

TS. Nguyễn Quang Đạt (Chủ biên),

Dương Hoàng Vũ, Nguyễn Cao Huy, Nguyễn Công Duy, Nguyễn Phúc Nguyên, Trần Đức Minh, Trương Diệu Anh

Sổ tay

KIẾN THỨC TIN HỌC

ÔN THI  
TỐT NGHIỆP THPT

LẬP TRÌNH WEB CƠ BẢN

CSS



NHÀ XUẤT BẢN BÁCH KHOA HÀ NỘI

Dương Hoàng Vũ, Nguyễn Cao Huy, Nguyễn Công Duy, Nguyễn Phúc Nguyên, Trần Đức Minh, Tr  
TS. Nguyễn Quang Đạt (Chủ biên),

*Sổ tay*  
**KIẾN THỨC TIN HỌC**

**ÔN THI  
TỐT NGHIỆP THPT**

**LẬP TRÌNH WEB CƠ BẢN**



NHÀ XUẤT BẢN BÁCH KHOA HÀ NỘI

## **Biên mục trên xuất bản phẩm của Thư viện Quốc gia Việt Nam**

**Sổ tay kiến thức Tin học ôn thi tốt nghiệp THPT - Lập trình Web cơ bản / Nguyễn Quang Đạt (ch.b.), Dương Hoàng Vũ, Nguyễn Cao Huy... - H. : Bách khoa Hà Nội, 2025. - 208 tr. : minh họa ; 24 cm**

1. Tin học 2. Lập trình 3. Trang Web 4. Trung học phổ thông 5. Sổ tay  
006.760712 - dc23

BKF0325p-CIP

## LỜI GIỚI THIỆU

Trong kỷ nguyên số, kiến thức về lập trình Web ngày càng trở thành nền tảng quan trọng đối với học sinh phổ thông. Việc nắm vững HTML, CSS và JavaScript không chỉ giúp các bạn hiểu cấu trúc, cách trình bày và hành vi của một trang Web hiện đại mà còn hỗ trợ trực tiếp cho việc học và ôn thi môn Tin học. Cuốn sách **Sổ tay kiến thức Tin học ôn thi tốt nghiệp THPT – Lập trình Web cơ bản** được biên soạn với mục tiêu cung cấp một bản đồ khái niệm rõ ràng, hệ thống và dễ tiếp cận, phù hợp để đọc nhanh khi cần cùng cố lý thuyết trọng tâm trước kỳ thi.

Năm học 2025 đánh dấu lần đầu tiên kỳ thi tốt nghiệp THPT được tổ chức theo Chương trình Giáo dục phổ thông 2018, đồng thời cũng là năm đầu tiên môn Tin học xuất hiện như một bài thi độc lập. Trong bối cảnh hệ thống tài liệu tham khảo còn chưa phong phú, cuốn sách này được xây dựng như một nỗ lực tiên phong nhằm tổng hợp mạch lạc mảng kiến thức lập trình Web – nội dung giữ vai trò trọng tâm và chiếm tỷ trọng lớn trong định hướng đánh giá của môn học ở cấp phổ thông.

### NHÓM TÁC GIẢ

Cuốn sách là kết quả làm việc chung của một nhóm bảy thành viên thuộc Trường THPT chuyên Hà Nội – Amsterdam, gồm một giáo viên hướng dẫn và sáu học sinh khối 11 chuyên Tin. Người thầy đảm nhiệm vai trò định hướng nội dung, chuẩn hóa thuật ngữ và kiểm tra độ chính xác kỹ thuật; sáu học sinh giữ vai trò nghiên cứu, biên soạn và đề xuất cách diễn đạt gần gũi với lứa tuổi trung học. Sự kết hợp giữa kinh nghiệm sư phạm và góc nhìn của người học giúp cuốn sách đảm bảo hai tiêu chí: chính xác về mặt chuyên môn và thân thiện trong cách trình bày.

## MỤC TIÊU VÀ PHẠM VI

Cuốn sách tập trung xây dựng nền tảng lý thuyết cốt lõi cho mảng lập trình Web cơ bản. Nội dung sách được trình bày theo trúc kiến trúc ba tầng của một trang Web: HTML đảm nhận cấu trúc và ngữ nghĩa, CSS chịu trách nhiệm trình bày và bố cục, JavaScript mang lại tính tương tác và khả năng phản hồi của trang. Các khái niệm được diễn giải bằng ngôn ngữ tiếng Việt trong sáng, dễ hiểu, đi kèm mã minh họa tối giản và chú giải ngay bên cạnh để người đọc có thể lần theo lập luận một cách liền mạch.

## CẤU TRÚC CUỐN SÁCH

Chương I mở đầu giới thiệu bức tranh tổng thể của lập trình Web, từ cách thức hoạt động của trình duyệt và máy chủ đến cấu trúc dự án, quy ước đặt tên tệp và vai trò của các công cụ cơ bản. Chương II trình bày HTML theo mạch từ cấu trúc tài liệu với các phần head và body đến hệ thống thẻ thường dùng, cách hiển thị văn bản, chèn hình ảnh, tạo liên kết, tổ chức bảng và nhúng nội dung từ nguồn khác. Chương III đi vào CSS với Box Model, các thuộc tính về chữ và màu, nền và đường viền, cùng những kỹ thuật căn chỉnh, sắp xếp thành phần giúp trang trở nên thống nhất và dễ đọc. Chương IV giới thiệu JavaScript ở mức nhập môn, bao gồm cú pháp, biến và kiểu dữ liệu, hàm và phạm vi, đối tượng và mảng, cùng các thao tác cơ bản với DOM để liên kết mã lệnh với phần tử trên trang. Toàn bộ chương mục được sắp xếp theo lộ trình tuyến tính để người đọc có thể đọc từ đầu tới cuối hoặc tra cứu từng khái niệm khi cần.

## ĐỐI TƯỢNG BẠN ĐỌC VÀ CÁCH SỬ DỤNG

Sách hướng đến học sinh lớp 12 đang ôn thi tốt nghiệp THPT môn Tin học, giáo viên cần tài liệu tham khảo để hệ thống hóa bài giảng, cũng như bạn đọc yêu thích lập trình muốn nắm bắt nhanh cốt lõi của HTML, CSS và JavaScript. Khi sử dụng sách, người đọc nên bắt đầu từ phần giới thiệu để có cái nhìn tổng quát, sau đó chuyển sang HTML và CSS trước khi tiếp cận JavaScript. Ở mỗi đoạn mã minh họa, việc đọc chậm và đối chiếu với giải thích kèm theo sẽ giúp làm rõ mối liên hệ giữa cấu trúc trang, lớp trình bày và hành vi tương tác. Việc ghi chú lại các quy tắc, từ khóa và lỗi điển hình theo cách riêng của mình cũng là một thói quen hữu ích để củng cố kiến thức.

## **LỜI CẢM ƠN VÀ KỲ VỌNG**

Nhóm tác giả xin trân trọng cảm ơn Ban Giám hiệu, Tổ Tin học Trường THPT chuyên Hà Nội – Amsterdam cùng các thầy cô và bạn bè đã động viên, góp ý trong suốt quá trình biên soạn sách. Chúng tôi coi cuốn sách như một dự án học thuật mở và sẽ tiếp tục lắng nghe phản hồi từ bạn đọc để bổ sung, hoàn thiện trong những lần xuất bản tiếp theo. Hy vọng tài liệu này sẽ trở thành người đồng hành tin cậy, giúp các bạn học sinh củng cố kiến thức nền tảng, tự tin bước vào kỳ thi tốt nghiệp THPT và rộng hơn là nuôi dưỡng niềm say mê với thế giới lập trình Web.

**Nhóm tác giả**

**Hà Nội, 2025**



## CHƯƠNG I

# GIỚI THIỆU CHUNG

## 1.1. LẬP TRÌNH WEB

### 1.1.1. Khái niệm

Những trang Web hay trang mạng là một tập hợp các văn bản, hình ảnh, tệp tin tài liệu hoặc ứng dụng được lưu trữ trên các máy chủ (server) và có thể truy cập thông qua một tên miền (ví dụ: google.com, facebook.com, chat.zalo.me) bằng cách sử dụng một công cụ trình duyệt (Chrome, Safari, Microsoft Edge, v.v.).

### 1.1.2. Lịch sử hình thành

#### 1.1.2.1. Ý tưởng ban đầu (1989)

Năm 1989, khi đang làm việc tại CERN (Tổ chức Nghiên cứu Hạt nhân châu Âu), nhà khoa học máy tính người Anh Tim Berners-Lee đã đề xuất một hệ thống chia sẻ thông tin toàn cầu dựa trên siêu văn bản (Hypertext). Mục tiêu là giúp các nhà khoa học dễ dàng truy cập và liên kết thông tin từ các máy tính khác nhau.

#### 1.1.2.2. Phát triển công nghệ (1990)

Đến cuối năm 1990, Berners-Lee đã phát triển các công nghệ nền tảng cho World Wide Web, bao gồm:

- HTML (HyperText Markup Language): Ngôn ngữ đánh dấu siêu văn bản.
- HTTP (HyperText Transfer Protocol): Giao thức truyền tải siêu văn bản.
- URL (Uniform Resource Locator): Định danh tài nguyên thống nhất.

Ông cũng tạo ra trình duyệt đầu tiên có tên World Wide Web (sau này đổi tên thành Nexus) và máy chủ Web đầu tiên tại địa chỉ info.cern.ch, chạy trên máy tính NeXT tại CERN.

#### **1.1.2.3. Ra mắt công chúng (1991)**

Ngày 06 tháng 8 năm 1991, Berners-Lee công bố dự án World Wide Web trên nhóm tin Usenet, cho phép cộng đồng Internet truy cập trang Web đầu tiên tại info.cern.ch. Trang Web này cung cấp thông tin về dự án và hướng dẫn cách tạo trang Web.

#### **1.1.2.4. Phổ biến rộng rãi (1993)**

Ngày 30 tháng 4 năm 1993, CERN quyết định đưa phần mềm World Wide Web vào phạm vi công cộng, cho phép mọi người sử dụng và phát triển miễn phí. Quyết định này đã thúc đẩy sự phát triển nhanh chóng của Web trên toàn thế giới.

#### **1.1.2.5. Thành lập W3C (1994)**

Năm 1994, Berners-Lee thành lập World Wide Web Consortium (W3C) tại Học viện Công nghệ Massachusetts (MIT) nhằm phát triển các tiêu chuẩn mở cho Web và đảm bảo tính tương thích giữa các hệ thống.

Tóm lại, trang Web đầu tiên được hình thành từ ý tưởng chia sẻ thông tin toàn cầu, phát triển thành công nhờ sự hỗ trợ của CERN và cộng đồng, đã mở đường cho kỷ nguyên Internet hiện đại.

### **1.1.3. Ứng dụng**

Để nói về ứng dụng cũng như những trang Web thì vô vàn, từ những trang chứa thông tin như Wikipedia, những trang mạng xã hội như Facebook, đến những trò chơi trực tuyến, những công cụ trí tuệ nhân tạo miễn phí như Chat GPT, v.v.. Bạn gần như có thể tìm kiếm mọi thông tin và công cụ thông qua những trang Web.

### **1.1.4. Cách tạo ra một trang Web**

Dù các trang Web ngày càng trở nên phức tạp với những công cụ, ngôn ngữ lập trình không ngừng được cải tiến, chúng về cơ bản đều được tạo nên từ ba nền tảng chính: HTML, CSS và JavaScript.

- HTML (HyperText Markup Language): là khung xương, quyết định cấu trúc của từng trang web.
- CSS (Cascading Style Sheets): là thứ tạo nên diện mạo cho từng trang Web thông qua việc thay đổi những yếu tố như màu sắc, bố cục hay phông chữ.

– JavaScript: là ngôn ngữ lập trình giúp người dùng có thể tương tác được với các trang Web.

Bằng cách tạo một tệp HTML từ ba công cụ trên và khởi chạy tệp bằng một trình duyệt, về cơ bản bạn đã có cho mình một trang Web. Tuy nhiên, cần lưu ý rằng để có thể truy cập trang Web của mình trên Internet, bạn vẫn cần mua một tên miền và một nền tảng hosting.

Để có cái nhìn tổng quát hơn, chúng ta hãy cùng quan sát đoạn code của một trang Web đơn giản sau đây:

```
<!DOCTYPE html>
<head>
<title>Giới thiệu HTML, CSS, JS</title>
<style>
body {
    font-family: 'Arial', sans-serif;
    background-color: #f0f4f8;
    color: #333;
    text-align: center;
    margin: 0;
    padding: 0;
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
}
/* Cải thiện kiểu chữ tiêu đề */
h1 {
    color: #2a6d9f;
    font-size: 3rem;
```

```
margin-bottom: 20px;
}

/* Cải thiện kiểu chữ đoạn văn */

p {
color: #555;
font-size: 1.2rem;
margin-bottom: 30px;
}

/* Thiết kế nút bấm */

button {
background-color: #4CAF50;
color: white;
font-size: 1.2rem;
padding: 15px 30px;
border: none;
border-radius: 8px;
cursor: pointer;
transition: background-color 0.3s, transform 0.2s;
}

/* Hiệu ứng khi di chuột vào nút */

button:hover {
background-color: #45a049;
transform: scale(1.1);
}

/* Hiệu ứng khi nhấn nút */

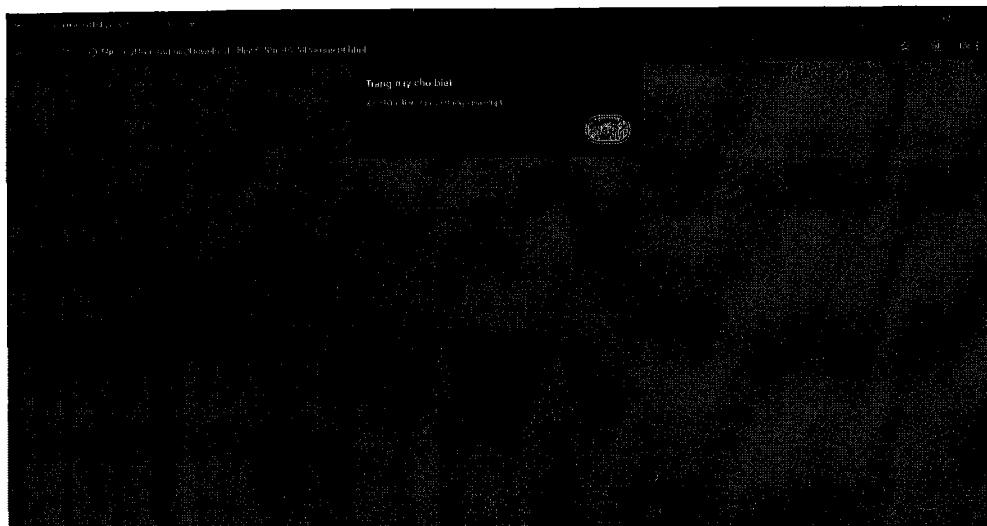
button:active {
transform: scale(1);
}
```

```
</style>
</head>
<body>
<div>
<h1>Chào mừng đến với Web!</h1>
<p>Đây là ví dụ đơn giản về HTML, CSS và JavaScript.</p>
<button onclick="sayHello()">Nhấn vào tôi!</button>
</div>
<script>
function sayHello(){
    alert("Xin chào! Bạn vừa sử dụng JavaScript!")
}
</script>
</body>
</html>
```

Trang Web sẽ hiển thị như sau:



Khi click vào nút điều hướng (Nhấn vào tôi) sẽ có thông báo hiện lên:



## 1.2. GIỚI THIỆU HTML

### 1.2.1. Giới thiệu về HTML

HTML (viết tắt của HyperText Markup Language) là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang Web. Được phát triển từ năm 1991 bởi Tim Berners-Lee và phát hành chính thức vào năm 1993, HTML đã trở thành nền tảng cốt lõi của mọi website trên World Wide Web, hay còn được gọi là Mạng lưới toàn cầu. Với cú pháp đơn giản là sử dụng các thẻ (hay còn gọi là "tags") như <html>, <head>, <body>, <p>, <a>, v.v., HTML cho phép người dùng dễ dàng tạo ra cấu trúc cơ bản cho trang Web bao gồm tiêu đề, đoạn văn liên kết, hình ảnh và nhiều thành phần khác.

Khác với các ngôn ngữ lập trình thông thường, HTML không có chức năng tính toán hay xử lý logic phức tạp mà chỉ tập trung vào việc định nghĩa và tổ chức nội dung. Để tạo một trang Web hoàn chỉnh, HTML thường được kết hợp với CSS (Cascading Style Sheets) để trình bày giao diện và JavaScript để thêm các chức năng tương tác.

Phiên bản mới nhất hiện nay là HTML5, được phát hành năm 2014, bổ sung nhiều chức năng quan trọng như hỗ trợ đa phương tiện (video, audio), các thẻ ngữ nghĩa (hay còn gọi là "semantic tags") giúp tối ưu SEO, cùng nhiều API mạnh mẽ cho phép xây dựng ứng dụng Web phức tạp.

Vì là ngôn ngữ cơ bản nhất trong lập trình Web nên việc sử dụng thuần thục HTML là bước đầu tiên và bắt buộc với bất kỳ ai muốn phát triển Web.

### 1.2.2. Lịch sử hình thành HTML

- Năm 1989: Tim Berners-Lee, một nhà khoa học tại CERN (Tổ chức Nghiên cứu Hạt nhân châu Âu), đã đề xuất và tạo mẫu một hệ thống để các nhà nghiên cứu ở đây có thể dễ dàng chia sẻ tài liệu, thứ mà sau này sẽ là nền móng giúp ông tạo ra HTML.
- Năm 1990: Berners-Lee và kỹ sư hệ thống dữ liệu CERN – Robert Cailliau đã hợp tác để cùng yêu cầu tài trợ dự án tạo HTML, nhưng dự án không được CERN chính thức thông qua.
- Năm 1991: Lần đầu tiên các mô tả về HTML xuất hiện trên Internet. Tập tài liệu này tên là HTML Tags và nó liệt kê 20 thẻ. Tuy nhiên, chỉ có 18 thẻ được sử dụng rộng rãi về sau, do một số thẻ như <nextid> không được chấp nhận vào tiêu chuẩn chính thức.
- Năm 1993: Phiên bản HTML đầu tiên được mô tả công khai và sử dụng rộng rãi, dù chưa chuẩn hóa chính thức. Đây là giai đoạn định hình nền tảng ban đầu của HTML. Phiên bản HTML này cũng không được đánh số, mà chỉ được gọi là HTML.
- Năm 1994: W3C (World Wide Web Consortium) được thành lập bởi Tim Berners-Lee để tiêu chuẩn hóa HTML và các công nghệ Web. Kể từ đây, HTML được phát triển dưới sự dẫn dắt của W3C.
- Năm 1995: HTML2 được phát hành và chuẩn hóa bởi IETF (Internet Engineering Task Force), nhằm thống nhất cú pháp và chức năng của HTML. Phiên bản này bổ sung thêm các tính năng như hỗ trợ các biểu mẫu thông qua các thẻ: <form>, <input>, <textarea>, v.v.. Đây cũng là phiên bản HTML đầu tiên được chính thức chuẩn hóa.
- Năm 1996: Cuối năm 1996, IETF ngừng phát triển HTML, quyền phát triển được chuyển giao hoàn toàn cho W3C. Dự thảo HTML3.2 sau đó đã được công bố.
- Năm 1997: Tháng 01 năm 1997, HTML3.2 được phát hành bởi W3C, thay cho IETF vốn đã dừng phát triển HTML từ cuối năm 1996. Phiên bản này bổ sung

nhiều cải tiến như hỗ trợ nhúng JavaScript thông qua thẻ <script>, cũng như cho phép sử dụng CSS để định dạng trang Web.

– Năm 1997: Tháng 12 năm 1997, HTML4.0 được phát hành bởi W3C, mang đến nhiều cải tiến như hỗ trợ khung (frames), bảng biểu nâng cao và phân tách nội dung khỏi trình bày (thúc đẩy sử dụng CSS). Tuy nhiên, phiên bản này còn gặp một số lỗi nhỏ.

– Năm 1998: W3C chuyển trọng tâm sang một phiên bản HTML dựa trên XML (Extensible Markup Language).

– Năm 1999: HTML4.01, một phiên bản hoàn thiện và ổn định hơn của HTML4.0, được phát hành và chuẩn hóa vào cuối năm 1999. Phiên bản này trở thành tiêu chuẩn thống trị trong hơn một thập kỷ tiếp theo.

– Năm 2000: W3C phát hành XHTML1.0, một phiên bản của HTML được viết lại theo cú pháp chặt chẽ của XML. Dù mục tiêu của sự đổi mới này là để đảm bảo tính hợp lệ của tài liệu và tăng khả năng tương thích với các công nghệ Web mới, nó cũng khiến cho trình duyệt gặp lỗi nếu cú pháp không được gõ đúng.

– Năm 2002 – 2006: W3C bắt đầu phát triển XHTML 2.0, nhưng phiên bản này không tương thích với các trình duyệt đang phổ biến lúc bấy giờ, gây ra nhiều khó khăn cho những lập trình viên mong muốn nâng cấp trang Web của mình. Dự án này sau đó bị hủy bỏ.

– Năm 2004: WHATWG (Web Hypertext Application Technology Working Group) được thành lập bởi Apple, Mozilla và Opera để tiếp tục phát triển HTML theo hướng linh hoạt và tương thích hơn – tiền thân của HTML5.

– Năm 2008: HTML5 Working Draft lần đầu tiên được WHATWG và W3C công bố, bắt đầu thu hút sự quan tâm lớn từ cộng đồng phát triển.

– Năm 2011: WHATWG đổi tên phiên bản HTML5 của họ thành HTML Living Standard. W3C tiếp tục với dự án phát hành HTML5 của họ.

– Năm 2012: WHATWG và W3C quyết định phát triển HTML theo hai hướng khác nhau. W3C hướng tới mục tiêu tạo ra một tiêu chuẩn thống nhất, trong khi WHATWG phát triển HTML5 của họ theo hướng một tiêu chuẩn sống – phiên bản này sẽ không bao giờ hoàn thiện và sẽ liên tục được cập nhật.

– Năm 2014: W3C phát hành chính thức HTML5 như một phiên bản hoàn chỉnh, trong khi WHATWG vẫn tiếp tục phát triển HTML như một tiêu chuẩn sống mà

không gắn mốc phiên bản cụ thể. HTML5 có nhiều bước tiến lớn so với HTML4.01, chẳng hạn như hỗ trợ đa phương tiện (audio, video), lưu trữ cục bộ, API phong phú và đặc biệt thân thiện với thiết bị di động.

– Năm 2019: W3C dừng phát triển HTML, WHATWG trở thành đơn vị duy nhất phát triển HTML. HTML Living Standard trở thành phiên bản HTML duy nhất, được duy trì và cập nhật liên tục. Sẽ không có HTML6 và 7 trong tương lai.

### 1.2.3. Các phiên bản

- HTML1: gồm các thẻ cơ bản:
  - + Cấu trúc và tài liệu cơ bản: <html>, <head>, <body>, <title>.
  - + Văn bản, tiêu đề: <p>, <h1> – <h6>.
  - + Liên kết và danh sách: <a>, <ul>, <li>.
  - + Ảnh: <img>.
- HTML2: được chuẩn hóa bởi IETF, bổ sung một vài tính năng mới:
  - + Form cơ bản: <form>, <input type = "text">, <select>, <option>, <textarea>.
  - + Table cơ bản: <table>, <tr>, <td>, <th>.
- HTML3.2: cho phép ứng dụng CSS và JavaScript, đồng thời bổ sung một vài tính năng mới:
  - + Định dạng trình bày: <font>, <b>/<i>/<u>, <center>.
  - + Nhúng script / style: <script>, <style>.
  - + Bổ sung form: <button>.
- HTML4.01:
  - + Cấu trúc bảng nâng cao: <thead>, <tbody>, <tfoot>, <colgroup>.
  - + Nhúng đối tượng: <iframe>, <object>, <param>.
  - + Ngữ nghĩa văn bản: <abbr>, <acronym>, <q>.
  - + Form nâng cao: <fieldset>, <legend>, <label>, <optgroup>.
- HTML5:
  - + Ngữ nghĩa layout: <header>, <footer>, <article>, <section>, <nav>, <aside>, <main>.

- + Đa phương tiện: <video>, <audio>, <canvas>, <svg>, <track>.
- + Form hiện đại: <input type = "email">, <input type = "date">, <datalist>, <progress>, <meter>, <output>.
- + Cho phép API và các tương tác như Geolocation, Local Storage, Drag and Drop, Web Workers, v.v..

#### **1.2.4. Nguồn gốc của ngôn ngữ HTML**

HTML (viết tắt của HyperText Markup Language – ngôn ngữ đánh dấu siêu văn bản) ra đời vào cuối thế kỷ XX, gắn liền với sự phát triển của Internet và nhu cầu chia sẻ thông tin giữa các nhà khoa học. Năm 1989, tại Tổ chức Nghiên cứu Hạt nhân châu Âu (CERN), nhà khoa học máy tính Tim Berners-Lee nhận thấy việc chia sẻ tài liệu giữa các nhóm nghiên cứu gặp nhiều trở ngại do thiếu một hệ thống chung. Ông đã đề xuất ý tưởng mang tên “Information Management: A Proposal”, với mục tiêu xây dựng một hệ thống có thể liên kết các tài liệu thông qua mạng, sử dụng các siêu liên kết (hyperlink).

Năm 1990, Berners-Lee chính thức bắt tay vào phát triển hệ thống đó và tạo ra ba thành phần chính: trình duyệt Web đầu tiên mang tên World Wide Web (về sau đổi thành Nexus), máy chủ Web đầu tiên đặt tại CERN và đặc biệt là HTML – một ngôn ngữ mới dùng để định dạng và trình bày nội dung trên các trang Web. HTML cho phép người dùng sử dụng các “thẻ” (tags) để tổ chức văn bản một cách có cấu trúc.

Tháng 11 năm 1991, ông công bố tài liệu đầu tiên mô tả chi tiết HTML với tên gọi “HTML Tags”, có chứa 20 thẻ, bao gồm 18 thẻ cơ bản cũng như một số thẻ đã bị loại bỏ như <nextid>. Dù mới ở giai đoạn sơ khai, HTML đã đặt nền móng cho sự ra đời của World Wide Web – mạng lưới toàn cầu cho phép các trang Web liên kết với nhau mà ngày nay chúng ta sử dụng hằng ngày. HTML không chỉ giải quyết nhu cầu chia sẻ thông tin trong giới nghiên cứu mà còn mở ra một cuộc cách mạng truyền thông và công nghệ, dẫn đến sự phát triển nhanh chóng của Internet trên toàn thế giới.

#### **1.2.5. Tình trạng hiện tại của HTML**

Hiện nay, HTML vẫn là nền tảng cốt lõi để tạo nên một trang Web. Phiên bản mới nhất HTML5 hỗ trợ thêm nhiều tính năng hữu dụng như hiển thị video, âm thanh, vẽ đồ họa, v.v. bên cạnh các tính năng hiển thị văn bản thông thường.

Lấy ví dụ, đoạn code sau cho phép người dùng nhúng một đoạn video vào trang Web mà không cần sử dụng các phần mềm phụ trợ như Flash:

```
<video controls>  
  <source src="video.mp4" type="video/mp4">  
  Trình duyệt của bạn hỗ trợ thẻ video.  
</video>
```

Các thẻ khác như `<header>`, `<article>`, v.v. giúp tổ chức nội dung một cách gọn gàng và rõ ràng. Kết hợp thêm CSS để tạo kiểu và JavaScript để thêm tính tương tác, người dùng có thể tạo nên các trang Web hiện đại, đẹp mắt và dễ sử dụng trên hầu hết mọi thiết bị.

Vì thế, HTML chính là nền tảng quan trọng nhất, đồng thời là ngôn ngữ thống trị ở thời điểm hiện tại khi một người muốn tạo một trang Web.

### 1.2.6. Độ khó của HTML

HTML không phải một ngôn ngữ lập trình mà là ngôn ngữ đánh dấu, được dùng để xây dựng cấu trúc Web đơn giản. Ở mức cơ bản, HTML rất dễ học và thường được đánh giá là một trong những ngôn ngữ thân thiện với người học mới nhất. Lấy ví dụ, chỉ với một vài dòng code sau:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Web cua toi</title>  
  </head>  
  <body>  
    <h1>Day la trang web cua toi:D<h1>  
    <p>Day la doan van dau tien.</p>  
  </body>  
</html>
```

người dùng đã có thể tạo một trang Web cơ bản để hiển thị tiêu đề và văn bản.

Tuy nhiên, một trang Web hoàn chỉnh còn cần tích hợp thêm CSS và JavaScript, hoặc cần tối ưu cho nhiều thiết bị hoặc trình duyệt khác nhau. Người học do đó cần hiểu rõ về cấu trúc thẻ, cú pháp, cũng như khả năng tương thích với các công nghệ khác. Vì vậy, mặc dù HTML dễ tiếp cận ở giai đoạn đầu nhưng lại đòi hỏi sự tỉ mỉ cao trong các dự án thực tế.

### 1.3. GIỚI THIỆU CSS

#### 1.3.1. Giới thiệu về CSS

CSS hay Cascading Style Sheets là một ngôn ngữ lập trình có thiết kế vô cùng đơn giản, thường được sử dụng để định vị và định dạng lại các phần tử được tạo bởi các ngôn ngữ đánh dấu. Mục tiêu chính của ngôn ngữ này 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, v.v..

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 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. Do đó, CSS là một công cụ thường được kết hợp với các ngôn ngữ như HTML và XHTML.

Cú pháp CSS được các chuyên gia đánh giá là rất đơn giản nên vô cùng dễ học và dễ hiểu. Hướng dẫn cũng như các tài liệu liên quan đến CSS hiện có sẵn và

hoàn toàn miễn phí nên các bạn có đam mê với ngành IT hay quan tâm tới CSS đều có thể tìm kiếm dễ dàng.

### 1.3.2. Lịch sử hình thành

Phiên bản	Tính năng
CSS1 (1996)	Cơ bản: thay đổi màu sắc, font chữ, kích thước văn bản và căn lề.
CSS2 (1998)	Tùy chỉnh bố cục phức tạp, điều chỉnh hiển thị cho các loại nội dung đa phương tiện, định dạng cho màn hình và máy in.
CSS3 (2011)	Flexbox, grid, animation, transition và gradient giúp nhà thiết kế tạo ra các hiệu ứng sinh động mà không cần sử dụng JavaScript.

### 1.3.3. Cách hoạt động

CSS giúp trang Web trở nên trực quan và chuyên nghiệp, dễ sử dụng với người dùng.

Trang Web không có CSS:



- [Đăng nhập](#)
- [Email](#)
- [Forums](#)
- [Sitemap](#)
- [Tuyển dụng](#)
- [Đối tác](#)
- [Contact](#)
- [Nhận bản tin](#)
- 
- 

#### TRƯỜNG THPT CHUYÊN HÀ NỘI - AMSTERDAM

*Nơi khởi đầu của những ước mơ*

- [Home](#)
- [Giới thiệu](#)
  - [Principal's Message](#)
  - [Sứ mệnh & Tâm nhín](#)
  - [Development History](#)
  - [Cơ cấu tổ chức](#)
    - [Đảng ủy](#)
    - [Ban giám hiệu](#)
    - [Công đoàn](#)
    - [Tổ chuyên môn](#)
    - [Chi đoàn giáo viên](#)
    - [Đoàn - Đội](#)

## Trang Web có CSS:

Trường THPT Chuyên Hà Nội - Amsterdam đào tạo theo hai hệ:

**Hệ phổ thông cơ sở:** Nhà trường tuyển chọn học sinh có năng lực vào học lớp 6. Qua 4 năm các em được các thầy cô nâng cao năng lực học tập và đạt được để có khả năng thi vào lớp 10 trường THPT Hà Nội - Amsterdam.

**Hệ phổ thông trung học:** Học sinh vào lớp 10 phải trải qua quá trình tuyển nghiêm ngặt do Sở Giáo dục và Đào tạo tổ chức. Lớp 10 có các bộ chuyên: Văn, Toán, Tin, Lý, Hóa, Sinh, Nga, Anh, Pháp. Các học sinh này phải học toàn diện và đạt yêu cầu khai giảng ở tất cả các môn học trên cơ sở đầu tư học môn chuyên để thi học sinh giỏi Quốc gia, Quốc tế. Sau ba năm học, da số các em đều vào các trường đại học mà mình trúng tuyển. Một số học sinh được đi học nước ngoài và được cấp học bổng.

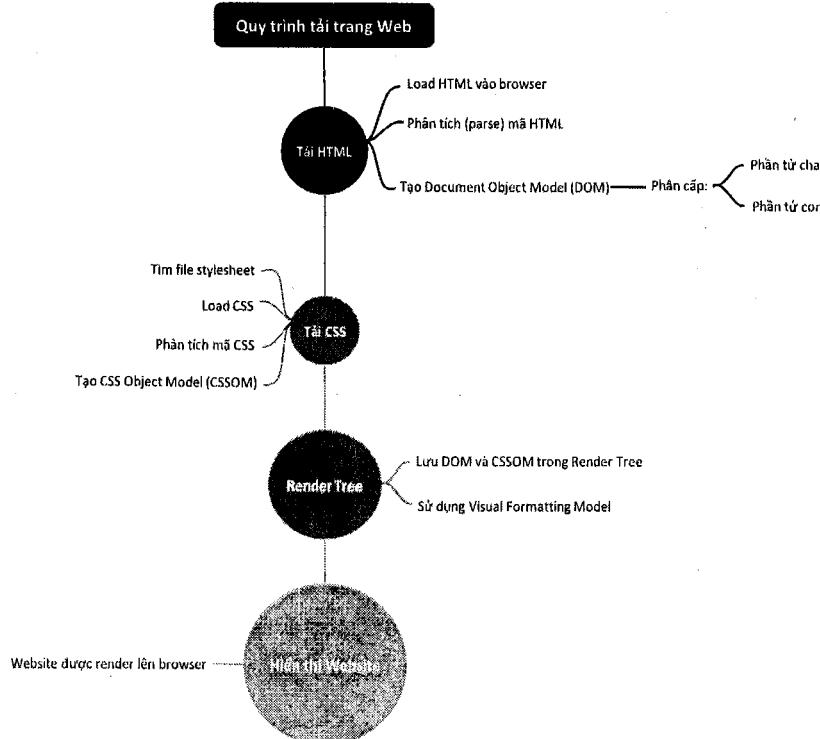
**Phương pháp đào tạo:** Trường THPT Hà Nội - Amsterdam thực hiện các phương pháp đào tạo như sau:

- Tuyển học sinh giỏi đầu vào.
- Giáo viên lấp trung trí luyện, nâng cao chất lượng hiệu quả giảng bài.
- Đổi mới phương pháp dạy, lấy học sinh làm trung tâm.

**TITLE**

Trường THPT Chuyên Hà Nội - Amsterdam chào mừng ngày hội thống nhất non sông

## Quy trình tải trang Web:



#### **1.3.4. Cách thiết lập CSS**

##### **1.3.4.1. CSS trong**

- Cách thiết lập này đưa toàn bộ các mẫu định dạng vào bên trong thẻ `<style>` và đặt trong phần tử `<head>` của tệp HTML.
- Các định dạng sẽ áp dụng cho tất cả các phần tử HTML trên trang Web phù hợp với mô tả của bộ chọn CSS.
- Với cách thiết lập CSS trong, các mẫu định dạng CSS chỉ được áp dụng cho tệp HTML hiện tại.

Ví dụ:

```
<!DOCTYPE html>
<html>
<head>
<style>
    h1 {
        color: green;
        text-align: center;
    }
    p {
        font-size: 16px;
    }
</style>
</head>
<body>
    <h1>Tiêu đề</h1>
    <p>Đây là đoạn văn.</p>
</body>
</html>
```

#### **1.3.4.2. CSS ngoài**

- Viết trong file.css riêng, ví dụ: style.css.
- Kết nối với HTML bằng thẻ `<link>` trong `<head>`.

Ví dụ:

File HTML:

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Đây là tiêu đề</h1>
    <p>Đây là đoạn văn.</p>
</body>
</html>
```

File style.css:

```
h1 {
    color: orange;
}
p {
    font-style: italic;
}
```

#### **1.3.4.3. CSS nội tuyến**

- Khai báo ngay trong thẻ HTML.
- Cú pháp: thuộc tính `style="..."`

Ví dụ:

```
<p style="color: blue; font-size: 18px;">Chào bạn!</p>
```

### **1.3.5. CSS thông dụng**

#### **1.3.5.1. Font**

Ví dụ:

```
p {  
    font-family: Arial;  
    font-size: 18px;  
    color: blue;  
}
```

#### **1.3.5.2. Căn chỉnh văn bản**

Ví dụ:

```
h1 {  
    text-align: center;  
}
```

#### **1.3.5.3. Màu nền, màu chữ**

Ví dụ:

```
body {  
    background-color: #f0f0f0;  
    color: #333;  
}
```

#### **1.3.5.4. Định dạng hộp**

Ví dụ:

```
div {  
    padding: 10px;  
    margin: 20px;  
    border: 2px solid black;  
}
```

### **1.3.5.5. Hover – hiệu ứng khi di chuột**

Ví dụ:

```
a:hover {  
    color: green;  
    text-decoration: underline;  
}
```

### **1.3.5.6. Tổng hợp**

Ví dụ:

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
  
    <style>  
  
        body {  
            background-color: #f5f5f5;  
            font-family: Arial;  
        }  
  
        h1 {  
            color: navy;  
            text-align: center;  
        }  
  
        .box {  
            padding: 15px;  
            margin: 20px auto;  
            width: 300px;  
            border: 1px solid #ccc;  
            background-color: white;  
        }  
    </style>  
</head>  
  
<body>  
  
    <h1>Hello World!</h1>  
  
    <div class="box">  
        <p>This is a sample text inside a box.</p>  
    </div>  
  
</body>
```

```
a:hover {  
    color: red;  
}  
</style>  
</head>  
<body>  
    <h1>Chúc bạn học tốt</h1>  
    <div class="box">  
        <p>Đây là một ví dụ CSS cơ bản.</p>  
    </div>  
</body>  
</html>
```

## 1.4. GIỚI THIỆU JAVASCRIPT

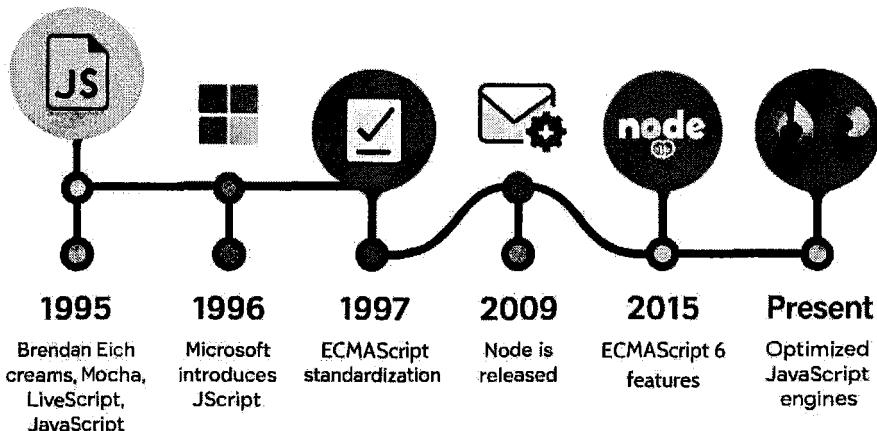
### 1.4.1. Nguồn gốc và lịch sử phát triển của JavaScript

JavaScript ban đầu được Brendan Eich phát triển tại Công ty Netscape vào năm 1995. Ông viết ngôn ngữ này trong vòng 10 ngày với tên gốc là Mocha, sau đó đổi thành LiveScript và cuối cùng là JavaScript. Mục tiêu ban đầu là giúp người dùng không chuyên có thể tạo trang Web tương tác. Cái tên JavaScript được đặt nhằm tạo đà theo danh tiếng Java.

JavaScript ban đầu được sử dụng trong trình duyệt Netscape Navigator để làm trang HTML vào thời điểm đó hấp dẫn hơn và có khả năng tương tác với người dùng. Năm 1996, Microsoft giới thiệu JScript để cạnh tranh, dẫn đến tình trạng phân mảnh theo trình duyệt. Năm 1997, ngôn ngữ được chuẩn hóa dưới tên ECMAScript do tổ chức ECMA quản lý. Từ đó, JavaScript được duy trì và phát triển bởi Ủy ban kỹ thuật TC39.

JavaScript ngày càng được cải tiến qua nhiều phiên bản chuẩn ECMAScript. XMLHttpRequest (1999) là bước đệm cho Ajax. Các trình duyệt như Firefox, Chrome, Edge tối ưu hóa JavaScript engine. JavaScript trở thành trụ cột phát triển Web.

Đến năm 2009, Node.js xuất hiện, cho phép chạy JavaScript trên máy chủ, mở ra lập trình full-stack.



#### *Sơ đồ minh họa sự phát triển của JavaScript*

JavaScript theo chuẩn ECMAScript đã phát triển qua nhiều phiên bản, trong đó quan trọng nhất là ECMAScript5 (2009) và ECMAScript6 hay ES2015 (2015).

#### **1.4.2. ECMAScript5 (2009)**

ES5 đánh dấu cột mốc quan trọng với nhiều cải tiến giúp mã JavaScript an toàn và dễ bảo trì hơn:

##### ***Strict mode:***

```
'use strict';  
  
// Gây lỗi nếu dùng biến chưa khai báo  
x = 10; // Lỗi!
```

##### ***Hàm mảng mới:***

```
let numbers = [1, 2, 3, 4];  
  
let even = numbers.filter(function(n) { return n % 2 === 0; });  
  
console.log(even); // [2, 4]
```

#### **1.4.3. ECMAScript 6/ES2015 (2015)**

ES2015 mang đến nhiều cú pháp hiện đại, giúp viết mã ngắn gọn và rõ ràng hơn:

### ***let/const thay thế var:***

```
let age = 18;  
const PI = 3.14;
```

### ***Arrow function:***

```
const square = x => x * x;  
console.log(square(5)); // 25
```

## **1.4.4. Các phiên bản sau ES2015**

Từ ES2016 đến ES2024, JavaScript liên tục được nâng cấp hằng năm, bổ sung các tính năng tiện lợi như:

- Async/await (ES2017);
- Spread/rest cho object (ES2018);
- Object.entries, Object.values;
- String.prototype.padStart, padEnd.

Mỗi phiên bản mặc dù bổ sung các tính năng khác nhau nhưng đều hướng tới việc cải thiện hiệu suất và trải nghiệm lập trình, giúp mã ngắn gọn, dễ đọc và dễ bảo trì hơn.

## **1.4.5. Tổng quan về JavaScript**

### **1.4.5.1. Cú pháp cơ bản**

JavaScript có cú pháp tương đối gần gũi với các ngôn ngữ như C hoặc Java, trong đó các khối lệnh được bao bọc trong cặp dấu ngoặc nhọn {} và mỗi lệnh kết thúc thông thường có dấu ;.

Việc khai báo biến có thể sử dụng ba từ khoá var, let và const. Chẳng hạn, ta có thể khai báo:

```
let year = 2025;  
const PI = 3.14;  
console.log("Năm nay: " + year);
```

Các câu lệnh điều khiển như if, for, hoặc while có cách viết giống với nhiều ngôn ngữ khác:

```
for (let i = 1; i <= 3; i++) {  
    console.log(i);  
}
```

#### **1.4.5.2. Môi trường chạy**

Ban đầu, JavaScript được sinh ra để chạy trên các trình duyệt Web, nhằm giúp trang Web tương tác với người dùng. Mã JavaScript có thể được nhúng trực tiếp trong HTML thông qua các thẻ `<script>...</script>` và sẽ được thực thi khi trang được tải trong trình duyệt. Từ năm 2009, nhờ vào Node.js, JavaScript đã có thể chạy ở môi trường ngoài trình duyệt, cho phép lập trình back-end, truy cập hệ thống file và kết nối cơ sở dữ liệu. Bên cạnh đó, JavaScript còn được dùng trong nhiều nền tảng khác như Deno, React Native (cho ứng dụng di động) và Electron (viết ứng dụng desktop).

#### **1.4.5.3. Kiểu dữ liệu, biến và hàm**

JavaScript hỗ trợ nhiều kiểu dữ liệu cơ bản như Number (số), String (chuỗi), Boolean (true/false), Object (đối tượng), Array (mảng), Null và Undefined. Một ví dụ minh họa cho các kiểu trên:

```
let x = 10;  
let s = "Hello";  
let ok = true;  
let arr = [1, 2, 3];  
let obj = {a: 1};  
let z = null;  
let t;
```

Việc khai báo biến có thể dùng var (kiểu cũ, phạm vi trong hàm), let (mới hơn, phạm vi khối lệnh) hoặc const (biến hằng):

```
var a = 1;  
let b = 2;  
const c = 3;
```

Cuối cùng, hàm trong JavaScript có thể được khai báo theo cách truyền thống với từ khóa function:

```
function cong(a, b) {  
    return a + b;  
}
```

..., hoặc theo cách ngắn hơn với arrow function:

```
const nhan = (a, b) => a * b;
```

#### **1.4.5.4. Tình trạng hiện tại của JavaScript trong thế giới lập trình**

JavaScript hiện là ngôn ngữ phổ biến nhất trên Web, được sử dụng để tạo giao diện tương tác cho hầu hết các trang Web hiện đại. Theo StackOverflow 2024, 62,3% lập trình viên dùng JavaScript, giúp nó giữ vị trí số 1. JavaScript phù hợp cả cho người mới bắt đầu nhờ dễ thử nghiệm trong trình duyệt.

Không chỉ giới hạn trong Web front-end, JavaScript còn được sử dụng rộng rãi trong back-end với Nodejs, ứng dụng di động, desktop và cả trong AI hay IoT.

JavaScript phổ biến nhờ được hỗ trợ trực tiếp trong trình duyệt, có hệ sinh thái thư viện mã nguồn mở phong phú và cộng đồng lập trình viên đông đảo. Nhờ vậy, JavaScript tiếp tục là ngôn ngữ trung tâm của Web hiện đại và sẽ còn phát triển mạnh mẽ trong tương lai.

#### **1.4.5.5. Độ khó khi học JavaScript**

So với Python, JavaScript linh hoạt hơn nhưng phức tạp hơn về các cơ chế như prototype, callback hay Promise. Python sử dụng thực đầu dòng thay vì dấu ngoặc nhọn, cú pháp đơn giản và dễ đọc nên phù hợp với người mới. Tuy nhiên, JavaScript lại có lợi thế chạy ngay trong trình duyệt, không cần cài đặt, giúp thử nghiệm dễ dàng qua DevTools.

So với Java, JavaScript có cú pháp ngắn gọn và linh hoạt hơn, không cần khai báo kiểu dữ liệu rõ ràng và dễ bắt đầu hơn. Java là ngôn ngữ nghiêm ngặt về định kiểu, tổ chức theo lớp/gói và yêu cầu biên dịch trước khi chạy. Trong khi Java phù hợp với các dự án lớn cần tính đồng nhất cao, JavaScript lại thích hợp cho những người muốn bắt đầu nhanh và trực quan.

Tóm lại, JavaScript dễ thử nghiệm ngay trên trình duyệt, nhiều tài liệu thân thiện, phù hợp với học sinh THPT. Python đơn giản, có cấu trúc rõ ràng và thích hợp cho khoa học dữ liệu. Java tuy mạnh mẽ về quy mô nhưng có thể khó hơn với người mới bắt đầu do cú pháp và môi trường phức tạp hơn.

JavaScript phù hợp với lập trình trung học phổ thông và thi tốt nghiệp.

## CHƯƠNG II

# HTML

### 2.1. TAG

#### 2.1.1. Giới thiệu về thẻ

Trong HTML, thẻ là thành phần cốt lõi để xây dựng cấu trúc và trình bày nội dung một trang Web. Phiên bản HTML hiện nay – HTML5 Living Standard – có khoảng 100 thẻ chính thức, mỗi thẻ có một vai trò riêng biệt, được chia thành nhiều nhóm chức năng như:

- Thẻ văn bản: <h1> – <h6>, <p>, <strong>, <em>, v.v.;
- Thẻ định dạng tài liệu: <html>, <head>, <title>, <body>;
- Thẻ liên kết và điều hướng: <a>, <nav>, <link>;
- Thẻ đa phương tiện: <img>, <audio>, <video>, <source>;
- Thẻ danh sách: <ul>, <ol>, <li>;
- Thẻ bảng: <table>, <tr>, <td>, <th>;
- Thẻ biểu mẫu: <form>, <input>, <textarea>, <button>, <label>;
- Thẻ cấu trúc trang: <header>, <footer>, <section>, <article>;

v.v..

Sự đa dạng về chức năng của các thẻ HTML giúp lập trình viên Web tạo nên những trang Web phong phú về nội dung và linh hoạt trong hiển thị. Việc nắm rõ vai trò, cách dùng và phạm vi áp dụng của từng thẻ là cần thiết khi tương tác và sử dụng HTML.

#### 2.1.2. Cấu trúc của mỗi thẻ

Trước khi tìm hiểu về chức năng các thẻ, chúng ta cùng tìm hiểu về cấu trúc của mỗi thẻ.

<tag\_name> Nội dung </tag\_name>

Ví dụ: <h1> Đây là ví dụ </h1>

Một phần tử (element) của HTML được xác định bởi các thẻ (tags). Một thẻ sẽ bao gồm tên phần tử trong các ngoặc góc <>. Các trình duyệt được lập trình để giấu đi các văn bản bên trong các cặp ngoặc góc và do đó chúng sẽ không xuất hiện trên Web.

Tên phần tử xuất hiện trong thẻ mở (opening/start tag) và xuất hiện lần nữa trong thẻ đóng (closing/end tag). Trong thẻ đóng, tên phần tử sẽ được xếp sau dấu "/". Thẻ đóng hoạt động như một công tắc đóng, giúp phân biệt các dòng code với nhau.

**Chú ý:** Các thẻ đóng của HTML sử dụng dấu "/", không phải dấu "\" hoặc dấu "|".

Các thẻ được sử dụng quanh phần nội dung được gọi là một đánh dấu (markup). Mặc dù một phần tử bao gồm cả phần markup và phần nội dung nhưng không phải phần tử nào cũng sẽ chứa nội dung. Một vài phần tử là rỗng theo mặc định, chẳng hạn như thẻ <img> dùng để chèn ảnh vào trang Web:

<img src = "URL" alt "van\_ban"> </img>

Cuối cùng, các thẻ của HTML không bị ảnh hưởng bởi việc viết hoa hay viết thường. Vì thế, các dòng code bên dưới có ý nghĩa như nhau đối với trình duyệt:

<img src = "URL" alt "van\_ban"> </img>

<IMG src = "URL" alt "van\_ban"> </IMG>

<iMg src = "URL" alt "van\_ban"> </Img>

Tuy nhiên, hầu hết lập trình viên đều thống nhất sử dụng cách viết thường toàn bộ, vừa để code nhìn đẹp vừa dễ quản lý.

**Chú ý:** Trong một số trường hợp, các dòng code được mở và đóng bởi một cặp ngoặc góc không phải là một phần tử. Ví dụ, dòng code thường gặp bên dưới:

<!DOCTYPE html>

không phải là một phần tử, mà nó xác định loại văn bản, giúp các trình duyệt hiện đại có thể chọn được phiên bản HTML phù hợp để chạy chương trình. Dòng code trên thể hiện rằng chương trình được viết bằng HTML5.

### **2.1.3. Các thẻ quan trọng trong HTML**

Mặc dù HTML có chứa hơn 100 thẻ chính thức, nhưng chỉ một số thẻ có vai trò quan trọng hơn cả, là cốt lõi giúp tạo nên một trang Web hoàn thiện. Các thẻ này có thể kể đến như:

- <html>: Còn được gọi là phần tử gốc (tên tiếng anh là root element), cặp thẻ <html> chứa toàn bộ các phần tử trong tài liệu trong khi không nằm bên trong bất kỳ phần tử nào.
- <head>: Chứa thông tin “ẩn” như tiêu đề trang hay các liên kết đến CSS, JavaScript, v.v.
- <title>: Tên trang hiển thị trên thanh tiêu đề trình duyệt.
- <body>: Chứa toàn bộ nội dung sẽ hiển thị trên trang Web.
- <h1> – <h6>: Dùng để tạo tiêu đề. <h1> là tiêu đề lớn nhất (quan trọng nhất), <h6> là tiêu đề nhỏ nhất (ít quan trọng nhất). HTML chỉ có từ <h1> – <h6>.
- <p>: Dùng để viết đoạn văn bản.
- <a>: Tạo liên kết (link) đến trang khác hoặc một phần khác cùng trang.
- <img>: Chèn hình ảnh vào trang Web.
- <ul>/<ol>/<li>: Tạo danh sách (dạng chấm hoặc số).
- <table>/<tr>/<td>: Tạo bảng và các ô trong bảng.
- <form>/<input>/<button>: Dùng để tạo biểu mẫu nhập liệu như đăng ký, đăng nhập.

Chỉ cần sử dụng những thẻ trên, người dùng đã có thể tạo một trang Web khá hoàn chỉnh.

## **2.2. HEAD**

### **2.2.1. Giới thiệu**

Trình duyệt Web xử lý một tài liệu HTML theo tuần tự từ trên xuống. Bằng việc phân tích (parse) nội dung của <head> đầu tiên, trình duyệt có thể thiết lập các ngữ cảnh và tài nguyên cần thiết: bộ ký tự, tiêu đề trang, các bảng định kiểu CSS, một số kịch bản JavaScript, trước khi bắt đầu quá trình kết xuất (render) nội dung trực quan trong <body>.

Các chức năng chính của `<head>` bao gồm:

- Khai báo siêu dữ liệu: cung cấp thông tin cho trình duyệt (user agent), các công cụ tìm kiếm (web crawlers) và các hệ thống xử lý tài liệu khác.
- Thiết lập ngữ cảnh tài liệu: định nghĩa tiêu đề, bộ ký tự và các thông tin nền tảng khác.
- Liên kết tài nguyên ngoài: khai báo các mối quan hệ với các tệp bên ngoài như CSS, JavaScript, phông chữ và biểu tượng trang.
- Nhúng mã nguồn: chứa các khối mã CSS hoặc JavaScript nội bộ.

– Kiểm soát hiển thị và hành vi: cung cấp các chỉ thị về cách hiển thị trên các thiết bị (viewport) và các chỉ thị tương đương với tiêu đề HTTP (HTTP pragmas).

Các phần tử hợp lệ có thể được lồng bên trong `<head>` bao gồm: `<title>`, `<style>`, `<meta>`, `<link>`, `<script>` và `<base>`. Mỗi phần tử này thực hiện một chức năng riêng biệt, góp phần xây dựng nên cấu trúc và hành vi của một trang Web.

### **2.2.2. Phần tử `<title>`**

Phần tử `<title>` là một thành phần bắt buộc trong mọi tài liệu HTML hợp lệ. Chức năng duy nhất của nó là định nghĩa tiêu đề cho trang Web. Tiêu đề này không được hiển thị trong khu vực nội dung chính của trang nhưng được các trình duyệt sử dụng ở nhiều vị trí quan trọng.

#### **2.2.2.1. Vai trò và ứng dụng**

- Thanh tiêu đề và tab trình duyệt: Đây là nơi hiển thị rõ ràng nhất, cho phép người dùng nhận diện và điều hướng giữa nhiều tab đang mở.
- Đánh dấu trang (Bookmarks): Khi người dùng lưu một trang web, nội dung của `<title>` được sử dụng làm tên mặc định cho bookmark đó.
- Lịch sử duyệt web: Tiêu đề được ghi lại trong lịch sử trình duyệt, hỗ trợ việc tìm kiếm và truy cập lại các trang đã xem.
- Kết quả của công cụ tìm kiếm (SERPs): Tiêu đề trang thường được các công cụ tìm kiếm như Google sử dụng làm tiêu đề có thể nhấp vào trong trang kết quả tìm kiếm.
- Khả năng truy cập (Accessibility): Đối với người dùng sử dụng công nghệ hỗ trợ như trình đọc màn hình, `<title>` là thông tin đầu tiên được đọc lên khi họ truy cập một trang mới, cung cấp ngữ cảnh tức thì về nội dung của trang.

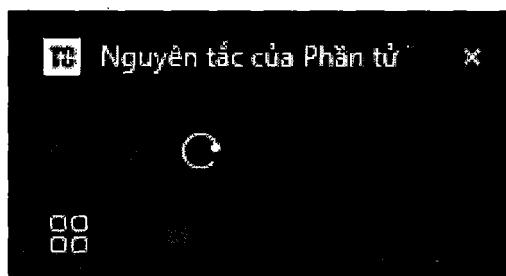
### **2.2.2.2. Quy tắc và lưu ý**

- Tính duy nhất: Mỗi tài liệu HTML chỉ được phép chứa một và chỉ một phần tử `<title>`.
- Nội dung: Nội dung bên trong `<title>` phải là văn bản thuần túy (plain text). Bất kỳ thẻ HTML nào được đặt bên trong sẽ được diễn giải dưới dạng văn bản thô.
- Tính súc tích: Mặc dù không có giới hạn kỹ thuật nghiêm ngặt về độ dài nhưng các trình duyệt thường cắt bớt các tiêu đề dài. Một tiêu đề hiệu quả cần ngắn gọn, mang tính mô tả cao và thường dưới 60 ký tự.

Ví dụ:

```
<head>
<meta charset="UTF-8">
<title>Nguyên tắc của Phần tử Title trong HTML</title>
</head>
<body>
<!-- Nội dung trang -->
</body>
```

Tiêu đề “Nguyên tắc của Phần tử Title trong HTML” đã được hiển thị.



### **2.2.3. Phần tử `<style>`**

Phần tử `<style>` là một cách để nhúng trực tiếp CSS vào một tài liệu HTML. Phương pháp này được gọi là CSS nội bộ (Internal CSS), khác với CSS ngoại tuyến (External CSS) được liên kết qua thẻ `<link>` và CSS nội tuyến (Inline CSS) được khai

báo trong thuộc tính style của một phần tử.

Các cú pháp CSS được định nghĩa trong thẻ `<style>` chỉ có phạm vi ảnh hưởng lên tài liệu chứa nó. Điều này khiến nó hữu ích với các kiểu dáng đặc thù chỉ áp dụng cho một trang duy nhất hoặc trong các tình huống không cần thiết phải tạo một tệp CSS riêng.

#### **2.2.3.1. Cú pháp và thuộc tính**

- Các cú pháp CSS được viết giữa thẻ mở `<style>` và thẻ đóng `</style>`.
- Type: Thuộc tính này chỉ định kiểu MIME của nội dung. Đối với CSS, giá trị là `text/css`. Trong HTML5, đây là giá trị mặc định và có thể bỏ qua.
- Media: Thuộc tính này cho phép bạn áp dụng các quy tắc CSS dành riêng cho một phương tiện cụ thể. Chẳng hạn, có thể dùng `media="screen"` cho màn hình hiển thị, `media="print"` cho phiên bản in hoặc `media="all"` cho mọi loại phương tiện."

Ví dụ:

```
<head>
<meta charset="UTF-8">
<title>Minh họa Thẻ Style</title>
<style media="screen">
body {
font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif;
background-color: #f8f9fa;
color: #212529;
}
h1 {
color: #0056b3;
border-bottom: 2px solid #dee2e6;
}
.text-muted {
```

```
color: #6c757d;  
}  
</style>  
</head>  
<body>  
<h1>Tiêu đề Chính của Trang</h1>  
<p>Một đoạn văn bản thông thường.</p>  
<p class="text-muted">Đoạn văn bản này có màu khác do áp dụng class.</p>  
</body>
```

## Tiêu đề Chính của Trang

Một đoạn văn bản thông thường.

Đoạn văn bản này có màu sắc khác do áp dụng class.

Trang Web sẽ hiển thị như trên.

### 2.2.3.2. Lời khuyên

Luôn đặt thẻ `<style>` bên trong phần tử `<head>`. Việc này đảm bảo rằng các quy tắc định dạng được tải và phân tích trước khi trình duyệt bắt đầu kết xuất (render) nội dung trong `<body>`. Điều này giúp ngăn hiện tượng “Flash of Unstyled Content” (FOUC), tình trạng trang web hiển thị chớp nhoáng mà không có định dạng CSS, gây ảnh hưởng tiêu cực đến trải nghiệm người dùng.

### 2.2.4. Phần tử `<meta>`

Phần tử `<meta>` là công cụ cực kỳ linh hoạt để cung cấp nhiều loại siêu dữ liệu khác nhau. Đây là một thẻ tự đóng (self-closing tag) và không hiển thị bất kỳ nội dung nào, nhưng nó lại truyền tải những thông tin thiết yếu cho trình duyệt và các dịch vụ Web khác.

## **Các ứng dụng phổ biến và thuộc tính tương ứng:**

### **a) Khai báo bộ ký tự**

Đây là ứng dụng quan trọng nhất của <meta> và luôn phải được đặt ở vị trí đầu tiên trong <head>. Nó đảm bảo trình duyệt giải mã chính xác các ký tự trong tài liệu. UTF-8 là bộ ký tự tiêu chuẩn được sử dụng toàn cầu.

```
<meta charset="UTF-8">
```

### **b) Cấu hình Viewport cho thiết kế đáp ứng (Responsive Design)**

Thẻ <meta> này kiểm soát kích thước và tỷ lệ của trang trên các thiết bị di động.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Giải thích:

width=device-width: Thiết lập chiều rộng của viewport bằng với chiều rộng màn hình của thiết bị.

initial-scale=1.0: Thiết lập mức thu phóng ban đầu là 100%, không phóng to hay thu nhỏ.

### **c) Mô tả trang và tác giả**

Các thẻ meta này sử dụng cặp thuộc tính name và content để định nghĩa siêu dữ liệu.

```
<!-- Giải thích: Cung cấp một bản tóm tắt ngắn gọn về nội dung trang -->
```

```
<meta name="description" content="Hướng dẫn chi tiết về các thẻ trong <head> HTML, bao gồm title, meta, link, style, script và base.">
```

```
<!-- Giải thích: Chỉ định tác giả của tài liệu -->
```

```
<meta name="author" content="Tên Tổ chức Giáo dục hoặc Tác giả">
```

Mặc dù description không còn là yếu tố xếp hạng quan trọng, nhưng nội dung của nó thường được các công cụ tìm kiếm sử dụng làm đoạn trích hiển thị bên dưới tiêu đề trang trong kết quả tìm kiếm, ảnh hưởng đến tỷ lệ nhấp chọn của người dùng.

### **d) Chỉ thị HTTP-Equiv**

Thuộc tính http-equiv cho phép thẻ <meta> mô phỏng một tiêu đề phản hồi HTTP (HTTP response header), ra lệnh cho trình duyệt thực hiện một hành động cụ thể.

<!--Giải thích: Yêu cầu trình duyệt Internet Explorer sử dụng chế độ render mới nhất (cho mục đích tương thích ngược) -->

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">  
<!--Giải thích: Tự động tải lại trang sau 300 giây -->  
<meta http-equiv="refresh" content="300">  
<!--Giải thích: Chuyển hướng đến một URL khác sau 5 giây -->  
<meta http-equiv="refresh" content="5;url=https://example.com">
```

## 2.2.5. Phần tử **<link>**

Phần tử **<link>** được sử dụng để định nghĩa mối quan hệ giữa tài liệu HTML và các tài nguyên bên ngoài. Đây là một thẻ tự đóng, vị trí của nó phải nằm trong **<head>**. Thuộc tính **rel** (relationship) là yếu tố cốt lõi giúp xác định bản chất của mỗi liên kết.

### **Ứng dụng chính:**

#### a) *Liên kết bảng định kiểu CSS (External CSS)*

Đây là công dụng phổ biến và quan trọng nhất, cho phép tách biệt mã CSS khỏi HTML, giúp mã nguồn trở nên sạch sẽ, dễ bảo trì và có khả năng tái sử dụng cao.

```
<link rel="stylesheet" href="/css/main.css" type="text/css" media="screen">
```

Giải thích:

- **rel="stylesheet"**: Xác định rằng tài nguyên được liên kết là một bảng định kiểu.
- **href**: Chỉ định đường dẫn URL đến tệp .css.
- **type="text/css"** (Tùy chọn trong HTML5): Chỉ định kiểu MIME.
- **media**: Chỉ định phương tiện áp dụng (ví dụ: screen, print).

#### b) *Liên kết biểu tượng trang (Favicon)*

Ví dụ:

```
<link rel="icon" href="/favicon.ico" type="image/x-icon">  
<link rel="icon" href="/favicon.png" type="image/png">  
<link rel="apple-touch-icon" href="/apple-touch-icon.png">
```

c) *Gợi ý tài nguyên (Resource Hints) để tối ưu hóa hiệu suất*

Các gợi ý này cho phép trình duyệt thực hiện các hành động tối ưu hóa một cách chủ động.

– dns-prefetch: Yêu cầu trình duyệt phân giải DNS cho một tên miền trước khi cần đến.

```
<link rel="dns-prefetch" href="//fonts.gstatic.com">
```

– preconnect: Yêu cầu trình duyệt thiết lập kết nối sớm đến một tên miền (DNS, TCP, TLS).

```
<link rel="preconnect" href="https://api.example.com" crossorigin>
```

– preload: Chỉ thị trình duyệt tải sớm một tài nguyên quan trọng với độ ưu tiên cao. Thuộc tính as (bắt buộc) chỉ định loại tài nguyên.

```
<link rel="preload" href="/fonts/critical-font.woff2" as="font" type="font/woff2" crossorigin>
```

```
<link rel="preload" href="/js/critical-script.js" as="script">
```

d) *Khai báo URL chuẩn (Canonical URL)*

Sử dụng rel="canonical" để chỉ định phiên bản URL ưu tiên của một trang, giúp các công cụ tìm kiếm xử lý các vấn đề về trùng lặp nội dung.

```
<link rel="canonical" href="https://example.com/san-pham/ao-thun">
```

## 2.2.6. Phần tử <script>

Phần tử <script> được dùng để nhúng mã kịch bản thực thi phía máy khách (client-side), phổ biến nhất là JavaScript. Nó giúp tạo ra các trang Web động, có tính tương tác cao.

### Vị trí và ảnh hưởng đến hiệu suất:

Việc đặt thẻ <script> có ảnh hưởng trực tiếp đến thời gian tải và hiển thị của trang.

#### a) Trong <head> (mặc định)

Một thẻ <script> không có thuộc tính async hoặc defer sẽ là render-blocking. Trình duyệt sẽ tạm dừng việc phân tích HTML, nó sẽ tải và thực thi kịch bản trước

khi tiếp tục. Điều này có thể làm chậm đáng kể thời gian hiển thị trang nếu trang có kích bản lớn hoặc tốc độ tải chậm.

b) Cuối thẻ `<body>`

Đây là phương pháp truyền thống để tránh chặn hiển thị. Trình duyệt sẽ phân tích và hiển thị toàn bộ nội dung HTML trước, sau đó mới tải và thực thi kịch bản.

Các thuộc tính `async` và `defer` (chỉ dành cho kịch bản ngoài): HTML5 giới thiệu hai thuộc tính để tối ưu hóa việc tải kịch bản khi đặt trong `<head>`.

– `defer`: Trình duyệt sẽ tải kịch bản song song với việc phân tích HTML (không chặn), nhưng chỉ thực thi nó sau khi toàn bộ tài liệu đã được phân tích xong và theo đúng thứ tự chúng xuất hiện. Đây là lựa chọn được ưu tiên cho các kịch bản cần tương tác với DOM.

```
<script src="app.js" defer></script>  
<script src="analytics.js" defer></script> <!-- Sẽ chạy sau app.js -->
```

– `async`: Trình duyệt cũng tải kịch bản song song (không chặn), nhưng sẽ thực thi nó ngay khi tải xong, không cần đợi phân tích HTML và không theo thứ tự. Phù hợp cho các kịch bản độc lập như quảng cáo hoặc phân tích.

```
<script src="ads.js" async></script>
```

Ví dụ (sử dụng `defer`):

```
<head>  
  <!-- ... Giải thích: Các thẻ meta, title, link ... -->  
  <script src="/js/main-library.js" defer></script>  
  <script src="/js/application-logic.js" defer></script>  
</head>
```

## 2.2.7. Phần tử `<base>`

Phần tử `<base>` chỉ định một URL cơ sở sẽ được sử dụng để phân giải tất cả các URL tương đối (relative URLs) trong tài liệu, bao gồm các liên kết (`<a>`), hình ảnh (`<img>`), kịch bản (`<script>`) và bảng định kiểu (`<link>`).

Chỉ có thể có một phần tử `<base>` trong một tài liệu và nó phải nằm trong `<head>`.

#### **2.2.7.1. Thuộc tính**

- href: Chỉ định URL cơ sở.
- target: Chỉ định ngữ cảnh duyệt Web mặc định cho các liên kết (ví dụ: \_blank, \_self).

Ví dụ:

```
<head>
<title>Ví dụ Thẻ Base</title>
<base href="https://www.example.com/assets/" target="_blank">
</head>
<body>
<!-- Đường dẫn này sẽ được phân giải thành "https://www.example.com/
assets/images/logo.png"-->

<!-- Liên kết này sẽ mở "https://www.example.com/assets/about.html" trong
một tab mới -->
<a href="about.html">Về chúng tôi</a>
</body>
```

#### **2.2.7.2. Lưu ý quan trọng**

Thẻ `<base>` có tác động toàn cục và có thể gây ra các hành vi không mong muốn, đặc biệt là với các liên kết neo (anchor links) trả đến các mảnh vụn (fragment) trong cùng một trang (ví dụ: `<a href="#section2">`). Vì lý do này, việc sử dụng nó cần được cân nhắc kỹ lưỡng và thường được thay thế bằng các giải pháp quản lý đường dẫn phía máy chủ hoặc trong quá trình xây dựng dự án (build process) trong các ứng dụng Web hiện đại.

### **2.3. HIỂN THỊ CÁC KÝ TỰ**

Để xử lý văn bản và chữ, chúng ta có hai loại thẻ: thẻ hiển thị chữ (Text Display Tags) và thẻ định dạng văn bản (Text Formatting Tags).

Các thẻ sau đây được sắp xếp theo mức độ thông dụng từ cao xuống thấp.

### **2.3.1. Các thẻ hiển thị văn bản phổ biến**

#### **2.3.1.1. Thẻ `<p>`**

`<p>` viết tắt cho Paragraphs (đoạn văn), là một trong những thẻ cơ bản và thông dụng nhất trong HTML. Để tạo một đoạn văn, người dùng chỉ cần bổ sung phần nội dung vào giữa cặp thẻ `<p>` và `</p>`.

Ví dụ: `<p> Đây là một đoạn văn. </p>` `<p> Đây là đoạn văn thứ hai </p>`.

Kết quả:

**Đây là một đoạn văn.**

**Đây là đoạn văn thứ hai**

Các trình duyệt gần như luôn luôn hiển thị các đoạn văn trên các dòng khác nhau, các dòng cách nhau một khoảng cách nhỏ theo mặc định (hay theo thuật ngữ của CSS – chúng được hiển thị dưới dạng các khối (blocks)).

Nội dung một đoạn văn không bị giới hạn chỉ trong các văn bản mà còn có thể bao gồm hình ảnh hay các phần tử nội tuyến (inline elements) khác. Tuy vậy, các đoạn văn không nên chứa các tiêu đề, danh sách hay bất cứ các phần tử nào được thể hiện dưới dạng khối theo mặc định.

**Chú ý:** Thẻ `<p>` không gây lỗi kể cả khi thiếu đi thẻ đóng, do các trình duyệt sẽ tự động coi thẻ `<p>` đã kết thúc khi bắt gặp một thẻ có dạng khối tiếp theo. Tuy nhiên, hầu hết lập trình viên đều đóng thẻ `<p>` để đảm bảo tính ổn định và mạch lạc của code.

#### **2.3.1.2. Thẻ tiêu đề `<h1>` – `<h6>`**

Giống thẻ `<p>`, các thẻ tiêu đề (Headings) cũng rất phổ biến trong lập trình Web. Có sáu cấp độ của các thẻ tiêu đề, với mức độ quan trọng của tiêu đề giảm dần theo số hạng của thẻ.

Các thẻ `<h1>` – `<h6>` rất thường được sử dụng kèm với thẻ `<p>`, lấy ví dụ đoạn code sau:

`<h1> Thẻ định dạng chữ </h1> <!—Tiêu đề cấp cao nhất —>`

`<h3> Thẻ hiển thị chữ </h3> <!—Tiêu đề cấp cao thứ ba —>`

`<p> Thẻ paragraph </p>`

```
<p> Thẻ headings </p>
<h3> Thẻ định dạng văn bản </h3><!--Tiêu đề cấp cao thứ ba -->
<p> Thẻ a </p>
<p> Thẻ b </p>
```

Kết quả:

## Thẻ định dạng chữ

Thẻ hiển thị chữ

Thẻ paragraph

Thẻ headings

Thẻ định dạng văn bản

Thẻ a

Thẻ b

Theo mặc định, các tiêu đề được bôi đen, đồng thời cỡ chữ của chúng sẽ giảm dần tương ứng với mức độ quan trọng của tiêu đề.

Thông tin phụ: Các thiết bị hỗ trợ việc đọc sẽ quét qua các tiêu đề trên trang Web để giúp người dùng có thể dễ dàng tiếp nhận thông tin. Các thuật toán tìm kiếm cũng sẽ xem xét mức độ quan trọng của các tiêu đề để quyết định mức độ ưu tiên của thông tin. Vì thế, các lập trình viên mới nên bắt đầu với việc sử dụng thẻ `<h1>`, sau đó chuyển sang các thẻ tiêu đề cấp thấp hơn để thiết kế Web hiệu quả.

### 2.3.1.3. Thẻ `<hr>` (thẻ kẻ đường ngang)

Khi muốn thể hiện rằng một phần nào đó đã kết thúc, người dùng có thể kẻ một đường ngang với thẻ `<hr>`. Phần tử `<hr>` sẽ phân chia một cách rõ ràng và logic các phần của một trang hoặc một đoạn mà không cần sử dụng một thẻ tiêu đề mới.

Khác với hai thẻ bên trên, thẻ `<hr>` rỗng và chỉ đứng một mình.

Kết hợp với đoạn code bên trên:

```
<h1> Thẻ định dạng chữ </h1> <!--Tiêu đề cấp cao nhất -->
<h3> Thẻ hiển thị chữ </h3> <!--Tiêu đề cấp cao thứ ba -->
```

```
<p> Thẻ paragraph </p>
<p> Thẻ headings </p>
<hr> <!-- Kẻ một đường ngang giữa đoạn -->
<h3> Thẻ định dạng văn bản </h3><!-- Tiêu đề cấp cao thứ ba -->
<p> Thẻ a </p>
<p> Thẻ b </p>
```

Kết quả:

## Thẻ định dạng chữ

### Thẻ hiển thị chữ

Thẻ paragraph

Thẻ headings

### Thẻ định dạng văn bản

Thẻ a

Thẻ b

### 2.3.1.4. Thẻ **<br>** (thẻ xuống dòng)

Để ngắt dòng, người dùng có thể sử dụng thẻ **<br>**. Thẻ này sẽ chuyển nội dung dang sau xuống dòng mới nhưng không tạo ra một đoạn văn mới như thẻ **<p>**.

Giống thẻ **<hr>**, thẻ này rỗng và không có thẻ đóng. Đồng thời, **<br>** không thêm khoảng cách như thẻ **<p>** và chỉ xuống dòng một lần.

Khi sử dụng trong một đoạn code:

```
<p> HTML (viết tắt của HyperText Markup Language) <br> là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang web. <br> Được phát triển từ năm 1991 bởi Tim Berners-Lee <br> và phát hành chính thức vào năm 1993, <br> HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW </p>
```

Kết quả:

HTML (viết tắt của HyperText Markup Language)  
là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang web.  
Được phát triển từ năm 1991 bởi Tim Berners-Lee  
và phát hành chính thức vào năm 1993 ,  
HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW

Thẻ này rất hữu dụng để hiển thị một đoạn văn lớn thành nhiều đoạn nhỏ, hoặc đơn giản là khi người dùng muốn viết thơ, lời bài hát, v.v..

### 2.3.2. Các thẻ định dạng văn bản phổ biến

#### 2.3.2.1. Thẻ **< b >**, **< i >**, **< u >** (các thẻ nhấn mạnh nhẹ)

Ba thẻ trên là những thẻ định dạng văn bản phổ biến. Thẻ **< b >** dùng để bôi đậm chữ, giúp làm nổi bật những nội dung quan trọng nhưng không mang ý nghĩa nhấn mạnh nội dung. Thẻ **< i >** dùng để in nghiêng chữ, sử dụng để viết các thuật ngữ, tên tác phẩm, hoặc đoạn trích dẫn. Thẻ **< u >** dùng để gạch chân văn bản – trước đây thường dùng để đánh dấu các liên kết, nhưng hiện nay chủ yếu dùng làm nổi bật từ ngữ theo kiểu trình bày cũ hoặc các yêu cầu định dạng đặc biệt.

Ba thẻ này đều thuộc nhóm thẻ định dạng trực quan và không làm thay đổi cấu trúc nội dung văn bản.

Ví dụ:

**< b >** Đoạn văn bản này được bôi đậm **< /b >** **< br >**

**< i >** Đoạn văn bản này được in nghiêng **< /i >** **< br >**

**< u >** Đoạn văn bản này được gạch chân **< /u >**

Kết quả:

**Đoạn văn bản này được bôi đậm**

**Đoạn văn bản này được in nghiêng**

**Đoạn văn bản này được gạch chân**

Ba thẻ này cũng có thể sử dụng chung với nhau để tăng tính nhấn mạnh. Mặc dù không có thứ tự mặc định để tránh gây lỗi, người dùng nên sắp xếp chúng theo thứ tự mạch lạc và rõ ràng để tránh gây nhầm lẫn. Ví dụ:

**< b >****< i >**Đoạn văn bản này được bôi đậm và in nghiêng**< /i >****< /b >** **< br >**

**< b >****< u >**Đoạn văn bản này được bôi đậm và gạch chân**< /u >****< /b >** **< br >**

**< b >****< u >****< i >**Đoạn văn bản này được bôi đậm, in nghiêng và gạch chân**< /i >****< /u >****< /b >** **< br >**

**< b >****< u >****< i >**Đoạn văn bản này được bôi đậm, in nghiêng và gạch chân**< /b >****< /u >****< /i >** **<!-- Không gây lỗi nhưng không nên viết như này-->**

Kết quả:

Đoạn văn bản này được bôi đậm và in nghiêng  
Đoạn văn bản này được bôi đậm và gạch chân  
Đoạn văn bản này được bôi đậm, in nghiêng và gạch chân  
Đoạn văn bản này được bôi đậm, in nghiêng và gạch chân

Chú ý: Không nên liên tục lồng nhiều thẻ nếu không thật sự cần thiết. Nếu dùng quá nhiều định dạng cùng lúc có thể gây rối mắt hoặc làm giảm hiệu quả nhấn mạnh.

### 2.3.2.2. Thẻ **, *(các thẻ nhấn mạnh mạnh)***

Thẻ  **được dùng để nhấn mạnh nội dung quan trọng. Khi sử dụng, phần văn bản nằm giữa hai cặp thẻ mở đóng sẽ được bôi đậm. Mặc dù bề ngoài nhìn giống thẻ **, thẻ  **còn mang ý nghĩa ngữ nghĩa (semantic), giúp các công cụ tìm kiếm hiểu được phần thông tin đó là quan trọng.******

Thẻ *, viết tắt của emphasis, dùng để nhấn mạnh nhẹ hoặc làm nổi bật cảm xúc trong đoạn văn bản. Khi hiển thị, văn bản sẽ được in nghiêng giống thẻ *, nhưng nó có ý nghĩa giống như thẻ  **bên trên.****

Ví dụ:

<p>HTML, viết tắt của <b><i>HyperText Markup Language</i></b>, <br> là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng web</p>

<p>HTML, viết tắt của <strong><em>HyperText Markup Language</em></strong>, <br> là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng web</p>

Kết quả:

**HTML , viết tắt của *HyperText Markup Language* ,  
là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng web**

**HTML , viết tắt của *HyperText Markup Language* ,  
là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng web**

### 2.3.2.3. Thẻ <sub>, <sup>(thẻ chỉ số dưới/trên)</sup></sub>

Thẻ  <sub>được dùng để hiển thị văn bản dưới dòng chuẩn, thường thấy trong các công thức hóa học, toán học hoặc các ghi chú nhỏ. Văn bản bên trong thẻ  <sub>thường nhỏ hơn và nằm thấp xuống so với dòng bình thường.</sub></sub>

Ngược lại, thẻ `<sup>` dùng để hiển thị văn bản nằm trên dòng chuẩn, sử dụng nhiều trong các công thức toán học, lũy thừa, số mũ, v.v.. Giống như thẻ `<sub>`, văn bản nằm trong thẻ `<sup>` nhỏ hơn và nằm cao hơn so với dòng bình thường.

Ví dụ sử dụng:

`<p>H<sub>2</sub>O là công thức của nước.</p>`

`<p>6<sup>2</sup> + 8<sup>2</sup> = 10 <sup>2</sup></p>`

Kết quả:

H<sub>2</sub>O là công thức của nước.

$$6^2 + 8^2 = 10^2$$

#### 2.3.2.4. Sử dụng các ký tự đặc biệt trong HTML

Khi lập trình Web, một vài ký tự đặc biệt như `<`, `>`, `&`, `”`, v.v. không thể sử dụng trực tiếp do các trình duyệt sẽ nhầm chúng với các mã HTML.

Vì thế, để hiển thị các ký tự này cần sử dụng cú pháp:

`&[mã-ký-tự];`

Ví dụ:

`<p>5 &lt; 10 &amp;&gt; 5</p>`

`<p>Copyright &copy; 2025 bởi nhóm viết sách</p>`

Kết quả:

5 < 10 &&gt; 5

Copyright © 2025 bởi nhóm viết sách

Danh sách các ký tự đặc biệt thường dùng:

Ký tự muốn hiển thị	Mã HTML	Kết quả hiển thị
Dấu “Nhỏ hơn”	<code>&amp;lt;</code>	<
Dấu “Lớn hơn”	<code>&amp;gt;</code>	>
Dấu “và”	<code>&amp;amp;</code>	&
Dấu ngoặc kép	<code>&amp;quot;</code>	”
Dấu ngoặc đơn	<code>&amp;apos;</code>	'

Dấu cách	&nbsp;	(Dấu cách)
Mũi tên trỏ trái	&rarr;	<-
Mũi tên trỏ phải	&larr;	->
Ký hiệu bản quyền	&copy;	©
Ký hiệu thương hiệu	&reg;	®

### 2.3.3. Hiển thị đặc biệt

Trong HTML, ngoài việc trình bày văn bản đơn thuần, bạn có thể định dạng nội dung theo nhiều cách khác nhau để làm cho trang Web trở nên rõ ràng, dễ đọc và hấp dẫn hơn. Dưới đây là một số kiểu hiển thị đặc biệt thường dùng.

#### 2.3.3.1. Danh sách gạch đầu dòng và danh sách có thứ tự

Danh sách là một phần quan trọng trong việc trình bày và tổ chức các thông tin theo một trật tự nhất định, ngoài ra, nó còn giúp thông tin được truyền tải một cách rõ ràng và dễ đọc hơn cho người dùng. Trong HTML, hai loại danh sách được sử dụng phổ biến là:

##### a) Danh sách không thứ tự (Unordered List)

Sử dụng thẻ `<ul>` kết hợp với `<li>` để tạo danh sách các mục không cần theo một trật tự cụ thể, thường đây là loại danh sách dùng khi thứ tự các mục không quan trọng. Mỗi mục trong danh sách được đánh dấu bằng một ký hiệu cho từng mục riêng biệt (thường là dấu chấm tròn, hoặc hình vuông theo mặc định của trình duyệt).

- Thẻ `<ul>`: là thẻ cha, dùng để bao bọc toàn bộ danh sách không có thứ tự.
- Thẻ `<li>`: là thẻ con, dùng để định nghĩa từng mục (item) trong danh sách.

Ví dụ:

```
<ul>
<li>HTML cơ bản</li>
<li>CSS cơ bản</li>
<li>JavaScript cơ bản</li>
</ul>
```

Kết quả: Hiển thị ba mục được đánh dấu bằng dấu chấm tròn mặc định và chứa nội dung lần lượt theo từng mục là “HTML cơ bản”, “CSS cơ bản” và “JavaScript cơ bản” như sau:

- HTML cơ bản
- CSS cơ bản
- JavaScript cơ bản

*b) Danh sách có thứ tự (Ordered List)*

Danh sách có thứ tự (`<ol>`) dùng khi thứ tự của các mục là quan trọng và cần theo một trình tự cụ thể (ví dụ: các bước hướng dẫn, danh sách xếp hạng). Mỗi mục trong danh sách sẽ được đánh số hoặc chữ theo một trình tự (ví dụ: 1, 2, 3,... hoặc a, b, c,...). Tương tự như danh sách không thứ tự, các mục cũng được đặt trong thẻ `<li>`.

- Thẻ `<ol>`: là thẻ cha, dùng để bao bọc toàn bộ danh sách có thứ tự.
- Thẻ `<li>`: là thẻ con, dùng để định nghĩa từng mục (item) trong danh sách.

Ví dụ:

```
<ol>
<li>Chuẩn bị tài liệu</li>
<li>Viết mã HTML</li>
<li>Kiểm tra trình duyệt</li>
</ol>
```

Kết quả sẽ hiển thị ba mục được đánh số theo thứ tự như sau:

- (1). Chuẩn bị tài liệu
- (2). Viết mã HTML
- (3). Kiểm tra trình duyệt

### **2.3.3.2. Định dạng văn bản: màu, cỡ chữ và phông chữ**

Để giúp trang Web trở nên phong phú và rực rỡ hơn, bạn nên kết hợp với CSS. CSS sẽ khiến trang Web có bố cục trình bày rõ ràng, đẹp mắt và dễ dàng truyền tải nội dung đến với người đọc. Ở mức cơ bản để minh họa chức năng, bạn có thể sử

dụng cách áp dụng trực tiếp (gọi là Inline CSS). Tuy nhiên, không nên lạm dụng Inline CSS vì nó sẽ khiến code bị rối, vì vậy ta thường thực hiện bằng CSS riêng biệt.

#### a) *Màu chữ (color)*

Thuộc tính `style="color: giá_trị_màu;"` được đặt bên trong thẻ HTML để thay đổi màu sắc của văn bản bên trong thẻ đó.

– Color: là thuộc tính CSS dùng để đặt màu chữ.

– Giá\_trị\_màu: có thể là tên màu tiếng Anh (ví dụ: blue, red, green) hoặc mã màu Hex (ví dụ: #FF0000 cho màu đỏ, #00FF00 cho màu xanh lá cây).

```
<p style="color: blue;">Đây là dòng chữ màu xanh dương.</p>
```

```
<p style="color: #FF0000;">Đây là dòng chữ màu đỏ.</p>
```

Kết quả:

Đây là dòng chữ màu xanh dương.

Đây là dòng chữ màu đỏ.

#### b) *Cỡ chữ (font-size)*

Thuộc tính `style="font-size: giá_trị_kích_thước;"` được dùng để điều chỉnh kích thước của chữ.

– Font-size: là thuộc tính CSS dùng để đặt cỡ chữ.

– Giá\_trị\_kích\_thước: thường dùng đơn vị px (pixel) để định nghĩa kích thước cố định (ví dụ: 16px, 24px). Ngoài ra, còn có các đơn vị tương đối như em hoặc % (ví dụ: 1.2em nghĩa là 1.2 lần kích thước font của thẻ cha).

```
<p style="font-size: 24px;">Chữ có cỡ lớn 24px.</p>
```

Kết quả:

Chữ có cỡ lớn 24px.

#### c) *Phông chữ (font-family)*

Thuộc tính `style="font-family: tên_phông_chữ;"` dùng để thay đổi kiểu phông chữ.

– Font-Family: là thuộc tính CSS dùng để chọn phông chữ.

– Tên\_phông\_chữ: Bạn có thể liệt kê nhiều tên phông chữ cách nhau bằng dấu phẩy. Trình duyệt sẽ cố gắng tìm và sử dụng phông chữ đầu tiên có sẵn trên máy

tính của người dùng. Nếu không tìm thấy, nó sẽ chuyển sang phông tiếp theo trong danh sách. Luôn kết thúc bằng một kiểu phông chung (generic family) như serif, sans-serif để đảm bảo có phông chữ được hiển thị.

< p style="font-family: Times New Roman, serif;">Chữ dùng phông Times New Roman.</p>

Kết quả:

Chữ dùng phông Times New Roman.

## 2.4. ẢNH VÀ LINK

### 2.4.1. Ảnh

#### 2.4.1.1. Thẻ <img> trong HTML

Trong HTML, thẻ <img> được sử dụng để chèn hình ảnh vào trang Web của bạn. Đây là một phần tử vô cùng quan trọng giúp nội dung trang trở nên sinh động, trực quan và hấp dẫn hơn, thu hút sự chú ý của người dùng.

Khi trình duyệt gặp thẻ <img>, nó sẽ gửi yêu cầu đến vị trí chứa ảnh (thường là một tệp có định dạng như .jpg, .png, .gif, .webp, v.v.) và hiển thị ảnh tại vị trí đó trên trang Web.

Ví dụ:

Bạn có thể chèn một hình ảnh vào giữa một đoạn văn bản như sau:

< p>Mùa hè này, bạn hãy thử làm bánh pizza < img src="pizza.png" alt="Bánh pizza" > trên bếp nướng.</p>

Kết quả:

Mùa hè này, bạn hãy thử làm bánh pizza trên bếp nướng.



#### **2.4.1.2. Các thuộc tính bắt buộc**

- src: Đây là thuộc tính quan trọng nhất, chỉ định đường dẫn đến tệp ảnh. Đường dẫn này có thể là URL tuyệt đối (địa chỉ đầy đủ trên Internet) hoặc URL tương đối (đường dẫn so với vị trí tệp HTML hiện tại).
- alt: Thuộc tính alt dùng để hiển thị dòng chữ thay thế nếu hình ảnh không hiển thị được (ví dụ: lỗi đường dẫn, kết nối mạng yếu, v.v.). Nó cũng giúp máy đọc màn hình đọc nội dung ảnh cho người khiếm thị.

#### **2.4.1.3. Các loại đường dẫn ảnh**

Tương tự như liên kết, đường dẫn ảnh trong thuộc tính src có thể là:

- Ảnh trong cùng thư mục:

```

```

- Ảnh trong thư mục con:

```

```

- Ảnh từ website khác (URL tuyệt đối):

```

```

*Cảnh báo:* Hạn chế dùng ảnh từ trang Web khác vì:

- Không ổn định: Ảnh có thể bị xóa, di chuyển hoặc thay đổi bất cứ lúc nào trên trang Web gốc, khiến ảnh trên trang của bạn bị lỗi.
- Gây chậm tải trang: Tốc độ tải ảnh phụ thuộc vào máy chủ của trang Web khác. Nếu máy chủ đó chậm hoặc gặp sự cố, trang của bạn cũng sẽ bị ảnh hưởng.
- Vi phạm bản quyền: Sử dụng ảnh của người khác mà không được phép có thể dẫn đến vi phạm bản quyền và các rắc rối pháp lý.

### **2.4.2. Link**

Khi tạo một trang Web, bạn sẽ thường cần tạo liên kết để chuyển đến trang khác hoặc tài liệu liên quan, như bài viết khác, hình ảnh, video, v.v.. Các liên kết này có thể nằm trong cùng trang Web hoặc từ một trang Web khác.

Trong chương này, chúng ta sẽ tìm hiểu cách viết mã HTML để tạo liên kết:

- Liên kết đến trang web khác;
- Liên kết nội bộ;
- Liên kết các tài liệu.

Tất cả những điều này đều được thực hiện thông qua một thẻ HTML duy nhất: thẻ liên kết –  (anchor).

#### **2.4.2.1. Cú pháp thẻ – tạo liên kết**

- Mục đích:

Thẻ  dùng để tạo các liên kết siêu văn bản (hyperlink) – kết nối giữa các trang Web hoặc các phần khác nhau trong cùng một trang.

- Cú pháp cơ bản:

`<a href="địa_chỉ_url">Nội dung liên kết</a>`

+ href (viết tắt của hypertext reference) là thuộc tính bắt buộc, chứa địa chỉ trang đích (URL) mà bạn muốn liên kết tới.

+ Phần nội dung đặt giữa hai thẻ  và  sẽ hiển thị như một liên kết có thể nhấp.

Ví dụ:

`<a href="https://www.thaydatdeptra.com">Truy cập trang thaydatdeptra</a>`

Dòng này tạo một liên kết đến website thaydatdeptra. Khi người dùng nhấp vào dòng chữ, trình duyệt sẽ mở trang Web đó.

#### **2.4.2.2. Liên kết đến các trang Web trên Internet**

Trong quá trình xây dựng một trang Web, bạn sẽ thường xuyên cung cấp các liên kết đến các trang Web khác trên Internet, ví dụ như trang tin tức, công thức nấu ăn, bài viết tham khảo hoặc các tài nguyên hữu ích khác. Những liên kết này được gọi là liên kết ngoài, vì trang đích nằm ngoài phạm vi website của bạn.

*Cách tạo liên kết ngoài:*

Để tạo một liên kết ngoài, bạn cần sử dụng địa chỉ URL tuyệt đối của trang đích. Địa chỉ URL tuyệt đối là địa chỉ đầy đủ của trang Web, bao gồm:

- Giao thức: thường là `http://` hoặc `https://`. `https://` là giao thức bảo mật hơn và được sử dụng phổ biến hiện nay.

- Tên miền của trang Web: ví dụ: www.google.com, vietnamnet.vn, github.com.
- Đường dẫn cụ thể (nếu có): là phần chỉ định đường dẫn đến một trang hoặc tài nguyên cụ thể trong tên miền đó, ví dụ: /tin-tuc/giao-duc.

Ví dụ:

Để tạo liên kết đến một trang Web cụ thể như http://www.thaydatdeptrai.com, bạn sẽ viết:

```
<a href="http://www.thaydatdeptrai.com">Trang Web của Thầy Đạt Đẹp Trai</a>
```

Khi người dùng nhấp vào dòng chữ “Trang Web của Thầy Đạt Đẹp Trai”, trình duyệt sẽ mở trang Web của kênh thaydatdeptraи trong tab hiện tại.

#### **2.4.2.3. Liên kết nội bộ trong website**

Liên kết nội bộ là những liên kết giữa các trang trong cùng một website – chẳng hạn như từ trang chủ đến trang “Giới thiệu”, từ trang sản phẩm đến trang liên hệ, v.v.. Việc sử dụng liên kết nội bộ giúp người dùng dễ dàng di chuyển giữa các phần của trang Web và là một kỹ thuật rất quan trọng trong thiết kế website.

Liên kết trong cùng thư mục, thư mục con và thư mục cha.

##### *a) Liên kết đến tệp tin trong cùng thư mục*

Giả sử bạn có hai tệp tin HTML nằm trong cùng một thư mục, bạn có thể liên kết giữa chúng như sau:



Trong tệp index.html, để liên kết đến trang about.html, bạn sẽ viết:

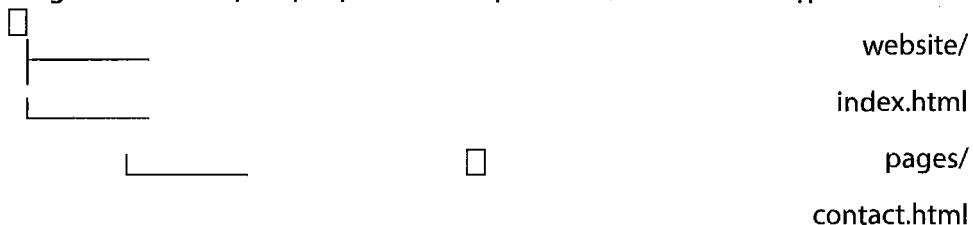
```
<a href="about.html">Giới thiệu</a>
```

Khi người dùng nhấp vào liên kết “Giới thiệu”, trình duyệt sẽ mở tệp about.html nằm cùng cấp với index.html.

##### *b) Liên kết đến tệp tin trong thư mục con*

Để tổ chức code gọn gàng hơn, bạn nên đặt các tệp liên quan vào trong các thư mục con. Để liên kết đến một tệp nằm trong thư mục con, bạn cần chỉ rõ

đường dẫn từ thư mục hiện tại đến thư mục con đó, sau đó là tên tệp.



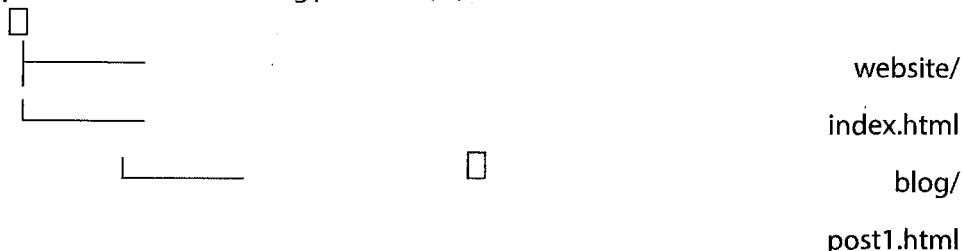
Trong tệp index.html, để liên kết đến trang contact.html nằm trong thư mục pages, bạn sẽ viết:

```
<a href="pages/contact.html">Liên hệ</a>
```

Ở đây, pages/ cho trình duyệt biết rằng tệp contact.html nằm bên trong thư mục pages. Dấu gạch chéo / được dùng để phân tách tên thư mục và tên tệp.

#### c) Liên kết đến tập tin trong thư mục cha

Để liên kết trở về một tệp ở thư mục cấp trên (thư mục cha), bạn sử dụng ký hiệu hai dấu chấm và dấu gạch chéo (../).



Trong tệp post1.html (nằm trong thư mục blog), để tạo liên kết trở về trang index.html (nằm ở thư mục website – thư mục cha), bạn sẽ viết:

```
<a href="../index.html">Trang chủ</a>
```

- Dấu .. nghĩa là quay lên thư mục cha.
- Dấu / dùng để phân tách thư mục.

#### 2.4.2.4. Liên kết đến một vị trí cụ thể trong cùng trang

Nếu bạn muốn tạo một liên kết mà khi nhấp vào đó, người dùng sẽ được đưa đến một vị trí cụ thể ngay trên cùng một trang Web, bạn có thể tạo mục lục ở đầu trang, khi người dùng nhấp vào một mục, họ sẽ được “nhảy” xuống đúng phần nội dung tương ứng mà không cần cuộn chuột. Hoặc từ cuối trang, bạn có thể tạo một liên kết để quay ngược về đầu trang một cách nhanh chóng.

Cách làm như sau:

- Bước 1: Gắn id cho phần tử đích.

Ví dụ:

Giả sử bạn có một tiêu đề “Liên hệ với chúng tôi” và muốn tạo liên kết nhảy đến đó:

```
<h2 id="contact">Liên hệ với chúng tôi</h2>
```

Trong ví dụ này, contact là giá trị của thuộc tính id. Bạn có thể đặt bất kỳ tên nào cho id miễn là nó duy nhất trên toàn bộ trang HTML của bạn.

- Bước 2: Tạo liên kết đến id đó.

Sau khi đã gắn id cho phần tử đích, bạn sẽ tạo một thẻ liên kết `<a>` và sử dụng dấu thăng (#) cùng với tên id bạn đã đặt trong thuộc tính href.

Ví dụ:

Để tạo liên kết đến phần “Liên hệ với chúng tôi” ở trên:

```
<a href="#contact">Đi đến phần Liên hệ</a>
```

Khi người dùng nhấp vào liên kết “Đi đến phần Liên hệ”, trình duyệt sẽ tự động cuộn trang xuống đến vị trí của phần tử có id=”contact”.

Lưu ý:

ID phải là duy nhất: Mỗi giá trị id trên một trang HTML phải là duy nhất. Bạn không thể có hai phần tử khác nhau cùng có chung một id. Nếu có, liên kết có thể hoạt động không như mong đợi hoặc gây ra lỗi.

## 2.5. TABLE (BẢNG)

### 2.5.1. Cách sử dụng bảng

Bảng (table) trong HTML là một cấu trúc cho phép trình bày dữ liệu dạng lưới với các hàng và cột trên trang Web. Cũng giống như bảng trong các ứng dụng soạn thảo (Word, Excel), bảng HTML giúp sắp xếp thông tin thành các ô, giúp người đọc dễ dàng so sánh và đối chiếu dữ liệu theo hàng ngang và cột dọc.

Mục đích sử dụng bảng: Bảng được dùng để hiển thị những dữ liệu có cấu trúc bảng biểu, ví dụ: bảng điểm của học sinh theo các môn, thời khóa biểu (lịch các

tiết học trong tuần), danh sách thông tin học sinh trong lớp, bảng so sánh đặc tính sản phẩm, thống kê số liệu, v.v.. Nhờ cách trình bày hàng – cột, bảng giúp thông tin được trình bày rõ ràng, dễ hiểu và ngắn gọn cho người xem.

### 2.5.2. Cấu trúc cơ bản của bảng

Một bảng HTML được định nghĩa bởi cặp thẻ `<table>...</table>`. Bên trong thẻ `<table>` là nội dung của bảng được tổ chức thành các hàng và các ô:

- Thẻ `<tr>` (table row): Đại diện cho một hàng của bảng. Mỗi hàng là một dòng ngang gồm nhiều ô dữ liệu.
- Thẻ `<td>` (table data): Đại diện cho một ô dữ liệu (cell) trong bảng, chứa nội dung (văn bản, số, hình ảnh, v.v.). Các thẻ `<td>` phải được đặt bên trong một hàng `<tr>`. Mỗi thẻ `<td>...</td>` tạo ra một ô và các ô trong cùng một hàng sẽ nằm cạnh nhau theo chiều ngang.
- Thẻ `<th>` (table header): Đại diện cho một ô tiêu đề trong bảng, cũng đặt bên trong `<tr>`. Thẻ `<th>` có chức năng tương tự `<td>` nhưng dùng cho ô ở hàng tiêu đề (thường là hàng đầu bảng hoặc đầu các nhóm dữ liệu). Nội dung trong `<th>` mặc định sẽ được trình duyệt căn giữa và in đậm để làm nổi bật là tiêu đề.
- Thẻ `<caption>`: Đại diện cho chú thích của bảng, thường được dùng để mô tả ngắn gọn nội dung hoặc mục đích của bảng. Thẻ `<caption>` (nếu có) phải được đặt ngay sau thẻ `<table>` mở và theo mặc định, chú thích sẽ được hiển thị căn giữa phía trên bảng.

Các thẻ nâng cao hơn như `<thead>`, `<tbody>`, `<tfoot>`, `<colgroup>`, `<col>` v.v. dùng để nhóm các phần của bảng hoặc định kiểu cho cột sẽ không đề cập trong bài học này.

Dưới đây là mã HTML cho một bảng cơ bản hiển thị thông tin dinh dưỡng:

```
<table> <!-- Tạo bảng dữ liệu -->  
<tr> <!-- Dòng 1: Dữ liệu cho món thứ nhất -->  
<td>Phở bò</td> <!-- Cột 1: Tên món ăn -->  
<td>500</td> <!-- Cột 2: Lượng calo của món ăn -->  
<td>20</td> <!-- Cột 3: Lượng chất béo (gram) -->
```

```

</tr>

<tr> <!-- Dòng 2: Dữ liệu cho món thứ hai -->
<td>Cơm rang</td> <!-- Cột 1: Tên món ăn -->
<td>400</td> <!-- Cột 2: Lượng calo của món ăn -->
<td>26</td> <!-- Cột 3: Lượng chất béo (gram) -->
</tr>

</table>

```

Nếu chỉ sử dụng cấu trúc bảng cơ bản, trang Web sẽ hiển thị như sau:

Phở bò	500	20
Cơm rang	400	26

- Một bảng với hai hàng và ba cột.
- Hai hàng chứa dữ liệu (“Phở bò”, “500”, “20” và “Cơm rang”, “400”, “26”).

### 2.5.3. Thuộc tính cơ bản của bảng

HTML cung cấp một số thuộc tính giúp định dạng trực tiếp bảng và các ô bảng. Dưới đây là các thuộc tính bảng cơ bản thường dùng:

– Border: Thuộc tính thêm đường viền cho bảng và ô bảng. Giá trị thường là một số nguyên biểu thị độ dày đường viền tính bằng pixel. Ví dụ: border="1" trên thẻ `<table>` sẽ vẽ đường viền ô dày 1 pixel bao quanh các ô của bảng. Nếu bỏ thuộc tính border hoặc đặt border="0" thì bảng sẽ không hiển thị đường viền (mặc định các bảng HTML không có viền hiển thị cho ô).

– Width: Thuộc tính quy định độ rộng cho bảng hoặc ô. Giá trị có thể là số pixel cố định (ví dụ: width="500" nghĩa là 500px) hoặc phần trăm (%) so với độ rộng vùng chứa. Ví dụ: `<table width="100%">` tạo bảng giãn hết chiều ngang vùng chứa, còn `<td width="50%">` tạo ô rộng bằng một nửa hàng chứa nó. Thuộc tính width có thể áp dụng trên `<table>`, `<td>` hoặc `<th>`. Nếu không chỉ định, độ rộng bảng sẽ tự động co giãn theo nội dung các ô.

- Align: Thuộc tính dùng để căn lề (align). Cách dùng:

+ Với thẻ `<table>`, thuộc tính align có thể nhận giá trị “left”, “center”, “right” để căn cảng sang trái, giữa hoặc phải trang Web. Thông thường, bảng mặc định căn trái. Muốn bảng nằm giữa trang, ta thêm align=“center” vào thẻ `<table>`.

+ Với thẻ ô `<td>` hoặc `<th>`, thuộc tính align dùng để căn chỉnh nội dung bên trong ô theo chiều ngang. Các giá trị thường dùng: “left” (căn trái nội dung ô), “center” (căn giữa), “right” (căn phải), “justify” (căn đều hai bên – thường ít dùng cho dữ liệu bảng). Mặc định, nội dung trong `<td>` căn trái, còn nội dung trong `<th>` thường được trình duyệt căn giữa và in đậm như đã đề cập.

– Colspan (column span): Thuộc tính gộp cột, cho phép một ô kéo rộng qua nhiều cột. Gắn thuộc tính này cho thẻ `<td>` hoặc `<th>` và đặt giá trị là số cột muốn ô chiếm. Ví dụ: `colspan=“2”` nghĩa là ô này sẽ trải rộng bằng độ rộng hai cột thường. Thuộc tính `colspan` giúp hợp nhất nhiều ô liền kề theo chiều ngang thành một ô lớn.

– Rowspan (row span): Thuộc tính gộp hàng, cho phép một ô kéo dài xuống nhiều hàng. Cũng gắn trong `<td>` hoặc `<th>` với giá trị là số hàng muốn ô chiếm. Ví dụ: `rowspan=“3”` nghĩa là ô sẽ kéo dài xuống ba hàng dọc. Thuộc tính `rowspan` hợp nhất các ô theo chiều dọc (trên dưới) thành một ô cao hơn bình thường.

Lưu ý: Giá trị của `colspan` và `rowspan` phải là số nguyên  $\geq 1$ , tương ứng với số ô muốn gộp. Khi gộp ô, cần bảo đảm các hàng/cột khác được bổ sung hoặc sắp xếp ô hợp lý để bảng không bị lệch cấu trúc. Nếu khai báo giá trị span không phù hợp (quá lớn so với số ô thực có hoặc bố trí không đúng hàng, cột tương ứng), bảng có thể bị lỗi hiển thị (các ô chồng chéo nhau).

Hiện nay, trong HTML5, các thuộc tính định dạng như `border`, `align`, `bgcolor`, v.v. được xem là không còn được khuyến khích sử dụng (deprecated), nên sử dụng CSS để định dạng thay thế. Tuy nhiên, các thuộc tính này vẫn còn hiệu lực trên nhiều trình duyệt và rất hữu ích để tạo bảng nhanh, đơn giản. Chúng ta có thể dùng kết hợp định nghĩa cấu trúc bảng bằng HTML, sau đó dùng CSS để trình bày cho đẹp. Phần sau sẽ hướng dẫn cách trang trí bảng bằng CSS.

## 2.5.4. Tạo bảng HTML cơ bản với ví dụ minh họa

### 2.5.4.1. Các bước để tạo một bảng HTML

1. Tạo khung bảng: Dùng cặp thẻ `<table>...</table>` để bao lấp toàn bộ nội dung bảng. Có thể thêm thuộc tính `border=“1”` vào thẻ `<table>` để bảng có đường viền dễ quan sát kết quả.

2. Tạo các hàng: Bên trong `<table>`, thêm các cặp thẻ `<tr>...</tr>` cho mỗi hàng của bảng. Mỗi hàng sẽ chứa các ô bên trong.

3. Tạo các ô: Bên trong mỗi cặp `<tr>`, sử dụng thẻ `<td>...</td>` cho mỗi ô dữ liệu của hàng đó. Điền nội dung phù hợp vào giữa cặp thẻ `<td>`. Đảm bảo số lượng thẻ `<td>` trong các hàng tương ứng với số cột cần có.

4. Hàng tiêu đề (nếu cần): Nếu bảng có hàng tiêu đề, sử dụng thẻ `<th>...</th>` thay cho `<td>` ở những ô thuộc hàng tiêu đề (thường là hàng đầu tiên). Nội dung trong `<th>` thường là tên cột hoặc mô tả ý nghĩa dữ liệu.

5. Chú thích bảng (tùy chọn): Nếu muốn thêm tiêu đề hoặc chú thích chung cho cả bảng, sử dụng thẻ `<caption>...</caption>` ngay sau thẻ `<table>` mở. Nội dung chú thích sẽ hiển thị phía trên bảng, mặc định căn giữa.

#### **2.5.4.2. Một số ví dụ cụ thể**

Ví dụ 1: Tạo bảng hai hàng ba cột đơn giản.

Giả sử cần tạo một bảng gồm hai hàng và ba cột, chứa các dữ liệu văn bản đơn giản. Ta thực hiện như sau:

html

```
<table border="1" width="50%">  
  <tr>  
    <td>Ô 1</td>  
    <td>Ô 2</td>  
    <td>Ô 3</td>  
  </tr>  
  <tr>  
    <td>Ô 4</td>  
    <td>Ô 5</td>  
    <td>Ô 6</td>  
  </tr>  
</table>
```

Ô 1	Ô 2	Ô 3
Ô 4	Ô 5	Ô 6

Kết quả: Bảng có hai hàng, ba cột với nội dung “Ô 1” đến “Ô 6” trong các ô tương ứng. Thuộc tính border=“1” giúp hiện đường viền nên mỗi ô được ngăn cách rõ ràng. Thuộc tính width=“50%” khiến bảng có độ rộng bằng một nửa chiều ngang vùng chứa (ví dụ trang Web), giúp bảng không quá rộng. Nếu bỏ width, bảng sẽ co theo độ rộng nội dung các ô.

Trong ví dụ trên, tất cả các ô đều dùng thẻ <td> nên nội dung là dữ liệu thông thường. Bảng này chưa có hàng tiêu đề.

Ví dụ 2: Bảng có hàng tiêu đề sử dụng thẻ <th>.

Xét trường hợp ta muốn tạo bảng danh sách liên hệ gồm ba cột: Họ, Tên, Số điện thoại. Hàng đầu tiên sẽ là tiêu đề của các cột, các hàng tiếp theo là dữ liệu. Ta có thể viết mã như sau:

html

```
<table border="1" width="60%">

<tr>
<th>Họ</th>
<th>Tên</th>
<th>Số điện thoại</th>
</tr>

<tr>
<td>Nguyễn</td>
<td>Phúc Nguyên</td>
<td>0123456789</td>
</tr>

<tr>
<td>Trần</td>
```

```

<td>Đức Minh</td>
<td>0987654321</td>
</tr>
</table>

```

Họ	Tên	Số điện thoại
Nguyễn	Phúc Nguyên	0123456789
Trần	Đức Minh	0987654321

Ở ví dụ này, hàng thứ nhất sử dụng thẻ **<th>** cho ba ô tiêu đề: Họ, Tên, Số điện thoại. Trình duyệt mặc định sẽ hiển thị các ô **<th>** này với chữ đậm và căn giữa, phân biệt với dữ liệu thường. Hai hàng sau sử dụng **<td>** cho dữ liệu là hai người: Nguyễn Phúc Nguyên và Trần Đức Minh cùng số điện thoại tương ứng.

Kết quả: Bảng được hiển thị với hàng đầu là tiêu đề cột (Họ, Tên, Số điện thoại) có định dạng nổi bật, các hàng dưới là dữ liệu cụ thể. Nhờ sử dụng **<th>**, hàng tiêu đề dễ nhìn hơn (chữ in đậm, căn giữa), giúp người xem nhận biết đó là tiêu đề của các cột.

Ví dụ 3: Bảng có ô gộp cột (colspan) và gộp hàng (rowspan).

HTML cho phép gộp các ô lại với nhau như đã giải thích ở phần thuộc tính. Sau đây là hai ví dụ nhỏ, một về gộp cột và một về gộp hàng.

– Gộp cột: Tạo bảng điểm đơn giản với cột “Họ và Tên” kéo dài qua hai cột con. Ta có thể thiết kế hàng tiêu đề gồm một ô “Họ và Tên” gộp hai cột và một ô “Số điện thoại” bình thường. Các hàng dưới sẽ chứa Họ, Tên và Tuổi.

html

```

<table border="1" width="60%">
<tr>
<th colspan="2">Họ và tên</th>
<th>Số điện thoại</th>
</tr>

```

```

<tr>
<td>Nguyễn</td>
<td>Phúc Nguyên</td>
<td>0123456789</td>
</tr>
<tr>
<td>Trần</td>
<td>Đức Minh</td>
<td>0987654321</td>
</tr>

```

Họ và tên		Số điện thoại
Nguyễn	Phúc Nguyên	0123456789
Trần	Đức Minh	0987654321

</table>

Ở đây, thuộc tính colspan="2" đặt trong ô <th> "Họ và Tên" giúp ô này trải rộng thành hai cột. Do đó, hàng đầu tiên chỉ có hai ô <th> nhưng tạo thành ba cột thị giác: ô "Họ và tên" chiếm hai cột, ô "Số điện thoại" chiếm cột thứ ba. Các hàng thứ hai và ba có 3 ô <td> mỗi hàng, tương ứng với Họ, Tên và Số điện thoại.

Kết quả: Bảng hiển thị ba cột: Họ và Tên (gộp hai cột con), Số điện thoại. (Lưu ý: Giá trị colspan="2" báo rằng ô Họ và Tên chiếm hai cột liên tiếp).

## 2.6. CĂN CHỈNH TRANG WEB

### 2.6.1. Căn chỉnh tiêu đề và đoạn văn

- HTML cung cấp các thẻ tiêu đề từ <h1> (to nhất) đến <h6> (nhỏ nhất).
- Đoạn văn được tạo bằng thẻ <p>.
- Các thẻ này có thể được căn chỉnh theo chiều ngang bằng cách đặt trong thẻ <div> có thuộc tính align="..."

Ví dụ:

Code:

```
<body>
```

```
  <div align="center">
```

```
    <h1>Chào mừng đến với website của tôi</h1>
```

```
    <p>Đoạn này được căn giữa. Việc sử dụng align giúp bạn định hướng cho nội dung rõ ràng hơn.</p>
```

```
  </div>
```

```
  <div align="left">
```

```
    <h2>Tiêu đề nhỏ căn trái</h2>
```

```
    <p>HTML mặc định căn trái cho nội dung. Tuy nhiên, bạn vẫn nên rõ ràng hoặc đặt trong div cho linh hoạt.</p>
```

```
  </div>
```

```
  <div align="right">
```

```
    <p>Và đây là đoạn văn căn phải, dùng cho chữ ký, số liệu hoặc một số giao diện đặc thù.</p>
```

```
  </div>
```

```
  <div align="justify">
```

```
    <p>Căn đều hai bên (justify) giúp đoạn văn trông giống sách vở hơn. Tuy nhiên, khi không có CSS, việc căn đều có thể không hoạt động hoàn hảo trên một số trình duyệt.</p>
```

```
  </div>
```

Hiển thị:



## Chào mừng đến với website của tôi

Đoạn này được căn giữa. Việc sử dụng align giúp bạn định hướng cho nội dung rõ ràng hơn.

### Tiêu đề nhỏ căn trái

HTML mặc định căn trái cho nội dung. Tuy nhiên, bạn vẫn nên rõ ràng hoặc đặt trong div cho linh hoạt.

Và đây là đoạn văn căn phải, dùng cho chữ ký, số liệu hoặc một số giao diện đặc thù.

Căn đều hai bên (justify) giúp đoạn văn trông giống sách vở hơn. Tuy nhiên, khi không có CSS, việc căn đều có thể không hoạt động hoàn hảo trên một số trình duyệt.

## 2.6.2. Căn giữa hình ảnh

Ví dụ:

```
<body>
<div align="center">

<p>Hình được căn giữa bằng div.</p>
</div>
</body>
```

Hiển thị:



## 2.6.3. Tạo bảng và căn chỉnh trong bảng

- HTML dùng `<table>` tạo bảng.
- Mỗi hàng dùng `<tr>`, ô tiêu đề dùng `<th>`, ô dữ liệu dùng `<td>`.
- Đặt `align="..."` vào để căn chỉnh nội dung.

Ví dụ:

```
<body>
<h2 align="center">Danh sách lớp</h2>
<table border="1" width="70%" align="center">
```

```

<tr>
<th align="center">Tên</th>
<th align="center">Tuổi</th>
<th align="center">Lớp</th>
</tr>
<tr>
<td align="left">Diệu Anh</td>
<td align="center">17</td>
<td align="right">11 Tin</td>
</tr>
<tr>
<td align="left">Ngọc Minh</td>
<td align="center">17</td>
<td align="right">11 Tin</td>
</tr>
</table>
</body>

```

Hiển thị:

Tên	Tuổi	Lớp
Điệu Anh	17	11 Tin
Ngọc Minh	17	11 Tin

## 2.6.4. Đổi màu nền và chữ

- Thuộc tính bgcolor và text được dùng trong thẻ <body>.

Code:

```

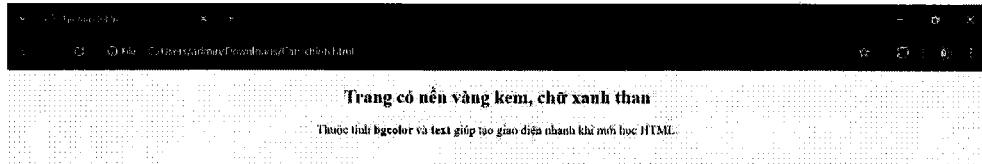
<body bgcolor="#fff8dc" text="#003366">
<h2 align="center">Trang có nền vàng kem, chữ xanh than</h2>

```

```
<p align="center">Thuộc tính <b>bgcolor</b> và <b>text</b> giúp tạo giao diện nhanh khi mới học HTML.</p>
```

```
</body>
```

Hiển thị:



## 2.7. MÃ NHÚNG

### 2.7.1. Mã nhúng là gì?

- Mã nhúng (embed code) là đoạn mã HTML, thường sử dụng thẻ `<iframe>`, `<script>` hoặc `<object>`, cho phép hiển thị nội dung từ nguồn bên ngoài (như YouTube, Facebook, Google Maps, v.v.) trên trang Web của bạn.
- Nội dung được hiển thị mà không cần tải file gốc về máy chủ của bạn.
- Mã nhúng giúp tích hợp nội dung động, có thể cập nhật theo thời gian thực.
- Việc sử dụng mã nhúng giúp giảm tải cho máy chủ lưu trữ của bạn.
- Các nền tảng thường cung cấp mã nhúng sẵn qua nút “Embed” hoặc “Chia sẻ”.
- Người dùng chỉ cần sao chép và dán mã nhúng vào trang HTML, blog hoặc ứng dụng Web.

### 2.7.2. Mã nhúng hoạt động như thế nào?

- Bước 1: Khởi tạo kết nối đến nguồn bên thứ ba.  
Trình duyệt gửi yêu cầu đến URL trong thẻ nhúng (ví dụ: `src="https://www.youtube.com/embed/..."`) để lấy tài nguyên từ máy chủ gốc.
- Bước 2: Tải tài nguyên từ máy chủ gốc.  
Nội dung như video, bài đăng, bản đồ hoặc widget sẽ được tải trực tiếp từ máy chủ của nhà cung cấp (YouTube, Facebook, Map, v.v.).

- Bước 3: Hiển thị nội dung trong vùng nhúng (viewport).

Dữ liệu được hiển thị trong một khung riêng biệt, thường là thẻ `<iframe>`, giúp nội dung hiển thị đúng định dạng mà không ảnh hưởng đến trang Web chính.

### 2.7.3. Nhúng video Youtube

- Src: Địa chỉ nguồn của nội dung (ở đây là link video YouTube).
- Allowfullscreen: Cho phép xem toàn màn hình.
- Width và height: Xác định kích thước khung hiển thị.

Ví dụ:

```
<body>
```

```
<h2>Nhúng video YouTube</h2>
```

```
<p>Dưới đây là một video được nhúng từ YouTube:</p>
```

```
<iframe width="560" height="315" src="link_youtube" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" referrerPolicy="strict-origin-when-cross-origin" allowfullscreen>
```

```
</iframe>
```

```
<p><i>Video trên không cần tải về, chỉ cần dán mã nhúng.</i></p>
```

```
</body>
```

Hiển thị:



Video trên không cần tải về, chỉ cần dán mã nhúng.

#### **2.7.4. Nhúng Google Maps**

**Code:**

<body>

**<h2 align="center">Những bản đồ Google Maps</h2>**

<p align="center">Ví dụ dưới đây chèn bản đồ của Trường THPT chuyên Hà Nội – Amsterdam:</p>

<div align="center">

```
<iframe src="link_google_map" width="600" height="450" style="border:0;" allowfullscreen="" loading="lazy" referrerpolicy="no-referrer-when-downgrade">
```

</iframe>

```
</div>
```

*Bản đồ này được tải trực tiếp từ Google Maps.*

</body>



## **CHƯƠNG III**

# **CSS**

### **3.1. GIỚI THIỆU VỀ CSS VÀ CÁC PHẦN TỬ HTML**

Cascading Style Sheets (CSS) là công cụ để tạo kiểu cho các trang Web. Trong khi HTML cung cấp cấu trúc và nội dung, CSS quy định cách nội dung đó trông như thế nào và được bố trí trên màn hình ra sao. Cốt lõi của mối quan hệ này là các phần tử HTML: Các khối xây dựng cơ bản của bất kỳ trang Web nào. Từ đoạn văn đến tiêu đề, hình ảnh đến liên kết điều hướng, mỗi phần nội dung đều nằm trong một phần tử HTML.

Hiểu được cách CSS tương tác với các phần tử là điều thiết yếu để tạo ra các trang Web có giao diện đẹp mắt và cấu trúc tốt. Chương này sẽ đi sâu vào các khái niệm cốt lõi về việc tạo kiểu trực tiếp cho các phần tử HTML, cách chúng ta nhắm mục tiêu cho chúng, hành vi mặc định của chúng và giới thiệu các phương pháp hay nhất để áp dụng kiểu cho các thẻ phổ biến như `<div>`, `<p>`, `<h1>` và nhiều thẻ khác.

### **3.2. VAI TRÒ CỦA CÁC THẺ HTML**

Trước khi đi sâu vào việc tạo kiểu, hãy cùng xem lại về các thẻ HTML. Một thẻ HTML được dùng để đánh dấu điểm bắt đầu và kết thúc của một phần tử HTML. Ví dụ: `<p>` là thẻ mở của một đoạn văn và `</p>` là thẻ đóng của nó. Nội dung giữa các thẻ này tạo thành phần tử đoạn văn.

Dưới đây là một số thẻ phổ biến và mục đích của chúng:

- `<div>` (Division): Một vùng chứa chung cho nội dung. Nó thường được dùng để nhóm các phần tử khác lại với nhau cho mục đích tạo kiểu. Theo mặc định, một `<div>` là một phần tử cấp khối (block-level), có nghĩa là nó bắt đầu trên một dòng mới và chiếm toàn bộ chiều rộng có sẵn.

- `<p>` (Paragraph): Định nghĩa một đoạn văn bản. Giống như `<div>`, `<p>` là một phần tử cấp khối.

– <h1> đến <h6> (Headings): Định nghĩa các tiêu đề HTML. <h1> là tiêu đề quan trọng nhất và <h6> là tiêu đề ít quan trọng nhất. Đây cũng là các phần tử cấp khống.

– <span> (Span): Một vùng chứa nội dòng (inline) cho nội dung. Nó được dùng để nhóm các phần tử nội dòng cho mục đích tạo kiểu. Không giống như <div>, <span> không bắt đầu trên một dòng mới và chỉ chiếm chiều rộng bằng nội dung của nó.

– <a> (Anchor): Định nghĩa một siêu liên kết, được dùng để liên kết từ trang này sang trang khác. Theo mặc định, <a> là một phần tử nội dòng.

– <img> (Image): Nhúng một hình ảnh vào tài liệu. Nó là một phần tử nội dòng - khống (inline-block) tự đóng.

– <ul> (Unordered List) và <ol> (Ordered List): Định nghĩa các vùng chứa danh sách.

– <li> (List Item): Định nghĩa một mục trong danh sách.

– <header>, <nav>, <main>, <article>, <section>, <footer>: Đây là các thẻ HTML5a cung cấp ý nghĩa cho cấu trúc của nội dung. Chúng hoạt động tương tự như <div> theo mặc định (vùng chứa cấp khống), nhưng mang giá trị ngữ nghĩa cao hơn, có lợi cho khả năng truy cập và SEO.

### 3.3. NHẮM MỤC TIÊU CÁC PHẦN TỬ HTML VỚI CSS

Cách cơ bản nhất để áp dụng CSS cho các phần tử HTML là thông qua bộ chọn kiểu (type selectors), còn được gọi là bộ chọn phần tử (element selectors). Phương pháp này cho phép chọn tất cả các phiên bản của một thẻ HTML cụ thể và áp dụng kiểu cho chúng.

#### 3.3.1. Bộ chọn kiểu

Một bộ chọn kiểu chỉ đơn giản sử dụng tên của thẻ HTML mà bạn muốn tạo kiểu.

/\* Giải thích: Chọn tất cả các phần tử <h1> \*/

```
h1 {  
    color: darkblue;  
    font-size: 2.5em;  
    text-align: center;  
}
```

```
/*Giải thích: Chọn tất cả các phần tử <p> */  
p {  
    font-family: Arial, sans-serif;  
    line-height: 1.6;  
}  
  
/*Giải thích: Chọn tất cả các phần tử <div> */  
div {  
    border: 1px solid #ccc;  
    background-color: #f9f9f9;  
}
```

Cách tiếp cận này giúp đặt các kiểu chung cho các phần tử phổ biến. Ví dụ, bạn có thể khiến tất cả các đoạn văn trên trang Web của mình có một phông chữ nhất định hoặc tất cả các tiêu đề được căn giữa.

### 3.3.2. Nhóm bộ chọn

Bạn có thể áp dụng cùng một bộ kiểu cho nhiều phần tử bằng cách nhóm các bộ chọn của chúng chỉ cách nhau bởi dấu phẩy.

```
/*Giải thích: Áp dụng các kiểu này cho tất cả các phần tử <h1>, <h2> và <h3> */  
h1, h2, h3 {  
    color: #333;  
    font-family: 'Georgia', serif;  
    margin-bottom: 0.5em; /* Thuộc tính này sẽ được thảo luận chi tiết ở nơi khác */  
}  
  
/*Giải thích: Áp dụng các kiểu này cho tất cả các phần tử <ul> và <ol> */  
ul, ol {  
    list-style-position: inside;  
    padding-left: 20px;  
}
```

### **3.3.3. Bộ chọn chung (\*)**

Bộ chọn chung (\*) chọn tất cả các phần tử HTML trên một trang. Mặc dù bộ chọn chung (\*) có thể hữu ích cho một số thiết lập lại chung nhưng nên sử dụng hạn chế vì nó có thể dẫn đến các vấn đề về hiệu suất và những vấn đề về việc áp dụng kiểu không mong muốn.

*/\*Giải thích: Áp dụng cho tất cả các phần tử \*/*

```
* {  
    box-sizing: border-box; /* Một thiết lập lại chung phổ biến */  
}
```

### **3.3.4. Hành vi mặc định của phần tử: khối, nội dòng và nội dòng – khối**

Mọi phần tử HTML đều có thuộc tính display mặc định quy định cách nó hoạt động trong luồng tài liệu. Hiểu các hành vi mặc định này là điều quan trọng để bố trí trang của bạn một cách chính xác.

#### **3.3.4.1. Phần tử cấp khối (Block-Level Elements)**

- Bắt đầu trên một dòng mới: Phần tử cấp khối luôn bắt đầu trên một dòng mới và thường chiếm toàn bộ chiều rộng có sẵn.
- Chiếm toàn bộ chiều rộng: Ngay cả khi nội dung của chúng hẹp. Các phần tử khối sẽ kéo dài để lấp đầy chiều rộng của vùng chứa phần tử cha của chúng.
- Có thể đặt width, height, margin, padding: Kiểm soát kích thước và khoảng cách của chúng.
- Có thể chứa các phần tử khối và nội dòng khác.

*Ví dụ:* <div>, <p>, <h1> đến <h6>, <ul>, <ol>, <li>, <form>, <header>, <footer>, <section>, <article>, <nav>.

*<!-- Ví dụ về các phần tử khối -->*

```
<div style="background-color: lightblue;">Đây là một div.</div>  
<p style="background-color: lightcoral;">Đây là một đoạn văn.</p>  
```css
```

```
/* Kiểu dáng để minh họa */  
div, p {  
    margin-bottom: 10px;  
    padding: 10px;  
    border: 1px solid blue;  
}
```

#### **3.4.4.2. Phần tử nội dòng (Inline Elements)**

- Không bắt đầu trên một dòng mới: Chúng chạy cùng với nội dung văn bản, xuất hiện trên cùng một dòng với văn bản xung quanh và các phần tử nội dòng khác.
- Chỉ chiếm chiều rộng cần thiết: Phần tử nội dòng chỉ chiếm chiều rộng bằng nội dung của chúng.
- Không thể đặt width hoặc height (một cách hiệu quả): Các cổ gắng đặt width hoặc height trên các phần tử nội dòng thường bị bỏ qua hoặc hoạt động không mong muốn.
- Margin-top và margin-bottom bị bỏ qua: Chỉ margin-left và margin-right có tác dụng. Padding-top và padding-bottom sẽ thêm khoảng trống xung quanh nội dung nhưng sẽ không ảnh hưởng đến chiều cao dòng.
- Chỉ có thể chứa dữ liệu và các phần tử nội dòng khác.

Ví dụ: <span>, <a>, <strong>, <em>, <img>, <sub>, <sup>, <label>.

<!-- Ví dụ về các phần tử nội dòng -->

```
<span style="background-color: lightgreen;">Đây là một span.</span>
```

Một số văn bản với một <a href="#" style="background-color: lightyellow;">liên kết nội dòng</a>.

```
/*Giải thích: Kiểu dáng để minh họa */
```

```
span, a {
```

```
    padding: 5px; /* Padding trên/dưới hoạt động nhưng không ảnh hưởng đến  
    chiều cao dòng */
```

```
    margin-left: 5px;
```

```
margin-right: 5px;  
border: 1px solid green;  
}
```

### **3.3.4.3. Phần tử nội dòng – khối (Inline-Block Elements)**

Các phần tử nội dòng – khối kết hợp các đặc điểm của cả phần tử khối và nội dòng:

- Không bắt đầu trên một dòng mới: Chúng chạy theo chiều ngang, tương tự như các phần tử nội dòng.
- Có thể đặt các thẻ width, height, margin, padding: Kiểm soát được kích thước và khoảng cách của chúng, giống như các phần tử khối.
- Chỉ chiếm chiều rộng cần thiết hoặc chiều rộng được chỉ định: Chúng sẽ không chiếm toàn bộ chiều rộng theo mặc định trừ khi được chỉ định.

Ví dụ: <img>, <input>, <button>, <select>. Bạn cũng có thể thay đổi bất kỳ phần tử nào thành display: inline-block;

```
<!-- Ví dụ về một phần tử nội dòng-khối -->
```

```
<button style="background-color: lightsalmon; width: 100px; height: 40px;">Bấm vào đây</button>
```

```
<span style="display: inline-block; background-color: lavender; width: 120px; height: 50px;">Nội dòng-khối tùy chỉnh</span>
```

```
```css
```

```
/*Giải thích: Kiểu dáng để minh họa */
```

```
button, span[style*="inline-block"] {  
margin: 10px;  
border: 1px solid purple;  
}
```

Hiểu các loại hiển thị này là việc rất quan trọng, vì thuộc tính display mặc định của một thẻ HTML xác định cách bạn có thể tạo kiểu hiệu quả cho kích thước, khoảng cách và vị trí của nó. Bạn có thể ghi đè thuộc tính display mặc định bằng CSS (ví dụ: display: block; trên một <span> để làm cho nó hoạt động như một <div>).

### **3.3.5. Các ví dụ thực tế**

#### **3.3.5.1. Tạo kiểu cho <h1> đến <h6> (tiêu đề)**

Các tiêu đề là nền tảng để cấu trúc nội dung và cải thiện khả năng đọc.

```
h1 {  
    font-size: 3em; /* Văn bản lớn hơn */  
    color: #2c3e50; /* Màu xám đậm */  
    text-align: center; /* Căn giữa văn bản */  
    text-transform: uppercase; /* Chuyển văn bản thành chữ hoa */  
    letter-spacing: 2px; /* Khoảng cách giữa các chữ cái */  
    border-bottom: 2px solid #3498db; /* Viền dưới màu xanh */  
    padding-bottom: 10px;  
    margin-top: 40px;  
    margin-bottom: 20px;  
}
```

#### **3.3.5.2. Tạo kiểu cho <p> (đoạn văn)**

Các đoạn văn là phương tiện chính cho nội dung văn bản.

```
p {  
    font-family: 'Open Sans', sans-serif;  
    font-size: 1.1em;  
    line-height: 1.8; /* Khoảng cách giữa các dòng */  
    color: #333;  
    text-align: justify; /* Căn đều văn bản để có giao diện khối sạch sẽ */  
    margin-bottom: 1em; /* Lê dưới tiêu chuẩn */  
}
```

#### **3.3.5.3. Tạo kiểu cho <div> (phản)**

Các phản tử <div> linh hoạt như các vùng chứa đa năng. Việc tạo kiểu cho chúng thường liên quan đến các thuộc tính bối cục.

Ví dụ:

```
/* Một div vùng chứa đơn giản */  
.container {  
width: 80%; /* Chiếm 80% chiều rộng của phần tử cha */  
max-width: 960px; /* Nhưng không rộng hơn 960px */  
margin: 0 auto; /* Căn giữa div theo chiều ngang */  
padding: 20px;  
background-color: white; /*Nền trắng/  
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* Bóng nhẹ */  
border-radius: 8px; /* Góc bo tròn */  
}
```

### **3.3.5.4. Tạo kiểu cho <span> (khoảng)**

Các phần tử <span> được sử dụng để áp dụng kiểu cho một phần nhỏ của văn bản hoặc một phần tử nội dòng cụ thể mà không làm phá vỡ luồng.

```
.highlight {  
background-color: #ffe082; /* Nền vàng nhạt */  
font-weight: bold;  
padding: 2px 5px;  
border-radius: 3px;  
}  
  
.important-text {  
color: #c0392b; /* Màu đỏ */  
font-style: italic;  
}
```

### **3.3.5.5. Tạo kiểu cho <a> (liên kết neo)**

Liên kết là các phần tử tương tác, việc tạo kiểu cho chúng rất quan trọng với trải nghiệm người dùng.

```
a {  
    color: #007bff; /* Màu liên kết xanh */  
    text-decoration: none; /* Bỏ gạch chân theo mặc định */  
    transition: color 0.3s ease-in-out; /* Chuyển đổi mượt mà cho hiệu ứng di chuột */  
}  
  
a:hover {  
    color: #0056b3; /* Màu xanh đậm hơn khi di chuột */  
    text-decoration: underline; /* Thêm gạch chân khi di chuột */  
}
```

### **3.3.5.6. Tạo kiểu cho <ul>, <ol>, <li> (danh sách)**

Danh sách giúp tổ chức thông tin một cách hiệu quả.

Ví dụ:

```
ul {  
    list-style-type: square; /* Dấu đầu dòng hình vuông */  
    margin-left: 20px;  
    line-height: 1.6;  
}  
  
ol {  
    list-style-type: decimal; /* Số thập phân */  
    margin-left: 25px;  
    line-height: 1.6;  
}  
  
li {  
    margin-bottom: 5px;  
    color: #444;  
}
```

```
/* Tạo kiểu cho danh sách lồng nhau */  
ul ul, ol ol {  
    margin-top: 5px;  
    margin-bottom: 5px;  
}
```

### 3.4. CÁC CÁCH NHÚNG CSS VÀO HTML

Có ba cách chính để nhúng CSS vào tài liệu HTML.

#### 3.4.1. Kiểu nội dòng (Inline Styles)

Kiểu nội dòng được áp dụng trực tiếp vào một phần tử HTML bằng cách sử dụng thuộc tính style.

##### \* **Ưu điểm**

- Nhanh chóng để áp dụng một kiểu độc lập cho một phần tử cụ thể.
- Có độ ưu tiên cao nhất, ghi đè các kiểu khác.

##### \* **Nhược điểm**

- Không thể tái sử dụng: Kiểu chỉ áp dụng cho một phần tử duy nhất.
- Phân tách kém: Trộn lẫn cấu trúc HTML và trình bày kiểu, làm cho mã khó đọc và khó bảo trì hơn.
- Không hiệu quả cho các dự án lớn.

Ví dụ:

```
<p style="color: blue; font-size: 16px;">Đây là một đoạn văn bản với kiểu nội  
dòng.</p>
```

```
<div style="background-color: lightgray; padding: 15px;">Đây là một div với  
kiểu nội dòng.</div>
```

Kết quả:

Đây là một đoạn văn bản với kiểu nội dòng.

Đây là một div với kiểu nội dòng.

### **3.4.2. Kiểu nội bộ (Internal/Embedded Styles)**

Kiểu nội bộ được đặt trong thẻ `<style>` bên trong phần `<head>` của tài liệu HTML. Các kiểu này chỉ áp dụng cho trang HTML đó.

\* ***Ưu điểm***

- Dễ dàng áp dụng kiểu cho một trang HTML duy nhất.
- Các kiểu được tập trung trong một phần của tài liệu.

\* ***Nhược điểm***

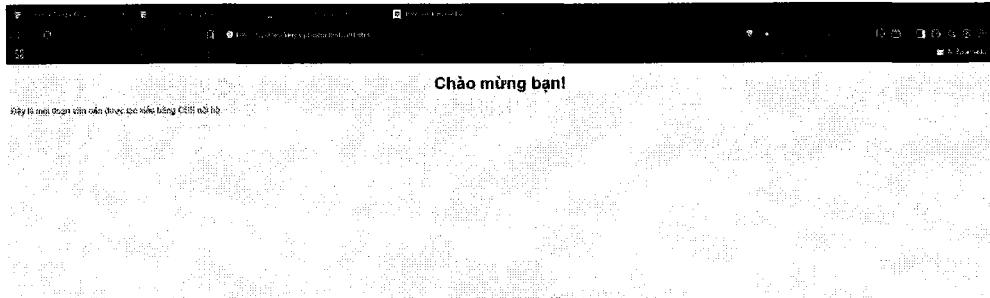
- Không thể tái sử dụng trên nhiều trang HTML.
- Vẫn trộn lẫn HTML và CSS trong cùng một tệp, mặc dù ở các phần khác nhau.
- Khi tài liệu lớn, thẻ `<style>` có thể trở nên rất dài.

Ví dụ:

```
<head>
<title>Trang với kiểu nội bộ</title>
<style>
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
}
h1 {
    color: #333;
    text-align: center;
}
p {
    color: #555;
    line-height: 1.5;
}
```

```
</style>
</head>
<body>
<h1>Chào mừng bạn!</h1>
<p>Đây là một đoạn văn bản được tạo kiểu bằng CSS nội bộ.</p>
</body>
```

Kết quả:



### 3.4.3. Kiểu bên ngoài (External Stylesheets)

Kiểu bên ngoài được viết trong một tệp .css riêng biệt và được liên kết với tài liệu HTML bằng thẻ `<link>` trong phần `<head>`. Đây là phương pháp được khuyên dùng và phổ biến nhất.

#### 3.4.3.1. Ưu điểm

- Tái sử dụng cao: Cùng một tệp CSS có thể được liên kết với nhiều trang HTML, đảm bảo tính nhất quán trên toàn bộ trang Web.
- Phân tách rõ ràng: Tách biệt hoàn toàn nội dung (HTML) khỏi trình bày (CSS), làm cho mã dễ đọc, dễ bảo trì và dễ gỡ lỗi hơn.
- Tải trang nhanh hơn: Trình duyệt có thể lưu trữ tệp CSS vào bộ nhớ đệm, giúp các trang tiếp theo tải nhanh hơn vì tệp CSS không cần tải lại.

#### 3.4.3.2. Nhược điểm

- Cần thêm một yêu cầu HTTP ban đầu để tải tệp CSS.

vv

Tệp index.html:

```
<head>
<title>Trang với kiểu bên ngoài</title>
<link rel="stylesheet" href="styles.css"> <!-- Liên kết đến tệp CSS bên ngoài -->
</head>
<body>
<h1>Chào mừng bạn!</h1>
<p>Đây là một đoạn văn bản được tạo kiểu bằng CSS bên ngoài.</p>
</body>
```

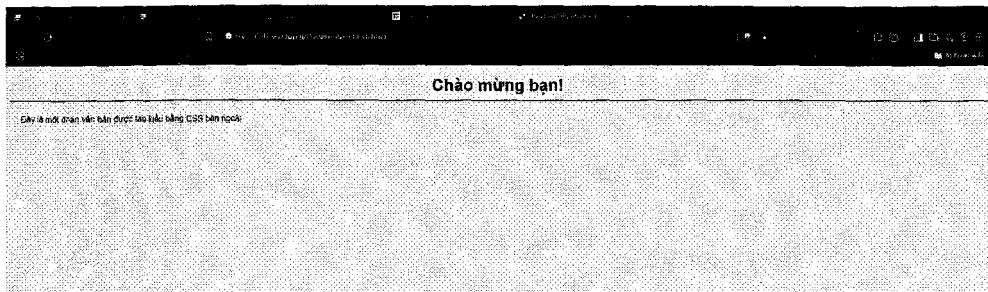
Tệp styles.css:

```
/* styles.css */
body {
    font-family: 'Open Sans', sans-serif;
    background-color: #e0f2f7;
}

h1 {
    color: #1a5276;
    text-align: center;
    border-bottom: 2px solid #1a5276;
    padding-bottom: 10px;
}

p {
    color: #333;
    line-height: 1.6;
    margin-left: 20px;
    margin-right: 20px;
}
```

Kết quả:



### 3.5. SELECTOR LÀ GÌ? (BỘ CHỌN TRONG CSS)

Trong CSS, selector (bộ chọn) là thành phần cốt lõi dùng để chỉ định chính xác phần tử HTML nào sẽ được áp dụng kiểu dáng (style). Nói cách khác, selector hoạt động như một “công cụ chọn lọc”, giúp bạn xác định các phần tử cụ thể trong trang Web để thay đổi cách chúng hiển thị, bao gồm các thuộc tính như màu sắc, kích thước chữ, khoảng cách, viền, vị trí và nhiều thuộc tính khác.

Hãy hình dung trang Web của bạn như một ngôi nhà với nhiều đồ đạc khác nhau (các phần tử HTML). Selector giống như việc bạn gọi tên hoặc chỉ đích danh một món đồ cụ thể (ví dụ: “cái ghế sofa màu xanh”, “tất cả các chiếc đèn trong phòng khách”, “cái bàn duy nhất có khắc tên”) để sau đó bạn có thể sơn lại màu, thay đổi kích thước hoặc di chuyển chúng.

Ví dụ:

Nếu bạn muốn tất cả các đoạn văn (`<p>`) trên trang có màu xanh, bạn dùng selector chọn thẻ `p` và áp dụng thuộc tính `màu` (`color`) cho chúng:

```
p {  
    color: blue; /* Đặt màu chữ thành màu xanh */  
    font-size: 16px; /* Đặt kích thước chữ là 16 pixel */  
}
```

Trong ví dụ trên:

- `p` là selector, có nhiệm vụ “chọn” tất cả các thẻ `<p>` (đoạn văn) có trong tài liệu HTML.

– { color: blue; font-size: 16px; } là khối khai báo. Khối này chứa một hoặc nhiều cặp thuộc tính: giá trị; (property: value;) được ngăn cách bởi dấu chấm phẩy. Mỗi cặp này gọi là một khai báo.

- color: blue; là một khai báo.
- font-size: 16px; là một khai báo.

Toàn bộ đoạn code p { color: blue; font-size: 16px; } được gọi là một quy tắc CSS (CSS Rule Set). Một quy tắc CSS luôn bắt đầu bằng một selector, sau là một khối khai báo được đặt trong cặp dấu ngoặc nhọn {}.

Bạn có thể dùng nhiều loại selector khác nhau để chọn phần tử theo nhiều cách: theo tên thẻ, theo class, theo ID, theo mối quan hệ giữa các phần tử, v.v.. Việc nắm vững các loại selector và cách chúng hoạt động là một trong những kiến thức nền tảng quan trọng nhất khi học CSS, đặc biệt là khi bạn phải giải quyết các bài toán thiết kế giao diện phức tạp và hiểu rõ cách CSS hoạt động trong các kỳ thi.

### **3.5.1. Bộ chọn cơ bản (Basic Selectors)**

CSS cung cấp một số bộ chọn (selector) cơ bản để bạn có thể nhanh chóng và trực quan nhắm mục tiêu vào các phần tử HTML để áp dụng kiểu dáng (style). Việc nắm vững ba loại bộ chọn này là nền tảng quan trọng cho mọi tùy chỉnh giao diện Web.

#### **3.5.1.1. Bộ chọn theo tên thẻ (Type Selector/Element Selector)**

Bộ chọn theo tên thẻ (còn gọi là Element Selector) được dùng để áp dụng kiểu dáng cho tất cả các phần tử có cùng tên thẻ HTML trên trang. Đây là cách đơn giản nhất để định kiểu cho các thành phần HTML chuẩn.

Cú pháp:

```
ten_the {  
    thuoc_tinh: gia_tri;  
}
```

Ví dụ:

```
p {  
    font-size: 16px; /* Tất cả đoạn văn sẽ có cỡ chữ 16px */
```

```
line-height: 1.5; /* Khoảng cách dòng là 1.5 lần kích thước chữ */  
}  
  
h1 {  
color: #333; /* Các tiêu đề cấp 1 có màu xám đậm */  
text-align: center; /* Căn giữa tiêu đề */  
}
```

*Giải thích:*

– Khi bạn khai báo p { ... }, mọi thẻ <p> trên trang sẽ nhận các thuộc tính được định nghĩa.

– Tương tự, h1 { ... } sẽ ảnh hưởng đến tất cả các thẻ <h1>.

Khi nào nên dùng:

– Khi bạn muốn áp dụng một kiểu dáng chung cho tất cả các thể hiện của một loại phần tử HTML (ví dụ: tất cả các đoạn văn, tất cả các nút bấm mặc định).

– Thích hợp cho việc thiết lập các kiểu dáng cơ bản, “chuẩn” cho các thẻ HTML.

### **3.5.1.2. Bộ chọn theo class (Class Selector)**

Class là một thuộc tính HTML linh hoạt dùng để gắn nhãn (phân loại) cho một hoặc nhiều phần tử HTML. Trong CSS, để chọn một class, bạn sử dụng dấu chấm(.) ngay trước tên class.

Cú pháp:

```
.ten_class {  
thuoc_tinh: gia_tri;  
}
```

Ví dụ:

HTML mẫu:

```
<h2 class="title">Chào mừng!</h2>  
<p class="title highlight">Đây là trang đầu tiên của bạn.</p>  
<span class="highlight">Một phần văn bản được làm nổi bật.</span>
```

CSS mẫu:

```
.title {  
    color: green; /* Chữ màu xanh */  
    font-weight: bold; /* Chữ in đậm */  
    text-transform: uppercase; /* Chữ viết hoa */  
}  
  
.highlight {  
    background-color: yellow; /* Nền màu vàng */  
    padding: 5px; /* Khoảng đệm 5px */  
}
```

*Giải thích:*

- Phần tử `<h2 class="title">` sẽ có màu xanh, chữ đậm và viết hoa.
- Phần tử `<p class="title highlight">` sẽ có cả kiểu dáng của `.title` và `.highlight`.
- Phần tử `<span class="highlight">` chỉ có kiểu dáng của `.highlight`.

### **3.5.1.3. Bộ chọn theo ID (ID Selector)**

ID là một định danh duy nhất cho một phần tử HTML cụ thể trên trang. Để chọn phần tử có ID trong CSS, bạn sử dụng dấu thăng (#) ngay trước tên ID.

Cú pháp:

```
#ten_id {  
    thuoc_tinh: gia_tri;  
}
```

Ví dụ:

HTML mẫu:

```
<div id="header">Trang của tôi</div>  
<div id="main-content">  
    <p>Nội dung chính của trang web.</p>
```

```
</div>

CSS mẫu:  
#header {  
background-color: lightgray; /* Nền màu xám nhạt */  
padding: 20px; /* Khoảng đệm 20px */  
text-align: center; /* Căn giữa nội dung */  
}  
#main-content {
```

```
width: 80%; /* Chiều rộng 80% */  
margin: 0 auto; /* Căn giữa theo chiều ngang */  
}
```

#### *Giải thích:*

- Phần tử `<div id="header">` sẽ nhận kiểu dáng của `#header`.
- Phần tử `<div id="main-content">` sẽ nhận kiểu dáng của `#main-content`.

#### *Lưu ý:*

– **Tính duy nhất:** Trong một tài liệu HTML, mỗi ID chỉ được sử dụng một lần duy nhất. Đây là quy tắc vàng của HTML và CSS. Nếu bạn cần áp dụng cùng một kiểu cho nhiều phần tử, hãy dùng class thay vì ID.

– **Độ ưu tiên cao:** Bộ chọn ID có độ ưu tiên (specificity) rất cao trong CSS. Điều này có nghĩa là kiểu dáng được định nghĩa bởi ID selector sẽ có khả năng ghi đè các kiểu dáng khác (như Type hoặc Class selector) nếu chúng cùng nhắm mục tiêu vào một phần tử. Đây là một khái niệm quan trọng sẽ được giải thích chi tiết hơn trong phần Specificity.

### **3.5.2. :hover và :active – tạo hiệu ứng tương tác**

Trong CSS, pseudo-class (giả lớp) là các “trạng thái đặc biệt” của một phần tử HTML mà bạn có thể áp dụng kiểu dáng (style) khi người dùng tương tác với chúng hoặc khi phần tử đạt một điều kiện nhất định. Thay vì định kiểu cho một phần tử cố định, pseudo-class cho phép bạn định kiểu cho các trạng thái động của phần tử đó.

Trong số rất nhiều pseudo-class, :hover và :active là hai trong số những loại phổ biến và quan trọng nhất để tạo ra các hiệu ứng tương tác tức thì, giúp giao diện trang Web trở nên sinh động và thân thiện hơn với người dùng.

### **3.5.2.1. :hover – khi di chuột vào phần tử**

Pseudo-class :hover được sử dụng để áp dụng kiểu dáng khi người dùng di chuột vào một phần tử mà không cần nhấp chuột. Đây là cách đơn giản và hiệu quả nhất để làm cho các nút, liên kết, hình ảnh hay bất kỳ vùng tương tác nào trở nên sinh động và thu hút sự chú ý.

Cú pháp:

```
selector:hover {  
    thuoc_tinh: gia_tri;  
}
```

**Cơ chế hoạt động:** Khi con trỏ chuột của người dùng đi vào vùng hiển thị của selector được chỉ định, các thuộc tính CSS trong khối khai báo của :hover sẽ được áp dụng. Khi con trỏ chuột rời khỏi vùng đó, các thuộc tính này sẽ tự động trở về trạng thái ban đầu của phần tử.

Ví dụ:

CSS

```
/* Kiểu dáng mặc định của liên kết */  
a {  
    color: blue;  
    text-decoration: none; /* Bỏ gạch chân mặc định */  
    font-size: 18px;  
}  
  
/* Kiểu dáng khi rê chuột vào liên kết */  
a:hover {  
    color: red; /* Chữ chuyển sang màu đỏ */  
    text-decoration: underline; /* Xuất hiện gạch chân */  
}
```

*Giải thích:* Khi người dùng di chuột vào một liên kết, chữ sẽ ngay lập tức chuyển sang màu đỏ, có gạch chân, báo hiệu rằng đây là một vùng có thể tương tác. Khi chuột rời đi, liên kết sẽ trở về màu xanh và không gạch chân.

### **3.5.2.2. :active – khi đang nhấp vào phần tử**

Pseudo-class:active được dùng để áp dụng kiểu dáng cho phần tử tại thời điểm nó đang được nhấn giữ (bằng chuột hoặc phím Enter/Spacebar). Nó cung cấp phản hồi tức thì cho người dùng rằng hành động nhấn đang diễn ra.

Cú pháp:

```
selector:active {  
    thuoc_tinh: gia_tri;  
}
```

**Cơ chế hoạt động:** :active được kích hoạt ngay khi người dùng nhấn chuột xuống (mousedown event) trên phần tử và giữ chuột. Kiểu dáng sẽ duy trì chừng nào người dùng còn giữ chuột trên phần tử đó. Khi chuột được thả ra (mouseup event) hoặc di chuyển ra khỏi phần tử trong khi vẫn giữ chuột, kiểu dáng :active sẽ mất đi.

Ví dụ:

CSS

```
/* Kiểu dáng mặc định của nút */  
button {  
    background-color: lightblue;  
    color: white;  
    padding: 10px 20px;  
    border: none;  
    border-radius: 5px;  
    font-size: 16px;  
    transition: background-color 0.2s ease; /* Tạo chuyển động mượt mà cho màu nền */  
}
```

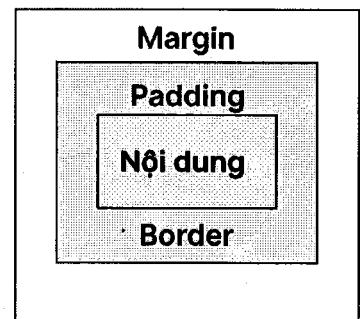
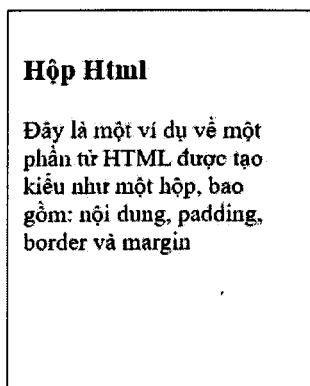
```
/* Kiểu dáng khi người dùng đang nhấn giữ nút */  
button:active {  
    background-color: dodgerblue; /* Nền nút chuyển sang xanh đậm hơn */  
    transform: translateY(1px); /* Nút dịch chuyển nhẹ xuống 1px, tạo cảm giác  
    “ấn xuống” */  
    box-shadow: 0 0 0 transparent; /* Loại bỏ đổ bóng nếu có để tăng cảm giác  
    nhấn */  
}
```

*Giải thích:* Khi người dùng nhấn giữ nút, màu nền nút sẽ chuyển sang màu xanh đậm hơn, nút sẽ dịch chuyển nhẹ xuống một pixel, tạo hiệu ứng thị giác của một nút đang bị “nhấn xuống”. Hiệu ứng này rất quan trọng để cung cấp phản hồi trực quan cho người dùng.

### 3.6. BOX MODEL

Mỗi phần tử HTML (ví dụ một đoạn văn, hình ảnh, nút bấm, v.v.) về cơ bản đều là một hình hộp chữ nhật gồm bốn lớp: phần nội dung, padding, border và margin.

Ví dụ:



Các thành phần của phần tử như sau.

### 3.6.1. Content Area (nội dung)

Content Area là phần trung tâm và quan trọng nhất của hộp, nơi chứa nội dung của phần tử hoặc các phần tử con khác.



#### Hộp Html

Đây là một ví dụ về một phần tử HTML được tạo kiểu như một hộp, bao gồm: nội dung, padding, border và margin

Phần văn bản trên chính là phần nội dung của phần tử. Ngoài văn bản, một phần tử hộp cũng có thể chứa hình ảnh, video, v.v..

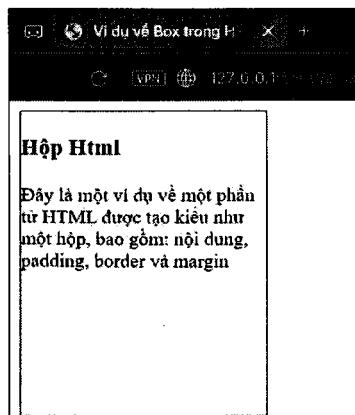
### 3.6.2. Kích thước của phần tử hộp

Kích thước của một phần tử được xác định bởi hai giá trị: width (chiều rộng) và height (chiều cao).

Kích thước của phần tử có thể là tuyệt đối (với đơn vị pixel (px), centimet (cm), v.v.) hoặc tương đối (so với phần tử cha (%)) hay với màn hình hiển thị (vw, vh)).

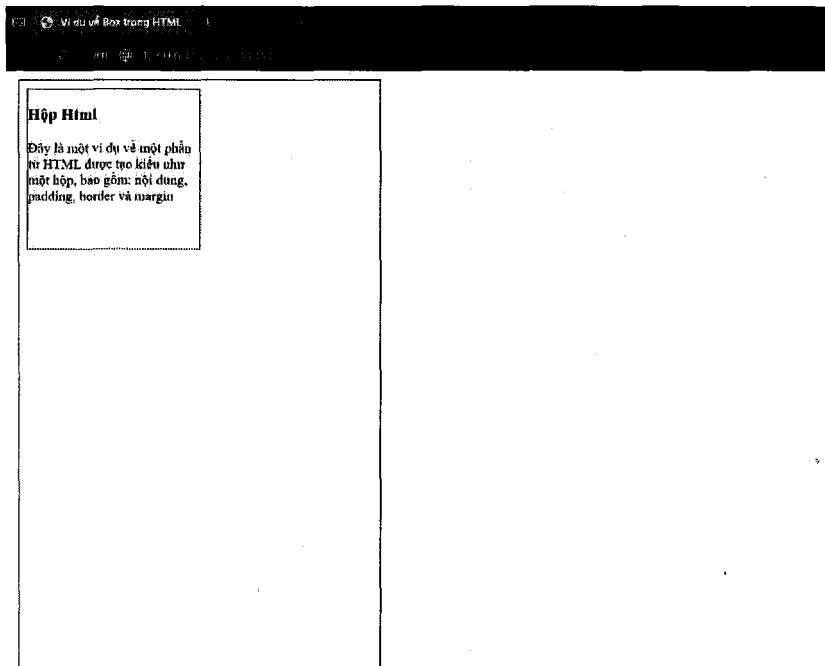
Ví dụ:

```
.box {  
width: 200px;  
height: 250px;  
border: 1px solid black;  
}
```



Phần tử có chiều rộng 200px và chiều cao 250px.

```
.box {  
    width: 50%;  
    height: 25vh;  
    border: 1px solid black;  
}
```



Phần tử có chiều rộng bằng một nửa (50%) so với phần tử cha và chiều rộng bằng 25% (¼) so với màn hình hiển thị.

### 3.6.3. Đường viền (Border)

Border là đường viền bao xung quanh dùng để làm nổi bật và phân biệt từng phần tử hộp riêng biệt với nhau.

Ta có thể điều chỉnh độ dày, màu sắc và các chi tiết khác của border.

Ví dụ:

```
.box-border {
```

```

width: 200px;

height: 300px;

border-top: 3px dashed #0d6efd; /* Nét đứt */

border-right: 2px dotted #198754; /* Nét chấm */

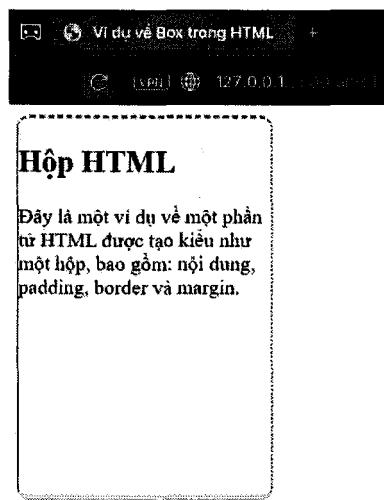
border-bottom: 4px double #ffc107; /* hai đường nét liền */

border-left: 1px solid #dc3545; /* nét liền */

border-radius: 10px; /* bo tròn góc */

}

```

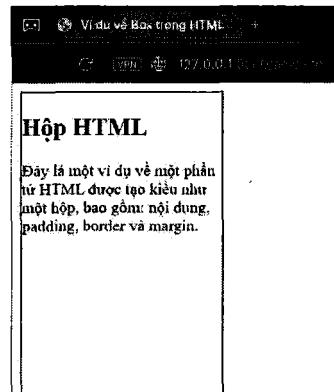


Ngoài cách cài đặt từng phía như trên, ta cũng có thể cài đặt chung cho cả bốn phía:

```

.box-border {
    width: 200px;
    height: 300px;
    border: 2px solid #333;
    box-sizing: border-box;
}

```



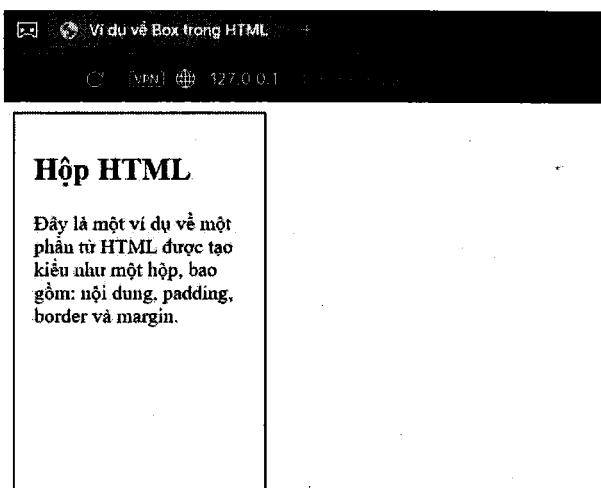
### 3.6.4. Padding (khoảng đệm trong)

Padding là khoảng trống giữa phần nội dung với phần border hoặc với các phần tử khác trong cùng phần tử cha.

Padding được sử dụng để giúp phần nội dung trở nên dễ nhìn và gọn gàng hơn.

Ví dụ:

```
.box-border {  
    width: 200px;  
    height: 300px;  
    padding-top: 10px;  
    padding-right: 15px;  
    padding-bottom: 10px;  
    padding-left: 15px;  
    border: 2px solid #333;;  
    box-sizing: border-box;  
}
```



Cũng như border, padding có nhiều cách điều chỉnh cho từng hướng hoặc chung:

```
/* Cú pháp: padding: trên phải dưới trái */  
.box-border {  
    padding: 10px 15px 10px 15px;  
}  
  
/* Nếu trên/dưới, trái/phải giống nhau */  
.box-border {  
    padding: 10px 15px; /* top/bottom: 10px, left/right: 15px */  
}  
  
/* Nếu tất cả các phía giống nhau */  
.box-border {
```

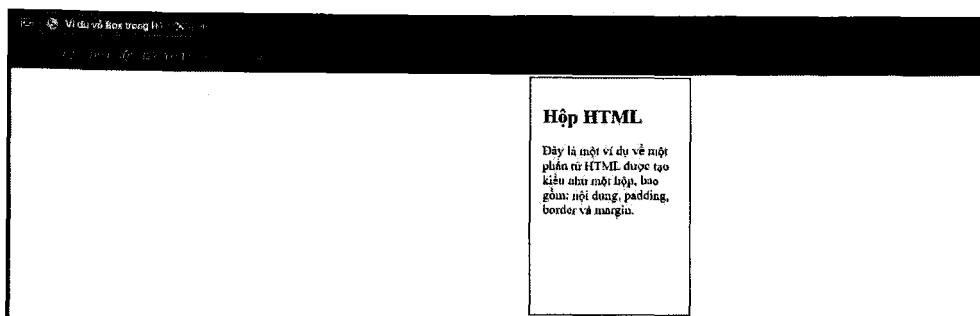
```
padding: 20px; /* tất cả 4 phía 20px */  
}
```

### 3.6.5. Margin (khoảng cách ngoài)

Margin là khoảng cách giữa đường viền của một phần tử với các phần tử khác hoặc với đường viền của phần tử cha.

Margin cũng có thể được sử dụng để căn ngang/dọc một phần tử.

```
.box-border {  
width: 200px;  
height: 300px;  
padding: 15px;  
border: 2px solid #333;  
margin: auto; /* Căn giữa ngang */  
margin-top: 5px; /* Margin âm */  
box-sizing: border-box;  
}
```



Bằng việc sử dụng lệnh margin:auto, phần tử sẽ được căn giữa so với phần tử cha (ở đây là phần tử body).

Ngoài cách trên thì những cách cài đặt từng phía hay trên/dưới, trái/phải như của padding và border cũng có thể áp dụng được.

### 3.6.6. Box-sizing

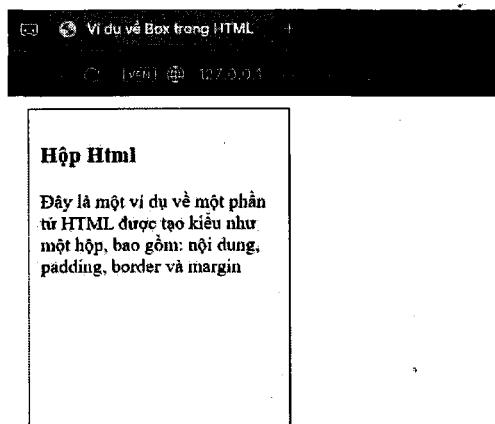
Ở trên, ta đã đề cập đến chiều rộng và chiều cao của một phần tử hộp, nhưng việc chiều rộng và chiều cao gồm chính xác những phần nào thì còn phụ thuộc vào thuộc tính box-sizing.

#### 3.6.6.1. Content-box

Content-box là giá trị mặc định trong CSS. Khi sử dụng giá trị content-box, phần chiều rộng và chiều cao sẽ chỉ tính cho phần nội dung, còn kích thước thật sẽ phải cộng thêm giá trị của padding và border.

Ví dụ:

```
.box {  
    width: 200px;  
    height: 250px;  
    border: 1px solid black;  
    padding: 10px;  
    margin: 10px;  
    box-sizing: content-box;  
}
```



Chiều rộng thực tế của phần tử trên sẽ là: 200 (nội dung) + 10\*2 (padding trái phải) + 1\*2 (border trái phải) = 222px.

Chiều cao thực tế là: 250 (nội dung) + 10\*2 (padding trên dưới) + 1\*2 (border trên dưới) = 272px.

#### 3.6.6.2. Border-box

Khác với box-content, khi sử dụng giá trị border-box, phần chiều rộng và chiều cao của phần tử hộp đã bao gồm cả giá trị padding và border.

Chính vì không phải thực hiện việc tính giá trị thực tế mà border-box thường được sử dụng nhiều hơn so với content-box.

Ví dụ:

```
.box {  
width: 200px;  
height: 250px;  
border: 1px solid black;  
padding: 10px;  
margin: 10px;  
box-sizing: border-box;  
}
```

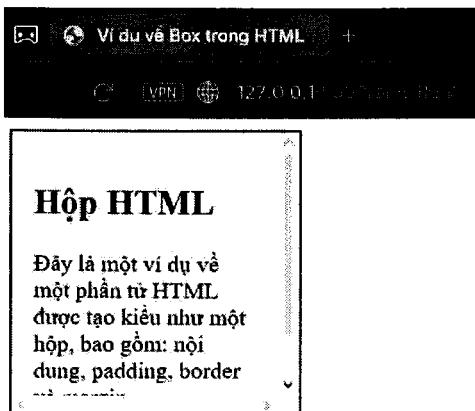
Có thể thấy, tuy cùng giá trị width và height, nhưng khi sử dụng border-box thì phần tử sẽ có kích thước nhỏ hơn do không phải cộng thêm giá trị padding và border.

### 3.6.7. Overflow – xử lý tràn nội dung

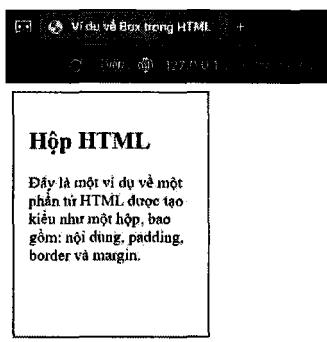
Khi phần nội dung dài vượt quá kích thước của phần tử, thuộc tính overflow sẽ quyết định cách mà ta xử lý chúng:

- Visible: Hiển thị phần thừa (mặc định);
- Hidden: Ẩn phần thừa;
- Scroll: Luôn hiện thanh cuộn;
- Auto: Tự động hiện thanh cuộn khi cần.

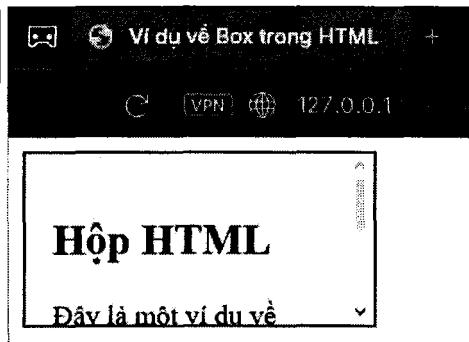
Khi sử dụng scroll:



Khi sử dụng overflow:hidden

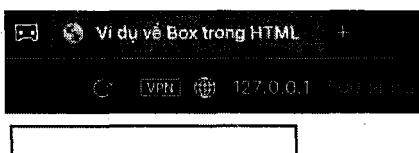


Khi sử dụng overflow:visible

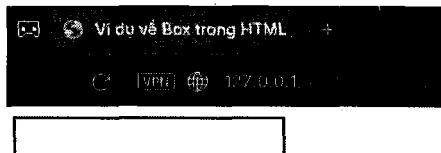


Còn overflow:auto thì giống như overflow:scroll nhưng chỉ hiện thanh cuộn khi cần thiết.

Khi không cần



Khi cần:



## 3.7. BACKGROUND

### 3.7.1. Tổng quan về background trong CSS

Trong thiết kế Web, background (nền) là yếu tố quan trọng giúp trang Web trở nên sinh động, thu hút và dễ đọc hơn. CSS cung cấp nhiều thuộc tính cho phép chúng ta thiết lập màu nền, hình nền, vị trí, kích thước và các hiệu ứng liên quan đến nền cho từng phần tử HTML.

Có hai thuộc tính background cơ bản và quan trọng nhất là:

- Background-color: Thiết lập màu nền.
- Background-image: Thiết lập hình ảnh nền.

### **3.7.2. Thiết lập màu nền với background-color**

#### **3.7.2.1. Khái niệm**

- Thuộc tính background-color dùng để đặt màu nền cho một phần tử HTML (thẻ div, body, p, h1, v.v.).
- Màu sắc có thể được khai báo bằng tên màu tiếng Anh, mã HEX, mã RGB, hoặc RGBA.

Cú pháp:

```
selector {  
background-color: giá_trị_màu;  
}
```

Ví dụ:

```
body {  
background-color: #f2f2f2; /* Màu xám nhạt */  
}  
  
h1 {  
background-color: yellow; /* Màu vàng */  
}  
  
p {  
background-color: rgb(173, 216, 230); /* Màu xanh da trời nhạt */  
}
```

*Chú ý:*

- Nếu không khai báo background-color, phần tử sẽ trong suốt, kế thừa màu nền từ phần tử cha.
- Màu nền nên lựa chọn hài hòa với màu chữ để đảm bảo dễ đọc.

#### **3.7.2.2. Cách khai báo màu sắc**

- Tên màu tiếng Anh: red, green, blue, yellow, pink, orange, violet, gray, black, white, v.v..

Ví dụ: background-color: red.

– Mã HEX: Sử dụng dấu # và 6 ký tự số hoặc chữ (0-9, A-F).

Ví dụ: background-color: #00FF00; (màu xanh lá cây).

– RGB: Dùng ba giá trị số từ 0 đến 255 cho Đỏ – Xanh lá – Xanh dương.

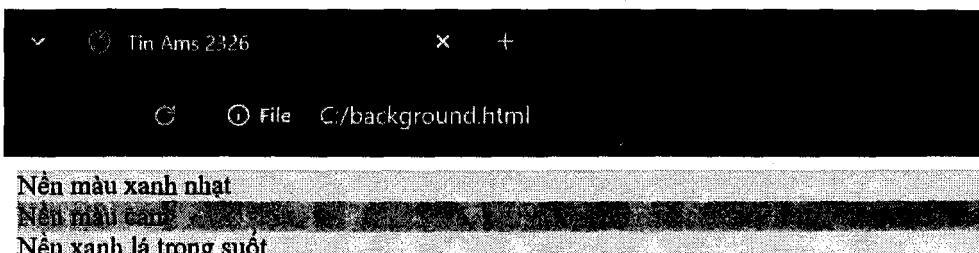
Ví dụ: background-color: rgb(255, 255, 0); (màu vàng).

– RGBA: Giống RGB, thêm giá trị alpha (độ trong suốt) từ 0 (trong suốt) đến 1 (đậm).

Ví dụ: background-color: rgba(0, 0, 0, 0.5); (đen mờ 50%).

Code:

```
<head>
    <title>Tin Ams 2326</title>
    <style>
        .box1 { background-color: lightblue; }
        .box2 { background-color: #ffa500; }
        .box3 { background-color: rgba(34, 139, 34, 0.3); }
    </style>
</head>
<body>
    <div class="box1">Nền màu xanh nhạt</div>
    <div class="box2">Nền màu cam</div>
    <div class="box3">Nền xanh lá trong suốt</div>
</body>
```



### **3.7.3. Thiết lập hình nền với background-image**

#### **3.7.3.1. Khái niệm**

- Thuộc tính background-image dùng để đặt hình ảnh làm nền cho một phần tử HTML.
- Hình ảnh có thể là file ảnh định dạng .jpg, .png, .gif, v.v. trên máy chủ hoặc một đường link trực tiếp.

Code:

```
selector {  
background-image: url("đường_dẫn_ảnh");  
}
```

#### **3.7.3.2. Thuộc tính**

- Background-repeat: xác định hình nền có lặp lại hay không.
- Repeat (mặc định): lặp cả chiều ngang và dọc.
- No-repeat: không lặp, chỉ hiển thị một ảnh.
- Repeat-x: lặp theo chiều ngang.
- Repeat-y: lặp theo chiều dọc.

Ví dụ: background-repeat: no-repeat.

- Background-position: Xác định vị trí hiển thị hình nền.
- Giá trị: left, right, center, top, bottom, hoặc dùng tọa độ px, %.

Ví dụ: background-position: right top.

- Background-size: Thiết lập kích thước ảnh nền.
- Giá trị: cover (phủ kín), contain (vừa đủ hiển thị), hoặc kích thước cụ thể (px, %).

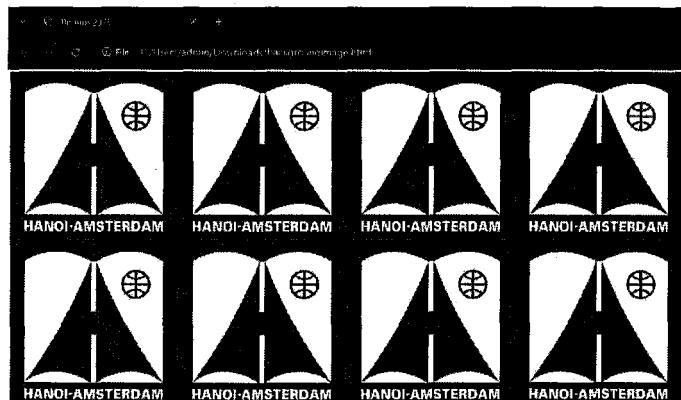
Ví dụ: background-size: cover.

- Background-attachment: Ảnh nền có cuộn theo trang không.
- Scroll: Ảnh nền cuộn cùng nội dung (mặc định).
- Fixed: Ảnh nền cố định khi cuộn trang.

Ví dụ: background-attachment: fixed.

Code:

```
<head>
<title>Tin Ams 2326</title>
<style>
body {
background: #fff;
}
.bg-demo-repeat {
width: 100vw;
height: 100vh;
background-image:url('https://dongphuchaianh.vn/wp-content/uploads/2023/06/y-nghia-logo-amd.jpg');
background-size: auto;
background-repeat: repeat;
background-position: top left;
}
</style>
</head>
<body>
<div class="bg-demo-repeat"></div>
</body>
```



## 3.8. FONT

### 3.8.1. Tổng quan về định dạng văn bản trong CSS

Font (phông chữ) là kiểu chữ được sử dụng để hiển thị văn bản trên trang Web. Việc lựa chọn font phù hợp rất quan trọng vì nó ảnh hưởng lớn đến tính thẩm mỹ và trải nghiệm người dùng. Một font dễ đọc và hài hòa sẽ giúp nội dung rõ ràng hơn, tăng tương tác và cải thiện trải nghiệm người dùng. Ngược lại, chọn font không phù hợp có thể khiến trang Web khó đọc và gây ấn tượng xấu. Ngày nay, typography (nghệ thuật sắp đặt chữ) chiếm phần lớn thiết kế Web, nên tối ưu font chữ cũng chính là tối ưu giao diện và hình ảnh thương hiệu của trang Web.

Ví dụ dưới đây minh họa sự khác biệt giữa font mặc định và font tùy chọn trên trang Web:

html

```
<p>Đoạn văn với font mặc định của trình duyệt.</p> <!-- default font -->
```

```
<p style="font-family:'Courier New', monospace;">
```

Đoạn văn với font tùy chọn Courier New.

```
</p> <!-- dùng font Courier New -->
```

Đoạn văn với font mặc định của trình duyệt.

Đoạn văn với font tùy chọn Courier

(Đoạn văn đầu dùng font mặc định, đoạn sau dùng font "Courier New" nên hiển thị kiểu chữ khác).

### 3.8.2. Các thuộc tính font cơ bản

CSS cung cấp nhiều thuộc tính để định dạng font chữ của văn bản, giúp trang Web hiển thị đẹp mắt và dễ đọc. Các thuộc tính quan trọng nhất gồm font-family, font-size, font-style, font-weight và line-height – chúng cho phép chọn kiểu chữ, cỡ chữ, kiểu dáng chữ (nghiêng/thẳng), độ đậm nhạt và khoảng cách dòng.

#### 3.8.2.1. Thuộc tính font-family

Thuộc tính font-family dùng để chỉ định font cho một phần tử. Bạn có thể liệt kê một danh sách nhiều font, trong đó trình duyệt sẽ dùng font đầu tiên có sẵn trên máy người dùng, nếu không có sẽ thử font tiếp theo. Thông thường, ta ghi

tên font chính trước, sau đó là các font dự phòng, kết thúc bằng một font generic (như serif, sans-serif, monospace, v.v.) để đảm bảo luôn có font hiển thị.

Ví dụ: font-family: "Times New Roman", Georgia, serif; nghĩa là ưu tiên dùng Times New Roman, nếu không có thì dùng Georgia, nếu cả hai đều không có thì trình duyệt sẽ dùng một font serif có sẵn bất kỳ.

Lưu ý: Tên font có khoảng trắng nên được đặt trong dấu ngoặc kép.

html

```
<p style="font-family: Arial, sans-serif;">
```

Văn bản này dùng font Arial làm chính, sans-serif làm dự phòng.

```
</p>
```

Ví dụ trên, trình duyệt sẽ cố dùng Arial; nếu máy không có Arial thì sẽ dùng font thuộc nhóm sans-serif có sẵn.

### **3.8.2.2. Thuộc tính font-size**

Thuộc tính font-size quy định kích cỡ chữ. Cỡ chữ có thể đặt bằng đơn vị tuyệt đối hoặc đơn vị tương đối. Đơn vị tuyệt đối phổ biến nhất là px (pixel) – kích thước cố định trên màn hình. Ngoài ra, còn có các đơn vị tương đối như %, em, rem để kích thước chữ có thể thay đổi tùy ngữ cảnh. Đơn vị em lấy kích cỡ chữ của phần tử cha làm tham chiếu, còn rem tham chiếu kích cỡ chữ gốc của trang (thường là của html, mặc định khoảng 16px).

Ví dụ: font-size: 150% nghĩa là bằng 150% cỡ chữ phần tử cha, font-size: 1.5em nghĩa là gấp 1.5 lần cỡ chữ cha, còn 1.5rem là gấp 1.5 lần cỡ chữ gốc. Việc dùng em, rem hoặc % giúp thiết kế responsive linh hoạt hơn, trong khi px cho kích cỡ cố định.

html

```
<p style="font-size: 16px;">Đoạn văn cỡ chữ 16px (cỡ chuẩn mặc định).</p>
```

```
<p style="font-size: 120%;">Đoạn văn cỡ chữ 120% (lớn hơn một chút).</p>
```

```
<p style="font-size: 1.5em;">Đoạn văn cỡ chữ 1.5em (lớn hơn nữa so với bình thường).</p>
```

**Đoạn văn cỡ chữ 16px (cỡ chuẩn mặc định).**

**Đoạn văn cỡ chữ 120% (lớn hơn một chút).**

**Đoạn văn cỡ chữ 1.5em (lớn hơn nữa so với bình thường).**

(Ba đoạn văn trên lần lượt có cỡ chữ 16px cố định, 120% so với cỡ chữ gốc và 1.5em so với cỡ chữ mặc định).

### **3.8.2.3. Thuộc tính font-style**

Thuộc tính font-style dùng để chọn kiểu dáng chữ nghiêng hay bình thường. Giá trị thường dùng là normal (bình thường, không nghiêng), italic (in nghiêng kiểu chữ nghiêng thực sự nếu font có hỗ trợ) và oblique (in nghiêng kiểu chéch). Về mặt hiệu ứng, italic và oblique đều làm chữ nghiêng. Thông thường, italic là kiểu chữ nghiêng được thiết kế riêng (như chữ viết tay hoặc viết kiểu in nghiêng), còn oblique thường là chữ bình thường được xoay nghiêng đi một góc. Nếu font không có phiên bản italic/oblique, trình duyệt có thể giả lập bằng cách tự động làm nghiêng chữ thường. Mặc định văn bản là normal. Ta dùng thuộc tính này để nhấn mạnh hoặc trích dẫn.

html

```
<p style="font-style: italic;">Đây là văn bản in nghiêng (italic).</p>
```

(Đoạn văn trên sẽ hiển thị chữ nghiêng. Nếu đổi italic thành oblique, kết quả trông tương tự nhưng cách trình duyệt xử lý có thể khác).

### **3.8.2.4. Thuộc tính font-weight**

Thuộc tính font-weight điều chỉnh độ đậm nhạt (độ dày nét chữ) của font. Giá trị thông dụng là normal (bình thường) tương đương độ nặng 400 và bold (đậm) tương đương 700. Ngoài ra, CSS cho phép dùng các giá trị số 100 đến 900 (tăng dần, mỗi bước 100) để chỉ độ dày cụ thể – ví dụ 100 (mỏng nhất), 300 (light), 400 (normal), 500 (medium), 700 (bold), 900 (đậm nhất, còn gọi là black). Không phải font nào cũng hỗ trợ đủ tất cả các mức trọng lượng; thường chỉ có một vài mức như 400 và 700. Khi đặt font-weight, nếu font không có đúng độ đậm yêu cầu, trình duyệt sẽ tự làm đậm hoặc làm mỏng gần đúng. Thuộc tính này hay được dùng để tạo điểm nhấn: ví dụ tiêu đề thường đậm hơn nội dung.

html

```
<h2 style="font-weight: bold;">Tiêu đề văn bản in đậm</h2>
<p style="font-weight: 400;">Đoạn văn với chữ thường (400).</p>
<p style="font-weight: 700;">Đoạn văn với chữ đậm (700).</p>
```

## Tiêu đề văn bản in đậm

Đoạn văn với chữ thường (400).

Đoạn văn với chữ đậm (700).

Ví dụ trên: Tiêu đề dùng bold tương đương 700, các đoạn văn minh họa (font-weight 400 và 700).

### 3.8.2.5. Thuộc tính line-height

Thuộc tính line-height quy định chiều cao dòng, tức khoảng cách dọc giữa các dòng chữ. Hiểu đơn giản, line-height tạo khoảng trống trên dưới giữa hai dòng văn bản liên tiếp, tính từ đường cơ sở (baseline) của chữ. Giá trị mặc định normal thường vào khoảng 1.2 (120% chiều cao chữ) tùy trình duyệt – tức dòng cao hơn cỡ chữ một chút để các dòng không dính sát nhau. Ta có thể tăng line-height để dòng thoáng, dễ đọc hơn, hoặc giảm để các dòng sát lại khi cần thiết kế đặc biệt. Thuộc tính này nhận các giá trị: số không đơn vị, độ dài cố định hoặc phần trăm. Khi dùng một số unitless (không kèm đơn vị), con số đó sẽ được nhân với kích cỡ chữ để tính khoảng cách dòng (ví dụ line-height: 1.5; nghĩa là mỗi dòng cao bằng 1.5 lần kích thước font hiện tại). Còn nếu dùng đơn vị như px hoặc %, ví dụ line-height: 24px; hoặc 150%, thì dòng cao tương ứng giá trị tuyệt đối đó. Thông thường, cách dùng unitless (không đơn vị) được khuyến khích vì nó thuận tiện trong tính toán và thừa kế line-height.

html

```
<p style="line-height: normal;">
Dòng thứ nhất của đoạn văn.<br>
Dòng thứ hai của đoạn văn.
</p> <!-- line-height mặc định (normal) -->
```

```
<p style="line-height: 1.5;">  
Dòng thứ nhất của đoạn văn.<br>  
Dòng thứ hai của đoạn văn.  
</p> <!-- line-height 1.5 (150%) -->
```

Dòng thứ nhất của đoạn văn.  
Dòng thứ hai của đoạn văn.

Dòng thứ nhất của đoạn văn.  
Dòng thứ hai của đoạn văn.

Đoạn đầu không thiết lập line-height nên dùng giá trị normal mặc định; đoạn sau đặt line-height = 1.5 làm tăng khoảng cách giữa hai dòng, giúp văn bản thoáng và dễ đọc hơn.

### 3.9. POSITION + FLOAT

Chúng ta đã tìm hiểu về nhiều tính chất của CSS giúp thay đổi ngoại hình của các phần tử văn bản cũng như những khối mà chúng tạo ra ở phần trước. Tuy nhiên, chúng ta mới chỉ định dạng các phần tử khi chúng xuất hiện theo tuần tự trên trình duyệt. Ở phần này, chúng ta sẽ tìm hiểu về Positioning và Floating trong CSS.

#### 3.9.1. Giới thiệu về Positioning và Floating trong CSS

Khi xây dựng một trang Web, việc hiển thị nội dung theo đúng vị trí mong muốn là rất quan trọng. Ví dụ, để hiển thị hình ảnh nằm bên trái và văn bản nằm bên phải, hay một hộp thông báo luôn cố định ở góc màn hình, người dùng cần phải sử dụng CSS để định vị phần tử.

Để phục vụ mục đích này, chúng ta có hai kỹ thuật rất phổ biến trong CSS để điều chỉnh vị trí các phần tử trong trang Web:

- Positioning (định vị): Giúp bạn đặt phần tử ở vị trí tuyệt đối, tương đối, cố định, v.v. tùy theo bố cục người dùng mong muốn.
- Floating (làm trôi): Dùng để đẩy phần tử sang trái hoặc phải, áp dụng khi người dùng muốn sắp xếp hình ảnh và văn bản nằm cạnh nhau.

Việc nắm vững hai kỹ thuật này sẽ giúp người dùng làm chủ việc trình bày trang Web một cách đẹp mắt và có logic.

### **3.9.2. Positioning – định vị phần tử**

#### **3.9.2.1. Khái niệm**

Positioning (hay còn gọi là “định vị”) là kỹ thuật giúp điều khiển chính xác vị trí của một phần tử HTML trên trang Web. Thay vì chỉ phụ thuộc vào luồng mặc định (từ trên xuống dưới, trái sang phải), người dùng có thể sử dụng position để di chuyển, xếp chồng hoặc cố định phần tử theo ý muốn. Positioning có thể định vị các phần tử với độ chính xác ở mức độ pixel.

#### **3.9.2.2. Giới thiệu các giá trị của Position**

CSS cung cấp một số các phương thức để định vị các phần tử trên trình duyệt. Các phần tử có thể được đặt ở vị trí có quan hệ với vị trí bình thường chúng xuất hiện trên luồng nội dung hoặc có thể đứng tách biệt với luồng và được đặt ở một vị trí xác định trên Web.

CSS có năm phương thức Position phổ biến là:

- Static: Position này sẽ hiển thị các phần tử ở vị trí tương ứng với luồng văn bản của code.
- Relative: Position này di chuyển phần tử tương ứng với vị trí mặc định của nó. Điểm đặc nhất của Position: Relative đó là vị trí mặc định của phần tử sẽ không bị chiếm bởi các phần tử.
- Absolute: Position này sẽ loại bỏ phần tử khỏi luồng văn bản và đặt ở vị trí tương ứng so với màn hình hiển thị hoặc theo một phần tử khác. Khác với Position: Relative, vị trí mặc định của phần tử sẽ được sử dụng bởi các phần tử khác.
- Fixed: Position này sẽ cố định phần tử ở một vị trí trên màn hình hiển thị kể cả khi người dùng kéo xuống. Các phần tử với Position: Fixed cũng sẽ được loại bỏ khỏi luồng văn bản và được đặt ở vị trí tương ứng với màn hình hiển thị (nhưng không tương ứng với các phần tử khác).
- Sticky: Position này là sự kết hợp giữa Relative và Fixed. Các phần tử với Position: Sticky sẽ được hiển thị như một phần tử Relative ở điều kiện thường, nhưng sẽ hoạt động như một phần tử Fixed khi người dùng kéo xuống một vị trí xác định.

Các phần tử được cài đặt mặc định là Position: Static.

### **3.9.2.3. Các từ khóa để xác định vị trí**

Sau khi đã thiết lập được phương thức Position, vị trí cuối cùng của phần tử còn có thể được xác định bởi việc sử dụng các thuộc tính bù đắp (tên tiếng Anh là offset properties). Các thuộc tính này chỉ áp dụng cho các phần tử đã được thiết lập Position khác “Static”.

Bốn thuộc tính này là: left (trái), right (phải), bottom (đáy), top (đỉnh).

Mỗi thuộc tính sẽ được gắn một giá trị để định nghĩa khoảng cách phần tử được dịch chuyển khỏi cạnh tương ứng. Ví dụ, giá trị của thuộc tính “top” sẽ định nghĩa khoảng cách giữa cạnh trên của phần tử so với trình duyệt hoặc với các phần tử khác. Các giá trị có thể có cả giá trị âm hoặc dương.

### **3.9.2.4. Đi sâu vào các Position của CSS**

#### *a) Position Static*

Static là Position mặc định của một phần tử. Các phần tử với Position: Static không bị ảnh hưởng bởi các thuộc tính bù đắp.

#### *b) Position Relative*

Định vị bằng Position: Relative di chuyển phần tử tương ứng với vị trí ban đầu của nó trên luồng văn bản.

Cách hoạt động của Position này là:

- Phần tử vẫn giữ chỗ trong bố cục ban đầu.
- Các thuộc tính bù đắp sẽ dịch chuyển nội dung của phần tử nhưng không thay đổi vị trí logic của nó trong tài liệu.

Ví dụ:

<p class =“position\_relative”>HTML (viết tắt của HyperText Markup Language) là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang Web. Được phát triển từ năm 1991 bởi Tim Berners-Lee và phát hành chính thức vào năm 1993, HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW.</p>

<style>

.position\_relative{

```
width: 400px;  
height: 500px;  
display: flex;  
margin: 20px;  
padding: 20px;  
position: relative;  
left: 100px;  
top: 50px;  
}  
</style>
```

### Vị trí cũ

HTML (viết tắt của HyperText Markup Language) là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang web. Được phát triển từ năm 1991 bởi Tim Berners-Lee và phát hành chính thức vào năm 1993. HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW

### Vị trí sau khi dùng Position: Relative

HTML (viết tắt của HyperText Markup Language) là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang web. Được phát triển từ năm 1991 bởi Tim Berners-Lee và phát hành chính thức vào năm 1993. HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW

Từ ví dụ, có thể thấy phần văn bản đã được dịch sang phải 100 pixel (tương ứng với left: 100px) và dịch xuống dưới 50 pixel (tương ứng với top: 50px).

Tuy vậy, do cách hoạt động của Position: Relative, việc các phần tử khi hiển thị bị đè lên nhau là không hiếm thấy.

Chẳng hạn với đoạn code sau:

```
<p class = "position_relative">HTML (viết tắt của HyperText Markup Language)  
là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang web.</p>
```

```
<p class = "position_relative_1">Được phát triển từ năm 1991 bởi Tim Berners-Lee  
và phát hành chính thức vào năm 1993, HTML đã trở thành nền tảng cốt lõi của mọi  
website trên WWW</p>
```

```
<style>

.position_relative{
width: 400px;
display: flex;
margin: 20px;
padding: 20px;
position: relative;
left: 100px;
top: 100px;
}

.position_relative_1{
width: 400px;
display: flex;
margin: 20px;
padding: 20px;
}

</style>
```

sẽ khiến hai đoạn văn bản chồng lên nhau như sau:

HTML (viết tắt của HyperText Markup Language) là ngôn  
Được phát triển bởi Tim Berners-Lee và một  
hành chính thức vào năm 1993, HTML đã trở thành nền tảng  
cốt lõi của mọi website trên WWW

Đồng thời, khoảng trống không được sử dụng bởi phần tử nhìn không đẹp  
mắt, nên Position này không được dùng nhiều như Position: Absolute. Tuy vậy,  
Position Relative lại thường được sử dụng để tạo các phần tử chứa.

### c) Position Absolute

Position Absolute có cách hoạt động khác ở một số chỗ so với Position Relative  
và bớt cứng nhắc hơn trong việc định vị và đặt phần tử trên Web.

### Cách hoạt động của Position Absolute:

- Khi một phần tử được gán Position: Absolute, nó thoát hoàn toàn khỏi luồng thông thường của trang.
- Khác với Position: Relative, phần tử với Position: Absolute sẽ không chiếm chỗ và các phần tử khác cũng sẽ không bị ảnh hưởng bởi phần tử này trong bố cục.
- Vị trí của phần tử có thể được điều chỉnh bởi các thuộc tính bù đắp.

Ví dụ:

```
<div class="pos_abso">HTML là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang web.</div>
```

```
<div class="pos_abso_1">Được phát triển từ năm 1991 bởi Tim Berners-Lee và phát hành chính thức vào năm 1993, HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW.</div>
```

```
<style>  
.pos_abso {  
    position: absolute;  
    width: 300px;  
    display: flex;  
    height: 100px;  
    top: 100px;  
    left: 350px;  
}  
.pos_abso_1 {  
    display: flex;  
    width: 300px;  
}  
</style>
```

## Trước khi dùng Position: Absolute

HTML là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang web.

Được phát triển từ năm 1991 bởi Tim Berners-Lee và phát hành chính thức vào năm 1993, HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW.

## Sau khi dùng Position: Absolute

Được phát triển từ năm 1991 bởi Tim Berners-Lee và phát hành chính thức vào năm 1993, HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW.

HTML là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang web.

Có thể thấy, phần tử với Position: Absolute đã được di chuyển ra chỗ khác bằng các thuộc tính bù đắp. Phần khoảng trống không được sử dụng đã được sử dụng bởi phần tử còn lại không cài đặt Position: Absolute.

Mặc dù ở ví dụ trên, phần tử được thiết lập Position: Absolute được định vị tương ứng với trình duyệt, Position: Absolute thực chất di chuyển phần tử tương ứng với các phần tử chứa khác. Vì thế, để sử dụng Position Absolute hiệu quả và đúng cách, cần phải xác định được chính xác phần tử nào đang chứa phần tử sẽ được thiết lập Position: Absolute.

### d) Position Fixed

Về cơ bản, Position Fixed hoạt động giống với Position Absolute. Tuy vậy, sử dụng các thuộc tính bù đắp sẽ di chuyển phần tử tương ứng với màn hình hiển thị, hoặc có thể nói là phần tử đứng yên kể cả khi người dùng kéo lên, xuống trang Web.

Position Fixed thường được sử dụng để tạo danh mục, menu, v.v. mà đứng yên một chỗ ở trên cùng, ở đáy hoặc ở hai bên màn hình kể cả khi người dùng di chuyển màn hình. Tuy vậy, nếu để phần tử được thiết lập Position: Fixed ở đáy màn hình thì cần phải dành ra đủ khoảng trống để các phần nội dung không bị giấu bởi phần tử.

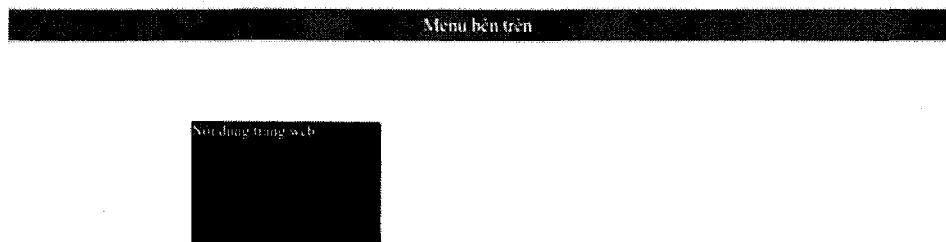
Ví dụ:

```
<div class = "pos_fixed">Menu bên trên</div>
```

```
<div class =“pos_fixed_1”>Nội dung trang web</div>
<style>
.pos_fixed{
width: 100%;
height: 50px;
background-color: fuchsia; /*Nền tím*/
position: fixed;
top: 20px;
z-index: 100; /*Phần này sẽ giải thích sau*/
font-size: 40px;
color: wheat;
display: flex;
justify-content: center;
align-items: center;
}
.pos_fixed_1{
width: 20%;
height: 200px;
background-color: black; /*Nền đen*/
position: relative;
top: 200px;
left: 300px;
color: white; /*Làm nổi bật chữ*/
font-size: 30px;
}
</style>
```

Kết quả:

Màn hình bình thường:



Màn hình sau khi người dùng di chuyển màn hình:



Các phần tử được thiết lập Position: Fixed cũng gây nhiều vấn đề khi tài liệu được in ấn vì chúng sẽ được in lên mọi trang. Vì thế, trước khi in, người dùng nên loại bỏ Position: Fixed.

### **3.9.2.5. *Đi sâu vào việc di chuyển các phần tử đã được thiết lập Position***

#### *a) Phần tử chứa*

Vị trí và kích thước của một phần tử nhiều lúc được tính toán dựa trên một phần tử chứa khác. Vì thế, việc xác định đúng phần tử chứa là tối quan trọng để định vị và di chuyển phần tử một cách chính xác.

Mặc dù việc xác định phần tử chứa có nhiều quy luật khác nhau nhưng chúng ta có thể tóm gọn chúng thành hai ý chính:

– Nếu phần tử được thiết lập Position không nằm trong một phần tử khác cũng được thiết lập Position, nó sẽ được định vị và di chuyển tương ứng với phần tử chứa mặc định (hoặc nói dễ hiểu hơn là phần tử được tạo bởi thẻ <html>).

– Nếu như phần tử (PT1) nằm trong phần tử khác (PT2) và PT2 này đã được thiết lập Position thì PT1 sẽ được định vị và di chuyển tương ứng với các cạnh của PT2.

Ví dụ:

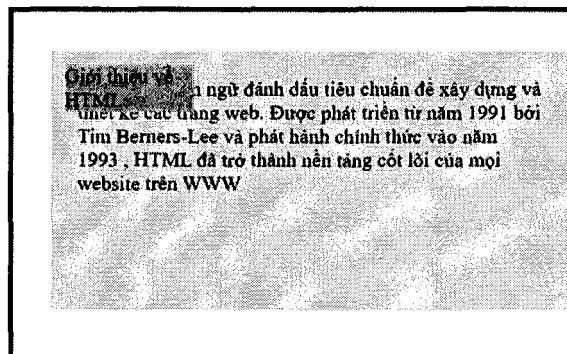
Đoạn code sau:

```
<div class = "pos_1">  
    <div class = "pos_2">Giới thiệu về HTML</div>
```

HTML là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang Web. Được phát triển từ năm 1991 bởi Tim Berners-Lee và phát hành chính thức vào năm 1993, HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW.

```
<style>  
  
body{  
    border: 10px solid blue; /*Thể hiện viền màn hình*/  
}  
  
.pos_1{  
    background-color: wheat;  
    width: 400px;  
    height: 200px;  
    padding: 20px;  
    margin: 30px;  
    /* position: relative;  
    top: 20px; left: 40px; */  
}  
  
.pos_2{  
    width: 100px;  
    height: auto;  
    background-color: aqua;  
    position: absolute;  
    top: 50px; left: 50px;  
}  
  
</style>
```

Kết quả:



Có thể thấy, do phần tử thứ hai không nằm trong phần tử chứa nào nên nó được căn lề cách 50 pixels so với cạnh trái và đỉnh của màn hình.

Trong khi đó, sau khi thiết lập thêm Position: Relative:

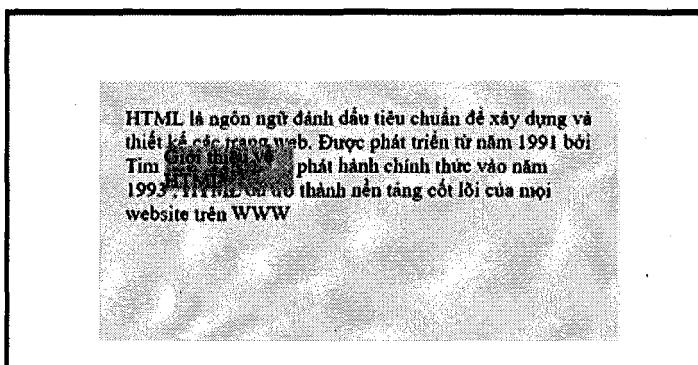
```
<div class = "pos_1">  
<div class = "pos_2">Giới thiệu về HTML</div>
```

HTML là ngôn ngữ đánh dấu tiêu chuẩn để xây dựng và thiết kế các trang Web. Được phát triển từ năm 1991 bởi Tim Berners-Lee và phát hành chính thức vào năm 1993, HTML đã trở thành nền tảng cốt lõi của mọi website trên WWW.</div>.

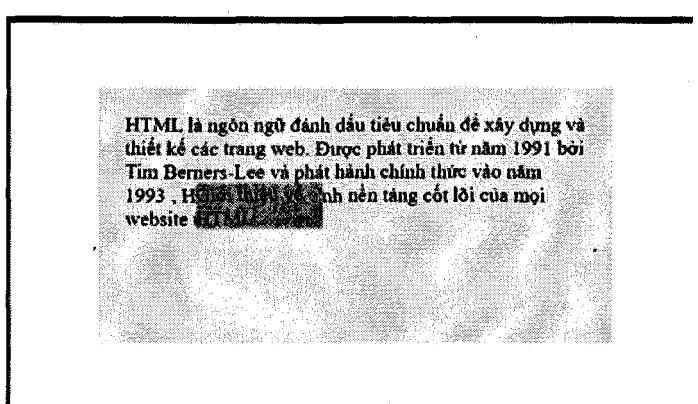
```
<style>  
body{  
border: 10px solid blue; /*Thể hiện viền màn hình*/  
}  
.pos_1{  
background-color: wheat;  
width: 400px;  
height: 200px;  
padding: 20px;  
margin: 30px;  
position: relative;
```

```
top: 20px; left: 40px;  
}  
  
.pos_2{  
width: 100px;  
height: auto;  
background-color: aqua;  
position: absolute;  
top: 50px; left: 50px;  
/*margin: 25px;*/  
}  
  
</style>
```

Kết quả:



Sau khi cho thêm margin: 25px:



Do phần tử thứ nhất đã được thiết lập Position nên phần tử thứ hai được căn lề tương ứng với cạnh của phần tử thứ nhất. Đồng thời, các thuộc tính khác của CSS như margin và padding cũng ảnh hưởng đến việc định vị và di chuyển phần tử.

Ví dụ trên cũng giải thích cho việc tại sao ở mục Position Relative bên trên có đề cập đến việc Position này thường được dùng để tạo các phần tử chứa.

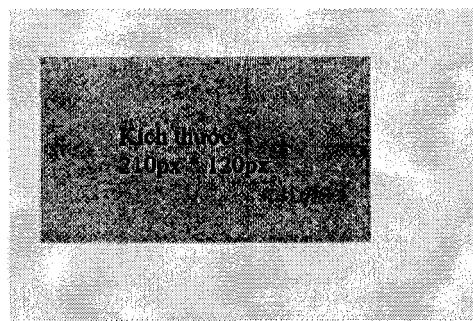
#### b) Định vị chính xác một phần tử

Các thuộc tính bù đắp sẽ được gắn một giá trị nhất định để chỉ khoảng cách cần phải di chuyển phần tử. Như bên trên đã thấy, nếu gán một giá trị dương, phần tử sẽ được đẩy xa khỏi cạnh xác định và ngược lại, một giá trị âm sẽ kéo phần tử lại gần cạnh xác định.

Khi không được gán giá trị, các thuộc tính sẽ được mặc định thiết lập ở auto. Các thuộc tính cũng có thể được sử dụng để thiết lập kích thước của một phần tử:

```
<div class = "box_1">  
  <div class = "box_2"> </div>  
</div>  
<style>  
  .box_1{  
    width: 300px;  
    height: 200px;  
    position: relative;  
    background-color: wheat;  
  }  
  .box_2{  
    position: absolute;  
    background-color: aqua;  
    top: 30px; bottom: 50px;  
    left: 20px; right: 70px;  
  }  
</style>
```

Kết quả:



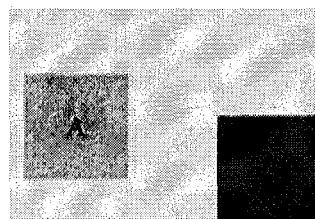
Bằng cách sử dụng các thuộc tính bù đắp, người dùng đã gián tiếp thiết lập kích thước của phần tử bên trong. Tuy nhiên, nếu phần tử đã được thiết lập chiều cao và chiều rộng sẵn thì việc lạm dụng các thuộc tính bù đắp có thể gây xung đột nội dung.

Các thuộc tính bù đắp cũng có thể sử dụng tỷ lệ phần trăm (%) để định vị. Chẳng hạn đoạn code sau:

```
<div class = "box_1">  
<div class = "box_2">A</div>  
<div class = "box_3">B</div>  
</div>  
<style>  
.box_1{  
width: 300px;  
height: 200px;  
position: relative;  
background-color: wheat;  
}  
.box_2{  
background-color: aqua;
```

```
width: 30px;  
height: 30px;  
position: absolute;  
top: 30%; left: 5%;  
}  
  
.box_3{  
background-color: fuchsia;  
width: 30px;  
height: 30px;  
position: absolute;  
bottom: 0%; right: 0%;  
}  
  
</style>
```

Kết quả:



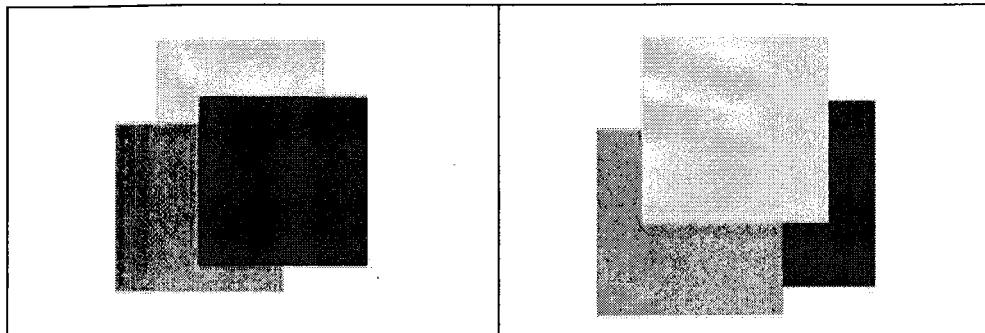
### c) Thứ tự các phần tử chồng lên nhau

Việc sử dụng các Position có thể khiến các phần tử chồng lên nhau. Theo mặc định, các phần tử sẽ chồng lên nhau theo thứ tự chúng xuất hiện trong tài liệu, phần tử xuất hiện sau sẽ chồng lên phần tử xuất hiện trước.

Để thay đổi thứ tự chúng chồng lên nhau, chúng ta sẽ sử dụng thuộc tính z-index của CSS. Z-index được gán giá trị số (âm hoặc dương), số càng lớn, thứ tự ưu tiên xuất hiện trên đỉnh sẽ càng cao.

Ví dụ:

Không sử dụng z-index	Sử dụng z-index
<pre>&lt;div class = "box_1"&gt;&lt;/div&gt; &lt;div class = "box_2"&gt;&lt;/div&gt; &lt;div class = "box_3"&gt;&lt;/div&gt; &lt;style&gt; .box_1{ width: 100px; height: 100px; position: absolute; background-color: wheat; top: 150px; left: 175px; } .box_2{ width: 100px; height: 100px; background-color: aqua; position: absolute; top: 200px;left: 150px; } .box_3{ width: 100px; height: 100px; position: absolute; background-color: fuchsia; top: 185px; left: 200px; } &lt;/style&gt;</pre>	<pre>&lt;div class = "box_1"&gt;&lt;/div&gt; &lt;div class = "box_2"&gt;&lt;/div&gt; &lt;div class = "box_3"&gt;&lt;/div&gt; &lt;style&gt; .box_1{ width: 100px; height: 100px; position: absolute; background-color: wheat; top: 150px; left: 175px; z-index: 100; } .box_2{ width: 100px; height: 100px; background-color: aqua; position: absolute; top: 200px;left: 150px; z-index: 25; } .box_3{ width: 100px; height: 100px; position: absolute; background-color: fuchsia; top: 185px; left: 200px; z-index: 1; } &lt;/style&gt;</pre>



### 3.9.3. Floating

#### 3.9.3.1. Khái niệm

Floating là kỹ thuật giúp đẩy phần tử sang bên trái hoặc bên phải của vùng hiển thị, từ đó cho phép các phần tử khác bao quanh nó. Tính năng này thường được dùng để dàn bố cục cho văn bản và hình ảnh, như khi bạn muốn hình ảnh nằm bên trái và đoạn văn nằm bên phải, trông gọn gàng và dễ nhìn hơn.

#### 3.9.3.2. Các giá trị của Floating

Floating trong CSS được cung cấp ba giá trị là left, right và none. Floating được áp dụng cho mọi phần tử và các phần tử được thiết lập mặc định là none.

Cách tốt nhất để thể hiện tác dụng của thuộc tính Float là ví dụ sau:

<b>Không có Float</b>	<b>Có Float</b>
<pre>&lt;div class = "float_demo"&gt; &lt;img src="html_logo.png" alt="logo_html_5"&gt; HTML (viết tắt của HyperText Markup Language) ra đời vào cuối thế kỷ 20, ... &lt;/div&gt; &lt;style&gt; /* Không có code */ &lt;/style&gt;</pre>	<pre>&lt;div class = "float_demo"&gt; &lt;img src="html_logo.png" alt="logo_html_5"&gt; HTML (viết tắt của HyperText Markup Language) ra đời vào cuối thế kỷ 20, ... &lt;/div&gt; &lt;style&gt; .float_demo img{ float: right; } &lt;/style&gt;</pre>

<b>Không có Float</b>	<b>Có Float</b>
 <p>HTML (viết tắt của HyperText Markup Language) ra đời vào cuối thế kỷ 20, gắn liền với sự phát triển của Internet và nhu cầu chia sẻ thông tin giữa các nhà khoa học. Vào năm 1989, tại tổ chức nghiên cứu hạt nhân châu Âu CERN, nhà khoa học máy tính Tim Berners-Lee nhận thấy việc chia sẻ tài liệu giữa các nhóm nghiên cứu gặp nhiều trở ngại do thiếu một hệ thống chung. Ông đã đề xuất một ý tưởng mang tên "Information Management: A Proposal", với mục tiêu xây dựng một hệ thống có thể liên kết các tài liệu thông qua mạng, sử dụng các siêu liên kết (hyperlink).</p>	 <p>HTML (viết tắt của HyperText Markup Language) ra đời vào cuối thế kỷ 20, gắn liền với sự phát triển của Internet và nhu cầu chia sẻ thông tin giữa các nhà khoa học. Vào năm 1989, tại tổ chức nghiên cứu hạt nhân châu Âu CERN, nhà khoa học máy tính Tim Berners-Lee nhận thấy việc chia sẻ tài liệu giữa các nhóm nghiên cứu gặp nhiều trở ngại do thiếu một hệ thống chung. Ông đã đề xuất một ý tưởng mang tên "Information Management: A Proposal", với mục tiêu xây dựng một hệ thống có thể liên kết các tài liệu thông qua mạng, sử dụng các siêu liên kết (hyperlink).</p>

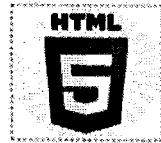
Có thể thấy khoảng trống không đẹp mắt phía trên phần văn bản đã không còn nữa. Tuy nhiên, hình ảnh logo lại đang quá sát so với phần văn bản. Để sửa vấn đề này, chúng ta có thể sử dụng thêm thuộc tính margin để chỉnh:

```
<div class = "float_demo">

HTML (viết tắt của HyperText Markup Language) ra đời vào cuối thế kỷ 20, ...
</div>
<style>
.float_demo img{
    float: right;
    margin: 10px;
}
</style>
```

Kết quả:

HTML (viết tắt của HyperText Markup Language) ra đời vào cuối thế kỷ 20, gắn liền với sự phát triển của Internet và nhu cầu chia sẻ thông tin giữa các nhà khoa học. Vào năm 1989, tại tổ chức nghiên cứu hạt nhân châu Âu CERN, nhà khoa học máy tính Tim Berners-Lee nhận thấy việc chia sẻ tài liệu giữa các nhóm nghiên cứu gặp nhiều trở ngại do thiếu một hệ thống chung. Ông đã đề xuất một ý tưởng mang tên "Information Management: A Proposal", với mục tiêu xây dựng một hệ thống có thể liên kết các tài liệu thông qua mạng, sử dụng các siêu liên kết (hyperlink).



Chúng ta có thể rút ra được những thuộc tính cơ bản sau của Float:

- Các phần tử được thiết lập thuộc tính Float hoạt động giống một vật cản giữa dòng nước. Điều này được thấy khi hình ảnh di chuyển thì phần văn bản cũng được thay đổi để bao quanh hình ảnh.

- Các phần tử được thiết lập Float chỉ di chuyển bên trong phần tử chứa.
- Các phần tử được thiết lập Float vẫn ảnh hưởng đến các phần tử khác thông qua margin.

### 3.9.3.3. Các tính chất khác của Float

#### a) Sử dụng Float với các phần tử văn bản

Ngoài việc sử dụng Float với một phần tử hình ảnh, ở mục này Float sẽ được sử dụng với các đoạn văn bản.

Ví dụ:

```
<div class = "float_text">  
    <span class = "tip"> Bạn có biết? ...</span>  
    Bánh chưng là loại bánh duy nhất có lịch sử lâu đời ...  
</div>  
  
<style>  
.float_text .tip{  
    float: right;  
    color: white;  
    background-color: lightseagreen;  
    margin: 10px;  
    padding: 10px;  
}  
</style>
```

Kết quả: Bánh chưng là loại bánh duy nhất có lịch sử lâu đời trong ẩm thực truyền thống Việt Nam còn được sử sách nhắc lại, bánh chưng có vị trí đặc biệt trong tâm thức của cộng đồng người Việt và nguồn gốc của nó về truyền thuyết liên

Bạn có biết? Chiếc bánh chưng nặng nhất thế giới nặng 1,4 tấn, được tạo bởi 50 nghệ nhân của Làng Uớc Lẽ.

quan đến hoàng tử Lang Liêu vào đời vua Hùng thứ 6. Sự tích trên muốn nhắc nhở con cháu về truyền thống của dân tộc; là lời giải thích ý nghĩa cũng như nguồn cội của Bánh Chưng, Bánh Giầy trong văn hóa, đồng thời nhấn mạnh tầm quan trọng của cây lúa và thiên nhiên trong nền văn hóa lúa nước.

Tuy nhiên, khi sử dụng Float cho các đoạn văn bản, việc làm rõ kích cỡ của phần tử là cần thiết. Nếu không, chiều rộng của phần tử sẽ được thiết lập là auto và kéo dài tương ứng với phần tử chứa của nó. Chẳng hạn, ở code bên trên, ta bỏ thuộc tính width thì sẽ được kết quả như sau:

Bạn có biết? Chiếc bánh chưng nặng nhất thế giới nặng 1,4 tấn, được tạo bởi 50 nghệ nhân của Làng Uớc Lễ.

Bánh chưng là loại bánh duy nhất có lịch sử lâu đời trong ẩm thực truyền thống Việt Nam còn được sữ sách nhắc lại, bánh chưng có vị trí đặc biệt trong tâm thức của cộng đồng người Việt và nguồn gốc của nó về truyền thuyết liên quan đến hoàng tử Lang Liêu vào đời vua Hùng thứ 6. Sự tích trên muôn nhắc nhớ con cháu về truyền thống của dân tộc; là lời giải thích ý nghĩa cũng như nguồn cội của Bánh Chung, Bánh Giầy trong văn hóa, đồng thời nhấn mạnh tầm quan trọng của cây lúa và thiên nhiên trong nền văn hóa lúa nước.

Các phần tử được thiết lập Float cũng sẽ được trình duyệt coi như một khối (block).

#### b) Sử dụng Float với các phần tử dạng khối (block)

Ở các ví dụ trên, chúng ta đã thấy việc sử dụng thuộc tính Float với các phần tử được chứa trong phần tử khác. Ở mục này, chúng ta sẽ xem tác dụng của Float trong luồng tài liệu bình thường.

Ví dụ:

<p>HTML (viết tắt của ... </p>

<p class = "float">HTML được tạo ra bởi Tim Berners-Lee nhưng về sau được W3C và WHATWG hợp tác phát triển </p>

<p>Vào năm 1989, ... </p>

<p>Ông đã đề xuất ... </p>

<p>Năm 1990, ... </p>

<style>

body p{

width: 600px;

```
height: auto;  
margin: 10px;  
padding: 10px;  
border: 2px red dashed; /*Làm nổi bật viền của phần tử*/  
}  
.float{  
float: left;  
width: 300px;  
height: auto;  
background-color: aquamarine;  
border: 2px black dashed; /*Làm nổi bật viền của phần tử*/  
}  
</style>
```

Kết quả:

HTML (viết tắt của HyperText Markup Language) ra đời vào cuối thế kỷ 20, gắn liền với sự phát triển của Internet và nhu cầu chia sẻ thông tin giữa các nhà khoa học.

Vào những năm 1989, tại tổ chức Nghiên cứu Hạt nhân châu Âu CERN, nhà khoa học máy tính Tim Berners-Lee nhận thấy việc chia sẻ tài liệu giữa các nhóm nghiên cứu gặp nhiều trở ngại do thiếu một hệ thống chung.

Ông đã đề xuất một ý tưởng mang tên “Information Management A Proposal”, với mục tiêu xây dựng một hệ thống có thể liên kết các tài liệu thông qua mạng, sử dụng các siêu liên kết (hyperlink).

Năm 1990, Berners-Lee chính thức bắt tay vào phát triển hệ thống đó và tạo ra ba thành phần chính: trình duyệt web đầu tiên có tên WorldWideWeb, đến sau đó được đổi thành World Wide Web; máy chủ web đầu tiên đặt tại CERN và đặc biệt là HTML được dùng để định dạng và trình bày nội dung trên các trang web.

Có thể thấy, đoạn văn bản thứ hai đã được di chuyển ra trái và các văn bản khác cũng được chỉnh để bao quanh phần tử trên. Tuy nhiên, phần viền của phần tử thể hiện rằng các phần tử khối vẫn chồng lên nhau như bình thường.

### c) Sử dụng Float với nhiều phần tử

Việc thiết lập Float với nhiều phần tử cùng lúc là có thể làm được, kể cả khi các phần tử đó cùng nằm bên trong một phần tử chứa khác. Đó là vì CSS được thiết kế để tự động sắp xếp các phần tử được thiết lập Float không đè lên nhau.

Ví dụ:

```
<div class = "container">  
    <p>[Đoạn 1] Lorem ipsum ... </p>  
    <p class = "float_demo">[Đ2] ....</p>  
    <p class = "float_demo">[Đ3] ...</p>  
    <p class = "float_demo">[Đ4] ...</p>  
    <p class = "float_demo">[Đ5] ...</p>  
    <p>[Đ6...</p>  
    <p>[Đ7] ...</p>  
    <p>[Đ8...</p>  
</div>  
<style>  
.container{  
    width: 950px;  
    height: auto;  
    padding: 0px;  
}  
.container p{  
    width: 100%;  
    height: auto;
```

```

padding: 10px;
}

.container .float_demo{
width: 250px;
float: left;
margin: 0;
background-color: wheat;
}

```

### Kết quả:

[Đoạn 1] Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Nam quam nunc, blandit vel, luctus pulvinar, hendrerit id, lorem

[Đ2] In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis premium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi.

[Đ3] Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Nam quam nunc.

[Đ4] Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Nam quam nunc Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum.

[Đ6] In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis premium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus.

[Đ7] Donec vitae sapien ut libero venenatis faucibus. Nullam quis ante. Etiam sit amet orci eget eros faucibus

[Đ5] Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Nam quam nunc, blandit vel, luctus pulvinar, hendrerit id, lorem. Maecenas nec odio et ante tincidunt tempus.

tincidunt. Duis leo. Sed fringilla mauris sit amet nibh. Donec sodales sagittis magna.

[Đ8] Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu

Các phần tử thiết lập float: left được sắp xếp về phía trái và trên cùng nhất có thể, nhưng sẽ xuống dòng và sắp xếp về phía bên trái nhất có thể khi không còn đủ chỗ nữa. Đồng thời, các phần tử không được thiết lập float như Đ6 cũng được sắp xếp lại để bao quanh phần vị trí không được sử dụng.

#### **3.9.3.4. Các thuộc tính khác của CSS có liên quan đến Float**

##### *a) Thuộc tính clear*

Sử dụng Float trong tình huống bình thường sẽ khiến các phần tử xung quanh nó thay đổi để bao quanh phần tử đó. Tuy nhiên, trong một số trường hợp, chúng

ta sẽ không muốn các phần tử thực hiện điều đó. Để tránh việc các văn bản khác bao quanh phần tử được thiết lập float, chúng ta sử dụng thuộc tính clear.

Clear có bốn giá trị là: left, right, both và none. Các phần tử được thiết lập mặc định là none.

Ví dụ:

<i>Không sử dụng Clear</i>	<i>Sử dụng Clear</i>
<pre>&lt;div class = "clear_demo"&gt; &lt;img src="html_logo.png" alt="html_5"&gt; &lt;p&gt;HTML ...&lt;/p&gt; &lt;h2&gt;Lịch sử hình thành HTML&lt;/h2&gt;</pre> <pre>&lt;p&gt; ... &lt;/p&gt; &lt;/div&gt; &lt;style&gt; .clear_demo img{ float: left; } &lt;/style&gt;</pre>  <p>HTML (viết tắt của HyperText Markup Language) ra đời vào cuối thế kỷ 20, gắn liền với sự phát triển của Internet và nhu cầu chia sẻ thông tin giữa các nhà khoa học.</p> <p><b>Lịch sử hình thành HTML</b>  Vào năm 1989, tại tổ chức nghiên cứu hạt nhân châu Âu CERN, nhà khoa học máy tính Tim Berners-Lee nhận thấy việc chia sẻ tài liệu giữa các nhóm nghiên cứu gặp nhiều trở ngại do thiếu một hệ thống chung</p>	<pre>&lt;div class = "clear_demo"&gt; &lt;img src="html_logo.png" alt="html_5"&gt; &lt;p&gt;...&lt;/p&gt; &lt;h2&gt;Lịch sử hình thành HTML&lt;/h2&gt;</pre> <pre>&lt;p&gt;...&lt;/p&gt; &lt;/div&gt; &lt;style&gt; .clear_demo img{ float: left; } .clear_demo h2{ clear: left; } &lt;/style&gt;</pre>  <p>HTML (viết tắt của HyperText Markup Language) ra đời vào cuối thế kỷ 20, gắn liền với sự phát triển của Internet và nhu cầu chia sẻ thông tin giữa các nhà khoa học.</p> <p><b>Lịch sử hình thành HTML</b>  Vào năm 1989, tại tổ chức nghiên cứu hạt nhân châu Âu CERN, nhà khoa học máy tính Tim Berners-Lee nhận thấy việc chia sẻ tài liệu giữa các nhóm nghiên cứu gặp nhiều trở ngại do thiếu một hệ thống chung</p>

Thuộc tính clear bắt buộc các phần tử bắt đầu ngay lập tức ở khoảng trống tiếp theo ở ngay bên dưới phần tử được thiết lập float. Giá trị left và right khiến phần tử bắt đầu ở bên trái hoặc phải, ngay dưới các phần tử được thiết lập float. Nếu có nhiều phần tử được thiết lập float, chúng ta sử dụng giá trị both để phần tử bắt đầu ở ngay bên dưới tất cả các phần tử trên.

*Chú ý:*

- Thuộc tính clear áp dụng cho phần tử người dùng muốn hiển thị bên dưới phần tử được thiết lập float, không áp dụng cho chính phần tử được thiết lập float.
- Nếu muốn thiết lập khoảng cách giữa phần tử float và phần tử clear, thiết lập margin-bottom cho phần tử float mới cho ra kết quả mong muốn thay vì sử dụng margin-top cho phần tử clear.

## CHƯƠNG IV

# JAVASCRIPT

### 4.1. GIỚI THIỆU VỀ JAVASCRIPT

Chương này sẽ giúp các bạn hiểu rõ về JavaScript – ngôn ngữ lập trình quan trọng thứ ba trong bộ ba công nghệ Web cốt lõi sau HTML và CSS. JavaScript đóng vai trò then chốt trong việc tạo ra những trang Web tương tác, sinh động và thân thiện với người dùng. Khác với HTML chịu trách nhiệm về cấu trúc nội dung và CSS xử lý giao diện, JavaScript mang đến khả năng tương tác và xử lý logic cho trang Web. Trong chương này, chúng ta sẽ tìm hiểu về bản chất của JavaScript, lịch sử phát triển, cách thức hoạt động và những ứng dụng thực tiễn quan trọng mà các bạn cần nắm vững để chuẩn bị cho kỳ thi tốt nghiệp THPT.

#### 4.1.1. JavaScript là gì?

##### 4.1.1.1. Định nghĩa và bản chất

JavaScript là một ngôn ngữ lập trình động, diễn giải (interpreted) và linh hoạt, được thiết kế chủ yếu để tạo ra các trang Web tương tác. Khác với Java mặc dù có tên gọi tương tự, JavaScript là một ngôn ngữ hoàn toàn độc lập, được tạo ra với mục đích khác biệt.

JavaScript thuộc loại ngôn ngữ loosely typed (kiểu dữ liệu linh hoạt), có nghĩa là các bạn không cần khai báo kiểu dữ liệu cụ thể cho biến. Điều này giúp việc lập trình trở nên dễ dàng hơn cho người mới bắt đầu, nhưng cũng đòi hỏi sự cẩn thận trong việc quản lý dữ liệu.

##### 4.1.1.2. Vai trò trong kiến trúc web

JavaScript đóng vai trò là "tầng hành vi" (behavioral layer) trong kiến trúc web ba tầng. Trong khi HTML cung cấp cấu trúc và nội dung, CSS đảm nhiệm việc trình

bày và thiết kế, JavaScript chịu trách nhiệm tạo ra những tương tác động và phản hồi với người dùng.

#### **4.1.1.3. Đặc điểm kỹ thuật quan trọng**

JavaScript là một ngôn ngữ lập trình phía máy khách (client-side scripting), nghĩa là mã lệnh được thực thi trực tiếp trên trình duyệt của người dùng thay vì trên máy chủ. Điều này mang lại nhiều lợi ích quan trọng cho trải nghiệm người dùng và hiệu suất của trang Web. Đầu tiên, việc xử lý mã lệnh ngay trên trình duyệt giúp phản hồi nhanh chóng đối với các thao tác của người dùng, như nhấp chuột hoặc nhập liệu mà không cần gửi yêu cầu đến máy chủ, từ đó giảm độ trễ và cải thiện tốc độ phản hồi. Thứ hai, việc xử lý dữ liệu trên máy khách giúp giảm tải cho máy chủ, vì không phải xử lý mọi yêu cầu từ người dùng, điều này đặc biệt hữu ích khi có nhiều người truy cập đồng thời. Cuối cùng, JavaScript cho phép tạo ra các hiệu ứng động và tương tác trực tiếp trên trang Web, như xác thực biểu mẫu, cập nhật nội dung mà không cần tải lại trang, mang đến trải nghiệm linh hoạt và hấp dẫn hơn cho người dùng.

### **4.1.2. Lịch sử hình thành và phát triển**

#### **4.1.2.1. Sự ra đời (1995)**

JavaScript được tạo ra bởi Brendan Eich tại Công ty Netscape vào tháng 5 năm 1995. Một sự thật thú vị là JavaScript được phát triển chỉ trong vòng 10 ngày để kịp thời hạn phát hành trình duyệt Netscape Navigator 2.0.

Ban đầu, ngôn ngữ này có tên là "Mocha", sau đó được đổi thành "LiveScript" vào tháng 9 năm 1995 và cuối cùng là "JavaScript" vào tháng 12 năm 1995 trong một thông báo chung của Netscape và Sun Microsystems.

#### **4.1.2.2. Chuẩn hóa và phát triển**

Để đảm bảo tính nhất quán và khả năng tương thích giữa các trình duyệt, JavaScript đã được chuẩn hóa dưới tên gọi ECMAScript (ECMA-262) bởi tổ chức Ecma International vào tháng 6 năm 1997. Kể từ đó, ngôn ngữ đã trải qua nhiều phiên bản nâng cấp với các tính năng mới:

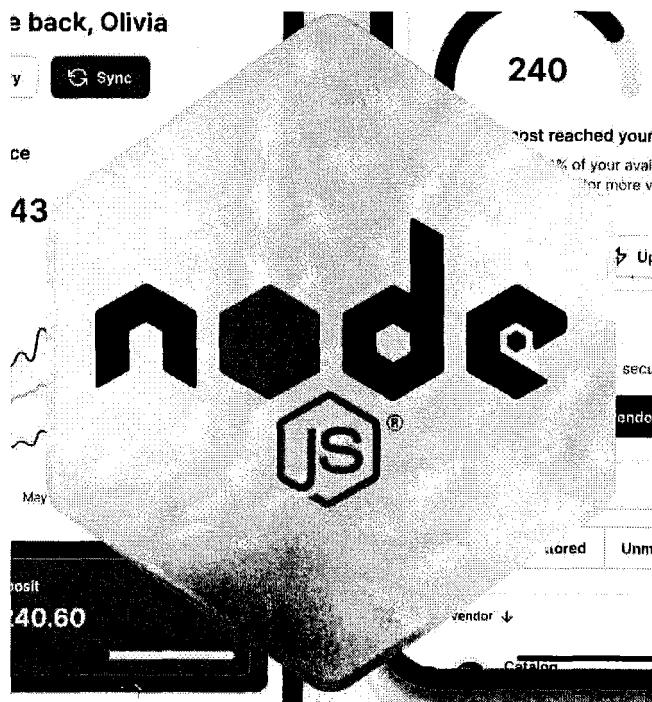
- ECMAScript 3 (ES3) – 1999: Bổ sung các tính năng như biểu thức chính quy (regular expressions), xử lý ngoại lệ với try/catch và các cải tiến về xử lý chuỗi.

- ECMAScript 5 (ES5) – 2009: Giới thiệu chế độ nghiêm ngặt ("strict mode") để kiểm tra lỗi chặt chẽ hơn, hỗ trợ JSON các phương thức mới cho mảng và chuỗi.
- ECMAScript 6 (ES6) hay ECMAScript 2015 – 2015: Được coi là một bước ngoặt lớn, ES6 bổ sung các tính năng hiện đại như khai báo biến với let và const, hàm mũi tên (arrow functions), lớp (classes) và template literals, giúp JavaScript trở nên mạnh mẽ và dễ sử dụng hơn.

Sau ES6, các phiên bản ECMAScript mới được phát hành hằng năm, tiếp tục mở rộng và cải thiện ngôn ngữ để đáp ứng nhu cầu ngày càng tăng của cộng đồng phát triển Web.

#### **4.1.2.3. Mở rộng ra ngoài trình duyệt**

Năm 2009, Node.js ra đời, cho phép JavaScript chạy trên máy chủ (server-side). Điều này đánh dấu bước ngoặt quan trọng, giúp JavaScript trở thành ngôn ngữ lập trình toàn diện có thể sử dụng cho cả front-end và back-end.



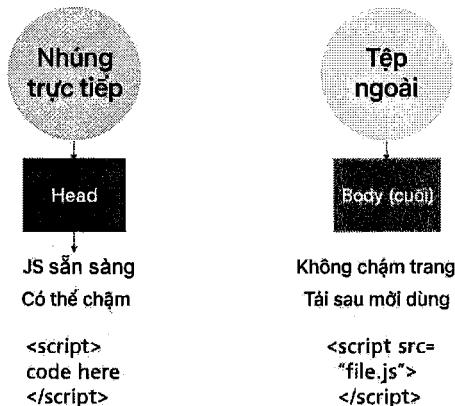
Node.js hỗ trợ phát triển nhiều ứng dụng đa dạng.

#### 4.1.3. Cách nhúng JavaScript vào trang HTML

##### 4.1.3.1. Hai phương pháp chính

## JavaScript in HTML

JS Embedding Methods



– Phương pháp 1. Nhúng trực tiếp (Inline JavaScript):

xml

```
<script>  
// Mã JavaScript được viết trực tiếp ở đây  
alert("Xin chào các bạn học sinh!");  
</script>
```

– Phương pháp 2. Liên kết từ tệp ngoài (External JavaScript):

xml

```
<script src="script.js"></script>
```

##### 4.1.3.2. Vị trí đặt script trong HTML

Trong tài liệu HTML, thẻ `<script>` có thể được đặt ở nhiều vị trí khác nhau, mỗi vị trí có những ưu điểm và hạn chế riêng:

– Trong phần `<head>`: Đặt script tại đây giúp mã JavaScript sẵn sàng khi cần sử dụng. Tuy nhiên, nếu script thực hiện các thao tác với các phần tử DOM chưa được

tải, có thể gây lỗi hoặc yêu cầu sử dụng các sự kiện như DOMContentLoaded để đảm bảo DOM đã sẵn sàng.

– Cuối phần <body> (được khuyến nghị): Đặt script tại đây giúp đảm bảo rằng toàn bộ nội dung HTML đã được tải trước khi script thực thi, giảm nguy cơ lỗi liên quan đến DOM và cải thiện tốc độ hiển thị trang.

– Sử dụng thuộc tính defer hoặc async: Khi liên kết đến các tệp JavaScript bên ngoài, có thể sử dụng thuộc tính defer để trì hoãn việc thực thi script cho đến khi toàn bộ HTML được phân tích cú pháp, hoặc async để tải và thực thi script song song với quá trình phân tích HTML. Lưu ý rằng defer đảm bảo thứ tự thực thi của các script, trong khi async không đảm bảo điều này.

#### **4.1.3.3. Thứ tự thực thi**

Trình duyệt Web thực thi các thẻ <script> theo thứ tự xuất hiện trong tài liệu HTML. Điều này có nghĩa là nếu một script phụ thuộc vào một script khác, script phụ thuộc phải được đặt sau script mà nó phụ thuộc. Việc sắp xếp thứ tự các script một cách hợp lý là rất quan trọng để đảm bảo logic hoạt động đúng và tránh lỗi trong quá trình thực thi.

### **4.1.4. Cú pháp cơ bản của JavaScript**

#### **4.1.4.1. Biến và kiểu dữ liệu**

a) Khai báo biến: var, let và const

Trong JavaScript, có ba từ khóa chính để khai báo biến:

– Var: Được sử dụng từ các phiên bản JavaScript cũ, biến khai báo bằng var có phạm vi toàn cục hoặc trong hàm.

javascript

var diem = 8;

– Let: Giới thiệu từ ES6, let cho phép khai báo biến với phạm vi khối (block scope), giúp tránh các lỗi liên quan đến phạm vi biến.

javascript

let tenHocSinh = "Nguyễn Văn A";

– Const: Cũng được giới thiệu từ ES6, const dùng để khai báo hằng số, tức là biến không thể thay đổi giá trị sau khi được gán.

javascript

```
const PI = 3.14159;
```

Lưu ý: Khi khai báo với const, bắt buộc phải gán giá trị ngay lập tức, nếu không sẽ gây lỗi.

b) Các kiểu dữ liệu cơ bản trong JavaScript

JavaScript là ngôn ngữ có kiểu dữ liệu động, nghĩa là biến có thể chứa bất kỳ kiểu dữ liệu nào và kiểu dữ liệu có thể thay đổi trong quá trình thực thi. Dưới đây là các kiểu dữ liệu cơ bản:

– String (Chuỗi): Đại diện cho chuỗi ký tự.

javascript

```
let hoTen = "Nguyễn Văn A";
```

– Number (Số): Đại diện cho cả số nguyên và số thực.

javascript

```
let diemToan = 9.5;
```

– Boolean (Logic): Chỉ có hai giá trị: true (đúng) hoặc false (sai).

javascript

```
let daNopBai = true;
```

– Null: Đại diện cho giá trị rỗng hoặc không tồn tại.

javascript

```
let duLieu = null;
```

– Undefined: Biến được khai báo nhưng chưa được gán giá trị.

javascript

```
let diemVan;
```

```
console.log(diemVan); // undefined
```

– Array (Mảng): Lưu trữ danh sách các giá trị.

javascript

```
let diemCacMon = [8, 9, 7.5];
```

– Object (Đối tượng): Lưu trữ các cặp khóa – giá trị.

javascript

```
let hocSinh = {  
    hoTen: "Nguyễn Văn A",  
    tuoi: 17,  
    lop: "12A1"  
};
```

c) *Kiểm tra kiểu dữ liệu*

Để kiểm tra kiểu dữ liệu của một biến, sử dụng toán tử `typeof`:

javascript

```
let x = 10;  
console.log(typeof x); // "number"  
let y = "Hello";  
console.log(typeof y); // "string"
```

#### 4.1.4.2 Toán tử

JavaScript hỗ trợ đầy đủ các loại toán tử:

javascript

```
// Toán tử số học  
let a = 10, b = 3;  
let cong = a + b; // 13  
let tru = a - b; // 7  
let nhan = a * b; // 30  
let chia = a / b; // 3.33...  
  
// Toán tử so sánh  
let lonHon = a > b; // true  
let bangNhau = a == b; // false
```

// Toán tử logic

```
let va = true && false; // false
```

```
let hoac = true || false; // true
```

#### **4.1.4.3. Câu lệnh điều khiển**

##### *a) Câu lệnh if*

javascript

```
let diem = 8.5;
```

```
if (diem >= 8.0) {
```

```
    console.log("Học sinh giỏi");
```

```
} else if (diem >= 6.5) {
```

```
    console.log("Học sinh khá");
```

```
} else {
```

```
    console.log("Cần cố gắng hơn");
```

```
}
```

##### *b) Vòng lặp*

javascript

```
// Vòng lặp for
```

```
for (let i = 1; i <= 5; i++) {
```

```
    console.log("Số thứ " + i);
```

```
}
```

```
// Vòng lặp while
```

```
let dem = 0;
```

```
while (dem < 3) {
```

```
    console.log("Lần thứ " + dem);
```

```
    dem++;
```

```
}
```

#### **4.1.4.4. Hàm (Functions)**

Hàm là khối mã có thể tái sử dụng để thực hiện một tác vụ cụ thể:

**javascript**

```
// Khai báo hàm  
  
function tinhTongDiem(toan, van, anh) {  
  
    return toan + van + anh;  
  
}  
  
// Gọi hàm  
  
let tongDiem = tinhTongDiem(9, 8, 7.5);  
  
console.log("Tổng điểm:" + tongDiem);
```

#### **4.1.5. Ứng dụng thực tiễn của JavaScript**

##### **4.1.5.1. Tương tác với người dùng**

JavaScript cho phép tạo ra những trải nghiệm tương tác phong phú:

- Phản hồi khi click chuột: Hiệu ứng khi di chuột qua nút bấm.
- Kiểm tra form: Xác thực dữ liệu trước khi gửi.
- Cập nhật nội dung động: Thay đổi nội dung mà không cần tải lại trang.
- Tạo menu thả xuống: Menu điều hướng tương tác.

The screenshot shows a web page titled "Form Validation". It contains two input fields: one for "Email Id" with the value "alexbond@gmail.com" and another for "Password" with the value "\*\*\*\*\*". To the right of the password field is a red exclamation mark icon. Below the password field, there is an error message: "Please enter correct password". At the bottom of the form is a "Submit" button.

Một ví dụ về xác thực biểu mẫu phía máy khách bằng JavaScript, minh họa thông báo lỗi cho mật khẩu không chính xác.

#### **4.1.5.2. Xử lý sự kiện**

JavaScript có thể phản hồi với nhiều loại sự kiện khác nhau:

javascript

// Xử lý khi nhấn nút

```
document.getElementById("myButton").addEventListener("click", function() {  
    alert("Bạn đã nhấn nút!");  
});
```

// Xử lý khi gửi form

```
document.getElementById("myForm").addEventListener("submit", function(event) {  
    // Kiểm tra dữ liệu form  
  
    if (!validateForm()) {  
  
        event.preventDefault(); // Ngăn gửi form nếu không hợp lệ  
  
    }  
  
});
```

#### **4.1.5.3. Thao tác DOM (Document Object Model)**

JavaScript có thể thay đổi cấu trúc và nội dung của trang Web:

javascript

// Thay đổi nội dung

```
document.getElementById("title").innerHTML = "Tiêu đề mới";
```

// Thay đổi kiểu dáng

```
document.getElementById("myDiv").style.color = "red";
```

// Thêm phần tử mới

```
let newElement = document.createElement("p");
```

```
newElement.innerHTML = "Đoạn văn mới";
```

```
document.body.appendChild(newElement);
```

#### **4.1.5.4. Ứng dụng trong thực tế**

JavaScript được sử dụng rộng rãi trong nhiều lĩnh vực:

- Trang web thương mại điện tử: Giỏ hàng, thanh toán trực tuyến.
- Mạng xã hội: Cập nhật feed, chat trực tuyến.
- Ứng dụng giáo dục: Bài tập tương tác, quiz trực tuyến.
- Game trên web: Trò chơi HTML5 Canvas.
- Ứng dụng di động: Thông qua các framework như React Native.

#### **4.1.5.5. Không tối ưu hóa hiệu suất**

JavaScript chạy trên máy của người dùng, vì vậy cần chú ý đến hiệu suất:

- Tránh vòng lặp quá phức tạp.
- Giảm thiểu thao tác DOM không cần thiết.
- Sử dụng cache khi có thể.

## **4.2. KIỂU DỮ LIỆU TRONG JAVASCRIPT**

### **4.2.1. Tổng quan về kiểu dữ liệu trong JavaScript**

Trong quá trình lập trình với JavaScript, bạn sẽ thường xuyên làm việc với dữ liệu, có thể là con số, văn bản, danh sách hoặc đối tượng. Mỗi loại dữ liệu đều thuộc một kiểu dữ liệu cụ thể, việc hiểu rõ những kiểu này là điều vô cùng quan trọng để viết mã đúng và hiệu quả.

#### **4.2.1.1. JavaScript là ngôn ngữ “động”**

Điểm đặc biệt của JavaScript là kiểu dữ liệu không cần khai báo trước khi sử dụng. Khác với một số ngôn ngữ lập trình như C hoặc Java – nơi bạn phải khai báo rõ ràng kiểu của biến (ví dụ: int, float, string, v.v.) thì trong JavaScript, bạn chỉ cần dùng từ khóa let, const hoặc var và gán giá trị – trình thông dịch sẽ tự động suy luận kiểu dữ liệu dựa trên giá trị đó.

Ví dụ:

```
let age = 25; // JavaScript hiểu age là kiểu Number.
```

```
let name = "Anna"; // JavaScript hiểu name là kiểu String.
```

Điều này mang lại sự linh hoạt cao, nhưng cũng dễ gây nhầm lẫn nếu bạn không nắm rõ cách mà JavaScript xác định và xử lý kiểu dữ liệu.

#### **4.2.1.2. Hai nhóm kiểu dữ liệu chính**

Các kiểu dữ liệu trong JavaScript được chia làm hai nhóm lớn:

##### *a) Kiểu nguyên thủy (Primitive Types)*

Đây là những kiểu đơn giản nhất, không có cấu trúc phức tạp và không thể thay đổi nội dung bên trong sau khi đã gán. Bao gồm:

- Number: Số (cả nguyên và thực).
- String: Chuỗi ký tự.
- Boolean: Giá trị đúng/sai.
- Null: Thể hiện không có gì cả.
- Undefined: Biến chưa được gán giá trị.
- Symbol: Kiểu dữ liệu đặc biệt, duy nhất (giới thiệu từ ES6).
- BigInt: Dùng để biểu diễn các số nguyên rất lớn (giới thiệu từ ES2020).

##### *b) Kiểu phức hợp (Object Types)*

Các kiểu này có thể chứa nhiều giá trị và cấu trúc dữ liệu khác nhau. Chúng linh hoạt và mạnh mẽ hơn kiểu nguyên thủy. Bao gồm:

- Object: Kiểu dữ liệu chung cho các đối tượng.
- Array: Mảng – tập hợp các phần tử theo thứ tự.
- Function: Hàm – đoạn mã có thể gọi lại.
- Date, RegExp, v.v..

#### **4.2.1.3. Tại sao cần hiểu rõ kiểu dữ liệu?**

Biết chính xác kiểu dữ liệu bạn đang làm việc giúp:

- Tránh lỗi logic và lỗi runtime.
- Sử dụng đúng phương pháp và thao tác với dữ liệu.
- Viết mã dễ đọc và bảo trì hơn.
- Tối ưu hiệu năng khi xử lý các cấu trúc lớn.

Trong các phần tiếp theo, chúng ta sẽ đi sâu vào từng kiểu dữ liệu, từ Number, String, đến Object và minh họa cách chúng hoạt động trong thực tế. Đây là bước đầu tiên cực kỳ quan trọng để bạn làm chủ ngôn ngữ JavaScript một cách bài bản.

### 4.2.2. Kiểu dữ liệu Number trong JavaScript

#### 4.2.2.1. Tổng quan

Trong JavaScript, Number là kiểu dữ liệu dùng để biểu diễn tất cả các giá trị số: số nguyên và số thực (số thập phân). Không giống như một số ngôn ngữ khác (như C, Java), JavaScript không phân biệt giữa int, float hay double. Mọi số đều là kiểu Number.

JavaScript

```
let age = 25; // Số nguyên
```

```
let price = 19.99; // Số thực
```

```
let temperature = -10; // Số nguyên âm
```

let bigNumber = 12345678901234567890n; // Ví dụ về BigInt (sẽ được giới thiệu sau), nhưng cần phân biệt với Number.

Tất cả các giá trị trên (ngoại trừ bigNumber nếu bạn có đề cập đến BigInt ở phần khác) đều thuộc kiểu Number.

#### 4.2.2.2. Operator (Toán tử với Number)

Các toán tử cơ bản với kiểu số bao gồm:

Toán tử	Ý nghĩa	Ví dụ
+	Cộng	5 + 2 // 7
-	Trừ	5 - 2 // 3
*	Nhân	5 * 2 // 10
/	Chia	5 / 2 // 2.5
%	Chia lấy dư	5 % 2 // 1
**	Lũy thừa (ES6)	2 ** 3 // 8
++	Tăng lên 1 đơn vị	a++ hoặc ++a
--	Giảm đi 1 đơn vị	a-- hoặc --a

Ví dụ:

JavaScript

```
let x = 10;
```

$x += 5; // x = x + 5 \rightarrow x = 15$  (Toán tử gán kết hợp)

```
let y = 7;
```

$y -= 2; // y = y - 2 \rightarrow y = 5$

```
let a = 5;
```

`console.log(a++); // 5 (sử dụng giá trị hiện tại của 'a' rồi mới tăng)`

```
console.log(a); // 6
```

```
let b = 5;
```

`console.log(++b); // 6 (tăng 'b' trước rồi mới sử dụng giá trị mới)`

```
console.log(b); // 6
```

### 4.2.3. Kiểu dữ liệu Boolean trong JavaScript

#### 4.2.3.1. Tổng quan

Kiểu dữ liệu Boolean là một trong những kiểu dữ liệu nguyên thủy (primitive data types) cơ bản nhất trong JavaScript, chỉ có hai giá trị duy nhất:

- True;

- False.

Các giá trị Boolean đóng vai trò cốt lõi trong logic điều khiển luồng chương trình, thường được sử dụng trong các phép so sánh hoặc các câu lệnh điều kiện như if, while, for, v.v..

Ví dụ:

```
let isPass = true;
```

```
let isLate = false;
```

```
if (isPass) {
```

```
    console.log("Bạn đã đậu kỳ thi!");
```

```
} else {
```

```
    console.log("Bạn thi rớt.");
}
```

*Giải thích:*

Biến isPass có giá trị true, nghĩa là học sinh đã đậu. Câu lệnh if kiểm tra điều này và in ra "Bạn đã đậu kỳ thi!". Nếu isPass là false, chương trình sẽ in "Bạn thi rớt..

#### **4.2.3.2. Truthy và Falsy**

Một trong những khái niệm quan trọng và thường xuyên gây nhầm lẫn nhất trong JavaScript là Truthy và Falsy. JavaScript có khả năng tự động chuyển đổi nhiều kiểu dữ liệu khác sang true hoặc false khi chúng được sử dụng trong ngữ cảnh Boolean (ví dụ: trong câu lệnh if, while, hoặc các toán tử logic).

Đây chính là lý do bạn thường thấy code JavaScript kiểm tra điều kiện một cách "ngắn gọn" mà không cần so sánh tường minh với true hay false.

##### *a) Truthy (được xem là true)*

Các giá trị sau được xem là true khi JavaScript thực hiện ép kiểu ngầm định sang Boolean:

- Chuỗi không rỗng: "hello", "false", "" (chuỗi có chứa khoảng trắng cũng là truthy bởi vì bất kỳ chuỗi nào, miễn là không phải chuỗi rỗng "", đều được coi là true).
- Số khác 0: 1, -1, 3.14, 0.0001 (ngoại trừ 0 và -0, tất cả các số khác đều là true).
- Mảng hoặc đối tượng (kể cả rỗng): [], {}, function() {}.
- Infinity, -Infinity: Các giá trị vô cùng.
- Date object, RegExp object.

##### *b) Falsy (được xem là false)*

Đây là danh sách các giá trị "đặc biệt" và rất quan trọng cần ghi nhớ, vì chúng được coi là false khi ép kiểu ngầm định:

Giá trị	Giải thích
false	Giá trị boolean false nguyên thủy
0, -0	Số không (cả dương 0 và âm 0)
""	Chuỗi rỗng

null	Giá trị đặc biệt chỉ ra “không có giá trị” hoặc “không tồn tại”
undefined	Biến đã được khai báo nhưng chưa được gán giá trị
NaN	Not-a-Number (không phải là số hợp lệ)

– Sử dụng toán tử !! để ép kiểu nhanh: Đây là một thủ thuật phổ biến trong JavaScript để chuyển đổi bất kỳ giá trị nào thành giá trị boolean tương ứng của nó (true/false) một cách ngắn gọn.

! là toán tử phủ định (NOT), chuyển đổi một giá trị sang boolean rồi đảo ngược.

!! là hai lần phủ định, sẽ trả về giá trị boolean “gốc” của nó.

#### 4.2.4. Kiểu dữ liệu String trong JavaScript

##### 4.2.4.1. Tổng quan

String là kiểu dữ liệu dùng để lưu trữ và thao tác với văn bản (chuỗi ký tự) trong JavaScript. Một chuỗi có thể được tạo bằng cách đặt nội dung trong:

- Dấu nháy đơn (...).
- Dấu nháy kép (...).
- Backticks (dấu huyền) (`...) – hay còn gọi là Template Literals.

Việc chọn loại dấu nháy thường phụ thuộc vào nội dung bên trong để tránh xung đột và làm code dễ đọc hơn:

Ví dụ:

```
let s1 = "I'm learning JavaScript"; // Dùng nháy kép để chứa nháy đơn.
```

```
let s2 = 'He said "Hello!"'; // Dùng nháy đơn để chứa nháy kép.
```

```
let s3 = `This is a "template" literal with 'quotes.'`; // Template literals linh hoạt hơn.
```

*Đặc điểm quan trọng của chuỗi trong JavaScript:*

- Immutable (bất biến):

Đây là một khái niệm cực kỳ quan trọng. Khi một chuỗi được tạo ra, bạn không thể thay đổi trực tiếp nội dung của chuỗi đó. Nếu bạn thực hiện bất kỳ thao tác nào dường như làm “thay đổi” chuỗi (ví dụ: nối chuỗi, thay thế), thực chất JavaScript sẽ

tạo ra một chuỗi mới với nội dung đã thay đổi và trả về chuỗi mới đó. Chuỗi gốc vẫn không bị ảnh hưởng.

– Hoạt động như mảng ký tự (Array-like object):

Một chuỗi có thể được coi như một mảng các ký tự, với chỉ số bắt đầu từ 0. Bạn có thể truy cập từng ký tự bằng chỉ số.

Ví dụ:

```
let s = "JavaScript";
console.log(s[0]); // "J" (truy cập ký tự đầu tiên)
console.log(s[4]); // "S"
console.log(s[s.length - 1]); // "t" (ký tự cuối cùng)
```

Nếu truy cập một chỉ số vượt quá độ dài chuỗi, kết quả là undefined, nhưng không gây lỗi và chương trình vẫn tiếp tục chạy.

#### 4.2.4.2. Escape Character (ký tự thoát)

Một số ký tự đặc biệt không thể viết trực tiếp vào chuỗi (vì chúng có thể xung đột với cú pháp chuỗi hoặc có ý nghĩa điều khiển) và cần dùng ký tự thoát (\) đứng trước để biểu diễn chúng.

Ký tự thoát	Ý nghĩa	Ví dụ
\"	Dấu nháy kép	"He said \\\"Hello!\\\""
'	Dấu nháy đơn	'I\\'m learning'
\\"	Dấu gạch chéo ngược (\")	"C:\\Program Files"
\n	Xuống dòng mới	"Line 1\\nLine 2"
\t	Ký tự Tab	"Column1\\tColumn2"
\r	Xuống đầu dòng (carriage return)	Không dùng, thường đi kèm với \n (\r\n)

Ví dụ:

JavaScript

```
let path = "C:\\Users\\Public\\Documents"; // Để biểu diễn dấu gạch chéo ngược
console.log(path); // C:\\Users\\Public\\Documents
```

```
let poem = "Twinkle, twinkle, little star,\n\tHow I wonder what you are.";  
console.log(poem);  
/* Output:  
Twinkle, twinkle, little star,  
How I wonder what you are.  
*/
```

#### **4.2.4.3. Các phương thức làm việc với String**

##### *a) Length Property (độ dài chuỗi)*

Thuộc tính length trả về số lượng ký tự trong chuỗi.

Ví dụ:

```
let s = "Hello";  
console.log(s.length); // 5 (bao gồm cả khoảng trắng nếu có)  
let emptyString = "";  
console.log(emptyString.length); // 0
```

##### *b) Nối chuỗi*

Dùng dấu + để ghép nhiều chuỗi lại với nhau.

Ví dụ:

```
let s1 = "Hello", s2 = "world";  
let s3 = s1 + " " + s2; // "Hello world"
```

##### *c) Chuyển chữ hoa/thường*

- toUpperCase(): Chuyển chuỗi thành chữ hoa.
- toLowerCase(): Chuyển chuỗi thành chữ thường.

Ví dụ:

```
let s = "Hello JavaScript";  
console.log(s.toUpperCase()); // "HELLO JAVASCRIPT"  
console.log(s.toLowerCase()); // "hello javascript"
```

Lưu ý: Các hàm này trả về chuỗi mới, không làm thay đổi chuỗi gốc.

d) *Tìm vị trí chuỗi con*

- `indexOf()`: Tìm vị trí xuất hiện đầu tiên của chuỗi con.
- `lastIndexOf()`: Tìm vị trí cuối cùng.
- `search()`: Tìm bằng biểu thức chính quy (regex).

Ví dụ:

```
let s = "Anh yeu em va yeu JS";
console.log(s.indexOf("yeu")); // 4
console.log(s.lastIndexOf("yeu")); // 14
console.log(s.search("em")); // 8
```

#### 4.2.5. Kiểu dữ liệu Array trong JavaScript

##### 4.2.5.1. Tổng quan

Array (mảng) là một trong những cấu trúc dữ liệu cơ bản và mạnh mẽ trong JavaScript, cho phép bạn lưu trữ một tập hợp nhiều giá trị trong một biến duy nhất. Khai báo array bằng cách gọi tên mảng và các giá trị trong cặp [], chúng được ngăn cách nhau bởi dấu phẩy. Các phần tử trong mảng được sắp xếp theo một thứ tự nhất định và được truy cập thông qua chỉ số (index), bắt đầu từ 0.

Ví dụ:

```
let fruits = ["Apple", "Banana", "Cherry"];
console.log(fruits[0]); // "Apple" – Truy cập phần tử đầu tiên
console.log(fruits[1]); // "Banana" – Truy cập phần tử thứ hai
console.log(fruits[2]); // "Cherry" – Truy cập phần tử thứ ba
console.log(fruits[3]); // undefined – Chỉ số không tồn tại
```

Đặc điểm quan trọng của mảng trong JavaScript:

a) *Tính hỗn hợp*: Mỗi phần tử trong mảng có thể thuộc bất kỳ kiểu dữ liệu nào, thậm chí là một mảng khác (tạo thành mảng đa chiều) hoặc một đối tượng. Điều này mang lại sự linh hoạt rất cao.

*b) Kích thước động:* Mảng trong JavaScript có thể thay đổi kích thước linh hoạt (co giãn) trong quá trình chạy chương trình. Bạn không cần phải khai báo trước kích thước của mảng.

*c) Mảng là đối tượng:* Mặc dù được sử dụng như một cấu trúc dữ liệu danh sách, Array trong JavaScript thực chất là một loại đối tượng đặc biệt. `typeof []` sẽ trả về "object". Điều này giải thích tại sao mảng có nhiều thuộc tính và phương thức tích hợp sẵn.

#### **4.2.5.2. Các thao tác cơ bản trong array**

##### *a) Length Property (thuộc tính độ dài)*

Thuộc tính `length` cho biết số lượng phần tử hiện có trong mảng. Đây là thuộc tính có thể đọc và ghi.

Ví dụ:

```
let colors = ["red", "green", "blue"];
console.log(colors.length); // 3

// Bạn có thể thay đổi length để cắt ngắn hoặc kéo dài mảng
colors.length = 2; // Cắt ngắn mảng, xóa phần tử cuối
console.log(colors); // ["red", "green"]

colors.length = 5; // Kéo dài mảng, các vị trí mới sẽ là undefined
console.log(colors); // ["red", "green", undefined, undefined, undefined]
```

*Lưu ý:* Thay đổi `length` để kéo dài mảng sẽ tạo ra các "lỗ hổng" chứa `undefined`, nhưng không thực sự thêm phần tử.

##### *b) Push() và Unshift() (thêm phần tử)*

– `Push()`: Thêm một hoặc nhiều phần tử vào cuối mảng và trả về độ dài mới của mảng.

– `Unshift()`: Thêm một hoặc nhiều phần tử vào đầu mảng và trả về độ dài mới của mảng. Các phần tử hiện có sẽ bị dịch chuyển về sau.

Ví dụ:

```
let numbers = [2, 3];
```

```
numbers.push(4); // numbers bây giờ là [2, 3, 4]
console.log(numbers.push(5, 6)); // 6 (độ dài mới), numbers là [2, 3, 4, 5, 6]
numbers.unshift(1); // numbers bây giờ là [1, 2, 3, 4, 5, 6]
console.log(numbers.unshift(0, -1)); // 8 (độ dài mới), numbers là [-1, 0, 1, 2, 3,
4, 5, 6]
```

*Lưu ý:* Unshift() có thể tốn kém về hiệu suất với mảng lớn vì nó phải di chuyển tất cả các phần tử hiện có.

c) *Pop()* và *Shift()* (xóa phần tử)

- Pop(): Xóa phần tử cuối cùng của mảng và trả về phần tử đã bị xóa.
- Shift(): Xóa phần tử đầu tiên của mảng và trả về phần tử đã bị xóa. Các phần tử còn lại sẽ bị dịch chuyển về trước.

Ví dụ:

```
let numbers = [1, 2, 3, 4];
let lastElement = numbers.pop(); // lastElement là 4, numbers bây giờ là [1, 2, 3]
console.log(numbers); // [1, 2, 3]
let firstElement = numbers.shift(); // firstElement là 1, numbers bây giờ là [2, 3]
console.log(numbers); // [2, 3]
```

*Lưu ý:* Tương tự như unshift(), shift() cũng có thể ảnh hưởng đến hiệu suất với mảng lớn.

d) Xóa phần tử bằng *delete*

Toán tử delete có thể dùng để xóa một phần tử tại một chỉ số cụ thể, nhưng nó sẽ để lại một "lỗ hổng" (empty item hoặc undefined) tại vị trí đó và không thay đổi thuộc tính length của mảng. Điều này thường dẫn đến các hành vi không mong muốn.

Ví dụ:

```
let numbers = [10, 20, 30];
delete numbers[1]; // numbers bây giờ là [10, <empty>, 30]
console.log(numbers); // [10, <empty>, 30]
```

```
console.log(numbers.length); // 3 (độ dài không đổi)  
console.log(numbers[1]); // undefined (giá trị tại vị trí bị xóa)
```

### e) Xóa toàn bộ mảng

Có nhiều cách để xóa toàn bộ nội dung của một mảng:

- Gán lại mảng rỗng: Cách nhanh và đơn giản nhất, nhưng nếu mảng ban đầu có nhiều tham chiếu, các tham chiếu khác vẫn sẽ trỏ đến mảng cũ (không bị xóa).

Ví dụ:

```
let numbers = [1, 2, 3];  
numbers = []; // ✓ Cách nhanh nhất để làm rỗng mảng.  
console.log(numbers); // []
```

- Đặt length = 0: Xóa tất cả phần tử và cắt ngắn mảng. Cách này hiệu quả và ảnh hưởng đến tất cả các tham chiếu đến mảng đó.

Ví dụ:

```
let numbers = [1, 2, 3];  
numbers.length = 0;  
console.log(numbers); // [](tham chiếu trỏ đến mảng rỗng)
```

### 4.2.5.3. Sắp xếp và tìm giá trị

#### a) Sort() – sắp xếp mảng

Phương thức sort() giúp chúng ta sắp xếp lại các phần tử của mảng và trả về giá trị trong mảng đã được sắp xếp theo trật tự nhất định.

- Sắp xếp mặc định (theo thứ tự chuỗi): Sort() sẽ tự động sắp xếp các phần tử, chuyển chúng thành chuỗi và sắp xếp chúng theo thứ tự từ điển. Do đó, điều này thường không trả về kết quả mong muốn với số, nên chúng ta phải sử dụng phương pháp numeric sort ở phần bên dưới.

Ví dụ:

```
let arr = [3, 1, 4, 10, 2];  
arr.sort(); // [1, 10, 2, 3, 4] (Sai với số vì "10" đứng trước "2")  
console.log(arr);
```

- Numeric sort hay sort mảng số: Để sắp xếp đúng theo thứ tự số, ta cần truyền thêm một hàm so sánh (compare function):

Ví dụ:

```
let numbers = [1, 10, 2, 5];
numbers.sort(function(a, b) {return a - b;});
console.log(numbers); // [1, 2, 5, 10]
```

a – b giúp sắp xếp tăng dần, còn b – a giúp sắp xếp giảm dần.

b) *Tìm giá trị lớn nhất/nhỏ nhất*

Có nhiều cách để tìm giá trị lớn nhất/nhỏ nhất trong một mảng số.

Sử dụng Math.max() và Math.min() với toán tử Spread (...): Cách hiệu quả và ngắn gọn nhất cho mảng số.

Ví dụ:

```
let arrNumbers = [10, 5, 8, 12, 3];
let max = Math.max(...arrNumbers); // 12
let min = Math.min(...arrNumbers); // 3
console.log("Max:", max, "Min:", min); lúc bạn cần {} và return rõ ràng.
```

## 4.3. JAVASCRIPT FUNCTION

### 4.3.1. Hàm (Function) là gì trong JavaScript?

Trong lập trình, hàm là một khối mã được định nghĩa sẵn để thực hiện một hoặc nhiều nhiệm vụ cụ thể. Mục tiêu chính khi sử dụng hàm là tái sử dụng mã, giảm trùng lặp và giúp mã trở nên gọn gàng, ngắn gọn, dễ bảo trì hơn.

Ví dụ:

```
function tinhTong(a, b) {
    return a + b;
}
console.log(tinhTong(5, 3)); // Kết quả: 8
```

Ở ví dụ trên, thay vì viết `5 + 3` mỗi lần, ta định nghĩa hàm `tinhTong()` để dùng lại nhiều lần khi cần tính tổng hai số.

### 4.3.2. Các cách khai báo hàm

#### 4.3.2.1. Function Declaration (*khai báo chuẩn*)

Đây là cách khai báo hàm truyền thống trong JavaScript:

```
function xinChao() {  
    console.log("Xin chào bạn!");  
}
```

Cách này cho phép hoisting, nghĩa là bạn có thể gọi hàm trước cả khi nó được định nghĩa trong mã.

```
xinChao(); // Vẫn hoạt động
```

```
function xinChao() {  
    console.log("Xin chào bạn!");  
}
```

#### 4.3.2.2. Function Expression (*hàm gán vào biến*)

Hàm cũng có thể được gán vào một biến như một giá trị:

```
const xinChao = function() {  
    console.log("Xin chào!");  
};
```

Ở đây, bạn không thể gọi hàm trước khi nó được gán vì nó không được hoisted.

#### 4.3.2.3. Arrow Function (*hàm mũi tên – ES6*)

Arrow Function là cú pháp ngắn gọn hơn, thường dùng trong các trường hợp đơn giản hoặc làm callback.

```
const xinChao = () => {  
    console.log("Xin chào!");  
};
```

Cần lưu ý rằng khi gọi hàm bằng cú pháp mũi tên thì hàm sẽ không được coi là một object riêng, do đó, bạn không thể sử dụng những cú pháp như "This" giống các cách khác.

Ví dụ sai:

```
const person = {  
    name: "Lan",  
    sayHi: () => {  
        console.log("Hi " + this.name);  
    }  
};  
  
person.sayHi(); // ✗ Hi undefined
```

Ví dụ đúng:

```
const person = {  
    name: "Lan",  
    sayHi: function() {  
        console.log("Hi " + this.name);  
    }  
};  
  
person.sayHi(); // ✓ Hi Lan
```

#### 4.3.3. Tham số và đối số

– Tham số (Parameter):

Tham số là biến tạm thời được khai báo trong phần định nghĩa hàm, dùng để nhận giá trị truyền vào khi hàm được gọi. Tham số giúp hàm làm việc với dữ liệu linh hoạt mà không cần viết lại hàm cho mỗi trường hợp cụ thể.

– Đối số (Argument):

Đối số là một giá trị cố định được truyền vào khi ta gọi hàm để thực hiện một tác vụ nào đấy.

Ví dụ:

```
function nhanDoi(x, y) {  
    return x * y;  
}
```

// Gọi hàm với đối số 3 và 4

```
let kq = nhanDoi(3, 4);  
x và y là tham số → khai báo trong hàm.
```

3 và 4 là đối số → giá trị thực được truyền khi gọi hàm.

– Tham số mặc định (Default Parameters):

Khi không truyền đối số, ta có thể đặt giá trị mặc định:

```
function greet(name = "bạn") {  
    console.log("Xin chào, " + name);  
}  
  
greet(); // Xin chào, bạn
```

Ở đây, đối số mặc định là “bạn” nên nếu không truyền đối số khác vào thì hàm sẽ sử dụng “bạn”, còn nếu truyền một giá trị khác thì đối số “bạn” sẽ bị bỏ qua.

#### 4.3.4. Trả về giá trị với return

Hàm không chỉ in ra kết quả mà còn có thể trả về giá trị cho phần còn lại của chương trình sử dụng bằng lệnh return.

```
function binhPhuong(x) {  
    return x * x;  
}
```

```
let kq = binhPhuong(5);  
console.log(kq); // Kết quả: 25
```

Lưu ý: Sau return, mọi dòng phía dưới sẽ không được thực thi.

### **4.3.5. Phạm vi (Scope) và biến cục bộ/toàn cục**

Trong JavaScript, phạm vi (scope) là nơi bạn có thể truy cập và sử dụng một biến. Có hai loại phạm vi phổ biến:

#### **4.3.5.1. Biến toàn cục (Global Variable)**

Biến toàn cục là một biến có thể được sử dụng ở mọi nơi trong chương trình đó. Biến toàn cục thường được sử dụng cho các giá trị quan trọng, giúp tránh việc phải khai báo lại một biến nhiều lần.

Ví dụ:

```
let ten = "Lan";  
  
function xinChao() {  
    console.log("Xin chào," + ten);  
}  
  
xinChao(); // Output: Xin chào, Lan
```

Ở đây, biến "ten" được khai báo ở ngoài và có thể được sử dụng ở bất kỳ đâu chứ không chỉ trong hàm "xinChao".

#### **4.3.5.2. Biến cục bộ (Local Variable)**

Biến cục bộ là một biến được khai báo ở trong hàm và chỉ có thể sử dụng ở hàm đó hoặc trong các hàm con.

Biến cục bộ giúp chương trình trở nên gọn gàng và việc quản lý các biến trở nên dễ dàng hơn.

Ví dụ:

```
function tinhToan() {  
    let a = 5;  
    console.log(a);  
}  
  
function xinChao() {  
    let ten = "Lan"; // biến local, chỉ tồn tại bên trong hàm
```

```
console.log("Xin chào," + ten);
}

xinChao(); // Output: Xin chào, Lan

console.log(ten); // X Lỗi: ten is not defined
```

Giải thích:

let ten = "Lan"; được khai báo bên trong hàm xinChao.

Vì vậy, ten là biến cục bộ (local variable) – chỉ có thể sử dụng được bên trong hàm.

Nếu bạn cố gọi ten bên ngoài hàm, chương trình sẽ báo lỗi: ten is not defined.

#### 4.3.6. Các hàm có sẵn trong JavaScript

JavaScript cung cấp rất nhiều hàm tích hợp sẵn (built-in functions) để xử lý nhanh chóng các tác vụ phổ biến.

Dưới đây là năm hàm quan trọng thường gặp:

Tên hàm	Mục đích
parseInt()	Chuyển chuỗi thành số nguyên
isNaN()	Kiểm tra một giá trị có phải là số
setTimeout()	Chờ một khoảng thời gian rồi thực hiện
setInterval()	Lặp lại hành động mỗi khoảng thời gian
console.log()	Ghi log ra màn hình console

Ví dụ:

```
setTimeout(() => {
    console.log("Thông báo sau 2 giây");
}, 2000);
```

Trong ví dụ trên: Sau 2 giây (2000 ms), dòng thông báo sẽ hiển thị.

## **4.4. JAVASCRIPT OBJECT**

### **4.4.1. Tổng quan về Object trong JavaScript**

- Object là tập hợp các cặp “key: value”, trong đó value có thể là dữ liệu hoặc function (còn được gọi là method).
- Object giống như một thực thể: có thuộc tính (property) và hành vi (method).

### **4.4.2. Cách tạo object**

Có nhiều cách để tạo ra một object trong JS:

- Tạo single object bằng object literal (initializer);
- Tạo object bằng new và Object constructor;
- Tạo object bằng một constructor tùy chỉnh;
- Tạo từ class;
- Dùng method Object.create().

#### **4.4.2.1. Object literal (object initializer)**

Cú pháp:

```
const obj = {  
    key1: value1,  
    "key n": valueN,  
    [expr]: valueX  
};
```

Ví dụ:

```
const user = { name: "An", age: 18 };
```

Chú ý: Cách viết nhanh gọn, chỉ phù hợp khi dùng cho object nhỏ, đơn giản.

#### **4.4.2.2. Dùng new Object()**

Công thức:

```
let obj = new Object();  
obj.key = value;
```

Ví dụ:

```
const person = new Object();
person.name = "DANH";
person.age = 18;
```

#### **4.4.2.3. Constructor function**

Công thức:

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}
let john = new Person("John", 20);
```

Ví dụ:

```
function Car(brand) {
    this.brand = brand;
}
let honda = new Car("Honda");
```

*Chú ý:* Dùng để tạo nhiều object cùng loại.

#### **4.4.2.4. Dùng class**

Ví dụ:

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
}
const p = new Person("An", 18);
```

#### **4.4.2.5. Object.create (prototype)**

Công thức:

```
let obj = Object.create(prototypeObject);
```

Ví dụ:

```
let vehicle = { run: function() { console.log("Running"); } };
```

```
let car = Object.create(vehicle);
```

```
car.run(); // Running
```

### **4.4.3. Thuộc tính và method**

#### **4.4.3.1. Thuộc tính**

– Gồm key (string/identifier) và value (bất kỳ kiểu dữ liệu).

– Truy cập:

+ obj.key

+ obj["key"] – sử dụng khi key là chuỗi phức tạp hoặc biến.

– Thêm/xóa:

+ obj.newProp = value;

+ delete obj.oldProp;

#### **4.4.3.2. Method**

a) Định nghĩa

```
let john = {
    name: "John",
    greet: function() { console.log("Hello"); },
    sayHi() { console.log("Hi"); }
};
```

b) Thêm method

```
john.sayBye = function() { console.log("Bye"); };
```

c) Gọi method

```
john.greet();
john.sayHi();
john.sayBye();
```

#### **4.4.3.3. Duyệt và liệt kê property**

- for (let k in obj) – duyệt cả thuộc tính trên prototype.
- Object.keys(obj)/Object.values(obj) – trả về mảng các key hoặc value.

#### **4.4.3.4. Kiểm tra sự tồn tại của property**

- key in obj – kiểm tra cả trên prototype.
- Own Property và Prototype Property.
- Own property: thuộc tính nằm trực tiếp trên object (do chính object đó tạo ra).
- Prototype property: thuộc tính nằm trên prototype mà object thừa hưởng.

```
function A() {}

A.prototype.x = 1;
const a = new A();

a.y = 2;

console.log("x" in a); // true (trên prototype)
console.log(a.hasOwnProperty("x")); // false
console.log(a.hasOwnProperty("y")); // true
```

#### **4.4.4. Getter/Setter**

Công thức:

```
let obj = {
  _prop: "value",
  get prop() {
    return this._prop;
  }
}
```

```
},  
set prop(val) {  
    console.log("Không cho sửa!");  
}  
};
```

Ví dụ:

```
console.log(obj.prop); // Đọc -> gọi getter  
obj.prop = "new"; // Ghi -> gọi setter
```

#### 4.4.5. Phép toán với object

##### 4.4.5.1. Sao chép & gộp object

```
Object.assign(target,...sources);
```

Ví dụ:

```
let target = { a: 1 };  
let source = { b: 2 };  
Object.assign(target, source);  
console.log(target); // { a: 1, b: 2 }
```

##### 4.4.5.2. Object.is

So sánh chính xác hơn ===

```
console.log(Object.is(NaN, NaN)); // true  
console.log(Object.is(+0, -0)); // false  
console.log(+0 === -0); // true
```

##### 4.4.5.3. Đóng băng & ngăn thêm property

- Object.freeze(obj) – không thể thay đổi hoặc xóa property.
- Object.seal(obj) – không thể thêm/xóa property, nhưng vẫn có thể thay đổi giá trị hiện có.
- Object.preventExtensions(obj) – không cho phép thêm property mới.

#### **4.4.5.4. Object.keys/Object.values**

```
Object.keys(target); // ['a', 'b']
```

```
Object.values(target); // [1, 2]
```

#### **4.4.5.5. Object.defineProperty**

Property metadata

Mỗi property có ba đặc tính:

- Writable: có thể thay đổi giá trị không.
- Enumerable: có xuất hiện khi duyệt bằng for...in hoặc Object.keys không.
- Configurable: có thể xóa, thay đổi metadata không.

```
Object.defineProperty(target, 'c', {  
    value: 3,  
    writable: false,  
    enumerable: true,  
    configurable: false  
});
```

Chú ý:

- Không enumerable => không hiện trong for...in, Object.keys;
- Không configurable => không xóa bằng delete.

#### **4.4.5.6. Object.defineProperties**

Định nghĩa nhiều property cùng lúc.

```
Object.defineProperties(obj, {  
    a: {  
        value: 1,  
        writable: true  
    },  
    b: {  
        value: 2,  
        writable: true  
    }  
});
```

```
    value: 2,  
    writable: false  
}  
};
```

#### 4.4.6. Prototype và kế thừa

- Mọi function có thuộc tính .prototype. Khi dùng new, object được khởi tạo có \_\_proto\_\_ trỏ tới prototype.
- Có thể thêm method/chung thuộc tính vào .prototype để chia sẻ giữa các thể hiện.

Ví dụ:

```
function Person(name) { this.name = name; }  
  
Person.prototype.greet = function() {  
    console.log("Hi, " + this.name);  
}  
  
const a = new Person("An");  
  
a.greet(); // "Hi, An"
```

– Kế thừa với extend và super.

```
class Animal {  
  
    constructor(name) {  
        this.name = name;  
    }  
  
    speak() {  
        console.log(`${this.name} makes a sound.`);  
    }  
}  
  
class Dog extends Animal {
```

```
constructor(name, breed) {  
    super(name);  
    this.breed = breed;  
}  
  
speak() {  
    console.log(` ${this.name} barks. `);  
}  
}  
  
const d = new Dog("Max", "Labrador");  
d.speak(); // Max barks.
```

– Static method.

```
class Util {  
  
    static sayHello() {  
        console.log("Hello!");  
    }  
}  
  
Util.sayHello();
```

*Chú ý:*

- Super phải gọi đầu tiên trong constructor con.
- Static gọi qua class, không gọi qua instance.

#### 4.4.7. Ví dụ và chú ý

Ví dụ:

```
// Tạo constructor + prototype  
  
function Student(name, score) {  
    this.name = name;  
    this.score = score;
```

```
}

Student.prototype.getAvg = function() {
    return this.score / 10;
};

// Khởi tạo object
const s1 = new Student("An", 85);
console.log(s1.name); // "An"
console.log(s1.getAvg()); // 8.5

// Thêm property động
s1.age = 18;

// Duyệt thuộc tính
for (let key in s1) {
    console.log(key, ":", s1[key]);
}

// Kiểm tra tồn tại prop
console.log("age" in s1); // true
console.log(s1.hasOwnProperty("getAvg")); // false (là method trên prototype)

// Gộp object
const extra = { school: "THPT", grade: "12" };
Object.assign(s1, extra);
console.log(s1);

// Đóng băng object
Object.freeze(s1);
// s1.newProp = 123; // không hiệu lực
```

## 4.5. JAVASCRIPT DOM

### 4.5.1. DOM là gì? Tại sao nó lại quan trọng?

Khi một trình duyệt tải tệp HTML của bạn, nó không chỉ đơn thuần đọc một chuỗi văn bản. Để có thể hiểu và làm việc với cấu trúc của trang, trình duyệt sẽ phân tích cú pháp (parse) mã HTML và xây dựng một phiên bản có tổ chức của nó trong bộ nhớ. Phiên bản này chính là DOM.

DOM biểu diễn tài liệu HTML dưới dạng một cấu trúc cây (tree structure), nơi mỗi phần của tài liệu (mỗi thẻ HTML, mỗi thuộc tính, mỗi đoạn văn bản) đều trở thành một nút (node) trong cây. Cấu trúc này cho phép JavaScript, một ngôn ngữ lập trình, có thể “nhìn thấy” và “chạm vào” từng phần của trang Web một cách logic. Nếu không có DOM, JavaScript sẽ chỉ thấy HTML như một tệp văn bản dài ngoằng và không thể biết được đâu là tiêu đề, đâu là đoạn văn để thay đổi.

### 4.5.2. Cấu trúc cây DOM

Hãy xem xét một tài liệu HTML cực kỳ đơn giản và xem trình duyệt “nhìn” nó như thế nào qua cây DOM:

#### 4.5.2.1. Mã HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Trang của tôi</title>
</head>
<body>
<h1>Tiêu đề chính</h1>
<p>Một đoạn văn.</p>
</body>
</html>
```

#### **4.5.2.2. Cây DOM được tạo ra trong bộ nhớ**

- document
  - |
    - <html>
  - /\ul>  - <head> • <body>
  - //\ul>  - <title> • <h1> • <p>
  - |||

“Trang của tôi” “Tiêu đề chính” “Một đoạn văn.”

(Text Node) (Text Node) (Text Node)

Trong cây này:

- Nút gốc (Root Node): Luôn là document. Đây là điểm khởi đầu để truy cập vào mọi thứ khác.
- Nút phần tử (Element Node): Mỗi thẻ HTML như <html>, <head>, <body>, <h1>, <p> v.v. là một nút phần tử. Chúng có mối quan hệ cha – con – anh em (parent – child – sibling) rất rõ ràng. Ví dụ, <body> là con của <html> và là anh em với <head>.
- Nút văn bản (Text Node): Nội dung văn bản nằm bên trong các thẻ, như “Trang của tôi”, cũng là các nút. Đây là “những chiếc lá” ở tận cùng của cành cây.

Nhờ cấu trúc này, JavaScript có thể bắt đầu từ document, đi xuống “cành” <body>, rồi đến “nhánh” <h1> để thay đổi nội dung của nó.

#### **4.5.3. Truy xuất các phần tử DOM**

Để thay đổi một phần tử, việc đầu tiên và quan trọng nhất là phải “tìm thấy” nó trong cây DOM. Giống như khi bạn muốn nói chuyện với một người trong đám đông, bạn phải biết tên hoặc đặc điểm nhận dạng của họ. JavaScript cung cấp cho chúng ta nhiều phương thức để làm việc này.

#### **4.5.3.1. Các phương thức truyền thống**

Đây là những phương thức đã có từ lâu, hoạt động nhanh và hiệu quả cho các tác vụ cụ thể.

`Document.getElementById(id)`: Đây là cách nhanh và chính xác nhất để chọn một phần tử, giống như gọi một người bằng số căn cước công dân của họ vậy. Vì thuộc tính id là duy nhất trong một trang, phương thức này luôn trả về một và chỉ một phần tử.

Ví dụ:

```
<div id="main-content">Đây là nội dung chính.</div>
```

```
```javascript
```

```
// Yêu cầu DOM tìm phần tử có id là 'main-content'
```

```
const mainContent = document.getElementById('main-content');
```

```
console.log(mainContent.innerText); // Kết quả: Đây là nội dung chính.
```

`Document.getElementsByTagName(tagName)`: Phương thức này giống như việc bạn yêu cầu: "Hãy cho tôi tất cả những người đang mặc áo p (thẻ `<p>`)". Nó sẽ trả về một `HTMLCollection` – một bộ sưu tập giống như mảng, chứa tất cả các phần tử có tên thẻ được chỉ định.

Ví dụ:

```
<p>Đoạn 1</p>
```

```
<p>Đoạn 2</p>
```

```
```javascript
```

```
const allParagraphs = document.getElementsByTagName('p');
```

```
console.log(allParagraphs.length); // Kết quả: 2
```

```
console.log(allParagraphs[0].innerText); // Kết quả: Đoạn 1
```

`Document.getElementsByClassName(className)`: Tương tự như trên, nhưng phương thức này tìm các phần tử dựa trên tên lớp CSS (class) của chúng. "Hãy cho tôi tất cả những ai thuộc nhóm item".

Ví dụ:

```
<div class="item">Sản phẩm 1</div>
```

```
<span class="item">Sản phẩm 2</span>
const allItems = document.getElementsByClassName('item');
for (let i = 0; i < allItems.length; i++) {
  allItems[i].style.color = 'blue'; // Đổi màu tất cả sản phẩm thành xanh
}
```

Lưu ý: HTMLCollection (trả về bởi getElementsByTagName và getElementsByClassName) là một bộ sưu tập động (live collection). Hãy tưởng tượng nó như một camera an ninh đang theo dõi trực tiếp. Nếu một phần tử mới được thêm vào trang và nó khớp với tiêu chí tìm kiếm (ví dụ, một thẻ `<p>` mới được tạo), HTMLCollection sẽ tự động cập nhật và chứa cả phần tử mới đó mà bạn không cần phải chạy lại lệnh tìm kiếm.

#### **4.5.3.2. Các phương thức hiện đại (Query Selectors)**

Các phương thức này là cách mạng vì chúng cho phép chúng ta sử dụng các bộ chọn CSS (CSS selectors) mạnh mẽ và linh hoạt để tìm kiếm phần tử. Nếu bạn đã biết CSS, bạn có thể dùng ngay kiến thức đó.

`Document.querySelector(selector):` Phương thức này hoạt động như một người tìm kiếm cẩn thận. Nó sẽ duyệt qua cây DOM từ trên xuống dưới và trả về phần tử đầu tiên mà nó tìm thấy khớp với bộ chọn CSS bạn đưa ra. Nếu không tìm thấy gì, nó sẽ trả về null.

Ví dụ:

```
<div class="container">
  <p class="text">Đoạn văn đầu tiên.</p>
  <p class="text">Đoạn văn thứ hai.</p>
</div>
```javascript
// Tìm thẻ p có class 'text', nằm bên trong một thẻ div có class 'container'
const firstText = document.querySelector('div.container p.text');
console.log(firstText.innerText); // Kết quả: Đoạn văn đầu tiên.
```

`Document.querySelectorAll(selector)`: Trong khi `querySelector` dừng lại ở kết quả đầu tiên, `querySelectorAll` sẽ tiếp tục tìm kiếm và trả về tất cả các phần tử khớp với bộ chọn, dưới dạng một `NodeList`.

Ví dụ:

```
const allTexts = document.querySelectorAll('p.text');

// NodeList có phương thức forEach rất tiện lợi
allTexts.forEach(textElement => {
    console.log(textElement.innerText);

});

// Kết quả sẽ in ra:
// Đoạn văn đầu tiên.
// Đoạn văn thứ hai.
```

Lưu ý: `NodeList` (trả về bởi `querySelectorAll`) là một bộ sưu tập tĩnh (static collection). Hãy tưởng tượng nó như một bức ảnh chụp nhanh. Nó ghi lại tất cả các phần tử khớp với bộ chọn tại thời điểm bạn gọi lệnh. Nếu sau đó bạn thêm một phần tử mới vào trang, nó sẽ không xuất hiện trong `NodeList` đã được tạo trước đó. Điều này thường làm cho việc lặp qua các phần tử trở nên dễ đoán và an toàn hơn so với `HTMLCollection`.

#### **4.5.4. Thay đổi nội dung và thuộc tính**

Một khi đã kiểm soát được một phần tử, chúng ta có thể thay đổi nó theo vô số cách.

##### **4.5.4.1. Thay đổi nội dung**

– `Element.innerHTML`: Cho phép bạn lấy hoặc thiết lập toàn bộ nội dung HTML bên trong một phần tử.

Cảnh báo bảo mật: Hãy cẩn trọng khi dùng `innerHTML` với dữ liệu do người dùng nhập. Nếu một kẻ xấu nhập vào mã `<script>alert('hack!')</script>`, `innerHTML` sẽ thực thi mã đó, gây ra lỗ hổng bảo mật nghiêm trọng gọi là Cross-Site Scripting (XSS).

– Element.textContent: Lấy hoặc thiết lập nội dung văn bản thô. Nó coi mọi thứ là văn bản thuần túy, kể cả các thẻ HTML. Đây là cách nhanh và an toàn nhất để chỉ làm việc với chữ.

– Element.innerText: Tương tự.textContent, nhưng “thông minh” hơn một chút. Nó chỉ lấy nội dung văn bản “có thể nhìn thấy được” trên trang, có tính đến các quy tắc CSS như display: none.

Ví dụ so sánh:

```
<div id="demo">Đây là <b>văn bản</b> gốc.</div>
const demoDiv = document.getElementById('demo');
console.log(demoDiv.innerHTML); // Kết quả: Đây là <b>văn bản</b> gốc.
console.log(demoDiv.textContent); // Kết quả: Đây là văn bản gốc.
console.log(demoDiv.innerText); // Kết quả: Đây là văn bản gốc.
```

// Thủ thay đổi nội dung

```
demoDiv.innerHTML = '<i>Văn bản mới được in nghiêng</i>'; // Sẽ hiển thị
chữ in nghiêng.
```

```
demoDiv.textContent = '<i>Văn bản mới được in nghiêng</i>'; // Sẽ hiển thị cả
các thẻ <i> ra màn hình.
```

#### **4.5.4.2. Thao tác với thuộc tính**

Thuộc tính là những thông tin bổ sung trong thẻ mở như href của thẻ  hay src của thẻ .

– Element.getAttribute(name): Đọc giá trị của một thuộc tính.

– Element.setAttribute(name, value): Thêm hoặc thay đổi giá trị của một thuộc tính.

– Element.removeAttribute(name): Xóa bỏ một thuộc tính.

Ví dụ:

```
<a id="my-link" href="https://google.com">Đến Google</a>
const link = document.getElementById('my-link');
// Đọc thuộc tính href
console.log(link.getAttribute('href')) // Kết quả: https://google.com
```

```
// Thay đổi link sang Bing  
link.setAttribute('href', 'https://bing.com');  
  
// Thêm thuộc tính target để mở trong tab mới  
link.setAttribute('target', '_blank');
```

#### **4.5.4.3. Thao tác với lớp CSS (classList)**

Việc thay đổi class của một phần tử để áp dụng các style CSS khác nhau là một tác vụ rất phổ biến. Thay vì gán lại toàn bộ chuỗi element.className, cách làm hiện đại là sử dụng element.classList. Nó cung cấp các phương thức an toàn và tiện lợi hơn nhiều.

- Element.classList.add('className'): Thêm một lớp mới.
- Element.classList.remove('className'): Xóa một lớp.
- Element.classList.toggle('className'): "Bật/tắt" một lớp. Nếu lớp đó đang có, nó sẽ bị xóa. Nếu chưa có, nó sẽ được thêm vào.
- Element.classList.contains('className'): Kiểm tra xem một lớp có tồn tại hay không, trả về true hoặc false.

Ví dụ:

```
<p id="message" class="info">Một thông báo.</p>  
const msg = document.getElementById('message');  
msg.classList.remove('info'); // Xóa class 'info'  
msg.classList.add('success'); // Thêm class 'success'  
// Bây giờ thẻ p sẽ có class="success"  
// Nếu trong CSS có định nghĩa .success { color: green; }, chữ sẽ đổi thành  
màu xanh.
```

#### **4.5.5. Tạo, thêm và xóa phần tử**

DOM không chỉ cho phép chúng ta sửa đổi những gì có sẵn, mà còn cho phép tạo ra những phần tử hoàn toàn mới và thêm chúng vào trang hoặc xóa đi những phần tử không cần thiết.

- (1) Tạo một viền gạch mới: document.createElement(tagName).

(2) Trang trí cho viên gạch: Dùng.textContent hoặc setAttribute để thêm nội dung và thuộc tính.

(3) Gắn viên gạch vào công trình: Dùng parentElement.appendChild(newElement) để thêm phần tử mới làm con cuối cùng của một phần tử cha đã có sẵn.

Ví dụ: Tạo một danh sách các loại trái cây từ một mảng.

```
<ul id="item-list"></ul>

const fruits = ['Táo', 'Cam', 'Chuối'];

const list = document.getElementById('item-list'); // Tìm "công trình" để gắn vào

fruits.forEach(fruitName => {

    // 1. Tạo một "viên gạch" <li> mới

    const listItem = document.createElement('li');

    // 2. "Trang trí" cho nó bằng cách gán nội dung

    listItem.textContent = fruitName;

    // 3. "Gắn" nó vào danh sách <ul>

    list.appendChild(listItem);

});
```

Để xóa một phần tử, bạn có thể dùng parentElement.removeChild(childElement). Hoặc đơn giản hơn, bạn chỉ cần gọi element.remove() trên chính phần tử muốn xóa.

#### 4.5.6. Xử lý Sự kiện (Event Handling)

Sự kiện là những hành động xảy ra trong trình duyệt mà chúng ta có thể phản ứng lại. Ví dụ: người dùng nhấp chuột vào một nút, di chuột qua một hình ảnh, gõ phím trong một ô nhập liệu hoặc đơn giản là trang Web đã tải xong.

Phương pháp hiện đại và được khuyến khích nhất để "lắng nghe" và xử lý các sự kiện này là sử dụng element.addEventListener().

Cú pháp: element.addEventListener(event, function, useCapture)

– Event: Tên của sự kiện, là một chuỗi văn bản (ví dụ: 'click', 'mouseover', 'keydown').

- Function: Hàm sẽ được gọi để thực thi khi sự kiện xảy ra. Hàm này còn được gọi là “event handler” hoặc “event listener”. Nó sẽ tự động nhận một đối tượng event chứa đầy đủ thông tin về sự kiện vừa diễn ra.
- UseCapture: Một giá trị boolean (không bắt buộc, mặc định là false) để xác định giai đoạn xử lý sự kiện.

Ví dụ: Tạo một nút bấm có thể thay đổi văn bản.

```
<p id="status">Chưa có gì xảy ra.</p>
<button id="action-btn">Bấm vào tôi!</button>
const statusP = document.getElementById('status');
const button = document.getElementById('action-btn');

// Gắn một "tai nghe" vào nút bấm để "lắng nghe" sự kiện 'click'
button.addEventListener('click', function(event) {
    statusP.textContent = 'Bạn vừa bấm vào nút!';
    statusP.style.color = 'green';
    // Đối tượng 'event' là một bản báo cáo chi tiết về sự kiện
    console.log('Sự kiện xảy ra trên phần tử:', event.target); // event.target chính
    là cái nút bấm
});
```

#### **4.5.7. Event Bubbling (sùi bọt) và Capturing (bắt giữ)**

Đây là một khái niệm nâng cao nhưng rất quan trọng để hiểu cách sự kiện di truyền trong DOM. Khi một sự kiện xảy ra trên một phần tử (ví dụ, bạn click vào một nút), nó không chỉ xảy ra ở riêng nút đó.

Hãy tưởng tượng bạn ném một viên sỏi vào ao. Sóng nước sẽ lan ra. Trong DOM cũng vậy, sự kiện sẽ lan truyền qua hai giai đoạn:

1. Giai đoạn Capturing (Bắt giữ – từ ngoài vào trong): Sự kiện bắt đầu từ “bầu trời” (window), đi xuống qua các phần tử cha (<html>, <body>, <div> v.v.) cho đến khi chạm tới đúng phần tử mục tiêu (cái nút bạn click).
2. Giai đoạn Bubbling (Sùi bọt – từ trong ra ngoài): Sau khi đến được mục tiêu, sự kiện lại “nổi bọt” ngược trở lên, từ phần tử mục tiêu ra các phần tử cha, cho đến khi quay trở lại window.

Mặc định, addEventListener xử lý sự kiện trong giai đoạn Bubbling. Đây là hành vi tự nhiên và được sử dụng nhiều nhất. Nếu bạn muốn xử lý sự kiện ngay trong giai đoạn Capturing, bạn có thể đặt tham số useCapture thành true.

#### **4.5.8. Event.preventDefault() và event.stopPropagation()**

Trong hàm xử lý sự kiện, đối tượng event cung cấp hai phương thức hữu ích:

- Event.preventDefault(): Ngăn chặn hành vi mặc định của trình duyệt. Ví dụ kinh điển là khi người dùng nhấn vào một thẻ <a>, hành vi mặc định là chuyển đến trang mới. Nếu bạn gọi event.preventDefault(), việc chuyển trang sẽ bị hủy bỏ. Điều này rất hữu ích khi bạn muốn dùng thẻ <a> như một nút bấm và xử lý logic bằng JavaScript.
- Event.stopPropagation(): Ngăn sự kiện tiếp tục lan truyền trong giai đoạn bubbling hoặc capturing. Giống như bạn nói: "Chuyện này dừng ở đây thôi, đừng báo cáo lên cấp trên nữa".

Ví dụ:

```
<a id="form-link" href="/submit">Gửi Form</a>

const formLink = document.getElementById('form-link');

formLink.addEventListener('click', function(event) {
    // Ngăn chặn hành vi mặc định của thẻ a (là chuyển trang)
    event.preventDefault();
    console.log('Link đã được click nhưng trang không tải lại!');

    // Tại đây, chúng ta có thể thêm logic để kiểm tra dữ liệu form
    // trước khi quyết định có gửi đi hay không.
});
```

### **4.6. VÙN VẶT TRONG JAVASCRIPT**

#### **4.6.1. Giới thiệu**

Trong các phần trước, chúng ta đã tìm hiểu những nền tảng cơ bản của JavaScript như biến, kiểu dữ liệu, toán tử, cấu trúc điều khiển, hàm và DOM. Những công cụ đó cho phép chúng ta viết các kịch bản (script) có thể thực thi các tác vụ

đơn giản. Tuy nhiên, để xây dựng các ứng dụng Web thực sự mạnh mẽ, tương tác và đáng tin cậy, chúng ta cần trang bị thêm những công cụ chuyên sâu hơn.

Mục này sẽ giới thiệu bốn khái niệm quan trọng:

(1) Đối tượng Math: Một thư viện tích hợp sẵn cung cấp các công cụ toán học cần thiết.

(2) Xử lý lỗi (Error Handling): Kỹ thuật giúp chương trình không bị sập (crash) khi có sự cố bất ngờ xảy ra, làm tăng tính ổn định cho ứng dụng.

(3) JSON (JavaScript Object Notation): Định dạng dữ liệu tiêu chuẩn để các chương trình và hệ thống khác nhau có thể “nói chuyện” và trao đổi thông tin một cách hiệu quả.

(4) Biểu thức chính quy (Regular Expressions): Một công cụ cực kỳ mạnh mẽ để tìm kiếm, xác thực và xử lý văn bản dựa trên các mẫu (pattern) phức tạp.

Việc nắm vững các chủ đề này sẽ là một bước tiến lớn, giúp bạn chuyển từ việc viết các đoạn mã đơn giản sang việc phát triển các tính năng phức tạp và chuyên nghiệp hơn cho website của mình.

#### **4.6.2. Đối tượng Math**

Trong lập trình, chúng ta thường xuyên phải thực hiện các phép toán. Thay vì phải tự mình định nghĩa lại các hằng số quen thuộc như số Pi ( $\pi$ ) hay viết các thuật toán phức tạp để làm tròn số, tính lũy thừa, JavaScript cung cấp sẵn một “hộp công cụ” toán học được gọi là đối tượng Math.

##### **4.6.2.1. Math là một đối tượng tĩnh (static object) tích hợp sẵn**

*Giải thích:*

– Tích hợp sẵn (built-in): Nó là một phần của ngôn ngữ JavaScript. Bạn không cần phải cài đặt hay khai báo gì thêm để sử dụng nó.

– Đối tượng tĩnh (static): Đây là một điểm quan trọng. Bạn không thể và không cần tạo ra một “thể hiện” (instance) của Math bằng từ khóa new (ví dụ: let myMath = new Math(); sẽ gây ra lỗi). Thay vào đó, bạn truy cập trực tiếp vào các thuộc tính và phương thức của nó thông qua chính tên Math. Hãy hình dung Math như một chiếc hộp đựng cụ duy nhất trong xưởng của bạn. Khi cần một cái búa, bạn chỉ cần lấy Math.bua, chứ không cần tạo ra một “cái búa mới”.

#### **4.6.2.2. Các Thuộc tính (hằng số toán học)**

Thuộc tính của đối tượng Math là các hằng số toán học đã được định nghĩa sẵn với độ chính xác cao.

##### *a) Math.PI*

Trả về giá trị của số Pi ( $\pi$ ), là tỷ số giữa chu vi của một đường tròn và đường kính của nó.

- Giá trị: xấp xỉ 3.141592653589793.
- Ứng dụng: Tính chu vi hoặc diện tích hình tròn.

Ví dụ: Tính diện tích của một hình tròn có bán kính là 5.

```
const banKinh = 5;
```

```
// Công thức diện tích hình tròn: PI * r^2
```

```
const dienTich = Math.PI * banKinh * banKinh;
```

```
console.log(`Diện tích hình tròn với bán kính ${banKinh} là: ${dienTich}`);
```

```
// Kết quả: Diện tích hình tròn với bán kính 5 là: 78.53981633974483
```

##### *b) Math.E*

Trả về hằng số Euler (còn gọi là số Napier), cơ số của logarit tự nhiên.

- Giá trị: xấp xỉ 2.718281828459045.
- Ứng dụng: Thường được sử dụng trong các bài toán về tăng trưởng, tài chính (lãi kép liên tục) và các lĩnh vực khoa học khác.

#### **4.6.2.3. Các phương thức (hàm toán học)**

Phương thức là các hàm được gắn với đối tượng Math để thực hiện một phép toán cụ thể.

##### *a) Math.round(x)*

Làm tròn số x đến số nguyên gần nhất theo quy tắc làm tròn thông thường (từ .5 trở lên thì làm tròn lên, dưới .5 thì làm tròn xuống).

Ví dụ:

- Console.log(Math.round(4.7)); // Kết quả: 5;

- `Console.log(Math.round(4.4)); // Kết quả: 4;`
- `Console.log(Math.round(4.5)); // Kết quả: 5;`
- `Console.log(Math.round(-4.2)); // Kết quả: -4.`

*b) Math.ceil(x)*

Làm tròn lên (Ceiling – Trần nhà). Luôn làm tròn x đến số nguyên lớn hơn hoặc bằng nó.

Ví dụ:

- `console.log(Math.ceil(4.7)); // Kết quả: 5;`
- `console.log(Math.ceil(4.4)); // Kết quả: 5 (vẫn làm tròn lên);`
- `console.log(Math.ceil(4.0)); // Kết quả: 4 (vì đã là số nguyên);`
- `console.log(Math.ceil(-4.2)); // Kết quả: -4 (-4 lớn hơn -4.2).`

*c) Math.floor(x)*

Làm tròn xuống (Floor – Sàn nhà). Luôn làm tròn x đến số nguyên nhỏ hơn hoặc bằng nó.

Ví dụ:

- `console.log(Math.floor(4.7)); // Kết quả: 4;`
- `console.log(Math.floor(4.4)); // Kết quả: 4;`
- `console.log(Math.floor(4.99)); // Kết quả: 4 (vẫn làm tròn xuống);`
- `console.log(Math.floor(-4.2)); // Kết quả: -5 (-5 nhỏ hơn -4.2).`

*d) Math.random()*

Trả về một số thực giả ngẫu nhiên trong khoảng [0, 1), nghĩa là số này lớn hơn hoặc bằng 0 và nhỏ hơn 1.

Ví dụ:

- `Console.log(Math.random()); // Có thể là 0.12345...`
- `Console.log(Math.random()); // Có thể là 0.98765...`
- `Console.log(Math.random()); // Mỗi lần gọi sẽ ra một số khác nhau.`

e) Công thức tổng quát để sinh số nguyên trong khoảng [min, max]

Math.floor(Math.random() \* (max - min + 1)) + min

Math.max(x, y,...) và Math.min(x, y,...)

– Math.max(): Trả về số lớn nhất trong danh sách các đối số được cung cấp.

– Math.min(): Trả về số nhỏ nhất trong danh sách các đối số được cung cấp.

Ví dụ:

– console.log(Math.max(5, 10, -3, 100, 42)); // Kết quả: 100;

– console.log(Math.min(5, 10, -3, 100, 42)); // Kết quả: -3.

f) Math.pow(x, y)

Trả về kết quả của x lũy thừa y (tức là  $x^y$ ).

Ví dụ:

– Console.log(Math.pow(2, 3)); //  $2 * 2 * 2 = 8$ ;

– Console.log(Math.pow(5, 2)); //  $5 * 5 = 25$ ;

– Console.log(Math.pow(4, 0.5)); // Căn bậc hai của 4 = 2.

g) Math.sqrt(x)

Trả về căn bậc hai của x. Tương đương với Math.pow(x, 0.5).

Lưu ý: Nếu x là một số âm, kết quả sẽ là NaN (Not a Number – Không phải là số).

Ví dụ:

– Console.log(Math.sqrt(64)); // Kết quả: 8;

– Console.log(Math.sqrt(2)); // Kết quả: 1.414...;

– Console.log(Math.sqrt(-9)); // Kết quả: NaN.

h) Math.abs(x)

Trả về giá trị tuyệt đối (module) của x. Về cơ bản, nó loại bỏ dấu âm của một số.

Ví dụ:

– Console.log(Math.abs(-15)); // Kết quả: 15;

– Console.log(Math.abs(15)); // Kết quả: 15;

– Console.log(Math.abs(0)); // Kết quả: 0.

### **4.6.3. Xử lý lỗi (Error Handling)**

Khi viết chương trình, dù cẩn thận đến đâu, lỗi vẫn có thể xảy ra. Lỗi trong lập trình có thể được chia thành hai loại chính:

#### **4.6.3.1. Lỗi cú pháp (Syntax Errors)**

Là những lỗi do gõ sai “ngữ pháp” của ngôn ngữ. Ví dụ: thiếu dấu ngoặc, sai tên từ khóa. Trình duyệt sẽ phát hiện ra các lỗi này ngay lập tức và không thực thi bất kỳ dòng mã nào.

```
// Lỗi cú pháp: thiếu dấu )
```

```
console.log("Xin chào");
```

#### **4.6.3.2. Lỗi thực thi (Runtime Errors/Exceptions)**

Là những lỗi xảy ra trong khi chương trình đang chạy. Cú pháp có thể hoàn toàn đúng, nhưng một tình huống không lường trước đã xảy ra. Ví dụ: cố gắng thực hiện một phép toán không hợp lệ (chia cho 0) hoặc cố gắng truy cập một tài nguyên không tồn tại.

Nếu không được xử lý, một lỗi thực thi có thể làm cho toàn bộ kịch bản JavaScript trên trang Web ngừng hoạt động, dẫn đến trải nghiệm rất tệ cho người dùng. Xử lý lỗi là kỹ thuật để chúng ta lường trước, bắt lấy và xử lý các lỗi thực thi này một cách có kiểm soát.

Công cụ chính để xử lý lỗi trong JavaScript là cấu trúc try...catch...finally.

a) *Cấu trúc try...catch...finally*

```
try {
```

```
    // Đặt đoạn mã có nguy cơ gây lỗi ở đây.
```

```
} catch (error) {
```

```
    // Khối này sẽ được thực thi NẾU có lỗi xảy ra trong khối try.
```

```
} finally {
```

```
    // Khối này LUÔN LUÔN được thực thi, dù có lỗi hay không.
```

```
}
```

### *Giải thích:*

– Try: Bạn đặt đoạn mã mà bạn nghi ngờ có thể gây ra lỗi vào trong khối này. JavaScript sẽ cố gắng thực thi nó.

– Catch (error): Nếu bất kỳ dòng lệnh nào trong khối try gây ra lỗi, quá trình thực thi của khối try sẽ dừng lại ngay lập tức và chuyển sang khối catch. Biến error (bạn có thể đặt tên khác) là một đối tượng chứa thông tin chi tiết về lỗi đã xảy ra.

– Finally: Khối này là tùy chọn. Các lệnh trong khối finally sẽ luôn được thực thi sau khi try và catch hoàn tất, bất kể có lỗi xảy ra hay không. Nó thường được dùng cho các tác vụ “dọn dẹp”, ví dụ như đóng một kết nối mạng hoặc giải phóng bộ nhớ, để đảm bảo chúng luôn được thực hiện.

### *b) Đối tượng error*

Khi một lỗi bị “bắt” trong khối catch, đối tượng error thường có hai thuộc tính quan trọng:

- Error.name: Tên hoặc loại của lỗi (ví dụ: TypeError, ReferenceError).
- Error.message: Một chuỗi văn bản mô tả chi tiết hơn về lỗi.

### *c) Từ khóa throw – Tự tạo ra lỗi*

Đôi khi, một tình huống không phải là lỗi theo định nghĩa của JavaScript, nhưng lại là lỗi theo logic của ứng dụng. Ví dụ, JavaScript cho phép chia một số cho 0 và kết quả là Infinity, nhưng trong một ứng dụng tính toán tài chính, việc chia cho 0 có thể là một lỗi logic nghiêm trọng cần phải được ngăn chặn.

Từ khóa throw cho phép bạn tự tạo ra (ném ra) một ngoại lệ của riêng mình. Khi throw được thực thi, nó sẽ dừng hàm hiện tại và tìm đến khối catch gần nhất để xử lý.

Ví dụ: Xây dựng hàm chia an toàn.

```
function divide(numerator, denominator) {  
    console.log(`Bắt đầu thực hiện phép chia ${numerator} / ${denominator}...`);  
    try {  
        // Kiểm tra logic của ứng dụng  
        if (denominator === 0) {
```

```
// Đây không phải lỗi cú pháp, nhưng là lỗi logic.  
// Chúng ta chủ động "ném" ra một đối tượng Error mới.  
throw new Error("Lỗi logic: Không thể chia cho số không!");  
}  
  
// Dòng này chỉ thực thi nếu không có lỗi nào được ném ra  
const result = numerator / denominator;  
console.log(`Kết quả thành công: ${result}`);  
} catch (error) {  
    // Bắt lấy lỗi (dù là do hệ thống hay do chúng ta tự throw)  
    console.error("!!! Đã có sự cố xảy ra !!!");  
    console.error(`- Tên lỗi: ${error.name}`); // Tên của loại lỗi  
    console.error(`- Thông điệp: ${error.message}`); // Thông điệp chúng ta đã tạo  
} finally {  
    // Khối này luôn chạy  
    console.log("...Hoàn thành quá trình xử lý phép chia.");  
    console.log("-----");  
}  
}  
  
// Trường hợp 1: Hoạt động bình thường  
divide(10, 2);  
  
// Trường hợp 2: Gây ra lỗi logic do chúng ta định nghĩa  
divide(10, 0);  
  
// Trường hợp 3: Gây ra lỗi thực thi của JavaScript (biến không tồn tại)  
// Lưu ý: Dòng này sẽ gây lỗi ReferenceError và cũng sẽ bị bắt bởi catch  
// divide(nonExistentVariable, 5);  
Kết quả trên console:  
Bắt đầu thực hiện phép chia 10 / 2...
```

Kết quả thành công: 5

...Hoàn thành quá trình xử lý phép chia.

Bắt đầu thực hiện phép chia 10 / 0...

!!! Đã có sự cố xảy ra !!!

– Tên lỗi: Error

– Thông điệp: Lỗi logic: Không thể chia cho số không!

...Hoàn thành quá trình xử lý phép chia.

#### **4.6.4. JSON (JavaScript Object Notation)**

##### **4.6.4.1. Làm thế nào để các hệ thống khác nhau trao đổi dữ liệu?**

Hãy tưởng tượng một trang Web thương mại điện tử. Trình duyệt của bạn (client, viết bằng JavaScript) cần hiển thị thông tin sản phẩm. Thông tin này lại được lưu trữ trong một cơ sở dữ liệu trên một máy chủ (server) ở rất xa và máy chủ này có thể được viết bằng một ngôn ngữ hoàn toàn khác như Python, Java, hay PHP.

Làm thế nào để máy chủ Python có thể gửi dữ liệu về một sản phẩm cho trình duyệt JavaScript hiểu được? Chúng cần một ngôn ngữ chung. JSON chính là ngôn ngữ chung đó.

JSON (JavaScript Object Notation) là một định dạng văn bản nhẹ, dễ đọc cho người và dễ phân tích cho máy, được sử dụng để trao đổi dữ liệu. Mặc dù tên của nó có chứa "JavaScript", JSON hoàn toàn độc lập về ngôn ngữ và đã trở thành tiêu chuẩn quốc tế cho việc truyền dữ liệu trên Web, đặc biệt là qua các API (Application Programming Interface).

##### **4.6.4.2. Cú pháp của JSON**

Cú pháp của JSON rất giống với cách khai báo đối tượng trong JavaScript, nhưng có một vài quy tắc nghiêm ngặt hơn:

(1) Cặp key/value: Dữ liệu được tổ chức thành các cặp khóa (key) và giá trị (value).

(2) Khóa (Key): Bắt buộc phải là một chuỗi (string) và được bao quanh bởi dấu ngoặc kép "". Đây là điểm khác biệt lớn so với đối tượng JavaScript (nơi dấu ngoặc kép cho key thường không bắt buộc).

(3) Giá trị (Value): Có thể là một trong các kiểu dữ liệu sau:

- Chuỗi (string, trong dấu "");
- Số (number);
- Đối tượng (object, trong dấu {});
- Mảng (array, trong dấu []);
- Boolean (true hoặc false);
- Null.

(4) Phân tách: Các cặp key/value trong một đối tượng, hoặc các phần tử trong một mảng, được phân tách với nhau bằng dấu phẩy.

(5) Không cho phép: JSON không thể chứa hàm (function), undefined, hay comment.

Ví dụ về một đối tượng JSON mô tả một học sinh lớp 12:

```
{  
    "studentId": "SV001",  
    "name": "Nguyễn Văn A",  
    "age": 18,  
    "isEnrolled": true,  
    "contact": {  
        "email": "a.nguyen@example.com",  
        "phone": null  
    },  
    "courses": [  
        {"courseId": "CS101", "courseName": "Tin học 12"},  
        {"courseId": "MA203", "courseName": "Giải tích II"}  
    ]  
}
```

#### **4.6.4.3. Sử dụng JSON trong JavaScript**

JavaScript cung cấp một đối tượng tích hợp sẵn tên là JSON với hai phương thức cực kỳ quan trọng.

a) `JSON.stringify(object)`: *Chuyển đổi JavaScript Object thành chuỗi JSON*

Phương thức này nhận vào một đối tượng (hoặc mảng, giá trị) của JavaScript và chuyển đổi nó thành một chuỗi văn bản có định dạng JSON. Quá trình này được gọi là “serialization” được sử dụng trong JavaScript

Khi bạn cần gửi dữ liệu từ ứng dụng JavaScript của mình lên một máy chủ.

Ví dụ:

// 1. Đây là một đối tượng JavaScript thông thường

```
const studentObject = {  
    name: "Trần Thị B",  
    age: 21,  
    major: "Information Technology",  
    // Hàm sẽ bị bỏ qua khi chuyển đổi  
    displayInfo: function() {  
        console.log(this.name);  
    }  
};
```

// 2. Chuyển đổi đối tượng này thành một chuỗi JSON

// Tham số thứ hai (replacer) và thứ ba (space) là tùy chọn.

// Ở đây, dùng `2` để chuỗi JSON được định dạng với 2 dấu cách thụt đầu dòng cho dễ đọc.

```
const jsonString = JSON.stringify(studentObject, null, 2);  
console.log("Đối tượng JavaScript ban đầu:", studentObject);  
console.log("Chuỗi JSON sau khi chuyển đổi:");  
console.log(jsonString);
```

Kết quả trên console:

Đối tượng JavaScript ban đầu: { name: 'Trần Thị B', age: 21, major: 'Information Technology', displayInfo: [Function: displayInfo] }

Chuỗi JSON sau khi chuyển đổi:

```
"name": "Trần Thị B",
"age": 21,
"major": "Information Technology"
}
```

b) `JSON.parse(jsonString)`: Chuyển đổi chuỗi JSON thành JavaScript Object

Phương thức này làm điều ngược lại: Nó nhận vào một chuỗi văn bản có định dạng JSON và phân tích cú pháp (parse) để tạo ra một đối tượng hoặc giá trị JavaScript tương ứng. Quá trình này gọi là “deserialization”, được sử dụng khi ứng dụng JavaScript của bạn nhận được dữ liệu từ một máy chủ hoặc API.

Lưu ý: Nếu chuỗi đầu vào không phải là một chuỗi JSON hợp lệ, `JSON.parse()` sẽ ném ra một lỗi. Do đó, đây là một nơi lý tưởng để sử dụng khối `try...catch`.

Ví dụ:

```
// 1. Giả sử đây là chuỗi văn bản chúng ta nhận được từ server
const receivedJsonString = '{"id": 101, "productName": "Laptop Pro", "price": 25000000, "inStock": true}';

try {
    // 2. Phân tích chuỗi JSON để tạo ra một đối tượng JavaScript
    const productObject = JSON.parse(receivedJsonString);

    // 3. Bây giờ chúng ta có thể sử dụng nó như một đối tượng bình thường
    console.log("Đối tượng JavaScript đã được phân tích:");
    console.log(productObject);
    console.log(`Tên sản phẩm: ${productObject.productName}`); // Output: Laptop Pro
    console.log(`Giá: ${productObject.price}`); // Output: 25000000
}
```

```
    } catch (error) {  
        console.error("Không thể phân tích chuỗi JSON. Lỗi:", error.message);  
    }  
}
```

#### 4.6.5. Biểu thức chính quy (Regular Expressions – Regex)

##### 4.6.5.1. Regex là gì?

Biểu thức chính quy (Regex) là một chuỗi các ký tự tạo thành một mẫu tìm kiếm (search pattern). Nó là một ngôn ngữ mini chuyên dùng để làm việc với văn bản. Thay vì tìm kiếm một chuỗi ký tự cố định, Regex cho phép bạn tìm kiếm những chuỗi khớp với một quy tắc nào đó.

Hãy tưởng tượng bạn muốn kiểm tra xem một chuỗi người dùng nhập vào có phải là một địa chỉ email hợp lệ hay không. Bạn không thể kiểm tra chuỗi đó bằng "test@example.com" vì có vô số địa chỉ email hợp lệ, mà cần cách định nghĩa "mẫu" của một email: (một vài ký tự)@(một vài ký tự).(hai đến sáu ký tự). Regex giúp bạn làm điều đó.

Biểu thức chính quy là một chủ đề sâu rộng, nhưng việc nắm vững những kiến thức cơ bản trên sẽ cung cấp cho bạn một công cụ cực kỳ mạnh mẽ để xử lý văn bản một cách linh hoạt.

##### 4.6.5.2. Các ứng dụng chính của Regex

(1) Xác thực dữ liệu (Validation): Kiểm tra xem dữ liệu người dùng nhập (email, số điện thoại, mật khẩu, ngày tháng) có tuân theo một định dạng cụ thể hay không.

(2) Tìm kiếm (Searching): Tìm kiếm một hoặc nhiều lần xuất hiện của một mẫu trong một đoạn văn bản lớn.

(3) Thay thế (Replacing): Tìm các chuỗi con khớp với một mẫu và thay thế chúng bằng một chuỗi khác.

##### 4.6.5.3. Cách tạo một Regex trong JavaScript

Có hai cách để tạo một đối tượng Regex:

(1) Dạng chữ (Literal): Mẫu được đặt giữa hai dấu gạch chéo /. Đây là cách phổ biến và hiệu quả nhất.

```
const regex = /pattern	flags/;
```

(2) Hàm khởi tạo (Constructor): Sử dụng khi mẫu của bạn được tạo ra một cách linh động từ một biến hoặc chuỗi khác.

```
const regex = new RegExp('pattern', 'flags');
```

Các cờ (flags) phổ biến: Cờ được đặt sau dấu / cuối cùng để sửa đổi hành vi tìm kiếm:

– g (Global): Tìm kiếm tất cả các kết quả khớp trong chuỗi, thay vì dừng lại ở kết quả đầu tiên.

– i (Case-Insensitive): Tìm kiếm không phân biệt chữ hoa, chữ thường.

– m (Multiline): Cho phép các ký tự ^ và \$ (sẽ học dưới đây) khớp với đầu và cuối của mỗi dòng, thay vì chỉ đầu và cuối của toàn bộ chuỗi.

#### 4.6.5.4. Các khái niệm cơ bản của một mẫu Regex

Ký tự	Mô tả	Ví dụ
Ký tự thường	a, b, 1, 2	/hello/ khớp với "hello"
.	Khớp với bất kỳ ký tự nào (trừ dòng mới)	/h.t/ khớp với "hat", "hot", "h8t"
\d	Khớp với bất kỳ chữ số nào (tương đương [0-9])	/\d\d/ khớp với "12", "99"
\w	Khớp với bất kỳ ký tự từ nào (chữ cái, số, dấu gạch dưới)	/\w+/ khớp với "word", "user_123"
\s	Khớp với bất kỳ ký tự khoảng trắng nào (dấu cách, tab, v.v.)	/hello\sworld/ khớp với "hello world"
[...]	Bộ ký tự: Khớp với một ký tự bất kỳ bên trong ngoặc	/[aeiou]/ khớp với bất kỳ nguyên âm nào
[^...]	Bộ ký tự phủ định: Khớp với ký tự không có trong ngoặc	/[^0-9]/ khớp với bất kỳ ký tự nào không phải là số

Ký tự	Mô tả	Ví dụ
+	Quantifier: Khớp với một hoặc nhiều lần xuất hiện của ký tự trước nó	/a+/ khớp với "a", "aa", "aaa"
*	Quantifier: Khớp với không hoặc nhiều lần xuất hiện	/a*/ khớp với "", "a", "aa"
?	Quantifier: Khớp với không hoặc một lần xuất hiện (tùy chọn)	/color?r/ khớp với "color" và "colour"
{n}	Quantifier: Khớp với đúng n lần xuất hiện	/d{4}/ khớp với một số có bốn chữ số
^	Anchor: Khớp với bắt đầu của chuỗi	/^Start/ khớp với "Start of line" nhưng không khớp "This is the Start"
\$	Anchor: Khớp với kết thúc của chuỗi	/end\$/ khớp với "This is the end" nhưng không khớp "end of the line"
()	Grouping: Nhóm các phần của mẫu lại với nhau	/^(abc)+\$/ khớp với "abc", "abcabc"

#### 4.6.5.5. Các phương thức sử dụng Regex

##### a) `regex.test(string)`

- Mục đích: Kiểm tra xem chuỗi có khớp với mẫu hay không.
- Trả về: true (nếu khớp) hoặc false (nếu không khớp).
- Ứng dụng: Hoàn hảo cho việc xác thực dữ liệu.

Ví dụ: Xác thực email đơn giản.

// ^ - Bắt đầu chuỗi

// [a-zA-Z0-9.\_-]+ - Một hoặc nhiều ký tự (chữ, số, ., \_ -)

// @ - Ký tự @

// [a-zA-Z0-9.-]+ - Một hoặc nhiều ký tự tên miền

// \. - Dấu chấm (dấu \ để nó không bị hiểu là "ký tự bất kỳ")

// [a-zA-Z]{2,6} - Từ 2 đến 6 chữ cái cho TLD (com, vn, net)

// \$ - Kết thúc chuỗi

```
const emailPattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;
```

```
const email1 = "test.user@example.com";
```

```
const email2 = "invalid-email@.com";
```

```
console.log(`Email "${email1}" có hợp lệ không?`, emailPattern.test(email1)); //
```

```
true
```

```
console.log(`Email "${email2}" có hợp lệ không?`, emailPattern.test(email2)); //
```

```
false
```

b) *string.match(regex)*

- Mục đích: Tìm kiếm trong chuỗi và trả về các kết quả khớp.

- Trả về:

+ Nếu không có cờ g: Một mảng chứa kết quả khớp đầu tiên cùng với thông tin chi tiết (vị trí, các nhóm con), hoặc null nếu không khớp.

+ Nếu có cờ g: Một mảng chứa tất cả các chuỗi con đã khớp, hoặc null.

Ví dụ: Tìm tất cả các số trong một chuỗi:

```
const text = "Sản phẩm A giá 15000đ, sản phẩm B giá 20000đ. Tổng cộng 35000đ.);
```

```
const numberPattern = /\d+/g; // \d: ký tự số, +: một hoặc nhiều, g: tìm tất cả  
const numbers = text.match(numberPattern);
```

```
console.log("Các số tìm thấy:", numbers); // Output: ["15000", "20000", "35000"]
```

c) *string.replace(regex, newSubstring)*

- Mục đích: Tìm kiếm các chuỗi con khớp với mẫu và thay thế chúng.

- Trả về: Một chuỗi mới đã được thay thế.

Ví dụ:

Thay đổi định dạng ngày tháng.

```
const dateString = "Hôm nay là ngày 25-12-2025";
// Mẫu tìm kiếm: (\d{2}) - (\d{2}) - (\d{4})
// - (\d{2}): Tìm và "bắt giữ" (capture) một nhóm gồm 2 chữ số.
// - $1, $2, $3: Tham chiếu đến các nhóm đã được bắt giữ theo thứ tự.
const formatedDate = dateString.replace(/(\d{2})-(\d{2})-(\d{4})/, 'năm $3,
tháng $2, ngày $1');
console.log("Chuỗi gốc:", dateString);
console.log("Chuỗi sau khi định dạng:", formatedDate);
// Output: Hôm nay là ngày năm 2025, tháng 12, ngày 25
```

## 4.7. BẤT ĐỒNG BỘ TRONG JAVASCRIPT

### 4.7.1. Giới thiệu về bất đồng bộ trong JavaScript

Trong lập trình Web, có nhiều thao tác không thể thực hiện được cùng lúc, chẳng hạn như việc tải dữ liệu từ Internet hay xử lý một tập tin lớn. Nếu để JavaScript chạy theo thứ tự mặc định, toàn bộ trang Web sẽ rơi vào trạng thái “đóng băng”, ngăn cản người dùng tiếp tục sử dụng.

Để tránh điều này xảy ra, JavaScript đã giới thiệu một cơ chế gọi là bất đồng bộ (tên tiếng Anh là “asynchronous”), giúp chương trình xử lý các tác vụ mất thời gian trong khi không ảnh hưởng đến các thao tác khác của người dùng.

Để đơn giản hóa, cơ chế bất đồng bộ giao nhiệm vụ hiện tại cho trình duyệt xử lý trong khi tiếp tục thực hiện các công việc khác. Kết quả sẽ được trả về ngay khi nhiệm vụ được hoàn thành, đảm bảo trang Web hoạt động mượt mà và ổn định.

### 4.7. 2. Cơ chế hoạt động của bất đồng bộ

Đầu tiên, cần làm rõ rằng JavaScript là một ngôn ngữ đơn luồng (tiếng Anh là “single-threaded”), hay còn có nghĩa là ngôn ngữ này chỉ có thể xử lý một tác vụ tại một thời điểm. Tuy nhiên, cơ chế bất đồng bộ kết hợp với hệ thống quản lý tác vụ để tạo nên vòng lặp sự kiện (tiếng Anh là “event loop”) vượt qua rào cản này.

Bất đồng bộ được tạo nên từ bốn thành phần chính:

- Call stack (ngăn gọi hàm): Nơi JavaScript thực hiện từng dòng code theo tuân tự.
- Web APIs (API của trình duyệt): Giúp xử lý các tác vụ bất đồng bộ như setTimeOut, fetch, v.v. hay các sự kiện của người dùng.
- Callback Queue (hàm đợi hàm gọi lại): Khu vực lưu trữ các hàm sẵn sàng được thực hiện khi tác vụ bất đồng bộ hoàn thành.
- Event loop (vòng lặp sự kiện): Cầu nối giữa Call Stack và Callback Queue, giúp kiểm tra và thực hiện các hàm khi Call Stack rảnh.

Ví dụ về cách bất đồng bộ hoạt động:

```
console.log("1. Ví dụ về Bất Đồng Bộ");
setTimeout(() => {
  console.log("2. Dòng code này được in ra sau 3 giây");
}, 3000);
console.log("3. Dòng code này được in ra sau dòng 1");
```

Kết quả:

```
1. Ví dụ về Bất Đồng Bộ
3. Dòng code này được in ra sau dòng 1
2. Dòng code này được in ra sau 3 giây
```

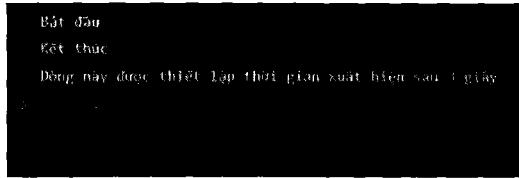
### 4.7.3. Các hàm trong JavaScript để xử lý bất đồng bộ

#### 4.7.3.1. Callback

Một hàm callback (dịch thô là “hàm gọi lại”) trong JavaScript là một hàm được truyền vào hàm khác như một đối số và sẽ được gọi lại sau khi một tác vụ được hoàn thành. Các hàm Callback là cách đơn giản nhất để xử lý bất đồng bộ trong JavaScript, chúng có thể được gọi bằng cách đồng bộ hoặc không đồng bộ.

JavaScript cung cấp một vài hàm gọi lại, nhưng người dùng cũng có thể tự tạo hàm gọi lại của riêng họ. Một vài hàm gọi lại được cung cấp trước đây hay dùng là setTimeout, addEventListener, forEach, v.v.. Để sử dụng chúng, người dùng phải cung cấp chức năng và logic của riêng họ.

Một số ví dụ về việc sử dụng hàm gọi lại:

Code ví dụ	Kết quả
<pre>console.log("Bắt đầu"); setTimeout(() =&gt; {   console.log("Đòng này được thiết lập thời gian xuất hiện sau 3 giây"); }, 3000); console.log("Kết thúc");  &lt;button&gt;Click Me!&lt;/button&gt; &lt;div id="message"&gt;&lt;/div&gt;  &lt;script&gt; button.addEventListener("click", function() {   messageBox.textContent = "Bạn đã bấm nút!"; }); &lt;/script&gt;</pre>	 <p>Bắt đầu Kết thúc Đòng này được thiết lập thời gian xuất hiện sau 3 giây</p> <p>Trước khi bấm nút:</p> <p><b>Click Me!</b></p> <p>Sau khi bấm nút:</p> <p><b>Click Me!</b></p>

Do tính đơn giản của hàm gọi lại nên chúng có thể gây ra lỗi callback hell hay pyramid of doom (do hình dáng nhìn giống kim tự tháp) khi các hàm gọi lại được lồng vào nhau:

```
doSomething(function(result) {
  doNext(result, function(nextResult) {
    doAnother(nextResult, function(finalResult) {
      console.log(finalResult);
    });
  });
});
```

```
});  
});  
});
```

Vì vấn đề này mà các hàm gọi lại không được sử dụng phổ biến trong lập trình bất đồng bộ như hàm Promise.

#### 4.7.3.2. Promise

Promise (tiếng Việt là “lời hứa”) là một đối tượng đại diện cho một giá trị sẽ có trong tương lai, không cần biết rằng tác vụ bất đồng bộ đó được thực hiện thành công hay thất bại. Nhờ tính chất này mà các hàm bất đồng bộ không cần trả ra kết quả ngay lập tức mà có thể trả về một Promise để thay cho một giá trị sẽ được trả ra trong tương lai.

Một promise, ở một thời điểm nhất định, sẽ ở một trong ba trạng thái:

- Pending: Trạng thái mặc định.
- Fulfilled: Trạng thái khi hàm được thực thi thành công.
- Rejected: Trạng thái khi hàm được thực thi thất bại.

Trạng thái cuối cùng của Promise luôn luôn là fulfilled (sẽ được trả ra cùng một giá trị) hoặc rejected (sẽ được trả ra cùng lý do hoặc lỗi). Một Promise thường đi kèm với hàm then để xử lý khi trả ra trạng thái fulfilled hoặc rejected.

Ví dụ về cách sử dụng Promise:

Ví dụ	Kết quả
<pre>// hàm Promise  function randomTask() {     return new Promise((resolve, reject) =&gt; {         const success = Math.random() &gt;         0.5; // 50% chance         setTimeout(() =&gt; {             if (success) {                 resolve("✓ Bạn đã đạt được giải thưởng");             } else {                 reject("✗ Thua: Chúc bạn may mắn lần sau");             }         }, 1000);     }); }</pre>	<p>Nếu ra số &gt; 0.5:</p> <p>Thắng: ✓ Bạn đã đạt được giải thưởng</p> <p>Nếu ra số &lt;= 0.5:</p> <p>Thua: ✗ Chúc bạn may mắn lần sau</p>

```

} else {
    reject("X Chúc bạn may mắn lần
sau");
}
}, 1000);
});
}

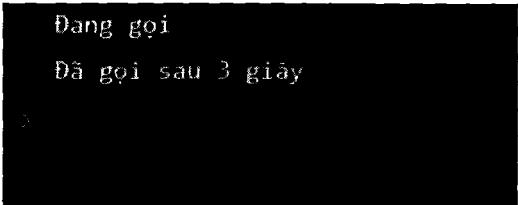
// Sử dụng Promise
randomTask()
.then((message) => {
    console.log("Thắng:", message);
})
.catch((error) => {
    console.error("Thua:", error);
});

```

#### 4.7.3.3. Async/Await

Hàm Async là một cú pháp mới trong JavaScript, giúp người dùng tạo một hàm bất đồng bộ với cú pháp dễ hiểu. Từ khóa Await có thể được sử dụng kèm bên trong hàm Async để tạo ra hành vi giống với Promise.

Ví dụ về cách sử dụng Async/Await:

Ví dụ	Kết quả
<pre> function returnAfter3Seconds() {     return new Promise((resolve) =&gt; {         setTimeout(() =&gt; {             resolve("Đã gọi sau 3 giây");         }, 3000);     }); }  async function asyncCall() { </pre>	<p><b>Kết quả</b></p> <p>Đang gọi</p> <p>Đã gọi sau 3 giây</p> 

```

console.log("Đang gọi");
const result = await returnAfter-
3Seconds();
console.log(result);
}
asyncCall();

```

#### 4.7.3.4. So sánh giữa ba cách lập trình bất đồng bộ

Tiêu chí	Hàm Callback	Promise	Async + Await
Độ phổ biến	Còn được dùng nhưng không phổ biến như trước	Phổ biến rộng rãi	Khác biệt không đáng kể so với Promise
Cú pháp	Cú pháp lồng hàm, có thể dẫn đến “callback hell”, làm code khó đọc	Sử dụng kèm .then(), .catch() giúp dễ đọc	Giúp luồng chương trình nhìn đồng bộ hơn, tăng tính dễ đọc
Xử lý khi gặp lỗi	Cần lập trình viên phải xử lý bằng tay, gây nhiều khó khăn	Các lỗi được xử lý trong .catch(). Việc xử lý lỗi được gộp vào một chỗ để dễ xử lý	Sử dụng cú pháp try...catch giúp đơn giản hóa việc xử lý lỗi
Hỗ trợ	Có thể sử dụng trong mọi môi trường JavaScript	Có thể sử dụng trong mọi môi trường JavaScript hiện đại	Mới chỉ được hỗ trợ bởi phiên bản ES2017 và các phiên bản sau đó
Hiệu quả	Sử dụng rất ít overhead* Có thể gọi hàm trực tiếp	Sử dụng rất ít overhead để tạo ra hàm Promise	Sử dụng rất ít overhead như Promise

\*Overhead: là một thuật ngữ dùng để mô tả chi phí hoặc gánh nặng mà một thao tác nào đó đặt lên hệ thống hoặc chương trình.

#### **4.7.4. Một vài lỗi thường gặp khi lập trình bất đồng bộ**

##### **4.7.4.1. Chặn luồng của chương trình**

Đây là lỗi khi viết các dòng code bất đồng bộ mà lại vô tình chặn luồng của chương trình, đi ngược lại mục đích của việc lập trình bất đồng bộ.

Ví dụ:

```
async function fetchData() {  
    let data = await fetch('link_bat_ky');  
    xuLyDuLieu(data);  
}
```

Để tránh lỗi này, chúng ta chỉ cần đảm bảo rằng các hàm xử lý dữ liệu cần thiết cũng được thiết lập bất đồng bộ:

```
async function fetchData() {  
    let data = await fetch('link_bat_ky');  
    await xuLyDuLieu(data);  
}
```

##### **4.7.4.2. Không xử lý lỗi**

Không lập trình chức năng xử lý lỗi trong các hàm bất đồng bộ có thể dẫn đến lỗi.

Ví dụ:

```
async function fetchData() {  
    let response = await fetch('https://api.example.com/data');  
    let data = await response.json();  
    // Chưa xử lý lỗi  
}
```

Để tránh lỗi này, chúng ta có thể sử dụng cú pháp try...catch để xử lý các lỗi tiềm năng:

```
async function fetchData() {
```

```
try {  
    let response = await fetch('link_bat_ky');  
    let data = await response.json();  
    } catch (error) {  
        console.error('Không lấy được dữ liệu:', error);  
    }  
}
```

#### **4.7.4.3. Sử dụng chung dữ liệu chưa đồng bộ**

Cho phép các hàm bất đồng bộ sử dụng và tương tác với các dữ liệu chưa đồng bộ có thể dẫn đến lỗi. Lỗi này thường chỉ gặp khi chúng ta lập trình ở cấp độ cao.

# MỤC LỤC

	Trang
LỜI GIỚI THIỆU .....	3
<b>CHƯƠNG I: GIỚI THIỆU CHUNG.....</b>	<b>7</b>
1.1. Lập trình Web.....	7
1.1.1. Khái niệm .....	7
1.1.2. Lịch sử hình thành .....	7
1.1.3. Ứng dụng .....	8
1.1.4. Cách tạo ra một trang Web .....	8
1.2. Giới thiệu HTML .....	12
1.2.1. Giới thiệu về HTML.....	12
1.2.2. Lịch sử hình thành HTML .....	13
1.2.3. Các phiên bản .....	15
1.2.4. Nguồn gốc của ngôn ngữ HTML.....	16
1.2.5. Tình trạng hiện tại của HTML .....	16
1.2.6. Độ khó của HTML .....	17
1.3. Giới thiệu CSS .....	18
1.3.1. Giới thiệu về CSS.....	18
1.3.2. Lịch sử hình thành.....	19
1.3.3. Cách hoạt động .....	19
1.3.4. Cách thiết lập CSS.....	21
1.3.5. CSS thông dụng .....	23
1.4. Giới thiệu JS.....	25
1.4.1. Nguồn gốc và lịch sử phát triển của JavaScript.....	25
1.4.2. ECMAScript5 (2009) .....	26
1.4.3. ECMAScript6/ES2015 (2015).....	26
1.4.4. Các phiên bản sau ES2015.....	27
1.4.5. Tổng quan về JavaScript .....	27

<b>CHƯƠNG II: HTML .....</b>	31
<b>2.1. Tag.....</b>	31
2.1.1. Giới thiệu về thẻ .....	31
2.1.2. Cấu trúc của mỗi thẻ .....	31
2.1.3. Các thẻ quan trọng trong HTML .....	33
<b>2.2. Head.....</b>	33
2.2.1. Giới thiệu .....	33
2.2.2. Phần tử <title>.....	34
2.2.3. Phần tử <style> .....	35
2.2.4. Phần tử <meta> .....	37
2.2.5. Phần tử <link> .....	39
2.2.6. Phần tử <script> .....	40
2.2.7. Phần tử <base> .....	41
<b>2.3. Hiển thị các ký tự .....</b>	42
2.3.1. Các thẻ hiển thị văn bản phổ biến .....	43
2.3.2. Các thẻ định dạng văn bản phổ biến.....	46
2.3.3. Hiển thị đặc biệt.....	49
<b>2.4. Ảnh và link .....</b>	52
2.4.1. Ảnh.....	52
2.4.2. Link.....	53
<b>2.5. Table (bảng).....</b>	57
2.5.1. Cách sử dụng bảng .....	57
2.5.2. Cấu trúc cơ bản của bảng.....	58
2.5.3. Thuộc tính cơ bản của bảng .....	59
2.5.4. Tạo bảng HTML cơ bản với ví dụ minh họa .....	60
<b>2.6. Căn chỉnh trang Web.....</b>	64
2.6.1. Căn chỉnh tiêu đề và đoạn văn .....	64
2.6.2. Căn giữa hình ảnh .....	66
2.6.3. Tạo bảng và căn chỉnh trong bảng.....	66
2.6.4. Đổi màu nền và chữ.....	67
<b>2.7. Mã nhúng .....</b>	68
2.7.1. Mã nhúng là gì .....	68

2.7.2. Mã nhúng hoạt động như thế nào.....	68
2.7.3. Nhúng video Youtube.....	69
2.7.4. Nhúng Google Map .....	70
<b>CHƯƠNG III: CSS .....</b>	<b>71</b>
3.1.Giới thiệu về CSS và các phần tử HTML.....	71
3.2. Vai trò của các thẻ HTML.....	71
3.3. Nhắm mục tiêu các phần tử HTML với CSS.....	72
3.3.1. Bộ chọn kiểu.....	72
3.3.2. Nhóm bộ chọn.....	73
3.3.3. Bộ chọn chung (*). ....	74
3.3.4. Hành vi mặc định của phần tử: khối, nội dòng và nội dòng – khối.....	74
3.3.5. Các ví dụ thực tế.....	77
3.4. Các cách nhúng CSS vào HTML.....	80
3.4.1. Kiểu nội dòng (Inline Styles) .....	80
3.4.2. Kiểu nội bộ (Internal/Embedded Styles) .....	81
3.4.3. Kiểu bên ngoài (External Stylesheets).....	82
3.5. Selector là gì? (Bộ chọn trong CSS) .....	84
3.5.1. Bộ chọn cơ bản (Basic Selectors) .....	85
3.5.2. :hover và :active – tạo hiệu ứng tương tác .....	88
3.6. Box model .....	91
3.6.1. Content Area (nội dung) .....	92
3.6.2. Kích thước của phần tử hộp .....	92
3.6.3. Đường viền (Border) .....	93
3.6.4. Padding (khoảng đệm trong).....	95
3.6.5. Margin (khoảng cách ngoài).....	96
3.6.6. Box-sizing .....	97
3.6.7. Overflow – xử lý tràn nội dung .....	98
3.7. Background .....	99
3.7.1. Tổng quan về background trong CSS.....	99
3.7.2. Thiết lập màu nền với background-color .....	100
3.7.3. Thiết lập hình nền với background-image.....	102

3.8. Font .....	104
3.8.1. Tổng quan về định dạng văn bản trong CSS .....	104
3.8.2. Các thuộc tính font cơ bản.....	104
3.9. Position + Float .....	108
3.9.1. Giới thiệu về Positioning và Floating trong CSS.....	108
3.9.2. Positioning – định vị phần tử .....	109
3.9.3. Floating .....	124
<b>CHƯƠNG IV: JAVASCRIPT .....</b>	<b>133</b>
4.1. Giới thiệu về JavaScript.....	133
4.1.1. JavaScript là gì? .....	133
4.1.2. Lịch sử hình thành và phát triển .....	134
4.1.3. Cách nhúng JavaScript vào trang HTML .....	136
4.1.4. Cú pháp cơ bản của JavaScript.....	137
4.1.5. Ứng dụng thực tiễn của JavaScript.....	141
4.2. Kiểu dữ liệu trong JavaScript .....	143
4.2.1. Tổng quan về kiểu dữ liệu trong JavaScript.....	143
4.2.2. Kiểu dữ liệu Number trong JavaScript.....	145
4.2.3. Kiểu dữ liệu Boolean trong JavaScript.....	146
4.2.4. Kiểu dữ liệu String trong JavaScript.....	148
4.2.5. Kiểu dữ liệu Array trong JavaScript.....	151
4.3. JavaScript function .....	155
4.3.1. Hàm (Function) là gì trong JavaScript? .....	155
4.3.2. Các cách khai báo hàm .....	156
4.3.3. Tham số và đối số .....	157
4.3.4. Trả về giá trị với return .....	158
4.3.5. Phạm vi (Scope) và biến cục bộ/toàn cục .....	159
4.3.6. Các hàm có sẵn trong JavaScript .....	160
4.4. JavaScript object .....	161
4.4.1. Tổng quan về object trong JavaScript .....	161
4.4.2. Cách tạo object .....	161
4.4.3. Thuộc tính và method.....	163
4.4.4. Getter/Setter .....	164

4.4.5. Phép toán với object .....	165
4.4.6. Prototype và kế thừa .....	167
4.4.7. Ví dụ và chú ý .....	168
4.5. JavaScript DOM.....	170
4.5.1. DOM là gì? Tại sao nó lại quan trọng? .....	170
4.5.2. Cấu trúc cây DOM .....	170
4.5.3. Truy xuất các phần tử DOM.....	171
4.5.4. Thay đổi nội dung và thuộc tính .....	174
4.5.5. Tạo, thêm và xóa phần tử.....	176
4.5.6. Xử lý sự kiện (Event Handling) .....	177
4.5.7. Event Bubbling (sủi bọt) và Capturing (bắt giữ) .....	178
4.5.8. Event.preventDefault() và event.stopPropagation() .....	179
4.6. Vụn vặt trong JavaScript .....	179
4.6.1. Giới thiệu .....	179
4.6.2. Đối tượng Math .....	180
4.6.3. Xử lý lỗi (Error Handling) .....	184
4.6.4. JSON (JavaScript Object Notation).....	187
4.6.5. Biểu thức chính quy (Regular Expressions – Regex) .....	191
4.7. Bất đồng bộ trong JavaScript .....	195
4.7.1. Giới thiệu về bất đồng bộ trong JavaScript .....	195
4.7.2. Cơ chế hoạt động của bất đồng bộ .....	195
4.7.3. Các hàm trong JavaScript để xử lý bất đồng bộ.....	196
4.7.4. Một vài lỗi thường gặp khi lập trình bất đồng bộ .....	201

# SỔ TAY KIẾN THỨC TIN HỌC

## ÔN THI TỐT NGHIỆP THPT

### LẬP TRÌNH WEB CƠ BẢN

NHÀ XUẤT BẢN BÁCH KHOA HÀ NỘI

Số 1 Đại Cồ Việt - Phường Bạch Mai - Thành phố Hà Nội

VPGD: Ngõ 17 Tạ Quang Bửu - Phường Bạch Mai - Thành phố Hà Nội

ĐT: 024. 38684569; Fax: 024. 38684570

<https://nxbbachkhoa.vn>

*Chịu trách nhiệm xuất bản:*

Giám đốc - Tổng Biên tập

**PSG. TS. BÙI ĐỨC HÙNG**

Biên tập: Đỗ Thanh Thùy

Sửa bản in Benito

Trình bày bìa: Benito

#### LIÊN KẾT XUẤT BẢN

Công ty cổ phần Văn hóa & Bản quyền Benito,  
số 10, ngách 43/5 Nguyễn Ngọc Nại, phường Phương Liệt, TP. Hà Nội

---

In 500 cuốn khổ (17 × 24) cm tại Công ty TNHH Thương mại và Dịch vụ in Huyền Linh.

Địa chỉ: Số 4B ngõ 145 phố Vĩnh Hưng, Phường Vĩnh Hưng, Thành phố Hà Nội.

Địa chỉ sản xuất: Lô 55-56 Khu dân cư Hương Mạc, đường 277, Phường Phù Khê, tỉnh Bắc Ninh.

Số xuất bản: 3794-2025/CXBIPH/02-54/BKHN; ISBN: 978-632-609-689-7.

Số QĐXB: 504/QĐ-ĐHBK-BKHN ngày 17/10/2025.

In xong và nộp lưu chiểu quý IV năm 2025.