

# Ngôn ngữ Javascript

Nguyễn Thị Thùy Liên

Khoa CNTT-ĐHSP Hà Nội

[lienntt@hnue.edu.vn](mailto:lienntt@hnue.edu.vn)

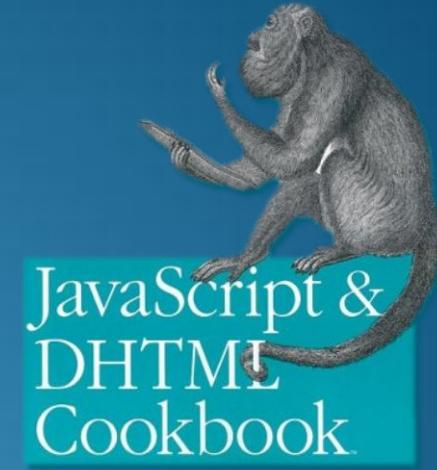
# Nội dung

- Giới thiệu JavaScript
  - JavaScript và ưu điểm của JavaScript
  - Thực thi JavaScript
- Cú pháp JavaScript
  - Kiểu dữ liệu
  - Khai báo biến
  - Các toán tử
  - Pop-up box
    - Alert, confirm, prompt
  - Cấu trúc điều khiển
  - Hàm
  - Đối tượng
- Mô hình DOM
- Debug

JAVA  
SCRIPT

# Giới thiệu về JavaScript

```
if (pop < 10)
{
    map.graphics.add(features[i].setSymbol(onePopSymbol));
}
else if (pop >= 10 && pop < 95)
{
    map.graphics.add(features[i].setSymbol(twoPopSymbol));
}
else if (pop >= 95 && pop < 365)
{
    map.graphics.add(features[i].setSymbol(threePopSymbol));
}
else if (pop >= 365 && pop < 1100)
{
    map.graphics.add(features[i].setSymbol(fourPopSymbol));
}
else
{
    map.graphics.add(features[i].setSymbol(fivePopSymbol));
}
```



JAVA  
SCRIPT

# JavaScript

- JavaScript là một ngôn ngữ kịch bản được hãng Sun Microsystems và Netscape phát triển giúp tạo các trang Web động và tương tác dễ dàng hơn.
- Client-site script:
  - Nhẹ, đơn giản, linh hoạt
  - Được thực thi bởi trình duyệt
  - Kết hợp với DOM



JavaScript

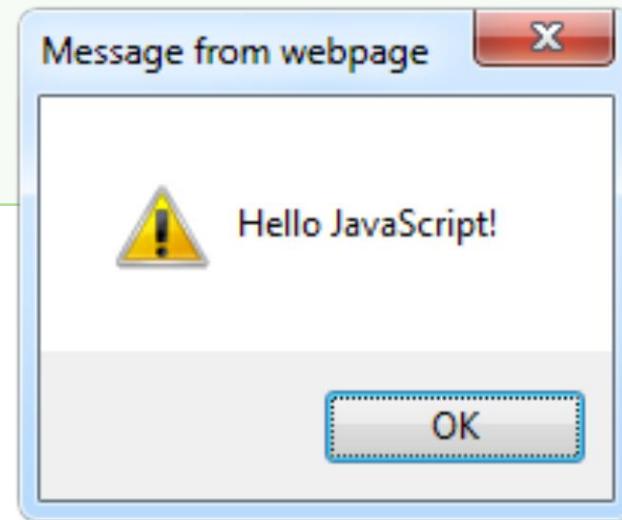
# Khả năng của Javascript

- JavaScript có thể tăng cường tính động và tính tương tác của các trang web.
  - Cung cấp sự tương tác người dùng: xử lý sự kiện nhấn các phím...
  - Thay đổi nội dung động: thay đổi hình ảnh khi di chuột....
  - Thực hiện các tính toán phức tạp
  - Xác nhận tính hợp lệ của dữ liệu
  - Điều khiển HTML tùy chỉnh
  - Thực hiện các chức năng AJAX

The screenshot shows a 'Signup' form with four fields: 'First Name \*', 'Last Name \*', 'Company Name \*', and 'Email address \*'. Each field has a red validation message to its right. The 'First Name' field has a message 'Please enter the first name'. The 'Last Name' field has a message 'Please enter the last name'. The 'Company Name' field has a message 'Please enter the company name'. The 'Email address' field has two messages: 'Please enter the email' and 'Invalid email address'. A 'Submit' button is located at the bottom right of the form.

# The First Script

```
<html>  
  
<body>  
  <script type="text/javascript">  
    alert('Hello JavaScript!');  
  </script>  
</body>  
  
</html>
```



# Chèn Javascript vào HTML

- Đoạn mã JavaScript có thể được đặt trong:
  - Thẻ **<script>** trong thẻ head
  - Thẻ **<script>** trong thẻ body (không khuyến khích)

```
<script language="JavaScript">
    <!--
        JavaScript statements;
    //-->
</script>
```

- Sử dụng các biểu thức JavaScript trong các giá trị thuộc tính của thẻ
- Sử dụng trong các trình điều khiển sự kiện

```

```

# Chèn Javascript vào HTML

- Trong file javascript được liên kết thông qua thẻ **<script>** nằm trong thẻ head
  - Các file chứa mã javascript thường có phần mở rộng là **.js**

```
<script src="filename.js" type="text/javascript">  
<!-- code placed here will not be executed! --&gt;<br/></script>
```

- Khuyến khích viết Javascript trong các file mở rộng
- Các file **.js** được cache bởi trình duyệt

# Thực thi mã Javascript

- Mã JavaScript được thực thi khi trang web được tải về hoặc khi có sự kiện xảy ra với trình duyệt.
  - Tất cả các câu lệnh được thực thi khi trang web được tải
  - Một số câu lệnh được khai báo thành các hàm sẽ được thực thi sau
- Các hàm hoặc các đoạn mã được khai báo trong “event handlers” thông qua thuộc tính của thẻ sẽ được thực thi khi sự kiện đó diễn ra

```

```

# Cú pháp JavaScript

```
if (pop < 10)
{
    map.graphics.add(features[i].setSymbol(onePopSymbol));
}
else if (pop >= 10 && pop < 95)
{
    map.graphics.add(features[i].setSymbol(twoPopSymbol));
}
else if (pop >= 95 && pop < 365)
{
    map.graphics.add(features[i].setSymbol(threePopSymbol));
}
else if (pop >= 365 && pop < 1100)
{
    map.graphics.add(features[i].setSymbol(fourPopSymbol));
}
else
{
    map.graphics.add(features[i].setSymbol(fivePopSymbol));
}
```



JavaScript &  
DHTML  
Cookbook

JAVA  
SCRIPT

# Cú pháp

- Cú pháp JavaScript tương tự như C# và Java
  - Toán tử (+, \*, =, !=, &&, ++, ...)
  - Biến (ít kiểu hơn)
  - Cấu trúc điều khiển (**if**, **else**)
  - Lặp (**for**, **while**)
  - Mảng (**my\_array[]**) và mảng liên kết (**my\_array['abc']**)
  - Hàm
- Chú ý: JavaScript phân biệt chữ hoa chữ thường

# Biến

- Biến được khai báo sử dụng từ khoá ‘var’.
  - var ten\_bien;
  - var ten\_bien = gia\_tri;
  - var ten\_bien kieu\_du\_lieu;
  - var ten\_bien = ten\_bien\_khac;

ví dụ: var A = 10 ;
- Tên biến bắt đầu là chữ cái hoặc dấu gạch dưới, hoặc dấu \$, theo sau là chữ cái, chữ số, dấu gạch dưới, hoặc dấu \$.
- Phạm vi:
  - Biến toàn cục
  - Biến cục bộ

# Các kiểu dữ liệu

- JavaScript có một tập các kiểu dữ liệu.
  - Số: integer, floating-point

```
var x1 = 34.00;          // Written with decimals
var x2 = 34;              // Written without decimals
var x3 = 123e3;            // 123000
var x4 = 123e-3;           // 0.123
```

- Giá trị logic: boolean (true/false)

```
var x1 = true;
var x2 = false;
```

- Giá trị rỗng: Null

# Các kiểu dữ liệu

- Chuỗi: string

```
// using double quotes  
var x1 = " a string variable";  
// using single quotes  
Var x2= ' a string variable' ;
```

- Mảng: array

```
var fruits= ["banana", "orange", "lemon"] ;
```

- Đối tượng: object

```
var person = {firstName:"John",  
lastName:"Doe", age:50, eyeColor:"blue"} ;
```

# Toán tử

- Toán tử số học: `+`, `-`, `*`, `/`, `%`, `++`, `--`
- Toán tử gán: `=`, `+=`, `-=`, `*=`, `/=`, `%=`
- Toán tử chuỗi: `+`, `+=`
- Toán tử so sánh: `==`, `====`, `!=`, `!==`, `>`, `<`, `>=`, `<=`, `?`
- Toán tử logic: `&&`, `||`, `!`
- Chú ý: cộng chuỗi và số
  - Kết quả phép cộng chuỗi và số trả về chuỗi

# Ví dụ

```
<HTML>
  <HEAD>
    <SCRIPT LANGUAGE = "Javascript">
      var A = "12" + 7.5;
      var B = 2 + 4 + "5.5";
      var C = 2 + (4 + "5.5");
      document.write(A);
      document.write(B);
      document.write(C);
    </SCRIPT>
  </HEAD>
</HTML>
```

# Ví dụ

```
<HTML>
    <HEAD>
        <SCRIPT LANGUAGE="JavaScript">
            var x = 10;
            var y = 5;
            alert ("The value of x is "
                    + x + "The value of y is " + y);
            alert("x AND y = " + (x && y));
            alert("x OR y = " + (x || y));
            alert("NOT x = " + (!x));
        </SCRIPT>
    </HEAD>
</HTML>
```

# Standard Popup Boxes

- Alert box with text and [OK] button
  - Just a message shown in a dialog box:

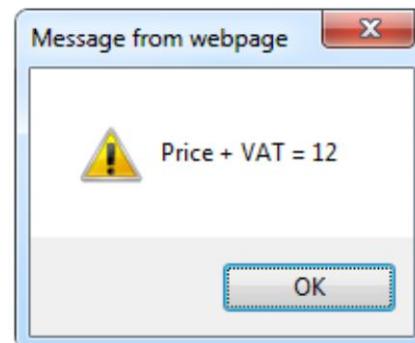
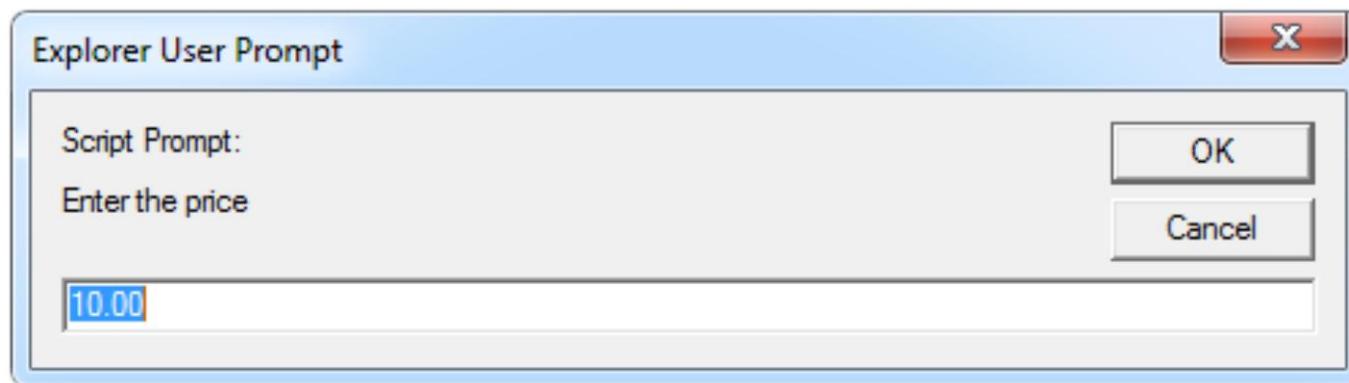
```
alert("Some text here");
```
- Confirmation box
  - Contains text, [OK] button and [Cancel] button:

```
confirm("Are you sure?");
```
- Prompt box
  - Contains text, input field with default value:

```
prompt ("enter amount", 10);
```

# JavaScript Prompt – Example

```
price = prompt("Enter the price", "10.00");
alert('Price + VAT = ' + price * 1.2);
```



# Lệnh rẽ nhánh (If)

```
if (điều kiện 1) {  
    mã Lệnh thực thi nếu  
    khi điều kiện 1 đúng  
} else if(điều kiện 2){  
    mã Lệnh thực thi nếu  
    khi điều kiện 2 đúng  
} else {  
    mã Lệnh được thực thi  
    nếu tất cả các điều  
    kiện sai  
}
```

```
if (quantity > 100) {  
    unitPrice = 1.00;  
} else if(quantity <50){  
    unitPrice = 1.20;  
} else {  
    unitPrice = 1.30;  
}
```

# Lệnh rẽ nhánh (switch)

```
switch(điều kiện) {  
    case 1 :  
        code block  
        break;  
    case 2 :  
        code block  
        break;  
    case 3 :  
        code block  
        break;  
    default:  
        code block  
        break;  
}
```

```
switch (quantity) {  
    case "50":  
        unitprice = 1.2;  
        break;  
    default:  
        unitprice = 1.2;  
        break;  
}
```

# Lăp (for/ for-in)

```
for(statement 1; statement2; statement 3) {  
    code block  
}
```

```
for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}
```

```
for(variable in object)  
{  
    code block  
}
```

```
var obj= {fname:"John",  
lname:"Doe"};  
for (x in obj) {  
    txt += person[x] + " ";  
}
```

# Lặp (while/ do-while)

```
while(condition) {  
    code block  
}
```

```
do {  
    code block  
} while(condition)
```

```
while (i < 10) {  
    text += "The number is " + i;  
    i++;  
};
```

```
do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);
```

# Hàm trong javascript

# Khai báo hàm

- Khai báo hàm trong javascript với từ khóa **function**

```
function functionName(parameters) {  
    code block  
}
```

```
function max (a, b) {  
    if(a < b) return b;  
    return a;  
}
```

# Khai báo hàm

- Hàm cũng có thể được khai báo theo cách thức sau:

```
var functionName = function(parameters) {  
    code block  
}
```

```
var max = function (a, b){  
    if(a < b) return b;  
    return a;  
};  
var z = max(5,7);
```

# Parameter và argument

```
function functionName(para1, para2, para3) {  
    code block  
}
```

- **parameters**: tên các tham số truyền vào trong khai báo hàm
- **arguments** : giá trị thực được truyền vào hàm

# Function Parameters

- Quy tắc :
  - Không chỉ rõ kiểu dữ liệu parameter khi khai báo hàm.
  - Không kiểm tra kiểu dữ liệu khi truyền biến trong hàm
  - Không kiểm tra số lượng tham số truyền vào hàm
- Mặc định:
  - Tất cả các tham số không được truyền vào khi gọi làm (số tham số ít hơn khai báo) đều được gán giá trị là:**undefined**

```
var max = function (a, b){  
    if(b === undefined) b=0;  
    return a + b;  
};
```

# Đối tượng argument

- Đối tượng argument chứa một mảng các đối số được truyền vào khi gọi hàm

```
x = sumAll(1, 123, 500, 115, 44, 88);
```

```
function sumAll() {  
    var i, sum = 0;  
    for (i = 0; i < arguments.length; i++){  
        sum += arguments[i];  
    }  
    return sum;  
}
```

# Đối tượng trong javascript

# Đối tượng trong javascript

- Trong JavaScript, "everything" đều được coi là object.
  - Các biến Booleans, numbers, strings có thể là đối tượng nếu được khai báo với từ khóa **new**
  - Date , Math, Regular expression, Array, Function, Object đều là các đối tượng

```
var test = "some string";
alert(test[7]);          // shows letter 'r'
alert(test.charAt(5));   // shows letter 's'
alert("test".charAt(1)); //shows letter 'e'
alert("test".substring(1,3)); //shows 'es'
```

```
var arr = [1,3,4];
alert (arr.length);      // shows 3
arr.push(7);            // appends 7 to end of array
alert (arr[3]);          // shows 7
```

# Thuộc tính

- Thuộc tính: các biến chứa dữ liệu đặc trưng của đối tượng.
- Bộ các thuộc tính là danh sách các biến không có thứ tự.
- Các thuộc tính có thể được thay đổi giá trị, thêm, xóa hoặc chỉ đọc.
- Truy xuất đến thuộc tính:

```
objectName.propertyName
```

```
objectName[“propertyName”]
```

```
objectName[expression]
```

# Phương thức

- Phương thức: các hàm thực thi công việc cụ thể nào đó của đối tượng.
- Phương thức là thuộc tính với khai báo hàm
- Truy xuất đến phương thức:

```
objectName.methodName()
```

# Khai báo đối tượng

- Sử dụng Object Literal

- Đây là cách dễ nhất để khai báo đối tượng Javascript.
- Khai báo các cặp **name:value** trong cặp dấu ngoặc nhọn {}

```
var person = { firstName:"John",
               lastName:"Doe",
               age:50,
               eyeColor:"blue",
               fullName: function(){
                   return firstName +
                     lastName;
               }
};
```

# Khai báo đối tượng

- Sử dụng từ khóa **new**

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
Person.fullName = function(){
    return firstName + lastName;
}
```

# Khai báo đối tượng

- Sử dụng hàm khởi tạo đối tượng

```
function person(first, last, age, eye) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eye;  
    this.fullName = function(){  
        return firstName + lastName;  
    }  
}  
var myFather = new person("John", "Doe", 50, "blue");
```

# Từ khóa this

- Từ khóa **this** đại diện cho chính đối tượng hiện tại đang được sử dụng hoặc đang truy cập tới.
  - Khi sử dụng trong hàm, **this** đại diện cho chính đối tượng chứa hàm đó.
  - Khi sử dụng trong đối tượng, **this** đại diện cho chính đối tượng đó.

# Đối tượng String

- Thông thường, chuỗi trong javascript là các dữ liệu nguyên thủy được tạo với cú pháp:
  - **var firstName = "John";**
- Chuỗi cũng được khai báo và khởi tạo như là các đối tượng với từ khóa **new**:
  - **var firstName = new String("John");**

# Thuộc tính và phương thức của chuỗi

- Thuộc tính **length** : trả về độ dài của chuỗi

```
var txt = "ABCD";  
var len = txt.length;
```

- search()**: Tìm kiếm một chuỗi con và trả về vị trí của chuỗi đó trong chuỗi gốc.

```
var str = "Please locate where 'locate' occurs!";  
var pos = str.search("Locate");
```

# Thuộc tính và phương thức của chuỗi

- **indexOf()**: trả về vị trí đầu tiên xuất hiện chuỗi con
- **lastIndexOf()**: trả về vị trí cuối cùng xuất hiện chuỗi con

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("Locate");
var pos = str.lastIndexOf("Locate");
var pos = str.lastIndexOf("Locate", 10);
```

- Return: -1 nếu không tìm thấy chuỗi con
- Tham số thứ 2 là vị trí bắt đầu tìm kiếm

# Thuộc tính và phương thức của chuỗi

- **slice(start, end)**: Lấy ra một phần của chuỗi từ vị trí start đến vị trí end
  - Nếu tham số là số âm: tính từ cuối chuỗi.
  - Nếu thiếu tham số end: lấy hết chuỗi

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13); //Banana
var res2 = str.slice(-12, -6);
var res3 = str.slice(7);
var res5 = str.slice(-12);
```

# Thuộc tính và phương thức của chuỗi

- **substring(start, end)**: tương tự như slide nhưng không áp dụng tham số là số âm

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13); //Banana
var res3 = str.substring(7);
```

- **substr (start, length of substring )**: lấy chuỗi con từ vị trí start và lấy length ký tự

```
var str = "Apple, Banana, Kiwi";
var res = str.substr (7, 6); //Banana
var res3 = str.substr(7);
```

# Thuộc tính và phương thức của chuỗi

- **replace(*find, replace*)**: thay thế chuỗi tìm kiếm bằng một chuỗi mới

```
var str = "Apple, Banana, Kiwi";
var res = str.replace("Banana", "Orange");
```

- **concat (*addition\_string*)**: nối chuỗi

```
var str = "Apple, Banana, Kiwi";
var res = str.concat("Orange");
var res3 = str + ", " + "Orange";
```

# Thuộc tính và phương thức của chuỗi

- **toUpperCase()**: chuyển sang chữ in hoa

```
var str = "Apple, Banana, Kiwi";
var res = str.toUpperCase();
```

- **toLowerCase ()**: chuyển sang chữ in thường

```
var str = "Apple, Banana, Kiwi";
var res = str.toLowerCase();
```

- **split (separator)**: phân tách chuỗi thành mảng dựa vào ký tự phân tách

```
var str = "Apple, Banana, Kiwi";
var res = str.split(",");
```

# Thuộc tính và phương thức của chuỗi

Phương thức	Mô tả
charAt()	Trả về ký tự ở vị trí xác định
charCodeAt()	Trả về mã Unicode của ký tự ở vị trí xác định
startsWith() endsWith()	Kiểm tra xem chuỗi có bắt đầu/kết thúc với một ký tự/chuỗi nào đó không
fromCharCode()	Chuyển đổi từ mã Unicode sang ký tự
includes()	Kiểm tra xem chuỗi có chứa chuỗi/ký tự con không
match()	Tìm kiếm chuỗi phù hợp với biểu thức chính quy và trả về kết quả tìm thấy
toString()	Chuyển đổi kiểu đối tượng String
trim()	Loại bỏ các khoảng trắng ở hai đầu chuỗi
valueOf()	Trả về kiểu giá trị nguyên thủy của đối tượng String

# Thuộc tính và phương thức của chuỗi

big()

bold()

fontcolor()

fontsize()

italics()

link()

small()

strike()

sub()

sup()

```
var txt = "Hello World!";

document.write("The original string: " + txt);
document.write("Big:" + txt.big());
document.write("Small:" + txt.small());
document.write("Bold:" + txt.bold());
document.write("Italic:" + txt.italics());
document.write("Fixed:" + txt.fixed());
document.write("Strike:" + txt.strike());
document.write("color:" + txt.fontcolor("red"));
document.write("Fontsize:" + txt.fontsize(6));
document.write("Subscript:" + txt.sub());
document.write("Superscript:" + txt.sup());
document.write("Link:" +
txt.link("https://abc.xyz")));
```

# Arrays

- Mảng: biến đặc biệt, chứa nhiều dữ liệu cùng lúc
- Khởi tạo mảng
  - Sử dụng cú pháp : cặp dấu ngoặc vuông []

```
var fruits = ["Banana", "Orange", "Apple"];
```

- Sử dụng từ khóa new

```
var fruits = new Array("Banana", "Orange");
var fruits2 = new Array();
fruits2[0] = "Apple";
fruits2[1] = "Grapes";
```

# Thuộc tính và phương thức của mảng

- **toString()**: chuyển đổi một mảng thành một chuỗi các giá trị phân tách nhau bởi dấu phẩy.

```
var fruits = ["Banana", "Orange", "Apple"];
var txt= fruits.toString();
//Banana,Orange,Apple
```

- **join("separator")**: chuyển đổi một mảng thành một chuỗi các giá trị phân tách nhau bởi dấu phân tách separator

```
var fruits = ["Banana", "Orange", "Apple"];
var txt= fruits.join('*');
//Banana*Orange*Apple
```

# Thuộc tính và phương thức của mảng

- **length** : trả về số phần tử của mảng

```
var fruits = ["Banana", "Orange", "Apple"];
var len = fruits.length; // 3
```

- **concat()**: Kết hợp hai hay nhiều mảng tạo thành một mảng mới

```
var fruits = ["Banana", "Orange", "Apple"];
var add_fruits = ["Mango", "Grapes"];
var x= fruits.concat(add_fruits);
```

# Thuộc tính và phương thức của mảng

- **push("new item"):** Thêm phần tử vào cuối mảng

```
var fruits = ["Banana", "Orange", "Apple"];
fruits.push("Mango");
fruits.[fruits.length] = "Lemon";
```

- **pop():** lấy ra phần tử cuối mảng

```
var fruits = ["Banana", "Orange", "Apple"];
var x= fruits.pop(); // x = Apple
```

# Thuộc tính và phương thức của mảng

- **shift()** : Lấy ra phần tử đầu mảng

```
var fruits = ["Banana", "Orange", "Apple"];
var newitem = fruits.shift(); // Banana
```

- **unshift(*“new item”*)**: thêm phần tử vào đầu mảng

```
var fruits = ["Banana", "Orange", "Apple"];
fruits.unshift("Mango");
```

# Thuộc tính và phương thức của mảng

- **slice(*from, to*)** : Lấy ra một phần của mảng từ *from* đến *to* và tạo thành mảng mới

```
var fruits = ["Banana", "Orange", "Apple"];
var newitem = fruits.slice(3); // Apple
var newitem2 = fruits.slice(1,3);
// Orange, Apple
```

- **sort()**: sắp xếp các phần tử của mảng (tăng dần)

```
var fruits = ["Banana", "Orange", "Apple"];
fruits.sort(); //Apple, Banana, Orange
```

# Đối tượng toán học trong Javascript

- Đối tượng toán học giúp thực thi các hàm toán học trong javascript

Math.E (2,718)/Math.PI (3.14)	Math.sin(x) /Math.asin(x)
Math.LN10/Math.LN2	Math.cos(x)/Math.acos(x)
Math.LOG2E /Math.LOG10E	Math.tan(x)/Math.atan(x)
Math.abs(x)	Math.max(x,y)/Math.min(x,y)
Math.sqrt(x)	Math.floor(x)/Math.ceil(x) // round downward/upward
Math.log(x)	Math.round(x)
Math.SQRT1_2/Math.SQRT2	Math.random() // rand from 0 to 1
Math.exp(x) // e^x	Math.pow(x,y) //x^y

# Đối tượng Date

- Đối tượng thời gian Date chứa: năm, tháng, ngày, giờ phút, giây, và mili giây
- Định dạng
  - Chuỗi: **Fri Aug 04 2017 11:47:45 GMT+0700 (SE Asia Standard Time)**
  - Số: **1501822065560**
    - Thời gian viết dưới dạng số được tính bằng mili giây tính từ January 1, 1970, 00:00:00.

# Khởi tạo đối tượng Date

- Đối tượng Date được khởi tạo bởi hàm khởi tạo **new Date()**

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours,
         minutes, seconds, milliseconds)
```

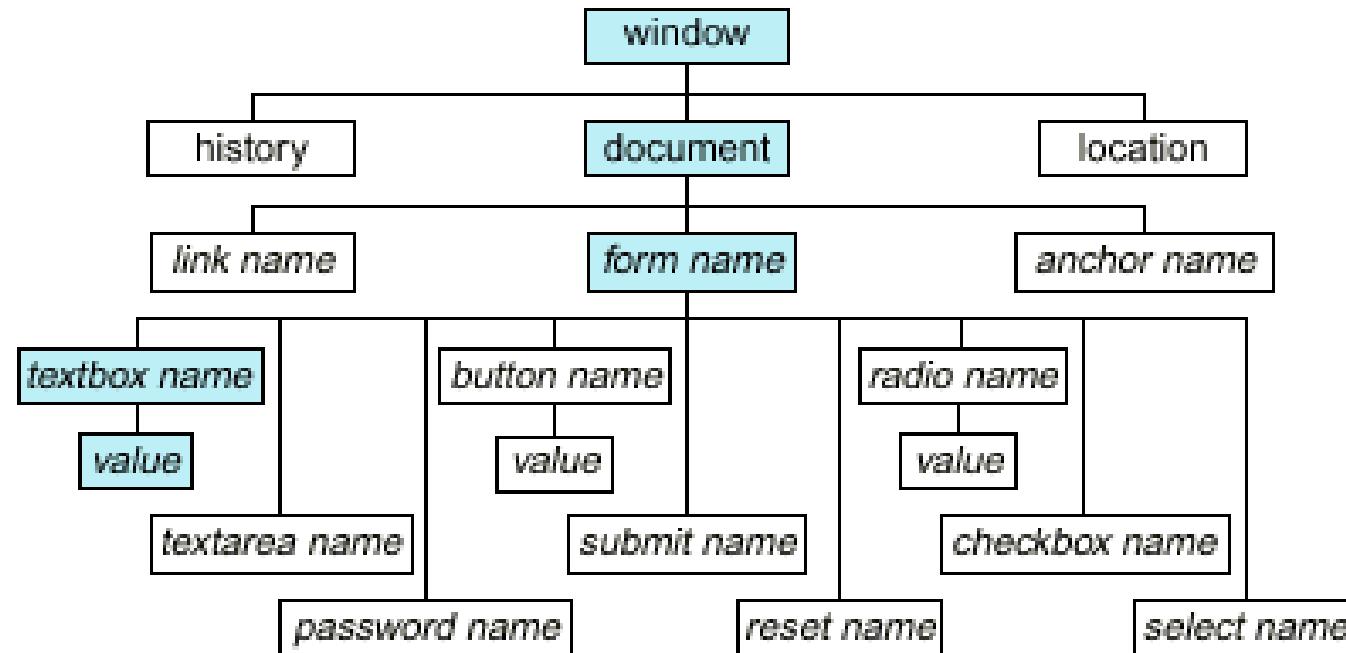
```
var d = new Date()
var d = new Date(86400000)
var d = new Date("October 17, 2017 11:13:00")
var d = new Date(99, 5, 24, 11, 33, 30, 0)
```

# Phương thức

Phương thức	Mô tả
getDate()	Trả về ngày trong tháng (1-31)
getDay()	Trả về ngày trong tuần (0-6)
getFullYear()	Trả về năm dưới dạng đầy đủ (yyyy)
getHours()	Trả về giờ (0-23)
getMilliseconds()	Trả về mili giây(0-999)
getMinutes()	Trả về phút (0-59)
getMonth()	Trả về tháng dạng số (0-11)
getSeconds()	Trả về giây (0-59)
getTime()	Trả về số mili giây tính từ mốc thời gian (milliseconds since January 1, 1970)

# Phương thức

Phương thức	Mô tả
toLocaleDateString()	Returns the date portion of a Date object as a string, using locale conventions
toLocaleTimeString()	Returns the time portion of a Date object as a string, using locale conventions
toLocaleString()	Converts a Date object to a string, using locale conventions
toString()	Converts a Date object to a string



The JavaScript Object Model

labelm labeld idchsevsl art

# Document Object Model (DOM)

# Document Object Model (DOM)

- Mọi phần tử HTML đều có thể được truy xuất thông qua JavaScript DOM API
- Các đối tượng DOM có thể được can thiệp, thay đổi dễ dàng bởi người lập trình
- Mô hình sự kiện giúp tạo các tương tác động trên trang với người sử dụng
- Ưu điểm
  - Tạo tương tác trên trang
  - Cập nhật động các đối tượng trên trang mà không cần tải lại trang

# Truy xuất các phần tử

Thuộc tính	Mô tả
document.anchors	Returns all <a> elements that have a name attribute
document.baseURI	Returns the absolute base URI of the document
document.body	Returns the <body> element
document.cookie	Returns the document's cookie
document.documentElement	Returns the <html> element
document.forms	Returns all <form> elements

# Truy xuất các phần tử

Thuộc tính	Mô tả
document.head	Returns the <head> element
document.images	Returns all <img> elements
document.title	Returns the <title> element
document.URL	Returns the complete URL of the document
document.doctype	Returns the document's doctype

# Truy xuất các phần tử

- Truy xuất phần tử thông qua thuộc tính ID

```
var elem = document.getElementById("some_id")
```

```
var myElement = document.getElementById("intro");
```

- Truy xuất phần tử thông qua tên thẻ

```
var elem = document.getElementsByTagName("some_id")
```

```
var x = document.getElementsByTagName("intro");
var y = x.getElementsByTagName("p");
```

# Truy xuất các phần tử

- Truy xuất phần tử thông qua tên lớp

```
var e = document.getElementsByClassName("cl_name")
```

```
var el = document.getElementsByClassName("intro");
```

- Truy xuất phần tử thông qua CSS selectors

```
var elem = document.querySelectorAll("selectors")
```

```
var x = document.querySelectorAll("p.intro");
```

# Ví dụ

```
<script>

var x = document.forms["frm1"];

var text = "";

var i;

for (i = 0; i < x.length; i++) {
    text += x.elements[i].value + "<br>";

}

document.getElementById("demo").innerHTML = text;

</script>
```

# Truy xuất phần tử thông qua mô hình DOM Tree Structure

- Truy xuất các phần tử trong DOM thông qua cấu trúc cây phân cấp:
  - element.childNodes
  - element.parentNode
  - element.nextSibling
  - element.previousSibling
  - element.firstChild
  - element.lastChild

# Truy xuất phần tử thông qua mô hình DOM Tree Structure – Ví dụ

```
var el = document.getElementById('div_tag');
alert (el.childNodes[0].value);
alert (el.childNodes[1].
      getElementsByTagName('span').id);
...
<div id="div_tag">
  <input type="text" value="test text" />
  <div>
    <span id="test">test span</span>
  </div>
</div>
```

- Chú ý: một số trình duyệt không hỗ trợ

# Thay đổi các phần tử HTML

- Thay đổi nội dung bên trong thẻ HTML

```
element.innerHTML = "new html content";
```

- Thay đổi giá trị thuộc tính trong thẻ HTML

```
element.setAttribute("attribute", "new value");
```

```
el.src = "new_src.jpg";
```

- Ví dụ. id, name, href, alt, title, src, etc...

```
element.setAttribute(attribute, "value");
```

```
el.setAttribute(src, "new_src.jpg");
```

# Thay đổi các phần tử HTML

- Thay đổi style của thẻHTML

```
element.style.property = "new style value";
```

- Ứng với định dạng inline style của phần tử
  - Quy tắc viết tên thuộc tính camelCase
- Example: style.width, style.marginTop, style.backgroundImage

```
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color = "blue";
</script>
```

# Thêm và xóa các phần tử

- Tạo một phần tử

```
document.createElement (element);
```

- Xóa một phần tử

```
document.removeChild(element);
```

- Thêm phần tử

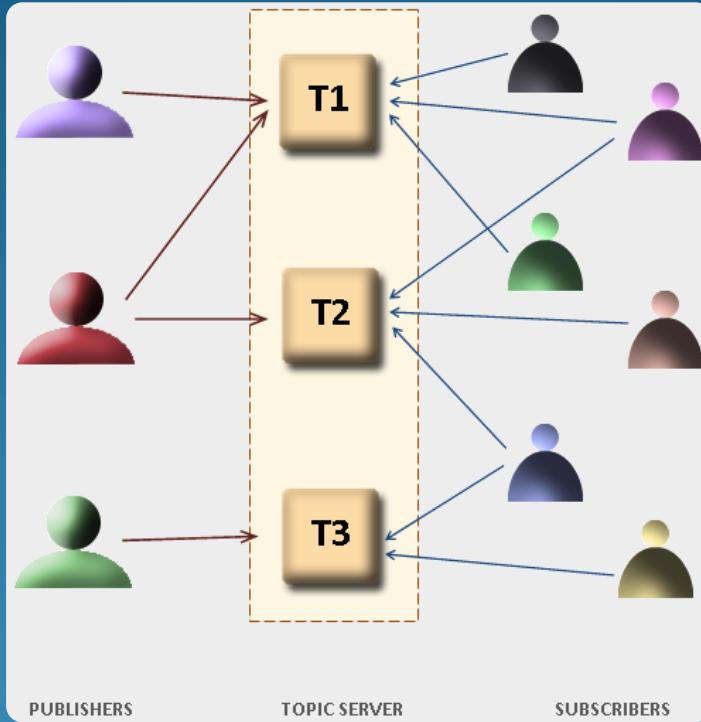
```
document.appendChild(element);
```

- Thay thế phần tử

```
document.replaceChild(element);
```

- Viết nội dung

```
document.write(text);
```



# The HTML DOM Event Model

# The HTML DOM Event Model

- Javascript cung cấp mô hình sự kiện cho phép quản lý và thực hiện các tác vụ nào đó khi sự kiện xảy ra
  - Các sự kiện được ghi nhận bởi trình duyệt và gọi đến các hàm, đoạn mã javascript thực thi tương ứng.
  - Sử dụng thuộc tính HTML:

```

```

- Sử dụng thông qua truy xuất DOM:

```
var img = document.getElementById("myImage");
img.onclick = imageClicked;
```

# The HTML DOM Event Model (2)

- Các trình xử lý sự kiện đều nhận một tham số
  - Chứa thông tin về sự kiện
  - Loại sự kiện (mouse click, key press, ....)
  - Vị trí, tọa độ nơi sự kiện xảy ra (e.g. mouse coordinates)
  - Tham chiếu đối tượng tạo ra sự kiện
    - Ví dụ: nút được nhấn
  - Trạng thái phím [Alt], [Ctrl] và [Shift]
  - Một số trình duyệt lưu trữ thông tin trong document.event



# Các sự kiện thông dụng

- Sự kiện với chuột:
  - onclick, onmousedown, onmouseup
  - onmouseover, onmouseout, onmousemove
- Sự kiện với bàn phím:
  - onkeypress, onkeydown, onkeyup
  - Only for input fields
- Sự kiện giao diện:
  - onblur, onfocus
  - onscroll

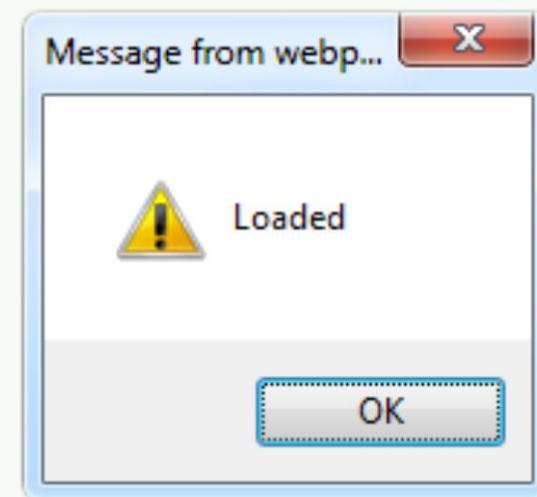
# Các sự kiện thông dụng(2)

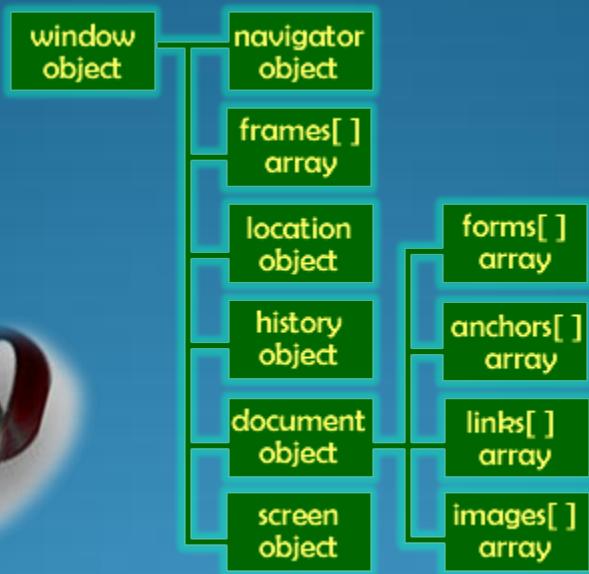
- Sự kiện trong form
  - onchange – áp dụng cho các trường input
  - onsubmit
    - Cho phép hủy bỏ submit form
    - Thường sử dụng trong kiểm tra dữ liệu trước khi gửi form
- Các sự kiện khác
  - onload, onunload
    - Chỉ áp dụng cho phần tử <body>
    - Xảy ra khi tất cả nội dung trên trang đã được tải/không được tải

# Ví dụ

- Sự kiện onload

```
<html>
<head>
  <script type="text/javascript">
    function greet() {
      alert("Loaded.");
    }
  </script>
</head>
<body onload="greet()" >
</body>
</html>
```



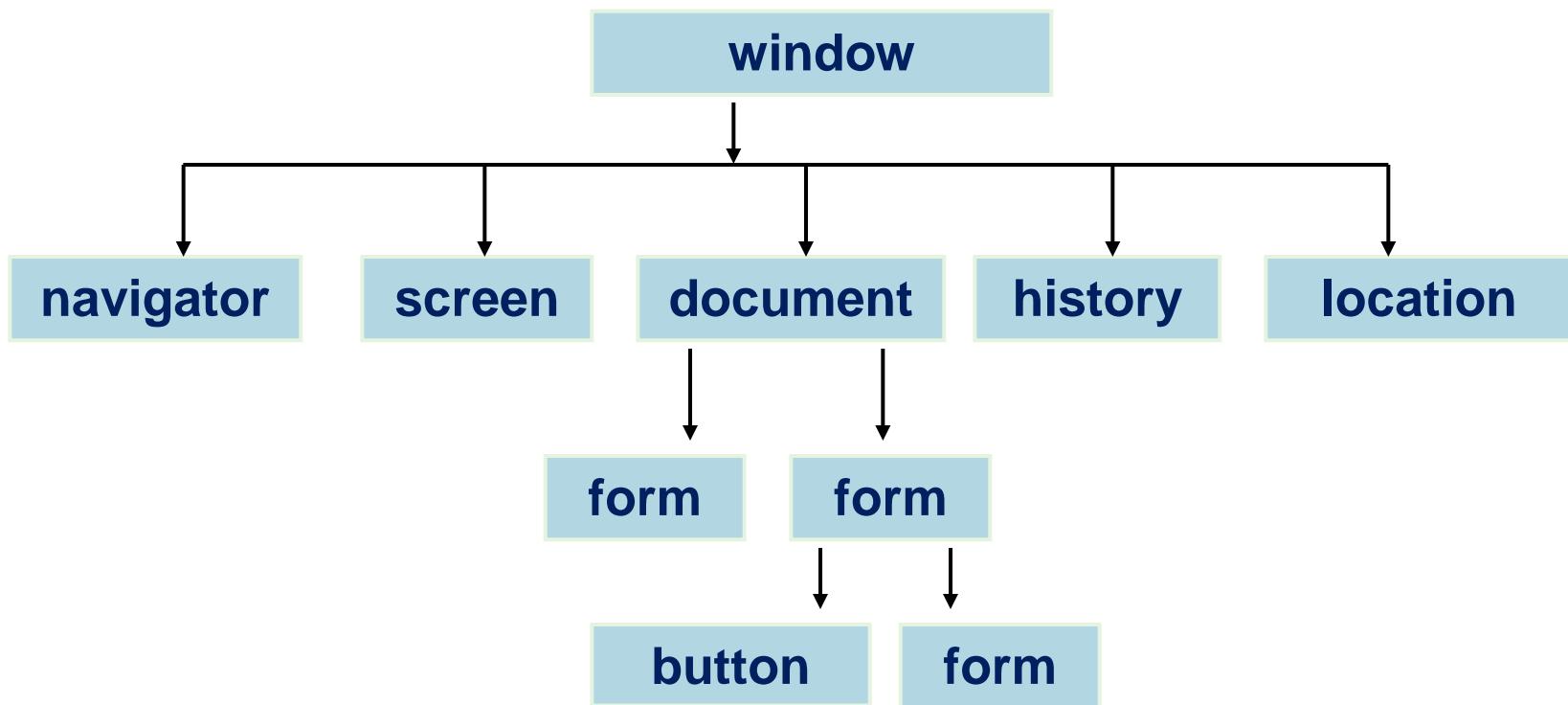


# The Built-In Browser Objects

# Built-in Browser Objects

- Trình duyệt cung cấp một số dữ liệu chỉ đọc thông qua
  - **window**
    - Nút trên cùng trong mô hình DOM tree
    - Đại diện cho cửa sổ trình duyệt
  - **document**
    - Nắm giữ thông tin về tài liệu đang được tải
  - **screen**
    - Thông tin hiển thị của người dùng
  - **browser**
    - Thông tin về trình duyệt

# DOM Hierarchy – Example



# Đối tượng window

- Đối tượng **window** đại diện cho cửa sổ trình duyệt.
- Tất cả các đối tượng, hàm, biến toàn cục đều là thuộc tính của đối tượng window.
- Window Size:
  - IE, Chrome, Firefox, Opera, Safari:
    - `window.innerHeight`
    - `window.innerWidth`
  - IE < 9
    - `document.documentElement.clientHeight`
    - `document.documentElement.clientWidth`
    - `document.body.clientHeight`
    - `document.body.clientWidth`

# The Window Object

- Thuộc tính:
  - frames
  - document
  - history
  - location
  - opener
  - status
  - cookie

# The Window Object

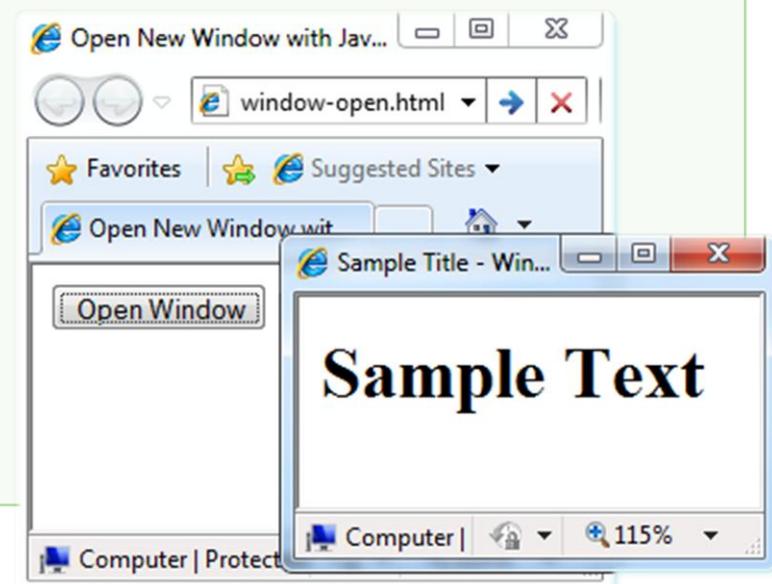
- Sự kiện:
  - onLoad()
  - onUnload()
- Phương thức:
  - alert(strMessage)
  - confirm(strMessage)
  - prompt(strMessage, defaultText)
  - open(url, name, option, replace)
  - close()
  - moveTo(x,y)
  - resizeTo()

# Ví dụ

- `window.open()`

```
var newWindow = window.open("", "sampleWindow",
    "width=300, height=100, menubar=yes,
    status=yes, resizable=yes");
```

```
newWindow.document.write(
    "<html><head><title>
        Sample Title</title>
    </head><body><h1>Sample
        Text</h1></body>");
newWindow.status =
    "Hello folks";
```



# Đối tượng Screen

- Đối tượng **screen** chứa thông tin về màn hình người sử dụng
- Properties
  - screen.width
  - screen.height
  - screen.availWidth
  - screen.availHeight
  - screen.colorDepth
  - screen.pixelDepth

# Đối tượng Screen

- Ví dụ

```
window.moveTo(0, 0);  
x = screen.availWidth;  
y = screen.availHeight;  
window.resizeTo(x, y);
```



# Đối tượng Document

- Đối tượng document chứa các thông tin về tài liệu đang được tải
- Thuộc tính
  - anchors[]
  - links[]
  - forms
  - images[]

```
document.links[0].href = "yahoo.com";
document.write(
    "This is some <b>bold text</b>");
```

# Form Validation – Ví dụ

```
function checkForm()
{
    var valid = true;
    if (document.mainForm.firstName.value == "") {
        alert("Please type in your first name!");
        document.getElementById("firstNameError").
            style.display = "inline";
        valid = false;
    }
    return valid;
}
...
<form name="mainForm" onsubmit="return checkForm()">
    <input type="text" name="firstName" />
    ...
</form>
```

# Đối tượng window Location

- Sử dụng để lấy địa chỉ trang hiện tại và chuyển hướng trình duyệt đến một trang mới
- Thuộc tính:
  - `window.location.href`: địa chỉ URL hiện tại
  - `window.location.hostname`: domain name của trang
  - `window.location.pathname` : phần đường dẫn và tên file
  - `window.location.protocol`: giao thức (`http:` or `https:`)
  - `window.location.assign`: tải một trang mới

# Đối tượng window history

- Đối tượng window.history chứa các thông tin về lịch sử duyệt web
- Phương thức
  - History.back(): tải lại trang trước trong lịch sử duyệt web.

```
<script>
    function goBack() {
        window.history.back()
    }
</script>

<body>
    <input type="button" value="Back" onclick="goBack()">
</body>
```

# Đối tượng window navigator

- Đối tượng window.navigator chứa các thông tin về trình duyệt
- Thuộc tính:
  - navigator.appName
  - navigator.appCodeName
  - navigator.platform
  - navigator.cookieEnabled
  - navigator.language
  - navigator.appVersion/userAgent

# Sự kiện thời gian

- Đối tượng window cho phép thực thi các đoạn mã theo các khoảng thời gian được chỉ định gọi là sự kiện thời gian.
- Phương thức
  - **setTimeOut**(*function, miliseconds*): Thực thi một hàm *function* sau khi đợi một số *miliseconds*.
  - **setInterval**(*function, milliseconds*)  
Giống hàm setTimeOut nhưng lặp lại việc gọi hàm sau một số *milliseconds*.

# Timers: setTimeout()

- Gọi một hàm sau một khoảng thời gian nhất định

```
var timer = setTimeout('bang()', 5000);
```

Sau 5 giây thì hàm bang() sẽ được gọi  
kể từ khi dòng lệnh này được chạy

- Dừng việc gọi hàm:

```
clearTimeout(timer);
```

Dừng việc gọi hàm đã  
khai báo qua setTimOut

# Timers: setInterval()

- Gọi hàm thực thi lặp đi lặp lại nhiều lần sau mỗi khoảng thời gian nhất định

```
var timer = setInterval('clock()', 1000);
```

Hàm được gọi sau mỗi 1 giây.

- Dừng thực thi:

```
clearInterval(timer);
```

Dừng lặp lại.

# Timer – Example

```
<script type="text/javascript">
    function timerFunc() {
        var now = new Date();
        var hour = now.getHours();
        var min = now.getMinutes();
        var sec = now.getSeconds();
        document.getElementById("clock").value =
            "" + hour + ":" + min + ":" + sec;
    }
    setInterval('timerFunc()', 1000);
</script>

<input type="text" id="clock" />
```

# Animation example

```
<style>
#container {
    width: 400px;
    height: 400px;
    position: relative;
    background: yellow;
}
#animate {
    width: 50px;
    height: 50px;
    position: absolute;
    background-color:
    red;
}
</style>
```

```
<p>
<button
onclick="myMove()">C1
ick Me</button>
</p>

<div id ="container">
<div id
="animate"></div>
</div>
```

# Animation example

```
<script>
function myMove() {
    var elem = document.getElementById("animate");
    var pos = 0;
    var id = setInterval(frame, 5);
    function frame() {
        if (pos == 350) {
            clearInterval(id);
        } else {
            pos++;
            elem.style.top = pos + 'px';
            elem.style.left = pos + 'px';
        }
    }
}
</script>
```



# Debugging JavaScript

The screenshot shows the Firebug extension for web development. At the top, there's a logo of a firebug (a reddish-orange insect) and the text "Firebug web development evolved". To the right are links for "blog", "discuss", "contribute", and "documentation". The main interface has tabs for "Inspect", "dom.js", "hasClass", "toggle", "onClick", and "onclick". The "Script" tab is active, displaying a portion of the "hasClass" function:

```
163 }
164
165 function hasClass(elt, className)
166 {
167     if (elt.className)
168     {
169         var classes = elt.className.split(" ");
170         for (var i in classes)
171         {
172             if (classes[i] == className)
173                 return true;
174         }
175     }
176 }
```

The "Watch" panel on the right shows the state of variables at the current line (line 172):

Variable	Value
1000+i	"10000"
this	Window joehwitt.com
className	"toggled"
classes	[ "commentLinkBox" ]
elt	div.commentLinkBox
id	"
className	"commentLinkBox"
nodeType	1
tagName	"DIV"
nodeName	"DIV"
localName	"DIV"
prefix	null
namespace	null

At the bottom left of the interface, it says "Done".

**JavaScript Debugging**

# Debugging JavaScript

- Các trình duyệt hiện nay có cửa sổ console cho phép quản lý, thông báo lỗi thực thi javascript
  - Lỗi có thể khác nhau trên các trình duyệt khác nhau
- Một số công cụ debug JavaScript
  - Microsoft Script Editor
    - Add-on for Internet Explorer
    - Hỗ trợ đặt breakpoints, watches
    - JavaScript statement **debugger**; opens the script editor

# Firebug

- Firebug – Firefox add-on for debugging JavaScript, CSS, HTML
  - Hỗ trợ breakpoints, watches, JavaScript console
  - Hữu ích cho chỉnh sửa HTML, CSS
    - Có thể chỉnh sửa trực tiếp: CSS, HTML, etc
    - Xem các khai báo CSS được áp dụng
  - Xem Ajax requests and responses

# Firebug (2)

The screenshot shows the Firebug developer toolbar with the "HTML" tab selected in the left pane. The tree view displays the DOM structure of a page, with the `<h1>` element highlighted. The right pane shows the "Style" tab, displaying the CSS rules applied to the selected element. The `h1` rule is defined in `debugcss.css (line 158)` with the following properties:

```
h1 {  
    color: #6600CC;  
    font-family: "Courier  
New", Courier, monospace, sans-serif;  
    font-size: 2em;  
    font-weight: 800;  
    margin-left: 20px;  
    text-align: center;  
}
```

The "Inherited from body" section shows the `body` rule from `debugcss.css (line 1)` with the following properties:

```
body {  
    color: #0A00B0;  
    font-family: "times new  
roman", Geneva, Arial, Helvetica, sans;  
    font-size: 1em;  
    font-size-adjust: none;  
}
```

At the bottom of the toolbar, there are several status icons: a pencil, a magnifying glass, a red circle with a white 'S', a green circle with a white checkmark, a blue square, and a red octagon with 'ABP'.

# JavaScript Console Object

- The `console` object exists only if there is a debugging tool that supports it
  - Used to write log messages at runtime
- Methods of the `console` object:
  - `debug(message)`
  - `info(message)`
  - `log(message)`
  - `warn(message)`
  - `error(message)`

# HTML, CSS and JavaScript Basics

Questions?