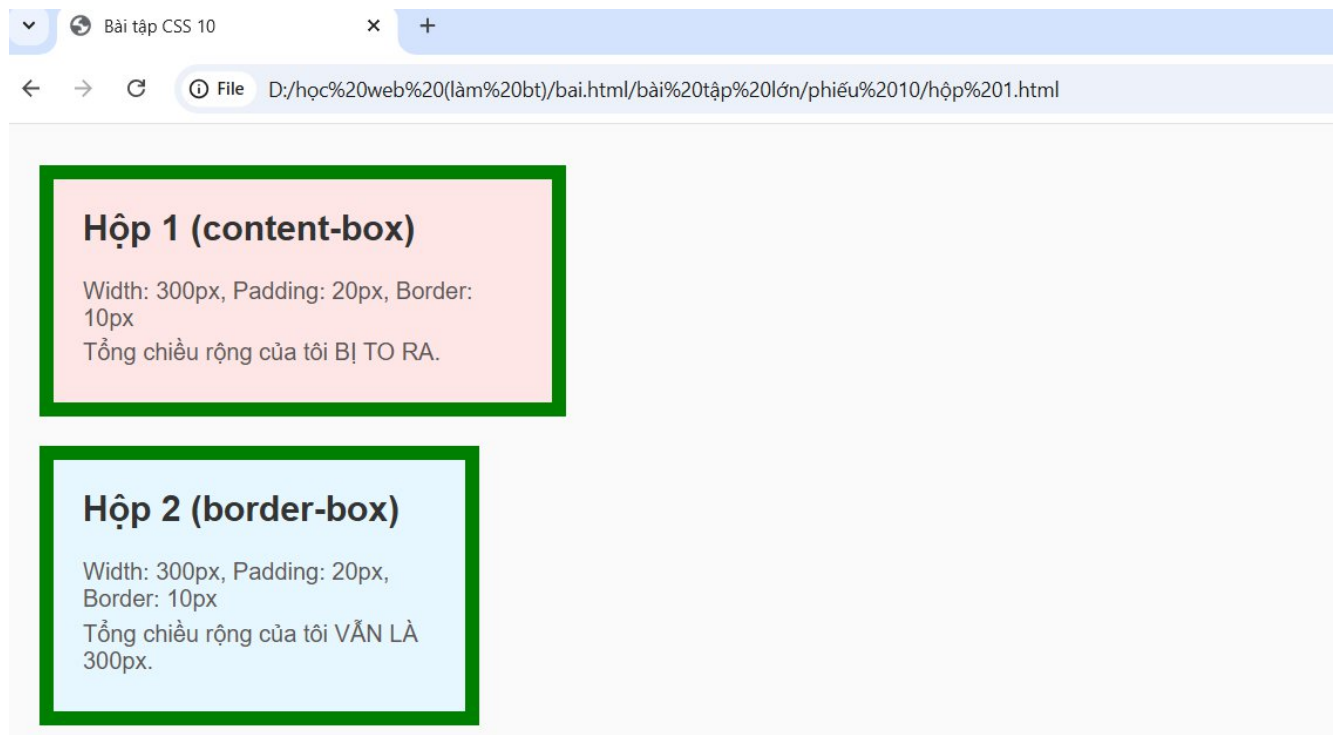
  <div class="box box-2">
    <h2>Hộp 2 (border-box)</h2>
    <p>Width: 300px, Padding: 20px, Border: 10px</p>
    <p>Tổng chiều rộng của tôi VẪN LÀ 300px.</p>
  </div>

</body>
```

</html>

Kết quả (Chụp ảnh màn hình và dán vào đây):

(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:



- *Hộp 1 (content-box) nhìn TO HƠN hẳn Hộp 2.*
- Hộp 2 (border-box) nhìn nhỏ gọn hơn, đúng 300px.

)

Bài tập tính toán (Bắt buộc):

1. Tổng chiều rộng thực tế của **Hộp 1** là bao nhiêu? (Tự tính) _____px
 2. Tổng chiều rộng thực tế của **Hộp 2** là bao nhiêu? _____px
-

4. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Tại sao border-box tốt như vậy mà trình duyệt không đặt nó làm mặc định?", hoặc "Em thấy nhiều người hay dùng `* { box-sizing: border-box; }` ở đầu file CSS. Đoạn code đó có ý nghĩa gì?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Bản chất của phần tử)

Mọi phần tử HTML đều có một giá trị display mặc định. Sau khi xem video, hãy hoàn thành bảng so sánh các giá trị display quan trọng:

Giá trị display	Đặc điểm (Hành vi)	Có thể đặt width/height?	Ví dụ thẻ mặc định
block	(Tự nhiên: Luôn bắt đầu trên một... và chiếm toàn bộ... có sẵn)	Có	div, p, h1...h6, ul, li
inline	(Tự nhiên: Nằm trên cùng một... với các phần tử khác, chỉ chiếm... cần thiết)	Không	span, a, img, strong
inline-block	(Tự nhiên: Kết hợp cả hai - Nằm trên cùng một... NHƯNG...)	Có	(Ít có thẻ mặc định, chủ yếu ta tự đặt)
none	(Tự nhiên: Ẩn hoàn toàn phần tử, coi như nó...)	(Không liên quan)	(Không có)

Câu hỏi bắt buộc:

- Đây là điểm khác biệt lớn nhất giữa inline và inline-block?

inline không thể đặt kích thước (width, height, margin-top/bottom) — nó chỉ co giãn theo nội dung.

inline-block có thể đặt kích thước như block, nhưng vẫn nằm trên cùng một dòng với các phần tử khác.

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS trong thẻ <style> để thay đổi hành vi mặc định của các phần tử HTML bên dưới.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1.0">
  <title>Bài tập CSS 11</title>
  <style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

    /* Yêu cầu 1:
      Các thẻ <span> mặc định là 'inline'.
      Hãy đổi chúng thành 'block' để chúng xếp chồng lên nhau
```

VÀ đặt cho mỗi thẻ width: 200px, background-color: #f9e79f;

*/

.bai-1 span {

display: block;
width: 200px;
background-color: #f9e79f;
margin-bottom: 5px;
padding: 5px;
border: 1px solid #d4ac0d;

```
}
```

```
/* Yêu cầu 2:
```

```
    Các thẻ <div> mặc định là 'block'.
```

```
    Hãy đổi chúng thành 'inline' để chúng xếp hàng ngang.
```

```
*/
```

```
.bai-2 div {
```

```
display: inline;
```

```
background-color: #abebc6;
```

```
border: 1px solid #27ae60;
```

```
padding: 3px;
```

```
margin-right: 5px;
```

```
}
```

```
/* Yêu cầu 3:
```

```
    Dùng 'inline-block' để tạo một menu hàng ngang.
```

```
    Chúng vừa xếp hàng ngang, vừa có thể đặt chiều
```

```
    rộng (width) và đệm (padding).
```

```
*/
```

```
.bai-3 li {
```

```
    width: 100px;
```

```
    padding: 10px;
```

```
    background-color: #d6eaf8;
```

```
    border: 1px solid #3498db;
```

```
    text-align: center;
```

```
}
```

```
/* Yêu cầu 4:
```

```
    Dùng 'display: none' để ẩn hoàn toàn phần
```

```
    tử có class "an-di".
```

```
*/
```

```
.an-di {
```

```
display: none;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div class="bai-1">
```

```
    <span>Mục 1</span>
```

```
    <span>Mục 2</span>
```

```
    <span>Mục 3</span>
```

```
</div>
```

<hr>

```
<div class="bai-2">  
  <div>Hộp 1</div>  
  <div>Hộp 2</div>  
  <div>Hộp 3</div>  
</div>
```

<hr>

```
<ul class="bai-3">  
  <li>Trang chủ</li>
```

```
<li>Sản phẩm</li>
<li>Liên hệ</li>
</ul>

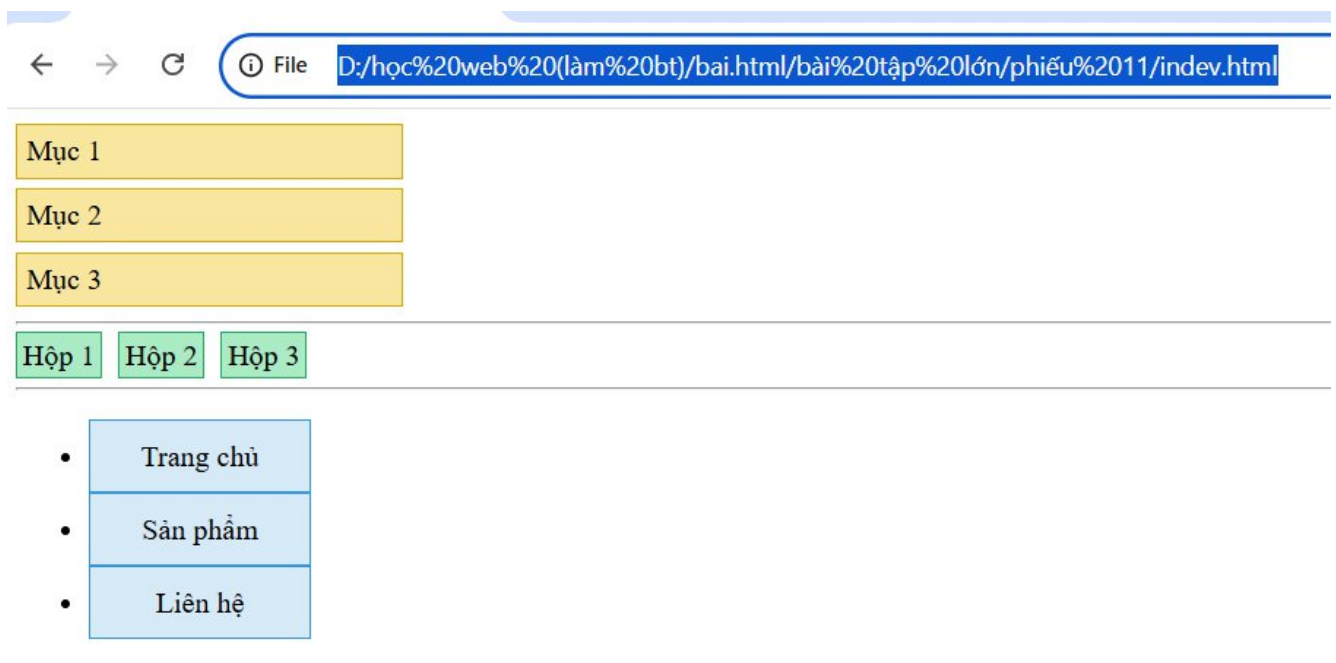
<hr>

<p>Bạn có thấy tôi không?</p>
<p class="an-di">Bạn không thể thấy tôi!</p>
<p>Bạn lại thấy tôi rồi.</p>

</body>
</html>
```

Kết quả (Chụp ảnh màn hình và dán vào đây):

(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:



Bạn có thấy tôi không?

Bạn lại thấy tôi rồi.

- Bài 1: 3 hộp vàng xếp dọc, mỗi hộp rộng 200px.
- Bài 2: 3 chữ "Hộp 1, Hộp 2, Hộp 3" nằm dính liền nhau trên 1 hàng.
- Bài 3: 3 mục menu (Trang chủ, Sản phẩm, Liên hệ) xếp hàng ngang, có nền xanh và viền.
- Bài 4: Dòng chữ "Bạn không thể thấy tôi!" hoàn toàn biến mất.

)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em nghe nói có visibility: hidden cũng dùng để ẩn. Vậy nó khác gì với display: none?")*

PHIẾU HỌC TẬP CSS [12] - ĐỊNH VỊ (POSITIONING)

Video đã xem (Link):

1. Kiến thức cốt lõi (Các hệ quy chiếu)

Sau khi xem video, hãy hoàn thành bảng so sánh 5 giá trị của thuộc tính position:

Giá trị position	Hệ quy chiếu (Nó di chuyển/định vị dựa theo cái gì?)	Bị rút khỏi dòng chảy? (Có chiếm diện tích không?)
static	(Tự nhiên: Dòng chảy bình thường của tài liệu)	Không
relative	(Tự nhiên: Vị trí ban đầu CỦA CHÍNH NÓ)	Không (Nó vẫn "giữ chỗ" ở vị trí gốc)
absolute	(Tự nhiên: "Tổ tiên" gần nhất có position KHÁC static)	Có (Bị nhấc bổng lên, không chiếm chỗ)
fixed	(Tự nhiên: Cửa sổ trình duyệt, hay còn gọi là Viewport)	Có
sticky	(Tự nhiên: Kết hợp static và fixed)	Không

Câu hỏi bắt buộc:

- Để position: absolute hoạt động đúng ý (tức là nằm gọn bên trong thẻ cha), thì thẻ cha (container) của nó **bắt buộc** phải có thuộc tính position là gì?
- Thuộc tính z-index dùng để làm gì? Nó có hoạt động với position: static không?

1. Thẻ cha phải có position: relative; (hoặc absolute, fixed, sticky) — miễn không phải static.
→ Khi đó phần tử con absolute sẽ định vị dựa vào khung của thẻ cha chứ không bay ra toàn trang.
2. z-index dùng để xác định thứ tự chồng lớp (lớp trên – lớp dưới) của các phần tử khi chúng bị chồng lên nhau.

z-index chỉ hoạt động với phần tử có position khác static, tức là relative, absolute, fixed, hoặc sticky.
Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS trong thẻ <style> để quan sát hành vi của 3 loại position chính: relative, absolute, và fixed.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 12</title>
  <style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

    /* Thêm style cho body để có thể cuộn trang */
```

```
body {  
  height: 1500px; /* Tạo chiều cao ảo để cuộn */
```

```

    font-family: Arial, sans-serif;
}

/* Yêu cầu 1: Thẻ cha (container)
   Đặt làm "điểm tựa" cho thẻ con 'absolute'
*/
.container {
    width: 500px;
    height: 300px;
    border: 3px dashed blue; margin:
    50px;

    /* TODO: Đặt position là 'relative' để làm
       "neo" cho .box-absolute
    */

}

/* Yêu cầu 2: Hộp 'relative'
   Dịch chuyển nó 20px so với VỊ TRÍ GỐC
*/
.box-relative { position:
    relative; top: 20px;
    left: 20px;

    width: 100px;
    height: 100px;
    background-color: #f9e79f; /* Vàng */
}

/* Yêu cầu 3: Hộp 'absolute'
   Định vị nó theo GÓC TRÊN BÊN PHẢI của
   thẻ cha (.container)
*/
.box-absolute {
    /* TODO: Đặt position là 'absolute' */

    top: 0;
    right: 0;

    width: 100px;
    height: 100px;
    background-color: #d6eaf8; /* Xanh */
}

/* Yêu cầu 4: Hộp 'fixed'
   "Đóng đinh" nó vào GÓC DƯỚI BÊN PHẢI
   của MÀN HÌNH TRÌNH DUYỆT

```

```
*/  
.box-fixed {  
    /* TODO: Đặt position là 'fixed' */
```

```
        bottom:20px; right:
        20px;

        width: 100px;
        height:100px;
        background-color: #d5f5e3; /* Xanh lá */
    }

</style>
</head>
<body>

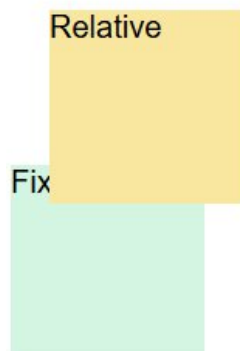
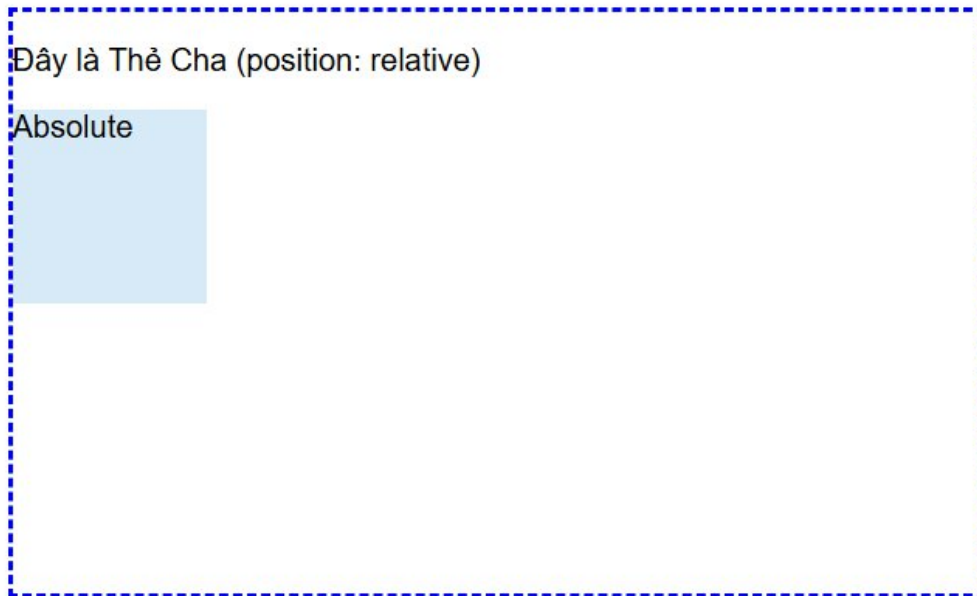
    <div class="container">
        <p>Đây là Thẻ Cha (position: relative)</p>
        <div class="box-absolute">Absolute</div>
    </div>

    <div class="box-relative">
        Relative
    </div>

    <div class="box-fixed"> Fixed
    </div>

</body>
</html>
```

Kết quả (Chụp ảnh màn hình và dán vào đây):



(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:

- Hộp màu xanh (.box-absolute) nằm ở góc trên-phải CỦA cái viền xanh dashed.
- Hộp màu vàng (.box-relative) bị dịch xuống dưới 20px và sang phải 20px so với vị trí bình thường của nó.
- Hộp màu xanh lá (.box-fixed) nằm ở góc dưới-phải CỦA CỬA SỔ TRÌNH DUYỆT. Khi bạn cuộn trang, nó VẪN ĐỨNG YÊN TẠI CHỖ.

)

2. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Nếu em đặt position: absolute cho .box-absolute nhưng em quên đặt position: relative cho .container, thì cái hộp màu xanh sẽ bay đi đâu? Tại sao?")*

PHIẾU HỌC TẬP CSS [13.1] - FLEXBOX: THUỘC TÍNH CONTAINER (CHA)

Video đã xem (Link):

1. Kiến thức cốt lõi (Kiểm soát từ thẻ Cha)

Flexbox là một mô hình layout một chiều. Để "kích hoạt" Flexbox, bạn chỉ cần đặt `display: flex;` cho thẻ cha (container). Mọi thẻ con (item) bên trong sẽ tffđ động "ngoan ngoãn" xếp hàng.

Sau khi xem video, hãy điền vào mục đích của các thuộc tính của thẻ **CHA**:

Thuộc tính (của Cha)	Mục đích sử dụng (Dùng để làm gì?)	Các giá trị phổ biến
display: flex;	(Tự điền: "Kích hoạt" môi trường flexbox cho các thẻ con)	flex
flex-direction	(Tự điền: Quyết định hướng xếp hàng của các con: ngang hay dọc?)	row (ngang), column (dọc)
justify-content	(Tự điền: Căn chỉnh các con theo... (trục chính))	flex-start, flex-end, center, space-between, space-around
align-items	(Tự điền: Căn chỉnh các con theo... (trục phụ))	flex-start, flex-end, center, stretch
flex-wrap	(Tự điền: Quyết định các con có được... khi không đủ chỗ hay không)	nowrap (không xuống), wrap (tự động xuống)

Câu hỏi bắt buộc:

- Khi flex-direction: row; (mặc định), thì **Trục chính** (Main Axis) là trục nào? (Ngang hay Dọc?)
- Trục Ngang
- Khi flex-direction: column;, thì **Trục chính** là trục nào? (Ngang hay Dọc?)
- Trục Dọc

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS cho thẻ cha `.flex-container` để sắp xếp các thẻ con `.item` theo đúng 3 yêu cầu bên dưới.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

<title>Bài tập CSS 13.1</title>

<style>

```

/* --- KHÔNG SỬA PHẦN NÀY --- */
body { font-family: Arial, sans-serif; }
.flex-container {
    border: 3px dashed #333;
    padding: 10px;
    margin-bottom: 20px; height:
    250px;
}
.item {
    width: 80px; height:
    80px;
    background-color: #3498db;
    color: white;
    font-size: 30px;
    /* Dùng display: flex lồng nhau để căn giữa số */
    display: flex;
    justify-content: center; align-
    items: center;
}
/* --- KẾT THÚC PHẦN KHÔNG SỬA --- */

```

/* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

/* Yêu cầu 1: Container 1

- Kích hoạt flex
- Xếp hàng ngang (mặc định)
- Căn giữa theo chiều ngang (trục chính)
- Căn giữa theo chiều dọc (trục phụ)

*/

```

.container-1 { display:
    flex;
    justify-content: center; align-
    items: center;
}

```

/* Yêu cầu 2: Container 2

- Kích hoạt flex
- Xếp hàng ngang
- Căn đều 2 bên, chừa khoảng trống Ở GIỮA (space-between)

*/

```

.container-2 { display:
    flex;
    justify-content: space-between;
}

```

/* Yêu cầu 3: Container 3

- Kích hoạt flex
- Cho phép XUỐNG HÀNG khi không đủ chỗ

```
*/  
.container-3 { display:  
    flex; flex-wrap:  
    wrap;  
}
```

```
</style>
</head>
<body>

  <h2>Yêu cầu 1: Căn giữa tuyệt đối</h2>
  <div class="flex-container container-1">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
  </div>

  <h2>Yêu cầu 2: Căn đều 2 bên</h2>
  <div class="flex-container container-2">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
  </div>

  <h2>Yêu cầu 3: Tự động xuống hàng</h2>
  <div class="flex-container container-3">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
    <div class="item">7</div>
  </div>

</body>
</html>
```

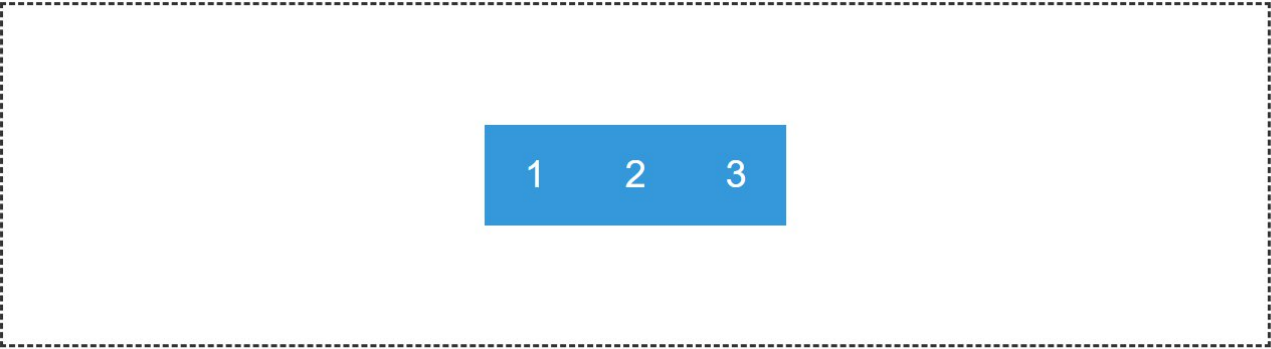
Kết quả (Chụp ảnh màn hình và dán vào đây):

(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:

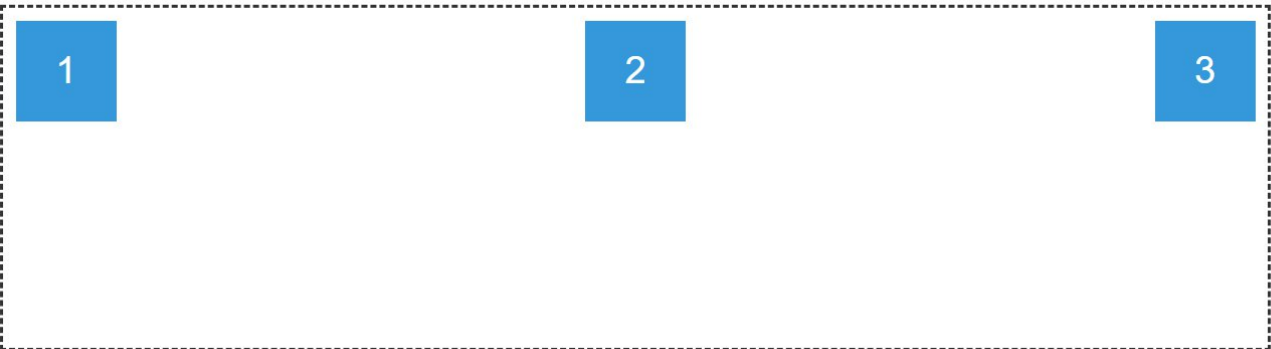
Bài tập CSS 13.1

File D:/học%20web%20(làm%20bt)/bai.html/bài%20tập%20lớn/phiếu%2013.1/index.html

Yêu cầu 1: Căn giữa tuyệt đối



Yêu cầu 2: Căn đều 2 bên



- *Hộp 1: 3 khối xanh nằm CĂN GIỮA (cả ngang và dọc) của viền dashed.*
- *Hộp 2: Khối 1 nằm sát lề trái, Khối 3 sát lề phải, Khối 2 nằm chính giữa.*
- *Hộp 3: Các khối 1-7 tự động xếp hàng, và những khối không đủ chỗ (từ 6, 7) SẼ TỰ RỐT XUỐNG HÀNG dưới.*

)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em vẫn bị lẫn lộn giữa justify-content và align-items. Làm sao để nhớ cái nào căn ngang, cái nào căn dọc? Nó có luôn luôn như vậy không?")*

PHIẾU HỌC TẬP CSS [13.2] - FLEXBOX: THUỘC TÍNH ITEMS (CON)

Video đã xem (Link):

1. Kiến thức cốt lõi (Kiểm soát từ thẻ Con)

Nếu như Bài 13.1 là thẻ Cha "ra lệnh" cho tất cả thẻ Con, thì bài này cho phép từng thẻ Con "cá nhân hóa" hành vi của mình.

Sau khi xem video, hãy điền vào mục đích của các thuộc tính của thẻ **CON** (item):

Thuộc tính (của Con)	Mục đích sử dụng (Dùng để làm gì?)
flex-grow	(Tự điền: Quyết định khả năng "phình to" của item để...
flex-shrink	(Tự điền: Quyết định khả năng "co lại" của item khi...)
flex-basis	(Tự điền: Quyết định kích thước... của item)
flex	(Tự điền: Là cú pháp viết tắt cho 3 thuộc tính nào?...)
align-self	(Tự điền: Cho phép 1 item "không vâng lời" và tự căn chỉnh...)
order	(Tự điền: Dùng để thay đổi... của các item)

Xuất sang Trang tính

Câu hỏi bắt buộc:

- Một item có flex-grow: 2; và item khác có flex-grow: 1;. Điều này có nghĩa là gì khi có không gian thừa?
 - Bạn muốn item thứ 1 (HTML: <div class="item1">) hiển thị ở vị trí thứ 3. Bạn sẽ đặt order cho nó là bao nhiêu?
-

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS cho các thẻ con (.item) để điều khiển chúng trong một flex-container.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 13.2</title>
  <style>
    /* --- KHÔNG SỬA PHẦN NÀY --- */
    body { font-family: Arial, sans-serif; }
    .flex-container { display:
      flex;
      border: 3px dashed #333;
```

```
padding: 10px; margin-
bottom: 20px; height:
150px;
background-color: #f0f0f0;
}
.item {
/* flex-basis: 100px; */ /* Kích thước cơ bản */ width: 100px;
/* Tạm dùng width để cố định */ height: 80px;
background-color: #e74c3c; color:
white;
font-size: 24px;
border: 2px solid #c0392b;
display: flex;
justify-content: center; align-
items: center;
}
/* --- KẾT THÚC PHẦN KHÔNG SỬA --- */
```

```
/* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */
```

```
/* Yêu cầu 1: Container 1
   Làm cho item ".grow" PHÌNH TO ra
   để chiếm hết không gian thừa.
*/
.container-1 .grow {
background-color: #2ecc71; /* Màu xanh lá */
}
```

```
/* Yêu cầu 2: Container 2 Thay
   đổi THỨ TỰ hiển thị.
   Làm cho item ".order-1" nhảy LÊN ĐẦU
   Làm cho item ".order-3" nhảy XUỐNG CUỐI
*/
.container-2 .order-1 {
```

```
display: flex;
justify-content: center;
align-items: center;
```

```
    }
    .container-2 .order-3 {
```

```
display: flex;
justify-content: space-between;
```

```
    }
```

/* Yêu cầu 3: Container 3

Thẻ cha đang căn giữa (align-items: center). Hãy dùng 'align-self' để BẮT item ".self" phải nhảy vọt LÊN TRÊN CÙNG (flex-start).

*/

.container-3 {

align-items: center;

}

.container-3 .self {

background-color: #9b59b6; /* Màu tím */

```
}

</style>
</head>
<body>

  <h2>Yêu cầu 1: Dùng `flex-grow` </h2>
  <div class="flex-container container-1">
    <div class="item">1</div>
    <div class="item grow">2 (Tôi sẽ lớn lên)</div>
    <div class="item">3</div>
  </div>

  <h2>Yêu cầu 2: Dùng `order` </h2>
  <div class="flex-container container-2">
    <div class="item order-3">(HTML là 1)</div>
    <div class="item">(HTML là 2)</div>
    <div class="item order-1">(HTML là 3)</div>
  </div>

  <h2>Yêu cầu 3: Dùng `align-self` </h2>
  <div class="flex-container container-3">
    <div class="item">Item A</div>
    <div class="item self">(Tôi khác biệt)</div>
    <div class="item">Item C</div>
  </div>


</body>
</html>
```

Kết quả (Chụp ảnh màn hình và dán vào đây): (Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:


Bài tập CSS 13.2

D:/học%20web%20(làm%20bt)/bai.html/bài%20tập%20lớn/phiếu%2013/index.html

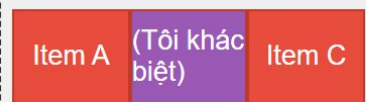
Yêu cầu 1: Dùng `flex-grow`



Yêu cầu 2: Dùng `order`



Yêu cầu 3: Dùng `align-self`



- *Hộp 1: Item 2 (màu xanh lá) tffđ động phình to ra, chiếm hết khoảng trống giữa Item 1 và 3.*
- *Hộp 2: Thứ tffđ thị giác bị đảo lộn. Chữ "(HTML là 3)" chạy lên đầu tiên, và chữ "(HTML là 1)" bị đẩy xuống cuối cùng.*
- *Hộp 3: Item A và C được căn giữa theo chiều dọc, nhưng Item màu tím "(Tôi khác biệt)" thì bị đẩy sát lên lề trên của hộp.)*

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em dùng order để thay đổi thứ tự. Vậy người dùng khiếm thị dùng trình đọc màn hình (screen reader) sẽ đọc theo thứ tự HTML hay thứ tự order của CSS?", hoặc "Sự khác biệt giữa width: 100px và flex-basis: 100px là gì?")*

PHIẾU HỌC TẬP CSS [14.1] - GRID: DỰNG KHUNG (CONTAINER)

Video đã xem (Link):

1. Kiến thức cốt lõi (Layout Hai chiều)

Nếu **Flexbox** là công cụ layout 1 chiều (hoặc ngang, hoặc dọc), thì **Grid** là công cụ layout 2 chiều (cả ngang và dọc CÙNG LÚC). Nó cho phép bạn tạo ra các bố cục cột và hàng phức tạp một cách dễ dàng.

Sau khi xem video, hãy điền vào mục đích của các thuộc tính của thẻ **CHA** (Grid Container):

Thuộc tính (của Cha)	Mục đích sử dụng (Dùng để làm gì?)	Ví dụ giá trị phổ biến
display: grid;	(Tự điền: "Kích hoạt" môi trường layout Grid)	grid
grid-template-columns	(Tự điền: Định nghĩa số lượng VÀ độ rộng của các CỘT)	100px 100px, 1fr 2fr, repeat(3, 1fr)
grid-template-rows	(Tự điền: Định nghĩa số lượng VÀ độ cao của các DÒNG)	200px 400px, auto
gap (hoặc grid-gap)	(Tự điền: Tạo khoảng trống...)	10px, 10px 20px

Xuất sang Trang tính

Câu hỏi bắt buộc:

1. Đơn vị fr (viết tắt của "Fraction" - Phân số) có nghĩa là gì?

fr là viết tắt của fractional unit — đơn vị “phân số”, dùng để chia phần còn lại của không gian khả dụng trong Grid container theo tỷ lệ.

2. Nếu bạn viết grid-template-columns: 1fr 2fr;, bố cục sẽ được chia như thế nào? (Gợi ý: Tổng là 3 phần, cột 1 chiếm... phần, cột 2 chiếm... phần).

Tổng cộng là 3 phần.

Cột 1 chiếm 1 phần,

Cột 2 chiếm 2 phần.

3. Hàm repeat(3, 1fr) tương đương với việc viết 1fr 1fr 1fr đúng không?

Đúng. Vì Cả hai cách đều tạo 3 cột bằng nhau, mỗi cột chiếm 1 phần (1fr) của chiều rộng tổng thể.

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS cho thẻ cha .grid-container để sắp xếp c thẻ con .item theo đúng 3 yêu cầu bố cục bên dưới.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 14.1</title>
  <style>
    /* --- KHÔNG SỬA PHẦN NÀY --- */
```

```
body { font-family: Arial, sans-serif; }
.grid-container {
  border: 3px dashed #c0392b;
  padding: 10px;
  margin-bottom: 20px;
  background-color: #f9f9f9;
}
.item {
  background-color: #f39c12; color:
  white;
  padding: 20px; font-
  size: 20px; text-align:
  center;
  border: 1px solid #e67e22;
}
/* --- KẾT THÚC PHẦN KHÔNG SỬA --- */
```

```
/* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */
```

```
/* Yêu cầu 1: Container 1
```

- Kích hoạt grid
- Tạo 3 CỘT, mỗi cột RỘNG 100px
- (Hàng sẽ tự động được tạo)

```
*/
```

```
.container-1 { display:
  grid;
```

```
grid-template-columns: 100px 100px 100px;
}
```

```
/* Yêu cầu 2: Container 2
```

- Kích hoạt grid
- Tạo 3 CỘT, chia ĐỀU toàn bộ không gian (dùng 'fr')

```
*/
```

```
.container-2 { display:
  grid;
```

```
grid-template-columns: 1fr 1fr 1fr;
}
```

```
/* Yêu cầu 3: Container 3
```

- Kích hoạt grid
- Tạo 2 CỘT (Cột 1 rộng 200px, Cột 2 chiếm hết phần còn lại)
- Tạo khoảng cách (gap) 10px giữa các ô

```
*/
```

```
.container-3 { display:
  grid;
```

```
grid-template-columns: 200px 1fr;
gap: 10px;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h2>Yêu cầu 1: 3 cột, mỗi cột 100px</h2>
```

```
<div class="grid-container container-1">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</div>
```

<h2>Yêu cầu 2: 3 cột, chia đều (dùng `fr`)</h2>

```
<div class="grid-container container-2">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</div>
```

<h2>Yêu cầu 3: 2 cột (200px và 1fr) + `gap` 10px</h2>

```
<div class="grid-container container-3">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</div>
```

```
</body>
```

```
</html>
```

Kết quả (Chụp ảnh màn hình và dán vào đây): (Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:

Bài tập CSS 14.1

File D:/hoc%20web%20(lam%20bt)/bai.html/bai%20tap%20lon/phiếu%2014/index.html

Yêu cầu 1: 3 cột, mỗi cột 100px

1	2	3
4	5	6

Yêu cầu 2: 3 cột, chia đều (dùng `fr`)

1	2	3
4	5	6

Yêu cầu 3: 2 cột (200px và 1fr) + `gap` 10px

1	2
3	4
5	6

- *Hộp 1: c khối cam xếp thành lưới 3 cột 2 hàng. Chiều rộng của 3 cột CỐ ĐỊNH là 100px.*
- *Hộp 2: c khối cam xếp thành lưới 3 cột 2 hàng. 3 cột này tffi động PHÌNH RA để lấp đầy chiều rộng của container.*
- *Hộp 3: c khối cam xếp thành lưới 2 cột 3 hàng. Cột 1 cố định 200px, cột 2 phình ra lấp đầy phần còn lại. Quan trọng: Có một KHOẢNG TRỐNG (gap) 10px giữa các khối cam.)*

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em thấy display: grid và display: flex rất giống nhau ở việc sắp xếp. Vậy khi nào thì em nên dùng Flexbox, khi nào thì nên dùng Grid?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Vượt ô - Spanning)

Phần mạnh nhất của Grid là cho phép 1 item (thẻ con) chiếm (span) nhiều hơn một ô (cell) trong lưới. Chúng ta làm điều này bằng cách chỉ định nó nên **bắt đầu từ đường kẻ (line) nào** và **kết thúc ở đường kẻ nào**.

Ghi nhớ quan trọng:

- Một lưới có **3 CỘT** sẽ có **4 ĐƯỜNG KẼ CỘT** (1, 2, 3, 4).
- Một lưới có **2 DÒNG** sẽ có **3 ĐƯỜNG KẼ DÒNG** (1, 2, 3).

Sau khi xem video, hãy điền vào mục đích của các thuộc tính của thẻ **CON** (item):

Thuộc tính (của Con)	Mục đích sử dụng (Dùng để làm gì?)	Ví dụ cú pháp
grid-column-start	(Tự điền: Chỉ định đường kẻ cột...)	1
grid-column-end	(Tự điền: Chỉ định đường kẻ cột...)	3
grid-column	(Tự điền: Cú pháp TẮT cho 2 thuộc tính trên)	1 / 3 (Bắt đầu ở 1, kết thúc ở 3)
grid-row-start	(Tự điền: Chỉ định đường kẻ dòng...)	1
grid-row-end	(Tự điền: Chỉ định đường kẻ dòng...)	2
grid-row	(Tự điền: Cú pháp TẮT cho 2 thuộc tính trên)	1 / 2

Các cú pháp đặc biệt:

- grid-column: 1 / 3; có nghĩa là "chiếm 2 cột đầu tiên" (từ vạch 1 đến vạch 3).
 - grid-column: span 2; có nghĩa là "chiếm 2 cột, bắt đầu từ vị trí mặc định của nó".
 - grid-column: 1 / -1; là một mẹo hay, có nghĩa là "bắt đầu từ vạch 1 và kết thúc ở vạch CUỐI CÙNG" (rất hữu ích khi bạn không biết có bao nhiêu cột).
-

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Dffing một bố cục (layout) trang web kinh điển (Header, Sidebar, Main Content, Footer) bằng cách sử dụng Grid. Thẻ cha đã được thiết lập sẵn, bạn chỉ cần ra lệnh cho các thẻ con phải nằm ở đâu.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 14.2</title>
  <style>
    /* --- KHÔNG SỬA PHẦN NÀY --- */
    body { font-family: Arial, sans-serif; margin: 0; }

    .layout {
      display: grid; width:
      100vw; height:
      100vh;

      /* Định nghĩa 2 CỘT:
        Cột 1 (sidebar) rộng 200px
        Cột 2 (main) chiếm hết phần còn lại (1fr)
      */
      grid-template-columns: 200px 1fr;

      /* Định nghĩa 3 DÒNG:
        Dòng 1 (header) cao 60px
        Dòng 2 (content) chiếm hết phần còn lại (1fr)
        Dòng 3 (footer) cao 40px
      */
      grid-template-rows: 60px 1fr 40px;
    }

    /* Style chung cho các ô */
    .layout > div { padding:
      10px; color: white;
      font-size: 20px;
    }
    /* --- KẾT THÚC PHẦN KHÔNG SỬA --- */

    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

    /* Yêu cầu 1: Header
      - Nằm ở DÒNG 1
      - Phải CHIẾM CẢ 2 CỘT (từ vạch 1 đến vạch 3, hoặc 1 / -1)
    */
    .header {
      background-color: #3498db; /* Xanh dương */
      grid-column: 1 / -1;
      grid-row: 1;
    }
```

```
/* Yêu cầu 2: Sidebar
  - Nằm ở DÒNG 2
  - Nằm ở CỘT 1
*/
.sidebar {
  background-color: #2ecc71; /* Xanh lá */
```

```

    grid-column: 1;
    grid-row: 2;
}

/* Yêu cầu 3: Main Content
   - Nằm ở DÒNG 2
   - Nằm ở CỘT 2
*/
.main-content {
    background-color: #f1c40f; /* Vàng */
grid-column: 2;
grid-row: 2;
}

/* Yêu cầu 4: Footer
   - Nằm ở DÒNG 3
   - Phải CHIẾM CẢ 2 CỘT (giống như Header)
*/
.footer {
    background-color: #34495e; /* Xám đậm */
grid-column: 1 / -1;
grid-row: 3;
}

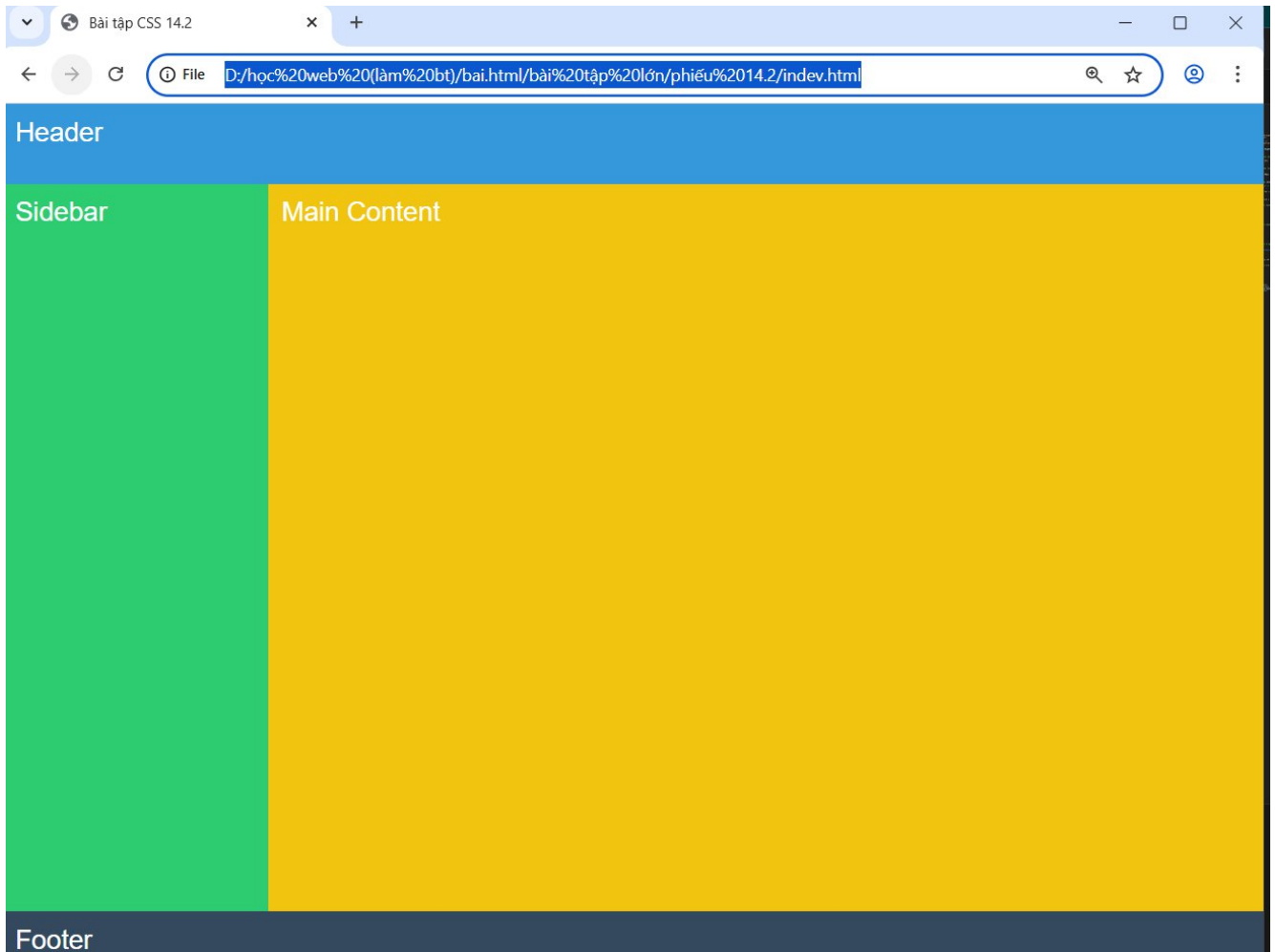
</style>
</head>
<body>

    <div class="layout">
        <div class="header">Header</div>
        <div class="sidebar">Sidebar</div>
        <div class="main-content">Main Content</div>
        <div class="footer">Footer</div>
    </div>

</body>
</html>

```

Kết quả (Chụp ảnh màn hình và dán vào đây):



(Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là một bố cục trang web hoàn chỉnh:

- *Header (xanh dương) chiếm toàn bộ chiều ngang ở trên cùng.*
- *Sidebar (xanh lá) là một cột hẹp 200px ở bên trái.*
- *Main Content (vàng) chiếm phần không gian chính còn lại, bên phải Sidebar.*
- *Footer (xám đậm) chiếm toàn bộ chiều ngang ở dưới cùng.*

)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em có thấy một cách viết khác là grid-template-areas. Cách đó so với cách dùng grid-column/row thì ưu nhược điểm là gì và khi nào nên dùng?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Trang web "co giãn")

Ngày nay, người dùng truy cập web bằng điện thoại, máy tính bảng, laptop, TV... Responsive Web Design (RWD) là kỹ thuật giúp trang web của bạn hiển thị đẹp và dễ dùng trên TẤT CẢ các thiết bị đó.

Sau khi xem video, hãy tffi diễn giải các khái niệm sau:

1. Responsive Web Design là gì?

- (Diễn giải bằng lời của bạn): cách thiết kế trang web sao cho nó tự động thay đổi bố cục và kích thước hiển thị phù hợp với mọi loại thiết bị — từ điện thoại, máy tính bảng đến máy tính để bàn.____

2. Viewport là gì?

- (Diễn giải bằng lời của bạn): vùng hiển thị nội dung của trang web trên màn hình thiết bị_____

3. Thẻ <meta name="viewport">:

- Đây là "dòng code ma thuật" **BẮT BUỘC** phải có trong thẻ <head> của mọi trang web responsive.
- Giải thích ý nghĩa của 2 thuộc tính trong thẻ: <meta name="viewport" content="width=device-width, initial-scale=1.0">
 - width=device-width: Yêu cầu trình duyệt đặt chiều rộng của trang web bằng_____
 - initial-scale=1.0: Yêu cầu trình duyệt hiển thị trang ở mức thu phóng 100% (tỷ lệ thật)

4. Triết lý "Mobile First" là g

- (Diễn giải bằng lời của bạn): thiết kế giao diện web bắt đầu từ phiên bản cho điện thoại trước,____

2. Ví dụ thực hành (Tác dụng "thần kỳ" của thẻ Viewport)

Yêu cầu: Quan sát sffi khác biệt của 1 trang web CÓ và KHÔNG CÓ thẻ <meta name="viewport"> trên màn hình điện thoại.

Cách thực hiện:

- Tạo file index.html với nội dung bên dưới (chưa có thẻ meta viewport).
- Mở file bằng trình duyệt (Chrome, Firefox...).
- Nhấn F12 (Developer Tools) và bật chế độ "Toggle device toolbar" (Biểu tượng điện thoại/tablet).
- Chọn một thiết bị (ví dụ: "iPhone 12 Pro").
- Chụp ảnh màn hình 1 (Không có meta tag).**
- Bây giờ, thêm thẻ <meta name="viewport"...> vào bên trong thẻ <head>.

7. Tải lại trang (vẫn ở chế độ điện thoại).

8. Chụp ảnh màn hình 2 (Có meta tag). Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Bài tập CSS 15</title>

  <style>
    body {
      font-size: 18px; line-
      height: 1.5;
      font-family: Arial, sans-serif;
    }
    .container {
      /* Cố tình đặt width RỘNG để xem trên desktop */
      width: 960px;
      margin: 0 auto; /* Căn giữa */
      border: 3px solid red; background-
      color: #f0f0f0;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Tiêu đề trang web</h1>
    <p>Đây là một đoạn văn bản. Nếu không có meta viewport, bạn sẽ
      thấy đoạn văn này và cả trang web
      bị thu nhỏ lại bé tí trên màn hình điện thoại.
      Người dùng sẽ phải "chạm tay" (pinch) để zoom lên.</p>
    <p>Sau khi thêm thẻ meta, chữ sẽ lập tức hiển
      thị với kích thước dễ đọc.</p>
  </div>
</body>
</html>
```

KẾT QUẢ QUAN SÁT:

Ảnh 1: KHÔNG CÓ <meta name="viewport"> (Dán ảnh chụp màn hình giả lập điện thoại của bạn vào đây)

- **Nhận xét:** (Gợi ý: Trang web bị nhỏ, bị thu lại. Chữ bé__không thể đọc được.)

Ảnh 2: ĐÃ CÓ <meta name="viewport"> (Dán ảnh chụp màn hình giả lập điện thoại của bạn vào đây)

- **Nhận xét:** (Gợi ý: Chữ đã rõ ràng và dễ đọc hơn. Tuy nhiên, cái hộp 960px màu đỏ đã bị tràn ra ra ngoài màn hình.)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Sau khi thêm thẻ meta, chữ đọc được rồi, nhưng cái hộp 960px của em bị tràn ra ngoài màn hình. Làm thế nào để cái hộp đó cũng tự động co lại cho vừa màn hình điện thoại?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Nếu... thì...)

Media Query là cú pháp "NẾU... THÌ..." của CSS. Nó cho phép bạn áp dụng các quy tắc CSS chỉ khi một điều kiện nhất định (ví dụ: màn hình có chiều rộng X) được đáp ứng.

Đây chính là chìa khóa để giải quyết vấn đề ở Bài 15.

Cú pháp cơ bản:

CSS

```
/* CSS Mặc định (cho Mobile First) */
.box {
    background-color: blue;
}

/* NẾU (màn hình có chiều rộng TỐI THIỂU là 768px) THÌ (áp dụng style bên trong)
*/
@media (min-width: 768px) {
    .box {
        background-color: red;
    }
}
```

Sau khi xem video, hãy giải thích các khái niệm sau:

1. @media: Dùng để làm gì?

Dùng để giúp website tự điều chỉnh giao diện để phù hợp với điện thoại, máy tính bảng, laptop, hay desktop.

- **2. (min-width: ...) (Triết lý Mobile First):**
 - (min-width: 768px) có nghĩa là: "Áp dụng style này cho các màn hình có chiều rộng lớn hơn 768px." (Lớn hơn hay nhỏ hơn?)
 - **3. (max-width: ...) (Triết lý Desktop First):**
 - (max-width: 600px) có nghĩa là: "Áp dụng style này cho các màn hình có chiều rộng nhỏ hơn 600px." (Lớn hơn hay nhỏ hơn?)
 - **4. Breakpoints (Điểm ngắt):**
 - Các giá trị như 768px (tablet), 992px (laptop) được gọi là gì? Breakpoints__
-

2. Ví dụ thực hành (Giải cứu layout)

Yêu cầu: Áp dụng triết lý "Mobile First". Chúng ta sẽ style cho điện thoại trước, sau đó dùng

min-width để "nâng cấp" layout cho màn hình lớn hơn.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 16</title>
  <style>
    /* --- Style Chung --- */
    body { font-family: Arial, sans-serif; margin: 0; }
    .container {
      width: 100%; /* Luôn vừa màn hình */
      max-width: 1200px; /* Giới hạn độ rộng tối đa */
      margin: 0 auto; background-
      color: #f0f0f0; padding: 10px;
    }

    .layout {
      display: flex;
      /* TODO Yêu cầu 1: (Mobile First)
      Mặc định, các item xếp DỌC (dùng flex-direction) và tự
      động xuống hàng (dùng flex-wrap).
      */

      flex-direction: column;
      flex-wrap: wrap;

    }

    .box {
      background-color: #3498db;
      color: white;
      padding: 20px;
      margin: 5px;

      /* TODO Yêu cầu 1 (Tiếp): (Mobile First) Mặc định,
      mỗi box chiếm 100% chiều rộng.
      (Gợi ý: dùng width: 100% hoặc flex-basis: 100%)
      */
      flex-basis: 100%;
    }

    /* TODO Yêu cầu 2: (Tablet / Desktop breakpoint)
    Viết một Media Query, NẾU màn hình RỘNG TỪ 768px trở lên
    thì...
    */
    @media (min-width: 768px) {
```

```
/* ...thì đổi .layout về xếp HÀNG NGANG (row)
*/
```

```
.layout {  
flex-direction: row;  
  
}
```

```
/* ...và đổi .box để mỗi box CHỈ chiếm 1/3
```

```

        (Gợi ý: dùng flex: 1; hoặc flex-basis: 32%;)
        */
        .box {
flex: 1;
        }
    }

</style>
</head>
<body>

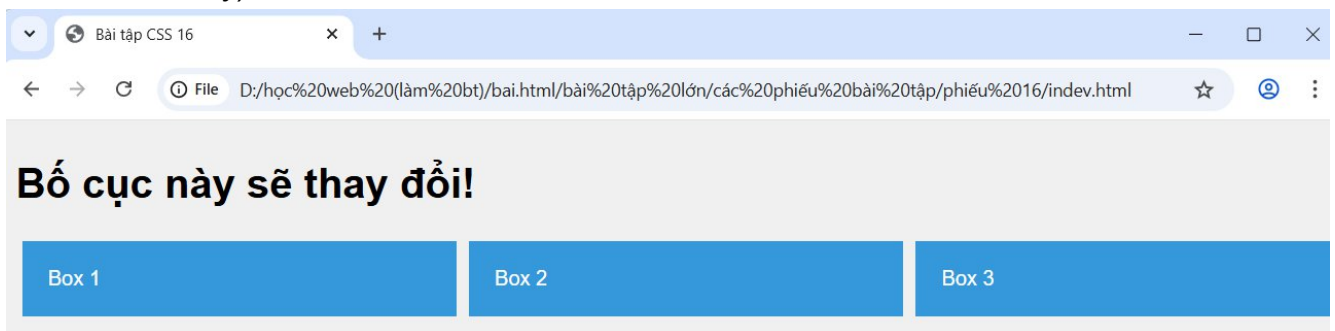
    <div class="container">
        <h1>Bố cục này sẽ thay đổi!</h1>
        <div class="layout">
            <div class="box">Box 1</div>
            <div class="box">Box 2</div>
            <div class="box">Box 3</div>
        </div>
    </div>

</body>
</html>

```

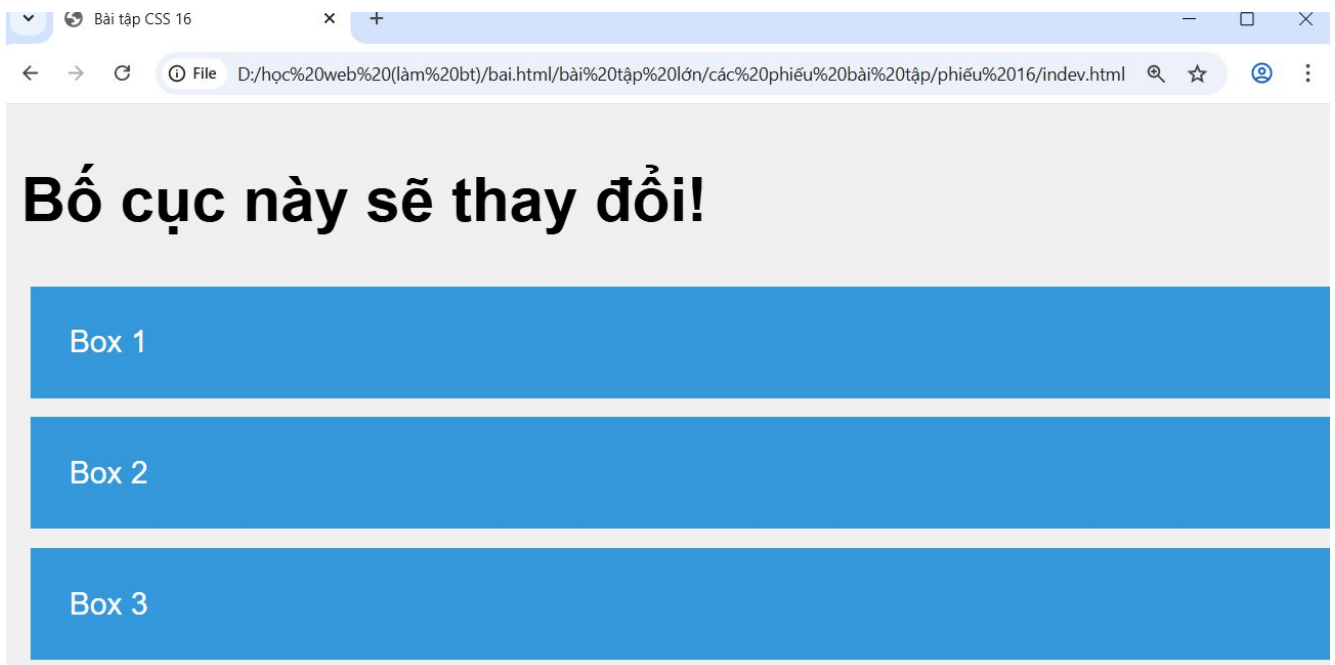
KẾT QUẢ QUAN SÁT:

Ảnh 1: Màn hình điện thoại (Dưới 768px) (Dán ảnh chụp màn hình giả lập điện thoại của bạn vào đây)



- **Nhận xét:** (Gợi ý: 3 cái hộp màu xanh xếp dọc _____ (ngang/dọc), mỗi hộp chiếm 100% _____ chiều rộng.)

Ảnh 2: Màn hình Desktop (Trên 768px) (Dán ảnh chụp màn hình trình duyệt ở chế độ desktop của bạn vào đây)



- **Nhận xét:** (Gợi ý: 3 cái hộp màu xanh xếp Ngang_____ (ngang/dọc), lấp đầy container.)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em thấy có 2 cách (min-width) và (max-width). Vậy khi nào thì nên dùng Mobile First (min-width) và khi nào nên dùng Desktop First (max-width)? Cách nào tốt hơn?")*

PHIẾU HỌC TẬP CSS [17] - HÌNH ẢNH s VIDEO ĐÁP ỨNG (FLUID)

Video đã xem (Link):

1. Vấn đề (The Problem)

Ở các bài trước, chúng ta đã làm cho các "cái hộp" (layout) co giãn. Nhưng nếu bạn đặt một tấm ảnh có kích thước cố định (ví dụ `width="800px"`) vào một cái hộp chỉ rộng 300px (trên điện thoại), tấm ảnh sẽ **bị tràn ra ngoài**, phá vỡ toàn bộ giao diện.

2. Kiến thức cốt lõi (Giải pháp "Chất lỏng")

Để giải quyết vấn đề này, chúng ta cần biến hình ảnh từ "cứng" (fixed) thành "lỏng" (fluid), tức là nó *tffi* co giãn theo vật chứa nó.

Quy tắc Vàng (The Magic Rule):

Bạn gần như SẼ LUÔN LUÔN thêm đoạn code này vào file CSS của mình cho mọi dự án:

CSS

img, video, iframe

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

Sau khi xem video, hãy giải thích ý nghĩa của 2 thuộc tính này:

Thuộc tính	Mục đích sử dụng (Giải thích bằng lời của bạn)
max-width: 100%;	(Tự điền: Nghĩa là chiều rộng của ảnh CÙNG LẮM là...)
height: auto;	(Tự điền: Dùng để giữ đúng... của ảnh khi chiều rộng thay đổi)

Một thuộc tính nâng cao: object-fit

Khi bạn BẮT BUỘC cái hộp phải có kích thước cố định (ví dụ: `width: 200px, height: 200px`), nhưng ảnh của bạn lại là hình chữ nhật, ảnh sẽ bị méo. object-fit dùng để xử lý việc này.

- object-fit: cover; (Phổ biến nhất): Lấp đầy hộp___(Giữ tỷ lệ, lấp đầy hộp, chấp nhận bị xén)
- object-fit: contain; (Ít dùng hơn): Vừa khít hộp _____(Giữ tỷ lệ, vừa khít trong hộp, chấp nhận bị hở)

3. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Tạo 2 cái hộp. Hộp 1 cho thấy vấn đề (ảnh bị tràn). Hộp 2 cho thấy giải pháp (ảnh co lại vừa vặn).

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 17</title>
  <style>
    /* Style chung cho 2 cái hộp */
    .container {
      width: 300px; /* Một cái hộp RẤT NHỎ */
      border: 3px solid red; margin:
      20px;
    }

    /* KHÔNG làm gì với ảnh này */
    .img-problem {
      /* Mặc định, nó sẽ giữ kích thước gốc */
    }

    /* TODO: Yêu cầu
      Áp dụng "Quy tắc Vàng" cho ảnh này
      để nó tự co lại VỪA KHÍT 300px của thẻ cha.
    */
    .img-responsive {
max-width: 100%;
height: auto;
    }
  </style>
</head>
<body>

  <h1>Hộp 1: Vấn đề (Ảnh tràn)</h1>
  <div class="container">
    
  </div>

  <h1>Hộp 2: Giải pháp (Ảnh đáp ứng)</h1>
  <div class="container">
    
  </div>

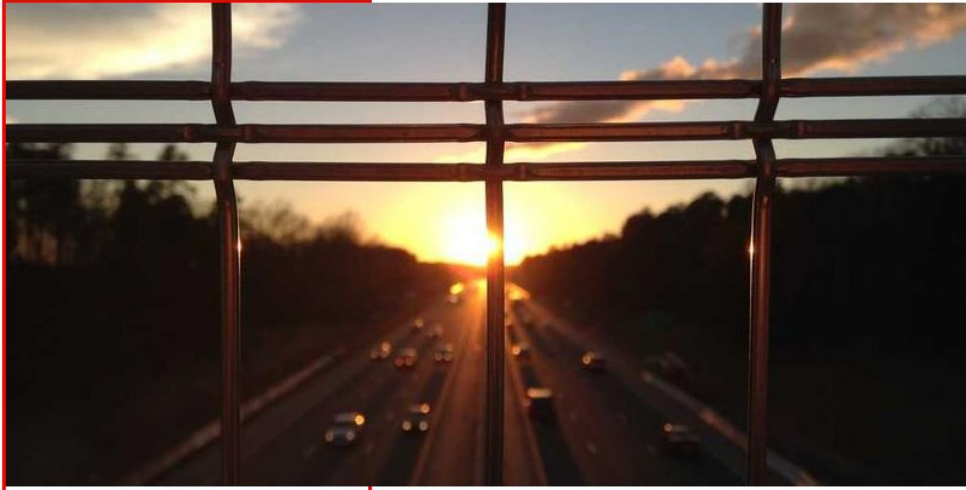
</body>
</html>
```

KẾT QUẢ QUAN SÁT:

Ảnh 1: Vấn đề (.img-problem)



Hộp 1: Vấn đề (Ảnh tràn)



(Dán ảnh chụp màn hình Hộp 1 của bạn vào đây)

- **Nhận xét:** (Gợi ý: Tầm ảnh 800px Vượt ra khỏi cái hộp màu đỏ 300px.)

Ảnh 2: Giải pháp (.img-responsive)

Hộp 2: Giải pháp (Ảnh đáp ứng)



(Dán ảnh chụp màn hình Hộp 2 của bạn vào đây)

- **Nhận xét:** (Gợi ý: Tầm ảnh 800px đã tự Thu nhỏ_để nằm Vừa khít bên trong cái hộp màu đỏ 300px.)

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Sự khác biệt thực sự giữa width: 100%; và max-width: 100%; là gì? Em dùng width: 100% cho ảnh có được không và tại sao max-width lại tốt hơn?")*

PHIẾU HỌC TẬP CSS [18] - CÁC ĐƠN VỊ TƯƠNG ĐỐI (REM, EM, VW, VH)

Họ và tên:

Lớp:

Video đã xem (Link):

1. Kiến thức cốt lõi (So sánh các đơn vị)

Cho đến nay, chúng ta chủ yếu dùng *px (pixels)*. Đây là đơn vị **tuyệt đối** (cố định). Để thiết kế *responsive* và linh hoạt, chúng ta cần các đơn vị **tương đối**.

Sau khi xem video, hãy hoàn thành bảng so sánh các đơn vị này:

Đơn vị	Tên đầy đủ	Hệ quy chiếu (Nó tính toán dựa trên cái gì?)
px	Pixel	(Tự điền: Đơn vị cố định, 1 pixel trên màn hình)
%	Percent	(Tự điền: Kích thước của thẻ CHA trực tiếp)
em	"M"	(Tự điền: font-size của thẻ CHA trực tiếp)
rem	Root "M"	(Tự điền: font-size của thẻ GỐC (là thẻ <html>))
vw	Viewport Width	(Tự điền: 1% của chiều RỘNG màn hình/viewport)
vh	Viewport Height	(Tự điền: 1% của chiều CAO màn hình/viewport)

Câu hỏi bắt buộc quan trọng:

1. Vấn đề "lồng nhau" (compounding) là gì? Nó xảy ra với em hay rem?

"Lồng nhau" nghĩa là khi bạn dùng em, kích thước của phần tử con phụ thuộc vào cha, nên nếu cha đã được nhân to bằng em, con lại tiếp tục nhân lên, dẫn đến kích thước phóng đại nhiều tầng.

Hiện tượng này xảy ra với em,

2. Tại sao rem thường được coi là lựa chọn tốt nhất cho font-size trong các dự án responsive?

Vì rem luôn dựa trên font-size của thẻ <html>, giúp bạn kiểm soát toàn bộ kích thước chữ chỉ bằng cách thay đổi 1 giá trị duy nhất. Điều này làm cho thiết kế ổn định, dễ bảo trì và không bị ảnh hưởng bởi việc lồng nhau như khi dùng em.

3. Ví dụ thực hành (Hiểm họa của em và sức mạnh của rem)

Yêu cầu: Tạo file HTML/CSS bên dưới, chạy và quan sát sffii khác biệt Cffii LỚN giữa em và rem khi chúng lồng vào nhau.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Bài tập CSS 18</title>
<style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */
```

```

/* Yêu cầu 1: Đặt font-size GỐC (root) là 16px */
html {
    font-size: 16px;
    line-height: 1.5;
}

body { font-family: Arial, sans-serif; }

/* Yêu cầu 2: Vấn đề của 'em'
    Mỗi thẻ con sẽ LỚN DẦN LÊN
    (vì nó = 1.2 lần font-size của cha nó)
*/
.container-em {
    /* 1.2 * 16px = 19.2px */ font-
    size: 1.2em; border: 2px
    solid blue; padding: 10px;
    margin: 20px;
}
.container-em .child {
    /* 1.2 * 19.2px = 23.04px */ font-
    size: 1.2em; background-color:
    #d6eaf8;
}

/* Yêu cầu 3: Giải pháp 'rem'
    Mọi thẻ con sẽ BẰNG NHAU
    (vì nó luôn = 1.2 lần font-size GỐC (16px))
*/
.container-rem {
    /* 1.2 * 16px = 19.2px */ font-
    size: 1.2rem; border: 2px
    solid red; padding: 10px;
    margin: 20px;
}
.container-rem .child {
    /* 1.2 * 16px = 19.2px */ font-
    size: 1.2rem; background-
    color: #fadedE;
}

/* Yêu cầu 4: Đơn vị 'vw'
    Đặt cỡ chữ bằng 5% chiều rộng màn hình. Hãy
    thử co giãn cửa sổ trình duyệt!
*/
.title-vw {
    font-size: 5vw;
}

```

</style>

</head>

<body>

<div class="container-em">

```

    Thẻ cha (font-size: 1.2em)
    <div class="child">
        Thẻ con (font-size: 1.2em) -> Chữ tôi BỊ TO HƠN cha!
    </div>
</div>

<div class="container-rem">
    Thẻ cha (font-size: 1.2rem)
    <div class="child">
        Thẻ con (font-size: 1.2rem) -> Chữ tôi BẰNG cha!
    </div>
</div>

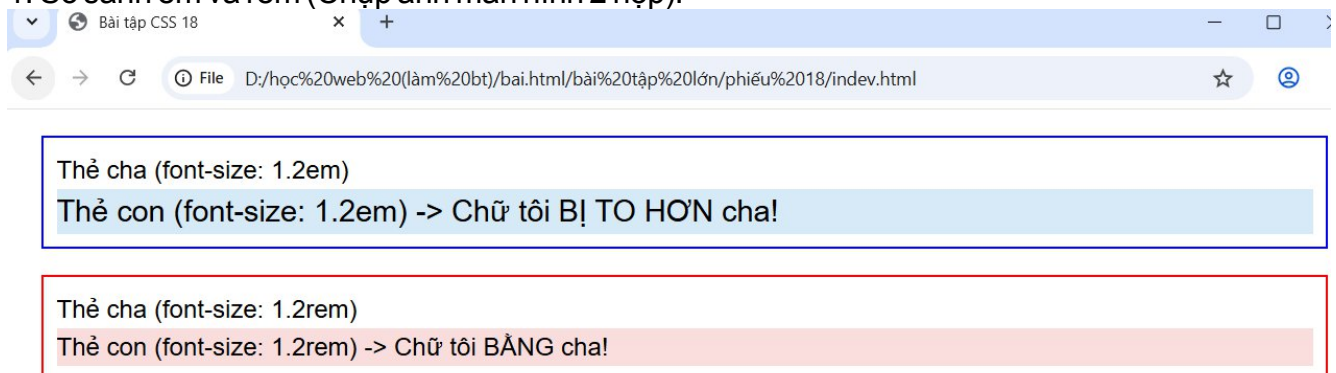
<h1 class="title-vw"> Hãy
    co giãn tôi!
</h1>

</body>
</html>

```

KẾT QUẢ QUAN SÁT:

1. So sánh em và rem (Chụp ảnh màn hình 2 hộp):



(Dán ảnh chụp màn hình Hộp 1 (xanh) và Hộp 2 (đỏ) vào đây)

- **Nhận xét (Bắt buộc):** Chữ "Thẻ con" ở hộp màu xanh (em) và hộp màu đỏ (rem) có kích thước như thế nào so với chữ "Thẻ cha" của chúng? Tại sao lại có sự khác biệt đó?
- Ở hộp **xanh (em)**: dòng chữ "Thẻ con" **to hơn** "Thẻ cha".
- Ở hộp **đỏ (rem)**: dòng chữ "Thẻ con" **bằng kích thước** "Thẻ cha".
- Vì Đơn vị em phụ thuộc vào **font-size của phần tử cha** → càng lồng sâu, kích thước càng nhân lên.
- Đơn vị rem phụ thuộc vào **font-size của root (html)** → không bị cộng dồn.

2. Thí nghiệm với vw (Video hoặc GIF):

(Hãy quay một video ngắn hoặc mô tả lại hành động của bạn)

- **Mô tả (Bắt buộc):** Khi bạn co/giãn cửa sổ trình duyệt, điều gì xảy ra với tiêu đề "Hãy co

giãn tôi!"?

- Khi co/giãn cửa sổ trình duyệt, dòng chữ “Hãy co giãn tôi!” thay đổi kích thước theo chiều rộng cửa sổ:

Khi cửa sổ nhỏ → chữ nhỏ lại.

Khi cửa sổ to → chữ phóng to theo.

4. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Vậy tóm lại, khi nào em nên dùng rem, khi nào dùng % và khi nào dùng px? Em có nên bỏ hoàn toàn px không?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Tạo phần tử "ảo")

Giải-phân tử (Pseudo-elements) là các "phần tử ảo" mà bạn có thể tạo và style bằng CSS. Chúng không tồn tại thực sự trong file HTML, nhưng hiển thị trên trang web. Hai cái phổ biến nhất là `::before` và `::after`.

Sau khi xem video, hãy hoàn thành bảng sau:

Tên Giải-phần tử	Cú pháp (Ví dụ)	Ý nghĩa (Nó chèn nội dung vào đâu?)
<code>::before</code>	<code>p::before { ... }</code>	(Tự điền: Chèn nội dung ảo VÀO BÊN TRONG thẻ p,...)
<code>::after</code>	<code>p::after { ... }</code>	(Tự điền: Chèn nội dung ảo VÀO BÊN TRONG thẻ p,...)

Xuất sang Trang tính

Câu hỏi BẮT BUỘC SỐNG CÒN:

- tính nào là **TUYỆT ĐỐI BẮT BUỘC** phải có (kể cả khi giá trị của nó là rỗng) để `::before` Thuộc hoặc `::after` có thể hiển thị?
 - Trả lời: Thuộc tính `(content: "");`
- Mặc định, các phần tử "ảo" này có display là gì? (inline hay block?)
 - Trả lời: `inline`

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS trong thẻ `<style>` để thêm các yếu tố trang trí cho tiêu đề và liên kết mà **không** được sửa file HTML.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 19</title>
  <style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

    /* Yêu cầu 1:
       Dùng '::before' để thêm một icon "liên kết" (🔗)
       vào TRƯỚC nội dung của thẻ .link (Nhớ
       thêm một khoảng trống sau icon)
    */
    .link::before {
```

```

        content: "🔪 ";
    }

    /* Yêu cầu 2:
        Dùng '::after' để tạo một đường gạch chân trang trí cho tiêu
        đề .title
    */
    .title {
        /* Cần 'relative' để định vị đường gạch (nâng cao) */
        position: relative;
        padding-bottom: 5px; /* Tạo không gian cho gạch */
    }

    .title::after {
        /* 1. BẮT BUỘC phải có */
        content: "";

        /* 2. Tạo hình dạng cho "đường gạch" */
        display: block; /* Biến nó thành block để có W/H */
        width: 50px; height:
        3px;
        background-color: #3498db; /* Màu xanh */

        /* 3. Định vị nó (nâng cao) */
        position: absolute;
        bottom: 0;
        left: 0;
    }

</style>
</head>
<body>

    <h2 class="title">Đây là Tiêu đề Trang trí</h2>

    <p>
        Hãy truy cập vào
        <a class="link" href="#">đường link quan trọng</a> này.
    </p>

</body>
</html>

```

Kết quả (Chụp ảnh màn hình và dán vào đây): (Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả phải là:

Đây là Tiêu đề Trang trí

Hãy truy cập vào [đường link quan trọng](#) này.

- *Tiêu đề <h2> có một đường gạch chân ngắn màu xanh bên dưới.*
 - *Chữ "đường link quan trọng" có một icon ³ nằm ngay trước nó.)*
-

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em thấy trên mạng có người viết :before (1 dấu hai chấm), có người viết ::before (2 dấu hai chấm). Chúng khác gì nhau và em nên dùng cái nào?")*

PHIẾU HỌC TẬP CSS [20] - BIẾN ĐỔI (TRANSFORMS)

Video đã xem (Link):

1. Kiến thức cốt lõi (Thay đổi hình dạng)

Thuộc tính *transform* cho phép bạn di chuyển, xoay, phóng to/thu nhỏ một phần tử mà **không ảnh hưởng đến vị trí** của các phần tử khác xung quanh nó.

Sau khi xem video, hãy điền vào mục đích của các hàm transform phổ biến:

Hàm transform	Mục đích sử dụng (Dùng để làm gì?)	Ví dụ cú pháp
translate()	(Tự điền: Dùng để DI CHUYỂN...)	translate(50px, 10px)
rotate()	(Tự điền: Dùng để XOAY...)	rotate(45deg)
scale()	(Tự điền: Dùng để PHÓNG TO/THU NHỎ...)	scale(1.5) (Phóng to 1.5 lần)
skew()	(Tự điền: Dùng để LÀM MÉO/NGHIÊNG...)	skew(10deg)

Câu hỏi bắt buộc:

- Đây là sự khác biệt lớn nhất giữa việc dùng transform: translate(50px); và margin-left: 50px; để di chuyển một cái hộp? (Gợi ý: Hãy nghĩ về các phần tử xung quanh nó).

translate() Dịch chuyển phần tử **trực quan (visual only)**, **không ảnh hưởng đến bố cục** xung quanh.

margin-left thì **thay đổi vị trí thật** trong bố cục trang

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Viết CSS để khi người dùng **DI CHUỘT (hover)** vào từng hộp, nó sẽ *thể hiện* một "biến đổi" khác nhau.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 20</title>
  <style>
    /* --- KHÔNG SỬA PHẦN NÀY --- */
    body { font-family: Arial, sans-serif; padding: 50px; }
    .box {
      width: 150px;
      height: 150px;
      background-color: #3498db;
      color: white;
      font-size: 20px;
      display: flex;
```

justify-content:center; align-
items:center;

```
        margin: 30px;
    }
    /* --- KẾT THÚC PHẦN KHÔNG SỬA --- */

    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

    /* Yêu cầu 1:
        Khi HƠ CHUỘT vào .box-1,
        hãy DI CHUYỂN (translate) nó sang phải 50px.
    */
    .box-1:hover {
transform: translateX(50px);

    }

    /* Yêu cầu 2:
        Khi HƠ CHUỘT vào .box-2, hãy
        XOAY (rotate) nó 45 độ.
    */
    .box-2:hover {
transform: rotate(45deg);

    }

    /* Yêu cầu 3:
        Khi HƠ CHUỘT vào .box-3,
        hãy PHÓNG TO (scale) nó lên 1.2 lần.
    */
    .box-3:hover {
transform: scale(1.2);

    }

</style>
</head>
<body>

    <div class="box box-1">Translate</div>

    <div class="box box-2">Rotate</div>

    <div class="box box-3">Scale</div>

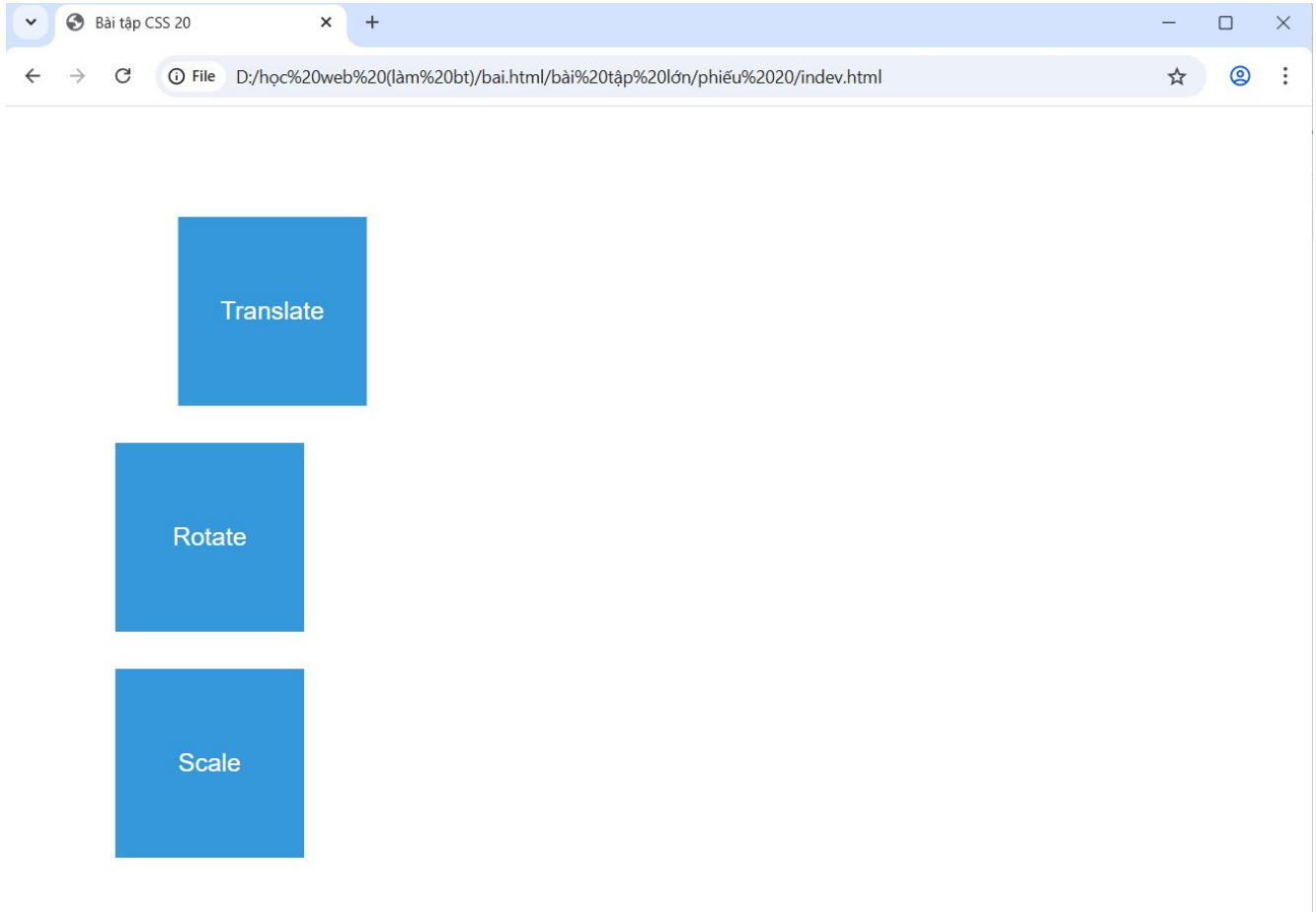
</body>
</html>
```

Kết quả (Chụp 3 ảnh màn hình):

Vì đây là hiệu ứng :hover, bạn hãy chụp 3 ảnh: 1 ảnh khi đang hover chuột vào Hộp 1, 1 ảnh khi đang hover chuột vào Hộp 2, và 1 ảnh khi đang hover chuột vào Hộp 3.

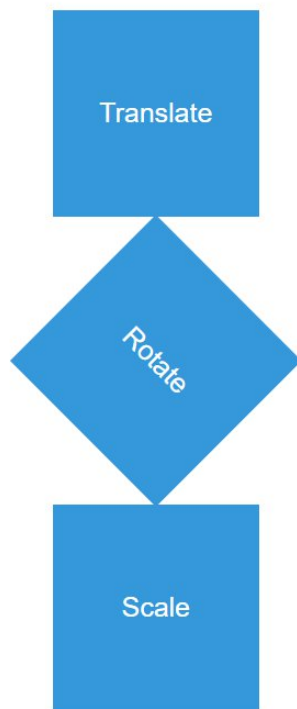
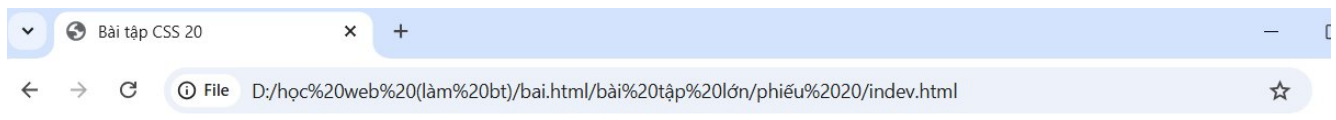
Ảnh 1: Hover chuột vào Hộp 1

(Dán ảnh chụp màn hình của bạn vào đây)



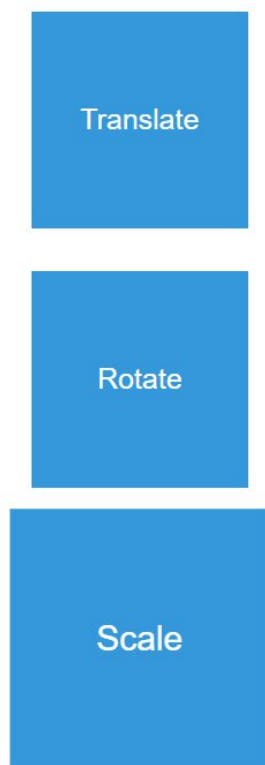
Ảnh 2: Hơ chuột vào Hộp 2

(Dán ảnh chụp màn hình của bạn vào đây)



Ảnh 3: Hơ chuột vào Hộp 3

(Dán ảnh chụp màn hình của bạn vào đây)



3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Ở Bài 20, cái hộp biến đổi rất nhanh. Làm thế nào để em làm cho nó XOAY hoặc DI CHUYỂN một cách MƯỢT MÀ (smooth) trong 0.5 giây thay vì giật một cái?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Làm mượt hiệu ứng)

Ở Bài 20, khi bạn :hover vào một phần tử, sự thay đổi (ví dụ transform) xảy ra ngay lập tức, trông rất "giật".

Thuộc tính transition (Chuyển tiếp) cho phép bạn báo với trình duyệt: "Này, nếu thuộc tính này (ví dụ background-color hay transform) thay đổi, hãy thay đổi nó một cách từ từ và mượt mà trong một khoảng thời gian nhất định."

Sau khi xem video, hãy điền vào mục đích của các thuộc tính con của transition:

Thuộc tính	Mục đích sử dụng (Dùng để làm gì?)	Ví dụ
transition-property	(Tự điền: Chỉ định thuộc tính nào...)	transform, background-color, all
transition-duration	(Tự điền: Chỉ định thời gian...)	0.5s (nửa giây), 300ms
transition-timing-function	(Tự điền: Chỉ định "đường cong tốc độ"...)	linear, ease, ease-in-out
transition-delay	(Tự điền: Chỉ định thời gian TRÌ HOÃN...)	1s (chờ 1 giây rồi mới chạy)

Cú pháp viết tắt (Shorthand):

Chúng ta gần như luôn dùng cú pháp tắt, viết tắt cả trên 1 dòng. Thứ tự phổ biến là:

transition: [property] [duration] [timing-function] [delay];

Ví dụ: transition: all 0.3s ease;

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Lấy lại BÀI TẬP CỦA BÀI 20, và thêm **một dòng** transition duy nhất để làm mượt tất cả các hiệu ứng.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 21</title>
  <style>
```

```
/* --- KHÔNG SỬA PHẦN NÀY (STYLE TỪ BÀI 20) --- */
```

```
body { font-family: Arial, sans-serif; padding: 50px; }
```

```
.box {  
    width: 150px;  
    height: 150px;  
    background-color: #3498db;  
    color: white;  
    font-size: 20px;  
    display: flex;  
    justify-content: center; align-  
    items: center; margin: 30px;  
}
```

```
/* Hiệu ứng HOVER (Từ Bài 20) */
```

```
.box-1:hover {  
    transform: translateX(50px);  
    background-color: #e74c3c; /* Thêm đổi màu */  
}  
.box-2:hover {  
    transform: rotate(45deg);  
}  
.box-3:hover {  
    transform: scale(1.2);  
}
```

```
/* --- KẾT THÚC PHẦN KHÔNG SỬA --- */
```

```
/* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */
```

```
/* Yêu cầu:
```

Thêm thuộc tính 'transition' vào class .box (thêm vào trạng thái gốc, KHÔNG PHẢI :hover)

- Áp dụng cho TẤT CẢ thuộc tính ('all')
- Thời gian chạy là 0.4 giây (0.4s)
- Hiệu ứng mượt là 'ease-in-out'

```
*/
```

```
.box {
```

```
transition: all 0.4s ease-in-out;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div class="box box-1">Translate</div>
```

```
<div class="box box-2">Rotate</div>
```

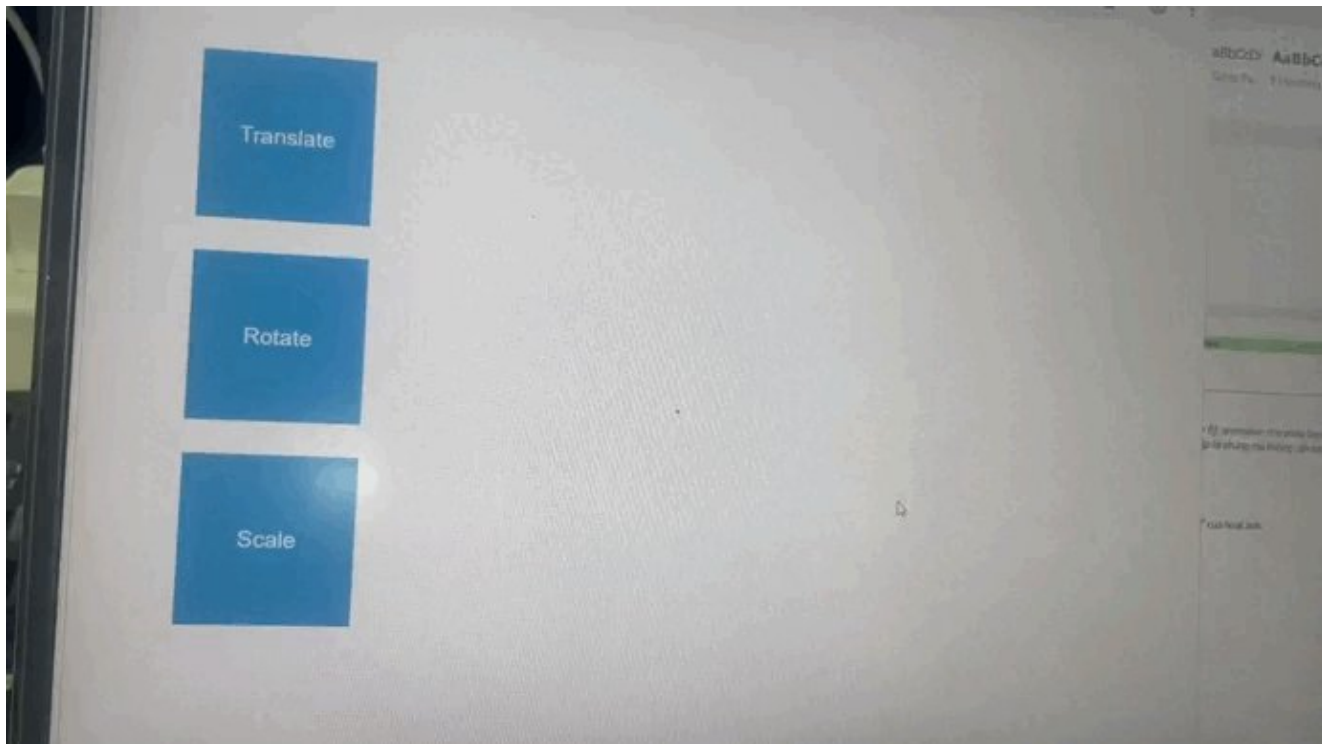
```
<div class="box box-3">Scale</div>
```

```
</body>
```

```
</html>
```

Kết quả (Video hoặc GIF):

Hiệu ứng tĩnh không thể hiện được bài này. Hãy quay một video ngắn (hoặc ảnh GIF) ghi lại cảnh bạn hover chuột lần lượt qua 3 cái hộp.



(Dán video/GIF của bạn vào đây)

Mô tả kết quả (Bắt buộc):

- Điều gì xảy ra khi bạn hover chuột vào Hộp 1? Nó còn "giật" như Bài 20 không?
- Khi hover chuột vào Hộp 1, hộp di chuyển sang phải 50px và đổi màu từ xanh sang đỏ một cách mượt mà, không còn bị giật như ở Bài 20
- Điều gì xảy ra khi bạn BỎ CHUỘT ra khỏi hộp? (Nó có mượt mà quay về không?)
- Khi rời chuột khỏi Hộp 1, hộp từ từ trượt trở lại vị trí ban đầu và đổi lại màu xanh cũng êm ái, không bị thay đổi đột ngột

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em đã làm mượt được khi hover. Nhưng nếu em muốn một cái hộp tự xoay vòng vòng liên tục mà không cần hover chuột thì làm thế nào?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Tạo chuyển động phức tạp)

transition (Bài 21) chỉ là chuyển đổi 2 trạng thái (A -> B). *animation* cho phép bạn tạo ra một chuỗi chuyển động phức tạp (A -> B -> C -> D...) và lặp lại chúng mà không cần tương tác của người dùng.

Tạo một animation luôn gồm 2 BƯỚC:

Bước 1: Định nghĩa "kịch bản" bằng @keyframes

- @keyframes là nơi bạn định nghĩa các "chặng" của hoạt ảnh.
- Cú pháp:

CSS

```
@keyframes ten-hoat-anh-cua-ban { from
  { /* Trạng thái bắt đầu */
    opacity: 0;
  }
  to { /* Trạng thái kết thúc */
    opacity: 1;
  }
}

/* Hoặc dùng % cho nhiều chặng */
@keyframes ten-phuc-tap
  { 0% { left: 0px; } 50%
    { left: 200px; } 100%
    { left: 0px; }
}
```

Bước 2: Áp dụng "kịch bản" đó cho phần tử

- Dùng thuộc tính animation để "gọi" @keyframes và ra lệnh cho nó chạy.

Thuộc tính animation	Mục đích sử dụng (Dùng để làm gì?)	Ví dụ
animation-name	(Tự điền: Tên của @keyframes...)	ten-hoat-anh-cua-ban
animation-duration	(Tự điền: Thời gian để hoàn thành 1 vòng...)	2s

animation-iteration-count	(Tự điền: Số lần lặp lại...)	3, infinite (vô tận)
Thuộc tính animation	Mục đích sử dụng (Dùng để làm gì?)	Ví dụ
animation-direction	(Tự điền: Hướng chạy...)	normal, alternate (chạy ngược lại)
animation-timing-function	(Tự điền: "Đường cong tốc độ"...)	ease, linear (đều đặn)

Cú pháp viết tắt (Shorthand):

animation: [name] [duration] [timing-function] [iteration-count] [direction]; Ví

dụ: animation: spin 2s linear infinite;

2. Ví dụ thực hành (Bắt buộc)

Yêu cầu: Tạo 2 hoạt ảnh chạy liên tục mà không cần hơ chuột:

1. Một hộp tự xoay 360 độ (.box-spin).
2. Một hộp tự "phập phồng" (lớn lên, nhỏ lại) (.box-pulse).

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 22</title>
  <style>
    /* --- Style chung --- */
    body { font-family: Arial, sans-serif; padding: 100px; }
    .box {
      width: 100px;
      height: 100px;
      background-color: #9b59b6; /* Tím */ color:
      white;
      display: flex;
      justify-content:center; align-
      items: center; margin: 50px;
    }

    /* --- VIẾT CODE CSS CỦA BẠN VÀO ĐÂY --- */
```

/* Yêu cầu 1: Định nghĩa kịch bản XOAY */

```
@keyframes spin { from  
  {  
    transform: rotate(0deg);  
  }  
}
```

```

        to {
            /* TODO: Xoay đến 360 độ */
transform: rotate(360deg);
        }
    }

    /* Yêu cầu 2: Định nghĩa kịch bản PHỒNG (pulse) */
    @keyframes pulse { 0%
        {
            transform: scale(1);
        }
        50% {
            /* TODO: Phóng to 1.2 lần */
transform: scale(1.2);
        }
        100% {
            transform: scale(1);
        }
    }

    /* Yêu cầu 3: Áp dụng kịch bản 'spin' cho .box-spin
        - Tên: spin
        - Thời gian: 2s
        - Tốc độ: linear (đều)
        - Lặp lại: infinite (vô tận)
    */
    .box-spin {
animation: spin 2s linear infinite;
    }

    /* Yêu cầu 4: Áp dụng kịch bản 'pulse' cho .box-pulse
        - Tên: pulse
        - Thời gian: 3s
        - Lặp lại: infinite (vô tận)
    */
    .box-pulse {
        background-color: #e74c3c; /* Đỏ */
animation: pulse 3s ease-in-out infinite;
    }

</style>
</head>
<body>

    <div class="box box-spin">Xoay...</div>

    <div class="box box-pulse">Phồng...</div>

</body>

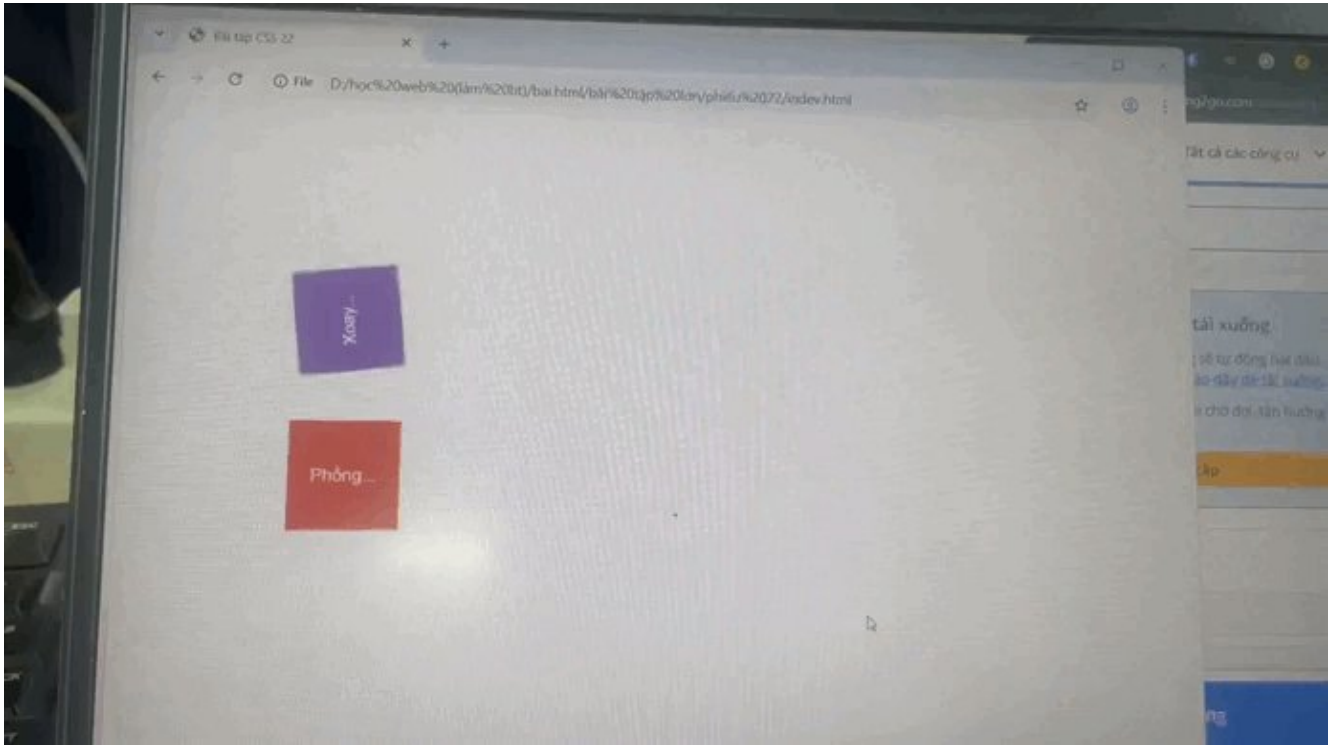
```

</html>

Kết quả (Video hoặc GIF):

Hãy quay một video ngắn (hoặc ảnh GIF) ghi lại 2 cái hộp đang tự chuyển động.

(Dán video/GIF của bạn vào đây)



Mô tả kết quả (Bắt buộc):

- Điều gì xảy ra với 2 cái hộp ngay khi tải trang (mà không cần hơ chuột)?
- hai hộp tự động bắt đầu chuyển động mà không cần hơ chuột.

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em thấy animation có alternate (chạy qua chạy lại), transition có làm được như vậy không? Hay transition chỉ chạy 1 chiều?")*

Video đã xem (Link):

1. Kiến thức cốt lõi (Sức mạnh của "Lưu trữ")

Giả sử dffí án của bạn dùng một "màu chủ đạo" (ví dụ: màu xanh) ở 50 nơi khác nhau. Khi sắp đổi ý muốn đổi sang "màu đỏ", bạn phải tìm và sửa 50 dòng code đó.

Biến CSS (CSS Variables), hay còn gọi là **Custom Properties**, cho phép bạn lưu giá trị đó (ví dụ: màu xanh) vào MỘT "biến" duy nhất. Khi cần đổi, bạn chỉ cần sửa 1 nơi duy nhất.

Sau khi xem video, hãy điền vào 2 cú pháp quan trọng:

1. Cú pháp ĐỊNH NGHĨA Biến (Thường đặt trong :root):

- :root là bộ chọn đại diện cho thẻ <html> (thẻ gốc), giúp biến có thể được dùng ở MỌI NƠI.
- Cú pháp (Tự điền):

CSS

```
:root {  
    --ten-bien-cua-ban: gia-tri;  
}
```

(Ví dụ: `--primary-color: #34S8db;`) Lưu ý: Tên biến **BẮT BUỘC** bắt đầu bằng 2 dấu gạch ngang_____

2. Cú pháp SỬ DỤNG Biến:

- Để gọi giá trị đã lưu, bạn dùng hàm var().
- Cú pháp (Tự điền):

CSS

```
.box {  
    background-color: var(--ten-bien-cua-ban);  
}
```

(Ví dụ: `background-color: var(--primary-color);`)

2. Ví dụ thực hành (Thay đổi giao diện trong 1 nốt nhạc)

Yêu cầu: Tạo một giao diện nhỏ dùng Biến CSS. Sau đó, thử thay đổi toàn bộ giao diện chỉ bằng cách sửa 2 dòng trong :root.

Mã HTML của tôi (index.html):

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bài tập CSS 23</title>
  <style>
    /* VIẾT CODE CSS CỦA BẠN VÀO ĐÂY */

    /* Yêu cầu 1: Định nghĩa 2 biến toàn cục
      - Biến '--main-color' có giá trị là 'darkblue'
      - Biến '--padding-size' có giá trị là '15px'
    */
    :root {

    }

    /* Yêu cầu 2: Sử dụng các biến
      (KHÔNG gõ 'darkblue' hay '15px' ở dưới đây)
    */
    body {
      font-family: Arial, sans-serif;
    }

    h1 {
      /* Dùng biến '--main-color' cho màu chữ */
      color: var(--main-color);
    }

    button {
      /* Dùng biến '--main-color' cho màu nền */
      background-color: var(--main-color);

      /* Dùng biến '--padding-size' cho padding */
      padding: var(--padding-size);

      color: white;
      border: none;
      font-size: 16px;
    }

    .note {
      margin-top: 20px;

      /* Dùng biến '--padding-size' cho padding */
      padding: var(--padding-size);

      /* Dùng biến '--main-color' cho viền */
```

```
border: 2px solid var(--main-color);
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

`<h1>Tiêu đề này dùng Biến CSS</h1>`

`<button>Nút này cũng dùng Biến CSS</button>`

`<div class="note">`

Cái hộp này cũng dùng Biến CSS

`</div>`

`</body>`

`</html>`

KẾT QUẢ QUAN SÁT:

Ảnh 1: Giao diện "Màu xanh" (Lần chạy đầu) (Dán ảnh chụp màn hình trình duyệt của bạn vào đây. Kết quả là tiêu đề, nút, và viền hộp đều có màu darkblue.)



Thí nghiệm (Bắt buộc):

- Bây giờ, quay lại code CSS, vào phần `:root` và sửa 2 dòng: `--main-color: darkred;` `--padding-size: 30px;`

Ảnh 2: Giao diện "Màu đỏ" (Sau khi sửa biến) (Dán ảnh chụp màn hình kết quả MỚI của bạn vào đây.)



- Nhận xét (Bắt buộc):**
- Sau khi sửa 2 dòng trong `:root`, điều gì đã xảy ra với Tiêu đề, Nút bấm và Cái hộp?

- Màu sắc của toàn bộ trang thay đổi:
 - tiêu đề (<h1>). Nút bấm (<button>) Và viền của khung (.note) đều chuyển từ màu xanh đậm (darkblue) sang màu đỏ đậm (darkred) — mà không cần chỉnh từng phần tử.
 - Kích thước padding của Nút và Hộp thay đổi như thế nào?
 - tăng gấp đôi — từ 15px lên 30px
-

3. Câu hỏi/Thắc mắc của tôi

Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm về chủ đề này:

- (Ví dụ: "Em có thể định nghĩa biến bên trong một class (ví dụ .container { --my-color: red; }) thay vì :root không? Nếu có thì nó khác gì?")*

Họ và tên: PHẠM LÊ HOÀNG

Lớp: CNTT 19-06

Link nộp bài (GitHub/CodePen): <https://github.com/huangphm-byte>

1. Mục tiêu Dự án

Dự án này là bài kiểm tra tổng hợp. Bạn phải vận dụng **TẤT CẢ** các kiến thức CSS đã học (từ Box Model, Flexbox, Grid, đến Responsive) để xây dựng một trang web hoàn chỉnh từ đầu.

Đề tài: Xây dựng một trang **Portfolio cá nhân** (trang giới thiệu bản thân/dự án) ở dạng *một trang duy nhất (single page)*.

2. Yêu cầu BẮT BUỘC (Checklist tính điểm)

Trang web của bạn phải đáp ứng đầy đủ các yêu cầu sau:

Yêu cầu về Cấu trúc HTML (Nội dung)

- ☐ Sử dụng HTML ngữ nghĩa (<header>, <nav>, <main>, <section>, <footer>).
- ☐ Trang web phải có ít nhất 5 phần (section):
 - Header s Navbar:** Logo (tên của bạn) và các link điều hướng (Giới thiệu, Dự án, Liên hệ...).
 - Giới thiệu (Hero):** Một khu vực lớn giới thiệu bạn là ai, có ảnh của bạn và một nút "Call to Action" (ví dụ: "Xem dự án").
 - Dự án (Projects):** Hiển thị ít nhất 3 dự án bạn đã làm (có thể là bài tập lớn, ảnh demo...).
 - Kỹ năng (Skills):** Liệt kê các kỹ năng (ví dụ: HTML, CSS, JavaScript...).
 - Footer:** Thông tin bản quyền, link mạng xã hội.

Yêu cầu về CSS (Kỹ thuật)

- ☐ **CSS Variables:** Phải sử dụng Biến CSS (:root) để quản lý màu sắc chủ đạo và font chữ. (Bài 23)
- ☐ **Box Model:** Sử dụng box-sizing: border-box; cho toàn bộ dự án. padding và margin phải được dùng hợp lý. (Bài 9 & 10)
- ☐ **Fonts:** Phải nhúng và sử dụng ít nhất 1 **Google Font**. (Bài 8)
- ☐ **Layout (BẮT BUỘC):**
 - Phải sử dụng **Flexbox** cho ít nhất 1 khu vực (ví dụ: căn chỉnh navbar, căn giữa nội dung hero). (Bài 13)

- Phải sử dụng **Grid** cho ít nhất 1 khu vực (ví dụ: chia lưới cho khu vực "Dự án"). (Bài 14)
- **[] Hiệu ứng:**
 - Phải sử dụng **Transition** (Bài 21) cho các hiệu ứng :hover (ví dụ: khi di chuột vào nút bấm, link, hoặc thẻ dự án).
 - (Tùy chọn, cộng điểm): Sử dụng ::before/::after (Bài 19) hoặc animation (Bài 22) để tạo các chi tiết trang trí.

Yêu cầu về Responsive (RẤT QUAN TRỌNG)

- **[] Meta Viewport:** Phải có thẻ <meta name="viewport"...> (Bài 15)
- **[] Hình ảnh đáp ứng:** Mọi hình ảnh () phải tự co giãn (max-width: 100%). (Bài 17)
- **[] Media Queries:** Phải sử dụng @media (Bài 16) để đảm bảo layout hiển thị tốt trên 2 kích thước:
 1. **Mobile (dưới 768px):** Layout hiển thị 1 cột, các dự án xếp chồng lên nhau.
 2. **Desktop (trên 768px):** Layout hiển thị nhiều cột (ví dụ: khu vực dự án hiển thị lưới 3 cột).

3. Hướng dẫn thực hiện (Gợi ý các bước)

Bạn nên thực hiện theo triết lý "Mobile First" để dễ quản lý.

1. **Bước 1 (Content):** Viết TOÀN BỘ nội dung HTML thô trước. Đảm bảo có đủ 5 section.
2. **Bước 2 (Setup CSS):**
 - Tạo file style.css và liên kết vào HTML.
 - Thiết lập box-sizing: border-box;
 - Định nghĩa các Biến CSS (:root) cho màu sắc.
 - Nhúng Google Font và setup body.
3. **Bước 3 (Mobile Layout - Dưới 768px):**
 - Bắt đầu style cho màn hình điện thoại (1 cột).
 - Style cho header, navbar (có thể dùng Flexbox cho navbar ngay).
 - Style cho từng section (chúng sẽ tự động xếp chồng lên nhau).
4. **Bước 4 (Desktop Layout - Trên 768px):**
 - Mở một @media (min-width: 768px) { ... }.
 - Bên trong media query này, viết các quy tắc để thay đổi layout.
 - Ví dụ: Dùng display: grid và grid-template-columns: 1fr 1fr 1fr; để chia lại lưới cho khu vực dự án.

5. Bước 5 (Transitions):

- Quay lại CSS gốc (ngoài media query).
- Thêm transition: all 0.3s ease; cho các nút bấm, các thẻ dự án.
- Viết các quy tắc :hover (ví dụ: transform: scale(1.05); cho thẻ dự án) để tạo hiệu ứng.

6. Bước 6 (Kiểm tra):

- Dùng F12 (Developer Tools) để kiểm tra trên nhiều kích thước màn hình.
- Đảm bảo không có gì bị tràn, vỡ layout.

4. Câu hỏi/Thắc mắc (Hãy hỏi trước khi bắt tay làm)

(Phần này để sinh viên điền vào các vướng mắc của mình trước khi bắt đầu code)

- (Ví dụ: "Em không biết tìm ảnh demo cho dự án ở đâu?", "Em không biết làm sao để navbar responsive trên mobile (tạo menu hamburger)?")*

Như vậy, chúng ta đã hoàn thành toàn bộ 24 bài học trong outline bạn yêu cầu. Chúc bạn và các sinh viên có một dự án cuối kỳ thành công!