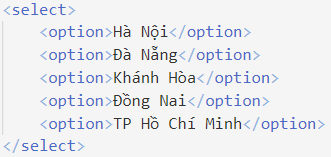
BÀI 1: NGÔN NGỮ ĐÁNH DẤU HTML

Thẻ <select />: Tạo 1 danh sách để cho chúng ta lựa chọn. Thông thường nó kết hợp với thẻ <option /> để tạo ra danh sách đó. Ví dụ:

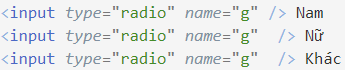


Kết quả:



Thẻ <input /> kết hợp với type=“checkbox” sẽ tạo ra 1 ô cho chúng ta check, lưu ý rằng check này có thể check được nhiều ô dù **name** có giống nhau.

Thẻ <input /> kết hợp với type=“radio” sẽ tạo ra 1 ô cho chúng ta tích vào, nếu trong 1 danh sách có nhiều ô nhưng các ô này có **name** giống nhau thì chỉ tích được vào 1 ô duy nhất. Nếu ta muốn khi mới vào trang web mà nó tự động tích vào ô nào đó thì ta sử dụng từ khóa **checked**. Ví dụ:



Kết quả:



Thẻ <sup />: Viết đoạn văn bên trong lên trên. Ví dụ:



Kết quả:



Thẻ <sub />: Viết đoạn văn bên trong thẻ xuống dưới. Ví dụ:



Kết quả:



**:first-child**: Lớp giả CSS đại diện cho phần tử đầu tiên trong số các phần tử trong 1 khu vực nhất định. Ví dụ:



Kết quả:



**:last-child**: Chọn phần tử cuối cùng. Ngược lại với **:first-child**.

**:nth-child(n)**: Nếu n là số thì chọn phần tử tương ứng với số đó. Nếu n là chuỗi thì chọn tất cả các phần tử.

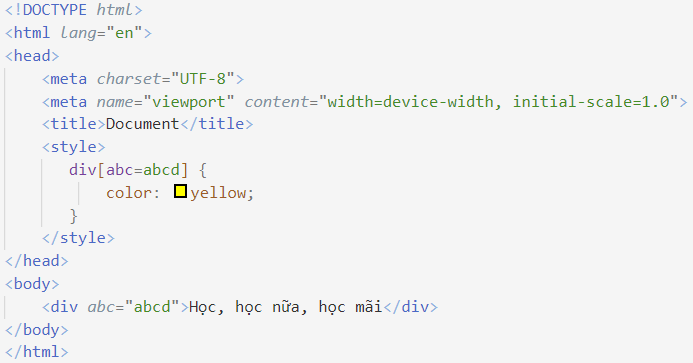
**:nth-child(2n)** với **2n** là chuỗi không phải là số: Chọn tất cả các phần tử chẵn.

**:nth-child(even)**: Giống như trên là chọn tất cả các phần tử chẵn.

**:nth-child(2n+1)**, **:nth-child(odd)**: Chọn tất cả các phần tử lẻ (**2n+1** là chuỗi ko phải là số).

Định dạng CSS bằng cách gọi thuộc tính. Cú pháp:

**tag[prop=value]** hoặc **tag[prop=“value”]**



Kết quả:



Ta có thể thấy prop có thể là bất cứ giá trị nào ta đặt và không nhất thiết phải các prop được cho như **name**, **id**, **type**.

Sử dụng dấu **+** để chọn sang thành phần tiếp theo. Cú pháp:

Selector + tag{

//code

}

Ví dụ:



Như ví dụ trên **div[abc=abcd]** là lựa chọn thành phần chữ “Học, học nữa, học mãi” nhưng **div[abc=abcd] + p** lại thành lựa chọn thành phần tiếp theo của thành phần đó là thành phần “Học là vô tận”.

Tuy nhiên nếu thành phần “Học là vô tận” mà bị ngăn cách với thành phần “Học, học nữa, học mãi” bởi 1 thẻ nào đó thì gọi **div[abc=abcd] + p** không chọn được thành phần “Học là vô tận”.

Chọn tất cả các thành phần phía sau 1 thẻ. Cú pháp:

Selector ~ \* {

//code

}



Kết quả:



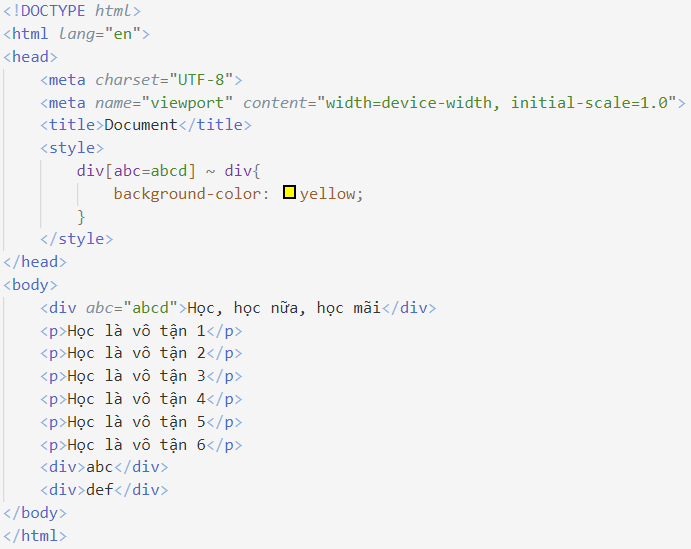
Chọn tất cả thành phần thẻ cùng loại sau 1 thẻ. Cú pháp:

Selector ~ tag {

//code

}

Ví dụ:



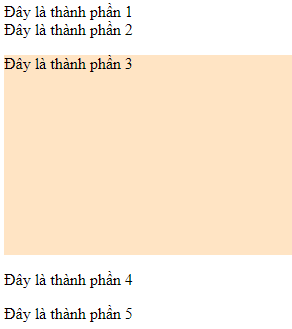
Kết quả:



Trường hợp ta chọn thành phần thẻ sau theo công thức trên và chọn theo số thứ tự thì số thứ tự là số tính từ trên đầu xuống chứ không phải tính từ thẻ **Selector**. Ví dụ:



Theo ví dụ trên thẻ <span> nằm ở vị trí thứ 3. Chọn thẻ thứ 3 là chọn thẻ thứ 3 tính từ trên đầu xuống chứ không phải tính từ thành phần <span>. Kết quả:



Một thành phần có class gồm 2 tên khác nhau thì để định dạng thành phần đó ta có thể gọi tên 1 trong 2 tên đó hoặc định dạng cả 2 không viết dấu cách:

Cách 1:



Cách 2:

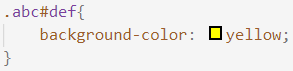


Cách 3:

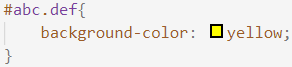


Lưu ý rằng với cách 3 thì chỉ có thành phần có class=“abc def” mới được hưởng tính chất định dạng còn thành phần có class=“abc” hoặc class=“def” thì không được hưởng.

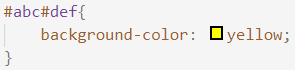
Với cách 3 này ta có thể mở rộng thêm là với cách định dạng:



Hoặc:



Hoặc:



Có nghĩa là chọn 1 thành phần có cả 2 thuộc tính trên.

Tóm gọn lại đối với cách định dạng CSS ta có các công thức chung như sau:

* Công thức 1:

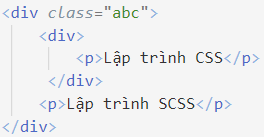
Selector1 selector2 {

//code

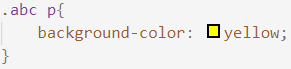
}

Ý nghĩa: Định dạng cho thành phần selector2 nằm bên trong thành phần selector1.

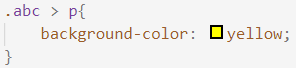
Với trường hợp sau:



Nếu ta định dạng:



Thì cả 2 thành phần **p** trong ví dụ trên được định dạng do cả 2 thẻ **p** đều nằm trong class **abc**. Để chọn 1 thành phần p nằm bên trong class **abc** mà ko trong thẻ nào trung gian (thành phần con trực tiếp) thì ta sử dụng định dạng sau:



* Công thức 2:

Selector1 + selector2 {

//code

}

Ý nghĩa: Định dạng cho thành phần selector2 nằm ngay sau thành phần selector1.

* Công thức 3:

Selector1 ~ selector2 {

//code

}

Ý nghĩa: Định dạng cho tất cả các thành phần selector2 nằm sau thành phần selector1.

* Công thức 4:

Selector1Selector2Selector3{

//code

}

Ý nghĩa: Định dạng cho tất cả các thành phần mà có cả 3 Selector1, Selector2, Selector3.

Lưu ý đối với cách định dạng này thì giữa các Selector phải có dấu ngăn cách do đó vị trí Selector2 và selector3 phải là id hoặc class vì nó có dấu # và dấu “.” Còn vị trí Selector1 có thể không phải là class hoặc id mà là thẻ hoặc thuộc tính cũng được.

* Công thức 5:

Selector1 > Selector2 {

//code

}

Ý nghĩa: Định dạng cho tất cả các thành phần Selector2 là thành phần con trực tiếp của thành phần Selector1.

Thuộc tính **box-sizing**: Quy định cách mà trình duyệt tính toán kích thước của element. Các giá trị của nó là:

**content-box**: Giá trị mặc định, nghĩa là giá trị width và height chỉ áp dụng cho khu vực nội dung bên trong, không bao gồm padding, border và margin.

**border-box**: Khi thiết lập giá trị này, thì width và height sẽ bao gồm cho cả phần nội dung, padding và border nhưng không bao gồm margin.

**padding-box**: Với giá trị này thì width và height chỉ bao gồm cho phần nội dung và padding, không bao gồm border và margin.

Lưu ý rằng **padding-box** chỉ có tác dụng với trình duyệt Firefox.

Có 2 loại element trong HTML:

* Block element:

+ Là element có chiều rộng bằng 100% chiều rộng của màn hình.

+ Nằm trên 1 hàng riêng và nó dùng để chứa các block element hoặc inline element khác.

+ Có thể sử dụng các thuộc tính như width, height, margin-top,…

+ Các loại thẻ block element có thể chuyển thành thẻ inline element nhờ thuộc tính **display: inline;**

+ Các loại thẻ block element phổ biến như: div, p, h, dl, dt, dd, ol, li, ul, nav, header, section, sidebar, footer, …

* Inline element:

+ Chiều rộng và dài chỉ bằng chiều rộng và dài của nội dung bên trong.

+ Chỉ có thể chứa các inline element khác.

+ Không thể sử dụng các thuộc tính như width, height, margin-top,…

+ Các loại thẻ inline element như: span, a,…

+ Các loại thẻ inline element có thể chuyển thành thẻ block element nhờ thuộc tính **display: block;**

Với thẻ inline element tuy không sử dụng được **margin-top** và **bottom** nhưng có thể sử dụng được **margin-left** và **margin-right** và **padding**.

Riêng đối với thẻ <img> thì tùy từng trình duyệt và tùy từng điều kiện nó có thể là block element hoặc là inline element.

1 số thẻ inline-block element nổi bật như: button, input,…

Đối với thuộc tính **display: inline-block;** thì nó hiển thị trên 1 dòng như inline nhưng có thể đặt **width** và **height** và **margin-top**.

Thẻ <a /> không nên nằm ngoài 1 thẻ inline-block vì điều đó khiến cho các thuộc tính như **text-decoration** không thể định dạng ở thẻ <a /> được. Những lúc cần 1 khu vực có thể click vào mà cần dùng thẻ <a /> thì ta nên chuyển a về dạng inline-block để có thể dùng được các thuộc tính khác. Tuy nhiên để có thể sử dụng được thuộc tính trên thì ta có thể trỏ đến thành phần bên trong được chuyển thành dạng inline-block đó và định dạng.

Trong HTML nếu ta viết nhiều dấu cách thì cũng chỉ được tính 1 dấu cách cho dù khi ta viết 1 đoạn văn bản sau:



Thì kết quả hiển thị vẫn là:



Và khi chúng ta xuống dòng cũng chỉ được tính là 1 dấu cách.

Với ví dụ sau cũng chỉ được tính 1 dấu cách:



Mặc dù xuống dòng là 1 dấu cách và trước chữ “Trình” là 5 6 dấu cách nhưng vẫn chỉ hiển thị 1 dấu cách:



Thuộc tính **float**: Đẩy phần tử sang bên trái hoặc bên phải tùy ý. Nhờ thuộc tính này các block element hoặc các inline element được chuyển thành các inline-block element.

Nhờ thuộc tính float mà các khối có thể ở cùng 1 hàng. Ví dụ:



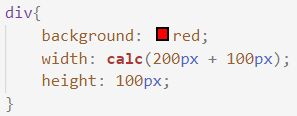
Phần sidebar, phần nội dung và phần quảng cáo có thể ở cùng 1 hàng là do quy định của thuộc tính float.

Thuộc tính float có các giá trị sau:

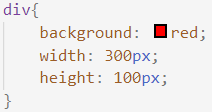
* left: Cố định phần tử về bên trái.
* right: Cố định phần tử về bên phải.
* none: Trạng thái mặc định (trạng thái bình thường)
* inherit: Phần tử kế thừa từ float cha.

Với giá trị left thì phần tử đó được đẩy sang bên trái. Khi đó các phần tử khác có thể được viết sang bên phải hoặc viết xuống dưới phần tử đó.

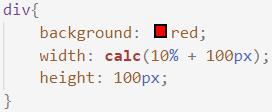
Hàm **calc()** trong CSS: Dùng để tính toán các giá trị CSS ngay trong thuộc tính CSS. Ví dụ:



Nó sẽ tương đương với:



Đặc biệt với hàm này ta có thể kết hợp giữa các đơn vị khác nhau:



Với 10% là 1/10 chiều rộng màn hình.

Hàm **calc()** phù hợp với các loại bài khó có thể đặt giá trị tĩnh cho thành phần. Ví dụ để đặt 1 block cách chính giữa màn hình 10px về bên trái thì ta dùng câu lệnh: **margin-left: calc(50% - 10px);**

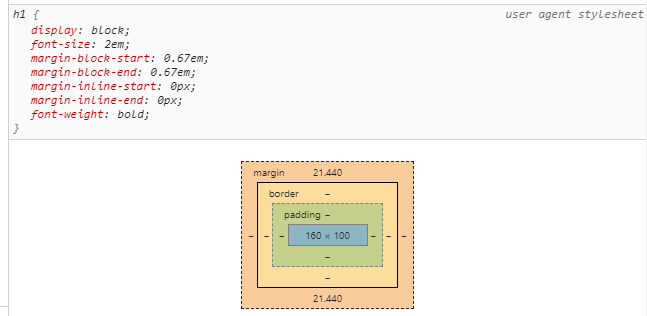
Trường hợp muốn thực hiện phép nhân hoặc phép chia trong giá trị thuộc tính bằng **calc()** thì ta làm như sau:

Phép tính bằng **calc()**:



* **margin-top = margin-block-start**
* **margin-bottom = margin-block-end**
* **margin-right = margin-inline-end**
* **margin-left = margin-inline-start**

Kiểm tra phần tử cho hình ảnh như sau:

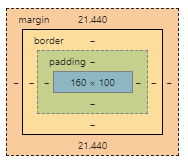


Ta thấy nếu cài đặt margin cho phần tử thì các giá trị trên kia sẽ không còn phản ánh chính xác nữa mà ta phải xem trong khối block ở hình trên.

Ban đầu khi chưa cài đặt margin thì nó sẽ có giá trị mặc định nhưng sau khi cài đặt thì nó sẽ được thay thế bằng giá trị đã cài.

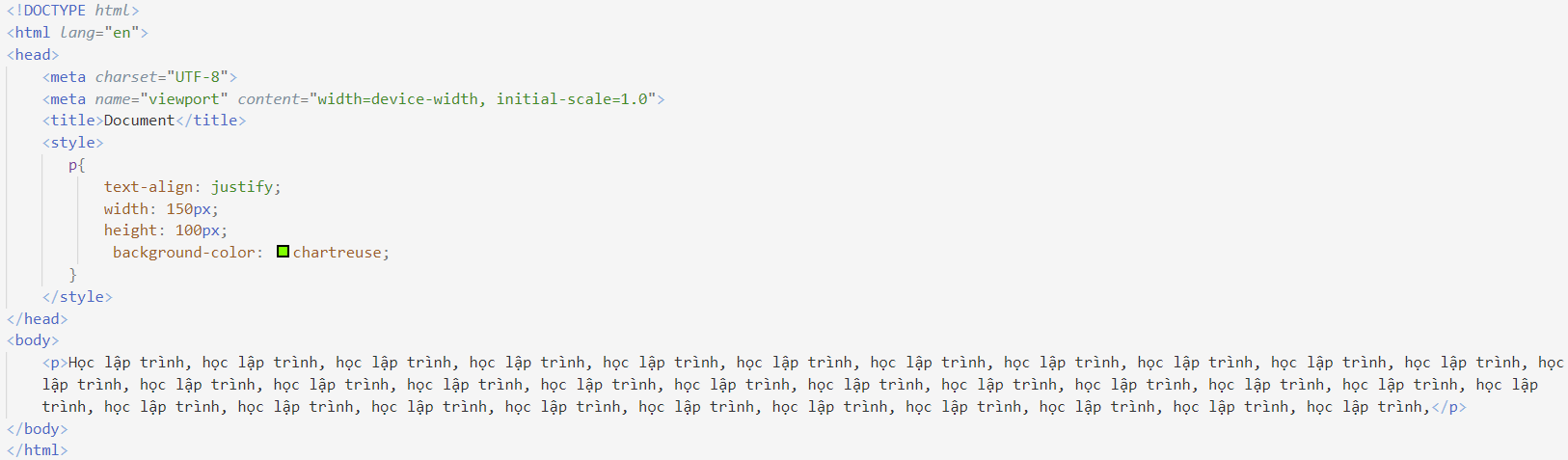
Với 1 phần tử có block cho trước đã được định dạng **float: left**. Tất cả các phần tử khác, phần content không được chồng lên phần block của phần tử cho trước đó. Tuy nhiên phần background của phần tử khác này có thể chồng lên phần block của phần tử cho trước (kể cả đó là phần content).

Phần block của 1 phần tử là phần tính cả margin của phần tử đó. Ví dụ:

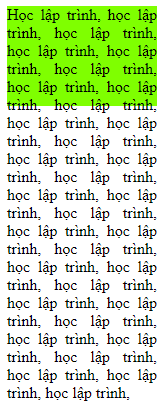


Đối với các phần tử không được định dạng **float** thì block của phần tử này không được xếp chồng lên phần border của phần tử kia (các margin có thể xếp chồng lên nhau).

Khi 1 phần tử block cùng được định dạng width và height. Nếu nội dung viết trong block đó mà ko đủ thì nó sẽ viết xuống dưới và lấy width làm chuẩn. Ví dụ:



Kết quả:



Chương trình ưu tiên đảm bảo content có width bằng width cài đặt còn height của content khác height cài đặt.

Độ rộng màn hình khoảng 1904px đến 1920px.

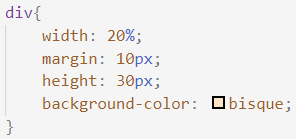
Trường hợp 1 block **float: left** và 1 block **float: right** thì các block không được chồng lên nhau (như 2 block không được định dạng float) và block **float: right** được xếp sang phải và lấy cạnh phải làm chuẩn để căn chỉnh. Nếu width của block phải + width block trái lớn hơn cạnh màn hình thì cả block đó bị đẩy xuống dưới.

1 block element có 3 cái đặc trưng là:

* Phần 1 gồm nội dung
* Phần 2 gồm phần 1, background và border.
* Phần 3 gồm phần 2 và các margin.

Ta có ví dụ cụ thể:

CSS viết:



Hiển thị:



Ta thấy phần 1 là phần nội dung “lập trình”.

Phần 2 là phần được định dạng màu sắc **bisque**. Phần này ta có thể định dạng được bằng các thuộc tính height, width, background và border của thẻ div.

Phần 3 là phần 2 cộng thêm các margin như hình dưới:



Thuộc tính **clear**: Dùng để xóa sự ảnh hưởng của thuộc tính float.

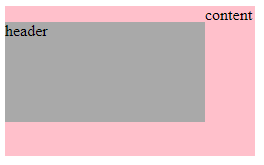
Thuộc tính này có 4 giá trị là **left, right, both** và **none**.

* left: clear sự ảnh hưởng của float left.
* right: clear sự ảnh hưởng của float right.
* both: clear sự ảnh hưởng của cả float left và right.
* none: Không clear sự ảnh hưởng nào.

Ta có thể hiểu tương đối rằng khi 1 thành phần đi trước được float thì các thành phần sau nếu không định nghĩa float thì sẽ bị ảnh hưởng bởi thành phần được float đằng trước. Nó cũng tương tự như quy tắc căn chỉnh text trong word, thành phần đi trước được căn chỉnh **center** thì các thành phần sau đó cũng căn chỉnh **center**. Khi thành phần đi sau được clear float thì các thành phần sau đó cũng được clear float. Từ thành phần được clear nó coi thành phần được float như là thành phần mặc định của nó. Ví dụ khi thành phần content chưa được clear float thì nó hiển thị phần nội dung sang bên trái của thành phần được float:



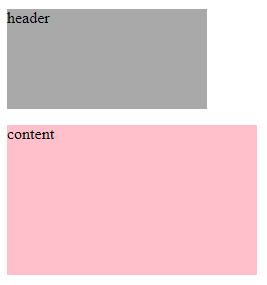
Kết quả:



Khi clear float cho thành phần content:



Kết quả:



Ta thấy khi clear thành phần content thì thành phần content coi thành phần header như thành phần chưa được float. Vì vậy nó hiển thị phần content xuống dưới, các thành phần sau thành phần content cũng hiển thị tương tự.

CSS Gradients: Có 2 kiểu là **linear-gradient** và **radial-gradient**.

**linear-gradient** là kiểu trải màu. Có 4 cách viết như sau:

* Từ trên xuống dưới (linear-gradient(to bottom, color1, color2,..))
* Từ dưới lên trên (linear-gradient(to top, color1, color2,..))
* Từ trái qua phải (linear-gradient(to right, color1, color2,..))
* Từ phải qua trái (linear-gradient(to left, color1, color2,..))

Ngoài ra ta có thể tùy chỉnh theo góc mà ta thay đổi như sau:

**linear-gradient(deg, color1, color2);**

Nếu không nhập góc độ thì mặc định là 180o.

Trong đó deg là góc giữa cạnh đáy và cạnh của màu bắt đầu và color1 là màu bắt đầu còn color2 là màu kết thúc. Cho ví dụ sau:



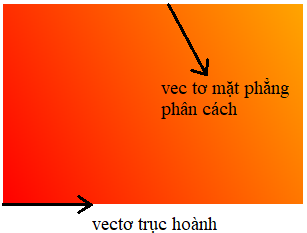
Kết quả:



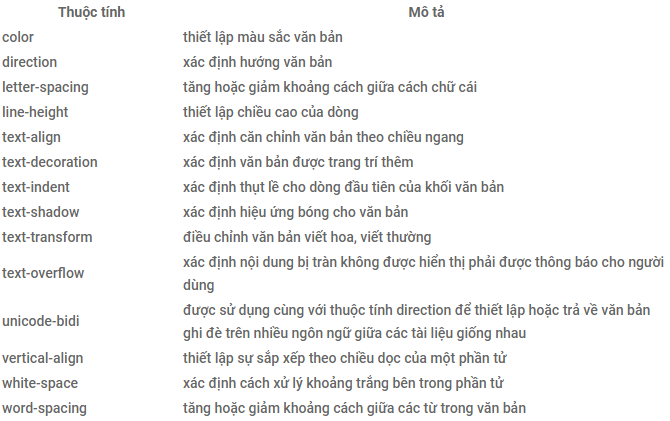
Ví dụ khác với 60o:



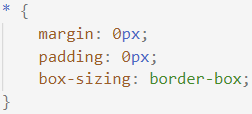
Ta có thể rút ra từ 2 ví dụ trên thì góc là góc giữa vectơ trục hoành có chiều từ trái qua phải và vectơ mặc phẳng phân cách có chiều từ trái qua phải nếu color1 nằm dưới theo chiều kim đồng hồ. Ví dụ:



Tất cả thuộc tính CSS của văn bản:



Mỗi trình duyệt có các thuộc tính khác nhau như border, padding hay margin, tuy nhiên để cho tất cả các trình duyệt đều giống nhau thì ta phải reset css. Tra google để có code mẫu. Để đơn giản nhất ta có thể cài đặt 3 thuộc tính như sau:



Thuộc tính **list-style:** Dùng để định dạng các kí tự trước 1 list nào đó.

Với giá trị **none**: Xóa các kí tự trước 1 danh sách. Ví dụ:

* Khi chưa có **list-style**:



* Khi có **list-style**:

Code này ta phải định dạng tại thẻ <ul> của danh sách cần định dạng:



Hiển thị:



Với thuộc tính **list-style-type:** Định dạng loại đánh dấu mục danh sách. Mặc định của nó là dấu **.**

Các giá trị:

* **disc**: giá trị mặc định.
* **circle**: Ví dụ:



* **square**: Ví dụ:



* **upper-roman**: Ví dụ:



* **lower-alpha**: Ví dụ:



Với thuộc tính **list-style-position**: Chỉ định nơi đặt điểm đánh dấu mục danh sách. Giá trị mặc định là **outside**.

Với thuộc tính **list-style-image**: Chỉ định loại đánh dấu mục danh sách. Giá trị mặc định là **none**.

Thuộc tính **text-decoration**: Dùng để trang trí cho văn bản.

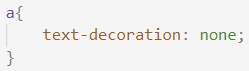
Với giá trị **none**: Xóa gạch dưới của các phần tử trong 1 danh sách. Ví dụ:

* Khi chưa có **text-decoration**:



* Khi có **text-decoration**:

Code này ta phải định dạng tại thẻ <a> của danh sách cần định dạng:



Hiển thị:



Để tránh trường hợp thay đổi code mà hiển thị trên trình duyệt không đúng như code thay đổi thì ta sử dụng lệnh **ctrl + shift + R** để xóa cache.

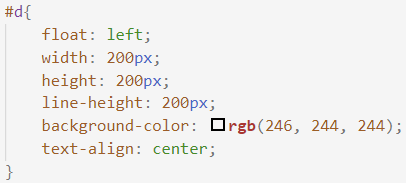
Thuộc tính **line-height**: Thiết lập chiều cao giữa các dòng. Với thuộc tính này mỗi dòng tương đương 1 khối có chiều cao bằng giá trị **line-height** và chữ được căn giữa khối đó theo chiều dọc.

Các giá trị:

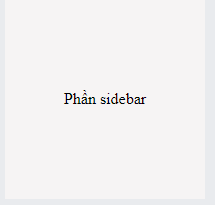
* Nomal: Không tăng khoảng cách giữa các kí tự cho chữ.
* Số: Tăng hoặc giảm khoảng cách giữa các dòng, có thể là số tự nhiên hoặc số thập phân.
* Khoảng cách: Tăng hoặc giảm khoảng cách giữa các dòng, đơn vị có thể là px, rem, em, %,…
* inherit: Xác định thừa hưởng thuộc tính từ thành phần cha.

Để cho 1 đoạn văn bản vào chính giữa 1 khối theo chiều dọc ta cho đoạn văn bản đó thuộc tính **line-height** bằng với giá trị **height** của khối.

Ví dụ:



Kết quả:



Thuộc tính **letter-spacing**: Thiết lập khoảng cách giữa các kí tự trong 1 chữ (giá trị là đơn vị đo khoảng cách).

Thuộc tính **word-spacing**: Thiết lập không gian giữa các từ trong 1 đoạn văn (giá trị là đơn vị đo khoảng cách).

Thuộc tính **word-break**: Làm cho các chữ trong các từ không còn là 1 thể thống nhất với nhau và xuống dòng ở bất cứ chữ nào trong từ.

Giá trị:

* break-all: Khi hết đoạn thì một từ sẽ tự động ngắt ở bất kỳ chữ nào để xuống hàng.
* [hyphenate](https://hocwebchuan.com/reference/cssSection/css3/example/ex_word-break_css3.php" \l "anchor02" \t "_blank): Một từ sẽ được xuống hàng tại một vị trí gạch nối thích hợp.
* normal: Trả về dạng mặc định ban đầu cho một từ.

Thuộc tính **display: flex;** : Tạo 1 flex box. Thuộc tính này được sử dụng cho thành phần cha (thành phần wrapper).

Khi thành phần cha được định dạng flex box thì các thành phần bên trong trở thành inline-block element còn thành phần cha vẫn là block element. Các thành phần bên trong của các thành phần inline-block này vẫn giữ nguyên định dạng block hoặc inline của nó.

Khi thành phần cha được định dạng flex thì các thành phần con nằm trên cùng 1 hàng và các margin của nó không trùng nhau. Trường hợp không được định dạng flex thì các thành phần con nếu khác hàng thì margin bottom của thành phần này có thể trùng với margin top của thành phần kia nhưng margin thành phần này không được trùng vào phần border của thành phần kia.

Thuộc tính **flex-grow: number;** với **number** là giá trị số bất kỳ dùng cho chính thành phần con: Thể hiện độ chiếm của background thành phần theo chiều rộng của background thành phần cha có **display: flex**. Ví dụ với 1 thành phần có **flex-grow: 0.5** thì background thành phần đó chiếm 0.5 độ rộng của tổng background của thành phần cha. Ví dụ:



Kết quả:



Thành phần có id=“ab” có **flex-grow: 0.5** có nghĩa là phần không có nội dung (không chứa “abcd”) của thành phần id=“ab” chiếm 50% chiều rộng của thành phần cha **display: flex** cũng không chứa nội dung (không chứa “abcd”).

Trường hợp các thành phần trong cùng 1 flex box có tổng flex-grow lớn hơn 1 thì tổng các thành phần đó chiếm hết chiều rộng của thành phần cha và được chia theo tỉ lệ như tỉ lệ các **flex-grow**.

Thuộc tính **flex-shrink**: Phân chia tỉ lệ khi thiếu.

Thuộc tính này giải thích rõ ràng là khi tổng chiều rộng các thành phần thực tế trên trình duyệt không bằng tổng chiều rộng các thành phần đó trên code thì phần thiếu đó sẽ được chia đều theo tỉ lệ các **flex-shrink** của các thành phần. Thuộc tính này là ngược lại so với thuộc tính **flex-grow**.

Để sắp xếp vị trí các content theo chiều ngang ta dùng thuộc tính **justify-content:**

Các giá trị:

+ **flex-start**: Tương tự như khi ta sử dụng bootstrap **justify-content-start**

+ **center**: Tương tự như khi ta sử dụng bootstrap **justify-content-center**

+ **flex**-**end**: Tương tự như khi ta sử dụng bootstrap **justify-content-end**

+ **space-between**: Tương tự như khi ta sử dụng bootstrap **justify-content-between**

+ **space-around**: Tương tự như khi ta sử dụng bootstrap **justify-content-around**

Lưu ý: các loại **justify-content:** được sử dụng khi thành phần đó là flex. Nếu thành phần đó không phải flex thì không sử dụng được.

Để sắp xếp vị trí các content theo chiều dọc ta dùng thuộc tính **align-items:**

Các giá trị cũng giống như **justify-content** nhưng không có **space-between** và **space-around**.

Thuộc tính **flex-direction**: Xác định hướng trục main để container được sắp xếp.

Các giá trị:

* **row**: mặc định, flex item được sắp xếp theo chiều ngang, từ trái qua phải (main axis nằm ngang).
* **row-reverse**: flex item được sắp xếp theo chiều ngang, từ phải qua trái (main axis nằm ngang).
* **column**: flex item được sắp xếp theo chiều dọc, từ trên xuống dưới (main axis đứng dọc).
* **column-reverse**: flex item được sắp xếp theo chiều dọc, từ dưới lên trên (main axis đứng dọc).

Thuộc tính **flex-basis**: Kích thước của 1 thành phần theo trục main.

Thuộc tính này chủ yếu được sử dụng trong trường hợp trục main được **flex-direction** thay đổi. Nếu **flex-direction** được định dạng kiểu **column** thì thuộc tính **flex-basis** là height của thành phần.

Thuộc tính **font-family**: Xác định giá trị font family cho chữ.

Các giá trị: Arial, Helvetica, …

Thuộc tính **font-size**: Xác định kích cỡ của chữ (đơn vị có thể là px, em, %, …).

Thuộc tính **font-style**: Xác định loại chữ.

Giá trị:

* **normal**.
* **italic**.
* **oblique**.

Thuộc tính **[font-variant](https://hocwebchuan.com/reference/cssSection/example/ex_font.php" \l "pr04" \t "_blank)**: Chuyển đổi kiểu chữ (thường thành hoa).

Giá trị:

* **normal**.
* **small-caps**.
* **inherit**.

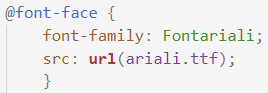
Thuộc tính **[font-weight](https://hocwebchuan.com/reference/cssSection/example/ex_font.php" \l "pr05" \t "_blank)**: Hiển thị chữ đậm hay thường.

Giá trị:

* **bold**.
* **normal**.
* Giá trị.

+ Thuộc tính **src**: Xác định đường dẫn font chữ được load.

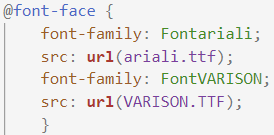
Thông thường **src** được kết hợp với **@font-face** và **font-family** để tạo 1 font mới cho định dạng CSS. Trong file định dạng CSS ta code như sau:



Với:

* **Fontariali** là tên font ta đặt để sử dụng nó trong định dạng font.
* **ariali.ttf** là đường dẫn đến font chữ đó. Trường hợp font đó mà được đặt trong 1 folder riêng giả sử nó có tên là **fonts** thì đường dẫn trở thành **fonts/ariali.ttf**.

Trường hợp ta cần đặt nhiều font thì ta viết các font liền kề nhau. Ví dụ:



Khi đó **Fontariali** là font **ariali.ttf**, **FontVARISON** là font **VARISON.TTF**.

Trường hợp ta đặt **src** cả ở trên và ở dưới **font-family** thì nó sẽ ưu tiên **src** ở dưới vì theo cơ chế overwrite thì **src** ở dưới được ưu tiên sử dụng, còn trường hợp không đặt ở dưới thì nó sẽ lấy **src** ở trên.

Trường hợp có 2 **font-family** trên và dưới 1 **src** thì cũng theo cơ chế overwrite nó sẽ tương ứng với **font-family** ở dưới.

Tốt nhất để tránh rườm rà thì ta nên viết **font-family** ở trên và viết **src** ở dưới.

Thuộc tính **background-image**: Dùng ảnh vào bên trong 1 thành phần.

Cách sử dụng: **background -image: url(**đường link hình ảnh**)**

Để hình ảnh có thể hiển thị lên trình duyệt trong 1 thẻ nào đó thì ta phải định dạng chiều cao và chiều rộng cho nó trừ khi nó là thẻ body. Nếu không định dạng thì chương trình sẽ hiểu các kích thước của nó là 0x0 thì không hiển thị lên trình duyệt được gì. Nó không giống như

Thuộc tính **background -repeat**: Tùy chỉnh có lặp lại hay không và lặp theo kiểu nào.

Các giá trị:

* **repeat**: Lặp lại từ trái qua phải và từ trên xuống dưới.
* **repeat**-**x**: Lặp lại theo trục hoành.
* **repeat**-**y**: Lặp lại theo trục tung.
* **No**-**repeat**: Không lặp lại.

Thuộc tính **background-position**: Xác định vị trí đặt ảnh so với thẻ HTML.

Đối với thuộc tính này khi ta cho nó giá trị **left top;** thì có nghĩa là vị trí của ảnh cách trái và trên 0px, cho giá trị **right top;** thì ảnh cách phải và trên 0px. Trường hợp đặt giá trị thì nó cách mép trái và mép trên bằng các giá trị nhập vào. Trường hợp nhập **center center** thì hình ảnh được căn giữa thành phần theo cả chiều ngang và chiều dọc.

Cách sử dụng ngắn:

Thay vì khai báo tất cả các thành phần rồi định dạng ta có thể khai báo như sau:



Với ví dụ trên chương trình hiểu rằng hình ảnh không lặp lại, màu nền là yellow, url(img24-png-20190726133727cbvncjKzsQ.png), cách trái 50px và trên 0px.

Lưu ý rằng ta có thể viết thứ tự thế nào cũng được vì các thuộc tính có các giá trị không thể trùng lại với nhau.

Bình thường nếu ta không sử dụng **background –repeat** thì chương trình sẽ tự động lặp.

Thuộc tính **background-size**: Quy định các kích thước của hình ảnh.

Các giá trị:

* **auto**: Giá trị mặc định. Hình ảnh được hiển thị với size gốc.
* Giá trị số: Trường hợp đặt 2 số thì giá trị trước là chiều rộng, giá trị sau là chiều cao. Đặt 1 số thì giá trị trước là chiều rộng, giá trị sau là chiều cao auto sao cho đúng tỉ lệ hình ảnh ban đầu.
* **cover**: Thay đổi kích thước hình ảnh sao cho kích thước của 1 chiều = kích thước phần tử cha và kích thước chiều còn lại lớn hơn hoặc bằng phần tử cha. Tuy nhiên nó sẽ không hiển thị ra ngoài quá phần tử cha mà nó sẽ bị cắt bớt đi phần thừa sao cho khớp với phần tử cha.
* **contain**: Thay đổi kích thước hình ảnh để hình ảnh được hiển thị đầy đủ. Khi đó cả 2 kích thước của hình ảnh ≤ kích thước của thành phần cha.
* **initial**: Đặt nó về giá trị mặc định ban đầu.

Thuộc tính **background-attachment**:

Các giá trị:

* **scroll**: Hình ảnh cuộn theo trang. Đây là dạng mặc định.
* **fixed**: Hình ảnh không được cuộn theo trang.
* **local**: Hình ảnh sẽ được cuộn với nội dung của phần tử.
* **initial**: Đặt thuộc tính này thành giá trị mặc định của nó.

**Tìm hiểu về 1 số lớp giả:**

**:active**: Thành phần được định dạng lớp giả này khi ta click chuột vào và giữ thì nó sẽ được thực thi. Khi nhả chuột ra thì trở lại trạng thái ban đầu.

**:visited**: Thành phần được định dạng lớp giả này khi nó đã được truy cập thì sau khi quay trở lại sẽ có những hiệu ứng từ định dạng css do **visited** mà không còn là hiệu ứng do css ban đầu định dạng.

**:focus**: Thành phần được định dạng lớp giả này khi ta sử dụng phím tab hướng trọng tâm vào thành phần này thì khi đó thành phần này được định dạng theo lớp giả. Ví dụ:

HTML viết:



CSS viết:



Trên trình duyệt khi ta focus bằng phím tab vào thành phần có tên Facebook:

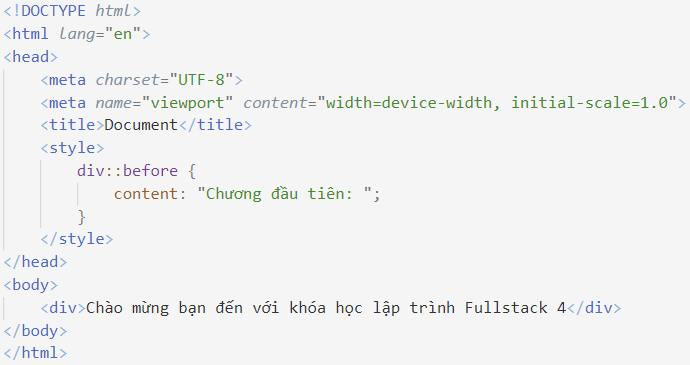


Khi đó màu của chữ **Facebook** đã đổi theo định dạng của **:focus**.

Lớp giả **::before**: Viết lên đằng trước trong thẻ HTML.

Lớp giả **::after**: Viết lên đằng sau trong thẻ HTML.

Ví dụ:



Kết quả:



Hoặc:



Kết quả của ví dụ này cũng giống với kết quả của ví dụ trước. Ta thấy với 2 lớp giả này nó được viết lần lượt là phía trước và phía sau trong thẻ HTML.

Để tạo con dấu tam giác  phục vụ cho việc tạo các dropdown hoặc Flyout menu thì ta có thể sử dụng lớp giả **::before** hoặc **::after**.

Đối với thành phần được tạo bằng **::before** hoặc **::after** nếu không được định dạng **display** thì nó là thành phần **inline**.

Xét ví dụ sau:



Kết quả:



Ta thấy rõ ràng width và height ta cho đều bằng 0, border 4 cạnh như nhau mà vẫn có sự khác nhau giữa kích thước của ô vuông màu đỏ. Vì vậy chiều cao tổng là bằng chiều cao content + chiều cao border top + border bottom. Để cho chiều cao thành phần màu đỏ trên bằng chiều cao đã định dạng thì phải chuyển thành phần màu đỏ này từ inline thành inline-block hoặc block.

Những thành phần bị hidden nhưng sau đó được hiện lên do **:hover** thì nó không được tính vào phần của phần tử cha dù cho sau khi di chuột vào phần tử đó.

Nếu 1 thành phần block nằm trong thành phần inline thì thành phần block và inline vẫn giữ nguyên kiểu của nó. Tuy nhiên thành phần block có margin hết chiều ngang màn hình.

Nếu 1 thành phần nằm trong 1 thành phần cha, nếu thành phần con không có margin thì khoảng cách từ các cạnh của thành phần con đến thành phần cha là padding.

Thuộc tính **text-align** căn chỉnh tất cả chữ có trong thành phần đó và tất cả thành phần con cháu của nó mặc dù thành phần con là block hay inline element. Tuy nhiên nếu thành phần con là các inline hoặc inline-block element thì các thành phần này cũng được căn giữa kể cả text bên trong các thành phần con đó. Ví dụ:

Khi chưa có **text-align**:



Khi đã có **text-align**:



Lưu ý: Để có thể căn chỉnh được thành phần inline-block như trên ta chỉ cần định dạng cho thành phần đó thành inline-block chứ không cần phải để các thành phần này thành flex (khác với thuộc tính **justify-content:** thành phần cha phải dạng flex).

Thuộc tính **text-decoration: line-through;** gạch ngang giữa đoạn text được định dạng thuộc tính này.

Thuộc tính **overflow:** Xác định sẽ làm gì với thành phần con nếu thành phần con có background lớn hơn thành phần cha.

Các giá trị:

* **scroll**: Phần thừa của thành phần con sẽ được hiển thị bằng cách sử dụng thanh cuộn. Ví dụ:

Ta cho thành phần cha có background nhỏ hơn thành phần con thì phần thừa ra của thành phần con phải sử dụng cuộn chuột để xem.



* **hidden**: Phần thừa của thành phần con sẽ bị ẩn đi.

Thuộc tính **overflow-x** và **overflow-y** làm tương tự đối với trục x và trục y.

Cách quy đổi màu HEX sang màu rgb:

1 thành phần của màu HEX gồm 6 chữ và số với red là 2 chữ và số đầu, green là 2 chữ và số thứ 2, blue là 2 chữ và số thứ 3.

Quy đổi 2 chữ và số của màu HEX sang rgb như sau:

Màu HEX là màu được tính theo hệ 16. Thành phần đằng trước bằng 16 lần thành phần đằng sau. Ví dụ 18 của màu HEX là 1\*16 + 8 = 24 đối với hệ 10 (là hệ của màu rgb). Với abcdef lần lượt là 10, 11, 12, 13, 14, 15. Giả sử 6b thì bằng 6\*16 + 11 = 107 = giá trị của màu rgb. Cho ví dụ quy đổi màu HEX sau:

#8b6d9f => red = 8\*16+11 = 139, green = 6\*16+13 = 109, blue = 9\*16+15 = 159

* rgb(139, 109, 159)

Thuộc tính **display: list-item** xác định thành phần đó là 1 danh sách và các thành phần con của thành phần này là các phần tử của danh sách đó.

Ví dụ:



Kết quả:



Nếu không có **display: list-item** thì chỉ có như sau:



Thuộc tính **border-collapse: collapse;** thay đổi đường kẻ giữa các ô trong 1 bảng.

Khi <table> chưa có thuộc tính **border-collapse: collapse;**:



Kết quả:



Khi <table> có thuộc tính **border-collapse: collapse;**:



Kết quả:



Nói tóm lại thuộc tính này để làm sụp đổ đường kẻ thừa của <table>.

Nếu cả 2 thành phần con và thành phần cha đều là dạng block thì thành phần con nằm trong thành phần cha nhưng background của thành phần cha không bao gồm margin của thành phần con và đường ngang trên của background 2 thành phần trùng nhau nếu thành phần cha để định dạng auto theo thành phần con.

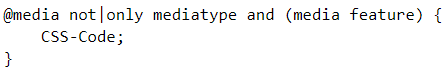
Trường hợp nếu 2 thành phần mà trong đó có 1 thành phần là inline-block thì thành phần cha auto sẽ bao gồm margin của thành phần con.

Thuộc tính **position:**

|  |  |  |  |
| --- | --- | --- | --- |
| Giá trị | Nằm trên các thành phần khác | Được tính vào thành phần cha | Tự động chuyển thành dạng inline-block |
| relative | Có | Không | Không |
| absolute | Có | Không | Có |
| fixed | Có | Không | Có |
| sticky | Có | Có (không bao giờ ra khỏi thành phần cha) | Không |

Một thành phần không nằm trong thành phần khác thì vẫn có thể lấy thành phần đó làm vị trí tuyệt đối để định dạng **position: relative;** và **position: absolute;**.

Sử dụng thuộc tính **@media** CSS: Cho phép ta tùy chỉnh từ các thiết bị khác nhau cho các giao diện khác nhau. Cú pháp:



Trong đó **@media** và and là từ khóa của thuộc tính này. **not** và **only** là lựa chọn để ta chọn, **mediatype** là 1 trong số các giá trị sau:

* **all**: Dùng cho mọi thiết bị
* **print**: Dùng cho máy in
* **screen**: Dùng cho máy tính và các thiết bị smart phone

**media feature** là điều kiện để đoạn code CSS phía dưới được thực thi.

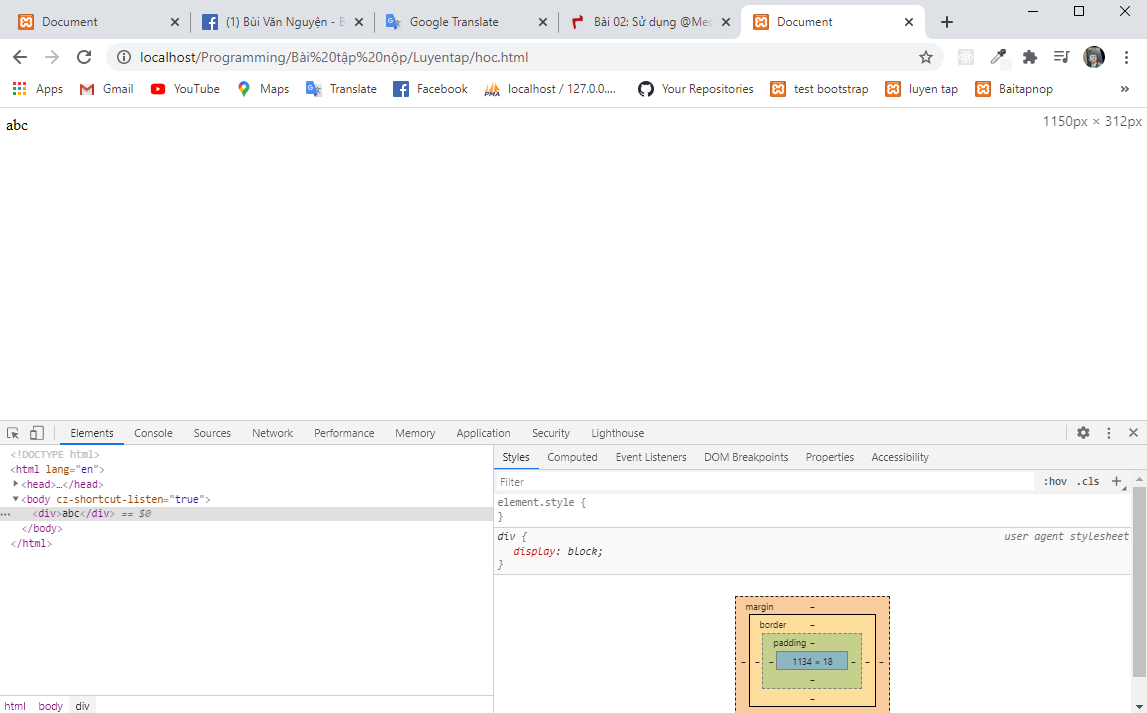
Ví dụ:



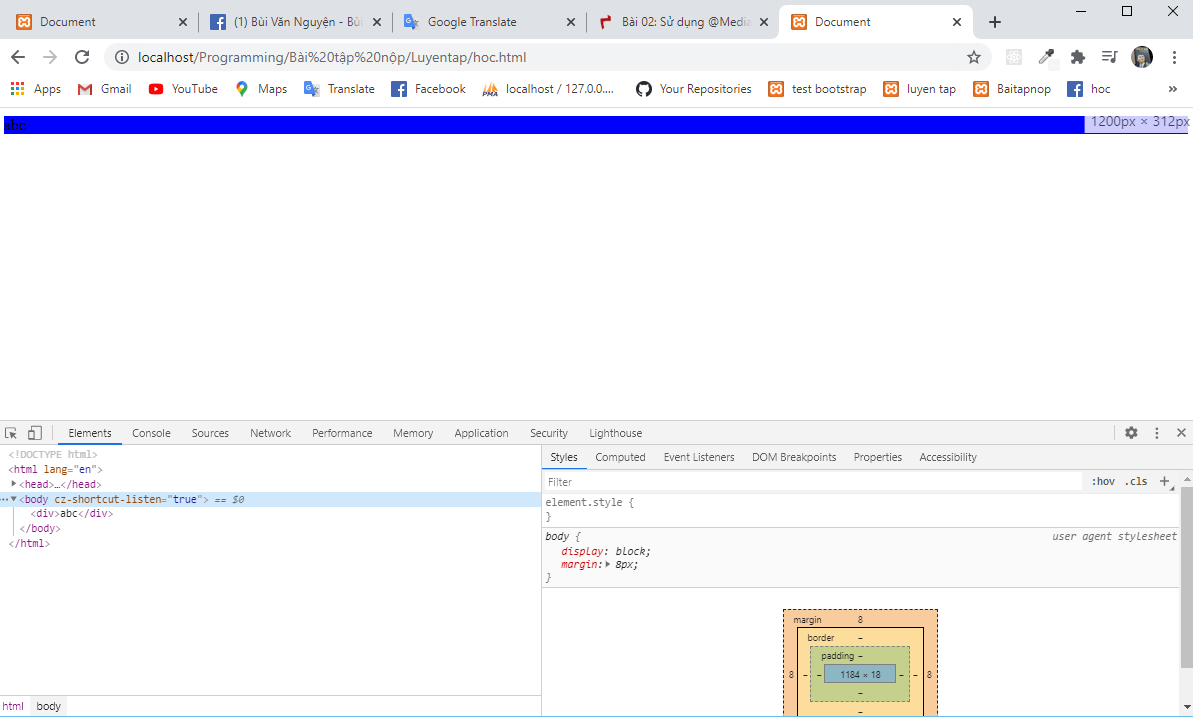
Với đoạn code trên ta có thể hiểu rằng với màn hình máy tính và các thiết bị smartphone và khi chiều rộng màn hình lớn hơn hoặc bằng 1200px thì thẻ **div** chứa đoạn text **abc** có màu background là màu **blue**.

Kết quả:

Khi màn hình nhỏ hơn 1200px:



Khi màn hình lớn hơn hoặc bằng 1200px:



Lưu ý rằng ta có thể bỏ chữ **only** trong đoạn code trên cũng được.

Trường hợp ta muốn thiết lập nhiều điều kiện thì ta thêm các từ **and** và thêm các điều kiện ở phía sau. Ví dụ ta muốn thiết lập với điều kiện ở ví dụ trên và thêm điều kiện màn hình phải nhỏ hơn 1300px ta làm như sau:



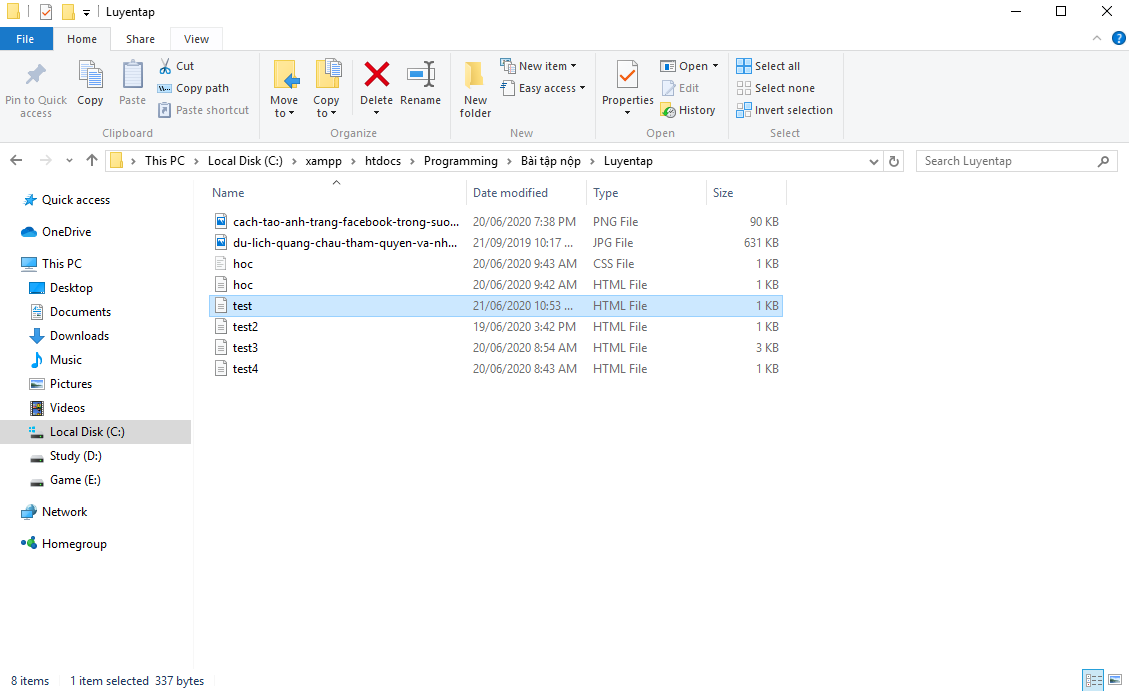
Sử dụng thuộc tính **media** trong thẻ HTML: Quy định thiết bị gì có thể được hiển thị trong tài liệu liên quan.

Ví dụ:

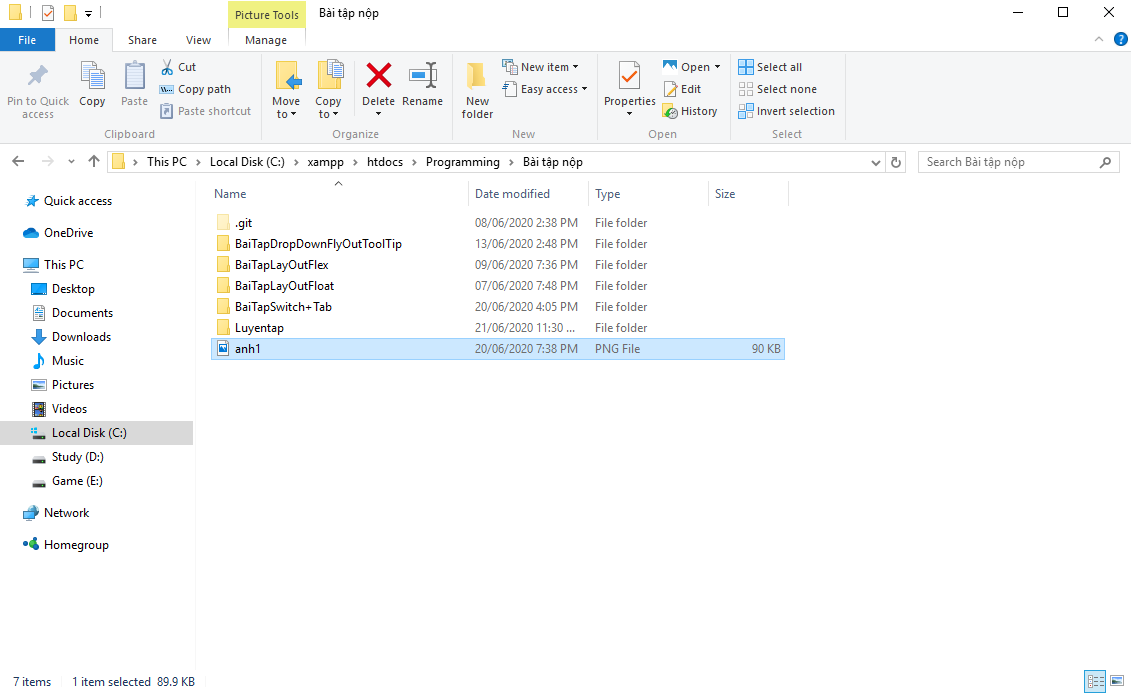


Lúc này thuộc tính **media** quy định thiết bị phải là máy tính hoặc các thiết bị smartphone, độ rộng màn hình nhỏ nhất là 900px thì thẻ link này mới có tác dụng.

Để lấy liên kết đến các file nằm ngoài thư mục chứa file đang code thì với mỗi lần chạy ra thư mục mẹ ta sử dụng 1 lần **../** ví dụ:



Giả sử ta đang code file test.html là file thuộc thư mục **Luyentap** là thư mục con của thư mục **Bài tập nộp** như hình trên. Ta cần lấy hình ảnh:



anh1.png nằm tại thư mục **Bài tập nộp** để cho nó hiển thị trên trình duyệt khi chạy code ở file test.html thì trong file test.html ta code như sau:



Như vậy hình ảnh được hiển thị bình thường trên trình duyệt.

Trường hợp file ảnh nằm trong thư mục **Programming** là thư mục cha của thư mục **Bài tập nộp** thì ta code như sau:



Như vậy với mỗi lần ra 1 thư mục cha thì ta sử dụng 1 lần **../**

Thẻ <base /> đặt đường dẫn tương đối cho các đường link khác trong file HTML.

Ví dụ:



Khi đó src của hình ảnh trong thẻ HTML là **<https://cdn.tgdd.vn/Files/2017/01/20/939784/cach-tao-anh-trang-facebook-trong-suot-9.png>**

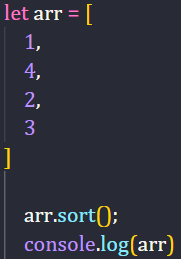
Như vậy ta có thể hiểu rằng thẻ <base /> đặt đường dẫn tương đối cho các đường dẫn khác truy cập được ngắn gọn hơn.

Chú ý: đường link trong thẻ base phải được kết thúc bằng dấu **“/”.**

**JAVASCRIPT**

Hàm **map()** khác với **forEach()** ở chỗ nó trả về mảng mới có số phần tử bằng với số phần tử của mảng ban đầu và mỗi phần tử trong mảng mới được tạo ra từ mỗi lần lặp trong **map**. Do đó ta có thể dùng biến để hứng các giá trị được lặp trên và tạo thành 1 mảng mới.

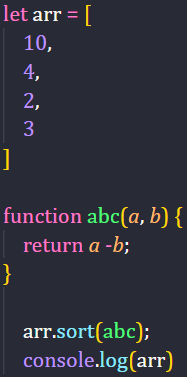
Hàm **sort**(): sắp xếp các giá trị theo alpha.b. Ví dụ:



Kết quả theo thứ tự alpha.b. Nó cũng tương tự như với các phần tử là chuỗi.

Tuy nhiên đối với trường hợp sắp xếp số giữa phần tử 10 và 4 thì 10 lại được sắp xếp trước do 1 < 4 và nó sắp xếp từ nhỏ đến lớn.

Đối với trường hợp này ta phải chèn thêm 1 hàm so sánh như sau:



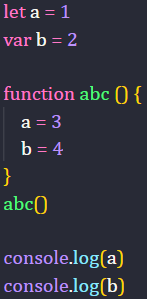
Tìm hiểu các nguyên tắc của var và let:

+ Phạm vi của var là bên trong hoặc bên ngoài hàm số. Nó là phạm vi của một hàm.

+ Phạm vi của let là bên trong phạm vi 1 khối, xác định bởi cặp dấu {}.

Chú ý rằng đối với câu điều kiện if và vòng lặp for không được tính là hàm. Vì vậy chủ yếu sự khác nhau giữa var và let được thể hiện bên trong câu điều kiện và vòng lặp này.

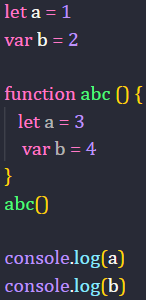
Trường hợp đặt biến bên ngoài (global) theo 2 cách mà trong hàm (kể cả câu điều kiện if và vòng lặp for) gán giá trị khác cho nó thì giá trị sẽ được lấy theo giá trị trong hàm. Ví dụ:



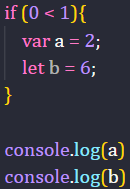
Kết quả:



Đối với trường hợp sau thì cả 2 không ảnh hưởng đến giá trị ban đầu:

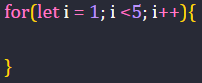


Đối với trường hợp như sau thể hiện sự khác nhau giữa var và let:



Kết quả a = 2 và b not defined. Nguyên nhân var a được khai báo toàn cục, let b được khai báo bên trong cặp dấu {} nên khi ra ngoài không có tác dụng.

Chú ý với trường hợp sau:



i được xem trong cặp dấu {} của vòng lặp for nên nó sẽ không có tác dụng khi đi ra khỏi vòng lặp.

Tóm lại ta được bảng sau:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | Khai báo và gán giá trị xong thì khai báo và gán giá trị tiếp | | Khai báo và gán giá trị xong thì gán giá trị tiếp | |
| Trong cùng phạm vi function | var | Được, thay đổi giá trị cũ | | Được, thay đổi giá trị cũ | |
| let | Không được | | Được, thay đổi giá trị cũ | |
| Khai báo bên ngoài function và thực hiện bên trong function | var | Được, không ảnh hưởng giá trị cũ | | Được, thay đổi giá trị cũ | |
| let | Được, không ảnh hưởng giá trị cũ | | Được, thay đổi giá trị cũ | |
| Khai báo bên ngoài {} và thực hiện bên trong {} | var | Tương đương trong cùng function | | Tương đương trong cùng function | |
| let | Được, không ảnh hưởng giá trị cũ | | Được, thay đổi giá trị cũ | |
| Khai báo bên trong {} và thực hiện bên ngoài {} (không tính {} của function) | var | Được, thay đổi giá trị cũ | | Được, thay đổi giá trị cũ | |
| let | Được, không ảnh hưởng giá trị cũ | | Không được | |

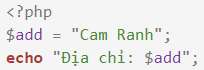
**BACK END**

Trong PHP thì dấu nháy đơn nhanh hơn dấu nháy đôi. Còn trong JavaScript thì như nhau và không phân biệt.

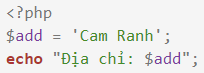
Phần view page source trong Google Chrome chỉ hiển thị những đoạn code của HTML, CSS và JavaScript chứ nó không hiển thị những thành phần backend. Những phần hiển thị này là những đoạn code được server response về.

Với ghép chuỗi trong PHP thì dấu nháy đôi có thể chứa biến khác trong chuỗi đó. Nhưng với dấu nháy đơn thì không thể chứa biến khác trong chuỗi. ví dụ:

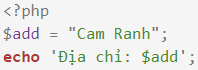
Như thế này là đúng:



Hoặc:



Như thế này là sai:



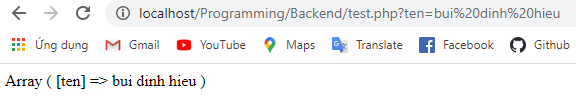
Với các biến môi trường ta có:

$\_GET: Là biến được lấy từ địa chỉ URL. Phần URL được dùng để lấy giá trị là phần sau dấu “?” của một số địa chỉ URL của 1 số trang web nhất định. Nó là 1 mảng. Xét ví dụ sau:

Bên PHP ta code:



Bên trình duyệt:



Phần sau dấu ‘?’ là phần mà $\_GET lấy. Phần đó tương đương **ten=bui dinh hieu** khi đó nó sẽ trả về 1 array với key là **ten** và value là **bui dinh hieu**.

Hàm **explode**: Hàm tách chuỗi. Cú pháp: **$arr = explode($character, $string);**

Trong đó **$character** là kí tự làm căn cứ để tách chuỗi, **$string** là chuỗi cần tách.

Ví dụ:



Kết quả:

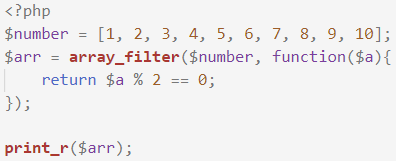


Hàm **implode**: Ngược lại với hàm **explode**.

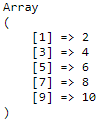
Đối với PHP thì tất cả các biến là biến tham trị, còn trong JavaScript thì tất cả các biến là biến tham chiếu. Tuy nhiên biến trong PHP có thể chuyển thành dạng tham chiếu bằng cách sử dụng **&**.

Hàm **array\_filter()**: Lọc phần tử thỏa mãn trong hàm và trả về mảng các phần tử thỏa mãn điều kiện.

Ví dụ:



Kết quả:



Hàm **array\_pop()**: Xóa phần tử cuối cùng trong mảng.

Hàm **array\_shift()**: Loại bỏ phần tử đầu tiên của mảng.

Hàm **array\_reduce()**: Tính toán dựa vào các phần tử trong mảng.

Hàm **array\_map()**: Lặp tất cả các phần tử của mảng và truyền vào hàm callback.

Hàm **array\_unshift()**: Thêm một hoặc nhiều phần tử vào đầu mảng. Hàm trả về số nguyên là số phần tử của mảng mới được thêm.

Hàm **array\_slice()**: Cắt bỏ phần tử của mảng, giữ lại những phần tử được chọn.

Hàm **array\_splice()**: Xóa phần tử trong mảng và thay thế bằng một hoặc một số phần tử khác.

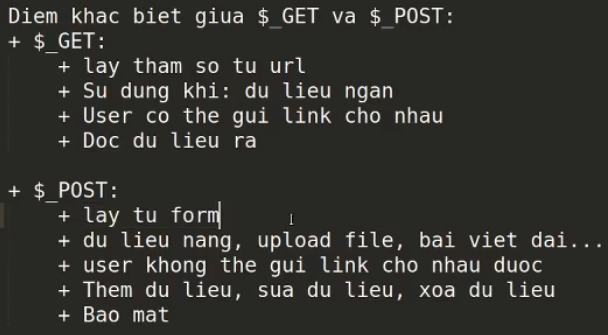
Hàm **array\_unique()**: Xóa những phần tử trùng lặp và để lại phần tử đầu tiên.

Hàm **array\_merge()**: Trộn 2 hoặc nhiều mảng.

Hàm **array\_keys()**: Trả về một mảng các keys.

Hàm **array\_values()**: Trả về một mảng các values.

Điểm khác nhau giữa $\_GET và $\_POST:



Khi ta muốn sử dụng session thì ta phải dùng hàm **session\_start();** để khởi tạo session. Tại mỗi trang code ta đều phải đặt **session\_start()** để có thể sử dụng session trong trang code đó.

Trong máy chủ ảo xampp thì session được đặt ở thư mục C:\xampp\tmp

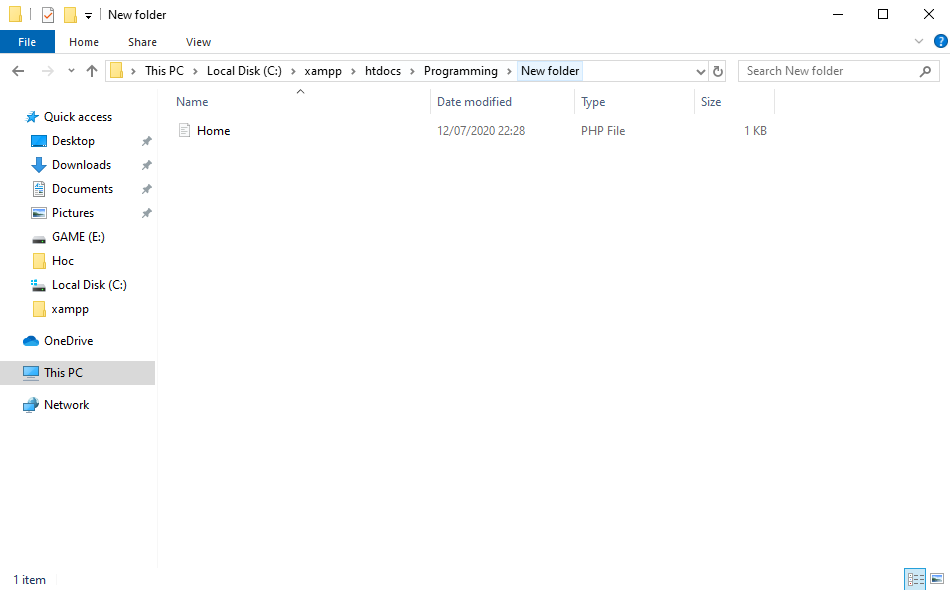
Khi gửi dữ liệu với form thì mặc định nó sử dụng phương thức get để gửi dữ liệu.

Hàm **header()**: Dùng để chuyển hướng trang web. Ví dụ:

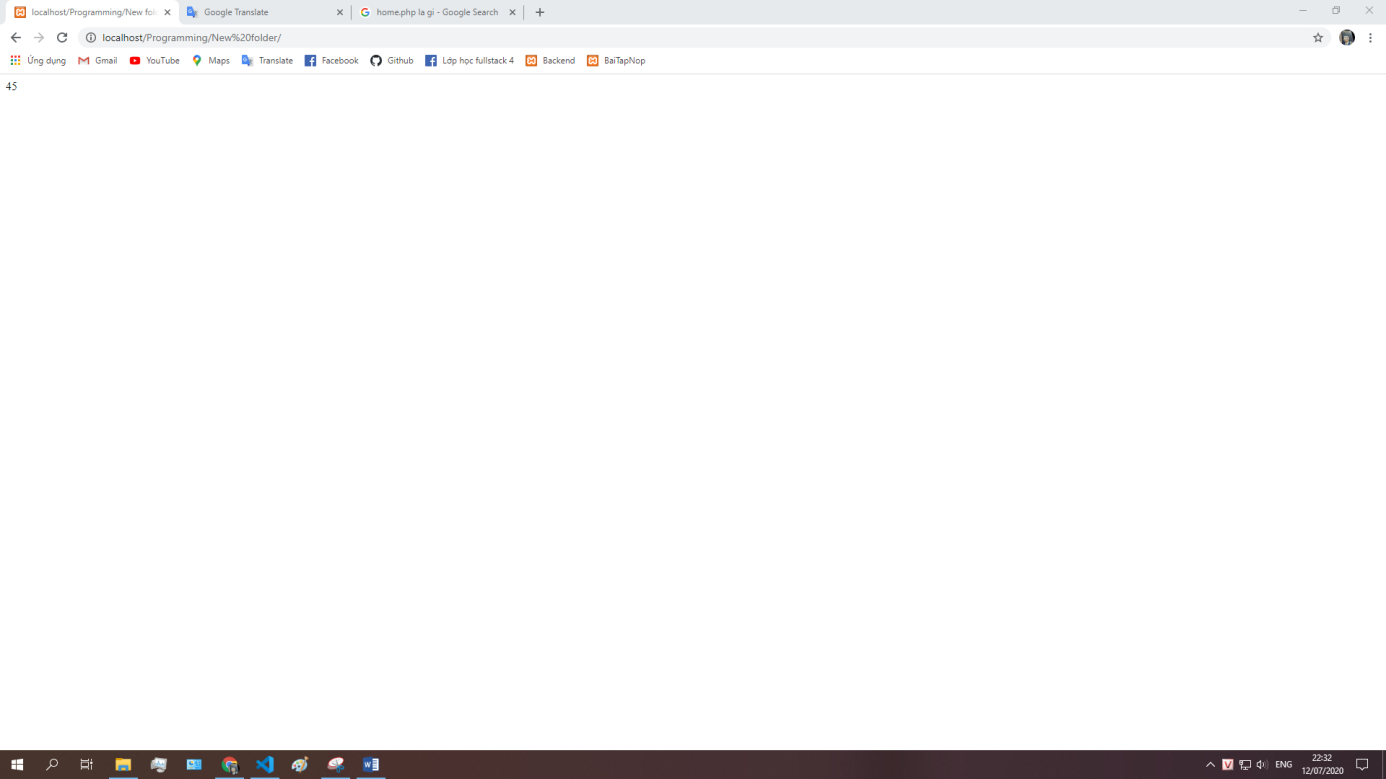


Từ khóa **Location:** là từ bắt buộc phải có. Phần đằng sau là địa chỉ đường link để chuyển hướng.

Trang **Home.php** là 1 trang đặc biệt do:

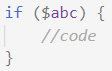


Khi này trang **Home.php** được đặt trong folder **New folder** nhưng khi lên trình duyệt ta vào thư mục **New folder** thì chương trình đã cho ra kết quả của file **Home.php** luôn:



Điều đó cho thấy rằng file **Home.php** là đại diện luôn cho thư mục **New folder** mà khi vào folder đó thì chương trình vào thẳng luôn file **Home.php**. Vì vậy để tránh xảy ra kết quả không như ý muốn thì ta không nên đặt tên file là **Home.php**. File tên **index.php** cũng xảy ra tình trạng tương tự.

Đối với câu điều kiện dạng như sau:



Thì điều kiện **true** khi $abc có giá trị và false khi $abc không có giá trị. Tuy nhiên $abc phải được định nghĩa trước, nếu chưa được định nghĩa mà đưa vào xét thì gây ra lỗi chương trình.

Hàm ép kiểu **intval()**: Ép cho dữ liệu đầu vào là 1 kiểu dữ liệu nhất định (ở đây là kiểu integer), tránh việc người dùng nhập dữ liệu kiểu khác gây ra lỗi.

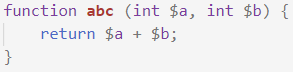
Hàm **gettype()**: Lấy ra kiểu dữ liệu mà người dùng nhập vào biến đó.

Ta có thể ép kiểu dữ liệu bằng cách sau:



Tuy kiểu dữ liệu mặc định khi nhập 12345 là integer nhưng khi ta nhập string phía trước biến đó trong ngoặc thì $abc thành kiểu chuỗi.

Cách này cũng có thể ép kiểu dữ liệu cho tham số. Ví dụ:



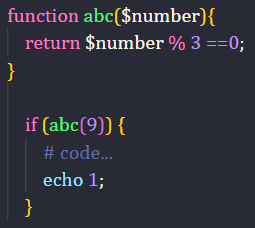
Nó được áp dụng từ PHP 7.

Để sử dụng hằng số ta dùng cú pháp: **const aa =** (giá trị nào đó)**;**

Do **aa** là hằng số nên ta không thể định nghĩa lại và gán cho nó giá trị khác. Để sử dụng hằng số này cho việc khác ta chỉ cần viết **$abc = aa;** trong đó **$abc** là biến số cần đặt giá trị.

Khi dùng thẻ <input> có type=“submit” thì nó mới có tác dụng gửi trong form. Nếu trong form có thẻ <button> thì nếu không đặt type cho button đó thì nó mặc định là submit và có thể gửi được. Nếu đặt type=“button” thì không thể gửi.

Thường những đoạn code so sánh trong hàm khi return những đoạn code đó thì chương trình sẽ trả về true hoặc false. Ví dụ:



Kết quả trả về là 1. Kiểu này áp dụng cho cả JavaScript.

Thuộc tính contextType: Sử dụng với class để gọi giá trị của context bên trong class đó. Ví dụ:

Có 1 đối tượng context **MyContext** được tạo như sau:

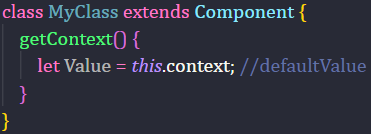
MyContext = React.createContext(defaultValue);

Và 1 class tên là **MyClass** khi đó ta đặt như sau:

MyClass.contextType = MyContext

Lưu ý: Đoạn code trên phải đặt bên ngoài class **MyClass** và phải được viết dưới phần khai báo component đó.

Khi đó trong class **MyClass** ta có thể gọi giá trị context đó bằng cú pháp sau:



Trong đó this.context là từ khóa. Khi đó **Value = defaultValue**

Phương pháp này dùng để truyền dữ liệu từ context vào 1 component mà không dùng đến context Consumer (context Consumer chỉ dùng được trong phần JSX mà không dùng được trong phần JS nên contextType làm điều này).