Quét để xem sơ đồ trực tuyến

FRONTEND

Lộ trình CSS hiện đại: Từ nền tảng đến giao diện thực chiến

Học CSS từ số 0 đến làm chủ giao diện sản phẩm: selectors, cascade, box model, typography, Flexbox, Grid, responsive, container queries, CSS nesting, cascade layers, animation, accessibility, hiệu năng và workflow làm việc với AI. Mỗi bài có ví dụ, bài tập, checklist và nguồn tham khảo.

Cơ bản · 10 giai đoạn · 50 chủ đề · Cập nhật 04/11/2026

Tài liệu nguồn chính: https://roadmap.sh/css

MỤC TIÊU

- Giải thích được CSS cascade, specificity, box model và tự debug bằng DevTools
- Dựng layout responsive với Flexbox, Grid, fluid type và container queries
- Tổ chức CSS bằng design tokens, nesting, @layer và cấu trúc dễ bảo trì
- Thiết kế trạng thái focus, form, motion và màu sắc có cân nhắc accessibility
- Kiểm tra tương thích trình duyệt, hiệu năng, lỗi responsive và visual regression
- Hoàn thành landing page, dashboard, component library và capstone có bằng chứng QA
- Dùng AI để giải thích/gợi ý/review nhưng tự kiểm chứng CSS và chịu trách nhiệm sản phẩm

CÁC GIAI ĐOẠN

Giai đoạn 1: 01 · Nhập môn CSS và công cụ
Hiểu CSS, cách trình duyệt áp dụng style và thiết lập môi trường thử nghiệm an toàn. (5 chủ đề)

Giai đoạn 2: 02 · Selectors, Cascade và Specificity
Chọn phần tử đúng và giải thích vì sao một declaration thắng declaration khác. (5 chủ đề)

Giai đoạn 3: 03 · Box model, kích thước và trình bày
Kiểm soát box sizing, đơn vị, kiểu chữ, màu sắc và media mà không gây tràn. (5 chủ đề)

Giai đoạn 4: 04 · Normal Flow, Display và Position
Hiểu mô hình bố cục của trình duyệt trước khi dùng Flexbox hay Grid. (5 chủ đề)

Giai đoạn 5: 05 · Flexbox và CSS Grid thực chiến
Làm chủ layout một chiều và hai chiều, căn chỉnh, wrap và responsive không hack. (5 chủ đề)

Giai đoạn 6: 06 · Responsive và CSS thích nghi theo component
Thiết kế mobile-first, fluid type, container queries và hỗ trợ mới theo điều kiện. (5 chủ đề)

Giai đoạn 7: 07 · CSS hiện đại và tổ chức code

Làm chủ selectors mới, nesting, cascade layers, custom properties và cấu trúc dễ bảo trì. (5 chủ đề)

Giai đoạn 8: 08 · Hiệu ứng, trạng thái và trải nghiệm sử dụng

Tạo UI chuyển động vừa đủ, biểu mẫu dùng được và các thành phần tương tác dễ tiếp cận. (5 chủ đề)

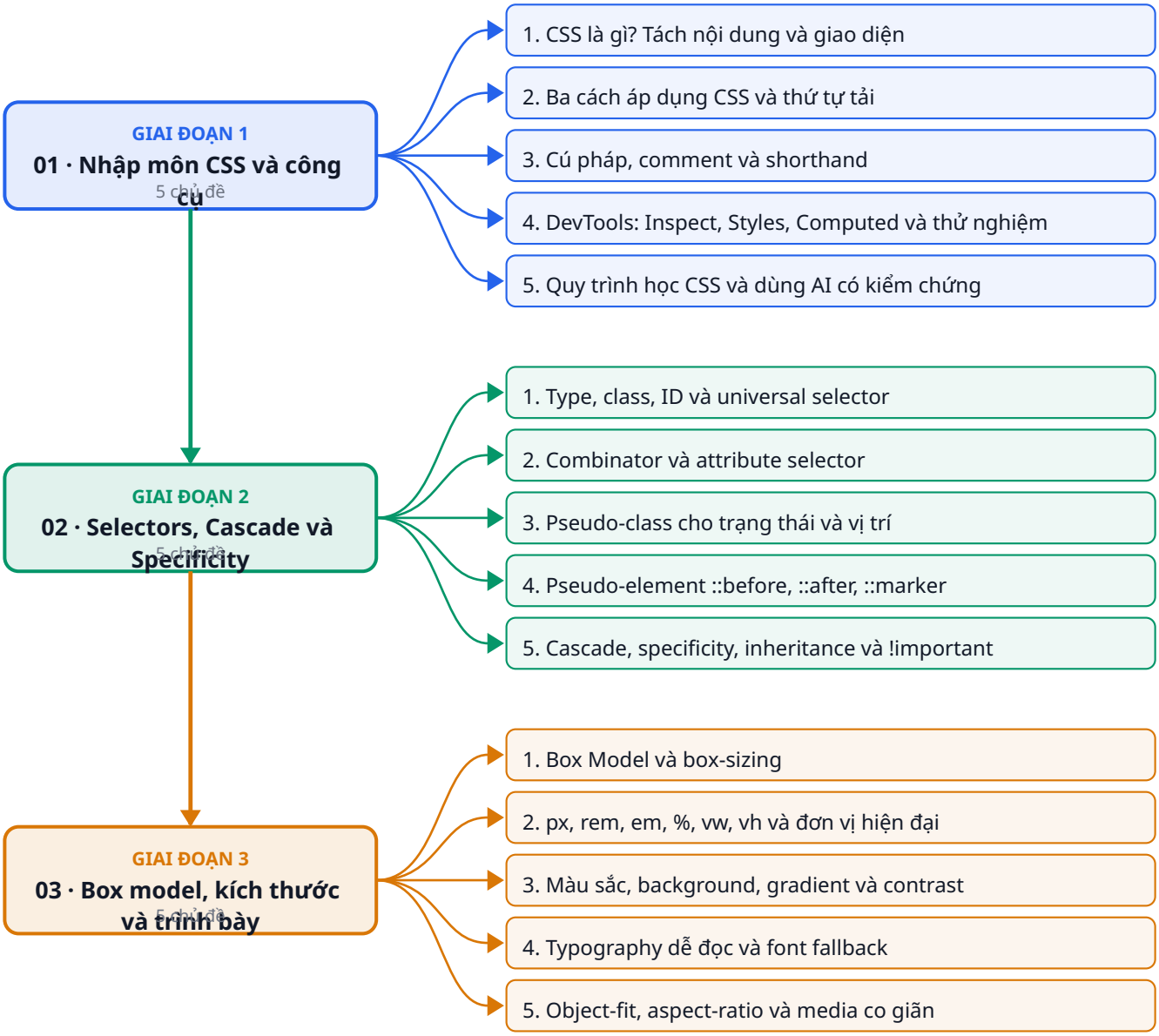
Giai đoạn 9: 09 · Accessibility, hiệu năng và CSS production

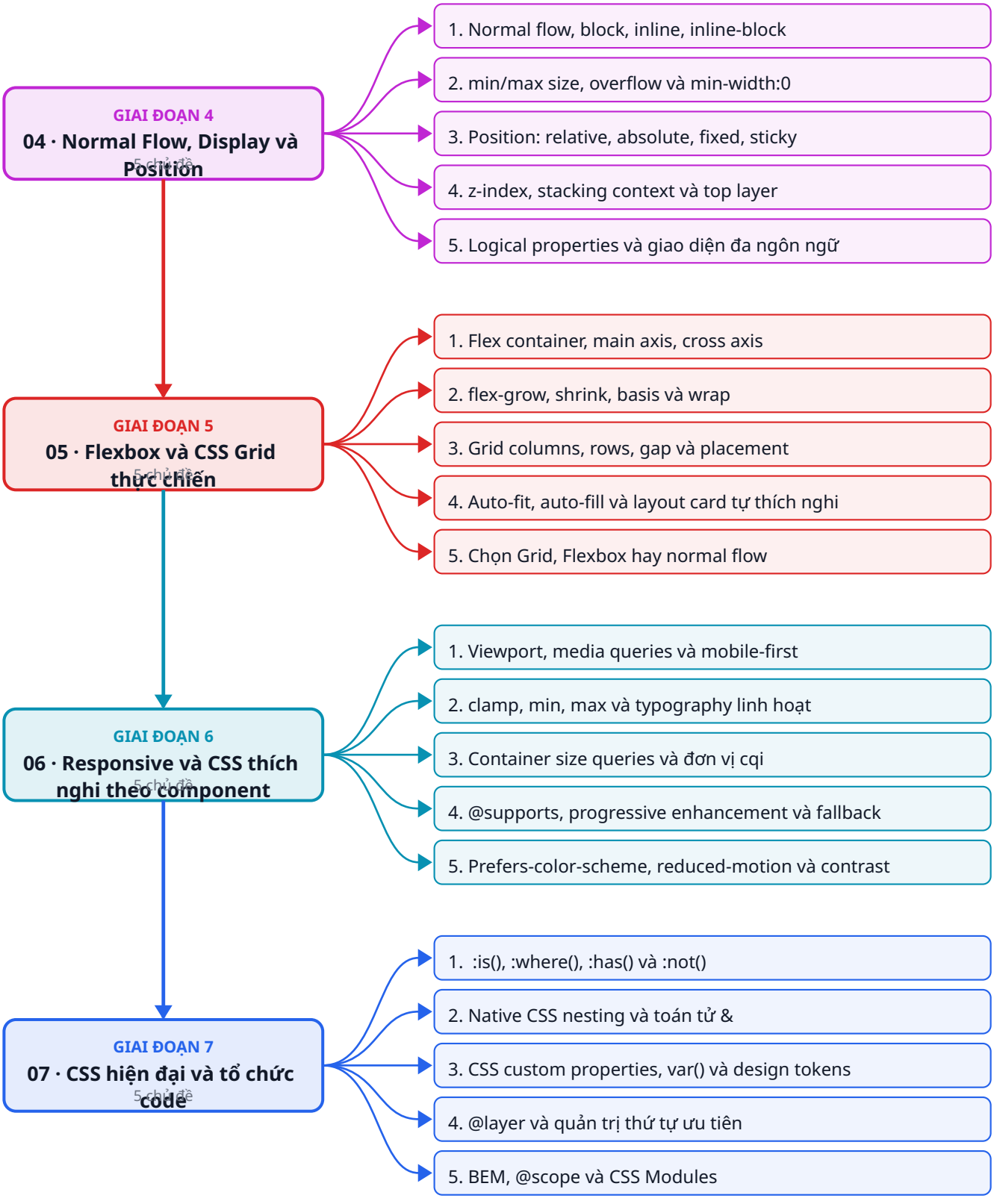
Đánh giá chất lượng trước khi giao diện đi vào sản phẩm thật. (5 chủ đề)

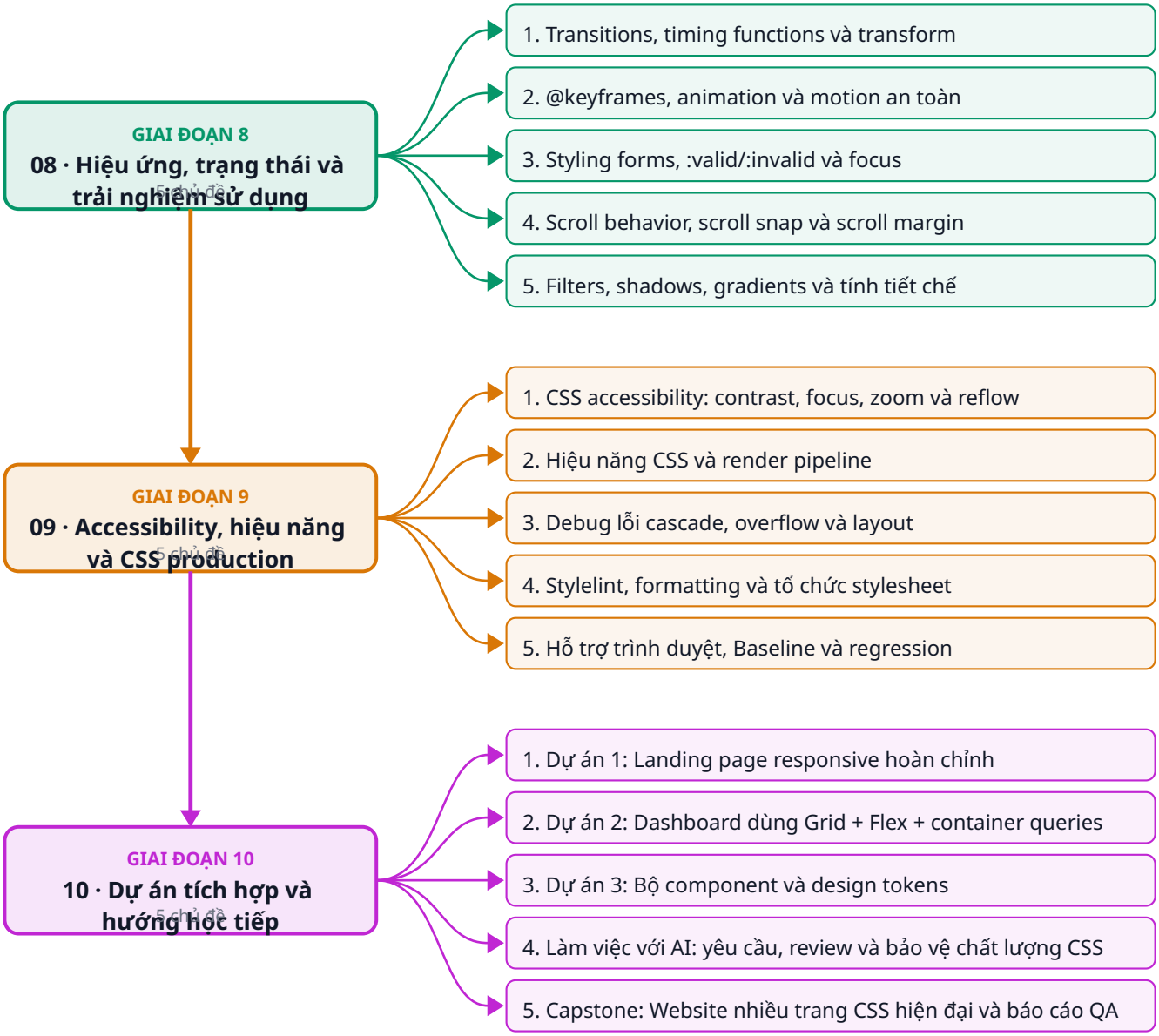
Giai đoạn 10: 10 · Dự án tích hợp và hướng học tiếp

Áp dụng kiến thức vào sản phẩm kiểm thử được, sử dụng AI mà không đánh mất năng lực CSS nền tảng. (5 chủ đề)

SƠ ĐỒ LỘ TRÌNH




GIAI ĐOẠN 4
04 · Normal Flow, Display và Position
5 chủ đề
1. Normal flow, block, inline, inline-block
2. min/max size, overflow và min-width:0
3. Position: relative, absolute, fixed, sticky
4. z-index, stacking context và top layer
5. Logical properties và giao diện đa ngôn ngữ
GIAI ĐOẠN 5
05 · Flexbox và CSS Grid thực chiến
5 chủ đề
1. Flex container, main axis, cross axis
2. flex-grow, shrink, basis và wrap
3. Grid columns, rows, gap và placement
4. Auto-fit, auto-fill và layout card tự thích nghi
5. Chọn Grid, Flexbox hay normal flow
GIAI ĐOẠN 6
06 · Responsive và CSS thích nghi theo component
5 chủ đề
1. Viewport, media queries và mobile-first
2. clamp, min, max và typography linh hoạt
3. Container size queries và đơn vị cqi
4. @supports, progressive enhancement và fallback
5. Prefers-color-scheme, reduced-motion và contrast
GIAI ĐOẠN 7
07 · CSS hiện đại và tổ chức code
5 chủ đề
1. :is(), :where(), :has() và :not()
2. Native CSS nesting và toán tử &
3. CSS custom properties, var() và design tokens
4. @layer và quản trị thứ tự ưu tiên
5. BEM, @scope và CSS Modules


GIAI ĐOẠN 8
08 · Hiệu ứng, trạng thái và trải nghiệm sử dụng
5 chủ đề
1. Transitions, timing functions và transform
2. @keyframes, animation và motion an toàn
3. Styling forms, :valid/:invalid và focus
4. Scroll behavior, scroll snap và scroll margin
5. Filters, shadows, gradients và tính tiết chế
GIAI ĐOẠN 9
09 · Accessibility, hiệu năng và CSS production
5 chủ đề
1. CSS accessibility: contrast, focus, zoom và reflow
2. Hiệu năng CSS và render pipeline
3. Debug lỗi cascade, overflow và layout
4. Stylelint, formatting và tổ chức stylesheet
5. Hỗ trợ trình duyệt, Baseline và regression
GIAI ĐOẠN 10
10 · Dự án tích hợp và hướng học tiếp
5 chủ đề
1. Dự án 1: Landing page responsive hoàn chỉnh
2. Dự án 2: Dashboard dùng Grid + Flex + container queries
3. Dự án 3: Bộ component và design tokens
4. Làm việc với AI: yêu cầu, review và bảo vệ chất lượng CSS
5. Capstone: Website nhiều trang CSS hiện đại và báo cáo QA

GIAI ĐOẠN 1

01 · Nhập môn CSS và công cụ

Hiểu CSS, cách trình duyệt áp dụng style và thiết lập môi trường thử nghiệm an toàn.

1. CSS là gì? Tách nội dung và giao diện

CSS mô tả cách trình bày của HTML, không thay thế ngữ nghĩa HTML.

Mục tiêu học tập

- Giải thích được Stylesheet và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "CSS là gì? Tách nội dung và giao diện" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Stylesheet · Rule · Declaration · Selector · Property

Giải thích chi tiết

Vì sao cần học?

CSS mô tả cách trình bày của HTML, không thay thế ngữ nghĩa HTML. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- CSS mô tả cách trình bày của HTML, không thay thế ngữ nghĩa HTML.
- Một stylesheet chứa các quy tắc selector { property: value; }.
- Học CSS qua quan sát kết quả và computed styles, không học thuộc danh sách thuộc tính.

Ví dụ để thử trên trang HTML

```
.notice { color: #164e63; background: #ecfeff; padding: 1rem; }
```

Thực hành từng bước

1. Tạo HTML có heading, paragraph và aside.
2. Viết CSS đổi màu, nền và khoảng cách.
3. Tắt stylesheet để chứng minh nội dung vẫn đọc được.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “CSS là gì? Tách nội dung và giao diện”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Nhầm CSS với logic tương tác hoặc dùng CSS để che nội dung quan trọng.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ CSS là gì? Tách nội dung và giao diện

```
.notice { color: #164e63; background: #ecfeff; padding: 1rem; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo HTML có heading, paragraph và aside.
[ ] Đã làm: Viết CSS đổi màu, nền và khoảng cách.
[ ] Đã làm: Tắt stylesheet để chứng minh nội dung vẫn đọc được.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: CSS mô tả cách trình bày của HTML, không thay thế ngữ nghĩa HTML.

Tài liệu tham khảo

Tài liệu MDN — CSS là gì? Tách nội dung và giao diện Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/What_is_CSS

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

---

2. Ba cách áp dụng CSS và thứ tự tải

External CSS dễ tái sử dụng và cache; internal style phù hợp demo nhỏ; inline style khó bảo trì.

Mục tiêu học tập

- Giải thích được External CSS và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Ba cách áp dụng CSS và thứ tự tải” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

External CSS · Internal CSS · Inline Styles · Link

Giải thích chi tiết

Vì sao cần học?

External CSS dễ tái sử dụng và cache; internal style phù hợp demo nhỏ; inline style khó bảo trì. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- External CSS dễ tái sử dụng và cache; internal style phù hợp demo nhỏ; inline style khó bảo trì.
- Thẻ link rel=stylesheet trong head tải CSS ngoài.
- CSS bị lỗi cú pháp có thể làm một phần declaration không áp dụng.

Ví dụ để thử trên trang HTML

```
<link rel="stylesheet" href="styles.css">
<style>h1 { color: teal; }</style>
```

Thực hành từng bước

1. Tạo styles.css riêng và kết nối HTML.
2. Thử gỡ link và xem giao diện thay đổi.
3. Quan sát Network để biết stylesheet có tải thành công không.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Ba cách áp dụng CSS và thứ tự tải”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Đặt sai path, tưởng file đã tải hoặc dùng inline style tràn lan.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Ba cách áp dụng CSS và thứ tự tải

```
<link rel="stylesheet" href="styles.css">
<style>h1 { color: teal; }</style>
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Tạo styles.css riêng và kết nối HTML.
- [ ] Đã làm: Thử gỡ link và xem giao diện thay đổi.
- [ ] Đã làm: Quan sát Network để biết stylesheet có tải thành công không.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: External CSS dễ tái sử dụng và cache; internal style phù hợp demo nhỏ; inline style khó bảo trì.

Tài liệu tham khảo

Tài liệu MDN — Ba cách áp dụng CSS và thứ tự tải Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Getting_started

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

CSS là gì? Tách nội dung và giao diện

---

3. Cú pháp, comment và shorthand

Một declaration có property và value; comment dùng /* ... */.

Mục tiêu học tập

- Giải thích được Declaration và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Cú pháp, comment và shorthand” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Declaration · Shorthand · Longhand · Comments

Giải thích chi tiết

Vì sao cần học?

Một declaration có property và value; comment dùng / ... /. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Một declaration có property và value; comment dùng / ... /.
- Shorthand như margin có 1–4 giá trị theo thứ tự top right bottom left.
- Property không hợp lệ bị bỏ qua; CSS có cơ chế phục hồi lỗi.

Ví dụ để thử trên trang HTML

```
/* Card spacing */
.card { margin: 1rem 2rem; padding: 12px 16px; border: 1px solid #ddd; }
```

Thực hành từng bước

1. Viết 5 quy tắc và đặt tên class rõ nghĩa.
2. Đổi shorthand sang longhand tương đương.
3. Cố ý viết lỗi và đọc cảnh báo DevTools.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Cú pháp, comment và shorthand”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Quên dấu chấm trong class selector hoặc hiểu sai thứ tự shorthand.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Cú pháp, comment và shorthand


```
/* Card spacing */
.card { margin: 1rem 2rem; padding: 12px 16px; border: 1px solid #ddd; }
```


Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Viết 5 quy tắc và đặt tên class rõ nghĩa.

[ ] Đã làm: Đổi shorthand sang longhand tương đương.

[ ] Đã làm: Cố ý viết lỗi và đọc cảnh báo DevTools.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Một declaration có property và value; comment dùng / ... /.

Tài liệu tham khảo

Tài liệu MDN — Cú pháp, comment và shorthand Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_syntax

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Ba cách áp dụng CSS và thứ tự tải

---

4. DevTools: Inspect, Styles, Computed và thử nghiệm

Elements chọn DOM node; Styles hiển thị quy tắc cùng phần bị ghi đè.

Mục tiêu học tập

- Giải thích được DevTools và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "DevTools: Inspect, Styles, Computed và thử nghiệm" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

DevTools · Computed Style · Matched Rules · Box Model

Giải thích chi tiết

Vì sao cần học?

Elements chọn DOM node; Styles hiển thị quy tắc cùng phần bị ghi đè. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Elements chọn DOM node; Styles hiển thị quy tắc cùng phần bị ghi đè.
- Computed cho giá trị tính cuối cùng; Box Model minh họa kích thước.
- Có thể bật/tắt declaration trực tiếp để tìm nguyên nhân lỗi.

Ví dụ để thử trên trang HTML

```
.debug-me { width: 240px; padding: 16px; border: 2px solid; }
```

Thực hành từng bước

1. Inspect một thẻ và lần theo selector.
2. Chỉnh thử padding trong Styles, lưu lại file nguồn.
3. Chụp lại computed width và giải thích box sizing.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “DevTools: Inspect, Styles, Computed và thử nghiệm”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Sửa CSS trong DevTools rồi quên cập nhật file thật.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ DevTools: Inspect, Styles, Computed và thử nghiệm

```
.debug-me { width: 240px; padding: 16px; border: 2px solid; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Inspect một thẻ và lần theo selector.
[ ] Đã làm: Chỉnh thử padding trong Styles, lưu lại file nguồn.
[ ] Đã làm: Chụp lại computed width và giải thích box sizing.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: Elements chọn DOM node; Styles hiển thị quy tắc cùng phần bị ghi đè.

Tài liệu tham khảo

Tài liệu MDN — DevTools: Inspect, Styles, Computed và thử nghiệm Tài liệu
https://developer.chrome.com/docs/devtools/css/

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Cú pháp, comment và shorthand

5. Quy trình học CSS và dùng AI có kiểm chứng

Làm giao diện nhỏ, dự đoán style trước, mở trình duyệt đối chiếu.

Mục tiêu học tập

- Giải thích được Experiment và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Quy trình học CSS và dùng AI có kiểm chứng" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Experiment · Verification · Iteration · Browser Testing

Giải thích chi tiết

Vì sao cần học?

Làm giao diện nhỏ, dự đoán style trước, mở trình duyệt đối chiếu. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Làm giao diện nhỏ, dự đoán style trước, mở trình duyệt đối chiếu.
- Dùng AI như người giải thích và reviewer, không như nguồn sự thật duy nhất.
- Đối chiếu MDN và kiểm tra giao diện ở nhiều viewport, zoom, bàn phím.

Ví dụ để thử trên trang HTML

```
.exercise { outline: 2px dashed tomato; }
```

Thực hành từng bước

1. Thiết lập thư mục index.html và styles.css.
2. Ghi nhật ký 5 lỗi CSS và cách tìm trong DevTools.
3. Nhờ AI giải thích lỗi nhưng tự xác minh kết quả.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Quy trình học CSS và dùng AI có kiểm chứng”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Copy nguyên CSS AI sinh và không hiểu cascade hay ảnh hưởng responsive.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Quy trình học CSS và dùng AI có kiểm chứng

```
.exercise { outline: 2px dashed tomato; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Thiết lập thư mục index.html và styles.css.

[ ] Đã làm: Ghi nhật ký 5 lỗi CSS và cách tìm trong DevTools.

[ ] Đã làm: Nhờ AI giải thích lỗi nhưng tự xác minh kết quả.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Làm giao diện nhỏ, dự đoán style trước, mở trình duyệt đối chiếu.

Tài liệu tham khảo

Tài liệu MDN — Quy trình học CSS và dùng AI có kiểm chứng Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

DevTools: Inspect, Styles, Computed và thử nghiệm

GIAI ĐOẠN 2

02 · Selectors, Cascade và Specificity

Chọn phần tử đúng và giải thích vì sao một declaration thắng declaration khác.

1. Type, class, ID và universal selector

Selector theo thẻ, .class, #id và * có phạm vi khác nhau.

Mục tiêu học tập

- Giải thích được Type selector và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Type, class, ID và universal selector" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Type selector · Class selector · ID selector · Universal selector

Giải thích chi tiết

Vì sao cần học?

Selector theo thẻ, .class, #id và * có phạm vi khác nhau. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Selector theo thẻ, .class, #id và * có phạm vi khác nhau.
- Ưu tiên class cho styling có thể tái sử dụng.
- Selector theo ID có specificity cao nên khó override.

Ví dụ để thử trên trang HTML

```
p { line-height: 1.6; }
.card { border-radius: 12px; }
#main-title { color: navy; }
```

Thực hành từng bước

1. Gán class cho ba card và áp dụng chung.
2. Dùng DevTools so sánh selector theo thẻ và class.
3. Refactor selector theo ID sang class.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Type, class, ID và universal selector”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng ID để style mọi thành phần hoặc thêm nhiều selector dư thừa.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Type, class, ID và universal selector

```
p { line-height: 1.6; }
.card { border-radius: 12px; }
#main-title { color: navy; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Gán class cho ba card và áp dụng chung.
[ ] Đã làm: Dùng DevTools so sánh selector theo thẻ và class.
[ ] Đã làm: Refactor selector theo ID sang class.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: Selector theo thẻ, .class, #id và * có phạm vi khác nhau.

Tài liệu tham khảo

Tài liệu MDN — Type, class, ID và universal selector Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_selectors

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Quy trình học CSS và dùng AI có kiểm chứng

---

2. Combinator và attribute selector

Dấu cách chọn hậu duệ; > chọn con trực tiếp; + anh em liền kề; ~ anh em sau.

Mục tiêu học tập

- Giải thích được Descendant và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Combinator và attribute selector” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng

- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Descendant · Child combinator · Sibling combinator · Attribute selector

Giải thích chi tiết

Vì sao cần học?

Dấu cách chọn hậu duệ; > chọn con trực tiếp; + anh em liền kề; ~ anh em sau. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Dấu cách chọn hậu duệ; > chọn con trực tiếp; + anh em liền kề; ~ anh em sau.
- [type=email] và [data-state=open] chọn theo thuộc tính.
- Selector có thể quá rộng gây hiệu ứng ngoài dự kiến.

Ví dụ để thử trên trang HTML

```
.menu > li { list-style: none; }
a[target="_blank"] { text-decoration-style: dotted; }
```

Thực hành từng bước

1. Viết menu lồng hai cấp và thử > so với dấu cách.
2. Chỉ style link mở tab mới bằng attribute selector.
3. Kiểm tra DOM có bị chọn ngoài ý muốn không.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Combinator và attribute selector”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Nhầm descendant với direct child, phụ thuộc cấu trúc DOM quá sâu.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Combinator và attribute selector

```
.menu > li { list-style: none; }
a[target="_blank"] { text-decoration-style: dotted; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Viết menu lồng hai cấp và thử > so với dấu cách.

[ ] Đã làm: Chỉ style link mở tab mới bằng attribute selector.

[ ] Đã làm: Kiểm tra DOM có bị chọn ngoài ý muốn không.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Dấu cách chọn hậu duệ; > chọn con trực tiếp; + anh em liền kề; ~ anh em sau.

Tài liệu tham khảo

Tài liệu MDN — Combinator và attribute selector Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_selectors/Selectors_and_combinators

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Type, class, ID và universal selector

---

3. Pseudo-class cho trạng thái và vị trí

Pseudo-class như :hover, :focus-visible, :checked, :nth-child() mô tả trạng thái.

Mục tiêu học tập

- Giải thích được Pseudo-class và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Pseudo-class cho trạng thái và vị trí” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Pseudo-class · Focus-visible · Hover · Nth-child

Giải thích chi tiết

Vì sao cần học?

Pseudo-class như :hover, :focus-visible, :checked, :nth-child() mô tả trạng thái. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Pseudo-class như :hover, :focus-visible, :checked, :nth-child() mô tả trạng thái.
- Focus-visible giúp người dùng bàn phím thấy tiêu điểm.

- Không chỉ dùng hover vì thiết bị cảm ứng và bàn phím hoạt động khác.

Ví dụ để thử trên trang HTML

```
button:hover { background: #ddd; }
button:focus-visible { outline: 3px solid royalblue; outline-offset: 3px; }
```

Thực hành từng bước

1. Thiết kế button có trạng thái hover/focus/disabled.
2. Tab qua các button và quan sát focus.
3. Dùng :nth-child(odd) tạo bảng xen kẽ.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Pseudo-class cho trạng thái và vị trí”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Xóa outline mà không thay focus indicator phù hợp.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Pseudo-class cho trạng thái và vị trí

```
button:hover { background: #ddd; }
button:focus-visible { outline: 3px solid royalblue; outline-offset: 3px; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Thiết kế button có trạng thái hover/focus/disabled.

[ ] Đã làm: Tab qua các button và quan sát focus.

[ ] Đã làm: Dùng :nth-child(odd) tạo bảng xen kẽ.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Pseudo-class như :hover, :focus-visible, :checked, :nth-child() mô tả trạng thái.

Tài liệu tham khảo

Tài liệu MDN — Pseudo-class cho trạng thái và vị trí Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Combinator và attribute selector

---

4. Pseudo-element ::before, ::after, ::marker

Pseudo-element tạo hoặc style phần cụ thể như ::first-letter, ::before, ::marker.

Mục tiêu học tập

- Giải thích được Pseudo-element và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Pseudo-element ::before, ::after, ::marker" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Pseudo-element · Generated content · Marker

Giải thích chi tiết

Vì sao cần học?

Pseudo-element tạo hoặc style phần cụ thể như ::first-letter, ::before, ::marker. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Pseudo-element tạo hoặc style phần cụ thể như ::first-letter, ::before, ::marker.
- Content trong ::before/after thường mang tính trang trí.
- Thông tin quan trọng nên xuất hiện trong DOM thực, không phụ thuộc content CSS.

Ví dụ để thử trên trang HTML

```
.tag::before { content: "• "; color: teal; }
li::marker { color: #0891b2; }
```

Thực hành từng bước

1. Tạo bullet trang trí cho danh sách.
2. Thử ::selection và ::placeholder.
3. Tắt CSS và bảo đảm nội dung chính còn đủ.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Pseudo-element ::before, ::after, ::marker”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Đặt nhãn form hoặc thông tin bắt buộc chỉ trong content pseudo-element.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Pseudo-element ::before, ::after, ::marker

```
.tag::before { content: "• "; color: teal; }
li::marker { color: #0891b2; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo bullet trang trí cho danh sách.

[ ] Đã làm: Thử ::selection và ::placeholder.

[ ] Đã làm: Tắt CSS và bảo đảm nội dung chính còn đủ.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Pseudo-element tạo hoặc style phần cụ thể như ::first-letter, ::before, ::marker.

Tài liệu tham khảo

Tài liệu MDN — Pseudo-element ::before, ::after, ::marker Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Pseudo-class cho trạng thái và vị trí

5. Cascade, specificity, inheritance và !important

Cascade quyết định ưu tiên theo nguồn, importance, layer, specificity và thứ tự (cùng các yếu tố như scope).

Mục tiêu học tập

- Giải thích được Cascade và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Cascade, specificity, inheritance và !important" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Cascade · Specificity · Inheritance · Origin · Importance

Giải thích chi tiết

Vì sao cần học?

Cascade quyết định ưu tiên theo nguồn, importance, layer, specificity và thứ tự (cùng các yếu tố như scope). Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Cascade quyết định ưu tiên theo nguồn, importance, layer, specificity và thứ tự (cùng các yếu tố như scope).
- Specificity không phải quy tắc ưu tiên duy nhất; style từ layer khác có thể thắng.
- Một số property như color kế thừa; margin thường không kế thừa.
- !important chỉ nên dùng có chủ ý thay vì vá mọi xung đột.

Ví dụ để thử trên trang HTML

```
p { color: gray; }
.article p { color: navy; }
/* Inspect declaration thắng trong DevTools */
```

Thực hành từng bước

1. Tạo ba quy tắc cùng đặt color cho p.
2. Dự đoán rồi dùng Computed xem quy tắc thắng.
3. Loại bỏ !important bằng cách chỉnh kiến trúc selector.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Cascade, specificity, inheritance và !important”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Nghĩ selector dài nhất luôn thắng hoặc dùng !important để chữa lỗi.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Cascade, specificity, inheritance và !important

```
p { color: gray; }
.article p { color: navy; }
/* Inspect declaration thắng trong DevTools */
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Tạo ba quy tắc cùng đặt color cho p.
- [ ] Đã làm: Dự đoán rồi dùng Computed xem quy tắc thắng.
- [ ] Đã làm: Loại bỏ !important bằng cách chỉnh kiến trúc selector.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: Cascade quyết định ưu tiên theo nguồn, importance, layer, specificity và thứ tự (cùng các yếu tố như scope).

Tài liệu tham khảo

Tài liệu MDN — Cascade, specificity, inheritance và !important Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Cascade/Introduction

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Pseudo-element ::before, ::after, ::marker

GIAI ĐOẠN 3

03 · Box model, kích thước và trình bày

Kiểm soát box sizing, đơn vị, kiểu chữ, màu sắc và media mà không gây tràn.

1. Box Model và box-sizing

Box model gồm content, padding, border, margin; margin nằm ngoài border.

Mục tiêu học tập

- Giải thích được Content box và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Box Model và box-sizing" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Content box · Padding · Border · Margin · Border-box

Giải thích chi tiết

Vì sao cần học?

Box model gồm content, padding, border, margin; margin nằm ngoài border. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Box model gồm content, padding, border, margin; margin nằm ngoài border.
- box-sizing: border-box khiến width/height bao gồm padding và border.
- Margin collapse có thể xảy ra giữa các margin dọc trong normal flow.

Ví dụ để thử trên trang HTML

```
*, *::before, *::after { box-sizing: border-box; }
.card { width: 300px; padding: 24px; border: 2px solid; }
```

Thực hành từng bước

1. Đo box width với content-box và border-box.
2. Thử hai block có margin dọc.
3. Giải thích vì sao phần tử tràn 300px khi dùng content-box.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Box Model và box-sizing”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Cộng width, padding, border sai hoặc coi margin là phần nội dung.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Box Model và box-sizing

```
*, *::before, *::after { box-sizing: border-box; }
.card { width: 300px; padding: 24px; border: 2px solid; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Đo box width với content-box và border-box.

[ ] Đã làm: Thử hai block có margin dọc.

[ ] Đã làm: Giải thích vì sao phần tử tràn 300px khi dùng content-box.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Box model gồm content, padding, border, margin; margin nằm ngoài border.

Tài liệu tham khảo

Tài liệu MDN — Box Model và box-sizing Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Box_model

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Cascade, specificity, inheritance và !important

---

2. px, rem, em, %, vw, vh và đơn vị hiện đại

rem liên hệ font-size gốc; em phụ thuộc ngữ cảnh; % phụ thuộc containing block/thuộc tính.

Mục tiêu học tập

- Giải thích được Absolute units và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “px, rem, em, %, vw, vh và đơn vị hiện đại” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Absolute units · Relative units · rem · em · dvh · svh

Giải thích chi tiết

Vì sao cần học?

rem liên hệ font-size gốc; em phụ thuộc ngữ cảnh; % phụ thuộc containing block/thuộc tính. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- rem liên hệ font-size gốc; em phụ thuộc ngữ cảnh; % phụ thuộc containing block/thuộc tính.
- vw/vh dựa viewport, svh/lvh/dvh phản ánh các biến thể viewport trên mobile.
- Ưu tiên kích thước linh hoạt cho text; tránh cố định chiều cao các khối chứa văn bản.

Ví dụ để thử trên trang HTML

```
.hero { min-height: 100svh; padding: 2rem; }
.card { width: min(100%, 38rem); }
```

Thực hành từng bước

1. Đặt spacing bằng rem và so với px.
2. Thử chiều cao 100vh và 100dvh trên mobile.
3. Zoom trình duyệt 200% để tìm chỗ bị cắt chữ.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “px, rem, em, %, vw, vh và đơn vị hiện đại”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng 100vh mọi nơi, fixed height cho card dài hoặc lạm dụng vw cho font.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ px, rem, em, %, vw, vh và đơn vị hiện đại

```
.hero { min-height: 100svh; padding: 2rem; }
.card { width: min(100%, 38rem); }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Đặt spacing bằng rem và so với px.
- [ ] Đã làm: Thử chiều cao 100vh và 100dvh trên mobile.
- [ ] Đã làm: Zoom trình duyệt 200% để tìm chỗ bị cắt chữ.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: rem liên hệ font-size gốc; em phụ thuộc ngữ cảnh; % phụ thuộc containing block/ thuộc tính.

Tài liệu tham khảo

Tài liệu MDN — px, rem, em, %, vw, vh và đơn vị hiện đại Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Values_and_units

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Box Model và box-sizing

---

3. Màu sắc, background, gradient và contrast

CSS hỗ trợ hex, rgb(), hsl(), màu có alpha và hệ màu hiện đại như oklch().

Mục tiêu học tập

- Giải thích được Color và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Màu sắc, background, gradient và contrast" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Color · Alpha · Gradient · Contrast · OKLCH

Giải thích chi tiết

Vì sao cần học?

CSS hỗ trợ hex, rgb(), hsl(), màu có alpha và hệ màu hiện đại như oklch(). Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- CSS hỗ trợ hex, rgb(), hsl(), màu có alpha và hệ màu hiện đại như oklch().
- Background image và gradient nên không che giảm độ dễ đọc của chữ.
- Contrast phải được kiểm tra bằng công cụ, không dựa cảm nhận.

Ví dụ để thử trên trang HTML

```
.hero { color: #fff; background: linear-gradient(120deg,#0f172a,#155e75); }
```

Thực hành từng bước

1. Tạo bộ màu chính/phụ cho trang.
2. Đổi nền gradient và đo contrast văn bản.
3. Thử màu alpha trên nền sáng và tối.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Màu sắc, background, gradient và contrast”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng màu chỉ để truyền đạt trạng thái hoặc chữ nhạt đến mức khó đọc.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Màu sắc, background, gradient và contrast

```
.hero { color: #fff; background: linear-gradient(120deg,#0f172a,#155e75); }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo bộ màu chính/phụ cho trang.
[ ] Đã làm: Đổi nền gradient và đo contrast văn bản.
[ ] Đã làm: Thử màu alpha trên nền sáng và tối.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: CSS hỗ trợ hex, rgb(), hsl(), màu có alpha và hệ màu hiện đại như oklch().

Tài liệu tham khảo

Tài liệu MDN — Màu sắc, background, gradient và contrast Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

px, rem, em, %, vw, vh và đơn vị hiện đại

---

4. Typography dễ đọc và font fallback

font-family dùng fallback; font-weight, line-height, letter-spacing ảnh hưởng khả năng đọc.

Mục tiêu học tập

- Giải thích được Font stack và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Typography dễ đọc và font fallback" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Font stack · Line height · Font display · Text wrap

Giải thích chi tiết

Vì sao cần học?

font-family dùng fallback; font-weight, line-height, letter-spacing ảnh hưởng khả năng đọc. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- font-family dùng fallback; font-weight, line-height, letter-spacing ảnh hưởng khả năng đọc.
- Line-height thường dùng số không đơn vị để kế thừa tỷ lệ tốt.
- Font web cần cân nhắc tải, font-display và fallback để tránh dịch chuyển.

Ví dụ để thử trên trang HTML

```
body { font: 400 1rem/1.6 system-ui, sans-serif; }
h1 { line-height: 1.15; text-wrap: balance; }
```

Thực hành từng bước

1. Thiết kế hệ thống chữ cho heading, body, caption.
2. Zoom 200% để kiểm tra ngắt dòng.
3. So sánh font hệ thống với web font và quan sát Network.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: "Typography dễ đọc và font fallback". Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Thiết kế chỉ dựa font đẹp mà bỏ khả năng đọc hoặc khóa chiều cao dòng.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Typography dễ đọc và font fallback

```
body { font: 400 1rem/1.6 system-ui, sans-serif; }
h1 { line-height: 1.15; text-wrap: balance; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Thiết kế hệ thống chữ cho heading, body, caption.
- [ ] Đã làm: Zoom 200% để kiểm tra ngắt dòng.
- [ ] Đã làm: So sánh font hệ thống với web font và quan sát Network.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: font-family dùng fallback; font-weight, line-height, letter-spacing ảnh hưởng khả năng đọc.

Tài liệu tham khảo

Tài liệu MDN — Typography dễ đọc và font fallback Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Text_styling/Fundamentals

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Màu sắc, background, gradient và contrast

5. Object-fit, aspect-ratio và media co giãn

Ảnh nên dùng max-width:100% và height:auto trong nội dung.

Mục tiêu học tập

- Giải thích được Responsive image và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Object-fit, aspect-ratio và media co giãn" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Responsive image · Object-fit · Aspect ratio · Intrinsic sizing

Giải thích chi tiết

Vì sao cần học?

Ảnh nên dùng max-width:100% và height:auto trong nội dung. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Ảnh nên dùng max-width:100% và height:auto trong nội dung.
- object-fit:cover cắt hình có chủ ý; contain giữ toàn bộ.
- aspect-ratio ổn định không gian trước khi nội dung tải xong.

Ví dụ để thử trên trang HTML

```
img { display:block; max-width:100%; height:auto; }
.avatar { width: 5rem; aspect-ratio: 1; object-fit: cover; }
```

Thực hành từng bước

1. Tạo ảnh card và avatar vuông.
2. So sánh cover/contain với ảnh khác tỷ lệ.
3. Kiểm tra ảnh bị bóp méo khi chỉ đặt width/height.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Object-fit, aspect-ratio và media co giãn”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Đặt width:100% cho mọi icon hoặc bóp tỷ lệ ảnh.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Object-fit, aspect-ratio và media co giãn

```
img { display:block; max-width:100%; height:auto; }
.avatar { width: 5rem; aspect-ratio: 1; object-fit: cover; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo ảnh card và avatar vuông.

[ ] Đã làm: So sánh cover/contain với ảnh khác tỷ lệ.

[ ] Đã làm: Kiểm tra ảnh bị bóp méo khi chỉ đặt width/height.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Ảnh nên dùng max-width:100% và height:auto trong nội dung.

Tài liệu tham khảo

Tài liệu MDN — Object-fit, aspect-ratio và media co giãn Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/object-fit

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Typography dễ đọc và font fallback

GIAI ĐOẠN 4

04 · Normal Flow, Display và Position

Hiểu mô hình bố cục của trình duyệt trước khi dùng Flexbox hay Grid.

1. Normal flow, block, inline, inline-block

Trong normal flow, block thường chiếm chiều ngang khả dụng và xuống dòng; inline chảy theo dòng chữ.

Mục tiêu học tập

- Giải thích được Normal flow và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Normal flow, block, inline, inline-block” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Normal flow · Block · Inline · Inline-block · Display

Giải thích chi tiết

Vì sao cần học?

Trong normal flow, block thường chiếm chiều ngang khả dụng và xuống dòng; inline chảy theo dòng chữ. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Trong normal flow, block thường chiếm chiều ngang khả dụng và xuống dòng; inline chảy theo dòng chữ.
- display thay đổi cách phần tử tham gia layout; display:none loại khỏi cây layout và thường khỏi accessibility tree.
- visibility:hidden giấu nhưng vẫn giữ chỗ bố cục.

Ví dụ để thử trên trang HTML

```
.badge { display: inline-block; padding: .25rem .5rem; }
.stack > * + * { margin-block-start: 1rem; }
```

Thực hành từng bước

1. Thử span display:inline so với inline-block.
2. Tạo layout article không cần position.
3. So sánh display:none và visibility:hidden.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Normal flow, block, inline, inline-block”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng absolute position để đặt mọi thành phần thay normal flow.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Normal flow, block, inline, inline-block

```
.badge { display: inline-block; padding: .25rem .5rem; }
.stack > * + * { margin-block-start: 1rem; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Thử span display:inline so với inline-block.
- [ ] Đã làm: Tạo layout article không cần position.
- [ ] Đã làm: So sánh display:none và visibility:hidden.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: Trong normal flow, block thường chiếm chiều ngang khả dụng và xuống dòng; inline chảy theo dòng chữ.

Tài liệu tham khảo

Tài liệu MDN — Normal flow, block, inline, inline-block Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Introduction

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Object-fit, aspect-ratio và media co giãn

2. min/max size, overflow và min-width:0

min-width/max-width giúp linh hoạt, overflow xử lý khi nội dung dài hơn box.

Mục tiêu học tập

- Giải thích được Overflow và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "min/max size, overflow và min-width:0" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Overflow · Min-content · Max-content · Min-width · Intrinsic size

Giải thích chi tiết

Vì sao cần học?

min-width/max-width giúp linh hoạt, overflow xử lý khi nội dung dài hơn box. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- min-width/max-width giúp linh hoạt, overflow xử lý khi nội dung dài hơn box.
- Trong flex/grid, min-width:auto đôi khi ngăn phần tử thu nhỏ; min-width:0 giúp xử lý.
- Tránh overflow:hidden như cách giấu lỗi layout; ưu tiên sửa nguyên nhân.

Ví dụ để thử trên trang HTML

```
.content { width: min(100%, 70rem); margin-inline:auto; }
.flex-child { min-width:0; overflow-wrap:anywhere; }
```

Thực hành từng bước

1. Thêm URL dài vào card và tìm nguyên nhân tràn.
2. So sánh overflow:auto, hidden và clip.
3. Đặt min-width:0 cho flex child bị tràn.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “min/max size, overflow và min-width:0”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Ẩn overflow toàn trang để che horizontal scrolling.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ min/max size, overflow và min-width:0

```
.content { width: min(100%, 70rem); margin-inline:auto; }
.flex-child { min-width:0; overflow-wrap:anywhere; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Thêm URL dài vào card và tìm nguyên nhân tràn.

[ ] Đã làm: So sánh overflow:auto, hidden và clip.

[ ] Đã làm: Đặt min-width:0 cho flex child bị tràn.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: min-width/max-width giúp linh hoạt, overflow xử lý khi nội dung dài hơn box.

Tài liệu tham khảo

Tài liệu MDN — min/max size, overflow và min-width:0 Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/overflow

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Normal flow, block, inline, inline-block

---

3. Position: relative, absolute, fixed, sticky

Relative dịch phần tử khỏi vị trí bình thường nhưng vẫn giữ chỗ.

Mục tiêu học tập

- Giải thích được Relative và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Position: relative, absolute, fixed, sticky” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Relative · Absolute · Fixed · Sticky · Containing block

Giải thích chi tiết

Vì sao cần học?

Relative dịch phần tử khỏi vị trí bình thường nhưng vẫn giữ chỗ. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Relative dịch phần tử khỏi vị trí bình thường nhưng vẫn giữ chỗ.
- Absolute định vị theo containing block thích hợp và thoát normal flow.
- Fixed gắn viewport trong nhiều trường hợp; sticky phụ thuộc scroll container và ngưỡng inset.

Ví dụ để thử trên trang HTML

```
.card { position:relative; }
.card__badge { position:absolute; inset-block-start:1rem; inset-inline-end:1rem; }
.header { position:sticky; top:0; }
```

Thực hành từng bước

1. Tạo badge nổi trong card.
2. Tạo sticky header và thử cuộn.
3. Kiểm tra sticky không hoạt động khi ancestor overflow thay đổi.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Position: relative, absolute, fixed, sticky”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Sử dụng fixed/absolute cho bố cục chính khiến chữ chồng nhau.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Position: relative, absolute, fixed, sticky

```
.card { position:relative; }
.card__badge { position:absolute; inset-block-start:1rem; inset-inline-end:1rem; }
.header { position:sticky; top:0; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo badge nổi trong card.

[ ] Đã làm: Tạo sticky header và thử cuộn.

[ ] Đã làm: Kiểm tra sticky không hoạt động khi ancestor overflow thay đổi.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Relative dịch phần tử khỏi vị trí bình thường nhưng vẫn giữ chỗ.

Tài liệu tham khảo

Tài liệu MDN — Position: relative, absolute, fixed, sticky Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Positioning

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

min/max size, overflow và min-width:0

---

4. z-index, stacking context và top layer

z-index có hiệu lực theo stacking context; giá trị lớn không luôn thắng context khác.

Mục tiêu học tập

- Giải thích được Z-index và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "z-index, stacking context và top layer" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Z-index · Stacking context · Top layer · Overlay

Giải thích chi tiết

Vì sao cần học?

z-index có hiệu lực theo stacking context; giá trị lớn không luôn thắng context khác. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- z-index có hiệu lực theo stacking context; giá trị lớn không luôn thắng context khác.
- Transform, opacity và một số thuộc tính có thể tạo stacking context mới.
- Dialog/popover top layer xử lý lớp giao diện theo cơ chế riêng.

Ví dụ để thử trên trang HTML

```
.overlay { position:fixed; inset:0; z-index:100; background:#0008; }
```

Thực hành từng bước

1. Tạo hai hộp chồng nhau và đổi z-index.
2. Thêm transform cho parent để quan sát context.
3. Kiểm tra dialog dùng top layer trong DevTools.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “z-index, stacking context và top layer”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Lạm dụng z-index:999999 để sửa vấn đề stacking context.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ z-index, stacking context và top layer


```
.overlay { position:fixed; inset:0; z-index:100; background:#0008; }
```


Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo hai hộp chồng nhau và đổi z-index.
[ ] Đã làm: Thêm transform cho parent để quan sát context.
[ ] Đã làm: Kiểm tra dialog dùng top layer trong DevTools.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: z-index có hiệu lực theo stacking context; giá trị lớn không luôn thắng context khác.

Tài liệu tham khảo

Tài liệu MDN — z-index, stacking context và top layer Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_positioned_layout/Stacking_context

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Position: relative, absolute, fixed, sticky

---

5. Logical properties và giao diện đa ngôn ngữ

margin-inline, padding-block, inset-inline-start thể hiện hướng logic thay vì left/right cứng.

Mục tiêu học tập

- Giải thích được Logical properties và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Logical properties và giao diện đa ngôn ngữ" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Logical properties · Inline axis · Block axis · RTL

Giải thích chi tiết

Vì sao cần học?

margin-inline, padding-block, inset-inline-start thể hiện hướng logic thay vì left/right cứng. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- margin-inline, padding-block, inset-inline-start thể hiện hướng logic thay vì left/right cứng.
- Writing mode và hướng văn bản ảnh hưởng bố cục.
- Thiết kế logic giúp mở rộng RTL và responsive dễ hơn.

Ví dụ để thử trên trang HTML

```
.panel { padding-inline:1.5rem; padding-block:1rem; margin-inline:auto; }
```

Thực hành từng bước

1. Thay margin-left/right bằng margin-inline.
2. Chuyển dir=rtl trong HTML để kiểm tra.
3. Quan sát border-inline-start ở cả hai hướng.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: "Logical properties và giao diện đa ngôn ngữ". Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Mặc định hướng trái-phải cho mọi giao diện hoặc đảo icon sai ngữ cảnh.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Logical properties và giao diện đa ngôn ngữ

```
.panel { padding-inline:1.5rem; padding-block:1rem; margin-inline:auto; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Thay margin-left/right bằng margin-inline.
- [ ] Đã làm: Chuyển dir=rtl trong HTML để kiểm tra.
- [ ] Đã làm: Quan sát border-inline-start ở cả hai hướng.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: margin-inline, padding-block, inset-inline-start thể hiện hướng logic thay vì left/right cứng.

Tài liệu tham khảo

Tài liệu MDN — Logical properties và giao diện đa ngôn ngữ Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_logical_properties_and_values

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

z-index, stacking context và top layer

GIAI ĐOẠN 5

05 · Flexbox và CSS Grid thực chiến

Làm chủ layout một chiều và hai chiều, căn chỉnh, wrap và responsive không hack.

1. Flex container, main axis, cross axis

display:flex biến children thành flex items.

Mục tiêu học tập

- Giải thích được Flex container và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Flex container, main axis, cross axis" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Flex container · Main axis · Cross axis · Gap · Alignment

Giải thích chi tiết

Vì sao cần học?

display:flex biến children thành flex items. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- display:flex biến children thành flex items.
- flex-direction xác định trục chính, justify-content căn theo trục chính; align-items theo trục ngang còn lại.
- gap tạo khoảng cách giữa items nhất quán.

Ví dụ để thử trên trang HTML

```
.toolbar { display:flex; align-items:center; justify-content:space-between; gap:1rem; }
```

Thực hành từng bước

1. Tạo navbar logo, menu, action.
2. Đổi flex-direction và lý giải trục.
3. Thử align-items:stretch, center, baseline.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Flex container, main axis, cross axis”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Nghĩ justify-content luôn là căn ngang bất kể flex-direction.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Flex container, main axis, cross axis

```
.toolbar { display:flex; align-items:center; justify-content:space-between; gap:1rem; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo navbar logo, menu, action.
[ ] Đã làm: Đổi flex-direction và lý giải trục.
[ ] Đã làm: Thử align-items:stretch, center, baseline.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: display:flex biến children thành flex items.

Tài liệu tham khảo

Tài liệu MDN — Flex container, main axis, cross axis Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Flexbox

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Logical properties và giao diện đa ngôn ngữ

---

2. flex-grow, shrink, basis và wrap

flex:1 là shorthand gồm grow/shrink/basis (giá trị cụ thể tùy shorthand).

Mục tiêu học tập

- Giải thích được Flex basis và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “flex-grow, shrink, basis và wrap” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Flex basis · Flex grow · Flex shrink · Flex wrap

Giải thích chi tiết

Vì sao cần học?

flex:1 là shorthand gồm grow/shrink/basis (giá trị cụ thể tùy shorthand). Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- flex:1 là shorthand gồm grow/shrink/basis (giá trị cụ thể tùy shorthand).
- min-width và nội dung dài ảnh hưởng việc shrink.
- flex-wrap:wrap tạo dòng mới khi thiếu chỗ, nhưng không thay thế mọi responsive logic.

Ví dụ để thử trên trang HTML

```
.columns { display:flex; flex-wrap:wrap; gap:1rem; }
.columns > * { flex:1 1 16rem; min-width:0; }
```

Thực hành từng bước

1. Tạo 3 card tự xuống dòng ở viewport hẹp.
2. Thử flex:1, flex:0 1 auto và flex-basis.
3. Nhập chữ dài và xử lý không tràn.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “flex-grow, shrink, basis và wrap”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Chỉ thay width cố định mà quên flex-basis/shrink.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ flex-grow, shrink, basis và wrap

```
.columns { display:flex; flex-wrap:wrap; gap:1rem; }
.columns > * { flex:1 1 16rem; min-width:0; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo 3 card tự xuống dòng ở viewport hẹp.

[ ] Đã làm: Thử flex:1, flex:0 1 auto và flex-basis.

[ ] Đã làm: Nhập chữ dài và xử lý không tràn.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: flex:1 là shorthand gồm grow/shrink/basis (giá trị cụ thể tùy shorthand).

Tài liệu tham khảo

Tài liệu MDN — flex-grow, shrink, basis và wrap Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/flex

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Flex container, main axis, cross axis

---

3. Grid columns, rows, gap và placement

Grid tạo lưới hai chiều với các track và line.

Mục tiêu học tập

- Giải thích được Grid track và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Grid columns, rows, gap và placement” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Grid track · fr · Minmax · Grid line · Placement

Giải thích chi tiết

Vì sao cần học?

Grid tạo lưới hai chiều với các track và line. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Grid tạo lưới hai chiều với các track và line.
- repeat(), minmax(), fr dùng phân phối kích thước linh hoạt.
- grid-column/grid-row đặt vị trí; ưu tiên nguồn DOM theo thứ tự có nghĩa.

Ví dụ để thử trên trang HTML

```
.dashboard { display:grid; grid-template-columns: 15rem minmax(0,1fr); gap:1.5rem; }
```

Thực hành từng bước

1. Dựng sidebar và khu nội dung.
2. Dùng fr, minmax và so với px.
3. Dùng Grid Inspector trong DevTools đọc track.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Grid columns, rows, gap và placement”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Cố gắng dùng Grid cho một hàng nút đơn giản hoặc đảo visual order sai keyboard order.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Grid columns, rows, gap và placement

```
.dashboard { display:grid; grid-template-columns: 15rem minmax(0,1fr); gap:1.5rem; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Dựng sidebar và khu nội dung.
[ ] Đã làm: Dùng fr, minmax và so với px.
[ ] Đã làm: Dùng Grid Inspector trong DevTools đọc track.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: Grid tạo lưới hai chiều với các track và line.

Tài liệu tham khảo

Tài liệu MDN — Grid columns, rows, gap và placement Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Grids

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

flex-grow, shrink, basis và wrap

---

4. Auto-fit, auto-fill và layout card tự thích nghi

repeat(auto-fit,minmax(...)) tạo cột dựa không gian khả dụng.

Mục tiêu học tập

- Giải thích được Auto-fit và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Auto-fit, auto-fill và layout card tự thích nghi" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Auto-fit · Auto-fill · Responsive grid · Intrinsic sizing

Giải thích chi tiết

Vì sao cần học?

repeat(auto-fit,minmax(...)) tạo cột dựa không gian khả dụng. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- repeat(auto-fit,minmax(...)) tạo cột dựa không gian khả dụng.
- auto-fill giữ track trống còn auto-fit có thể co track trống.
- min(100%, size) giúp tránh min track lớn hơn viewport.

Ví dụ để thử trên trang HTML

```
.cards { display:grid; grid-template-columns:repeat(auto-fit,minmax(min(100%,16rem),1fr));
gap:1rem; }
```

Thực hành từng bước

1. Tạo gallery 2–5 card ở các viewport.
2. So sánh auto-fit và auto-fill với ít items.
3. Kiểm tra chữ dài, ảnh rộng và zoom 200%.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Auto-fit, auto-fill và layout card tự thích nghi”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Đặt grid-template-columns:repeat(4,1fr) ở mọi màn hình.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Auto-fit, auto-fill và layout card tự thích nghi

```
.cards { display:grid; grid-template-columns:repeat(auto-fit,minmax(min(100%,16rem),1fr));
gap:1rem; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo gallery 2–5 card ở các viewport.
[ ] Đã làm: So sánh auto-fit và auto-fill với ít items.
[ ] Đã làm: Kiểm tra chữ dài, ảnh rộng và zoom 200%.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: repeat(auto-fit,minmax(...)) tạo cột dựa không gian khả dụng.

Tài liệu tham khảo

Tài liệu MDN — Auto-fit, auto-fill và layout card tự thích nghi Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/repeat

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Grid columns, rows, gap và placement

5. Chọn Grid, Flexbox hay normal flow

Normal flow tốt cho bài viết; Flexbox cho cấu trúc một chiều; Grid cho bố cục hai chiều.

Mục tiêu học tập

- Giải thích được Layout strategy và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Chọn Grid, Flexbox hay normal flow" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Layout strategy · DOM order · Grid vs Flex · Normal flow

Giải thích chi tiết

Vì sao cần học?

Normal flow tốt cho bài viết; Flexbox cho cấu trúc một chiều; Grid cho bố cục hai chiều. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Normal flow tốt cho bài viết; Flexbox cho cấu trúc một chiều; Grid cho bố cục hai chiều.
- Có thể kết hợp Grid ở cấp trang, Flex ở component.
- Layout cần giữ thứ tự đọc của DOM để hỗ trợ bàn phím và screen reader.

Ví dụ để thử trên trang HTML

```
.page { display:grid; grid-template-columns:1fr minmax(0,60rem) 1fr; }
.page > main { grid-column:2; }
```

Thực hành từng bước

1. Phác thảo layout dashboard và chọn công cụ.
2. Chuyển navbar Grid sang Flex để so sánh.
3. Kiểm tra thứ tự Tab và thứ tự đọc sau khi sắp xếp.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Chọn Grid, Flexbox hay normal flow”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng order/grid-area để thay DOM order và làm khó điều hướng.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Chọn Grid, Flexbox hay normal flow

```
.page { display:grid; grid-template-columns:1fr minmax(0,60rem) 1fr; }
.page > main { grid-column:2; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Phác thảo layout dashboard và chọn công cụ.
- [ ] Đã làm: Chuyển navbar Grid sang Flex để so sánh.
- [ ] Đã làm: Kiểm tra thứ tự Tab và thứ tự đọc sau khi sắp xếp.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: Normal flow tốt cho bài viết; Flexbox cho cấu trúc một chiều; Grid cho bố cục hai chiều.

Tài liệu tham khảo

Tài liệu MDN — Chọn Grid, Flexbox hay normal flow Tài liệu
https://web.dev/learn/css/layout

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Auto-fit, auto-fill và layout card tự thích nghi

GIAI ĐOẠN 6

06 · Responsive và CSS thích nghi theo component

Thiết kế mobile-first, fluid type, container queries và hỗ trợ mới theo điều kiện.

1. Viewport, media queries và mobile-first

Viewport meta trong HTML rất quan trọng với layout mobile.

Mục tiêu học tập

- Giải thích được Viewport và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Viewport, media queries và mobile-first" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Viewport · Media query · Mobile-first · Breakpoint

Giải thích chi tiết

Vì sao cần học?

Viewport meta trong HTML rất quan trọng với layout mobile. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Viewport meta trong HTML rất quan trọng với layout mobile.
- @media (min-width:...) mở rộng thiết kế từ nhỏ đến lớn; breakpoint dựa nội dung, không theo tên thiết bị cố định.
- Kiểm tra cả chiều ngang và zoom lớn.

Ví dụ để thử trên trang HTML

```
.nav { display:flex; flex-direction:column; }
@media (min-width:48rem) { .nav { flex-direction:row; } }
```

Thực hành từng bước

1. Dựng header trên màn 360px.
2. Tăng viewport đến điểm nội dung cần đổi layout.
3. Kiểm tra 320px, tablet, desktop và zoom 200%.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: "Viewport, media queries và mobile-first". Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Chỉ test vài mẫu thiết bị hoặc cố định mọi breakpoint theo iPhone.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Viewport, media queries và mobile-first

```
.nav { display:flex; flex-direction:column; }
@media (min-width:48rem) { .nav { flex-direction:row; } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Dựng header trên màn 360px.
[ ] Đã làm: Tăng viewport đến điểm nội dung cần đổi layout.
[ ] Đã làm: Kiểm tra 320px, tablet, desktop và zoom 200%.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: Viewport meta trong HTML rất quan trọng với layout mobile.

Tài liệu tham khảo

Tài liệu MDN — Viewport, media queries và mobile-first Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Media_queries

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Chọn Grid, Flexbox hay normal flow

---

2. clamp, min, max và typography linh hoạt

clamp(min,preferred,max) tạo giá trị linh hoạt có giới hạn.

Mục tiêu học tập

- Giải thích được Clamp và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "clamp, min, max và typography linh hoạt" trong trình duyệt

- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Clamp · Min · Max · Fluid typography

Giải thích chi tiết

Vì sao cần học?

clamp(min,preferred,max) tạo giá trị linh hoạt có giới hạn. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- clamp(min,preferred,max) tạo giá trị linh hoạt có giới hạn.
- min()/max() hữu ích cho width và padding responsively.
- Với font-size, kết hợp đơn vị tương đối để hỗ trợ zoom và người dùng tùy chỉnh chữ.

Ví dụ để thử trên trang HTML

```
h1 { font-size:clamp(2rem,1.2rem + 3vw,4rem); }
.container { padding-inline:clamp(1rem,3vw,2.5rem); }
```

Thực hành từng bước

1. Tạo heading fluid và đo ở 3 viewport.
2. Thử min/max cho width container.
3. Zoom chữ và xác nhận không bị che/cắt.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “clamp, min, max và typography linh hoạt”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng viewport units thuần cho chữ khiến người dùng khó phóng to.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ clamp, min, max và typography linh hoạt

```
h1 { font-size:clamp(2rem,1.2rem + 3vw,4rem); }
.container { padding-inline:clamp(1rem,3vw,2.5rem); }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Tạo heading fluid và đo ở 3 viewport.
- [ ] Đã làm: Thử min/max cho width container.
- [ ] Đã làm: Zoom chữ và xác nhận không bị che/cắt.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: clamp(min,preferred,max) tạo giá trị linh hoạt có giới hạn.

Tài liệu tham khảo

Tài liệu MDN — clamp, min, max và typography linh hoạt Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/clamp

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Viewport, media queries và mobile-first

---

3. Container size queries và đơn vị cqi

Container queries phản ứng với kích thước vùng chứa, không phải toàn viewport.

Mục tiêu học tập

- Giải thích được Container query và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Container size queries và đơn vị cqi” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Container query · Query container · Container-type · cqi

Giải thích chi tiết

Vì sao cần học?

Container queries phản ứng với kích thước vùng chứa, không phải toàn viewport. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Container queries phản ứng với kích thước vùng chứa, không phải toàn viewport.
- Đặt container-type:inline-size cho parent; @container dùng để đổi layout child.

- Một component có thể hiển thị khác nhau trong sidebar và main cùng viewport.

Ví dụ để thử trên trang HTML

```
.widget-wrapper { container-type:inline-size; }
.widget { display:grid; gap:1rem; }
@container (min-width:30rem) { .widget { grid-template-columns:8rem 1fr; } }
```

Thực hành từng bước

1. Tạo card dùng lại ở sidebar và main.
2. So sánh hành vi @media và @container.
3. Thay chiều rộng parent để card tự đổi layout.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Container size queries và đơn vị cqi”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Đặt container-type lên chính phần tử cần truy vấn rồi mong nó query chính mình.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Container size queries và đơn vị cqi

```
.widget-wrapper { container-type:inline-size; }
.widget { display:grid; gap:1rem; }
@container (min-width:30rem) { .widget { grid-template-columns:8rem 1fr; } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo card dùng lại ở sidebar và main.

[ ] Đã làm: So sánh hành vi @media và @container.

[ ] Đã làm: Thay chiều rộng parent để card tự đổi layout.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Container queries phản ứng với kích thước vùng chứa, không phải toàn viewport.

Tài liệu tham khảo

Tài liệu MDN — Container size queries và đơn vị cqi Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_containment/Container_size_and_style_queries

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

clamp, min, max và typography linh hoạt

---

4. @supports, progressive enhancement và fallback

@supports kiểm tra khả năng hiểu khai báo CSS, không chứng minh mọi trải nghiệm hoàn hảo.

Mục tiêu học tập

- Giải thích được Feature query và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "@supports, progressive enhancement và fallback" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Feature query · Progressive enhancement · Fallback · Baseline

Giải thích chi tiết

Vì sao cần học?

@supports kiểm tra khả năng hiểu khai báo CSS, không chứng minh mọi trải nghiệm hoàn hảo. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- @supports kiểm tra khả năng hiểu khai báo CSS, không chứng minh mọi trải nghiệm hoàn hảo.
- Viết fallback cơ bản trước, bổ sung tính năng hiện đại khi được hỗ trợ.
- Baseline/Can I use giúp quyết định tính năng và mức hỗ trợ mục tiêu.

Ví dụ để thử trên trang HTML

```
.card { background:#eef2ff; }
@supports (color:oklch(60% .15 250)) { .card { background:oklch(95% .02 250); } }
```

Thực hành từng bước

1. Tạo fallback màu hex cho màu hiện đại.
2. Thử vô hiệu @supports và xem giao diện.
3. Tra cứu hỗ trợ tính năng trước khi dùng production.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “@supports, progressive enhancement và fallback”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng CSS mới không có fallback khi sản phẩm cần hỗ trợ browser cũ.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ @supports, progressive enhancement và fallback

```
.card { background:#eef2ff; }
@supports (color:oklch(60% .15 250)) { .card { background:oklch(95% .02 250); } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Tạo fallback màu hex cho màu hiện đại.
- [ ] Đã làm: Thử vô hiệu @supports và xem giao diện.
- [ ] Đã làm: Tra cứu hỗ trợ tính năng trước khi dùng production.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: @supports kiểm tra khả năng hiểu khai báo CSS, không chứng minh mọi trải nghiệm hoàn hảo.

Tài liệu tham khảo

Tài liệu MDN — @supports, progressive enhancement và fallback Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/@supports

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Container size queries và đơn vị cqi

---

5. Prefers-color-scheme, reduced-motion và contrast

Media feature phát hiện một số sở thích người dùng như color scheme và reduced motion.

Mục tiêu học tập

- Giải thích được Color scheme và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Prefers-color-scheme, reduced-motion và contrast" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Color scheme · Reduced motion · Contrast preference · User preferences

Giải thích chi tiết

Vì sao cần học?

Media feature phát hiện một số sở thích người dùng như color scheme và reduced motion. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Media feature phát hiện một số sở thích người dùng như color scheme và reduced motion.
- Dark mode phải kiểm tra độ tương phản, không chỉ đảo màu.
- Reduced motion giảm chuyển động không thiết yếu, không loại bỏ tín hiệu trạng thái quan trọng.

Ví dụ để thử trên trang HTML

```
@media (prefers-color-scheme:dark) { :root { color-scheme:dark; } }
@media (prefers-reduced-motion:reduce) { *,*::before,*::after { scroll-behavior:auto; } }
```

Thực hành từng bước

1. Thêm dark mode dựa biến CSS.
2. Bật reduced motion trong OS và kiểm tra animation.
3. Đo contrast ở cả hai chế độ.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Prefers-color-scheme, reduced-motion và contrast”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Buộc animation mạnh cho mọi người hoặc coi dark mode là đảo màu bằng filter.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Prefers-color-scheme, reduced-motion và contrast

```
@media (prefers-color-scheme:dark) { :root { color-scheme:dark; } }
@media (prefers-reduced-motion:reduce) { *,*::before,*::after { scroll-behavior:auto; } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Thêm dark mode dựa biến CSS.
- [ ] Đã làm: Bật reduced motion trong OS và kiểm tra animation.
- [ ] Đã làm: Đo contrast ở cả hai chế độ.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: Media feature phát hiện một số sở thích người dùng như color scheme và reduced motion.

Tài liệu tham khảo

Tài liệu MDN — Prefers-color-scheme, reduced-motion và contrast Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/@media/prefers-reduced-motion

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

@supports, progressive enhancement và fallback

GIAI ĐOẠN 7

07 · CSS hiện đại và tổ chức code

Làm chủ selectors mới, nesting, cascade layers, custom properties và cấu trúc dễ bảo trì.

1. :is(), :where(), :has() và :not()

:is() gộp các selector, specificity lấy theo đối số phù hợp nhất.

Mục tiêu học tập

- Giải thích được is() và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài " :is(), :where(), :has() và :not()" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

is() · where() · has() · not() · Relational selectors

Giải thích chi tiết

Vì sao cần học?

:is() gộp các selector, specificity lấy theo đối số phù hợp nhất. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- :is() gộp các selector, specificity lấy theo đối số phù hợp nhất.
- :where() luôn có specificity bằng 0.
- :has() chọn phần tử dựa vào quan hệ con/hậu duệ; dùng có chủ ý vì tác động và độ phức tạp.

Ví dụ để thử trên trang HTML

```
.card:has(img) { display:grid; }
:where(.prose h2,.prose h3) { scroll-margin-top:5rem; }
```

Thực hành từng bước

1. Viết selector cho card có ảnh.
2. So sánh :is() và :where() trong DevTools.
3. Tạo form-group:has(input:invalid) với trạng thái lỗi.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “ :is(), :where(), :has() và :not()”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Giả định :where() tăng specificity giống :is().
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ :is(), :where(), :has() và :not()

```
.card:has(img) { display:grid; }
:where(.prose h2,.prose h3) { scroll-margin-top:5rem; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Viết selector cho card có ảnh.

[ ] Đã làm: So sánh :is() và :where() trong DevTools.

[ ] Đã làm: Tạo form-group:has(input:invalid) với trạng thái lỗi.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: :is() gộp các selector, specificity lấy theo đối số phù hợp nhất.

Tài liệu tham khảo

Tài liệu MDN — :is(), :where(), :has() và :not() Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/:has

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Prefers-color-scheme, reduced-motion và contrast

---

2. Native CSS nesting và toán tử &

CSS nesting cho phép lồng selector mà không cần Sass.

Mục tiêu học tập

- Giải thích được CSS nesting và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Native CSS nesting và toán tử &” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

CSS nesting · Ampersand · Nested rule · Specificity

Giải thích chi tiết

Vì sao cần học?

CSS nesting cho phép lồng selector mà không cần Sass. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- CSS nesting cho phép lồng selector mà không cần Sass.
- & xác định vị trí selector cha trong selector mới.
- Không lồng quá sâu khiến specificity và bảo trì khó.

Ví dụ để thử trên trang HTML

```
.card { padding:1rem; & h2 { margin:0; } &:hover { box-shadow:0 4px 20px #0002; } }
```

Thực hành từng bước

1. Refactor 5 selector phẳng sang nesting.
2. Dùng &:hover và & .child đúng ngữ nghĩa.
3. Kiểm tra cascade sau refactor.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Native CSS nesting và toán tử &”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Nhầm nesting CSS với cú pháp Sass trong mọi trường hợp.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Native CSS nesting và toán tử &

```
.card { padding:1rem; & h2 { margin:0; } &:hover { box-shadow:0 4px 20px #0002; } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Refactor 5 selector phẳng sang nesting.

[ ] Đã làm: Dùng &:hover và & .child đúng ngữ nghĩa.

[ ] Đã làm: Kiểm tra cascade sau refactor.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: CSS nesting cho phép lồng selector mà không cần Sass.

Tài liệu tham khảo

Tài liệu MDN — Native CSS nesting và toán tử & Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_nesting/Using_CSS_nesting

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

:is(), :where(), :has() và :not()

---

3. CSS custom properties, var() và design tokens

--token là custom property; var(--name,fallback) đọc giá trị.

Mục tiêu học tập

- Giải thích được Custom property và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "CSS custom properties, var() và design tokens" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Custom property · var() · Design tokens · Theming

Giải thích chi tiết

Vì sao cần học?

--token là custom property; var(--name,fallback) đọc giá trị. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- --token là custom property; var(--name,fallback) đọc giá trị.
- Custom properties có thể kế thừa và thay theo theme/context.
- Token đặt theo ý nghĩa giúp thống nhất spacing, color, radius.

Ví dụ để thử trên trang HTML

```
:root { --brand:#0f766e; --space-3:.75rem; }
.button { background:var(--brand); padding:var(--space-3); }
```

Thực hành từng bước

1. Tạo bảng token màu, spacing, radius.
2. Đổi --brand và quan sát toàn trang.
3. Dùng fallback var() cho token tùy chọn.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “CSS custom properties, var() và design tokens”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Đặt tên token --blue-500 cho mọi ý nghĩa hoặc quên fallback khi giá trị chưa có.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ CSS custom properties, var() và design tokens


```
:root { --brand:#0f766e; --space-3:.75rem; }
.button { background:var(--brand); padding:var(--space-3); }
```


Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo bảng token màu, spacing, radius.

[ ] Đã làm: Đổi --brand và quan sát toàn trang.

[ ] Đã làm: Dùng fallback var() cho token tùy chọn.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: --token là custom property; var(--name,fallback) đọc giá trị.

Tài liệu tham khảo

Tài liệu MDN — CSS custom properties, var() và design tokens Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_cascading_variables/Using_CSS_custom_properties

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Native CSS nesting và toán tử &

---

4. @layer và quản trị thứ tự ưu tiên

Cascade layers kiểm soát thứ tự stylesheet ở cấp kiến trúc trước specificity.

Mục tiêu học tập

- Giải thích được Cascade layers và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "@layer và quản trị thứ tự ưu tiên" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Cascade layers · Layer order · Unlayered rules · Important

Giải thích chi tiết

Vì sao cần học?

Cascade layers kiểm soát thứ tự stylesheet ở cấp kiến trúc trước specificity. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Cascade layers kiểm soát thứ tự stylesheet ở cấp kiến trúc trước specificity.
- Với normal declarations, layer khai báo sau thường ưu tiên cao hơn; unlayered normal styles ưu tiên hơn layered normal styles.
- Với !important, thứ tự layer đảo ngược; cần đọc kỹ.

Ví dụ để thử trên trang HTML

```
@layer reset, base, components, utilities;
@layer base { body { font-family:system-ui; } }
@layer components { .button { padding:.75rem 1rem; } }
```

Thực hành từng bước

1. Phân file reset/base/components/utilities.
2. Tạo hai selector xung đột và dự đoán layer thắng.
3. Thử một declaration !important để quan sát đảo ưu tiên.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: "@layer và quản trị thứ tự ưu tiên". Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Nghĩ tăng ID specificity luôn thắng layer khác.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ @layer và quản trị thứ tự ưu tiên

```
@layer reset, base, components, utilities;
@layer base { body { font-family:system-ui; } }
@layer components { .button { padding:.75rem 1rem; } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Phân file reset/base/components/utilities.

[ ] Đã làm: Tạo hai selector xung đột và dự đoán layer thắng.

[ ] Đã làm: Thử một declaration !important để quan sát đảo ưu tiên.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Cascade layers kiểm soát thứ tự stylesheet ở cấp kiến trúc trước specificity.

Tài liệu tham khảo

Tài liệu MDN — @layer và quản trị thứ tự ưu tiên Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Cascade_layers

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

CSS custom properties, var() và design tokens

5. BEM, @scope và CSS Modules

BEM là quy ước tên class để giảm va chạm; CSS Modules tạo tên lớp cục bộ trong build tooling.

Mục tiêu học tập

- Giải thích được BEM và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "BEM, @scope và CSS Modules" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

BEM · CSS Modules · @scope · Scoping

Giải thích chi tiết

Vì sao cần học?

BEM là quy ước tên class để giảm va chạm; CSS Modules tạo tên lớp cục bộ trong build tooling. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- BEM là quy ước tên class để giảm va chạm; CSS Modules tạo tên lớp cục bộ trong build tooling.
- @scope giới hạn vùng áp dụng theo ngữ cảnh DOM (cần kiểm tra hỗ trợ mục tiêu).
- Không công cụ nào thay thế việc hiểu cascade.

Ví dụ để thử trên trang HTML

```
.card__title { font-weight:700; }
.card--featured { border-color:gold; }
@scope (.article) { h2 { color:teal; } }
```

Thực hành từng bước

1. Tạo component card theo BEM.
2. Thử @scope trong browser mục tiêu.
3. So sánh class toàn cục và CSS Module ở dự án framework.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “BEM, @scope và CSS Modules”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Đặt selector toàn cục quá rộng hoặc tin CSS Modules tự giải quyết layout.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ BEM, @scope và CSS Modules

```
.card__title { font-weight:700; }
.card--featured { border-color:gold; }
@scope (.article) { h2 { color:teal; } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Tạo component card theo BEM.
- [ ] Đã làm: Thử @scope trong browser mục tiêu.
- [ ] Đã làm: So sánh class toàn cục và CSS Module ở dự án framework.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: BEM là quy ước tên class để giảm va chạm; CSS Modules tạo tên lớp cục bộ trong build tooling.

Tài liệu tham khảo

Tài liệu MDN — BEM, @scope và CSS Modules Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/@scope

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

@layer và quản trị thứ tự ưu tiên

GIAI ĐOẠN 8

08 · Hiệu ứng, trạng thái và trải nghiệm sử dụng

Tạo UI chuyển động vừa đủ, biểu mẫu dùng được và các thành phần tương tác dễ tiếp cận.

1. Transitions, timing functions và transform

transition tạo chuyển đổi giữa trạng thái, ví dụ hover/focus.

Mục tiêu học tập

- Giải thích được Transition và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Transitions, timing functions và transform" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Transition · Timing function · Transform · Opacity

Giải thích chi tiết

Vì sao cần học?

transition tạo chuyển đổi giữa trạng thái, ví dụ hover/focus. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- transition tạo chuyển đổi giữa trạng thái, ví dụ hover/focus.
- transform/opacity thường phù hợp hơn thay đổi kích thước layout liên tục.
- Đảm bảo trạng thái quan trọng không chỉ thể hiện bằng chuyển động.

Ví dụ để thử trên trang HTML

```
.button { transition:transform .18s ease,background-color .18s ease; }
.button:hover { transform:translateY(-2px); }
```

Thực hành từng bước

1. Tạo nút có hover transition.
2. So sánh animate top với transform.
3. Tắt motion để bảo đảm vẫn nhìn rõ trạng thái.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: "Transitions, timing functions và transform". Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- transition:all vô điều kiện hoặc animation thay thế focus indicator.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Transitions, timing functions và transform

```
.button { transition:transform .18s ease,background-color .18s ease; }
.button:hover { transform:translateY(-2px); }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo nút có hover transition.
[ ] Đã làm: So sánh animate top với transform.
[ ] Đã làm: Tắt motion để bảo đảm vẫn nhìn rõ trạng thái.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: transition tạo chuyển đổi giữa trạng thái, ví dụ hover/focus.

Tài liệu tham khảo

Tài liệu MDN — Transitions, timing functions và transform Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_transitions/Using_CSS_transitions

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

BEM, @scope và CSS Modules

---

2. @keyframes, animation và motion an toàn

@keyframes định nghĩa các mốc thay đổi; animation có duration, iteration, direction.

Mục tiêu học tập

- Giải thích được Keyframes và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "@keyframes, animation và motion an toàn" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Keyframes · Animation · Iteration · Reduced motion

Giải thích chi tiết

Vì sao cần học?

@keyframes định nghĩa các mốc thay đổi; animation có duration, iteration, direction. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- @keyframes định nghĩa các mốc thay đổi; animation có duration, iteration, direction.
- Hiệu ứng loading chỉ nên dùng khi có trạng thái loading thật.
- Tôn trọng prefers-reduced-motion và tránh nội dung chớp nháy nguy hiểm.

Ví dụ để thử trên trang HTML

```
@keyframes pulse { from { opacity:.5; } to { opacity:1; } }
.skeleton { animation:pulse 1s ease-in-out infinite alternate; }
```

Thực hành từng bước

1. Tạo skeleton loading không quá gây phân tâm.
2. Dùng DevTools inspect animation.
3. Viết override trong reduced-motion.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “@keyframes, animation và motion an toàn”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Animation vô hạn gây mất tập trung hoặc không có tín hiệu hoàn thành thật.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ @keyframes, animation và motion an toàn

```
@keyframes pulse { from { opacity:.5; } to { opacity:1; } }
.skeleton { animation:pulse 1s ease-in-out infinite alternate; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo skeleton loading không quá gây phân tâm.

[ ] Đã làm: Dùng DevTools inspect animation.

[ ] Đã làm: Viết override trong reduced-motion.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: @keyframes định nghĩa các mốc thay đổi; animation có duration, iteration, direction.

Tài liệu tham khảo

Tài liệu MDN — @keyframes, animation và motion an toàn Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animations/Using_CSS_animations

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Transitions, timing functions và transform

---

3. Styling forms, :valid/:invalid và focus

Form control cần có label HTML, CSS không thay thế cấu trúc truy cập được.

Mục tiêu học tập

- Giải thích được Form states và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Styling forms, :valid/:invalid và focus” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Form states · User-invalid · Disabled · Focus-visible

Giải thích chi tiết

Vì sao cần học?

Form control cần có label HTML, CSS không thay thế cấu trúc truy cập được. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Form control cần có label HTML, CSS không thay thế cấu trúc truy cập được.
- :required, :invalid và :focus-visible cho trạng thái; :user-invalid chỉ hiện lỗi sau tương tác ở browser hỗ trợ.
- Đừng dựa duy nhất màu đỏ để báo lỗi; cần text thông báo.

Ví dụ để thử trên trang HTML

```
input:focus-visible { outline:2px solid #2563eb; }
input:user-invalid { border-color:#b91c1c; }
input:disabled { opacity:.6; cursor:not-allowed; }
```

Thực hành từng bước

1. Style form đăng ký có input/select/button.
2. Kiểm tra Tab, error, disabled và zoom.
3. Xem lỗi có được giải thích bằng text HTML không.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Styling forms, :valid/:invalid và focus”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Loại bỏ focus ring, chỉ đổi border đỏ mà không báo lỗi.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Styling forms, :valid/:invalid và focus


```
input:focus-visible { outline:2px solid #2563eb; }
input:user-invalid { border-color:#b91c1c; }
input:disabled { opacity:.6; cursor:not-allowed; }
```


Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Style form đăng ký có input/select/button.

[ ] Đã làm: Kiểm tra Tab, error, disabled và zoom.

[ ] Đã làm: Xem lỗi có được giải thích bằng text HTML không.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Form control cần có label HTML, CSS không thay thế cấu trúc truy cập được.

Tài liệu tham khảo

Tài liệu MDN — Styling forms, :valid/:invalid và focus Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Forms/Styling_web_forms

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

@keyframes, animation và motion an toàn

---

4. Scroll behavior, scroll snap và scroll margin

scroll-margin hỗ trợ anchor không bị sticky header che.

Mục tiêu học tập

- Giải thích được Scroll-margin và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Scroll behavior, scroll snap và scroll margin" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Scroll-margin · Scroll-snap · Scroll-padding · Smooth scrolling

Giải thích chi tiết

Vì sao cần học?

scroll-margin hỗ trợ anchor không bị sticky header che. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- scroll-margin hỗ trợ anchor không bị sticky header che.
- scroll-snap có thể hữu ích cho carousel nhưng không nên khóa việc cuộn.
- Cuộn mượt phải cân nhắc reduced motion.

Ví dụ để thử trên trang HTML

```
html { scroll-padding-top:5rem; }
section { scroll-margin-top:5rem; }
```

Thực hành từng bước

1. Tạo mục lục anchor với sticky header.
2. Thử scroll-snap cho gallery ngang.
3. Kiểm tra trải nghiệm bàn phím và reduced motion.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Scroll behavior, scroll snap và scroll margin”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Buộc scroll-snap khiến người dùng không đọc được nội dung ở giữa.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Scroll behavior, scroll snap và scroll margin

```
html { scroll-padding-top:5rem; }
section { scroll-margin-top:5rem; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo mục lục anchor với sticky header.

[ ] Đã làm: Thử scroll-snap cho gallery ngang.

[ ] Đã làm: Kiểm tra trải nghiệm bàn phím và reduced motion.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: scroll-margin hỗ trợ anchor không bị sticky header che.

Tài liệu tham khảo

Tài liệu MDN — Scroll behavior, scroll snap và scroll margin Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_scroll_snap

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Styling forms, :valid/:invalid và focus

5. Filters, shadows, gradients và tính tiết chế

box-shadow/filters hỗ trợ phân cấp thị giác nhưng có thể tốn hiệu năng.

Mục tiêu học tập

- Giải thích được Filter và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Filters, shadows, gradients và tính tiết chế" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Filter · Backdrop-filter · Shadow · Gradient

Giải thích chi tiết

Vì sao cần học?

box-shadow/filters hỗ trợ phân cấp thị giác nhưng có thể tốn hiệu năng. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- box-shadow/filters hỗ trợ phân cấp thị giác nhưng có thể tốn hiệu năng.
- backdrop-filter cần cân nhắc tương phản, fallback và hỗ trợ.
- Dùng pseudo-element cho trang trí thay nhiều DOM thừa.

Ví dụ để thử trên trang HTML

```
.glass { background:rgb(255 255 255 / .85); }
@supports (backdrop-filter:blur(12px)) { .glass { backdrop-filter:blur(12px); } }
```

Thực hành từng bước

1. Tạo banner với shadow và gradient.
2. Đo hiệu ứng blur trong Performance.
3. Tắt backdrop-filter và xem chữ còn dễ đọc không.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Filters, shadows, gradients và tính tiết chế”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Lạm dụng blur/filter lớn làm chữ khó đọc và tăng chi phí render.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Filters, shadows, gradients và tính tiết chế

```
.glass { background:rgb(255 255 255 / .85); }
@supports (backdrop-filter:blur(12px)) { .glass { backdrop-filter:blur(12px); } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Tạo banner với shadow và gradient.

[ ] Đã làm: Đo hiệu ứng blur trong Performance.

[ ] Đã làm: Tắt backdrop-filter và xem chữ còn dễ đọc không.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: box-shadow/filters hỗ trợ phân cấp thị giác nhưng có thể tốn hiệu năng.

Tài liệu tham khảo

Tài liệu MDN — Filters, shadows, gradients và tính tiết chế Tài liệu
https://developer.mozilla.org/en-US/docs/Web/CSS/backdrop-filter

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Scroll behavior, scroll snap và scroll margin

GIAI ĐOẠN 9

09 · Accessibility, hiệu năng và CSS production

Đánh giá chất lượng trước khi giao diện đi vào sản phẩm thật.

1. CSS accessibility: contrast, focus, zoom và reflow

WCAG yêu cầu nhiều tiêu chí về độ tương phản, focus và reflow; không thể đạt chỉ bằng CSS.

Mục tiêu học tập

- Giải thích được WCAG và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "CSS accessibility: contrast, focus, zoom và reflow" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

WCAG · Focus indicator · Reflow · Contrast · Keyboard

Giải thích chi tiết

Vì sao cần học?

WCAG yêu cầu nhiều tiêu chí về độ tương phản, focus và reflow; không thể đạt chỉ bằng CSS. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- WCAG yêu cầu nhiều tiêu chí về độ tương phản, focus và reflow; không thể đạt chỉ bằng CSS.
- Kiểm tra zoom 200%, 400%, keyboard, chế độ tương phản cao khi có.
- display:none/visibility:hidden có tác động khả năng truy cập; cần dùng đúng ngữ nghĩa.

Ví dụ để thử trên trang HTML

```
a:focus-visible { outline:3px solid #1d4ed8; outline-offset:3px; }
.skip-link:focus { position:static; }
```

Thực hành từng bước

1. Kiểm tra một trang chỉ bằng bàn phím.
2. Đo contrast từng nút, placeholder, body text.
3. Zoom 400% và kiểm tra reflow, scroll ngang.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “CSS accessibility: contrast, focus, zoom và reflow”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Nghĩ color contrast đủ là đạt toàn bộ accessibility.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ CSS accessibility: contrast, focus, zoom và reflow

```
a:focus-visible { outline:3px solid #1d4ed8; outline-offset:3px; }
.skip-link:focus { position:static; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Kiểm tra một trang chỉ bằng bàn phím.
- [ ] Đã làm: Đo contrast từng nút, placeholder, body text.
- [ ] Đã làm: Zoom 400% và kiểm tra reflow, scroll ngang.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: WCAG yêu cầu nhiều tiêu chí về độ tương phản, focus và reflow; không thể đạt chỉ bằng CSS.

Tài liệu tham khảo

Tài liệu MDN — CSS accessibility: contrast, focus, zoom và reflow Tài liệu
https://www.w3.org/WAI/WCAG22/quickref/

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Filters, shadows, gradients và tính tiết chế

2. Hiệu năng CSS và render pipeline

Trình duyệt có các bước style, layout, paint, composite; một thay đổi có thể kích hoạt bước khác nhau.

Mục tiêu học tập

- Giải thích được Render pipeline và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Hiệu năng CSS và render pipeline" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Render pipeline · Layout · Paint · Composite · Core Web Vitals

Giải thích chi tiết

Vì sao cần học?

Trình duyệt có các bước style, layout, paint, composite; một thay đổi có thể kích hoạt bước khác nhau. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Trình duyệt có các bước style, layout, paint, composite; một thay đổi có thể kích hoạt bước khác nhau.
- Giảm layout thrashing do JavaScript và animation không cần thiết.
- Theo dõi LCP/CLS/INP; CSS góp phần nhưng không phải yếu tố duy nhất.

Ví dụ để thử trên trang HTML

```
.animatable { transform:translate3d(0,0,0); }
/* Dùng transform có mục đích, không lạm dụng will-change */
```

Thực hành từng bước

1. Ghi trace trong Performance khi hover.
2. Thay animation width bằng transform rồi so sánh.
3. Kiểm tra font/ảnh làm CLS và ghi số đo.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Hiệu năng CSS và render pipeline”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng will-change cho mọi element hoặc cho rằng CSS nhỏ là luôn nhanh.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Hiệu năng CSS và render pipeline

```
.animatable { transform:translate3d(0,0,0); }
/* Dùng transform có mục đích, không lạm dụng will-change */
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Ghi trace trong Performance khi hover.
- [ ] Đã làm: Thay animation width bằng transform rồi so sánh.
- [ ] Đã làm: Kiểm tra font/ảnh làm CLS và ghi số đo.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: Trình duyệt có các bước style, layout, paint, composite; một thay đổi có thể kích hoạt bước khác nhau.

Tài liệu tham khảo

Tài liệu MDN — Hiệu năng CSS và render pipeline Tài liệu
https://web.dev/learn/performance/

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

CSS accessibility: contrast, focus, zoom và reflow

---

3. Debug lỗi cascade, overflow và layout

Đi từ DOM matched styles computed box model layout inspector.

Mục tiêu học tập

- Giải thích được Debugging và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Debug lỗi cascade, overflow và layout” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Debugging · Computed styles · Grid inspector · Overflow

Giải thích chi tiết

Vì sao cần học?

Đi từ DOM matched styles computed box model layout inspector. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Đi từ DOM matched styles computed box model layout inspector.
- Horizontal overflow thường do fixed width, min-width, nội dung dài, viewport unit, hoặc transform.
- Phân biệt lỗi CSS với HTML sai cấu trúc.

Ví dụ để thử trên trang HTML

```
* { /* chỉ dùng tạm để debug */ outline: 1px solid rgb(255 0 0 / .12); }
```

Thực hành từng bước

1. Sửa 3 lỗi: card tràn, sticky không dính, button không nhận màu.
2. Ghi giả thuyết trước khi sửa.
3. Dùng DevTools xác định đúng declaration gây lỗi.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Debug lỗi cascade, overflow và layout”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Fix bằng !important, overflow-x:hidden hay z-index vô hạn mà không tìm gốc.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Debug lỗi cascade, overflow và layout

```
* { /* chỉ dùng tạm để debug */ outline: 1px solid rgb(255 0 0 / .12); }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Sửa 3 lỗi: card tràn, sticky không dính, button không nhận màu.
[ ] Đã làm: Ghi giả thuyết trước khi sửa.
[ ] Đã làm: Dùng DevTools xác định đúng declaration gây lỗi.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: Đi từ DOM matched styles computed box model layout inspector.

Tài liệu tham khảo

Tài liệu MDN — Debug lỗi cascade, overflow và layout Tài liệu
https://developer.chrome.com/docs/devtools/css/

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Hiệu năng CSS và render pipeline

---

4. Stylelint, formatting và tổ chức stylesheet

Formatter giữ phong cách nhất quán; linter phát hiện quy tắc đáng nghi.

Mục tiêu học tập

- Giải thích được Stylelint và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Stylelint, formatting và tổ chức stylesheet” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Stylelint · Prettier · PostCSS · Sass · CSS Modules · Tailwind

Giải thích chi tiết

Vì sao cần học?

Formatter giữ phong cách nhất quán; linter phát hiện quy tắc đáng nghi. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Formatter giữ phong cách nhất quán; linter phát hiện quy tắc đáng nghi.
- CSS Modules, Sass, PostCSS và Tailwind là công cụ tùy bối cảnh, không thay nền tảng CSS.
- Tránh phụ thuộc framework trước khi tự làm layout cơ bản.

Ví dụ để thử trên trang HTML

```
/* architecture example */
@import url("./tokens.css");
@import url("./components.css");
```

Thực hành từng bước

1. Thiết lập Prettier và Stylelint cho repo demo.
2. Chạy lint trên file có lỗi cố ý.
3. Đánh giá khi nào nên dùng CSS thuần, Modules hay utility framework.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Stylelint, formatting và tổ chức stylesheet”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Cài nhiều công cụ mà không có lý do hoặc coi Sass là CSS browser chạy trực tiếp.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Stylelint, formatting và tổ chức stylesheet

```
/* architecture example */
@import url("./tokens.css");
@import url("./components.css");
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Thiết lập Prettier và Stylelint cho repo demo.
[ ] Đã làm: Chạy lint trên file có lỗi cố ý.
[ ] Đã làm: Đánh giá khi nào nên dùng CSS thuần, Modules hay utility framework.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Formatter giữ phong cách nhất quán; linter phát hiện quy tắc đáng nghi.

Tài liệu tham khảo

Tài liệu MDN — Stylelint, formatting và tổ chức stylesheet Tài liệu
https://stylelint.io/user-guide/get-started

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Debug lỗi cascade, overflow và layout

---

5. Hỗ trợ trình duyệt, Baseline và regression

Các tính năng CSS mới cần kiểm tra hỗ trợ theo browser target và thời điểm sản phẩm.

Mục tiêu học tập

- Giải thích được Browser compatibility và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Hỗ trợ trình duyệt, Baseline và regression" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Browser compatibility · Baseline · Regression · Fallback

Giải thích chi tiết

Vì sao cần học?

Các tính năng CSS mới cần kiểm tra hỗ trợ theo browser target và thời điểm sản phẩm. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Các tính năng CSS mới cần kiểm tra hỗ trợ theo browser target và thời điểm sản phẩm.
- Progressive enhancement bảo đảm trải nghiệm cốt lõi hoạt động khi nâng cao không sẵn.
- Visual regression so sánh ảnh nhiều kích thước, nhưng vẫn cần test tương tác và accessibility.

Ví dụ để thử trên trang HTML

```
@supports (display:grid) { .layout { display:grid; } }
```

Thực hành từng bước

1. Lập browser support matrix cho dự án.
2. Kiểm tra :has(), container queries và @scope theo mục tiêu.
3. Chụp UI 3 viewport trước/sau khi refactor.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Hỗ trợ trình duyệt, Baseline và regression”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Cho rằng chạy trên Chrome mới nhất đồng nghĩa chạy ở mọi nơi.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Hỗ trợ trình duyệt, Baseline và regression

```
@supports (display:grid) { .layout { display:grid; } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Lập browser support matrix cho dự án.
[ ] Đã làm: Kiểm tra :has(), container queries và @scope theo mục tiêu.
[ ] Đã làm: Chụp UI 3 viewport trước/sau khi refactor.
[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
[ ] Tự trả lời được: Các tính năng CSS mới cần kiểm tra hỗ trợ theo browser target và thời điểm sản phẩm.

Tài liệu tham khảo

Tài liệu MDN — Hỗ trợ trình duyệt, Baseline và regression Tài liệu
https://web.dev/baseline

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Stylelint, formatting và tổ chức stylesheet

GIAI ĐOẠN 10

10 · Dự án tích hợp và hướng học tiếp

Áp dụng kiến thức vào sản phẩm kiểm thử được, sử dụng AI mà không đánh mất năng lực CSS nền tảng.

1. Dự án 1: Landing page responsive hoàn chỉnh

Kết hợp typography, hệ spacing, hero, CTA, card, footer và mobile-first.

Mục tiêu học tập

- Giải thích được Landing page và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Dự án 1: Landing page responsive hoàn chỉnh" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Landing page · Mobile-first · Typography scale · Responsive QA

Giải thích chi tiết

Vì sao cần học?

Kết hợp typography, hệ spacing, hero, CTA, card, footer và mobile-first. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Kết hợp typography, hệ spacing, hero, CTA, card, footer và mobile-first.
- Tránh phụ thuộc animation để truyền thông tin.
- Tiêu chí hoàn thành: không tràn ngang, keyboard focus rõ, contrast đạt mục tiêu.

Ví dụ để thử trên trang HTML

```
.landing { width:min(100% - 2rem,72rem); margin-inline:auto; }
```

Thực hành từng bước

1. Thiết kế landing một dịch vụ giả định với 5 section.
2. Làm responsive từ 320px đến desktop và kiểm tra zoom.
3. Ghi screenshot và checklist accessibility, performance.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Dự án 1: Landing page responsive hoàn chỉnh”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Chỉ dựng bản desktop đẹp nhưng mobile không dùng được.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Dự án 1: Landing page responsive hoàn chỉnh

```
.landing { width:min(100% - 2rem,72rem); margin-inline:auto; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Thiết kế landing một dịch vụ giả định với 5 section.
- [ ] Đã làm: Làm responsive từ 320px đến desktop và kiểm tra zoom.
- [ ] Đã làm: Ghi screenshot và checklist accessibility, performance.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: Kết hợp typography, hệ spacing, hero, CTA, card, footer và mobile-first.

Tài liệu tham khảo

Tài liệu MDN — Dự án 1: Landing page responsive hoàn chỉnh Tài liệu
https://web.dev/learn/design/

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Hỗ trợ trình duyệt, Baseline và regression

---

2. Dự án 2: Dashboard dùng Grid + Flex + container queries

Trang dashboard có sidebar, stats, bảng, card responsive.

Mục tiêu học tập

- Giải thích được Dashboard và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài “Dự án 2: Dashboard dùng Grid + Flex + container queries” trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Dashboard · Grid · Flexbox · Container queries · Overflow

Giải thích chi tiết

Vì sao cần học?

Trang dashboard có sidebar, stats, bảng, card responsive. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Trang dashboard có sidebar, stats, bảng, card responsive.
- Grid tổ chức bố cục trang; Flex căn nội dung trong component; container queries cho card tái sử dụng.
- Cần xử lý overflow bảng và trạng thái điều hướng focus.

Ví dụ để thử trên trang HTML

```
.dashboard { display:grid; grid-template-columns:15rem minmax(0,1fr); gap:1rem; }
@media (max-width:48rem) { .dashboard { grid-template-columns:1fr; } }
```

Thực hành từng bước

1. Dựng dashboard không dùng framework CSS.
2. Tạo stat card dùng được ở sidebar và main.
3. Test bảng rộng, chữ dài, màn nhỏ, keyboard navigation.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Dự án 2: Dashboard dùng Grid + Flex + container queries”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng Grid order thay thứ tự DOM hoặc che bảng bằng overflow hidden.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Dự án 2: Dashboard dùng Grid + Flex + container queries

```
.dashboard { display:grid; grid-template-columns:15rem minmax(0,1fr); gap:1rem; }
@media (max-width:48rem) { .dashboard { grid-template-columns:1fr; } }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Dựng dashboard không dùng framework CSS.

[ ] Đã làm: Tạo stat card dùng được ở sidebar và main.

[ ] Đã làm: Test bảng rộng, chữ dài, màn nhỏ, keyboard navigation.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Trang dashboard có sidebar, stats, bảng, card responsive.

Tài liệu tham khảo

Tài liệu MDN — Dự án 2: Dashboard dùng Grid + Flex + container queries Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Dự án 1: Landing page responsive hoàn chỉnh

---

3. Dự án 3: Bộ component và design tokens

Tạo token màu, font, spacing; các component button/card/input/alert và các trạng thái.

Mục tiêu học tập

- Giải thích được Design tokens và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Dự án 3: Bộ component và design tokens" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Design tokens · Component states · Cascade layers · Theming

Giải thích chi tiết

Vì sao cần học?

Tạo token màu, font, spacing; các component button/card/input/alert và các trạng thái. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Tạo token màu, font, spacing; các component button/card/input/alert và các trạng thái.
- Dùng @layer, custom properties và naming để giảm xung đột.
- Mỗi component có ví dụ trạng thái, disabled, focus và theme.

Ví dụ để thử trên trang HTML

```
@layer base,components,utilities;
:root { --radius: .75rem; --accent:#0f766e; }
```

Thực hành từng bước

1. Xây style guide có ít nhất 6 component.
2. Thêm light/dark mode và reduced motion.
3. Ghi chú các trường hợp thiếu tương phản hoặc focus.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Dự án 3: Bộ component và design tokens”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Có đẹp một trạng thái nhưng thiếu hover/focus/error/disabled.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Dự án 3: Bộ component và design tokens

```
@layer base,components,utilities;
:root { --radius: .75rem; --accent:#0f766e; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Xây style guide có ít nhất 6 component.

[ ] Đã làm: Thêm light/dark mode và reduced motion.

[ ] Đã làm: Ghi chú các trường hợp thiếu tương phản hoặc focus.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Tạo token màu, font, spacing; các component button/card/input/alert và các trạng

thái.

Tài liệu tham khảo

Tài liệu MDN — Dự án 3: Bộ component và design tokens Tài liệu
https://web.dev/learn/design/

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Dự án 2: Dashboard dùng Grid + Flex + container queries

---

4. Làm việc với AI: yêu cầu, review và bảo vệ chất lượng CSS

Giao AI tác vụ nhỏ, nêu viewport, HTML hiện có và tiêu chí kiểm thử.

Mục tiêu học tập

- Giải thích được AI assisted CSS và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Làm việc với AI: yêu cầu, review và bảo vệ chất lượng CSS" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

AI assisted CSS · Review · Verification · Prompt · Testing

Giải thích chi tiết

Vì sao cần học?

Giao AI tác vụ nhỏ, nêu viewport, HTML hiện có và tiêu chí kiểm thử. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Giao AI tác vụ nhỏ, nêu viewport, HTML hiện có và tiêu chí kiểm thử.
- Kiểm soát CSS sinh ra: specificity, selector toàn cục, !important, browser support và accessibility.
- Không đưa dữ liệu cá nhân/secrets vào prompt; kiểm tra diff/test trước khi nhận.

Ví dụ để thử trên trang HTML

```
/* Review AI-generated CSS: cascade, responsive, focus, reduced-motion */
```

Thực hành từng bước

1. Viết prompt tạo 1 component từ HTML có sẵn.
2. Yêu cầu AI giải thích từng quyết định layout.
3. Tự đọc diff, thử breakpoint và refactor phần gây lỗi.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: "Làm việc với AI: yêu cầu, review và bảo vệ chất lượng CSS". Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Dùng AI sinh 1000 dòng CSS rồi chấp nhận mà không đọc.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Làm việc với AI: yêu cầu, review và bảo vệ chất lượng CSS

```
/* Review AI-generated CSS: cascade, responsive, focus, reduced-motion */
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

[ ] Đã làm: Viết prompt tạo 1 component từ HTML có sẵn.

[ ] Đã làm: Yêu cầu AI giải thích từng quyết định layout.

[ ] Đã làm: Tự đọc diff, thử breakpoint và refactor phần gây lỗi.

[ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.

[ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.

[ ] Tự trả lời được: Giao AI tác vụ nhỏ, nêu viewport, HTML hiện có và tiêu chí kiểm thử.

Tài liệu tham khảo

Tài liệu MDN — Làm việc với AI: yêu cầu, review và bảo vệ chất lượng CSS Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Dự án 3: Bộ component và design tokens

5. Capstone: Website nhiều trang CSS hiện đại và báo cáo QA

Sản phẩm tối thiểu 3 trang với design system, mobile-first, responsive images và component tái sử dụng.

Mục tiêu học tập

- Giải thích được Capstone và vai trò của nó trong giao diện thực tế
- Viết và thử được ví dụ CSS của bài "Capstone: Website nhiều trang CSS hiện đại và báo cáo QA" trong trình duyệt
- Tự tìm được nguyên nhân lỗi bằng DevTools và sửa mà không dùng cách vá mù quáng
- Kiểm tra được responsive, khả năng dùng bàn phím và giới hạn hỗ trợ phù hợp với bài học

Kiến thức cần học

Capstone · QA · Responsive · Accessibility · Documentation

Giải thích chi tiết

Vì sao cần học?

Sản phẩm tối thiểu 3 trang với design system, mobile-first, responsive images và component tái sử dụng. Đây là năng lực quan trọng khi xây giao diện thật: có thể giải thích điều gì xảy ra trong trình duyệt, thay vì chỉ sao chép một đoạn CSS chạy tình cờ.

Kiến thức cốt lõi

- Sản phẩm tối thiểu 3 trang với design system, mobile-first, responsive images và component tái sử dụng.
- Có checklist từ DevTools, kiểm tra accessibility, trình duyệt, hiệu năng, screenshots trước/sau.
- Bàn giao README giải thích kiến trúc CSS và những giới hạn chưa giải quyết.

Ví dụ để thử trên trang HTML

```
.site-shell { width:min(100% - 2rem,72rem); margin-inline:auto; }
```

Thực hành từng bước

1. Xây 3 trang: trang chủ, danh sách, chi tiết/đăng ký.
2. Đạt checklist ở 320px, 768px, 1440px, zoom 200%, keyboard.
3. Viết README và báo cáo bug đã sửa, bảo đảm không dựa duy nhất vào đánh giá AI.

Tư duy ứng dụng trong dự án

Trước khi viết mã, hãy xác định nội dung HTML, bố cục mong muốn, trạng thái tương tác và kích thước vùng chứa. Thử trường hợp nội dung dài, màn hình hẹp, zoom lớn và điều hướng bàn phím. So sánh kết quả thật trong DevTools với dự đoán ban đầu.

Làm việc với AI có kiểm chứng

Có thể nhờ AI: Giải thích cơ chế, gợi ý hai cách triển khai khác nhau, tạo bộ dữ liệu giả để test và review đoạn CSS bạn tự viết.

Tự kiểm chứng: Mở trình duyệt, kiểm tra Computed và layout inspector, so sánh với MDN và thử ít nhất ba kích thước viewport. Kiểm tra các trạng thái focus/hover/reduced motion khi liên quan.

Không giao toàn quyền: Không chấp nhận !important, z-index cực lớn, overflow:hidden hoặc lối tắt bỏ focus chỉ vì AI nói là đã sửa. Không đưa dữ liệu nhạy cảm vào công cụ ngoài.

Prompt mẫu:

> Tôi đang học CSS: “Capstone: Website nhiều trang CSS hiện đại và báo cáo QA”. Hãy dùng kiến thức MDN để giải thích cơ chế qua một ví dụ nhỏ; hỏi tôi dự đoán kết quả trước khi giải. Sau đó review CSS tôi gửi theo cascade, responsive, accessibility và hỗ trợ trình duyệt. Đừng tự thêm framework.

Sai lầm cần tránh

- Nộp screenshot đẹp nhưng thiếu source, kiểm thử và khả năng tái hiện.
- Chỉ kiểm tra một viewport hoặc tin vào mô tả mà không chạy mã.

Điều kiện hoàn thành

Tự thực hiện các bước thực hành, giải thích vì sao CSS chạy như vậy và chứng minh kết quả qua DevTools hoặc ảnh kiểm thử. Nêu được ít nhất một giới hạn hay trường hợp đặc biệt của kỹ thuật đã học.

Ví dụ minh họa

Ví dụ Capstone: Website nhiều trang CSS hiện đại và báo cáo QA

```
.site-shell { width:min(100% - 2rem,72rem); margin-inline:auto; }
```

Dán ví dụ vào một trang HTML thử nghiệm, quan sát Styles/Computed trong DevTools và thay đổi từng thuộc tính.

Checklist

- [ ] Đã làm: Xây 3 trang: trang chủ, danh sách, chi tiết/đăng ký.
- [ ] Đã làm: Đạt checklist ở 320px, 768px, 1440px, zoom 200%, keyboard.
- [ ] Đã làm: Viết README và báo cáo bug đã sửa, bảo đảm không dựa duy nhất vào đánh giá AI.
- [ ] Đã kiểm tra CSS trên viewport nhỏ và lớn, nội dung dài và zoom.
- [ ] Đã đối chiếu ví dụ với MDN hoặc nguồn chuẩn.
- [ ] Tự trả lời được: Sản phẩm tối thiểu 3 trang với design system, mobile-first, responsive images và component tái sử dụng.

Tài liệu tham khảo

Tài liệu MDN — Capstone: Website nhiều trang CSS hiện đại và báo cáo QA Tài liệu
https://developer.mozilla.org/en-US/docs/Learn_web_development

CSS Roadmap 2026 — roadmap.sh Website
https://roadmap.sh/css

Learn CSS — web.dev Khóa học
https://web.dev/learn/css

Nên học trước

Làm việc với AI: yêu cầu, review và bảo vệ chất lượng CSS