Phiếu 16

PHIẾU HỌC TẬP CSS [16] - MEDIA QUERIES (Bí thuật Responsive)

**I. Kiến thức cốt lõi**

1. @media: Dùng để làm gì?
   * @media dùng để “ràng buộc” (điều kiện “nếu... thì...”) áp dụng CSS chỉ khi điều kiện về môi trường thiết bị (ví dụ loại màn hình, chiều rộng, chiều cao, độ phân giải...) được đáp ứng. Nhờ đó, giao diện web có thể tự động thay đổi linh hoạt, phù hợp nhiều loại thiết bị khác nhau.​
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 HOẶC BẰNG 768px.
   * Tức là: viết CSS cơ bản cho mobile, rồi “nâng cấp” (enhance) bố cục/kiểu dáng khi màn hình rộng 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 HOẶC BẰNG 600px.
   * Tức là: mặc định giao diện desktop, nhưng khi màn hình thu nhỏ (vd. mobile) thì đổi sang style khác.​
4. Breakpoints (Điểm ngắt)
   * Các giá trị như 768px, 992px,... được gọi là Breakpoints (điểm ngắt): Là các mốc chiều rộng màn hình mà tại đó layout hoặc style sẽ thay đổi, giúp giao diện hiển thị đẹp trên từng loại thiết bị (mobile, tablet, desktop...).​

**II. Ví dụ thực hành (áp dụng triết lý Mobile First)**

Mã CSS mẫu:

css

*/\* Mặc định cho mobile (dưới 768px) \*/*

.box {

background-color: blue;

color: white;

margin: 8px 0;

width: 100%;

}

*/\* Khi màn hình lớn hơn hoặc bằng 768px (tablet/desktop) \*/*

@media (min-width: 768px) {

.container {

display: flex;

gap: 12px;

}

.box {

background-color: red;

width: 33.33%;

margin: 0;

}

}

Nhận xét thực hành:

* Ảnh 1: Màn hình điện thoại (dưới 768px)
* Ảnh có chứa văn bản, máy tính xách tay, Thiết bị hiển thị, đa phương tiện

  Nội dung do AI tạo ra có thể không chính xác.
  + 3 hộp màu xanh xếp dọc, mỗi hộp chiếm 100% chiều rộng container.

**III. Giải đáp thắc mắc về Mobile First & Desktop First**

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

* Mobile First (min-width):
  + CSS mặc định dành cho thiết bị nhỏ; dùng min-width để nâng cấp cho thiết bị lớn hơn.
  + Ưu điểm: dễ tối ưu hiệu suất (ít CSS hơn cho mobile), phù hợp xu hướng hiện đại (đa số truy cập web từ mobile).
  + Được khuyến khích dùng cho web mới, chuẩn hiện đại.
* Desktop First (max-width):
  + CSS mặc định dành cho desktop; dùng max-width để “hạ cấp” cho thiết bị nhỏ hơn.
  + Hợp với các trang web cũ hoặc tập trung vào trải nghiệm desktop/phức tạp ở desktop, mobile chỉ là phụ.​

=> Tóm lại:  
Mobile First (min-width) là xu hướng chủ đạo hiện nay, giúp web tối ưu trên mọi thiết bị. Desktop First phù hợp cho trường hợp đặc biệt cần ưu tiên desktop trước.

Nếu cần ví dụ cụ thể về breakpoints thường dùng hay cách chia layout responsive cho nhiều loại màn hình, mình sẽ hướng dẫn thêm nhé!

Phiếu 17

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

**1. Ý nghĩa hai thuộc tính “thần thánh”**

| **Thuộc tính** | **Mục đích sử dụng (giải thích bằng lời)** |
| --- | --- |
| max-width: 100%; | Nghĩa là chiều rộng của ảnh CÙNG LẮM chỉ bằng 100% vật chứa—ảnh sẽ co nhỏ lại vừa khít hộp nếu hộp nhỏ, nhưng không bao giờ nở lớn hơn kích thước gốc của nó nếu hộp rộng hơn ​. |
| height: auto; | Dùng để giữ đúng tỷ lệ/cạnh gốc của ảnh (không bị méo mó) khi chiều rộng thay đổi ​. |

**2. object-fit: cover vs contain**

* object-fit: cover;
  + Ảnh SẼ GIỮ đúng tỷ lệ gốc, tự động phóng to/thu nhỏ sao cho lấp đầy hộp, nhưng chấp nhận bị xén những phần ngoài hộp (ưu tiên tràn ra ngoài để điền đủ hộp—thường dùng cho avatar, banner, thumbnail...).
* object-fit: contain;
  + Ảnh SẼ GIỮ đúng tỷ lệ gốc, luôn nằm vừa khít trong hộp, nhưng sẽ có thể bị hở trắng hai bên hoặc phía trên/dưới nếu hộp và ảnh không đồng tỷ lệ (ưu tiên nhìn đủ toàn bộ ảnh, không quan tâm “đầy hộp”).​

**3. Góc so sánh width: 100% và max-width: 100% với ảnh**

* width: 100%; ép ảnh luôn căng hết vật chứa (nếu hộp to hơn ảnh, ảnh sẽ bị kéo giãn, nát!).
* max-width: 100%; chỉ cho phép ảnh co nhỏ đến cỡ hộp nếu hộp nhỏ, nhưng không bao giờ nở lớn hơn kích thước thật của chính nó nếu vật chứa quá to, nên luôn giữ nét.​
* => max-width: 100% là “an toàn” hơn, ảnh responsive đẹp trên hầu hết mọi giao diện!
* Code
* <!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 {

* }
* </style>
* </head>
* <body>
* <h1>Hộp 1: Vấn đề (Ảnh tràn)</h1>
* <div class="container">
* <img class="img-problem"
* src="https://picsum.photos/800/400"
* alt="Ảnh bị lỗi vỡ layout">
* </div>
* <h1>Hộp 2: Giải pháp (Ảnh đáp ứng)</h1>
* <div class="container">
* <img class="img-responsive"
* src="https://picsum.photos/800/400"
* alt="Ảnh đã responsive">
* </div>
* </body>
* </html>

**Mẫu CSS thực hành và nhận xét:**

Ảnh 1: Vấn đề (.img-problem)

* Nhận xét: [Tấm ảnh 800px] tràn ra khỏi cái hộp màu đỏ 300px.

Ảnh 2: Giải pháp (.img-responsive)

* Nhận xét: [Tấm ảnh 800px] đã tự co nhỏ lại để nằm trọn vẹn bên trong cái hộp màu đỏ 300px.
* Ảnh
* Ảnh có chứa văn bản, máy tính xách tay, Thiết bị hiển thị, đa phương tiện

  Nội dung do AI tạo ra có thể không chính xác.

Mẫu CSS giải pháp:

css

.img-responsive {

max-width: 100%;

height: auto;

}

**Gợi ý trả lời thắc mắc:**

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

* width: 100% sẽ luôn ép ảnh căng ra vừa hộp, dù hộp lớn hay nhỏ, gây vỡ hình nếu hộp lớn quá.

max-width: 100% thì chỉ “giới hạn trên”, khiến ảnh sẽ không bao giờ lớn hơn bình thường, không bị vỡ nét, và vẫn tự co nhỏ lại nếu hộp nhỏ—nên max-width: 100% thường tốt hơn cho ảnh responsive!

Phiếu 18

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

Họ và tên: Nguyễn Văn Trọng

Lớp : CNTT19-07

**Bảng so sánh các đơn vị trong CSS**

| **Đơn vị** | **Tên đầy đủ** | **Hệ quy chiếu (Nó tính toán dựa trên cái gì?)** |
| --- | --- | --- |
| px | Pixel | Đơn vị cố định, 1 pixel trên màn hình |
| % | Percent | Kích thước của thẻ CHA trực tiếp |
| em | "M" | font-size của thẻ CHA trực tiếp |
| rem | Root "M" | font-size của thẻ GỐC (<html>) |
| vw | Viewport Width | 1% của chiều RỘNG màn hình/viewport |
| vh | Viewport Height | 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?
   * Vấn đề compounding = “kết quả bị cộng dồn khi lồng nhiều cấp”. Đơn vị em sẽ bị ảnh hưởng bởi từng cấp cha trực tiếp, nên khi lồng nhiều lớp với font-size em, kích thước sẽ tăng/tăng lũy tiến ngoài ý muốn. rem thì KHÔNG bị lồng nhau, luôn dựa vào thẻ gốc (<html>) nên ổn định.​
2. Tại sao rem thường được coi là lựa chọn tốt nhất cho font-size trong các dự án responsive?
   * rem giúp font-size ổn định, dễ truy xuất, luôn dựa vào <html> (đổi root là đổi cả site). rem tối ưu cho responsive & accessibility vì người dùng tăng font trình duyệt thì toàn trang cùng lớn lên.​

**Nhận xét thực hành em vs rem**

* Chữ "Thẻ con" ở hộp màu xanh (em) sẽ lớn hơn cha vì bị compounding — mỗi cấp cộng thêm tỷ lệ!
* Chữ "Thẻ con" ở hộp màu đỏ (rem) sẽ bằng cha vì luôn lấy font-size <html> làm chuẩn, KHÔNG bị cộng dồn.​

**Thí nghiệm với vw (viewport width)**

* Khi co/giãn cửa sổ trình duyệt, tiêu đề "Hãy co giãn tôi!" sẽ thay đổi kích thước liên tục theo chiều rộng viewport—càng thu nhỏ cửa sổ thì tiêu đề càng nhỏ lại, càng rộng thì lớn ra—rất phù hợp cho tiêu đề, banner responsive!

**Tóm tắt khi nào dùng rem, %, px?**

* rem: dùng cho font-size cơ bản, padding, margin cần đồng bộ & scale toàn site (tốt cho responsive, accessibility).
* %: dùng cho width, height, spacing khi muốn tỉ lệ theo thẻ cha trực tiếp (khi bố cục linh hoạt theo vùng chứa).
* px: dùng cho các giá trị cần cố định tuyệt đối như viền, icon, hình ảnh, spacing nhỏ hoặc giao diện cần chính xác (không bỏ hoàn toàn px mà dùng đúng chỗ, ưu tiên rem/% cho các kích thước tuỳ biến hoặc font-size).​

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?

• \_\_\_\_\_\_\_\_\_\_\_

Hình ảnh minh họa

Ảnh có chứa văn bản, máy tính, ảnh chụp màn hình, bảng trắng

Nội dung do AI tạo ra có thể không chính xác.

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!"?

• \_\_\_\_\_\_\_\_\_\_\_

Video co dãn



3. 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?")\*

Phiếu 19

PHIẾU HỌC TẬP CSS [19] - GIẢ-PHẦN TỬ (::before & ::after)

+Giả-phần 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 Giả-phần tử | Cú pháp (Ví dụ) | Ý nghĩa (Nó chèn nội dung vào đâu?) |
| --- | --- | --- |
| ::before | p::before { ... } | Chèn nội dung ảo vào bên trong, ngay đầu thẻ p |
| ::after | p::after { ... } | Chèn nội dung ảo vào bên trong, ngay cuối thẻ p |

Câu hỏi bắt buộc:

Thuộc tính nào là TUYỆT ĐỐI BẮT BUỘC phải có (kể cả khi giá trị của nó là rỗng) để ::before 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

Ví dụ thực hành (CSS):

Giả sử HTML có đoạn sau:

xml

<h1 class="tieu-de">Đây là Tiêu đề Trang trí</h1>

<a href="#" class="link-qtrong">đường link quan trọng</a>

CSS cần dùng:

css

.tieu-de {

position: relative;

display: inline-block; /\* Để trang trí gạch dưới chính xác \*/

}

.tieu-de::after {

content: "";

display: block;

width: 60px;

height: 3px;

background: #26a7de;

margin: 6px auto 0 auto;

border-radius: 2px;

}

.link-qtrong::before {

content: "🔗 "; /\* hoặc dùng background-image/icon font nếu muốn đẹp hơn \*/

font-size: 1em;

vertical-align: middle;

}

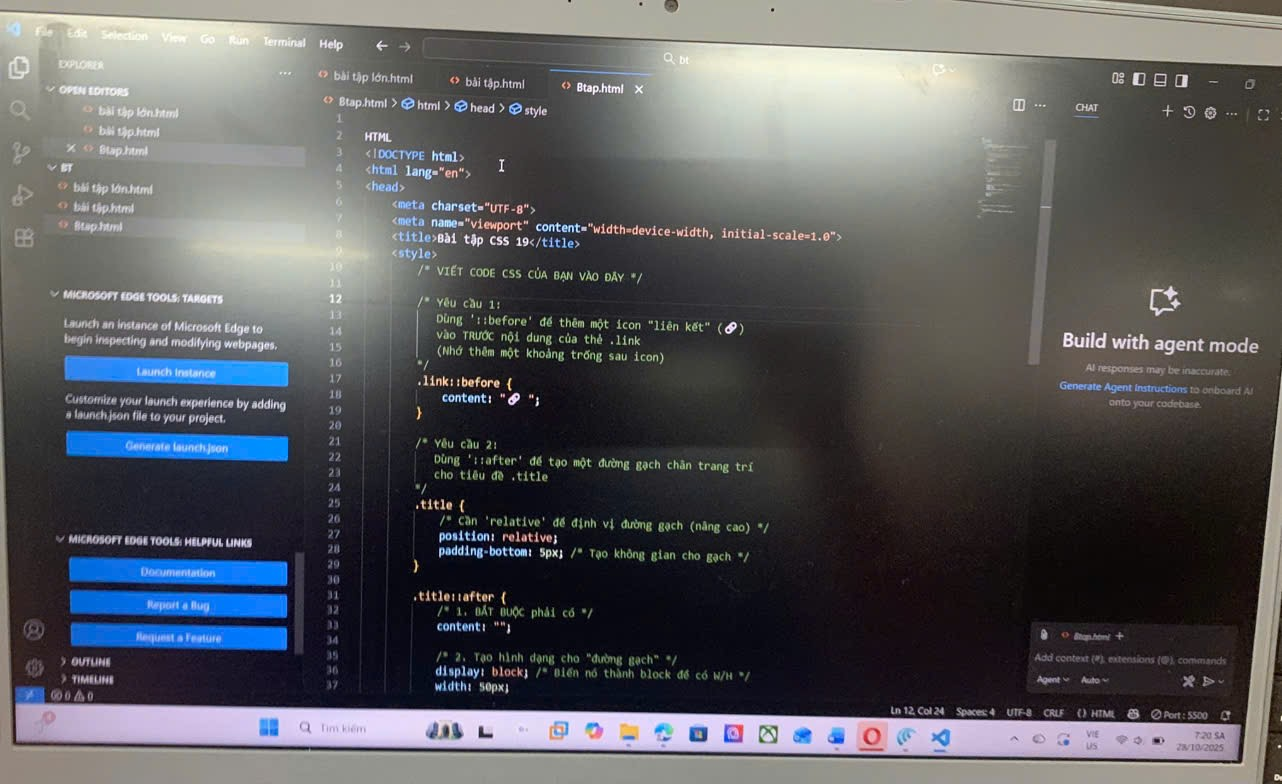
Kết quả mong muốn:

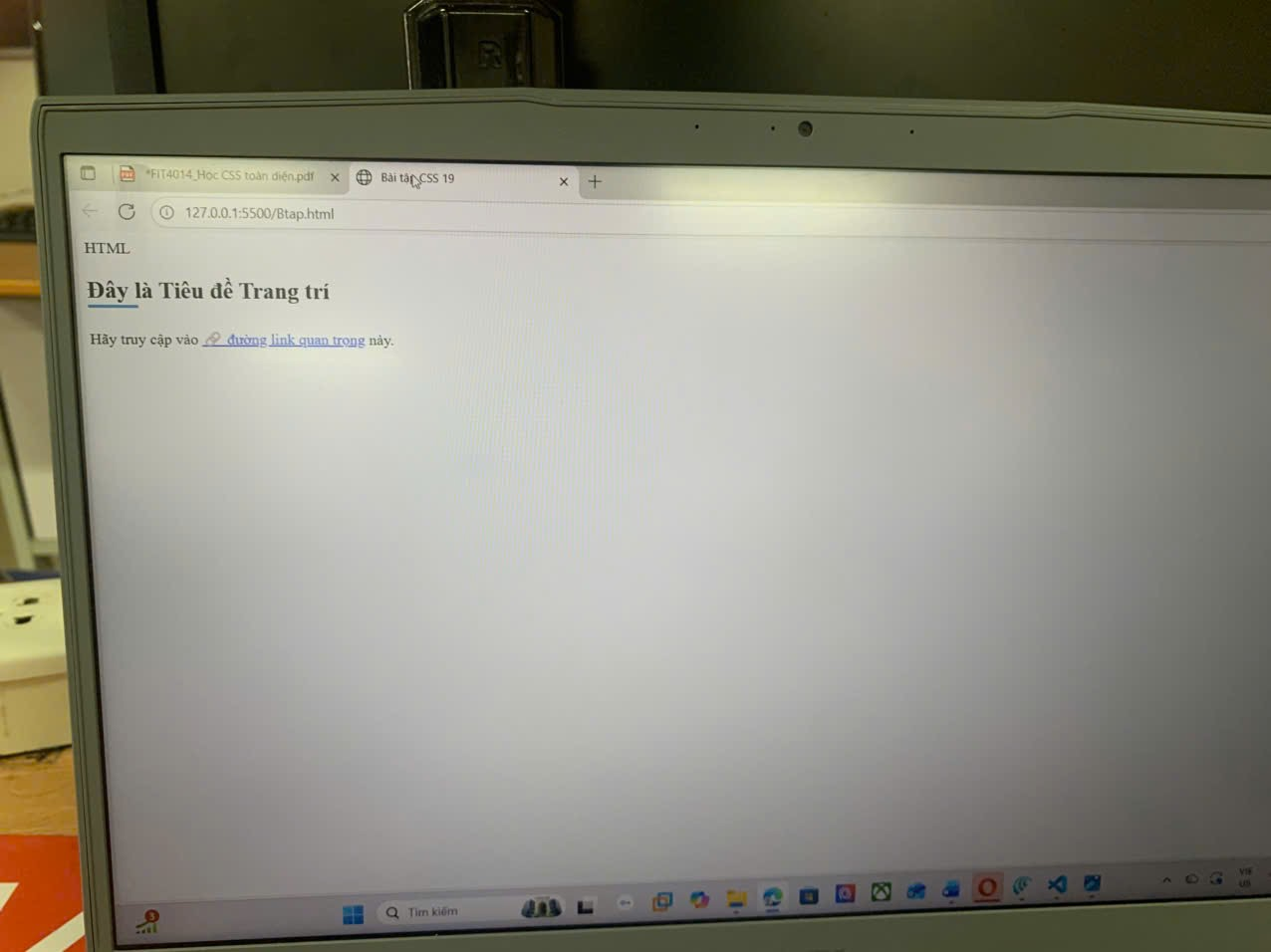
Tiêu đề có một đường gạch chân ngắn màu xanh bên dưới.

Chữ "đường link quan trọng" có một icon nằm ngay trước nó.

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

Hình minh họa code và chạy





: (Dán ảnh chụp màn hình trình duyệt của bạn

vào đây. Kết quả phải là:

• Tiêu đề <h2> có một đường gạch chân ngắn màu xanh bên dưới.

• Chữ "đường link quan trọng" có một icon nằm ngay trước nó. )

Phiếu 20

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

+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() | Dùng để di chuyển phần tử trên trục X hoặc Y | translate(50px, 10px) |
| rotate() | Dùng để xoay phần tử quanh một tâm | rotate(45deg) |
| scale() | Dùng để phóng to/thu nhỏ kích thước phần tử | scale(1.5) |
| skew() | Dùng để làm méo/nghiêng phần tử theo hướng X/Y | skew(10deg) |

Câu hỏi bắt buộc

Sự khác biệt lớn nhất giữa

transform: translate(50px); và

margin-left: 50px;

Trả lời:

Sử dụng transform: translate(50px); chỉ di chuyển hình dạng trực quan của phần tử đó, không làm dịch chuyển vị trí thực của nó trong luồng bố cục của trang web. Các phần tử xung quanh vẫn coi nó nằm ở vị trí cũ.

Trong khi đó, margin-left: 50px; sẽ đẩy vị trí thực sự của phần tử, làm thay đổi vị trí của các phần tử xung quanh trong bố cục.

Ví dụ thực hành (CSS):

Giả sử HTML của bạn:

xml

<div class="box box1">Translate</div>

<div class="box box2">Rotate</div>

<div class="box box3">Scale</div>

CSS mẫu:

css

.box {

width: 120px;

height: 60px;

background: #cec;

border-radius: 8px;

margin: 20px;

display: inline-block;

line-height: 60px;

font-weight: bold;

font-size: 1.2em;

text-align: center;

transition: transform 0.5s; /\* Làm hiệu ứng mượt mà \*/

cursor: pointer;

}

.box1:hover {

transform: translate(30px, 0);

}

.box2:hover {

transform: rotate(15deg);

}

.box3:hover {

transform: scale(1.3);

}

Hình ảnh code

Ảnh có chứa văn bản, màn hình, Thiết bị hiển thị, đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.

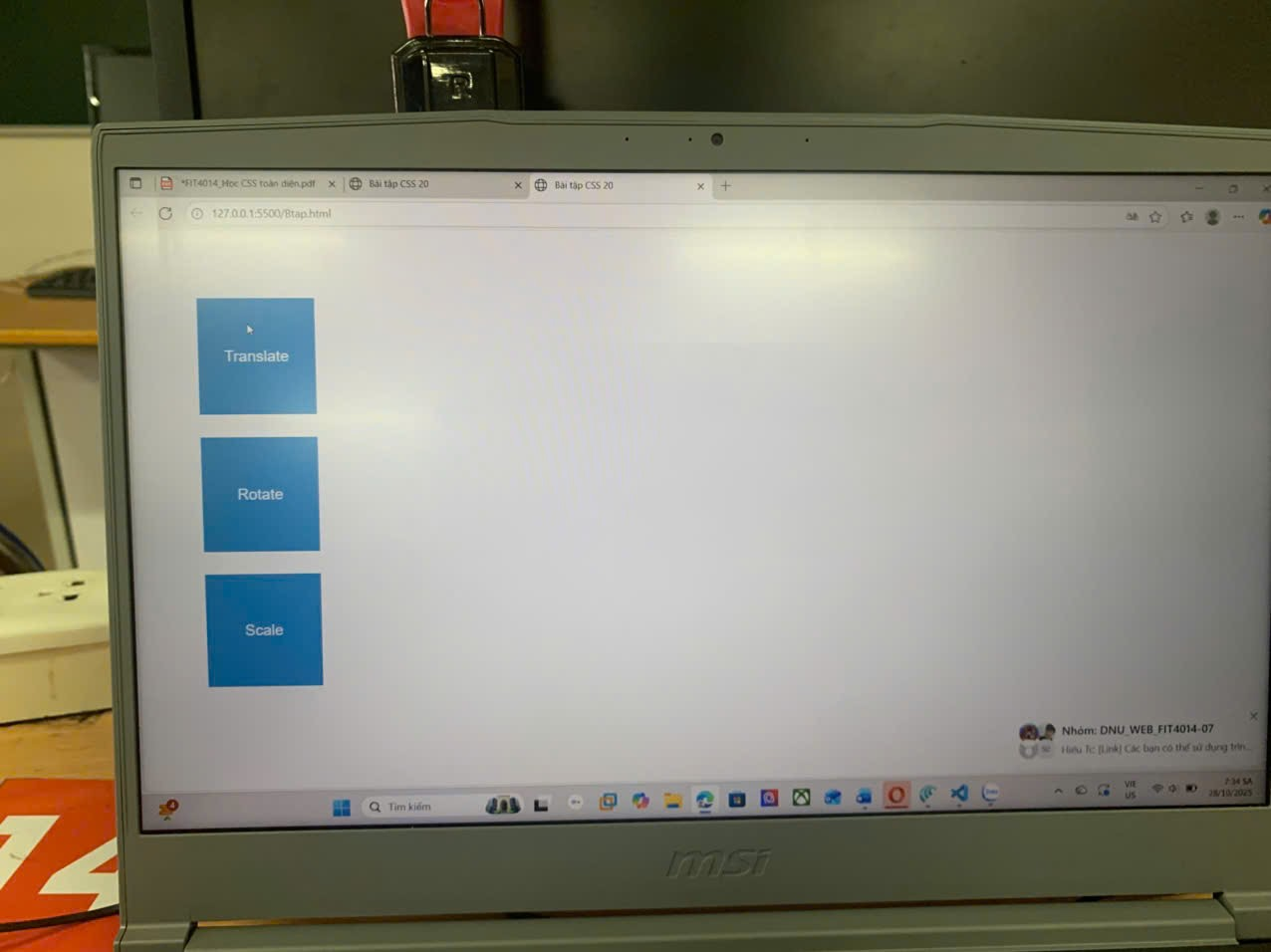
Kết quả (Chụp 3 ảnh màn hình):

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

đang hơ chuột vào Hộp 2, và 1 ảnh khi đang hơ chuột vào Hộp 3.

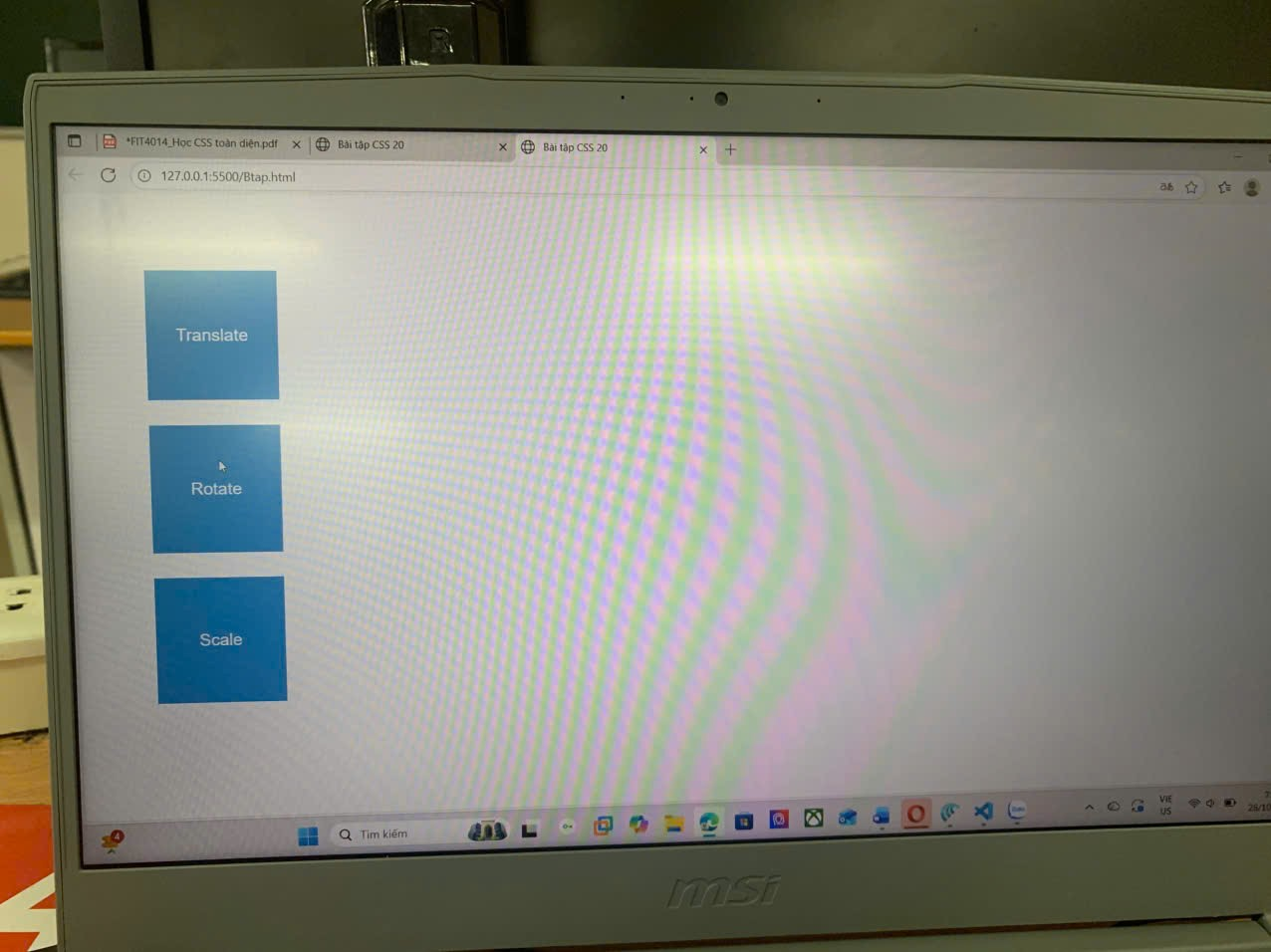
Ảnh 1: Hơ chuột vào Hộp 1

(Dán ảnh chụp màn hình của bạn vào đây)



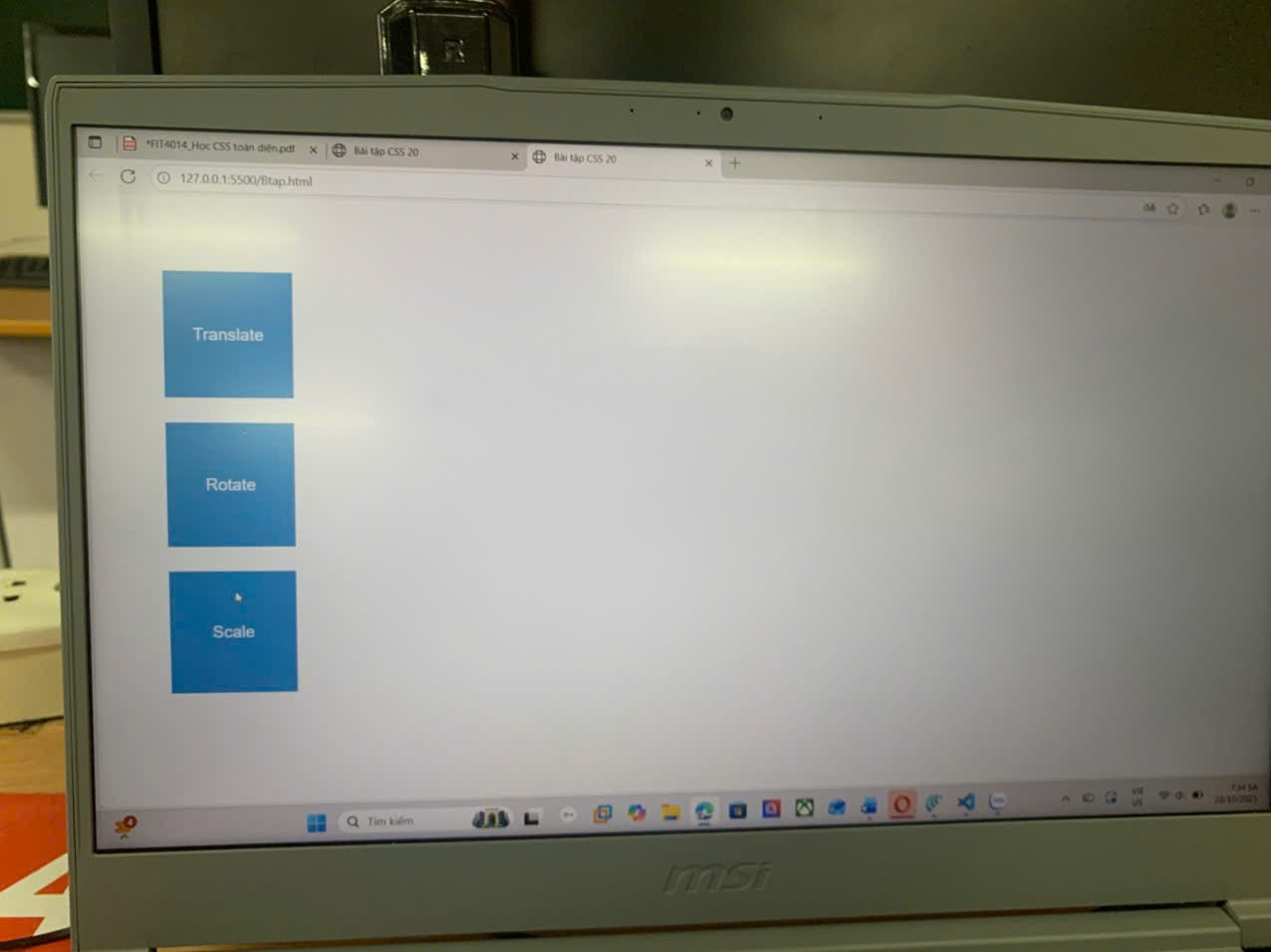
Ảnh 2: Hơ chuột vào Hộp 2

(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)



Câu hỏi thêm:

Ví dụ bạn hỏi: "Làm sao để box biến đổi MƯỢT MÀ?"

► Trả lời:

Để chuyển động mượt, hãy dùng CSS transition cho thuộc tính transform, ví dụ:

css

transition: transform 0.5s;

Thêm vào class chung hoặc từng box, hiệu ứng sẽ diễn ra trong 0.5 giây khi hover, không bị giật.

Phiếu 21

PHIẾU HỌC TẬP CSS [21] - CHUYỂN TIẾP MƯỢT MÀ (TRANSITIONS)

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 | Chỉ định thuộc tính nào được làm mượt | transform, background-color, all |
| transition-duration | Chỉ định thời gian chuyển tiếp | 0.5s (nửa giây), 300ms |
| transition-timing-function | Chỉ định "đường cong tốc độ" (cách diễn tiến) | linear, ease, ease-in-out |
| transition-delay | Chỉ định thời gian TRÌ HOÃN bắt đầu hiệu ứng | 1s (chờ 1 giây rồi mới chạ |

Cú pháp viết tắt (Shorthand - hay dùng nhất):

css

transition: all 0.3s ease;

Thứ tự: [property] [duration] [timing-function] [delay]

Ví dụ thực hành (thêm transition - CSS):

css

.box {

/\* Các style như trong Bài 20 \*/

width: 120px;

height: 60px;

background: #cec;

border-radius: 8px;

margin: 20px;

display: inline-block;

line-height: 60px;

font-weight: bold;

font-size: 1.2em;

text-align: center;

cursor: pointer;

/\* Thêm duy nhất 1 dòng này để mọi hiệu ứng chuyển động đều mượt \*/

transition: all 0.5s ease;

}

.box1:hover { transform: translate(30px, 0); }

.box2:hover { transform: rotate(15deg); }

.box3:hover { transform: scale(1.3); }

Mô tả kết quả:

Khi hơ chuột vào Hộp 1: Hộp di chuyển sang phải một cách mượt mà, không còn "giật" như Bài 20 nữa.

Khi bỏ chuột ra khỏi hộp: Hộp mượt mà quay về vị trí ban đầu, không bị giật hay chạy quá nhanh.

Mẹo bổ sung (Câu hỏi thêm được giải thích luôn):

Ví dụ nếu muốn một cái hộp tự xoay liên tục (không cần hover), hãy dùng CSS animation chứ không dùng transition. Ví dụ:

css

@keyframes spin {

from { transform: rotate(0deg); }

to { transform: rotate(360deg); }

}

.box-auto-spin {

animation: spin 2s linear infinite;

}

Dùng class box-auto-spin cho hộp cần xoay liên tục.

Kết quả (Video hoặc GIF):

+Code

<!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 {

}

</style>

</head>

<body>

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

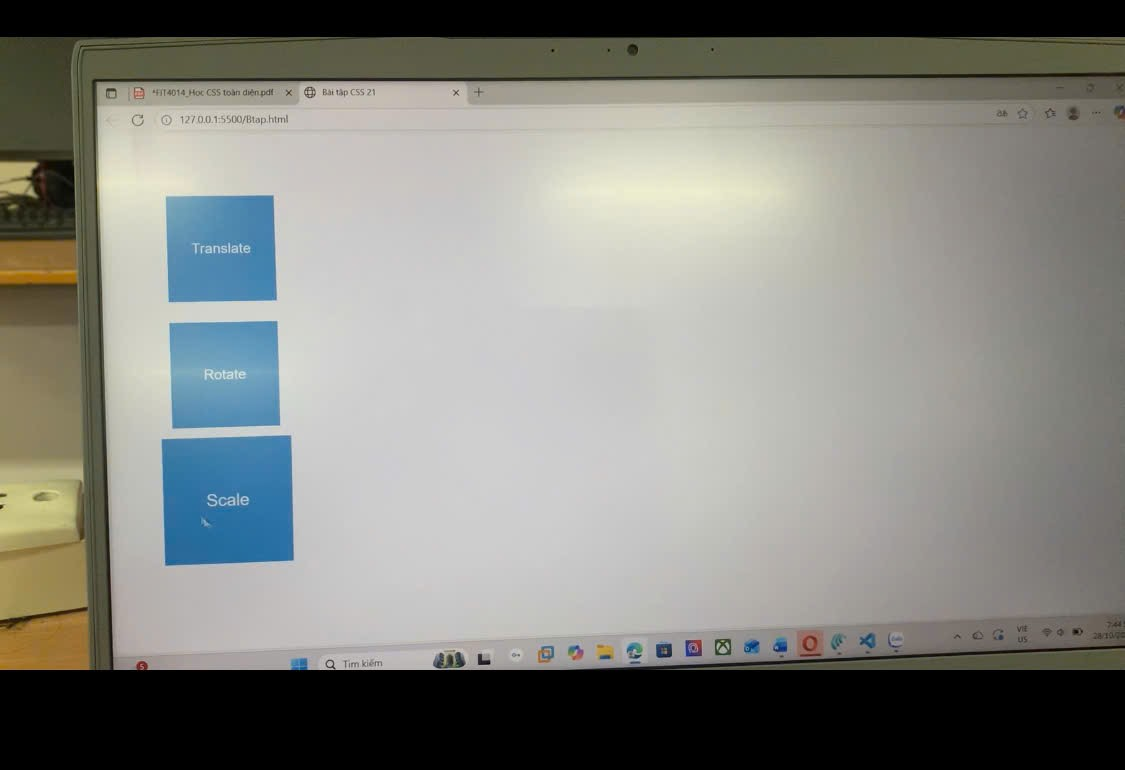
<div class="box box-2">Rotate</div>

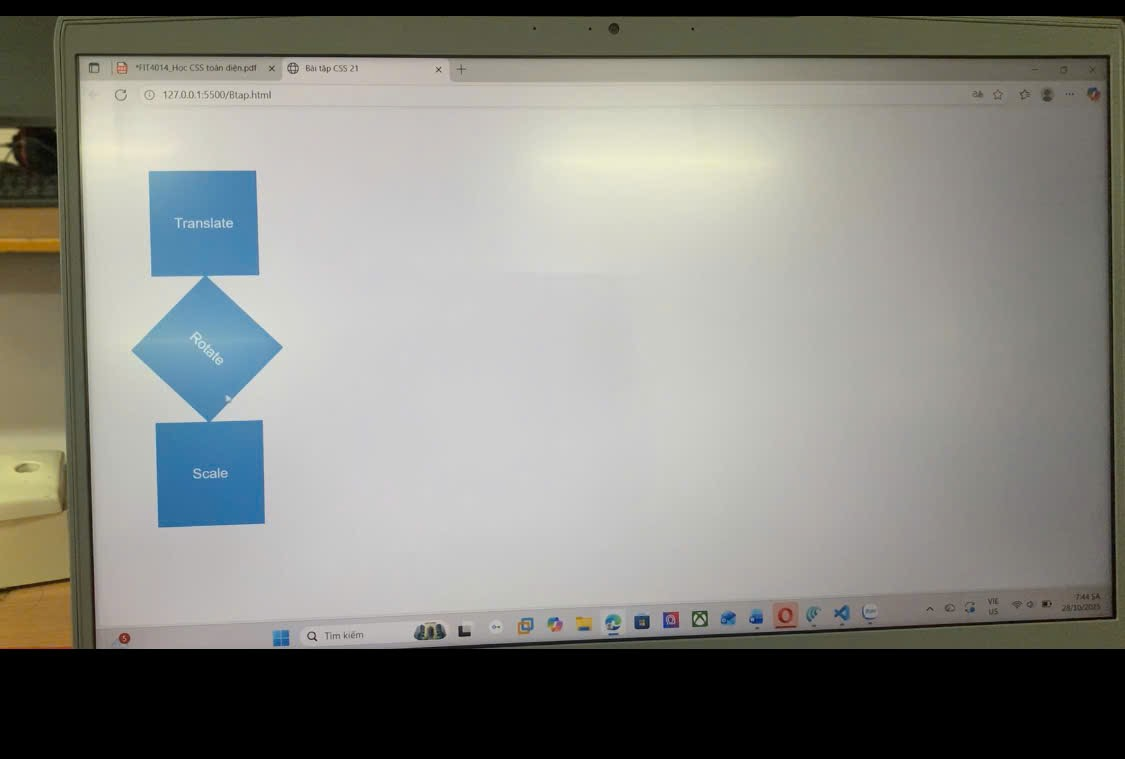
<div class="box box-3">Scale</div>

</body>

</html>

Ảnh





+ Video :



Hiệu ứng tĩnh không thể hiện được bài này. Hãy quay một video ngắn (hoặc ảnh GIF) ghi lại

cảnh bạn hơ 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 hơ chuột vào Hộp 1? Nó còn "giật" như Bài 20 không?

• \_không ,bài 20 không có hiệu ứng giật .giật\_\_\_\_\_\_\_\_\_\_

• Điều gì xảy ra khi bạn BỎ CHUỘT ra khỏi hộp? (Nó có mượt mà quay về không?)

• \_Có nó sẽ quay về vị trí bạn đầu của trang\_\_\_\_\_\_\_\_\_\_

+ 3 hộp có chuyển động như sau

- Hộp 1 sẽ chuyển động sang bên khi ta di chuột vào

- Hộp 2 sẽ xoay tròn khi ta di chuột vào

- Hộp 3 sẽ phình to khi ta di chuột vào

Phiếu 22

PHIẾU HỌC TẬP CSS [22] - HOẠT ẢNH VỚI @keyframes

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 | Tên của @keyframes cần gọi | spin, pulse |
| animation-duration | Thời gian để hoàn thành 1 vòng hoạt ảnh | 2s, 1.2s |
| animation-iteration-count | Số lần lặp lại hoạt ảnh | 3, infinite |
| animation-direction | Hướng chạy của hoạt ảnh (bình thường hoặc qua lại) | normal, alternate |
| animation-timing-function | "Đường cong tốc độ" hoạt ảnh | linear, ease |

Cú pháp viết tắt (shorthand animation):

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

Ví dụ:

animation: spin 2s linear infinite;

Ví dụ thực hành HTML & CSS:

Hình ảnh code

Ảnh có chứa văn bản, Thiết bị hiển thị, đa phương tiện, màn hình

Nội dung do AI tạo ra có thể không chính xác.

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)?

• nó vẫn sẽ chuyện động

Phiếu 23

PHIẾU HỌC TẬP CSS [23] - BIẾN TRONG CSS (CSS VARIABLES)

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

Giả sử dự án của bạn dùng một "màu chủ đạo" (ví dụ: màu xanh) ở 50 nơi khác nhau. Khi sếp

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:

+code

<!DOCTYPE html>

<html>

<head>

  <title>Bài tập CSS 23</title>

  <style>

    :root {

      --main-color: darkblue;

      --padding-size: 16px;

    }

    h1 {

      color: var(--main-color);

      padding-bottom: var(--padding-size);

    }

    button {

      background: var(--main-color);

      color: #fff;

      border: none;

      padding: var(--padding-size) 32px;

      border-radius: 6px;

      font-size: 1.1em;

    }

    .box {

      border: 3px solid var(--main-color);

      padding: var(--padding-size);

      margin-top: 18px;

      border-radius: 10px;

      font-size: 1.1em;

    }

  </style>

</head>

<body>

  <h1>Tiêu đề này dùng Biến CSS</h1>

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

  <div class="box">Cái hộp này cũng dùng Biến CSS</div>

</body>

</html>

+ Hình ảnh của code

Ảnh có chứa văn bản, máy tính, Thiết bị hiển thị, màn hình

Nội dung do AI tạo ra có thể không chính xác.

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 có chứa văn bản, ảnh chụp màn hình, hàng

Nội dung do AI tạo ra có thể không chính xác.

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?

• sẽ làm thay đổi

• Kích thước padding của Nút và Hộp thay đổi như thế nào?

• \_to hơn và khác nhau về kích thước



Phiếu 24

PHIẾU HỌC TẬP [24] - DỰ ÁN CUỐI KHÓA: XÂY DỰNG TRANG PORTFOLIO CÁ NHÂN

Họ và tên: nguyễn văn trọng

Lớp CNTT19-07

Link nộp bài (GitHub/CodePen):

https://github.com/trong456/2025\_FIT4014\_ThietkeWeb

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):

1. Header & Navbar: Logo (tên của bạn) và các link điều hướng (Giới thiệu, Dự án,

Liên hệ...).

2. 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").

3. Dự án (Projects): Hiển thị ít nhất 3 dự án bạn đã làm (có thể là bài tập lớn, ảnh

demo...).

4. Kỹ năng (Skills): Liệt kê các kỹ năng (ví dụ: HTML, CSS, JavaScript...).

5. 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):

o Phải sử dụng Flexbox cho ít nhất 1 khu vực (ví dụ: căn chỉnh navbar, căn giữa

nội dung hero). (Bài 13)

o Phải sử dụng Grid cho ít nhất 1 khu vực (ví dụ: chia lưới cho khu vực "Dự án").

(Bài 14)

• [ ] Hiệu ứng:

o Phải sử dụng Transition (Bài 21) cho các hiệu ứng :hover (ví dụ: khi di chuột vào

nút bấm, link, hoặc thẻ dự án).

o (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 (<img>) 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):

o Tạo file style.css và liên kết vào HTML.

o Thiết lập box-sizing: border-box;.

o Định nghĩa các Biến CSS (:root) cho màu sắc.

o Nhúng Google Font và setup body.

3. Bước 3 (Mobile Layout - Dưới 768px):

o Bắt đầu style cho màn hình điện thoại (1 cột).

o Style cho header, navbar (có thể dùng Flexbox cho navbar ngay).

o Style cho từng section (chúng sẽ tự động xếp chồng lên nhau).

4. Bước 4 (Desktop Layout - Trên 768px):

o Mở một @media (min-width: 768px) { ... }.

o Bên trong media query này, viết các quy tắc để thay đổi layout.

o 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):

o Quay lại CSS gốc (ngoài media query).

o Thêm transition: all 0.3s ease; cho các nút bấm, các thẻ dự án.

o Viết các quy tắc :hover (ví dụ: transform: scale(1.05); cho thẻ dự án) để tạo hiệu

ứng.

6. Bước 6 (Kiểm tra):

o Dùng F12 (Developer Tools) để kiểm tra trên nhiều kích thước màn hình.

o 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!