BỘ CÔNG THƯƠNG

TRƯỜNG CAO ĐẲNG CÔNG NGHIỆP HUẾ

GIÁO TRÌNH

MÔN HỌC: LẬP TRÌNH JAVASCRIPT

Ngành: **ỨNG DỤNG PHẦN MỀM**

TRÌNH ĐỘ: CAO ĐẲNG

Ban hành kèm theo Quyết định số: /QĐ-... ngày…….tháng…. năm ..... của Hiệu trưởng Trường Cao đẳng Công nghiệp Huế

Thừa Thiên Huế, năm 2017

MỤC LỤC

Chương 1. Tổng quan về Javascript

1.1. Giới thiệu về Javascript

JavaScript là một ngôn ngữ lập trình đa nền tảng (cross-platform), ngôn ngữ lập trình kịch bản, hướng đối tượng. JavaScript là một ngôn ngữ nhỏ và nhẹ (small and lightweight). Khi nằm bên trong một môi trường (host environment), JavaScript có thể kết nối tới các object của môi trường đó và cung cấp các cách quản lý chúng (object).

JavaScript chứa các thư viện tiêu chuẩn cho các object, ví dụ như: Array, Date, và Math, và các yếu tố cốt lõi của ngôn ngữ lập trình như: toán tử (operators), cấu trúc điều khiển (control structures), và câu lệnh. JavaScript có thể được mở rộng cho nhiều mục đích bằng việc bổ sung thêm các object; ví dụ:

Client-side JavaScript - JavaScript phía máy khách, JavaScript được mở rộng bằng cách cung cấp các object để quản lý trình duyệt và Document Object Model (DOM) của nó. Ví dụ, phần mở rộng phía máy khách cho phép một ứng dụng tác động tới các yếu tố trên một trang HTML và phản hồi giống các tác động của người dùng như click chuột, nhập form, và chuyển trang.

Server-side JavaScript - JavaScript phía Server, JavaScript được mở rộng bằng cách cung cấp thêm các đối tượng cần thiết để để chạy JavaScript trên máy chủ. Ví dụ, phần mở rộng phía server này cho phép ứng dụng kết nối với cơ sở dữ liệu (database), cung cấp thông tin một cách liên tục từ một yêu cầu tới phần khác của ứng dụng, hoặc thực hiện thao tác với các tập tin trên máy chủ.

JavaScript và Java

JavaScript và Java thì giống nhau ở những cái này nhưng lại khác nhau ở cái khác. Ngôn ngữ JavaScript có lẽ giống giống với ngôn ngữ Java nhưng JavaScript không có khai báo static cũng như không có "tính mạnh về kiểu" (strong type checking) như Java. Cú pháp (syntax) lập trình, đặt tên công thức và xây dựng điều khiển lưu lượng (control-flow) cơ bản của JavaScript phần lớn dựa theo ngôn ngữ lập trình Java, đó cũng là lý do tại sao JavaScript được đổi tên từ LiveScript thành JavaScript.

Ngược lại với hệ thống thời gian biên dịch (compile-time) Java của các lớp được xây dựng bởi các khai báo, JavaScript hỗ trợ nền tảng hệ thống thời gian chạy dựa trên một số lượng nhỏ các loại dữ liệu đại diện cho số, boolean và dữ liệu các chuỗi. JavaScript có một mô hình ít phổ biến hơn là mô hình đối tượng dựa trên nguyên mẫu (prototype-based) thay vì các mô hình đối tượng dựa trên lớp (class-based). Các mô hình dựa trên nguyên mẫu cung cấp khả năng thừa kế năng động; nghĩa là, những gì được kế thừa có thể khác nhau cho các đối tượng khác nhau. JavaScript cũng hỗ trợ các phương thức (function) không khai báo bất

cứ gì ở trong. Phương thức có thể là một trong các thuộc tính (property) của các đối tượng, thực thi như là một phương thức đã được định kiểu (loosely typed methods).

JavaScript là một ngôn ngữ rất tự do so với Java. Bạn có thể không cần khai báo tất cả biến (variable), lớp (class) và cả phương thức (method). Bạn không cần quan tâm cho dù phương thức đó là public, private hoặc protected, và bạn không cần phải implement interfaces. Biến, tham số (parameters), và kiểu trả về của phương thức (function return) cũng không cần phải rõ ràng.

1.2. Lịch sử hình thành và phát triển của Javascript

1.2.1. Lịch sử hình thành Javascript

Javascript được biết đến lần đầu tiên là Mocha và chỉ ngay sau đó một thời gian nó được đổi tên thành Livescript và cuối cùng Netscape đã đổi tên thành Javascript, bởi vì sự phổ biến của Java như là một hiện tượng lúc bấy giờ.

Sau khi ngôn ngữ mới được tạo ra, nhóm marketing của Netscape đã yêu cầu Sun cho phép họ đặt tên ngôn ngữ là Javascript để truyền thông và quảng bá, và sau đó cũng chính là lý do mà tại sao hầu hết người dùng chưa sử dụng Javascript bao giờ, đã nghĩ rằng Javascript có liên quan đến Java.

Khoảng 1, 2 năm sau khi phát hành Javascript trên trình duyệt, trình duyệt IE của Microsoft đã lấy lại ngôn ngữ này vào bắt đầu tạo ra ngôn ngữ của riêng mình có tên là Jscript. Cũng tại thời điểm đó, IE đã "thống trị" thị trường và không lâu sau đó Netscape đã phải đóng cửa dự án của mình.

Trước khi Netscape xuống dốc, vào năm 1997 họ đã chuyển Javascript đến ECMA International để làm công tác chuẩn hóa và viết đặc tả,… Cái tên ECMAScript đã được hình thành từ đây.

Trước đó Netscape đã phát hành một vài phiên bản ECMAScript và đến năm 1999 thì phiên bản ECMAScript 3 được phát hành – phiên bản cuối cùng trước khi Netscape đi vào "chế độ ngủ đông" trong suốt 10 năm tiếp theo. Trong suốt 10 năm qua, Microsoft đã "thống trị" toàn bộ thị trường và hãng cũng không ngừng cải tiến các sản phẩm của mình. Tiếp sau đó là sự ra đời của một loạt các trình duyệt khác như Firefox, Chrome và Opera.

Phiên bản ECMAScript 4 đã bị "lãng quên". Phiên bản ECMAScript 5 được phát hành vào năm 2009, ECMAScript 6 được phát hành vào năm 2015 và phiên bản ECMAScript 7, 8, 9 cũng được phát hành vào các năm 2016, 2017, 2018.

1.2.2. ECMAScript là gì

ECMAScript là tên chính thức của ngôn ngữ, là một bộ đặc tả tiêu chuẩn dành cho Javascript do hiệp hội các nhà sản xuất máy tính Châu Âu (European Computer Manufaturers Association - ECMA) đề xuất. Ta cứ nghĩ xem hiện nay có khá nhiều trình duyệt browser ra đời và nếu mỗi browser lại có cách chạy Javascript khác nhau thì các trang web không thể

hoạt động trên tất cả các trình duyệt đó được, vì vậy cần có một chuẩn chung để bắt buộc các browser phải phát triển dựa theo chuẩn đó.

ECMAScript thường được viết tắt là ES.

1.2.3. Các trình duyệt hỗ trợ chạy ECMASript/Javascript

ECMAScript 3 được hỗ trợ đầy đủ trong tất cả các trình duyệt.

ECMAScript 5 được hỗ trợ đầy đủ trong tất các các trình duyệt hiện đại.

Danh sách trình duyệt hỗ trợ cho ES5 (2009)

Trình duyệt	Phiên bản	Ngày
Chrome	23	Tháng 9/2012
Firefox	21	Tháng 4/2013
IE	9*	Tháng 3/2011
IE/Edge	10	Tháng 9/2012
Safari	6	Tháng 7/2012
Opera	15	Tháng 7/2013

1.3. Vai trò của Javascript trong ngành công nghiệp phần mềm hiện nay

JavaScript là một ngôn ngữ lập trình nhẹ. Trình duyệt web nhận code JavaScript ở dạng văn bản gốc và chạy tập lệnh từ đó. Từ quan điểm kỹ thuật, hầu hết các trình biên dịch JavaScript hiện đại thực sự sử dụng một kỹ thuật gọi là just-in-time compiling để cải thiện hiệu suất; mã nguồn JavaScript được biên dịch thành định dạng nhị phân nhanh hơn trong khi tập lệnh đang được sử dụng, để có thể chạy nhanh nhất có thể. Tuy nhiên, JavaScript vẫn được coi là ngôn ngữ được diễn giải, vì quá trình biên dịch được xử lý trong thời gian chạy, thay vì trước thời hạn.

Javascript cho phép ta xây dựng các trang web tương tác. Nếu nghĩ về cấu trúc của một trang web, ta có HTML - mô tả và xác định nội dung cấu trúc cơ bản của trang web, ta có CSS - cho trình duyệt biết nội dung HTML này sẽ được hiển thị như thế nào khi xác định những thứ như màu sắc hay phông chữ. Chỉ với HTML và CSS, ta có một trang web có vẻ tốt nhưng thực sự thì thiếu rất nhiều. JavaScript làm cho trang web trở nên sống động bằng

cách thêm chức năng. JavaScript chịu trách nhiệm cho các yếu tố mà người dùng có thể tương tác, chẳng hạn như trình đơn thả xuống, cửa sổ phương thức và biểu mẫu liên hệ. Nó cũng được sử dụng để tạo ra những thứ như hình động, trình phát video và bản đồ tương tác.

Javascript là một ngôn ngữ lập trình đa năng, có nghĩa là nó chạy trên đa nền tảng, đa trình duyệt. Hơn nữa, bây giờ Javascript đã mở rộng để phát triển ứng dụng Mobile, Desktop và thậm chí cả về mảng game nữa. Ứng dụng phổ biến nhất của JavaScript là ở phía máy khách (hay còn gọi là frontend), nhưng vì Node.js xuất hiện, nhiều người cũng chạy JavaScript ở phía máy chủ (còn gọi là phụ trợ). Khi được sử dụng ở phía client-side, mã JavaScript được đọc, giải thích và được thực thi trong trình duyệt web của người dùng. Khi được sử dụng ở phía máy chủ, nó được chạy trên một máy tính từ xa.

1.4. Sử dụng trình soạn thảo Javascript

1.4.1. Giới thiệu về IDE

Người ta sử dụng khái niệm IDE (Integrated Development Environment) để nói về trình soạn thảo Javascript.

IDE là môi trường lập trình tích hợp nhiều công cụ khác nhau như code editor, debugger, simulator…. Tổng thể mà nói, IDE là một phần mềm bao gồm những gói phần mềm khác giúp phát triển ứng dụng phần mềm.

IDE JavaScript là các phần mềm chuyên dụng cung cấp môi trường lập trình JavaScript cho lập trình viên. Khi sử dụng IDE JavaScript, các developers được hỗ trợ code JavaScript tốt nhất.

1.4.2. Các loại IDE để code JavaScript

Ta có thể sử dụng notepad++ nhưng lời khuyên cho bạn nên sử dụng các IDE chuyên dụng lập trình web. Visual Studio là một công cụ khá phổ biến mà hầu hết lập trình viên nào cũng biết nên mình sẽ không liệt kê vào danh sách sau. Một số phần mềm IDE hay cho bạn ví dụ như:

- Visual Studio Code: Là một trình biên tập lập trình code miễn phí dành cho Windows, Linux và macOS, Visual Studio Code được phát triển bởi Microsoft. Nó được xem là một sự kết hợp hoàn hảo giữa IDE và Code Editor. Visual Studio Code hỗ trợ chức năng debug, đi kèm với Git, có syntax highlighting, tự hoàn thành mã thông minh, snippets, và cải tiến mã nguồn. Nhờ tính năng tùy chỉnh, Visual Studio Code cũng cho phép người dùng thay đổi theme, phím tắt, và các tùy chọn khác. Visual Studio Code hỗ trợ nhiều ngôn ngữ lập trình, hỗ trợ đa nền tảng, cung cấp kho tiện ích mở rộng, kho lưu trữ an toàn, hỗ trợ git,…
- PHPdesigner: PHPdesigner giúp bạn có thể chỉnh sửa, thiết kế, truyền tải các PHP, HTML5, CSS3 và JavaScript vô cùng đơn giản. Chức năng tìm ra các lỗi trong dòng mã PHP của bạn phù hợp với Xdebug. Ngoài ra, phần mềm này còn

hỗ trợ tất cả framework PHP phổ biến như Zend, CodeIgniter, Yii, Symfony và Prado.

- Dreamweaver: Dreamweaver hỗ trợ được rất nhiều loại ngôn ngữ như PHP, ASP.NET, JSP, ASP… Hầu hết các ngôn ngữ lập trình web. Dreamweaver có khá nhiều tiện ích dễ dàng thực hiện các thao tác kéo thả di chuyển các phần tử, các khung của một trang web hay viết code, thẻ tag, bảng mã màu dễ dàng chỉnh sửa, thanh công cụ đầy đủ chức năng tiện ích… tiết kiệm được rất nhiều thời gian cho học viên lập trình. Download Dreamweaver.
- PHP Sublime Text: IDE này được viết dựa trên ngôn ngữ lập trình Python và C++. Sublime Text là một Text Editor cực kỳ hiệu quả dành cho các lập trình viên không những làm tăng hiệu suất làm việc mà còn tiết kiệm thời gian gõ code với các Plugin hữu ích. Download PHP Sublime Text.
- Dưới đây là một số công cụ chỉnh sửa JavaScript đẹp, được đánh giá cao khác: Microsoft FrontPage, Macromedia Dreamweaver MX, Macromedia HomeSite 5

1.5. Viết chương trình Javascript đầu tiên

1.5.1. Cách đặt mã Javascript vào trang web

1.5.1.1. Thẻ <script> trong HTML

Thẻ <script> trong HTML được sử dụng để chỉ định kịch bản phía máy khách như JavaScript. Nó tạo điều kiện cho ta đặt một kịch bản trong tài liệu HTML. Có 2 cách khai báo thẻ script

Cách 1: Đặt trong head hay body

```
<html>
   <head>
      <script type="text/javascript">
         ….
    </script>
   </head>
   <body>
     <script type="text/javascript">
         ….
    </script>
         ….
   </body>
</html>
```

Cách 2: Tạo tập tin bên ngoài và liên kết tập tin đó trong phần head.

```
<html>
   <head>
      <script src="YourJsFile.js"></script>
```

```
    </head>
    <body>
        ….
    </body>
</html>
```

hoặc:

```
<html>
    <head>
        ….
    </head>
    <body>
        ….
        <script src="YourJsFile.js"></script>
    </body>
</html>
```

1.5.1.2. Các thuộc tính của thẻ <script>

- Thẻ <script> có bốn thuộc tính cơ bản.
- Dưới đây là bảng mô tả sơ lược về bốn thuộc tính đó:

src	Xác định đường dẫn đến tập tin JavaScript mà bạn muốn sử dụng cho trang web
async	Đảm bảo việc mã lệnh JavaScript chỉ được thực thi sau khi trang web đã được tải xong hoàn toàn
defer	Đảm bảo việc mã lệnh JavaScript chỉ được thực thi sau khi trang web đã được tải xong hoàn toàn
charset	Xác định kiểu mã hóa ký tự được sử dụng trong tập tin JavaScript

1.5.2. Các ví dụ

Ví dụ 1: Tạo file index.html với phần tử <script> bên trong chứa bốn câu lệnh JavaScript

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
</head>
<body>
    <script>
        var name = "Lê Chi Đó";
        var year = 1993;
        var info = name + " sinh năm " + year;
        document.write(info);
```

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

Ví dụ 2:

- Ta tạo file myscript.js với nội dung như sau:

```
var name = "Lê Chi Đó";
var year = 1993;
var info = name + " sinh năm " + year;
document.write(info);
```

- Tạo file index.html (có cùng thư mục cha với file myscript.js) với nội dung sau:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
</head>
<body>
    <script src="myscript.js"> </script>
</body>
</html>
```

1.6. Cài đặt và cấu hình Xampp

1.6.1. XAMPP là gì?

XAMPP là một phần mềm mã nguồn mở miễn phí, cung cấp một cách dễ dàng cho các nhà thiết kế và phát triển web để cài đặt các thành phần cần thiết để chạy phần mềm dựa trên PHP như WordPress, Drupal, Joomla và các phần mềm khác trên Windows 10, macOS và Linux.

Nếu bạn là nhà phát triển web hoặc bạn đang cố gắng viết blog, XAMPP sẽ tiết kiệm thời gian cài đặt và cấu hình Apache, MySQL, PHP và Perl trên máy tính của bạn để tạo môi trường thử nghiệm.

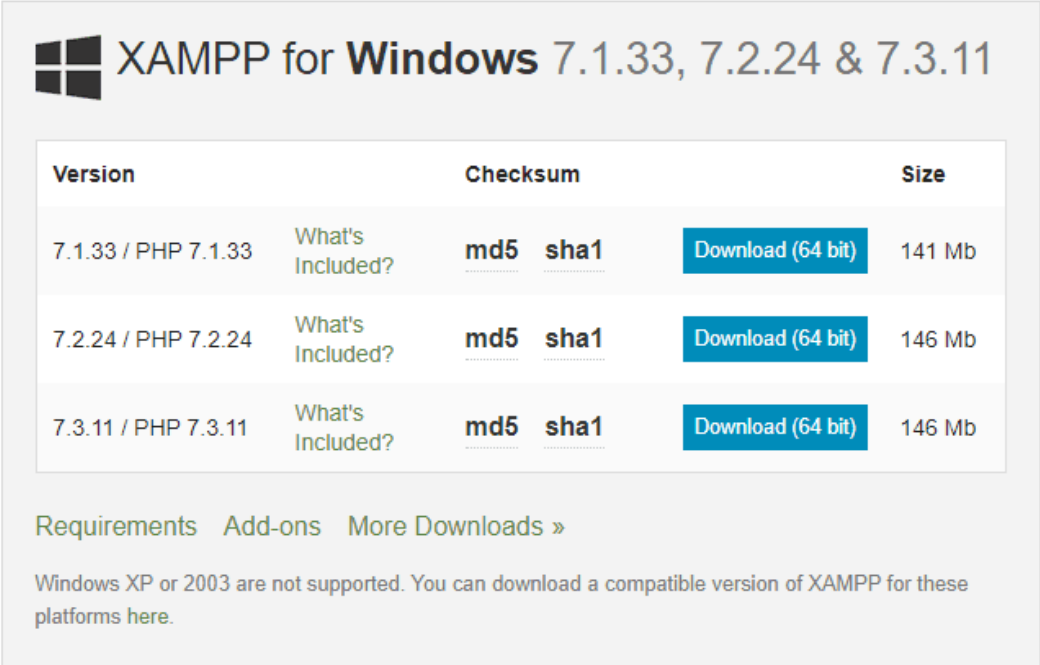
Trong bài học này, ta sẽ tìm hiểu các bước để cài đặt XAMPP trên Windows 10 cũng như các hướng dẫn để cấu hình giải pháp và khắc phục các sự cố thường gặp sau khi thiết lập.

1.6.2. Cách cài đặt XAMPP trên Windows 10

Sử dụng các bước sau để tải xuống và cài đặt XAMPP trên Windows 10:

Bước 1. Mở trang web Apache Friends.

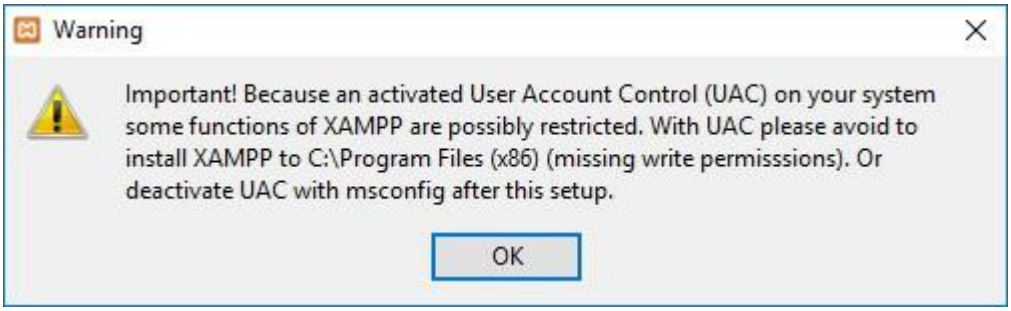
Bước 2. Nhấp vào mục XAMPP for Windows, chọn phiên bản XAMPP tương ứng với phiên bản PHP



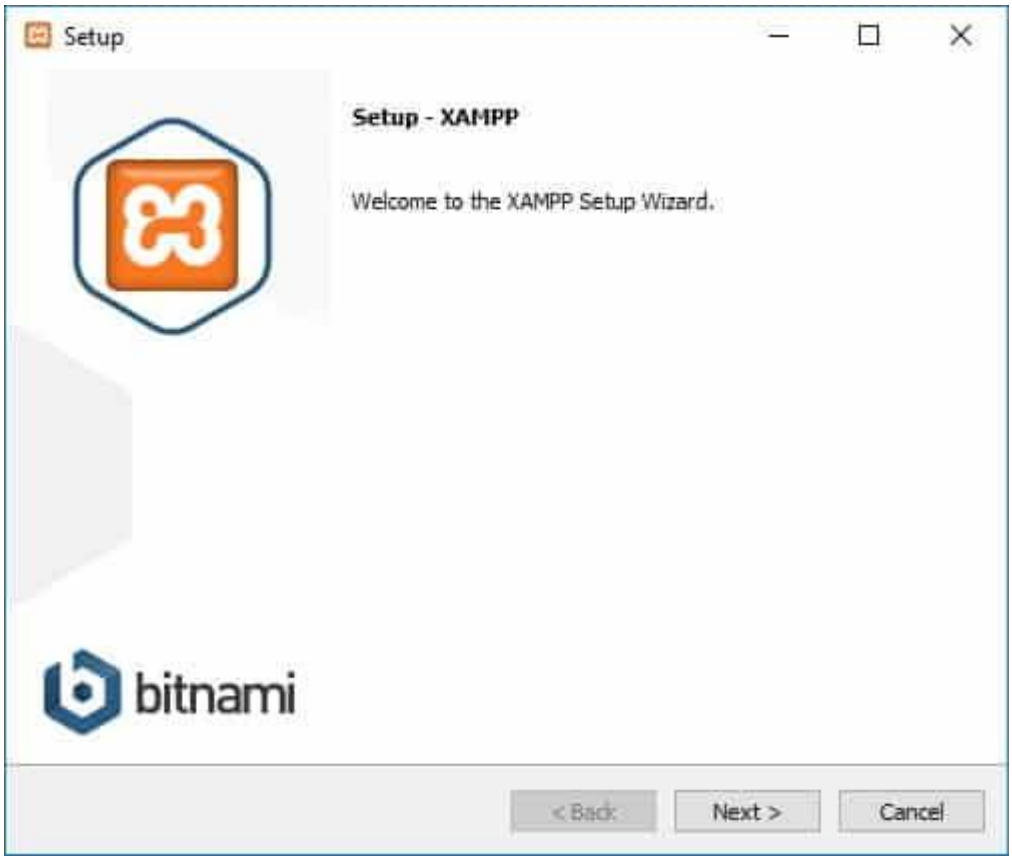
Nên chọn từ phiên bản PHP 7.3 trở lên và tiến hành tải xuống để lưu tệp trên máy tính của bạn.

Bước 3. Bấm đúp chuột vào tệp đã tải xuống để khởi chạy trình cài đặt.

Bước 4. Nhấp vào nút OK.

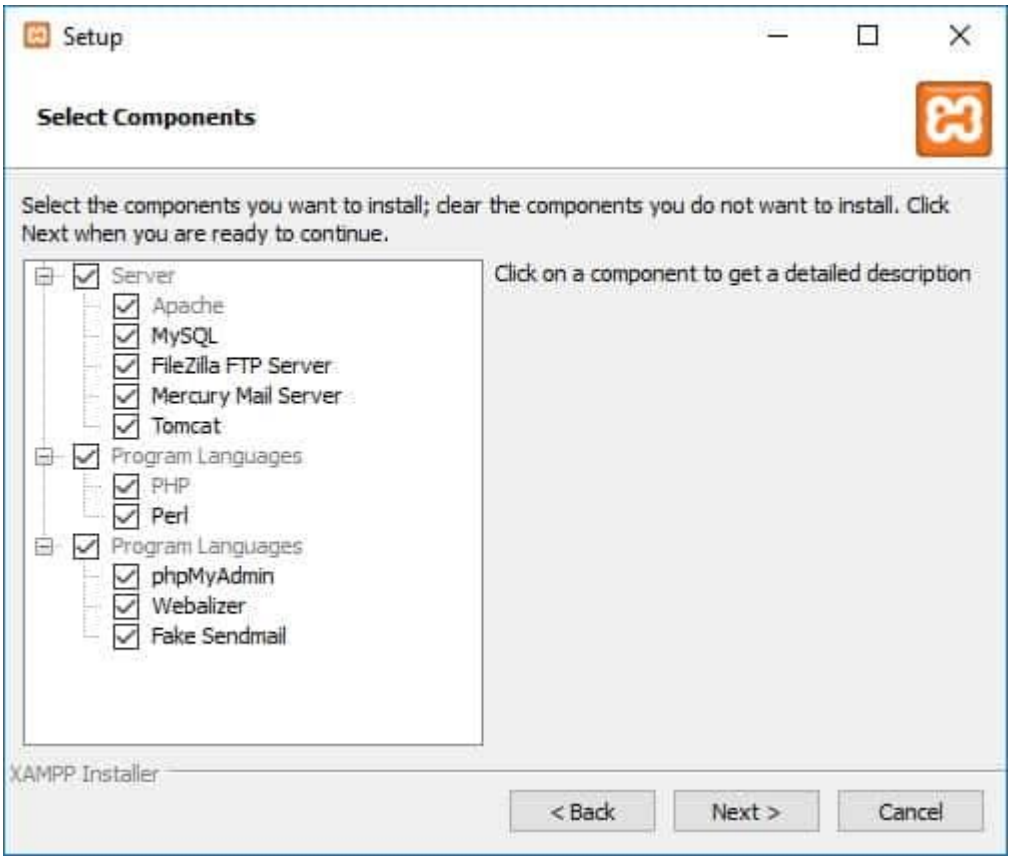


Bước 5. Nhấp vào nút Next.

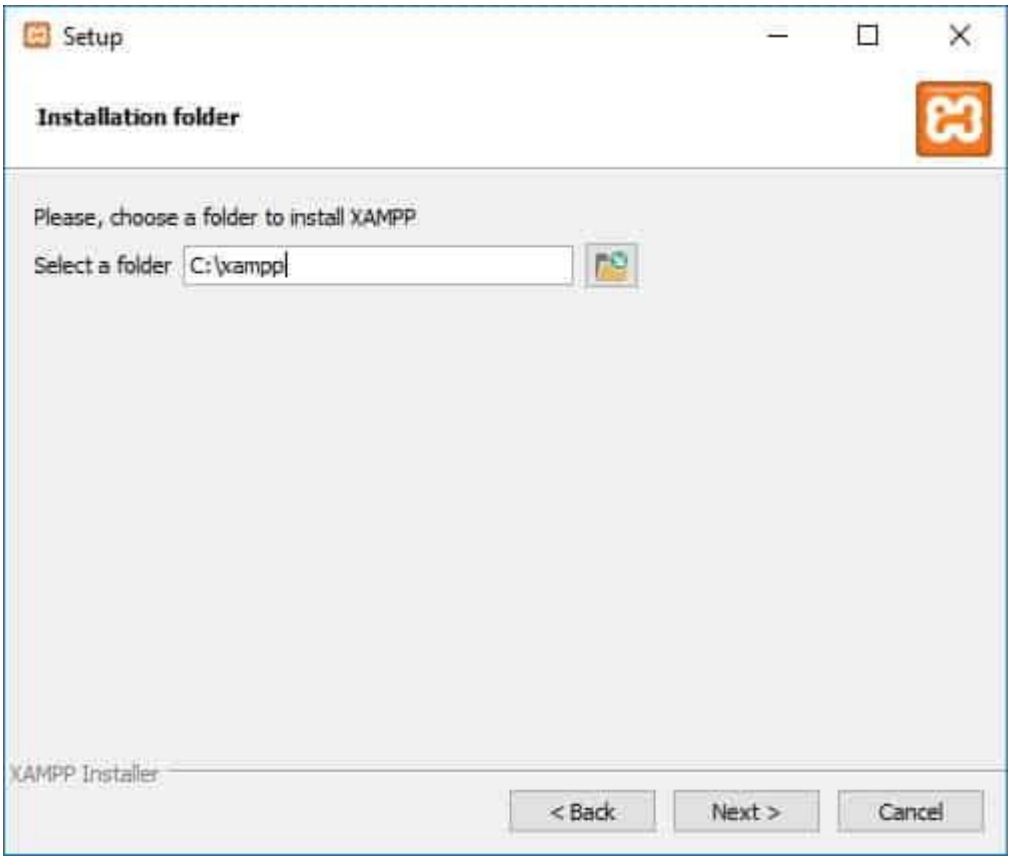


Bước 6. XAMPP cung cấp một loạt các thành phần mà bạn có thể cài đặt, chẳng hạn như MySQL, phpMyAdmin, PHP, Apache, v.v. Đối với hầu hết các phần, bạn sẽ sử dụng hầu hết các thành phần này, vì vậy nên để các tùy chọn mặc định.

Bước 7. Nhấp vào nút Next.



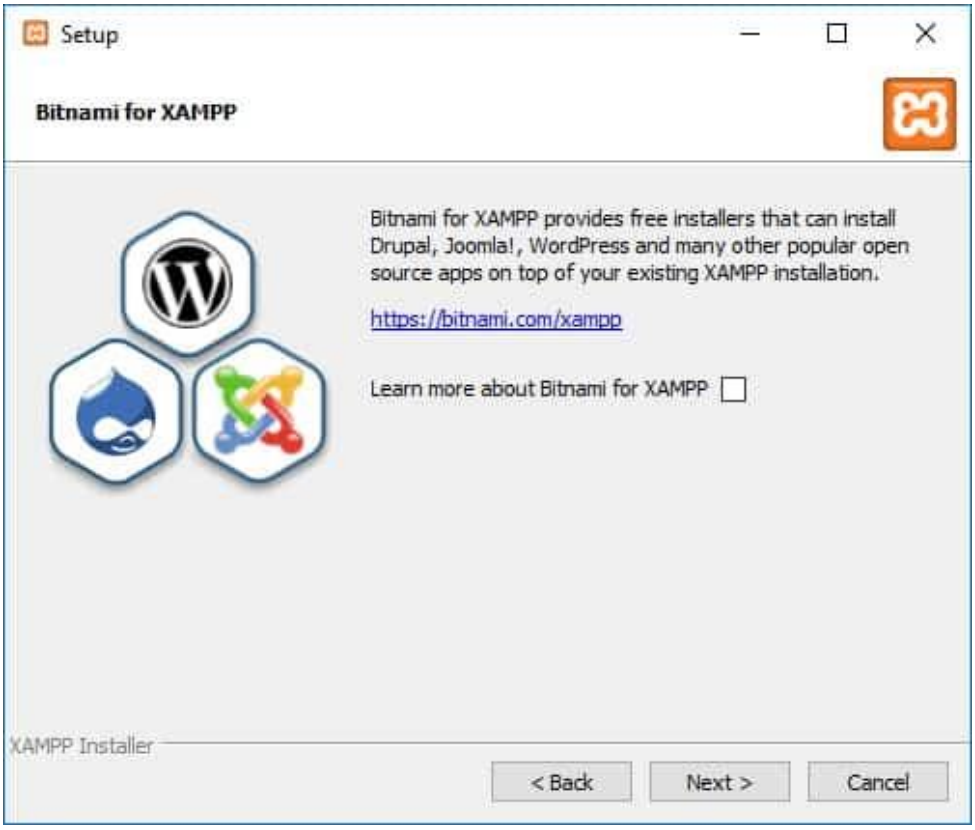
Bước 8. Sử dụng vị trí cài đặt mặc định hoặc chọn thư mục khác để cài đặt phần mềm trong trường Select a folder.



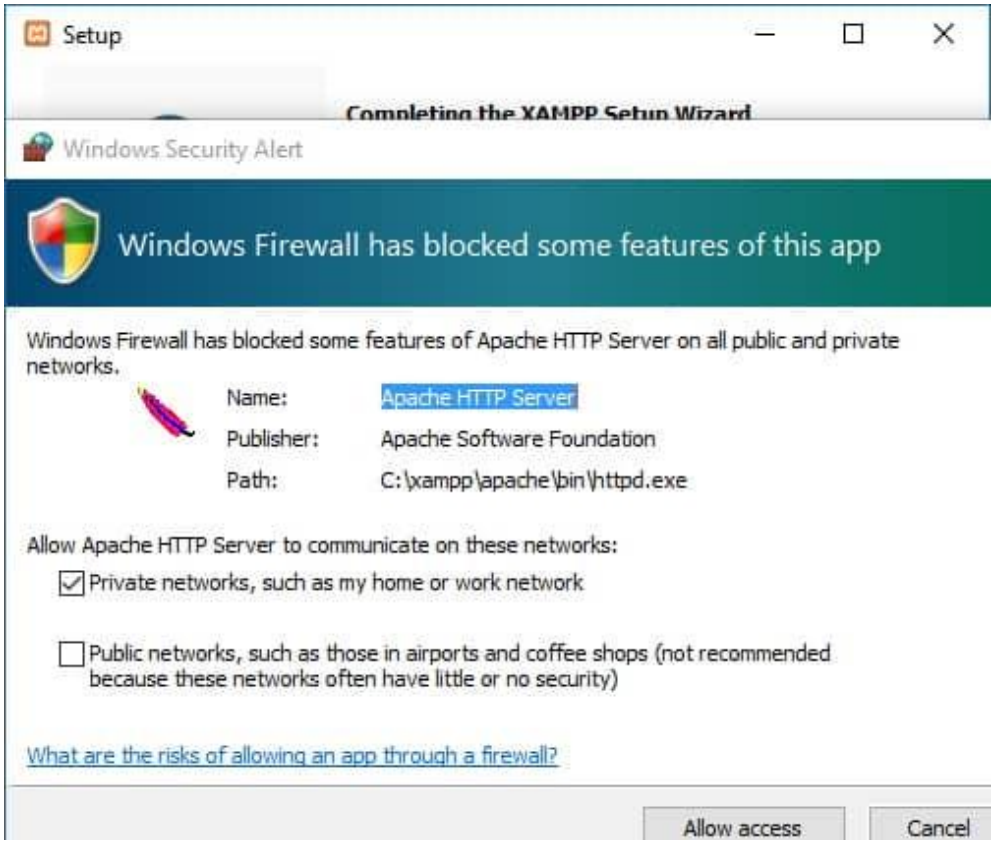
Bước 9. Nhấp vào nút Next.

Bước 10. Tích bỏ tùy chọn Learn more about Bitnami for XAMPP.

Bước 11. Nhấp vào nút Next.



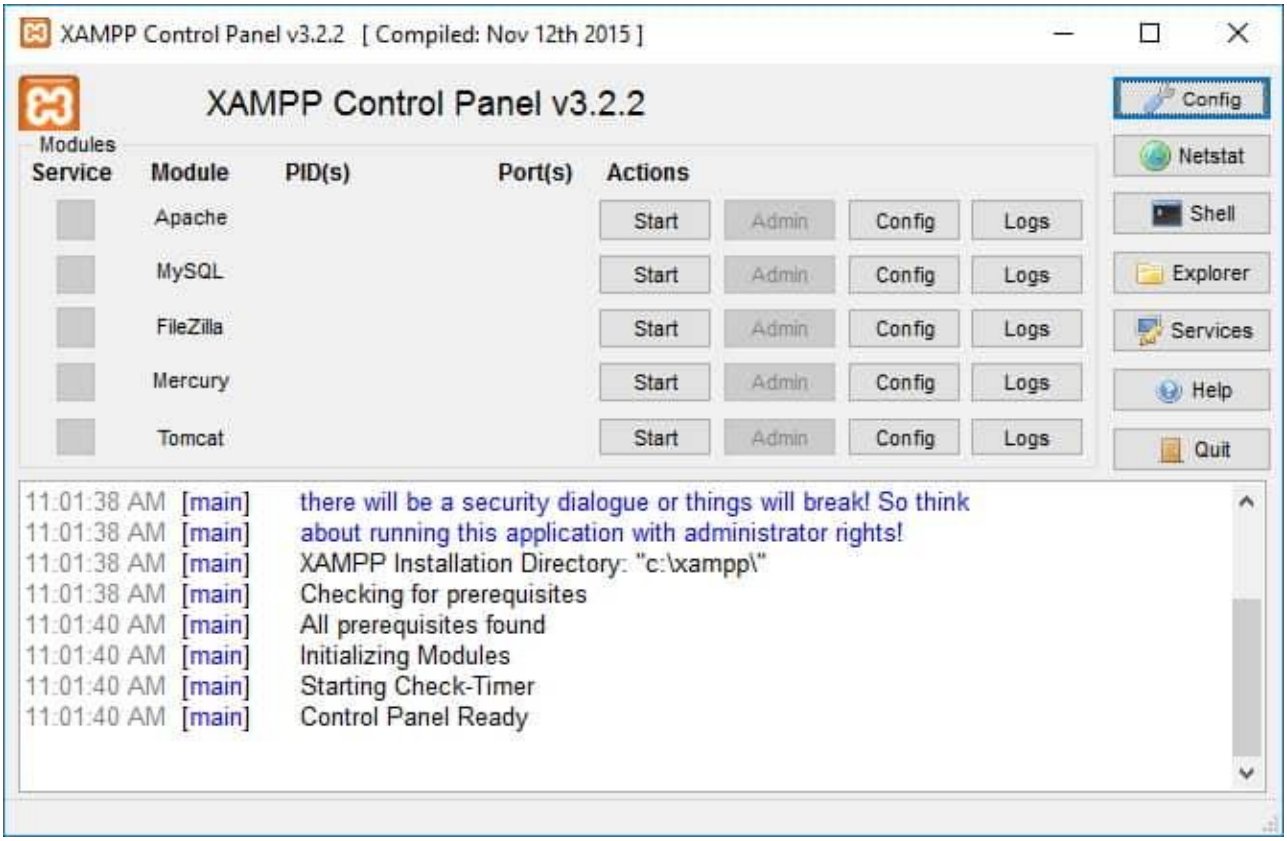
Bước 12. Nhấp vào nút Allow access để cho phép ứng dụng thông qua tường lửa (nếu có).



Bước 13. Nhấp vào nút Finish.

Bước 14. Chọn ngôn ngữ (tiếng Anh hoặc tiếng Đức).

Bước 15. Nhấp vào nút Save.



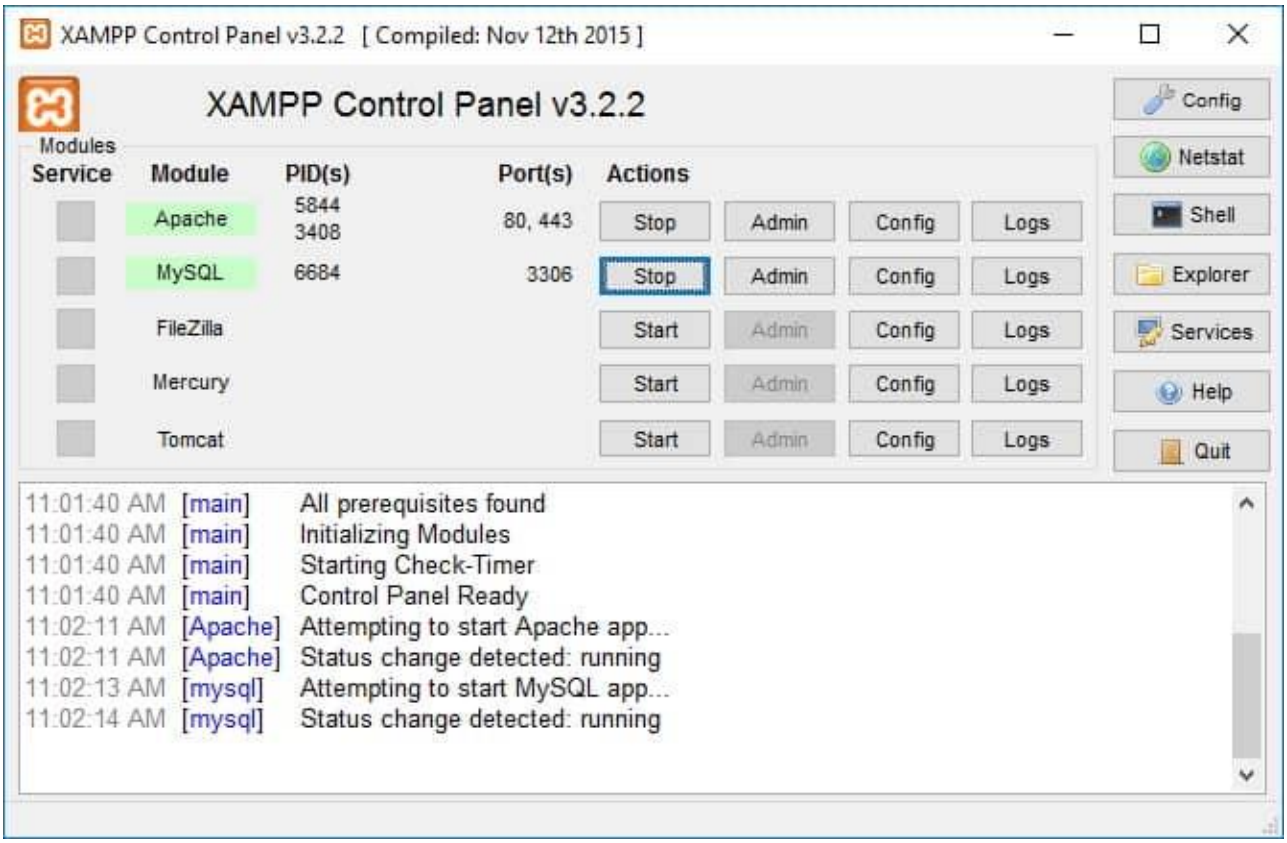
Khi bạn hoàn thành các bước, Bảng điều khiển XAMPP sẽ khởi chạy và bạn có thể bắt đầu cấu hình môi trường máy chủ web.

1.6.3. Cách cấu hình XAMPP trên Windows 10

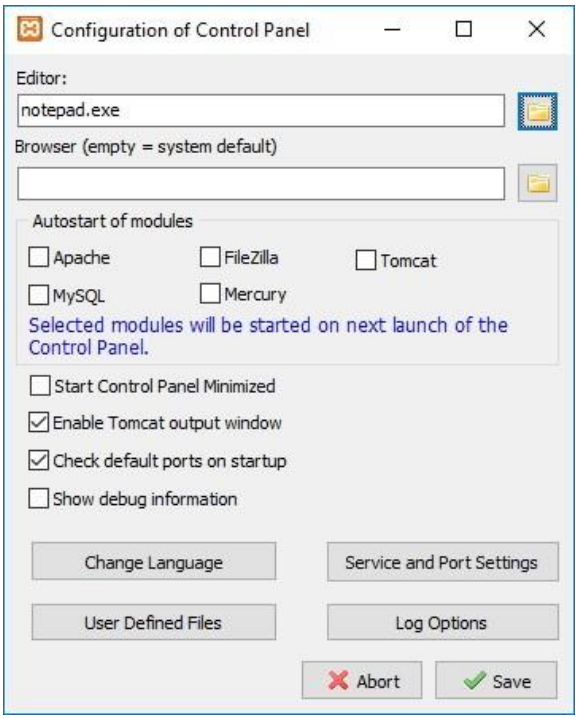
Bảng điều khiển XAMPP bao gồm ba phần chính. Trong phần Mô-đun, bạn sẽ tìm thấy tất cả các dịch vụ web có sẵn. Bạn có thể bắt đầu mỗi dịch vụ bằng cách nhấp vào nút Start.

Khi bạn bắt đầu một số dịch vụ, bao gồm cả Apache và MySQL, ở phía bên phải, bạn cũng sẽ thấy số ID quá trình (PID) và số cổng TCP/IP (Cổng) mà mỗi dịch vụ đang sử dụng. Ví dụ, theo mặc định, Apache sử dụng cổng TCP/IP 80 và 443, trong khi MySQL sử dụng cổng TCP/IP 3306 .

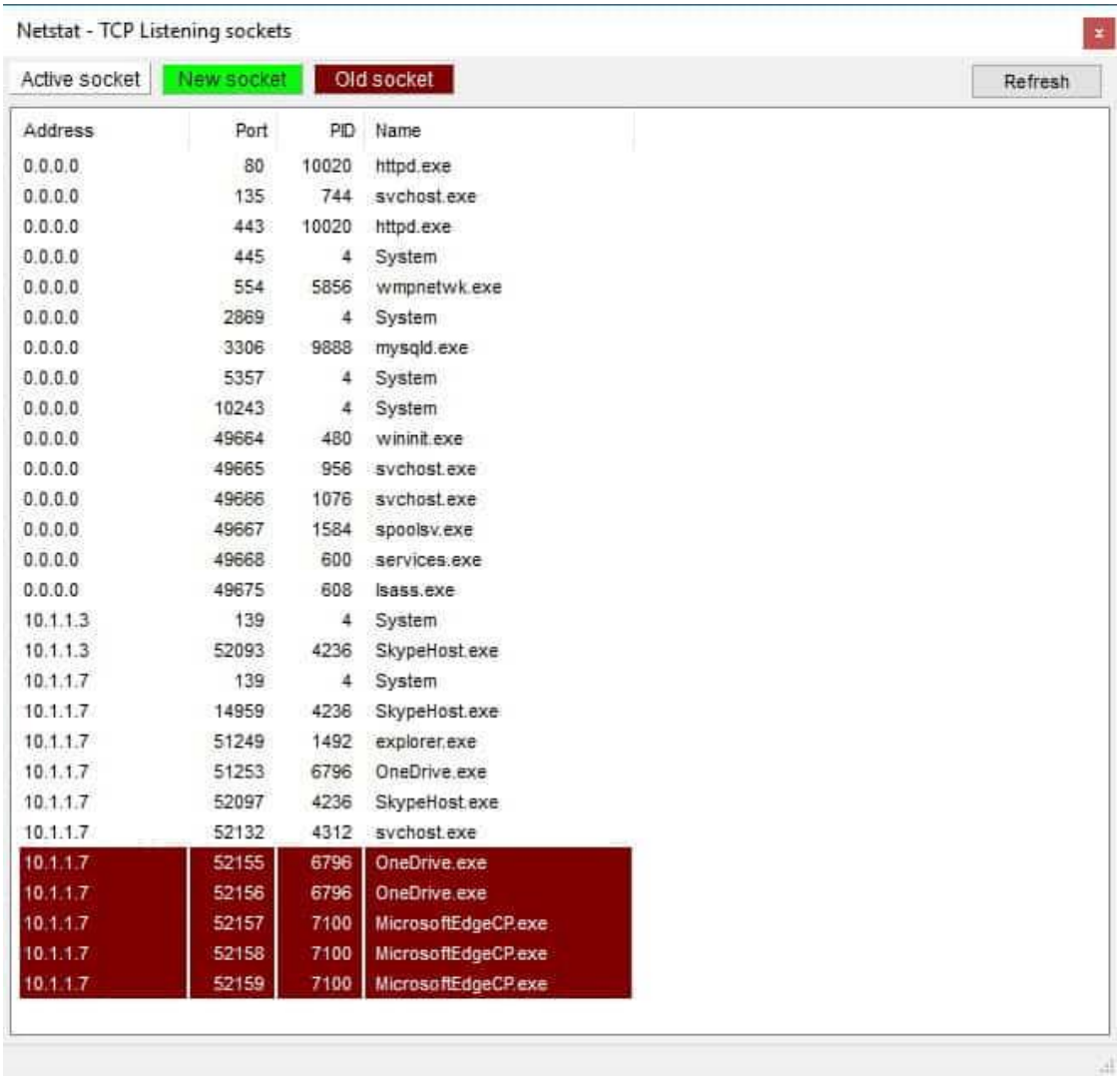
Bạn cũng có thể nhấp vào nút Admin để có quyền truy cập vào bảng điều khiển quản trị cho từng dịch vụ và xác minh rằng mọi thứ đều hoạt động chính xác.



Ở bên phải có một danh sách các nút để định cấu hình các khía cạnh khác nhau của bảng điều khiển, bao gồm nút Config để định cấu hình mô-đun nào sẽ tự động bắt đầu khi bạn khởi chạy XAMPP.



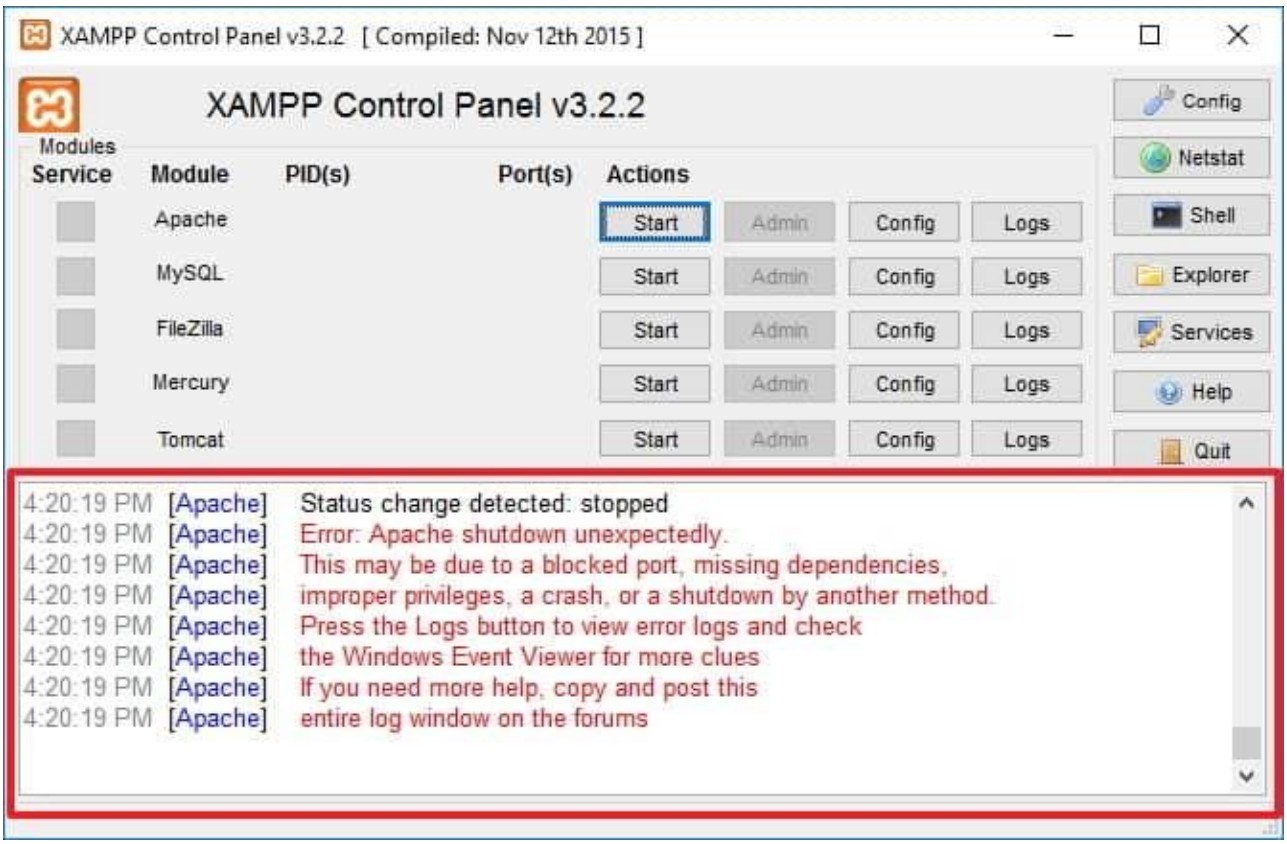
Nhấp vào nút Netstart sẽ cung cấp cho bạn danh sách các dịch vụ hiện đang truy cập mạng, bao gồm địa chỉ TCP/IP, cổng và xử lý thông tin ID.



Address	Port	PID	Name
0.0.0.0	80	10020	httpd.exe
0.0.0.0	135	744	svchost.exe
0.0.0.0	443	10020	httpd.exe
0.0.0.0	445	4	System
0.0.0.0	554	5856	wmpnetwk.exe
0.0.0.0	2869	4	System
0.0.0.0	3306	9888	mysqld.exe
0.0.0.0	5357	4	System
0.0.0.0	10243	4	System
0.0.0.0	49664	480	wininit.exe
0.0.0.0	49665	956	svchost.exe
0.0.0.0	49666	1076	svchost.exe
0.0.0.0	49667	1584	spoolsv.exe
0.0.0.0	49668	600	services.exe
0.0.0.0	49675	608	lsass.exe
10.1.1.3	139	4	System
10.1.1.3	52093	4236	SkypeHost.exe
10.1.1.7	139	4	System
10.1.1.7	14959	4236	SkypeHost.exe
10.1.1.7	51249	1492	explorer.exe
10.1.1.7	51253	6796	OneDrive.exe
10.1.1.7	52097	4236	SkypeHost.exe
10.1.1.7	52132	4312	svchost.exe
10.1.1.7	52155	6796	OneDrive.exe
10.1.1.7	52156	6796	OneDrive.exe
10.1.1.7	52157	7100	MicrosoftEdgeCP.exe
10.1.1.7	52158	7100	MicrosoftEdgeCP.exe
10.1.1.7	52159	7100	MicrosoftEdgeCP.exe

Ngoài ra, từ bảng điều khiển, cũng có các nút truy cập nhanh để mở tiện ích dòng lệnh shell, thư mục cài đặt XAMPP, dịch vụ và đóng ứng dụng.

Cuối cùng, bạn có được phần nhật ký, nơi bạn có thể xem qua những gì xảy ra mỗi khi bạn khởi động một mô-đun hoặc thay đổi cài đặt. Đây cũng là nơi đầu tiên để xem xét khi một cái gì đó không hoạt động.



Các cài đặt mặc định sẽ hoạt động đối với hầu hết mọi người sử dụng XAMPP để tạo môi trường thử nghiệm để chạy trang web. Tuy nhiên, tùy thuộc vào cấu hình thiết lập của bạn, bạn có thể cần thay đổi số cổng TCP/IP cho máy chủ Apache, kích thước tải lên cơ sở dữ liệu hoặc đặt mật khẩu cho phpMyAdmin.

Để thay đổi các cài đặt này, bạn sẽ cần sử dụng nút Config cho dịch vụ tương ứng. Ví dụ: bạn sẽ cần mở tệp httpd.conf để thay đổi cài đặt trên máy chủ Apache và tệp my.ini để thay đổi cài đặt MySQL.

1.6.4. Cách sửa lỗi Apache không start trên XAMPP

Việc cài đặt XAMPP rất đơn giản, nhưng đôi khi Windows 10 có thể không cho phép máy chủ Apache chạy. Thông thường, đó là vì World Wide Publishing Service đang chạy trên cổng 80 trên máy tính của bạn, đây cũng là cổng TCP/IP mặc định mà Apaches sử dụng trên XAMPP và hai ứng dụng không thể sử dụng cùng một cổng.

Nếu bạn gặp phải vấn đề này, có một vài cách để khắc phục nó. Bạn có thể gỡ cài đặt dịch vụ xuất bản hoặc bạn có thể thay đổi cổng mặc định trên Apache.

1.6.4.1. Gỡ cài đặt World Wide Web Services

Nếu bạn không có bất kỳ ứng dụng nào tùy thuộc vào tính năng World Wide Web Services, bạn có thể gỡ cài đặt ứng dụng đó bằng các bước sau:

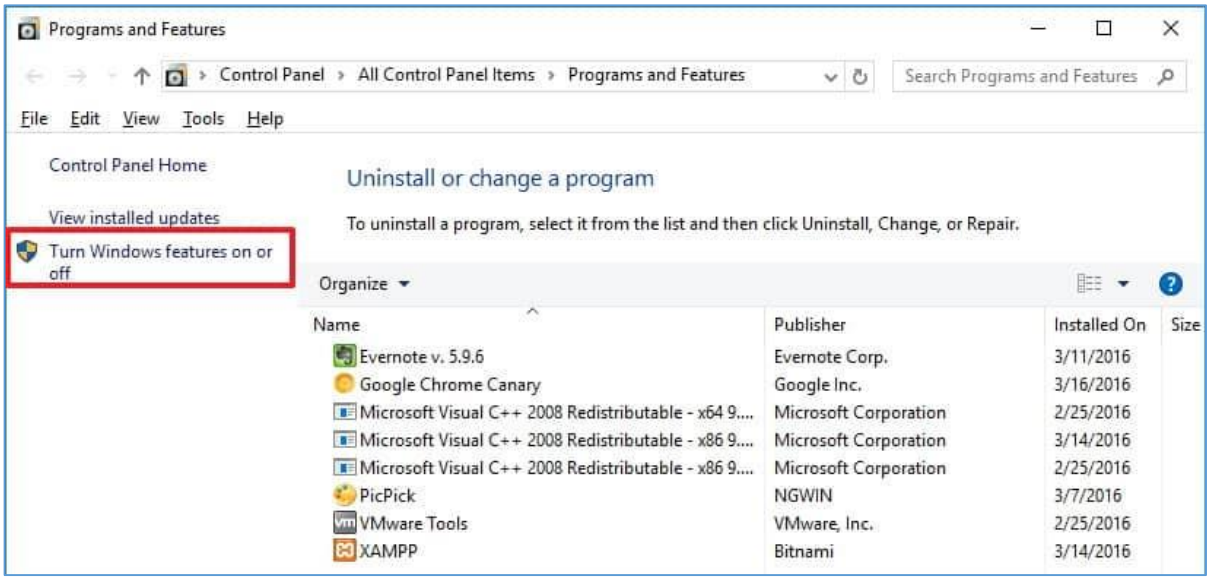
1) Mở Start.

2) Tìm kiếm Control Panel và nhấp vào kết quả trên cùng để mở trải nghiệm.

3) Bấm vào Programs.

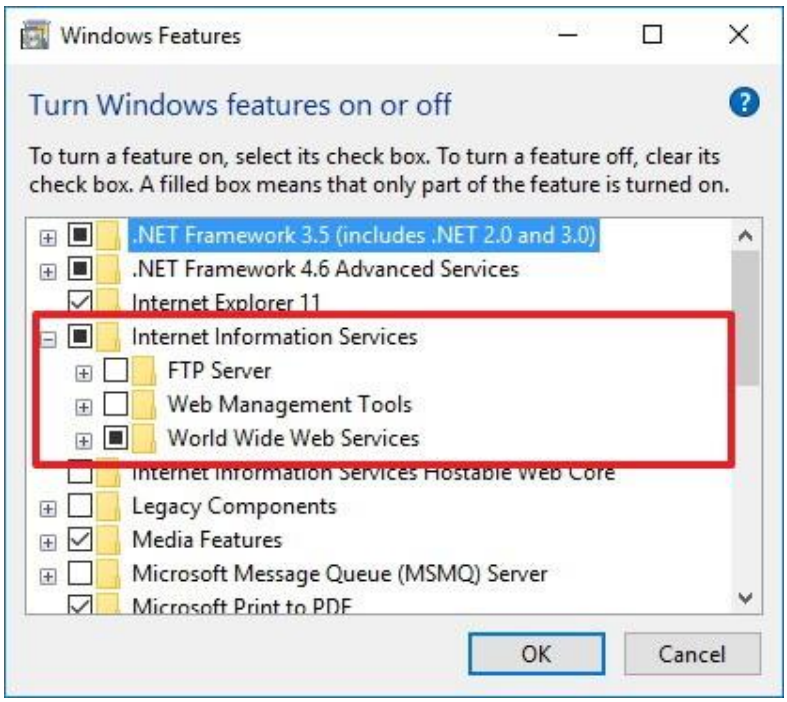
4) Nhấp vào Programs and Features.

5) Nhấp vào Turn Windows features on or off của Windows.



6) Mở rộng Internet Information Services và xóa tùy chọn World Wide Web Services.

7) Nhấp vào nút OK .



8) Khởi động lại máy tính của bạn.

9) Mở XAMPP Control Panel.

10) Nhấp vào nút Start trên Apache.

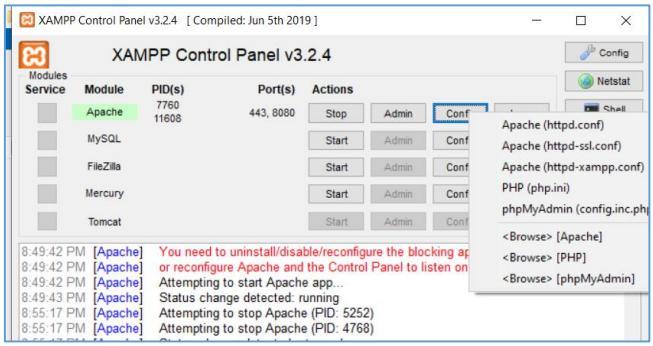
Sau khi bạn hoàn thành các bước, Apache sẽ có thể chạy trong cổng 80 trên thiết bị của bạn.

1.6.4.2. Thay đổi cổng TCP/IP mặc định của Apache

Ngoài ra, có thể định cấu hình Apache để chạy trên một cổng TCP/IP khác bằng các bước sau:

1) Mở XAMPP.

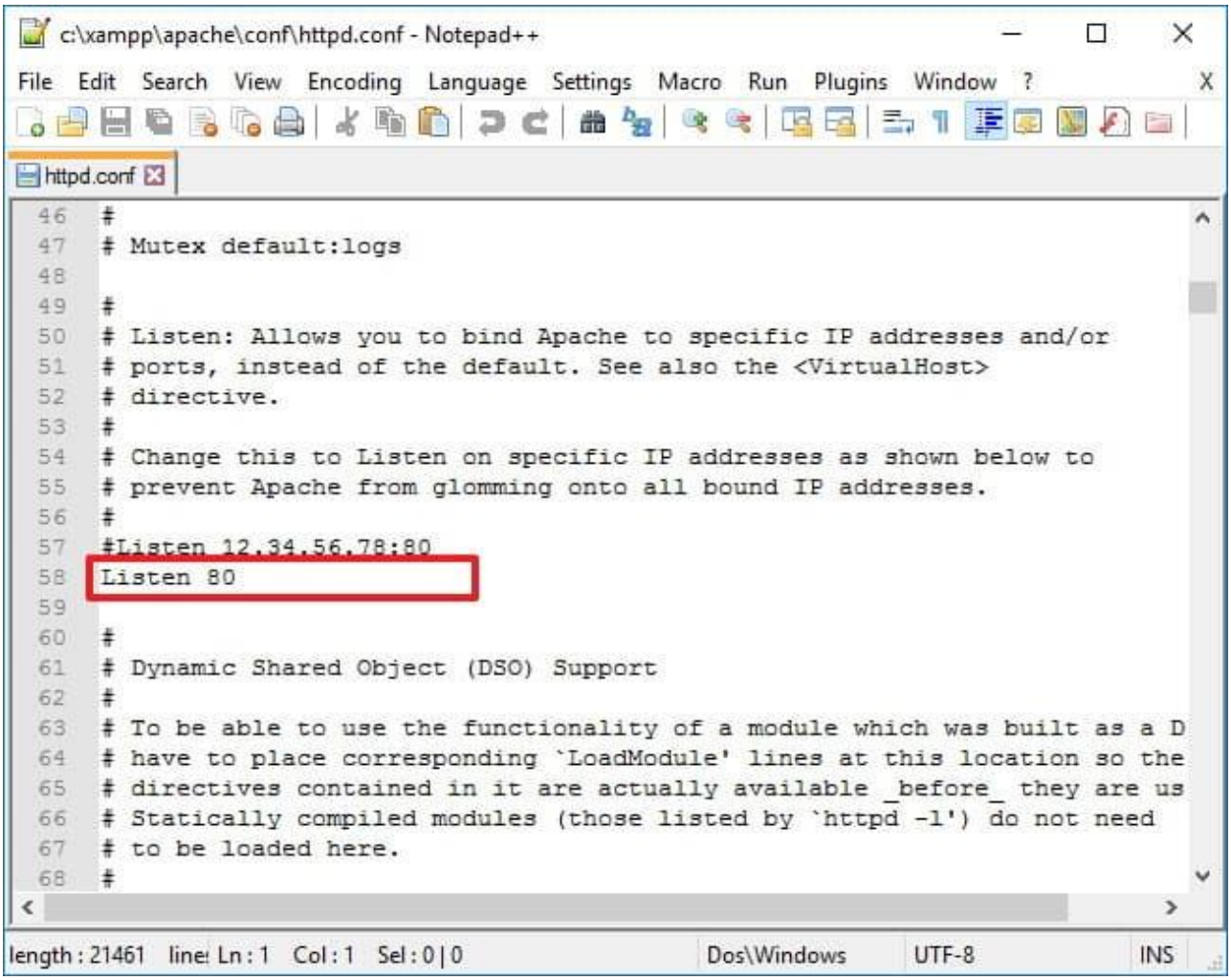
2) Nhấp vào nút Config trên Apache.



3) Nhấp vào tùy chọn Apache (httpd.conf).

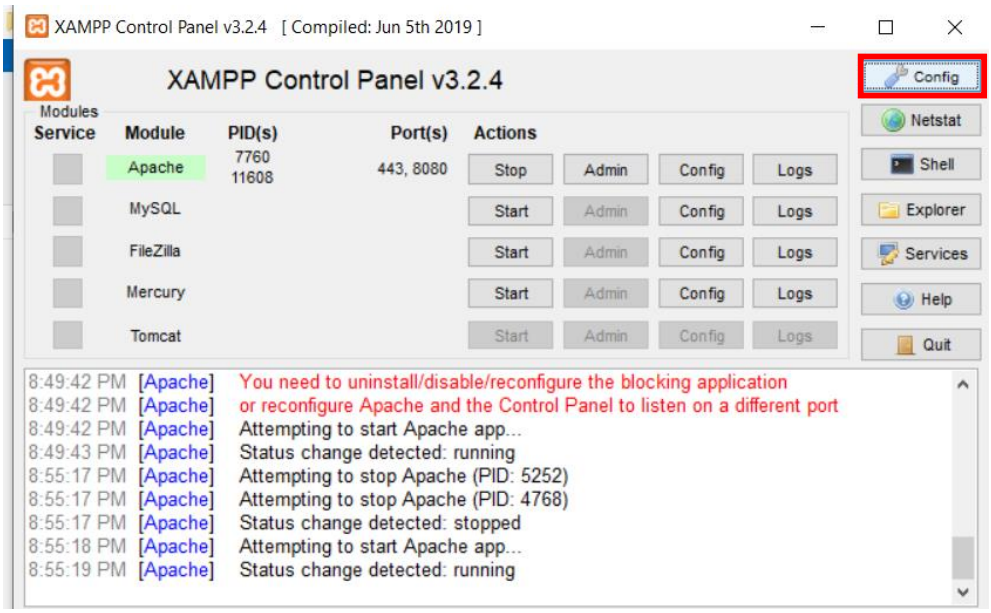
4) Cuộn xuống và tìm dòng ***Listen 80*** và thay bằng ***Listen 8080***

5) Cuộn xuống và tìm dòng ***ServerName localhost: 80*** và thay bằng ***ServerName localhost: 8080***

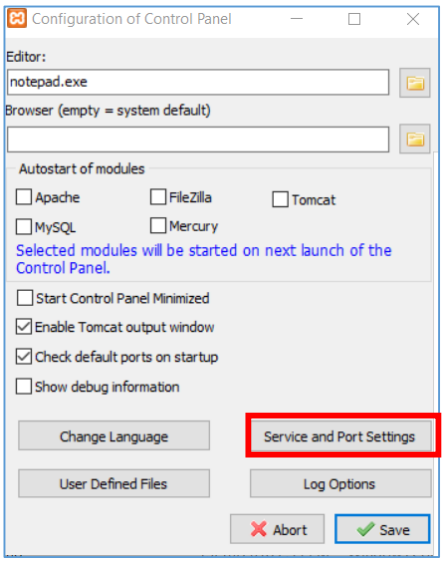


6) Lưu và đóng tệp Apache (httpd.conf).

7) Trên cửa sổ Xampp, nhấp vào nút Config trên cùng bên phải.



8) Trong cửa sổ Configuration of Control Panel chọn Service and Port Settings



9) Trong cửa sổ Service Settings sử ***Main Port*** thành ***8080.***

10) Lưu để thoát khỏi cửa sổ Service Settings và cửa sổ Configuration of Control Panel.

Sau hoàn thành các bước, máy chủ Apache sẽ chạy mà không gặp sự cố nào trên cổng TCP/IP mới đã được chỉ định.

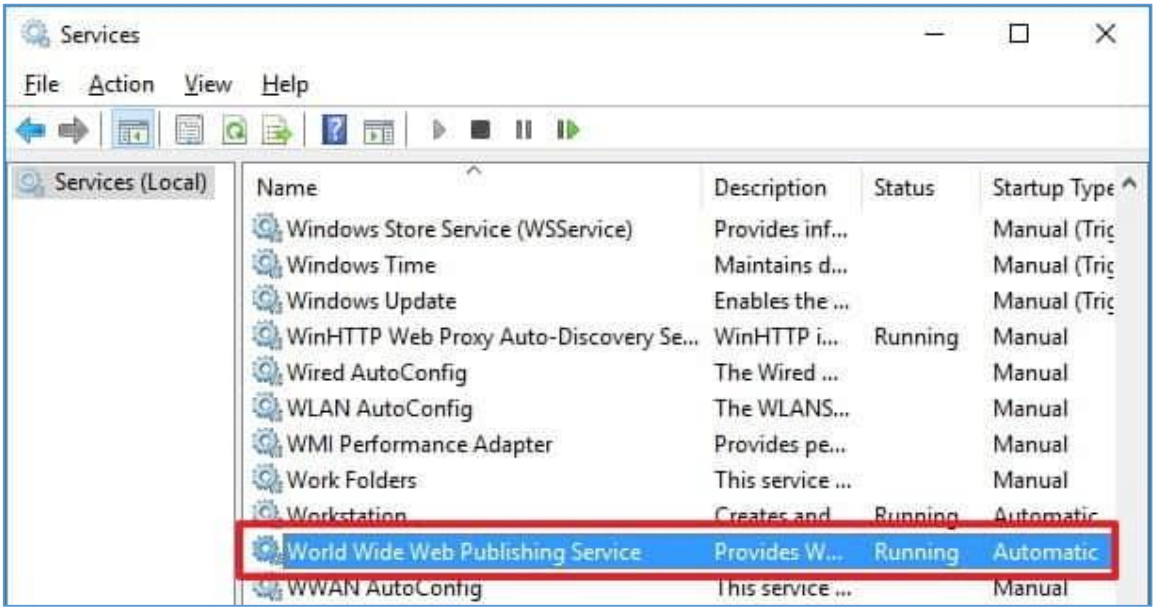
1.6.4.3. Dừng thủ công World Wide Web Publishing Service

Một cách khác để khắc phục sự cố cổng là dừng dịch vụ và thay đổi cài đặt của nó để chỉ khởi động dịch vụ theo cách thủ công bằng các bước sau:

1) Mở Start.

2) Tìm kiếm Services và nhấp vào kết quả hàng đầu để mở trải nghiệm.

3) Bấm đúp vào World Wide Web Publishing Service.

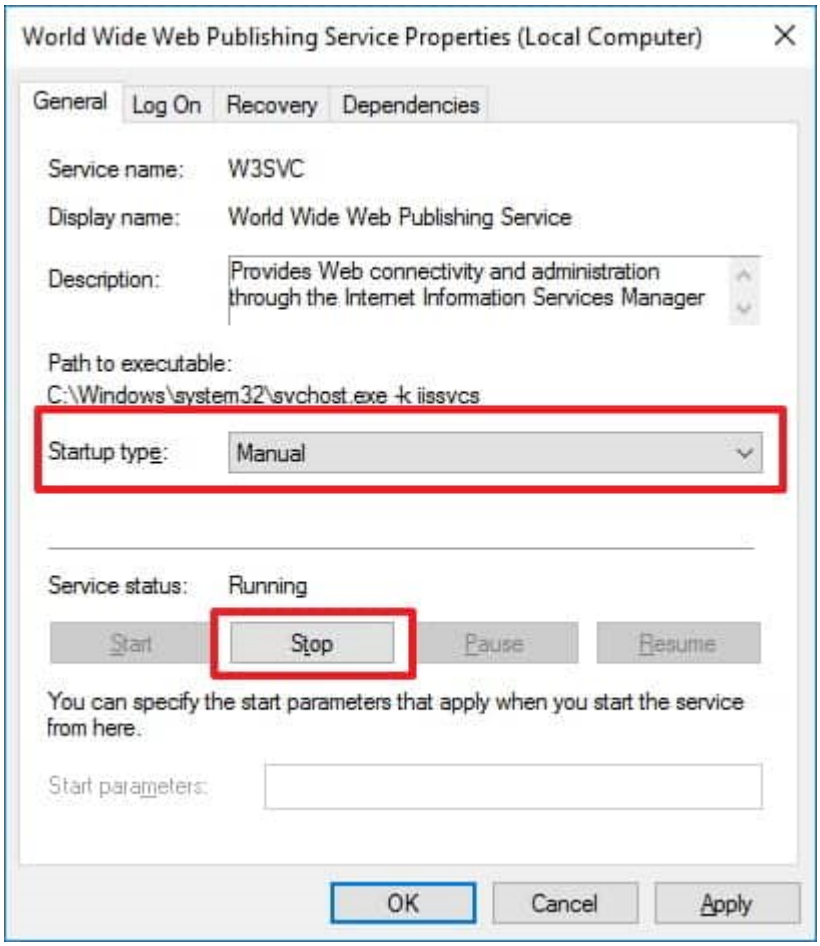


4) Nhấp vào Stop.

5) Thay đổi loại khởi động thành Manual.

6) Nhấp vào nút Apply.

7) Nhấp vào nút OK.



8) Mở XAMPP Control Panel.

9) Nhấp vào nút Start để chạy máy chủ Apache.

Cách tốt nhất để khiến Apache hoạt động trở lại là bằng cách gỡ cài đặt World Wide Web Publishing Service, nhưng khi không thể, bạn có thể thay đổi số cổng TCP/IP hoặc dừng dịch vụ khi cần thiết.

1.6.5. Cách tăng giới hạn upload phpMyAdmin trên XAMPP

1) Mở XAMPP Control Panel.

2) Nhấp vào nút Explorer.

3) Mở thư mục php .

4) Mở tệp php.ini bằng bất kỳ trình soạn thảo văn bản nào.

5) Thay đổi giá trị kích thước mà bạn muốn, chẳng hạn.

```
post_max_size = 200M
upload_max_filesize = 200M
max_execution_time = 3000
max_input_time = 3000
```

```
memory_limit = 200M
```

6) Nhấp vào nút Stop cho MySQL và Apache.

7) Nhấp vào nút Start để khởi động lại Apache và MySQL.

Khi bạn hoàn thành các bước, bây giờ bạn có thể nhập các tệp cơ sở dữ liệu lớn trên phpMyAdmin.

1.6.6. Cách thay đổi mật khẩu phpMyAdmin trên XAMPP

Theo mặc định, phpMyAdmin (MySQL) sử dụng user root không có mật khẩu, có nghĩa là nếu bạn đang cài đặt một trang web WordPress, khi được hỏi bạn chỉ cần nhập root làm tên người dùng và để trống trường mật khẩu.

Sử dụng các bước sau để thay đổi mật khẩu phpMyAdmin trên XAMPP:

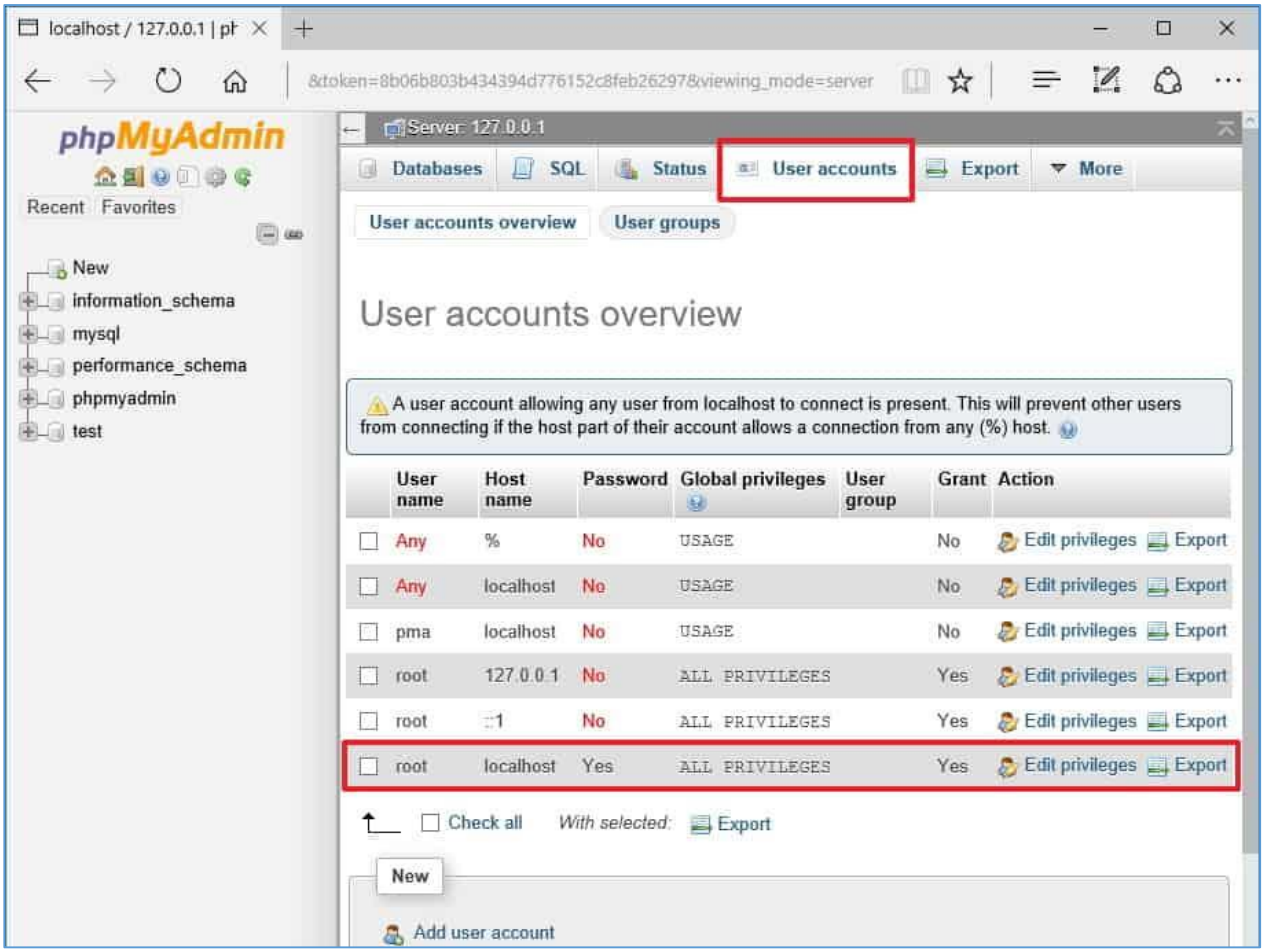
1) Mở XAMPP Control Panel.

2) Nhấp vào nút Admin cho MySQL.

3) Nhấp vào tab User accounts.

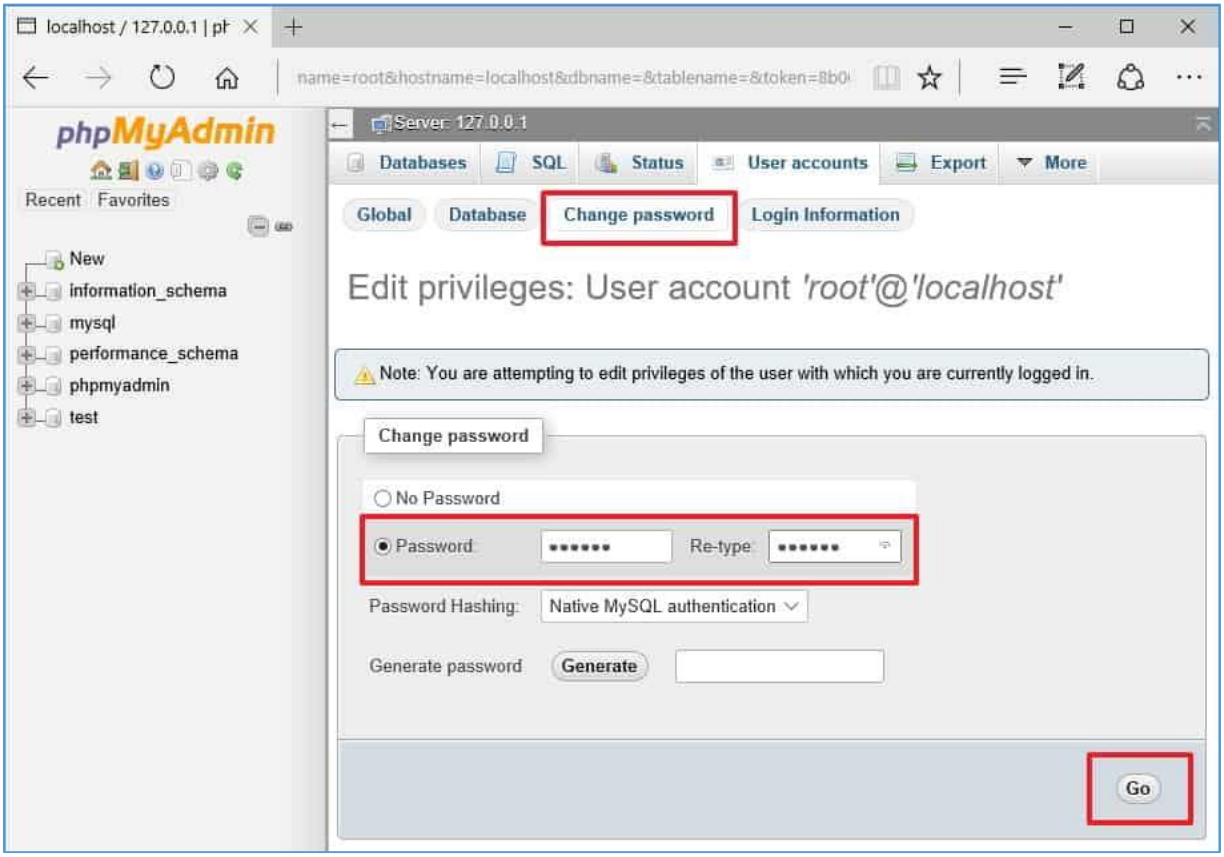
4) Nhấp vào liên kết Edit privileges cho user root nhưng liên kết có tên máy chủ localhost .

5) Nhấp vào nút Change password.

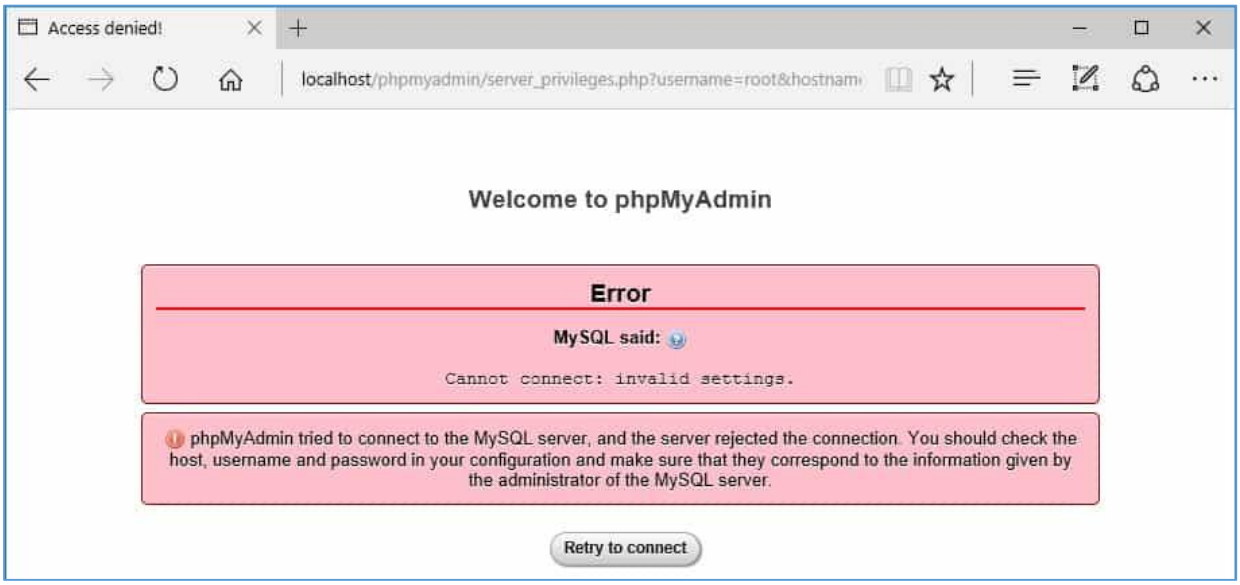


6) Chỉ định mật khẩu mới.

7) Nhấp vào nút Go.



Sau khi bạn hoàn thành các bước, nếu bạn cố gắng đăng nhập vào phpMyAdmin, bạn sẽ nhận được một thông báo từ chối truy cập: “Cannot connect: invalid settings“.



1.6.7. Sửa lỗi “Cannot connect: invalid settings” trên phpMyAdmin

Sử dụng các bước sau để thay đổi cài đặt cấu hình trong tệp config.inc.php để khắc phục sự cố đăng nhập:

1) Mở XAMPP Control Panel.

2) Nhấp vào nút Explorer .

3) Trong thư mục XAMPP, mở thư mục phpMyAdmin .

4) Mở tệp config.inc.php bằng bất kỳ trình soạn thảo văn bản nào.

5) Bên dưới loại và thông tin xác thực, hãy và cập nhật các dòng sau:

Tìm đoạn

```
$cfg['Servers'][$i]['auth_type'] = 'config';
```

thay thế

```
config
```

thành

```
cookie
```

Tìm đoạn

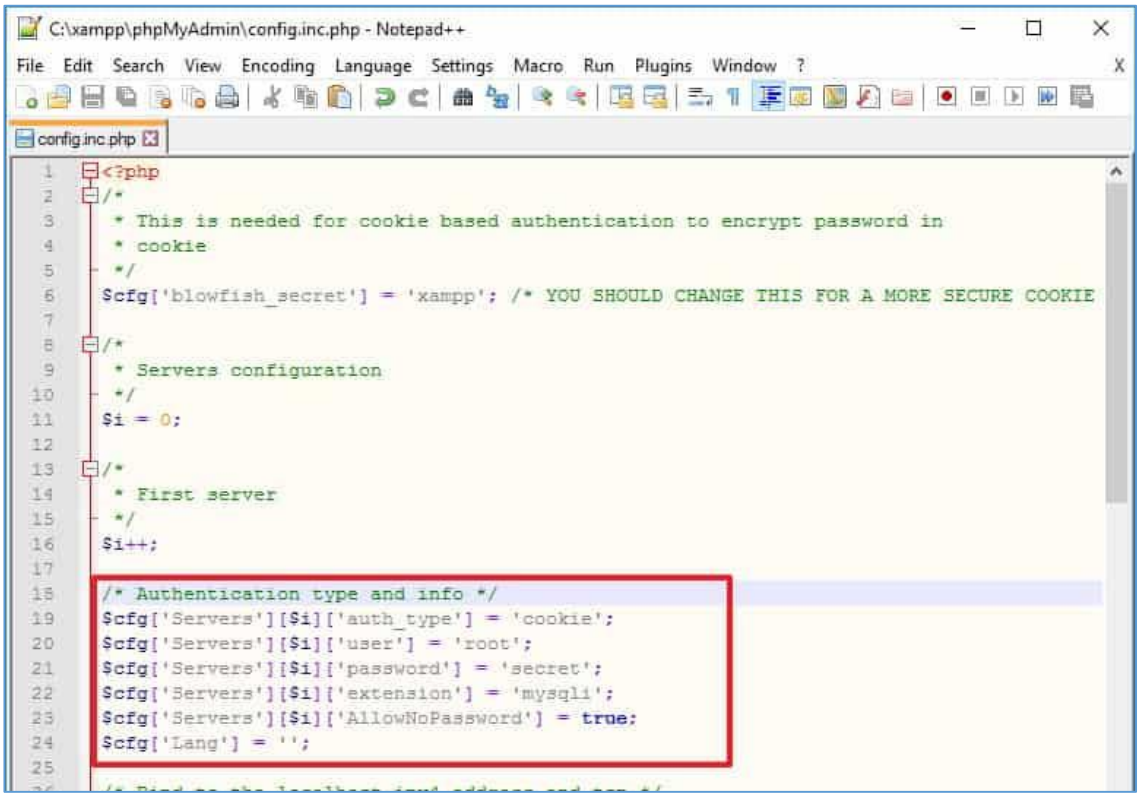
```
$cfg['Servers'][$i]['AllowNoPassword'] = true;
```

thay thế

```
true
```

thành

```
false
```



6) Lưu và đóng tập tin.

Khi hoàn thành các bước, ta có thể đăng nhập vào bảng điều khiển phpMyAdmin.

Bây giờ XAMPP đã được cấu hình và sẵn sàng để sử dụng với bất kỳ phần mềm dựa trên PHP nào được hỗ trợ.

Chương 2. Các thành phần cơ bản

2.1. Cấu trúc chương trình Javascript

2.1.1. Khối lệnh và dòng lệnh

Giống như các ngôn ngữ họ C khác như Java, .NET, v.v .JavaScript cũng tương tự, các khối mã lệnh được đặt trong cặp dấu ngoặc nhọn { code block }. Kết thúc mỗi dòng lệnh sẽ là dấu chấm phẩy ;.

Ví dụ 2.1. Khối lệnh và dòng lệnh.

```
function doSomethingCool() {
  // Đây là một khối lệnh trong cặp dấu ngoặc nhọn.
  // Dòng lệnh trong khối lệnh kết thúc bằng dấu chấm phẩy.
  console.log('Line 1');
  console.log('Line 2');
  console.log('Line 3');
}
```

2.1.2. Từ khóa

Từ khóa là từ có sẵn trong Javascript, khi lập trình ta không được sử dụng từ khóa để đặt tên. Trong Javascript có bộ từ khóa là:

Từ khóa	Ý nghĩa
break	Thoát khỏi câu lệnh Switch hoặc vòng lặp
continue	Nhảy ra khỏi vòng lặp hiện tại và quay trở lại đầu vòng lặp kế tiếp
debugger	Dùng để đặt một break point khi debug
do ... while	Thực thi câu lệnh trong khối Do khi điều kiện trong While còn đúng
if ... else	Khối lệnh điều kiện rẽ nhánh.
return	Trả về giá trị và thoát khỏi một hàm.
switch	Đánh dấu một block gồm nhiều khối lệnh khác nhau sẽ được thực thi phụ thuộc vào từng trường hợp cụ thể của dữ liệu vào.
try ... catch	Bắt và xử lý lỗi (Ngoại lệ)

var	Định nghĩa biến.
…	…

Ví dụ 2.2.

```
var a = 1;
var b = 2;

console.log(a + b); // 3
```

Câu lệnh trên khi thực thi sẽ in ra màn hình terminal giá trị là 3.

2.1.3. Comment (Viết bình luận hoặc mô tả cho mã lệnh)

Khi viết chương trình thì việc comment hay viết lời bình luận cho câu lệnh là không thể thiếu. Cũng như các ngôn ngữ khác JS có hai kiểu comment là comment trên một dòng hoặc comment trên nhiều dòng.

```
comment trên một dòng bắt đầu với //
còn comment trên nhiều dòng bắt đầu với /* và kết thúc với * /
```

Ví dụ 2.3. Viết document cho hàm Add thực hiện việc cộng hai số a và b.

```
/**
* @name Add
* @description
* Add two given number {a} and {b} .
*
* @param  {number} a First number
* @param  {number} b Second number
* @constructor
*/
function Add(a, b) {
  // return sum of a and b.
  return a + b;
};
```

2.2. Biến

2.2.1. Giới thiệu

Biến (variable) là một "vị trí bộ nhớ" (Memory location) mà một chương trình sử dụng để lưu trữ các giá trị. Tên của biến được gọi là một định danh (identifier).

Tên của biến phải được đặt theo quy tắc, dưới đây là các quy tắc để đặt tên biến:

- Tên biến không được phép giống với các từ khóa (keyword).
- Tên biến có thể chứa các chữ cái (alphabets) hoặc các con số, nhưng không bắt đầu bởi một con số.

- Tên biến không thể chứa các khoảng trắng (white space), hoặc các ký tự đặc biệt ngoại trừ ký tự ngạch dưới (underscore) (_) và ký tự dolar ($).

2.2.2. Khai báo biến

Khai báo biến là cần thiết vì bạn cần phải làm như vậy trước khi sử dụng nó. Cú pháp **ES5** sử dụng từ khóa **var** để khai báo một biến. **ES6** thêm vào 2 từ khóa mới là **let** & **const** để khai báo một biến.

Ví dụ 2.4.

```
// Declare variables:
var a1;
console.log(a1); // undefined

var a2 = 100;
console.log(a2); // 100


var a3, a4;
var a5, a6 = 200;

console.log(a5); // undefined
console.log(a6); // 200

var a7= 100, a8 = 300;


let b1;
console.log(b1); // undefined

let b2 = "Hello";
console.log(b2); // Hello



const c = 100;
console.log(c); // 100
```

- **Khối lệnh**

Khối lệnh (block) là một một tập hợp các lệnh nằm trong một "cặp ngoặc xoắn" (Curly brackets) { }.

- **let**

Từ khóa let được sử dụng để khai báo một biến có phạm vi khối (block scope), điều này có nghĩa là biến này sẽ được chương trình nhận biết trong khối đó hoặc bên trong các khối con, nhưng không được chương trình nhận biết bên ngoài khối đã định nghĩa ra nó.

Ví dụ 2.5.

```
if(true) {
   let a = 200;
   console.log(a); // 200
}
// Program will ignore this statement:
console.log(a);
```

Nếu sử dụng từ khóa let để định nghĩa 2 biến có tên giống nhau, một biến khai báo ở khối cha, một biến khai báo ở khối con, chương trình sẽ coi đó là 2 biến khác nhau.

Ví dụ 2.6.

```
let i = 1000;
let j = 2000;
if(true)  {
   i = 100; // Assign new value to 'i'
   let j = 200; // A new variable (***)
   console.log("Test1: " + i + " : " + j); // Test1: 100 : 200
}
console.log("Test2: "+ i + " : " + j); // Test2: 100 : 2000
```

- **var**

Từ khóa var được sử dụng để khai báo một biến, biến này có phạm vi (scope) rộng hơn so với biến let. Nó được chương trình nhận biết ở bên trong khối định nghĩa ra nó, trong các khối con, thậm trí được nhận biết ở bên ngoài khối đã khai báo nó.

Ví dụ 2.7.

```
if(true) {
   var a = 200;
   console.log(a); // 200
}
console.log(a); // 200
```

Nếu khai báo 2 biến cùng tên với từ khóa var, một biến khai báo trong khối cha, một biến trong khối con, chương trình sẽ coi 2 biến đó là giống nhau (Cùng một vị trí trên bộ nhớ).

Ví dụ 2.8.

```
var i = 1000;
var j = 2000;
if(true)  {
   i = 100; // Assign new value
   var j = 200; // Assign new value
   console.log("Test1: " + i + " : " + j); // Test1: 100 : 200
```

```
}
console.log("Test2: "+ i + " : " + j); // Test2: 100 : 200
```

Các biến được khai báo với từ khóa var trong một hàm sẽ chỉ được chương trình nhận biết trong hàm đó, nó không được nhận biết bên ngoài hàm.

Ví dụ 2.9.

```
// A function
var test = function()  {
    var a = 200;

    console.log(a); // 200
}

// Call function.
test();

console.log(a); // Not work!!
```

- **const**

Từ khóa const được sử dụng để khai báo một hằng số (constant). Khi khai báo một hằng số bạn phải gán luôn giá trị cho nó. Nhưng bạn không thể gán giá trị mới cho biến này. Chú ý: Giống với biến let, biến const có phạm vi khối (block scope).

Ví dụ 2.10.

```
// Declare a constant with a value
const greeting = "Hello";
// Assign new value to 'greeting'
greeting = "Hi"; // ==> Error!!
// Declare a constant without a value
const i  ; // ==> Error!!
```

Một biến được khai báo với từ khóa const, nó sẽ là một hằng số theo nghĩa bạn không thể gán giá trị mới cho nó, nhưng nó không phải là bất biến (imutable), bạn vẫn có thể thay đổi các property của nó.

Hãy xem ví dụ 2.11, một biến được khai báo với từ khóa const, giá trị của nó là một đối tượng có nhiều property. Bạn có thể gán giá trị mới cho các property của đối tượng này.

Ví dụ 2.11.

```
// Declaring a constant is an object
const person = {
    name: "Clinton",
    gender : "Female"
};
```

```
console.log(person.name); // Clinton

// Can assign new values to properties of const object.
person.name = "Trump";
person.gender = "Male";

console.log(person.name); // Trump


// ** Error! (Cannot assign new value to const variable).
person =  {
   name : "Trump";
}
```

+ const: Object.freeze()

Phương thức Object.freeze() giúp đóng băng (freeze) một đối tượng, bạn không thể thay đổi gán giá trị mới cho các property của nó.

Ví dụ 2.12.

```
// Declaring a constant is an object
const person = {
    name: "Clinton",
    gender : "Female"
};

console.log(person.name); // Clinton

Object.freeze(person); // Freeze object 'person'.

person.name = "Trump";

console.log(person.name); // Clinton
```

2.3. Kiểu dữ liệu

2.3.1. Giới thiệu

Kiểu dữ liệu là một cách phân loại dữ liệu cho trình biên dịch hoặc thông dịch hiểu các lập trình viên muốn sử dụng dữ liệu. Hầu hết các ngôn ngữ hỗ trợ nhiều kiểu dữ liệu khác nhau, như só thực, nguyên hay Boolean. Một kiểu dữ liệu cung cấp một bộ các giá trị mà từ đó một biểu thức (ví dụ như biến, hàm…) có thể lấy giá trị của nó. Kiểu định nghĩa các toán tử có thể được thực hiện trên dữ liệu của nó, ý nghĩa của dữ liệu, và cách mà giá trị của kiểu có thể được lưu trữ.

Kiểu dữ liệu trong javascript không phức tạp như trong các ngôn ngữ lập trình khác. Ví dụ là kiểu số (Number) thì nó không phân biệt kiểu nguyên hay không nguyên gì cả. Javascript có 6 kiểu dữ liệu cơ bản:

- **number:** dữ liệu kiểu số, các số bất kỳ loại nào, số nguyên hoặc dấu phẩy động.
- **string:** dữ liệu kiểu chuỗi, chuỗi có thể có một hoặc nhiều ký tự, không có loại ký tự đơn riêng biệt.
- **boolean:** dữ liệu kiểu logic cho giá trị ***true/false***.
- null: dữ liệu kiểu rỗng biểu diễn các giá trị không xác định – một loại độc lập có một giá trị duy nhất null.
- undefined: dữ liệu kiểu chưa xác định, các giá trị chưa được gán – một kiểu độc lập có một giá trị duy nhất undefined.
- object: dữ liệu kiểu đối tượng các cấu trúc dữ liệu phức tạp hơn.

2.3.2. Dữ liệu kiểu Number

2.3.2.1. Khai báo

Number là dữ liệu kiểu số, không phân biệt số nguyên hay số thực.

Ví dụ 2.13.

```
var x = 20;
// gán giá trị cho x = 20, vậy x là kiểu Number
var y = 20.234;
// gán giá trị cho y = 20.234 vậy y là kiểu Number
console.log(x);
//Hiện thị x ra màn hình console của trình duyệt
console.log(y);
//Hiện thị y ra màn hình console của trình duyệt
console.log(x+y);
//Hiện thị tổng x + y ra màn hình console của trình duyệt
```

Một số "giá trị số đặc biệt" : Infinity và NaN

- Infinity: không xác định

```
console.log(2/0) //infinity
```

- NaN là viết tắt Not And Number nghĩa là không phải số.

```
console.log('cmm'/2); //NaN
```

2.3.2.2. Đối tượng Math để thao tác trên số.

Trong JS Math là một biến Global các bạn có thể dùng luôn mà không cần khai báo.

- **Làm tròn số:**

Để làm tròn số thì trong JS có hỗ trợ 3 kiểu, làm tròn gần nhất (round), làm tròn lên (ceil) và làm tròn xuống (floor).

Làm tròn gần nhất sẽ trả về giá trị nguyên gần nhất với giá trị hiện tại.

```
Math.round(4.7);    // trả về 5
Math.round(4.4);    // trả về 4
```

Làm tròn lên và làm tròn xuống.

```
Math.floor(4.7);    // trả về 4
Math.ceil(4.4);     // trả về 5
```

- **Tính trị tuyệt đối**

```
Math.abs(-5); // trả về 5
```

- **Tính sin và cos**

Hàm sin và cos nhận và giá trị radians, nếu bạn muốn dùng bằng giá trị độ, thì đơn giản là nhân cho số PI rồi chia cho 180.

```
Math.sin(90 * Math.PI / 180);     // trả về 1 (Sin góc 90 độ)
```

- **Tính giá trị mũ (x mũ y)**

```
Math.pow(8, 2);      // trả về 64 vì là 8 mũ 2
```

- **Tính giá trị căn bật hai**

```
Math.sqrt(64);      // trả về 8
```

- **Lấy giá trị ngẫu nhiên**

Hàm random() sẽ trả về ngẫu nhiên một giá trị trong khoảng từ 0 đến 1. Chẳng hạn để lấy ngẫu nhiên một số trong khoảng từ 0 đến 1000 thì có thể làm như sau.

```
Math.round(Math.random() * 1000);
```

2.3.3. Kiểu dữ liệu String

2.3.3.1. Khai báo

String là kiểu chuỗi. Hằng chuỗi được đặt trong cặp dấu nháy đơn'…' hoặc nháy kép "…", sử dụng dấu + để nối các chuỗi lại với nhau.

Ví dụ 2.14.

```
var x = "con ga" //x được gán giá trị "con ga", vậy x là kiểu String
var y = 'mai' // y được gán giá trị là "mai", vậy x là kiểu String
var z = '911' // z được gán giá trị là '911', vậy z là kiểu String
console.log(x+y+z+'20kg')
// Hiện thị trên màn hình console "con ga mai 911 20kg"
```

Một số ngôn ngữ có kiểu dữ liệu kí tự, riêng Javascript không có kiểu dữ liệu ký tự, Javascript chỉ có một loại string. Một chuỗi có thể chỉ bao gồm một ký tự hoặc nhiều ký tự.

2.3.3.2. Các thuộc tính và phương thức hay dùng trên chuỗi

- Lấy độ dài chuỗi.

```
var _string = 'THIS IS STRING';
_string.length // Get string length.
```

- Viết thường hoặc viết hoa tất cả.

```
var _string = 'THIS IS STRING';
var a = _string.toLowerCase(); // a = 'this is string'
var b = _string.toUpperCase(); // b = 'THIS IS STRING'
```

- Cắt bỏ ký tự trống hai đầu.

Phương thức này được dùng thường xuyên để xử lý chuỗi vào, tránh bị dư thừa ký tự trống hai đầu.

```
var _string = '         THIS IS STRING         ';
var a = _string.trim(); // a = 'THIS IS STRING'
```

2.3.4. Boolean (kiểu logic)

Boolean nó chỉ trả về 2 giá trị là ” True ” or ” False ”

Ví dụ 2.15.

```
var x = true; // x được gán giá trị cho true, vậy x là kiểu Boolean
var y = false; // y được gán giá trị cho false, vậy y là kiểu Boolean
var m = 1;
var n = 2;
console.log(m = n);
// Hiện thị ra màn hình console của trình duyệt kết quả m = n,
// ở đây nó sẽ trả về false
```

2.3.5. Null

Null là giá trị đặc biệt không thuộc bất kỳ loại nào được mô tả ở trên. Nó chỉ là một giá trị đặc biệt có ý nghĩa “không có gì”, “trống rỗng” hoặc “giá trị không rõ”.

Ví dụ 2.16.

```
var n = null; // n được gán giá trị cho null, vậy n là kiểu Null
console.log(n);
// Hiện thị ra màn hình console của trình duyệt giá trị của n,
// ở đây nó sẽ trả về null
```

2.3.6. Underfined

Undefined là biến chưa được có giá trị.

Ví dụ 2.17.

```
 var x; // x chưa được gán giá trị nào
console.log(x);
// Hiện thị ra màn hình console giá trị của x,
// ở đây kết quả trả về là underfined
```

2.3.7. Object

2.3.7.1 Khai báo

Object là kiểu dữ liệu đối tượng, một đối tượng ở đây là chỉ một đối tượng thật trong cuộc sống. Ví dụ một chiếc xe tăng, một con mèo, một lá cờ Đảng … Đối tượng được chia làm 2 phần là: thuộc tính (properties) và các phương thức (methods) hay còn gọi là hành vi của đối tượng.

Ví dụ 2.18.

```
var meo = {
ten: 'meomeo',
maulong: 'den',
tuoi: 3,
fullInfo: function getFullInfo() {
    return 'Ten: ' + this.ten + ', Mau long: ' + this.maulong + ',
Tuoi: ' + this.tuoi;
  }
};
//khởi tạo đối tượng meo có tên là 'meomeo' màu lông là 'den'
// và có tuổi là 3
console.log(meo.ten);
```

Trong ví dụ 2.18 ta thấy đối tượng **meo** có 3 thuộc tính là **ten, maulong và tuoi** trong đó thuộc tính **ten** có giá trị (Property Value) là **meomeo**.

Ngoài ra object **meo** còn có một phương thức đó chính là **fullInfo** để trả về thông tin đầy đủ của đối tượng là một chuỗi gồm tên, màu lông và tuổi.

2.3.7.2. Thiết lập và truy xuất dữ liệu (Get và Set Data)

Để thiết lập giá trị (Set) cho thuộc tính của đối tượng ta có hai cách như sau:

```
meo.ten = 'New Name';
meo['ten'] = 'New Name';
```

Việc lấy dữ liệu cũng đơn giản

```
var _ten = meo.ten;
```

hoặc

```
var _ten = meo['ten];
```

2.3.7.3 Thực thi phương thức

Để thực thi phương thức của object ta làm như sau.

```
var _fullInfo = meo.fullInfo();
```

2.4. Toán tử

2.4.1. Giới thiệu

Chúng ta xét biểu thức đơn giản sau 4 + 5 bằng 9. Tại đây 4 và 5 là các operands - toán hạng và '+' được gọi là operator - toán tử. JavaScript hỗ trợ các kiểu toán tử sau: toán tử số học, toán tử so sánh, toán tử logic (hoặc quan hệ), toán tử gán, toán tử điều kiện.

2.4.2. Toán tử gán

Toán tử gán được dùng để gán giá trị ở bên phải toán tử vào biến ở bên trái toán tử. Có các toán tử gán sau:

Toán tử	Ví dụ	Ý nghĩa
=	x = y	gán y vào x
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

2.4.3. Các toán tử số học

Các toán tử số học này thực hiện trên các số - dữ liệu dạng số (cụ thể hoặc biến).

Toán tử	Mô tả	Ví dụ
+	phép cộng	25 + 5 = 30
-	phép trừ	10 - 5 = 5
*	phép nhân	2*3 = 6
/	phép chia	20 / 2 = 10
%	lấy phần dư của phép chia	56 / 3 = 2
++	Tăng thêm 1	var a = 10; a ++; //giá trị a là 11

--	giảm đi 1	var a = 10; a --; //giá trị a là 9

Chú ý về phép toán tăng thêm 1++ và giảm 1--: Khi viết biểu thức thì toán tử có thể ở bên trước hoặc sau biến cần tăng giảm, kết quả trả về của biểu thức có sự khác nhau tùy cách viết

Viết sau biến như: a = var++, b = var-- thì giá trị trả về của biểu thức (giá trị gán vào a, b) là giá trị gốc của var, còn bản thân var vẫn được tăng, giảm

Ví dụ 2.19.

```
var a = 0; b = 10;
var a = b++;
//a sẽ là 10; b là 11
```

Nếu toán tử ++-- viết bên trái biến, thì giá trị biểu thức trả về chính là biến sau khi tăng giảm

Ví dụ 2.20.

```
var a = 0; b = 10;
var a = ++b;
//a sẽ là 11; b là 11
```

2.4.4. Toán tử so sánh

Toán tử so sánh sử dụng trong các biểu thức về logic để so sánh bằng nhau, khác nhau. Nó trả về giá trị true false

JavaScript có một số toán tử so sánh, ví dụ so sánh bằng ==

Ví dụ 2.21.

```
var num = 10;
// num == 8 will return false
```

Bảng toán tử so sánh

Toán tử	Diễn tả	Ví dụ
==	so sánh bằng	5 == 10 false
===	giống nhau (cùng giá trị và kiểu dữ liệu)	5 === 10 false
!=	khác giá trị	5 != 10 true
!==	Khác giá trị và kiểu	10 !== 10 false
>	lớn hơn	10 > 5 true

>=	lớn hơn hoặc bằng	10 >= 5 true
<	nhỏ hơn	10 < 5 false
<=	nhỏ hơn	10 <= 5 false

Khi sử dụng các toán tử này, hãy chắc chắn các số hạng có cùng kiểu; số so sánh với số; chuỗi so sánh với chuỗi ...

2.4.5. Toán tử logic

Bảng toán tử logic gồm các phép toán **và** (&&) - **hoặc** (||) - **phủ định** (!).

Toán tử	Diễn tả
&&	phép và trả về true nếu 2 số hạng là true: a && b
||	phép hoặc trả về true nếu 1 trong 2 số hạng là true : a || b
!	phủ định; trả về giá trị ngược với biểu thức !a

Ví dụ 2.22.

```
var a = (4 > 2) && (10 < 15);
//a nhận giá trị false: vì 4 > 2 là true, 10 < 15 là false;
//a = true && false;
```

2.4.6. Toán tử điều kiện

Cú pháp:

variable = (condition) ? value1: value2;

Nhận giá trị value1 nếu điều kiện là true, nhận value2 nếu điều kiện false

Ví dụ 2.23.

```
var isAdult = (age < 18) ? "Too young": "Old enough";
//isAdult là Too young nếu age nhỏ hơn 18
//isAdult là Old enough nếu age lớn hơn hoặc bằng 18
```

2.4.7. Toán tử với chuỗi

Toán tử với chuỗi sử dụng nhiều là nối hai chuỗi lại với nhau, sử dụng toán tử + để nối. Lưu ý toán tử này có thể nối số vào chuỗi.

Ví dụ 2.24.

```
var mystring1 = "Học viết mã ";
var mystring2 = "JavaScript.";
document.write(mystring1 + mystring2);
```

```
//sẽ viết ra: Học viết mã JavaScript.
```

Xuất biến ra chuỗi với kỹ thuật Template Literal, ta có thể đưa vào chuỗi nằm giữa dấu `` (không phải " hay ""), trong chuỗi đó có thể chèn biểu thức với ký hiệu ${biểu-thức}

Ví dụ 2.25.

```
let tb = `Hai nhân hai là ${2*2}`;
let name = "XuanThuLab";
let msg  = `Xin chao ${name}`;
console.log(tb);
console.log(msg);
```

2.4.8. Toán tử typeof

Toán tử typeof trả về kiểu dữ liệu cần kiểm tra của một biến, một giá trị.

Ví dụ 2.26.

```
var a = 1;
var b = "Hi";
var c = true;
var d = null;
var e;

console.log(typeof(a)); // trả về number
console.log(typeof(b)); // trả về string
console.log(typeof(c)); // trả về boolean
console.log(typeof(d)); // trả về object
console.log(typeof(e)); // trả về undefined
console.log(typeof(f)); // trả về undefined
```

2.5. Biểu thức

Tập hợp các literal, biến và các toán tử nhằm đánh giá một giá trị nào đó được gọi là một biểu thức (expression). Về cơ bản có ba kiểu biểu thức trong JavaScript:

- Số học: Nhằm để lượng giá giá trị số. Ví dụ (3+4)+(84.5/3) được đánh giá bằng 197.1666666667.
- Chuỗi: Nhằm để đánh giá chuỗi. Ví dụ "The dog barked" + barktone + "!" là The dog barked ferociously!.
- Logic: Nhằm đánh giá giá trị logic. Ví dụ temp>32 có thể nhận giá trị sai. JavaScript cũng hỗ trợ biểu thức điều kiện, cú pháp như sau:

```
(condition) ? valTrue : valFalse
```

Nếu điều kiện condition được đánh giá là đúng, biểu thức nhận giá trị valTrue, ngược lại nhận giá trị valFalse.

Ví dụ 2.27.

```
state = (temp>32) ? "liquid" : "solid"
```

Trong ví dụ 2.27 biến state được gán giá trị "liquid" nếu giá trị của biến temp lớn hơn 32; trong trường hợp ngược lại nó nhận giá trị "solid".

2.6. Các cấu trúc điều khiển

2.6.1. Các cấu trúc chọn lựa

2.6.1.1. Cấu trúc chọn lựa if

Câu lệnh if else dùng để kiểm tra một mệnh đề nào đó có đúng hay không, nếu đúng thì thực thi những câu lệnh bên trong và ngược lại nếu sai thì nó sẽ bỏ qua những câu lệnh đó.

Cú pháp:

```
if (condition){
    // Code cho lệnh if
}
[else{
    // Code cho lệnh else
}]
```

Trong đó:

- Nếu **condition** có giá trị **true** thì **Code cho lệnh if** được thực hiện, ngược lại (**condition** có giá trị **false**) thì **Code cho lệnh else** được thực hiện.
- Câu lệnh trong ngoặc vuông có thể có hoặc không.

Ví dụ 2.28.

```
var a = 12;
var b = 12;

if (a == b)
    alert('a và b bằng nhau');
```

Ví dụ 2.29.

```
var a = 12;
var b = 10;

if (a == b){
    alert('a và b bằng nhau');
}
else{
    alert('a và b khác nhau');
}
```

Ví dụ 2.30.

```
var a = 12;

if (a > 12){
    alert('a > 12');
}
else if (a < 12){
    alert('a < 12');
}
else{
    alert('a = 12');
}
```

Ví dụ 2.31.

```
var a = 13;

// Nếu a > 12
if (a > 12)
{
    // Khai báo biến b
    var b = 20;

    // Nếu a bằng b
    if (a == b)
    {
        alert(' a = b ');
    }
    else { // ngược lại a khác b
        alert(' a != b ');
    }
}
```

2.6.1.2. Cấu trúc chọn lựa switch

Lệnh switch case có công dụng giống như lệnh if else đó là đều dùng để rẽ nhánh chương trình, ứng với mỗi nhánh sẽ có một điều kiện cụ thể và điều kiện đó phải sử dụng toán tử so sánh bằng, còn đối với lệnh if else thì bạn có thể truyền vào một mệnh đề bất kì và sử dụng nhiều toán tử khác nhau.

Cú pháp:

```
switch (variable)
{
    case value_1 : {
        // do some thing
        break;
    }
    case value_2 : {
        // do some thing
```

```
        break;
    }
    default : {
        // do something
    }
}
```

Nếu như trong tất cả các case không có case nào phù hợp thì nó sẽ chạy lệnh ở default, ngược lại nếu có case nào phù hợp thì chương trình sẽ chạy trong case đó, đồng thời lệnh break sẽ giúp chương trình thoát khỏi lệnh switch, còn nếu bạn không thêm lệnh break thì chương trình sẽ tiếp tục kiểm tra và chạy tiếp ở case tiếp theo.

Quy trình chạy như sau:

- Nếu tham số variable có giá trị là value_1 thì những đoạn code nằm bên trong case 1 sẽ được thực hiện, ngược lại nó sẽ nhảy xuống case tiếp theo.
- Lúc này nếu variable có giá trị là value_2 thì những đoạn code trong case 2 sẽ được thực hiện, ngược lại nó kiểm tra tiếp xem còn case nào không.
- Nhận thấy không còn case nào nữa nên nó sẽ kiểm tra có lệnh default không? Vì có lệnh default nên nó sẽ chạy đoạn code trong lệnh default đó rồi thoát khỏi switch case.

Ví dụ 2.32. Chương trình cho người dùng nhập vào một số, kiểm tra số đó là số chẵn hay số lẻ.

Bài toán này ta kết hợp lệnh prompt() để lấy thông tin từ người dùng, đồng thời kết hợp lệnh switch case để hiển thị kết quả. Có một lưu ý là nên sử dụng hàm parseInt() để chuyển dữ liệu người dùng nhập sang number.

```
var number = parseInt(prompt("Nhập số cần kiểm tra"));
var mod = (number % 2);
switch (mod)
{
    case 0 : {
        document.write(number + " là số chẵn");
        break;
    }
    case 1: {
        document.write(number + " là số lẽ");
        break;
    }
    default: {
        document.write("Ký tự bạn nhập không phải số");
    }
}
```

Ví dụ 2.32 ta có thể sử dụng lệnh if else để thực hiện.

```
var number = parseInt(prompt("Nhập số cần kiểm tra"));
var mod = (number % 2);
if (mod == 0){
    document.write(number + " là số chẵn");
}
else if (mod == 1){
    document.write(number + " là số lẽ");
}
else{
    document.write("Ký tự bạn nhập không phải số");
}
```

Ví dụ 2.33. Chương trình cho người dùng nhập một màu, kiểm tra màu đó có phải màu đỏ (red) hay màu vàng (yellow) hay không? Nếu không phải thì thông báo cho người dùng biết nhập sai màu.

Chúng ta sẽ giải bài này bằng nhiều cách khác nhau và mỗi cách bạn sẽ học được một kinh nghiệm xử lý lệnh switch case.

Trường hợp không có default

Trường hợp này nếu bạn nhập một màu khác với màu đỏ (red) và vàng (yellow) thì sẽ không có thông báo gì.

```
var color = prompt("Nhập màu cần kiểm tra");
switch (color){
    case 'red' :
        document.write("Bạn nhập màu đỏ, đúng rồi đó");
        break;
    case 'yellow' :
        document.write("Bạn nhập màu vàng, đúng rồi đó");
        break;
}
```

Trường hợp không có break

Trường hợp này nếu nhập vào màu đỏ (red) thì chương trình sẽ in ra cả lệnh ở case màu vàng (yellow) phía dưới, lý do là trong case màu đỏ ta không sử dụng lệnh break để thoát khỏi lệnh switch nên nó sẽ chạy thẳng xuống case phía dưới luôn mà không cần kiểm tra điều kiện.

```
var color = prompt("Nhập màu cần kiểm tra");
switch (color){
    case 'red' :
        document.write("Bạn nhập màu đỏ, đúng rồi đó");
    case 'yellow' :
        document.write("Bạn nhập màu vàng, đúng rồi đó");
        break;
    default :
```

```
        document.write("Màu bạn nhập không có trong hệ thống");
}
```

Giả sử bạn thêm một case nữa cho màu xanh (blue), lúc nếu bạn nhập vào màu đỏ (red) thì kết quả chỉ in thêm lệnh màu vàng (yellow) thôi. Từ đó suy ra rằng nếu không có break thì nó chỉ chạy luôn case đầu tiên phía dưới nó.

```
var color = prompt("Nhập màu cần kiểm tra");
switch (color){
    case 'red' :
        document.write("Bạn nhập màu đỏ, đúng rồi đó");
    case 'yellow' :
        document.write("Bạn nhập màu vàng, đúng rồi đó");
        break;
    case 'blue' :
        document.write("Bạn nhập màu xanh, đúng rồi đó");
        break;
    default :
        docment.write("Màu bạn nhập không có trong hệ thống");
}
```

Trường hợp gom nhóm case

Nếu để ý kỹ hơn thì thì thấy rằng nếu người dùng nhập vào màu đỏ (red), vàng (yellow) và xanh (blue) thì đều có thông báo nhập đúng. Vậy tai sao mình không gom ba trường hợp đó thành một thôi.

```
var color = prompt("Nhập màu cần kiểm tra");
switch (color){
    case 'red' :
    case 'yellow' :
    case 'blue' :
        document.write("Bạn nhập màu " + color + ", đúng rồi đó");
        break;
    default :
        document.write("Màu bạn nhập không có trong hệ thống");
}
```

2.6.2. Các cấu trúc lặp

2.6.2.1. Giới thiệu

Vòng lặp dùng để thực thi một hành động lặp đi lặp lại. Bên cạnh đó, các bài toán từ đơn giản đến phức tạp, không bài nào là không bắt gặp vòng lặp. Nó xuất hiện ở mọi bài toán, mọi vấn đề.

Cũng như những ngôn ngữ lập trình khác, Javascript có rất nhiều cấu trúc lặp khác nhau while, for, do...while,... Tuy nhiên, trong Javascript còn cung cấp thêm cho chúng ta 2 cấu trúc lặp rất đặc biệt, có cú pháp ngắn gọn và dễ sử dụng. Đó chính là for...of và for...in.

2.6.2.2. Vòng lặp for

Vòng lặp for thực hiện liên tục cho tới khi điều kiện ban đầu trả về false. Vòng for trong JavaScript tương tự như vòng for trong Java hoặc C.

Cú pháp:

```
for ([biểu_thức_khởi_tạo]; [điều_kiện]; [biểu_thức_tăng_tiến])
  lệnh
```

Vòng lặp for thực thi như sau:

- Biểu thức khởi tạo (biểu_thức_khởi_tạo), nếu có, được thực thi. Biểu thức này thường khởi tạo một hoặc nhiều biến đếm cho vòng lặp, nhưng cú pháp của nó vẫn có thể tạo ra biểu thức có độ phức tạp. Biểu thức này còn có thể khai báo biến.

- Biểu thức điều kiện (điều_kiện) được tính toán. Nếu giá trị của điều_kiện trả về true, các lệnh trong vòng lặp được thực thi. Nếu giá trị của điều_kiện trả về false, vòng lặp for bị dừng lại. Nếu biểu thức điều_kiện bị khuyết (bỏ qua không đặt), điều kiện sẽ được gán giả định là true.

- lệnh sẽ được thực thi. Để thực thi nhiều lệnh, dùng khối lệnh ({ ... }) để gom nhóm các lệnh này.

- Nếu không có lỗi nào xảy ra sau khi thực thi lệnh, biểu thức cập nhật (biểu_thức_tăng_tiến) được thực thi.

- Quay lại bước 2.

Ví dụ 2.34.

```
for(let i=0; i<10; i++){
      console.log(i);
}
```

Trong đó:

- let i = 0 : khởi tạo biến cho vòng lặp

- i < 10 : điều kiện để vòng lặp thực hiện

- i++ : tăng giá trị biến chạy lên 1 mỗi khi thực hiện xong hành động

Ta có thể bỏ trống giá trị ban đầu trong cú pháp của vòng lặp for nếu trước đó đã gán giá trị của biến chạy

```
var i=0;
```

```
for(; i<10; i++){
      console.log(i);
}
```

Ta cũng có thể bỏ trống giá trị thứ hai trong cú pháp của vòng lặp for. Lúc này, nếu giá trị thứ hai trả về giá trị true thì vòng lặp tiếp tục thực thi, nếu là false thì vòng lặp sẽ dừng lại.

```
for(let i=0;; i++){
      console.log(i);
      if(i===10)break;
}
```

Nếu bỏ trống giá trị thứ hai thì bắt buộc trong vòng lặp phải có lệnh break, nếu không thì vòng lặp sẽ chạy mãi không dừng lại.

Giá trị thứ ba trong cú pháp vòng lặp for cũng không thực sự cần thiết nếu như bạn thay đổi giá trị của biến chạy trong vòng lặp.

```
for(let i=0;;){
      console.log(i);
      if(i===10)break;
      i++;
}
```

Tuy nhiên, việc bỏ trống các giá trị trong cú pháp của vòng lặp for là không nên. Dù ngắn gọn nhưng nếu dùng không đúng lúc sẽ khiến code của chúng ta khó đọc lại.

Ví dụ 2.35.

```
//Khai báo mảng các objects
var students = [
{
            name = ‘Man’,
            age = 19
},
{
            name = ‘Huy’,
            age = 19
},
{
            name = ‘Minh’,
            age = 21
}
]
//in ra giá trị name trong mỗi object
for(let i=0; i<students.length; i++){
            console.log(students[i].name);
}
```

Kết quả in ra:

```
Man
Huy
Minh
```

2.6.2.3. Vòng lặp for...of

Vòng lặp này được ra mắt ở phiên bản ES6. Tương tự như for, vòng lặp này được sử dụng để duyệt từng phần tử của đối tượng duyệt. Số lượng lặp bằng với số phần tử của đối tượng.

Cú pháp:

```
for(let student of students){
            //do something
}
```

Trong đó:

- let student: khai báo biến chạy
- students: một Array, String, Map, WeakMap, Set (không có Object)

Ví dụ 2.36.

```
//Khai báo mảng các objects
var students = [
{
            name = ‘Man’,
            age = 19
},
{
            name = ‘Huy’,
            age = 19
},
{
            name = ‘Minh’,
            age = 21
}
]
//in ra giá trị name trong mỗi object
for(let student of students){
            console.log(student.name);
}
```

Kết quả in ra:

```
Man
Huy
Minh
```

Và kết quả cũng giống như chúng ta sử dụng vòng lặp for nhưng có thể thấy cú pháp của for...of tường minh, dễ hiểu hơn là for.

Tuy nhiên for...of không duyệt theo kiểu giảm dần giống như trong for được.

2.6.2.4. Vòng lặp for...in

Vòng lặp này thường được sử dụng với mục đích là lặp trong một object chứ không phải trong array hay string giống như hai vòng lặp trên. Số lượng lặp tương ứng với số thuộc tính của object mà ta duyệt.

Cú pháp:

```
for(let key in obj){
            //do something
}
```

Trong đó:

- let key: khai báo biến chạy
- obj: đối tượng duyệt (thường là object)

Ví dụ 2.37.

```
//Khai báo mảng các objects
var students = [
{
            name = ‘Man’,
            age = 19
},
{
            name = ‘Huy’,
            age = 19
},
{
            name = ‘Minh’,
            age = 21
}
]
//in ra giá trị name trong mỗi object
for(let student of students){
      for(let key in student){
            console.log(student.name);
      }
}
```

Kết quả in ra:

```
name
age
```

```
name
age
name
age
```

Kết quả in ra là những thuộc tính của object

for...in cũng có thể dùng với array, lúc này các thuộc tính của mỗi phần từ trong array chính là index.

Ví dụ 2.38.

```
var animals = [‘cat’, ‘dog’, ‘mouse’];
for(animal in animals){
      console.log(animal);
}
```

Kết quả in ra:

```
0
1
2
```

2.6.2.5. Cấu trúc lặp while

Lệnh while thực thi các lệnh bên trong nó ngay khi điều kiện xác định trả về true.

Cú pháp:

```
while (điều_kiện)
  lệnh
```

Nếu điều_kiện trả về false, chương trình sẽ ngừng thực thi lệnh bên trong vòng lặp và truyền điều khiển xuống lệnh liền sau vòng lặp.

Chương trình kiểm tra điều kiện trước khi thực thi lệnh bên trong. Nếu điều kiện trả về true, lệnh sẽ được thực thi và kiểm tra lại điều kiện. Nếu điều kiện trả về false, ngừng thực thi và truyền điều khiển xuống lệnh liền sau vòng lặp while.

Để thực thi nhiều lệnh, dùng khối lệnh ({ ... }) để gom nhóm các câu lệnh.

Ví dụ 2.39. Vòng while lặp lại cho tới khi n nhỏ hơn 3

```
var n = 0;
var x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

Ứng với mỗi lần lặp, tăng n thêm một và cộng giá trị đó vào x. Bởi vậy, x và n có giá trị như sau:

- Sau lần lặp thứ nhất: n = 1 và x = 1
- Sau lần lặp thứ hai: n = 2 và x = 3
- Sau lần lặp thứ ba: n = 3 và x = 6

Sau khi hoàn thành lần lặp thứ ba, điều kiện n < 3 không còn trả về true nữa nên vòng lặp bị kết thúc.

Ví dụ 2.40. Hãy tránh lặp vô hạn. Hãy đảm bảo rằng điều kiện của vòng lặp sẽ dần trả về false; bằng không, vòng lặp sẽ không bao giờ kết thúc. Lệnh trong vòng lặp while dưới đây được thực thi mãi mãi vì điều kiện không bao giờ trả về false:

```
while (true) {
  console.log('Hello, world!');
}
```

2.6.2.6. Cấu trúc lặp do…while

Lệnh do...while thực hiện liên tục cho tới khi điều kiện xác định trả về false.

Cú pháp:

```
do{
  lệnh
}while (điều_kiện);
```

lệnh luôn được thực thi một lần trước khi kiểm tra điều kiện (và rồi cứ thế cho tới khi điều kiện trả về false). Để thực thi nhiều lệnh, dùng khối lệnh ({ ... }) để gom nhóm các câu lệnh. Nếu điều_kiện trả về true, lệnh lại được thực thi một lần nữa. Sau mỗi lần thực thi, chương trình kiểm tra lại điều kiện. Khi điều kiện trả về false, chương trình dừng thực thi vòng lặp và truyền điều khiển xuống lệnh liền sau vòng do...while.

Ví dụ 2.41. lặp ít nhất một lần và tái duyệt cho tới khi i không nhỏ hơn 5.

```
var i = 0;
do {
  i += 1;
  console.log(i);
} while (i < 5);
```

2.6.2.7. Lệnh break

Dùng lệnh break để kết thúc vòng lặp, kết thúc switch, hoặc liên hợp với một lệnh được gán nhãn.

- Khi dùng break không kèm với nhãn, chương trình kết thúc ngay tức thì vòng while, do-while, for, hoặc switch trong cùng bọc lệnh break và truyền điều khiển xuống lệnh liền sau.

- Khi dùng break kèm với nhãn, nó sẽ chấm dứt việc thực thi lệnh được gắn nhãn đó.

Cú pháp lệnh break:

```
break;
break [nhãn];
```

Kiểu cú pháp đầu tiên kết thúc vòng lặp trong cục bao bọc hoặc switch; kiểu thứ hai kết thúc lệnh gắn với nhãn.

Ví dụ 2.42. Lệnh lặp sẽ lặp qua từng phần tử của mảng (thông qua chỉ mục của từng phần tử) cho tới khi gặp phần tử có giá trị bằng theValue:

```
for (var i = 0; i < a.length; i++) {
  if (a[i] == theValue) {
    break;
  }
}
```

Ví dụ 2.43. Áp dụng break cho nhãn

```
var x = 0;
var z = 0;
labelCancelLoops: while (true) {
  console.log('Outer loops: ' + x);
  x += 1;
  z = 1;
  while (true) {
    console.log('Inner loops: ' + z);
    z += 1;
    if (z === 10 && x === 10) {
      break labelCancelLoops;
    } else if (z === 10) {
      break;
    }
  }
}
```

2.6.2.8. Lệnh continue

Lệnh continue có thể dùng để tái duyệt các vòng while, do-while, for, hoặc label.

- Khi dùng continue không kèm theo label, chương trình ngừng thực thi lần lặp hiện tại của vòng while, do-while, hoặc for gần nhất bọc lệnh continue và tiếp tục thực thi lần lặp tiếp theo. Trái với lệnh break, continue không dừng vòng lặp

hoàn toàn. Trong vòng lặp while, chương trình nhảy về đoạn xét điều kiện. Trong vọng lặp for, chương trình nhảy tới biểu_thức_tăng_tiến.

- Khi dùng continue có kèm theo label, lệnh continue sẽ áp dụng cho vòng lặp được gán nhãn đó.

Cú pháp lệnh continue:

```
continue [label];
```

Ví dụ 2.44. Vòng while với lệnh continue thực thi khi giá trị của i bằng 3. Thế nên, n sẽ có giá trị là 1, 3, 7 và 12.

```
var i = 0;
var n = 0;
while (i < 5) {
  i++;
  if (i == 3) {
    continue;
  }
  n += i;
  console.log(n);
}
//1,3,7,12


var i = 0;
var n = 0;
while (i < 5) {
  i++;
  if (i == 3) {
     // continue;
  }
  n += i;
  console.log(n);
}
// 1,3,6,10,15
```

Ví dụ 2.45. Lệnh nhãn checkiandj chứa nhãn checkj. Nếu chương trình chạy tới continue, chương trình sẽ ngừng thực hiện lần lặp hiện tại của checkj và bắt đầu thực hiện lần lặp kế tiếp. Mỗi khi chạy tới continue, checkj tái duyệt cho tới khi biểu thức điều kiện của nó trả về false. Khi false được trả về, phần còn lại của lệnh checkiandj được hoàn thành, và checkiandj tái duyệt cho tới khi điều kiện của nó trả về false. Khi false được trả về, chương trình tiếp tục thực thi lệnh liền sau checkiandj.

Nếu continue gắn với nhãn checkiandj, chương trình sẽ tiếp tục ở đầu lệnh checkiandj .

```
var i = 0;
```

```
var j = 10;
checkiandj:
  while (i < 4) {
    console.log(i);
    i += 1;
    checkj:
      while (j > 4) {
        console.log(j);
        j -= 1;
        if ((j % 2) == 0) {
          continue checkj;
        }
        console.log(j + ' is odd.');
      }
      console.log('i = ' + i);
      console.log('j = ' + j);
  }
```

2.7. Các câu lệnh xử lý ngoại lệ

2.7.1. Giới thiệu

Giống bất kỳ ngôn ngữ nào, Javascript cũng sẽ gặp các tình huống lỗi như: undefined, null, JSON.parse() fail...

Điều này nghĩa là chúng ta phải xử lý những lỗi ngoại lệ này một cách nuột nà nếu chúng ta muốn có một trải nghiệm người dùng tốt cho ứng dụng của mình.

Giải pháp cho vấn đề này là dùng try...catch blocks, việc này cũng giúp ứng dụng của chúng ta ko bị crash khi xảy ra lỗi.

2.7.2. Sử dụng lệnh try...catch

Cú pháp try…catch:

```
try{
  // các đoạn xử lý mà có thể phát sinh ngoại lệ
  // phải có ít nhất 1 dòng code trong khối này
}
catch (error){
  // xử lý lỗi ngoại lệ ở đây
  // khối này có thể không cần nếu khối finally đươck khai báo
}
finally {
  // phần này có thể có hoặc không
  // khối này sẽ được chạy sau khi xử lý xong khối try hoặc catch
}
```

Ví dụ 2.46.

```
try {
  undefined.prop
```

```
} catch (error) {
  console.log(error);
}
```

Trong ví dụ 2.46 chúng ta khi lấy thuộc tính từ undefined, nên sẽ sinh ra một lỗi ngoại lệ. Trong catch block, chúng ta bắt sự kiện error và log nó ra, kết quả như sau:

```
TypeError: Cannot read property ‘prop’ of undefined
```

Khi ứng dụng chạy dòng lệnh undefined.prop, chúng ta có được error message log ra thay vì crashing ứng dụng.

Như trong phần comments trên, try block phải có ít nhất một xử lý nào đó, sau đó là catch block hoặc finally block được khai báo. Điều này có nghĩa là chúng ta có thể có các trường hợp sau:

```
try {
  ...
}
catch {
  ...
}
try {
  ...
}
finally{
  ...
}
try {
  ...
}
catch {
  ...
}
finally {
  ...
}
```

Khối catch chứa các đoạn code mà chỉ ra app sẽ xử lý cái gì khi mà phát sinh lỗi ngoại lệ trong try block. Nếu try block phát sinh lỗi thì catch block sẽ được chạy, nếu try block không phát sinh lỗi thì catch block sẽ được bỏ qua.

Khối finally sẽ được chạy sau khi tất cả code trong phần try block hoặc catch block chạy xong. Nó luôn chạy dù có phát sinh lỗi ngoại lệ hay không.

Khối try có thể được lồng bên trong nhau. Nếu khối try..catch bên trong không xử lý catch lỗi thì khối try...catch bên ngoài phải làm điều đó.

Ví dụ 2.47.

```
try {
```

```
    try {
      undefined.prop
    } finally {
      console.log('Inner finally block runs');
    }
  } catch (error) {
    console.log('Outer catch block caught:', error);
  }
```

Khi chạy đoạn code trên, chúng ta sẽ thấy log ra:

```
Inner finally block runs
Outer catch block caught: TypeError: Cannot read property ‘prop’ of undefined’
```

Có thể thấy là khi khối try...catch bên trong không catch lỗi nên khối try...catch bên ngoài có khai báo catch nên đã xử lý lỗi, và khối finally bên trong chạy trước khối catch bên ngoài. Try...catch...finally chạy tuần tự từ trên xuống dưới theo thứ tự.

Các khối catch ở trên tất cả đều theo kiểu unconditional (vô điều kiện), tức là chúng có thể bắt tất cả các ngoại lệ xảy ra. Object error giữ dữ liệu về lỗi ngoại lệ. Nó chỉ giữ dữ liệu bên trong khối catch(nó theo kiểu variable scope), nếu muốn sử dụng error object bên ngoài khối catch thì phải gán nó cho 1 biến bên ngoài. Sau khi catch block chạy xong, error object sẽ được giải phóng khỏi bộ nhớ.

Khối finally chứa các đoạn code mà được chạy sau khi code trong try block hoặc catch block chạy, và nó được thực thi bất kể có lỗi ngoại lệ hay không có. Do đó khối finally làm cho trải nghiệm người dùng ít bị ảnh hưởng nhất khi mà ứng dụng của chúng ta xảy ra lỗi (hoặc có thể do các yếu tố khách quan như mất kết nối mạng...). Ví dụ, chúng ta có thể đặt các code clean up ví dụ như close file khi đọc file dù có hay không xảy ra lỗi khi đọc. Khi chạy tới dòng mà phát sinh lỗi ngoại lệ thì toàn bộ code sau dòng đó trong try block sẽ không được thực thi nữa, nên nếu chúng ta xử lý close file ở cuối của try block thì có thể nó sẽ không được thực thi (nếu xảy ra lỗi). Chúng ta có thể chỉ đặt code sẽ chạy mà không cần quan tâm liệu có phát sinh lỗi hay không như cleanup code, release memory... trong finally block, khi đó chúng ta sẽ không phải dupplicate code khi mà phải xử lý những việc đó trong try và catch block.

Ví dụ 2.48.

```
openFile();
try {
  // tie up a resource
  writeFile(data);
}
finally {
  closeFile();
  // always close the resource
}
```

Ta có thể lồng nhiều block try...catch với nhau:

Ví dụ 2.49.

```
try {
  try {
    throw new Error('error');
  }
  finally {
    console.log('first finally runs');
  }
  try {
    throw new Error('error2');
  }
  finally {
    console.log('second finally runs');
  }
}
catch (ex) {
  console.error('exception caught', ex.message);
}
```

Trong ví dụ 2.49 chúng ta sẽ không thấy log ra:

```
second finally runs
```

bởi vì khối try...catch đầu tiên không catch lỗi ngoại lệ nên khi xảy ra lỗi thì lỗi đó được truyền ra và được catch bởi khối try...catch cha bên ngoài. Nếu chúng ta muốn khối try...catch thứ 2 được thực thi thì chúng ta phải thêm catch block vào khối try...catch thứ nhất, như sau:

```
try {
  try {
    throw new Error('error');
  }
  catch {
    console.log('first catch block runs');
  }
  finally {
    console.log('first finally runs');
  }
  try {
    throw new Error('error2');
  }
  finally {
    console.log('second finally runs');
  }
}
catch (ex) {
  console.error('exception caught', ex.message);
}
```

Ta còn có thể rethrow các lỗi ngoại lệ mà được bắt ở catch block.

Ví dụ 2.50.

```
try {
  try {
    throw new Error('error');
  } catch (error) {
    console.error('error', error.message);
    throw error;
  } finally {
    console.log('finally block is run');
  }
} catch (error) {
  console.error('outer catch block caught', error.message);
}
```

Như ví dụ 2.50, ta có 3 output là

```
error error, finally block is run outer catch block caught error
```

bởi vì khối catch của block try...catch bên trong đã throw lỗi ngoại lệ một lần nữa sau khi log nó ra, do đó khối catch bên ngoài vẫn được thực thi do nhận được lỗi rethrow từ block catch bên trong.

Vì code chạy tuần tự, chúng ta có thể sử dụng biểu thức return ở cuối try block. Ví dụ, nếu chúng ta muốn parse một chuỗi JSON thành một object, chúng ta muốn return một object rỗng nếu xảy ra lỗi nào đó.

```
const parseJSON = (str) => {
  try {
    return JSON.parse(str);
  }
  catch {
    return {};
  }
}
```

và chúng ta chạy thử hàm trên:

```
console.log(parseJSON(undefined));
console.log(parseJSON('{"a": 1}'))
```

ta sẽ được object rỗng ở dòng thứ nhất và {a: 1} ở dòng thứ 2.

2.7.3. Try...Catch trong xử lý bất đồng bộ

Ta thường sử dụng async - await cho việc xử lý bất đồng bộ trong Javascript, và chúng ta còn có thể sử dụng try...catch block để catch rejected promises và xử lý rejected promises một cách uyển chuyển.

Ví dụ 2.51.

```
(async () => {
  try {
    await new Promise((resolve, reject) => {
      reject('something wrong!')
    })
  } catch (error) {
    console.log(error);
  }
})();
```

Trong đoạn code trên, vì chúng ta reject promise trong try block, nên catch block bắt được lỗi trong reject promise và log ra:

```
something wrong!
```

Hàm trên trông có vẻ giống với try...catch block bình thường, nhưng thực sự thì điều đó không đúng, vì đây là một async funcion.

Một async function chỉ trả về các promises, nên chúng ta không thể return bất cứ thứ gì thay vì promises trong try...catch block. Khối catch trong async function chỉ là một cách viết tắt cho catch function, hàm mà được chuỗi cùng với then function. Đoạn code trên tương đương với cách viết sau trong thế giới promises:

```
(() => {
  new Promise((resolve, reject) => {
      reject('error')
    })
    .catch(error => console.log(error))
})()
```

Tương tự thì finally block của try...catch trong async cũng tương đương với cách viết finally function trong promise. Ta có 2 đoạn code sau là tương đương:

```
(async () => {
  try {
    await new Promise((resolve, reject) => {
      reject('error')
    })
  } catch (error) {
    console.log(error);
  } finally {
    console.log('finally is run');
  }
})();
(() => {
  new Promise((resolve, reject) => {
      reject('error')
    })
    .catch(error => console.log(error))
```

```
    .finally(() => console.log('finally is run'))
})()
```

Quy tắc try...catch lồng nhau ở trên cũng áp dụng được cho async function, nên chúng ta có thể viết code như sau:

```
(async () => {
  try {
    await new Promise((resolve, reject) => {
      reject('outer error')
    })
    try {
      await new Promise((resolve, reject) => {
        reject('inner error')
      })
    } catch (error) {
      console.log(error);
    } finally {
    }
  } catch (error) {
    console.log(error);
  } finally {
    console.log('finally is run');
  }
})();
```

Điều này giúp ta dễ dàng lồng các promises và xử lý các lỗi tùy theo scope của chúng. Cách này giúp code sạch sẽ và dễ đọc hơn so với việc chain các then, catch và finally function của promise.

Tóm lại, để xử lý lỗi ngoại lệ trong JavaScript, chúng ta sử dụng try...catch...finally block để bắt lỗi. Cách này có thể áp dụng cho cả code đồng bộ và bất đồng bộ. Chúng ta để các đoạn code có thể phát sinh lỗi trong try block, và viết các code xử lý lỗi ngoại lệ trong catch block. Trong finally block, chúng ta viết các đoạn code thực thi mà không quan tâm tới có xảy ra lỗi hay không. Các async function có thể sử dụng try...catch block nhưng chúng chỉ trả về các promises giống như các async funciton khác, nhưng try...catch block trong các function bình thường thì có thể trả về bất cứ thứ gì.

Chương 3. Mảng

3.1. Khái niệm

Mảng là một tập hợp các phần tử cố định có cùng một kiểu, được lưu trữ liên tiếp nhau trong các ô nhớ. Kiểu phần tử có thể là có các kiểu bất kỳ: ký tự, số, chuỗi ký tự…; cũng có khi ta sử dụng kiểu mảng để làm kiểu phần tử cho một mảng.

Mảng có chức năng như biến và được sử dụng để chứa dữ liệu. Tuy nhiên thì mảng nó to hơn, một biến chỉ chứa duy nhất một giá trị còn mảng thì thích bao nhiêu có bấy nhiêu.

Chẳng hạn, một bài tập có 1000 cái tham số chẳng lẽ bạn lại ngồi nhập 1000 biến? Thay vì đó chúng ta sử dụng mảng để thực hiện. Ở javascript, trong một mảng các giá trị không cần cùng kiểu dữ liệu.

3.2. Khai báo mảng

Có 3 cách khai báo mảng:

- Đưa giá trị vào khi khai báo.
- Khai báo rỗng rồi đưa giá trị vào.
- Khai báo theo hướng đối tượng.

3.2.1. Đưa giá trị vào khi khai báo

Cú pháp:

```
var tên_mảng = [giá trị 1, giá trị 2, giá trị 3, ... ];
```

Ví dụ 3.1. Tạo mảng

```
var Ar = [1,2,3,4,'teemo','',];
console.log(Ar[0]);// 1
console.log(Ar[4]);// 'teemo'
console.log(Ar[5]);// null
console.log(Ar[6]);// undefined
```

Trong ví dụ 3.1, ta tạo mảng và truy xuất giá trị của các phần tử của mảng Array. Ar[0] chính là phần tử thứ nhất của mảng có giá trị = 1,...

2.3.2. Khai báo rỗng rồi đưa giá trị vào

Cú pháp:

```
var tên_mảng = [];
tên_mảng[0] = giá trị 1;
tên_mảng[1] = giá trị 2;
..
```

```
tên_mảng[n] = giá trị n;
```

Ví dụ 3.2.

```
var Ar = [];
Ar[0] = 1;
Ar[2] = 'teemo';
Ar[3] = '';
Ar[10] = ;
```

Ta có thể dùng hàm console.log để truy xuất các giá trị đã khởi tạo của mảng Ar.

2.3.3. Khai báo theo hướng đối tượng:

Cú pháp:

```
Ar = new Array();
```

Ví dụ 3.3.

```
Ar = new Array(1,2,3,4);//Nhập giá trị trực tiếp
Br = new Array(); //Khai báo rỗng rồi nhập giá trị
Br[0] = 1;
Br[1] = 'Teemo';
```

Tương tự, ta có thể dùng hàm console.log() để truy xuất giá trị của mảng Ar và Br.

3.3. Truy suất các phần tử trong mảng

Sau khi đã tạo được mảng, ta cần phải truy xuất thông tin của mảng. Để có thể lấy ra giá trị của một thành phần trong mảng thì chúng ta sử dụng cú pháp như bên dưới.

Cú pháp:

```
arr[index];
```

Trong đó: arr là tên biến mảng, index là vị trí của mảng(bắt đầu từ 0).

Ví dụ 3.4.

```
var arr = new Array(1, 2, 4, 5, 9, 6);
alert(arr[1]);
//result: 2
```

3.4. Các phương thức

3.4.1. Phương thức Length

Đây là phương thức trong mảng, trả về số lượng phần tử đang có trong mảng.

Cú pháp:

```
arr.length;
```

Trong đó: arr là tên của biến mảng.

Ví dụ 3.5.

```
var arr = new Array(1, 2, 4, 5, 9, 6);
alert(arr.length);
```

3.4.2. Phương thức join()

Phương thức này có tác dụng gộp tất cả các phần tử có trong mảng thành một chuỗi.

Cú pháp:

```
arr.join(string);
```

Trong đó:

- arr là tên biến mảng mà chúng ta cần gộp.
- string là chuỗi phân cách giữa các phần tử sau khi gộp (để trống là ngăn cách bằng dấu ,).

Ví dụ 3.6.

```
//gộp mảng mặc định
document.write(arr.join() + '<br>');
//result: 1,2,4,5,9,6
//gộp mảng ngăn cách nhau bằng -
document.write(arr.join('-') + '<br>');
//result: 1-2-4-5-9-6
```

3.4.3. Phương thức valueOf()

Phương thức valueof sẽ trả về đối tượng mảng. Đây là phương thức mặc định của đối tượng mảng, nó sẽ tương đương với việc chỉ gọi tên mảng. Ví dụ: subject.valueOf() sẽ tương đương với subject.

Chính vì là phương thức mặc định của đối tượng mảng, nên chúng ta sẽ ít khi sử dụng phương thức valueof mà thường gọi thẳng luôn tên mảng.

Phương thức valueof sẽ không làm thay đổi mảng ban đầu.

Cú pháp:

```
array.valueOf()
```

Phương thức này không có tham số truyền vào.

Ví dụ 3.7. sử dụng phương thức valueof để gọi đối tượng mảng.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```

```
        </head>
        <body>
            <h1>Học lập trình miễn phí tại freetuts.net</h1>
            <p>Click vào button để in mảng.</p>

            <button onclick="myFunction()">Run</button>

            <p id="demo"></p>

            <script>
                var subject = ["php", "css", "html", "js"];

                function myFunction() {
                    document.getElementById("demo").innerHTML =
subject.valueOf();
                }
            </script>
        </body>
    </html>
```

Kết quả:

```
    php,css,html,js
```

3.4.4. Phương thức push()

Phương thức này có tác dụng thêm một hoặc nhiều phần tử vào cuối mảng.

Ví dụ 3.8.

```
    var arr = new Array(1, 2, 4, 5, 9, 6);
    document.write(arr.join() + '<br>');
    //thêm 7 vào mảng
    arr.push(7);
    document.write(arr.join() + '<br>');
    //thêm 3,8 vào mảng
    arr.push(3,8);
    document.write(arr.join() + '<br>');
```

Kết quả:

```
    1,2,4,5,9,6
    1,2,4,5,9,6,7
    1,2,4,5,9,6,7,3,8
```

3.4.5. Phương thức pop()

Phương thức này có tác dụng xóa phần tử cuối cùng trong mảng.

Ví dụ 3.9.

```
var arr = new Array(1, 2, 4, 5, 9, 6);
document.write(arr.join() + '<br>');
//xóa phần tử cuối
arr.pop();
document.write(arr.join() + '<br>');
```

Kết Quả:

```
1,2,4,5,9,6
1,2,4,5,9
```

3.4.6. Phương thức unshift()

Phương thức này có tác dụng thêm một hoặc nhiều phần tử vào đầu mảng.

Ví dụ 3.10.

```
var arr = new Array(1, 2, 4, 5, 9, 6);
document.write(arr.join() + '<br>');
//thêm 7 vào mảng
arr.unshift(7);
document.write(arr.join() + '<br>');
//thêm 3,8 vào mảng
arr.unshift(3, 8);
document.write(arr.join() + '<br>');
```

Kết Quả:

```
1,2,4,5,9,6
7,1,2,4,5,9,6
3,8,7,1,2,4,5,9,6
```

3.4.7. Phương thức shift()

Phương thức này có tác dụng xóa phần tử đầu tiên của mảng.

Ví dụ 3.11.

```
var arr = new Array(1, 2, 4, 5, 9, 6);
document.write(arr.join() + '<br>');
//xóa phần tử đầu
arr.shift();
document.write(arr.join() + '<br>');
```

Kết Quả:

```
1,2,4,5,9,6
2,4,5,9,6
```

3.4.8. Phương thức splice()

Phương thức này có tác dụng thêm phần tử vào mảng theo một vị trí xác định.

Cú pháp:

```
arr.splice(index, howmany, item1, ....., itemX)
```

Trong đó:

- arr là tên biến mảng các bạn muốn thêm vào.
- index là vị trí mà các bạn muốn thêm phần tử vào (bắt đầu từ 0).
- howmany là số phần tử sẽ xóa tính tự vị trí thêm mảng, để 0 nếu không muốn xóa phần tử nào.
- item1,...itemX là giá trị của các phần tử muốn thêm vào.

Ví dụ 3.12.

```
var arr = new Array(1, 2, 4, 5, 9, 6);
document.write(arr.join() + '<br>');
//thêm 2 phần tử vào vị trí thứ 1 và 0 xóa phần tử nào
arr.splice(1, 0, 3, 8);
document.write(arr.join() + '<br>');
```

Kết Quả:

```
1,2,4,5,9,6
1,3,8,2,4,5,9,6
```

3.4.9. Phương thức concat()

Phương thức này có tác dụng ghép 2 mảng lại với nhau.

Ví dụ 3.13.

```
var arr1 = new Array(1, 2, 4, 5, 9, 6);
var arr2 = new Array(3, 8, 7);
document.write(arr1.concat(arr2) + '<br>');
```

Kết Quả:

```
1,2,4,5,9,6,3,8,7
```

3.4.10. Phương thức slice()

Phương thức này có tác dụng lấy ra một hoặc một số phần tử trong mảng.

Cú pháp:

```
arr.slice(begin,end);
```

Trong đó:

- arr là tên biến mảng các bạn muốn thêm vào.

- begin là vị trí bắt đầu.
- end là vị trí kết thúc, nếu muốn cắt từ vị trí đầu đến hết thì bỏ trống giá trị này.

Ví dụ 3.14.

```
var arr = new Array(1, 2, 4, 5, 9, 6);
document.write(arr.join() + '<br>');
//cắt mảng từ vị trí thứ 3 đến hết
document.write(arr.slice(3) + '<br>');
```

Kết Quả:

```
1,2,4,5,9,6
5,9,6
```

3.4.11. Phương thức sort()

Phương thức này có tác dụng sắp xếp lại mảng theo chiều tăng dần, nếu là số thì từ bé đến lớn, chữ thì sắp xếp theo alpha(a-z).

Ví dụ 3.15.

```
var arr = new Array(1, 2, 4, 5, 9, 6);
document.write(arr.join() + '<br>');
//sắp xếp lại mảng
document.write(arr.sort() + '<br>');
```

Kết Quả:

```
1,2,4,5,9,6
1,2,4,5,6,9
```

3.4.12. Phương thức reverse()

Phương thức này có tác dụng đảo ngược vị trí các phần tử của mảng.

Ví dụ 3.16.

```
var arr = new Array(1, 2, 4, 5, 9, 6);
document.write(arr.join() + '<br>');
//cắt mảng từ vị trí thứ 3 đến hết
document.write(arr.reverse() + '<br>');
```

3.5. Enum

3.5.1. Các khái niệm căn bản của enum

Javascript chỉ có đúng một kiểu mà giá trị bị ràng buộc rất cụ thể: boolean, giá trị chỉ được phép là true hoặc false, không chấp nhận một giá trị nào khác. Enum là phiên bản mở rộng với công dụng tương tự như boolean

Ví dụ 3.17.

```
enum NoYes {
    No,
    Yes
}
```

2 giá trị No Yes được gọi là thành viên của hội enum NoYes. Dùng nó như một object, chúng ta có thể chấm đến giá trị đó

Ví dụ 3.18.

```
function toGerman(value: NoYes) {
  switch (value) {
    case NoYes.No:
      return 'Nein';
    case NoYes.Yes:
      return 'Ja';
  }
}
```

Tất cả thành phần của enum đều được cấp một thẻ thành viên gồm name và value. Ở trên chúng ta chỉ đang khai báo phần name cho enum, value lúc đó sẽ lấy mặc định là số từ 0 trở đi.

Ví dụ 3.19.

```
enum NoYes {
  No,
  Yes,
}
// nếu khai báo một cách tường minh hơn
enum NoYes {
  No = 0,
  Yes = 1,
}

assert.equal(NoYes.No, 0);
assert.equal(NoYes.Yes, 1);
```

Nếu khai báo như thế này

```
enum Enum {
  A,
  B,
  C = 4,
  D,
  E = 8,
  F,
```

```
}
```

Đồng nghĩa là các giá trị kế tiếp sẽ tự động tăng lên một, so với giá trị khai báo trước đó

```
assert.deepEqual(
  [Enum.A, Enum.B, Enum.C, Enum.D, Enum.E, Enum.F],
  [0, 1, 4, 5, 8, 9]
);
```

Về cách đặt tên (name) trong enum, cũng có nhiều lựa chọn

- Đặt theo kiểu JavaScript, viết hoa hết, Number,MAX_VALUE
- Đặt theo kiểu symbol, viết hoa đầu mỗi từ, chữ đầu viết thường, Symbol.asyncIterator
- Kiểu TypeScript, viết hoa đầu mỗi từ, chữ đầu viết hoa, Number.MaxValue

Tương tự như object, chúng ta có thể truy xuất đến một thành viên của enum bằng cách viết sau

Ví dụ 3.20.

```
enum HttpRequestField {
  'Accept',
  'Accept-Charset',
  'Accept-Datetime',
  'Accept-Encoding',
  'Accept-Language',
}
assert.equal(HttpRequestField['Accept-Charset'], 1);
```

Giá trị value của enum, không bắt buộc là một number, có thể là một string

Ví dụ 3.21.

```
enum NoYes {
  No = 'No',
  Yes = 'Yes',
}
assert.equal(NoYes.No, 'No');
assert.equal(NoYes.Yes, 'Yes');
```

Còn một cách khai báo, ít được sử dụng, các giá trị trong enum có thể là số cũng có thể là chữ

Ví dụ 3.22.

```
enum Enum {
  A,
```

```
  B,
  C = 'C',
  D = 'D',
  E = 8,
  F,
}
assert.deepEqual(
  [Enum.A, Enum.B, Enum.C, Enum.D, Enum.E, Enum.F],
  [0, 1, 'C', 'D', 8, 9]
);
```

Theo như kinh nghiệm từ các người đi trước, sử dụng enum thì nên dùng kiểu giá trị string

```
enum NoYes { No = 'NO', Yes = 'YES' }
```

Nếu có log ra chúng ta cũng biết được giá trị chính xác là gì, đỡ mất công ngồi đếm số thứ tự

```
console.log(NoYes.No);
console.log(NoYes.Yes);
```

Thêm nữa, ràng buộc được luôn kiểu giá trị

```
function func(noYes: NoYes) {}
func('abc');
func('Yes');
```

3.5.2. Các trường hợp sử dụng enum

3.5.2.1. Hằng số nhiều giá trị

Nếu chúng ta có nhiều hằng số, có quan hệ họ hàng gần với nhau

Ví dụ 3.23.

```
const fatal = Symbol('fatal');
const error = Symbol('error');
const warn = Symbol('warn');
const info = Symbol('info');
const debug = Symbol('debug');
const trace = Symbol('trace');
```

Có thể dùng enum

```
enum LogLevel {
  fatal = 'fatal',
  error = 'error',
  warn = 'warn',
  info = 'info',
  debug = 'debug',
```

```
  trace = 'trace',
}
```

Lợi ích mang lại: nhóm lại với nhau một thể, Javascript dễ đàng kiểm tra giúp chúng ta

3.5.2.2. Cho các kiểu tường minh

Chúng ta hay dùng boolean để làm cờ bật tắt hoặc đảo ngược giá trị

```
class List1 { isOrdered: boolean; }
```

Nếu dùng enum, nó sẽ rõ nghĩa hơn, nhiều lựa chọn hơn

```
enum ListKind { ordered, unordered }
class List2 {
  listKind: ListKind;

}
```

3.5.2.3. Cho hằng String

String là một hằng số đúng nghĩa, an toàn hơn vì không thể thay đổi được giá trị

Ví dụ 3.24. hàm bên dưới dùng const

```
const GLOBAL = 'g';
const NOT_GLOBAL = '';
type Globalness = typeof GLOBAL | typeof NOT_GLOBAL;

function createRegExp(source: string,
  globalness: Globalness = NOT_GLOBAL) {
    return new RegExp(source, 'u' + globalness);
  }

assert.deepEqual(
  createRegExp('abc', GLOBAL),
  /abc/ug);
```

dùng enum tiện hơn

```
enum Globalness {
  GLOBAL = 'g',
  NOT_GLOBAL = '',
}

function createRegExp(source: string, globalness =
Globalness.NOT_GLOBAL) {
  return new RegExp(source, 'u' + globalness);
}

assert.deepEqual(
  createRegExp('abc', Globalness.GLOBAL),
```

```
  /abc/ug);
```

3.5.3. Enum lúc chạy thì sẽ trở thành gì?

Sau khi Javascript đã compile, enum sẽ được được chuyển thành javascript object

Ví dụ 3.25.

```
enum NoYes {
  No,
  Yes,
}
var NoYes;
(function (NoYes) {
  NoYes[NoYes["No"] = 0] = "No";
  NoYes[NoYes["Yes"] = 1] = "Yes";
})(NoYes || (NoYes = {}));
```

Chương 4. Hàm và Lập trình hướng đối tượng

4.1. Hàm

4.1.1. Khái niệm

- Hàm là một tập hợp gồm nhiều câu lệnh, các câu lệnh này được sắp xếp theo một thứ tự xác định để xây dựng thành một chức năng cụ thể

- Mỗi hàm sẽ có một cái tên và hàm chỉ thực thi khi nó được gọi đến tên.

4.1.2. Xây dựng hàm

Để xây dựng hàm, ta sử dụng cú pháp như bên dưới.

Cú pháp:

```
function name_of_function(var1, var2, var3, ...)
{
    // Some code
}
```

Trong đó:

- function: là từ khóa của javascript nên bắt buộc phải như vậy
- name_of_function: là tên của function, thông thường chúng ta tạo những tên có ý nghĩa như find_max, find_min, ...
- var1, var2 var3, ... là các tham số truyền vào hàm. Ví dụ viết hàm kiểm tra số chẵn hay lẽ thì ta sẽ có một tham số đó là số cần kiểm tra.

Ví dụ 4.1. Viết hàm kiểm tra một số chẵn hay lẻ.

```
// Tạo hàm
function check_number(number)
{
    if (number % 2 == 0){
        alert(number + ' là số chẵn');
    }
    else {
        alert(number + 'Số lẻ');
    }
}

// Sử dụng hàm kiểm tra cho 5 số
check_number(1);
check_number(2);
check_number(3);
check_number(4);
check_number(5);
```

Ta đã tạo một hàm với tham số truyền vào có tên là number. Như vậy khi sử dụng nếu ta truyền số 1 vào thì lúc này biến number trong hàm sẽ có giá trị là 1, tương tự cho 2, 3,4,5.

Lưu ý rằng trong javascript không tồn tại khái niệm con trỏ và tham chiếu trong hàm.

4.1.3. Hàm có return và hàm không có return

Hàm có return là hàm có sử dụng từ khóa return để đặt ở cuối hàm với mục đích trả kết quả về để sử dụng tiếp ở những đoạn code bên ngoài. Ví dụ ta cần viết một hàm tính tổng của hai số a và b thì hàm này phải trả về giá trị là tổng của hai số a, b.

Ví dụ 4.2.

```
// Khai báo hàm
function tinh_tong(a, b)
{
    // trả về kết quả là a + b
    return a + b;
}

// Sử dụng
var so1 = 1;
var so2 = 2;

// truyền so1 và so2 vào hàm
var ketqua = tinh_tong(so1, so2);

alert(ketqua);
```

Hàm không có return là hàm không có sử dụng từ khóa return đặt trong hàm.

Ví dụ 4.3. Viết chương trình in ra tổng của hai số a và b.

```
// Khai báo hàm
function tinh_tong(a, b)
{
    document.write('Tổng là ' + (a + b));
}

// Sử dụng
var so1 = 1;
var so2 = 2;

// truyền so1 và so2 vào hàm
tinh_tong(so1, so2);
```

Như vậy tùy vào mục đích mà ta dùng có return hay không có return. Nhưng thông thường ta sử dụng return ở những trường hợp cần lấy kết quả đó để xử lý tiếp, như ở ví dụ trên đó là mình lấy kết quả để in thông báo.

4.1.4. Giá trị mặc định của tham số

Có một số trường hợp bạn muốn một tham số nào đó có thể được truyền hoặc không cần truyền vào đều được, lúc này chúng ta phải sử dụng nó như một tham số mặc định. Javascript không có cú pháp gán giá trị mặc định như PHP mà thay vào đó chúng ta sử dụng toán tử ||.

Ví dụ 4.4. Hàm hiển thị một thông báo.

```
function showMessage(message)
{
   // Nếu message không được truyền vào hoặc giá trị nó là rỗng
   // thì sẽ được thay thế bằng giá trị 'Không có tin nhắn'
   message = message || 'Không có tin nhắn <br/>';
   document.write(message);
}

// Cách 1: không truyền tham số
showMessage();

// Cách 2: Truyền tham số
showMessage('Chào mừng bạn đến với freetuts.net');
```

Toán tử || gồm hai vế, trong đó nếu vế TRÁI có giá trị rỗng (undefined, null, false, ...) kết quả sẽ trả về vế PHẢI, ngược lại nó sẽ trả về vế TRÁI.

4.1.5. Rest parameters

Rest parameter cho phép chúng ta dùng 1 mảng để đại diện cho số lượng vô hạn các đối số.

Trong ví dụ 4.5, hàm multiply sử dụng rest parameters để thu thập các đối số kể từ đối số hứ hai trở về đến hết. Hàm này sau đó sẽ nhân những đối số này với đối số đầu tiên.

Ví dụ 4.5.

```
function multiply(multiplier, ...theArgs) {
  return theArgs.map(x => multiplier * x);
}

var arr = multiply(2, 1, 2, 3);
console.log(arr); // [2, 4, 6]
```

4.1.6. Arrow Function

Trong ES6, arrow function là một cú pháp mới dùng để viết các hàm trong JavaScript. Nó giúp tiết kiệm thời gian phát triển và đơn giản hóa phạm vi function (function scope).

Arrow function - còn được gọi là "fat arrow", là cú pháp được mượn từ CoffeeScript (một ngôn ngữ chuyển tiếp), cú pháp này là cách ngắn gọn hơn dùng để viết function. Ở đây

sử dụng kí tự =>, trông giống như một mũi tên "béo". Arrow function là một hàm vô danh và nó thay đổi cách this bind đến function. Arrow function làm code của ta trông ngắn gọn hơn, giúp đơn giản hóa function scoping cũng như từ khóa this.

Dưới đây là 1 số ví dụ về việc sử dụng Arrow fucntion trong Javascript

Ví dụ 4.6. Trường hợp có nhiều tham số

```
// (param1, param2, paramN) => expression

// ES5
var multiply = function(x, y) {
    return x * y;
};

// ES6
var multiply = (x, y) => { return x * y };
```

Ví dụ 4.6 cho cùng một kết quả, tuy nhiên cú pháp với arrow function tốn ít dòng mã hơn. Trong trường hợp chỉ có một biểu thức thì không cần tới dấu ngoặc nhọn: Ví dụ 4.6 có thể viết lại như sau:

```
var multiply = (x, y) => x * y ;
```

Ví dụ 4.7. Trường hợp có 1 tham số. Dấu ngoặc đơn là không bắt buộc khi chỉ có 1 tham số.

```
//ES5
var phraseSplitterEs5 = function phraseSplitter(phrase) {
    return phrase.split(' ');
};

//ES6
var phraseSplitterEs6 = phrase => phrase.split(" ");

//["Love", "Codelearn"]
console.log(phraseSplitterEs6("Love Codelearn"));
```

Ví dụ 4.8. Trường hợp không có tham số. Dấu ngoặc đơn là bắt buộc khi không có tham số.

```
// ES5
var hello = function sayHello() {
     console.log("Hello World");
}

// ES6
var hello = () => { console.log("Hello World"); }

hello(); // Hello World
```

4.1.7. Closures

Closures là một trong những chức năng quyền lực nhất của JavaScript. JavaScript cho phép lồng các function vào nhau, và cấp quyền cho function con, để function con có toàn quyền truy cập vào tất cả các biến và function được định nghĩa bên trong function cha (và tất cả biến và function mà function cha được cấp quyền truy cập đến).

Tuy nhiên, function cha không có quyền truy cập đến các biến và function được định nghĩa bên trong function con. Điều này tạo nên một dạng bảo mật khép kín cho các biến của function con.

Bên cạnh đó, vì function con có quyền truy cập đến scope của function cha, các biến và function được định nghĩa bên trong function cha sẽ vẫn tồn tại dù việc thực thi function cha đã kết thúc, nếu function con xoay sở để tồn tại lâu hơn thời gian sống của function cha. Một closure được tạo ra khi một function con bằng cách nào đó trở nên khả dụng với bất kỳ scope nào bên ngoài function cha.

Ví dụ 4.9.

```
function numberGenerator() {
  // Local “free” variable that ends up within the closure
  var num = 1;
  function checkNumber() {
    console.log(num);
  }
  num++;

  return checkNumber;
}

var number = numberGenerator();
number(); // 2
```

Trong ví dụ 4.9, hàm numberGenerator() tạo ra một biến local num và checkNumber() (một hàm in ra num trong console). Hàm checkNumber() không có bất kỳ biến local nào trong nó. Tuy nhiên, nó có quyền truy cập vào các biến bên ngoài function, bởi vì numberGenerator() là một closure. Do đó, nó có thể sử dụng biến num được khai báo trong numberGenerator() để log num trong console sau khi numberGenerator() được trả lại.

Ví dụ 4.10.

```
function sayHello() {
  var say = function() { console.log(hello); }
  // Local variable that ends up within the closure
  var hello = 'Hello, world!';
```

```
  return say;
}
var sayHelloClosure = sayHello();
sayHelloClosure(); // ‘Hello, world!’
```

Chú ý, biến hello được khai báo sau anonymous function nhưng vẫn có thể truy cập biến hello. Điều này là do biến hello đã được khai báo trong function scope tại thời điểm được tạo ra, làm cho nó có sẵn khi anonymous function được thực thi.

4.1.8. Callback Function

Callback function có thể được hiểu nôm na như sau: callback tức là ta truyền một đoạn code (Hàm A) này vào một đoạn code khác (Hàm B). Tới một thời điểm nào đó, Hàm A sẽ được hàm B gọi lại (callback). Javascript là một ngôn ngữ lập trình hướng sự kiện và bất đồng bộ nên callback function đóng vai trò rất quan trọng, bạn sẽ truyền một callback function vào các sự kiện và xử lý bất đồng bộ đó..

4.1.9. Một số ví dụ tạo hàm trong javascript

Ví dụ 4.11. Viết chương trình kiểm tra một năm có phải là năm nhuận hay không

Năm nhuận là năm chia hết cho 4, nếu chia hết cho 100 thì nó phải chia hết cho 400. Đây là định nghĩa năm nhuận còn chính xác hay không thì mình không biết nhé :D vì có trường hợp nó sai.

```
// khai báo hàm
function kiem_tra_nam_nhuan(nam)
{
    // nếu năm chia hết cho 100
    // thì kiểm tra nó có chia hết cho 400 hay không
    if (nam % 100 == 0)
    {
        // nêu chia hết cho 400 thì là năm nhuận
        if (nam % 400 == 0){
            alert(nam + ' là năm nhuận');
        }
        else { // ngược lại không phải năm nhuận
            alert(nam + ' không phải năm nhuận');
        }
    }
    else if (nam % 4 == 0){ // trường hợp chia hết cho 4 thì là năm
nhuận
        alert(nam + ' là năm nhuận');
    }
    else { // cuối cùng trường hợp không phải năm nhuận
        alert(nam + 'không phải là năm nhuận');
    }
```

```
}
// sử dụng
kiem_tra_nam_nhuan(4);
```

Ví dụ 4.12. Thực hiện lại ví dụ 4.5 nhưng sử dụng return để trả kết quả vè, nếu true thì là năm nhuận, false thì không phải năm nhuận.

```
// khai báo hàm
function kiem_tra_nam_nhuan(nam)
{
    // nếu năm chia hết cho 100
    // thì kiểm tra nó có chia hết cho 400 hay không
    if (nam % 100 == 0)
    {
        // nêu chia hết cho 400 thì là năm nhuận
        if (nam % 400 == 0){
            return true;
        }
        else { // ngược lại không phải năm nhuận
            return false;
        }
    }
    else if (nam % 4 == 0){ // tr.hợp chia hết cho 4 thì là năm nhuận
        return true;
    }
    else { // cuối cùng trường hợp không phải năm nhuận
        return false;
    }
}
// sử dụng
var flag = kiem_tra_nam_nhuan(4);

if (flag){
    alert('là năm nhuận');
}
else {
    alert('không phải là năm nhuận');
}
```

Trong ví dụ 4.12 thay vì alert trực tiếp trong hàm thì ta trả về kết quả true/false, sau đó kiểm tra kết quả này nếu true thì là năm nhuận, nếu false thì không phải là năm nhuận.

Tóm lại, qua bài này ta phải hiểu được cách sử dụng hàm và tạo hàm trong javascript, hiểu được trong javascript không tồn tại khái niệm con trỏ và tham chiếu, hiểu được hai cách sử dụng hàm là hàm có return và hàm không có return.

4.2. Lập trình hướng đối tượng

4.2.1. Tạo class

ECMAScript 5 không có khái niệm về lớp (Class) một cách tường minh, thay vào đó nó mô phỏng một lớp dựa trên 2 khái niệm là function & prototype. Bởi vì ES5 được giới thiệu năm 2011, cho đến nay cú pháp của nó rất phổ biến, tuy nhiên chúng ta không thể phủ nhận rằng tạo ra một lớp theo cách như vậy là khó hiểu đối với các lập trình viên. Trong khi đó, các ngôn ngữ như Java, C#,.. có cách hiện đại hơn để tạo ra một lớp. Phiên bản ES6 ra đời năm 2015 đã kịp thời vá vấn đề này, nó đưa ra cú pháp hiện đại để định nghĩa một lớp.

Với cú pháp mới ECMAScript 6 đã làm giảm sự phức tạp cho lập trình viên khi tạo ra một lớp, và làm giảm sự phức tạp khi tạo ra một lớp con (Subclass). Nhưng về mặt kỹ thuật không có gì thay đổi so với phiên bản trước.

Trong ví dụ 4.13, xây dựng lớp Rectangle (hình chữ nhật), có 2 thuộc tính (property) là width (chiều rộng) và height (chiều cao).

Ví dụ 4.13. Tạo một file nguồn rectangle.js:

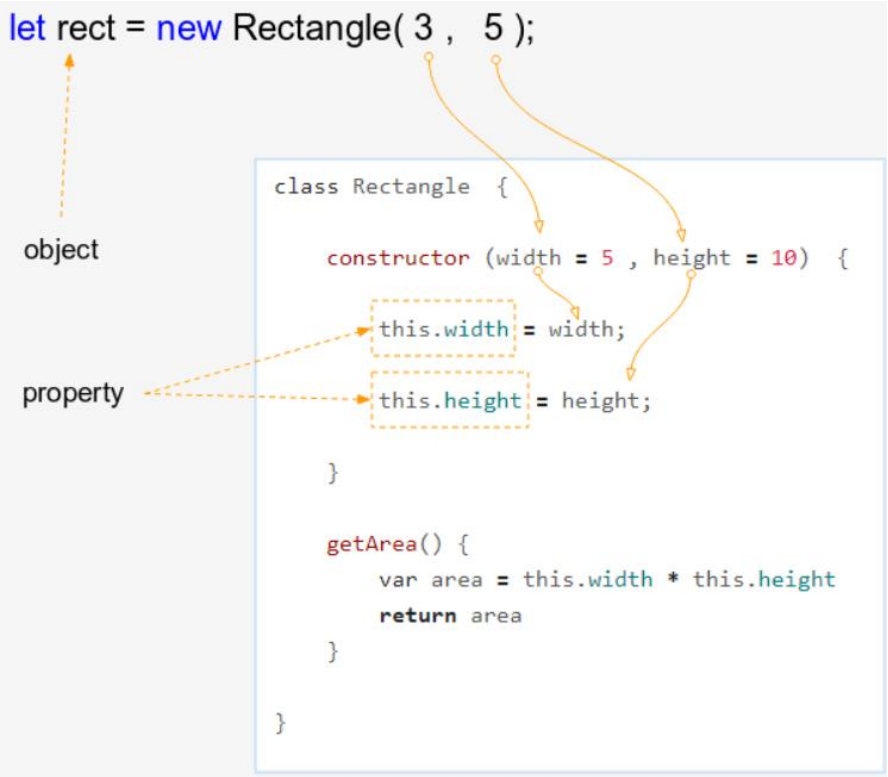
```
// Define a class.
class Rectangle  {

    // Một Constructor có 2 tham số.
    // (Được sử dụng để tạo ra đối tượng)
    // this.width trỏ tới property (thuộc tính) width của lớp.
    constructor (width = 5 , height = 10)  {
        this.width = width;
        this.height = height;
    }

    // Phương thức dùng để tính diện tích hình chữ nhật.
    getArea() {
        var area = this.width * this.height
        return area
    }

}

// Tạo 1 đối tượng của lớp Rectangle thông qua Constructor.
var rect = new Rectangle(3, 5);

console.log("Height "+ rect.height);
console.log("Width "+ rect.width);

// Gọi phương thức
let area = rect.getArea();
console.log("Area "+ area );
```

Kết quả:

```
Height 5
Width 3
```

```
Area 15
```

Điều gì xảy ra khi tạo một đối tượng từ constructor của lớp?



Hình 4.1. Gọi constructor của một lớp

Khi bạn gọi Constructor của lớp, một đối tượng mới sẽ được tạo ra, và các property của đối tượng sẽ được gán giá trị từ các tham số.

Trong ECMAScript mỗi lớp chỉ có nhiều nhất một constructor. Cũng giống như hàm, các tham số của constructor cũng có thể có giá trị mặc định. Vì vậy bạn có thể tạo đối tượng theo nhiều cách khác nhau.

```
// width = 3, height = 5
let rect = new Rectangle(3, 5);

// width = 15, height = 10 (Default)
let rect2 = new Rectangle(15);

// width = 5 (Default), height = 50
let rect3 = new Rectangle(undefined, 50);


// width = 5 (Default), height = 10 (Default)
let rect4 = new Rectangle();
```

4.2.2. Getter & Setter

Trước khi đưa ra khái niệm về Getter & Setter, chúng ta hãy phân tích một tình huống:

Giả sử rằng chúng ta có lớp Person, lớp này có một property là name.

```
class Person  {

    constructor(name) {
      this.name = name;
    }
}
```

Và ta có thể truy cập vào các property của đối tượng, hoặc gán giá trị mới cho nó mà không gặp bất cứ một vấn đề gì.

```
// Create an object
let person = new Person("Tom");

// Access to property name.
console.log( person.name); // Tom

// Assign new value to property name.
person.name = "Jerry"; // !!!

console.log( person.name ); // Jerry
```

Việc truy cập tự do vào một property và thay đổi giá trị của nó từ bên ngoài lớp là thực sự nguy hiểm. Đôi khi bạn muốn có một property mà bên ngoài lớp không thể truy cập được vào nó, hoặc bên ngoài lớp không thể gán giá trị mới cho nó. Getter/Setter cho phép bạn tạo ra một property như vậy.

Từ khóa get đặt trước một phương thức không có tham số của lớp giúp tạo ra một property với tên là tên của phương thức. Phương thức này sẽ được gọi mỗi khi khi chương trình truy cập vào property này.

```
class Person  {

    constructor (name)  {
       // property: __name
       this.__name = name;
    }

    // Getter of property name
    get name()  {
       console.log("Call getter of property 'name'!!");
       return this.__name;
    }

}

// ------------ TEST -----------------
```

```
let person = new Person("Tom");

// Access to property 'name' ==> Call getter
console.log( person.name); // Tom

// Assign new value to property name.
person.name = "Jerry"; // Not Working!!!!

// Access to property 'name' ==> Call getter
console.log( person.name); // Tom
```

Từ khóa set đặt trước một phương thức có 1 tham số của lớp giúp tạo ra một property với tên là tên của phương thức. Phương thức này sẽ được gọi mỗi khi chương trình gán giá trị mới cho property này.

```
class Person  {

    constructor (name)  {
       // property: __name
       this.__name = name;
    }

    // Setter of property name
    set name(newName)  {
       console.log("Call setter of property 'name'!!");
       this.__name = newName;
    }

    // A method
    showInfo()  {
       console.log("Person: " + this.__name);
    }
}

// ------------ TEST -----------------

let person = new Person("Tom");

// Can not access to property 'name'
console.log( person.name); // undefined

// Set new value to property 'name' ==> Call setter
person.name = "Jerry";

person.showInfo(); // Person: Tom
```

Ví dụ 4.14. property với cả hai Getter & Setter. Lưu tại file getter-setter-example.js

```
class Rectangle  {

    constructor (width = 5 , height = 10)  {
        this.__width = width;
        this.height = height;
    }

    // Getter of property 'width'
    get width()  {
      return this.__width;
    }

    // Setter of property 'width'
    set width(newWidth)  {
      if(newWidth > 0) {
          this.__width = newWidth;
      } else {
          console.log("Invalid width " + newWidth);
      }
    }

}

// ------------ TEST ------------------


var rect = new Rectangle(3, 5);

console.log("Height "+ rect.height); // Height: 5
console.log("Width "+ rect.width); // Width: 3

rect.width = -100;

console.log("Height "+ rect.height); // Height: 5
console.log("Width "+ rect.width); // Width: 3


rect.width = 100;

console.log("Height "+ rect.height); // Height: 5
console.log("Width "+ rect.width); // Width: 100
```

Kết quả:

```
Height 5
Width 3
Invalid width -100
Height 5
Width 3
```

```
Height 5
Width 100
```

Các property với tiếp đầu ngữ là 2 dấu gạch dưới (__) thường được các lập trình viên quy ước với nhau rằng đừng sử dụng nó ở bên ngoài lớp. Tuy nhiên quy ước đó có thể bị ai đó phá vỡ. Và vì vậy sử dụng các property như vậy là nguy hiểm.

Nếu muốn có một property thực sự riêng tư (Private) nó lên được đặt tên với tiếp đầu ngữ là dấu thăng (hashtag) (#). Tuy nhiên code này chỉ có thể chạy được với sự trợ giúp của Babel 7 hoặc mới hơn.

Private property

```
class Something {

  constructor(){

    this.#someProperty = "test";
  }

}

const instance = new Something();

console.log(instance.#someProperty); // undefined
```

Ví dụ 4.15. Lưu tại file getter-example2.js

```
class Person  {

    constructor (name)  {
       // Private property: #name
       this.#name = name;
    }

    // Getter of property name
    get name()  {
       console.log("Call getter of property 'name'!!");
       return this.#name;
    }

}
```

4.2.3. Trường tĩnh (Static Field)

Từ khóa static xuất hiện trong khai báo Getter hoặc Setter giúp bạn định nghĩa một trường tĩnh (static field). Bạn có thể truy cập vào các static field thông qua tên lớp.

Các trường tĩnh (static field) có giá trị cố định (Không thể thay đổi) được gọi là các trường hằng số tĩnh (constant static field).

Ví dụ 4.16. File static-field-example1.js

```
class Employee {

   constructor (fullName, age)  {
     this.fullName = fullName;
     if(age < Employee.MIN_AGE || age > Employee.MAX_AGE)  {
        throw "Invalid Age " + age;
     }
     this.age = age;
   }

   // A static field: MIN_AGE
   static get MIN_AGE() {
      return 18;
   }

   // A static field: MAX_AGE
   static get MAX_AGE() {
     if(!Employee.__MAXA)  {
        Employee.__MAXA = 60;
     }
     return Employee.__MAXA;
   }

   static set MAX_AGE(newMaxAge)  {
      Employee.__MAXA = newMaxAge;
   }

}

// ---- TEST ---------

console.log("Mininum Age Allowed: " + Employee.MIN_AGE);
console.log("Maximum Age Allowed: " + Employee.MAX_AGE);

// Set new Maximum Age:
Employee.MAX_AGE = 65;

console.log("Maximum Age Allowed: " + Employee.MAX_AGE);

let baby = new Employee("Some Baby", 1); // Error!!
```

Ta có một cách khác để khai báo một trường tĩnh cho lớp, tuy nhiên các trường tĩnh được tạo ra bởi cách này sẽ không là một hằng số (Constant) vì giá trị của nó có thể thay đổi.

Ví dụ 4.17. File static-field-example2.js

```
class Employee {
```

```
   constructor (fullName, age)  {
     this.fullName = fullName;
     if(age < Employee.MIN_AGE || age > Employee.MAX_AGE)  {
        throw "Invalid Age " + age;
     }
     this.age = age;
   }
}

Employee.MIN_AGE = 18;
Employee.MAX_AGE = 60;

// ---- TEST ---------

console.log("Mininum Age Allowed: " + Employee.MIN_AGE);
console.log("Maximum Age Allowed: " + Employee.MAX_AGE);

// Set new Maximum Age:
Employee.MAX_AGE = 65;

console.log("Maximum Age Allowed: " + Employee.MAX_AGE);

let baby = new Employee("Some Baby", 1); // Error!!
```

4.2.4. Phương thức tĩnh

Từ khóa static xuất hiện trong khai báo một phương thức của lớp giúp bạn định nghĩa một phương thức tĩnh (static method). Bạn có thể gọi phương thức tĩnh thông qua tên của lớp. Phương thức tĩnh không thể gọi thông qua đối tượng của lớp. Phương thức tĩnh thường được sử dụng như là một hàm tiện ích của ứng dụng.

Ví dụ 4.18. File point.js

```
class Point {

  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  // Tính khoảng cách giữa 2 điểm
  static distance( point1, point2) {
    const dx = point1.x - point2.x;
    const dy = point1.y - point2.y;

    return Math.hypot(dx, dy);
  }
}
```

```
// --- TEST ---
let point1 = new Point( 5, 10);
let point2 = new Point( 15, 20);

// Distance
let d = Point.distance(point1, point2);

console.log("Distance between 2 points: " + d);
```

4.2.5. Toán tử so sánh đối tượng

Trong ECMAScript, khi bạn tạo một đối tượng thông qua constructor sẽ có một thực thể thực sự được tạo ra nằm trên bộ nhớ, nó có một địa chỉ xác định.

Một phép toán gán đối tượng AA bởi một đối tượng BB không tạo ra thêm thực thể trên bộ nhớ, nó chỉ là trỏ địa chỉ của AA tới địa chỉ của BB.

Toán tử === dùng để so sánh địa chỉ 2 đối tượng trỏ đến, nó trả về true nếu cả 2 đối tượng cùng trỏ tới cùng một địa chỉ trên bộ nhớ. Toán tử !== cũng sử dụng để so sánh 2 địa chỉ của 2 đối tượng trỏ đến, nó trả về true nếu 2 đối tượng trỏ tới 2 địa chỉ khác nhau.

Ví dụ 4.19. File identify-operator-example.js

```
// Define a class.
class Rectangle  {
    constructor (width = 5 , height = 10)  {
        this.width = width;
        this.height = height;
    }
    getArea() {
        var area = this.width * this.height;
        return area;
    }
}

// Tạo đối tượng: r1
let r1 = new Rectangle( 20,  10);

// Tạo đối tượng: r2
let r2 = new Rectangle( 20, 10);

let r3 = r1;


let  b12  =   r1 === r2; // false
let  b13  =   r1 === r3; // true
```

```
console.log("r1 === r2 ? " + b12); // false
console.log("r1 === r3 ? " + b13); // true


var bb12  = r1 !== r2; // true
var bb13  = r1 !== r3; // false


console.log("r1 !== r2 ? " + bb12); // true
console.log("r1 !== r3 ? " + bb13); // false
```

4.2.6. Thừa kế và đa hình

4.2.6.1. Thừa kế trong ECMAScript

ECMAScript cho phép tạo một lớp mở rộng từ một hoặc nhiều lớp khác. Lớp này được gọi là lớp dẫn xuất (Derived class), hoặc đơn giản gọi là lớp con.

Lớp con sẽ thừa kế các property, phương thức và các thành viên khác từ lớp cha. Nó cũng có thể ghi đè (override) các phương thức từ lớp cha. Trong ECMAScript mỗi lớp chỉ có duy nhất 1 constructor. Nếu một lớp không tự định nghĩa một constructor của riêng nó, nó sẽ được thừa kế constructor của lớp cha. Ngược lại nếu lớp con định nghĩa một constructor, nó sẽ không được thừa kế constructor của lớp cha.

Khác với Java, CSharp và một vài ngôn ngữ khác, ECMAScript cho phép đa thừa kế. Một lớp có thể được mở rộng (extends) từ một hoặc nhiều lớp cha.

Chúng ta cần một vài class tham gia vào minh họa.

- Animal: Class mô phỏng một lớp Động vật.
- Duck: Class mô phỏng lớp vịt, là một class con của Animal.
- Cat: Class mô phỏng lớp mèo, là một class con của Animal
- Mouse: Class mô phỏng lớp chuột, là một class con của Animal.

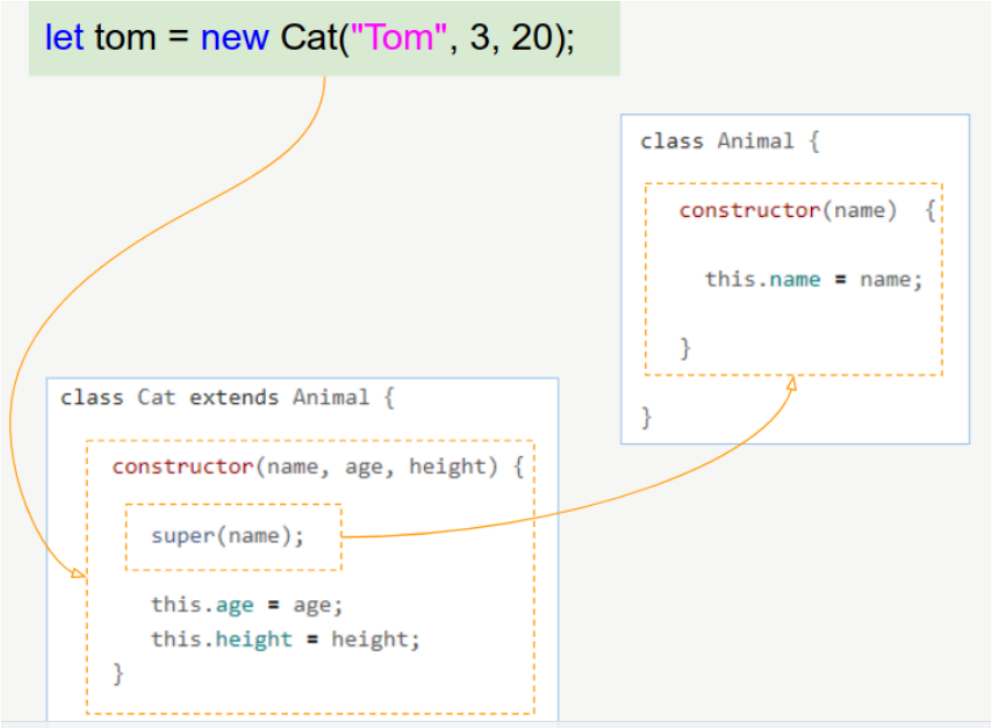
Trong ECMAScript, constructor sử dụng để tạo một đối tượng, và gán giá trị cho các property. Constructor của lớp con bao giờ cũng gọi tới constructor của lớp cha để gán giá trị cho các property của lớp cha, sau đó nó mới gán giá trị cho các property của nó.

Ví dụ 4.20. Cat là lớp con thừa kế từ lớp Animal, nó cũng có các property riêng của nó. File inherit-example1.js

```
class Animal {

   constructor(name)  {
     this.name = name;
   }

   showInfo()  {
```

```
      console.log("I'm " + this.name);
    }

    move()  {
      console.log("Moving..");
    }

}

class Cat extends Animal {
    constructor(name, age, height) {
       super(name);
       // Cat's properties:
       this.age = age;
       this.height = height;
    }

    // Ghi đè (override) phương thức cùng tên của lớp cha.
    showInfo()  {
      console.log("My name is " + this.name);
    }

    // Other method...
    sleep()  {
       console.log("Sleeping..");
    }
}

// ------------- TEST --------------------
let tom = new Cat("Tom", 3, 20);

console.log("Call move() method");

tom.move();

console.log("\n");
console.log("Call showInfo() method");

tom.showInfo();
```

Điều gì đã xảy ra khi khởi tạo một đối tượng từ constructor? Nó sẽ gọi lên constructor của lớp cha như thế nào? Bạn hãy xem hình minh họa dưới đây.



Hình 4.2. Gọi constructor lớp cha trong lớp con

Với hình minh họa trên ta thấy rằng, constructor của lớp cha được gọi trong constructor của lớp con, nó sẽ gán giá trị cho các property của lớp cha, sau đó các property của lớp con cũng được gán giá trị.

4.2.6.2. Ghi đè phương thức

Mặc định lớp con được thừa kế các phương thức từ lớp cha, tuy nhiên lớp con có thể ghi đè (override) phương thức của lớp cha.

Ví dụ 4.21. File inheritance-example2.js

```
class Animal {

   constructor(name)  {
     this.name = name;
   }

   showInfo()  {
     console.log("I'm " + this.name);
   }

   move()  {
     console.log("Moving..");
   }

}

class Mouse extends Animal {
```

```
    constructor(name, age, height) {
       super(name);

       this.age = age;
       this.height = height;
    }

    // Ghi đè (override) phương thức cùng tên của lớp cha.
    showInfo()  {
      // Gọi phương thức showInfo() của lớp cha.
      super.showInfo();
      console.log ("Age: " + this.age);
      console.log ("Height: " + this.height);
    }

    // Ghi đè (override) phương thức cùng tên của lớp cha.
    move()  {
      console.log("Mouse Moving..");
    }
}

// ------------- TEST --------------------
let jerry = new Mouse("Jerry", 3, 5);

console.log("Call move() method");

jerry.move();

console.log("\n");
console.log("Call showInfo() method");

jerry.showInfo();
```

4.2.6.3. Phương thức trừu tượng

Khái niệm về một phương thức trừu tượng (abstract method) hoặc một lớp trừu tượng.(abstract class) có trong các ngôn ngữ như Java, C#. Nhưng nó không có một cách rõ ràng trong ECMAScript. Tuy nhiên chúng ta có cách để định nghĩa nó.

Một lớp được gọi là trừu tượng (abstract) định nghĩa ra các phương thức trừu tượng và lớp con phải ghi đè (override) các phương thức này nếu muốn sử dụng chúng. Các phương thức trừu tượng luôn ném ra ngoại lệ "Not Implemented Error".

Ví dụ 4.22. File abstract-example.js

```
// Một lớp trừu tượng (Abstract class).
class AbstractDocument {

    constructor(name) {
        this.name = name
```

```
    }

    // Một phương thức không thể sử dụng được, vì nó luôn ném ra lỗi.
    show() {
        // Not Implemented Error
        throw "Subclass must implement abstract method";
    }
}

class PDF extends AbstractDocument {

    // Ghi đè phương thức của lớp cha
    show() {
        console.log("Show PDF document:", this.name);
    }
}



class Word extends AbstractDocument {
    show() {
        console.log("Show Word document:", this.name)
    }
}



// -------------------- TEST -------------------------
let doc1 = new PDF("Python tutorial");
let doc2 = new Word("Java IO Tutorial");
let doc3 = new PDF("ECMAScript Date & Time Tutorial");

let documents = [doc1, doc2, doc3];


for (let i = 0; i < documents.length; i++) {
    documents[i].show();
}

let doc4 = new AbstractDocument("An Abstract Document");

doc4.show(); // Throw Error!!!
```

Ví dụ 4.22 thể hiện tính đa hình (Polymorphism) trong ECMAScript. Một đối tượng Document (tài liệu) có thể được thể hiện ở các hình thái khác nhau (PDF, Word, Excel, ...).

Một ví dụ khác minh họa về tính đa hình: Khi tôi nói về một người Châu Á, nó khá trừu tượng, anh ta có thể là một người Nhật Bản, người Việt Nam, hoặc một người Ấn Độ, ... Tuy nhiên đều có đặc điểm đặc trưng của người Châu Á.

4.2.6.4. Toán tử instanceof, typeof

Toán tử instanceof giúp kiểm tra xem một "cái gì đó" có phải là một đối tượng của một lớp nào đó hay không.

Ví dụ 4.23. instanceof-example.js

```
class Person {

}

// A Child class of Person
class AsianPerson extends Person {

}

class Triangle {

}

let person = new Person();
let asianPerson = new AsianPerson();

let triangle = new Triangle();

let isPerson = person instanceof Person;
console.log( isPerson ); // true

console.log( asianPerson instanceof Person ); // true

console.log( person instanceof AsianPerson ); // false
console.log( triangle instanceof Person ); // false
console.log( triangle instanceof AsianPerson ); // false
```

Toán tử typeof được sử dụng để kiểm tra kiểu của một "cái gì đó", kết quả nhận được là tên của kiểu.

Type	Result
undefined	"undefined"
null	"object"
boolean	"boolean"
number	"number"

String	"string"
Symbol (new in ECMAScript 6)	"symbol"
Host object (provided by the JS environment)	Implementation-dependent
Function object (implements [[Call]] in ECMA-262 terms)	"function"
Any other object	"object"

Ví dụ 4.24. File typeof-example.js

```
// A Class:
class Person {

}
// An Object:
let person = new Person();

console.log( typeof Person ); // function
console.log( typeof person ); // object

// null:
console.log( typeof null ); // object

// A Number:
let intValue = 100;

console.log( typeof intValue ); // number

// NaN (Not a Number)
console.log( typeof NaN); // number

// A String
let strValue = "Hello";

console.log( typeof strValue ); // string

// A Boolean:
let boolValue = true;

console.log( typeof boolValue ); // boolean

// undefined
console.log( typeof undefined); // undefined

// An Array
let years = [1980, 2003, 2018];
```

```
console.log( typeof years ); // object

// A Function
let myFunc = function()  {

}

console.log( typeof myFunc ); // function

// A Symbol
let mySymbol =   Symbol();

console.log( typeof mySymbol ); // symbol
```

Is Sub Class?

Giả sử bạn có 2 lớp A & B. Ví dụ dưới đây giúp bạn kiểm tra xem A có phải là lớp con của B hay không.

```
let isChild = A === B || A.prototype instanceof B;
```

Ví dụ 4.25. File issubclass-example.js

```
class Animal {

}

class Cat extends Animal {

}

class AsianCat extends Cat {

}

// ------------- TEST --------------------

console.log("AsianCat === Animal? " + (AsianCat === Animal)); // false

let isSubClass1 = AsianCat === Animal || AsianCat.prototype instanceof
Animal;

console.log("AsianCat is child of Animal? " + isSubClass1); // true

let isSubClass2 = AsianCat === Animal || Animal.prototype instanceof
AsianCat;

console.log("Animal is child of AsianCat? " + isSubClass2); // false
```

4.2.6.5. Đa hình với hàm

Các ngôn ngữ như Java, C# rất chặt chẽ về kiểu dữ liệu. Vì vậy khi gọi một phương thức (Hàm) bạn phải truyền vào đúng kiểu dữ liệu ứng với các tham số. Trong ECMAScript khi gọi một hàm bạn có thể truyền vào tham số với kiểu dữ liệu bất kỳ, rủi ro có thể xẩy ra, và người lập trình phải tự quản lý các rủi ro đó.

Ở đây ta tạo ra 2 lớp English và French. Cả hai lớp này đều có phương thức greeting(). Cả hai tạo ra các lời chào khác nhau. Tạo ra 2 đối tượng tương ứng từ 2 lớp trên và gọi hành động của 2 đối tượng này trong cùng một hàm (Hàm intro)

Ví dụ 4.26. File polymorphism-example.js

```
class English {

    greeting() {
        console.log("Hello");
    }
}

class French {
    greeting() {
        console.log("Bonjour");
    }
}

function intro(language) {
    language.greeting();
}

// -------------- TEST ----------------------

let flora = new English();
let aalase = new French();

// Call function:
intro(flora);
intro(aalase);

let someObject = {};

intro(someObject);// Error!!!
```

Chạy ví dụ: Lỗi

Sửa code của ví dụ 4.26, thêm vào các kiểm tra cần thiết để tránh các rủi ro cho ứng dụng của bạn:

Ví dụ 4.27. File polymorphism-example2.js

```
class English {

    greeting() {
        console.log("Hello");
    }
}

class French {
    greeting() {
        console.log("Bonjour");
    }
}

function intro(language) {
    // Check type of 'language' object.
    if(language instanceof English || language instanceof French)  {
      language.greeting();
    }
}

// -------------- TEST ----------------------

let flora = new English();
let aalase = new French();

// Call function:
intro(flora);
intro(aalase);

let someObject = {};

intro(someObject);
```

Áp dụng kiến trúc "Lớp & Thừa kế" giúp ứng dụng của bạn dễ dàng quản lý hơn và tránh các sai sót.

Ví dụ 4.28. File polymorphism-example3.js

```
// A Base class
class Language {

}
class English extends Language {

    greeting() {
        console.log("Hello");
    }
}
```

```
class French  extends Language {
    greeting() {
        console.log("Bonjour");
    }
}

function intro(language) {
    // Check type of 'language' object.
    if(language instanceof Language)  {
      language.greeting();
    }
}

// -------------- TEST ----------------------

let flora = new English();
let aalase = new French();

// Call function:
intro(flora);
intro(aalase);

let someObject = {};

intro(someObject);
```

Chương 5. HTML - DOM

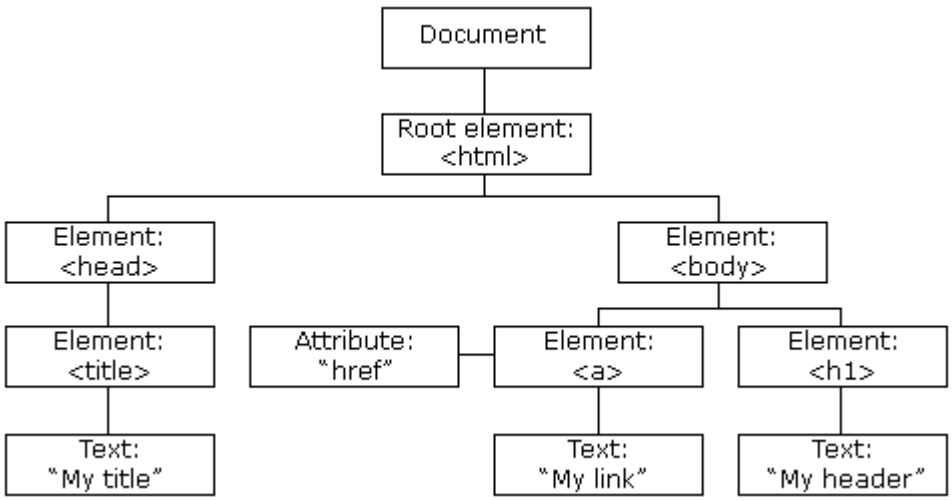
5.1. Sơ đồ cây HTML DOM

5.1.1. HTML là gì?

HTML đã quá quen với lập trình viên, vậy nó được hiểu như thế nào? Như các bạn đã biết HTML là ngôn ngữ đánh dấu siêu văn bản, nó là một XML namespace, hay được hiểu là tập các thẻ XML mà trình duyệt nào cũng có thể đọc được. Chúng ta nhìn vào một file HTML thì nhìn thấy text, còn trình duyệt nhìn vào sẽ thấy cây DOM.

5.1.2. DOM là gì?

DOM là viết tắt của chữ Document Object Model, dịch tạm ra là mô hình các đối tượng trong tài liệu HTML. Như các bạn biết trong mỗi thẻ HTML sẽ có những thuộc tính (Properties) và có phân cấp cha - con với các thẻ HTML khác. Sự phân cấp và các thuộc tính của thẻ HTML này ta gọi là selector và trong DOM sẽ có nhiệm vụ xử lý các vấn đề như đổi thuộc tính của thẻ, đổi cấu trúc HTML của thẻ,... DOM được dùng để truy xuất các tài liệu dạng HTML và XML, có dạng một cây cấu trúc dữ liệu, và thông thường mô hình DOM độc lập với hệ điều hành và dựa theo kỹ thuật lập trình hướng đối tượng để mô tả tài liệu.



Hình 5.1. Cây DOM HTML của các đối tượng

Chúng ta có thể thấy tất cả các thẻ HTML sẽ được quản lý trong đối tượng document (DOM), thẻ cao nhất là thẻ html, tiếp đến là phân nhánh body và head. Bên trong head thì có những thẻ như style, title,... và bên trong body chứa bất kì một thẻ nào đó là thành phần của HTML. Như vậy ta có thể hiểu trong Javascript để thao tác được với các thẻ HTML ta phải thông qua đối tượng documnent (DOM).

Với DOM, JavaScript được tất cả sức mạnh cần thiết để tạo ra HTML động:

- JavaScript có thể thay đổi tất cả các phần tử HTML trong trang

- JavaScript có thể thay đổi tất cả các thuộc tính HTML trong trang
- JavaScript có thể thay đổi tất cả các phong cách CSS trong trang
- JavaScript có thể loại bỏ các yếu tố HTML và thuộc tính hiện tại
- JavaScript có thể thêm các yếu tố HTML mới và các thuộc tính
- JavaScript có thể phản ứng với tất cả các sự kiện HTML hiện trong trang
- JavaScript có thể tạo ra các sự kiện HTML mới trong trang

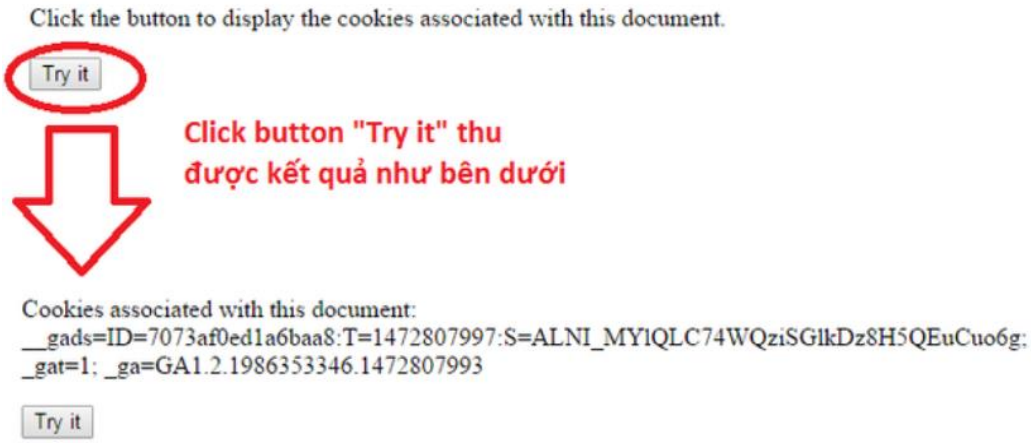
Document Object Model - DOM ("Mô hình Đối tượng Tài liệu"), là một giao diện lập trình ứng dụng (API). DOM được dùng để truy xuất các tài liệu dạng HTML và XML, có dạng một cây cấu trúc dữ liệu, và thông thường mô hình DOM độc lập với hệ điều hành và dựa theo kỹ thuật lập trình hướng đối tượng để mô tả tài liệu.

Thời kì sơ khai các thành phần trong một tài liệu HTML mô tả bằng các phiên bản khác nhau của DOM được hiển thị bởi các chương trình duyệt web thông qua JavaScript vì chưa có một chuẩn thống nhất nào. Điều này buộc World Wide Web Consortium (W3C) phải đưa ra một loạt các mô tả kĩ thuật về tiêu chuẩn cho DOM để thống nhất mô hình này.

Ví dụ 5.1.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo">Click the button to display the cookies associated with
this document.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML =
    "Cookies associated with this document: " + document.cookie;
}
</script>

</body>
</html>
```

Kết quả:



5.1.3. HTML DOM là gì?

HTML DOM là một chuẩn mô hình object và programming interface cho HTML. nó định nghĩa:

- HTML elements như là objects
- properties của tất cả HTML elements
- methods để truy cập đến tất cả HTML elements
- events cho tất cả HTML elements

HTML DOM là một tiêu chuẩn cho phép bạn thực hiện những công việc thao tác với bất kì một trang web: get, change, add, or delete các thành phần của HTML.

5.1.4. DOM Attributes

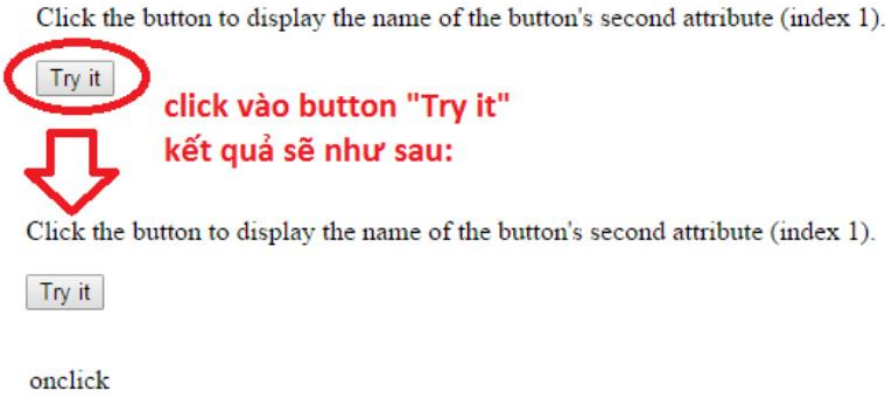
Attributes property là một khái niệm của DOM trả về một tập hợp các thuộc tính của nút được chỉ định, như một đối tượng NamedNodeMap. Các nút có thể được truy cập bởi các con số chỉ số, và chỉ số bắt đầu từ 0. Và bằng số chỉ mục là hữu ích cho đi qua tất cả các thành phần của Attributes: Bạn có thể sử dụng các property của đối tượng NamedNodeMap để xác định số lượng các thuộc tính, lặp qua tất cả sau đó bạn có thể tính các nút và trích xuất các thông tin mà bạn muốn.

Ví dụ 5.2.

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to display the name of the button's second
attribute (index 1).</p>

<button id="myBtn" onclick="myFunction()">Try it</button>
```

```
<p id="demo"></p>

<script>
function myFunction() {
    var x = document.getElementById("myBtn").attributes[1].name;
    document.getElementById("demo").innerHTML = x;
}
</script>

</body>
</html>
```

Kết quả:



Nói tóm lại, attribute là thuộc tính của các phần tử DOM. Attribute cho biết các đặc điểm của phần tử DOM đó.

5.1.5. Property

Property cung cấp thêm thông tin về các thành phần trong HTML, các phần tử DOM được ánh xạ thành các đối tượng Javascript khi ta sử dụng Javascript để thao tác với DOM.

```
var paragraphs = document.getElementsByTagName('P'); // (1)
var firstParagraph = paragraphs[0]; // (2)
```

Phần tử <P> đầu tiên của document được ánh xạ thành đối tượng Javascript được trỏ bởi biến firstParagraph, (getElementsByTagName() trả về một cấu trúc dữ liệu tương tự 1 mảng các Node được gọi là NodeList, tức là có thuộc tính length, và truy xuất thông qua chỉ số). Phần tử DOM được ánh xạ thành đối tượng có thuộc tính và phương thức trong Javascript. Thuộc tính của đối tượng Javascript, được gọi là property. Chung quy lại thì:

Attribute là thuộc tính các phần tử DOM còn Property là thuộc tính của đối tượng Javascript.

Chú ý :

- Attribute của DOM element và property của Javascript object tương ứng thì không có quan hệ 1 - 1. Chẳng hạn như attribute class được ánh xạ thành property className và attribute for được ánh xạ thành htmlFor
- Dùng phương thức getAttribute(name) và setAttribute('name', 'value'). Để thao tác với property để tương tác với attribute, dùng dot notation (element.property = value)
- Attribute value của phần tử <input type="text" value="type to search"/> chính vì vậy thay đổi property không chắc chắn làm thay đổi attribute và ngược lại.

Nói một cách khái quát thì nếu giá trị trong input được định nghĩa là 'type to search', thì propery tương ứng cũng như vậy. Sau khi người dùng nhập dữ liệu, 'abc' chẳng hạn, thì property sẽ được thiết lập thành 'abc', tuy vậy, attribute vẫn không thay đổi.

```
console.log(input.getAttribute('value'));
// type to search

console.log(input.value);
// 'abc'
```

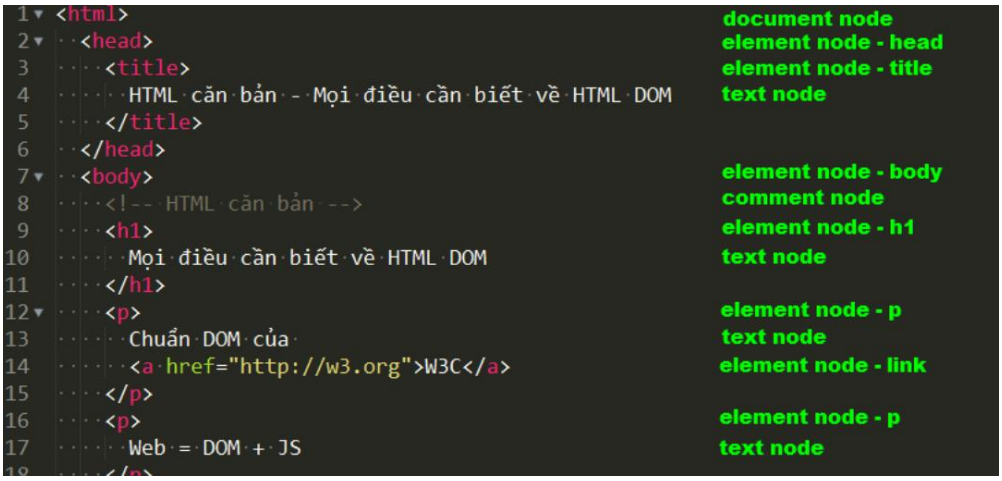
Mặc dù nghĩa dịch sang tiếng việt giống nhau nhưng attribute và property thuộc về 2 thế giới hoàn toàn khác nhau. Cần nắm rõ để tránh các hiểu lầm không cần thiết.

5.1.6. Cây cấu trúc trong DOM

Đối với HTML DOM, cấu trúc dạng cây gọi là DOM Tree có nghĩa là mọi thành phần đều được xem là 1 nút (node), được biểu diễn trên 1 cây . Các phần tử khác nhau sẽ được phân loại nút khác nhau nhưng quan trọng nhất là 3 loại: nút gốc (document node), nút phần tử (element node), nút văn bản (text node).

- Nút gốc: thường được biểu diễn bởi thẻ <html> là thành phần của HTML.
- Nút phần tử: biểu thị cho 1 phần tử HTML.
- Nút văn bản: mỗi đoạn kí tự trong tài liệu HTML, bên trong 1 thẻ HTML đều là 1 nút văn bản. Đó có thể là tên trang web trong thẻ <title>, tên đề mục trong thẻ <h1>, hay một đoạn văn trong thẻ

Ngoài ra còn có nút thuộc tính (attribute node) và nút chú thích (comment node).



Quan hệ giữa các nút

- Nút gốc (root document) luôn luôn là nút đầu tiên.
- Tất cả các nút không phải là nút gốc và đều chỉ có 1 nút cha (parent).
- Một nút có thể có một hoặc nhiều con, hoặc cũng có thể không có con nào.
- Những nút anh em (siblings) thì có cùng nút cha.
- Trong các nút anh em (siblings), nút đầu tiên được gọi là anh cả (firstChild) và nút cuối cùng là em út (lastChild).

5.2. Phương thức và thuộc tính

5.2.1. Thao tác với DOM

Việc thao tác với DOM cho bạn sức mạnh "thay đổi thế giới", vì mọi nội dung đều có thể được cập nhật động thông qua các thuộc tính và phương thức của DOM. Tất tần tật từ thay đổi định dạng chữ, nội dung chữ đến thay đổi cấu trúc các nút và cả thêm nút mới, bạn đều có thể làm được. Do đó, để sáng tạo nội dung tốt, bạn cần hiểu rõ cách thao tác DOM và ý nghĩa của từng thuộc tính, phương thức.

5.2.2. Các Thuộc tính và Phương thức thường gặp

5.2.2.1. Thuộc tính

- id: Định danh – là duy nhất cho mỗi phần tử nên thường được dùng để truy xuất DOM trực tiếp và nhanh chóng.
- className: Tên lớp – Cũng dùng để truy xuất trực tiếp như id, nhưng 1 className có thể dùng cho nhiều phần tử.
- tagName: Tên thẻ HTML.
- innerHTML: Trả về mã HTML bên trong phần tử hiện tại. Đoạn mã HTML này là chuỗi kí tự chứa tất cả phần tử bên trong, bao gồm các nút phần tử và nút văn bản.

- outerHTML: Trả về mã HTML của phần tử hiện tại. Nói cách khác, outerHTML = tagName + innerHTML.
- textContent: Trả về 1 chuỗi kí tự chứa nội dung của tất cả nút văn bản bên trong phần tử hiện tại.
- attributes: Tập các thuộc tính như id, name, class, href, title…
- style: Tập các định dạng của phần tử hiện tại
- value: Lấy giá trị của thành phần được chọn thành một biến.

5.2.2.2. Phương thức

- getElementById(id): Tham chiếu đến 1 nút duy nhất có thuộc tính id giống với id cần tìm.
- getElementsByTagName(tagname): Tham chiếu đến tất cả các nút có thuộc tính tagName giống với tên thẻ cần tìm, hay hiểu đơn giản hơn là tìm tất cả các phần tử DOM mang thẻ HTML cùng loại. Nếu muốn truy xuất đến toàn bộ thẻ trong tài liệu HTML thì hãy sử dụng document.getElementsByTagName('*').
- getElementsByName(name): Tham chiếu đến tất cả các nút có thuộc tính name cần tìm.
- getAttribute(attributeName): Lấy giá trị của thuộc tính.
- setAttribute(attributeName, value): Sửa giá trị của thuộc tính.
- appendChild(node): Thêm 1 nút con vào nút hiện tại.
- removeChild(node): Xóa 1 nút con khỏi nút hiện tại.

Mặt khác, các phần tử DOM đều là các nút trên cây cấu trúc DOM. Chúng sở hữu thêm các thuộc tính quan hệ để biểu diễn sự phụ thuộc giữa các nút với nhau. Nhờ các thuộc tính quan hệ này, chúng ta có thể truy xuất DOM gián tiếp dựa trên quan hệ và vị trí của các phần tử:

5.2.2.3. Thuộc tính quan hệ

- parentNode: Nút cha
- childNodes: Các nút con
- firstChild: Nút con đầu tiên
- lastChild: Nút con cuối cùng
- nextSibling: Nút anh em liền kề sau

- previousSibling: Nút anh em liền kề trước

Ta có thể xem đầy đủ ở website: https://www.w3schools.com/jsref/dom_obj_all.asp

5.3. Đối tượng document

5.3.1. Giới thiệu

Đối tượng document trong JavaScript đại diện cho toàn bộ tài liệu HTML. Khi tài liệu html được tải trong trình duyệt, nó sẽ trở thành một đối tượng document. Nó là phần tử gốc đại diện cho tài liệu html. Nó có các thuộc tính và phương thức. Nhờ sự giúp đỡ của đối tượng document, chúng tôi có thể thêm nội dung động vào trang web.

Đối tượng document là thuộc tính của đối tượng window, vì vậy nó có thể được truy cập bằng cách:

```
window.document
```

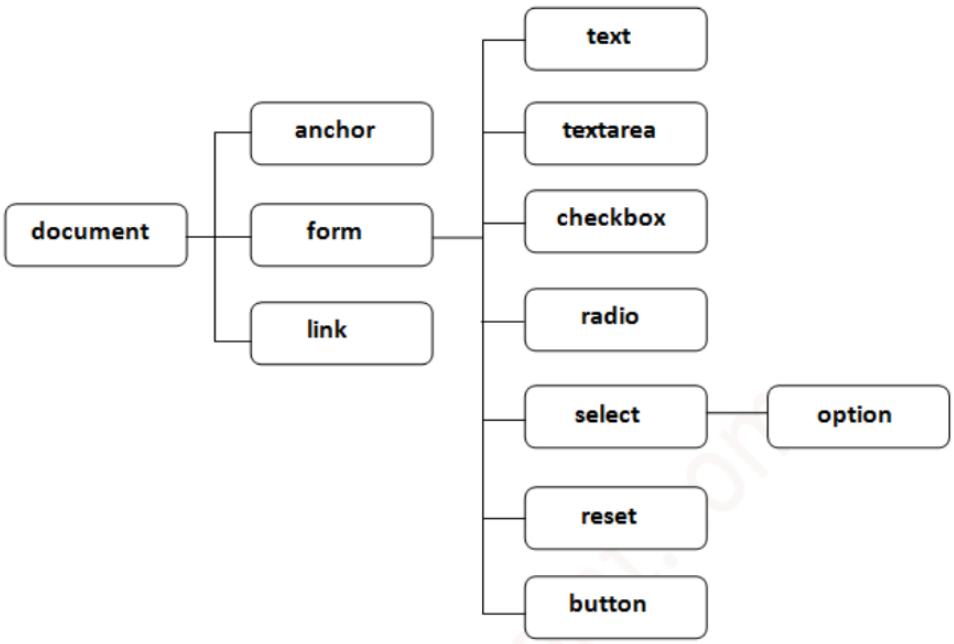
hoặc:

```
document
```

Theo W3C - "Document Object Model (DOM) - Mô hình đối tượng document (DOM) là giao diện nền tảng và trung lập ngôn ngữ cho phép các chương trình và tập lệnh tự động truy cập và cập nhật nội dung, cấu trúc và css của tài liệu".

5.3.2. Các thuộc tính của đối tượng document

Các thuộc tính của đối tượng docment được thể hiện trong hình 5.2.



Hình 5.2. Các thuộc tính của đối tượng document

5.3.3. Các phương thức của đối tượng document

Chúng ta có thể truy cập và thay đổi nội dung của trang web bằng các phương thức của đối tượng document.

Các phương thức quan trọng của đối tượng document như sau:

Phương thức	Mô tả
write("string")	viết chuỗi đã cho trên doucment.
writeln("string")	viết chuỗi đã cho trên doucment với ký tự newline ở cuối.
getElementById()	trả về phần tử có giá trị id đã cho.
getElementsByName()	trả về tất cả các phần tử có giá trị name đã cho.
getElementsByTagName()	trả về tất cả các phần tử có tên thẻ đã cho.
getElementsByClassName()	trả về tất cả các phần tử có class đã cho.

5.3.4. Truy cập giá trị trường bằng đối tượng document

Trong ví dụ 5.3, chúng ta sẽ nhận được giá trị của văn bản đầu vào từ người dùng. Ở đây, chúng ta đang sử dụng document.form1.name.value để lấy giá trị của trường name:

- document: là phần tử gốc đại diện cho tài liệu html.
- form1 : là tên của biểu mẫu.
- name : là tên thuộc tính của văn bản đầu vào.
- value : là thuộc tính, trả về giá trị của văn bản đầu vào.

Ví dụ 5.3. Sử dụng đối tượng document

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Vi du doi tuong document trong JavaScript</title>
<script type="text/javascript">
    function printValue() {
        var name = document.form1.name.value;
        alert("Welcome: " + name);
    }
</script>
</head>
<body>
    <form name="form1">
        Enter Name: <input type="text" name="name" />
```

```
            <input type="button" onclick="printValue()" value="Display
Name" />
        </form>
    </body>
    </html>
```

Kết quả:

Enter Name: Display Name

5.4. DOM elements

5.4.1. Tìm thẻ HTML theo ID

Để truy xuất tới một thẻ HTML theo ID ta sử dụng cú pháp sau:

```
var element = document.getElementById('idname');
```

Ví dụ 5.4.

```
<html>
  <body>
    <input type="text" value="Freetuts.net" id="website" />

    <script language="javascript">
      // Lấy thẻ input
      var element = document.getElementById('website');

      // Lấy giá trị của thẻ input
      document.write(element.value);
    </script>
  </body>
</html>
```

5.4.2. Tìm thẻ HTML theo tên của thẻ HTML

Tên thẻ HTML chính là tên các thẻ như p, a, div, ... Và ta sẽ truy xuất tới nó bằng cú pháp sau:

```
var element = document.getElementsByTagName('tagname');
```

Ví dụ 5.5.

```
<html>
  <body>

    <input type="text" value="Freetuts.net"/>

    <script language="javascript">
      // Lấy thẻ input
      var element = document.getElementsByTagName('input');
```

```
      // Lấy giá trị của thẻ input
      document.write(element[0].value);
    </script>
  </body>
</html>
```

Trong ví dụ này có sự khác biệt đó là đoạn code element[0].value. Tại sao lại như vậy? Như bạn biết trong một trang web có thể có nhiều thẻ HTML giống nhau (ví dụ có hai thẻ div) nên hàm getElementsByTagName() sẽ trả về một mảng các object (xem bài mảng trong javascript) chứ không phải là một object nữa, chính vì vậy ta sẽ lấy input thứ nhất nên truyền số 0 vào.

5.4.3. Tìm thẻ HTML theo tên class

Để tìm các thẻ HTML có class nào đó thì ta dùng cú pháp sau:

```
var element = document.getElementsByClassName('input');
```

Tương tự như tìm theo tên thẻ HTML thì tìm theo tên class sẽ trả về một mảng các object nên bạn sẽ phải sử dụng cú pháp truy xuất mảng để chọn đúng đối tượng muốn lấy.

Ví dụ 5.6.

```
<html>
  <body>

    <input type="text" value="Freetuts.net" class="website"/>

    <script language="javascript">
      // Lấy thẻ input
      var element = document.getElementsByClassName('website');

      // Lấy giá trị của thẻ input
      document.write(element[0].value);
    </script>
  </body>
</html>
```

5.4.4. Tìm thẻ HTML theo cú pháp của Selector CSS

Khi chọn các thẻ HTML theo class thường sẽ trả về hàng loạt các kết quả, như vậy đôi khi sẽ có những kết quả mà ta không mong đợi. Chính vì vậy javascript DOM có một phương thức kết hợp với CSS Selector để truy vấn có độ chính xác cao hơn.

Cú pháp sử dụng:

```
var element = document.querySelectorAll("selector.css");
```

Ví dụ 5.7.

```
<html>
  <body>
    <input type="text" value="thẻ không cần lấy" class="website"/>
    <div>
        <input type="text" value="Thẻ Cần Lấy" class="website"/>
        <input type="text" value="thẻ không cần lấy"/>
    </div>
  </body>
</html>
```

Câu hỏi đặt ra là làm thế nào có thể chọn đúng một thẻ input nằm trong thẻ div và có class="website"?

Trước tiên ta quay lại chút với CSS Selector đã nhé. Trong CSS để chọn thẻ input nằm trong thẻ div và có class="website" thì cú pháp là:

```
div input.website
```

Như vậy vấn đề được giải quyết như sau:

```
<html>
  <body>
    <input type="text" value="thẻ không cần lấy" class="website"/>
    <div>
            <input type="text" value="Thẻ Cần Lấy" class="website"/>
        <input type="text" value="thẻ không cần lấy"/>
    </div>
    <script language="javascript">
      // Lấy thẻ input
      var element = document.querySelectorAll("div input.website");

      // Lấy giá trị của thẻ input
      document.write('<h3>Nội dung thẻ cần lấy là: ' +
element[0].value + '</h3>');
    </script>
  </body>
</html>
```

5.5. DOM html

5.5.1. Giới thiệu

DOM HTML chuyên xử lý các vấn đề liên quan đến nội dung, thuộc tính của thẻ HTML.

5.5.2. Thay đổi và lấy nội dung bên trong thẻ HTML

Để lấy nội dung bên trong một thẻ HTML thì chúng ta sử dụng cú pháp như sau:

```
var html = document.getElementById("content").innerHTML
```

Và để thay đổi nội dung cho một thẻ HTML thì ta dùng cú pháp sau:

```
var html = document.getElementById("content").innerHTML = "<h1>Nội dung</h1>";
```

Ví dụ 5.8. Trong ví dụ này ta sẽ viết chương trình thay đổi nội dung HTML của một thẻ DIV và lấy nội dung bên trong của một thẻ DIV

```
<html>
  <body>
    <script language="javascript">
      // Hàm thiết lập nội dung cho thẻ div#content
      function set_content()
      {
        document.getElementById("content").innerHTML = "<h1>Nội dung đã được thay đổi</h1>";
      }

      // Hàm lấy nội dung cho thẻ div#content
      function get_content()
      {
        var html = document.getElementById("content").innerHTML;
        alert("Nội dung cần lấy là: " + html);
      }

    </script>
    <div id="content">Nội dung của thẻ DIV</div>
    <input type="button" value="Lấy nội dung" id="get_content" onclick="get_content()" />
    <input type="button" value="Thay đổi nội dung" id="set_content" onclick="set_content()" />
  </body>
</html>
```

5.5.3. Thay đổi và lấy giá trị thuộc tính thẻ HTML bằng Javascript

Để thay đổi giá trị của một thuộc tính HTML bất kì thì ta sử dụng cú pháp sau:

```
document.getElementById("element").attributeName = "new value";
```

Để lấy giá trị của một thuộc tính HTML ta sử dụng cú pháp sau:

```
var value = document.getElementById("element").attributeName;
```

Rất giống với cách thay đổi và lấy nội dung bên trong thẻ HTML. Từ đây có thể suy ra rằng trong Javascript để thiết lập (set) và lấy (get) thì sử dụng chung một cú pháp, chỉ khác nhau ở chỗ gán bằng và không có gán bằng.

Ví dụ 5.9. Xây dựng chương trình khi click vào một button thì chuyển nó thành textbox, và tiếp tục click vào textbox thì sẽ đổi thành button

```
<html>
    <body>
        <script language="javascript">
          function change()
          {
             // Lấy đối tượng
             var object = document.getElementById("object");

             // lấy thuộc tính type
             var type = object.type;

             // kiểm tra thuộc tính type và thay đổi
            if (type == "button"){

               object.type = 'text';
            }
            else{
                object.type = "button";
            }

          }
        </script>
        <input type="button" value="CLick me" onclick="change()"
id="object" />
    </body>
</html>
```

Như vậy là ta đã biết hai cách xử lý thuộc tính và nội dung của thẻ HTML bằng javascript rất thông dụng rồi đấy, thực tế bạn có thể sử dụng những mánh khóe để xử lý bài toán một cách tốt hơn. Ví dụ bạn cần bổ sung một đoạn text vào trong một thẻ HTML thì sẽ không có hàm nào bổ trợ, tuy nhiên bạn có thể sử dụng một mẹo nhỏ thế này:

```
var content_append = 'nội dung cần thêm vào';

// Lấy đối tượng
var object = document.getElementById("object");

// Lấy nội dung hiện tại
var content_current = object.innerHTML;

// Bổ sung nội dung vào đối tượng
object.innerHTML = content_current + content_current;
```

5.6. DOM css

5.6.1. Thay đổi CSS bằng Javascript

Style bản chất nó cũng là một thuộc tính của các thẻ HTML nhưng bạn không thể sử dụng DOM HTML để thiết lập hay xóa bỏ CSS được mà phải thông qua một đối tượng biệt khác đó là style.

Đối tượng style này sẽ chứa tất cả các thuộc tính của CSS và chúng ta sẽ dễ dàng thao tác với chúng bằng cú pháp riêng, và như thường lệ chúng ta có hai thao tác chính đó là thiết lập CSS và lấy giá trị CSS hiện tại.

Cú pháp thiết lập CSS bằng Javascript:

```
document.getElementById("object").style.cssName = 'something';
```

Cú pháp lấy giá trị CSS bằng Javascript:

```
var value = document.getElementById("object").style.cssName;
```

Trường hợp thuộc tính có dấu gạch ngang như: font-size, line-height, margin-bottom thì thuộc tính đó trong style sẽ có tên là fontSize, lineHeight, marginBottom ,nghĩa là sẽ bỏ đi dấu gạch ngang và viết hoa ký tự đầu tiên của chữ thứ hai.

```
document.getElementById("object").style.fontSize = 'something';
document.getElementById("object").style.lineHeight = 'something';
document.getElementById("object").style.marginBottom = 'something';
<br />
<br />
```

Lưu ý rằng có những thuộc tính nếu bạn chưa thiết lập CSS cho nó thì khi bạn lấy giá trị sẽ là một giá trị rỗng.

5.6.2. Các ví dụ thay đổi CSS bằng Javascript

Ví dụ 5.10. Viết một chương trình gồm 4 buttons và 1 thẻ div, khi click vào từng button thì sẽ thiết lập màu sắc, background, chiều cao, font size của thẻ div.

Để làm bài này thì chúng ta phải sử dụng sự kiện trong javascript đó là onclick, mỗi khi click vào mỗi button sẽ giải quyết một vấn đề của bài toán.

Các bước thực hiện:

- Tạo 4 functions thực hiện 4 nhiệm vụ như đề bài yêu cầu
- Gán mỗi function vào sự kiện onclick của mỗi button

```
<html>
    <body>
        <script language="javascript">
            function change_background()
            {
            document.getElementById("message").style.background='red';
```

```
            }

            function change_color()
            {
              document.getElementById("message").style.color = 'blue';
            }

            function change_height()
            {
              document.getElementById("message").style.height='500px';
            }

            function change_font_size()
            {
            document.getElementById("message").style.fontSize='500px';
            }
        </script>
        <div id="message">
          Chào mừng các bạn đến với freetuts.net
        </div>
        <input type="button" value="Change backgroud"
                             onclick="change_background()"/>
        <input type="button" value="Change color"
                             onclick="change_color()"/>
        <input type="button" value="Change height"
                             onclick="change_height()"/>
        <input type="button" value="Change fontsize"
                             onclick="change_font_size()"/>
    </body>
</html>
```

Ví dụ 5.11. Viết chương trình đăng nhập và validate thông tin username, password. Nếu người dùng không nhập username hoặc mật khẩu thì hiển thị thông báo message trong một thẻ div và chữ màu đỏ, ngược lại thì thông báo validate thành công và chữ màu xanh

Với bài này bạn cần phải sử dụng thêm DOM Element để lấy giá trị của các ô input, đồng thời sử dụng DOM HTML để thay đổi nội dung của message và sử dụng DOM CSS để thay đổi màu sắc.

```
<html>
    <body>
        <script language="javascript">
          function validate()
          {
            // Lấy giá trị input
             var username = document.getElementById("username").value;
             var password = document.getElementById("password").value;
```

```
            // Lấy đối tượng message
            var message = document.getElementById("message");

            // Validate
            if (username == "" || password == ""){
                message.innerHTML = "Bạn chưa nhập đầy đủ thông tin!";
                message.style.color = "red";
            }

            else{
                message.innerHTML="Chúc mừng, validate thành công!";
                message.style.color = "blue";
            }

          }
        </script>
        Username: <input type="text" value="" id="username" /> <br/>
        Username: <input type="password" value="" id="password" />
        <br/>
        <div id="message"></div>
        <input type="button" value="Login" onclick="validate()"/>
    </body>
</html>
```

Tóm lại, những bài toán dạng như thế này thì chúng ta hay gọi là DHTML, nghĩa là HTML động có các hiệu ứng thay đổi giao diện bằng cách sử dụng Javascript kết hợp với giá trị CSS của HTML. Tất cả các thuộc tính của CSS được lưu trong đối tượng style (có thể gọi là thuộc tính style) của đối tượng HTML nên bạn tránh ghi nhầm bỏ đi chữ style nhé.

5.7. DOM animation

5.7.1. Giới thiệu

Animations dịch sang tiếng việt có nghĩa là chuyển động, đúng với cái tên của nó, các hiệu ứng Animations sẽ khiến cho các phần tử thực hiện các chuyển động. Việc thêm các hiệu ứng Animations sẽ khiến website của bạn trở nên chuyên nghiệp hơn rất nhiều. Ví dụ như khi bạn thêm một sản phẩm vào giỏ hàng, việc hình ảnh sản phẩm di chuyển đến giỏ hàng sẽ sinh động hơn rất nhiều so với việc chỉ hiển thị thông báo thông thường.

Việc thêm các animation vào giao diện web làm cho web pages và các apps trông có vẻ tương tác nhiều hơn với người dùng. Một thanh side menu với khả năng trượt vào/ra mượt mà sẽ mang đến trải nghiệm người dùng tốt hơn nhiều trường hợp cũng là thanh side menu đó, chỉ xuất hiện rồi biến mất.

Từ trước đến nay, việc tạo web animation thường được thực hiện bằng CSS transition, CSS keyframe hoặc một thư viện bên ngoài như Animate.css, Velocity.js. Những API mới của native JavaScript có thể tạo animation cho các phần tử HTML gói gọn trong file .js

5.7.2. Tạo animation

Tạo file index.html và đặt đoạn mã HTML dưới đây vào trong thẻ body.

```
<div class="container">
  <h2>Freetuts.net hướng dẫn tạo hiệu ứng Animation</h2>
  <button id="btn" onclick="myMove()">Xem kết quả</button>
  <div id ="myContainer">
    <div id ="myAnimation"></div>
  </div>
</div>
```

Giờ ta đã có các thành phần nền rồi, mình tiếp tục sử dụng CSS để định dạng cho các thành phần dễ nhìn hơn, các bạn thêm đoạn mã CSS dưới đây vào bên trong thẻ style nhé:

```
.container {
  width: 500px;
  margin: auto;
  text-align: center;
}
#myContainer {
  width: 400px;
  height: 400px;
  margin: auto;
  position: relative;
  background: #222222;
}
#myAnimation {
  width: 50px;
  height: 50px;
  position: absolute;
  background-color: #FF0000;
}
#btn {
  width: 150px;
  height: 40px;
  margin: 20px;
}
```

Đoạn mã này rất cơ bản nên mình cũng không giải thích nữa. Giờ ta sẽ chuyển đến bước tiếp theo.

Xử lí với Javascript:

Sau khi đã hoàn thành phần giao diện, việc tiếp theo của chúng ta là thêm các xử lí javascript để các danh sách có thể hoạt động, các bạn đặt đoạn mã script dưới đây vào sau thẻ body.

```
<script type="text/javascript">
  function myMove() {
```

```
    var btn = document.getElementById("btn");
    btn.style.display = "none";
    var elem = document.getElementById("myAnimation");
    var pos = 0;
    var id = setInterval(frame, 10);
    function frame() {
      if (pos == 350) {
        clearInterval(id);
      } else {
        pos++;
        elem.style.top = pos + 'px';
        elem.style.left = pos + 'px';
      }
    }
  }
</script>
```

Thuật toán sử dụng ở đây khá đơn giản:

- Đầu tiên mình sẽ lấy thành phần chuyển động elem, tiếp đến là khai báo biến pos = 0 đại diện cho vị trí hiện tại và biến id = setInterval(frame, 10); tức lại cứ sau 10 mili giây sẽ gọi hàm frame 1 lần.
- Tiếp đến mình sẽ định nghĩa hàm frame, Nếu vị trí đã bằng 350 thì dừng việc gọi hàm frame, ngược lại thì tăng biến pos lên 1 đơn vị và đặt vị trí cho elem theo pos. Như vậy khi chưa đến vị trí 350, thì mỗi 10 mili giây phần elem sẽ di chuyển 1px.

Xong rồi giờ các bạn chạy file index.html để xem thành quả nhé.

5.8. DOM events

5.8.1. Sự kiện trong javascript là gì?

Về định nghĩa dưới dạng nghe hiểu thì sự kiện là một hành động nào đó tác động lên đối tượng HTML mà ta có thể bắt được sự kiện này và thực hiện những hành động nào đó.

Mỗi sự kiện chúng ta có thể có nhiều hành động. Ví dụ bạn có một sự kiện là tết âm lịch thì bạn sẽ có những hành động trong sự kiện đó là mua đồ mới, dọn dẹp nhà cửa, mua bao lì xì, ..

Xét về mặt thực tế thì ta có dụ thế này: Giả sử bạn xây dựng một form đăng ký tài khoản và bạn muốn bắt sự kiện khi người dùng CLICK vào button đăng ký thì hiện những hành động như validate dữ liệu, thông báo nếu người dùng nhập nội dung không đúng, .. Như vậy bạn cần nhớ rằng mỗi sự kiện chúng ta có thể thực hiện nhiều hành động khác nhau và bao nhiêu hành động thì phụ thuộc vào từng chức năng cụ thể.

5.8.2. Các sự kiện (Events) trong javascrpt

Có rất nhiều sự kiện và mỗi đối tượng HTML thì lại có những sự kiện khác nhau nên mình chỉ liệt kê được những sự kiện thông dụng hay sử dụng khi các bạn làm việc với javascript để lập trình phía client.

Bảng các sự kiện thông dụng trong javascript.

STT	Event Name	Description
1	onclick	Xảy ra khi click vào thẻ HTML
2	ondbclick	Xảy ra khi double click vào thẻ HTML
3	onchange	Xảy ra khi giá trị (value) của thẻ HTML đổi. Thường dùng trong các đối thẻ form input
4	onmouseover	Xảy ra khi con trỏ chuột bắt đầu đi vào thẻ HTML
5	onmouseout	Xảy ra khi con trỏ chuột bắt đầu rời khỏi thẻ HTML
6	onmouseenter	Tương tự như onmouseover
7	onmouseleave	Tương tự như onmouseout
8	onmousemove	Xảy ra khi con chuột di chuyển bên trong thẻ HTML
9	onkeydown	Xảy ra khi gõ một phím bất kì vào ô input
10	onload	Sảy ra khi thẻ HTML bắt đầu chạy, nó giống như hàm khởi tạo trong lập trình hướng đối tượng vậy đó.
11	onkeyup	Xảy ra khi bạn gõ phím nhưng lúc bạn nhã phím ra sẽ được kích hoạt
12	onkeypress	Xảy ra khi bạn nhấn môt phím vào ô input
14	onblur	Xảy ra khi con trỏ chuột rời khỏi ô input
15	oncopy	Xảy ra khi bạn copy nội dung của thẻ
16	oncut	Xảy ra khi bạn cắt nội dung của thẻ
17	onpaste	Xảy ra khi bạn dán nội dung vào thẻ

5.8.3. Các ví dụ về xử lý sự kiện trong javascript

Ví dụ 5.12. Viết chương trình gồm một ô input và một thẻ div dùng để hiển thị nội dung (giá trị của ô input) khi người dùng gõ vào ô input

Vì đề bài yêu cầu khi nhập dữ liệu vào ô input thì hiển thị nội dung bên trong thẻ DIV nên ta có thể sử dụng sự kiện onkeyup. Thứ hai nữa là chúng ta sẽ sử dụng các hàm DOM Element để truy xuất các đối tượng HTML.

```
<html>
  <body>
    <script language="javascript">
      // Hàm show kết quả
      function show_result()
      {
        // Lấy hai thẻ HTML
        var input = document.getElementById("message");
        var div = document.getElementById("result");

        // Gán nội dung ô input vào thẻ div
        div.innerHTML = input.value;
      }
    </script>
    <input type="text" id="message" value="" onkeyup="show_result()"/>
    <div id="result"></div>
  </body>
</html>
```

Nếu như bài này bạn sử dụng sự kiện onkeypress hoặc onkeydown thì sẽ có kết quả sai, lý do là những sự kiện này xảy ra khi bạn nhấn phím xuống nên nó sẽ lấy giá trị chưa được cập nhật. Còn sự kiện onkeyup xảy ra khi bạn nhả phím ra nên nó sẽ lấy được giá trị mới.

Ví dụ 5.13. Viết chương trình khi người dùng copy nội dung của website thì thông báo là bạn đã copy thành công

Trong danh sách các sự kiện trên thì có sự kiện oncopy nên ta sẽ sử dụng nó để giải bài này.

```
<html>
  <body>
    <script language="javascript">
      // Hàm show kết quả
      function show_message()
      {
        alert("Bạn đã copy thành công");
      }
    </script>
    <h3>Hãy copy dòng chữ dưới đây:</h3>
    <div oncopy="show_message()">Chào mừng các bạn đến với website
freetuts.net</div>
  </body>
</html>
```

Ví dụ 5.14. Viết chương trình tính tổng của hai số nhập vào (tính tự động)

Bài này ta phải tạo 3 ô input và gán sự kiện onkeyup cho 2 ô input đầu tiên, trong sự kiện này sẽ thực hiện tính tổng của hai ô và in kết quả vào ô input thứ 3.

```
<html>
  <body>
    <script language="javascript">
      // Hàm tính kết quả
      function tinh()
      {
        // Lấy 3 ô input
        var a = document.getElementById("a");
        var b = document.getElementById("b");
        var result = document.getElementById("result");

        // Tính tổng hai ô đầu tiên
        var tong = parseInt(a.value) + parseInt(b.value);

        // Gán giá trị vào ô thứ ba
        // Phải kiểm tra tổng hai số này có bị lỗi hay không
        if (!isNaN(tong)){
          result.value = tong;
        }
      }
    </script>
    a: <input type="text" id="a" value="" onkeyup="tinh()"/>
    b: <input type="text" id="b" value="" onkeyup="tinh()"/>
    Kết quả: <input type="text" id="result" value=""/>
  </body>
</html>
```

5.9. DOM Node

5.9.1. DOM Node - document.createElement()

Khi sử dụng DOM Element để truy vấn tới một đối tượng HTML nào đó thì kết quả nó sẽ trả về một object và object đó ta gọi là DOM Nodes. Chẳng hạn ta có khai báo:

```
var node = document.getElementById("some-id");
```

Với cách này bắt buộc phải tồn tại một đối tượng HTML đang hiển thị trên website thì mới khởi tạo thành công. Giả sử nếu bạn muốn tạo một Node mới hoàn toàn v không liên quan tới những thẻ HTML đang hiển thị trên website thì làm thế nào? Rất đơn giản chúng ta sẽ sử dụng phương thức document.createElement() với tham số truyền vào là tên của thẻ HTML cần tạo.

```
var p = document.createElement("p");
```

Sau khi khởi tạo xong bạn hoàn toàn có thể sử dụng các phương thức, thuộc tính của DOM HTML, DOM CSS.

```
p.innerHTML = "Học DOM Nodes tại freetuts.net"
```

Để thêm Node này vào trang web thì chúng ta sử dụng phương thức appendChild (sẽ học ở bên dưới). Giả sử tôi thêm vào thẻ body thì làm như sau:

```
document.getElementsByTagName('body')[0].appendChild(p);
```

Và đây là toàn bộ code trong ví dụ 5.15.

Ví dụ 5.15.

```
<html>
    <body>
        <script language="javascript">

          // Tạo mới một thẻ p
          var p = document.createElement("p");

          // Thêm nội dung HTML vào thẻ p
          p.innerHTML = "Học DOM Nodes tại freetuts.net"

          // Đưa thẻ P vào trong thẻ body
          document.getElementsByTagName('body')[0].appendChild(p);

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

5.9.2. DOM Node - document.createTextNode()

Text node là một node đặc biệt, nó không phải là một thẻ HTML thông thường mà chỉ là một chuỗi (string) nên thông thường chúng ta sử dụng nó để thay thế cách gán thông thường node.innerHTML.

Ví dụ 5.16.

```
<html>
    <body>
        <script language="javascript">

          // Tạo mới một thẻ p
          var p = document.createElement("p");

          // Tạo text node
          var text = document.createTextNode("Học DOM Nodes tại
freetuts.net");

          // Thêm nội dung HTML vào thẻ p
          p.appendChild(text);
```

```
            // Đưa thẻ P vào trong thẻ body
            document.getElementsByTagName('body')[0].appendChild(p);

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

Trong ví dụ 5.16 thay vì sử dụng cách thông thường như ví dụ ở phần 1 thì mình đã thay thế bằng cách sử dụng text node.

5.9.3. DOM Node - các phương thức khác

5.9.3.1. Phương thức appendChild()

Dùng để thêm (bổ sung) vào vị trí cuối cùng của đối tượng một thẻ HTML nào đó.

Ví dụ 5.17.

```
<html>
    <body>
        <div id="TOP">
            Xin chào các bạn!
        </div>

        <input type="button" value="Append" id="btn-append"/>

        <script language="javascript">
          // Lấy button
          var button = document.getElementById("btn-append");

          // Thêm sự kiện click cho button
          button.addEventListener("click", function(){
            // Tạo mới một thẻ h1
            var h1 = document.createElement("h1");

            // Thêm nội dung HTML vào thẻ h1
            h1.innerHTML = "Học DOM Nodes tại freetuts.net"

            // Đưa thẻ h1 vào trong thẻ div có id=TOP
            document.getElementById('TOP').appendChild(h1);
          });
        </script>
    </body>
</html>
```

5.9.3.2. Phương thức insertBefore()

Được dùng để thêm một Node vào đằng trước một node con nào đó. Phương thức này có hai tham số truyền vào insertBefore(node_insert, node_child), trong đó:

- node_insert là node bạn muốn thêm vào
- node_child là node con mà bạn muốn thêm vào đằng trước nó.

Ví dụ 5.18.

```
<html>
    <body>
        <div id="content">
            <h5>Xin chào các bạn</h5>
            <h5>chúc vui vẻ khi học bài</h5>
        </div>

        <input type="button" value="View" id="btn-view"/>

        <script language="javascript">
          // Lấy button
          var button = document.getElementById("btn-view");

          // Thêm sự kiện click cho button
          button.addEventListener("click", function(){
            // Tạo mới một thẻ h1
            var h1 = document.createElement("h1");

            // Thêm nội dung HTML vào thẻ h1
            h1.innerHTML = "Chào mừng bạn đến với freetuts.net"

            // Lấy thẻ ngoài cùng
            var element = document.getElementById("content");

            // Lấy thẻ cần isertBefore
            var element_child =
document.getElementsByTagName("h5")[1];

            // Insert Before
            element.insertBefore(h1, element_child);
          });
        </script>
    </body>
</html>
```

5.9.3.3. Phương thức removeChild()

Được dùng để xóa một node con ra khỏi node hiện tại.

Ví dụ 5.19.

```
<html>
    <body>
        <div id="content">
```

```
                <h5>chúc vui vẻ khi học bài</h5>
            </div>

            <input type="button" value="Remove" id="btn-remove"/>

            <script language="javascript">
              // Lấy button
              var button = document.getElementById("btn-remove");

              // Thêm sự kiện click cho button
              button.addEventListener("click", function(){

                // Lấy thẻ cần remove
                var need_remove = document.getElementsByTagName("h5")[0];

                // Remove

document.getElementById("content").removeChild(need_remove);
              });
            </script>
        </body>
    </html>
```

5.9.3.4. Phương thức replaceChild()

Dùng để replace (thay thế) một node con nào đó bằng một node khác mới hoàn toàn.

Ví dụ 5.20.

```
    <html>
        <body>
            <div id="content">
                <h5>chúc vui vẻ khi học bài</h5>
            </div>

            <input type="button" value="Replace" id="btn-replace"/>

            <script language="javascript">
              // Lấy button
              var button = document.getElementById("btn-replace");

              // Thêm sự kiện click cho button
              button.addEventListener("click", function(){

                // Tạo mới một thẻ
                var p = document.createElement("h1");
                p.innerHTML = "Xin chào!";

                // Lấy thẻ cần replace
```

```
                var replace = document.getElementsByTagName("h5")[0];

                // Replace
                document.getElementById("content").replaceChild(p,
replace);
            });
        </script>
    </body>
</html>
```

Tóm lại, việc sử dụng thành thạo các phương thức xử lý DOM Node rất quan trọng khi bạn làm việc với các thẻ HTML bằng Javascript. Nếu bạn sử dụng một JS Library như jQuery thì điều này khá đơn giản bởi vì nó đã viết ra những hàm cho chúng ta sử dụng sẵn nên hầu hết các lập trình viên đều không quan tâm đến những kiến thức thuần javascript như thế này.

Ngoài các phương thức mình liệt kê trên thì còn khá nhiều phương thức nên bạn có thể sử dụng Google để tìm kiếm trong quá trình học và làm việc của mình. Bản thân mình cũng vậy thôi không tài nào nhớ hết được nhung phải biết từ khóa để tìm kiếm.

5.10. DOM collections

5.10.1. Đối tượng HTMLCollection

Phương thức getElementsByTagName() trả về một đối tượng HTMLCollection

Một đối tượng HTMLCollection là một danh sách mảng gồm các thành phần HTML.

Ví dụ 5.21 sẽ chọn tất cả các thành phần <p> trong tài liệu:

Ví dụ 5.21.

```
var x = document.getElementsByTagName("p");
```

Các thành phần trong collection được truy cập bởi chỉ số. Để truy cập đến thành phần<p> thứ 2 ta viết:

```
y = x[1];
```

Chú ý: Chỉ mục bắt đầu tại 0.

5.10.2. Số phần tử trong HTMLCollection

Thuộc tính length xác định số phần tử trong HTMLCollection:

Ví dụ 5.22.

```
var myCollection = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = myCollection.length;
```

Giải thích:

- Tạo một collection cho tất cả các thành phần <p>

- Hiển thị số phần tử có trong collection.

Thuộc tính length rất hữu dụng khi ta muốn lặp qua tất cả các phần tử trong collection.

Ví dụ 5.23. Thay đổi màu nền của tất cả các thành phần <p>

```
var myCollection = document.getElementsByTagName("p");
var i;
for (i = 0; i < myCollection.length; i++) {
  myCollection[i].style.color = "red";
}
```