Giáo trình CSS nâng cao

Mục tiêu

- Cung cấp kiến thức sâu về các kỹ thuật CSS nâng cao, giúp người học xây dựng giao diện web hiện đại, tối ưu và dễ bảo trì.
- Giải thích chi tiết cách hoạt động của từng kỹ thuật, kèm ví dụ thực tế và ứng dụng trong dự án.
- Hỗ trợ người học áp dụng CSS vào các dự án thực tế, từ giao diện đơn giản đến phức tạp.

Đối tượng

- Sinh viên hoặc lập trình viên đã nắm vững CSS cơ bản (selector, box model, media queries).
- Người muốn nâng cao kỹ năng để làm việc với các dự án lớn, tích hợp với JavaScript frameworks, hoặc theo xu hướng thiết kế hiện đại (2025).

---

1. BEM (Block, Element, Modifier)

Định nghĩa

BEM là một phương pháp đặt tên class trong CSS, giúp tổ chức mã theo cách có cấu trúc, dễ đọc, và dễ bảo trì. BEM chia giao diện thành các khối độc lập (Block), các phần tử con (Element), và các biến thể/trạng thái (Modifier).

Cách hoạt động

- **Block**: Là một thành phần độc lập, có thể tái sử dụng (ví dụ: một nút, một card). Block không phụ thuộc vào ngữ cảnh.
- **Element**: Là phần tử con trong block, chỉ có ý nghĩa khi nằm trong block. Ký hiệu bằng `__` (ví dụ: `.card__title`).
- **Modifier**: Thay đổi giao diện hoặc trạng thái của block/element, ký hiệu bằng `--` (ví dụ: `.button--disabled`).
- **Cơ chế**: BEM sử dụng tên class dài, mô tả rõ ràng cấu trúc và chức năng, giúp tránh xung đột tên class trong dự án lớn. Trình duyệt xử lý các class BEM như class thông thường, nhưng cấu trúc tên giúp lập trình viên dễ dàng quản lý và mở rộng mã.
- **Lưu ý**:
  - Không lồng class quá sâu để tránh mã phức tạp.
  - Modifier thường chỉ áp dụng cho block hoặc element, không tạo modifier riêng lẻ.

Cú pháp

```
.block { /* Kiểu dáng cho block */ }
.block__element { /* Kiểu dáng cho element */ }
.block--modifier { /* Kiểu dáng cho block với trạng thái/biến thể */ }
.block__element--modifier { /* Kiểu dáng cho element với trạng thái/biến thể */ }
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một card với tiêu đề và nội dung, sử dụng BEM.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>BEM Basic Example</title>
  <style>
    .card {
      padding: 20px;
      background-color: #f9f9f9;
      border-radius: 8px;
    }
    .card__title {
      font-size: 1.5rem;
      color: #333;
    }
    .card__content {
      font-size: 1rem;
      color: #666;
    }
  </style>
</head>
<body>
  <div class="card">
    <h2 class="card__title">Tiêu đề</h2>
    <p class="card__content">Nội dung card đơn giản.</p>
  </div>
</body>
</html>
```

Ví dụ nâng cao 1: Card với trạng thái nổi bật

Tạo một card với trạng thái "featured" sử dụng modifier.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>BEM Advanced Example 1</title>
  <style>
    .card {
      padding: 20px;
      background-color: #f9f9f9;
      border-radius: 8px;
      box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    }
    .card__title {
      font-size: 1.5rem;
      color: #333;
      margin-bottom: 10px;
```

```
    }
    .card__content {
      font-size: 1rem;
      color: #666;
    }
    .card--featured {
      border: 2px solid #007bff;
      background-color: #e3f2fd;
    }
    .card__title--large {
      font-size: 2rem;
    }
  </style>
</head>
<body>
  <div class="card card--featured">
    <h2 class="card__title card__title--large">Tiêu đề nổi bật</h2>
    <p class="card__content">Nội dung của card nổi bật.</p>
  </div>
  <div class="card">
    <h2 class="card__title">Tiêu đề thường</h2>
    <p class="card__content">Nội dung card thường.</p>
  </div>
</body>
</html>
```

Ví dụ nâng cao 2: Menu điều hướng với BEM

Tạo một menu điều hướng với trạng thái active.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>BEM Advanced Example 2</title>
  <style>
    .nav {
      display: flex;
      gap: 10px;
      padding: 10px;
      background-color: #333;
    }
    .nav__item {
      padding: 10px 20px;
      color: white;
      text-decoration: none;
      border-radius: 5px;
    }
    .nav__item--active {
      background-color: #007bff;
      font-weight: bold;
    }
    .nav__item:hover {
```

```
      background-color: #555;
    }
  </style>
</head>
<body>
  <nav class="nav">
    <a href="#" class="nav__item">Trang chủ</a>
    <a href="#" class="nav__item nav__item--active">Giới thiệu</a>
    <a href="#" class="nav__item">Liên hệ</a>
  </nav>
</body>
</html>
```

Ứng dụng thực tế

- **Dự án nhóm**: BEM giúp nhiều lập trình viên làm việc trên cùng một codebase mà không gây xung đột.
- **E-commerce**: Tạo các component như card sản phẩm, menu điều hướng, hoặc form nhập liệu với cấu trúc rõ ràng.
- **Design systems**: Xây dựng thư viện giao diện với các thành phần tái sử dụng, như button, card, modal.

Bài tập

1. Tạo một giao diện nút (button) với BEM, có 2 trạng thái: "primary" và "disabled".
2. Xây dựng một form đăng nhập với các trường input và nút submit, sử dụng BEM để đặt tên class.

---

2. CSS Grid Layout

Định nghĩa

CSS Grid là một hệ thống bố cục 2D mạnh mẽ, cho phép tạo lưới với các hàng và cột linh hoạt, giúp thiết kế giao diện phức tạp mà không cần framework.

Cách hoạt động

- **Container**: Phần tử cha được đặt `display: grid`, định nghĩa cấu trúc lưới qua `grid-template-columns`, `grid-template-rows`, và `gap`.
- **Items**: Các phần tử con được đặt trong lưới, có thể chỉ định vị trí bằng `grid-column`, `grid-row`, hoặc `grid-area`.
- **Cơ chế**: Trình duyệt tính toán kích thước và vị trí của các ô lưới dựa trên thuộc tính được định nghĩa. Các hàm như `repeat()`, `minmax()`, `auto-fit` giúp lưới tự động điều chỉnh theo nội dung hoặc kích thước container.
- **Lưu ý**:
  - CSS Grid phù hợp cho bố cục toàn trang hoặc các thành phần lớn.
  - Kết hợp với media queries để tạo giao diện responsive.

Cú pháp

```
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr); /* 3 cột đều nhau */
```

```
  grid-template-rows: auto; /* Hàng tự động */
  gap: 20px; /* Khoảng cách giữa các ô */
}
.grid-item {
  grid-column: 1 / 3; /* Chiếm từ cột 1 đến cột 3 */
  grid-row: 1 / 2; /* Chiếm hàng 1 */
}
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một lưới 3 cột đơn giản.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Grid Basic Example</title>
  <style>
    .grid-container {
      display: grid;
      grid-template-columns: repeat(3, 1fr);
      gap: 15px;
      padding: 20px;
    }
    .grid-item {
      background-color: #e0f7fa;
      padding: 15px;
      text-align: center;
      border-radius: 5px;
    }
  </style>
</head>
<body>
  <div class="grid-container">
    <div class="grid-item">Item 1</div>
    <div class="grid-item">Item 2</div>
    <div class="grid-item">Item 3</div>
  </div>
</body>
</html>
```

Ví dụ nâng cao 1: Bố cục trang web

Tạo bố cục trang web với header, sidebar, main, và footer.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Grid Advanced Example 1</title>
  <style>
```

```
    .grid-container {
      display: grid;
      grid-template-columns: 200px 1fr;
      grid-template-rows: 80px 1fr 60px;
      gap: 10px;
      height: 100vh;
      grid-template-areas:
        "header header"
        "sidebar main"
        "footer footer";
    }
    .header { grid-area: header; background-color: #333; color: white;
padding: 20px; }
    .sidebar { grid-area: sidebar; background-color: #f0f0f0; padding:
20px; }
    .main { grid-area: main; background-color: #e0f7fa; padding: 20px; }
    .footer { grid-area: footer; background-color: #333; color: white;
padding: 20px; }
  </style>
</head>
<body>
  <div class="grid-container">
    <header class="header">Header</header>
    <aside class="sidebar">Sidebar</aside>
    <main class="main">Nội dung chính</main>
    <footer class="footer">Footer</footer>
  </div>
</body>
</html>
```

Ví dụ nâng cao 2: Grid responsive với auto-fit

Tạo một gallery ảnh responsive.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Grid Advanced Example 2</title>
  <style>
    .gallery {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
      gap: 15px;
      padding: 20px;
    }
    .gallery__item {
      background-color: #b2ebf2;
      padding: 10px;
      border-radius: 5px;
      text-align: center;
    }
    .gallery__item img {
```

```
      max-width: 100%;
      border-radius: 5px;
    }
  </style>
</head>
<body>
  <div class="gallery">
    <div class="gallery__item"><img src="https://via.placeholder.com/200"
alt="Image 1"></div>
    <div class="gallery__item"><img src="https://via.placeholder.com/200"
alt="Image 2"></div>
    <div class="gallery__item"><img src="https://via.placeholder.com/200"
alt="Image 3"></div>
    <div class="gallery__item"><img src="https://via.placeholder.com/200"
alt="Image 4"></div>
  </div>
</body>
</html>
```

Ứng dụng thực tế

- **Dashboard quản trị**: Sắp xếp các widget (biểu đồ, bảng dữ liệu) trong bố cục linh hoạt.
- **E-commerce**: Hiển thị danh sách sản phẩm theo lưới responsive.
- **Portfolio**: Tạo gallery ảnh hoặc dự án với bố cục masonry.

Bài tập

1. Tạo một lưới 4x2 với các ô có màu nền khác nhau, sử dụng `minmax()` để giới hạn chiều rộng.
2. Xây dựng một trang web với bố cục gồm header, 2 cột nội dung, và footer, sử dụng `grid-template-areas`.

---

3. Flexbox nâng cao

Định nghĩa

Flexbox là hệ thống bố cục 1D, lý tưởng để sắp xếp các phần tử trong một hàng hoặc cột, với khả năng kiểm soát linh hoạt thứ tự, kích thước, và căn chỉnh.

Cách hoạt động

- **Container**: Phần tử cha được đặt `display: flex`, xác định hướng (`flex-direction`), căn chỉnh ngang (`justify-content`), và căn chỉnh dọc (`align-items`).
- **Items**: Phần tử con có thể điều chỉnh kích thước (`flex-grow`, `flex-shrink`), thứ tự (`order`), hoặc căn chỉnh riêng lẻ (`align-self`).
- **Cơ chế**: Trình duyệt phân bổ không gian dựa trên thuộc tính flex, tự động điều chỉnh khi kích thước container thay đổi.
- **Lưu ý**:
  - Flexbox phù hợp cho bố cục tuyến tính (hàng/cột), không lý tưởng cho lưới 2D.
  - Kết hợp với media queries để tạo bố cục responsive.

Cú pháp

```
.flex-container {
  display: flex;
  flex-direction: row; /* Hoặc column */
  justify-content: space-between; /* Căn chỉnh ngang */
  align-items: center; /* Căn chỉnh dọc */
  gap: 10px;
}
.flex-item {
  flex: 1; /* flex-grow: 1, flex-shrink: 1, flex-basis: auto */
  order: 2; /* Thứ tự hiển thị */
}
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một hàng flex với 3 phần tử căn đều.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Flexbox Basic Example</title>
  <style>
    .flex-container {
      display: flex;
      justify-content: space-between;
      gap: 10px;
      padding: 20px;
    }
    .flex-item {
      flex: 1;
      background-color: #ffccbc;
      padding: 10px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">Item 1</div>
    <div class="flex-item">Item 2</div>
    <div class="flex-item">Item 3</div>
  </div>
</body>
</html>
```

Ví dụ nâng cao 1: Navigation bar responsive

Tạo một menu điều hướng responsive với Flexbox.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Flexbox Advanced Example 1</title>
  <style>
    .nav {
      display: flex;
      justify-content: space-around;
      align-items: center;
      background-color: #333;
      padding: 10px;
    }
    .nav__item {
      color: white;
      text-decoration: none;
      padding: 10px 20px;
      border-radius: 5px;
    }
    .nav__item:hover {
      background-color: #555;
    }
    .nav__item--active {
      background-color: #007bff;
    }
    @media (max-width: 600px) {
      .nav {
        flex-direction: column;
        gap: 10px;
      }
    }
  </style>
</head>
<body>
  <nav class="nav">
    <a href="#" class="nav__item">Trang chủ</a>
    <a href="#" class="nav__item nav__item--active">Giới thiệu</a>
    <a href="#" class="nav__item">Liên hệ</a>
  </nav>
</body>
</html>
```

Ví dụ nâng cao 2: Card layout với tỷ lệ linh hoạt

Tạo một hàng card với tỷ lệ co giãn khác nhau.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Flexbox Advanced Example 2</title>
  <style>
    .card-container {
```

```
      display: flex;
      gap: 15px;
      padding: 20px;
    }
    .card {
      background-color: #e0f7fa;
      padding: 15px;
      border-radius: 5px;
      text-align: center;
    }
    .card--large {
      flex: 2; /* Chiếm gấp đôi không gian */
    }
    .card--small {
      flex: 1;
    }
  </style>
</head>
<body>
  <div class="card-container">
    <div class="card card--small">Card nhỏ 1</div>
    <div class="card card--large">Card lớn</div>
    <div class="card card--small">Card nhỏ 2</div>
  </div>
</body>
</html>
```

Ứng dụng thực tế

- **Navigation bar**: Tạo menu ngang hoặc dọc, responsive trên mobile.
- **Card layout**: Sắp xếp danh sách bài viết, sản phẩm, hoặc hình ảnh.
- **Form UI**: Căn chỉnh các trường input và nút trong form.

Bài tập

1. Tạo một hàng flex với 4 phần tử, căn giữa và có khoảng cách 15px, thay đổi thành cột trên màn hình nhỏ.
2. Tạo một footer với 3 cột (thông tin, liên hệ, mạng xã hội), sử dụng Flexbox để phân bổ không gian linh hoạt.

---

4. Container Queries

Định nghĩa

Container Queries cho phép áp dụng kiểu dáng dựa trên kích thước của container cha, thay vì kích thước viewport (như media queries), giúp thiết kế component-based linh hoạt hơn.

Cách hoạt động

- **Container**: Phần tử cha được khai báo `container-type` (thường là `inline-size` hoặc `size`) để xác định phạm vi query.

- **Query**: Sử dụng `@container` để áp dụng kiểu dáng khi container đáp ứng điều kiện kích thước (ví dụ: `max-width`, `min-width`).
- **Cơ chế**: Trình duyệt kiểm tra kích thước của container (không phải viewport) và áp dụng kiểu dáng tương ứng. Điều này cho phép các component tự điều chỉnh độc lập.
- **Lưu ý**:
  - Container Queries được hỗ trợ tốt từ Chrome/Edge 105, Firefox 110 (2023 trở đi).
  - Phù hợp với các ứng dụng có nhiều component lồng nhau.

Cú pháp

```
.container {
  container-type: inline-size; /* Hoặc size */
}
@container (max-width: 300px) {
  .child {
    /* Kiểu dáng khi container nhỏ hơn 300px */
  }
}
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một card thay đổi kiểu dáng dựa trên kích thước container.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Container Queries Basic Example</title>
  <style>
    .container {
      container-type: inline-size;
      width: 400px;
      padding: 10px;
      background-color: #f0f0f0;
    }
    .card {
      background-color: #b2ebf2;
      padding: 15px;
      font-size: 16px;
    }
    @container (max-width: 300px) {
      .card {
        font-size: 14px;
        background-color: #ffccbc;
      }
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="card">Nội dung card</div>
```

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

Ví dụ nâng cao 1: Card với bố cục thay đổi

Tạo một card thay đổi bố cục (ngang/dọc) dựa trên kích thước container.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Container Queries Advanced Example 1</title>
  <style>
    .container {
      container-type: inline-size;
      width: 500px;
      padding: 10px;
      background-color: #f0f0f0;
    }
    .card {
      display: flex;
      gap: 10px;
      background-color: #b2ebf2;
      padding: 15px;
    }
    .card__image {
      width: 100px;
      height: 100px;
      background-color: #ccc;
    }
    .card__content {
      font-size: 16px;
    }
    @container (max-width: 400px) {
      .card {
        flex-direction: column;
        align-items: center;
      }
      .card__image {
        width: 80px;
        height: 80px;
      }
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="card">
      <div class="card__image"></div>
      <div class="card__content">Nội dung card thay đổi bố cục</div>
    </div>
  </div>
```

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

Ví dụ nâng cao 2: Nhiều container với kích thước khác nhau

Tạo hai container với các card điều chỉnh kiểu dáng khác nhau.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Container Queries Advanced Example 2</title>
  <style>
    .container--large {
      container-type: inline-size;
      width: 600px;
      padding: 10px;
      background-color: #f0f0f0;
      margin-bottom: 20px;
    }
    .container--small {
      container-type: inline-size;
      width: 300px;
      padding: 10px;
      background-color: #f0f0f0;
    }
    .card {
      background-color: #b2ebf2;
      padding: 15px;
      font-size: 16px;
      text-align: center;
    }
    @container (max-width: 400px) {
      .card {
        font-size: 14px;
        background-color: #ffccbc;
        padding: 10px;
      }
    }
  </style>
</head>
<body>
  <div class="container--large">
    <div class="card">Card trong container lớn</div>
  </div>
  <div class="container--small">
    <div class="card">Card trong container nhỏ</div>
  </div>
</body>
</html>
```

Ứng dụng thực tế

- **Dashboard**: Các widget tự điều chỉnh dựa trên kích thước container cha (ví dụ: sidebar hoặc main content).
- **E-commerce**: Card sản phẩm thay đổi bố cục (hình ảnh, văn bản) tùy thuộc vào không gian có sẵn.
- **Component libraries**: Tạo các component tái sử dụng trong React/Vue với kiểu dáng linh hoạt.

Bài tập

1. Tạo một container với card bên trong, thay đổi kích thước chữ và màu nền khi container nhỏ hơn 350px.
2. Tạo một giao diện với 3 container có kích thước khác nhau, mỗi container chứa một card với bố cục thay đổi qua Container Queries.

---

5. CSS Animations

Định nghĩa

CSS Animations cho phép tạo các hiệu ứng chuyển động mượt mà bằng cách sử dụng `@keyframes` để định nghĩa các trạng thái thay đổi của phần tử.

Cách hoạt động

- **@keyframes**: Xác định các trạng thái của hoạt ảnh (0%, 100%, hoặc các bước trung gian).
- **Animation properties**: Bao gồm `animation-name`, `animation-duration`, `animation-timing-function`, `animation-iteration-count`, v.v.
- **Cơ chế**: Trình duyệt tính toán và render các trạng thái chuyển đổi dựa trên `@keyframes`. Sử dụng `transform` và `opacity` để tối ưu hiệu suất, vì chúng không gây reflow (tái tính toán bố cục).
- **Lưu ý**:
    - Tránh sử dụng các thuộc tính như `width`, `margin` trong animation vì gây tốn tài nguyên.
    - Sử dụng `will-change` để báo trước cho trình duyệt tối ưu hóa.

Cú pháp

```
@keyframes animation-name {
  0% { /* Trạng thái ban đầu */ }
  50% { /* Trạng thái giữa */ }
  100% { /* Trạng thái cuối */ }
}
.element {
  animation: animation-name 2s ease infinite;
}
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một hình vuông di chuyển qua lại.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
```

```
  <title>CSS Animation Basic Example</title>
  <style>
    .box {
      width: 100px;
      height: 100px;
      background-color: #ff5722;
      animation: slide 3s ease-in-out infinite;
      will-change: transform;
    }
    @keyframes slide {
      0% { transform: translateX(0); }
      50% { transform: translateX(200px); }
      100% { transform: translateX(0); }
    }
  </style>
</head>
<body>
  <div class="box"></div>
</body>
</html>
```

Ví dụ nâng cao 1: Loading spinner

Tạo một vòng tròn xoay liên tục.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Animation Advanced Example 1</title>
  <style>
    .spinner {
      width: 50px;
      height: 50px;
      border: 5px solid #f0f0f0;
      border-top: 5px solid #007bff;
      border-radius: 50%;
      animation: spin 1s linear infinite;
      will-change: transform;
    }
    @keyframes spin {
      0% { transform: rotate(0deg); }
      100% { transform: rotate(360deg); }
    }
  </style>
</head>
<body>
  <div class="spinner"></div>
</body>
</html>
```

Ví dụ nâng cao 2: Hiệu ứng cuộn xuất hiện

Tạo hiệu ứng các phần tử xuất hiện khi cuộn trang.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Animation Advanced Example 2</title>
  <style>
    .section {
      height: 100vh;
      display: flex;
      justify-content: center;
      align-items: center;
      background-color: #f0f0f0;
    }
    .fade-in {
      opacity: 0;
      transform: translateY(50px);
      animation: fadeIn 1s ease forwards;
      animation-delay: 0.5s;
    }
    @keyframes fadeIn {
      0% { opacity: 0; transform: translateY(50px); }
      100% { opacity: 1; transform: translateY(0); }
    }
  </style>
</head>
<body>
  <div class="section">
    <h2 class="fade-in">Chào mừng đến với trang web</h2>
  </div>
  <div class="section">
    <p class="fade-in">Nội dung xuất hiện khi cuộn</p>
  </div>
</body>
</html>
```

Ứng dụng thực tế

- **Micro-interactions**: Tạo hiệu ứng hover trên nút, loading spinners, hoặc chuyển đổi trạng thái.
- **Storytelling websites**: Hiệu ứng xuất xuất hiện khi cuộn để tăng trải nghiệm người dùng.
- **Game UI**: Tạo hoạt ảnh cho các thành phần giao diện như nút, nhân vật.

Bài tập

1. Tạo một hình tròn nhấp nháy (thay đổi opacity) liên tục.
2. Tạo một card với hiệu ứng zoom-in khi hover, sử dụng `transform: scale`.

---

6. CSS Custom Properties (Variables)

Định nghĩa

CSS Custom Properties (biến CSS) là các giá trị được lưu trữ trong biến (bắt đầu bằng `--`) để tái sử dụng, giúp quản lý màu sắc, kích thước, và các thuộc tính khác dễ dàng.

Cách hoạt động

- **Khai báo**: Biến được định nghĩa trong `:root` (toàn cục) hoặc trong một phạm vi cụ thể (ví dụ: một class).
- **Sử dụng**: Truy xuất biến bằng `var(--variable-name)`.
- **Cơ chế**: Trình duyệt thay thế giá trị biến vào thuộc tính tương ứng khi render. Biến có thể được cập nhật động (qua JavaScript) hoặc kế thừa trong phạm vi.
- **Lưu ý**:
  - Biến CSS được kế thừa từ phần tử cha, trừ khi bị ghi đè.
  - Kết hợp với `calc()` để tạo giá trị động.

Cú pháp

```
:root {
  --variable-name: value;
}
.element {
  property: var(--variable-name);
}
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một hộp sử dụng biến CSS cho màu nền và padding.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Variables Basic Example</title>
  <style>
    :root {
      --primary-color: #2196f3;
      --padding: 15px;
    }
    .box {
      background-color: var(--primary-color);
      padding: var(--padding);
      color: white;
      border-radius: 5px;
    }
  </style>
</head>
<body>
  <div class="box">Hộp sử dụng biến CSS</div>
</body>
</html>
```

Ví dụ nâng cao 1: Theme switching

Tạo một giao diện với khả năng chuyển đổi theme sáng/tối.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Variables Advanced Example 1</title>
  <style>
    :root {
      --bg-color: #ffffff;
      --text-color: #333333;
    }
    [data-theme="dark"] {
      --bg-color: #333333;
      --text-color: #ffffff;
    }
    body {
      background-color: var(--bg-color);
      color: var(--text-color);
      transition: all 0.3s;
    }
    .container {
      padding: 20px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Chuyển đổi theme</h1>
    <button onclick="document.body.setAttribute('data-theme',
'dark')">Dark Theme</button>
    <button onclick="document.body.removeAttribute('data-theme')">Light
Theme</button>
  </div>
</body>
</html>
```

Ví dụ nâng cao 2: Responsive padding với calc()

Tạo một card với padding thay đổi dựa trên viewport.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Variables Advanced Example 2</title>
  <style>
    :root {
      --base-padding: 10px;
    }
    .card {
      background-color: #e0f7fa;
```

```
      padding: calc(var(--base-padding) * 2);
      border-radius: 5px;
      text-align: center;
    }
    @media (max-width: 600px) {
      :root {
        --base-padding: 5px;
      }
    }
  </style>
</head>
<body>
  <div class="card">Card với padding responsive</div>
</body>
</html>
```

Ứng dụng thực tế

- **Theme switching**: Chuyển đổi giao diện sáng/tối trong các ứng dụng web.
- **Design systems**: Quản lý màu sắc, font, khoảng cách trong các dự án lớn.
- **Responsive design**: Tạo các giá trị động (kích thước, khoảng cách) dựa trên viewport hoặc trạng thái.

Bài tập

1. Tạo một giao diện với 3 nút, sử dụng biến CSS để quản lý màu nền và thay đổi khi hover.
2. Tạo một card với padding và font-size sử dụng `calc()` và biến CSS, thay đổi dựa trên media query.

---

7. Glassmorphism

Định nghĩa

Glassmorphism là xu hướng thiết kế tạo hiệu ứng kính mờ trong suốt, sử dụng `backdrop-filter: blur` và màu nền `rgba` để tạo giao diện hiện đại, sang trọng.

Cách hoạt động

- **Background**: Sử dụng `rgba` để tạo màu nền trong suốt, cho phép nhìn xuyên qua.
- **Backdrop-filter**: Áp dụng hiệu ứng mờ (blur) cho nền phía sau phần tử, tạo cảm giác kính mờ.
- **Cơ chế**: Trình duyệt render lớp nền trong suốt và áp dụng bộ lọc (blur) cho khu vực phía sau phần tử. Viền mỏng và bóng đổ nhẹ thường được thêm để tăng tính thẩm mỹ.
- **Lưu ý**:
  - Hỗ trợ tốt trên Chrome, Edge, Safari (2023 trở đi), nhưng cần kiểm tra tính tương thích.
  - Đảm bảo tỷ lệ tương phản để văn bản dễ đọc.

Cú pháp

```
.glass {
  background: rgba(255, 255, 255, 0.2);
  backdrop-filter: blur(10px);
  border: 1px solid rgba(255, 255, 255, 0.3);
```

```
  border-radius: 10px;
}
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một card với hiệu ứng glassmorphism.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Glassmorphism Basic Example</title>
  <style>
    body {
      background: url('https://via.placeholder.com/500') center/cover;
    }
    .glass-card {
      background: rgba(255, 255, 255, 0.2);
      backdrop-filter: blur(10px);
      padding: 20px;
      border-radius: 10px;
      border: 1px solid rgba(255, 255, 255, 0.3);
      color: white;
      max-width: 300px;
      margin: 20px;
    }
  </style>
</head>
<body>
  <div class="glass-card">
    <h2>Card Kính Mờ</h2>
    <p>Nội dung với hiệu ứng glassmorphism.</p>
  </div>
</body>
</html>
```

Ví dụ nâng cao 1: Header glassmorphism

Tạo một header cố định với hiệu ứng glassmorphism.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Glassmorphism Advanced Example 1</title>
  <style>
    body {
      background: url('https://via.placeholder.com/1000') center/cover;
      margin: 0;
    }
    .header {
```

```
      position: fixed;
      top: 0;
      width: 100%;
      background: rgba(255, 255, 255, 0.1);
      backdrop-filter: blur(12px);
      padding: 15px;
      display: flex;
      justify-content: space-between;
      align-items: center;
      border-bottom: 1px solid rgba(255, 255, 255, 0.3);
    }
    .header__logo {
      color: white;
      font-size: 1.5rem;
    }
    .header__nav a {
      color: white;
      margin-left: 20px;
      text-decoration: none;
    }
    .header__nav a:hover {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <header class="header">
    <div class="header__logo">Logo</div>
    <nav class="header__nav">
      <a href="#">Trang chủ</a>
      <a href="#">Giới thiệu</a>
      <a href="#">Liên hệ</a>
    </nav>
  </header>
</body>
</html>
```

Ví dụ nâng cao 2: Card với hover effect

Tạo một card glassmorphism với hiệu ứng hover.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Glassmorphism Advanced Example 2</title>
  <style>
    body {
      background: url('https://via.placeholder.com/1000') center/cover;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
```

```
      margin: 0;
    }
    .card {
      background: rgba(255, 255, 255, 0.2);
      backdrop-filter: blur(10px);
      padding: 20px;
      border-radius: 10px;
      border: 1px solid rgba(255, 255, 255, 0.3);
      color: white;
      max-width: 300px;
      transition: backdrop-filter 0.3s, transform 0.3s;
    }
    .card:hover {
      backdrop-filter: blur(20px);
      transform: scale(1.05);
    }
  </style>
</head>
<body>
  <div class="card">
    <h2>Card Hover</h2>
    <p>Hiệu ứng glassmorphism khi hover.</p>
  </div>
</body>
</html>
```

Ứng dụng thực tế

- **UI hiện đại**: Tạo card sản phẩm, modal, hoặc dashboard với giao diện trong suốt.
- **Portfolio cá nhân**: Tăng tính thẩm mỹ cho header hoặc section nổi bật.
- **Ứng dụng mobile**: Hiệu ứng kính mờ trong các ứng dụng web progressive (PWA).

Bài tập

1. Tạo một modal với hiệu ứng glassmorphism, căn giữa màn hình.
2. Tạo một danh sách 3 card glassmorphism, thêm hiệu ứng hover thay đổi độ blur và scale.

---

8. Accessible CSS

Định nghĩa

Accessible CSS là cách sử dụng CSS để đảm bảo giao diện web dễ tiếp cận với người dùng khuyết tật, tuân theo các tiêu chuẩn như WCAG (Web Content Accessibility Guidelines).

Cách hoạt động

- **Tỷ lệ tương phản**: Đảm bảo màu chữ và nền có tỷ lệ tương phản tối thiểu (4.5:1 cho văn bản thường).
- **Focus states**: Sử dụng `:focus` và `:focus-visible` để hiển thị trạng thái focus rõ ràng cho người dùng bàn phím.
- **Reduced motion**: Sử dụng `prefers-reduced-motion` để tắt hoặc giảm hoạt ảnh cho người nhạy cảm với chuyển động.

- **Cơ chế**: Trình duyệt kiểm tra cài đặt hệ thống của người dùng (như chế độ high contrast, reduced motion) và áp dụng kiểu dáng phù hợp.
- **Lưu ý**:
  - Kiểm tra tương phản bằng công cụ như WebAIM Contrast Checker.
  - Đảm bảo focus states không bị che khuất bởi các hiệu ứng khác.

Cú pháp

```
.element {
  background-color: #007bff;
  color: white; /* Tỷ lệ tương phản cao */
}
.element:focus {
  outline: 2px solid #ffff00;
  outline-offset: 2px;
}
@media (prefers-reduced-motion: reduce) {
  .element {
    transition: none; /* Tắt hoạt ảnh */
  }
}
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một nút với focus state và hỗ trợ reduced motion.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Accessible CSS Basic Example</title>
  <style>
    .button {
      background-color: #007bff;
      color: white;
      padding: 10px 20px;
      border: none;
      border-radius: 5px;
      transition: background-color 0.3s;
    }
    .button:hover {
      background-color: #0056b3;
    }
    .button:focus {
      outline: 2px solid #ffff00;
      outline-offset: 2px;
    }
    @media (prefers-reduced-motion: reduce) {
      .button {
        transition: none;
      }
```

```
    }
  </style>
</head>
<body>
  <button class="button">Click me</button>
</body>
</html>
```

Ví dụ nâng cao 1: Form với focus states

Tạo một form với các trường input có focus states rõ ràng.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Accessible CSS Advanced Example 1</title>
  <style>
    .form {
      display: flex;
      flex-direction: column;
      gap: 10px;
      max-width: 300px;
      padding: 20px;
    }
    .form__input {
      padding: 10px;
      border: 2px solid #ccc;
      border-radius: 5px;
    }
    .form__input:focus {
      border-color: #007bff;
      outline: 2px solid #ffff00;
      outline-offset: 2px;
    }
    .form__button {
      background-color: #007bff;
      color: white;
      padding: 10px;
      border: none;
      border-radius: 5px;
      transition: background-color 0.3s;
    }
    .form__button:hover {
      background-color: #0056b3;
    }
    .form__button:focus {
      outline: 2px solid #ffff00;
      outline-offset: 2px;
    }
    @media (prefers-reduced-motion: reduce) {
      .form__button {
        transition: none;
```

```
      }
    }
  </style>
</head>
<body>
  <form class="form">
    <input type="text" class="form__input" placeholder="Tên">
    <input type="email" class="form__input" placeholder="Email">
    <button class="form__button">Gửi</button>
  </form>
</body>
</html>
```

Ví dụ nâng cao 2: High contrast mode

Tạo một card hỗ trợ chế độ high contrast.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Accessible CSS Advanced Example 2</title>
  <style>
    .card {
      background-color: #ffffff;
      color: #000000;
      padding: 20px;
      border: 2px solid #000000;
      border-radius: 5px;
      max-width: 300px;
    }
    .card__title {
      font-size: 1.5rem;
    }
    .card__content {
      font-size: 1rem;
    }
    @media (prefers-contrast: high) {
      .card {
        background-color: #000000;
        color: #ffffff;
        border-color: #ffffff;
      }
    }
  </style>
</head>
<body>
  <div class="card">
    <h2 class="card__title">Tiêu đề</h2>
    <p class="card__content">Nội dung hỗ trợ high contrast.</p>
  </div>
</body>
</html>
```

Ứng dụng thực tế

- **Website công cộng**: Đảm bảo người khiếm thị hoặc nhạy cảm với chuyển động có thể sử dụng.
- **Ứng dụng doanh nghiệp**: Tăng khả năng tiếp cận cho người dùng đa dạng.
- **E-learning platforms**: Hỗ trợ người học với các nhu cầu đặc biệt.

Bài tập

1. Tạo một form với 2 trường input và 1 nút, thêm focus states rõ ràng với outline màu vàng.
2. Tạo một card hỗ trợ `prefers-contrast: high` và `prefers-reduced-motion`.

---

9. CSS-in-JS

Định nghĩa

CSS-in-JS là cách viết CSS trong JavaScript, thường sử dụng các thư viện như Styled-Components để quản lý kiểu dáng theo component, tránh xung đột tên class.

Cách hoạt động

- **Khai báo kiểu dáng**: CSS được viết trong template literals (JavaScript) và gắn với component.
- **Scope cục bộ**: Mỗi component có kiểu dáng riêng, không ảnh hưởng đến phần tử khác.
- **Cơ chế**: Thư viện CSS-in-JS (như Styled-Components) tạo ra các class duy nhất cho mỗi component khi render, đảm bảo không xung đột. Có thể sử dụng props để tạo kiểu dáng động.
- **Lưu ý**:
  - Phù hợp với các framework như React, Vue.
  - Có thể làm tăng thời gian tải nếu không tối ưu hóa.

Cú pháp

```
import styled from 'styled-components';

const StyledDiv = styled.div`
  background-color: #e0f7fa;
  padding: 20px;
  border-radius: 8px;
`;
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một component card với Styled-Components.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS-in-JS Basic Example</title>
```

```
  <script
src="https://cdn.jsdelivr.net/npm/react@17/umd/react.development.js"></scr
ipt>
  <script src="https://cdn.jsdelivr.net/npm/react-dom@17/umd/react-
dom.development.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/styled-
components@5/dist/styled-components.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script>
    const { styled } = window.styled;
    const Card = styled.div`
      background-color: #e0f7fa;
      padding: 20px;
      border-radius: 8px;
      text-align: center;
    `;
    ReactDOM.render(<Card>Card CSS-in-JS</Card>,
document.getElementById('root'));
  </script>
</body>
</html>
```

Ví dụ nâng cao 1: Button với props động

Tạo một nút với màu nền thay đổi dựa trên props.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS-in-JS Advanced Example 1</title>
  <script
src="https://cdn.jsdelivr.net/npm/react@17/umd/react.development.js"></scr
ipt>
  <script src="https://cdn.jsdelivr.net/npm/react-dom@17/umd/react-
dom.development.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/styled-
components@5/dist/styled-components.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script>
    const { styled } = window.styled;
    const Button = styled.button`
      background-color: ${props => props.primary ? '#007bff' : '#6c757d'};
      color: white;
      padding: 10px 20px;
      border: none;
      border-radius: 5px;
      &:hover {
        opacity: 0.8;
```

```
      }
    `;
    ReactDOM.render(
      <>
        <Button primary>Primary Button</Button>
        <Button>Secondary Button</Button>
      </>,
      document.getElementById('root')
    );
  </script>
</body>
</html>
```

Ví dụ nâng cao 2: Card với theme

Tạo một card sử dụng theme trong Styled-Components.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS-in-JS Advanced Example 2</title>
  <script
src="https://cdn.jsdelivr.net/npm/react@17/umd/react.development.js"></scr
ipt>
  <script src="https://cdn.jsdelivr.net/npm/react-dom@17/umd/react-
dom.development.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/styled-
components@5/dist/styled-components.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script>
    const { styled, ThemeProvider } = window.styled;
    const Card = styled.div`
      background-color: ${props => props.theme.bgColor};
      color: ${props => props.theme.textColor};
      padding: 20px;
      border-radius: 8px;
      text-align: center;
    `;
    const lightTheme = { bgColor: '#e0f7fa', textColor: '#333' };
    const darkTheme = { bgColor: '#333', textColor: '#fff' };
    ReactDOM.render(
      <ThemeProvider theme={lightTheme}>
        <Card>Card với theme sáng</Card>
      </ThemeProvider>,
      document.getElementById('root')
    );
  </script>
</body>
</html>
```

Ứng dụng thực tế

- **React/Vue apps**: Quản lý kiểu dáng cho các component độc lập.
- **Single-page applications**: Tăng tốc phát triển với kiểu dáng gắn liền component.
- **Dynamic UI**: Thay đổi kiểu dáng dựa trên trạng thái hoặc dữ liệu người dùng.

Bài tập

1. Tạo một component React với Styled-Components, định dạng một nút với 2 trạng thái (primary, secondary) dựa trên props.
2. Tạo một card sử dụng ThemeProvider để hỗ trợ theme sáng/tối.

---

10. CSS Nesting

Định nghĩa

CSS Nesting cho phép viết CSS lồng nhau trực tiếp trong file CSS, tương tự Sass, giúp mã ngắn gọn và dễ đọc hơn mà không cần preprocessor.

Cách hoạt động

- **Nesting**: Các kiểu dáng con được viết trong khối của phần tử cha, sử dụng `&` để tham chiếu đến cha.
- **Cơ chế**: Trình duyệt phân tích các quy tắc lồng nhau và chuyển thành các selector thông thường khi render. Điều này giảm sự lặp lại và tăng tính tổ chức.
- **Lưu ý**:
  - Hỗ trợ tốt trên Chrome 112, Firefox 117 (2023 trở đi).
  - Tránh lồng quá sâu để không làm tăng độ phức tạp selector.

Cú pháp

```
.parent {
  /* Kiểu dáng của parent */
  & .child {
    /* Kiểu dáng của child */
  }
  &:hover {
    /* Kiểu dáng khi hover */
  }
}
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một card với CSS Nesting.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
```

```
  <title>CSS Nesting Basic Example</title>
  <style>
    .card {
      padding: 20px;
      background-color: #f0f0f0;
      border-radius: 8px;
      & .title {
        font-size: 1.5rem;
        color: #333;
      }
      &:hover {
        background-color: #e0e0e0;
      }
    }
  </style>
</head>
<body>
  <div class="card">
    <h2 class="title">Tiêu đề</h2>
    <p>Nội dung card</p>
  </div>
</body>
</html>
```

Ví dụ nâng cao 1: Menu với nesting

Tạo một menu điều hướng với hiệu ứng hover.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Nesting Advanced Example 1</title>
  <style>
    .nav {
      display: flex;
      gap: 10px;
      padding: 10px;
      background-color: #333;
      & .nav__item {
        color: white;
        text-decoration: none;
        padding: 10px 20px;
        border-radius: 5px;
        &:hover {
          background-color: #555;
        }
        &--active {
          background-color: #007bff;
          font-weight: bold;
        }
      }
    }
```

```
  </style>
</head>
<body>
  <nav class="nav">
    <a href="#" class="nav__item">Trang chủ</a>
    <a href="#" class="nav__item nav__item--active">Giới thiệu</a>
    <a href="#" class="nav__item">Liên hệ</a>
  </nav>
</body>
</html>
```

Ví dụ nâng cao 2: Card với hiệu ứng phức tạp

Tạo một card với hiệu ứng hover và pseudo-elements.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>CSS Nesting Advanced Example 2</title>
  <style>
    .card {
      padding: 20px;
      background-color: #e0f7fa;
      border-radius: 8px;
      position: relative;
      & .title {
        font-size: 1.5rem;
        color: #333;
      }
      &:hover {
        transform: translateY(-5px);
        box-shadow: 0 4px 8px rgba(0,0,0,0.2);
        &::after {
          content: '';
          position: absolute;
          bottom: 0;
          left: 0;
          width: 100%;
          height: 4px;
          background-color: #007bff;
        }
      }
    }
  </style>
</head>
<body>
  <div class="card">
    <h2 class="title">Tiêu đề</h2>
    <p>Nội dung card với hiệu ứng</p>
  </div>
</body>
</html>
```

Ứng dụng thực tế

- **Dự án lớn**: Giảm sự lặp lại trong mã CSS, tăng tính dễ đọc.
- **Component styling**: Định dạng các component có cấu trúc lồng nhau (như
- **Thay thế Sass**: Sử dụng native CSS nesting trong các dự án không muốn dùng preprocessor.

Bài tập

1. Tạo một menu với 4 mục, sử dụng CSS Nesting để định dạng và thêm hiệu ứng `hover`.
2. Tạo một card với tiêu đề và nội dung, sử dụng CSS Nesting để thêm pseudo-elements (`::before`, `::after`) khi hover.

Dự án cuối khóa

Yêu cầu

Xây dựng một trang portfolio cá nhân sử dụng ít nhất 5 chủ đề trên:

- **BEM**: Đặt tên class cho tất cả các thành phần (header, card, button).
- **CSS Grid**: Sử dụng cho bố cục chính (header, main, footer).
- **Glassmorphism**: Áp dụng cho header hoặc card.
- **Container Queries**: Điều chỉnh bố cục card dựa trên kích thước container.
- **CSS Animations**: Thêm hiệu ứng xuất hiện khi cuộn hoặc hover.
- Đảm bảo khả năng tiếp cận với **Accessible CSS** (focus states, tương phản).

Hướng dẫn nộp bài

- Nộp file HTML, CSS, và một file README giải thích cách áp dụng các chủ đề.
- Giao diện phải responsive và hoạt động tốt trên desktop và mobile.

Tài nguyên bổ sung

- **MDN Web Docs**: Tài liệu chi tiết về CSS (developer.mozilla.org).
- **CSS Tricks**: Bài viết chuyên sâu về CSS nâng cao (css-tricks.com).
- **CanIUse**: Kiểm tra tính tương thích trình duyệt (caniuse.com).
- **CodePen**: Thực hành và chia sẻ mã (codepen.io).