

Nội dung

- Giới thiệu về Javascript
- Nhúng Javascript vào trang web
- Cú pháp và kiểu dữ liệu Javascript
- Xử lý sự kiện
- DOM HTML với Javascript
- Ví dụ

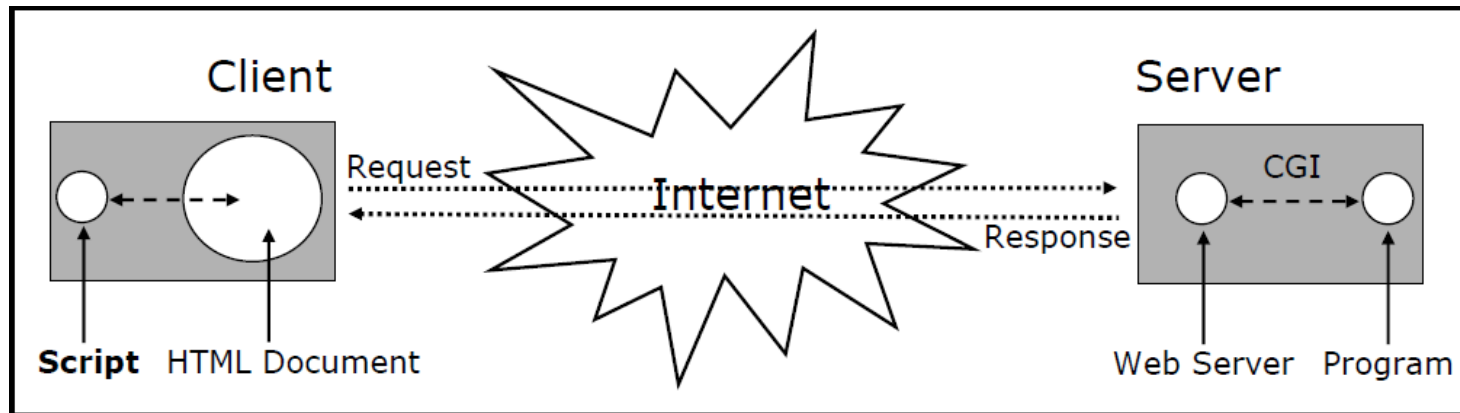


Giới thiệu Javascript



Ngôn ngữ kịch bản

- Kịch bản (Script) là những chương trình đơn giản hoạt động trên trình duyệt của người dùng (Client), hoặc ở phía server.



Ngôn ngữ kịch bản

- Ngôn ngữ kịch bản (Script language) là một ngôn ngữ lập trình, được dùng cho cả Client và Server
- **Client Side Script:** thực thi tại Client-Side (trình duyệt):
 - Thực hiện các tương tác với người dùng (tạo menu chuyển động, ...), kiểm tra dữ liệu nhập, ...
 - JavaScript (Netscape, ban đầu chỉ hỗ trợ cho trình duyệt Netscape, nhưng nay đã hỗ trợ nhiều trình duyệt)
 - VBScript (Microsoft, hỗ trợ bởi IE)
- **Server Side Script:** xử lý tại Server-Side
 - Nhằm tạo các trang web có khả năng phát sinh nội dung động.
 - Một số xử lý chính: kết nối CSDL, truy cập hệ thống file trên server, phát sinh nội dung html trả về người dùng...



Giới thiệu Javascript

- Là một ngôn ngữ kịch bản được phát triển Netscape Communications vào năm 1995, tên gọi ban đầu là LiveScript
- Sau này hợp tác với Sun Microsystems và đổi tên thành JavaScript.
- Chạy trên trình duyệt Netscape Navigation 2.0.
- Được hỗ trợ từ Internet Explorer 3.0.
- Có một số điểm chung với Java và sử dụng cú pháp tương tự ngôn ngữ C
- Được nhúng hoặc liên kết vào file HTML .
- Là ngôn ngữ Client-side script hoạt động trên trình duyệt của người dùng (client)
- Các ứng dụng client chạy trên trình duyệt như Netscape Navigator hoặc Internet Explorer, firefox, Chrome...



Giới thiệu Javascript

- JavaScript không phải là Java
- Là một ngôn ngữ lập trình đơn giản – ngôn ngữ kịch bản
- Chia sẻ xử lý trong ứng dụng web. Giảm các xử lý không cần thiết trên server.
- Điều khiển tính năng của trình duyệt
 - Hiện một cửa sổ hoặc một thông điệp đơn giản
- Giúp tạo các hiệu ứng, tương tác cho trang web
- Lưu trữ và sử dụng thông tin của người dùng
 - Tạo và đọc Cookie
- JavaScript làm cho việc tạo các trang Web động và tương tác dễ dàng hơn
 - Cung cấp sự tương tác người dùng
 - Thay đổi nội dung động
 - Xác nhận tính hợp lệ của dữ liệu

Giới thiệu Javascript

- Client-Side Script:
 - Script được thực thi tại Client-Side (trình duyệt): thực hiện các tương tác với người dùng (tạo menu chuyển động, ...) , kiểm tra dữ liệu nhập, ...
- Server-Side Script:
 - Script được xử lý tại Server-Side: nhằm tạo các trang web có khả năng phát sinh nội dung động. Một số xử lý chính: kết nối CSDL, truy cập hệ thống file trên server, phát sinh nội dung html trả về người dùng...
- Khi trình duyệt (Client browser) truy cập trang web có chứa các đoạn mã xử lý tại server-side. Server (run-time engine) sẽ thực hiện các lệnh Server-side Scripts và trả về nội dung HTML cho trình duyệt.
- Nội dung html trả về chủ yếu bao gồm: mã html, client-script.

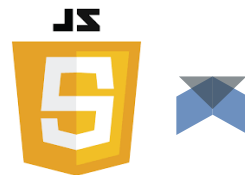


Java và Javascript

- Java và Javascript khác nhau thế nào?
- Tại sao lại có “Java” trong tên Javascript?



Java và Javascript



Java

Java là ngôn ngữ lập trình tĩnh, hướng đối tượng, hoạt động trên nhiều nền tảng.

Java là ngôn ngữ được biên dịch (compiled)

Java là một ngôn ngữ độc lập

JavaScript

JavaScript là ngôn ngữ lập trình động (hay ngôn ngữ kịch bản - scripted language) được sử dụng để làm cho các trang web trở nên sinh động

JavaScript là ngôn ngữ được diễn giải/thông dịch (interpreted).

JavaScript phụ thuộc nhiều hơn, nghĩa là nó hoạt động với HTML và CSS trên các trang web để tạo nội dung động.

Giới thiệu Javascript

- Client-Side Script:
 - Script được thực thi tại Client-Side (trình duyệt): thực hiện các tương tác với người dùng (tạo menu chuyển động, ...) , kiểm tra dữ liệu nhập, ...
- Server-Side Script:
 - Script được xử lý tại Server-Side: nhằm tạo các trang web có khả năng phát sinh nội dung động. Một số xử lý chính: kết nối CSDL, truy cập hệ thống file trên server, phát sinh nội dung html trả về người dùng...
- Khi trình duyệt (Client browser) truy cập trang web có chứa các đoạn mã xử lý tại server-side. Server (run-time engine) sẽ thực hiện các lệnh Server-side Scripts và trả về nội dung HTML cho trình duyệt.
- Nội dung html trả về chủ yếu bao gồm: mã html, client-script.

Nhúng Javascript vào HTML

- Sử dụng thẻ SCRIPT:

```
<script>
```

```
    JavaScript statements;
```

```
</script>
```

- Sử dụng một file JavaScript ở ngoài

```
<script src="filename.js"></script>
```

Nhúng Javascript vào HTML

```
<html>
  <head>
    <script>
      some statements
    </script>
  </head>
  <body>
    <script>
      some statements
    </script>
    <script src="Tên_file_script.js">method()</script>
    <script>
      //gọi thực hiện các phương thức được định nghĩa
      //trong "Tên_file_script.js"
    </script>
  </body>
</html>
```

Nhúng Javascript vào HTML

- Đặt giữa tag `<head>` và `</head>`: script sẽ **thực thi** ngay khi trang **web được mở**.
- Đặt giữa tag `<body>` và `</body>`: script trong phần body được **thực thi sau** khi thực thi các **đoạn script** có trong phần `<head>`.
- Số lượng đoạn client-script chèn vào trang **không hạn chế**.

Ví dụ

```
<!DOCTYPE html>

<html>

<head>

  <title>First page Javascript</title>

  <script>

    document.writeln("<H2>Hello JavaScript</H2>")

    document.write("HelloWorld");

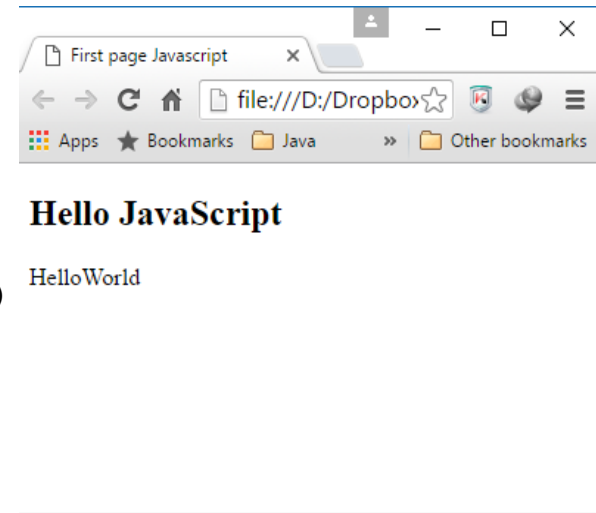
  </script>

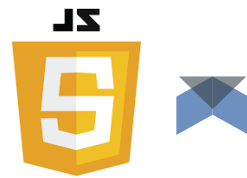
</head>

<body>

</body>

</html>
```





Cú pháp và kiểu dữ liệu trong Javascript



Cú pháp

- Phân biệt chữ hoa và thường
- Khoảng trắng, tab, xuống dòng chỉ dùng trong chuỗi.
- Kết thúc câu lệnh là dấu chấm phẩy “ ; ”
- Dấu phẩy để phân biệt các phần tử trong mảng

Các quy tắc chung

- Khối lệnh được bao trong dấu {}
- Mỗi lệnh nên kết thúc bằng dấu ;
- Cách ghi chú thích:
 - // Chú thích 1 dòng
 - /* Chú thích
nhiều dòng */

Biến

- **Phân biệt** chữ hoa và thường.
 - Ví dụ : Hai biến Java, java là khác nhau
- Cách đặt tên biến
 - Tuân thủ theo nguyên tắc đặt tên như lập trình C
 - A..Z,a..z,0..9,_ : phân biệt HOA, Thường
- Khai báo biến
 - Sử dụng từ khóa var
 - Ví dụ: **var count=10, amount;**
 - Không cần khai báo biến trước khi sử dụng, biến thật sự tồn tại khi bắt đầu sử dụng lần đầu tiên

Kiểu dữ liệu

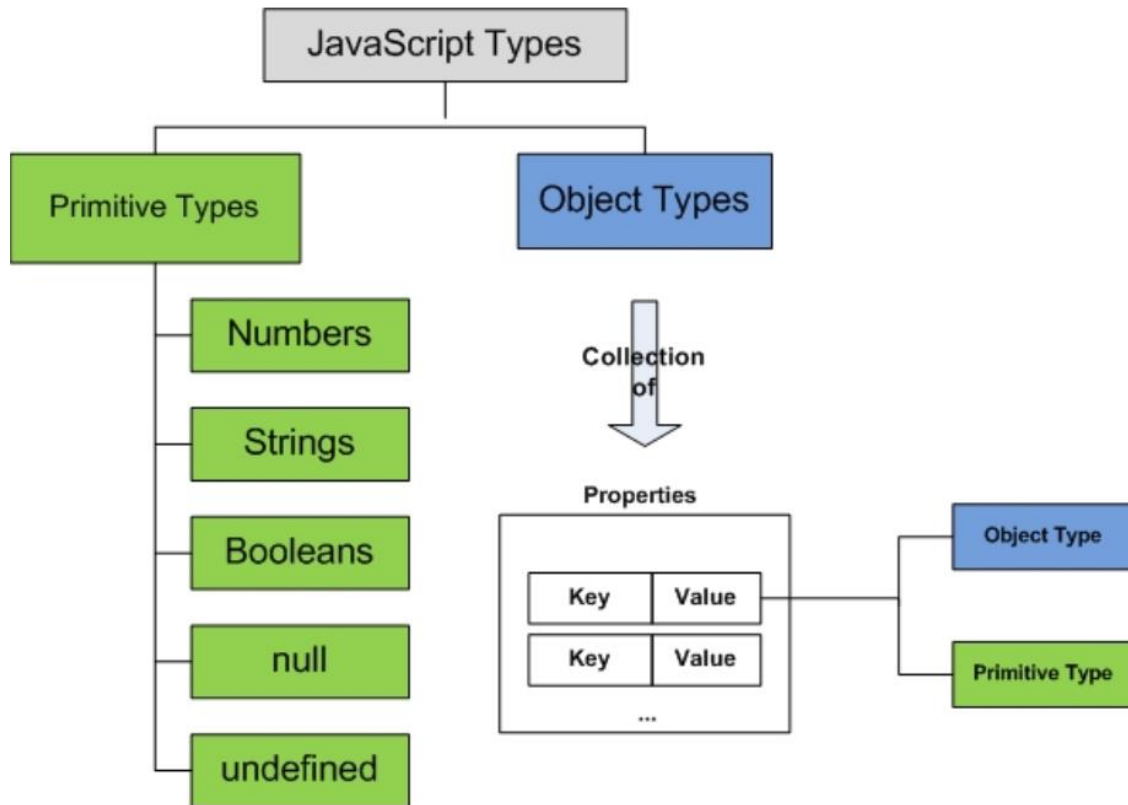
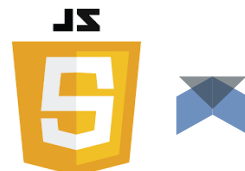
Kiểu dữ liệu	Ví dụ	Mô tả
Object	<code>var listBooks = new Array(10) ;</code>	Trước khi sử dụng, phải cấp phát bằng từ khóa new
String	<code>"The cow jumped over the moon."</code> <code>"40"</code>	Chứa được chuỗi unicode Chuỗi rỗng ""
Number	0.066218 12	Theo chuẩn IEEE 754
boolean	true / false	
undefined	<code>var myVariable ;</code>	<code>myVariable = undefined</code>
null	<code>connection.Close();</code>	<code>connection = null</code>

Kiểu dữ liệu

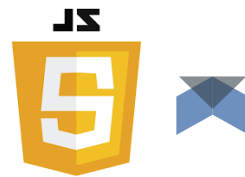
- Một biến trong Javascript có thể lưu bất kỳ kiểu dữ liệu nào
- Trong JavaScript, hai biến khác kiểu có thể kết hợp với nhau.
 - Ví dụ: $A = \text{" This apple costs Rs."} + 5$

Kết quả: một chuỗi với giá trị là "This apple costs Rs. 5".

Kiểu dữ liệu



Null, Undefined và NaN



Null, Undefined và NaN

- Null là đại diện cho một giá trị không tồn tại. Nó cần phải được gán cho một biến.

```
var a = null;  
console.log(a); //null
```

Null, Undefined và NaN

- undefined có nghĩa là không xác định
 - 1 biến được khai báo nhưng không được gán giá trị
 - Truy cập giá trị của một thuộc tính không tồn tại
 - Giá trị trả về của 1 hàm không trả về 1 giá trị:
 - Giá trị tham số của hàm đã được khai báo nhưng bị bỏ qua tham số khi gọi hàm

Null, Undefined và NaN

- Sự khác biệt giữa null và undefined
 - undefined nghĩa là một biến đã được khai báo nhưng chưa được gán giá trị, null là được gán cho một biến.

Null, Undefined và NaN

- NaN là viết tắt của “Not a Number”. Khi một function hoặc operation trong JavaScript không thể trả về một số cụ thể, nó sẽ trả về giá trị NaN thay thế. NaN sẽ được coi như một kiểu Number
- Một số trường hợp sinh ra NaN:
 - Lấy số 0 chia cho số 0
 - Lấy vô cùng (infinity) chia cho vô cùng (infinity)
 - Nhân vô cùng (infinity) với số 0
 - Bất kỳ phép tính toán nào trong đó NaN là một toán hạng
 - Chuyển đổi một chuỗi non-numeric hoặc undefined về dạng number.

Đổi kiểu dữ liệu

- Biến tự đổi kiểu dữ liệu khi giá trị mà nó lưu trữ thay đổi

- Ví dụ:

```
var x = 10;           // x kiểu Number  
x = "hello world !";  // x kiểu String
```

- Có thể cộng 2 biến khác kiểu dữ liệu

- Ví dụ:

```
var x;  
x = "12" + 34.5;      // KQ: x = "1234.5"
```

- Hàm **parseInt(...)**, **parseFloat(...)** : Đổi kiểu dữ liệu từ chuỗi sang số.

Kiểu dữ liệu số

- Kiểu số có hai loại thông dụng là **kiểu số nguyên** và **kiểu số thực**.
 - Ví dụ: `var a = 10, b = 100.08;`
- Các phép toán trên kiểu số
 - `+`, `+=`, `-`, `-=`, `*`, `*=`, `/`, `/=`, `%` (chia lấy phần dư), `++` (phép tăng một đơn vị), `--` (phép giảm một đơn vị).
 - Các phép so sánh: `<` (nhỏ), `<=` (nhỏ hơn hay bằng), `>` (lớn), `>=` (lớn hơn hay bằng), `==` (bằng), `!=` (khác).
 - Phép `===`

Null, Undefined và NaN

```
null == undefined; // true  
null === undefined; // false
```

```
NaN == NaN // false  
NaN === NaN // false
```

Kiểu ký tự

- Các ký tự được nằm giữa 2 nháy đơn.
 - Ví dụ : **var ch= 'A', c = 'B';**
- Ngoài ra còn có các ký tự đặc biệt sau đây:

Ký tự	Ý nghĩa
\n	Xuống dòng mới
\t	Ký tự Tab
\r	Về đầu dòng
\b	Ký tự khoảng trắng

- Các phép toán trên ký tự: +, += (cộng 2 ký tự)
- Phép toán so sánh : >, >=, <, <=, ==, !=

Kiểu chuỗi

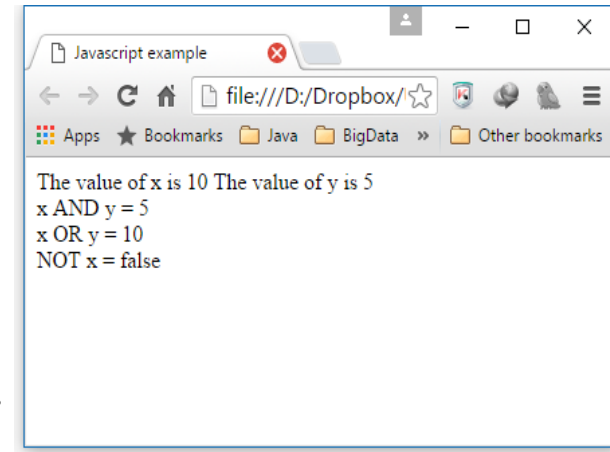
- Một hằng chuỗi được nằm giữa hai dấu nháy đôi "".
 - Ví dụ: **var chuoi = “Đây là kiểu Chuỗi”;**
- Các phép toán trên chuỗi
 - Phép nối chuỗi: +, +=
 - Phép so sánh: <, <= , >, >=, ==, !=

Kiểu luận lý

- Một biến có kiểu luận lý tồn tại 1 trong 2 trạng thái : **true**, **false**.
 - Ví dụ: **var t = true, f = false;**
- Các phép toán trên kiểu luận lý
 - Phép so sánh : <, <=, >, >=, ==, !=
 - Phép logic : && (và), || (hoặc), ! (phủ định).

Kiểu luận lý – Ví dụ

```
<script>
    var x = 10;
    var y = 5;
    document.write("The value of x is " + x + "
The value of y is " + y + "<br>");
    document.write("x AND y = " + (x &&
y) + "<br>");
    document.write("x OR y = " + (x || y) + "<br>");
    document.write("NOT x = " + (!x) + "<br>");
</script>
```



Mảng

- **Mảng một chiều:**

- **var A = new Array(10)**
- Mảng A nói trên có 10 phần tử, và chỉ số phần tử đầu tiên của mảng bắt đầu 0, muốn truy xuất đến phần tử có chỉ số i, ta dùng **A[i]**.

- **Mảng hai chiều**

- Khai báo A là mảng 2 chiều có 10 dòng, 20 cột.

```
var A = new Array(10), i = 0;  
for (i = 0; i < 10; i++)  
    A[i] = new Array(20);
```
- Để truy xuất đến phần tử có chỉ số dòng i, chỉ số cột j ta dùng **A[i][j]**.

Lệnh rẽ nhánh

- Câu lệnh điều kiện được dùng để kiểm tra điều kiện. Kết quả xác định câu lệnh hoặc khối lệnh được thực thi.
- Các câu lệnh điều kiện bao gồm:
 - **if**
 - **if..... else**
 - **Switch**

Câu lệnh if

- Dùng để xử lý lệnh khi biểu thức của if trả về giá trị true

```
if (biểu thức điều kiện)  
    Khối lệnh;
```

- Ví dụ:

```
a= "mon";  
if (a=="tue")  
    document.write("Hôm nay được nghỉ");
```

Câu lệnh if...else

```
if (biểu thức điều kiện)
    Khởi lệnh 1;
else
    Khởi lệnh 2;
```

- Ví dụ:

```
a=5;
if (a%2==0)
    document.write(a, "là số chẵn");
else
    document.write(a, "là số lẻ");
```

Câu lệnh switch...case

switch (*biến hoặc biểu thức*)

```
{  
    case giá trị 1:  
        Khởi lệnh 1;  
        break;  
    case giá trị 2:  
        Khởi lệnh 2;  
        break;  
    ...  
    default:  
        Khởi lệnh n;  
}
```

Câu lệnh switch...case – Ví dụ

```
var diem = "G";
switch (diem) {
    case "Y":
        document.write("Yếu");
        break;
    case "TB":
        document.write("Trung bình");
        break;
    case "K":
        document.write("Khá");
        break;
    case "G" :
        document.write("Giỏi");
        break;
    default:
        document.write("Xuất sắc")
}
```

Lệnh lặp

- Cấu trúc điều khiển lặp trong chương trình là các lệnh lặp
- Các kiểu lệnh lặp bao gồm:
 - for
 - do While
 - while

Lệnh lặp for

```
for (biểu thức1; biểu thức 2; biểu thức3){  
    Khối lệnh;  
}
```

- Biểu thức 1: khởi tạo giá trị
- Biểu thức 2: điều kiện
- Biểu thức 3: cập nhật giá trị
- Khối lệnh được thực hiện khi biểu thức 2 còn đúng.
- Ví dụ: **for** (i = 0; i < 10; i++)
 s+=2*i;

Lệnh lặp while

```
while (biểu thức điều kiện) {  
    Khối lệnh;  
}
```

- Khối lệnh được thực hiện khi biểu thức trong while còn đúng.
- Ví dụ:

```
i=0;  
while (i<20) {  
    s+=i;  
    i++;  
}
```

Hàm

- Dùng từ khóa **function** để khai báo hàm. Muốn trả về giá trị của hàm ta dùng từ khóa **return**
- Dạng thức khai báo chung:

```
function Tên_hàm(thamso1, thamso2,...)
{
    Khối lệnh;
}
```

- Hàm có giá trị trả về:

```
function Tên_hàm(thamso1, thamso2,...)
{
    Khối lệnh;
    return (value);
}
```

Ví dụ

```
function Add(x,y)
{
    return(x+y);
}

var t;

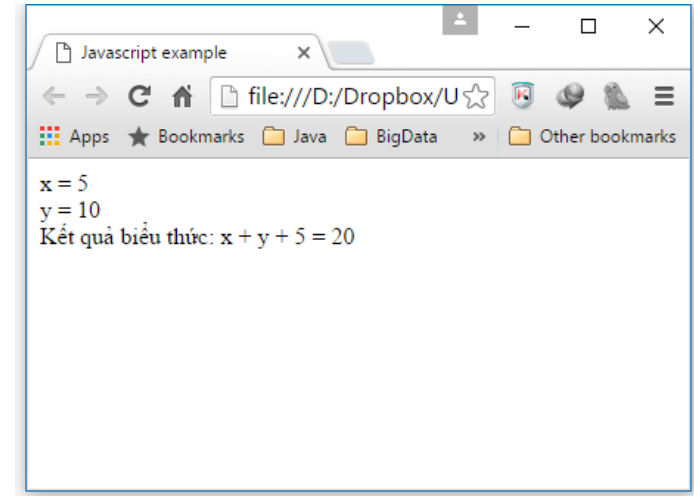
t = Add(4,8);

document.write(t);
```

Hàm eval

- Biến chuỗi thành biểu thức
- Biến chuỗi thành lệnh
- Biểu thức có thể bao gồm nhiều biến và nhiều thuộc tính của một đối tượng.

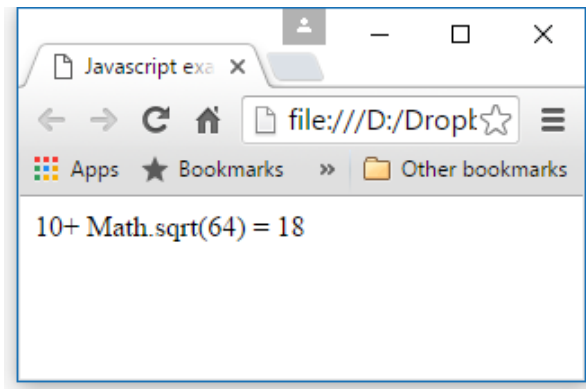
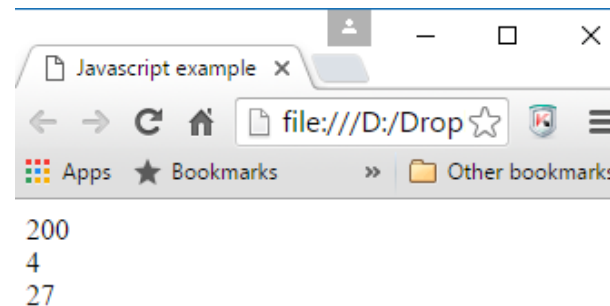
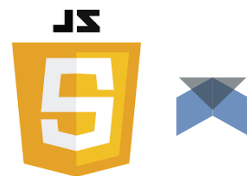
```
var x = 5;  
var y = 10;  
document.write(eval("x+y+5"));
```



Ví dụ hàm eval

```
<script>
    eval("x=10;y=20;document.write(x*y)");
    document.write("<br>" + eval("2+2"));
    document.write("<br>" + eval(x+17));
</script>
```

```
<script>
    var string= "10+ Math.sqrt(64)";
    document.write(string + "=" + eval(string));
</script>
```



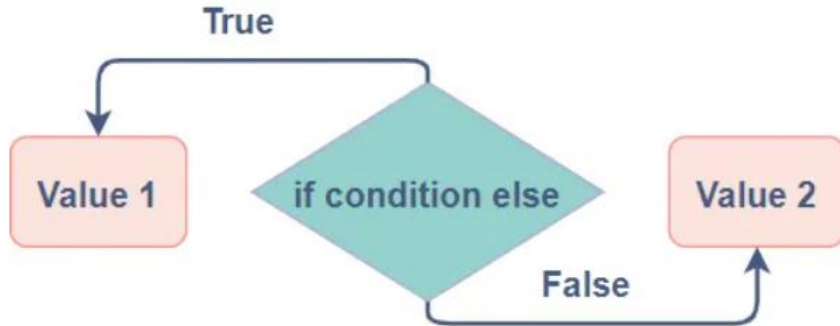
Toán tử ba ngôi

Conditional (ternary) operator

Cú pháp: **(Điều kiện) ? value1: value2**

Nếu biểu thức điều kiện đúng thì trả về giá trị value 1

Nếu biểu thức điều kiện sai thì trả về giá trị value 2



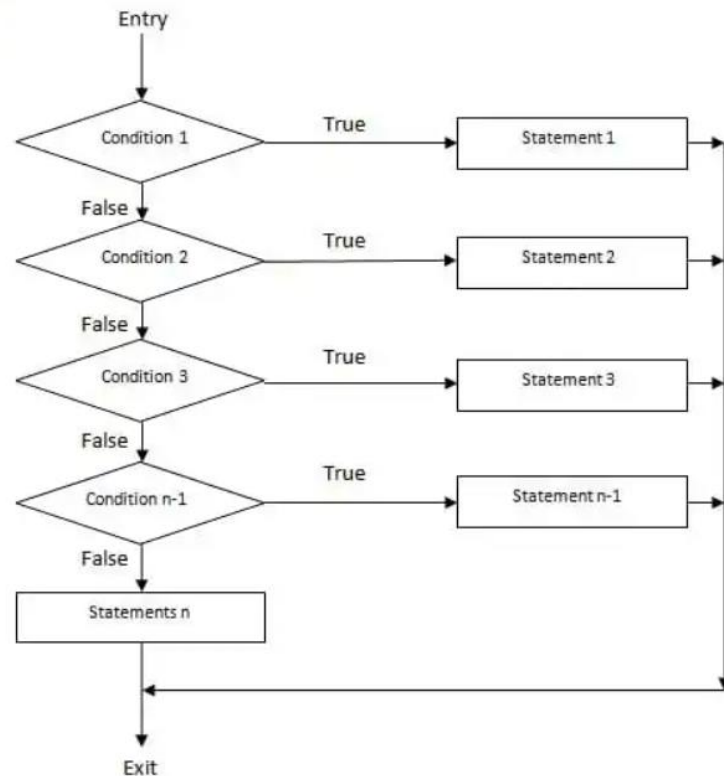
```

<script>
  var exp = 1;
  var salary = exp > 3 ? 1000 : 500;
  console.log(salary)    //500
</script>
  
```

Toán tử ba ngôi

Nó có thể được gọi “nối tiếp” theo cách sau, tương tự như với If – else If – else If – else nối tiếp nhau

```
<script>
  var exp = 2;
  var salary = exp < 1 ? 1000 :
    exp < 2 ? 1500 :
      exp < 3 ? 2000 : 3000;
  console.log(salary)    //2000
</script>
```



Nullish Coalescing Operator (??)

Toán tử điều kiện

Nullish chính là NULL or UNDEFINED

Cú pháp: **a ?? b**

Nếu a được định nghĩa thì kết quả sẽ là a

Nếu a là Nullish thì kết quả sẽ là b

```
let a = NULL
console.log(a??50) //50
console.log(a) //NULL
let a=10
let c=30
console.log(a??b??c??d) //10
```

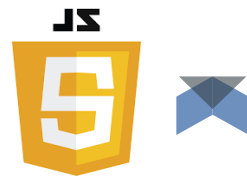
Logical Nullish Assignment (??=)

Cú pháp: **a ??= b** tương đương **a ??(a=b)**

Nếu a không phải là NULLISH (NULL or UNDEFINED) thì kết quả sẽ là a.

Nếu a là NULLISH thì kết quả sẽ là b, nhưng b sẽ có nhiệm vụ gán giá trị cho a. Hay gọi là assigned.

```
let a = NULL
console.log(a??=50) //50
console.log(a) //50
```



Các đối tượng cơ bản trong Javascript

- Lớp đối tượng là tập các đối tượng có cùng thuộc tính và hành vi (phương thức)
- Thuộc tính (biến) dùng để định nghĩa đối tượng.
- Phương thức (hàm): những hoạt động của lớp đối tượng



Thuộc tính và phương thức

- Để truy cập thuộc tính của đối tượng, chúng ta phải chỉ ra tên đối tượng và thuộc tính của nó:

`objectName.propertyName`

- Để truy cập phương thức của đối tượng, chúng ta phải chỉ ra tên đối tượng và thuộc tính của nó:

`objectName.method()`

- Tìm hiểu về Optional Chaining Operator (?.)
 - Cú pháp: `obj ?. prop`

Sử dụng đối tượng

- Khi tạo trang web, chúng ta cần sử dụng:
 - Các đối tượng trình duyệt
 - Các đối tượng có sẵn (thay đổi phụ thuộc vào ngôn ngữ kịch bản được sử dụng)
 - HTML elements
- Chúng ta cũng có thể tạo ra các đối tượng để sử dụng theo yêu cầu của mình.

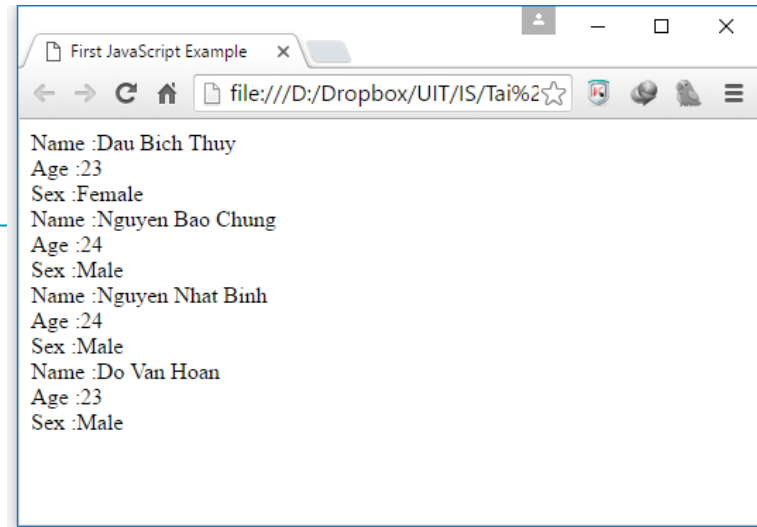
Từ khóa this

- Giá trị của nó chỉ ra đối tượng hiện hành và có thể có các thuộc tính chuẩn, chẳng hạn như tên, độ dài, và giá trị được áp dụng phù hợp

```
<script>
function person(first_name, last_name, age, sex){
    this.first_name=first_name;
    this.last_name=last_name;
    this.age=age;
    this.sex=sex;
    this.printStats=printStats;
}
function printStats(){
    document.write("Name : " + this.last_name + " " + this.first_name + "<br>" );
    document.write("Age : "+this.age+"<br>");
    document.write("Sex : "+this.sex+"<br>");
}
```

Từ khóa this

```
person1= new person("Thuy", "Dau Bich", "23", "Female");
person2= new person("Chung", "Nguyen Bao", "24", "Male");
person3= new person("Binh", "Nguyen Nhat", "24", "Male");
person4= new person("Hoan", "Do Van", "23", "Male");
person1.printStats();
person2.printStats();
person3.printStats();
person4.printStats();
</script>
```



Lệnh for...in

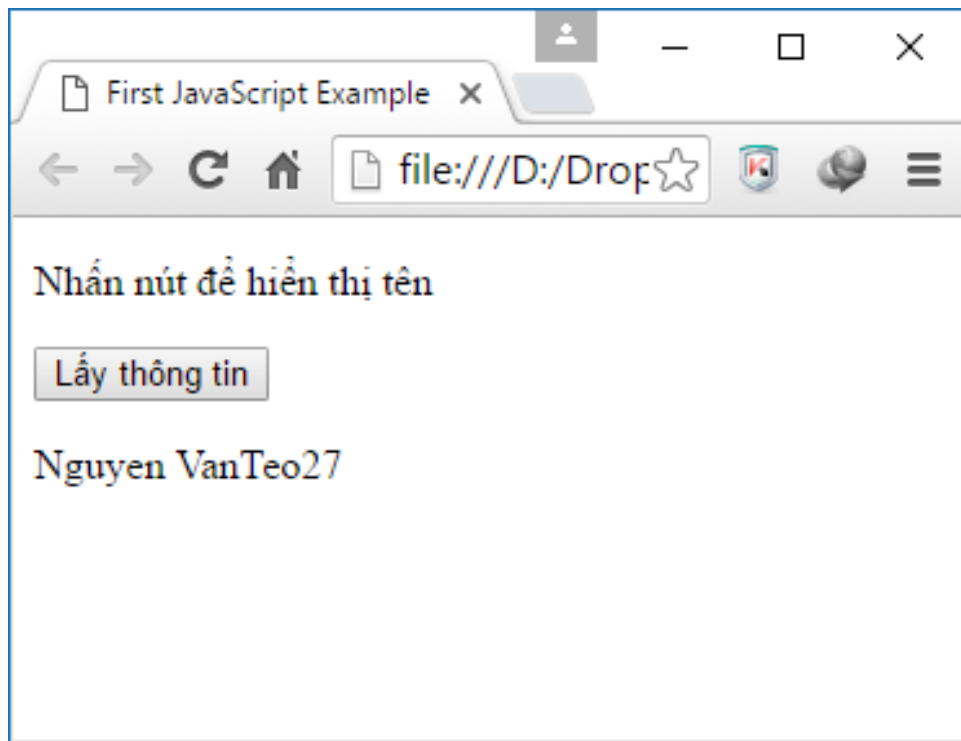
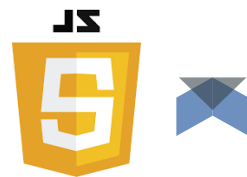
- Câu lệnh **for...in** được dùng để lặp mỗi thuộc tính của đối tượng hoặc mỗi phần tử của một mảng.
- Cú pháp:

```
for (variable in object) {  
    statements;  
}
```


Lệnh for...in

```
<script>
function myFunction() {
    var x;
    var txt="";
    var person={Ho:"Nguyen Van",Ten:"Teo",age:27};
    for (x in person){
        txt=txt + person[x];
    }
    document.getElementById("demo").innerHTML=txt;
}
</script>
</head>
<body>
<p>Nhấn nút để hiển thị tên</p>
<button onclick="myFunction()">Lấy thông tin</button>
<p id="demo">day la hien thi..</p>
</body>
```

Lệnh for...in



Câu lệnh with

- Câu lệnh **with** được dùng để thực thi tập hợp các lệnh mà các lệnh này dùng các phương thức của cùng một loại đối tượng.
- Thuộc tính được gán cho đối tượng đã được xác định trong câu lệnh with.
- Cú pháp:

```
with (object) {  
    statements;  
}
```

Câu lệnh with

```
<script>
with (document) {
    write("This is an exemple of the things that can be done <br>");
    write("With the with statment. <br>");
    write("This can really save some typing");
}
</script>
```

Kết quả:

This is an exemple of the things that can be done

With the with statment

This can really save some typing

Toán tử new

- Toán tử **new** được dùng để tạo ra một thực thể mới của một loại đối tượng
- Đối tượng có thể có sẵn hoặc do người dùng định nghĩa

objectName = new objectType(param1 [,param2] ... [,paramN])

- Trong đó:
 - **objectName** là tên của thực thể đối tượng mới.
 - **objectType** là một hàm quyết định loại của đối tượng. Ví dụ Array.
 - **Param[1, 2, ..]** là các giá trị thuộc tính của đối tượng.

Đối tượng string

- Đối tượng **String** được dùng để thao tác và làm việc với chuỗi văn bản.
- Chúng ta có thể tách chuỗi ra thành các chuỗi con và biến đổi chuỗi đó thành các chuỗi hoa hoặc thường trong một chương trình.
- Cú pháp tổng quát:

stringName.propertyName

hay

stringName.methodName

Đối tượng string

- Có 3 phương thức khác nhau để tạo ra chuỗi.
 - Dùng lệnh **var** và gán cho nó một giá trị.
`var string = "Test";`
 - Dùng một toán tử **(=)** có gán với một tên biến.
`string = "test";`
 - Dùng hàm khởi tạo **String(string)**
`string = new String("Test");`

Đối tượng Math

- Đối tượng **Math** có các thuộc tính và phương thức biểu thị các phép tính toán học nâng cao.

```
function doCalc(x) {  
    var a;  
    a = Math.PI * x * x;  
    alert ("The area of a circle with a radius of " + x + " is " + a);  
}
```


Đối tượng Date

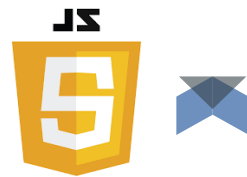
- Mô tả thông tin về: Ngày, Tháng, Năm, giờ, phút, giây của hệ thống.
- Ví dụ: `var now = new Date();`

Các hàm lấy ngày giờ trong đối tượng Date

Tên hàm	Mô tả
<code>getDate()</code>	Ngày: 1..31
<code>getDay()</code>	Ngày trong tuần: 0 (chủ nhật), 1 (thứ 2)
<code>getHours()</code>	Giờ: 0..23
<code>getMinutes()</code>	Phút: 0..59
<code>getMonth()</code>	Tháng: 0 (tháng 1)...11 (tháng 12)
<code>getSeconds()</code>	Giây: 0..59
<code>getTime()</code>	Giờ theo mili giây
<code>getFullYear()</code>	Năm

Ví dụ

```
<script>
    var now= new Date();
    var ngay="";
    ngay="hom nay la ngay"+ now.getDate();
    ngay+=" thang "+now.getMonth();
    ngay+=" nam " + now.getFullYear();
    document.write(ngay) ;
</script>
```



Xử lý sự kiện



Các sự kiện thông dụng

- Các sự kiện được hỗ trợ bởi hầu hết các đối tượng
 - onClick
 - onFocus
 - onChange
 - onBlur
 - onMouseOver
 - onMouseOut
 - onMouseDown
 - onMouseUp
 - onLoad
 - onSubmit
 - onResize
 -

Xử lý sự kiện cho các thẻ HTML

- Cú pháp:

```
<TAG eventHandler = "JavaScript Code">
```

- Ví dụ:

```
<body>
```

```
  <input type="button" name="Button1" value="OpenSesame!"  
    onClick="window.open('mydoc.html');">
```

```
</body>
```

- Lưu ý: Dấu “...” và ‘...’ (nháy kép và nháy đơn)

Xử lý sự kiện bằng hàm

```
<head>
  <script>
    function GreetingMessage() {
      window.alert("Welcome to my world");
    }
  </script>
</head>
<body onload="GreetingMessage()">
</body>
```

Xử lý sự kiện bằng thuộc tính

- Gán tên hàm xử lý cho 1 object event

```
object.eventhandler = function_name;
```

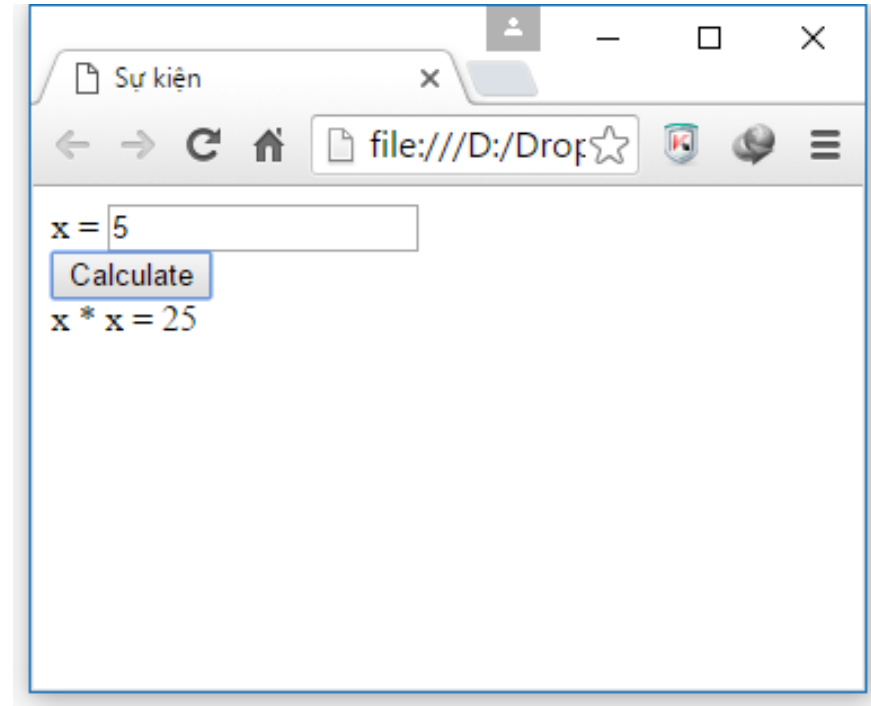
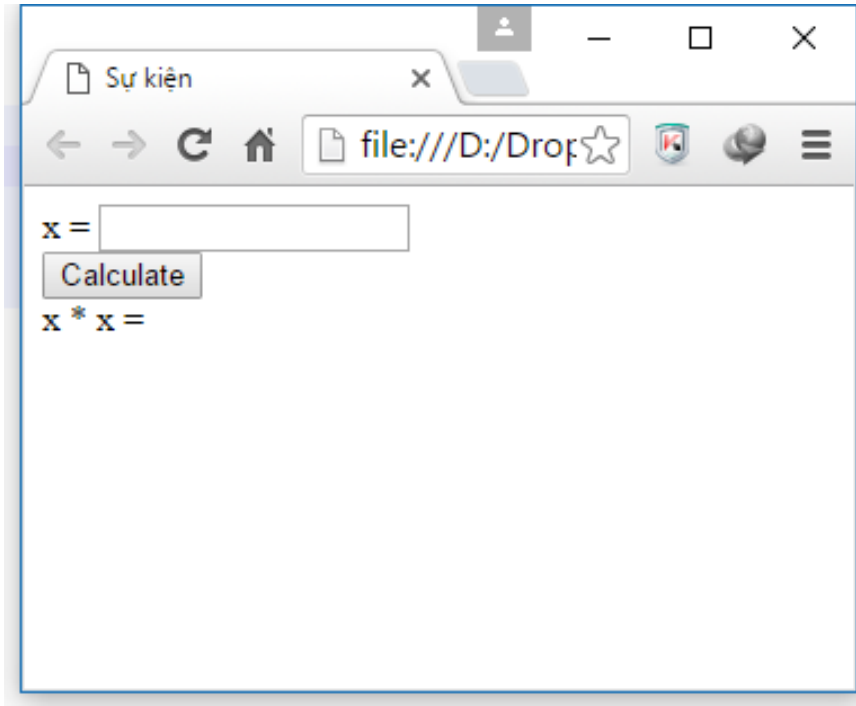
- Ví dụ:

```
<head>
  <script>
    function GreetingMessage() {
      window.alert("Welcome to my world");
    }
    window.onload = GreetingMessage();
  </script>
</head>
```


Ví dụ: sự kiện onclick

```
<script>
    function compute() {
        var x = frm.expr.value;
        result.innerHTML = x*x;
    }
</script>
<form name="frm">
    x = <input type="text" name="expr" size=15>
    <br>
    <input type="button" value="Calculate" onclick="compute()" ><br>
    x * x = <span id="result"></span>
</form>
```

Ví dụ: sự kiện onclick



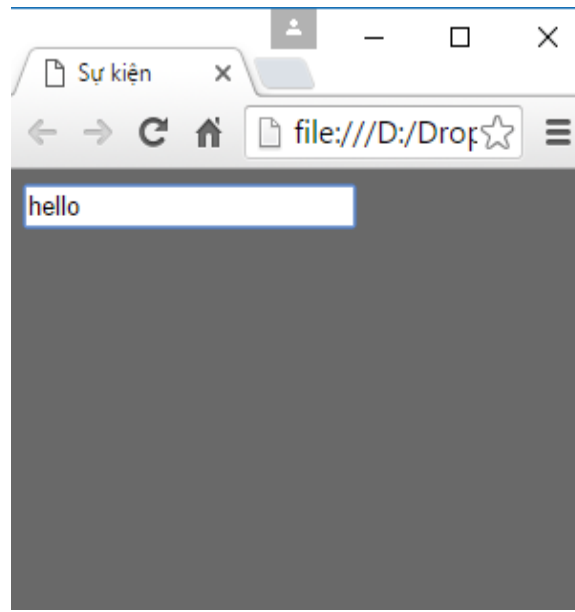
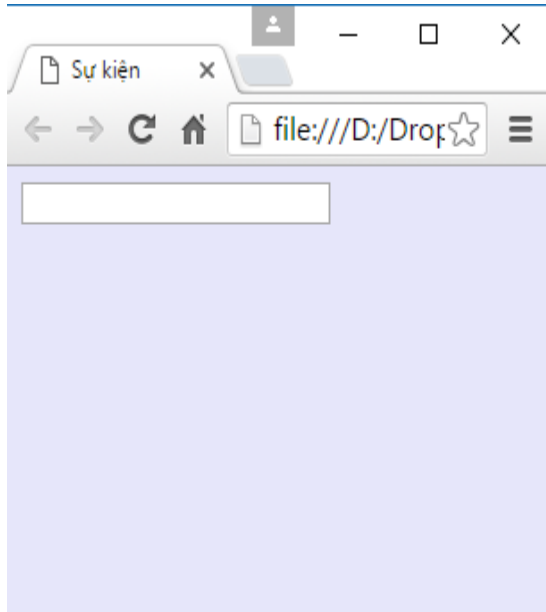
Ví dụ: sự kiện onFocus - onBlur

- Xảy ra khi một thành phần HTML nhận focus (onFocus) và mất focus (onBlur)
- Ví dụ:

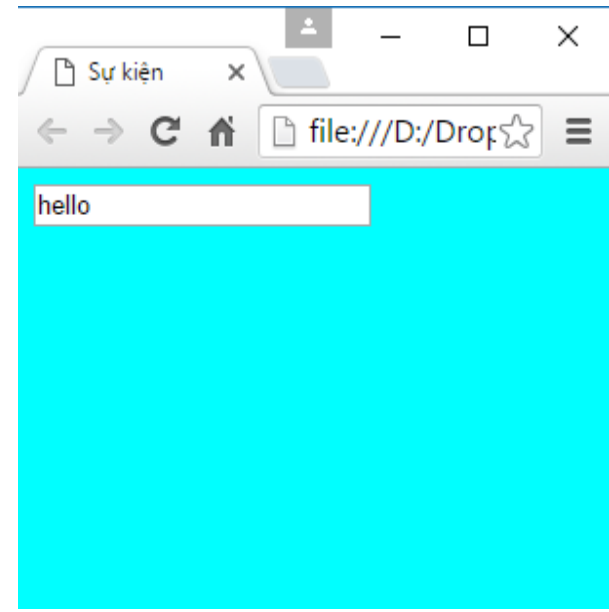
```
<body bgcolor="lavender">
  <form>
    <input type="text" name="myTextbox"
      onBlur="(document.bgColor='aqua') "
      onFocus="(document.bgColor='dimgray') " />
  </form>
</body>
```

Ví dụ: sự kiện onFocus - onBlur

onFocus



onBlur



Danh sách sự kiện

Event	Occurs when...
<code>onabort</code>	a user aborts page loading
<code>onblur</code>	a user leaves an object
<code>onchange</code>	a user changes the value of an object
<code>onclick</code>	a user clicks on an object
<code>ondblclick</code>	a user double-clicks on an object
<code>onfocus</code>	a user makes an object active
<code>onkeydown</code>	a keyboard key is on its way down
<code>onkeypress</code>	a keyboard key is pressed
<code>onkeyup</code>	a keyboard key is released

Danh sách sự kiện

Event	Occurs when...
<code>onload</code>	a page is finished loading.
<code>onmousedown</code>	a user presses a mouse-button
<code>onmousemove</code>	a cursor moves on an object
<code>onmouseover</code>	a cursor moves over an object
<code>onmouseout</code>	a cursor moves off an object
<code>onmouseup</code>	a user releases a mouse-button
<code>onreset</code>	a user resets a form
<code>onselect</code>	a user selects content on a page
<code>onsubmit</code>	a user submits a form
<code>onunload</code>	a user closes a page



DOM HTML với Javascript



Đối tượng HTML DOM

- DOM = Document Object Model
- Là tập hợp các đối tượng HTML chuẩn được dùng để truy xuất và thay đổi thành phần HTML trong trang web (thay đổi nội dung tài liệu của trang)
- Một số đối tượng của DOM: window, document, history, link, form, frame, location, event, ...

Đối tượng Window - DOM

- Là thể hiện của đối tượng **cửa sổ trình duyệt**
- Tồn tại khi mở 1 tài liệu HTML
- Sử dụng để truy cập thông tin của các đối tượng trên cửa sổ trình duyệt (tên trình duyệt, phiên bản trình duyệt, thanh tiêu đề, thanh trạng thái ...)

Đối tượng Window - DOM

- Properties

- document
- event
- history
- location
- name
- navigator
- screen
- status

- Methods

- alert
- confirm
- prompt
- blur
- close
- focus
- open

Đối tượng Window - DOM

- Ví dụ:

```
<html>
```

```
<body>
```

```
<script>
```

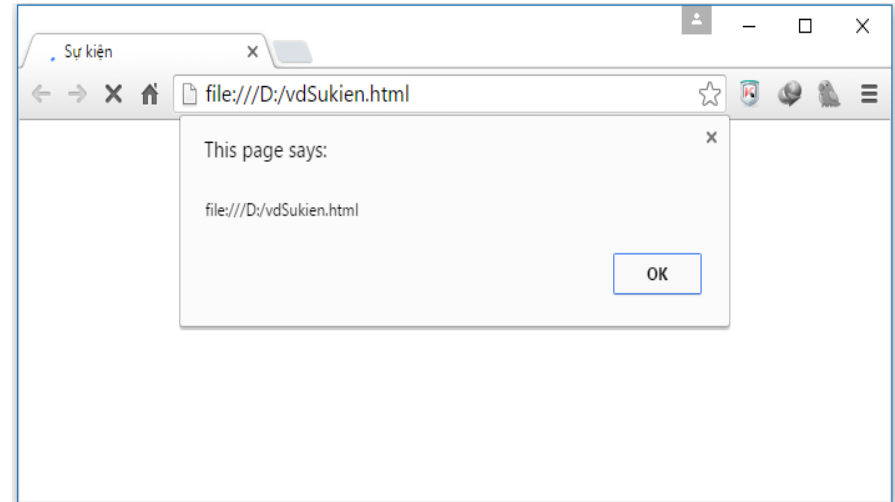
```
    var curURL = window.location;
```

```
    window.alert (curURL);
```

```
</script>
```

```
</body>
```

```
</html>
```





Đối tượng Document - DOM

- Biểu diễn cho **nội dung trang HTML** đang được hiển thị trên trình duyệt
- Dùng để lấy thông tin về tài liệu, các thành phần HTML và xử lý sự kiện



Đối tượng Document - DOM

• Properties

- document
- aLinkColor
- bgColor
- body
- fgColor
- linkColor
- title
- URL
- vlinkColor
- forms[]
- images[]
- childNodes[]
- documentElement
- cookie
-

• Methods

- close
- open
- createTextNode("text")
- createElement("HTMLtag")
- getElementById("id")
- getElementsByName(ten)
- getElementsByTagName(Ten_The)
- document.tenform.tencontrol.t
huoctinh

Đối tượng Document - DOM

- Truy xuất đến các form:
 - `document.tên_form`
- Truy xuất các đối tượng trong form:
 - `Tên_form.Tên_DT`
- Thuộc tính đối tượng:
 - `Tên_thuộc_tính`

Đối tượng Document - DOM

- `getElementById(id1)`
 - Trả về node có giá trị thuộc tính `id`
- Ví dụ:

```
<p id="id1" >
    some text
</p>
<script>
    var node = document.getElementById("id1");
    var nodeName = node.nodeName; // p
    var nodeType = node.nodeType; // 1
    var nodeValue = node.nodeValue; // null
    var text = node.innerText; // some text
</script>
```

Text Node

Đối tượng Document - DOM

- `createElement (nodeName)`
 - Cho phép tạo ra **1 node HTML** mới tùy theo đối số **nodeName** đầu vào

- Ví dụ:

```
<script>  
    var imgNode = document.createElement("img") ;  
    imgNode.src = "images/test.gif";  
</script>  
  

```


Đối tượng Document - DOM

- `createTextNode (content)`
 - Cho phép tạo ra **nội dung** cho một **node**,
- Ví dụ:

```
<script>  
    var textNode=document.createTextNode("New text");  
    var pNode = document.createElement("p");  
    pNode.appendChild(textNode);  
</script>
```

```
<p>New text</p>
```

Đối tượng Document - DOM

- `appendChild(newNode)`
 - Chèn node mới (**newNode**) vào cuối danh sách các node con của một node
- Ví dụ:

```
<p id="id1" >
    some text
</p>
<script>
    var pNode = document.getElementById("id1");
    var imgNode = document.createElement("img");
    imgNode.src = "images/test.gif";
    pNode.appendChild(imgNode);
</script>
<p id="id1" >
    some text
</p>
```

Đối tượng Document - DOM

- `innerHTML`
 - Chỉ định **nội dung HTML** bên trong một node.

- Ví dụ:

```
<p id="para1" >
    Some text
</p>
<script>
    var theElement = document.getElementById("para1");
    theElement.innerHTML = "Some <b> new </b> text";
</script>
```

Kết quả : Some **new** text

Đối tượng Document - DOM

- `innerText`
 - Tương tự `innerHTML`, tuy nhiên bất kỳ nội dung nào đưa vào cũng được xem như là **text** hơn là các thẻ **HTML**.
- Ví dụ:

```
<script>
```

```
var theElement=document.getElementById("para1");
```

```
theElement.innerText = "Some <b> new </b> text";
```

```
</script>
```

```
// Kết quả hiển thị trên trình duyệt
```

```
"Some <b> new </b> text"
```

Đối tượng Document - DOM

- Outer là phần inner và bản thân đối tượng chứa ID.
- Outer gồm có
 - outerHTML lấy nội dung text và tag HTML của cả đối tượng ID
 - outerText : lấy nội dung text
- Inner là nội dung chứa bên trong của đối tượng chứa ID.
- Inner gồm có
 - InnerHTML lấy nội dung text và tag HTML bên trong đối tượng ID
 - innerText: chỉ lấy nội dung text bên trong đối tượng ID

<Div ID=Intro>Monitor SAMSUNG</Div>

inner

outer

Đối tượng Document - DOM

- `s1=Intro.outerText`
`s2=Intro.innerText`
 thì `s1` và `s2` đều nhận giá trị Monitor SAMSUNG
- `s1=Intro.outerHTML`
`s2=Intro.innerHTML`
 Thì `s1=<Div Id=Intro>MonitorSAMSUNG</Div>`
 Và `s2=Monitor SAMSUNG`

<Div ID=Intro>Monitor SAMSUNG</Div>

inner

outer

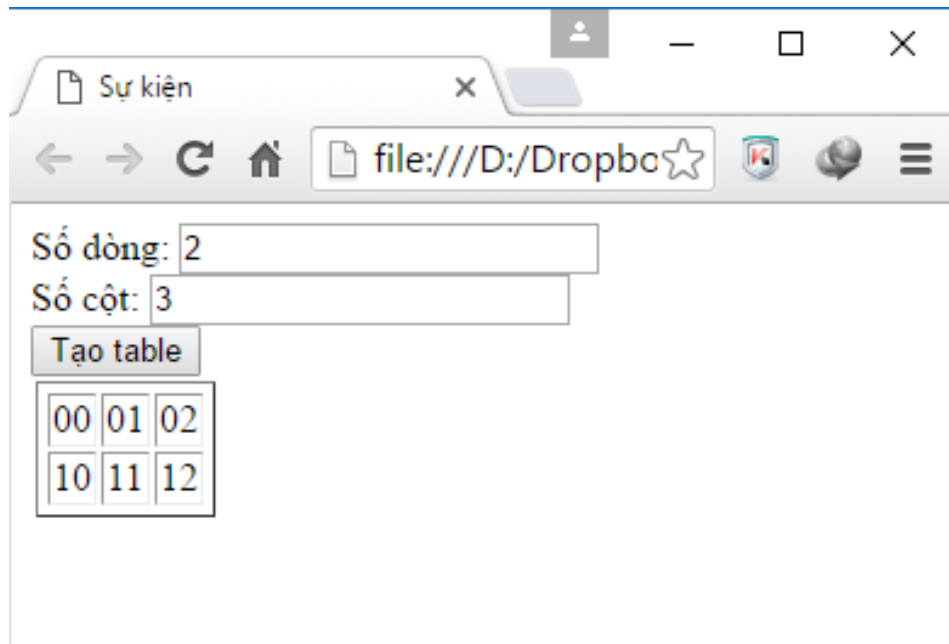


Một số ví dụ



Ví dụ: Dynamic table

- Trang web cho phép tạo table có **số dòng**, **số cột** do người dùng nhập.



Số dòng:

Số cột:

00	01	02
10	11	12

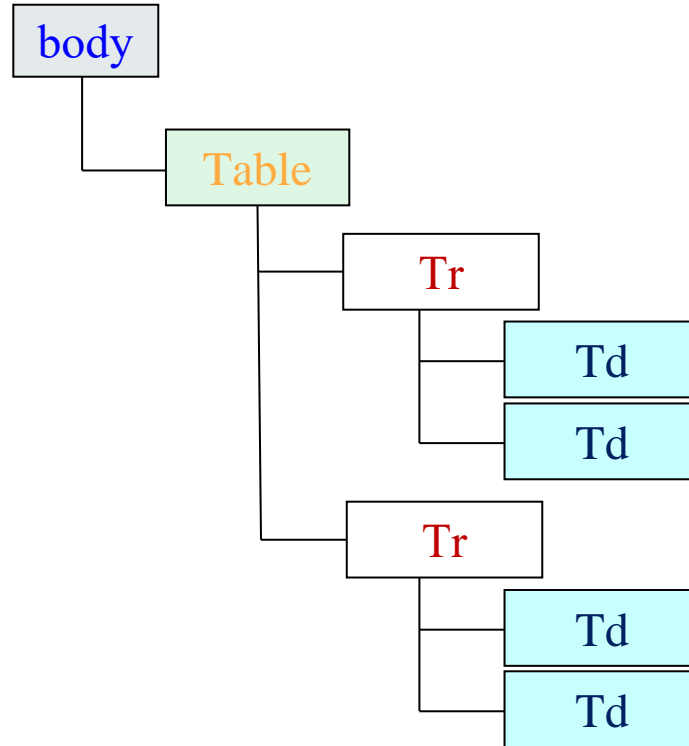
Ví dụ: Dynamic table

<table>

```
<tr>
  <td> 12 </td>
  <td> 13 </td>
</tr>
```

```
<tr>
  <td> 21 </td>
  <td> 22 </td>
</tr>
```

</table>



Ví dụ: Dynamic table

- `document.createElement(...)` :Tạo một đối tượng thẻ DOM HTML
- `object.appendChild(...)` : Thêm một đối tượng thẻ DOM HTML như là nút con.

Ví dụ: Dynamic table (1)

```
<body>
<form name="frm">
Số dòng: <input type="text" name="txtdong"><br>
Số cột: <input type="text" name="txtcot"><br>
<input type="button" name="taoTable" value="Tao table"
onclick="createTable(document.getElementById('divTable'))">
</form>
<div id="divTable">
</div>
</body>
```

Ví dụ: Dynamic table (1)

```
function createTable(divTable){
    var tagTable = document.createElement("table");
    tagTable.border=1;
    var nDong = frm.txtdong.value;
    var nCot = frm.txtcot.value;
    for (i=0;i<nDong;i++){
        var tr = document.createElement("tr");
        for (j=0;j<nCot;j++){
            var td = document.createElement("td");
            var textTd = document.createTextNode(i+" "+j);
            td.appendChild(textTd);
            tr.appendChild(td);
        }
        tagTable.appendChild(tr);
    }
    divTable.appendChild(tagTable);
}
```

Ví dụ: Dynamic table (2)

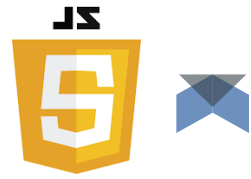
```
<body>  
<form name="frm">  
Số dòng: <input type="text" name="txtdong"><br>  
Số cột: <input type="text" name="txtcot"><br>  
<input type="button" name="taoTable" value="Tao table" onclick="createTable()">  
</form>  
<table id="divTable">  
</table>  
</body>
```

Ví dụ: Dynamic table (2)

```
function createTable() {
    var tagTable = document.getElementById("divTable");
    tagTable.border = 3;
    var nDong = frm.txtdong.value;
    var nCot = frm.txtcot.value;
    for (i=0; i<nDong; i++) {
        var tr = document.createElement("tr");
        for (j=0; j<nCot; j++) {
            var td = document.createElement("td");
            var textTd = document.createTextNode(i+" "+j);
            td.appendChild(textTd);
            tr.appendChild(td);
        }
        tagTable.appendChild(tr);
    }
}
```

Ví dụ

- Hiệu ứng chữ chạy trong trang web



Lý thuyết

- Lệnh **setTimeout(f, n)** quy định sau khoảng thời gian n mili giây hàm f sẽ được gọi. (f là chuỗi lưu lệnh cần thực hiện)
- Vài lệnh khác cùng nhóm setTimeout
 - `timeID = setTimeout(f, n)`
 - **clearTimeout(timeID)**: Hủy setTimeout
 - **intervalID = setInterval(f, n)**: Sau mỗi khoảng thời gian n ms lệnh f được gọi và tiếp tục cho đến khi thoát chương trình
 - **clearInterval(intervalID)**: Hủy interval.
- Giả sử str là một chuỗi ta có
 - **str.length**: Thuộc tính cho biết độ dài chuỗi
 - **str.substr(i, n)**: lấy ra n ký tự kể từ vị trí thứ i (Ký tự đầu tiên được đánh số là 0)

Giải thuật

- Ý tưởng giải thuật
 - Để có được cảm giác chữ chạy trong một thẻ `<div>` hoặc `<p>` ta cần copy ký tự đầu của dòng chữ hiện tại đưa xuống cuối cùng và lặp lại như vậy sau mỗi khoảng thời gian.
- Giải thuật: Giả sử ta có biến `str` đang lưu chuỗi cần chạy. Công việc thực hiện như sau:
 - **B1:** Thể hiện chuỗi `str` trong thẻ `<div>`. Chuyển sang bước 2
 - **B2:** Chuyển ký tự đầu của **str** về cuối (bằng cách gán `str = xâu con kể từ vị trí thứ 2 của str đến cuối + ký tự đầu tiên của str`). Chuyển sang bước 3
 - **B3:** Trễ một khoảng thời gian rồi quay lại bước 1

Mã lệnh

```
<script>
  var str= 'Dai Hoc Cong Nghe Thong Tin';
  for (i=str.length; i<100; i++){
    str = str + ' ';
  }
  function ChuChay(){
    var tagDiv = document.getElementById("chaychu");
    tagDiv.innerText = str;
    str = str.substr(1,str.length-1) + str.substr(0,1);
    setTimeout(ChuChay,100);
  }
</script>
<body onload="ChuChay()" >
<div id="chaychu">
</div>
</body>
```

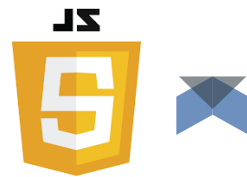
Phát triển



- Thay bằng nhiều dòng chữ chạy khác nhau (sử dụng mảng để lưu trữ)
- Chữ chạy theo nhiều cách khác nhau
- Cho chữ chạy trên thanh tiêu đề (dùng **document.title**)
- Cho chữ chạy trên một đối tượng khác



Cơ chế Xử lý Bất đồng bộ trong JS



Xử lý Đồng bộ và Bất đồng bộ

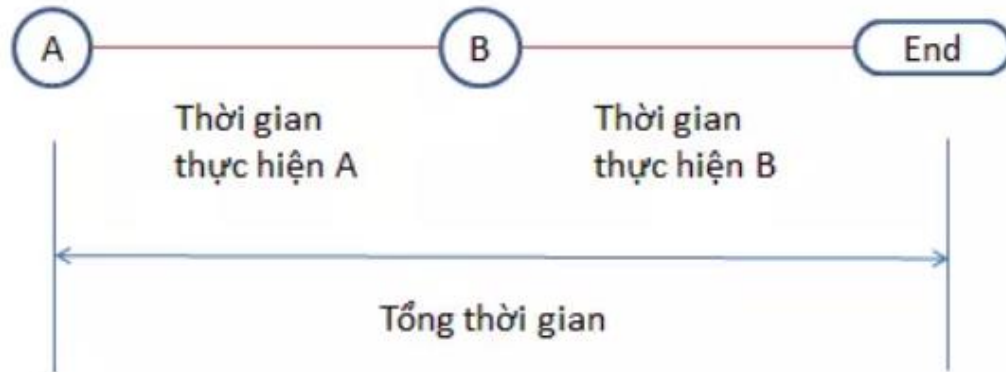
- Xử lý đồng bộ: Synchronous
- Xử lý bất đồng bộ: Asynchronous



Xử lý Đồng bộ và Bất đồng bộ

• Xử lý đồng bộ (Synchronous)

- Thực hiện các công việc một cách tuần tự, công việc này xong thì mới được thực hiện các công việc khác.
- Ví dụ có 2 công việc A và B thì khi có nghĩa là A thực hiện xong trước rồi mới tới lượt B.



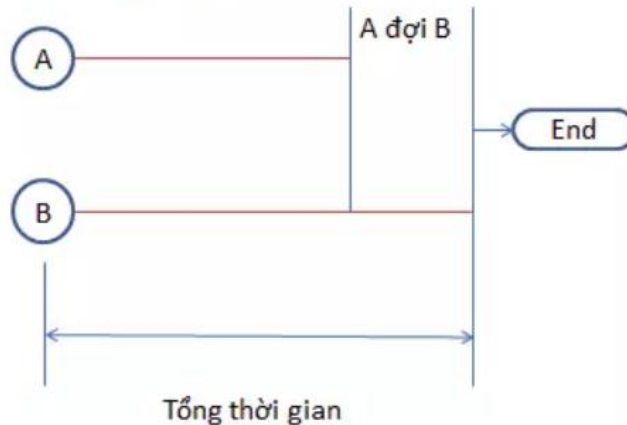
Xử lý Đồng bộ và Bất đồng bộ

- Điều này sẽ ảnh hưởng đến hiệu suất của người dùng.
- Giả sử một request gửi lên server yêu cầu server thực hiện chức năng như import file hoặc đọc ghi file thì lúc này server sẽ mất nhiều thời gian để xử lý những việc này. Đồng nghĩa với việc trong lúc server thực hiện chức năng đó thì sẽ không thể thực hiện thêm một hành động nào khác.

Xử lý Đồng bộ và Bất đồng bộ

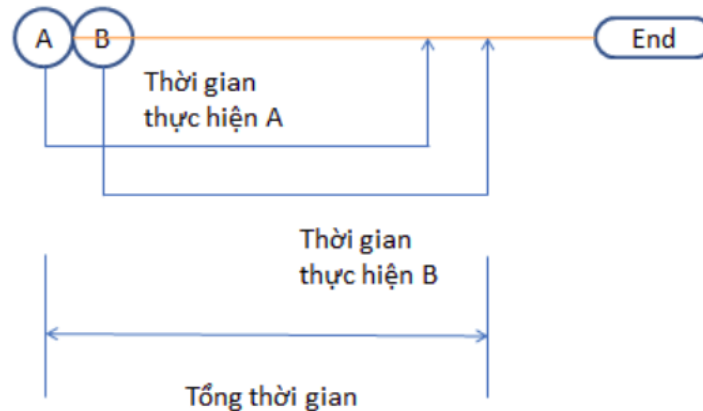
• Multi-thread

- Để khắc phục tình trạng nói trên, các ngôn ngữ lập trình như C/C++, Java,... sẽ sử dụng cơ chế đa luồng (multi-thread). Nghĩa là mỗi công việc tốn thời gian sẽ được thực hiện trên một thread riêng biệt mà không can thiệp vào thread chính. Vẫn có thể thực hiện các công việc tốn thời gian mà vẫn có thể bắt các sự kiện ở thread chính.



Xử lý Đồng bộ và Bất đồng bộ

- **Xử lý bất đồng bộ: Asynchronous**
- Javascript là ngôn ngữ Single thread
- Với cách xử lý bất đồng bộ, khi A bắt đầu thực hiện, chương trình tiếp tục thực hiện B mà không đợi A kết thúc. Chúng ta sẽ cung cấp một phương thức để chương trình thực hiện khi A hoặc B kết thúc.



Xử lý Đồng bộ và Bất đồng bộ

- Cơ chế thực hiện việc này trong JavaScript có thể là sử dụng
 - **Callback**
 - **Promise**
 - **Async/await**

Q & A



Cảm ơn đã theo dõi

Hy vọng cùng nhau đi đến thành công.

