JavaScript

Một số khái niệm cơ bản

- Ngôn ngữ kịch bản
  - Dạng ngôn ngữ lập trình cấp cao
  - Ngôn ngữ thông dịch, mã lệnh được thông dịch trực tiếp ngay khi thực thi.
    - Ngôn ngữ biên dịch sẽ dịch mã nguồn sang mã máy, hay mã trung gian trước khi thực thi.
- Các ngôn ngữ script thông dụng
  - JavaScript, VBScript, ASP, PHP, JSP, ActionScript…

Một số khái niệm cơ bản

- Ngôn ngữ kịch bản trên server (server-side scripting)
  - Công nghệ thực thi trên web server dùng để xử lý các yêu cầu của user bằng cách tạo ra các trang HTML động chứa kết quả xử lý trả về cho user
  - Ngôn ngữ phía server thường cung cấp khả năng tương tác với CSDL
  - Các ngôn ngữ phổ biến: CGI, Cold Fusion, ASP, ASP.NET, PHP, JSP...

Một số khái niệm cơ bản

- Ngôn ngữ kịch bản trên client (client-side scripting)
  - Ngôn ngữ thực thi trên trình duyệt, phía client. Dùng để xử lý các yêu cầu của người dùng.
  - Các mã lệnh được nhúng vào HTML hay file riêng. User hoàn toàn có thể xem source code của ngôn ngữ kịch bản phía client.
  - JavaScript và VbScript là hai ngôn ngữ script thông dụng hiện nay.

JavaScript - tổng quan

- JS là ngôn ngữ script ở client, dùng để xử lý và tương tác với các thành phần HTML.
- JS là dạng ngôn ngữ thông dịch
- JS không liên quan đến ngôn ngữ Java
  - JS được phát triển bởi Netscape
  - Chỉ thực thi trên trình duyệt
  - Không có đầy đủ tính năng của ngôn ngữ lập trình
  - Cú pháp đơn giản, gần giống với ngôn ngữ C

Mocha→LiveScript→JavaScript

JavaScript - tổng quan

- JS có thể làm được gì?
  - Cung cấp cho người thiết kế HTML công cụ lập trình
  - Cho phép đặt đoạn văn bản động vào trang web
  - Có thể tác động các sự kiện trong trang HTML
  - Có thể đọc/ghi các thành phần của HTML
  - Dùng để check dữ liệu từ người dùng
  - Có thể check phiên bản trình duyệt
  - Có thể thao tác cookie của trang web.

JavaScript - tổng quan

- Các bước thực thi của JS
  1. Trình duyệt tải trang web về
  2. Trình duyệt kiểm tra xem có mã JS trong web hay không
  3. Nếu có, trình duyệt sẽ chuyển mã JS cho bộ thông dịch
  4. Bộ thông dịch xử lý và thực thi các mã lệnh JS
  5. Các mã lệnh có thể tác động đến các thành phần của trang web.
  6. Trình duyệt hiển thị toàn bộ nội dung web.

JavaScript - tổng quan

- Cách đặt mã lệnh JS vào trang web
  - Internal: đặt trong head hay body

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

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

JavaScript - tổng quan

- Cách đặt mã lệnh JS vào trang web
  - External: tạo tập tin bên ngoài và liên kết tập tin đó trong phần head.

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

Toán tử (operator)

- Các toán tử toán học

y = 5

Toán tử	Mô tả	Ví dụ	Kết quả
+	Cộng (addition)	x=y+2	x=7
-	Trừ (subtraction)	x=y-2	x=3
*	Nhân (multiplication)	x=y*2	x=10
/	Chia (division)	x=y/2	x=2.5
%	Chia lấy dư (modulus)	x=y%2	x=1
++	Tăng (increment)	x=++y	x=6
--	Giảm (decrement)	x=--y	x=4

Toán tử (operator)

- Các toán tử gán

Toán tử	Ví dụ	Tương tự với
=	x=y	
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Toán tử (operator)

- Các toán tử so sánh

Toán tử	Mô tả	Ví dụ
==	bằng	x==8
===	bằng (so sánh vừa giá trị vừa kiểu biến)	x===5
!=	không bằng	x!=8
>	lớn hơn	x>8
<	nhỏ hơn	x<8
>=	lớn hơn hay bằng	x>=8
<=	nhỏ hơn hay bằng	x<=8

Toán tử (operator)

- Toán tử logic

Toán tử	Mô tả	Ví dụ
&&	and	(x < 10 && y > 1) → true
||	or	(x==5 || y==5) → false
!	not	!(x==y) → true

- Toán tử điều kiện

Cú pháp: variablename=(condition)?value1:value2

Ví dụ: greeting=(visitor=="PRES")?"Dear President ":"Dear ";

Điều kiện đúng chọn value1, ngược lại chọn value2

Cấu trúc điều khiển

- Cấu trúc điều khiển if

```
if (biểu_thức_điều kiện_1)
{
   //khối lệnh được thực hiện nếu biểu thức 1 đúng;
}
else if (biểu_thức_điều_kiện_2)
{
   //khối lệnh được thực hiện nếu biểu thức 2 đúng;
}
else
{
   //khối lệnh được thực hiện nếu cả hai biểu thức trên đều không đúng;
}
```

Cấu trúc điều khiển

- Cấu trúc điều khiển switch

```
switch (biểu_thức_điều_kiện)
{
   case kết_quả_1 :
      //khối lệnh cần thực hiện nếu biểu_thức_điều_kiện bằng kết_quả_1;
      break;
   case kết_quả_2 :
      //khối lệnh cần thực hiện nếu biểu_thức_điều_kiện bằng kết_quả_2;
      break;
   default :
      //khối lệnh cần thực hiện nếu biểu_thức_điều_kiện cho ra một kết quả khác;
}
```

Cấu trúc điều khiển

- Cấu trúc điều khiển for

```
for (bt_khởi_tạo; bt_điều_kiện; bt_thay_đổi_giá_trị)
{
        //Khối lệnh cần lặp;
}
for (biến in đối tượng)
{
        //Khối lệnh cần lặp;
}
var x;
var mycars = ["Saab", "Volvo", "BMW"];
for (x in mycars){
    document.write(mycars[x]);
}
```

Cấu trúc điều khiển

- Cấu trúc điều khiển while

```
while (biểu_thức_điều_kiện)
{
    //khối lệnh lặp cần thực hiện nếu
    //biểu_thức_điều_kiện trả về true;
}
```

- Cấu trúc điều khiển do while

```
do
{
    //khối lệnh lặp cần thực hiện
} while (biểu_thức_điều_kiện);
```

Cấu trúc dữ liệu - biến

- Khai báo biến

Cú pháp	Ví dụ
var <tên biến>;	**var** n; n = 1;
var <tên biến> = <giá trị>;	**var** s = "JavaScript string";
<tên biến> = <giá trị>;	PI = 3.14;

- Cách đặt tên biến
  - Dùng các ký tự a..z, A..Z, 1..9, dấu gạch dưới '_', dấu '$'
  - Tên biến không trùng với từ khóa JS
  - Tên biến không bắt đầu bởi con số
  - Tên biến không có ký tự khoảng trắng
  - Tên biến là case sensitive.

Hàm (function)

- Hàm là khối câu lệnh với một danh sách tham số (hoặc không có tham số)
- Trong JS cho phép hàm không tên
- Hàm có thể trả về một giá trị

```
function tên_hàm(đối_số_1, đối_số_2)
{
    //các câu lệnh cần thực hiện mỗi khi hàm được gọi;
    return giá_trị_cần_trả_về;
}
```

Hàm (function)

- Gọi hàm trong JS
  - Gọi tên hàm và truyền tương ứng các tham số vào
    - VD: tên_hàm(đối số 1, đối số 2)
  - Khi gọi hàm không nhất thiết phải truyền đủ các đối số khi định nghĩa hàm. Nếu số đối số ít hơn khi định nghĩa hàm, khi đó những đối số không được truyền cho hàm sẽ mang giá trị undefined
  - Các kiểu cơ bản sẽ được truyền vào hàm theo giá trị, đối tượng sẽ được chuyển vào hàm theo tham chiếu.

Hàm (function)

```
<script type="text/javascript">
function myfunction(txt)
{
    alert(txt);
}
</script>

<body>
  <form>    <input type="button" onclick="myfunction('Hello')"
                value="Call function">
  </form>
</body>
```

Khai báo sử dụng biến

- Kiểu dữ liệu của biến
  - JS không quy định kiểu biến khi khai báo biến, kiểu của biến sẽ được tự động xác định khi gán dữ liệu cho biến
  - Các kiểu dữ liệu của JS
    - Kiểu số (number): số nguyên, số thực
    - Kiểu chuỗi (string)
    - Kiểu luận lý (boolean): true/false
    - Kiểu đối tượng (object)
    - Kiểu hàm (function)

Khai báo sử dụng biến

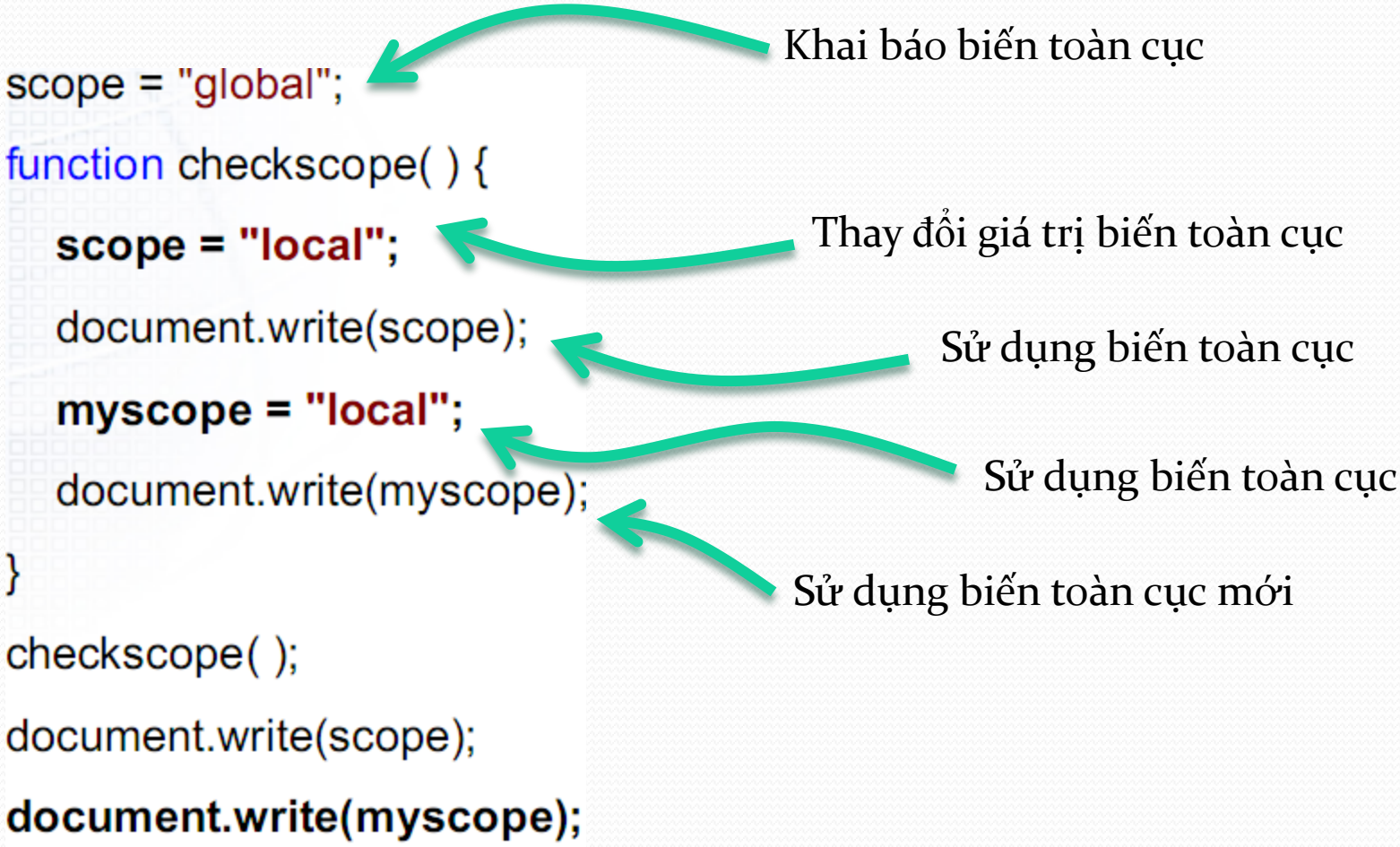
- Xác định kiểu của biến
  - Các giá trị trả về của toán tử typeof
    - number
    - string
    - boolean
    - object
    - function
    - undefined

```
var x = 1;
if (typeof(x) == "number")
    document.write("x is a number");
else
    document.write("x is not a number");
```

Kiểm tra xem x có phải là con số không?

Khai báo sử dụng biến

- Tầm vực của biến



Khai báo sử dụng biến

- Tầm vực của biến

```
var numberCars = 3;              // global
numberTrees = 15;                // global
if (numberTrees > numberCars) {
        var numberRoads = 4; // global
} else {
        var numberLakes = 9; // global, nhưng ko định nghĩa do đoạn code ko làm.
}
function simpleFunction()
{
    var colorCar = 'blue';       // local
    colorTree = 'green';         // global, chỉ khi hàm được gọi
    if (colorCar != colorTree) {
        var colorRoad = 'grey'; // local, từ dòng này
    } else {
        var colorLake = 'aqua'; // local, nhưng ko định nghĩa do code ko làm.
    }
}
```

Đối tượng mảng (array object)

- Mảng trong JS là dạng đối tượng
- Cách khai báo mảng 1 chiều

Cách 1: regular array

```
var myfriends=new Array();
myfriends[0]="John";      myfriends[1]="Bob";   myfriends[2]="Sue";
```

Cách 2: condensed array

```
var myfriends=new Array("John", "Bob", "Sue") ;
```

Cách 3: literal array

```
var myfriends=["John", "Bob", "Sue"] ;
```

Đối tượng mảng (array object)

- Khai báo mảng 2 chiều
  - Mảng 2 chiều được xem như mảng 1 chiều với mỗi phần tử của mảng 1 chiều này là một mảng 1 chiều khác
  - Ví dụ khai báo mảng 2 chiều 3x3
  - Cách 1

```
var b = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
```

  - Cách 2

```
var c = new Array(3);
dem = 0;
for (i=0; i < 3; i++)
{
    c[i] = new Array(3);
    for (j=0; j < 3; j++)
        c[i][j] = ++dem;
}
```

Chú ý: Các phần tử của mảng không nhất thiết phải cùng kiểu với nhau
Ví dụ:
var c = [1, 2.2, "chuỗi"];

Đối tượng mảng (array object)

- Các thuộc tính của đối tượng mảng

Thuộc tính	Mô tả
length	Là thuộc tính đọc/ghi, cho biết số lượng phần tử của mảng. Có thể gán số lượng phần tử cho thuộc tính này để thay đổi kích thước của mảng (mảng động)
prototype	Sử dụng thuộc tính này để thêm vào các thuộc tính mới hay phương thức mới cho đối tượng mảng

```
var myfriends=["John", "Bob", "Sue"];
document.write(myfirends.length);
```

$\Rightarrow$**3**

Đối tượng mảng (array object)

- Các phương thức của đối tượng mảng

Phương thức	Mô tả
concat(value1,...)	Nối các giá trị value1, value2... vào mảng hiện tại. Kết quả trả về một mảng mới chứa tất cả phần tử
every(testfunction [thisobj]) (JS1.6, FireFox)	Duyệt qua các phần tử của mảng, kiểm tra tất cả các phần tử có thoả điều kiện được hàm testfunction hay không ? Nếu thoả trả về true, ngược lại trả về false
filter(testfunction [thisobj]) (JS1.6, FireFox)	Lọc ra các phần tử thoả điều kiện được quy định trong hàm testfunction testfunction(element, index, array){....}
join([separator])	Chuyển tất cả các phần tử của mảng ra kiểu chuỗi và kết chúng lại thành một chuỗi dài chứa tất cả các phần tử

Đối tượng mảng (array object)

- Các phương thức của đối tượng mảng

Phương thức	Mô tả
indexOf(target, [startIndex])	Tìm phần tử target trong mảng từ vị trí startIndex đến vị trí cuối mảng (JS1.6, FireFox)
lastIndexOf(target, [startIndex])	Tìm phần tử target trong mảng từ vị trí startIndex trở về vị trí đầu mảng (JS1.6, FireFox)
map(testfunction [thisobj])	Trả về một mảng mới với các phần tử được tính toán lại từ hàm testfunction (JS1.6, FireFox)
pop()	Trả về phần tử nằm cuối mảng và xoá phần tử đó ra khỏi mảng
push(value1, ...)	Thêm các phần tử vào cuối mảng, trả về số lượng phần tử của mảng sau khi thêm vào

Đối tượng mảng (array object)

- Các phương thức của đối tượng mảng

Phương thức	Mô tả
reverse()	Đảo ngược các phần tử trong mảng
shift()	Xoá và trả về phần tử đầu tiên của mảng
slice(start, [end])	Trả về một mảng các phần tử từ vị trí start đến vị trí end của mảng, nếu không xác định ví end xem như lấy đến cuối mảng (vị trí end có thể là số âm)
splice(startIndex, [how_many], [value1, ...])	Xoá [how_many] phần tử tại vị trí startIndex và thay thế bằng các giá trị [value1, value2....] (chèn các giá trị mới vào vị trí đã xoá)
some(testfunction [thisobj]) (JS1.6, FireFox)	Duyệt qua các phần tử của mảng, nếu tồn tại bất kỳ phần tử nào thoả điều kiện trong hàm testfunction thì trả về giá trị true, ngược lại trả về giá trị false

Đối tượng mảng (array object)

- Các phương thức của đối tượng mảng

Phương thức	Mô tả
sort([sortFunction])	Sắp xếp các phần tử của mảng tăng/giảm theo điều kiện của hàm sortFunction(a, b). Hàm sortFunction trả về 3 giá trị: < 0: sắp vị trí của a trước b = 0: không cần sắp xếp (a == b) > 0: sắp vị trí của b trước a
toString()	Tạo chuỗi biểu diễn mảng và các phần tử của nó
unshift(value1, ...)	Chèn phần tử vào vị trí đầu mảng, trả về số lượng phần tử của mảng sau khi chèn
valueOf()	Trả về các giá trị của mảng ở dạng chuỗi

Đối tượng mảng (array object)

- VD1: sử dụng concat nối hai mảng

```
var fruits=["Apple", "Oranges"]; var meat=["Pork", "Chicken"];
var result1=fruits.concat(meat) //concat another array
Kết quả → result1 = ["Apple", "Oranges", "Pork", "Chicken"]
var result2=result1.concat(1, 2, 3) //concat plain values
Kết quả  → result2 = ["Apple", "Oranges", "Pork", "Chicken", 1, 2, 3]
```

Đối tượng mảng (array object)

- VD1: sử dụng every để kiểm tra các phần tử của mảng có thỏa điều kiện không.

```
var numbersarray=[2, 4, 5, -1, 34]
function isZeroAbove(element, index, array) {
    return (element > 0)
}
if (numbersarray.every(isZeroAbove)) //evaluates to false
    alert("All elements inside array is above 0 in value!")
```

Đối tượng mảng (array object)

- VD3: sử dụng phương thức join để kết các phần tử của mảng thành chuỗi.

```
var fruits=["Apple", "Oranges"]
var result1=fruits.join() //creates the String "Apple,Oranges"
var result2=fruits.join("*") //creates the String "Apple*Oranges"
```

- VD4: sử dụng phương thức indexOf để tìm phần tử trong mảng

```
var fruits=["Apple", "Oranges", "Pork", "Chicken"]
alert(fruits.indexOf("Pork")) //alerts 2
```

Đối tượng mảng (array object)

- VD5: Sử dụng phương thức map để tính lại các phần tử trong mảng

```
var numbersarray=[-3, 5, 34, 19]
function doubleIt(element, index, array) {
   return (element*2)
}
var doubledarray=numbersarray.map(doubleIt)
```

Kết quả → [-6, 10, 68, 38]

Đối tượng mảng (array object)

- VD6: sử dụng phương thức splice để xóa và thay thế phần tử trong mảng

```
var myarray=[13, 36, 25, 52, 83]

myarray.splice(2, 3, 42, 15) //myarray is now [13, 36 , 42, 15]. The 3rd, 4th and 5th element is removed, replaced with 42 and 15.
```

Đối tượng mảng (array object)

- VD7: sử dụng phương thức sort để sắp xếp mảng theo thứ tự tăng/giảm dần.
  - Sắp xếp mảng chuỗi ký tự tăng dần

```
var myarray=["Bob","Bully","Amy"]
myarray.sort() //Array now becomes ["Amy", "Bob", "Bully"]
```

  - Sắp xếp giảm dần, sắp tăng sau đó đảo mảng

```
var myarray=["Bob","Bully","Amy"]
myarray.sort()
myarray.reverse() //Array now becomes ["Bully", "Bob", "Amy"]
```

Đối tượng mảng (array object)

- Sắp xếp mảng số nguyên tăng dần

```
var myarray2=[25, 8, 7, 41]
myarray2.sort(function(a,b){return a - b}) // → [7, 8, 25, 41]
```

- Sắp giảm

```
var myarray2=[25, 8, 7, 41]
myarray2.sort(function(a,b){return b - a}) //→ [41, 25, 8, 7]
```

- Xáo trộn phần tử trong mảng

```
var myarray3=[25, 8, "George", "John"]
myarray3.sort(function() {return 0.5 - Math.random()})
```

Đối tượng chuỗi (string object)

- Các thuộc tính của đối tượng chuỗi

Thuộc tính	Mô tả
length	Trả về chiều dài của chuỗi (tổng số ký tự trong chuỗi)
prototype	Sử dụng thuộc tính này để thêm vào các thuộc tính mới hay phương thức mới cho đối tượng chuỗi

```
var str="John";
document.write(str.length);
```

Kết quả → 4

Đối tượng chuỗi (string object)

- Các phương thức của đối tượng chuỗi

Phương thức	Mô tả
anchor(name)	Trả về chuỗi với thẻ <a name="name"> bao quanh nó
big()	Trả về chuỗi với thẻ <big> bao quanh nó
blink()	Trả về chuỗi với thẻ <blink> bao quanh nó
bold()	Trả về chuỗi với thẻ <b> bao quanh nó
fixed()	Trả về chuỗi với thẻ <tt> bao quanh nó
fontcolor(color)	Trả về chuỗi với thẻ <font color="color"> bao quanh nó
fontsize(size)	Trả về chuỗi với thẻ <font size=" size "> bao quanh nó
italics()	Trả về chuỗi với thẻ <i> bao quanh nó
link(url)	Trả về chuỗi với thẻ <a href="url"> bao quanh nó
small()	Trả về chuỗi với thẻ <small> bao quanh nó

Đối tượng chuỗi (string object)

- Các phương thức của đối tượng chuỗi

Phương thức	Mô tả
strike()	Trả về chuỗi với thẻ <strike> bao quanh nó
sub()	Trả về chuỗi với thẻ <sub> bao quanh nó
sup()	Trả về chuỗi với thẻ <sup> bao quanh nó
charAt(x)	Trả về ký tự tại vị trí "x" của chuỗi
charCodeAt(x)	Trả về giá trị Unicode của ký tự tại vị trí "x" của chuỗi
concat(v1, v2,...)	Kết hợp một hay nhiều chuỗi (tham số v1, v2, ...) vào một chuỗi đã tồn tại và trả về chuỗi đã được kết hợp (không làm thay đổi chuỗi nguồn)
fromCharCode (c1, c2,...)	Trả về chuỗi được tạo bởi một chuỗi các giá trị Unicode (tham số c1, c2...). Đây là phương thức tĩnh của lớp String, do đó không cần phải tạo đối tượng cụ thể để sử dụng (vd: String.fromCharCode(....))

Đối tượng chuỗi (string object)

Phương thức	Mô tả
indexOf(substr, [start])	Tìm chuỗi con substr bên trong chuỗi, nếu tìm thấy trả về ký tự được tìm hay chuỗi con của chuỗi, nếu không tìm thấy trả về giá trị -1
lastIndexOf(substr, [start])	[start] là tham số tuỳ chọn xác định vị trí bắt đầu tìm (mặc định là 0)
match(regexp)	Tương tự phương thức indexOf() nhưng tìm bắt đầu từ cuối chuỗi trở về đầu chuỗi ([start] mặc định là string.length-1)
replace(regexp, replacetext)	Tìm một giá trị xác định trong chuỗi. Trả về mảng các thông tin hay trả về giá trị null nếu không tìm thấy
search(regexp)	Tìm và thay thế một số ký tự trong chuỗi (nếu thoả biểu thức regular expression – biểu thức chính quy)

Đối tượng chuỗi (string object)

Phương thức	Mô tả
slice(start, [end])	Trả về chuỗi con của chuỗi dựa trên tham số start và tham số end. (mặc định vị trí end là cuối chuỗi, giá trị của tham số end có thể là số âm)
split(delimiter, [limit])	Trả về mảng chứa các chuỗi con được phân cách bởi delimiter của chuỗi. Tham số [limit] là tham số tuỳ chọn, xác định số lượng thành phần tối đa được trả về
substr(start, [length])	Trả về chuỗi con của chuỗi bắt đầu tại vị trí start và có chiều dài là length. Tham số [length] là tuỳ chọn, nếu không xác định, mặc định lấy chiều dài từ start đến cuối chuỗi
substring(from, [to])	Trả về các ký tự của chuỗi giữa vị trí from và to. Nếu không xác định giá trị [to], mặc định lấy đến cuối chuỗi
toLowerCase()	Chuyển các ký tự trong chuỗi ra chữ thường
toUpperCase()	Chuyển các ký tự trong chuỗi ra chữ hoa

Đối tượng chuỗi (string object)

- VD1: định dạng kiểu cho chuỗi

```
var txt="Hello World!";
document.write("<p>Big: " + txt.big() + "</p>");
document.write("<p>Small: " + txt.small() + "</p>");
document.write("<p>Bold: " + txt.bold() + "</p>");
document.write("<p>Italic: " + txt.italics() + "</p>");
document.write("<p>Blink: " + txt.blink() +
                " (does not work in IE)</p>");
document.write("<p>Fixed: " + txt.fixed() + "</p>");
document.write("<p>Strike: " + txt.strike() + "</p>");
document.write("<p>Fontcolor: " + txt.fontcolor("Red") + "</p>");
document.write("<p>Fontsize: " + txt.fontsize(16) + "</p>");
document.write("<p>Lowercase: " + txt.toLowerCase() + "</p>");
document.write("<p>Uppercase: " + txt.toUpperCase() + "</p>");
document.write("<p>Subscript: " + txt.sub() + "</p>");
document.write("<p>Superscript: " + txt.sup() + "</p>");
document.write("<p>Link: " + txt.link("http://www.w3schools.com") + "</p>");
```

Big: Hello World!

Small: Hello World!

Bold: **Hello World!**

Italic: *Hello World!*

Blink: Hello World! (does not work in IE)

Fixed: `Hello World!`

Strike: ~~Hello World!~~

Fontcolor: Hello World!

Fontsize: Hello World!

Đối tượng chuỗi (string object)

- VD2: Tách chuỗi thành mảng các phần tử

```
var str="One;Two;Three;Four;Five;Six;Seven;Eight;Nine;Ten";
var arr = str.split(";");
for (i=0; i < arr.length; i++)
{
   document.write(arr[i]+"<br>");
}
```

- VD3: tìm kiếm chuỗi con bên trong chuỗi

```
var str="Hello world!";
document.write(str.indexOf("Hello") + "<br />");
document.write(str.indexOf("World") + "<br />");
document.write(str.indexOf("world"));
```

Kết quả:

```
0
-1
6
```

Chuỗi và số

- Có thể lấy giá trị chuỗi và số cộng lại với nhau

```
var r = "Hello world !" + 123;      → r = "Hello world !123"
var r = 123 + "456"                 → r = "123456";
```

- Chuyển số ra chuỗi

```
Cách 1:  Dùng hàm toString()
var a = 123; var b = a.toString() + 456;  → b = "123456"
Cách 2:  Cộng với giá trị số một chuỗi rỗng ""
var a = 123; var b = a + "" + 456;        → b = "123456"
```

- Chuyển chuỗi ra số

```
Cách 1: Dùng hàm eval()
var a = 123;  var b = eval(a) + 456;      → b = 579
Cách 2: Nhân chuỗi số với 1
var a = 123;  var b = (a*1) + 456;        → b = 579
```

Đối tượng Date

- Dùng để thao tác với ngày, giờ, thời gian
- Có hai cách để tạo đối tượng date

Cách 1: Sử dụng hàm tạo không đối

```
var d = new Date();
```

➔ d chứa giá trị ngày/tháng/năm hiện hành

Cách 2: Sử dụng hàm tạo với 3 đối số

```
var d = new Date(year, month, day);
```

- VD: khai báo đối tượng Date để in ra ngày tháng hiện hành

```
var d = new Date();
document.write(d);
```

Đối tượng Date

- Các phương thức của Date

Phương thức	Mô tả
getDate()	Trả về ngay của tháng từ (từ 1-31)
getDay()	Trả về ngày trong tuần (0: chủ nhật → 6: thứ 7)
getMonth()	Trả về tháng (từ 0-11)
getFullYear()	Trả về năm dạng 4 chữ số
getYear()	Trả về năm dạng 2 chữ số
getHours()	Trả về giờ (từ 0-23)
getMinutes()	Trả về phút (từ 0-59)
getSeconds()	Trả về phút giây (từ 0-59)
getMilliseconds()	Returns the milliseconds of a Date object (from 0-999)
getTime()	Trả về số millisecond (phần nghìn giây) (tính từ 0 giờ ngày 1/1/1970)

Đối tượng Date

parse(datestring)	Nhận vào ngày dạng chuỗi và trả về số millisecond của ngày đó (tính từ 0 giờ ngày 1/1/1970)
setDate(day)	Thiết lập ngày của tháng (từ 1-31)
setMonth(month)	Thiết lập tháng (từ 0-11)
setFullYear(year)	Thiết lập năm (4 chữ số)
setYear(year)	Thiết lập năm (2 chữ số)
setHours(hour)	Thiết lập giờ (0-23)
setMinutes(minute)	Thiết lập phút (từ 0-59)
setSeconds(second)	Thiết lập giây (từ 0-59)
setMilliseconds(ms)	Thiết lập phần ngàn giay (từ 0-999)
setTime(millisec)	Tính lại giá trị ngày giờ sau khi cộng hay trừ một giá trị millisecond với 0 giờ ngày 1/1/1970
toString()	Chuyển đối tượng Date ra dạng chuỗi

Đối tượng math

- Các thuộc tính của đối tượng Math

Thuộc tính	Mô tả
E	Hằng số Euler (khoảng 2.718)
LN2	Logarit tự nhiên của 2 (khoảng 0.693)
LN10	Logarit tự nhiên của 10 (khoảng 2.302)
LOG2E	Logarit cơ số 2 của e (khoảng 1.414)
LOG10E	Logarit cơ số 10 của e (khoảng 0.434)
PI	Số PI (khoảng 3.14159)
SQRT1_2	Căn bậc 2 của 1/2 (khoảng 0.707)
SQRT2	Căn bậc 2 của 2 (khoảng 1.414)

Đối tượng math

- Các phương thức của Math

Phương thức	Mô tả
abs(x)	Trả về trị tuyệt đối của một số
acos(x)	Trả về arccosine của một số
asin(x)	Trả về arcsine của một số
atan(x)	Trả về arctangent của x là một giá trị số giữa -PI/2 và PI/2 radians
ceil(x)	Làm tròn lên với số nguyên gần nhất
cos(x)	Trả về cosine của một số
exp(x)	Trả về giá trị E^x
floor(x)	Làm tròn xuống với số nguyên gần nhất

Đối tượng math

Phương thức	Mô tả
log(x)	Trả về log cơ số E của x
max(x,y)	Trả về giá trị lớn nhất giữa x và y
min(x,y)	Trả về giá trị nhỏ nhất giữa x và y
pow(x,y)	Tính x mũ y (x^y)
random()	Trả về số ngẫu nhiên từ 0 đến 1
round(x)	Làm tròn x đến số nguyên gần nhất
sin(x)	Trả về sin của x
sqrt(x)	Trả về căn bậc 2 của x
tan(x)	Trả về tangent của x