BỘ THÔNG TIN VÀ TRUYỀN THÔNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

TÌM HIỂU VỀ HTML, CSS, JAVASCRIPT

ĐƠN VỊ: CÔNG TY CỔ PHẦN VCCORP

Giáo viên hướng dẫn: TS. Nguyễn Quang Hưng

Sinh viên : Nguyễn Quốc Đại - B21DCCN025

Phụ lục

CHƯƠNG 1: GIỚI THIỆU HTML, CSS, JAVASCRIPT

1.1. Giới thiệu chung

HTML, CSS, và JavaScript là ba trụ cột không thể thiếu trong phát triển web hiện đại, mỗi thành phần đóng một vai trò riêng biệt nhưng lại bổ trợ hoàn hảo cho nhau để tạo nên một trải nghiệm trực tuyến toàn diện. HTML là ngôn ngữ đánh dấu, đóng vai trò như bộ xương và cấu trúc của một trang web, định nghĩa nội dung và ý nghĩa của các thành phần như tiêu đề, đoạn văn, hình ảnh hay liên kết. Sau khi HTML dựng lên khung sườn, CSS sẽ được áp dụng để trang trí và tạo kiểu, quyết định giao diện trực quan của trang, từ màu sắc, phông chữ, kích thước cho đến bố cục tổng thể. Cuối cùng, JavaScript mang đến sự sống và tính tương tác cho trang web, biến những nội dung tĩnh thành các yếu tố động, có khả năng phản hồi lại hành động của người dùng, xử lý dữ liệu và tạo ra trải nghiệm phong phú hơn.

CHƯƠNG 2. TÌM HIỂU VỀ HTML

2.1. Giới thiệu về HTML.

2.1.1. HTML là gì?

- HTML là viết tắt của HyperText Markup Language — tức là Ngôn ngữ đánh dấu siêu văn bản.
- HTML là ngôn ngữ chính dùng để xây dựng cấu trúc của các trang web. Nó không phải là ngôn ngữ lập trình (như JavaScript), mà là ngôn ngữ đánh dấu, giúp trình duyệt hiểu được nội dung và cách hiển thị nội dung trên một trang web.

2.1.2. Lịch sử của HTML.

- Vào cuối năm 1991, phiên bản HTML đầu tiên do Tim Berners-Lee phát triển đã được công khai với tên HTML Tags. Phiên bản này có thiết kế vô cùng đơn giản, mô tả 18 phần tử. Tiếp đến vào năm 1995, IETF đã hoàn thành "HTML 2.0".

- Sau đó phiên bản HTML 4.01 được công bố vào năm 1999. Đến năm 2000, các phiên bản HTML đã được các nhà phát triển thay thế bằng XHTML. Năm 2014, HTML được nâng cấp lên chuẩn HTML5 với sự cải tiến rõ rệt. Điều này được thể hiện trong việc đã có nhiều tag được thêm vào markup để giúp xác định rõ nội dung thuộc thể loại gì.

2.1.3. Vai trò của HTML.

2.1.3.1. Tạo nội dung cho trang web

Nếu trang web của bạn muốn hiển thị nội dung cho người truy cập thì sẽ phải cần đến HTML. HTML cho phép trang web của bạn có thể lưu trữ âm thanh, video, văn bản và một số ứng dụng khác.

2.1.3.2. Thiết kế giao diện cho trang web

HTML5 hiện đang là phiên bản HTML mới nhất với những cải tiến rõ rệt so với các phiên bản trước. Bởi vậy hiện nay, mọi người thường sử dụng HTML5 để thiết kế giao diện website.
Tuy nhiên, bạn chỉ có thể dùng HTML để tạo bộ khung sườn cho trang web mà thôi. Để thiết kế hoàn thiện một trang web, bạn cần sử dụng đến CSS để chỉnh sửa màu sắc, kích thước, vị trí của các biểu tượng và một số vấn đề phức tạp khác. Bên cạnh đó, bạn còn cần phải có kiến thức chuyên môn về lập trình.

2.1.3.3. Lập trình tương tác cho trang web

Thông qua HTML, bạn có thể lập trình tương tác giữa người dùng với trang web. Để làm được điều này, bạn cần dùng code JavaScript. JavaScript sẽ tạo ra những hiệu ứng khi người dùng nhấp và di chuyển chuột trên website.

2.1.4. Cách hoạt động của HTML với trình duyệt

Khi người dùng truy cập một trang web bằng trình duyệt (như Chrome, Firefox, Safari...), HTML là thành phần đầu tiên được tải về và xử lý. Quá trình hoạt động của HTML với trình duyệt diễn ra theo các bước chính như sau:

- **Gửi yêu cầu đến máy chủ**: Khi bạn nhập địa chỉ trang web vào thanh địa chỉ và nhấn Enter, trình duyệt sẽ gửi một yêu cầu HTTP hoặc HTTPS đến máy chủ chứa trang web đó.
- **Nhận tệp HTML từ máy chủ**: Máy chủ tiếp nhận yêu cầu và gửi lại một tệp HTML – thường là index.html, chứa nội dung và cấu trúc cơ bản của trang web.
- **Trình duyệt phân tích HTML (parsing)**: Sau khi nhận được tệp HTML, trình duyệt sẽ đọc mã HTML từ trên xuống dưới. Khi gặp các thẻ khác như <link> (CSS), <script> (JavaScript), hoặc <img>, trình duyệt sẽ gửi thêm yêu cầu để tải về các tệp liên quan.
- **Xây dựng cây DOM (Document Object Model)**: Trong quá trình phân tích HTML, trình duyệt đồng thời xây dựng một mô hình cây gọi là DOM, mô phỏng cấu trúc của trang web. DOM là nền tảng để JavaScript tương tác và thay đổi nội dung động của trang.
- **Kết hợp với CSS và JavaScript**: Trình duyệt sử dụng HTML để tạo cấu trúc, CSS để xử lý giao diện và định dạng hiển thị (như màu sắc, kích thước, bố cục), và JavaScript để thêm chức năng tương tác (như nút bấm, kéo thả, hiệu ứng...).
- **Hiển thị trang web cho người dùng (rendering)**: Cuối cùng, trình duyệt sẽ "vẽ" nội dung của trang web lên màn hình theo thông tin từ HTML, CSS và JavaScript. Người dùng sẽ thấy một trang web hoàn chỉnh với nội dung, hình ảnh, màu sắc, và chức năng tương tác như mong muốn.

2.2. Các loại thẻ trong HTML.

2.2.1. Thẻ cấu trúc cơ bản của trang

Những thẻ này định hình "bộ khung" của một tài liệu HTML:

- <html>: Là thẻ gốc bao ngoài toàn bộ tài liệu HTML. Mọi nội dung trong trang web đều nằm bên trong thẻ này.
- <head>: Chứa các thông tin "ẩn" không hiển thị trực tiếp lên giao diện, như tiêu đề, bộ mã hóa ký tự, CSS, JavaScript, favicon...
- <title>: Đặt tiêu đề của trang web, nội dung này sẽ hiển thị trên tab trình duyệt.
- <meta>: Dùng để khai báo thông tin meta như charset, mô tả trang, từ khóa SEO, tác giả...
- <body>: Chứa toàn bộ nội dung thực tế của trang web – những gì người dùng nhìn thấy trên trình duyệt (văn bản, hình ảnh, nút bấm...).

2.2.2. Thẻ nội dung văn bản

Các thẻ này dùng để trình bày và định nghĩa nội dung dạng văn bản:

- <h1> đến <h6>: Thẻ tiêu đề, <h1> là quan trọng nhất (tiêu đề chính), <h6> là nhỏ nhất. Giúp chia bố cục nội dung hợp lý và hỗ trợ SEO.
- <p>: Tạo đoạn văn bản. Là thẻ cơ bản để hiển thị nội dung dạng văn xuôi.
-
: Ngắt dòng. Không cần thẻ đóng.
- <hr>: Tạo đường kẻ ngang phân cách nội dung.

- : Làm đậm văn bản và nhấn mạnh ngữ nghĩa (có ý nghĩa với SEO).
- : Làm nghiêng văn bản và nhấn mạnh nội dung (có ý nghĩa với trình đọc màn hình).
- : Thẻ nội tuyến, không mang ý nghĩa ngữ nghĩa, thường dùng để định dạng một phần nhỏ văn bản bằng CSS.
- <blockquote>: Dùng để trích dẫn một đoạn nội dung từ nguồn khác.
- <pre>: Hiển thị văn bản có định dạng sẵn (giữ nguyên khoảng trắng, xuống dòng, thường dùng để trình bày mã lệnh).

2.2.3. Thẻ liên kết và hình ảnh

• <a>: Tạo liên kết (hyperlink) đến một trang web, một phần tử trong trang hoặc file tài liệu.

• <img>: Chèn hình ảnh vào trang web, sử dụng thuộc tính src để chỉ đường dẫn, và alt để mô tả hình ảnh (tốt cho SEO và khả năng truy cập).

2.2.4. Thẻ danh sách

- : Tạo danh sách **không có thứ tự** (unordered list).
- : Tạo danh sách **có thứ tự** (ordered list, đánh số).
- : Mỗi phần tử trong danh sách, dùng trong cả <ul> và <ol>.

2.2.5. Thẻ bảng

- <table>: Khởi tạo bảng.
- <tr>: Dòng trong bảng (table row).
- <th>: Ô tiêu đề (table header), thường là dòng đầu tiên.
- <td>: Ô dữ liệu (table data), nằm trong các dòng tiếp theo.

2.2.6. Thẻ biểu mẫu

Dùng để thu thập dữ liệu người dùng:

- <form>: Bao quanh toàn bộ biểu mẫu, có thể gửi dữ liệu đến máy chủ qua các phương thức như GET hoặc POST.
- <input>: Tạo các trường nhập liệu (văn bản, mật khẩu, checkbox, radio, file...).
- <label>: Gắn nhãn cho input, giúp tăng khả năng truy cập.
- <textarea>: Trường nhập văn bản nhiều dòng.
- <select> và <option>: Tạo menu chọn lựa (drop-down).
- <button>: Tạo nút bấm (có thể gửi form, hoặc chạy JavaScript).

2.2.7. Thẻ ngữ nghĩa

Các thẻ này giúp trình duyệt và công cụ tìm kiếm hiểu rõ hơn về ý nghĩa của nội dung:

- <header>: Khu vực đầu trang hoặc đầu mục nội dung (chứa logo, menu...).
- <nav>: Vùng chứa các liên kết điều hướng.
- <main>: Nội dung chính của trang.
- <article>: Một phần nội dung độc lập (bài viết, tin tức...).

- <section>: Phân vùng nội dung, thường theo chủ đề.
- <aside>: Nội dung phụ, như thanh bên (sidebar).
- <footer>: Phần cuối của trang, thường chứa thông tin bản quyền, liên hệ.

2.2.8. Thẻ đa phương tiện

- <video>: Chèn video, có thể kèm thuộc tính controls để hiển thị trình phát.
- <audio>: Chèn âm thanh, cũng có thể thêm controls.
- <source>: Chỉ định nguồn file cho thẻ video hoặc audio.
- <canvas>: Vùng vẽ đồ họa động bằng JavaScript (dùng cho game, đồ thị...).
- <iframe>: Nhúng trang web khác vào trang hiện tại (như nhúng video YouTube).

2.3. Quy tắc sắp xếp thẻ

- Trong HTML, việc sắp xếp và tổ chức các thẻ theo đúng thứ tự là vô cùng quan trọng để đảm bảo trang web hiển thị chính xác, dễ bảo trì, và tối ưu hóa cho trình duyệt cũng như công cụ tìm kiếm. Một tài liệu HTML chuẩn thường bắt đầu với dòng <!DOCTYPE html> ở ngay trên cùng để thông báo cho trình duyệt biết đây là tài liệu HTML5. Ngay sau đó là cặp thẻ <html>...</html> bao quanh toàn bộ nội dung của trang web.
- Bên trong thẻ <html>, cấu trúc được chia làm hai phần chính: phần <head> và phần <body>. Phần <head> nên được đặt đầu tiên và chứa các thông tin không hiển thị trực tiếp lên trang, như thẻ <meta> (mã hóa ký tự, mô tả, từ khóa SEO), <title> (tiêu đề hiển thị trên tab trình duyệt), các liên kết đến file CSS thông qua <link>, hoặc các đoạn mã JavaScript nếu cần tải sớm. Việc sắp xếp các thẻ trong phần <head> nên theo thứ tự: meta, title, link, và cuối cùng là script (nếu có).
- Tiếp đến là phần <body>, nơi chứa toàn bộ nội dung hiển thị ra trình duyệt cho người dùng. Trong phần này, các thẻ nên được sắp xếp theo cấu trúc lô-gic và ngữ nghĩa của trang web. Cụ thể, phần đầu trang thường sử dụng thẻ <header> để chứa logo và tiêu đề; nếu có menu điều hướng thì đặt bên trong <nav>. Nội dung chính của trang nên được bao quanh bởi thẻ <main>, bên trong đó có thể phân chia thành các phần riêng biệt bằng <section>, hoặc các bài viết độc lập bằng <article>. Nếu có nội dung phụ như quảng cáo, liên kết ngoài, hay thông tin bên lề, thì nên đặt vào thẻ <aside>. Cuối cùng, phần chân trang (footer) được đánh dấu bằng thẻ <footer>, thường chứa thông tin liên hệ, bản quyền, hoặc các liên kết phụ.
- Ngoài việc sắp xếp đúng thứ tự, người viết HTML cũng cần lưu ý đến việc **lồng thẻ đúng quy tắc**. Ví dụ, không nên đặt thẻ block như vào bên trong thẻ inline như . Thứ tự đóng thẻ phải theo nguyên tắc "mở trước, đóng sau", giống như cấu trúc ngăn xếp. Bên cạnh đó, việc **đóng thẻ đầy đủ**, **viết mã rõ ràng**, và **thụt lề hợp lý** sẽ giúp cho mã HTML dễ đọc, dễ bảo trì hơn, đặc biệt là khi làm việc theo nhóm hoặc dự án lớn.

2.4. Các loại HTML hiện nay

2.4.1. HTML4

HTML 4 ra đời vào năm 1997 là phiên bản thứ 4 của ngôn ngữ đánh dấu siêu văn bản. Phiên bản HTML này được xuất bản dưới dạng như một W3C Recommendation. HTML 4 áp dụng cho nhiều phần tử và thuộc tính khác nhau cho trình duyệt web.

2.4.2. XHTML

XHTML có tên đầy đủ là Extensible HyperText Markup Language, nghĩa là ngôn ngữ

đánh dấu siêu văn bản mở rộng. Đây là một ngôn ngữ thay thế của HTML với cú pháp chặt chẽ hơn. Cụ thể, XHTML yêu cầu mọi phần tử được đóng bằng thẻ đóng hoặc cú pháp tự đóng riêng và phân biệt được chữ in hoa hoặc chữ in thường, trong khi đó HTML không có điều này.

2.4.3. HTML5

HTML5 phiên bản thứ 5 của HTML được công bố bởi World Wide Web Consortium (W3C). HTML5 là sự kết hợp giữa HTML 4, XHTML, DOM cấp 2 và JavaScript. HTML5 hỗ trợ chạy trên mọi trình duyệt.

2.5. Các thuộc tính trong HTML

Trong HTML, **thuộc tính** là những thành phần mở rộng được sử dụng để cung cấp thông tin bổ sung cho các thẻ (element). Chúng giúp điều chỉnh cách hoạt động, cách hiển thị hoặc cách một phần tử tương tác trong trang web. Mỗi thuộc tính bao gồm hai phần: **tên thuộc tính (attribute name)** và **giá trị (value)**, được viết dưới dạng tên="giá trị", và luôn đặt **bên trong thẻ mở** của phần tử. Một phần tử HTML có thể chứa nhiều thuộc tính khác nhau, và các thuộc tính này phải được cách nhau bằng dấu cách.

2.5.1. Thuộc tính toàn cục

Những thuộc tính này có thể áp dụng cho **mọi thẻ HTML**:

- id: Xác định danh tính duy nhất cho một phần tử.
- class: Gán một hoặc nhiều lớp CSS cho phần tử.
- style: Viết CSS trực tiếp bên trong phần tử.
- title: Hiển thị chú thích khi rê chuột.
- hidden: Ẩn phần tử khỏi giao diện.
- data-*: Tạo các thuộc tính tùy chỉnh (dùng trong JavaScript).

2.5.2. Thuộc tính riêng cho từng thẻ

Thẻ <a>:
- href: Liên kết đến địa chỉ URL.
- target: Mở liên kết trong cửa sổ/tab mới (_blank) hoặc hiện tại (_self).

Thẻ <img>:
- src: Đường dẫn đến ảnh.
- alt: Văn bản mô tả ảnh (dùng khi ảnh lỗi hoặc cho người khiếm thị).
- width, height: Kích thước ảnh.

Thẻ <input>:
- type: Loại trường nhập (text, password, checkbox...).
- name: Tên trường (dùng khi gửi form).
- value: Giá trị mặc định.
- placeholder: Gợi ý nội dung cần nhập.
- required: Bắt buộc nhập.

Thẻ <form>:
- action: URL xử lý dữ liệu form.
- method: Phương thức gửi dữ liệu (GET hoặc POST).

CHƯƠNG 3. CSS

3.1. CSS là gì

- CSS là chữ viết tắt của Cascading Style Sheets, được đề xuất ra đời bởi lập trình viên Harkon Wium Lie và được sản xuất chính thức vào năm 1996. Đây là một **ngôn ngữ lập trình** có thiết kế vô cùng đơn giản và thường được sử dụng để định vị và định dạng lại cho các phần tử được tạo bởi các ngôn ngữ đánh dấu. Mục tiêu chính của ngôn ngữ này chính là thực hiện và xử lý giao diện của một trang web cụ thể như màu sắc, cách đổi màu chữ, thay đổi bố cục, màu nền,…
- Hiện nay có nhiều kiểu khác nhau được đưa vào sử dụng trong CSS. Nhưng về cơ bản chúng được chia thành các loại phổ biến nhất bao gồm:

- Tùy chỉnh hình nền – Background
- Tùy chỉnh cách hiển thị đoạn text – Text
- Tùy chỉnh kiểu chữ và kích thước – Font
- Tùy chỉnh bảng – Table
- Tùy chỉnh danh sách – Link
- Mô hình box model có kết hợp với padding, margin, border – Box model

- Theo các chuyên gia đánh giá CSS giữ một vai trò vô cùng quan trọng và cần thiết cho việc phát triển và thiết kế website. Nếu như HTML giữ vai trò trong việc định dạng các phần tử bên trong website nhưng lại không thể thay đổi được các cấu trúc, font chữ hay màu chữ, màu sắc của trang thì CSS có thể giúp ích rất nhiều trong việc này. Nhờ vậy, chúng ta có thể kiểm soát việc hiển thị một tài liệu **HTML** nhất định một cách mạnh mẽ và hiệu quả. Vì lẽ đó, nó là một công cụ thường được kết hợp với các ngôn ngữ như HTML và XHTML.

3.2. Tại sao cần sử dụng CSS

Cung cấp nhiều thuộc tính khác nhau

Khi nói đến lý do tại cần sử dụng CSS thì không bao giờ có thể bỏ qua thông tin này. Dễ thấy rằng các thuộc tính do CSS cung cấp chi tiết hơn HTML trên cùng một giao diện trang web. Điều này cho phép CSS cung cấp thêm nhiều phong cách, kiểu dáng khác nhau và đem lại khả năng tùy chỉnh các trang web theo cách thuận tiện và bắt mắt nhất.

Thời gian sử dụng sẽ được rút lại

Thời gian sử dụng được tiết kiệm tối đa bởi bạn có thể được dùng chương trình với mã code ngắn lại hơn. Nhờ đó giúp kiểm soát nhanh hơn, thuận tiện hơn và ít lo lắng hơn về loại mã không cần thiết xuất hiện trong chương trình. Thêm vào đó số lượng các stylesheet mà web mang đến là vô cùng đa dạng, đảm bảo hạn chế tối đa sự trùng lặp gây mất kiểm soát và không ấn tượng.

Giải quyết được nhiều vấn đề khác nhau

Khi bắt đầu sử dụng, sẽ thấy trang web của mình được tổ chức một cách gọn gàng và có tổ chức hơn nhiều. Những nội dung được hiển thị trên trang này sẽ được tách biệt hơn, giúp định dạng nhanh chóng và dễ dàng.

3.3 Bố cục và cấu trúc của ngôn ngữ lập trình CSS

3.3.1. Bố cục một đoạn CSS

Bố cục *CSS* thường dựa trên các hình hộp, trong đó mỗi hộp chiếm không gian trên trang và có các thuộc tính như:

- Padding (vùng đệm): Gồm không gian xung quanh nội dung (ví dụ: xung quanh đoạn văn bản).
- Border (đường viền): Là đường liền nằm ngay bên ngoài phần đệm.
- Margin (lề): Là khoảng cách xung quanh bên ngoài của phần tử.

3.3.2. Cấu trúc một đoạn CSS

Về cơ bản cấu trúc của một đoạn CSS bao gồm các phần như sau:

- Phần thứ 1: Vùng chọn {
- Phần thứ 2: Thuộc tính:
- Phần thứ 3: Giá trị;
- Phần thứ 4: }
- Phần thứ 5: ….

Các phần này có ý nghĩa như sau:

- **Bộ chọn tên tiếng anh là Selector:** Tên của phần tử HTML kích hoạt bộ quy tắc và thực hiện lựa chọn phần tử đã được tạo kiểu. Từ đó, có thể tạo kiểu cho các phần tử khác chỉ bằng cách thay đổi bộ chọn.
- **Tuyên bố (Declaration):** Xác định các thuộc tính của phần tử để định dạng.
- **Thuộc tính:** Là cách tạo kiểu cho phần tử HTML. Vì vậy, trong CSS, chỉ cần lựa chọn những thuộc tính mà muốn tác động ảnh hưởng lên nhất trong bộ quy tắc của mình.
- **Giá trị thuộc tính:** Ở bên phải của thuộc tính sau dấu hai chấm (:) là giá trị thuộc tính và các lựa chọn xuất hiện nhiều lần để cho phép chỉ định một thuộc tính cụ thể.

3.4 Các cách để triển khai CSS vào HTML

3.4.1. Inline CSS

Nên dùng kiểu định dạng này nếu áp dụng một kiểu định dạng riêng cho một phần tử riêng. Sử dụng CSS bằng cách này không được khuyến khích, vì mỗi tag HTML cần được styles độc lập. Để áp dụng, thêm thuộc tính style cho phần tử liên quan, thuộc tính style có thể chứa đặc tính CSS. Vì cách này không được khuyến khích dùng nên chúng ta chỉ đi sơ qua về ví dụ:

```
<html>
<head>
    <title>CSS</title>
    <meta charset="utf-8">
</head>
<body>
    <h1 style="color:red;padding:30px;"> SUNTECH.EDU.VN </h1>
    <h2 style="color:blue;">Học Lập Trình Miễn Phí.</h2>
</body>
</html>
```

3.4.2. Internal CSS

Khi dùng cách này CSS code được đặt trong mục <head> của một trang nhất định để áp dụng style cho toàn bộ trang ấy. Khi đó chỉ cần đặt đoạn CSS vào trong cặp thẻ <style> rồi đặt vào trong phần header của trang HTML (giữa <head> và </head>):

```
<html>
<head>
    <title>CSS</title>
    <meta charset="utf-8">
<style type="text/css">
    h1
        {color:red;padding:30px;}
    h2
        {color:blue;}
</style>
</head>
<body>
    <h1> SUNTECH.EDU.VN </h1>
    <h2>Học Lập Trình Miễn Phí.</h2>
</body>
</html>
```

Khi sử dụng cách này :
- Chỉ một trang ảnh hưởng bởi stylesheet.
- Classe và ID có thể được dùng bởi internal stylessheet.
- Không cần phải upload nhiều files. Lúc này HTML và CSS có thể là một file.

3.4.3. External CSS

Đây là cách được khuyên dùng nhiều nhất để thêm style css vào một trang web.
Với một file .css ở ngoài có thay đổi diện mạo của cả website mà chỉ cần thay đổi một tập tin duy nhất. Mỗi trang sẽ có tham chiếu tới tập tin ngoài này trong phần tử <link> nằm trong phần <head>. Thông thường ta sẽ đặt thẻ link này trong phần cuối cùng của thẻ head.

```
<head>
  <link rel="stylesheet" type="text/css" href="style.css" />
</head>
```

3.5. CSS SELECTOR

Selector là các lựa chọn dùng để truy vấn đến các thẻ HTML nhằm tác động vào chúng.
Ví dụ: Selector đến tất cả các thẻ h1 trong HTML chúng ta sẽ sử dụng cú pháp:

```
h1
{
  property: value;
}
```

Các dạng selector:

Selector	Ý Nghĩa	Ví Dụ
Universal Selector	Loại selector này sẽ tác động đến tất cả các thẻ HTML trên trang web	*{ property: value; }
Type Selector	Loại CSS này sẽ tác động đến các thẻ HTML mà chúng ta muốn tác động	h1, h2, h3 {property: value;}
Class Selector	Loại selector này sẽ tác động đến tất cả các thẻ HTML có class mà nó quy định	.note {property: value;} ; p.note {};...
ID Selector	Loại selectors này sẽ tác động đến tất cả các thẻ HTML có id mà nó quy định	#introduction {property: value;}
Child Selector	Loại css này sẽ cho phép chúng ta css vào các thẻ HTML theo cấp cha con	li>a {property: value;}
Descendant Selector	Chọn bất kì phần tử con nào trong phần tử cha	p a {property: value;}
Adjacent Sibling Selector	Nhắm chọn các phần tử anh chị em	h1+p {property: value;}
General Sibling Selector	Nhắm chọn các phần tử anh chị em mặc dù nó không nhất thiết phải là phần tử đứng trước trực tiếp	h1~p {property: value;}

3.6. Animation

Animation trong CSS là kỹ thuật cho phép tạo hiệu ứng chuyển động, biến đổi thuộc tính của phần tử HTML theo thời gian mà không cần JavaScript. Khác với transition (chỉ có 2 trạng thái bắt đầu/kết thúc), animation hỗ trợ **nhiều bước trung gian** (keyframes) và kiểm soát chi tiết tốc độ, hướng, lặp lại...

3.6.1. Cơ chế hoạt động

- Định nghĩa tự chuyển động với @keyframe
- Xác định các trạng thái tại các điểm phần trăm(0% đến 100%)
- Áp dụng animation cho phần tử
- Sử dụng các thuộc tính animation hoặc sub-properties

3.6.2. Các thuộc tính điều khiển

- animation-name: Tên keyframe đã định nghĩa (bắt buộc)
- animation-duration: Thời gian hoàn thành 1 chu kỳ (vd: 2s)
- animation-timing-function: Hàm điều phối tốc độ:
  - Giá trị cơ bản: ease, linear, ease-in, ease-out, ease-in-out
  - Tùy chỉnh nâng cao: cubic-bezier(0.17,0.67,0.83,0.67)
- animation-delay: Thời gian trễ trước khi bắt đầu (vd: 1s)
- animation-iteration-count: Số lần lặp:
  - Giá trị số (vd: 3) hoặc infinite (lặp vô hạn)
- animation-direction: Hướng chuyển động:
  - normal (mặc định), reverse (ngược), alternate (đảo chiều luân phiên)
- animation-fill-mode: Xác định trạng thái trước/sau animation:
  - forwards (giữ trạng thái cuối), backwards (áp dụng trạng thái đầu ngay lập tức)
- animation-play-state: Tạm dừng/tiếp tục animation (thường kết hợp với JS)

3.7. Transition

Transitions CSS là một tính năng cho phép nhà phát triển web tạo ra hiệu ứng chuyển đổi và kiểm soát tốc độ chuyển đổi của các phần tử HTML khi thuộc tính CSS của chúng thay đổi.
Chẳng hạn, muốn thay đổi màu của một phần tử từ trắng sang đen. Thông thường, sự thay đổi sẽ diễn ra tức thời. Tuy nhiên khi áp dụng transition CSS, các thay đổi này sẽ diễn ra theo các khoảng thời gian đã được định sẵn.

3.7.1. Cú pháp và thuộc tính

- **Cú pháp:**

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

- **Transition CSS gồm 4 thuộc tính:**

- transition-property (thuộc tính chuyển đổi)
- transition-duration (thời gian chuyển đổi)
- transition-timing-function (hàm thời gian chuyển đổi)
- transition-delay (trì hoãn chuyển đổi)

3.7.2. So sánh transition với animation

Tiêu chí	Transition CSS	Animation CSS
Bản chất	Chỉ có thể chuyển từ trạng thái ban đầu sang trạng thái cuối cùng, không có bước trung gian.	Tạo các hoạt động phức tạp hơn bằng cách điều khiển nhiều trạng thái thuộc tính tại các thời điểm khác nhau.
Độ phức tạp	Đơn giản hơn, với việc triển khai trực tiếp cho các thay đổi trạng thái cơ bản.	Phức tạp hơn, phù hợp với các hiệu ứng hoạt hình chi tiết và tuỳ biến hơn.
Tần suất	Chỉ có thể chạy một lần.	Có thể lặp vô hạn nhờ thuộc tính animation-interation-count.
Kích hoạt	Yêu cầu kích hoạt để hoạt động, chẳng hạn thao tác di chuyển chuột vào phần tử div.	Khởi chạy tự động khi trang hoặc phần tử được tải hoặc bằng cách thêm các lớp động vào phần tử.
Sự thay đổi	Chạy tiến khi được kích hoạt và chạy ngược lại khi bỏ kích hoạt.	Có thể chạy tiến, lùi hoặc đổi chiều.
Tương tác với JavaScript	Dễ sử dụng với JavaScript hơn.	Khó sử dụng với JavaScript hơn.
Phạm vi ứng dụng	Thích hợp để cải thiện các yếu tố giao diện người dùng, phản hồi các hành động cụ thể của người dùng như di chuột hoặc nhấp chuột.	Được áp dụng trong các tình huống yêu cầu chuyển động liên tục hoặc hiệu ứng hình ảnh phức tạp hơn, do tương tác của người dùng kích hoạt hoặc chạy tự động.

3.8. Các framework CSS

3.8.1. CSS framework là gì

CSS Frameworks là các thư viện mã nguồn giúp tăng tốc độ phát triển giao diện web bằng cách cung cấp các công cụ, thành phần giao diện được thiết kế sẵn. Các framework này thường bao gồm:

- Hệ thống lưới (grid system): Giúp tạo bố cục giao diện dễ dàng.
- Các class CSS: Cung cấp kiểu dáng cho các thành phần HTML thông dụng như nút bấm, bảng, biểu mẫu.
- Các tiện ích mở rộng JavaScript: Thường được tích hợp sẵn các tính năng động như dropdown, modal.

3.8.2. Mục đích sử dụng framework

Sử dụng CSS Frameworks không chỉ giúp tiết kiệm thời gian mà còn mang lại nhiều lợi ích nổi bật:

1. **Tăng tốc độ phát triển**: Với các thành phần sẵn có, bạn có thể nhanh chóng tạo ra các giao diện chuyên nghiệp mà không cần viết CSS từ đầu.

2. **Đảm bảo tính nhất quán**: CSS Frameworks cung cấp các quy tắc và kiểu dáng đồng nhất, giúp các trang web hoặc ứng dụng của bạn giữ được phong cách thiết kế liền mạch.

3. **Tương thích tốt trên nhiều thiết bị**: Hầu hết các framework đều hỗ trợ responsive design, giúp giao diện hiển thị tốt trên cả máy tính, tablet, và điện thoại.

4. **Cộng đồng hỗ trợ mạnh mẽ**: Các framework phổ biến như Bootstrap hay Tailwind CSS có cộng đồng rộng lớn, tài liệu chi tiết và nhiều nguồn hỗ trợ khác.

5. **Giảm thiểu lỗi CSS**: Với các quy tắc được định nghĩa rõ ràng, bạn sẽ tránh được những lỗi thông thường khi tự viết CSS.

3.8.3. Các CSS Frameworks phổ biến và tính năng nổi bật

3.8.3.1. Bootstrap

Bootstrap là một trong những CSS Frameworks lâu đời và phổ biến nhất. Được phát triển bởi Twitter, Bootstrap cung cấp mọi công cụ cần thiết để tạo giao diện hiện đại.

- **Điểm mạnh**:
  - Grid system mạnh mẽ.
  - Đầy đủ các thành phần UI như navbar, cards, modal.
  - Hỗ trợ tốt cho responsive design.

- **Nhược điểm**:

  - Ít linh hoạt, khó tùy chỉnh nếu bạn không quen với cấu trúc của Bootstrap.

3.8.3.2. Tailwind CSS

Tailwind CSS là framework theo phương pháp utility-first, cho phép bạn kiểm soát chi tiết giao diện bằng cách sử dụng các class nhỏ gọn.

- **Điểm mạnh**:

  - Linh hoạt tối đa.
  - Có thể tùy chỉnh toàn bộ thiết kế theo ý muốn.
  - Hỗ trợ theme và plugin.

- **Nhược điểm**:

  - Yêu cầu học cách sử dụng nhiều class CSS mới.

3.8.3.3. Foundation

Được phát triển bởi Zurb, Foundation là một framework mạnh mẽ dành cho các dự án lớn.

- **Điểm mạnh**:

  - Tập trung vào accessibility.
  - Cung cấp nhiều công cụ hỗ trợ phát triển ứng dụng web.

- **Nhược điểm**:

  - Độ phức tạp cao, cần nhiều thời gian để học.

3.8.3.4. Bulma

Bulma là một CSS Framework nhẹ nhàng, tập trung vào việc sử dụng class-based CSS để định kiểu.

- **Điểm mạnh**:

  - Dễ học, dễ sử dụng.
  - Thiết kế đơn giản, thân thiện.

- **Nhược điểm**:

  - Thiếu các tính năng JavaScript tích hợp.

3.8.3.5. Materialize

Materialize được xây dựng dựa trên triết lý Material Design của Google.

- **Điểm mạnh**:

- Phong cách hiện đại, tối ưu cho thiết kế web.
- Đầy đủ các thành phần UI.

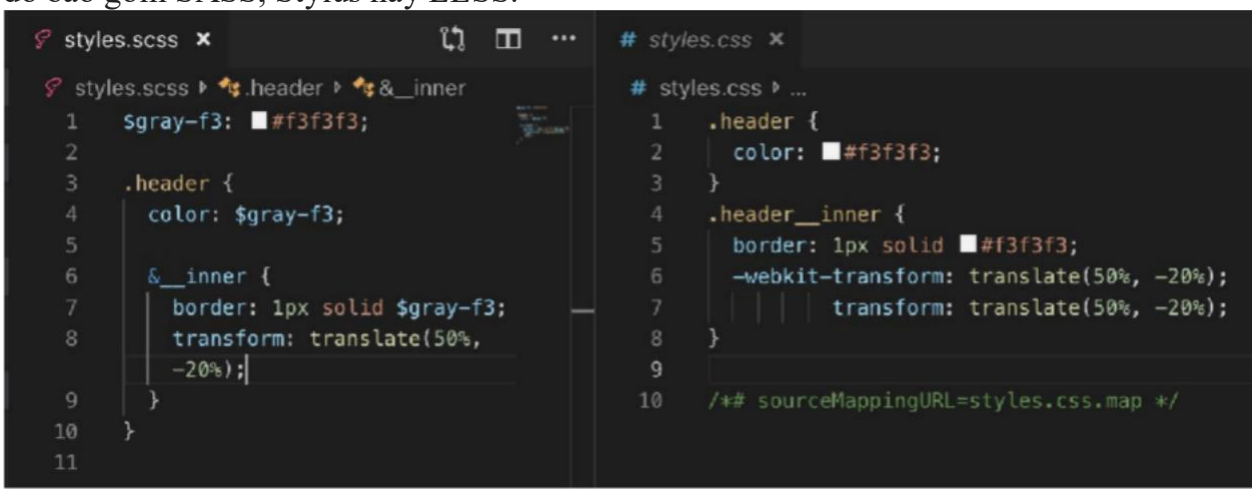
- **Nhược điểm**:

- Hạn chế trong việc tùy chỉnh.

3.9. SASS/SCSS

3.9.1. SASS/SCSS là gì

SASS/SCSS là một chương trình tiền xử lý CSS (*CSS preprocessor)*. Nó giúp bạn viết CSS theo cách của một ngôn ngữ lập trình, có cấu trúc rõ ràng, rành mạch, dễ phát triển và bảo trì code hơn. Ngoài ra nó có rất nhiều các thư viện hỗ trợ kèm theo giúp bạn viết code CSS một cách dễ dàng vào đơn giản hơn. Có rất nhiều loại CSS Preprocessor trong đó bao gồm SASS, Stylus hay LESS.



3.9.2. Các tính năng cơ bản của SCSS

3.9.2.1. Biến(Variables)

SCSS cho phép sử dụng biến để lưu trữ các giá trị tái sử dụng như màu sắc, font chữ hay kích thước. Bằng cách khai báo biến với ký hiệu $, bạn có thể quản lý thiết kế một cách nhất quán. Khi cần thay đổi giá trị, chỉ cần cập nhật ở nơi khai báo biến và thay đổi sẽ áp dụng tự động mọi nơi sử dụng biến đó. Ví dụ:
`$primary-color: #3498db;` giúp sử dụng màu xanh dương nhất quán toàn bộ dự án.

3.9.2.2. Mixins

Mixins hoạt động như các hàm có thể tái sử dụng, cho phép đóng gói các khối mã CSS thành các module. Chúng có thể nhận tham số đầu vào để tạo kiểu dáng linh hoạt, giúp giải quyết các mẫu thiết kế phức tạp hay các thuộc tính cần vendor prefix. Ví dụ mixin `flex-center` có thể tái sử dụng cho nhiều thành phần cần canh giữa.

3.9.2.3. Kế thừa (Extend)

Với `@extend`, các selector có thể kế thừa thuộc tính từ selector khác. Tính năng này giúp tránh trùng lặp mã CSS bằng cách nhóm các selector có chung thuộc tính lại với nhau trong output. Điều này không chỉ giúp mã ngắn gọn hơn mà còn tối ưu hóa kích thước file CSS cuối cùng.

3.9.2.4. Import

Cú pháp import của SASS rất hữu dụng và thường xuyên được sử dụng trong các project. Nó tương tự cách bạn require hay include file này vào file khác trong PHP.

Đặt trường hợp có 1 trang index, bao gồm header, body, footer. Thay vì sử dụng CSS cho những cái trên vào một style.css thì với SASS bạn sẽ thực hiện như sau, nhớ có dấu _ trước tên file được import:

1. Tạo 1 file **_header.scss** để CSS riêng cho header.
2. **_body.scss** để CSS riêng cho body.
3. **_footer.scss** để CSS riêng cho footer.
4. Rồi ở file style.css ta chỉ cần @import 3 file trên là được

3.9.3. Trình compile SASS

Hiện nay tồn tại tương đối nhiều trình biên dịch SASS sang CSS thuần. Trong đó có hai trình biên dịch được sử dụng thường xuyên:

1. Koala

- Đây là một phần mềm dùng để compile CSS Preprocessor như SASS, LESS… mình hay dùng nó khi viết SASS.

2. Laravel Mix

- Nếu bạn đang làm việc bằng Laravel thì bạn không cần đến phần mềm thứ 3 đâu vì bản thân Laravel đã tích hợp một công cụ tên là Laravel Mix rất đa năng, compile các CSS Preprocessor sang CSS thuần là một trong những tính năng xịn xò của nó.

CHƯƠNG 4. JAVASCRIPT

4.1. Giới thiệu chung

4.1.1. JavaScript là gì?

Javascript chính là một ngôn ngữ lập trình web rất phổ biến ngày nay. Javascript được tích hợp đồng thời nhúng vào HTML để hỗ trợ cho website trở nên sống động hơn. Chúng cũng đóng vai trò tương tự như một phần của website, cho phép Client-side Script từ người dùng tương tự máy chủ (Nodejs) để tạo ra những website động.

4.1.2. Lịch sử phát triển của JavaScript

- Brendan Eich chính là người đã phát triển Javascript tại Netscape với tiền thân là Mocha. Sau đó, Mocha được đổi thành LiveScript và cuối cùng mới đổi thành JavaScript.
- **Năm 1998**, JavaScript với phiên bản mới nhất là ECMAScript 2 phát hành và đến năm 1999 thì ECMAScript 3 được ra mắt.
- **Năm 2016**, ứng dụng JavaScript đã đạt kỷ lục lên tới 92% website sử dụng, đồng thời cũng được đánh giá là một công cụ cực kỳ quan trọng đối với lập trình viên.

4.1.3. JavaScript dùng để làm gì?

Việc nắm bắt được mục đích của ngôn ngữ đặc biệt này sẽ giúp dễ dàng sử dụng chúng hơn trong công việc. Cụ thể như sau:

- **Thay đổi nội dung HTML:** Một trong số nhiều phương thức HTML JavaScript chính là getElementById (). Chúng được sử dụng để tìm một phần tử của HTML với id =”demo" và dùng để thay đổi nội dung của phần từ (Internal HTML) sang thành “Hello JavaScript”
- **Thay đổi giá trị thuộc tính HTML:** *Tổng quan về javascript* còn có thể sử dụng để thay đổi các giá trị của thuộc tính. Ví dụ: thay đổi thuộc tính src (source) của tag<img>.
- **Thay đổi kiểu HTML:** Đây chính là một hoạt động biến thể của việc thay đổi thuộc tính của HTML ở trên. Ví dụ: document.getElementById(‘demo’).style.fontSize = ’35px;
- **Ẩn các phần tử HTML:** Một hoạt động tiếp theo là Javascript có thể ẩn được các phần tử HTML. Chúng có thể được thực hiện thông qua hoạt động thay đổi kiểu hiển thị các phần tử HTML.
- **Hiển thị các phần tử HTML:** Một điểm đặc biệt là JavaScript có thể hiển thị được các yếu tố HTML ẩn. Đồng thời, cũng có thể thực hiện được thông qua cách thay đổi kiểu hiển thị phần tử.

4.1.4. Cách JavaScript hoạt động trên trang web

Cách hoạt động của *javascript là*

- Thông thường, **JavaScript** sẽ được nhúng trực tiếp vào một website hoặc chúng được tham chiếu qua file .js hoặc .JavaScript.
- Đây là một ngôn ngữ đến từ phía Client nên Script sẽ được download về máy client khi truy cập.
- Tại đây, chúng sẽ được hệ thống xử ý. Vì vậy, không cần phải tải về máy server rồi chờ cho chúng xử lý xong mới phản hồi được kết quả đến client.

4.2. ES5, ES6, Typescript

4.2.1. ES5

ES (ECMAScript) là một ngôn ngữ được chuẩn hóa bởi tổ chức ECMA và được giám sát bởi hội đồng TC39. Và Javascript là cài đặt cụ thể của chuẩn ECMAScript này và trở thành một ngôn ngữ thông dụng trong lập trình web hiện này. ES trải qua rất nhiều phiên bản khác nhau và nó dần được cải tiến để mạnh mẽ và phù hợp hơn. Javascript là cài đặt cụ thể của ES, tuy nhiên nó không gắn liền với một phiên bản ES cụ thể mà có thể bao gồm các phiên bản ES từ cũ đến phiên bản hiện tại.

ES5 là phiên bản thứ năm cửa ECMAScript, nó được giới thiệu vào năm 2009. Nó được chúng ta sử dụng nhiều nhất trong nhiều năm để làm chuẩn cài đặt ra Javascript được hỗ trợ hầu hết bởi tất cả các trình duyệt hiện tại.

ES5 sử dụng lập trình hàm (Functional Programming) để là tiểu chuẩn cho Javascript, do đó hầu hết hiện này chúng ta vẫn sử dụng Javascript theo cách lập trình hàm là chủ yếu, trong khi các ngôn ngữ khác chạy theo xu hướng OOP.

ES5 rất mềm dẻo và dễ dàng để lập trình với các đặc trưng như Scoping, closures, IIFE (immediately-invoked function expression), loosely typed (đặc trưng của Javascript). Tuy nhiên với ES5 chúng ta cũng gặp không ít khó khăn khi lập trình với ES5. Đó là trong quá trình lập trình rất khó để kiểm tra các vấn đề liên quan tới ES5 (hay còn biết tới việc debug), rất khó để debug đối với Javascript, và các công cụ hỗ trợ cho việc này cũng rất ít và khá là phức tạp để sử dụng.

Nói tóm lại có thể xem ES5 chính là Javascript mà chúng ta sử dụng hiện nay để lập trình các web động.

Đây là những tính năng của ES5:

- The "use strict" Directive
- String.trim()
- Array.isArray()
- Array.forEach()
- Array.map()
- Array.filter()
- Array.reduce()
- Array.reduceRight()
- Array.every()
- Array.some()
- Array.indexOf()
- Array.lastIndexOf()
- JSON.parse()
- JSON.stringify()
- Date.now()
- Property Getters and Setters
- New Object Property Methods

4.2.2. ES6/ ES2015

ES6 là phiên bản thứ sáu của ECMAScript, nó được ra mặt vào 2015. ES6 được xem là một bước nhảy lớn của ES5 khi nó được thêm các khái niệm mới, chức năng mới vào cho

Javascript. Các tính năng mới giải quyết các vấn đề còn tồn tại và là thách thức của ES5. Các tính năng này là tùy chọn vì vậy chúng ta vẫn có thể sử dụng ES5 để lập trình trong ES6. Dưới đây là là danh sách cụ thể các tính năng của ES6:

- arrows
- classes
- enhanced object literals
- template strings
- destructuring
- default + rest + spread
- let + const
- iterators + for..of
- generators
- unicode
- modules
- module loaders
- map + set + weakmap + weakset
- proxies
- symbols
- subclassable built-ins
- promises
- math + number + string + array + object APIs
- binary and octal literals
- reflect api
- tail calls

Với một lượng lớn các tính năng mới ta có thể sử dụng để viết code bằng javascript. Các trình duyệt hiện đại ngày nay cũng đang chạy đua để hỗ trợ các tính năng trên của ES6. Do là chuẩn mới nên còn một số trình duyệt chưa hỗ trợ ES6 tuy nhiên ta có thể viết code bằng ES6 rồi sử dụng các tool để dịch code xuống các phiên bản Javascript cũ hơn. Một tool cụ thể cho việc này là Babe. Như vậy ES6 cũng chính là Javascript nhưng nó được tiến hóa hơn khi đưa thêm một số khái niệm và tính năng mới vào so với Javascript của ES5.

4.2.3. Typescript

TypeScript được coi là tập cha của Javascript, sau khi được biện nó sẽ biên dịch ra javascript thuần. Có thể coi TypeScript bao gồm ES6 cùng với việc thêm vào các khái niệm mới là:

- Types (Các kiểu dữ liệu)
- Interface (Giao diện)

Một trong những vấn đề xảy ra khi làm việc đối với Javascript đó là rất khó để xác định lỗi trong quá trình code (có thể gọi là debug). Và TypeScript ra đời để góp phần vào giải quyết vấn đề này. TypeScript không giúp chúng ta code đơn giản đi mà nó giúp ta code an toàn hơn giảm tối thiểu lỗi và phát hiện sớm các lỗi tiềm ẩn. Ngoài ra nó giúp chúng ta code hiểu quả hơn bằng việc sử dụng lợi thế của các tool, editor để xác định các lỗi, tự động sinh và gán các biến, hàm, thuộc tính ...

Như vậy có thể nói TypeScript về mặt bản chất nó cũng là JavaScript tuy nhiên, nó đưa vào một số khái niệm, nguyên tắc giúp ta là việc với JavaScript một cách an toàn, nhanh chóng và hiểu quả hơn

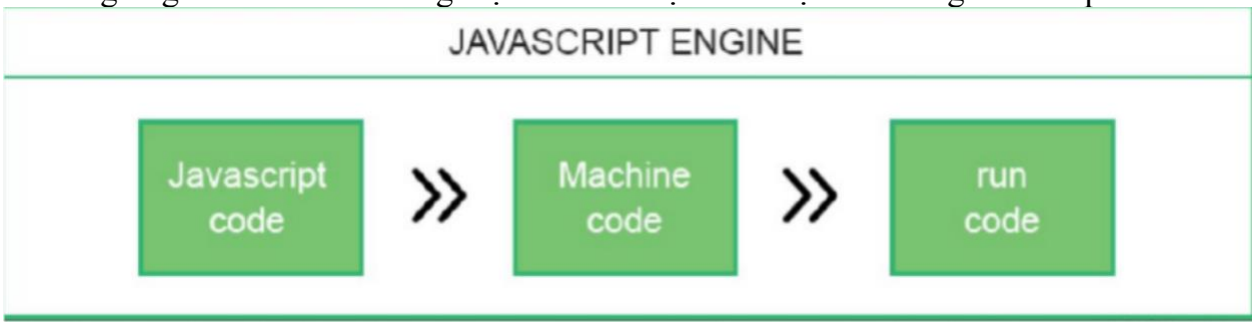
4.2.4. So sánh

Tiêu Chí	ES5	ES6	TypeScript
Bản chất	JavaScript	JavaScript	JavaScript
Sử dụng	Viết trực tiếp và chạy không cần biên dịch	Cần biên dịch để chạy	Cần biên dịch để chạy
Tính năng	Sử dụng lập trình hàm để thực hiện viết code	Tương tự ES5 tuy nhiên đưa vào một số khái niệm mới	Cũng có thể sử dụng lập trình hàm như ES5 và các tính năng của ES6, tuy nhiên còn sử dụng thêm các types và interface

4.3. Cách JavaScript thực thi trong môi trường browser

4.3.1. Javascript parsers và engines

Khi chúng ta viết code Javascript và chạy đoạn code đó, một javascript engine sẽ nhận đoạn mã đó mà thực thi nó, trong trường hợp bạn chưa biết thì javascript engine là một chương trình để thực thi code được viết bằng Javascript, một số javascript engine nổi tiếng như Google's V8 Engine, SpiderMonkey, JavascriptCore,... Các Engine này sẽ nhận vào code được viết bằng Javascript và sẽ phân tích từng dòng một, nếu có lỗi về syntax, nó sẽ ném ra lỗi và chương trình dừng hoạt động, nếu đoạn code đó đã được sửa hết lỗi Engine sẽ chuyển các đoạn mã đó thành mã máy để máy tính có thể hiểu và thực thi chúng. Các Engine khác nhau sẽ có chút khác biệt về cách hoạt động, nhưng nó đều có vai trò giống nhau là môi trường thực thi các đoạn mã được viết bằng Javascript.

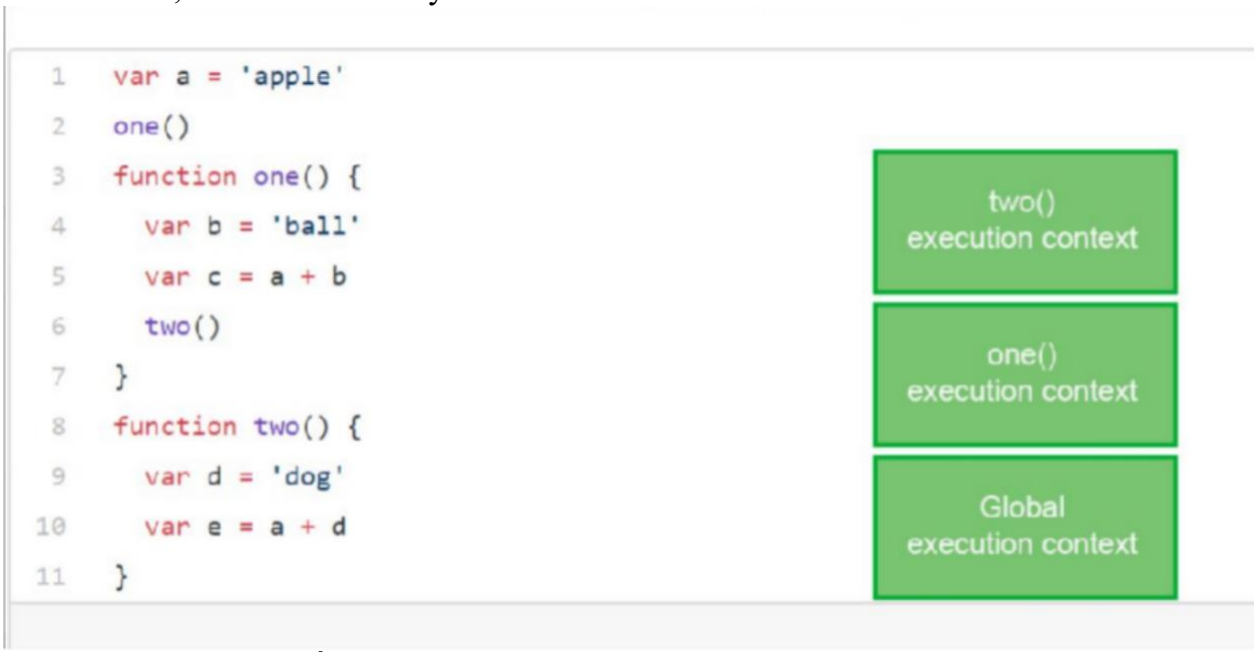


4.3.2. Execution context và Execution stack

Các đoạn code Javascript được chạy trong một môi trường (ở đây là Browser) và các môi trường đó còn được gọi là các Execution context (EC). Mặc định, các đoạn code không nằm trong bất kỳ một function nào đều nằm trong Global Execution context (GEC) và được gán trực tiếp với global object, và global object trong môi trường Browser chính là window object !

```
var helloBuddy;
helloBuddy === window.helloBuddy
// true
```

Execution stack, đó là một cấu trúc dữ liệu nơi mà một EC mới được đẩy lên trên GEC nếu có một function nào đó được thực thi, và cứ tiếp tục như vậy, nếu trong function đó lại có môt function khác cần thực thi, thì function "con" đó sẽ tạo ra một EC mới ở trên EC trước đó, như hình dưới đây:



Khi các Engine gặp biến a và function one(), nó sẽ được chưa trong GEC vì nó không nằm trong bất kỳ một function nào, khi thực thi function one(), nó tạo một EC nằm trên GEC, tiếp tục, trng function one() có 2 biến là b và c, cả 2 biến này đều nằm trong one() execution context , đến dòng thực thi function two() nó sẽ lại sinh ra một EC khác nằm trên one() execution context . và sau khi đã thực hiện xong, các EC của one() và two() sẽ mất đi và chỉ còn lại GEC.

4.3.3. Creation phases, Execution phases và Hoisting

Một EC có thể xem như một object và object đó bao gồm 3 thuộc tính:

- Variable Object (VO)
- Scope Chain
- từ khóa "this"

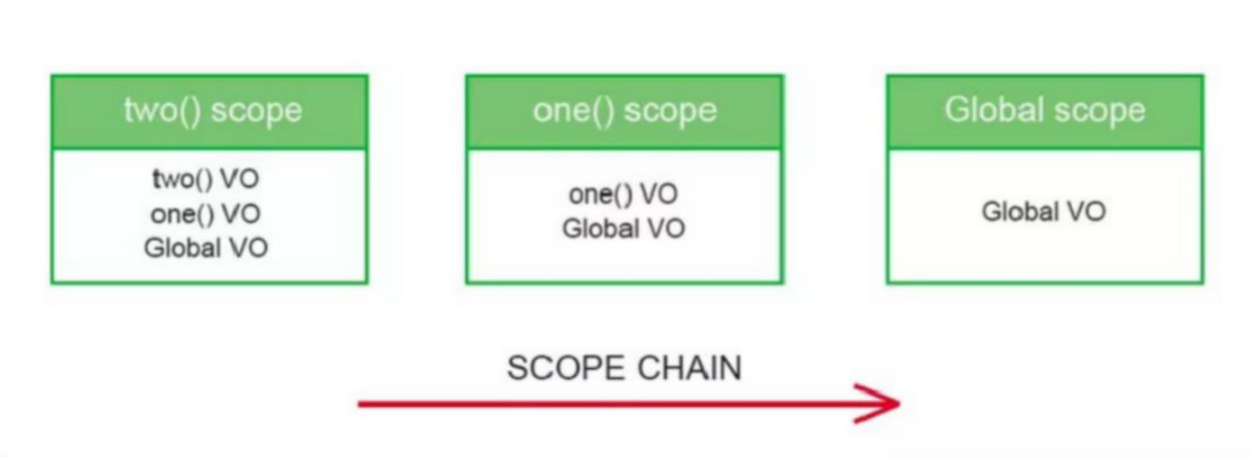
4.3.3.1. Variable Object(VO)

Khi một function được gọi bên trong một fuction khác, một EC mới được tạo ra, quá trình này gồm 2 phần gọi là Creation phase và Execution phase. Tất cả các thuộc tính như VO, Scope Chain và từ khóa "this" được hình thành ở phần Creation, sau đó đến phần Execution là lúc mà các đoạn code sẽ được thực thi.

Khi code được thực thi, mỗi khi khai báo một function, một thuộc tính sẽ được tạo ra trong VO và thuộc tính này sẽ trỏ đến function đó, nghĩa là các function sẽ được lưu giữ trong các VO trước khi đoạn code trong function đó được thực hiện hay nói cách khác ta có thể sử dụng function đó trước khi chúng ta thực sự khai báo. Nhưng khi ta khai báo một biến, một thuộc tính sẽ được tạo ra trong VO và thuộc tính này sẽ có giá trị là undefined, trong Javascript 2 quá trình này gọi là *Hoisting*.

4.3.3.2. Scope Chain

Scope là phạm vi mà một biến có thể được truy cập tới, trong mỗi function có một Scope của riêng nó, nơi mà các biến được khai báo



4.3.3.3. "This"

Mỗi một EC mới được tạo ra đều có một biến "this", trong một Regular function call biến "this" được trỏ tới GEC, là window object trong browser. Biến "this" chỉ thực sự trỏ tới một object khi nó nằm trong một Method call của object đó.

4.4. Cách JavaScript thực thi trong môi trường ngoài trình duyệt (NodeJs)

Trong Node.js, môi trường thực thi JavaScript được thiết kế để xử lý các tác vụ phía máy chủ một cách hiệu quả, đặc biệt là các thao tác I/O (Input/Output) không chặn (non-blocking I/O).

4.4.1. Engine V8 của Google

Trái tim của Node.js là **engine V8** của Google, cùng loại engine được sử dụng trong trình duyệt Chrome. V8 có trách nhiệm:

- **Biên dịch (Compilation)**: Chuyển đổi mã JavaScript thành mã máy (machine code) trực tiếp thay vì thông qua một trình thông dịch (interpreter). V8 sử dụng kỹ thuật **Just-In-Time (JIT) Compilation** để biên dịch mã JavaScript trong thời gian chạy, giúp tối ưu hóa hiệu suất.
- **Thực thi (Execution)**: Chạy mã máy đã được biên dịch.
- **Quản lý bộ nhớ (Memory Management)**: V8 có một **garbage collector** để tự động giải phóng bộ nhớ không còn được sử dụng, giúp ngăn chặn rò rỉ bộ nhớ (memory leaks).
- **Quản lý Call Stack (Call Stack Management)**: V8 duy trì một **Call Stack** (hay Execution Stack) để theo dõi các hàm đang được thực thi. Khi một hàm được gọi, nó được đẩy vào stack, và khi hoàn thành, nó được bật ra khỏi stack.

4.4.2. Libuv: Thư viện I/O Không Chặn

Trong khi JavaScript bản chất là đơn luồng (single-threaded) ở cấp độ thực thi mã người dùng, Node.js sử dụng thư viện **libuv** để thực hiện các thao tác I/O bất đồng bộ và không chặn. Đây là một điểm cực kỳ quan trọng làm nên hiệu suất của Node.js:

- **Non-blocking I/O**: libuv cho phép Node.js thực hiện các tác vụ như đọc/ghi file, truy vấn cơ sở dữ liệu, yêu cầu mạng mà không làm chặn luồng chính của JavaScript. Thay vì chờ đợi tác vụ I/O hoàn thành, Node.js sẽ ủy quyền nó cho libuv và tiếp tục thực thi các mã khác.
- **Thread Pool (Luồng Pool)**: Đối với một số tác vụ I/O tốn thời gian hoặc các hoạt động tính toán nặng nề (ví dụ: cryptographic operations), libuv sử dụng một nhóm các luồng (thread pool) bên trong. Các tác vụ này được đẩy vào thread pool để thực hiện song song, giải phóng luồng chính của JavaScript. Khi tác vụ hoàn thành, kết quả sẽ được đưa trở lại luồng chính thông qua Event Loop.
- **Cross-platform**: libuv cung cấp một lớp trừu tượng (abstraction layer) cho các hệ thống I/O cơ bản của hệ điều hành (Windows, macOS, Linux), giúp Node.js hoạt động nhất quán trên các nền tảng khác nhau.

4.4.3. Event Loop (Vòng Lặp Sự Kiện)

Event Loop là cơ chế trung tâm giúp Node.js xử lý đồng thời (concurrency) các tác vụ bất đồng bộ một cách hiệu quả, mặc dù nó chỉ có một luồng thực thi JavaScript chính. Event Loop liên tục kiểm tra **Call Stack** và **Callback Queue** (hay Task Queue):

- **Call Stack**: Nơi lưu trữ các hàm JavaScript đang được thực thi.
- **Web APIs (trong trình duyệt) vs. Node.js APIs**: Trong Node.js, thay vì Web APIs như trình duyệt (DOM, setTimeout của trình duyệt), chúng ta có **Node.js APIs** (ví dụ: fs.readFile, http.get, setTimeout, setImmediate, process.nextTick). Khi các hàm này được gọi, chúng được ủy quyền cho libuv (hoặc các module nội bộ C++ khác) để xử lý bất đồng bộ.
- **Callback Queue (Task Queue / Message Queue)**: Khi một tác vụ bất đồng bộ được hoàn thành (ví dụ: file đã được đọc xong, request HTTP đã nhận được phản hồi), hàm callback liên quan của nó sẽ được đưa vào Callback Queue.
- **Microtask Queue (Job Queue)**: Đây là một hàng đợi có độ ưu tiên cao hơn Callback Queue, thường chứa các **Promises** (.then(), .catch(), .finally()) và process.nextTick(). Event Loop sẽ ưu tiên xử lý tất cả microtasks trước khi chuyển sang các macrotasks trong Callback Queue.
- **Quá trình hoạt động**:

1. Event Loop liên tục kiểm tra xem **Call Stack** có trống không.
2. Nếu Call Stack trống, Event Loop sẽ kiểm tra **Microtask Queue**. Nó sẽ đưa tất cả các microtask từ hàng đợi này vào Call Stack và thực thi chúng.
3. Sau khi Microtask Queue trống, Event Loop sẽ kiểm tra **Callback Queue**. Nó sẽ lấy một macrotask (callback) từ đầu hàng đợi và đẩy vào Call Stack để thực thi.
4. Quá trình này lặp lại liên tục, cho phép Node.js xử lý nhiều tác vụ mà không bị chặn.

4.4.4. Các Module Core của Node.js

Node.js cung cấp một tập hợp phong phú các module tích hợp sẵn được viết bằng C++ và JavaScript, cung cấp các chức năng cốt lõi:

- **fs (File System)**: Để tương tác với hệ thống file (đọc, ghi, xóa file).
- **http / https**: Để tạo máy chủ web hoặc thực hiện các yêu cầu HTTP/HTTPS.
- **path**: Để làm việc với đường dẫn file và thư mục.
- **url**: Để phân tích cú pháp URL.
- **events**: Để làm việc với Event Emitters, một mẫu thiết kế phổ biến trong Node.js.

- **stream**: Để xử lý dữ liệu dưới dạng luồng, rất hiệu quả cho các tác vụ I/O lớn.
- **buffer**: Để làm việc với dữ liệu nhị phân.
- Và nhiều module khác.

4.4.5. npm (Node Package Manager)

Mặc dù không phải là một phần của môi trường thực thi cốt lõi, **npm** là một công cụ không thể thiếu khi làm việc với Node.js. Nó là trình quản lý gói lớn nhất thế giới, cho phép bạn:

- **Cài đặt và quản lý các thư viện (packages)**: Dễ dàng thêm các thư viện của bên thứ ba vào dự án của bạn.
- **Chia sẻ mã nguồn**: Đăng tải các module của riêng bạn để cộng đồng sử dụng.
- **Quản lý dependency**: npm giúp quản lý các phụ thuộc giữa các gói khác nhau.

4.5. Các Paradigms Lập Trình Phổ Biến

4.5.1. Lập trình hướng mệnh lệnh - Imperative programming

Lập trình mệnh lệnh bao gồm các tập hợp các hướng dẫn chi tiết được cung cấp cho máy tính để thực thi theo một thứ tự nhất định. Nó được gọi là "mệnh lệnh" bởi vì là lập trình viên, bạn ra lệnh chính xác những gì máy tính phải làm, theo một cách rất cụ thể.

Lập trình mệnh lệnh tập trung vào việc mô tả *cách* một chương trình hoạt động, từng bước.

4.5.2. Lập trình hướng thủ tục - Procedural programming

Lập trình thủ tục là một dẫn xuất của lập trình mệnh lệnh, thêm vào đó tính năng của các hàm (còn được gọi là "thủ tục" hoặc "chương trình con").

Trong lập trình thủ tục, người dùng được khuyến khích chia nhỏ việc thực thi chương trình thành các chức năng, như một cách để cải thiện tính mô-đun và tổ chức.

4.5.3. Lập trình hướng chức năng - Functional programming

Lập trình hướng chức năng có khái niệm về hàm xa hơn.

Trong lập trình hướng chức năng, các hàm được coi là **first-class citizens**, có nghĩa là chúng có thể được gán cho các biến, được truyền dưới dạng đối số và được trả về từ các hàm khác.

Một khái niệm quan trọng khác là ý tưởng về các **chức năng thuần túy**. Một hàm **thuần túy** là một hàm chỉ dựa vào các đầu vào của nó để tạo ra kết quả của nó. Và được cung cấp cùng một đầu vào, nó sẽ luôn tạo ra cùng một kết quả. Bên cạnh đó, nó không tạo ra tác dụng phụ (bất kỳ thay đổi nào bên ngoài môi trường của chức năng).

Với những khái niệm này, lập trình hướng chức năng khuyến khích các chương trình được viết chủ yếu bằng các hàm . Nó cũng bảo vệ ý tưởng rằng tính mô-đun code và sự vắng mặt của các tác dụng phụ làm cho việc xác định và phân tách các trách nhiệm trong cơ sở code trở nên dễ dàng hơn. Do đó, nó cải thiện khả năng bảo trì code.

4.5.4. Lập trình hướng khai báo - Declarative programming

Lập trình khai báo là tất cả về việc che giấu sự phức tạp và đưa các ngôn ngữ lập trình đến gần hơn với ngôn ngữ và tư duy của con người. Nó đối lập trực tiếp với lập trình mệnh lệnh theo nghĩa là lập trình viên không đưa ra hướng dẫn về cách máy tính sẽ thực thi tác vụ, mà là về kết quả cần thiết.

4.5.5. Lập trình hướng đối tượng

Một trong những mô hình lập trình phổ biến nhất là lập trình hướng đối tượng (OOP).

Khái niệm cốt lõi của OOP là tách các mối quan tâm thành các thực thể được code hóa thành các đối tượng. Mỗi thực thể sẽ nhóm một tập hợp thông tin (thuộc tính) và các hành động (method) nhất định có thể được thực hiện bởi thực thể.

OOP sử dụng nhiều lớp (Class là một cách tạo các đối tượng mới dựa trên bản thiết kế hoặc bản mẫu mà bạn đặt ra). Các đối tượng được tạo từ một lớp được gọi là **instances**.