

GIÁO TRÌNH LẬP TRÌNH ASP.NET MVC 5 TỪ A – Z
TÁC GIẢ: NGUYỄN VĂN PHÚC

Ngày cập nhật: 5/2018

Mục lục

Chương 1: Collections trong C#	3
1. ArrayList	3
2. List<T>	4
3. HashSet<T>	5
4. Hashtable	6
5. SortedList<Tkey,Tvalue>	7
6. Dictionary<Tkey,Tvalue>	8
7. Stack<T>	10
8. Queue<T>	12
Chương 2: Delegate trong C#	14
Chương 3: CSS Display	17
1. Thuộc tính display	17
a. Block	17
b. Inline	18
c. Inline – block	18
d. Table	19
e. Inline – table	20
f. Float	21
2. Margin giữa các display	22
3. Sự khác nhau về thuộc tính width, height và padding giữa các display ...	26
a. Block	26
b. Inline	28
c. Table	29
d. Inline-Block	30
e. Inline-Table	30
Chương 4: CSS Position	31
1. Khái niệm position	31
2. Relative	31
3. Absolute	33

4. Fixed	35
5. Static	36
Chương 5: CSS căn giữa các phần tử	37
1. Theo chiều ngang.....	37
2. Theo chiều dọc.....	38
3. Cả chiều ngang lẫn chiều dọc.....	40
Chương 6: Responsive	42
1. Responsive là gì?	42
2. Responsive Viewport.....	42
3. Responsive Grid View.....	43
4. Responsive Media Queries	47
Chương 7: Bộ chọn selector và các hàm truy xuất selector trong jQuery	50
1. Selector là gì?	50
2. Một số selector thường gặp	51
a. Hàm cơ sở.....	52
b. Các hàm truy xuất đến thành phần con và truy xuất ngược đến thành phần cha	53
c. Hàm truy xuất đến các thành phần cùng cấp.....	57
d. Các hàm xử lý thuộc tính của thành phần	58
3. Các hàm bắt sự kiện trong JQuery	62
4. Bài tập.....	67
Chương 8: Biến và khai báo biến trong Javascript	70
1. Khai báo biến.....	70
2. Gán giá trị cho biến	70
3. Gán kiểu giá trị cho biến	71
4. Biến cục bộ và biến toàn cục	71

Chương 1: Collections trong C#

1. ArrayList

ArrayList là một collection lưu trữ và quản lý một danh sách các đối 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 tại 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 vị trí start đến end trong ArrayList ban đầu
IndexOf(object value)	Trả về vị trí 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

GIAO TRÌNH LẬP TRÌNH ASP.NET MVC 5 TỪ A - Z

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:

```
DienThoai dt1 = new DienThoai("Zenfone 3 Max", "Asus");
DienThoai dt2 = new DienThoai("Zenfone 4 Max", "Asus");

ArrayList arrayList1 = new ArrayList();
arrayList1.Add(dt1);
arrayList1.Add(dt2);

ArrayList arrayList2 = new ArrayList(arrayList1); //sao chép từ arrayList1

foreach (DienThoai dt in arrayList2)
{
    Console.WriteLine(dt.HangDT + " " + dt.TenDT);
}

ArrayList arrayList3 = new ArrayList(1);

Console.WriteLine("Trước khi thêm phần tử");
Console.WriteLine("Số phần tử hiện có " + arrayList3.Count); //Trả về số phần tử hiện có
Console.WriteLine("Sức chứa " + arrayList3.Capacity); //Trả về sức chứa, tự động tăng gấp đôi nếu thêm phần tử vượt sức chứa hiện tại

arrayList3.Add(new DienThoai("X1", "Iphone"));
arrayList3.Add(new DienThoai("X2", "Iphone"));
arrayList3.Insert(1,new DienThoai("X3", "Iphone")); //Chèn vào index 1

Console.WriteLine("Sau khi thêm phần tử");
Console.WriteLine("Số phần tử hiện có " + arrayList3.Count);
Console.WriteLine("Sức chứa " + arrayList3.Capacity);

foreach (DienThoai dt in arrayList3)
{
    Console.WriteLine(dt.HangDT + " " + dt.TenDT);
}
Console.Read();
```

2. 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:

```
string[] str = {"Nguyễn ", "Quang ", "Dũng "};
List<string> list = new List<string>(str);

//Thêm phần tử (Remove tương tự)
list.Add("Nguyễn ");
//Thêm nhiều phần tử (RemoveRange(index, so phần tử))
string[] str2 = { "Quang ", "Dũng " };
list.AddRange(str2);
Console.WriteLine(list.Count);

//Sort() sắp xếp, BinarySearch(value) tìm kiếm nhị phân trả về vị trí của phần tử nếu tìm thấy, ngược lại -1, cần phải sort trước
list.Sort();
foreach(string s in list)
{
    Console.WriteLine(s);
}
Console.WriteLine(list.BinarySearch("Dũng "));

//sao chép sang mảng 1 chiều
string[] copy = new string[list.Count];
list.CopyTo(copy, 0);
Console.WriteLine(copy.Length);
//hoặc
copy = list.ToArray();

Console.WriteLine(list.LastIndexOf("Dũng "));
Console.WriteLine(list.Contains("Dũng "));
Console.ReadKey();
```

3. 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:

```
Console.OutputEncoding = System.Text.Encoding.UTF8;
//HashSet giống với List chỉ lưu giá trị duy nhất thích hợp với dữ liệu lớn
HashSet<string> hs = new HashSet<string>();
hs.Add("1");
hs.Add("1");
hs.Add("1");
hs.Add("2");

foreach(string s in hs)
{
    Console.WriteLine(s);
}
Console.WriteLine("Số phần tử: " + hs.Count);
Console.ReadKey();
```

4. 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 dụng:

- + 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 tại 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:

```
Console.OutputEncoding = System.Text.Encoding.UTF8;
//Hash lưu trữ dữ liệu dưới dạng Key - Value
//truy xuất phần tử thông qua key
KhoaHoc kh1 = new KhoaHoc("J", "Java");
KhoaHoc kh2 = new KhoaHoc("C", "C#");
KhoaHoc kh3 = new KhoaHoc("R", "Ruby");

Hashtable ht = new Hashtable();
ht.Add(kh1.MaKH, kh1.TenKH);
ht.Add(kh2.MaKH, kh2.TenKH);
ht.Add(kh3.MaKH, kh3.TenKH);

//truy xuất phần tử không tồn tại -> null
Console.WriteLine(ht["S"]);

//string str = ht.Keys[0]; -> không duyệt theo index

Console.WriteLine("Hashtable");
foreach(DictionaryEntry de in ht)
{
    Console.WriteLine(de.Key + " " + de.Value);
}

//gán giá trị cho key không tồn tại, tự động thêm vào
ht["S"] = "SQL";

Console.WriteLine("Hashtable sau khi gán giá trị cho key không tồn tại");
foreach (DictionaryEntry de in ht)
{
    Console.WriteLine(de.Key + " " + de.Value);
}
Console.Read();
```

5. 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:

```
//Vừa có thể tìm kiếm theo key vừa có thể tìm kiếm theo vị trí index
Console.OutputEncoding = System.Text.Encoding.UTF8;
SortedList<string,string> sl = new SortedList<string,string>();
sl.Add("hi", "Xin chào");
sl.Add("hello", "Xin chào");
// tự động sắp xếp
foreach (KeyValuePair<string,string> de in sl)
{
    Console.WriteLine(de.Key+" "+de.Value);
}
//danh sách key (tương tự values sl.Values)
Console.WriteLine("Danh sách key");
foreach(string key in sl.Keys)
{
    Console.WriteLine(key);
}
//lấy giá trị theo key
string str = sl["hi"];
//str= sl.Keys[0]; //tìm được theo index
Console.WriteLine("Value " + str);
string str2;
if(sl.TryGetValue("dt2", out str2))
{
    Console.WriteLine(str2);
}
if (sl.ContainsKey("hi"))
{
    Console.WriteLine("Sort List có chứa Key hi");
} else
{
    Console.WriteLine("Sort List không chứa Key hi");
}
sl.Remove("hi"); //Xóa đối tượng có key là hi
sl.RemoveAt(1); //Xóa đối tượng tại vị trí index 1
sl.Clear();//Xóa tất cả các phần tử
Console.ReadKey();
```

6. 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 tại 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:

```
Console.OutputEncoding = System.Text.Encoding.UTF8;
Dictionary<string, string> dtn = new Dictionary<string, string>();
//Thêm phần tử
dtn.Add("Tên 1", "Nội dung sách 1");
dtn.Add("Tên 3", "Nội dung sách 2");
dtn.Add("Tên 2", "Nội dung sách 2");
dtn.Add("Tên 4", "Nội dung sách 4");
Console.WriteLine("Kích thước: " + dtn.Count);
//danh sách key(danh sách value tương tự)
Console.WriteLine("Danh sách key");
foreach(string key in dtn.Keys)
{
    //Hiển thị theo thứ tự thêm, không tự động sắp xếp
    Console.WriteLine(key);
}
// Console.WriteLine(dtn["Tên"]); // truy xuất đến phần tử ko tồn tại -> lỗi khi chạy
//truy xuất phần tử không tồn tại
//Console.WriteLine(dtn["e"]); //lỗi
//kiểm tra key có tồn tại hay không (value tương tự)
Console.WriteLine("Kiểm tra key 'Tên' có trong dtn hay không: ");
Console.WriteLine(dtn.ContainsKey("Tên"));
//kiểm tra tồn tại và trả về value nếu true
string value;
Console.WriteLine(dtn.TryGetValue("Tên 1", out value) + " Value: " + value);
//remove key
dtn.Remove("Tên 2");
Console.WriteLine("Kích thước sau khi remove key 'Tên 2': " + dtn.Count);
dtn.Clear();
Console.WriteLine("Kích thước sau khi clear: " + dtn.Count);
//sử dụng khi muốn tìm kiếm/thêm/xóa nhanh dữ liệu bị ràng buộc
Console.ReadKey();
```

7. Stack<T>

Stack hay ngăn xếp là một tập hợp các đối 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:

```
Console.OutputEncoding = System.Text.Encoding.UTF8;
Stack<string> st = new Stack<string>();
st.Push("Nguyễn");
st.Push("Quang");
st.Push("Dũng");
foreach(string s in st)
{
    Console.WriteLine(s);
}

Console.WriteLine("Độ dài mảng: " + st.Count);

//Kiểm tra tồn tại (true/false)
Console.WriteLine("Tồn tại value 'Dũng': " + st.Contains("Dũng"));

Console.WriteLine("Giá trị peek được: " + st.Peek()); //Peek() lấy giá trị của đối tượng trên cùng nhưng ko loại khỏi stack
Console.WriteLine("Độ dài stack sau khi peek: " + st.Count);

Console.WriteLine("Giá trị pop được: " + st.Pop()); //Pop() lấy giá trị của đối tượng trên cùng và loại khỏi stack
Console.WriteLine("Độ dài stack sau khi pop: " + st.Count);

//Làm việc với mảng
string[] mang = new string[5];
st.CopyTo(mang, 3);

foreach(object o in mang)
{
    Console.WriteLine(o);
}
//Trả về 1 mảng object chứa các phần tử trong stack
string[] mang2 = st.ToArray();
Console.WriteLine("Độ dài mảng 2: " + mang2.Length);

//Xóa tất cả các phần tử
st.Clear();

//-> quản lý phần tử tuần tự cụ thể theo nguyên lý vào sau ra trước -> dễ dàng quay lui(undo) việc thêm xóa phần tử theo thứ tự
Console.ReadKey();
```

8. 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:

```
Console.OutputEncoding = System.Text.Encoding.UTF8;
Queue<string> qu = new Queue<string>();

//thêm người vào hàng chờ thanh toán tiền
qu.Enqueue("Người thứ nhất");
qu.Enqueue("Người thứ hai");
qu.Enqueue("Người thứ ba");

foreach(string s in qu)
{
    Console.WriteLine(s);
}

//trả về phần tử đầu tiên nhưng ko xóa khỏi hàng đợi
Console.WriteLine("Người đứng đầu hàng đợi: " + qu.Peek());

//trả về phần tử đầu tiên và xóa khỏi hàng đợi
Console.WriteLine(qu.Dequeue() + " ra đi. Còn lại: " + qu.Count);

//Xóa tất cả các phần tử
qu.Clear();

//Contains(object value), CopyTo(Array array, int index), ToArray() tương tự Stack

//-> quản lý phần tử tuần tự cụ thể theo nguyên lý vào trước ra trước
Console.ReadKey();
```

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:

```
18 references
class SinhVien
{
    private string name;
    private string maSV;
    0 references
    public SinhVien()
    {
    }

    4 references
    public SinhVien(string name, string maSV)
    {
        this.name = name;
        this.maSV = maSV;
    }

    1 reference
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    1 reference
    public string MaSV
    {
        get { return maSV; }
        set { maSV = value; }
    }

    2 references
    public override string ToString()
    {
        return "Tên: " + name + " Mã Sv: " + maSV;
    }
}
```

Class ThôngTinSinhVien

```
3 references
class ThôngTinSinhVien
{
    private List<SinhVien> listSV;

    0 references
    public List<SinhVien> ListSV
    {
        get { return listSV; }
    }

    1 reference
    public ThôngTinSinhVien(List<SinhVien> listSV)
    {
        this.listSV = listSV;
    }

    1 reference
    public SinhVien SinhVienTheoTen(string name)
    {
        foreach(SinhVien sv in listSV)
        {
            if(sv.Name == name)
            {
                return sv;
            }
        }
        return null;
    }

    0 references
    public SinhVien SinhVienTheoMa(string maSV)
    {
        foreach(SinhVien sv in listSV)
        {
            if(sv.MaSV == maSV)
            {
                return sv;
            }
        }
        return null;
    }
}
```


Thực thi:

```
//khai báo delegate
delegate SinhVien SearchSV(string search);
0 references
static void Main(string[] args)
{
    Console.OutputEncoding = System.Text.Encoding.UTF8;
    List<SinhVien> ls = new List<SinhVien>();
    ls.Add(new SinhVien("Dũng", "Mã 1"));
    ls.Add(new SinhVien("Quang", "Mã 2"));
    ls.Add(new SinhVien("Nguyễn", "Mã 3"));
    ls.Add(new SinhVien("Quang", "Mã 4"));

    ThôngTinSinhVien ttstv = new ThôngTinSinhVien(ls);

    SinhVien sv1 = TìmKiemSinhVien("Quang",ttstv.SinhVienTheoTen);
    Console.WriteLine(sv1.ToString());

    Console.ReadKey();
}
1 reference
static SinhVien TìmKiemSinhVien(string str, SearchSV search)
{
    return search(str);
}
```

Phương thức tìm kiếm SinhVienTheoTen(string name) của class ThôngTinSinhVien 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

1. 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.

a. 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:

```
<body>
  <div style="margin: 10px; padding:10px; background-color:aqua; width:50%;height:200px;">
    Khối Block 1
  </div>
  <div style="margin: 10px; padding:10px; background-color:green; width:40%;height:200px;">
    Khối Block 2
  </div>
</body>
```

Kết quả:



Các thẻ mặc định có display mặc định là block:

<address>	<article>	<aside>	<blockquote>	<canvas>	<dd>
<div>	<dl>	<dt>	<fieldset>	<figcaption>	<figure>
<footer>	<form>	<h1>-<h6>	<header>	<hr>	
<main>	<nav>	<noscript>		<output>	<p>
<pre>	<section>	<table>	<tfoot>		<video>

b. Inline

Thành phần 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:

```
<span style="margin: 10px; padding:10px; background-color:aqua;">
  Inline 1
</span>
<span style="margin: 10px; padding:10px; background-color:green;">
  Inline 2
</span>
```

Kết quả:

Inline 1

Inline 2

Các thẻ có display mặc định inline:

<a>	<abbr>	<acronym>		<bdo>	<big>
 	<button>	<cite>	<code>	<dfn>	
<i>		<input>	<kbd>	<label>	<map>
<object>	<q>	<samp>	<script>	<select>	<small>
		<sub>	<sup>	<textarea>	<time>
<tt>	<var>				

c. Inline – block

Thành phần sẽ hiển thị như một khối, nhưng là một khối nội tuyến. Khi sử dụng inline – block thành phần 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:

```
<span style="display:inline-block; margin: 10px; padding:10px; background-color:aqua; width:200px">
    Inline - Block 1
</span>
<span style="display:inline-block; margin: 10px; padding:10px; background-color:green; height: 100px">
    Inline - Block 2
</span>
```

Kết quả:



d. Table

Thành phần được đối xử như một <table>, tích hợp các display:table-children để hiển 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 height 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:

```
<div style="display:table; margin:20px; padding:20px; border:1px solid #000000">
  <div style="display:table-row; padding:10px; height:50px;">
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 1 1
    </div>
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 2 1
    </div>
  </div>
  <div style="display:table-row; padding:10px; height:50px;">
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 2 1
    </div>
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 2 2
    </div>
  </div>
</div>
```

Kết quả:

Ô 1 1	Ô 2 1
Ô 2 1	Ô 2 2

e. Inline – table

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

Ví dụ:

```
<div style="display:inline-table; margin:20px; padding:20px; border:1px solid #000000">
  <div style="display:table-row; padding:10px; height:50px;">
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 1 1
    </div>
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 2 1
    </div>
  </div>
  <div style="display:table-row; padding:10px; height:50px;">
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 2 1
    </div>
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 2 2
    </div>
  </div>
</div>

<div style="display:inline-table; margin:20px; padding:20px; border:1px solid #000000">
  <div style="display:table-row; padding:10px; height:50px;">
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 1 1
    </div>
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 2 1
    </div>
  </div>
  <div style="display:table-row; padding:10px; height:50px;">
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 2 1
    </div>
    <div style="display:table-cell; border:1px solid #000000; margin:10px; width:100px">
      Ô 2 2
    </div>
  </div>
</div>
```

Kết quả:

Ô 1 1	Ô 2 1
Ô 2 1	Ô 2 2

Ô 1 1	Ô 2 1
Ô 2 1	Ô 2 2

f. Float

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

Ví dụ:

```
<div style="float:left;background-color:green;width:200px;">
  Block 1
</div>
<div style="float:left;background-color:aqua;width:200px;">
  Block 2
</div>
```

Kết quả:



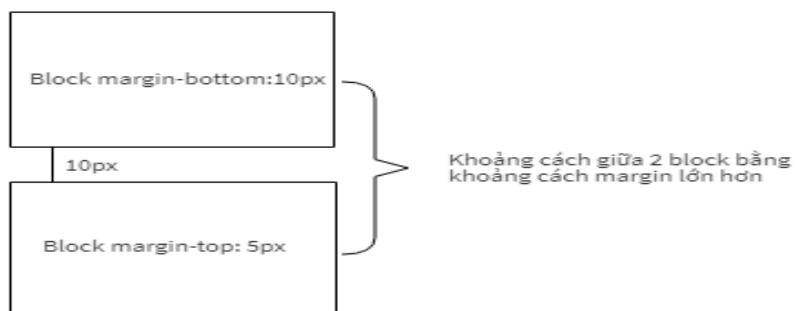
2. Margin giữa các display

Có 2 loại margin:

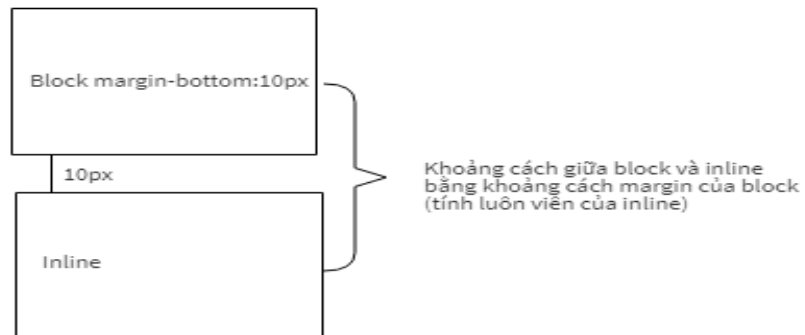
+ Khoảng cách margin giữa các thẻ bằng thẻ có thuộc tính margin lớn hơn.

Bao gồm:

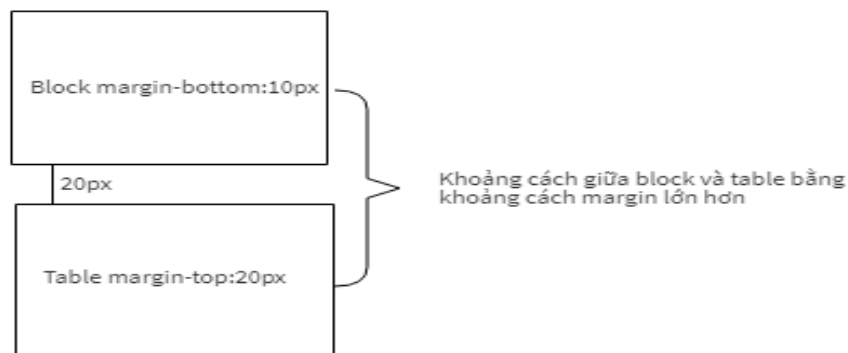
- block vs block



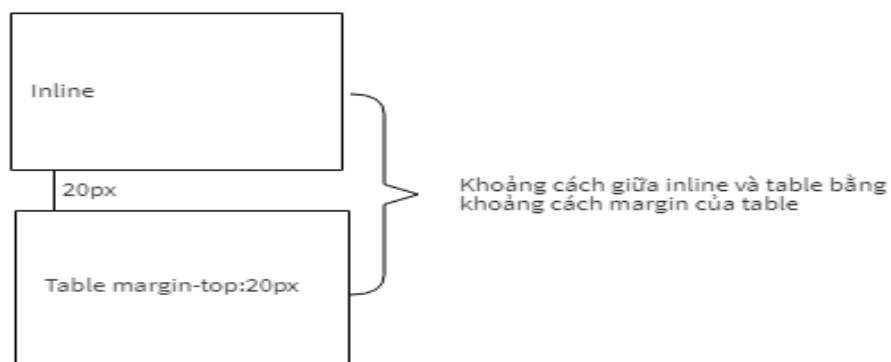
- block vs inline



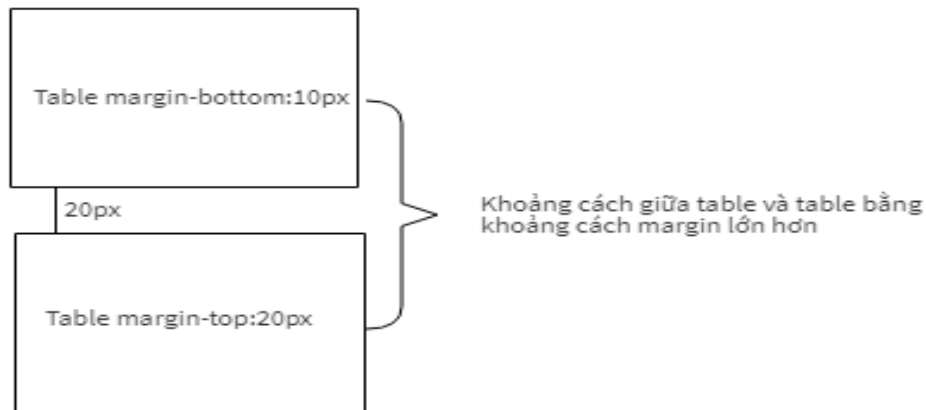
- block vs table



- table vs inline

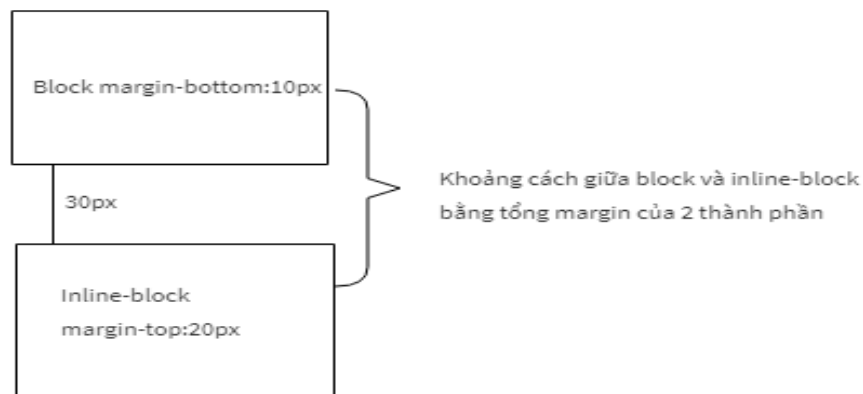


- table vs table

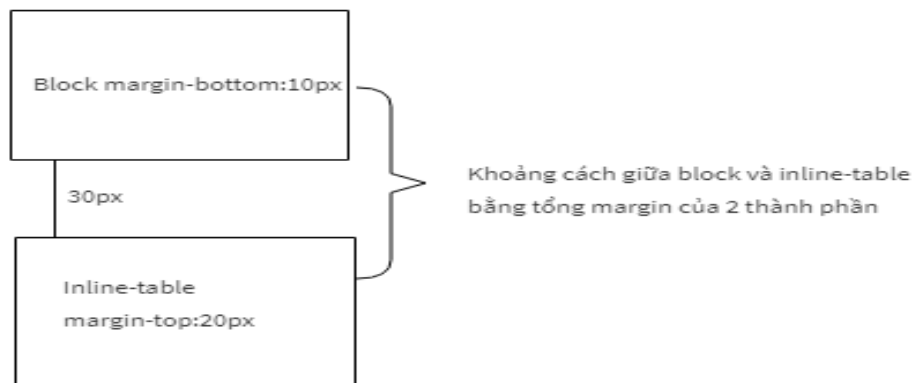


+ Khoảng cách margin giữa các thẻ bằng tổng margin của các thẻ. Bao gồm:

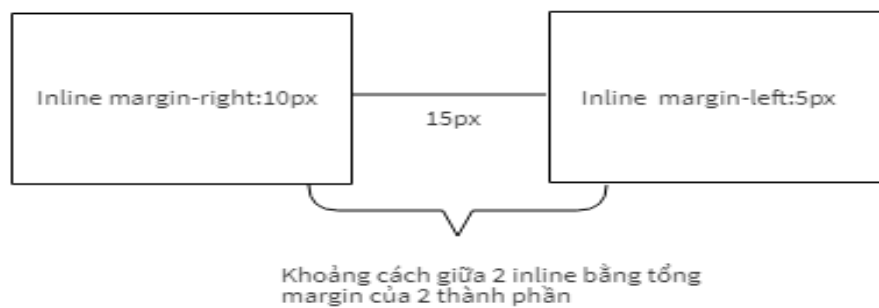
- block vs inline-block



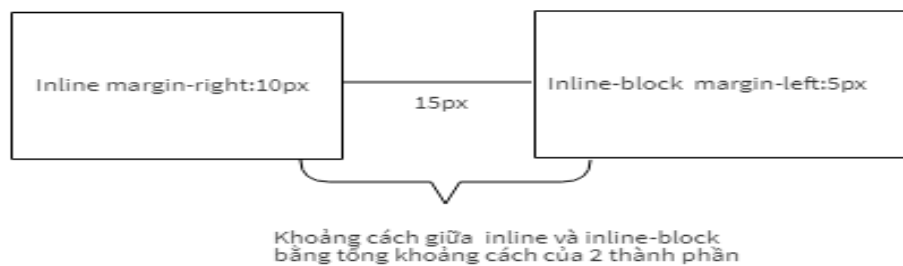
- block vs inline-table



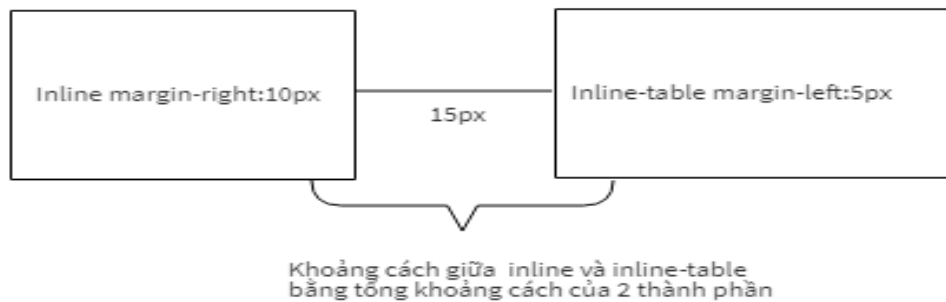
- inline vs inline



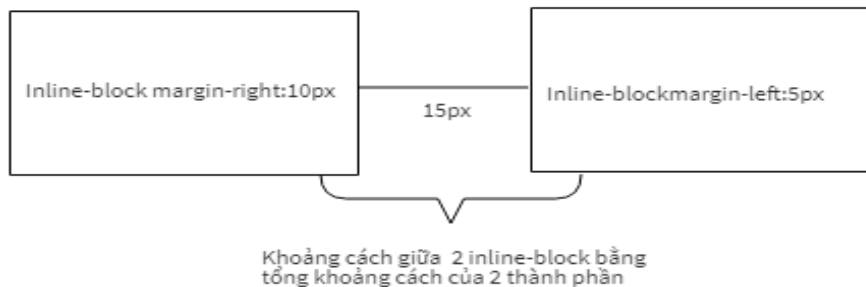
- inline vs inline-block



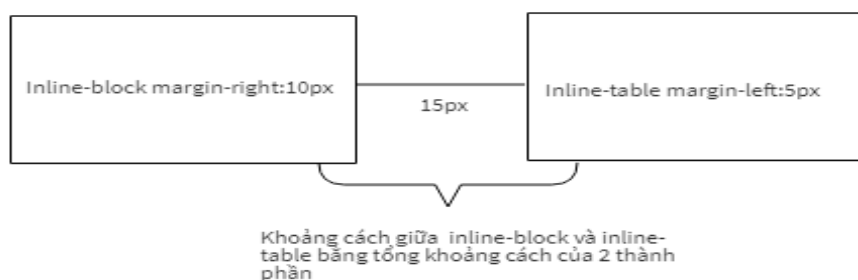
- inline vs inline table



- inline-block vs inline-block



- inline-block vs inline-table



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

a. Block

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

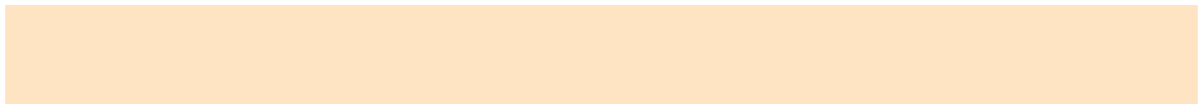
Width bằng giá trị mặc định là 100% so với thẻ chứa nó.

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:

```
<div style="background-color:bisque;">
  <div style="height:100px"></div>
</div>
```

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 màn hình hiện tại)

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

Ví dụ:

```
<div style="background-color:bisque;width:1000px;height:200px;">
  <p style="background-color:yellow;width:1200px;height:100px;"></p>
</div>
```

Kết quả:



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

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

- ➔ Kích cỡ 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:

```
<div style="background-color:bisque;padding:100px;"></div>
```

Kết quả:



Width = 1355px (theo màn hình hiện tại)

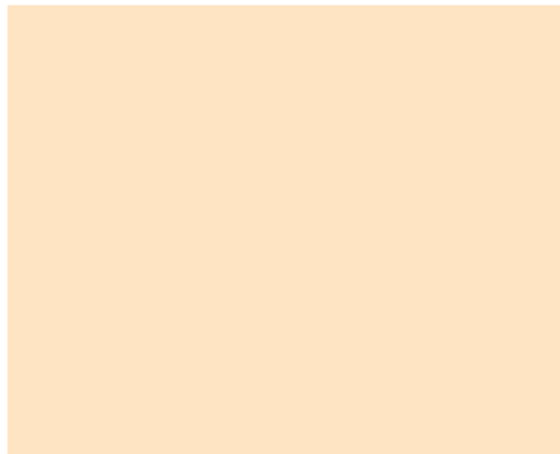
Height = 200px

- ➔ 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:

```
<div style="background-color:bisque;padding:100px; width:100px; height:100px"></div>
```

Kết quả:



Độ rộng của thẻ <div> bằng 300px

Độ cao của thẻ <div> bằng 300px

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

b. Inline

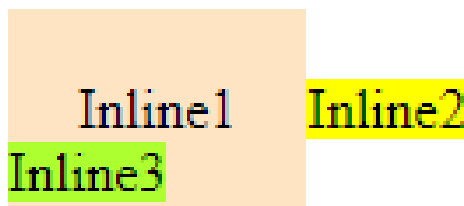
+ Vì thẻ inline không nhận các giá trị width, height khi đặt nên width và height lấy mặc định bằng tổng kích cỡ của các thành phần bên trong. Nếu không có thẻ con, mặc định kích cỡ bằng 0 x 0.

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

Ví dụ:

```
<div style="display:inline;background-color:bisque;padding:20px;">Inline1</div><div style="display:inline;background-color:yellow">Inline2</div>  
<div style="display:inline;background-color:greenyellow">Inline3</div>
```

Kết quả:



c. Table

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

Ví dụ:

```
<div style="display:table;background-color:aqua;">  
  <div style="width:200px;height:100px"></div>  
</div>
```

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:

```
<div style="display:table;background-color:aqua;width:300px;height:200px">  
  <div style="width:400px;height:100px;background-color:yellow"></div>  
</div>
```

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.

d. 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.

e. Inline-Table

- ➔ Width và height mặc định lấy theo kích cỡ của thành phần bên trong, có thể đặt và nhận giá 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: CSS Position

1. Khái niệm position

Position là thuộc tính xác định vị trí tương đối và tuyệt đối cho thành phần, vị trí này phụ thuộc vào các giá trị khai báo của thành phần và thành phần bao ngoài nó. Vị trí có gốc tính từ 4 phía, top, bottom, left, right.

Các giá trị của thuộc tính position: relative, absolute, fixed, static,...

Cấu trúc:

```
//Đứng độc lập
position: relative;

//Kèm vị trí:
position: relative;
right: value;
left: value;
```

Các vị trí có thể sử dụng số âm hay dương đều được.

2. Relative

Relative định vị tương đối cho thành phần, khi sử dụng thuộc tính này các thành phần sẽ định vị theo mốc vùng hiển thị của chính nó, vị trí này không phụ thuộc vào vùng không gian, vùng không gian vẫn được giữ như ban đầu.

Ví dụ:

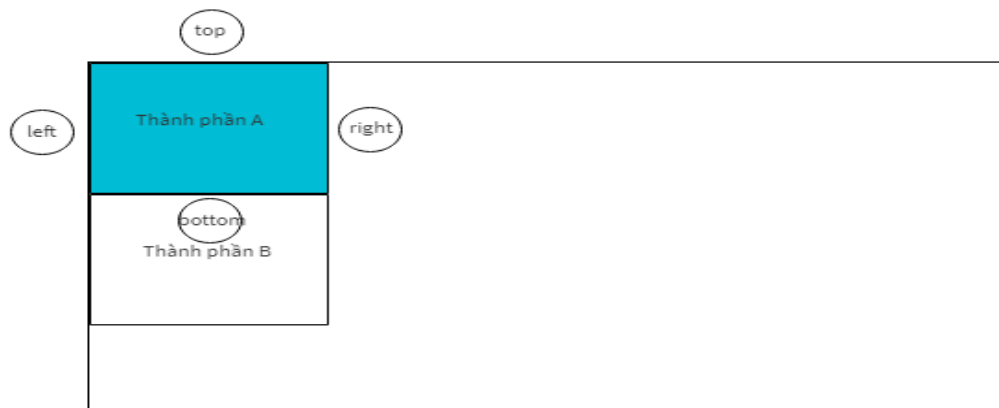
```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
  <style>
    * {
      margin: 0;
      padding: 0;
    }

    .pA{
      position: relative;
    }
  </style>
</head>
<body>
  <div style="border:1px solid black; width:500px;height:300px;margin:100px auto;">
    <p class="pA" style="background-color:aqua; width:100px;height:100px">Thành phần A.</p>
    <p class="pB" style="background-color:yellow; width:100px;height:100px">Thành phần B.</p>
  </div>
</body>
</html>
```


Kết quả:



Nhìn vào hình, ta thấy chưa có gì xảy ra nhưng thực chất ta đã định mốc cho thành phần A.



Ta thêm thuộc tính định vị trí để dễ nhìn:

```
.pA{  
  position: relative;  
  left: 10px;  
  top: 20px;  
}
```

Kết quả:



Vì phần không gian của A vẫn giữ như cũ nên vị trí của thành phần B không thay đổi. Phần hiển thị của A thay đổi vị trí chồng lên B. Ta có thể di chuyển ở bất kỳ vị trí nào tùy vào các thuộc tính định vị trí so với gốc.

3. Absolute

Absolute định vị tuyệt đối cho thành phần. khi sử dụng vùng không gian sẽ biến mất trở thành vùng trống. Thành phần sẽ thoát khỏi vùng không gian của nó.

Trường hợp 1: Nếu không có các thuộc tính định vị trí thì thành phần lấy gốc theo vùng hiển thị với giá trị mặc định top:0 và left:0.

```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
  <style>
    * {
      margin: 0;
      padding: 0;
    }

    .pA{
      position: absolute;
    }
  </style>
</head>
<body>
  <div style="border:1px solid black; width:500px;height:300px;margin:100px auto;">
    <p class="pA" style="background-color:aqua; width:100px;height:100px">Thành phần A.</p>
    <p class="pB" style="background-color:yellow; width:200px;height:100px">Thành phần B.</p>
  </div>
</body>
</html>
```

Kết quả:



Vùng không gian của thành phần A được giải phóng, nên thành phần B chiếm chỗ của thành phần A. Còn thành phần A nằm ở vị trí ban đầu.

Trường hợp 2: Sử dụng thuộc tính định vị trí thì thành phần sẽ lấy gốc theo vùng không gian của trình duyệt

```
.pA{
  position: absolute;
  top: 50px;
  left: 500px;
}
```



Trường hợp 3: Nếu thành phần bao ngoài được định vị tương đối (relative) thì gốc của thành phần absolute được tính theo vùng hiển thị của thành phần bao ngoài.

```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
  <style>
    * {
      margin: 0;
      padding: 0;
    }
    div {
      position: relative;
    }
    .pA {
      position: absolute;
      top: 50px;
      left: 50px;
    }
  </style>
</head>
<body>
  <div style="border: 1px solid black; width: 500px; height: 300px; margin: 10px auto;">
    <p class="pA" style="background-color: aqua; width: 100px; height: 100px;">Thành phần A.</p>
    <p class="pB" style="background-color: yellow; width: 200px; height: 100px;">Thành phần B.</p>
  </div>
</body>
</html>
```

Kết quả:



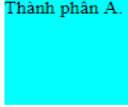
4. Fixed

Fixed định vị tuyệt đối cho thành phần với gốc vị trí tính theo vùng hiển thị của trình duyệt.

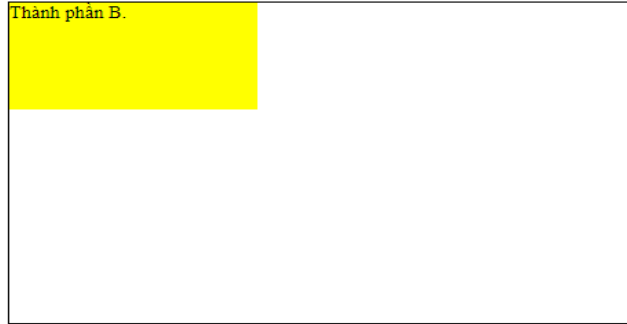
```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
  <style>
    * {
      margin: 0;
      padding: 0;
    }
    div {
      position: relative;
    }
    .pA {
      position: fixed;
      top: 50px;
      left: 50px;
    }
  </style>
</head>
<body>
  <div style="border: 1px solid black; width: 500px; height: 300px; margin: 10px auto;">
    <p class="pA" style="background-color: aqua; width: 100px; height: 100px;">Thành phần A.</p>
    <p class="pB" style="background-color: yellow; width: 200px; height: 100px;">Thành phần B.</p>
  </div>
</body>
</html>
```

Kết quả:

Thành phần A.



Thành phần B.



5. Static

Đây là dạng mặc định của thành phần, thường được sử dụng để phục hồi dạng position của thành phần. Khi sử dụng dạng này, nó sẽ không bị ảnh hưởng bởi các thuộc tính định vị trí như top, bottom, left, right. Nó định vị theo luồng thông thường của trang.

Chương 5: CSS căn giữa các phần tử

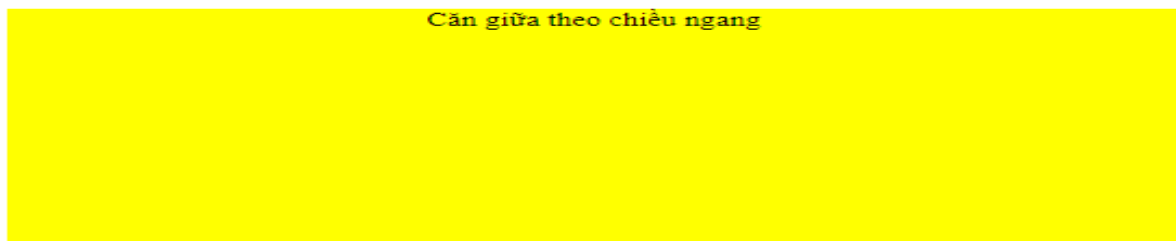
1. Theo chiều ngang

Trường hợp 1: Căn giữa các phần tử inline và inline-* theo chiều ngang bên trong phần tử block với text-align.

```
<style>
    .content{
        text-align:center;
        background-color:yellow;
        width:500px;
        height:200px;
    }
</style>
```

```
<div class="content">
    <span>Căn giữa theo chiều ngang</span>
</div>
```

Kết quả:



Trường hợp 2: Căn giữa phần tử block bằng việc gán giá trị margin-left và margin-right bằng auto.

```
<style>
    .content{
        background-color:yellow;
        width:500px;
        height:200px;
        margin: 0 auto;
    }
</style>
```

Kết quả:

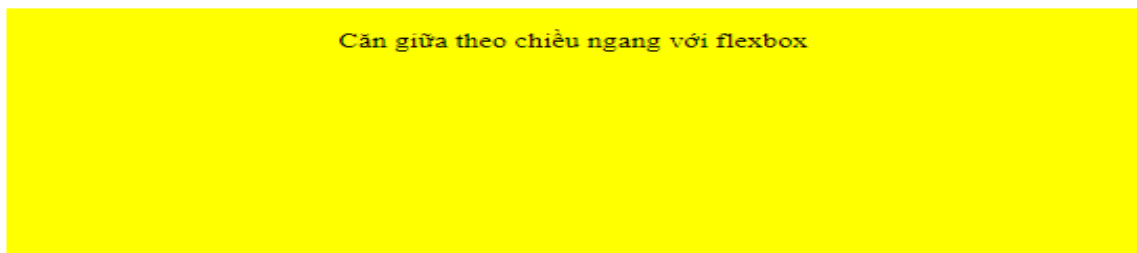


Trường hợp 3: Căn giữa các phần tử block trong block với flexbox

```
<style>
    .content{
        display:flex;
        justify-content:center;
        background-color:yellow;
        height:200px;
    }
</style>
```

```
<body>
    <div class="content">
        <p>Căn giữa theo chiều ngang với flexbox</p>
    </div>
</body>
```

Kết quả:

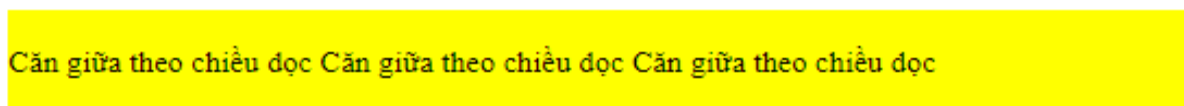


2. Theo chiều dọc

Trường hợp 1: Căn giữa các phần theo chiều dọc với padding-top và padding-bottom bằng nhau.

```
<style>
    .content{
        padding: 20px 0;
        background-color:yellow;
    }
</style>
```

Kết quả:



Trường hợp 2: Căn giữa theo chiều dọc khi không biết height của phần tử.

```
<style>
    .content {
        position: relative;
        background-color: yellow;
        height: 200px;
        width: 200px;
    }
    .content .center {
        position: absolute;
        top: 50%;
        transform: translateY(-50%);
        background-color: red;
    }
</style>
```

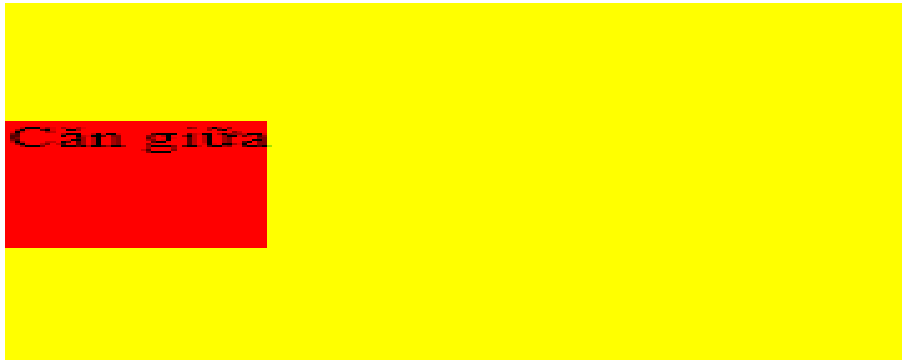
Kết quả:



Trường hợp 3: Căn giữa theo chiều dọc khi biết height. Margin-top = âm 1 nửa giá trị height.

```
<style>
    .content {
        position: relative;
        background-color: yellow;
        height: 200px;
        width: 200px;
    }
    .content .center {
        position: absolute;
        top: 50%;
        height: 70px;
        margin-top: -35px;
        background-color: red;
    }
</style>
```


Kết quả:



3. Cả chiều ngang lẫn chiều dọc

Trường hợp 1: Sử dụng margin âm có giá trị tuyệt đối bằng một nửa width và height nếu width và height cố định hoặc dùng transform: translate(-50%, -50%) nếu không biết width và height. Sau đó đặt position kiểu absolute với top và left 50%.

```
<style>
  .content {
    position: relative;
    background-color: yellow;
    height: 300px;
    width: 300px;
  }
  .content .center {
    position: absolute;
    top: 50%;
    left: 50%;
    width: 200px;
    height: 100px;
    margin: -50px 0 0 -100px;
    background-color: red;
  }
</style>
```

Kết quả:



Trường hợp 2: Sử dụng flexbox

```
<style>
  .content {
    background-color: yellow;
    height: 500px;
    width: 500px;
    display: flex;
    justify-content: center;
    align-items: center;
  }
  .content .center {
    width: 200px;
    height: 100px;
    background-color: red;
  }
</style>
```

Kết quả:



Chương 6: Responsive

1. Responsive là gì?

Responsive dùng để thiết kế trang web cho dễ nhìn trên mọi kích thước hiển thị của trình duyệt. Hay nói cách khác là nó thay đổi kích thước, ẩn, thu nhỏ phóng to hoặc di chuyển nội dung để trông đẹp hơn trên bất kỳ màn hình nào như máy tính, điện thoại, máy tính bảng...

2. Responsive Viewport

Khai báo meta viewport

```
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>
```

Thẻ meta viewport thiết lập cho trình duyệt hiển thị tương ứng với kích thước màn hình. Thẻ trên sẽ thiết lập trình duyệt hiển thị cố định và tương ứng trên tất cả các thiết bị dựa vào chiều rộng của thiết bị (device-width) và không cho phép người dùng phóng to (initial-scale=1).

Ngoài ra thẻ meta viewport còn có các giá trị sau:

- + width: thiết lập chiều rộng của viewport
- + device-width: thiết lập chiều rộng cố định của thiết bị
- + height: thiết lập chiều cao của viewport
- + device-height: thiết lập chiều cao cố định của thiết bị
- + initial-scale: thiết lập mức phóng to ban đầu, giá trị 1 nghĩa là không phóng to và khi giá trị được thiết lập thì người dùng không thể phóng to vì nó đã được cố định.
- + minimum-scale: Mức phóng to tối thiểu của thiết bị với trình duyệt
- + maximum-scale: Mức phóng to tối đa của thiết bị với trình duyệt
- + user-scalable: Cho phép người dùng phóng to, giá trị yes hoặc no.

Khi không sử dụng viewport và sử dụng viewport.

Viewport



Không sử dụng viewport

Viewport



Sử dụng viewport

3. Responsive Grid View

Responsive Grid View dùng để chia trang web thành các cột, tổng chiều rộng là 100% và sẽ thu nhỏ hoặc phóng to khi thay đổi kích thước cửa sổ trình duyệt. Thường thì sẽ chia thành 12 cột.

Triển khai:

Bước 1: Đảm bảo các phần tử html có thuộc tính `box-sizing: border-box`. Điều này để đảm bảo padding và border được bao gồm trong width và height của các phần tử.

```
*{  
  box-sizing: border-box;  
}
```

Bước 2: Chia cột và điều chỉnh kích cỡ. Kích cỡ các cột nên sử dụng tương đối, mỗi cột sẽ có độ rộng = $100/\text{số cột}$ (%) kích cỡ màn hình.

Ví dụ: Chia thành 4 cột.

```
.col-1{
    width:25%;
}
.col-2{
    width:50%;
}
.col-3{
    width:75%;
}
.col-4{
    width:100%;
}
```

Bước 3: Mỗi hàng nên được bao bọc trong một <div>. Số lượng cột trong một hàng nên thêm tối đa.

```
<div class="row">
    <div class="col-1">
        ...
    </div>
    <div class="col-3">
        ...
    </div>
</div>
```

Ta sử dụng thuộc tính float để làm cho các cột thoát khỏi luồng của trang trở thành trôi nổi, để lấp vùng trống. Và mỗi hàng thì dùng thuộc tính clear để phân biệt hàng này với hàng khác và không cho các phần tử khác chèn vào nếu hàng có chỗ trống.

```
.row::after{
    display:table;
    clear:both;
}
[class*="col-"]{
    float:left;
}
```

Demo: HTML

```
<body>
  <div>
    <header class="col-4"><h1>Thiết kế website với Responsive</h1></header>
    <div class="main">
      <div class="row">
        <div class="col-1">
          <ul class="menu">
            <li>
              Menu 1
            </li>
            <li>
              Menu 2
            </li>
            <li>
              Menu 3
            </li>
            <li>
              Menu 4
            </li>
          </ul>
        </div>
        <div class="col-3">
          <div class="content">
            <p>Nội dung</p>
          </div>
        </div>
      </div>
    </div>
    <footer class="col-4"><h2>Copyright &copy; 2018 by NQD</h2></footer>
  </div>
</body>
```

CSS:

```
<style>
*{
    box-sizing: border-box;
}
header {
    background-color: darkgray;
    color: white;
    text-align: center;
    height: 100px;
}
header h1 {
    vertical-align: middle;
}
footer {
    background-color: darkgray;
    color: white;
    text-align: center;
    height: 100px;
}
footer h2 {
    vertical-align: middle;
}

.col-1{
    width:25%;
}
.col-2{
    width:50%;
}
.col-3{
    width:75%;
}
.col-4{
    width:100%;
}

.row::after{
    display:table;
    clear:both;
}
[class*="col-"]{
    float:left;
}
.menu {
    list-style:none;
    background-color:darkgrey;
    color:white;
}
.content {
    margin: 20px;
    color: white;
    background-color: darkgrey;
}
</style>
```

Kết quả:



4. Responsive Media Queries

Media query là 1 kỹ thuật css trong CSS3. Nó sử dụng cú pháp @media để thêm khối lệnh css nếu điều kiện đúng. Thường dùng để tạo điểm ngắt, thay đổi cách bố trí dựa trên hướng của trình duyệt, ẩn thành phần hoặc thay đổi một số thuộc tính của thành phần.

Cú pháp @media dùng để áp dụng các style khác nhau cho các phương tiện khác nhau. Ví dụ như width và height của viewport, width và height của thiết bị, hướng của thiết bị (nằm ngang hay dọc) và độ phân giải.

Cú pháp:

```
@media not | only mediatype and (media feature and | or | not media feature){  
    // CSS  
}
```

Các loại media types:

Tên	Mô tả
All	Mặc định. Dùng cho tất cả thiết bị
Print	Sử dụng cho máy in
Screen	Sử dụng cho màn hình máy tính, máy tính bảng, điện thoại...
Speech	Sử dụng cho các trình đọc màn hình, máy chiếu...

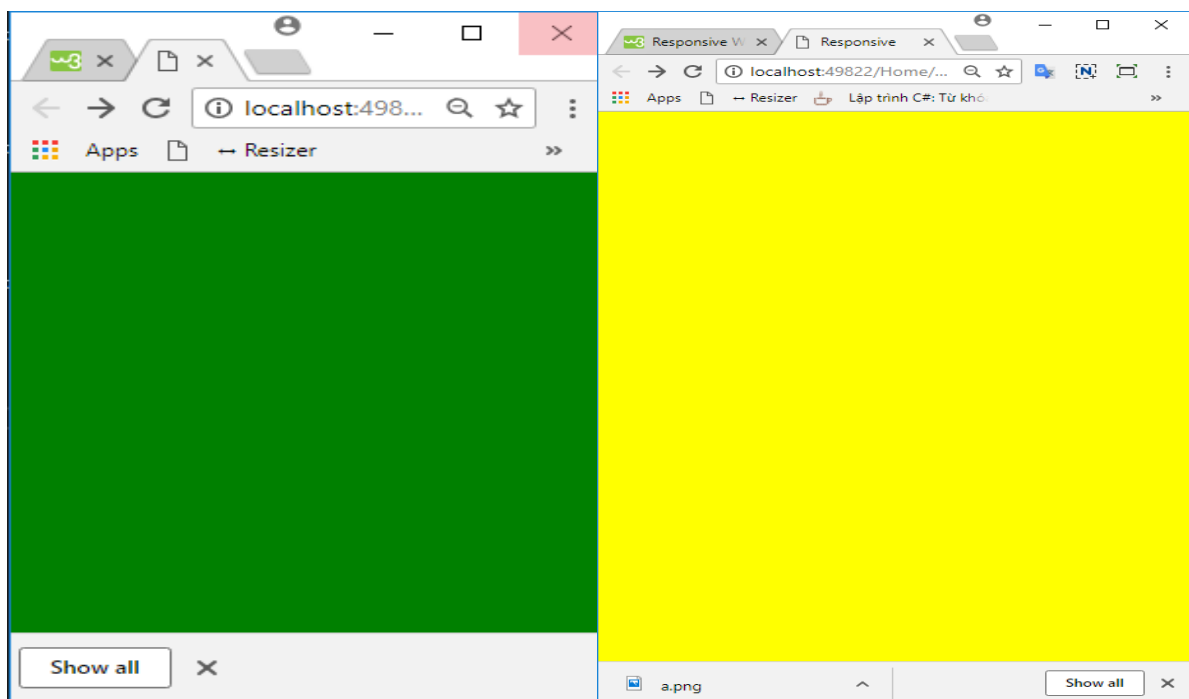
Các loại media features:

Tên	Mô tả
Width	Độ rộng của viewport
Height	Độ cao của viewport
Max-width	Chiều rộng tối đa của vùng hiển thị
Max-height	Chiều cao tối đa của vùng hiển thị
Min-width	Chiều rộng tối thiểu của vùng hiển thị
Min-height	Chiều cao tối thiểu của vùng hiển thị
Orientation	Hướng của viewport
Resolution	Độ phân giải của thiết bị

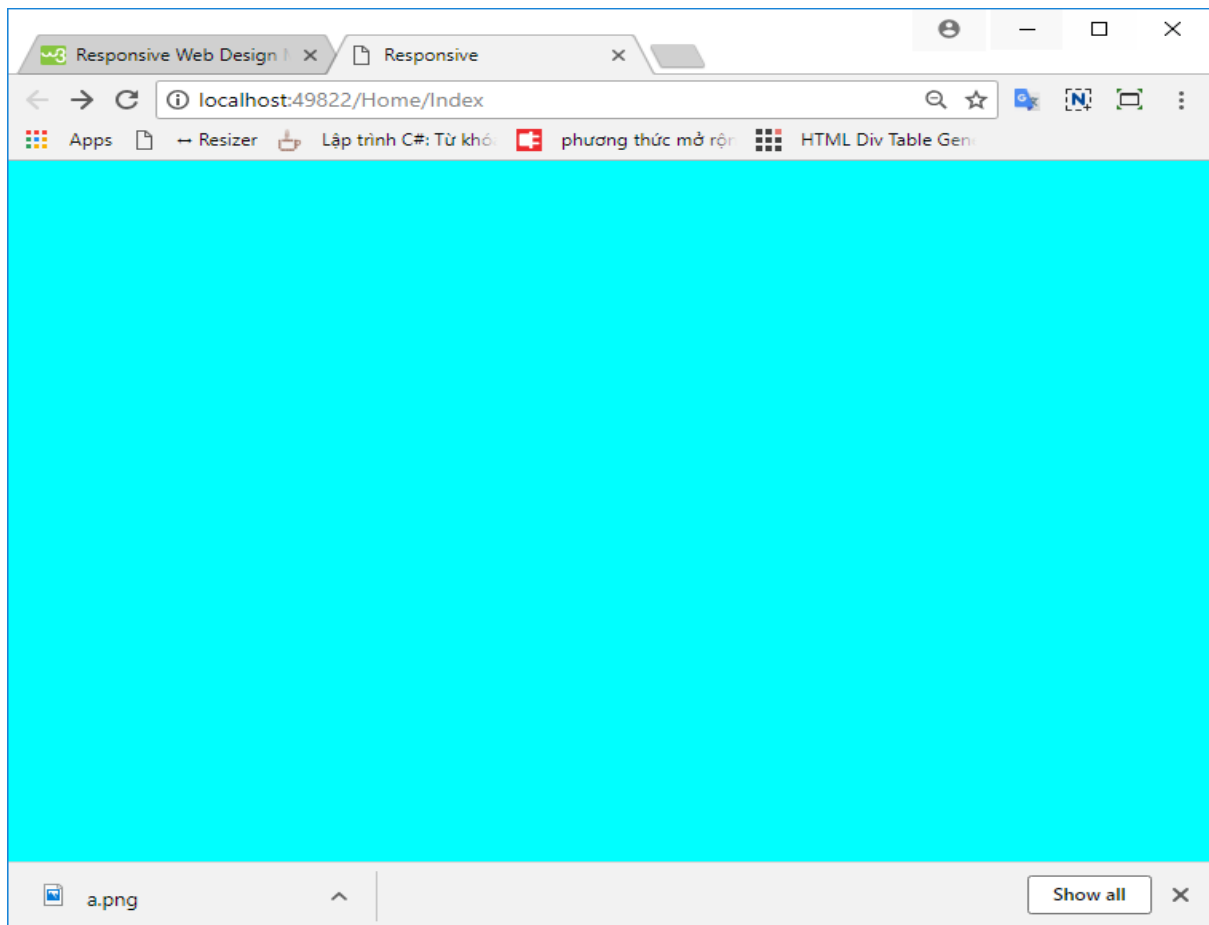
Demo:

```
@media screen and (min-width:1180px) {  
    body {  
        background-color: white;  
    }  
}  
@media screen and (min-width:1024px) and (max-width:1179px) {  
    body {  
        background-color: darkgray;  
    }  
}  
@media screen and (min-width:768px) and (max-width:1023px) {  
    body {  
        background-color: aqua;  
    }  
}  
@media screen and (min-width:480px) and (max-width:767px) {  
    body {  
        background-color: yellow;  
    }  
}  
@media screen and (min-width:320px) and (max-width:479px) {  
    body {  
        background-color: green;  
    }  
}
```

Kết quả: Background đổi màu khi phóng to, thu nhỏ màn hình.



GIÁO TRÌNH LẬP TRÌNH ASP.NET MVC 5 TỪ A - Z



Chương 7: Bộ chọn selector và các hàm truy xuất selector trong jQuery

1. 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: selector 1 selector 2 ... selector n

Selector 1 có thể là class, id hoặc tag name.

Selector 2 có thể là class, id hay tag name nhưng phải là con của selector 1.

Selector n cũng có thể là class, id hay tag name nhưng phải là con của các selector trước nó.

Từ trái qua phải là cha đến con, các thế hệ cách nhau bởi dấu cách.

Vị trí selector cuối cùng là những thành phần mà người dùng muốn tác động.

Ví dụ: 'html body div p'

Theo ví dụ trên, selector 2: 'body' là con của selector 1: 'html', selector 3: 'div' là con của selector 1 và 2, và selector cuối cùng 'p' là con của các selector còn lại. Nghĩa là lấy các thẻ <p> được bao bởi thẻ <div> nằm trong thẻ <body> và <body> nằm trong <html>

Việc thay thế các vị trí selector nếu không hợp lý sẽ làm ảnh hưởng đến hiệu năng.

Ví dụ: Ta có đoạn html sau:

```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>selector</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</head>
<body>
  <div class="content">
    <p id="myId">Tối ưu cách chọn</p>
    <p>Selector</p>
    <ul>
      <li>Link 1</li>
      <li>Link 2</li>
      <li>Link 3</li>
    </ul>
  </div>
</body>
</html>
```

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 của thẻ hoặc thuộc tính tự định nghĩa để lấy các thành phần muốn tác động.

Ví dụ:

```
<div>
  <p my-name="dung">Nguyễn Quang Dũng</p>
</div>

<script>
  $(document).ready(function(){
    console.log($('p[my-name=dung]'));
  });
</script>
```

Kết quả:

```
▼ init [p, prevObject: init(1), context: document, selector: "p[my-name=dung"] ⓘ
  ► 0: p
  ► context: document
    length: 1
  ► prevObject: init [document, context: document]
    selector: "p[my-name=dung]"
  ► __proto__: Object(0)
```

2. 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 trí 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](#).

Các hàm truy xuất selector trong jQuery

a. Hàm cơ sở

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

Ví dụ:

```
<div>
  <p>Nguyễn Quang Dũng</p>
</div>

<script>
  $(document).ready(function(){
    console.log($('p'));
  });
</script>
```

- b. 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:

```
<div class="content">
  <h1>Selector</h1>
  <div>
    <p>Nguyễn Quang Dũng</p>
  </div>
</div>
<script>
  $(document).ready(function(){
    console.log($('.content').children());
  });
</script>
```

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

```
▼ init(2) [h1, div, prevObject: init(1), context: document] ⓘ
  ► 0: h1
  ► 1: div
  ► context: document
  ► length: 2
  ► prevObject: init [div.content, prevObject: init(1), context: document, selector: ".content"]
  ► __proto__: Object(0)
```

- Hàm `find('selector')`: Lấy tất cả các thành phần con phù hợp với bộ `selector` truyền vào.

Ví dụ:

```
<div class="content">
  <h1>Selector</h1>
  <div>
    <p>Nguyễn Quang Dũng</p>
  </div>
</div>
<script>
  $(document).ready(function(){
    console.log($('.content').find('p'));
  });
</script>
```

Kết quả: Lấy được thành phần p

```
▼ init [p, prevObject: init(1), context: document, selector: ".content p"] ⓘ
  ▶ 0: p
  ▶ context: document
  ▶ length: 1
  ▶ prevObject: init [div.content, prevObject: init(1), context: document, selector: ".content"]
  ▶ selector: ".content p"
  ▶ __proto__: Object(0)
```

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

Ví dụ:

```
<div class="content">
  <h1>Selector</h1>
  <div>
    <p>Nguyễn Quang Dũng</p>
  </div>
</div>
<script>
  $(document).ready(function(){
    console.log($('h1').parent());
  });
</script>
```

Kết quả: Lấy đc thẻ div gần nhất

```
▼ init [div.content, prevObject: init(1), context: document] ⓘ
  ▶ 0: div.content
  ▶ context: document
  ▶ length: 1
  ▶ prevObject: init [h1, prevObject: init(1), context: document, selector: "h1"]
  ▶ __proto__: Object(0)
```

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

Ví dụ:

```
<html>
  <head>
    <title>Home</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  </head>
  <body style="margin:20px;">

    <div class="content">
      <h1>Selector</h1>
      <div>
        <p>Nguyễn Quang Dũng</p>
      </div>
    </div>
    <script>
      $(document).ready(function(){
        console.log($('h1').parents());
      });
    </script>

  </body>
</html>
```

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:

```
<div class="content">
  <h1>Selector</h1>
  <div>
    <p>Nguyễn Quang Dũng</p>
  </div>
</div>
<script>
  $(document).ready(function(){
    console.log($('p').closest('.content'));
  });
</script>
```

Kết quả:

```
▼ init [div.content, prevObject: init(2), context: document] ⓘ
  ► 0: div.content
  ► context: document
  ► length: 1
  ► prevObject: init(2) [p, p, prevObject: init(1), context: document, selector: "p"]
  ► __proto__: Object(0)
length: 4
► prevObject: init [h1, prevObject: init(1), context: document, selector: "h1"]
► __proto__: Object(0)
```


- Hàm `append()`: Thành phần được chèn thêm nội dung, nội dung này được sắp xếp ở dưới cùng
- Hàm `appendTo()`: Chèn thêm nội dung ở dưới cùng vào thành phần đã có

Ví dụ:

```
<script>
    $(document).ready(function(){
        $('.content').append('<p>Nội dung được thêm</p>');
    });
</script>
```

```
<script>
    $(document).ready(function(){
        $('<p>Nội dung được thêm</p>').appendTo('.content');
    });
</script>
```

Kết quả:

```
<div class="content">
    <h1>Selector</h1>
    <div>
        <p>Nguyễn Quang Dũng</p>
    </div>
    <p>Nội dung được thêm</p>
</div>
```

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

Ví dụ:

```
<div class="content">
    <h1>Selector</h1>
    <div>
        <a href="google.com">Selector</a>
    </div>
</div>
<script>
    $(document).ready(function () {
        $('a').before('<p>Nội dung được thêm</p>');
    });
</script>
```

Kết quả:

```
<div>
  <p>Nội dung được thêm</p>
  <a href="google.com">Selector</a>
</div>
```

- Hàm filter(): Lọc các thành phần

Ví dụ:

```
<div class="content">
  <h1>Selector</h1>
  <div>
    <p>Nguyễn Quang Dũng</p>
    <p class="lop">Lớp 14CNTT</p>
  </div>
</div>
<script>
  $(document).ready(function(){
    console.log($('p').filter('.lop'));
  });
</script>
```

Kết quả:

```
▼ init [p.lop, prevObject: init(3), context: document] ⓘ
  ▶ 0: p.lop
  ▶ context: document
    length: 1
  ▶ prevObject: init(3) [p, p.lop, p, prevObject: init(1), context: document, selector: "p"]
  ▶ __proto__: Object(0)
```

- c. Hàm truy xuất đến các thành phần cùng cấp
 - Hàm siblings() / siblings('selector'): Lấy các thành phần cùng cấp nhưng không chọn chính nó.

Ví dụ:

```
<div class="content">
  <ul>
    <li class="link-1">Link 1</li>
    <li>Link 2</li>
    <li>Link 3</li>
  </ul>
</div>
<script>
  $(document).ready(function () {
    ...
    $('.link-1').siblings().css('background', 'green');
  });
</script>
```

Kết quả:

- [Link 1](#)
- [Link 2](#)
- [Link 3](#)

➔ Chỉ lấy các thẻ li cùng cấp khác, không lấy chính mình.

d. 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:

```
<div class="content">
  <h1>Selector</h1>
  <div>
    <p>Nguyễn Quang Dũng</p>
  </div>
</div>
<script>
  $(document).ready(function(){
    $('p').addClass('my-class');
  });
</script>
```

Kết quả:

```
<div>
  <p class="my-class">Nguyễn Quang Dũng</p>
</div>
```

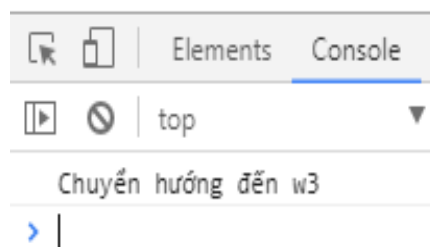
- Hàm `attr('attribute')`: Lấy thuộc tính của thành phần (get)
- Hàm `attr('attribute','value')`: Đặt giá trị cho thuộc tính của thành phần (set)

Ví dụ:

```
<div class="content">
  <h1>Selector</h1>
  <div>
    <a href="google.com">Selector</a>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('a').attr('title', 'Chuyển hướng đến w3');
    console.log($('a').attr('title'));
  });
</script>
```

Kết quả:

```
<div>
  <a href="google.com" title="Chuyển hướng đến w3">Selector</a>
</div>
```



- 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:

```
<div class="content">
  <div>
    <p>
      Nguyễn Quang Dũng
    </p>
  </div>
</div>
<script>
  $(document).ready(function () {
    ...
    $('p').css('color', 'red');
  });
</script>
```

Kết quả:

```
<p style="color: red;">
  Nguyễn Quang Dũng
</p>
```

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

Ví dụ:

```
<div class="content">
  <div>
    <p>
      Nguyễn Quang Dũng
    </p>
  </div>
</div>
<script>
  $(document).ready(function () {
    var height = $('.content').innerHeight();
    $('.div.content').append('<div>Độ cao của div.content là: ' + height + '</div>');
  });
</script>
```

Kết quả:

```
<div class="content">
  <div>...</div>
  <div>Độ cao của div.content là: 18</div>
</div>
```

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

Ví dụ:

```
<div class="content">
  <div>
    <p>
      Nguyễn Quang Dũng
    </p>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('.content div p').height(100);
    $('.content div p').width(100);
    $('.div.content').append('<div>Độ cao của p là: ' + $('.content div p').height() + '</div>');
    $('.div.content').append('<div>Độ rộng của p là: ' + $('.content div p').width() + '</div>');
  });
</script>
```

Kết quả:

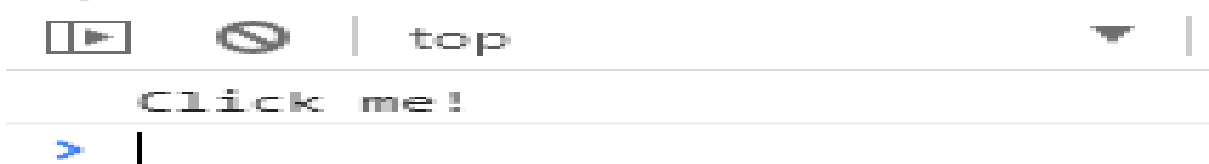
```
<div class="content">
  <div>
    <p style="height: 100px; width: 100px;">
      Nguyễn Quang Dũng
    </p>
  </div>
  <div>Độ cao của p là: 100</div>
  <div>Độ rộng của p là: 100</div>
</div>
```

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

Ví dụ:

```
<div class="content">
  <div>
    <input type="button" value="Click me!"/>
  </div>
</div>
<script>
  $(document).ready(function () {
    console.log($('input[type=button]').val());
  });
</script>
```

Kết quả:



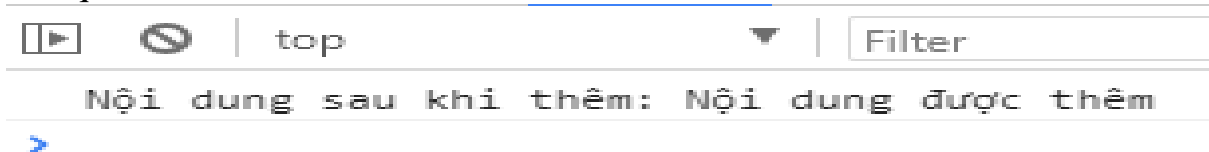
The screenshot shows a web browser interface. At the top, there are navigation icons (back, forward, stop) and a search bar. Below the navigation bar, there is a button labeled "Click me!". Below the button, there is a console window with a blue prompt character and a vertical line, indicating that the console is active and ready to display output.

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

Ví dụ:

```
<div class="content">
  <div>
    <p></p>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('p').text('Nội dung được thêm');
    console.log('Nội dung sau khi thêm: ' + $('p').text());
  });
</script>
```

Kết quả:



The screenshot shows a web browser interface. At the top, there are navigation icons (back, forward, stop) and a search bar. Below the navigation bar, there is a paragraph of text that reads "Nội dung được thêm". Below the paragraph, there is a console window with a blue prompt character and a vertical line, indicating that the console is active and ready to display output.

- 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\(selector\)](#).

3. 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:

```
<div class="content">
  <div>
    <input type="submit" value="Click me!"/>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('input[type=submit]').click(function () {
      $(this).val('Đã click');
    });
  });
</script>
```

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

Ví dụ:

```
<div class="content">
  <div>
    <input type="submit" value="Click me!"/>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('input[type=submit]').dblclick(function () {
      $(this).val('Đã click double');
    });
  });
</script>
```

- 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:

```
<div class="content">
  <div style="background-color:aqua; width:30%;height:100px;">
    <p>Mouse Enter</p>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('p').parent().mouseenter(function () {
      $(this).css('background-color','yellow');
    });
    $('p').parent().mouseleave(function () {
      $(this).css('background-color', 'aqua');
    });
  });
</script>
```

Kết quả:

Mouseleave():

Mouse

Mouse

Mouseenter():

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

Ví dụ:

```
<div class="content">
  <div style="background-color:aqua; width:30%;height:100px;">
    <p>Mouse</p>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('p').parent().mousedown(function () {
      $(this).css('background-color','yellow');
    });
    $('p').parent().mouseup(function () {
      $(this).css('background-color', 'aqua');
    });
  });
</script>
```

- Hàm `hover()`: Kết hợp hàm `mouseenter()` và `mouseleave()`

Ví dụ:

```
<div class="content">
  <div style="background-color:aqua; width:30%;height:100px;">
    <p>Mouse</p>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('p').parent().hover(function () {
      $(this).css('background-color','yellow');
    }, function () {
      $(this).css('background-color','aqua');
    });
  });
</script>
```

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:

```
<div class="content">
  <div width:30%;height:100px;">
    Tên: <input type="text" name="ten"/><br>
    Tuổi: <input type="text" name="tuoi"/>
  </div>
</div>
<script>
  $(document).ready(function () {
    $("input").focus(function () {
      $(this).css("background-color", "#cccccc");
    });
    $("input").blur(function () {
      $(this).css("background-color", "#ffffff");
    });
  });
</script>
```

Kết quả:

Tên:

Tuổi:

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

Ví dụ:

```
<div class="content">
  <div style="background-color:aqua; width:30%;height:100px;">
    <p>Mouse</p>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('p').on('click', function () {
      $('p').parent().css('background-color', 'yellow');
    });
  });
</script>
```

Hoặc nhiều sự kiện:

```
<div class="content">
  <div style="background-color:aqua; width:30%;height:100px;">
    <p>Mouse</p>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('p').on({
      mouseenter: function () {
        $(this).parent().css('background-color', 'yellow');
      },
      mouseleave: function () {
        $(this).parent().css('background-color', 'aqua');
      },
      click: function () {
        $(this).text('Click click click');
      }
    });
  });
</script>
```

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

Ví dụ:

```
<div class="content">
  <div style="background-color:aqua; width:30%;height:100px;">
    <p>Mouse</p>
    <input type="button" value="Toggle"/>
  </div>
</div>
<script>
  $(document).ready(function () {
    $('input').click(function () {
      $(this).parent().find('p').toggle(1000);
    });
  });
</script>
```

4. 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:

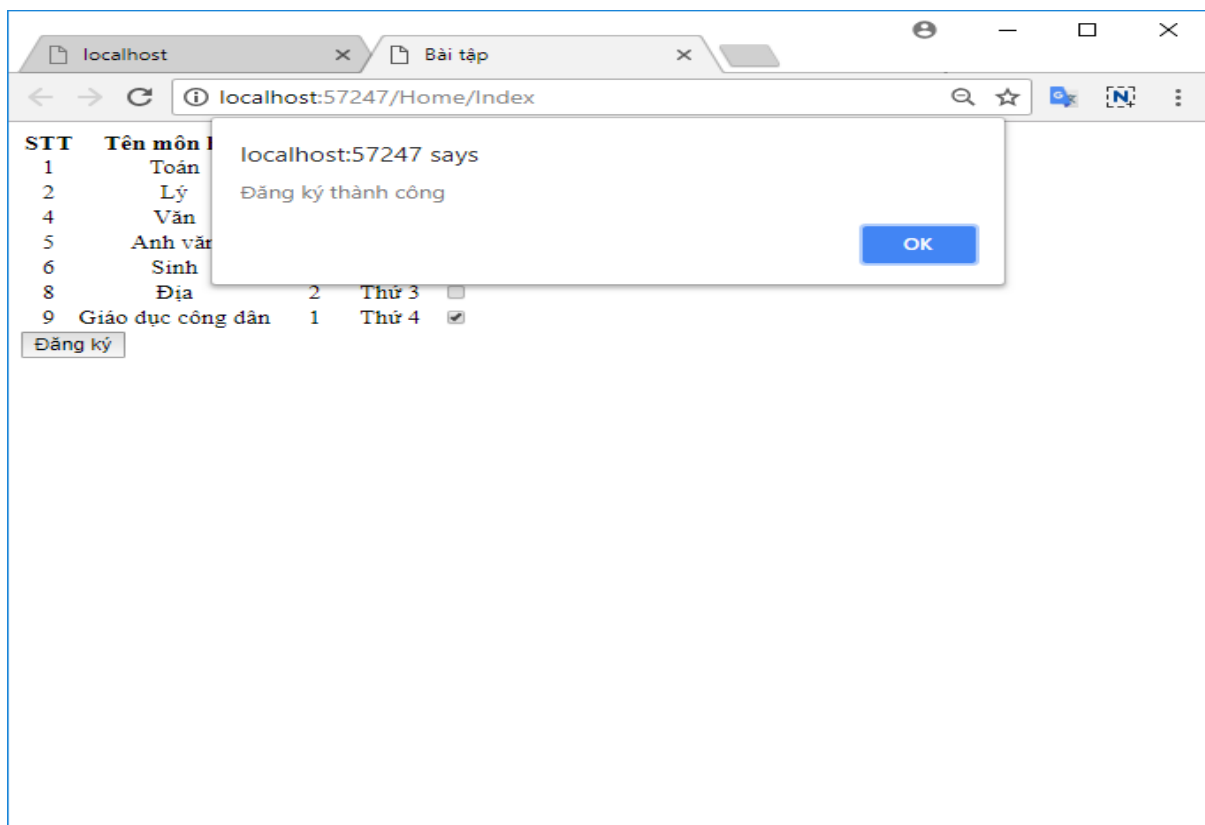
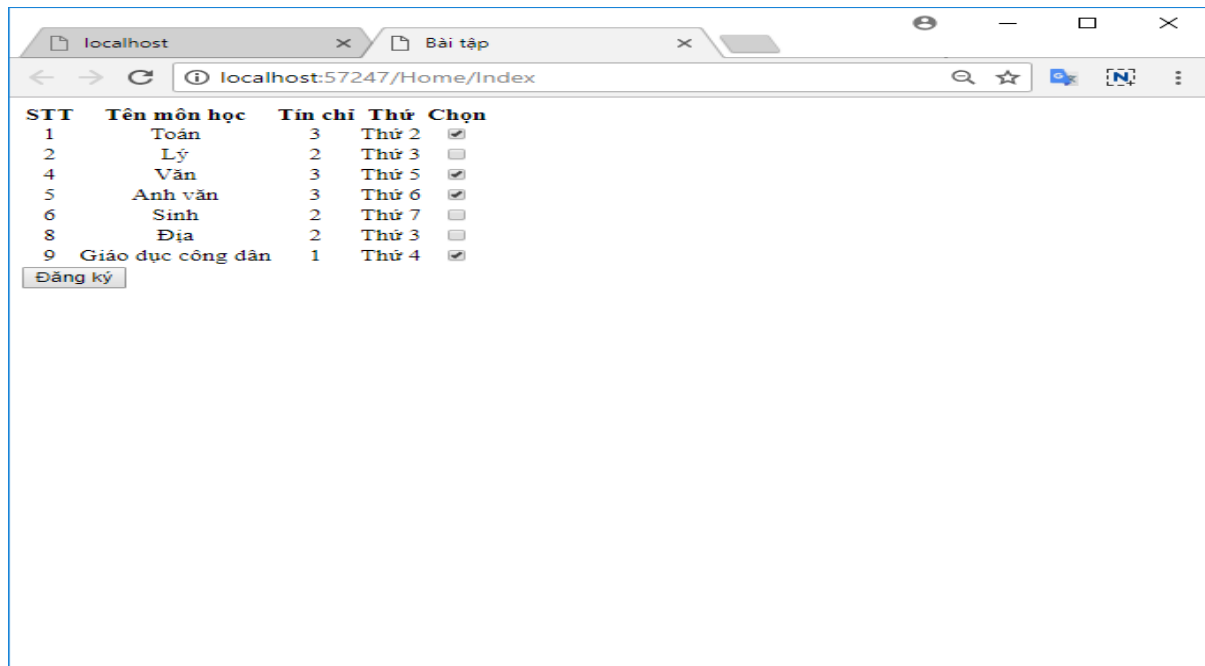
STT	Tên môn học	Tín chỉ	Thứ	Chọn
1	Toán	3	Thứ 2	<input type="checkbox"/>
2	Lý	2	Thứ 3	<input type="checkbox"/>
3	Hóa	2	Thứ 4	<input type="checkbox"/>
4	Văn	3	Thứ 5	<input type="checkbox"/>
5	Anh văn	3	Thứ 6	<input type="checkbox"/>
6	Sinh	2	Thứ 7	<input type="checkbox"/>
7	Sử	2	Thứ 2	<input type="checkbox"/>
8	Địa	2	Thứ 3	<input type="checkbox"/>
9	Giáo dục công dân	1	Thứ 4	<input type="checkbox"/>

Đoạn script:

```
<script>
$(document).ready(function() {
    var soTinChi = 0;
    var thoiGian = [];
    $(document).on('click', 'input[type=checkbox]', function () {
        var tr = $(this).closest('tr');
        if ($(this).prop('checked') == true) {
            //cộng số tín chỉ
            soTinChi += parseInt(tr.find('td:eq(2)').text());
            //lấy thứ và bỏ vào array
            var strThoiGian = tr.find('td:eq(3)').text();
            thoiGian.push(strThoiGian);
            //lấy các row chưa checked và kiểm tra thứ trùng thì ẩn
            $('input:checkbox:not(:checked)').each(function () {
                if ($(this).closest('tr').find('td:eq(3)').text() == strThoiGian) {
                    $(this).closest('tr').hide();
                }
            });
        } else {
            //trừ số tín chỉ
            soTinChi -= parseInt($(this).closest('tr').find('td:eq(2)').text());
            //loại thứ khỏi array
            var strThoiGian = tr.find('td:eq(3)').text();
            thoiGian.pop(strThoiGian);
            //lấy các row chưa checked và kiểm tra thứ trùng thì hiện
            $('input:checkbox:not(:checked)').each(function () {
                if ($(this).closest('tr').find('td:eq(3)').text() == strThoiGian) {
                    $(this).closest('tr').show();
                }
            });
        }
        if (soTinChi >= 10) {
            $('#dangky').show();
        } else {
            $('#dangky').hide();
        }
    });

    $('#dangky').on('click', function () {
        alert('Đăng ký thành công');
    });
});
</script>
```

Kết quả:



Chương 8: Biến và khai báo biến trong Javascript

1. Khai báo biến

Để khai báo biến ta sử dụng từ khóa 'var'.

```
//var tên biến;  
  
var firstname;
```

Cách đặt tên biến:

- Tên biến là các chữ không dấu viết hoa hoặc viết thường, các chữ số từ 0-9 và dấu gạch dưới (_).
- Tên biến bắt đầu phải là chữ hoặc dấu gạch dưới, không thể bắt đầu bằng số.

Ví dụ:

```
var firstname; // Đúng  
var _Firstname; //Đúng  
var __firstname; //Đúng  
var firstname01; // Đúng  
var 01firstname; //Sai
```

Có thể khai báo nhiều biến 1 lúc trên 1 dòng, mỗi biến cách nhau bởi dấu phẩy:

```
var firstname, lastname, age;
```

2. Gán giá trị cho biến

```
//Cách 1: Khai báo xong rồi gán  
  
var firstname; // Khi chưa gán giá trị, mặc định là undefined  
firstname = "Quang Dũng"  
  
//Cách 2: Vừa khai báo vừa gán  
  
var firstname = "Quang Dũng"
```

3. Gán kiểu giá trị cho biến

Trong javascript có cơ chế xử lý kiểu dữ liệu rất linh hoạt giúp chuyển đổi dễ dàng, bởi mọi kiểu dữ liệu đều có thể quy về đối tượng và mỗi đối tượng ta có thể bổ xung các phương thức xử lý riêng.

Kiểu dữ liệu của biến được xác định dựa trên giá trị của biến đó.

Ví dụ:

```
var firstname = "Quang Dũng"; //kiểu string  
var age = 22; //kiểu int  
var height = 1.65; //kiểu float
```

4. Biến cục bộ và biến toàn cục

Biến cục bộ được khai báo nằm bên trong một hàm cụ thể, lúc này biến đó sẽ không sử dụng được ở bên ngoài hàm.

Ví dụ:

```
<script>  
function thong_bao() {  
    var noi_dung = "Nội dung thông báo";  
    alert(noi_dung); //Đúng vì biến noi_dung tồn tại  
}  
alert(noi_dung); //Sai vì không tồn tại biến noi_dung  
</script>
```

Biến toàn cục là biến được khai báo bên ngoài và không nằm trong hàm cụ thể nào.

Ví dụ:

```
<script>  
var noi_dung = "Nội dung thông báo";  
function thong_bao() {  
    alert(noi_dung); //Đúng vì biến noi_dung đã tồn tại  
}  
alert(noi_dung); //Đúng vì biến noi_dung đã tồn tại  
</script>
```