ĐỀ CƯƠNG ÔN TẬP HỌC PHẦN LẬP TRÌNH WEB

Nội dung

A. THIẾT KẾ WEB FRONTEND

Tài liệu này được biên soạn nhằm mục đích cung cấp đề cương ôn tập chi tiết cho môn học Thiết kế Web. Nội dung bao gồm các kiến thức lý thuyết trọng tâm về HTML, CSS, JavaScript (phía frontend) và các bước căn bản để thiết kế một trang web tĩnh.

Phần lý thuyết sẽ hệ thống hóa các khái niệm, cú pháp và vai trò của từng công nghệ. Đi kèm với đó là ngân hàng 100 câu hỏi ôn tập được trình bày ở dạng bảng. Các câu hỏi được thiết kế để bao quát toàn bộ nội dung học phần.

Phần 1: HTML

HTML (HyperText Markup Language) là ngôn ngữ đánh dấu siêu văn bản, được coi là nền tảng cơ bản nhất của mọi trang web. Nó không phải là một ngôn ngữ lập trình theo nghĩa truyền thống vì không thể tạo ra các chức năng "động", mà chủ yếu dùng để cấu trúc và định dạng nội dung trên trang web. HTML sử dụng các "thẻ" (tags) để mô tả các thành phần như tiêu đề, đoạn văn, liên kết, hình ảnh, ... Phiên bản mới nhất và được sử dụng rộng rãi là HTML5, mang đến nhiều thẻ và thuộc tính mới, đặc biệt là các thẻ ngữ nghĩa giúp cấu trúc trang web rõ ràng hơn.

1.1. Cấu trúc cơ bản của một tài liệu HTML

Một tài liệu HTML điển hình luôn tuân theo một cấu trúc cơ bản để trình duyệt có thể hiểu và hiển thị chính xác. Cấu trúc này bao gồm các thành phần cốt lõi sau:

- Khai báo <!DOCTYPE html>: Đây là khai báo bắt buộc ở dòng đầu tiên của mọi tài liệu HTML5. Nó cho trình duyệt biết rằng trang web đang sử dụng phiên bản HTML5, giúp trình duyệt vào chế độ chuẩn (standards mode) để hiển thị trang một cách nhất quán.
- Phần tử <html>: Là phần tử gốc, bao bọc toàn bộ nội dung của trang web. Thuộc tính lang thường được thêm vào thẻ này để xác định ngôn ngữ chính của tài liệu (ví dụ: <html lang="vi"> cho tiếng Việt).
- Phần tử <head>: Chứa các thông tin meta về tài liệu HTML, không hiển thị trực tiếp trên trang web nhưng quan trọng cho trình duyệt và công cụ tìm kiếm. Các thẻ thường gặp trong <head> bao gồm:
    - <meta>: Cung cấp siêu dữ liệu như bộ ký tự (ví dụ: <meta charset="UTF-8">),

mô tả trang, từ khóa, thông tin viewport cho thiết kế đáp ứng.
    - <title>: Xác định tiêu đề của trang web, hiển thị trên tab trình duyệt hoặc thanh tiêu đề cửa sổ.
    - <link>: Dùng để liên kết đến các tài nguyên bên ngoài, phổ biến nhất là các file CSS.
    - <style>: Chứa mã CSS nội bộ.
    - <script>: Dùng để nhúng hoặc liên kết đến các file JavaScript.
- Phần tử <body>: Chứa toàn bộ nội dung hiển thị của trang web mà người dùng có thể nhìn thấy và tương tác, ví dụ như văn bản, hình ảnh, liên kết, bảng biểu, video, …

Việc phân tách rõ ràng giữa <head> (chứa thông tin mô tả) và <body> (chứa nội dung hiển thị) là một nguyên tắc cơ bản trong cấu trúc HTML, giúp tổ chức mã nguồn một cách logic và dễ bảo trì.

1.2. Các thẻ HTML phổ biến

HTML cung cấp một bộ thẻ phong phú để cấu trúc các loại nội dung khác nhau. Dưới đây là một số thẻ phổ biến và vai trò của chúng:

- Tiêu đề (Headings): Từ <h1> đến <h6> được sử dụng để định nghĩa các cấp độ tiêu đề. <h1> là tiêu đề quan trọng nhất và <h6> là ít quan trọng nhất. Các thẻ này không chỉ giúp định dạng kích thước chữ mà còn quan trọng cho SEO và khả năng truy cập.
- Đoạn văn (Paragraphs): Thẻ <p> dùng để định nghĩa một đoạn văn bản.
- Liên kết (Links/Anchors): Thẻ <a> tạo ra các siêu liên kết đến các trang web khác hoặc các phần khác trong cùng một trang. Thuộc tính quan trọng nhất là href, chỉ định URL đích của liên kết.
- Hình ảnh (Images): Thẻ <img> dùng để nhúng hình ảnh vào trang web. Các thuộc tính bắt buộc bao gồm src (đường dẫn đến file hình ảnh) và alt (văn bản thay thế mô tả hình ảnh, quan trọng cho khả năng truy cập và SEO).
- Danh sách (Lists):
    - (Unordered List): Tạo danh sách không có thứ tự (thường hiển thị với dấu đầu dòng).
    - (Ordered List): Tạo danh sách có thứ tự (thường hiển thị với số hoặc chữ cái).
    - (List Item): Định nghĩa một mục trong danh sách, được sử dụng bên trong <ul> hoặc <ol>.
- Phân chia khối và nội tuyến (Division and Span):

    - : Là một thẻ khối (block-level element), thường được sử dụng như một container để nhóm các phần tử HTML khác lại với nhau nhằm mục đích tạo bố cục hoặc áp dụng CSS.
    - : Là một thẻ nội tuyến (inline element), thường được sử dụng để nhóm các phần tử nội tuyến hoặc một phần văn bản nhỏ nhằm mục đích áp dụng CSS hoặc JavaScript. Sự khác biệt chính giữa block-level và inline elements là block-level elements luôn bắt đầu trên một dòng mới và chiếm toàn bộ chiều rộng có sẵn, trong khi inline elements không bắt đầu trên dòng mới và chỉ chiếm chiều rộng cần thiết cho nội dung của nó.
- Ngắt dòng và Đường kẻ ngang:
    -
: Chèn một dấu ngắt dòng đơn.
    - <hr>: Tạo một đường kẻ ngang, thường dùng để phân tách các phần nội dung.
- Bảng (Tables): Thẻ <table> dùng để tạo bảng dữ liệu. Các thẻ con phổ biến bao gồm <tr> (table row - hàng), <td> (table data - ô dữ liệu), <th> (table header - ô tiêu đề), <thead>, <tbody>, <tfoot> để cấu trúc bảng một cách rõ ràng hơn.

1.3. Thuộc tính HTML (Attributes)

Thuộc tính HTML cung cấp thông tin bổ sung về các phần tử HTML. Chúng luôn được chỉ định trong thẻ mở và thường đi theo cặp tên/giá trị như name="value".

- Khái niệm và Cú pháp: Mỗi thuộc tính có một tên và một giá trị. Tên thuộc tính xác định loại thông tin được cung cấp, và giá trị là dữ liệu cụ thể cho thuộc tính đó. Ví dụ, trong <img src="image.jpg" alt="Mô tả hình ảnh">, src và alt là tên thuộc tính, còn "image.jpg" và "Mô tả hình ảnh" là các giá trị tương ứng.
- Các thuộc tính cốt lõi (Core Attributes):
    - id: Cung cấp một định danh duy nhất cho một phần tử HTML trong một trang. Giá trị của id phải là duy nhất. Nó thường được sử dụng bởi CSS để áp dụng kiểu cụ thể cho một phần tử và bởi JavaScript để thao tác với phần tử đó.
    - class: Chỉ định một hoặc nhiều tên lớp cho một phần tử. Nhiều phần tử có thể chia sẻ cùng một tên lớp. Thuộc tính class rất quan trọng cho việc áp dụng CSS cho một nhóm các phần tử và cũng được JavaScript sử dụng để chọn và thao tác với các nhóm phần tử. Một phần tử có thể có nhiều lớp, cách nhau bởi khoảng trắng, ví dụ: class="tenClass1 tenClass2".
    - style: Cho phép áp dụng các quy tắc CSS nội tuyến trực tiếp cho một phần tử. Ví dụ: <p style="color: blue; font-size: 16px;">. Mặc dù tiện lợi cho các thay đổi nhỏ, việc sử dụng rộng rãi thuộc tính style không được khuyến khích vì nó trộn

lẫn cấu trúc và trình bày, làm giảm khả năng bảo trì.
    - title: Cung cấp thông tin bổ sung về một phần tử, thường hiển thị dưới dạng tooltip khi người dùng di chuột qua phần tử đó.
- Các thuộc tính quan trọng khác:
    - alt (cho thẻ <img>): Cung cấp văn bản thay thế cho hình ảnh nếu hình ảnh không thể hiển thị. Điều này rất quan trọng cho khả năng truy cập (accessibility) cho người dùng sử dụng trình đọc màn hình và cũng có lợi cho SEO.
    - href (cho thẻ <a>): Chỉ định URL đích của một siêu liên kết.
    - src (cho thẻ <img>, <script>, <audio>, <video>): Chỉ định nguồn (URL) của tài nguyên cần nhúng hoặc liên kết.

Việc hiểu và sử dụng đúng các thuộc tính HTML là rất quan trọng để tạo ra các trang web có cấu trúc tốt, dễ dàng tạo kiểu và tương tác.

1.4. HTML Forms (Biểu mẫu)

Biểu mẫu HTML là một thành phần thiết yếu của các trang web tương tác, cho phép người dùng nhập và gửi dữ liệu đến máy chủ.

- Thẻ <form>: Là thẻ bao bọc tất cả các phần tử của một biểu mẫu. Hai thuộc tính quan trọng của thẻ <form> là:
    - action: Chỉ định URL của trang hoặc script trên máy chủ sẽ xử lý dữ liệu được gửi từ biểu mẫu.
    - method: Chỉ định phương thức HTTP được sử dụng để gửi dữ liệu, phổ biến nhất là GET và POST.
        - GET: Dữ liệu biểu mẫu được nối vào URL dưới dạng các cặp tên/giá trị. Thường dùng cho các biểu mẫu không gây ra tác dụng phụ (ví dụ: tìm kiếm).
        - POST: Dữ liệu biểu mẫu được gửi trong phần thân của yêu cầu HTTP. Thường dùng cho các biểu mẫu gửi dữ liệu nhạy cảm hoặc dữ liệu lớn, hoặc khi việc gửi biểu mẫu sẽ thay đổi trạng thái trên máy chủ (ví dụ: đăng ký, đăng nhập).
- Thẻ <input>: Là thẻ đa năng nhất trong biểu mẫu, loại trường nhập liệu được xác định bởi thuộc tính type. Một số giá trị type phổ biến bao gồm:
    - text: Trường nhập văn bản một dòng.
    - password: Trường nhập mật khẩu, các ký tự nhập vào được che đi.
    - checkbox: Hộp kiểm, cho phép người dùng chọn một hoặc nhiều tùy chọn.
    - radio: Nút radio, cho phép người dùng chọn một tùy chọn duy nhất từ một nhóm.

Các nút radio trong cùng một nhóm phải có cùng giá trị cho thuộc tính name.
    - submit: Nút gửi biểu mẫu.
    - button: Nút bấm thông thường, có thể được sử dụng với JavaScript.
    - file: Cho phép người dùng chọn một tệp để tải lên.
    - date: Trường chọn ngày.
    - email: Trường nhập địa chỉ email, có kiểm tra định dạng cơ bản.
    - number: Trường nhập số.
- Thẻ <label>: Cung cấp nhãn mô tả cho một phần tử biểu mẫu. Thuộc tính for của <label> được liên kết với thuộc tính id của phần tử biểu mẫu tương ứng, giúp cải thiện khả năng truy cập và cho phép người dùng nhấp vào nhãn để focus vào trường nhập liệu.
- Thẻ <textarea>: Tạo một trường nhập văn bản nhiều dòng.
- Thẻ <select> và <option>:
    - <select>: Tạo một danh sách thả xuống.
    - <option>: Định nghĩa một tùy chọn trong danh sách thả xuống, được đặt bên trong thẻ <select>.
- Thẻ <button>: Tạo một nút bấm. Thuộc tính type của nó có thể là button, submit, hoặc reset.
- Thuộc tính name: Quan trọng cho các phần tử biểu mẫu vì nó được sử dụng làm tên cho dữ liệu khi gửi đến máy chủ.
- Thuộc tính value: Xác định giá trị ban đầu hoặc giá trị được gửi đi của một phần tử biểu mẫu.
- Thuộc tính required: Nếu có mặt, thuộc tính này chỉ định rằng trường nhập liệu đó phải được điền trước khi gửi biểu mẫu.

1.5. Thẻ HTML5 ngữ nghĩa (Semantic Elements)

HTML5 giới thiệu nhiều thẻ ngữ nghĩa mới, giúp mô tả cấu trúc của trang web một cách rõ ràng và có ý nghĩa hơn cho cả trình duyệt và lập trình viên, cũng như các công nghệ hỗ trợ (như trình đọc màn hình) và công cụ tìm kiếm. Việc sử dụng thẻ ngữ nghĩa thay vì chỉ dùng các thẻ và chung chung giúp cải thiện khả năng truy cập, SEO và khả năng bảo trì mã nguồn.

- <header>: Đại diện cho phần đầu của một tài liệu hoặc một section. Thường chứa tiêu đề, logo, thông tin giới thiệu, hoặc các liên kết điều hướng. Một tài liệu có thể có nhiều thẻ <header>.
- <nav>: Định nghĩa một tập hợp các liên kết điều hướng chính của trang web, chẳng

hạn như menu chính.

- <main>: Xác định nội dung chính, độc nhất của tài liệu. Nội dung bên trong <main> phải là trung tâm của trang và không nên lặp lại trên các trang khác (ví dụ: không bao gồm thanh bên, menu điều hướng, logo, thông tin bản quyền). Nên chỉ có một thẻ <main> trên mỗi trang.
- <article>: Đại diện cho một mẩu nội dung độc lập, có thể tự nó tồn tại và được phân phối riêng biệt, ví dụ như một bài đăng blog, một bài báo, một bình luận của người dùng.
- <section>: Biểu thị một phần (section) chung trong tài liệu, thường có một tiêu đề riêng. Nó nhóm các nội dung có liên quan về mặt chủ đề. Một <article> có thể chứa nhiều <section>, và ngược lại, một <section> cũng có thể chứa nhiều <article> tùy theo ngữ cảnh.
- <aside>: Chứa nội dung có liên quan một cách gián tiếp đến nội dung chính xung quanh nó, ví dụ như thanh bên (sidebar), các ghi chú, quảng cáo, hoặc thông tin tiểu sử tác giả.
- <footer>: Đại diện cho phần chân của một tài liệu hoặc một section. Thường chứa thông tin về tác giả, bản quyền, liên kết liên quan, hoặc thông tin liên hệ.
- <figure> và <figcaption>:
  - <figure>: Dùng để bao bọc nội dung độc lập, thường là hình ảnh, sơ đồ, đoạn mã, trích dẫn, ..., có thể được tham chiếu từ nội dung chính.
  - <figcaption>: Cung cấp chú thích hoặc tiêu đề cho nội dung bên trong thẻ <figure>.

Sử dụng các thẻ này không chỉ làm cho mã HTML dễ đọc và dễ hiểu hơn mà còn giúp các công cụ tự động (như trình đọc màn hình cho người khiếm thị, hoặc các bot của công cụ tìm kiếm) hiểu rõ hơn về cấu trúc và ý nghĩa của nội dung trang web.

Phần 2: CSS

CSS (Cascading Style Sheets) là một ngôn ngữ được sử dụng để mô tả cách trình bày và định dạng giao diện của các tài liệu được viết bằng HTML hoặc XML (bao gồm cả các ngôn ngữ dựa trên XML như SVG hoặc XHTML). CSS cho phép tách biệt nội dung (HTML) khỏi phần trình bày (CSS), giúp mã nguồn dễ quản lý, bảo trì và linh hoạt hơn.

2.1. Giới thiệu CSS: Cú pháp và cách nhúng CSS vào HTML

- Cú pháp CSS (CSS Syntax): Một quy tắc CSS (CSS Rule) bao gồm hai phần chính:

1. Bộ chọn (Selector): Là một mẫu (pattern) dùng để xác định (các) phần tử HTML nào sẽ được áp dụng kiểu dáng.
2. Khối khai báo (Declaration Block): Được đặt trong cặp dấu ngoặc nhọn {}. Bên trong khối này chứa một hoặc nhiều khai báo (declarations), được phân tách bởi dấu chấm phẩy ;.
    - Mỗi khai báo (Declaration) bao gồm một thuộc tính CSS (Property) và một giá trị (Value), được phân tách bởi dấu hai chấm :. Ví dụ: color: blue;. Thuộc tính và giá trị trong CSS thường không phân biệt chữ hoa chữ thường, nhưng việc viết nhất quán (thường là chữ thường) được coi là thực hành tốt.

- Cách nhúng CSS vào HTML: Có ba cách chính để thêm CSS vào tài liệu HTML:
    1. External CSS (CSS bên ngoài):
        - Các quy tắc CSS được định nghĩa trong một tệp riêng biệt có phần mở rộng là .css (ví dụ: styles.css).
        - Tệp CSS này được liên kết với tài liệu HTML bằng cách sử dụng thẻ <link> đặt trong phần <head> của HTML: <link rel="stylesheet" href="styles.css">. Thuộc tính type="text/css" thường được bỏ qua trong HTML5 vì nó là giá trị mặc định.
        - Đây là phương pháp phổ biến và được khuyến khích nhất vì nó cho phép sử dụng cùng một tệp CSS cho nhiều trang web, giúp dễ dàng bảo trì, quản lý tập trung và tận dụng cơ chế cache của trình duyệt để tải trang nhanh hơn.
    2. Internal CSS (CSS nội bộ):
        - Các quy tắc CSS được định nghĩa bên trong thẻ <style>, thẻ này cũng được đặt trong phần <head> của tài liệu HTML.
        - Kiểu CSS này chỉ ảnh hưởng đến tài liệu HTML mà nó được nhúng vào. Nó hữu ích cho việc định kiểu một trang duy nhất hoặc khi không tiện tạo tệp CSS riêng.
    3. Inline CSS (CSS nội tuyến):
        - Các quy tắc CSS được áp dụng trực tiếp cho một phần tử HTML cụ thể thông qua thuộc tính style. Ví dụ: <p style="color: red; font-size: 16px;">.
        - Inline CSS có độ ưu tiên (specificity) cao nhất, có thể gây khó khăn khi muốn ghi đè kiểu. Nó cũng làm trộn lẫn nội dung và trình bày, khiến mã nguồn khó đọc và bảo trì hơn. Do đó, việc sử dụng inline CSS nên được hạn chế cho các trường hợp đặc biệt hoặc các thay đổi nhỏ, cục bộ.
    4. Quy tắc @import trong CSS:
        - Cho phép nhập một tệp stylesheet này vào một tệp stylesheet khác: @import url("navigation.css");

    - Quy tắc @import phải được đặt ở đầu tệp CSS (hoặc khối <style>).
    - So với việc sử dụng thẻ <link>, @import có thể ảnh hưởng tiêu cực đến hiệu suất tải trang vì nó có thể chặn việc tải xuống song song các tài nguyên. Do đó, thẻ <link> thường được ưu tiên hơn.

Khái niệm "Cascading" (theo tầng) trong CSS là một cơ chế cốt lõi, quy định cách trình duyệt giải quyết các xung đột khi nhiều quy tắc CSS cùng áp dụng cho một phần tử. Các yếu tố chính ảnh hưởng đến quá trình này bao gồm độ ưu tiên của bộ chọn (specificity), tính kế thừa (inheritance), và thứ tự xuất hiện của các quy tắc. Hiểu rõ cơ chế này là rất quan trọng để viết CSS hiệu quả và dễ gỡ lỗi. Lựa chọn cách nhúng CSS cũng ảnh hưởng trực tiếp đến hiệu suất, khả năng bảo trì và mở rộng của trang web.

2.2. Bộ chọn CSS (Selectors)

Bộ chọn CSS là các mẫu được sử dụng để "chọn" hoặc "nhắm mục tiêu" các phần tử HTML mà bạn muốn áp dụng kiểu dáng. Việc sử dụng thành thạo các bộ chọn giúp viết mã CSS hiệu quả, linh hoạt và dễ bảo trì hơn.

- Các loại bộ chọn cơ bản:
    - Type Selector (Bộ chọn theo tên thẻ/phần tử): Chọn tất cả các phần tử HTML có cùng tên thẻ. Ví dụ: h1 sẽ chọn tất cả các thẻ <h1>; p sẽ chọn tất cả các thẻ <p>.
    - ID Selector (Bộ chọn theo ID): Chọn một phần tử duy nhất dựa trên giá trị của thuộc tính id. Cú pháp: #ten_id. Ví dụ: #header sẽ chọn phần tử có id="header". Giá trị id phải là duy nhất trong một trang HTML.
    - Class Selector (Bộ chọn theo Class): Chọn tất cả các phần tử có thuộc tính class chứa một tên lớp cụ thể. Cú pháp: .ten_class. Ví dụ: .highlight sẽ chọn tất cả các phần tử có class="highlight". Một phần tử có thể có nhiều lớp, và nhiều phần tử có thể chia sẻ cùng một lớp.
    - Attribute Selector (Bộ chọn theo thuộc tính): Chọn các phần tử dựa trên sự hiện diện hoặc giá trị của một thuộc tính. Ví dụ: input[type="text"] chọn tất cả các thẻ <input> có thuộc tính type với giá trị là "text".
    - Universal Selector (Bộ chọn *): Chọn tất cả các phần tử HTML trên trang. Cú pháp: *. Thường được sử dụng trong các tệp CSS reset để loại bỏ kiểu mặc định của trình duyệt.
    - Grouping Selector (Bộ chọn nhóm): Áp dụng cùng một bộ quy tắc CSS cho nhiều bộ chọn khác nhau. Các bộ chọn được phân tách bằng dấu phẩy. Ví dụ: h1, h2, p { color: gray; }.
- Combinators (Bộ kết hợp): Dùng để thiết lập mối quan hệ giữa các bộ chọn.

    - Descendant Combinator (khoảng trắng): Chọn các phần tử là con cháu (nằm bên trong, ở bất kỳ cấp độ nào) của một phần tử được chỉ định. Ví dụ: div p chọn tất cả các phần tử <p> nằm bên trong bất kỳ phần tử nào.
    - Child Combinator (>): Chọn các phần tử là con trực tiếp (con cấp một) của một phần tử được chỉ định. Ví dụ: ul > li chọn tất cả các phần tử <li> là con trực tiếp của phần tử <ul>.
    - Adjacent Sibling Combinator (+): Chọn một phần tử đứng ngay sau một phần tử được chỉ định và cả hai có cùng phần tử cha. Ví dụ: h1 + p chọn phần tử <p> đầu tiên đứng ngay sau một phần tử <h1>.
    - General Sibling Combinator (~): Chọn tất cả các phần tử là anh em (cùng cấp, cùng cha) đứng sau một phần tử được chỉ định. Ví dụ: h1 ~ p chọn tất cả các phần tử <p> đứng sau <h1> và có cùng cha với <h1>.
- Pseudo-classes (Lớp giả): Chọn các phần tử dựa trên trạng thái hoặc vị trí của chúng trong cấu trúc tài liệu, mà không cần thêm class hay id vào HTML. Cú pháp: selector:pseudo-class.
    - Trạng thái liên kết: :link (liên kết chưa được truy cập), :visited (liên kết đã được truy cập).
    - Trạng thái tương tác người dùng: :hover (khi di chuột qua), :active (khi được nhấp), :focus (khi nhận focus, ví dụ trường input).
    - Cấu trúc/Vị trí: :first-child (con đầu tiên), :last-child (con cuối cùng), :nth-child(n) (con thứ n), :only-child (con duy nhất).
    - Trạng thái input: :checked, :disabled, :enabled, :required.
- Pseudo-elements (Phần tử giả): Cho phép tạo kiểu cho các phần cụ thể của một phần tử, ngay cả khi chúng không phải là các phần tử HTML riêng biệt. Cú pháp CSS3 khuyến nghị dùng hai dấu hai chấm :: để phân biệt với pseudo-classes (ví dụ: selector::pseudo-element), mặc dù một số trình duyệt vẫn hỗ trợ cú pháp cũ với một dấu hai chấm.
    - ::before: Chèn nội dung vào trước nội dung của phần tử được chọn.
    - ::after: Chèn nội dung vào sau nội dung của phần tử được chọn.
    - ::first-letter: Áp dụng kiểu cho chữ cái đầu tiên của một khối văn bản.
    - ::first-line: Áp dụng kiểu cho dòng đầu tiên của một khối văn bản.

Việc sử dụng đa dạng các bộ chọn CSS cho phép kiểm soát giao diện một cách chi tiết mà không làm phức tạp mã HTML. Tuy nhiên, cần lưu ý rằng việc lạm dụng các bộ chọn quá cụ thể (ví dụ, chuỗi dài các bộ chọn con cháu hoặc sử dụng ID cho mục đích tạo kiểu quá nhiều) có thể dẫn đến CSS khó ghi đè và bảo trì, gây ra "cuộc chiến độ ưu tiên" (specificity wars). Do đó, một thực hành tốt là giữ độ ưu tiên của bộ chọn ở mức thấp

nhất có thể mà vẫn đạt được mục tiêu, thường ưu tiên sử dụng các bộ chọn dựa trên class để tăng tính tái sử dụng.

2.3. Mô hình hộp (Box Model) trong CSS

Mọi phần tử HTML trên trang web đều có thể được coi là một "hộp" hình chữ nhật. Mô hình hộp CSS mô tả cách các hộp này được hiển thị, cách tính toán kích thước (chiều rộng, chiều cao) và khoảng cách của chúng.

- Các thành phần của Box Model:
  - Content (Nội dung): Khu vực chứa nội dung thực sự của phần tử, như văn bản, hình ảnh. Kích thước của nó có thể được kiểm soát bởi các thuộc tính width và height.
  - Padding (Đệm): Là khoảng không gian trong suốt bao quanh vùng nội dung, nằm bên trong đường viền. Padding tạo ra khoảng trống giữa nội dung và đường viền. Được kiểm soát bởi các thuộc tính padding, padding-top, padding-right, padding-bottom, padding-left.
  - Border (Đường viền): Là đường viền bao quanh padding và nội dung. Được kiểm soát bởi các thuộc tính border, border-width, border-style, border-color, và các thuộc tính riêng cho từng cạnh (ví dụ: border-top-width).
  - Margin (Lề): Là khoảng không gian trong suốt nằm bên ngoài đường viền. Margin tạo ra khoảng cách giữa hộp hiện tại và các phần tử khác xung quanh nó. Được kiểm soát bởi các thuộc tính margin, margin-top, margin-right, margin-bottom, margin-left.
- Tính toán Kích thước và box-sizing:
  - box-sizing: content-box; (mặc định): Khi bạn đặt giá trị cho width và height, các giá trị này chỉ áp dụng cho vùng content của phần tử. Tổng chiều rộng thực tế của phần tử sẽ là width (của content) + padding (trái + phải) + border (trái + phải). Tương tự cho chiều cao. Điều này có nghĩa là nếu bạn thêm padding hoặc border, kích thước tổng thể của phần tử trên màn hình sẽ tăng lên, có thể gây khó khăn khi tính toán layout.
  - box-sizing: border-box;: Khi sử dụng giá trị này, thuộc tính width và height sẽ xác định kích thước tổng thể của phần tử, bao gồm cả content, padding và border. Vùng content sẽ tự động co lại để nhường chỗ cho padding và border. Ví dụ, nếu bạn đặt width: 200px; và padding: 10px; border: 1px solid black; với box-sizing: border-box;, thì tổng chiều rộng hiển thị của phần tử vẫn là 200px. Mô hình này thường được coi là trực quan và dễ làm việc hơn, đặc biệt khi xây dựng các

layout phức tạp và đáp ứng, vì kích thước bạn chỉ định chính là kích thước bạn thấy trên màn hình. Nhiều nhà phát triển thường đặt * { box-sizing: border-box; } như một quy tắc reset phổ biến.

- Margin Collapsing (Gộp lề): Đây là một hành vi đặc trưng của CSS, nơi các lề dọc (top và bottom margins) của các phần tử khối (block-level elements) liền kề nhau có thể "gộp" lại thành một lề duy nhất. Kích thước của lề gộp này sẽ bằng giá trị lớn nhất trong số các lề riêng lẻ (hoặc giá trị lớn nhất dương và nhỏ nhất âm nếu có lề âm). Hiện tượng này không xảy ra với lề ngang, hoặc với các phần tử được float hay định vị tuyệt đối (absolutely positioned). Hiểu rõ margin collapsing giúp tránh những khoảng trống không mong muốn hoặc khó hiểu trong layout.
  Việc nắm vững Mô hình hộp, đặc biệt là sự khác biệt giữa content-box và border-box cùng với hiện tượng gộp lề, là nền tảng để giải quyết nhiều vấn đề phổ biến về layout trong CSS. Các công cụ phát triển của trình duyệt (developer tools) thường cung cấp một giao diện trực quan để kiểm tra các thành phần của box model cho từng phần tử, đây là một công cụ gỡ lỗi vô cùng hữu ích.

2.4. Định dạng Văn bản, Màu sắc, Nền và Đường viền

CSS cung cấp một loạt các thuộc tính để kiểm soát chi tiết giao diện của văn bản, màu sắc, nền và đường viền của các phần tử HTML.

- Định dạng Văn bản (Text Formatting):
  - color: Thiết lập màu cho văn bản.
  - font-family: Chỉ định một danh sách ưu tiên các họ font (ví dụ: "Arial", "Verdana") và/hoặc các họ font chung (ví dụ: sans-serif, serif). Trình duyệt sẽ sử dụng font đầu tiên trong danh sách mà nó tìm thấy trên hệ thống của người dùng.
  - font-size: Thiết lập kích thước của font chữ (ví dụ: 16px, 1.2em, 100%).
  - font-weight: Thiết lập độ đậm của font (ví dụ: normal, bold, hoặc các giá trị số như 400 cho normal, 700 cho bold).
  - font-style: Thiết lập kiểu của font (ví dụ: normal, italic, oblique).
  - text-align: Căn chỉnh văn bản theo chiều ngang (ví dụ: left, right, center, justify).
  - line-height: Thiết lập khoảng cách giữa các dòng văn bản, có thể cải thiện khả năng đọc.
  - text-decoration: Thêm các đường trang trí cho văn bản (ví dụ: none, underline, overline, line-through). Thuộc tính text-decoration-color có thể dùng để chỉ định màu cho đường trang trí này.
  - text-transform: Kiểm soát việc viết hoa các ký tự (ví dụ: none, capitalize,

uppercase, lowercase).
    - letter-spacing: Điều chỉnh khoảng cách giữa các ký tự.
    - word-spacing: Điều chỉnh khoảng cách giữa các từ.
    - text-shadow: Thêm hiệu ứng đổ bóng cho văn bản.
    - font: Là thuộc tính viết tắt (shorthand) cho phép thiết lập nhiều thuộc tính font cùng lúc (ví dụ: font: italic bold 16px/1.5 Arial, sans-serif;).
- Màu sắc (Colors): CSS hỗ trợ nhiều cách để xác định màu:
    - Tên màu (Named colors): Sử dụng các tên màu được định nghĩa sẵn (ví dụ: red, blue, lightgray). Có khoảng 140 tên màu được hỗ trợ.
    - Mã Hexadecimal (HEX): Dạng #RRGGBB hoặc #RGB (viết tắt). Ví dụ: #FF0000 cho màu đỏ, #00F cho màu xanh dương.
    - RGB / RGBA:
        - rgb(red, green, blue): Giá trị từ 0 đến 255 cho mỗi thành phần màu. Ví dụ: rgb(255, 0, 0).
        - rgba(red, green, blue, alpha): Tương tự RGB nhưng có thêm giá trị alpha (từ 0.0 đến 1.0) để xác định độ trong suốt. Ví dụ: rgba(255, 0, 0, 0.5).
    - HSL / HSLA:
        - hsl(hue, saturation, lightness): Hue là góc trên vòng tròn màu (0-360), saturation và lightness là tỷ lệ phần trăm. Ví dụ: hsl(0, 100%, 50%).
        - hsla(hue, saturation, lightness, alpha): Tương tự HSL với thêm giá trị alpha cho độ trong suốt.
    - Từ khóa currentcolor: Đại diện cho giá trị của thuộc tính color của chính phần tử đó, hữu ích cho việc đồng bộ màu sắc của các thành phần khác (như border) với màu chữ. Sự đa dạng trong cách xác định màu, đặc biệt là với kênh alpha (trong RGBA và HSLA), mang lại khả năng kiểm soát tinh vi về bảng màu và hiệu ứng lớp trong thiết kế web.
- Nền (Backgrounds):
    - background-color: Thiết lập màu nền cho một phần tử.
    - background-image: Thiết lập một hoặc nhiều hình ảnh nền (có thể là URL đến file ảnh hoặc gradient CSS).
    - background-repeat: Kiểm soát việc lặp lại của hình nền (ví dụ: repeat, no-repeat, repeat-x, repeat-y).
    - background-position: Thiết lập vị trí bắt đầu của hình nền (ví dụ: center, top left, 50% 50%).
    - background-attachment: Xác định hình nền sẽ cuộn cùng trang hay cố định (scroll, fixed).

    - background-size: Chỉ định kích thước của hình nền (ví dụ: auto, cover, contain, hoặc kích thước cụ thể).
    - background: Thuộc tính viết tắt để thiết lập nhiều thuộc tính nền cùng lúc.
- Đường viền (Borders):
    - border-width: Thiết lập độ dày của đường viền (ví dụ: thin, medium, thick, hoặc giá trị cụ thể như 1px).
    - border-style: Thiết lập kiểu của đường viền (ví dụ: none, solid, dotted, dashed, double).
    - border-color: Thiết lập màu cho đường viền.
    - border: Thuộc tính viết tắt để thiết lập đồng thời border-width, border-style, và border-color cho cả bốn cạnh (ví dụ: border: 1px solid black;).
    - Có thể định dạng riêng cho từng cạnh: border-top, border-right, border-bottom, border-left và các thuộc tính con của chúng (ví dụ: border-top-width, border-left-style).
    - border-radius: Dùng để tạo các góc bo tròn cho phần tử.
    - box-shadow: Thêm hiệu ứng đổ bóng xung quanh khung của phần tử.

Các thuộc tính viết tắt như font, background, border rất hữu ích để viết CSS ngắn gọn. Tuy nhiên, cần hiểu rõ các thuộc tính thành phần mà chúng đại diện và giá trị mặc định của chúng. Nếu bỏ sót một giá trị trong thuộc tính viết tắt, thuộc tính thành phần tương ứng có thể bị reset về giá trị ban đầu, có khả năng ghi đè lên một khai báo riêng lẻ trước đó. Điều này đòi hỏi người học phải nắm vững cả cú pháp viết tắt và các thuộc tính riêng lẻ để sử dụng chúng một cách hiệu quả.

2.5. Kỹ thuật Bố cục CSS (CSS Layout Techniques)

CSS cung cấp nhiều kỹ thuật để sắp xếp và định vị các phần tử trên trang web, từ các phương pháp truyền thống đến các module hiện đại như Flexbox và Grid.

- Thuộc tính display: Thuộc tính này xác định cách một phần tử được hiển thị và hành vi bố cục của nó. Một số giá trị quan trọng:
    - block: Phần tử tạo ra một hộp khối, bắt đầu trên một dòng mới và chiếm toàn bộ chiều rộng có sẵn. Các thuộc tính width và height được tôn trọng.
    - inline: Phần tử tạo ra một hộp nội tuyến, không bắt đầu trên dòng mới và chỉ chiếm chiều rộng cần thiết cho nội dung. width và height không áp dụng; padding, margin, border theo chiều dọc không ảnh hưởng đến bố cục của các phần tử nội tuyến xung quanh.
    - inline-block: Phần tử được định dạng như một khối nội tuyến. Nó chảy cùng nội

dung xung quanh như một phần tử inline, nhưng width, height, margin, padding, và border được tôn trọng như một phần tử block.
    - none: Phần tử bị loại bỏ hoàn toàn khỏi luồng tài liệu và không được hiển thị.
    - flex: Kích hoạt Flexbox layout cho các con trực tiếp của phần tử.
    - grid: Kích hoạt CSS Grid layout cho các con trực tiếp của phần tử.
- Thuộc tính position: Chỉ định phương thức định vị được sử dụng cho một phần tử. Thường được sử dụng cùng với các thuộc tính top, right, bottom, left, và z-index.
    - static: Giá trị mặc định. Phần tử được định vị theo luồng bình thường của tài liệu. Các thuộc tính top, right, bottom, left, z-index không có tác dụng.
    - relative: Phần tử được định vị theo luồng bình thường, sau đó được dịch chuyển tương đối so với vị trí bình thường của nó bằng các thuộc tính top, right, bottom, left. Không gian mà phần tử chiếm giữ trong luồng bình thường vẫn được bảo toàn.
    - absolute: Phần tử bị loại bỏ khỏi luồng tài liệu bình thường. Nó được định vị tương đối so với tổ tiên được định vị gần nhất (tổ tiên có position khác static). Nếu không có tổ tiên nào được định vị, nó sẽ được định vị tương đối so với khối chứa ban đầu (thường là phần tử <html>). Không chiếm không gian trong luồng bình thường.
    - fixed: Phần tử bị loại bỏ khỏi luồng tài liệu bình thường. Nó được định vị tương đối so với viewport của trình duyệt (khung nhìn). Phần tử sẽ giữ nguyên vị trí ngay cả khi trang được cuộn. Không chiếm không gian trong luồng bình thường.
    - sticky: Là sự kết hợp giữa relative và fixed. Phần tử được coi là relative cho đến khi nó cuộn đến một vị trí bù đắp (offset) được chỉ định (ví dụ: top: 0), tại thời điểm đó nó trở thành fixed.
- Floats (Kỹ thuật cũ):
    - float: left; hoặc float: right;
    - Ban đầu được thiết kế để văn bản bao quanh hình ảnh. Sau đó được sử dụng rộng rãi để tạo layout nhiều cột trước khi Flexbox và Grid ra đời.
    - Yêu cầu kỹ thuật "clearing floats" (ví dụ: sử dụng clear: both; hoặc các "clearfix hack") để tránh các vấn đề về layout.
    - Mặc dù ít phổ biến hơn cho layout chính của trang hiện nay, float vẫn hữu ích cho mục đích ban đầu của nó và việc hiểu về float quan trọng khi làm việc với các codebase cũ. Các layout hiện đại nên ưu tiên Flexbox và Grid.
- Flexbox (Flexible Box Layout):
    - Là một mô hình layout một chiều, được thiết kế để phân phối không gian giữa các item trong một container và cung cấp khả năng căn chỉnh mạnh mẽ.

    - Khái niệm chính: Flex container (phần tử cha có display: flex hoặc display: inline-flex) và flex items (các con trực tiếp của flex container).
    - Thuộc tính của Container: flex-direction (xác định trục chính: row, column, etc.) 40, flex-wrap (cho phép các item xuống dòng), justify-content (căn chỉnh item dọc theo trục chính), align-items (căn chỉnh item dọc theo trục phụ), align-content (căn chỉnh các dòng item khi có nhiều dòng).
    - Thuộc tính của Item: order (thay đổi thứ tự hiển thị), flex-grow (khả năng co giãn để lấp đầy không gian), flex-shrink (khả năng co lại khi không đủ không gian), flex-basis (kích thước mặc định), align-self (ghi đè căn chỉnh mặc định cho item cụ thể).
- CSS Grid Layout:
    - Là một hệ thống layout hai chiều cho web, cho phép sắp xếp nội dung thành các hàng và cột một cách rõ ràng.
    - Khái niệm chính: Grid container (phần tử cha có display: grid hoặc display: inline-grid), grid items (các con trực tiếp), grid lines (đường lưới), grid tracks (hàng/cột), grid cells (ô lưới), grid areas (vùng lưới).
    - Thuộc tính của Container: grid-template-columns, grid-template-rows (định nghĩa cấu trúc cột và hàng với các đơn vị như px, %, fr - đơn vị phần), grid-template-areas (định nghĩa các vùng lưới có tên), gap (hoặc grid-gap, row-gap, column-gap - khoảng cách giữa các track), justify-items, align-items (căn chỉnh item bên trong cell), justify-content, align-content (căn chỉnh toàn bộ lưới bên trong container).
    - Thuộc tính của Item: grid-column-start, grid-column-end, grid-row-start, grid-row-end (xác định vị trí và kích thước item dựa trên đường lưới), grid-column, grid-row (viết tắt), grid-area (đặt item vào vùng có tên hoặc viết tắt cho vị trí).

Flexbox và CSS Grid không loại trừ lẫn nhau mà bổ sung cho nhau. Flexbox rất mạnh cho các layout một chiều (hoặc hàng hoặc cột), lý tưởng cho các thành phần như thanh điều hướng, danh sách các mục. Trong khi đó, Grid vượt trội cho các layout hai chiều phức tạp (cả hàng và cột đồng thời), phù hợp cho cấu trúc tổng thể của trang. Thường thì một grid item có thể lại là một flex container để sắp xếp các phần tử con bên trong nó. Sự phát triển từ float/positioning đến Flexbox và Grid phản ánh nỗ lực không ngừng nhằm cung cấp cho nhà phát triển web những công cụ mạnh mẽ, trực quan và đáng tin cậy hơn để tạo ra các thiết kế web phức tạp và đáp ứng.

2.6. Thiết kế Web Responsive và Media Queries

Thiết kế Web Đáp ứng (RWD) là một phương pháp thiết kế nhằm mục đích làm cho các trang web hiển thị tốt trên nhiều loại thiết bị và kích thước màn hình khác nhau, từ máy tính để bàn lớn đến điện thoại di động nhỏ. Mục tiêu là đảm bảo trải nghiệm người dùng tốt và tính khả dụng trên mọi thiết bị bằng cách tự động điều chỉnh bố cục và nội dung. RWD không phải là một công nghệ đơn lẻ mà là một tập hợp các thực hành và kỹ thuật.

- Các yếu tố chính của RWD:
  - Lưới linh hoạt (Fluid Grids / Flexible Layouts): Sử dụng các đơn vị tương đối như phần trăm (%) hoặc đơn vị fr (trong CSS Grid) cho các phần tử layout, thay vì các đơn vị cố định như pixel (px), để chúng có thể thay đổi kích thước một cách tỷ lệ.
  - Hình ảnh/Media linh hoạt (Flexible Images/Media): Đảm bảo hình ảnh và các đối tượng media khác co giãn vừa vặn trong phần tử chứa chúng. Kỹ thuật phổ biến là sử dụng max-width: 100%; height: auto; cho hình ảnh, giúp chúng thu nhỏ khi container hẹp lại nhưng không vượt quá kích thước gốc khi container rộng ra.
  - Media Queries: Là tính năng cốt lõi của CSS cho phép áp dụng các kiểu dáng khác nhau dựa trên các đặc điểm của thiết bị, chủ yếu là chiều rộng của viewport.
- Thẻ Meta Viewport:
  - Rất cần thiết cho RWD trên thiết bị di động: <meta name="viewport" content="width=device-width, initial-scale=1.0">
  - width=device-width: Thiết lập chiều rộng của viewport bằng chiều rộng màn hình của thiết bị.
  - initial-scale=1.0: Thiết lập mức thu phóng ban đầu khi trang được tải lần đầu.
  - Nếu không có thẻ này, các trình duyệt di động thường hiển thị trang web ở chiều rộng của màn hình máy tính để bàn rồi thu nhỏ lại, khiến nội dung khó đọc và tương tác.
- Media Queries:
  - Cú pháp: @media media-type and (media-feature-rule) { /* CSS rules */ }.
  - media-type: Chỉ định loại phương tiện (ví dụ: screen cho màn hình máy tính/điện thoại, print cho máy in, all cho tất cả). Tham số này là tùy chọn, mặc định là all.
  - media-feature-rule: Là một điều kiện phải được đáp ứng để các quy tắc CSS bên trong được áp dụng. Các đặc điểm phổ biến:
    - width, min-width, max-width: Dựa trên chiều rộng của viewport (phổ biến nhất cho RWD).
    - height, min-height, max-height: Dựa trên chiều cao của viewport.
    - orientation: portrait (dọc) hoặc landscape (ngang).

        - hover: Phát hiện xem cơ chế nhập liệu chính có khả năng di chuột qua các phần tử hay không (giúp phân biệt người dùng chuột và người dùng cảm ứng).
    - Toán tử logic: and (và), not (phủ định), , (dấu phẩy, tương đương OR - hoặc).
- Breakpoints (Điểm ngắt): Là các điểm cụ thể (thường là chiều rộng viewport) mà tại đó layout của trang web thay đổi thông qua media queries. Việc lựa chọn breakpoints nên dựa trên nhu cầu của nội dung và thiết kế, thay vì dựa trên kích thước của các thiết bị cụ thể.
    - Phương pháp Mobile-First: Thiết kế cho màn hình nhỏ (di động) trước, sau đó thêm các cải tiến và độ phức tạp cho màn hình lớn hơn thông qua media queries sử dụng min-width. Đây là cách tiếp cận được khuyến khích vì nó giúp tập trung vào nội dung cốt lõi và tối ưu hóa hiệu suất cho các thiết bị có tài nguyên hạn chế hơn.

Lợi ích của RWD bao gồm cải thiện trải nghiệm người dùng trên mọi thiết bị, tối ưu hóa cho công cụ tìm kiếm (SEO), tiết kiệm chi phí phát triển và bảo trì (chỉ cần một phiên bản website). Sự kết hợp giữa các kỹ thuật layout linh hoạt như Flexbox và Grid cùng với Media Queries là chìa khóa để tạo ra các thiết kế web đáp ứng tinh vi và dễ bảo trì.

Phần 3: JavaScript

JavaScript là một ngôn ngữ lập trình kịch bản, nhẹ, được thông dịch hoặc biên dịch just-in-time, và có các hàm hạng nhất (first-class functions). Nó nổi tiếng nhất với vai trò là ngôn ngữ kịch bản cho các trang Web, cho phép tạo ra các trang web động và tương tác. Cùng với HTML và CSS, JavaScript là một trong ba công nghệ cốt lõi của World Wide Web.

3.1. Giới thiệu JavaScript: Vai trò và cách thêm JavaScript vào trang HTML

- Vai trò của JavaScript trong Thiết kế Web:
    - Tạo tương tác: Thêm các yếu tố tương tác vào trang web như menu thả xuống, slideshow hình ảnh, hiệu ứng khi di chuột, ...
    - Thao tác DOM (Document Object Model): Thay đổi nội dung HTML và cấu trúc của trang web một cách linh hoạt sau khi trang đã tải.
    - Thay đổi CSS: Điều chỉnh kiểu dáng CSS của các phần tử một cách động.
    - Xử lý sự kiện: Phản hồi lại các hành động của người dùng như nhấp chuột, nhấn phím, di chuyển chuột.
    - Kiểm tra dữ liệu Form (Client-side Validation): Kiểm tra tính hợp lệ của dữ liệu

người dùng nhập vào biểu mẫu ngay tại trình duyệt trước khi gửi lên máy chủ, giúp giảm tải cho máy chủ và cung cấp phản hồi nhanh hơn cho người dùng.

- Yêu cầu bất đồng bộ (Asynchronous Requests - AJAX): Gửi và nhận dữ liệu từ máy chủ mà không cần tải lại toàn bộ trang, cho phép cập nhật các phần của trang một cách linh hoạt.

- Cách thêm JavaScript vào trang HTML:
  1. Internal JavaScript (JS nội bộ): Mã JavaScript được đặt trực tiếp bên trong cặp thẻ <script> và </script> trong tài liệu HTML.
     - Có thể đặt trong phần <head> hoặc <body>. Việc đặt các thẻ <script> ở cuối phần <body> (ngay trước thẻ đóng </body>) thường được khuyến nghị. Điều này cho phép nội dung HTML được phân tích cú pháp và hiển thị trước, cải thiện tốc độ tải trang cảm nhận được. Nếu một script trong <head> cố gắng truy cập các phần tử DOM chưa được tải, nó sẽ thất bại.
     - Ví dụ: <script> alert('Hello World!'); </script>.
  2. External JavaScript (JS bên ngoài): Mã JavaScript được viết trong một tệp riêng biệt có phần mở rộng là .js (ví dụ: myscript.js).
     - Tệp này được liên kết với tài liệu HTML bằng cách sử dụng thuộc tính src của thẻ <script>: <script src="myscript.js"></script>.
     - Đây là phương pháp được ưu tiên cho các đoạn mã lớn vì nó giúp tổ chức mã tốt hơn, có thể tái sử dụng trên nhiều trang HTML và cho phép trình duyệt lưu trữ tệp .js vào bộ nhớ đệm, giúp tải trang nhanh hơn cho các lần truy cập sau.
  3. Inline JavaScript (JS nội tuyến - Không khuyến khích cho mã phức tạp): Mã JavaScript được đặt trực tiếp trong các thuộc tính xử lý sự kiện của HTML (ví dụ: <button onclick="alert('Clicked!')">Click Me</button>).
     - Cách này trộn lẫn hành vi với cấu trúc, khó bảo trì và đọc hiểu. Thường chỉ nên dùng cho các hành động rất đơn giản.
  - Thuộc tính async và defer cho thẻ <script> (khi liên kết tệp bên ngoài):
    - async: Tải tệp script một cách bất đồng bộ (không chặn việc phân tích HTML) và thực thi ngay khi tải xong. Thứ tự thực thi của nhiều script async không được đảm bảo.
    - defer: Tải tệp script một cách bất đồng bộ nhưng chỉ thực thi sau khi toàn bộ tài liệu HTML đã được phân tích cú pháp hoàn toàn, và trước sự kiện DOMContentLoaded. Các script có defer sẽ thực thi theo thứ tự xuất hiện trong tài liệu.
    - Việc sử dụng async và defer rất quan trọng để tối ưu hiệu suất tải trang, tránh

việc script chặn hiển thị nội dung. defer thường được ưu tiên khi thứ tự script quan trọng hoặc khi script cần truy cập toàn bộ DOM.

3.2. Cú pháp cơ bản: Biến, Kiểu dữ liệu, Toán tử

- Biến (Variables): Là các tên tượng trưng dùng để lưu trữ các giá trị dữ liệu.
  - Khai báo:
    - var: Có phạm vi hàm (function-scoped) hoặc toàn cục (globally-scoped). Biến khai báo bằng var được "hoisted" (khai báo được đưa lên đầu phạm vi của nó trong quá trình biên dịch, nhưng không phải giá trị khởi tạo). Có thể được khai báo lại và cập nhật giá trị.
    - let: Có phạm vi khối (block-scoped - giới hạn trong cặp dấu {}). Được giới thiệu trong ES6. Được hoisted nhưng không được khởi tạo (nằm trong "temporal dead zone" cho đến khi dòng khai báo được thực thi). Không thể khai báo lại trong cùng một phạm vi, nhưng có thể cập nhật giá trị.
    - const: Có phạm vi khối. Được giới thiệu trong ES6. Được hoisted nhưng không được khởi tạo. Phải được khởi tạo giá trị ngay khi khai báo. Không thể khai báo lại hoặc gán lại giá trị mới (binding bất biến, nhưng nội dung của object/array có thể thay đổi). Việc giới thiệu let và const trong ES6 đã giải quyết nhiều vấn đề liên quan đến var (như phạm vi và hoisting), giúp mã nguồn dễ dự đoán và bảo trì hơn. Do đó, thực hành hiện đại khuyến khích sử dụng let và const thay cho var.
  - Tên biến (Identifiers): Phải bắt đầu bằng một chữ cái, dấu gạch dưới (_), hoặc dấu đô la ($). Các ký tự tiếp theo có thể là chữ cái, số, hoặc dấu gạch dưới. JavaScript phân biệt chữ hoa chữ thường.
- Kiểu dữ liệu (Data Types): JavaScript là một ngôn ngữ có kiểu động (dynamically typed), nghĩa là kiểu của biến được xác định tại thời điểm chạy dựa trên giá trị được gán.
  - Kiểu nguyên thủy (Primitive Types):
    - String: Chuỗi ký tự (ví dụ: "Hello", 'World'). Có các thuộc tính và phương thức như length, concat(), toUpperCase(), toLowerCase(), split().
    - Number: Giá trị số (bao gồm số nguyên và số thực, ví dụ: 42, 3.14). Có các giá trị đặc biệt như NaN (Not-a-Number), Infinity. Các phương thức như isNaN(), toFixed(). Đối tượng Math cung cấp nhiều hàm toán học.
    - Boolean: Giá trị logic true hoặc false.
    - undefined: Đại diện cho một biến đã được khai báo nhưng chưa được gán giá

trị, hoặc một hàm không trả về giá trị.
        - null: Đại diện cho sự cố ý không có giá trị đối tượng nào.
        - Symbol (ES6): Giá trị nguyên thủy duy nhất và bất biến.
        - BigInt (ES2020): Dành cho các số nguyên có độ dài tùy ý.
    - Kiểu đối tượng (Object Type):
        - Object: Tập hợp các cặp khóa-giá trị (thuộc tính).
        - Array: Danh sách các giá trị có thứ tự, là một loại đối tượng đặc biệt. Khai báo: let arr =; hoặc let arr = new Array();. Truy cập phần tử: arr[index]. Các thuộc tính/phương thức: length, push(), pop(), join().
        - Function: Một đối tượng có thể gọi được.
- Toán tử (Operators):
    - Toán tử số học: + (cộng/nối chuỗi), - (trừ), * (nhân), / (chia), % (chia lấy dư), ** (lũy thừa), ++ (tăng), -- (giảm).
    - Toán tử gán: =, +=, -=, *=, /=, %=, **=.
    - Toán tử so sánh:
        - == (so sánh bằng về giá trị, có ép kiểu - loose equality).
        - != (so sánh khác về giá trị, có ép kiểu).
        - === (so sánh bằng nghiêm ngặt cả về giá trị và kiểu, không ép kiểu - strict equality).
        - !== (so sánh khác nghiêm ngặt cả về giá trị và kiểu).
        - >, <, >=, <=. Do tính năng ép kiểu tự động của JavaScript khi dùng ==, có thể dẫn đến các lỗi khó tìm. Vì vậy, việc sử dụng toán tử so sánh nghiêm ngặt === và !== thường được coi là an toàn hơn và là một thực hành tốt.
    - Toán tử logic: && (AND logic), || (OR logic), ! (NOT logic).
    - Toán tử điều kiện (Ba ngôi): condition? exprIfTrue : exprIfFalse.
    - Toán tử typeof: Trả về một chuỗi cho biết kiểu của toán hạng.

3.3. Cấu trúc điều khiển: Câu lệnh điều kiện, Vòng lặp

Cấu trúc điều khiển cho phép thay đổi luồng thực thi của chương trình dựa trên các điều kiện hoặc lặp lại một khối mã nhiều lần.

- Câu lệnh điều kiện (Conditional Statements):
    - if statement: Thực thi một khối mã nếu một điều kiện cụ thể là đúng (truthy). Cú pháp: if (condition) { /* code to execute */ }.
    - if...else statement: Thực thi một khối mã nếu điều kiện là đúng, và một khối mã khác nếu điều kiện là sai (falsy). Cú pháp: if (condition) { /* code if true */ } else

{ /* code if false */ }.

- if...else if...else statement: Cho phép kiểm tra nhiều điều kiện theo tuần tự. Cú pháp: if (condition1) { /* code */ } else if (condition2) { /* code */ } else { /* code */ }. Trong JavaScript, các giá trị không nhất thiết phải là true hoặc false để được đánh giá trong một điều kiện. Các giá trị như false, 0, "" (chuỗi rỗng), null, undefined, NaN được coi là "falsy". Tất cả các giá trị khác (bao gồm cả đối tượng và mảng, ngay cả khi rỗng) được coi là "truthy". Hiểu rõ điều này là rất quan trọng để viết logic điều kiện chính xác.
- switch statement: Đánh giá một biểu thức và so khớp giá trị của nó với một loạt các mệnh đề case. Thực thi các câu lệnh liên quan đến case đầu tiên khớp. Cú pháp:

  JavaScript

```
switch (expression) {
  case value1:
    // statements
    break; // Ngăn chặn "fall-through"
  case value2:
    // statements
    break;
  default: // Tùy chọn
    // statements nếu không có case nào khớp
}
```

Từ khóa break rất quan trọng để thoát khỏi khối switch sau khi một case được thực thi.

- Vòng lặp (Loops): Dùng để thực thi một khối mã lặp đi lặp lại.
  - for loop: Lặp lại một khối mã một số lần đã biết trước. Cú pháp: for (initialization; condition; increment/decrement) { /* code to execute */ } Ví dụ: for (let i = 0; i < 5; i++) { console.log(i); }.
  - while loop: Lặp lại một khối mã miễn là một điều kiện cụ thể còn đúng. Điều kiện được kiểm tra trước khi thực thi thân vòng lặp. Cú pháp: while (condition) { /* code to execute */ }.
  - do...while loop: Tương tự while, nhưng thân vòng lặp được thực thi ít nhất một lần trước khi điều kiện được kiểm tra. Cú pháp: do { /* code to execute */ } while (condition);.

- for...of loop (ES6): Lặp qua các đối tượng có thể lặp (iterable objects) như Mảng (Arrays), Chuỗi (Strings), Maps, Sets. Ví dụ: for (const item of myArray) { console.log(item); }.
- break statement: Thoát khỏi vòng lặp hiện tại hoặc câu lệnh switch.
- continue statement: Bỏ qua phần còn lại của lần lặp hiện tại và chuyển sang lần lặp tiếp theo.

Việc lựa chọn loại vòng lặp phù hợp (for, while, do...while, for...of) phụ thuộc vào yêu cầu cụ thể của bài toán lặp, ví dụ như số lần lặp đã biết, lặp dựa trên điều kiện, hay lặp qua các phần tử của một tập hợp.

3.4. Hàm (Functions) trong JavaScript

Hàm là một khối mã được thiết kế để thực hiện một tác vụ cụ thể. Hàm có thể được gọi nhiều lần, giúp tái sử dụng mã và làm cho chương trình có cấu trúc hơn.

- Khai báo hàm (Function Declaration): Cú pháp:
  JavaScript
  ```
  function tenHam(thamSo1, thamSo2,...) {
    // Các câu lệnh thực thi
    return giaTriTraVe; // Tùy chọn
  }
  ```

* function: Từ khóa để khai báo hàm.

* tenHam: Tên của hàm.

* thamSo1, thamSo2,... (Parameters): Là các biến được liệt kê trong phần định nghĩa hàm, hoạt động như các placeholder cho các giá trị sẽ được truyền vào khi hàm được gọi.

* Thân hàm (Function body): Các câu lệnh JavaScript nằm trong cặp dấu {}.

* return giaTriTraVe: Từ khóa return dùng để trả về một giá trị từ hàm. Nếu hàm không có câu lệnh return hoặc return không có giá trị, hàm sẽ trả về undefined.

- Biểu thức hàm (Function Expression): Hàm cũng có thể được định nghĩa như một biểu thức và gán cho một biến.
  JavaScript
  ```
  const tenHam = function(thamSo1, thamSo2) {
    //...
    return giaTriTraVe;
  ```

};

Hàm trong biểu thức hàm có thể là hàm ẩn danh (anonymous function - không có tên).

- Gọi hàm (Calling a Function): Để thực thi một hàm, ta gọi nó bằng tên, theo sau là cặp dấu ngoặc đơn (), bên trong có thể chứa các đối số. Ví dụ: let ketQua = tenHam(doiSo1, doiSo2);
- Tham số (Parameters) và Đối số (Arguments):
  - Tham số (Parameters): Là các tên biến được liệt kê trong định nghĩa hàm.
  - Đối số (Arguments): Là các giá trị thực tế được truyền vào hàm khi nó được gọi.
  - JavaScript cho phép truyền số lượng đối số khác với số lượng tham số đã khai báo. Nếu truyền ít đối số hơn, các tham số còn lại sẽ có giá trị undefined. Nếu truyền nhiều hơn, các đối số thừa có thể được truy cập thông qua đối tượng arguments (trong các hàm thông thường).
  - Tham số mặc định (Default Parameters - ES6): Cho phép gán giá trị mặc định cho tham số nếu không có đối số nào được truyền hoặc đối số là undefined. Cú pháp: function ham(param1 = defaultValue1) {... }.
- Phạm vi biến (Variable Scope):
  - Biến toàn cục (Global Scope): Biến được khai báo bên ngoài bất kỳ hàm nào, có thể được truy cập từ bất kỳ đâu trong chương trình.
  - Biến cục bộ (Local/Function Scope): Biến được khai báo bên trong một hàm (sử dụng var, let, hoặc const), chỉ có thể được truy cập từ bên trong hàm đó.
  - Phạm vi khối (Block Scope - với let và const): Biến được khai báo với let hoặc const bên trong một khối {} (ví dụ: trong if, for) chỉ có thể truy cập được bên trong khối đó.
- Hoisting: Trong JavaScript, các khai báo hàm (function declarations) được "hoisted" lên đầu phạm vi của chúng, có nghĩa là bạn có thể gọi một hàm trước khi nó được khai báo trong mã. Tuy nhiên, điều này không áp dụng cho biểu thức hàm (function expressions).

3.5. Tương tác với DOM (Document Object Model)

DOM (Document Object Model) là một giao diện lập trình cho các tài liệu HTML và XML. Nó biểu diễn cấu trúc của tài liệu dưới dạng một cây các đối tượng (nodes), nơi mỗi node đại diện cho một phần của tài liệu (ví dụ: phần tử, thuộc tính, văn bản). JavaScript có thể sử dụng DOM để truy cập và thay đổi nội dung, cấu trúc và kiểu dáng

của tài liệu HTML một cách động.

- Chọn phần tử HTML (Selecting Elements): Để thao tác với một phần tử HTML, trước tiên bạn cần chọn nó.
  - document.getElementById('id_cua_phan_tu'): Chọn một phần tử duy nhất dựa trên thuộc tính id của nó.
  - document.getElementsByTagName('ten_the'): Trả về một HTMLCollection (giống mảng) gồm tất cả các phần tử có tên thẻ được chỉ định.
  - document.getElementsByClassName('ten_class'): Trả về một HTMLCollection gồm tất cả các phần tử có tên lớp được chỉ định.
  - document.querySelector('css_selector'): Trả về phần tử đầu tiên trong tài liệu khớp với bộ chọn CSS được chỉ định. Đây là một phương thức rất mạnh mẽ và linh hoạt.
  - document.querySelectorAll('css_selector'): Trả về một NodeList (giống mảng) gồm tất cả các phần tử trong tài liệu khớp với bộ chọn CSS được chỉ định.
- Thay đổi nội dung HTML:
  - element.innerHTML: Lấy hoặc thiết lập nội dung HTML (bao gồm cả các thẻ HTML) bên trong một phần tử.
  - element.textContent: Lấy hoặc thiết lập nội dung văn bản thuần túy bên trong một phần tử và tất cả các con của nó, bỏ qua các thẻ HTML.
  - element.innerText: Tương tự textContent nhưng có một số khác biệt về cách xử lý khoảng trắng và các phần tử ẩn.
- Thay đổi thuộc tính HTML:
  - element.attributeName (ví dụ: element.src, element.href): Truy cập trực tiếp các thuộc tính phổ biến.
  - element.getAttribute('attributeName'): Lấy giá trị của một thuộc tính.
  - element.setAttribute('attributeName', 'newValue'): Thiết lập giá trị mới cho một thuộc tính.
  - element.removeAttribute('attributeName'): Xóa một thuộc tính.
- Thay đổi kiểu dáng CSS:
  - element.style.propertyName: Cho phép truy cập và thay đổi các thuộc tính CSS nội tuyến của một phần tử. Tên thuộc tính CSS có dấu gạch nối (ví dụ: background-color) sẽ được chuyển thành dạng camelCase trong JavaScript (ví dụ: backgroundColor). Ví dụ: document.getElementById('myDiv').style.color = 'blue';
  - element.classList: Cung cấp các phương thức để thao tác với danh sách các lớp

của phần tử:
    - element.classList.add('className'): Thêm một lớp.
    - element.classList.remove('className'): Xóa một lớp.
    - element.classList.toggle('className'): Thêm lớp nếu chưa có, xóa nếu đã có.
    - element.classList.contains('className'): Kiểm tra xem phần tử có chứa lớp đó không.
- Tạo và thêm/xóa phần tử:
    - document.createElement('tagName'): Tạo một phần tử HTML mới.
    - parentNode.appendChild(childNode): Thêm childNode làm con cuối cùng của parentNode.
    - parentNode.insertBefore(newNode, existingNode): Chèn newNode vào trước existingNode trong parentNode.
    - element.remove() hoặc parentNode.removeChild(childNode): Xóa một phần tử.

3.6. Xử lý sự kiện (Event Handling)

Xử lý sự kiện là quá trình phản hồi lại các hành động xảy ra trong trình duyệt, thường là do người dùng tương tác (ví dụ: nhấp chuột, nhấn phím) hoặc do các thay đổi trạng thái của trình duyệt hoặc tài liệu (ví dụ: trang tải xong, kích thước cửa sổ thay đổi).

- Sự kiện (Events): Là các tín hiệu được hệ thống (trình duyệt) phát ra khi có điều gì đó xảy ra. Ví dụ về các sự kiện phổ biến:
    - click: Người dùng nhấp chuột vào một phần tử.
    - mouseover: Con trỏ chuột di chuyển vào một phần tử.
    - mouseout: Con trỏ chuột di chuyển ra khỏi một phần tử.
    - keydown, keyup, keypress: Người dùng nhấn hoặc nhả một phím.
    - load: Tài liệu hoặc một tài nguyên (như hình ảnh) đã tải xong.
    - submit: Một biểu mẫu được gửi đi.
    - focus: Một phần tử nhận được focus.
    - blur: Một phần tử mất focus.
    - change: Giá trị của một phần tử biểu mẫu (như <input>, <select>) thay đổi.
- Trình xử lý sự kiện (Event Handlers/Listeners): Là các hàm JavaScript được thực thi khi một sự kiện cụ thể xảy ra trên một phần tử cụ thể.
- Cách gán trình xử lý sự kiện:
    1. Thuộc tính HTML on-event (Inline event handlers - không khuyến khích): Gán trực tiếp mã JavaScript hoặc tên hàm vào thuộc tính HTML của sự kiện. Ví dụ: <button onclick="alert('Button clicked!')">Click Me</button>. Cách này trộn lẫn

HTML và JavaScript, khó bảo trì.

2. Thuộc tính DOM on-event (Event handler properties): Gán một hàm cho thuộc tính sự kiện của đối tượng DOM. Ví dụ:

```
JavaScript
const myButton = document.getElementById('myBtn');
myButton.onclick = function() {
  alert('Button clicked!');
};
```

Với cách này, chỉ có thể gán một trình xử lý cho mỗi sự kiện trên một phần tử.
Phương thức addEventListener() (Khuyến khích sử dụng):
Đây là cách hiện đại và linh hoạt nhất để đăng ký trình xử lý sự kiện. Nó cho phép gán nhiều trình xử lý cho cùng một sự kiện trên một phần tử.
Cú pháp: element.addEventListener(event, listenerFunction, useCapture);* event: Tên sự kiện dưới dạng chuỗi (ví dụ: "click", "mouseover").* listenerFunction: Hàm sẽ được gọi khi sự kiện xảy ra. Hàm này tự động nhận một đối tượng sự kiện (event object) làm tham số đầu tiên, chứa thông tin về sự kiện.* useCapture (tùy chọn): Một giá trị boolean xác định giai đoạn bắt sự kiện (capturing phase - true) hay giai đoạn nổi bọt (bubbling phase - false, mặc định).

Ví dụ:
javascript const myButton = document.getElementById('myBtn');
function handleClick() { alert('Button clicked via addEventListener!'); } myButton.addEventListener('click', handleClick);
Để gỡ bỏ trình xử lý sự kiện đã thêm bằng addEventListener(), sử dụng phương thức removeEventListener(event, listenerFunction, useCapture).
Điều quan trọng là hàm listener được truyền vào removeEventListener phải là cùng một hàm đã được truyền vào addEventListener (không phải là một hàm ẩn danh mới).

- Đối tượng sự kiện (Event Object): Khi một sự kiện xảy ra và một trình xử lý sự kiện được gọi, trình duyệt sẽ tự động truyền một đối tượng sự kiện vào hàm xử lý. Đối tượng này chứa thông tin chi tiết về sự kiện, ví dụ:
  - event.target: Phần tử DOM mà sự kiện bắt nguồn.
  - event.type: Loại sự kiện (ví dụ: "click").
  - event.preventDefault(): Ngăn chặn hành vi mặc định của trình duyệt đối với sự kiện đó (ví dụ: ngăn form submit khi nhấp nút submit).
  - event.stopPropagation(): Ngăn sự kiện nổi bọt (bubbling) lên các phần tử cha.
  - Tọa độ chuột (ví dụ: event.clientX, event.clientY), thông tin phím (ví dụ:

event.key, event.keyCode).

Phần 4: Các bước cơ bản để thiết kế một trang web tĩnh

Thiết kế một trang web tĩnh, mặc dù đơn giản hơn web động, vẫn đòi hỏi một quy trình có cấu trúc để đảm bảo sản phẩm cuối cùng đáp ứng được mục tiêu và yêu cầu đề ra.

4.1. Lập kế hoạch và xác định yêu cầu

Đây là bước nền tảng và quan trọng nhất trong mọi dự án thiết kế web.

- Xác định mục đích của website: Trang web được tạo ra để làm gì? (ví dụ: giới thiệu công ty, portfolio cá nhân, blog, trang thông tin sản phẩm).
- Xác định đối tượng người dùng mục tiêu: Ai sẽ sử dụng trang web này? Nhu cầu và mong đợi của họ là gì? Điều này sẽ ảnh hưởng đến thiết kế giao diện, nội dung và cấu trúc.
- Xác định nội dung chính: Những thông tin nào cần được truyền tải? Các trang chính sẽ là gì (ví dụ: Trang chủ, Giới thiệu, Dịch vụ, Liên hệ)?.
- Phân tích đối thủ cạnh tranh (nếu có): Xem xét các trang web tương tự để học hỏi điểm mạnh, điểm yếu và tìm ra lợi thế cạnh tranh.
- Lập sơ đồ trang web (Sitemap): Phác thảo cấu trúc phân cấp của các trang và cách chúng liên kết với nhau.

4.2. Thiết kế Wireframe và Mockup

Sau khi có kế hoạch rõ ràng, bước tiếp theo là trực quan hóa cấu trúc và giao diện.

- Wireframe (Khung sườn): Là bản phác thảo cấu trúc cơ bản, ở mức độ trung thực thấp (low-fidelity), tập trung vào bố cục, vị trí của các thành phần chính (như header, footer, sidebar, các khối nội dung, nút bấm) và luồng người dùng. Wireframe không chú trọng vào màu sắc, font chữ hay hình ảnh chi tiết. Mục đích là để thống nhất về cấu trúc và chức năng cơ bản trước khi đi vào chi tiết thiết kế đồ họa.
- Mockup (Mô hình thiết kế): Là bản trình bày trực quan, ở mức độ trung thực cao (high-fidelity), thể hiện giao diện người dùng (UI) sẽ trông như thế nào. Mockup bao gồm màu sắc, font chữ, hình ảnh, icon và các yếu tố đồ họa khác, gần giống với sản phẩm cuối cùng nhưng thường là tĩnh (không có tương tác). Mockup giúp hình dung rõ hơn về "cảm giác" của trang web và nhận phản hồi về mặt thẩm mỹ.

Wireframe giúp xác định "cái gì ở đâu", trong khi mockup thể hiện "nó trông như thế

nào". Cả hai đều quan trọng để đảm bảo thiết kế đi đúng hướng và giảm thiểu việc sửa đổi tốn kém ở các giai đoạn sau.

4.3. Xây dựng cấu trúc HTML

Dựa trên wireframe và mockup đã được duyệt, tiến hành viết mã HTML để tạo ra cấu trúc sườn cho trang web.

- Sử dụng các thẻ HTML cơ bản và các thẻ ngữ nghĩa HTML5 (như <header>, <nav>, <main>, <article>, <section>, <footer>, <aside>) để tổ chức nội dung một cách logic và có ý nghĩa.
- Đảm bảo mã HTML hợp lệ (validate HTML) để tránh các lỗi hiển thị không mong muốn trên các trình duyệt khác nhau.
- Tạo các tệp HTML riêng biệt cho từng trang của website (ví dụ: index.html, about.html, contact.html).

4.4. Áp dụng CSS để tạo kiểu dáng

Sau khi có cấu trúc HTML, sử dụng CSS để định dạng giao diện, màu sắc, font chữ, bố cục và các yếu tố trực quan khác theo thiết kế mockup.

- Viết mã CSS trong một hoặc nhiều tệp .css bên ngoài và liên kết chúng vào các tệp HTML (phương pháp External CSS được khuyến khích).
- Sử dụng các bộ chọn CSS (selectors) một cách hiệu quả để nhắm mục tiêu các phần tử HTML cần tạo kiểu.
- Áp dụng các khái niệm về Box Model, Flexbox, Grid Layout để xây dựng bố cục trang.
- Triển khai Responsive Web Design bằng Media Queries để đảm bảo trang web hiển thị tốt trên các kích thước màn hình khác nhau.
- Tổ chức mã CSS một cách khoa học, dễ đọc và dễ bảo trì, có thể sử dụng các quy ước đặt tên (như BEM) hoặc chia nhỏ CSS thành các module.

4.5. Thêm tương tác cơ bản với JavaScript (nếu cần)

Đối với trang web tĩnh, JavaScript có thể được sử dụng để thêm các tương tác nhỏ phía client nhằm nâng cao trải nghiệm người dùng, mà không cần đến xử lý phía máy chủ.

- Ví dụ:
    - Tạo menu thả xuống (dropdown menu).

    - Tạo slideshow hình ảnh đơn giản.
    - Hiển thị/ẩn các phần tử khi người dùng nhấp chuột.
    - Kiểm tra dữ liệu form cơ bản trước khi (giả sử) gửi qua email hoặc một dịch vụ bên thứ ba đơn giản (đối với web tĩnh thực sự, việc gửi form phức tạp thường cần backend).
    - Thay đổi kiểu dáng CSS của phần tử dựa trên hành động của người dùng.
- Viết mã JavaScript trong các tệp .js bên ngoài và liên kết vào HTML.
- Sử dụng các kỹ thuật thao tác DOM và xử lý sự kiện.

4.6. Kiểm thử và gỡ lỗi (Testing and Debugging)

Các bước quan trọng để đảm bảo chất lượng của trang web trước khi triển khai.

- Kiểm tra hiển thị (Visual Testing): Đảm bảo trang web hiển thị đúng như thiết kế trên các trình duyệt phổ biến (Chrome, Firefox, Safari, Edge) và trên các thiết bị khác nhau (desktop, tablet, mobile).
- Kiểm tra chức năng (Functional Testing): Kiểm tra tất cả các liên kết, nút bấm, form (nếu có tương tác JS) và các tính năng khác hoạt động chính xác.
- Kiểm tra đáp ứng (Responsive Testing): Thay đổi kích thước cửa sổ trình duyệt hoặc sử dụng công cụ developer tools để kiểm tra layout có thích ứng tốt không.
- Kiểm tra lỗi HTML, CSS, JavaScript: Sử dụng các công cụ validator (W3C HTML Validator, W3C CSS Validator) và console của trình duyệt để phát hiện và sửa lỗi cú pháp hoặc lỗi logic.
- Kiểm tra khả năng truy cập (Accessibility Testing): Đảm bảo trang web có thể sử dụng được bởi người khuyết tật (ví dụ: kiểm tra độ tương phản màu, điều hướng bằng bàn phím, văn bản thay thế cho hình ảnh).
- Kiểm tra hiệu suất cơ bản (Performance Testing): Kiểm tra tốc độ tải trang, tối ưu hóa hình ảnh. Công cụ như Google PageSpeed Insights có thể hữu ích.

4.7. Triển khai (Deployment - khái niệm cơ bản)

Sau khi trang web đã được kiểm thử kỹ lưỡng, bước cuối cùng là đưa nó lên mạng Internet để người dùng có thể truy cập.

- Chọn nhà cung cấp dịch vụ lưu trữ web (Hosting): Đối với web tĩnh, có nhiều lựa chọn hosting miễn phí hoặc chi phí thấp như GitHub Pages, Netlify, Vercel, hoặc các dịch vụ hosting truyền thống.
- Đăng ký tên miền (Domain Name): Nếu chưa có, cần đăng ký một tên miền cho

trang web (ví dụ: yourwebsite.com).

- Tải tệp lên máy chủ: Sử dụng các công cụ FTP (File Transfer Protocol) hoặc các giao diện quản lý tệp do nhà cung cấp hosting cung cấp để tải các tệp HTML, CSS, JavaScript và hình ảnh lên máy chủ. Nhiều nền tảng hiện đại cho web tĩnh hỗ trợ triển khai trực tiếp từ các kho chứa mã nguồn như Git.

Quy trình này là một vòng lặp, sau khi triển khai, việc bảo trì, cập nhật nội dung và tối ưu hóa dựa trên phản hồi của người dùng hoặc phân tích dữ liệu là cần thiết để trang web luôn hoạt động hiệu quả.

NGÂN HÀNG CÂU HỎI ÔN TẬP

Một số câu hỏi ôn tập lý thuyết, bao gồm các kiến thức về HTML, CSS, JavaScript (frontend) và các bước cơ bản để thiết kế một trang web tĩnh.

TT	Nội dung câu hỏi
1	HTML là viết tắt của cụm từ nào? Vai trò chính của HTML trong phát triển web là gì?
2	Nêu ít nhất 2 điểm khác biệt giữa HTML và HTML5.
3	HTML có phải là một ngôn ngữ lập trình không? Giải thích ngắn gọn.
4	Trình bày mục đích chính của việc sử dụng HTML trong một trang web.
5	Vẽ sơ đồ cấu trúc cơ bản của một tài liệu HTML, bao gồm các thẻ chính yếu.
6	Thẻ <!DOCTYPE html> có ý nghĩa gì trong một tài liệu HTML5?
7	Nội dung nào thường được đặt trong thẻ <head> của một tài liệu HTML? Nêu ít nhất 3 ví dụ.
8	Thẻ <body> trong HTML dùng để làm gì?

9	Phân biệt sự khác nhau cơ bản giữa thẻ <head> và thẻ <body> trong HTML.
10	Thuộc tính lang trong thẻ <html> (ví dụ: <html lang="vi">) có ý nghĩa gì?
11	Liệt kê 3 thẻ HTML dùng để định dạng tiêu đề (headings) và cho biết thẻ nào có mức độ quan trọng cao nhất.
12	Thẻ <p> dùng để làm gì? Cho ví dụ minh họa.
13	Thẻ <a> trong HTML được sử dụng với mục đích gì? Thuộc tính quan trọng nhất của thẻ <a> là gì?
14	Thẻ <img> dùng để làm gì? Nêu hai thuộc tính bắt buộc của thẻ <img>.
15	Phân biệt giữa thẻ và thẻ . Thẻ có vai trò gì khi kết hợp với chúng?
16	Thẻ và thẻ khác nhau như thế nào về cách hiển thị mặc định và mục đích sử dụng?
17	Nêu công dụng của thẻ và thẻ <hr>.
18	Thẻ <table> được sử dụng để làm gì? Kể tên 3 thẻ con thường dùng bên trong thẻ <table> để tạo cấu trúc bảng.
19	Block-level elements và Inline elements trong HTML là gì? Nêu 2 đặc điểm khác biệt chính.
20	Kể tên 2 thẻ HTML là block-level element và 2 thẻ là inline element.
21	Thuộc tính HTML là gì? Cung cấp thông tin gì cho phần tử HTML?
22	Nêu cú pháp chung để khai báo một thuộc tính HTML cho một phần tử. Cho ví dụ.

23	Thuộc tính id trong HTML có đặc điểm gì quan trọng? Nó thường được sử dụng cho mục đích gì?
24	Thuộc tính class trong HTML dùng để làm gì? Một phần tử có thể có nhiều class không?
25	Thuộc tính style trong HTML cho phép làm gì? Cho ví dụ về việc sử dụng thuộc tính style để thay đổi màu chữ của một đoạn văn bản.
26	Thuộc tính title trong HTML có công dụng gì?
27	Thuộc tính alt của thẻ <img> có ý nghĩa gì và tại sao nó quan trọng?
28	Thẻ <form> trong HTML dùng để làm gì? Nêu hai thuộc tính quan trọng thường được sử dụng với thẻ <form> và giải thích ý nghĩa của chúng.
29	Thẻ <input> là gì? Kể tên 3 giá trị phổ biến cho thuộc tính type của thẻ <input> và mô tả công dụng của mỗi loại.
30	Làm thế nào để tạo một trường nhập văn bản (text input field) trong HTML form?
31	Thẻ <label> trong HTML form có vai trò gì? Thuộc tính for của thẻ <label> liên kết với thuộc tính nào của trường nhập liệu?
32	Nêu sự khác biệt giữa <input type="radio"> và <input type="checkbox">.
33	Thẻ <textarea> dùng để làm gì trong một biểu mẫu HTML?
34	Thẻ <button> và <input type="submit"> có điểm gì giống và khác nhau khi sử dụng trong form?
35	Thuộc tính required trong các trường input của form có ý nghĩa gì?

36	Thẻ ngữ nghĩa trong HTML5 là gì? Nêu lợi ích của việc sử dụng thẻ ngữ nghĩa.
37	Kể tên 5 thẻ ngữ nghĩa mới được giới thiệu trong HTML5 và mô tả ngắn gọn mục đích sử dụng của mỗi thẻ.
38	Thẻ <header> trong HTML5 thường được dùng để chứa nội dung gì? Nó có thể xuất hiện ở đâu trong một tài liệu HTML?
39	Thẻ <nav> trong HTML5 dùng để xác định phần nào của trang web?
40	Phân biệt mục đích sử dụng của thẻ <article> và <section> trong HTML5.
41	Thẻ <footer> thường chứa những thông tin gì?
42	Thẻ <aside> trong HTML5 dùng để chứa loại nội dung nào?
43	CSS là viết tắt của cụm từ nào? Vai trò chính của CSS là gì?
44	Trình bày cú pháp cơ bản của một quy tắc CSS (CSS Rule), bao gồm selector và khối khai báo (declaration block).
45	Nêu 3 cách nhúng CSS vào tài liệu HTML. Cách nào được khuyến khích sử dụng nhất và tại sao?
46	Để nhúng CSS từ một file bên ngoài (external CSS), ta sử dụng thẻ HTML nào và đặt ở đâu trong tài liệu HTML? Thuộc tính nào của thẻ đó dùng để chỉ đường dẫn đến file CSS?
47	Internal CSS được khai báo như thế nào trong tài liệu HTML?
48	Inline CSS là gì? Nêu một ưu điểm và một nhược điểm của việc sử dụng Inline CSS.

49	Bộ chọn (selector) trong CSS dùng để làm gì?
50	Viết bộ chọn CSS để chọn tất cả các phần tử <p> trong tài liệu.
51	Viết bộ chọn CSS để chọn một phần tử có id là "main-title".
52	Viết bộ chọn CSS để chọn tất cả các phần tử có class là "highlight".
53	Phân biệt sự khác nhau giữa ID selector và Class selector về tính duy nhất và cách sử dụng.
54	Giải thích sự khác biệt giữa padding và margin trong CSS Box Model.
55	Liệt kê 3 thuộc tính CSS dùng để định dạng font chữ (ví dụ: tên font, kích thước, độ đậm).
56	Thuộc tính color trong CSS dùng để định dạng màu cho thành phần nào của một phần tử?
57	Nêu 2 cách để xác định giá trị màu trong CSS (ví dụ: tên màu, mã HEX).
58	Thuộc tính background-image cho phép làm gì?
59	Thuộc tính border là shorthand cho những thuộc tính nào? Cho ví dụ về cách sử dụng shorthand border.
60	Thuộc tính border-radius dùng để làm gì?
61	Nêu ý nghĩa của các giá trị block, inline, và inline-block của thuộc tính display.
62	Thuộc tính position: relative; và position: absolute; khác nhau như thế nào về cách xác định vị trí và ảnh hưởng đến luồng tài liệu?
63	CSS Grid Layout là gì? Nó khác biệt cơ bản với Flexbox như thế nào?

64	Khi nào nên sử dụng Flexbox và khi nào nên sử dụng CSS Grid?
65	Responsive Web Design (RWD) là gì? Mục tiêu chính của RWD là gì?
66	Media Query trong CSS là gì? Cú pháp cơ bản của một Media Query?
67	Thẻ <meta name="viewport"...> có vai trò gì trong RWD?
68	Giải thích ý nghĩa của min-width và max-width trong Media Query.
69	JavaScript là gì? Vai trò chính của JavaScript trong phát triển web front-end là gì?
70	Nêu 2 cách để thêm mã JavaScript vào một trang HTML. Cách nào thường được ưu tiên cho các đoạn mã lớn?
71	Tại sao việc đặt thẻ <script> ở cuối phần <body> thường được khuyến khích?
72	Trình bày sự khác biệt giữa let, const và var khi khai báo biến trong JavaScript.
73	Kể tên 3 kiểu dữ liệu nguyên thủy (primitive types) trong JavaScript.
74	Toán tử === (so sánh nghiêm ngặt) khác với toán tử == (so sánh lỏng) như thế nào trong JavaScript? Nên ưu tiên dùng toán tử nào?
75	Cho ví dụ về toán tử logic AND và OR
76	Viết cú pháp của câu lệnh if...else if...else trong JavaScript.
77	Vòng lặp for và vòng lặp while trong JavaScript khác nhau cơ bản ở điểm nào?
78	Hàm (function) trong JavaScript là gì? Nêu cú pháp khai báo một hàm cơ bản.
79	Tham số (parameter) và đối số (argument) của hàm khác nhau như thế nào?

80	Từ khóa return trong hàm dùng để làm gì?
81	DOM là gì? JavaScript sử dụng DOM để làm gì?
82	Nêu một phương thức JavaScript dùng để chọn một phần tử HTML dựa trên id của nó.
83	Làm thế nào để thay đổi nội dung văn bản của một phần tử HTML bằng JavaScript? (Ví dụ: thay đổi text của một thẻ <p>).
84	Làm thế nào để thay đổi một thuộc tính CSS (ví dụ: màu nền) của một phần tử HTML bằng JavaScript?
85	Xử lý sự kiện (event handling) trong JavaScript là gì? Cho ví dụ về một sự kiện phổ biến (ví dụ: click chuột).
86	Phương thức addEventListener() dùng để làm gì? Nêu các tham số chính của phương thức này.
87	Wireframe và Mockup khác nhau như thế nào trong quá trình thiết kế web?
88	Tại sao việc kiểm thử (testing) trang web trên nhiều trình duyệt và thiết bị khác nhau lại quan trọng?