

Lập trình website sử dụng c#



Mục lục

[Chương 1: Collections trong C# 2](#_Toc514860519)

[1. ArrayList 2](#_Toc514860520)

[2. List<T> 3](#_Toc514860521)

[3. HashSet<T> 4](#_Toc514860522)

[4. HashTable 4](#_Toc514860523)

[5. SortedList<Tkey,Tvalue> 6](#_Toc514860524)

[6. Dictionary<Tkey,Tvalue> 7](#_Toc514860525)

[7. Stack<T> 9](#_Toc514860526)

[8. Queue<T> 10](#_Toc514860527)

[Chương 2: Delegate trong C# 12](#_Toc514860528)

[Chương 3: CSS Display 14](#_Toc514860529)

[1. Thuộc tính display 14](#_Toc514860530)

[a. Block 14](#_Toc514860531)

[b. Inline 15](#_Toc514860532)

[c. Inline – block 16](#_Toc514860533)

[d. Table 17](#_Toc514860534)

[e. Inline – table 18](#_Toc514860535)

[f. Float 19](#_Toc514860536)

[2. Margin giữa các display 20](#_Toc514860537)

[3. Sự khác nhau về thuộc tính width, height và padding giữa các display 24](#_Toc514860538)

[a. Block 24](#_Toc514860539)

[b. Inline 26](#_Toc514860540)

[c. Table 26](#_Toc514860541)

[d. Inline-Block 27](#_Toc514860542)

[e. Inline-Table 28](#_Toc514860543)

[Chương 4: Bộ chọn selector và các hàm truy xuất selector trong jQuery 29](#_Toc514860544)

[1. Selector là gì? 29](#_Toc514860545)

[2. Một số selector thường gặp 30](#_Toc514860546)

[a. Hàm cơ sở 31](#_Toc514860547)

[b. Các hàm truy xuất đến thành phần con và truy xuất ngược đến thành phần cha 31](#_Toc514860548)

[c. Các hàm xử lý thuộc tính của thành phần 36](#_Toc514860549)

[3. Các hàm bắt sự kiện trong Jquery 39](#_Toc514860550)

[4. Bài tập 44](#_Toc514860551)

# Chương 1: Collections trong C#

## ArrayList

ArrayList là một collection lưu trữ và quản lý một danh sách các đố tượng theo kiểu mảng. Ta có thể truy cập phần tử thông qua chỉ số index.

ArrayList có thể thêm, xóa phần tử một cách linh hoạt và có thể tự điều chỉnh kích cỡ tự động.

Cú pháp khởi tạo:

//Cách 1: Khởi tạo 1 arraylist rỗng

ArrayList arrList = new ArrayList();

//Cách 2: Khởi tạo và chỉ định sức chứa ban đầu

Arraylist arrList = new ArrayList(10);

//Cách 3: Khởi tạo dựa trên mảng có sẵn

ArrayList arrList = new ArrayList(arr);

Một số thuộc tính thông dụng:

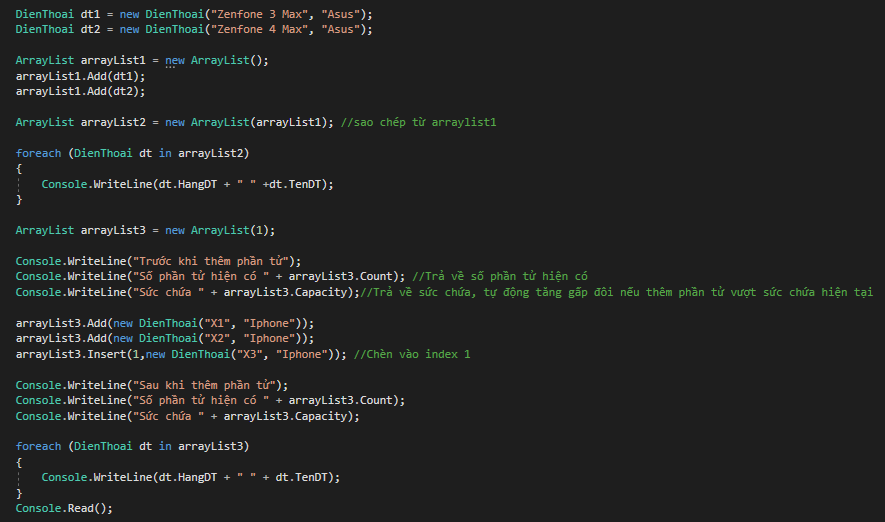
+ Count: Trả về 1 số nguyên là số lượng phần tử đang có trong ArrayList

+ Capacity: Trả về 1 số nguyên là sức chứa của ArrayList. Nếu số phần tử thêm vào vượt mức thì sức chứa sẽ tự động tăng lên

Một số phương thức thông dụng:

|  |  |
| --- | --- |
| **Tên phương thức** | **Mô tả** |
| Add(object value) | Thêm đối tượng vào cuối ArrayList |
| AddRange(Icollection ListObject) | Thêm nhiều đối tượng vào cuối ArrayList |
| Clear() | Xóa tất cả các phần tử |
| Constains(object value) | Kiểm tra đối tượng có tồn tài hay không. Trả về true nếu có, false nếu không |
| GetRange(int start,int end) | Trả về 1 ArrayList bao gồm các phần tử từ ví trí start đến end trong ArrayList ban đầu |
| IndexOf(object value) | Trả về ví trị xuất hiện đầu tiên của đối tượng trong ArrayList |
| LastIndexOf(object value) | Trả về vị trí xuất hiện cuối cùng của đối tượng trong ArrayList |
| Remove(object value) | Xóa đối tượng xuất hiện đầu tiên trong ArrayList |
| Reverse() | Đảo ngược tất cả phần tử |
| Sort() | Sắp xếp các phần tử theo thứ tự tăng dần |
| ToArray() | Trả về 1 mảng chứa các phần tử |
| Insert(int index,object value) | Chèn đối tượng vào vị trí index |
| InsertRange(int index, listObject) | Chèn danh sách các đối tượng từ vị trí index |

Demo:



## List<T>

List là 1 generic collections thay thế cho ArrayList.

Sự khác nhau giữa List và Arraylist.

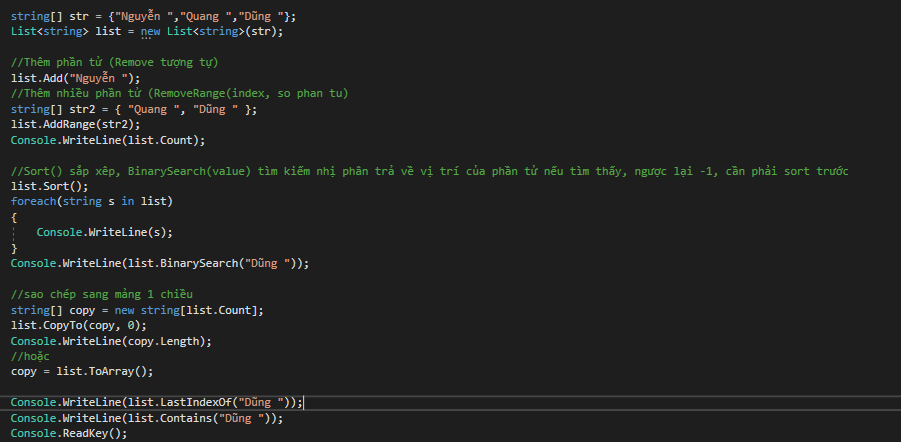
|  |  |
| --- | --- |
| **List** | **ArrayList** |
| Phải khai báo kiểu lưu trữ của các đối tượng. Nếu thêm các đối tượng khác kiểu sẽ xuất lỗi | Lưu trữ các đối tượng kiểu object |
| Giải quyết được việc boxing / unboxing khi truy xuất phần tử | Boxing / Unboxing khi truy xuất phần tử |

Đặc điểm chung của List<T> và ArrayList:

+ Truy xuất các phần tử bằng chỉ số index.

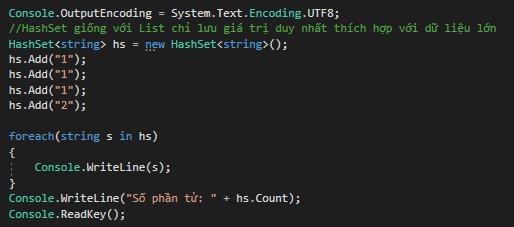
+ Việc thêm, chèn, xóa các phần tử (không phải ở cuối danh sách) tốn nhiều chi phí vì phải dịch các phần tử khác lên hoặc xuống trong danh sách.

+ List<T> thường được sử dụng khi không có ràng buộc nào.

Demo:

## HashSet<T>

HashSet<T> giống với List<T>. Điều khác biệt là nó chỉ lưu trữ các đối tượng duy nhất. Nghĩa là các đối tượng không trùng nhau. Thích hợp cho việc lưu trữ dữ liệu lớn. Nhanh hơn trong việc tìm kiếm vì các phần tử không trùng nhau.

Demo:

## HashTable

HashTable là một collection lưu trữ đối tượng dưới dạng Key – Value. Key đại diện cho 1 khóa như chỉ số index của mảng và Value là giá trị tương ứng của khóa đó.

Một cặp Key – Value được định nghĩa là 1 đối tượng có kiểu DictionaryEntry. Trong đó có 2 thuộc tính là Key và Value.

Cú pháp khởi tạo:

//Cách 1: Khởi tạo Hashtable rỗng

Hashtable ht = new Hashtable();

//Cách 2: Khởi tạo và chỉ định sức chứa ban đầu

Hashtable ht = new Hashtable(10);

//Cách 3: Khởi tạo dựa trên hashtable có sẵn

Hashtable ht = new Hashtable(hashtable);

Một số thuộc tính thông dung:

+ Count: Trả về 1 số nguyên là số phần tử hiện có trong Hashtable.

+ Keys: Trả về 1 danh sách chứa key.

+ Values: Trả về 1 danh sách chứa value.

Một số phương thức thông dụng:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| Add(key,value) | Thêm 1 cặp Key-Value vào Hashtable |
| Clear() | Xóa tất cả các phần tử |
| ContainsKey(key) | Kiểm tra key có tồn tại trong Hashtable không |
| ContainsValue(value) | Kiểm tra value có tồn tài trong Hashtable không |
| Remove(key) | Xóa đối tượng có key trong Hashtable |

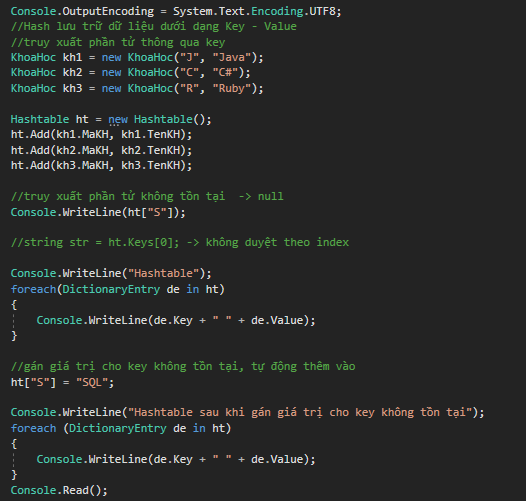
Ta có thể truy xuất đến các phần tử trong Hashtable thông qua Key. Ví dụ:

ht[“key”] // xuất ra value tương ứng

Nếu ta thực hiện lấy giá trị với key không tồn tại thì kết quả sẽ trả về null và không thông báo lỗi.

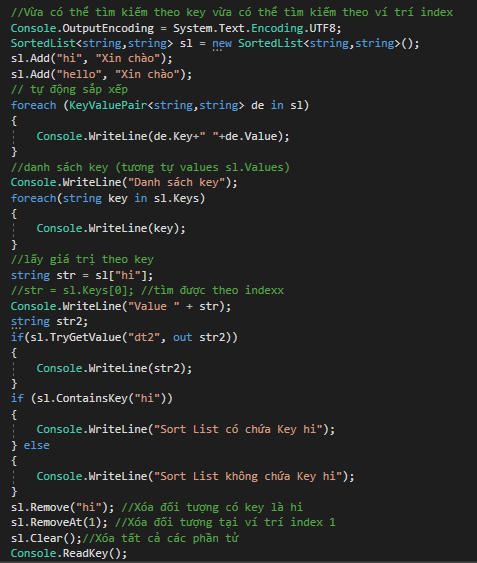
Nếu ta thực hiện gán giá trị cho key không tồn tại thì Hashtable sẽ tự động thêm 1 phần tử mới. Điều này làm phát sinh các phần tử không mong muốn.

Demo:



## SortedList<Tkey,Tvalue>

SortedList lưu trữ các đối tượng Key – Value dưới dạng mảng. Các phần tử sẽ được sắp xếp theo Key. Việc sắp xếp sẽ thực hiện một cách tự động khi thêm xóa các phần tử. Điều này dẫn đến tốn thời gian khi thêm, xóa phần tử, nhưng việc liên kết các đối tượng được sắp xếp theo chỉ số của mảng sẽ giúp tìm kiếm nhanh hơn.

Demo:

## Dictionary<Tkey,Tvalue>

Dictionary<Tkey,Tvalue> là một collections thay thế cho Hashtable. Thời gian thêm, xóa, tìm kiếm rất nhanh vì nó sử dụng 1 bảng băm để băm các key. Nhược điểm là các phần tử không được sắp xếp nên không dễ dàng duyệt theo một thứ tự nào đó.

Cú pháp khởi tạo:

//Cách 1: khởi tạo Dictionary rỗng

Dictionary<string,string> myDic = new Dictionary<string,string>();

//Cách 2: khởi tạo và gán sức chứa ban đầu

Dictionary<string,string> myDic = new Dictionary<string,string>(10);

//Cách 3: khởi tạo từ 1 dictionary có sẵn

Dictionary<string,string> myDic = new Dictionary<string,string>(myDic2);

Một số thuộc tính thông dụng:

+ Count: Đếm số lượng phần tử đang có trong Dictionary.

+ Keys: Trả về 1 danh sách chứa các Key trong Dictionary.

+ Values: Trả về 1 danh sách chứa các Value trong Dictionary.

Một số phương thức thông dụng:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| Add(key,value) | Thêm 1 cặp key – value |
| Clear() | Xóa tất cả phần tử |
| Remove(key) | Xóa đối tượng có key |
| ContainsKey(key) | Kiểm tra đối tượng Key có tồn tại không |
| ContainsValue(value) | Kiểm tra value có tồn tài không |
| TryGetValue(Tkey,TValue) | Kiểm tra key có tồn tại không. Nếu true trả về giá trị value tương ứng. Ngược lại false |

Sự khác biệt giữa Dictionary và Hashtable

|  |  |
| --- | --- |
| **Dictionary** | **Hashtable** |
| Key-value phải xác định kiểu cụ thể | Key – value kiểu object |
| Truy xuất phần tử không tồn tại sẽ thông báo lỗi | Truy xuất phần tử không tồn tại sẽ không thông báo lỗi mà trả về null |

Demo:

## Stack<T>

Stack hay ngăn xếp là một tập hợp các đôi tượng quản lý tuần tự theo cơ chế vào sau ra trước (LIFO). Các phần tử được thêm và xóa từ trên cùng của Stack, không thể truy xuất các phần tử theo index. Thích hợp cho việc muốn quay lui các thao tác theo thứ tự.

Cú pháp khởi tạo:

//Cách 1: khởi tạo Stack rỗng

Stack<string> myStack = new Stack<string>();

//Cách 2: khởi tạo và gán sức chứa ban đầu

Stack<string> myStack = new Stack<string>(5);

//Cách 3: khởi tạo từ 1 dictionary có sẵn

Stack<string> myStack = new Stack<string>(array);

Một số phương thức thông dụng:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| Push(obj) | Thêm đối tượng vào vị trí trên cùng |
| Pop() | Lấy đối tượng trên cùng và loại khỏi stack |
| Peek() | Lấy giá trị của đối tượng trên cùng nhưng không loại khỏi stack |

Demo:

## Queue<T>

Queue<T> hay hàng đợi là một collection quản lý đối tượng tuần tự theo nguyên lý vào trước ra trước (FIFO). Thích hợp quản lý theo thứ tự mà chúng được thêm vào như máy in hay xếp hàng thanh toán.

Cú pháp khởi tạo:

//Cách 1: khởi tạo Queue rỗng

Queue<string> myQueue = new Queue <string>();

//Cách 2: khởi tạo và gán sức chứa ban đầu

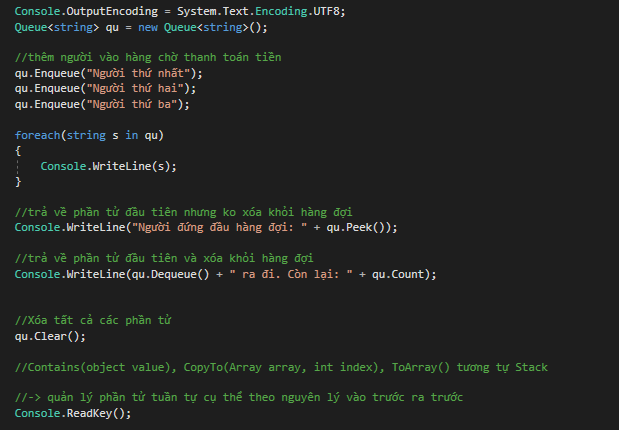
Queue <string> myQueue = new Queue <string>(5);

//Cách 3: khởi tạo từ 1 dictionary có sẵn

Queue <string> myQueue = new Queue <string>(array);

Một số phương thức thông dụng

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| Enqueue() | Thêm đối tượng |
| Dequeue() | Lấy đối tượng đầu tiên và loại khỏi queue |
| Peek() | Lấy giá trị của đối tượng đầu tiên nhưng không loại khỏi Queue |

Demo:

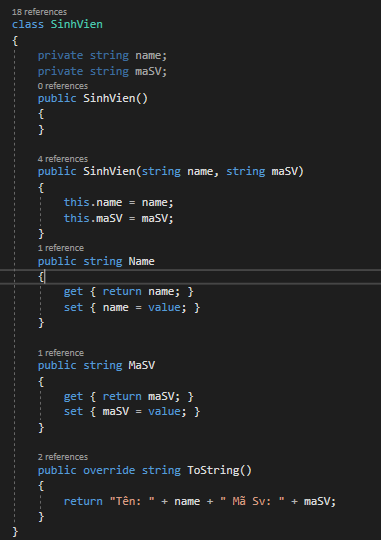
# Chương 2: Delegate trong C#

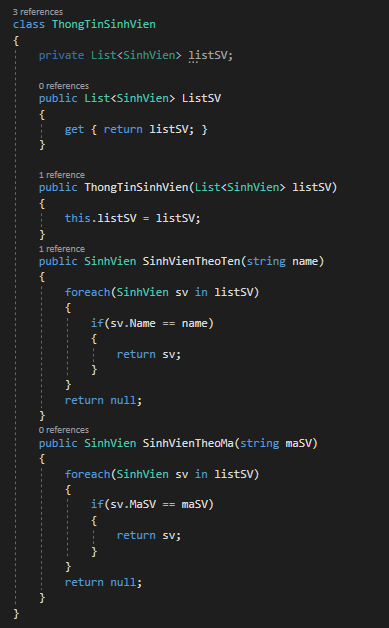
Delegate là các đối tượng chứa các tham chiếu đến các phương thức mà nó gần được gọi thay cho tên của phương thức thực sự.

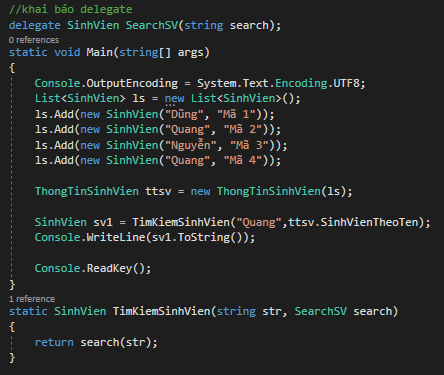
Một delegate có thể được dùng để tạo một bao đóng cho bất kỳ phương thức nào, miễn là nó phù hợp (kiểu trả về, tham số).

Delegate có thể gọi phương thức ở bất kỳ nơi nào: từ class này đến class khác, từ thread này đến thread khác.

Demo:

Ta có class SinhVien:

Class ThongTinSinhVien

Thực thi:

Phương thức tìm kiếm SinhVienTheoTen(string name) của class ThongTinSinhVien được truyền như một tham số. Một đối tượng delegate sẽ gọi phương thức này và thực thi phương thức này.

# Chương 3: CSS Display

## Thuộc tính display

Display là thuộc tính xác định kiểu hiển thị của các thành phần trong html.

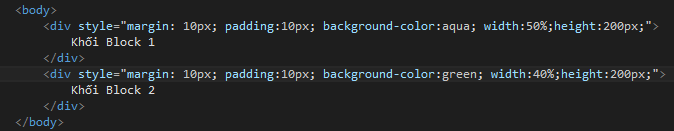
Một số kiểu hay sử dụng: block, inline, inline-block, table, inline-table.

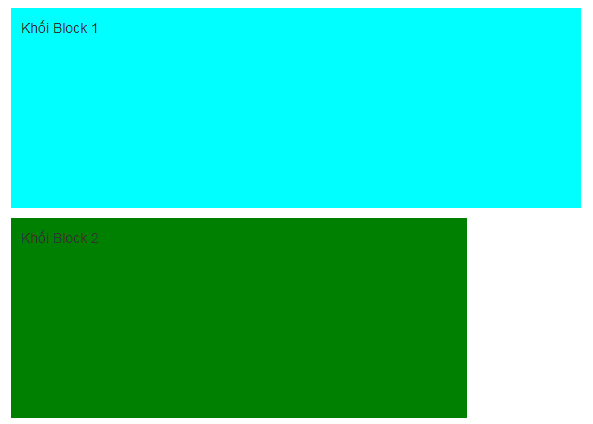
### Block

Thành phần hiển thị như một khối, khi sử dụng giá trị block, thành phần sẽ đứng một hàng độc lập so với thành phần trước và sau nó.

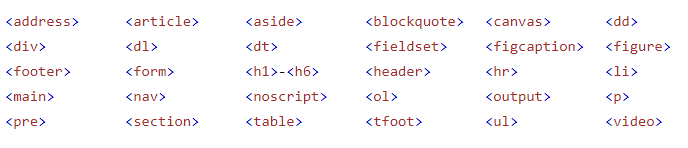
Có thể đặt giá trị tương đối (%) hoặc tuyệt đối (pt,px,..) cho thuộc tính width và height.

Thuộc tính margin và padding ảnh hưởng đến tất cả các mặt của thành phần.

Ví dụ:

Kết quả:

Các thẻ mặc định có display mặc định là block:

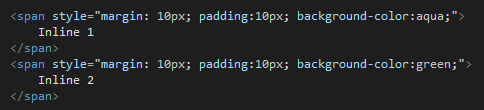


### Inline

Thành phẩn hiển thị nội tuyến, không ngắt dòng. Khi sử dụng giá trị inline các thành phần sẽ nối đuôi nhau.

Không nhận các giá trị của thuộc tính width và height.

Thuộc tính margin, padding chỉ ảnh hưởng theo chiều ngang.

Ví dụ:

Kết quả:

Các thẻ có display mặc định inline:

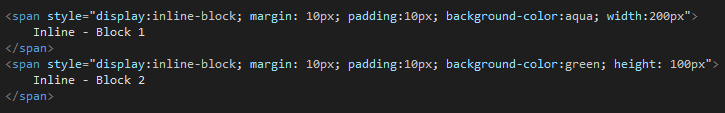
### Inline – block

Thành phần sẽ hiển thị như một khối, nhưng là một khối nội tuyến. Khi sử dụng inline – block thành phẩn các thành phần khối sẽ nối đuôi nhau.

Nhận giá trị của các thuộc tính width và height.

Thuộc tính margin và padding ảnh hưởng tất cả các mặt của thành phần.

Ví dụ:

Kết quả:

### Table

Thành phần được đối xử như một <table>, tích hợp các display:table-children để hiện thị các thành phần như một bảng.

Độ rộng và độ cao phụ thuộc vào độ rộng và độ cao của nội dung bên trong.

Các thành phần con:

+ table-cell

+ table-column

+ table-colgroup

+ table-caption

+ table-row

+ table-row-group

+ table-header-group

+ table-footer-group

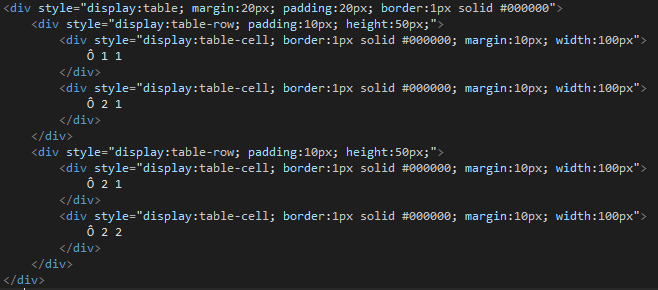
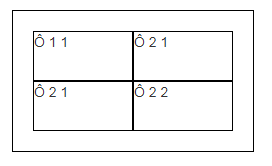
Display:table nhận các giá trị width, height và margin, padding như block.

Display:table-row không nhận giá trị width chỉ nhận giá trị heght và không nhận giá trị padding chỉ nhận margin.

Display:table-cell nhận giá trị width và height, không nhận giá trị margin chỉ nhận giá trị padding

Các bảng sẽ bắt đầu bằng dòng mới như block.

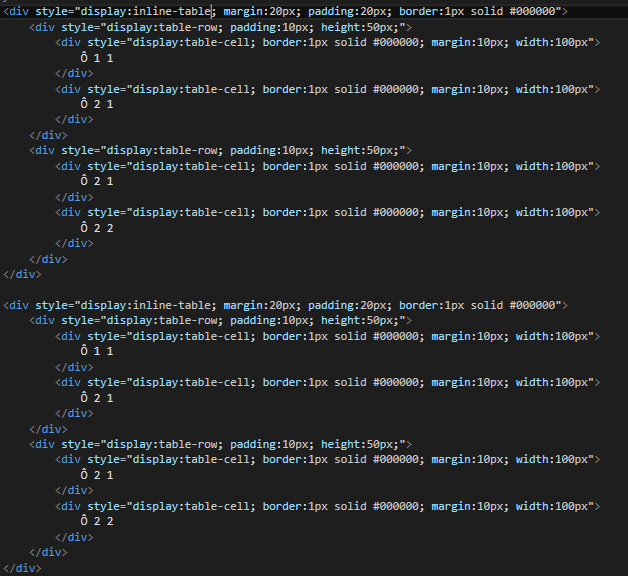
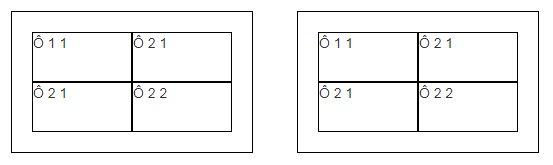
Ví dụ:

Kết quả:

### Inline – table

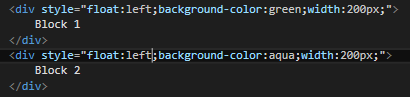
Giống với table nhưng các table sẽ được hiển thị nội tuyến cho phép các bảng nối đuôi nhau.

Ví dụ:

Kết quả:

### Float

Thành phần được hiển thị trôi nổi, cho phép các thành phần khác chiếm vùng không gian của nhau.

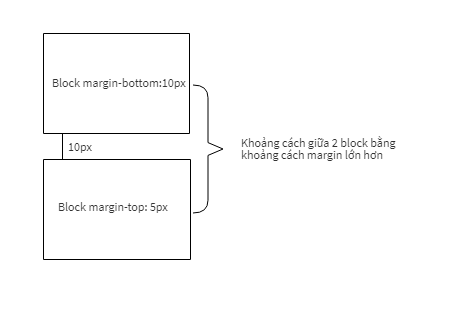
Ví dụ:

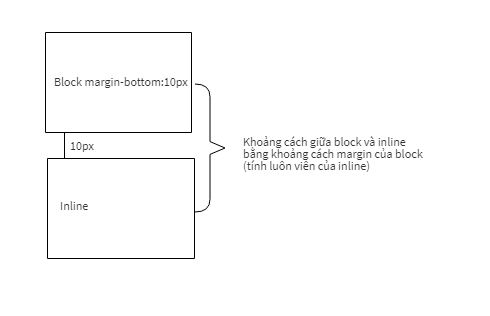
Kết quả:

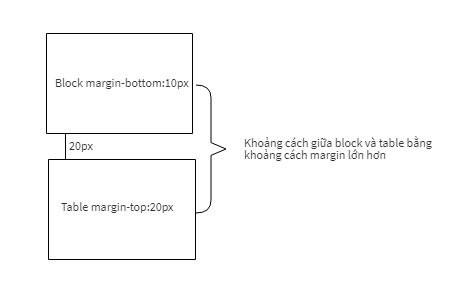
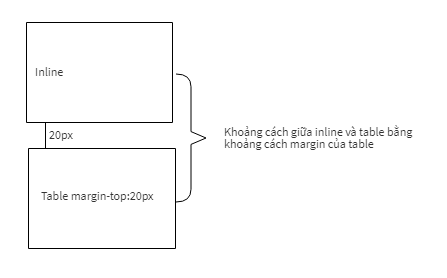
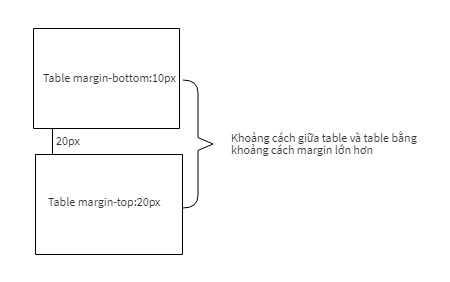
## Margin giữa các display

Có 2 loại margin:

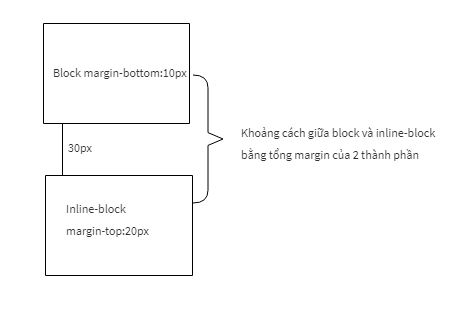
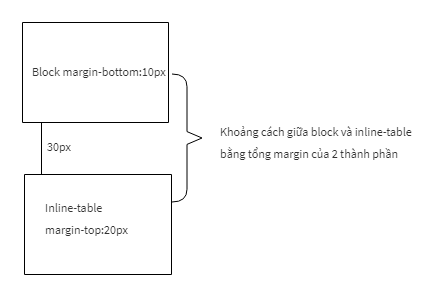
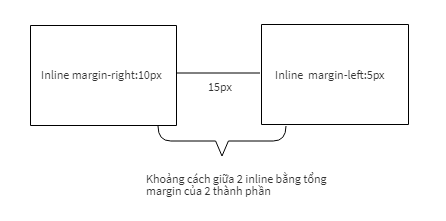
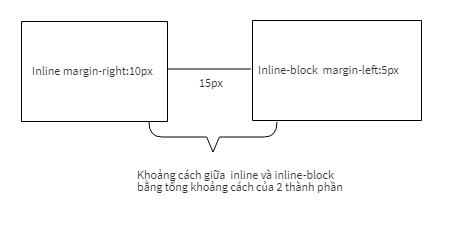
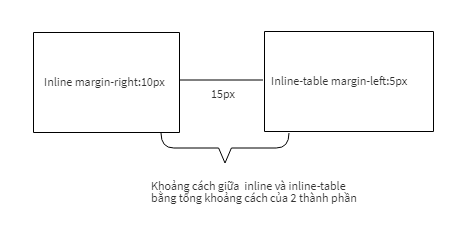
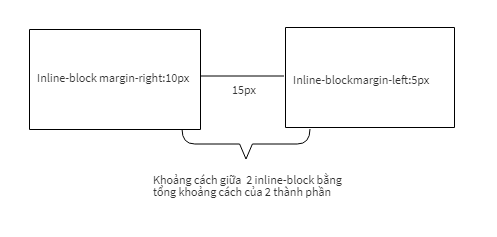
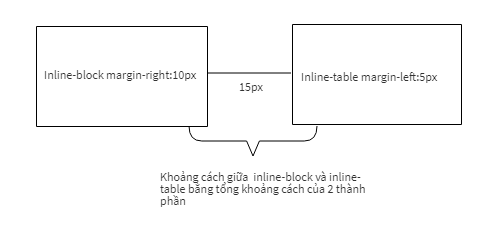
+ Khoảng cách margin giữa các thẻ bằng thẻ có thuộc tính margin lớn hơn. Bao gồm:

* block vs block
* block vs inline



* block vs table
* table vs inline
* table vs table

+ Khoảng cách margin giữa các thẻ bằng tổng margin của các thẻ. Bao gồm:

* block vs inline-block
* block vs inline-table
* inline vs inline
* inline vs inline-block
* inline vs inline table
* inline-block vs inline-block
* inline-block vs inline-table

## Sự khác nhau về thuộc tính width, height và padding giữa các display

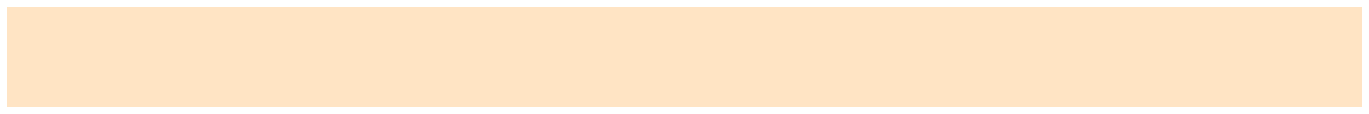
### Block

+ Trường hợp 1: Không đặt giá trị width, height, padding.

Width bằng giá trị mặc định là 100% so với thẻ chứa nó.

Height bằng tổng height của các thành phần bên trong nó và bằng 0 nếu không có thành phần nào bên trong kể cả text.

Ví dụ:

Kết quả:

Thẻ <div> bao ngoài có height = 100px

Thẻ <div> bên trong có height = 0px

Cả 2 thẻ <div> đều có width bằng nhau (1355px theo màng hình hiện tại)

+ Trường hợp 2: Đặt giá trị cho width, height

Ví dụ:

Kết quả:

Thẻ <div> vẫn nhận width = 1000px, height = 200px;

Thẻ <p> bên trong div nhận width = 1200px, height = 100px;

* Kích cở của thẻ cha không phụ thuộc vào kích cỡ của thẻ con
* Thẻ con có thể có kích cỡ lớn hơn cha

+ Trường hợp 3: Chỉ đặt giá trị padding

Ví dụ:

Kết quả:

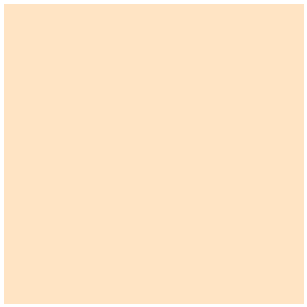
Width = 1355px (theo màng hình hiện tại)

Height = 200px

* Width = 100% so với thẻ cha
* Height = Padding-top + Padding-bottom + Height nội dung (nếu có)

+ Trường hợp 4: Đặt giá trị cho width, height, padding

Ví dụ:

Kết quả:

Độ rộng của thẻ <div> bằng 300px

Độ cao của thẻ <div> bằng 300px

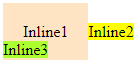
* Độ rộng = width + padding-left + padding-top
* Độ cao = height + padding-bottom + padding-top

### Inline

+ Vì thẻ inline không nhận các giá trị width, height khi đặt nên width và height lấy mặc định bằng tổng kích cở của các thành phần bên trong. Nếu không có thẻ con, mặc định kích cở bằng 0 x 0.

+ Padding ảnh hưởng đến tất cả các phía nhưng chỉ ảnh hướng đến nội dung xung quanh ở 2 bên trái, phái.

Ví dụ:

Kết quả:

### Table

+ Trường hợp 1: Không đặt giá trị width, height, padding

Ví dụ:

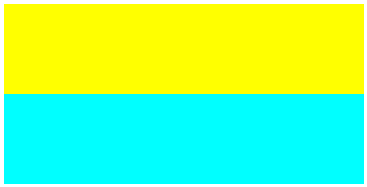
Kết quả:

Thẻ <div> bao ngoài có width = 200px, height = 100px.

* Mặc định lấy theo kích cỡ của thành phần bên trong

+ Trường hợp 2: Đặt giá trị width, height

Ví dụ:

Kết quả:

Thẻ <div> bao ngoài đặt width = 300px nhưng lại có giá trị 400px của thẻ <div> bên trong.

* Width và Height phụ thuộc vào width và height của các thành phần bên trong
* Padding ảnh hưởng đến tất cả các phía.

### Inline-Block

* Width và height mặc định lấy theo kích cỡ của thành phần bên trong

nhưng lại có thể đặt và nhận giá trị như phần tử block.

Padding ảnh hưởng đến tất cả các phía như phần tử block.

* Width và Height không phụ thuộc vào thành phần bên trong.

### Inline-Table

* Width và height mặc định lấy theo kích cỡ của thành phần bên trong, có thể đặt và nhận gái trị như phần tử block.
* Padding ảnh hưởng đến tất cả các phía.
* Width và Height phụ thuộc vào thành phần bên trong.

# Chương 4: Bộ chọn selector và các hàm truy xuất selector trong jQuery

## Selector là gì?

Selector là các chuỗi mẫu được sử dụng để chọn các thành phần mà người dùng muốn tác động.

Nguyên lý viết selector:

Ta có selector: element element … element thứ n

Ví dụ: ‘html body div p’

Nguyên lý từ trái qua phải là cha đến con, các thế hệ cách nhau bởi dấu cách. Theo ví dụ trên, html là cha của body, div, p. Body là cha của div, p. Div là cha của p. P là thành phần có bậc thấp nhất.

Ta có thể thay thế element bằng các selector trên như id, class, tên thẻ. Thành phần ở vị trí cuối cùng là thành phần ta muốn truy xuất.

Việc thay thế nếu không hợp lý sẽ làm ảnh hưởng đến hiệu năng.

Ví dụ: Ta có đoạn html sau:

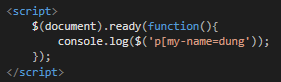
Chọn ra các thẻ p:

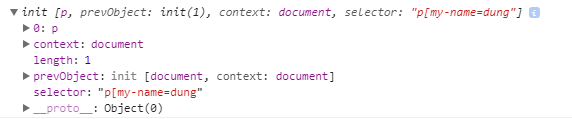
Cách 1: ‘html body div p’ -> đúng nhưng quá chi tiết (overqualifying)

Cách 2: ‘.content p’ -> gọn và tối ưu giúp tìm ra các thẻ p nhanh hơn

* Nên sử dụng class và id ở các vị trí đầu sẽ đạt hiệu năng tốt hơn.

Ngoài ra, ta có thể sử dụng thuộc tính tự định nghĩa.

Ví dụ:

Kết quả:

## Một số selector thường gặp

|  |  |  |
| --- | --- | --- |
| Selector | Ví dụ | Mô tả |
| \* | \* | Lấy tất cả các thành phần |
| #id | #myId | Lấy thành phần có id=’myId’ |
| .class | .content | Lấy tất cả các thành phần có class=’content’ |
| element | div | Lấy tất cả các thành phần div |
| element, element | div, p | Lấy tất cả các thành phần div và p |
| element element | div p | Lấy tất cả các thành phần p nằm trong thành phần div |
| element>element | div > p | Lấy tất cả các thành phần p có cha là div |
| [attribute] | [target] | Lấy tất cả các thành phần có thuộc tính target |
| [attribute=value] | [type=checkbox] | Lấy tất cả các thành phần có thuộc tính type = checkbox |
| :checked or :not:checked | input:checked | Lấy tất cả các thành phần input đã check |
| :disabled or :enabled | input:disabled | Lấy tất cả các thành phần input đã disabled |
| :empty | p:empty | Lấy tất cả các thành phần p không có children (kể cả text) |
| :first-child or :last-child | p:first-child | Lấy thành phần p đầu tiên trong thành phần cha |
| :nth-child | p:nth-child(2) | Lấy thành phần p là con thứ 2 của thành phần cha |
| tag:eq() | p:eq(2) | Lấy thành phần p có index = 2 trong mảng các thành phần p |
| tag:gt() | p:gt(2) | Lấy các thành phần p có index > 2 trong mảng các thành phần p |
| tag:lt() | p:lt(2) | Lấy các thành phần p có index <2 trong mảng các thành phần p |
| tag:even | li:even | Lấy các thành phần li ở vị trí lẻ |
| tag:odd | li:odd | Lấy các thành phần li ở vị trị chẵn |
| tag:first | p:first | Lấy thành phần p ở vị trí đầu tiên |
| tag:last | p:last | Lấy thành phần p ở vị trí cuối cùng |
| tag:first-of-type | li:first-of-type | Lấy thành phần con đầu tiên hoặc duy nhất trong thành phần cha |
| tag:last-of-type | p:last-of-type | Lấy thành phần con cuối cùng hoặc duy nhất trong thành phần cha |
| tag:parent | div:parent | Lấy thành phần có ít nhất một thành phần con (bao gồm cả text) |
| tag:empty | div:empty | Lấy thành phần không có thành phần con (kể cả text) |

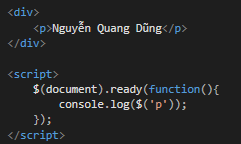
Bảng 2.1: Bảng các selector thường gặp

Ngoài ra còn rất nhiều selector khác dành cho css, javascript, jquery, tham khảo thêm tại [bộ chọn Selector](https://hocwebchuan.com/reference/jquery/jquery_selectors.php).

Các hàm truy xuất selector trong jQuery

### Hàm cơ sở

$() hay jQuery() là hàm cơ sở để lấy thành phần theo bộ selector truyền vào.

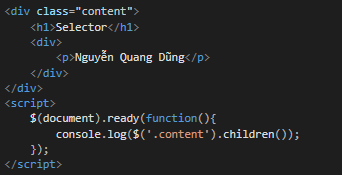
Ví dụ:

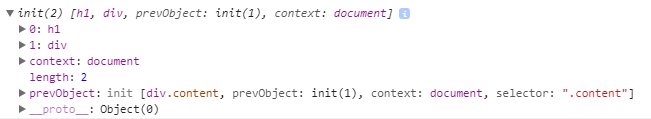
### Các hàm truy xuất đến thành phần con và truy xuất ngược đến thành phần cha

Sau khi lấy được đối tượng, ta có thể truy xuất đến các thành phần con hoặc truy xuất ngược đến các thành phần cha thông qua các hàm sau:

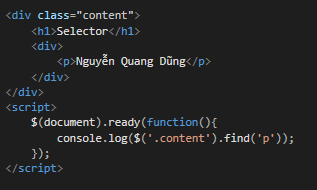
* Hàm children(): Lấy tất cả thành phần con cách cha 1 bậc.

Ví dụ:

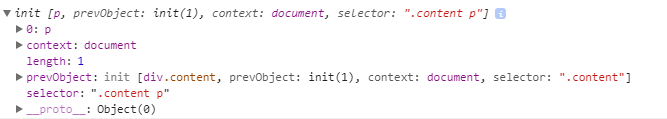
Kết quả: Lấy được 2 thành phần là h1 và div, không lấy thẻ p



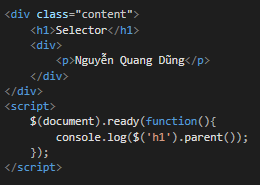
* Hàm find(‘selector’): Lấy tất cả các thành phần con phù hợp với bộ selector truyền vào

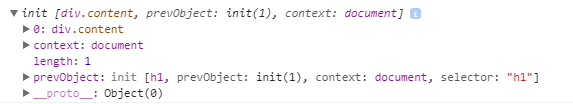
Ví dụ:

Kết quả: Lấy được thành phần p

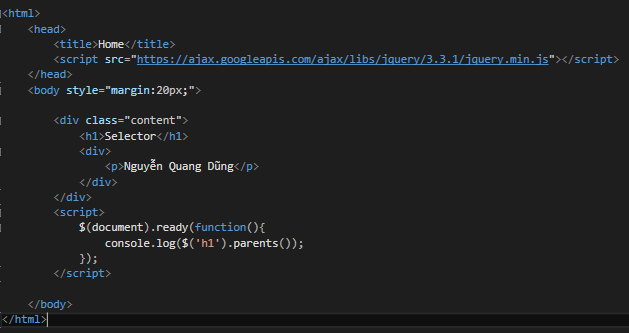


* Hàm parent(): Lấy thành phần cha cách thành phần con 1 bậc.

Ví dụ:

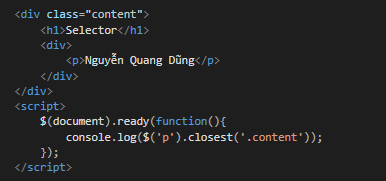
Kết quả: Lấy đc thẻ div gần nhất

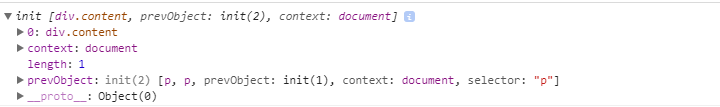
* Hàm parents(): Lấy tất cả các thành phần cha

Ví dụ:

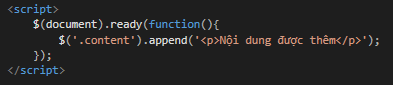
Kết quả: Lấy được tất cả thành phần cha

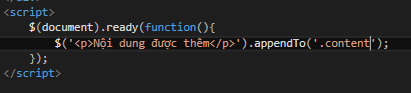
* Hàm closest(‘selector’): Lấy thành phần cha gần nhất phù hợp với selector truyền vào

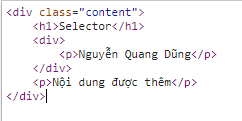
Ví dụ:

Kết quả:

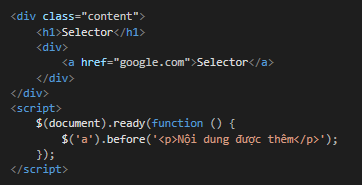
* Hàm appent(): Thành phần được chèn thêm nội dung, nội dung này được sắp xếp ở dưới cùng
* Hàm appentTo(): Chèn thêm nội dung ở dưới cùng vào thành phần đã có

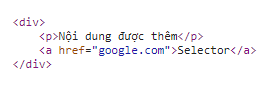
Ví dụ:



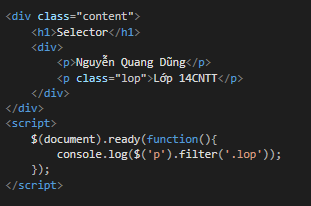
Kết quả:

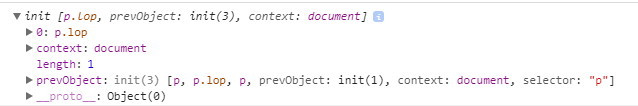
* Hàm before(): Thêm thành phần vào ngay trước thành phần đã có

Ví dụ:

Kết quả:

* Hàm filter(): Lọc các thành phần

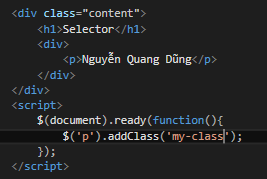
Ví dụ:

Kết quả:

### Các hàm xử lý thuộc tính của thành phần

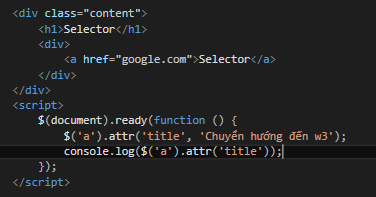
Sau khi lấy được các thành phần, ta có thể thay đổi các giá trị thuộc tính, hoặc thêm, xóa các thuộc tính của các thành phần đó.

* Hàm addClass() / removeClass(): Thêm / Xóa thuộc tính class của thẻ.

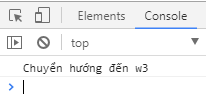
Ví dụ:

Kết quả:

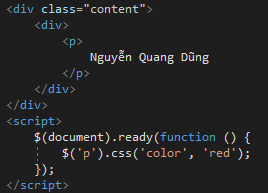
* Hàm attr(‘attribute’): Lấy thuộc tính của thành phần (get)
* Hàm attr(‘attribute’,’value’): Đặt giá trị cho thuộc tính của thành phần (set)

Ví dụ:

Kết quả:



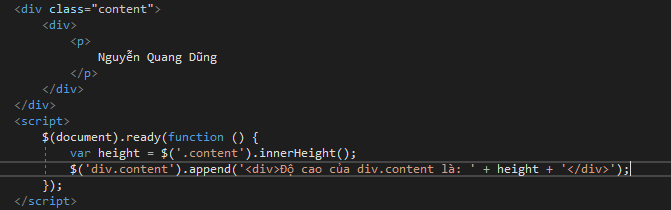
* Hàm css(): Được dùng để thêm một hoặc nhiều style cho thành phần, ngoài ra còn có thể sử dụng .css() để lấy giá trị style của thành phần.

Ví dụ:

Kết quả:

* Hàm innerHeight() / innerWidth(): Lấy độ cao / độ rộng của thành phần (không bao gồm border hay margin). ~ (outerHeight/outerWidth)

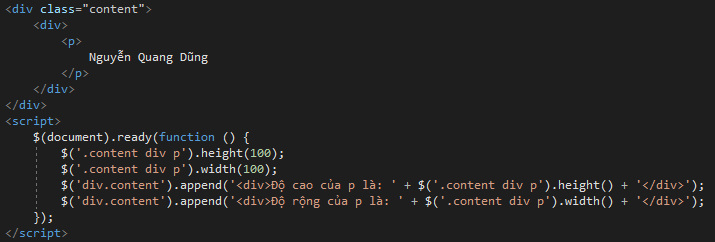
Ví dụ:

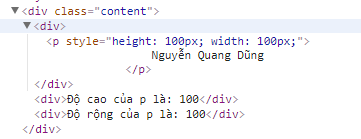


Kết quả:

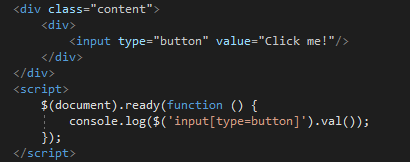
* Hàm height() / width(): get và set giá trị cho thuộc tính height / width của thành phần

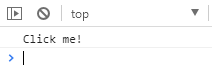
Ví dụ:



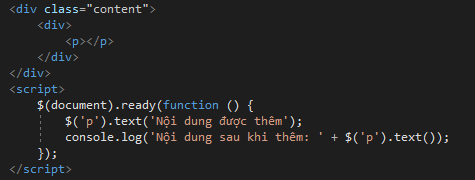
Kết quả:

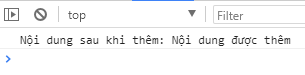
* Hàm val(): get và set giá trị cho thuộc tính value của thành phần

Ví dụ:

Kết quả:

* Hàm text(): get và set nội dung text của thành phần.

Ví du:

Kết quả:

* Hàm html(): get và set nội dung HTML cho thành phần gần giống text()

Ngoài ra còn 1 số các hàm khác, tham khảo thêm tại: [Jquery function(selector)](https://hocwebchuan.com/reference/jquery/jquery_function.php).

## Các hàm bắt sự kiện trong Jquery

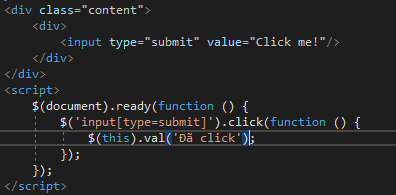
Các sự kiện hỗ trợ

|  |  |
| --- | --- |
| Sự kiện | Mô tả |
| Onchange | Thành phần html bị thay đổi |
| Onclick | Người dùng click vào thành phần html |
| Onmouseover | Người dùng di chuyển chuột qua thành phần html |
| Onmouseout | Người dùng di chuyển chuột ra khỏi thành phần html |
| Onkeydown | Người dùng nhấn 1 phím |
| Onload | Trình duyệt load trong trang |

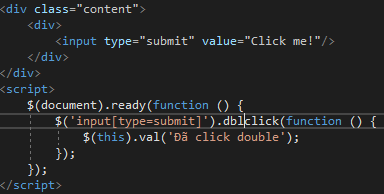
Các hàm xử lý sự kiện thường gặp

|  |  |
| --- | --- |
| Sự kiện | Mô tả |
| Blur() | Xảy ra khi ra khỏi đối tượng |
| Change() | Xảy ra khi giá trị bị thay đổi |
| Click() | Xảy ra khi click vào đối tượng |
| Contentmenu() | Xảy ra khi click vào chuột phải |
| Dblclick() | Xảy ra khi click double |
| Die() | Xóa sự kiện ra khỏi đối tượng |
| Error() | Xảy ra khi xuất hiện lỗi trên đối tượng |
| Focus() | Xảy ra khi focus vào đối tượng |
| Hover() | Xảy ra khi hover chuột vào đối tượng |
| Keydown() | Xảy ra khi bàn phím nhấn xuống |
| Keypress() | Xảy ra khi bàn phím nhấn xuống |
| Keyup() | Xảy ra khi nhả bàn phím |
| On() | Bổ sung sự kiện vào đối tượng |
| Mousedown() | Xảy ra khi nhấn chuột trái xuống |
| Mouseup() | Xảy ra khi nhả chuột trái ra |
| Mouseenter() | Xảy ra khi con trỏ chuột đi vào phạm vi của đối tượng |
| Mouseleave() | Xảy ra khi con trỏ chuột đi ra ngoài phạm vi của đối tượng |
| Mousemove() | Xảy ra khi con trỏ chuột đang di chuyển |
| Mouseover() | Xảy ra một lần duy nhất khi con trỏ chuột bắt đầu đi vào phạm vi đối tượng |
| Mouseout() | Xảy ra một lần duy nhất khi con trỏ chuột đi ra ngoài phạm vi đối tượng |
| Ready() | Khi browser đã load xong |
| Resize() | Xảy ra khi thay đổi kích thước của browser |
| Scroll() | Xảy ra khi kéo thanh cuộn |
| Submit() | Xảy ra khi form được submit |
| Toggle() | Xảy ra khi click vào đối tượng |

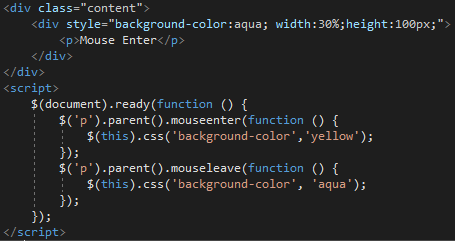
* Hàm click(): Hàm sẽ được thực thi khi người dùng nhấn chuột trái vào phần tử và thả ra.

Ví dụ:

* Hàm dblclick(): Hàm sẽ được thực khi khi người dùng nhấn double chuột trái vào phần tử và thả ra.

Ví dụ:

* Hàm mouseenter(): Hàm sẽ thực thi khi người di chuyển chuột vào phần tử kể cả các phần tử con.
* Hàm mouseleave(): Hàm sẽ thực thi khi người dùng di chuyển chuột ra ngoài phần tử kể cả các phần tử con.

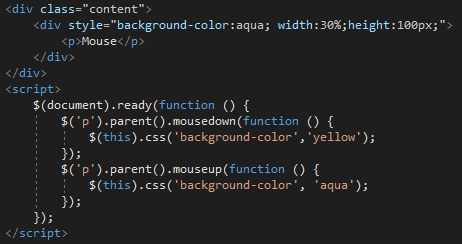
Ví dụ:

Kết quả:

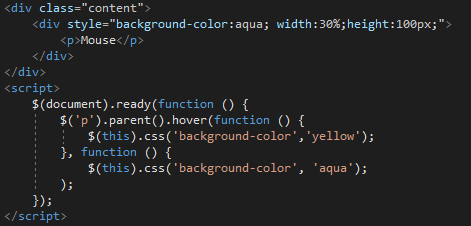
Mouseleave():

Mouseenter():

* Hàm mousedown()/mouseup(): Hàm sẽ thực thi khi người dùng nhấn/thả chuột trái, phải hoặc giữa.

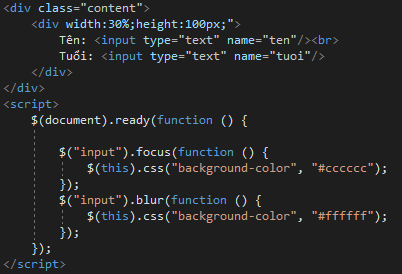
Ví dụ:

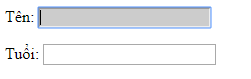
* Hàm hover(): Kết hợp hàm mouseenter() và mouseleave()

Ví dụ:

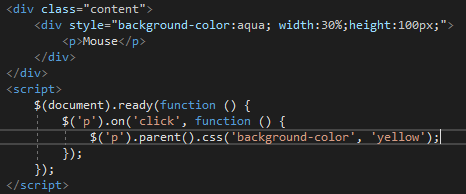
Hàm đầu tiên sẽ xử lý mouse enter và hàm thứ 2 sẽ xử lý mouse leave

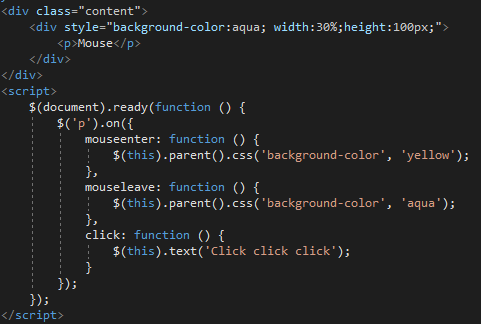
* Hàm focus() / blur(): Hàm thường được dùng để xử lý các form field trong html. Hàm sẽ thực thi khi get focus / lose focus đối tượng.

Ví dụ:

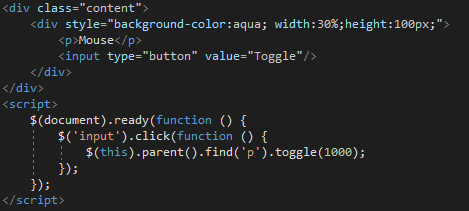
  
Kết quả:

* Hàm on(): Hàm xử lý một hoặc nhiều sự kiện cho thành phần được chọn.

Ví dụ:

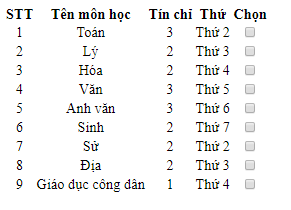
Hoặc nhiều sự kiện:

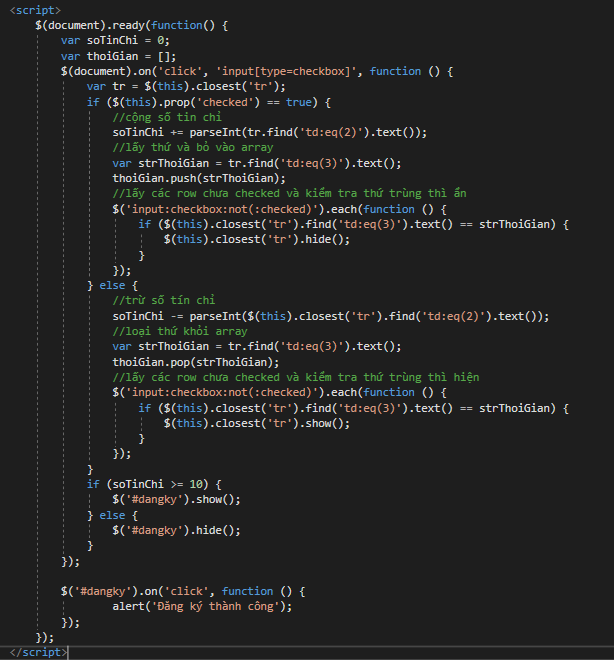
* Hàm toggle(): Xử lý ẩn hiện các phần tử

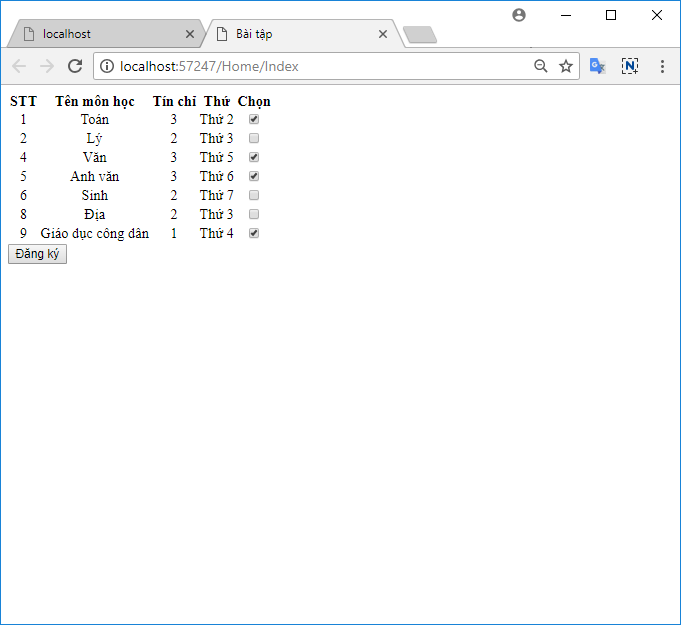
Ví dụ:

## Bài tập

Bài tập đăng ký tín chỉ. Trong bảng danh sách các môn học. Khi chọn đủ trên 10 tín chỉ hiện nút đăng ký, đồng thời khi chọn môn học ẩn các môn học có cùng thứ. Bấm đăng ký hiện thông báo đăng ký thành công.

Demo:

Đoạn script:

Kết quả:

