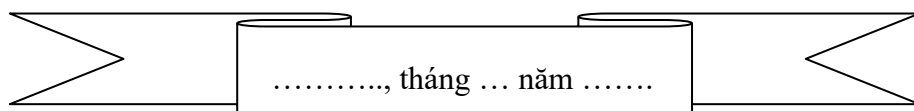




Giáo trình JavaScript



Giáo trình JavaScript

CH-ÔNG 1 LỜI NÓI ĐẦU.....	4
CH-ÔNG 2 NHẬP MÔN JAVASCRIPT.....	5
2.1. Nhúng JavaScript vào file HTML.....	5
2.3. Thẻ <NoScript> và </NOSCRIPT>.....	6
2.3. Hiển thị một dòng text	7
2.4. Giao tiếp với ng-ời sử dụng.....	9
2.5. Điểm lại các lệnh và mở rộng	12
CH-ÔNG 3 BIẾN TRONG JAVASCRIPT.....	13
3.1. Biến và phân loại biến.....	13
3.2. Biểu diễn từ tổ trong JavaScript	13
3.3. Kiểu dữ liệu.....	13
1.1.1. Kiểu nguyên (Interger).....	14
1.1.2. Kiểu dấu phẩy động (Floating Point)	14
1.1.3. Kiểu logic (Boolean)	15
1.1.4. Kiểu chuỗi (String)	15
2. XÂY DỰNG CÁC BIỂU THỨC TRONG JAVASCRIPT	16
ĐỊNH NGHĨA VÀ PHÂN LOẠI BIỂU THỨC	16
CÁC TOÁN TỬ (OPERATOR).....	16
2.1.1. Gán	16
2.1.2. So sánh.....	16
2.1.3. Số học.....	17
2.1.4. Chuỗi	17
2.1.5. Logic	17
2.1.6. Bitwise	18
BÀI TẬP.....	19
2.1.7. Câu hỏi	19
2.1.8. Trả lời.....	19
3. CÁC LỆNH	20
CÂU LỆNH ĐIỀU KIỆN.....	20
CÂU LỆNH LẶP.....	21
3.1.1. Vòng lặp for	21
3.1.2. while.....	22
3.1.3. Break	22
3.1.4. continue	23

CÁC CÂU LỆNH THAO TÁC TRÊN ĐỐI TƯỢNG	23
3.1.5. for...in.....	23
3.1.6. new	25
3.1.7. this	26
3.1.8. with	27
CÁC HÀM (FUNCTIONS).....	28
CÁC HÀM CÓ SẴN	30
3.1.9. eval	30
3.1.10. parseInt.....	31
3.1.11. parseFloat.....	32
MẢNG (ARRAY)	33
SỰ KIỆN	35
BÀI TẬP	38
3.1.12. Câu hỏi	38
3.1.13. Trả lời.....	40
4. CÁC ĐỐI TƯỢNG TRONG JAVASCRIPT	43
ĐỐI TƯỢNG NAVIGATOR	45
ĐỐI TƯỢNG WINDOW.....	46
4.1.1. Các thuộc tính.....	46
4.1.2. Các ph-ơng thức	47
4.1.3. Các ch-ơng trình xử lý sự kiện.....	48
ĐỐI TƯỢNG LOCATION.....	48
ĐỐI TƯỢNG FRAME.....	49
4.1.4. Các thuộc tính.....	49
4.1.5. Các ph-ơng thức	49
4.1.6. Sử dụng Frame	49
ĐỐI TƯỢNG DOCUMENT	51
4.1.7. Các thuộc tính.....	52
4.1.8. Các ph-ơng thức	52
ĐỐI TƯỢNG ANCHORS	52
ĐỐI TƯỢNG FORMS	53
4.1.9. Các thuộc tính.....	53
4.1.10. Các ph-ơng thức	53
4.1.11. Các ch-ơng trình xử lý sự kiện.....	53
ĐỐI TƯỢNG HISTORY	54
4.1.12. Các thuộc tính.....	54
4.1.13. Các ph-ơng thức	54
ĐỐI TƯỢNG LINKS.....	54

4.1.14.	Các thuộc tính.....	54
4.1.15.	Các chương trình xử lý sự kiện.....	55
ĐỐI TƯỢNG MATH		55
4.1.16.	Các thuộc tính.....	55
4.1.17.	Các phương thức	55
ĐỐI TƯỢNG DATE		56
4.1.18.	Các phương thức	56
ĐỐI TƯỢNG STRING		57
4.1.19.	Các phương thức	57
CÁC PHẦN TỬ CỦA ĐỐI TƯỢNG FORM		58
4.1.20.	Thuộc tính type	59
4.1.21.	Phần tử button	59
4.1.22.	Phần tử checkbox	60
4.1.23.	Phần tử File Upload	62
4.1.24.	Phần tử hidden	62
4.1.25.	Phần tử Password	62
4.1.26.	Phần tử radio	63
4.1.27.	Phần tử reset	65
4.1.28.	Phần tử select.....	65
5.	MÔ HÌNH ĐỐI TƯỢNG (OBJECT MODEL)	75
ĐỐI TƯỢNG VÀ THUỘC TÍNH		75
TẠO CÁC ĐỐI TƯỢNG MỚI		76
5.1.1.	Sử dụng khởi tạo đối tượng.....	76
5.1.2.	Sử dụng một hàm xây dựng(Constructor Function).....	77
5.1.3.	Lập mục lục cho các thuộc tính của đối tượng	78
5.1.4.	Định nghĩa thêm các thuộc tính cho một kiểu đối tượng.....	79
5.1.5.	Định nghĩa các cách thức	79
5.1.6.	Sử dụng cho các tham chiếu đối tượng (Object References).....	80
5.1.7.	Xoá đối tượng	80
6.	BẢNG TỔNG KẾT CÁC TỪ KHOÁ	81
7.	TỔNG KẾT	82

CH-ƠNG 1 LỜI NÓI ĐẦU

Với HTML and Microsoft FrontPage bạn đã biết cách tạo ra trang Web U tuy nhiên chỉ mới ở mức biểu diễn thông tin chứ ch-a phải là các trang Web động có khả năng đáp ứng các sự kiện từ phía ng-ời dùng. Hãng Netscape đã đ-a ra ngôn ngữ script có tên là LiveScript để thực hiện chức năng này. Sau đó ngôn ngữ này đ-ọc đổi tên thành JavaScript

để tận dụng tính đại chúng của ngôn ngữ lập trình Java. Mặc dù có những điểm t-ong đồng giữa Java và JavaScript, nh-ng chúng vẫn là hai ngôn ngữ riêng biệt.

JavaScript là ngôn ngữ d-ới dạng script có thể gắn với các file HTML. Nó không đ-ọc biên dịch mà đ-ọc trình duyệt diễn dịch. Không giống Java phải chuyển thành các mã để biên dịch, trình duyệt đọc JavaScript d-ới dạng mã nguồn. Chính vì vậy bạn có thể dễ dàng học JavaScript qua ví dụ bởi vì bạn có thể thấy cách sử dụng JavaScript trên các trang Web.

JavaScript là ngôn ngữ dựa trên đối t-ượng, có nghĩa là bao gồm nhiều kiểu đối t-ượng, ví dụ đối t-ượng **Math** với tất cả các chức năng toán học. Tuy vậy JavaScript không là ngôn ngữ h-óng đối t-ượng nh- C++ hay Java do không hỗ trợ các lớp hay tính thừa kế.

JavaScript có thể đáp ứng các sự kiện nh- tải hay loại bỏ các form. Khả năng này cho phép JavaScript trở thành một ngôn ngữ script động.

Giống với HTML và Java, JavaScript đ-ọc thiết kế độc lập với hệ điều hành. Nó có thể chạy trên bất kỳ hệ điều hành nào có trình duyệt hỗ trợ JavaScript. Ngoài ra JavaScript giống Java ở khía cạnh an ninh: JavaScript không thể đọc và viết vào file của ng-ời dùng.

Các trình duyệt web nh- Netscape Navigator 2.0 trở đi có thể hiển thị những câu lệnh JavaScript đ-ọc nhúng vào trang HTML. Khi trình duyệt yêu cầu một trang, server sẽ gửi đầy đủ nội dung của trang đó, bao gồm cả HTML và các câu lệnh JavaScript qua mạng tới client. Client sẽ đọc trang đó từ đầu đến cuối, hiển thị các kết quả của HTML và xử lý các câu lệnh JavaScript khi nào chúng xuất hiện.

Các câu lệnh JavaScript đ-ọc nhúng trong một trang HTML có thể trả lời cho các sự kiện của ng-ời sử dụng nh- kích chuột, nhập vào một form và điều h-óng trang. Ví dụ bạn có thể kiểm tra các giá trị thông tin mà ng-ời sử dụng đ-a vào mà không cần đến bất cứ một quá trình truyền trên mạng nào. Trang HTML với JavaScript đ-ọc nhúng sẽ kiểm tra các giá trị đ-ọc đ-a vào và sẽ thông báo với ng-ời sử dụng khi giá trị đ-a vào là không hợp lệ.

Mục đích của phần này là giới thiệu về ngôn ngữ lập trình JavaScript để bạn có thể viết các script vào file HTML của mình.

CH-ƠNG 2 NHẬP MÔN JAVASCRIPT

2.1.Nhúng JavaScript vào file HTML

Bạn có thể nhúng JavaScript vào một file HTML theo một trong các cách sau đây:

- ã Sử dụng các câu lệnh và các hàm trong cặp thẻ **<SCRIPT>**
- ã Sử dụng các file nguồn JavaScript
- ã Sử dụng một biểu thức JavaScript làm giá trị của một thuộc tính HTML
- ã Sử dụng thẻ sự kiện (event handlers) trong một thẻ HTML nào đó

Trong đó, sử dụng cặp thẻ **<SCRIPT>...</SCRIPT>** và nhúng một file nguồn JavaScript là đọc sử dụng nhiều hơn cả.

2.1.1.Sử dụng thẻ SCRIPT

Script đọc đưa vào file HTML bằng cách sử dụng cặp thẻ **<SCRIPT>** và **<\SCRIPT>**. Các thẻ **<SCRIPT>** có thể xuất hiện trong phần **<HEAD>** hay **<BODY>** của file HTML. Nếu đặt trong phần **<HEAD>**, nó sẽ đọc tải và sẵn sàng tr-ớc khi phần còn lại của văn bản đọc tải.

Thuộc tính duy nhất đọc định nghĩa hiện thời cho thẻ **<SCRIPT>** là **"LANGUAGE="** dùng để xác định ngôn ngữ script đọc sử dụng. Có hai giá trị đọc định nghĩa là "JavaScript" và "VBScript". Với ch-ơng trình viết bằng JavaScript bạn sử dụng cú pháp sau :

```
<SCRIPT LANGUAGE="JavaScript">
// INSERT ALL JavaScript HERE
</SCRIPT>
```

Chú ý:

Ghi chú không đọc đặt trong cặp thẻ <U vớU> nh- ghi chú trong file HTML. Cú pháp của JavaScript t-ơng tự cú pháp của C nên có thể sử dụng // hay /* ... */.

Điểm khác nhau giữa cú pháp viết các ghi chú giữa HTML và JavaScript là cho phép bạn ẩn các mã JavaScript trong các ghi chú của file HTML, để các trình duyệt cũ không hỗ trợ cho JavaScript có thể đọc đọc nó nh- trong ví dụ sau đây:

```
<SCRIPT LANGUAGE="JavaScript">
<!U'U' From here the JavaScript code hidden
// INSERT ALL JavaScript HERE
// This is where the hidden ends U'U'>
</SCRIPT>
```

Dòng cuối cùng của script cần có dấu // để trình duyệt không diễn dịch dòng này d-ới dạng mã JavaScript. Các ví dụ trong ch-ơng này không chứa đặc điểm ẩn của JavaScript để mã có thể dễ hiểu hơn.

2.1.2. Sử dụng một file nguồn JavaScript

Thuộc tính *SRC* của thẻ `<SCRIPT>` cho phép bạn chỉ rõ file nguồn JavaScript được sử dụng (dùng phương pháp này hay hơn nhúng trực tiếp một đoạn lệnh JavaScript vào trang HTML).

Cú pháp:

```
<SCRIPT SRC="file_name.js">
....
</SCRIPT>
```

Thuộc tính này rất hữu dụng cho việc chia sẻ các hàm dùng chung cho nhiều trang khác nhau. Các câu lệnh JavaScript nằm trong cặp thẻ `<SCRIPT>` và `</SCRIPT>` có chứa thuộc tính *SRC* trừ khi nó có lỗi. Ví dụ bạn muốn đưa dòng lệnh sau vào giữa cặp thẻ `<SCRIPT SRC="...">` và `</SCRIPT>`:

```
document.write("Không tìm thấy file JS đưa vào!");
```

Thuộc tính *SRC* có thể được định rõ bằng địa chỉ *URL*, các liên kết hoặc các đường dẫn tuyệt đối, ví dụ:

```
<SCRIPT SRC=" http://cse.com.vn ">
```

Các file JavaScript bên ngoài không được chứa bất kỳ thẻ HTML nào. Chúng chỉ được chứa các câu lệnh JavaScript và định nghĩa hàm.

Tên file của các hàm JavaScript bên ngoài cần có đuôi *.js*, và server sẽ phải ánh xạ đuôi *.js* đó tới kiểu MIME `application/x-javascript`. Đó là những gì mà server gửi trở lại phần Header của file HTML. Để

ánh xạ đuôi này vào kiểu MIME, ta thêm dòng sau vào file `mime.types` trong đường dẫn cấu hình của server, sau đó khởi động lại server:

```
type=application/x-javascript
```

Nếu server không ánh xạ đuôi *.js* tới kiểu MIME `application/x-javascript`, Navigator sẽ tải file JavaScript được chỉ ra trong thuộc tính *SRC* về không đúng cách.

Trong ví dụ sau, hàm `bar` có chứa chuỗi "left" nằm trong một cặp dấu nháy kép:

```
function bar(widthPct){
document.write(" <HR ALIGN='LEFT' WIDTH="+widthPct+"%>")
}
```

2.3. Thẻ `<NoScript>` và `</NoScript>`

Cặp thẻ này dùng để định rõ nội dung thông báo cho người sử dụng biết trình duyệt không hỗ trợ JavaScript. Khi đó trình duyệt sẽ không hiểu thẻ `<NoScript>` và nó bị bỏ đi, còn đoạn mã nằm trong cặp thẻ này sẽ được Navigator hiển thị. Ngược lại, nếu trình duyệt có hỗ trợ JavaScript thì đoạn mã trong cặp thẻ `<NoScript>` sẽ được bỏ qua. Tuy nhiên, điều này cũng có thể xảy ra nếu người sử dụng không sử dụng JavaScript trong trình duyệt của mình bằng cách tắt nó đi trong hộp *Preferences/Advanced*.

Chú ý

Khi bạn muốn chỉ ra một chuỗi trích dẫn trong một chuỗi khác cần sử dụng dấu nháy đơn (') để phân định chuỗi đó. Điều này cho phép script nhận ra chuỗi ký tự đó.

Ví dụ:

```
<NOSCRIPT>
<B> Trang này có sử dụng JavaScript. Do đó bạn cần sử dụng trình duyệt Netscape
Navigator từ version 2.0 trở đi!
<BR>
<A HREF="http://home.netscape.com/comprd/mirror/index.html">
Hãy kích chuột vào đây để tải về phiên bản Netscape mới hơn
</A>
</BR>
Nếu bạn đã sử dụng trình duyệt Netscape từ 2.0 trở đi mà vẫn đọc đ-ợc dòng chữ này thì
hãy bật Preferences/Advanced/JavaScript lên
</NOSCRIPT>
```

Hình 2.3: Minh hoạ thẻ NOSCRIPT

2.3. Hiển thị một dòng text

Trong hầu hết các ngôn ngữ lập trình, một trong những khả năng cơ sở là hiển thị ra màn hình một dòng text. Trong JavaScript, người lập trình cũng có thể điều khiển việc xuất ra màn hình của client một dòng text tuần tự trong file HTML. JavaScript sẽ xác định điểm

mà nó sẽ xuất ra trong file HTML và dòng text kết quả sẽ đ-ợc dịch nh- các dòng HTML khác và hiển thị trên trang.

Hơn nữa, JavaScript còn cho phép người lập trình sinh ra một hộp thông báo hoặc xác nhận gồm một hoặc hai nút. Ngoài ra, dòng text và các con số còn có thể hiển thị trong tr-ờng TEXT và TEXTAREA của một form.

Trong phần này, ta sẽ học cách thức `write()` và `writeln()` của đối tượng `document`.

Đối tượng `document` trong JavaScript được thiết kế sẵn hai cách thức để xuất một dòng text ra màn hình client: `write()` và `writeln()`. Cách gọi một cách thức của một đối tượng như sau:

`object_name.property_name`

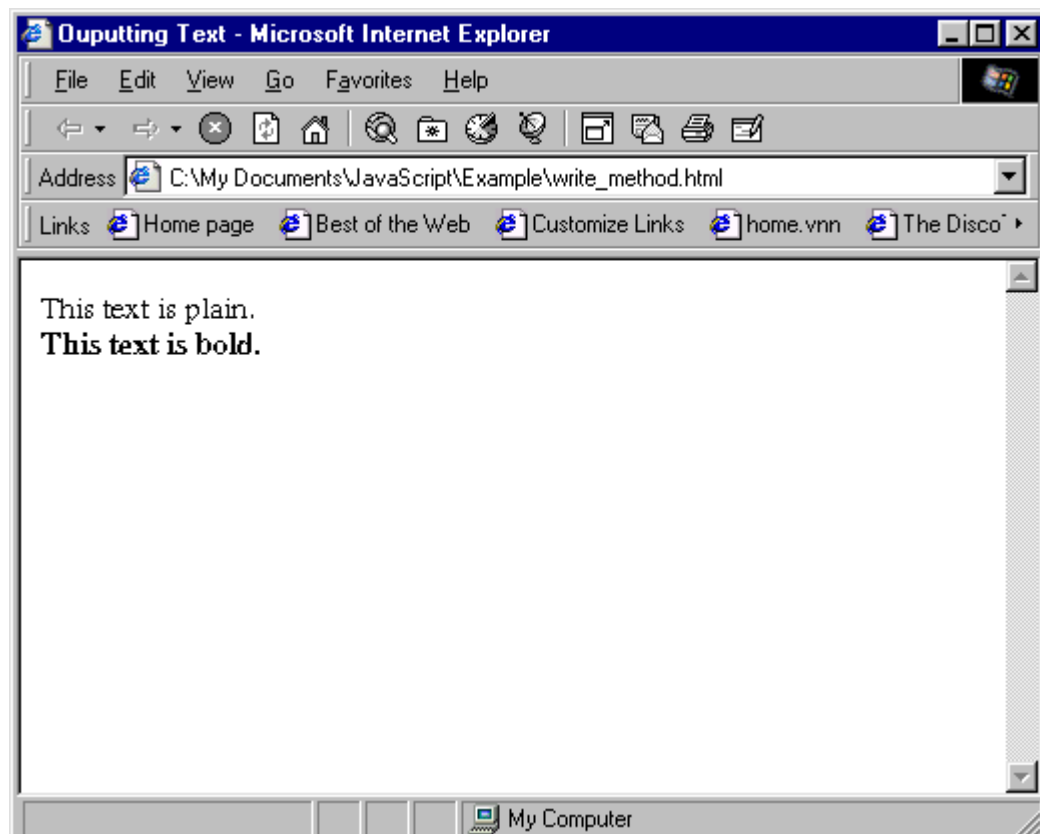
Dữ liệu mà cách thức dùng để thực hiện công việc của nó được đưa vào dòng tham số, ví dụ:

```
document.write("Test");  
document.writeln('Test');
```

Cách thức `write()` xuất ra màn hình xâu Text như không xuống dòng, còn cách thức `writeln()` sau khi viết xong dòng Text tự động xuống dòng. Hai cách thức này đều cho phép xuất ra thẻ HTML.

Ví dụ: Cách thức `write()` xuất ra thẻ HTML

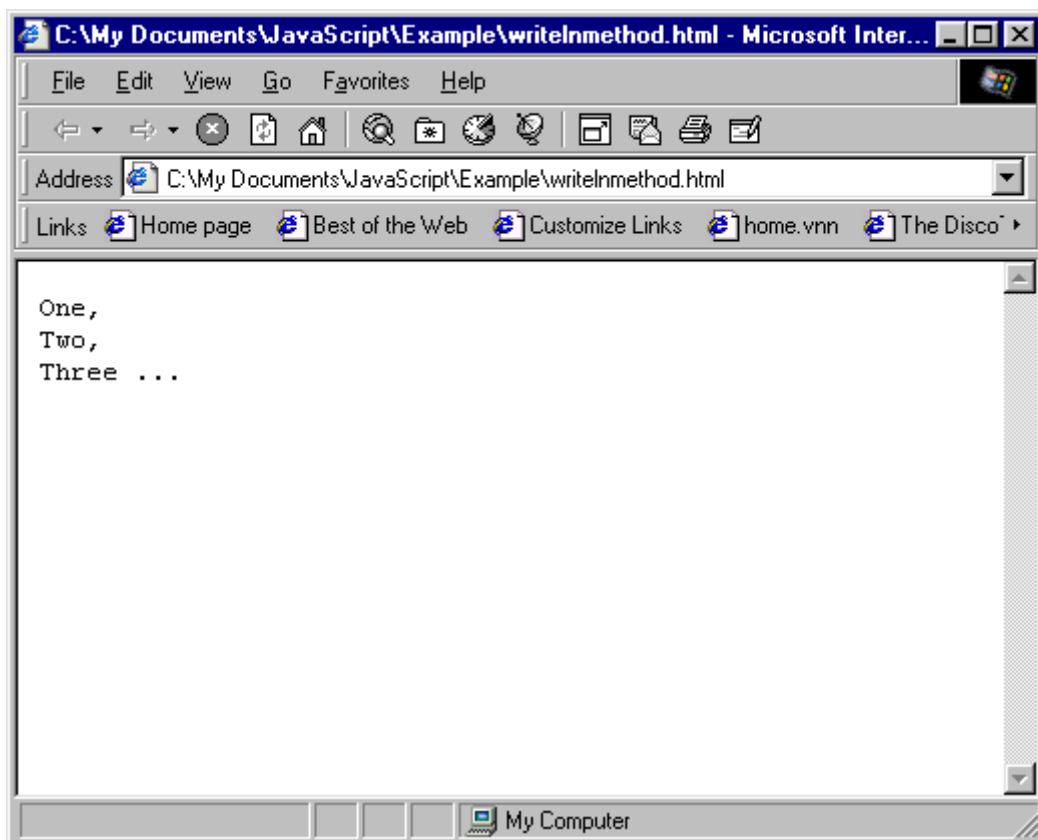
```
<HTML>  
<HEAD>  
<TITLE>Outputting Text</TITLE>  
</HEAD>  
<BODY> This text is plain.<BR> <B>  
<SCRIPT LANGUAGE="JavaScript">  
<!-- HIDE FROM OTHER BROWSERS  
document.write("This text is bold.</B>");  
// STOP HIDING FROM OTHER BROWSERS -->  
</SCRIPT>  
</BODY>  
</HTML>
```



Ví dụ: Sự khác nhau của write() và writeln():

```
<PRE>
<SCRIPT LANGUAGE="JavaScript">
<!-- HIDE FROM OTHER BROWSERS
document.writeln("One,");
document.writeln("Two,");
document.write("Three ");
document.write("...");
// STOP HIDING FROM OTHER BROWSERS -->
</SCRIPT>
</PRE>
```

Khi duyệt sẽ đ-ọc kết quả:



Hình 2.5: Sự khác nhau của write() và writeln()

2.4. Giao tiếp với ng-ời sử dụng

JavaScript hỗ trợ khả năng cho phép ng-ời lập trình tạo ra một hộp hội thoại. Nội dung của hộp hội thoại phụ thuộc vào trang HTML có chứa đoạn script mà không làm ảnh h-ởng đến việc xuất nội dung trang.

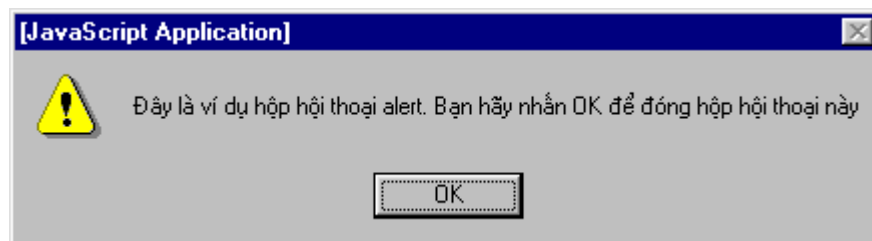
Cách đơn giản để làm việc đó là sử dụng cách thức alert(). Để sử dụng đ-ọc cách thức này, bạn phải đ-a vào một dòng text nh- khi sử dụng document.write() và document.writeln() trong phần tr-ớc. Ví dụ:

```
alert("Nhấn vào OK để tiếp tục");
```

Khi đó file sẽ chờ cho đến khi người sử dụng nhấn vào nút OK rồi mới tiếp tục thực hiện. Thông thường, cách thức `alert()` được sử dụng trong các trường hợp:

- Thông tin đưa và form không hợp lệ
- Kết quả sau khi tính toán không hợp lệ
- Khi dịch vụ chưa sẵn sàng để truy nhập dữ liệu

Tuy nhiên cách thức `alert()` mới chỉ cho phép thông báo với người sử dụng chứ chưa thực sự giao tiếp với người sử dụng. JavaScript cung cấp một cách thức khác để giao tiếp với người sử dụng là `prompt()`. Tương tự như `alert()`, `prompt()` tạo ra một hộp hội thoại với một dòng thông báo do bạn đưa vào, ngoài ra nó còn cung cấp một trường để nhập dữ liệu vào. Người sử dụng có thể nhập vào trường đó rồi kích vào OK. Khi đó, ta có thể xử lý dữ liệu do người sử dụng vừa đưa vào.

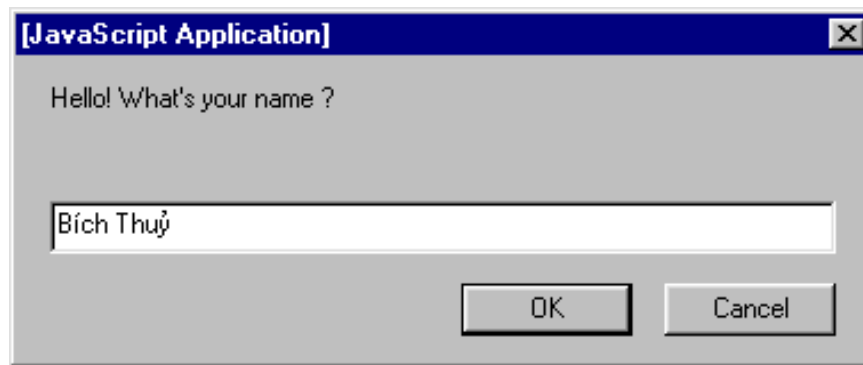


Ví dụ: Hộp hội thoại gồm một dòng thông báo, một trường nhập dữ liệu, một nút OK và một nút Cancel

Chương trình này sẽ hỏi tên người dùng và sau đó sẽ hiển thị một thông báo nhắc sử dụng tên mới đưa vào. Ví dụ được lưu vào file `Hello.html`

```
<HTML>
<HEAD>
<TITLE> JavaScript Exemple </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
var  name=window.prompt("Hello! What's your name ?", "");
document.write("Hello ' + name + ' ! I hope you like JavaScript ");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

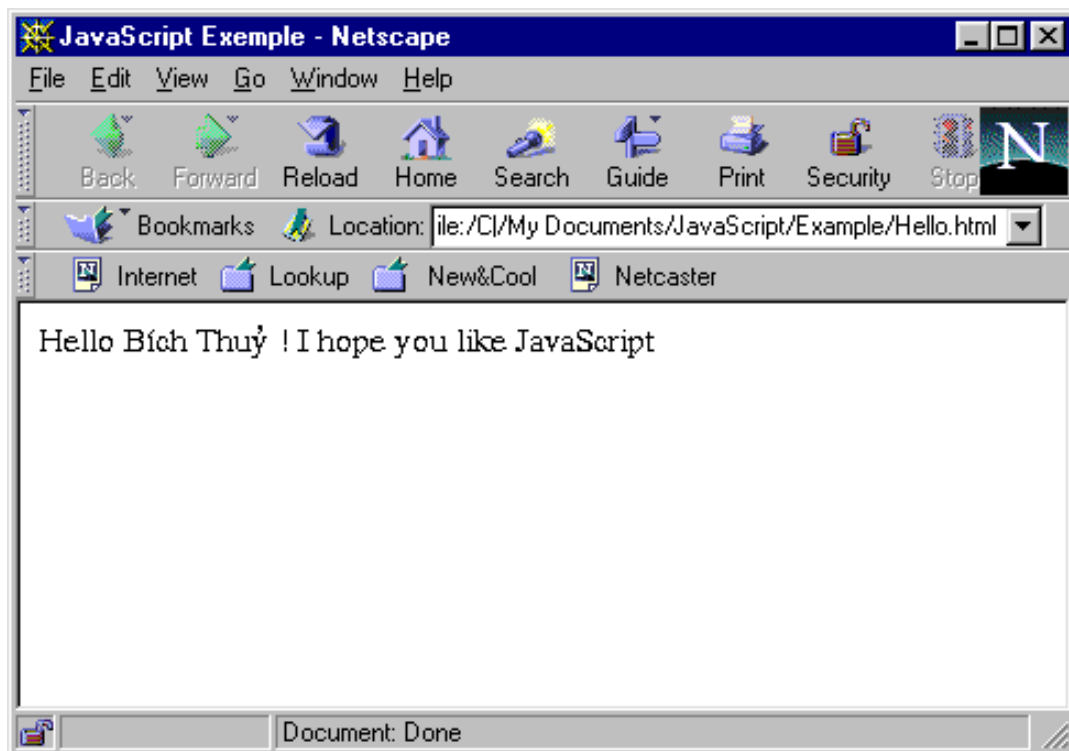
Khi duyệt có kết quả:



Hình2.1: Hiển thị cửa sổ nhập tên

Ví dụ này hiển thị dấu nhắc nhập vào tên với ph-ơng thức `window.prompt`. Giá trị đạt đ-ợc sẽ đ-ợc ghi trong biến có tên là `name`.

Biến `name` đ-ợc kết hợp với các chuỗi khác và đ-ợc hiển thị trong cửa sổ của trình duyệt nhờ ph-ơng thức `document.write`.



Hình 2.2: Hiển thị lời chào ng-ời nhập

Bây giờ bạn đã có ý t-ởng về các chức năng có thể đạt đ-ợc qua JavaScript, chúng ta hãy tiếp tục tìm hiểu thêm về chính ngôn ngữ này.

2.5. Điềm lại các lệnh vỡ mở rộng

LỆNH/MỞ RỘNG	KIỂU	MÔ TẢ
SCRIPT	thẻ HTML	Hộp chứa các lệnh JavaScript
SRC	Thuộc tính của thẻ SCRIPT	Giữ địa chỉ của file JavaScript bên ngoài. File này phải có phần đuôi .js
LANGUAGE	thuộc tính của thẻ SCRIPT	Định rõ ngôn ngữ script đ-ợc sử dụng (JavaScript hoặc VBScript)
//	Ghi chú trong JavaScript	Đánh dấu ghi chú một dòng trong đoạn script
/*...*/	Ghi chú trong JavaScript	Đánh dấu ghi chú một khối trong đoạn script
document.write()	cách thức JavaScript	Xuất ra một xâu trên cửa sổ hiện thời một cách tuần tự theo file HTML có đoạn script đó
document.writeln()	Cách thức JavaScript	T-ơng tự cách thức document.write() nh-ng viết xong tự xuống dòng.
alert()	Cách thức của JavaScript	Hiện thị một dòng thông báo trên hộp hội thoại
prompt()	Cách thức JavaScript	Hiện thị một dòng thông báo trong hộp hội thoại đồng thời cung cấp một tr-ờng nhập dữ liệu để ng-ời sử dụng nhập vào.

CH-ƠNG 3 BIẾN TRONG JAVASCRIPT

3.1. Biến vớ phân loại biến

Tên biến trong JavaScript phải bắt đầu bằng chữ hay dấu gạch d-ới. Các chữ số không đ-ọc sử dụng để mở đầu tên một biến nh-ng có thể sử dụng sau ký tự đầu tiên.

Phạm vi của biến có thể là một trong hai kiểu sau:

- ã *Biến toàn cục*: Có thể đ-ọc truy cập từ bất kỳ đâu trong ứng dụng. đ-ọc khai báo nh- sau :

x = 0;

- ã *Biến cục bộ*: Chỉ đ-ọc truy cập trong phạm vi ch-ơng trình mà nó khai báo. Biến cục bộ đ-ọc khai báo trong một hàm với từ khoá **var** nh- sau:

var x = 0;

Biến toàn cục có thể sử dụng từ khoá `var`, tuy nhiên điều này không thực sự cần thiết.

3.2. Biểu diễn từ tổ trong JavaScript

Từ tổ là các giá trị trong ch-ơng trình không thay đổi. Sau đây là các ví dụ về từ tổ:

8

"The dog ate my shoe"

true

Chú ý

Khác với C, trong JavaScript không có kiểu hằng số `CONST` để biểu diễn một giá trị không đổi nữa đây

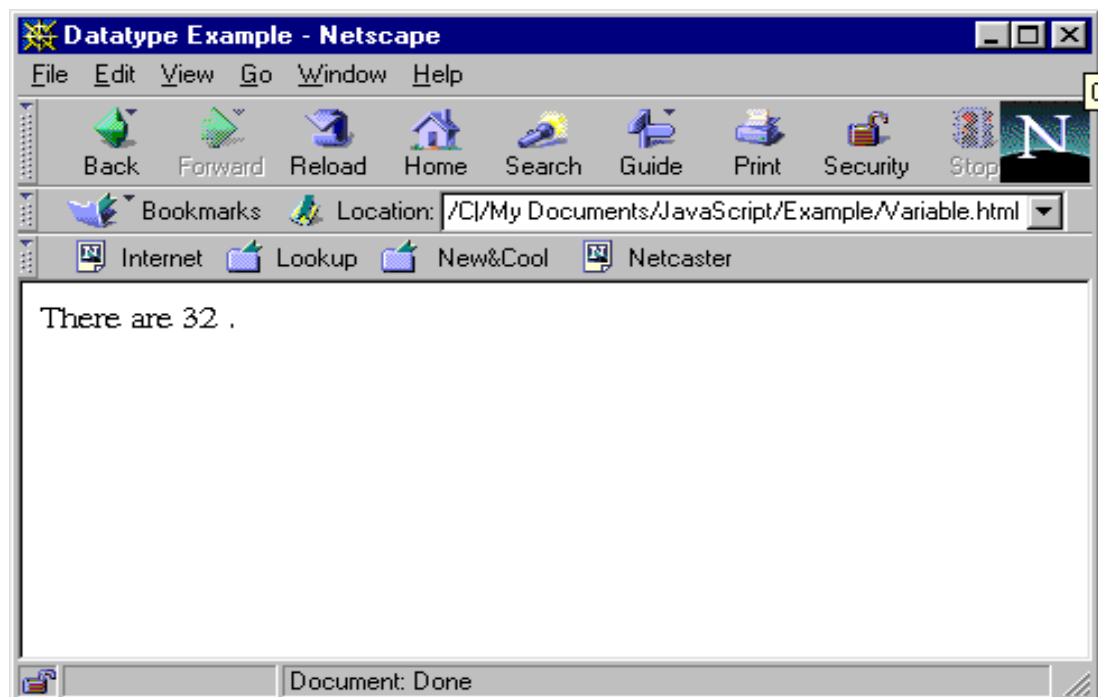
3.3. Kiểu dữ liệu

Khác với C++ hay Java, JavaScript là ngôn ngữ có tính định kiểu thấp. Điều này có nghĩa là không phải chỉ ra kiểu dữ liệu khi khai báo biến. Kiểu dữ liệu đ-ọc tự động chuyển thành kiểu phù hợp khi cần thiết.

Ví dụ file Variable.Html:

```
<HTML>
<HEAD>
<TITLE> Datatype Example </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
var fruit='apples';
var numfruit=12;
numfruit = numfruit + 20;
var temp ="There are " + numfruit + " " + ".";
document.write(temp);
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Các trình duyệt hỗ trợ JavaScript sẽ xử lý chính xác ví dụ trên và đ-a ra kết quả d-ới đây:



Trình diễn dịch JavaScript sẽ xem biến numfruit có kiểu nguyên khi cộng với 20 và có kiểu chuỗi khi kết hợp với biến temp.

Trong JavaScript, có bốn kiểu dữ liệu sau đây: *kiểu số nguyên*, *kiểu dấu phẩy động*, *kiểu logic* và *kiểu chuỗi*.

1.1.1. Kiểu nguyên (Integer)

Số nguyên có thể đ-ọc biểu diễn theo ba cách:

- ã *Hệ cơ số 10* (hệ thập phân) U có thể biểu diễn số nguyên theo cơ số 10, chú ý rằng chữ số đầu tiên phải khác 0.
- ã *Hệ cơ số 8* (hệ bát phân) U số nguyên có thể biểu diễn d-ới dạng bát phân với chữ số đầu tiên là số 0.
- ã *Hệ cơ số 16* (hệ thập lục phân) U số nguyên có thể biểu diễn d-ới dạng thập lục phân với hai chữ số đầu tiên là 0x.

1.1.2. Kiểu dấu phẩy động (Floating Point)

Một literal có kiểu dấu phẩy động có 4 thành phần sau:

Hình 3.1: Kết quả của xử lý dữ liệu

- ã Phần nguyên thập phân.
 - ã Dấu chấm thập phân (.).
 - ã Phần d-.
 - ã Phần mũ.
-

Để phân biệt kiểu dấu phẩy động với kiểu số nguyên, phải có ít nhất một chữ số theo sau dấu chấm hay **E**. Ví dụ:

9.87

U0.85E4

9.87E14

.98EU3

1.1.3. Kiểu logic (Boolean)

Kiểu logic đ-ọc sử dụng để chỉ hai điều kiện : đúng hoặc sai. Miền giá trị của kiểu này chỉ có hai giá trị

ã true.

ã false.

1.1.4. Kiểu chuỗi (String)

Một literal kiểu chuỗi đ-ọc biểu diễn bởi không hay nhiều ký tự đ-ọc đặt trong cặp dấu " ... " hay '... '. Ví dụ:

“The dog ran up the tree”

‘The dog barked’

“100”

Để biểu diễn dấu nháy kép ("), trong chuỗi sử dụng (\ "), ví dụ:

document.write(“ \”This text inside quotes.\””);

2. XÂY DỰNG CÁC BIỂU THỨC TRONG JAVASCRIPT

ĐỊNH NGHĨA VỀ PHÂN LOẠI BIỂU THỨC

Tập hợp các literal, biến và các toán tử nhằm đánh giá một giá trị nào đó được gọi là một biểu thức (expression). Về cơ bản có ba kiểu biểu thức trong JavaScript:

- ã **Số học:** Nhằm để l-ợng giá giá trị số. Ví dụ $(3+4)+(84.5/3)$ được đánh giá bằng *197.1666666667*.
- ã **Chuỗi:** Nhằm để đánh giá chuỗi. Ví dụ *"The dog barked"* + *barktone* + *"!"* là *The dog barked ferociously!*.
- ã **Logic:** Nhằm đánh giá giá trị logic. Ví dụ *temp* > 32 có thể nhận giá trị sai. JavaScript cũng hỗ trợ biểu thức điều kiện, cú pháp nh- sau:

(condition) ? valTrue : valFalse

Nếu điều kiện *condition* được đánh giá là đúng, biểu thức nhận giá trị *valTrue*, ngược lại nhận giá trị *valFalse*. Ví dụ:

```
state = (temp>32) ? "liquid" : "solid"
```

Trong ví dụ này biến *state* được gán giá trị *"liquid"* nếu giá trị của biến *temp* lớn hơn 32; trong tr-ờng hợp ngược lại nó nhận giá trị *"solid"*.

CÁC TOÁN TỬ (OPERATOR)

Toán tử được sử dụng để thực hiện một phép toán nào đó trên dữ liệu. Một toán tử có thể trả lại một giá trị kiểu số, kiểu chuỗi hay kiểu logic. Các toán tử trong JavaScript có thể được nhóm thành các loại sau đây: *gán*, *so sánh*, *số học*, *chuỗi*, *logic* và *logic bitwise*.

2.1.1. Gán

Toán tử gán là dấu bằng (=) nhằm thực hiện việc gán giá trị của toán hạng bên phải cho toán hạng bên trái. Bên cạnh đó JavaScript còn hỗ trợ một số kiểu toán tử gán rút gọn.

Kiểu gán thông th-ờng

$x = x + y$

$x = x \cup y$

$x = x * y$

$x = x / y$

$x = x \% y$

Kiểu gán rút gọn

$x += y$

$x \cup= y$

$x *= y$

$x /= y$

$x \% = y$

2.1.2. So sánh

Người ta sử dụng toán tử so sánh để so sánh hai toán hạng và trả lại giá trị đúng hay sai phụ thuộc vào kết quả so sánh. Sau đây là một số toán tử so sánh trong JavaScript:

==	Trả lại giá trị đúng nếu toán hạng bên trái bằng toán hạng bên phải
!=	Trả lại giá trị đúng nếu toán hạng bên trái khác toán hạng bên phải
>	Trả lại giá trị đúng nếu toán hạng bên trái lớn hơn toán hạng bên phải
>=	Trả lại giá trị đúng nếu toán hạng bên trái lớn hơn hoặc bằng toán hạng bên phải
<	Trả lại giá trị đúng nếu toán hạng bên trái nhỏ hơn toán hạng bên phải
<=	Trả lại giá trị đúng nếu toán hạng bên trái nhỏ hơn hoặc bằng toán hạng bên phải

2.1.3. Số học

Bên cạnh các toán tử cộng (+), trừ (-), nhân (*), chia (/) thông thường, JavaScript còn hỗ trợ các toán tử sau đây:

var1% var2	Toán tử phần dư, trả lại phần dư khi chia var1 cho var2
U'	Toán tử phủ định, có giá trị phủ định toán hạng
var++	Toán tử này tăng var lên 1 (có thể biểu diễn là ++var)
varU'	Toán tử này giảm var đi 1 (có thể biểu diễn là U'var)

Chú ý

Nếu bạn gán giá trị của toán tử ++ hay U' vào một biến, như `y = x++`, có thể có các kết quả khác nhau phụ thuộc vào vị trí xuất hiện trước hay sau của ++ hay U' với tên biến (là x trong trường hợp này). Nếu ++ đứng trước x, x sẽ được tăng hoặc giảm trước khi giá trị x được gán cho y. Nếu ++ hay U' đứng sau x, giá trị của x được gán cho y trước khi nó được tăng hay giảm.

2.1.4. Chuỗi

Khi được sử dụng với chuỗi, toán tử + được coi là kết hợp hai chuỗi, ví dụ:

```
"abc" + "xyz" được "abcxyz"
```

2.1.5. Logic

JavaScript hỗ trợ các toán tử logic sau đây:

<code>expr1 && expr2</code>	Là toán tử logic AND, trả lại giá trị đúng nếu cả expr1 và expr2 cùng đúng.
-------------------------------------	---

<code>expr1 expr2</code>	Là toán tử logic OR, trả lại giá trị đúng nếu ít nhất một trong hai <code>expr1</code> và <code>expr2</code> đúng.
<code>! expr</code>	Là toán tử logic NOT phủ định giá trị của <code>expr</code> .

2.1.6. Bitwise

Với các toán tử thao tác trên bit, đầu tiên giá trị được chuyển d-ới dạng số nguyên 32 bit, sau đó lần l-ợt thực hiện các phép toán trên từng bit.

- &** Toán tử bitwise AND, trả lại giá trị 1 nếu cả hai bit cùng là 1.
- |** Toán tử bitwise OR, trả lại giá trị 1 nếu một trong hai bit là 1.
- ^** Toán tử bitwise XOR, trả lại giá trị 1 nếu hai bit có giá trị khác nhau

Ngoài ra còn có một số toán tử dịch chuyển bitwise. Giá trị được chuyển thành số nguyên 32 bit tr-ớc khi dịch chuyển. Sau khi dịch chuyển, giá trị lại được chuyển thành kiểu của toán hạng bên trái. Sau đây là các toán tử dịch chuyển:

- <<** Toán tử dịch trái. Dịch chuyển toán hạng trái sang trái một số l-ợng bit bằng toán hạng phải. Các bit bị chuyển sang trái bị mất và 0 thay vào phía bên phải. Ví dụ: `4<<2` trở thành 16 (số nhị phân 100 trở thành số nhị phân 10000)
- >>** Toán tử dịch phải. Dịch chuyển toán hạng trái sang phải một số l-ợng bit bằng toán hạng phải. Các bit bị chuyển sang phải bị mất và dấu của toán hạng bên trái được giữ nguyên. Ví dụ: `16>>2` trở thành 4 (số nhị phân 10000 trở thành số nhị phân 100)
- >>>** Toán tử dịch phải có chèn 0. Dịch chuyển toán hạng trái sang phải một số l-ợng bit bằng toán hạng phải. Bit dấu được dịch chuyển từ trái (giống `>>`). Những bit được dịch sang phải bị xoá đi. Ví dụ: `U8>>>2` trở thành 1073741822 (bởi các bit dấu đã trở thành một phần của số). Tất nhiên với số d-ơng kết quả của toán tử `>>` và `>>>` là giống nhau.

Có một số toán tử dịch chuyển bitwise rút gọn:

<i>Kiểu bitwise thông th-ờng</i>	<i>Kiểu bitwise rút gọn</i>
<code>x = x << y</code>	<code>x <<= y</code>
<code>x = x >> y</code>	<code>x U' >> y</code>
<code>x = x >>> y</code>	<code>x >>>= y</code>
<code>x = x & y</code>	<code>x &= y</code>
<code>x = x ^ y</code>	<code>x ^= y</code>
<code>x = x y</code>	<code>x = y</code>

BỒI TẬP

2.1.7. Câu hỏi

Hãy đánh giá các biểu thức sau:

1. a. $7 + 5$
b. `"7" + "5"`
c. $7 == 7$
d. $7 >= 5$
e. $7 <= 7$
2. f. $(7 < 5) ? 7 : 5$
g. $(7 >= 5) \&\& (5 > 5)$
h. $(7 >= 5) \parallel (5 > 5)$

2.1.8. Trả lời

Các biểu thức được đánh giá như sau:

1. a. 12
b. `"75"`
c. `true`
d. `true`
e. `true`
 2. f. 5
g. `false`
h. `true`
-

3. CÁC LỆNH

Có thể chia các lệnh của JavaScript thành ba nhóm sau:

- Lệnh điều kiện.
- Lệnh lặp.
- Lệnh thao tác trên đối tượng.

CÂU LỆNH ĐIỀU KIỆN

Câu lệnh điều kiện cho phép chương trình ra quyết định và thực hiện công việc nào đấy dựa trên kết quả của quyết định. Trong JavaScript, câu lệnh điều kiện là ***if...else***

if ... else

Câu lệnh này cho phép bạn kiểm tra điều kiện và thực hiện một nhóm lệnh nào đấy dựa trên kết quả của điều kiện vừa kiểm tra. Nhóm lệnh sau **else** không bắt buộc phải có, nó cho phép chỉ ra nhóm lệnh phải thực hiện nếu điều kiện là sai.

Cú pháp

```
if ( <điều kiện> )
{
    //Các câu lệnh với điều kiện đúng
}
else
{
    //Các câu lệnh với điều kiện sai
}
```

Ví dụ:

```
if (x==10) {
    document.write("x bằng 10, đặt lại x bằng 0.");
    x = 0;
}
else
    document.write("x không bằng 10.");
```

Chú ý

Ký tự { và } đợc sử dụng để tách các khối mã.

CÂU LỆNH LẶP

Câu lệnh lặp thể hiện việc lặp đi lặp lại một đoạn mã cho đến khi biểu thức điều kiện đ-ợc đánh giá là đúng. JavaScript cung cấp hai kiểu câu lệnh lặp:

ã for loop

ã while loop

3.1.1. Vòng lặp for

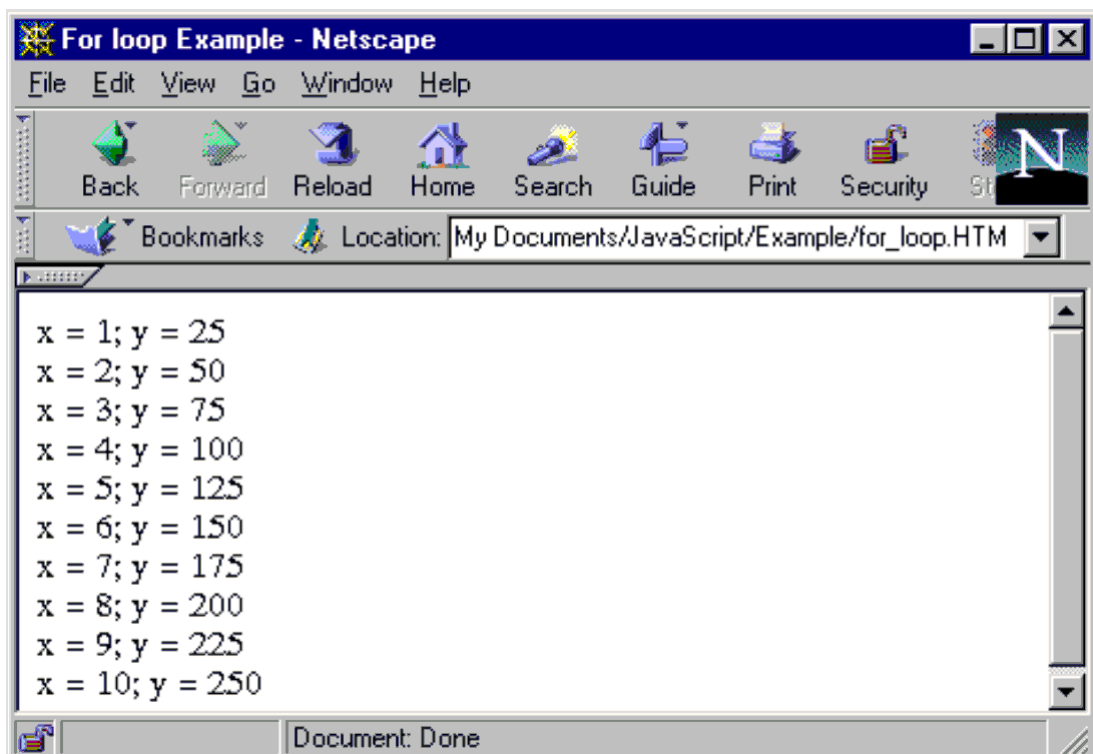
Vòng lặp for thiết lập một biểu thức khởi đầu *U* *initExpr*, sau đó lặp một đoạn mã cho đến khi biểu thức <điều kiện> đ-ợc đánh giá là đúng. Sau khi kết thúc mỗi vòng lặp, biểu thức *incrExpr* đ-ợc đánh giá lại.

Cú pháp:

```
for (initExpr; <điều kiện> ; incrExpr){  
    //Các lệnh đ-ợc thực hiện trong khi lặp  
}
```

Ví dụ:

```
<HTML> <HEAD>  
<TITLE>For loop Example </TITLE>  
<SCRIPT LANGUAGE= "JavaScript">  
for (x=1; x<=10 ; x++) {  
    y=x*25;  
    document.write("x =" + x + ";y= " + y + "<BR>");  
}  
</SCRIPT>  
</HEAD>  
<BODY></BODY>  
</HTML>
```



2

Hình 5.1: Kết quả của lệnh for...loop

Ví dụ này l-u vào file *for_loop.Html*.

Vòng lặp này sẽ thực hiện khối mã lệnh cho đến khi $x > 10$.

3.1.2. while

Vòng lặp while lặp khối lệnh chừng nào <điều kiện> còn đ-ợc đánh giá là đúng

Cú pháp:

```
while (<điều kiện>)  
{  
    //Các câu lệnh thực hiện trong khi lặp  
}
```

Ví dụ:

```
x=1;  
while (x<=10){  
    y=x*25;  
    document.write("x="+x +"; y = " + y + "<BR>");  
    x++;  
}
```

Kết quả của ví dụ này giống nh- ví dụ tr-ớc.

3.1.3. Break

Câu lệnh **break** dùng để kết thúc việc thực hiện của vòng lặp **for** hay **while**. Ch-ơng trình đ-ợc tiếp tục thực hiện tại câu lệnh ngay sau chỗ kết thúc của vòng lặp.

Cú pháp

```
break;
```

Đoạn mã sau lặp cho đến khi x lớn hơn hoặc bằng 100. Tuy nhiên nếu giá trị x đ-a vào vòng lặp nhỏ hơn 50, vòng lặp sẽ kết thúc

Ví dụ:

```
while (x<100)
{
    if (x<50) break;
    x++;
}
```

3.1.4. continue

Lệnh **continue** giống lệnh **break** nh-ng khác ở chỗ việc lặp đ-ợc kết thúc và bắt đầu từ đầu vòng lặp. Đối với vòng lặp **while**, lệnh **continue** điều khiển quay lại <điều kiện>; với **for**, lệnh **continue** điều khiển quay lại incrExpr.

Cú pháp

```
continue;
```

Ví dụ:

Đoạn mã sau tăng x từ 0 lên 5, nhảy lên 8 và tiếp tục tăng lên 10

```
x=0;
while (x<=10)
{
    document.write("Giá trị của x lờ:" + x + "<BR>");
    if (x=5)
    {
        x=8;
        continue;
    }
    x++;
}
```

CÁC CÂU LỆNH THAO TÁC TRÊN ĐỐI T-ỢNG

JavaScript là một ngôn ngữ dựa trên đối t-ợng, do đó nó có một số câu lệnh làm việc với các đối t-ợng.

3.1.5. for...in

Câu lệnh này đ-ợc sử dụng để lặp tất cả các thuộc tính (properties) của một đối t-ợng. Tên biến có thể là một giá trị bất kỳ, chỉ cần thiết khi bạn sử dụng các thuộc tính trong vòng lặp. Ví dụ sau sẽ minh họa điều này

Cú pháp

```
for (<variable> in <object>)
{
```

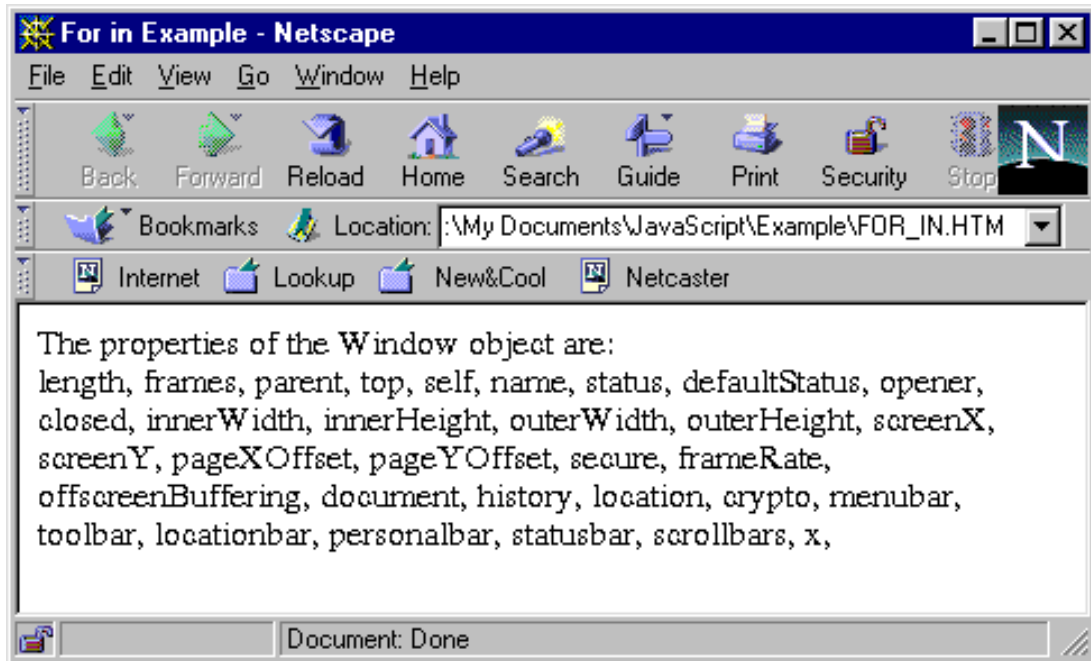


```
        //Các câu lệnh  
    }
```

Ví dụ

Ví dụ sau sẽ lấy ra tất cả các thuộc tính của đối tượng Window và in ra tên của mỗi thuộc tính. Kết quả được minh họa trên hình 5.2.

```
<HTML>  
<HEAD>  
<TITLE>For in Example </TITLE>  
<SCRIPT LANGUAGE= "JavaScript">  
    document.write("The properties of the Window    object are:  
                    <BR>");  
    for (var x in window)  
        document.write("        "+ x + ", ");  
</SCRIPT>  
</HEAD>  
<BODY>  
</BODY>  
</HTML>
```



Hình 5.2: Kết quả của lệnh for...in

3.1.6. new

Biến **new** được thực hiện để tạo ra một thể hiện mới của một đối tượng

Cú pháp

```
objectvar = new object_type ( param1 [,param2]...
                             [,paramN])
```

Ví dụ sau tạo đối tượng **person** có các thuộc tính *firstname*, *lastname*, *age*, *sex*. Chú ý rằng từ khoá **this** được sử dụng để chỉ đối tượng trong hàm **person**. Sau đó ba thể hiện của đối tượng **person** được tạo ra bằng lệnh **new**

```
<HTML>
<HEAD>
<TITLE>New Example </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
function person(first_name, last_name, age, sex){
    this.first_name=first_name;
    this.last_name=last_name;
    this.age=age;
    this.sex=sex;
}

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("Hoàn", "Đỗ Văn", "24", "Male");
    document.write ("1. "+person1.last_name+" " +
                    person1.first_name + "<BR>" );

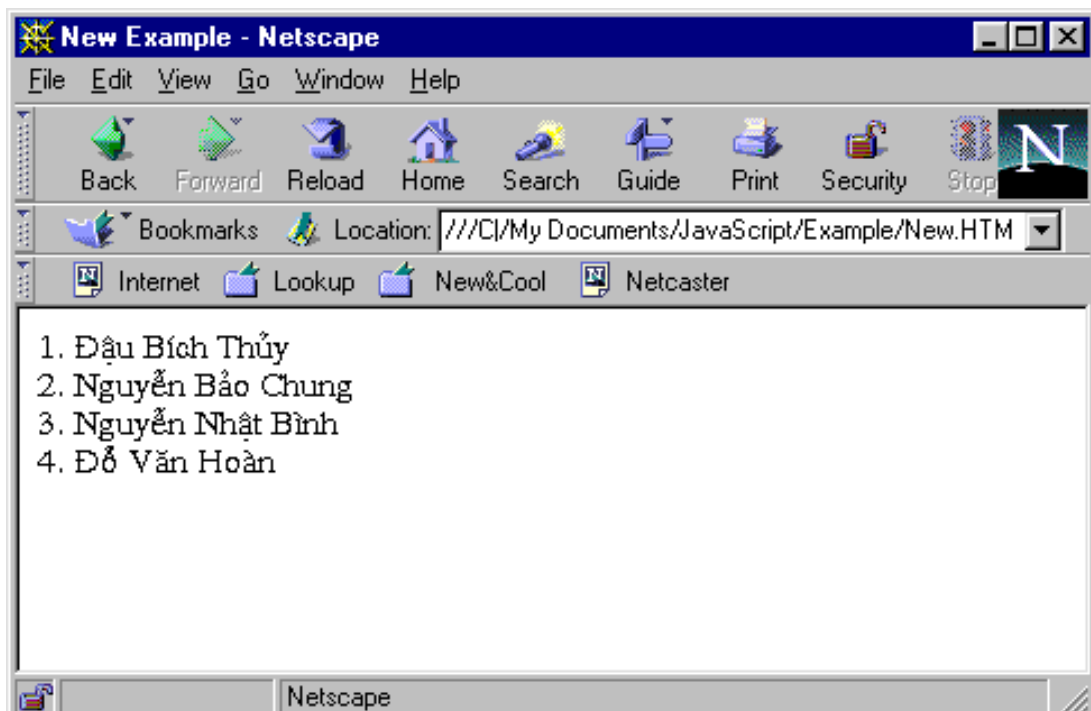
    document.write("2. "+person2.last_name + " " +
                    person2.first_name + "<BR>");

    document.write("3. "+ person3.last_name + " " +
                    person3.first_name + "<BR>");

    document.write("4. "+ person4.last_name + " " +
                    person4.first_name+"<BR>");

</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>

```



Hình 5.3: Kết quả của ví dụ lệnh New

3.1.7. **this**

Từ khoá **this** đ-ợc sử dụng để chỉ đối t-ợng hiện thời. Đối t-ợng đ-ợc gọi th-ờng là đối t-ợng hiện thời trong ph-ơng thức hoặc trong hàm.

Cú pháp

```
this [.property]
```

Có thể xem ví dụ của lệnh new.

3.1.8. with

Lệnh này đợc sử dụng để thiết lập đối t-ợng ngầm định cho một nhóm các lệnh, bạn có thể sử dụng các thuộc tính mà không đề cập đến đối t-ợng.

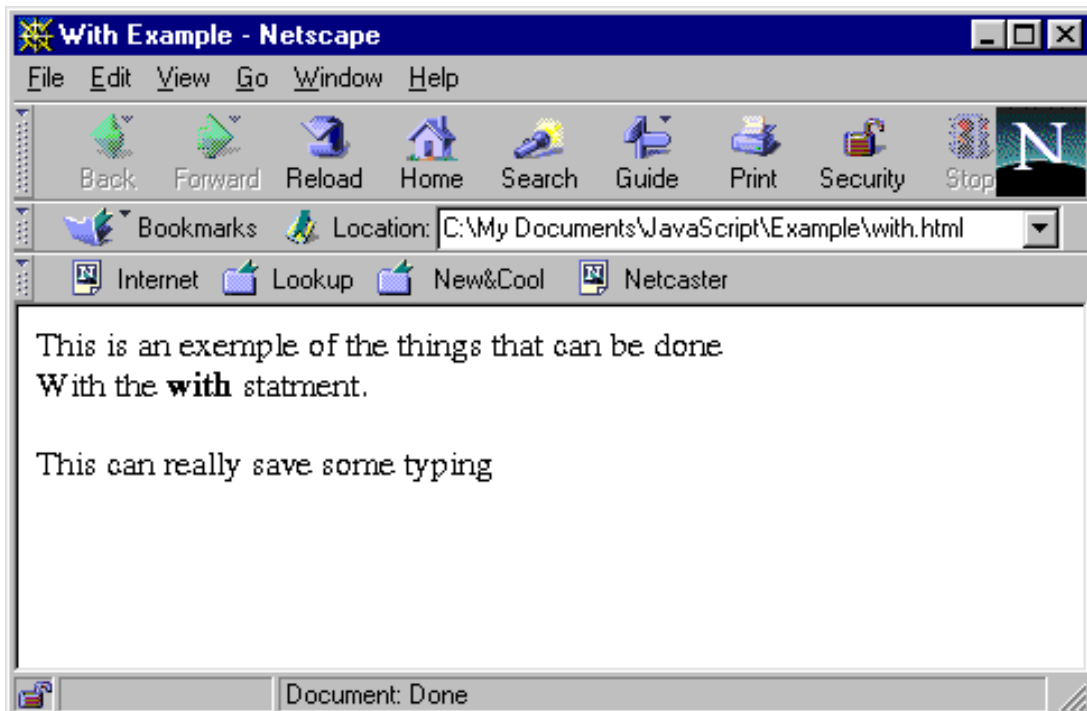
Cú pháp

```
with (object)
{
    // statement
}
```

Ví dụ:

Ví dụ sau chỉ ra cách sử dụng lệnh with để thiết lập đối t-ợng ngầm định là *document* và có thể sử dụng ph-ơng thức *write* mà không cần đề cập đến đối t-ợng document

```
<HTML>
<HEAD>
<TITLE>With Example </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
with (document){
    write("This is an exemple of the things that can be
    done <BR>");
    write("With the <B>with<B> statment. <P>");
    write("This can really save some typing");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```



Hình 5.4: Kết quả của ví dụ lệnh with

CÁC HỒM (FUNCTIONS)

JavaScript cũng cho phép sử dụng các hàm. Mặc dù không nhất thiết phải có, song các hàm có thể có một hay nhiều tham số truyền vào và một giá trị trả về. Bởi vì JavaScript là ngôn ngữ có tính định kiểu thấp nên không cần định nghĩa kiểu tham số và giá trị trả về của hàm. Hàm có thể là thuộc tính của một đối tượng, trong trường hợp này nó được xem như là phương thức của đối tượng đó.

Lệnh **function** được sử dụng để tạo ra hàm trong JavaScript.

Cú pháp

```
function fnName([param1], [param2], ..., [paramN])
{
    //function statement
}
```

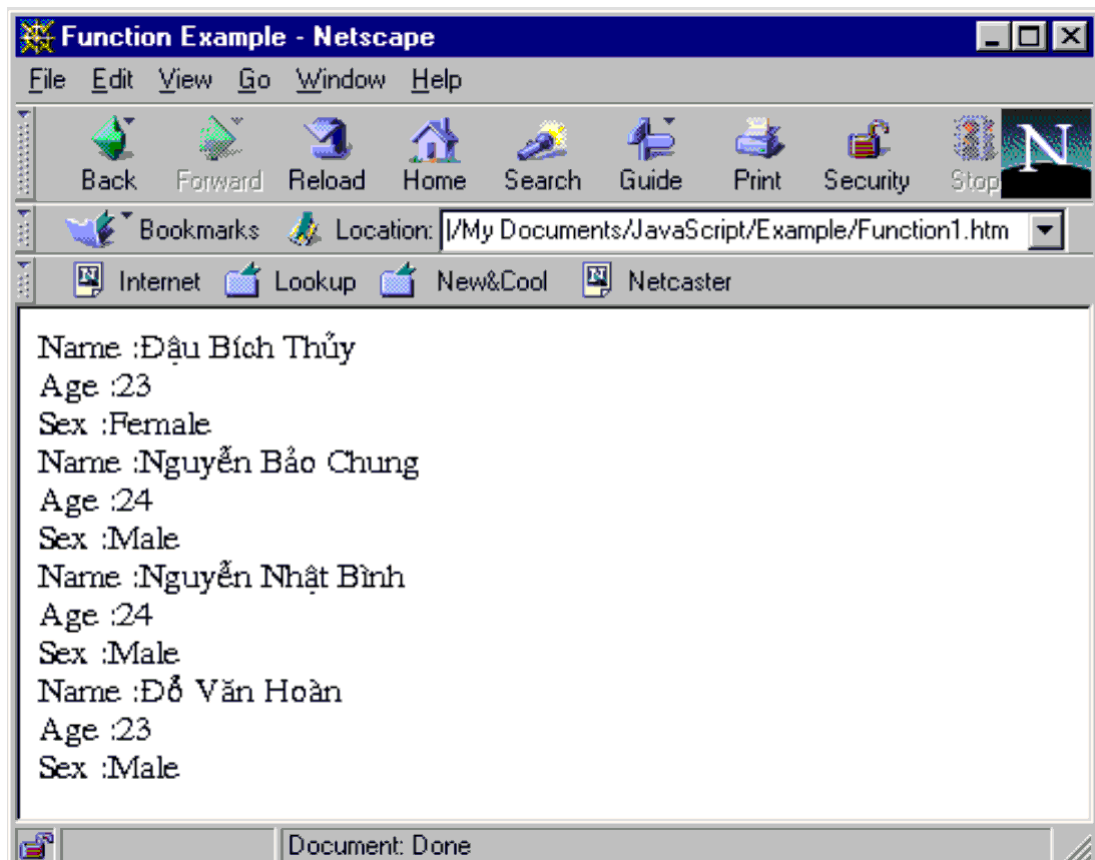
Ví dụ:

Ví dụ sau minh họa cách thức tạo ra và sử dụng hàm như là thành viên của một đối tượng. Hàm *printStats* được tạo ra là phương thức của đối tượng *person*

```
<HTML> <HEAD>
<TITLE>Function Example </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
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() {
        with (document) {
            write (" Name :" + this.last_name + " " +
                this.first_name + "<BR>" );
            write("Age :"+this.age+"<BR>");
            write("Sex :"+this.sex+"<BR>");
        }
    }

    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>
</HEAD>
<BODY> </BODY>
</HTML>
```



Hình 5.5: Kết quả việc sử dụng hàm

CÁC HẸM CỐ SẴN

JavaScript có một số hàm có sẵn, gắn trực tiếp vào chính ngôn ngữ và không nằm trong một đối tượng nào:

- eval
- parseInt
- parseFloat

3.1.9. eval

Hàm này được sử dụng để đánh giá các biểu thức hay lệnh. Biểu thức, lệnh hay các đối tượng của thuộc tính đều có thể được đánh giá. Đặc biệt hết sức hữu ích khi đánh giá các biểu thức do người dùng đưa vào (ngọc lại có thể đánh giá trực tiếp).

Cú pháp:

`returnval=eval (bất kỳ biểu thức hay lệnh hợp lệ trong Java)`

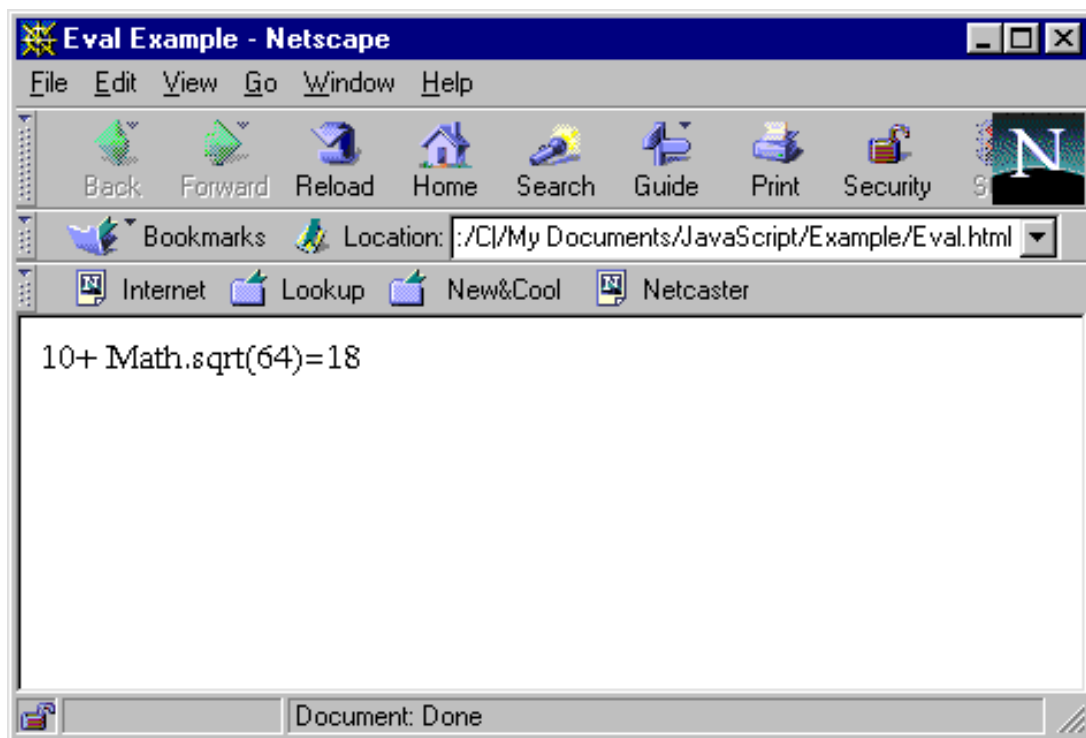
Ví dụ:

```
<HTML>
<HEAD>
```

```

<TITLE>Eval Example </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
    var string="10+ Math.sqrt(64)";
    document.write(string+ "="+ eval(string));
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>

```



Hình 5.6 Ví dụ hàm Eval

3.1.10. parseInt

Hàm này chuyển một chuỗi số thành số nguyên với cơ số là tham số thứ hai (tham số này không bắt buộc). Hàm này thường được sử dụng để chuyển các số nguyên sang cơ số 10 và đảm bảo rằng các dữ liệu được nhập dưới dạng ký tự được chuyển thành số trước khi tính toán. Trong trường hợp dữ liệu vào không hợp lệ, hàm parseInt sẽ đọc và chuyển dạng chuỗi đến vị trí nó tìm thấy ký tự không phải là số. Ngoài ra hàm này còn cắt dấu phẩy động.

Cú pháp

```
parseInt (string, [, radix])
```

Ví dụ:

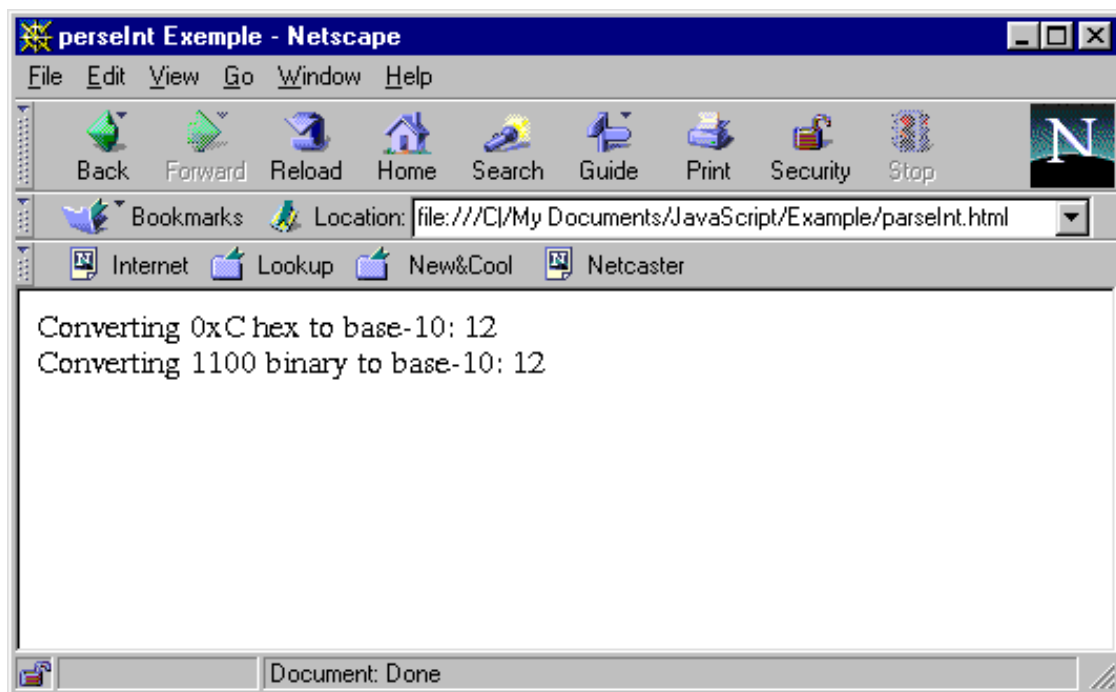
```
<HTML>
```



```

<HEAD>
<TITLE> parseInt Exemple </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
    document.write("Converting 0xC hex to base10: " +
        parseInt(0xC,10) + "<BR>");
    document.write("Converting 1100 binary to base10: " +
        parseInt(1100,2) + "<BR>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>

```



Hình 5.7: Ví dụ parInt

3.1.11. parseFloat

Hàm này giống hàm parseInt nh-ng nó chuyển chuỗi thành số biểu diễn d-ới dạng dấu phẩy động.

Cú pháp

```
parseFloat (string)
```

Ví dụ:

Ví dụ sau minh hoạ cách thức xử lý của parseFloat với các kiểu chuỗi khác nhau. Hình 5.8 minh hoạ kết quả

```
<HTML> <HEAD>
```

```

<TITLE> parseFloat Example </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
    document.write("This script will show how different strings
        are ");
document.write("Converted using parseFloat<BR>");
document.write("137= " + parseFloat("137") + "<BR>");
    document.write("137abc= " + parseFloat("137abc") + "<BR>");
    document.write("abc137= " + parseFloat("abc137") + "<BR>");
    document.write("1abc37= " + parseFloat("1abc37") + "<BR>");
</SCRIPT>
</HEAD>
<BODY> </BODY>
</HTML>

```

Hình 5.8 : Kết quả của ví dụ parseFloat

MẢNG (ARRAY)

Mặc dù JavaScript không hỗ trợ cấu trúc dữ liệu mảng nh-ng Netscape tạo ra ph-ong thức cho phép bạn tự tạo ra các hàm khởi tạo mảng nh- sau:

```

function InitArray(NumElements) {
    this.length = numElements;
    for (var x=1; x<=numElements; x++) {
        this[x]=0
    }
    return this;
}

```

Nó tạo ra một mảng với kích thước xác định trước và điền các giá trị 0. Chú ý rằng thành phần đầu tiên trong mảng là độ dài mảng và không được sử dụng.

Để tạo ra một mảng, khai báo như sau:

```
myArray = new InitArray (10)
```

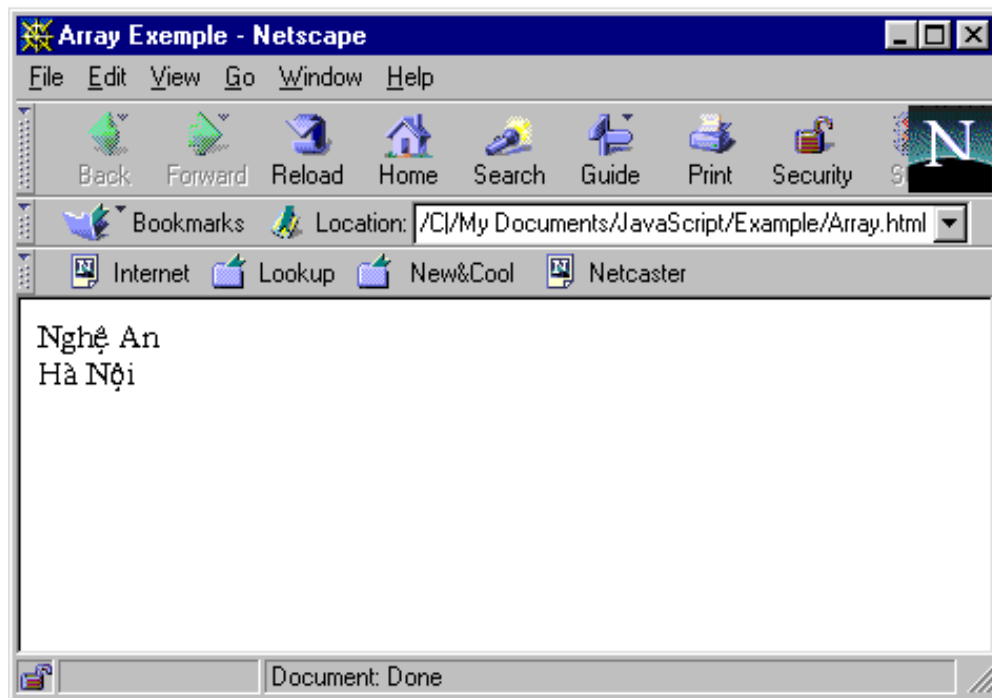
Nó tạo ra các thành phần từ myArray[1] đến myArray[10] với giá trị là 0. Giá trị các thành phần trong mảng có thể được thay đổi như sau:

```
myArray[1] = "Nghệ An"
```

```
myArray[2] = "Lỗ"
```

Sau đây là ví dụ đầy đủ:

```
<HTML> <HEAD>
<TITLE> Array Example </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
function InitArray(numElements) {
    this.length = numElements;
    for (var x=1; x<=numElements; x++){
        this[x]=0
    }
    return this;
}
myArray = new InitArray(10);
myArray[1] = "Nghệ An";
myArray[2] = "Hồ Nội";
document.write(myArray[1] + "<BR>");
document.write(myArray[2] + "<BR>");
</SCRIPT>
</HEAD>
<BODY> </BODY>
</HTML>
```



Hình 5.9: Ví dụ mảng

SỰ KIẾN

JavaScript là ngôn ngữ định hướng sự kiện, nghĩa là sẽ phản ứng trước các sự kiện xác định trước như: kích chuột hay tải một văn bản. Một sự kiện có thể gây ra việc thực hiện một đoạn mã lệnh (gọi là các chương trình xử lý sự kiện) giúp cho chương trình có thể phản ứng một cách thích hợp.

Chương trình xử lý sự kiện (Event handler): Một đoạn mã hay một hàm được thực hiện để phản ứng trước một sự kiện gọi là chương trình xử lý sự kiện. Chương trình xử lý sự kiện được xác định là một thuộc tính của một thẻ HTML:

```
<tagName eventHandler = "JavaScript Code or Function">
```

Ví dụ sau gọi hàm **CheckAge()** mỗi khi giá trị của trường văn bản thay đổi:

```
<INPUT TYPE=TEXT NAME="AGE" onChange="CheckAge()">
```

Đoạn mã của chương trình xử lý sự kiện không là một hàm; nó là các lệnh của JavaScript cách nhau bằng dấu chấm phẩy. Tuy nhiên cho mục đích viết thành các module nên viết dưới dạng các hàm.

Một số chương trình xử lý sự kiện trong JavaScript:

onBlur	Xảy ra khi input focus bị xóa từ thành phần form
onClick	Xảy ra khi người dùng kích vào các thành phần hay liên kết của form.

onChange	Xảy ra khi giá trị của thành phần đ-ợc chọn thay đổi
onFocus	Xảy ra khi thành phần của form đ-ợc focus(làm nổi lên).
onLoad	Xảy ra trang Web đ-ợc tải.
onMouseOver	Xảy ra khi di chuyển chuột qua kết nối hay anchor.
onSelect	Xảy ra khi ng-ời sử dụng lựa chọn một tr-ờng nhập dữ liệu trên form.
onSubmit	Xảy ra khi ng-ời dùng đ-a ra một form.
onUnLoad	Xảy ra khi ng-ời dùng đóng một trang

Sau đây là bảng các ch-ơng trình xử lý sự kiện có sẵn của một số đối t-ợng. Các đối t-ợng này sẽ đ-ợc trình bày kỹ hơn trong phần sau.

Đối t-ợng	Ch-ơng trình xử lý sự kiện có sẵn
Selection list	onBlur, onChange, onFocus
Text	onBlur, onChange, onFocus, onSelect
Textarea	onBlur, onChange, onFocus, onSelect
Button	onClick
Checkbox	onClick
Radio button	onClick
Hypertext link	onClick, onMouseOver, onMouseOut
Clickable Imagemap area	onMouseOver, onMouseOut
Reset button	onClick
Submit button	onClick
Document	onLoad, onUnload, onError
Window	onLoad, onUnload, onBlur, onFocus
Framesets	onBlur, onFocus
Form	onSubmit, onReset
Image	onLoad, onError, onAbort

Ví dụ sau là một đoạn mã script đơn giản của ch-ơng trình xử lý sự kiện thẩm định giá trị đ-a vào trong tr-ờng text. Tuổi của ng-ời sử dụng đ-ợc nhập vào trong tr-ờng này và ch-ơng trình xử lý sự kiện sẽ thẩm định tính hợp lệ của dữ liệu đ-a vào. Nếu không hợp lệ sẽ xuất hiện một thông báo yêu cầu nhập lại. Ch-ơng trình xử lý sự kiện đ-ợc gọi mỗi khi tr-ờng AGE bị thay đổi và focus chuyển sang tr-ờng khác. Hình 5.10 minh họa kết quả của ví dụ này

<HTML>

<HEAD>

```

<TITLE> An Event Handler Exemple </TITLE>

<SCRIPT LANGUAGE= "JavaScript">

function CheckAge(form) {

if ( (form.AGE.value<0)|| (form.AGE.value>120) )

    {

        alert("Tuổi nhập vào không hợp lệ! Mời bạn nhập
        lại");

        form.AGE.value=0;

    }

}

</SCRIPT>

</HEAD>

<BODY>

<FORM NAME="PHIEU_DIEU_TRA">

Nhập vào tên của bạn : <BR>
Tên <INPUT TYPE=TEXT NAME="TEN" MAXLENGTH=10 SIZE=10><BR>
Đệm <INPUT TYPE=TEXT NAME="DEM" MAXLENGTH=15 SIZE=10><BR>
Họ <INPUT TYPE=TEXT NAME="HO" MAXLENGTH=10 SIZE=10><BR>
    Age <INPUT TYPE=TEXT NAME="AGE" MAXLENGTH=3 SIZE=2
        onChange="CheckAge (PHIEU_DIEU_TRA) "><BR>

<P>

Bạn thích mùa nào nhất : <BR>
Mùa xuân<INPUT TYPE=RADIO NAME="MUA" VALUE="Mua xuan">
Mùa hạ<INPUT TYPE=RADIO NAME="MUA" VALUE="Mua ha">
Mùa thu<INPUT TYPE=RADIO NAME="MUA" VALUE="Mua thu">
Mùa đông<INPUT TYPE=RADIO NAME="MUA" VALUE="Mua dong">

<P>

Hãy chọn những hoạt động ngoài trời mà bạn yêu thích : <BR>
Đi bộ<INPUT TYPE=CHECKBOX NAME="HOAT_DONG" VALUE="Di bo">

    Tr-ợt tuyết<INPUT TYPE=CHECKBOX NAME="HOAT_DONG"
        VALUE="Truot tuyet">

```

```

    Thẻ thao d-ới n-ớc<INPUT TYPE=CHECKBOX NAME="HOAT_DONG"
        VALUE="Duoi nuoc">

    Đạp xe<INPUT TYPE=CHECKBOX NAME="HOAT_DONG" VALUE="Đạp xe">

<P>

<INPUT TYPE=SUBMIT>

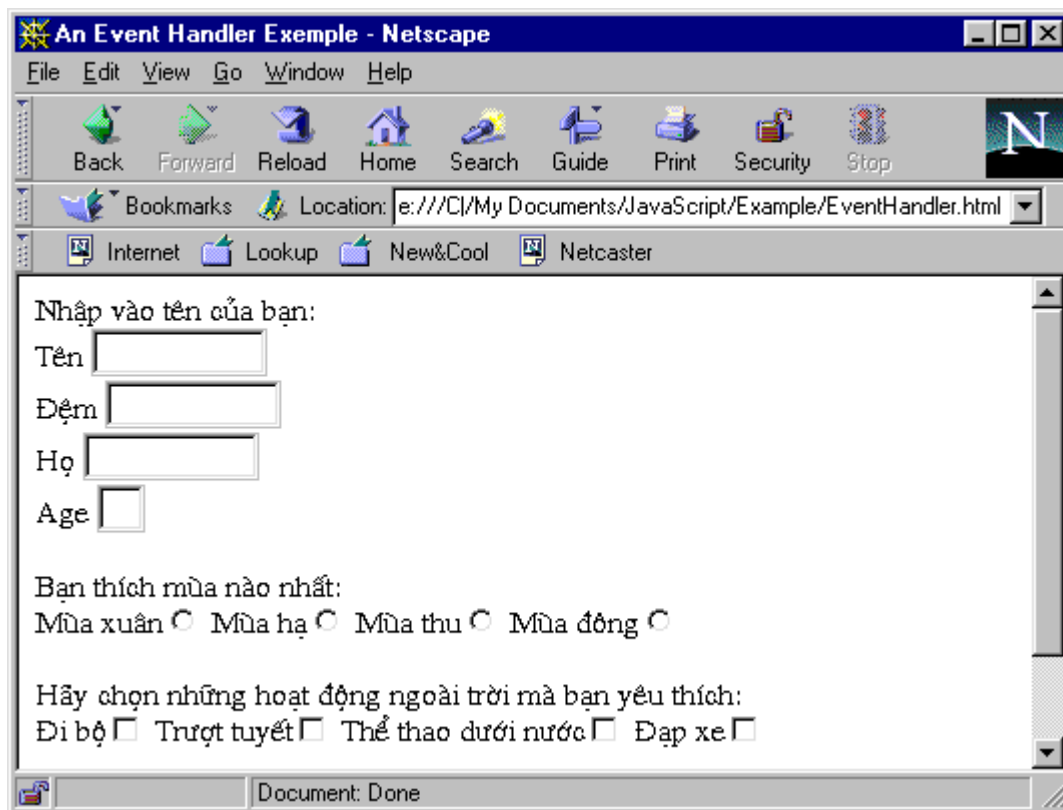
<INPUT TYPE=RESET>

</FORM>

</BODY>

</HTML>

```



Hình 5.10: Minh họa cho ví dụ Event Handler

BỒI TẬP

3.1.12. Câu hỏi

1. Viết một đoạn lệnh JavaScript sử dụng cách thức confirm() và câu lệnh if...then để thực hiện:

Hỏi ng-ời sử dụng có muốn nhận đ-ọc một lời chào không

Nếu có thì hiện ảnh wellcome.jpg và một lời chào.

Nếu không thì viết ra một lời thông báo

2. Viết một đoạn lệnh JavaScript để thực hiện:

- ã Hỏi ng-ời sử dụng: "10+10 bằng bao nhiêu?"
- ã Nếu đúng thì hỏi tiếp: Có muốn trả lời câu thứ hai không?"
- ã Nếu muốn thì hỏi: "10*10 bằng bao nhiêu?"
- ã Đánh giá kết quả bằng biểu thức logic sau đó viết ra màn hình:
Đúng: "CORRECT"; Sai: "INCORRECT"

Gợi ý: Sử dụng biến toàn cục l-u kết quả đúng để so sánh với kết quả đ-a vào.

3. Câu lệnh nào trong các câu sau đây sử dụng sai thẻ sự kiện

- a. <BODY onClick="doSomething();">
- b. <INPUT TYPE=text onFocus="doSomething();">
- c. <INPUT TYPE=textarea onLoad="doSomething();">
- d. <BODY onUnload="doSomething();">
- e. <FORM onLoad="doSomething();">
- f. <FORM onSubmit="doSomething();">

4. Điều gì xảy ra khi thực hiện script sau:

```
<HTML>
  <HEAD>
    <TITLE>Exercise 5.4</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
      <!-- HIDE FROM OTHER BROWSERS
      var name = "";
      function welcome() {
        name = prompt("Welcome to my page! What's Your
                        Name?", "name");
      }
      function farewell() {
        alert("Goodbye " + name + ". Thanks for visiting my
              page.");
      }
      // STOP HIDING FROM OTHER BROWSERS -->
    </SCRIPT>
  </HEAD>
  <BODY onLoad="welcome();" onUnload="farewell();">
    This is my page!
  </BODY>
</HTML>
```

5. Sử dụng vòng lặp while để mô phỏng các vòng lặp for sau:

a.


```
for (j = 4; j > 0; j UU) {
    document.writeln(j + "<BR>");
}
```

b.

```
for (k = 1; k <= 99; k = k*2) {
    k = k/1.5;
}
```

c.

```
for (num = 0; num <= 10; num ++) {
    if (num == 8)
        break;
}
```

3.1.13. Trả lời

1. Sử dụng cách thức confirm() và cấu trúc if...then:

```
<HTML>
<HEAD>
<TITLE>Execise 5.1</TITLE>
<HEAD>
<BODY>
<P>
<SCRIPT LANGUAGE="JavaScript">
    var conf=confirm("Click OK to see a wellcome
message!")
    if (conf){
        document.write("<IMG SRC='wellcome.jpg'>");
        document.write("<BR>Wellcome you come to CSE's
class");
    }
    else
        document.write("What a pity! You have just click
Cancel button");
</SCRIPT>
</P>
</BODY>
</HTML>
```

2. Thực hiện hỏi ng-ời sử dụng:

```
<HTML>
<HEAD>
<TITLE>Exercise 3.3</TITLE>
<SCRIPT LANGUAGE="JavaScript">
```

```

<!UU HIDE FROM OTHER BROWSERS
// DEFINE VARIABLES FOR REST OF SCRIPT
var question="What is 10+10?";
var answer=20;
var correct='CORRECT';
var incorrect='INCORRECT';
// ASK THE QUESTION
var response = prompt(question,"0");
// chECK THE ANSWER THE FIRST TIME
if (response != answer) {
// THE ANSWER WAS WRONG: OFFER A SECOND chAncE
    if (confirm("Wrong! Press OK for a second chance."))
        response = prompt(question,"0");
    } else {
        // THE ANSWER WAS RIGHT: OFFER A SECOND QUESTION
        if (confirm("Correct! Press OK for a second
question.")) {
            question = "What is 10*10?";
            answer = 100;
            response = prompt (question,"0");
        }
    }
// chECK THE ANSWER
var output = (response == answer) ? correct :
incorrect;
// STOP HIDING FROM OTHER BROWSERS UU>
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!UU HIDE FROM OTHER BROWSERS
// OUTPUT RESULT
document.write(output);
// STOP HIDING FROM OTHER BROWSERS UU>
</SCRIPT>
</BODY>
</HTML>

```

3. Các câu sai: a, c, e. Các câu đúng: b, d, f

4. Khi ch-ơng trình đ-ọc chạy (load), hàm wellcome sẽ thực hiện hỏi tên ng-ời sử dụng, l-u tên đó vào biến toàn cục name. Khi ng-ời sử dụng sang một địa chỉ URL khác, hàm farewell() sẽ thực hiện gửi một lời cảm ơn tới ng-ời sử dụng.

5. Sử dụng vòng lặp while nh- sau:

a.

```
j = 5;
while (UUj > 0) {
    document.writeln(j + "<BR>");
}
```

b.

```
k = 1;
while (k <= 99) {
    k = k * 2 / 1.5;
}
```

c.

```
num = 0;
while (num <= 10) {
    if (num++ == 8)
        break;
}
```

4. CÁC ĐỐI TƯỢNG TRONG JAVASCRIPT

Như đã nói JavaScript là ngôn ngữ lập trình *dựa trên đối tượng*, nhưng không *hướng đối tượng* bởi vì nó không hỗ trợ các lớp cũng như tính thừa kế. Phần này nói về các đối tượng trong JavaScript và hình 6.1 chỉ ra sơ đồ phân cấp các đối tượng.

Trong sơ đồ phân cấp các đối tượng của JavaScript, các đối tượng con thực sự là các thuộc tính của các đối tượng bố mẹ. Trong ví dụ về chương trình xử lý sự kiện trước đây form tên PHIEU_DIEU_TRA là thuộc tính của đối tượng *document* và trường *text AGE* là thuộc tính của form PHIEU_DIEU_TRA. Để tham chiếu đến giá trị của AGE, bạn phải sử dụng:

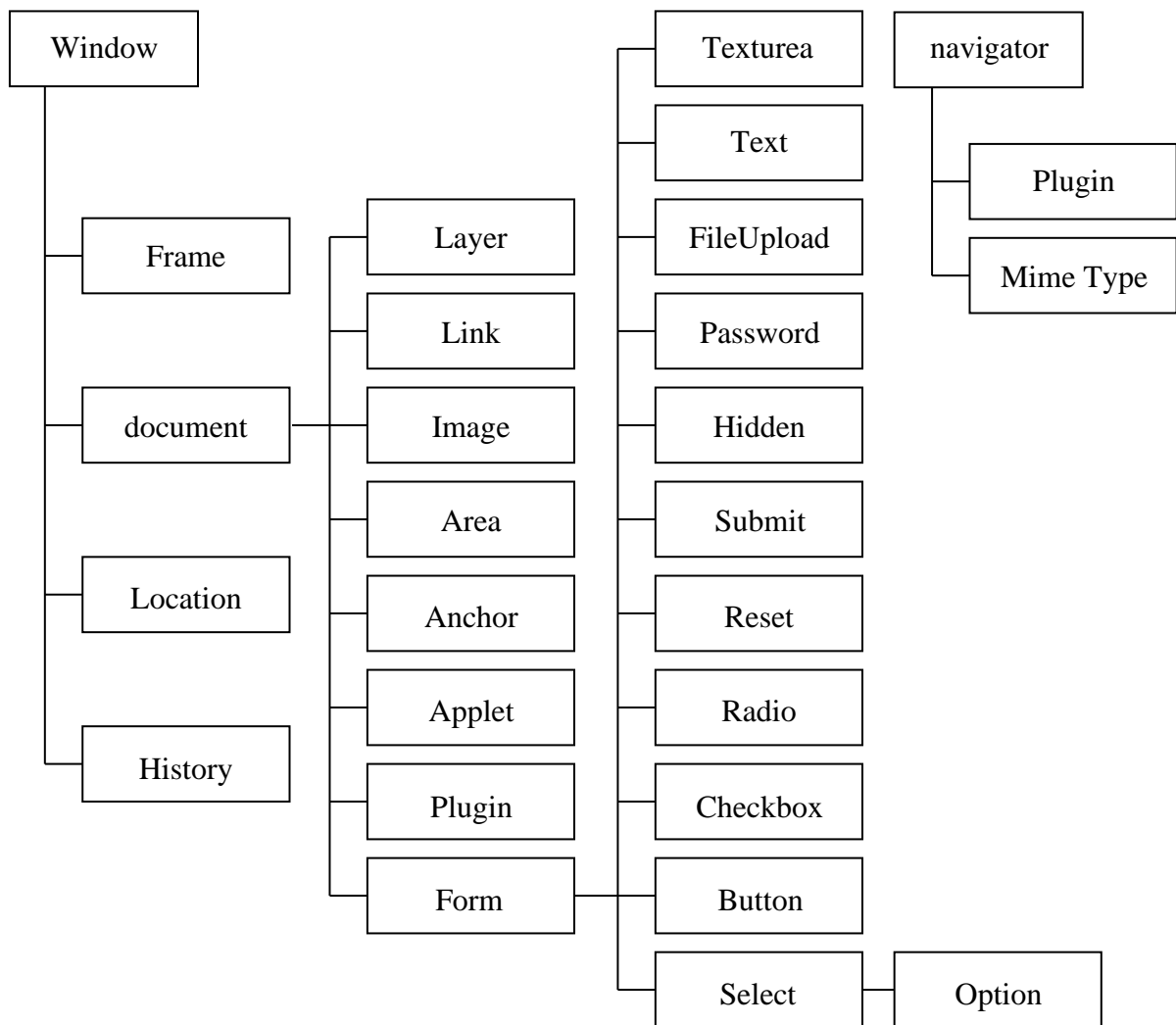
```
document.PHIEU_DIEU_TRA.AGE.value
```

Các đối tượng có thuộc tính (properties), phương thức (methods), và các chương trình xử lý sự kiện (event handlers) gắn với chúng. Ví dụ đối tượng *document* có thuộc tính *title* phản ánh nội dung của thẻ *<TITLE>* của *document*. Bên cạnh đó bạn thấy phương thức *document.write* được sử dụng trong nhiều ví dụ để đưa văn bản kết quả ra *document*.

Đối tượng cũng có thể có các chương trình xử lý sự kiện. Ví dụ đối tượng *link* có hai chương trình xử lý sự kiện là *onClick* và *onMouseOver*. *onClick* được gọi khi có đối tượng *link* được kích, *onMouseOver* được gọi khi con trỏ chuột di chuyển qua link.

Khi bạn tải một document xuống Navigator, nó sẽ tạo ra một số đối tượng cùng với những giá trị các thuộc tính của chúng dựa trên file HTML của document đó và một vài thông tin cần thiết khác. Những đối tượng này tồn tại một cách có cấp bậc và phản ánh chính cấu trúc của file HTML đó.

Sơ đồ sau sẽ minh họa sự phân cấp của các đối tượng này



Hình 6.1: Sơ đồ 1 U Phân cấp đối tượng Navigator

Trong sơ đồ phân cấp này, các đối tượng con chính là các thuộc tính của một đối tượng cha. Ví dụ như một form tên là `form1` chính là một đối tượng con của đối tượng `document` và được gọi tới là `document.form1`

Tất cả các trang đều có các đối tượng sau đây:

- ã **navigator**: có các thuộc tính tên và phiên bản của Navigator đang được sử dụng, dùng cho *MIME type* được hỗ trợ bởi client và plugin được cài đặt trên client.
- ã **window**: là đối tượng ở mức cao nhất, có các thuộc tính thực hiện áp dụng vào toàn bộ cửa sổ.
- ã **document**: chứa các thuộc tính dựa trên nội dung của document như tên, màu nền, các kết nối và các forms.
- ã **location**: có các thuộc tính dựa trên địa chỉ URL hiện thời

• **history**: Chứa các thuộc tính về các URL mà client yêu cầu tr-ớc đó.

Sau đây sẽ mô tả các thuộc tính, ph-ơng thức cũng nh- các ch-ơng trình xử lý sự kiện cho từng đối t-ợng trong JavaScript.

ĐỐI T-ỢNG NAVIGATOR

Đối t-ợng này đ-ợc sử dụng để đạt đ-ợc các thông tin về trình duyệt nh- số phiên bản. Đối t-ợng này không có ph-ơng thức hay ch-ơng trình xử lý sự kiện.

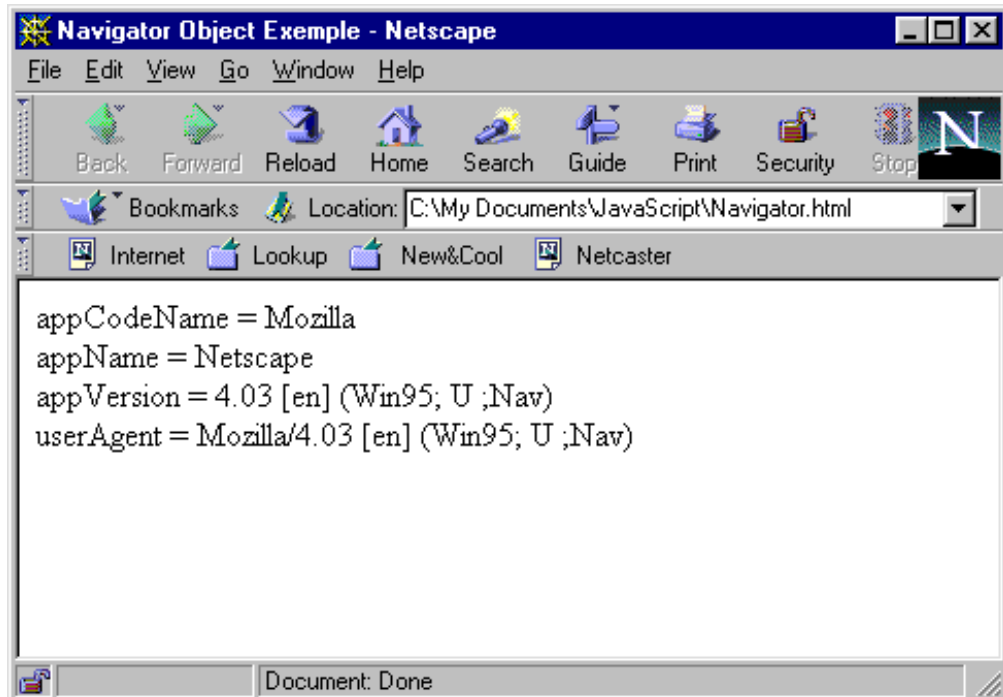
Các thuộc tính

<i>appCodeName</i>	Xác định tên mã nội tại của trình duyệt (Atlas).
<i>AppName</i>	Xác định tên trình duyệt.
<i>AppVersion</i>	Xác định thông tin về phiên bản của đối t-ợng navigator.
<i>userAgent</i>	Xác định header của user U agent.

Ví dụ

Ví dụ sau sẽ hiển thị các thuộc tính của đối t-ợng navigator

```
<HTML>
<HEAD>
<TITLE> Navigator Object Exemple </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
    document.write("appCodeName = "+navigator.appCodeName +
        "<BR>");
    document.write("appName = "+navigator.appName + "<BR>");
    document.write("appVersion = "+navigator.appVersion +
        "<BR>");
    document.write("userAgent = "+navigator.userAgent + "<BR>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```



Hình 6.2: Minh hoạ cho đối tượng Navigator

ĐỐI TƯỢNG WINDOW

Đối tượng window nh- đã nói ở trên là đối tượng ở mức cao nhất. Các đối tượng document, frame, vị trí đều là thuộc tính của đối tượng window.

4.1.1. Các thuộc tính

- ã **defaultStatus** U Thông báo ngầm định hiển thị lên trên thanh trạng thái của cửa sổ
- ã **Frames** U Mảng xác định tất cả các frame trong cửa sổ.
- ã **Length** U Số lượng các frame trong cửa sổ cha mẹ.
- ã **Name** U Tên của cửa sổ hiện thời.
- ã **Parent** U Đối tượng cửa sổ cha mẹ
- ã **Self** U Cửa sổ hiện thời.
- ã **Status** U Được sử dụng cho thông báo tạm thời hiển thị lên trên thanh trạng thái cửa sổ. Được sử dụng để lấy hay đặt lại thông báo trạng thái và ghi đè lên defaultStatus.
- ã **Top** U Cửa sổ ở trên cùng.
- ã **Window** U Cửa sổ hiện thời.

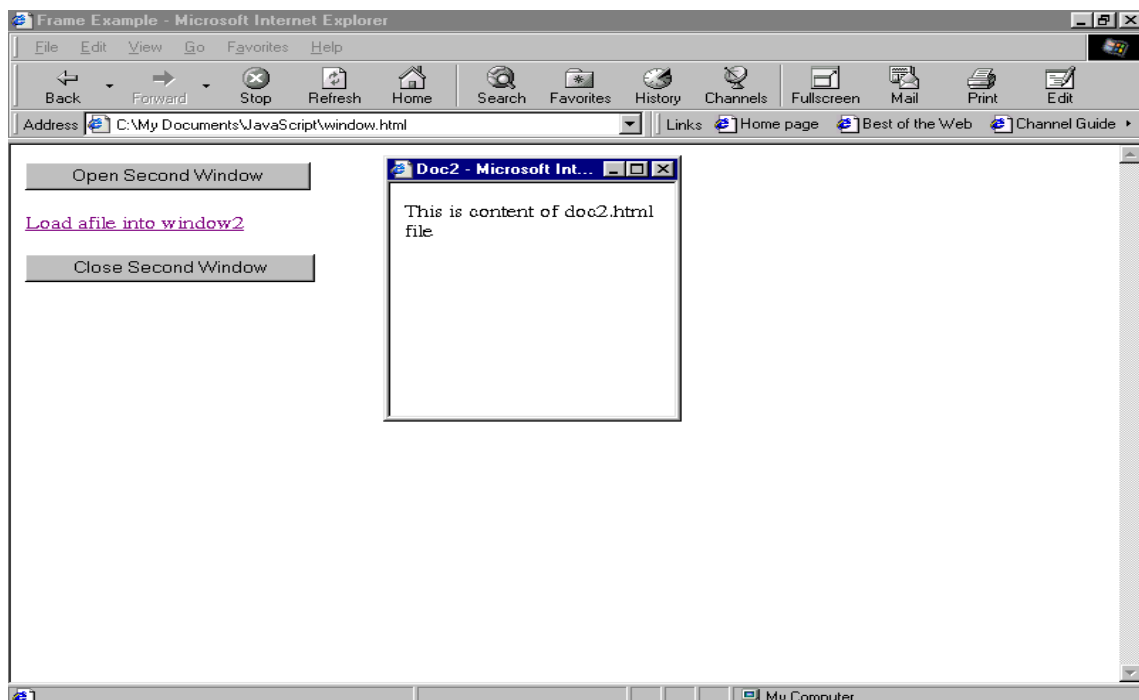
4.1.2. Các phương thức

- ã `alert ("message")` Ứ Hiển thị hộp hội thoại với chuỗi "message" và nút OK.
- ã `clearTimeout(timeoutID)` Ứ Xóa timeout do `setTimeout` đặt. `setTimeout` trả lại `timeoutID`
- ã `windowReference.close` Ứ Đóng cửa sổ `windowReference`.
- ã `confirm("message")` Ứ Hiển thị hộp hội thoại với chuỗi "message", nút OK và nút Cancel. Trả lại giá trị True cho OK và False cho Cancel.
- ã `[window/Var =][window]. open("URL", "windowName", ["windowFeatures"])` Ứ Mở cửa sổ mới.
- ã `prompt ("message" [, "defaultInput"])` Ứ Mở một hộp hội thoại để nhận dữ liệu vào trường text.
- ã `TimeoutID = setTimeout(expression, msec)` Ứ Đánh giá biểu thức `expression` sau thời gian `msec`.

Ví dụ: Sử dụng tên cửa sổ khi gọi tới nó nh- là đích của một form submit hoặc trong một Hypertext link (thuộc tính TARGET của thẻ FORM và A).

Trong ví dụ tạo ra một tới cửa sổ thứ hai, nh- nút thứ nhất để mở một cửa sổ rỗng, sau đó một liên kết sẽ tải file `doc2.html` xuống cửa sổ mới đó rồi một nút khác dùng để đóng cửa sổ thứ hai lại, ví dụ này l-a vào file *window.html*:

```
<HTML>
<HEAD>
<TITLE>Frame Example </TITLE>
</HEAD>
<BODY>
<FORM>
  <INPUT TYPE="button" VALUE="Open Second Window"
    onClick="msgWindow=window.open('','window2','resizable=
      no,width=200,height=200') ">
</FORM>
<P>
<A HREF="doc2.html" TARGET="window2">
Load a file into window2 </A>
</P>
<INPUT TYPE="button" VALUE="Close Second Window"
  onClick="msgWindow.close() ">
</FORM>
</BODY>
</HTML>
```

Hình 6.3: Minh họa cho đối tượng cửa sổ

4.1.3. Các chương trình xử lý sự kiện

- onLoad Ƴ Xuất hiện khi cửa sổ kết thúc việc tải.
- onUnload Ƴ Xuất hiện khi cửa sổ đợc loại bỏ.

ĐỐI TƯỢNG LOCATION

Các thuộc tính của đối tượng location duy trì các thông tin về URL của document hiện thời. Đối tượng này hoàn toàn không có các phương thức và chương trình xử lý sự kiện đi kèm. Ví dụ:

`http:// www.abc.com/chap1/page2.html#topic3`

Các thuộc tính

- hash Ƴ Tên anchor của vị trí hiện thời (ví dụ topic3).
- Host Ƴ Phần hostname:port của URL (ví dụ www.abc.com). Chú ý rằng đây thường là cổng ngầm định và ít khi đợc chỉ ra.
- Hostname Ƴ Tên của host và domain (ví dụ www.abc.com).
- href Ƴ Toàn bộ URL cho document hiện tại.
- Pathname Ƴ Phần đường dẫn của URL (ví dụ /chap1/page2.html).
- Port Ƴ Cổng truyền thông đợc sử dụng cho máy tính host, thường là cổng ngầm định.
- Protocol Ƴ Giao thức đợc sử dụng (cùng với dấu hai chấm) (ví dụ http:).
- Search Ƴ Câu truy vấn tìm kiếm có thể ở cuối URL cho các script CGI.

ĐỐI TƯỢNG FRAME

Một cửa sổ có thể có một vài frame. Các frame có thể cuộn một cách độc lập với nhau và mỗi frame có URL riêng. frame không có các chương trình xử lý sự kiện. Sự kiện onLoad và onUnload là của đối tượng window.

4.1.4. Các thuộc tính

- frames *U* Mảng tất cả các frame trong cửa sổ.
- Name *U* Thuộc tính NAME của thẻ <FRAME>
- Length *U* Số lượng các frame con trong một frame.
- Parent *U* Cửa sổ hay frame chứa nhóm frame hiện thời.
- self *U* frame hiện thời.
- Window *U* frame hiện thời.

4.1.5. Các phương thức

- clearTimeout (timeoutID) *U* Xóa timeout do setTimeout lập. SetTimeout trả lại timeoutID.
- TimeoutID = setTimeout (expression,msec) *U* Đánh giá expression sau khi hết thời gian msec.

4.1.6. Sử dụng Frame

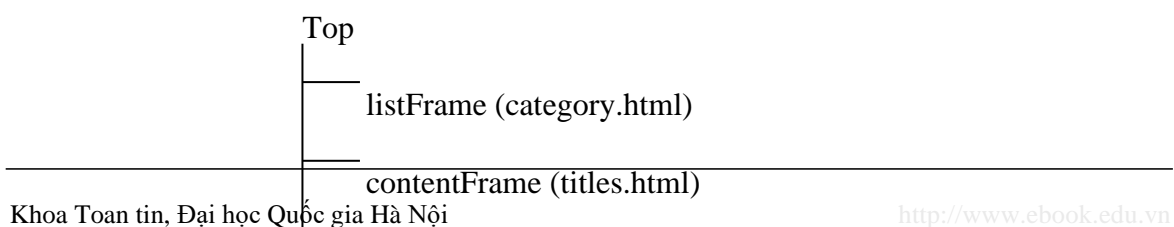
4.1.6.1. a) Tạo một frame (create)

Để tạo một frame, ta sử dụng thẻ **FRAMESET**. Mục đích của thẻ này là định nghĩa một tập các frame trong một trang.

Ví dụ 1: tạo frame (hình 17)

```
<HTML>
<HEAD>
<TITLE>Frame Example </TITLE>
<FRAMESET ROWS="90%,10%">
  <FRAMESET COLS="30%,70%">
    <FRAME SRC=CATEGORY.HTM NAME="ListFrame">
    <FRAME SRC=TITLES.HTM NAME="contentFrame">
  </FRAMESET >
  <FRAME SRC=NAVIGATOR.HTM NAME="navigateFrame">
</FRAMESET >
</HEAD>
<BODY> </BODY>
</HTML>
```

Sơ đồ sau hiển thị cấu trúc của các frame: Cả 3 frame đều trên cùng một cửa sổ cha, mặc dù 2 trong số các frame đó nằm trong một frameset khác.

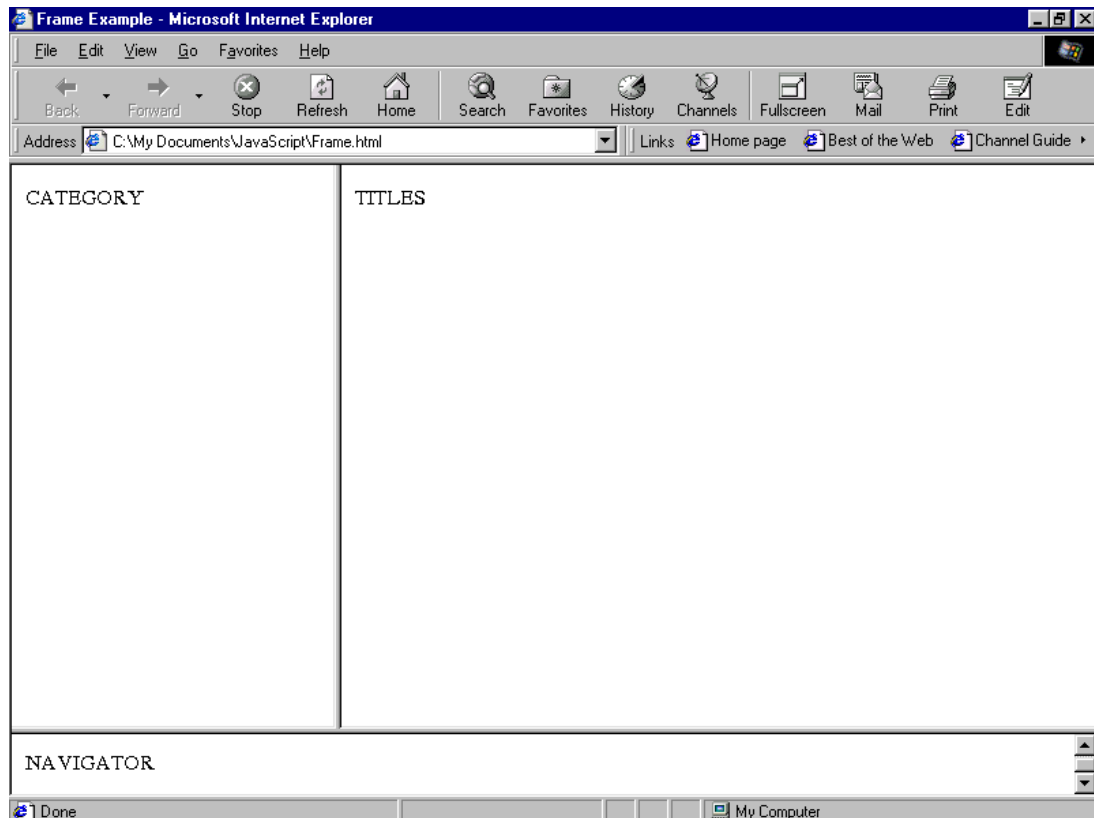


Bạn có thể gọi tới những frame tr-ớc đó bằng cách sử dụng thuộc tính **frames** nh- sau:

listFrame chính là `top.frames[0]`

contentFrame chính là `top.frames[1]`

navigatorFrame chính là `top.frames[2]`



Hình 6.4: Kết quả việc tạo frame trong

Ví dụ 2: Cũng giống nh- một sự lựa chọn, bạn có thể tạo ra một cửa sổ giống nh- ví dụ tr-ớc nh-ng trong mỗi đỉnh của hai frame lại có một cửa sổ cha riêng từ **navigateFrame**. Mức frameset cao nhất có thể đ-ọc định nghĩa nh- sau:

```
<HTML>
<HEAD>
<TITLE>Frame Example </TITLE>
<FRAMESET ROWS="90%,10%">
<FRAME SRC=muskel3.HTML NAME="upperFrame">
<FRAME SRC=NAVIGATOR.HTM NAME="navigateFrame">
</FRAMESET >
</HEAD>
<BODY>
</BODY>
</HTML>
```

Trong file muskel3.html lại tiếp tục đặt một frameset:

```
<HTML>
<HEAD>
```

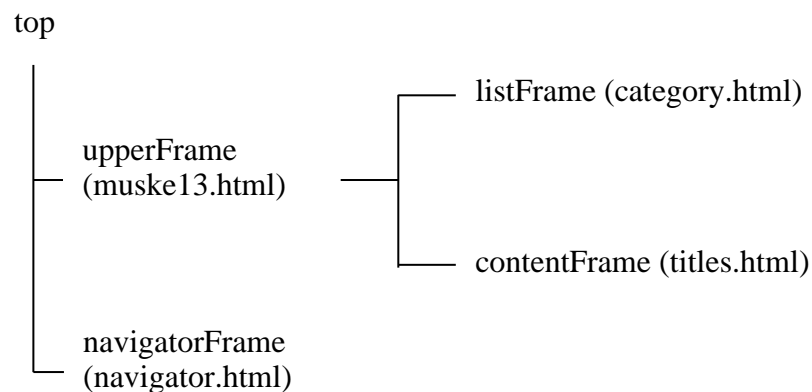
```

<TITLE>Frame Example </TITLE>
<FRAMESET COLS="30%,70%">
<FRAME SRC=CATEGORY.HTM NAME="ListFrame">
<FRAME SRC=TITLES.HTM NAME="contentFrame">
</FRAMESET >
</HEAD>
<BODY>
</BODY>
</HTML>

```

Khi đó kết quả hiển thị của ví dụ 2 giống ví dụ 1 nh-ng sự phân cấp của các frames lại khác hẳn:

Bạn có thể gọi tới các frame trên bằng cách sử dụng thuộc tính mảng **frames** nh- sau:



upperFrame chính là **top.frames[0]**

navigatorFrame chính là **top.frames[1]**

listFrame chính là **upperFrame.frames[0]**
hoặc **top.frames[0].frames[0]**

contentFrame chính là **upperFrame.frames[1]**
hoặc **top.frames[0].frames[1]**

4.1.6.2. b) Cập nhật một frame (update)

Bạn có thể cập nhật nội dung của một frame bằng cách sử dụng thuộc tính **location** để đặt địa chỉ URL và phải định chỉ rõ vị trí của frame trong cấu trúc.

Trong ví dụ trên, nếu bạn thêm một dòng sau vào **navigatorFrame**:

```

<INPUT TYPE="button" VALUE="Titles only"
onClick="top.frames[0].location='artist.html'">

```

thì khi nút “Titles only” đ-ợc nhấn, file **artist.html** sẽ đ-ợc tải vào **upperFrame**, và hai frame **listFrame**, **contentFrame** sẽ bị đóng lại nh- chúng ch-a bao giờ tồn tại.

ĐỐI T-ỌNG DOCUMENT

Đối t-ợng này chứa các thông tin về document hiện thời và cung cấp các ph-ơng thức để đ-a thông tin ra màn hình. Đối t-ợng document đ-ợc tạo ra bằng cặp thẻ **<BODY>** và **</BODY>**. Một số các thuộc tính gắn với thẻ **<BODY>**.

Các đối tượng `anchor`, `forms`, `history`, `links` là thuộc tính của đối tượng `document`. Không có các chương trình xử lý sự kiện cho các `frame`. Sự kiện `onLoad` và `onUnload` là cho đối tượng `window`.

4.1.7. Các thuộc tính

- `alinkColor` Ứ Giống nh- thuộc tính `ALINK`.
- `anchor` Ứ Mảng tất cả các `anchor` trong `document`.
- `bgColor` Ứ Giống thuộc tính `BGCOLOR`.
- `cookie` Ứ Sử dụng để xác định `cookie`.
- `fgColor` Ứ Giống thuộc tính `TEXT`.
- `forms` Ứ Mảng tất cả các `form` trong `document`.
- `lastModified` Ứ Ngày cuối cùng văn bản đ-ợc sửa.
- `linkColor` Ứ Giống thuộc tính `LINK`.
- `links` Ứ Mảng tất cả các `link` trong `document`.
- `location` Ứ URL đầy đủ của văn bản.
- `referrer` Ứ URL của văn bản gọi nó.
- `title` Ứ Nội dung của thẻ `<TITLE>`.
- `vlinkColor` Ứ Giống thuộc tính `VLINK`.

4.1.8. Các ph-ơng thức

- `document.clear` Ứ Xóa `document` hiện thời.
- `document.close` Ứ Đóng dòng dữ liệu vào và đ-a toàn bộ dữ liệu trong bộ đệm ra màn hình.
- `document.open(["mimeType"])` Ứ Mở một stream để thu thập dữ liệu vào của các ph-ơng thức `write` và `writeln`.
- `document.write(expression1 [,expression2]...[,expressionN])` Ứ Viết biểu thức HTML lên văn bản trong một cửa sổ xác định.
- `document.writeln (expression1 [,expression2] ... [,expressionN])` Ứ Giống ph-ơng thức trên nh-ng khi hết mỗi biểu thức lại xuống dòng.

ĐỐI TƯỢNG ANCHORS

`anchor` là một đoạn văn bản trong `document` có thể dùng làm đích cho một siêu liên kết. Nó đ-ợc xác định bằng cặp thẻ `<A>` và ``. Đối tượng `anchor` không có thuộc tính, ph-ơng thức cũng nh- chương trình xử lý sự kiện. Mảng `anchor` tham chiếu đến mỗi `anchor` có tên trong `document`. Mỗi `anchor` đ-ợc tham chiếu bằng cách:

```
document.anchors [index]
```

Mảng `anchor` có một thuộc tính duy nhất là `length` xác định số l-ợng các `anchor` trong `document`, nó có thể đ-ợc xác định nh- sau:

```
document.anchors.length.
```

ĐỐI TƯỢNG FORMS

Các form được tạo ra nhờ cặp thẻ `<FORM>` và `</FORM>`. Phần lớn các thuộc tính của đối tượng form phản ánh các thuộc tính của thẻ `<FORM>`. Có một vài phần tử (elements) là thuộc tính của đối tượng forms:

```
button
checkbox
hidden
password
radio
reset
select
submit
text
textarea
```

Các phần tử này sẽ được trình bày sau.

Nếu document chứa một vài form, chúng có thể được tham chiếu qua mảng forms. Số lượng các form có thể được xác định như sau:

```
document.forms.length.
```

Mỗi một form có thể được tham chiếu như sau:

```
document.forms[index]
```

4.1.9. Các thuộc tính

action	thuộc tính ACTION của thẻ FORM.
elements	Mảng chứa tất cả các thành phần trong một form (như checkbox, trường text, danh sách lựa chọn)
encoding	Xâu chứa kiểu MIME được sử dụng để mã hoá nội dung của form gửi cho server.
length	Số lượng các thành phần trong một form.
method	Thuộc tính METHOD.
target	Xâu chứa tên của cửa sổ đích khi submit form

4.1.10. Các phương thức

`formName.submit ()` Xuất dữ liệu của một form tên formName tới trang xử lý. Phương thức này mô phỏng một click vào nút submit trên form.

4.1.11. Các chương trình xử lý sự kiện

onSubmit Chương trình xử lý sự kiện này được gọi khi người sử dụng chuyển dữ liệu từ form đi.

ĐỐI TƯỢNG HISTORY

Đối tượng này được sử dụng để lưu giữ các thông tin về các URL trước được người sử dụng sử dụng. Danh sách các URL được lưu trữ theo thứ tự thời gian. Đối tượng này không có chương trình xử lý sự kiện.

4.1.12. Các thuộc tính

`length` Số lượng các URL trong đối tượng.

4.1.13. Các phương thức

- `history.back()` Được sử dụng để tham chiếu tới URL mới được thăm trước đây.
- `history.forward()` Được sử dụng để tham chiếu tới URL kế tiếp trong danh sách. Nó sẽ không gây hiệu ứng gì nếu đã đến cuối của danh sách.
- `history.go (delta | "location")` Được sử dụng để chuyển lên hay chuyển xuống `delta` bậc hay di chuyển đến URL xác định bởi `location` trong danh sách. Nếu `delta` được sử dụng thì việc dịch chuyển lên phía trên khi `delta` dương và xuống phía dưới khi `delta` âm. nếu sử dụng `location`, URL gần nhất có chứa `location` là chuỗi con sẽ được tham chiếu.

ĐỐI TƯỢNG LINKS

Đối tượng link là một đoạn văn bản hay một ảnh được xem là một siêu liên kết. Các thuộc tính của đối tượng link chủ yếu xử lý về URL của các siêu liên kết. Đối tượng link cũng không có phương thức nào.

Mảng link chứa danh sách tất cả các liên kết trong document. Có thể xác định số lượng các link qua

```
document.links.length()
```

Có thể tham chiếu tới một liên kết qua

```
document.links [index]
```

Để xác định các thuộc tính của đối tượng link, có thể sử dụng URL tương tự:

```
http://www.abc.com/chap1/page2.html#topic3
```

4.1.14. Các thuộc tính

- `hash` Tên anchor của vị trí hiện thời (ví dụ `topic3`).
- `Host` Phần `hostname:port` của URL (ví dụ www.abc.com). Chú ý rằng đây thường là cổng ngầm định và ít khi được chỉ ra.
- `Hostname` Tên của host và domain (ví dụ `ww.abc.com`).
- `href` Toàn bộ URL cho document hiện tại.
- `Pathname` Phần đường dẫn của URL (ví dụ `/chap1/page2.html`).
- `port` Cổng truyền thông được sử dụng cho máy tính host, thường là cổng ngầm định.

- ã Protocol *U* Giao thức đ-ợc sử dụng (cùng với dấu hai chấm) (ví dụ http:).
- ã Search *U* Câu truy vấn tìm kiếm có thể ở cuối URL cho các script CGI.
- ã Target *U* Giống thuộc tính TARGET của <LINK>.

4.1.15. Các ch-ơng trình xử lý sự kiện

- ã onClick *U* Xảy ra khi ng-ời sử dụng nhấn vào link.
- ã onMouseOver *U* Xảy ra khi con chuột di chuyển qua link.

ĐỐI T-ỌNG MATH

Đối t-ợng Math là đối t-ợng nội tại trong JavaScript. Các thuộc tính của đối t-ợng này chứa nhiều hằng số toán học, các hàm toán học, l-ợng giác phổ biến. Đối t-ợng Math không có ch-ơng trình xử lý sự kiện.

Việc tham chiếu tới **number** trong các ph-ơng thức có thể là số hay các biểu thức đ-ợc đánh giá là số hợp lệ.

4.1.16. Các thuộc tính

- ã E *U* Hằng số Euler, khoảng 2,718.
- ã LN2 *U* logarit tự nhiên của 2, khoảng 0,693.
- ã LN10 *U* logarit tự nhiên của 10, khoảng 2,302
- ã LOG2E *U* logarit cơ số 2 của e, khoảng 1,442.
- ã PI *U* Giá trị của p, khoảng 3,14159.
- ã SQRT1_2 *U* Căn bậc 2 của 0,5, khoảng 0,707.
- ã SQRT2 *U* Căn bậc 2 của 2, khoảng 1,414.

4.1.17. Các ph-ơng thức

- ã Math.abs (*number*) *U* Trả lại giá trị tuyệt đối của *number*.
- ã Math.acos (*number*) *U* Trả lại giá trị arc cosine (theo radian) của *number*. Giá trị của *number* phải nằm giữa -1 và 1.
- ã Math.asin (*number*) *U* Trả lại giá trị arc sine (theo radian) của *number*. Giá trị của *number* phải nằm giữa -1 và 1.
- ã Math.atan (*number*) *U* Trả lại giá trị arc tan (theo radian) của *number*.
- ã Math.ceil (*number*) *U* Trả lại số nguyên nhỏ nhất lớn hơn hoặc bằng *number*.
- ã Math.cos (*number*) *U* Trả lại giá trị cosine của *number*.
- ã Math.exp (*number*) *U* Trả lại giá trị e^{number} , với e là hằng số Euler.
- ã Math.floor (*number*) *U* Trả lại số nguyên lớn nhất nhỏ hơn hoặc bằng *number*.

- ã `Math.log (number)` Trả lại logarit tự nhiên của *number*.
- ã `Math.max (num1,num2)` Trả lại giá trị lớn nhất giữa *num1* và *num2*
- ã `Math.min (num1,num2)` Trả lại giá trị nhỏ nhất giữa *num1* và *num2*.
- ã `Math.pos (base,exponent)` Trả lại giá trị *base* lũy thừa *exponent*.
- ã `Math.random (r)` Trả lại một số ngẫu nhiên giữa 0 và 1. Phworng thức này chỉ thực hiện đợc trên nền tảng UNIX.
- ã `Math.round (number)` Trả lại giá trị của *number* làm tròn tới số nguyên gần nhất.
- ã `Math.sin (number)` Trả lại sin của *number*.
- ã `Math.sqrt (number)` Trả lại căn bậc 2 của *number*.
- ã `Math.tan (number)` Trả lại tag của *number*.

ĐỐI T-ỢNG DATE

Đối t-ợng Date là đối t-ợng có sẵn trong JavaScript. Nó cung cấp nhiều ph-ơng thức có ích để xử lý về thời gian và ngày tháng. Đối t-ợng Date không có thuộc tính và ch-ơng trình xử lý sự kiện.

Phần lớn các ph-ơng thức date đều có một đối t-ợng Date đi cùng. Các ph-ơng thức giới thiệu trong phần này sử dụng đối t-ợng Date *dateVar*, ví dụ:

```
dateVar = new Date ('August 16, 1996 20:45:04');
```

4.1.18. Các ph-ơng thức

- ã `dateVar.getDate()` Trả lại ngày trong tháng (1U31) cho *dateVar*.
- ã `dateVar.getDay()` Trả lại ngày trong tuần (0=chủ nhật,...6=thứ bảy) cho *dateVar*.
- ã `dateVar.getHours()` Trả lại giờ (0U23) cho *dateVar*.
- ã `dateVar.getMinutes()` Trả lại phút (0U59) cho *dateVar*.
- ã `dateVar.getSeconds()` Trả lại giây (0U59) cho *dateVar*.
- ã `dateVar.getTime()` Trả lại số l-ợng các mili giây từ 00:00:00 ngày 1/1/1970.
- ã `dateVar.getTimeZoneOffset()` Trả lại độ dịch chuyển bằng phút của giờ địa ph-ơng hiện tại so với giờ quốc tế GMT.
- ã `dateVar.getYear()` Trả lại năm cho *dateVar*.
- ã `Date.parse (dateStr)` Phân tích chuỗi *dateStr* và trả lại số l-ợng các mili giây tính từ 00:00:00 ngày 01/01/1970.
- ã `dateVar.setDay(day)` Đặt ngày trong tháng là *day* cho *dateVar*.
- ã `dateVar.setHours(hours)` Đặt giờ là *hours* cho *dateVar*.
- ã `dateVar.setMinutes(minutes)` Đặt phút là *minutes* cho *dateVar*.
- ã `dateVar.setMonths(months)` Đặt tháng là *months* cho *dateVar*.

- ã `dateVar.setSeconds(seconds)` Ư Đặt giây là *seconds* cho *dateVar*.
- ã `dateVar.setTime(value)` Ư Đặt thời gian là *value*, trong đó *value* biểu diễn số lượng mili giây từ 00:00:00 ngày 01/01/10970.
- ã `dateVar.setYear(years)` Ư Đặt năm là *years* cho *dateVar*.
- ã `dateVar.toGMTString()` Ư Trả lại chuỗi biểu diễn *dateVar* d-ới dạng GMT.
- ã `dateVar.toLocaleString()` Ư Trả lại chuỗi biểu diễn *dateVar* theo khu vực thời gian hiện thời.
- ã `Date.UTC (year, month, day [,hours] [,minutes] [,seconds])` Ư Trả lại số lượng mili giây từ 00:00:00 01/01/1970 GMT.

ĐỐI T-ỢNG STRING

Đối t-ợng String là đối t-ợng đ-ọc xây dựng nội tại trong JavaScript cung cấp nhiều ph-ơng thức thao tác trên chuỗi. Đối t-ợng này có thuộc tính duy nhất là độ dài (length) và không có ch-ơng trình xử lý sự kiện.

4.1.19. Các ph-ơng thức

- ã `str.anchor (name)` Ư Đ-ọc sử dụng để tạo ra thẻ <A> (một cách động). Tham số *name* là thuộc tính NAME của thẻ <A>.
- ã `str.big()` Ư Kết quả giống nh- thẻ <BIG> trên chuỗi *str*.
- ã `str.blink()` Ư Kết quả giống nh- thẻ <BLINK> trên chuỗi *str*.
- ã `str.bold()` Ư Kết quả giống nh- thẻ <BOLD> trên chuỗi *str*.
- ã `str.charAt(a)` Ư Trả lại ký tự thứ *a* trong chuỗi *str*.
- ã `str.fixed()` Ư Kết quả giống nh- thẻ <TT> trên chuỗi *str*.
- ã `str.fontcolor()` Ư Kết quả giống nh- thẻ <FONTCOLOR = *color*>.
- ã `str.fontSize(size)` Ư Kết quả giống nh- thẻ <FONTSIZE = *size*>.
- ã `str.indexOf(srchStr [,index])` Ư Trả lại vị trí trong chuỗi *str* vị trí xuất hiện đầu tiên của chuỗi *srchStr*. Chuỗi *str* đ-ọc tìm từ trái sang phải. Tham số *index* có thể đ-ọc sử dụng để xác định vị trí bắt đầu tìm kiếm trong chuỗi.
- ã `str italics()` Ư Kết quả giống nh- thẻ <I> trên chuỗi *str*.
- ã `str.lastIndexOf(srchStr [,index])` Ư Trả lại vị trí trong chuỗi *str* vị trí xuất hiện cuối cùng của chuỗi *srchStr*. Chuỗi *str* đ-ọc tìm từ phải sang trái. Tham số *index* có thể đ-ọc sử dụng để xác định vị trí bắt đầu tìm kiếm trong chuỗi.
- ã `str.link(href)` Ư Đ-ọc sử dụng để tạo ra một kết nối HTML động cho chuỗi *str*. Tham số *href* là URL đích của liên kết.
- ã `str.small()` Ư Kết quả giống nh- thẻ <SMALL> trên chuỗi *str*.
- ã `str.strike()` Ư Kết quả giống nh- thẻ <STRIKE> trên chuỗi *str*.
- ã `str.sub()` Ư Tạo ra một subscript cho chuỗi *str*, giống thẻ <SUB>.

- ã `str.substring(a,b)` ⌈ Trả lại chuỗi con của `str` là các ký tự từ vị trí thứ `a` tới vị trí thứ `b`. Các ký tự đ-ọc đếm từ trái sang phải bắt đầu từ 0.
- ã `str.sup()` ⌈ Tạo ra superscript cho chuỗi `str`, giống thẻ `<SUP>`.
- ã `str.toLowerCase()` ⌈ Đ-ổi chuỗi `str` thành chữ th-ờng.
- ã `str.toUpperCase()` ⌈ Đ-ổi chuỗi `str` thành chữ hoa.

CÁC PHẦN TỬ CỦA Đ-ỐI T-ƯỢNG FORM

Form đ-ọc tạo bởi các phần tử cho phép ng-ời sử dụng đ-a thông tin vào. Khi đó, nội dung (hoặc giá trị) của các phần tử sẽ đ-ọc chuyển đến một ch-ơng trình trên server qua một giao diện đ-ọc gọi là Common Gateway Interface(Giao tiếp qua một cổng chung) gọi tắt là CGI

Sử dụng JavaScript bạn có thể viết những đoạn scripts chèn vào HTML của bạn để làm việc với các phần tử của form và các giá trị của chúng.

Bảng ??: Các phần tử của form

Phần tử	Mô tả
button	Là một nút bấm hơn là nút submit hay nút reset (<code><INPUT TYPE="button"></code>)
checkbox	Một checkbox (<code><INPUT TYPE="checkbox"></code>)
FileUpload	Là một phần tử tải file lên cho phép ng-ời sử dụng gửi lên một file (<code><INPUT TYPE="file"></code>)
hidden	Một tr-ờng ẩn (<code><INPUT TYPE="hidden"></code>)
password	Một tr-ờng text để nhập mật khẩu mà tất cả các ký tự nhập vào đều hiển thị là dấu (*) (<code><INPUT TYPE="password"></code>)
radio	Một nút bấm (<code><INPUT TYPE="radio"></code>)
reset	Một nút reset(<code><INPUT TYPE="reset"></code>)
select	Một danh sách lựa chọn (<code><SELECT><OPTION>option1</OPTION><OPTION>option2</OPTION></SELECT></code>)
submit	Một nút submit (<code><INPUT TYPE="submit"></code>)
text	Một tr-ờng text (<code><INPUT TYPE="text"></code>)
textArea	Một tr-ờng text cho phép nhập vấp nhiều dòng <code><TEXTAREA>default text</TEXTAREA></code>)

Mỗi phần tử có thể đ-ọc đặt tên để JavaScript truy nhập đến chúng qua tên

4.1.20. Thuộc tính type

Trong mỗi phần tử của form đều có thuộc tính type, đó là một chuỗi chỉ định rõ kiểu của phần tử đ-ợc đ-a vào nh- nút bấm, một tr-ờng text hay một checkbox...

Chuỗi đó có thể là một trong các giá trị sau:

Text field: "text"

Radio button: "radio"

Checkbox: "checkbox"

Hidden field: "hidden"

Submit button: "submit"

Reset button: "reset"

Password field: "password"

Button: "button"

Select list: "selectUone"

Multiple select lists: "selectUmultiple"

Textarea field: "textarea"

4.1.21. Phần tử button

Trong một form HTML chuẩn, chỉ có hai nút bấm có sẵn là submit và reset bởi vì dữ liệu trong form phải đ-ợc gửi tới một vài địa chỉ URL (th-ờng là CGIUBIN script) để xử lý và l-u trữ.

Một phần tử button đ-ợc chỉ định rõ khi sử dụng thẻ INPUT:

```
<INPUT TYPE="button" NAME="name" VALUE= "buttonName">
```

Trong thẻ INPUT, name là tên của button, thuộc tính VALUE có chứa nhãn của button sẽ đ-ợc hiển thị trên Navigator của browser.

Chỉ có một thẻ sự kiện duy nhất đối với button là **onClick**. Kết hợp với nó là cách thức duy nhất **click**. Phần tử button có khả năng mở rộng cho phép ng-ời lập trình JavaScript có thể viết đ-ợc một đoạn mã lệnh JavaScript để thực thi việc thêm vào một nút bấm trong một script.

Trong ví dụ sau, thay vì sử dụng onChange, bạn có thể chỉnh sửa script để định giá biểu thức khi button đ-ợc bấm.

Ví dụ: Định giá một form sử dụng phần tử button.

```
<HTML>
<HEAD>
<TITLE>button Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!UU HIDE FROM OTHER BROWSERS
function calculate(form) {
form.results.value = eval(form.entry.value);
}
```

```
// STOP HIDING FROM OTHER BROWSERS UU>
</SCRIPT>
</HEAD>
<BODY>
<FORM METHOD=POST>
Enter a JavaScript mathematical expression:
<INPUT TYPE="text" NAME="entry" VALUE="">
<BR>
The result of this expression is:
<INPUT TYPE=text NAME="results" onFocus="this.blur();">
<BR>
<INPUT TYPE="button" VALUE="Calculate"
      onClick="calculate(this.form);">
</FORM>
</BODY>
</HTML>
```

4.1.22. Phần tử checkbox

Các phần tử checkbox có khả năng bật tắt dùng để chọn hoặc không chọn một thông tin. Các checkbox có nhiều thuộc tính và cách thức hơn button. Bảng dưới đây là danh sách các thuộc tính và các cách thức của phần tử checkbox.

Bảng . Các thuộc tính và cách thức của phần tử checkbox.

Cách thức và thuộc tính	Mô tả
checked	Cho biết trạng thái hiện thời của checkbox (thuộc tính)
defaultChecked	Cho biết trạng thái mặc định của phần tử (thuộc tính)
name	Cho biết tên của phần tử để chỉ định trong thẻ INPUT (thuộc tính)
value	Cho biết giá trị hiện thời của phần tử để chỉ định trong thẻ INPUT (thuộc tính)
click()	Mô tả một click vào checkbox (Cách thức)

Phần tử checkbox chỉ có một thẻ sự kiện là onClick

Ví dụ: Tạo hộp checkbox để nhập vào một số rồi lựa chọn tính nhân đôi và bình phương:

```
<HTML>
<HEAD>
<TITLE>checkbox Example</TITLE>
<SCRIPT>
<!UU HIDE FROM OTHER BROWSERS
function calculate(form,callingField) {
if (callingField == "result") { // if(1)
```

```

        if (form.square.checked) { // if(2)
            form.entry.value = Math.sqrt(form.result.value);
        }
        else {
            form.entry.value = form.result.value / 2;
        } //end if(2)
    }
else{
    if (form.square.checked) { // if(3)
        form.result.value=form.entry.value*form.entry.value;
    }
        else {
            form.result.value = form.entry.value * 2;
        } //enfzd if(3)
} //end if(1)

} //end function
// STOP HIDING FROM OTHER BROWSERS UU>
</SCRIPT>
</HEAD>
<BODY>
<FORM METHOD=POST>
Value: <INPUT TYPE="text" NAME="entry" VALUE=0
onChange="calculate(this.form,this.name);">
<BR>
Action (default double): <INPUT TYPE=checkbox NAME=square
onClick="calculate(this.form,this.name);">
Square
<BR>
Result: <INPUT TYPE="text" NAME="result" VALUE=0
onChange="calculate(this.form,this.name);">
</FORM>
</BODY>
</HTML>

```

Trong script này, bạn đã thấy cách sử dụng thẻ sự kiện onClick cũng như thuộc tính checked là một giá trị kiểu Boolean có thể dùng làm điều kiện trong câu lệnh if...else

Bạn có thể thêm một checkbox tên là square vào form. Nếu hộp này được check, chương trình sẽ lấy giá trị của nó, nếu không, một thực thi được mặc định sẽ nhân đôi giá trị của nó. Thẻ sự kiện onClick trong checkbox được định nghĩa:

```

(<INPUT TYPE=checkbox NAME=square onClick= "calculate( this.form,
this.name);"> )

```

Khi đó nếu người dùng thay đổi một câu lệnh khác, form sẽ đ-ọc tính toán lại.

Để tạo ra sự mở rộng cho checkbox, bạn có thể thay đổi hàm calculate() nh- sau:

```
function calculate(form,callingField) {
if (callingField == "result") { // if (1)
    if (form.square.checked) { // if (2)
        form.entry.value = Math.sqrt(form.result.value);
    }
    else {
        form.entry.value = form.result.value / 2;
    } //end if(2)
}

    else {
if (form.square.checked) { // if (3)
    form.result.value=form.entry.value*form.entry.value;
}
    else {
        form.result.value = form.entry.value * 2;
    } // end if (3)
} // end if (1)
}
```

4.1.23. Phần tử File Upload

Phần tử này cung cấp cho form một cách để người sử dụng có thể chỉ rõ một file đ-a vào form xử lý. Phần tử file Upload đ-ọc chỉ định rõ trong JavaScript bằng đối tượng **FileUpload**.

Đối tượng chỉ có hai thuộc tính là **name** và **value**, cả hai đều là giá trị xâu nh- các đối tượng khác. Không có cách thức hay thẻ file cho đối tượng này.

4.1.24. Phần tử hidden

Phần tử hidden là phần tử duy nhất trong số tất cả các phần tử của form không đ-ọc hiển thị trên Web browser. Tr-ờng hidden có thể sử dụng để l-u các giá trị cần thiết để gửi tới server song song với sự xuất ra từ form (form submission) nh-ng nó không đ-ọc hiển thị trên trang. Mọi người có thể sử dụng trong JavaScript để l-u các giá trị trong suốt một script và để tính toán không cần form.

Đối tượng hidden chỉ có hai thuộc tính là **name** và **value**, đó cũng là những giá trị xâu giống các đối tượng khác. Không có cách thức hay thẻ sự kiện nào cho đối tượng này.

4.1.25. Phần tử Password

Đối tượng Password là đối tượng duy nhất trong các đối tượng của form mà khi gõ bất kỳ ký tự nào vào cũng đều hiển thị dấu sao(*). Nó cho phép đ-a vào những thông tin bí mật nh- đăng ký mật khẩu...

Đối tượng Password có 3 thuộc tính giống trường text là: defaultValue, name và value. Không giống với hai phần tử ở trên, trường Password có nhiều cách thức hơn(focus(), blur(), and select()) và tương ứng với các thẻ sự kiện: onFocus, onBlur, and onSelect.

Phần này sẽ được nói kỹ hơn trong đối tượng text.

4.1.26. Phần tử radio

Đối tượng radio gần giống sự bật tắt checkbox khi có hai nút radio kết hợp thành một nhóm. Khi nhiều radio được kết hợp thành một nhóm, chỉ có một nút được chọn trong bất kỳ một thời điểm nào. Ví dụ dòng lệnh sau tạo ra một nhóm radio có ba nút tên là test:

```
<INPUT TYPE="radio" NAME="test" VALUE="1" checked>1<BR>
<INPUT TYPE="radio" NAME="test" VALUE="2">2<BR>
<INPUT TYPE="radio" NAME="test" VALUE="3">3<BR>
```

Nhóm các nút radio lại bằng cách đặt cho chúng có cùng một tên trong các thẻ INPUT.

Có một vài thuộc tính để kiểm tra trạng thái hiện thời của một nhóm nút radio. Bảng sau hiển thị các thuộc tính và cách thức của đối tượng radio.

Bảng? . Các thuộc tính và cách thức của đối tượng radio.

Thuộc tính và cách thức	Mô tả
checked	Mô tả trạng thái hiện thời của phần tử radio (thuộc tính)
defaultChecked	Mô tả trạng thái mặc định của phần tử (thuộc tính)
index	Mô tả thứ tự của nút radio được chọn hiện thời trong một nhóm
length	Mô tả tổng số nút radio trong một nhóm
name	Mô tả tên của phần tử được chỉ định trong thẻ INPUT (thuộc tính)
value	Mô tả giá trị hiện thời của phần tử được định ra trong thẻ INPUT (thuộc tính)
click()	Mô phỏng một click trên nút radio (cách thức)

Cũng như checkbox, radio chỉ có một thẻ sự kiện là onClick.

Không có bất kỳ một đối tượng form nào có thuộc tính index và length. Do một nhóm radio gồm nhiều phần tử radio, nên chúng được đặt trong một mảng các nút radio và được đánh số từ 0. Trong ví dụ nhóm radio có tên test ở trên, nếu nhóm đó nằm trong một form có tên là "testform", bạn có thể gọi tới nút radio thứ hai bằng tên "testform.test[1]" và có thể kiểm tra giá trị của nó bằng "testform.test[1].checked"

Để minh họa rõ cách dùng đối tượng radio, ta xem ví dụ sau:

Ví dụ:

```
<HTML>
<HEAD>
<TITLE>radio button Example</TITLE>
<SCRIPT>
<!-- HIDE FROM OTHER BROWSERS
function calculate(form,callingField) {
```

```

        if (callingField == "result") {
            if (form.action[1].checked) {
                form.entry.value = Math.sqrt(form.result.value);
            } else {
                form.entry.value = form.result.value / 2;
            }
        } else {
            if (form.action[1].checked) {
                form.result.value=form.entry.value*form.entry.value;
            } else {
                form.result.value = form.entry.value * 2;
            }
        }
    }
}

// STOP HIDING FROM OTHER BROWSERS UU>
</SCRIPT>
</HEAD>
<BODY>
<FORM METHOD=POST>
Value: <INPUT TYPE="text" NAME="entry" VALUE=0
           onChange="calculate(this.form,this.name);">
<BR>
Action:<BR>
<INPUT TYPE="radio" NAME="action" VALUE="twice"
      onClick="calculate(this.form,this.name);"> Double<BR>
<INPUT TYPE="radio" NAME="action" VALUE="square"
      onClick="calculate(this.form,this.name);"> Square <BR>
Result: <INPUT TYPE=text NAME="result" VALUE=0
        onChange="calculate(this.form,this.name);">

</FORM>
</BODY>
</HTML>

```

Trong ví dụ này, sự thay đổi từ checkbox ở trên là rất khó nhận biết. Thay cho một checkbox trong ví dụ trước, ở đây ta sử dụng hai nút radio với hai giá trị khác nhau: double và square

Nh- ta đã biết có thể truy nhập đến các nút radio qua một mảng, do đó hai nút này có thể truy nhập bằng **action[0]** và **action[1]**. Bằng cách này, bạn chỉ cần thay đổi tham chiếu đến hàm *calculate()* từ **form.square.checked** thành **form.action[1].checked**.

4.1.27. Phần tử reset

Sử dụng đối tượng reset, bạn có thể tác động ngược lại để click vào nút Reset. Cũng giống đối tượng button, đối tượng reset có hai thuộc tính là name và value, và một cách thức click(), một thẻ sự kiện onClick.

Hầu hết những người lập trình không sử dụng thẻ sự kiện onClick của nút reset để kiểm tra giá trị của nút này, đối tượng reset thường dùng để xóa form.

Ví dụ sau minh họa cách sử dụng nút reset để xóa các giá trị của form.

Ví dụ:

```
<HTML>
<HEAD>
<TITLE>reset Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!-- HIDE FROM OTHER BROWSERS
function clearForm(form) {
form.value1.value = "Form";
    form.value2.value = "Cleared";
}
// STOP HIDING FROM OTHER BROWSERS -->
//SCRIPT>
</HEAD>
<BODY>
<FORM METHOD=POST>
<INPUT TYPE="text" NAME="value1"><BR>
<INPUT TYPE="text" NAME="value2"><BR>
<INPUT TYPE="reset" VALUE="Clear Form"
    onClick="clearForm(this.form);">
</FORM>
</BODY>
</HTML>
```

4.1.28. Phần tử select

Danh sách lựa chọn trong các form HTML xuất hiện menu dropdown hoặc danh sách cuộn dọc của các đối tượng có thể được lựa chọn. Các danh sách dọc xây dựng bằng cách sử dụng hai thẻ SELECT và OPTION. Ví dụ:

```
<SELECT NAME="test">
    <OPTION SELECTED>1
    <OPTION>2
    <OPTION>3
</SELECT>
```

tạo ra ba thành phần của menu thả dropdown với ba lựa chọn 1,2 và 3. Sử dụng thuộc tính SIZE bạn có thể tạo ra một danh sách cuộn với số phần tử hiển thị ở lần thứ nhất. Để bật

menu dropdown trong một menu cuộn với hai thành phần hiển thị, bạn có thể sử dụng như sau:

```
<SELECT NAME="test" SIZE=2>
  <OPTION SELECTED>1
  <OPTION>2
  <OPTION>3
</SELECT>
```

Trong cả hai ví dụ trên, người sử dụng chỉ có thể có một lựa chọn. Nếu sử dụng thuộc tính **MULTIPLE**, bạn có thể cho phép người sử dụng lựa chọn nhiều hơn một giá trị trong danh sách lựa chọn:

```
<SELECT NAME="test" SIZE=2 MULTIPLE>
<OPTION SELECTED>1
<OPTION>2
<OPTION>3
</SELECT>
```

Danh sách lựa chọn trong JavaScript là đối tượng **select**. Đối tượng này tạo ra một vài thành phần tương tự các button và radio.

Với các thành phần lựa chọn, danh sách các lựa chọn được chứa trong một mảng được đánh số từ 0. Trong trường hợp này, mảng là một thuộc tính của đối tượng **select** gọi là **options**.

Cả việc lựa chọn các option và từng phần tử option riêng biệt đều có những thuộc tính. Bổ sung thêm vào mảng option, phần tử select có thuộc tính **selectedIndex**, có chứa số thứ tự của option được lựa chọn hiện thời.

Mỗi option trong danh sách lựa chọn đều có một vài thuộc tính:

- **DEFAULTSELECTED**: cho biết option có được mặc định là lựa chọn trong thẻ **OPTION** hay không.
- **INDEX**: chứa giá trị số thứ tự của option hiện thời trong mảng option.
- **SELECTED**: cho biết trạng thái hiện thời của option
- **TEXT**: có chứa giá trị của dòng text hiển thị trên menu cho mỗi option, và thuộc tính value mọi giá trị chỉ ra trong thẻ **OPTION**.

Đối tượng **select** không có các cách thức được định nghĩa sẵn. Tuy nhiên, đối tượng **select** có ba thẻ sự kiện, đó là **onBlue**, **onFocus**, **onChange**, chúng đều là những đối tượng text.

Ví dụ bạn có danh sách lựa chọn sau:

```
<SELECT NAME="example" onFocus="react();">
  <OPTION SELECTED VALUE="Number One">1
  <OPTION VALUE="The Second">2
  <OPTION VALUE="Three is It">3
</SELECT>
```

Khi lần đầu tiên hiển thị bạn có thể truy nhập tới các thông tin sau:

```
example.options[1].value = "The Second"
```

```
example.options[2].text = "3"
example.selectedIndex = 0
example.options[0].defaultSelected = true
example.options[1].selected = false
```

Nếu người sử dụng kích vào menu và lựa chọn option thứ hai, thì thẻ onFocus sẽ thực hiện, và khi đó giá trị của thuộc tính sẽ là:

```
example.options[1].value = "The Second"
example.options[2].text = "3"
example.selectedIndex = 1
example.options[0].defaultSelected = true
example.options[1].selected = true
```

Sửa các danh sách lựa chọn

Navigator 3.0 cho phép thay đổi nội dung của danh sách lựa chọn từ JavaScript bằng cách liên kết các giá trị mới cho thuộc tính text của các thực thể trong danh sách.

Ví dụ, trong ví dụ trước, bạn đã tạo ra một danh sách lựa chọn như sau:

```
<SELECT NAME="example" onFocus="react();">
  <OPTION SELECTED VALUE="Number One">1
  <OPTION VALUE="The Second">2
  <OPTION VALUE="Three is It">3
</SELECT>
```

Có thể thay đổi dòng text hiển thị trên nút thứ hai thành "two" bằng:

```
example.options[1].text = "two";
```

Có thể thêm các lựa chọn mới vào danh sách bằng cách sử dụng đối tượng xây dựng Option() theo cú pháp:

```
newOptionName = new Option(optionText, optionValue, defaultSelected,
selected);
```

```
selectListName.options[index] = newOptionName;
```

Việc tạo đối tượng option() này với dòng text được chỉ trước, defaultSelected và selected như trên đã định ra những giá trị kiểu Boolean. Đối tượng này được liên kết vào danh sách lựa chọn được thực hiện bằng **index**.

Các lựa chọn có thể bị xóa trong danh sách lựa chọn bằng cách gán giá trị **null** cho đối tượng muốn xóa

```
selectListName.options[index] = null;
```

1.1 Phần tử submit

Nút Submit là một trường hợp đặc biệt của button, cũng như nút Reset. Nút này đưa thông tin hiện tại từ các trường của form tới địa chỉ URL được chỉ ra trong thuộc tính ACTION của thẻ form sử dụng cách thức METHOD chỉ ra trong thẻ FORM.

Giống như đối tượng button và reset, đối tượng submit có sẵn thuộc tính name và value, cách thức click() và thẻ sự kiện onClick.

1.2 Phần tử Text

Phần tử này nằm trong những phần tử hay được sử dụng nhất trong các form HTML. Tương tự như trường Password, trường text cho phép nhập vào một dòng đơn, như các ký tự của nó hiện ra bình thường.

Đối tượng text có ba thuộc tính: defaultValue, name và value. Ba cách thức mô phỏng sự kiện của người sử dụng: focus(), blur() và select(). Có 4 thẻ sự kiện là: onBlur, onFocus, onChange, onSelect. Chú ý các sự kiện này chỉ thực hiện khi con trỏ đã được kích ra ngoài trường text.

Bảng sau mô tả các thuộc tính và cách thức của đối tượng text.

Bảng .Các thuộc tính và cách thức của đối tượng text.

Cách thức và thuộc tính	Mô tả
defaultValue	Chỉ ra giá trị mặc định của phần tử được chỉ ra trong thẻ INPUT (thuộc tính)
name	Tên của đối tượng được chỉ ra trong thẻ INPUT (thuộc tính)
value	Giá trị hiện thời của phần tử (thuộc tính)
focus()	Mô tả việc con trỏ tới trường text (cách thức)
blur()	Mô tả việc con trỏ rời trường text (cách thức)
select()	Mô tả việc lựa chọn dòng text trong trường text (cách thức)

Một chú ý quan trọng là có thể gán giá trị cho trường text bằng cách liên kết các giá trị với thuộc tính value. Trong ví dụ sau đây, dòng text được đưa vào trường đầu tiên được lặp lại trong trường text thứ hai, và mọi dòng text được đưa vào trường text thứ hai lại được lặp lại trong trường text thứ nhất. Khả năng này của nó có thể áp dụng để tự động cập nhật hoặc thay đổi dữ liệu.

Ví dụ. Tự động cập nhật các trường text .

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>text Example</TITLE>
```

```
<SCRIPT LANGUAGE="JavaScript">
```

```

<!U' HIDE FROM OTHER BROWSERS
function echo(form,currentField) {
    if (currentField == "first")
        form.second.value = form.first.value;
    else
        form.first.value = form.second.value;
}
// STOP HIDING FROM OTHER BROWSERS U'>
</SCRIPT>
</HEAD>
<BODY>
<FORM>
    <INPUT                TYPE=text                NAME="first"
onChange="echo(this.form,this.name);">
    <INPUT                TYPE=text                NAME="second"
onChange="echo(this.form,this.name);">
</FORM>
</BODY>
</HTML>

```

1.3 Phần tử **Textarea**

Thẻ TEXTAREA cung cấp một hộp cho phép nhập số dòng text do người thiết kế định trước. Ví dụ:

```

<TEXTAREA NAME="fieldName" ROWS=10 COLS=25>
    Default Text Here
</TEXTAREA>

```

Ví dụ này tạo ra một trường text cho phép đưa vào 10 hàng, mỗi hàng 25 ký tự. Dòng "Default Text Here" sẽ xuất hiện trong trường này vào lần hiển thị đầu tiên.

Cũng như phần tử text, JavaScript cung cấp cho bạn các thuộc tính defaultValue, name, và value, các cách thức focus(), select(), và blur(), các thẻ sự kiện onBlur, onFocus, onChange, onSelect.

2. Mảng **elements[]**

Các đối tượng của form có thể được gọi tới bằng mảng elements[]. Ví dụ bạn tạo ra một form sau:

```

<FORM METHOD=POST NAME=testform>

```

```

<INPUT TYPE="text" NAME="one">
<INPUT TYPE="text" NAME="two">
<INPUT TYPE="text" NAME="three">
</FORM>

```

bạn có thể gọi tới ba thành phần này nh- sau: `document.elements[0]`, `document.elements[1]`, `document.elements[2]`, hơn nữa còn có thể gọi `document.testform.one`, `document.testform.two`, `document.testform.three`.

Thuộc tính này th-ờng đ-ọc sử dụng trong các mối quan hệ tuần tự của các phần tử hơn là dùng tên của chúng.

3. Mảng `form[]`

Các thẻ sự kiện đ-ọc thiết kế để làm việc với các form riêng biệt hoặc các tr-ờng ở một thời điểm, nó rất hữu dụng để cho phép gọi tới các form có liên quan trong cùng một trang.

Mảng `form[]` đề cập đến ở đây có thể có nhiều xác định các nhân của form trên cùng một trang và **have information in a single field match in all three forms**. Có thể gọi bằng `document.forms[]` thay vì gọi bằng tên form. Trong script này, bạn có hai tr-ờng text để nhập và nằm trên hai form độc lập với nhau. Sử dụng mảng form bạn có thể t-ơng tác trên các giá trị của các tr-ờng trong hai form cùng một lúc khi ng-ời sử dụng thay đổi giá trị trên một form.

```

<HTML>
<HEAD>
<TITLE>forms[] Example</TITLE>
</HEAD>
<BODY>
<FORM METHOD=POST>
  <INPUT                                     TYPE=text
onChange="document.forms[1].elements[0].value = this.value;">
</FORM>
<FORM METHOD=POST>
  <INPUT                                     TYPE=text
onChange="document.forms[0].elements[0].value = this.value;">
</FORM>
</BODY>
</HTML>

```


Mặt khác, bạn cũng có thể truy nhập đến form bằng tên form đ-ợc đặt trong thẻ FORM:

<FORM METHOD=POST NAME="name">

Khi đó bạn có thể gọi là document.forms["name"] hoặc document.name

4. Xem lại các lệnh và mở rộng

Lệnh/ Mở rộng	Kiểu	Mô tả
blur()	cách thức JavaScript	Mô tả việc dịch chuyển con trỏ từ một phần tử
form.action	cách thức JavaScript	Xâu chứa giá trị của thuộc tính ACTION trong thẻ FORM
form.elements	thuộc tính JavaScript	mảng chứa danh sách các phần tử trong form (nh- checkbox, tr-ờng text, danh sách lựa chọn)
form.encoding	thuộc tính JavaScript	xâu chứa kiểu MIME sử dụng khi chuyển thông tin từ form tới server
form.name	thuộc tính JavaScript	Xâu chứa giá trị thuộc tính NAME trong thẻ FORM
form.target	thuộc tính JavaScript	Xâu chứa tên cửa sổ đích bởi một form submission
form.submit	cách thức JavaScript	Mô tả việc submit một form HTML
type	thuộc tính JavaScript	ánh xạ kiểu của một phần tử form thành một xâu.
onSubmit	Thẻ sự kiện	thẻ sự kiện cho việc submit
button	thuộc tính HTML	Thuộc tính kiểu cho các nút bấm của HTML (<INPUT TYPE=button>)
checkbox	thuộc tính HTML	Thuộc tính kiểu cho các checkbox của HTML (<INPUT TYPE=checkbox>)
password	thuộc tính HTML	Thuộc tính kiểu cho các dòng password của HTML(<INPUT TYPE=password>)
radio	thuộc tính HTML	Thuộc tính kiểu cho các nút radio của HTML (<INPUT TYPE=radio>)
reset	thuộc tính HTML	Thuộc tính kiểu cho các nút reset của HTML (<INPUT TYPE=reset>)
SELECT	thẻ HTML	Hộp thẻ cho danh sách lựa chọn

OPTION	thẻ HTML	chỉ ra các lựa chọn trong danh sách lựa chọn(<SELECT><OPTION>Option 1<OPTION>Option 2</SELECT>)
submit	thuộc tính HTML	Thuộc tính kiểu của nút submit (<INPUT TYPE=submit>)
text	thuộc tính HTML	Thuộc tính kiểu của trường trong form (<INPUT TYPE=text>)
TEXTAREA	Thẻ HTML	Hộp thẻ cho nhiều dòng text (<TEXTAREA> default text </TEXTAREA>)
name	thuộc tính JavaScript	Xâu chứa tên phần tử HTML (button, checkbox, password...)
value	thuộc tính JavaScript	Xâu chứa giá trị hiển thị của một phần tử HTML(button, checkbox, password...)
click()	cách thức JavaScript	Mô tả việc kích vào một phần tử trên form (button, checkbox,password)
onClick	thuộc tính JavaScript	Thẻ sự kiện cho sự kiện kích (button, checkbox, radio button, reset, selection list, submit)
checked	thuộc tính JavaScript	Giá trị kiểu Boolean mô tả một lựa chọn check(checkbox, radio button)
defaultChecked	thuộc tính JavaScript	Xâu chứa giá trị mặc định của một phần tử HTML (password, text, textarea)
focus()	cách thức JavaScript	Mô tả việc con trỏ tới một phần tử (password, text, textarea)
blur()	cách thức JavaScript	Mô tả việc con trỏ rời khỏi một phần tử (password, text, textarea)
select()	cách thức JavaScript	Mô tả việc lựa chọn dòng text trong một trường (password, text, textarea)
onFocus()	Thẻ sự kiện	Thẻ sự kiện cho sự kiện focus (password, selection list, text, textarea)
onBlur	Thẻ sự kiện	Thẻ sự kiện cho sự kiện blur (password, selection list, text, textarea)

onChange	Thẻ sự kiện	Thẻ sự kiện cho sự kiện khi giá trị của trường thay đổi (password, selection list, text, textarea)
onSelect	Thẻ sự kiện	Thẻ sự kiện khi người sử dụng chọn dòng text trong một trường (password, text, textarea)
index	thuộc tính JavaScript	Là một số nguyên mô tả lựa chọn hiện thời trong một nhóm lựa chọn (radio button)
length	thuộc tính JavaScript	Số nguyên mô tả tổng số các lựa chọn trong một nhóm các lựa chọn (radio button)
defaultSelected	thuộc tính JavaScript	Giá trị Boolean mô tả khi có một lựa chọn được đặt là mặc định (selection list)
options	thuộc tính JavaScript	Mảng các lựa chọn trong danh sách lựa chọn
text	thuộc tính JavaScript	Dòng text hiển thị cho một thành phần của menu trong danh sách lựa chọn
TABLE	thẻ HTML	Hộp thẻ cho các bảng HTML
TR	thẻ HTML	Hộp thẻ cho các hàng của một bảng HTML
TD	thẻ HTML	Hộp thẻ cho các ô của một hàng trong một bảng HTML
COLSPAN	thuộc tính HTML	Là thuộc tính của thẻ TD mô tả trong một ô của bảng có nhiều cột
ROWSPAN	thuộc tính HTML	Là thuộc tính của thẻ TD mô tả trong một ô của bảng có nhiều hàng
BORDER	thuộc tính HTML	Là thuộc tính của thẻ TABLE mô tả độ rộng đường viền của bảng
document.forms[]	thuộc tính JavaScript	mảng của các đối tượng form với một danh sách các form trong một document
string.substring()	cách thức JavaScript	Trả lại một xâu con của xâu string từ tham số vị trí ký tự đầu đến vị trí ký tự cuối
Math.floor()	cách thức JavaScript	Trả lại một giá trị nguyên tiếp theo

		nhỏ hơn giá trị của tham số đưa vào.
string.length	thuộc tính JavaScript	Giá trị nguyên của số thứ tự ký tự cuối cùng trong chuỗi string

5. MÔ HÌNH ĐỐI TƯỢNG (OBJECT MODEL)

ĐỐI TƯỢNG VỚI THUỘC TÍNH

Như đã biết, một đối tượng trong JavaScript có các thuộc tính đi kèm với nó. Bạn có thể truy nhập đến các thuộc tính của nó bằng cách gọi :

```
objectName.propertyName
```

Cả tên đối tượng và tên thuộc tính đều nhạy cảm. Bạn định nghĩa một thuộc tính bằng cách gán cho nó một giá trị. Ví dụ, giả sử có một đối tượng tên là `myCar` (trong trường hợp này giả sử đối tượng này đã tồn tại sẵn sàng). Bạn có thể lấy các thuộc tính có tên `make`, `model` và `year` của nó như sau:

```
myCar.make = "Ford"
myCar.model = "Mustang"
myCar.year = 69;
```

Có một mảng l-u trữ tập hợp các giá trị tham chiếu tới từng biến. Thuộc tính và mảng trong JavaScript có quan hệ mật thiết với nhau, thực ra chúng chỉ khác nhau về cách giao tiếp với cùng một cấu trúc dữ liệu. Ví dụ cũng có thể truy nhập tới các thuộc tính của đối tượng `myCar` ở trên bằng mảng như sau:

```
myCar[make] = "Ford"
myCar[model] = "Mustang"
myCar[year] = 69;
```

Kiểu mảng này được hiểu như một mảng có khả năng liên kết bởi mỗi một phần tử trong đó đều có thể liên kết đến một giá trị xâu nào đó. Để minh họa việc này được thực hiện như thế nào, hàm sau đây sẽ hiển thị các thuộc tính của một đối tượng thông qua tham số về kiểu đối tượng đó và tên đối tượng.

```
function show_props (obj, obj_name)
{
    var result=""
    for (i in obj)
        result=result+ obj_name + "." + i + "=" +
            obj[i] + "\n"
    return result
}
```

Khi gọi hàm `show_props(myCar,"myCar")` sẽ hiện lên:

```
myCar.make = Ford
myCar.model = Mustang
myCar.year = 69;
```

TẠO CÁC ĐỐI TƯỢNG MỚI

Cả JavaScript clientUside và serverUside đều có một số đối tượng được định nghĩa trước. Tuy nhiên, bạn cũng có thể tạo ra những đối tượng của riêng bạn. Trong JavaScript 1.2, nếu bạn chỉ muốn tạo ra một đối tượng duy nhất của một kiểu đối tượng, bạn có thể tạo nó bằng cách sử dụng khởi tạo đối tượng. Hoặc nếu bạn muốn tạo ra nhiều cá thể của một kiểu đối tượng, bạn có thể tạo ra một hàm xây dựng trước, sau đó tạo ra các đối tượng có kiểu của hàm đó bằng toán tử **new**

5.1.1. Sử dụng khởi tạo đối tượng

Trong những phiên bản trước của Navigator, bạn chỉ có thể tạo ra một đối tượng bằng cách sử dụng hàm xây dựng chúng hoặc sử dụng một hàm được cung cấp bởi một vài đối tượng khác để đạt được mục đích.

Tuy nhiên, trong Navigator 4.0, bạn có thể tạo ra một đối tượng bằng cách sử dụng một khởi tạo đối tượng. Bạn sử dụng cách này khi bạn chỉ muốn tạo ra một cá thể đơn lẻ chứ không phải nhiều cá thể của đối tượng.

Cú pháp để tạo ra một đối tượng bằng cách khởi tạo đối tượng (Object Initializers):

```
objectName={property1: value1, property2: value2,
            ..., propertyN: valueN}
```

Trong đó **objectName** là tên của đối tượng mới, mỗi **propertyI** là một xác minh (có thể là một tên, một số hoặc một chuỗi ký tự) và mỗi **valueI** là một biểu thức mà giá trị của nó được gán cho **propertyI**. Có thể lựa chọn khởi tạo bằng tên đối tượng hoặc chỉ bằng các khai báo. Nếu nh- bạn không cần dùng đến đối tượng đó trong mọi chỗ, bạn không cần phải gán nó cho một biến.

Nếu một đối tượng được tạo bằng cách khởi tạo đối tượng ở mức cao nhất, mỗi lần đối tượng đó xuất hiện trong các biểu thức, JavaScript sẽ đánh giá lại nó một lần. Ngoài ra, nếu sử dụng việc khởi tạo này trong một hàm thì mỗi lần gọi hàm, đối tượng sẽ được khởi tạo một lần

Giả sử bạn có câu lệnh sau:

```
if (condition)
    x={hi: "there."}
```

Trong trường hợp này, JavaScript sẽ tạo ra một đối tượng và gán nó vào biến x nếu biểu thức **condition** được đánh giá là đúng

Còn ví dụ sau tạo ra một đối tượng myHonda với 3 thuộc tính:

```
myHonda={color:"red", wheels:4, engine:{cylinder:4, size:2.2}}
```

Chú ý rằng thuộc tính **engine** cũng là một đối tượng với các thuộc tính của nó

Trong Navigator 4.0, bạn cũng có thể sử dụng một khởi tạo để tạo một mảng. Cú pháp để tạo mảng bằng cách này khác với tạo đối tượng:

```
arrayName=[element0, element1, ..., elementN]
```

Trong đó, **arrayName** là tên của mảng mới, và mỗi **elementI** là giá trị của phần tử ở vị trí đó của mảng. Khi bạn tạo một mảng bằng cách sử dụng ph-ơng pháp khởi tạo, thì nó sẽ coi mỗi giá trị là một phần tử trên mảng, và chiều dài của mảng chính là số các tham số.

Bạn không cần phải chỉ định rõ tất cả các phần tử trên mảng mới. Nếu bạn đặt hai dấu phẩy vào hàng, thì mảng sẽ được tạo với những chỗ trống cho những phần tử chưa được định nghĩa như ví dụ dưới đây:

Nếu một mảng được tạo bằng cách khởi tạo (initializer) ở mức cao nhất, mỗi lần mảng đó xuất hiện trong các biểu thức, JavaScript sẽ đánh giá lại nó một lần. Ngoài ra, nếu sử dụng việc khởi tạo này trong một hàm thì mỗi lần gọi hàm, mảng sẽ được khởi tạo một lần

Ví dụ 1: Tạo một mảng coffees với 3 phần tử và độ dài của mảng là 3:

```
coffees = ["French Roast", "Columbian", "Kona"]
```

Ví dụ 2: Tạo ra một mảng với 2 phần tử được khởi đầu và một phần tử rỗng:

```
fish = ["Lion", , " Surgeon"]
```

Với biểu thức này, `fish[0]` là "Lion", `fish[2]` là " Surgeon", và `fish[2]` chưa được định nghĩa

5.1.2. Sử dụng một hàm xây dựng (Constructor Function)

Bạn có thể tạo ra đối tượng của riêng mình với hai bước sau:

1. Định nghĩa kiểu của đối tượng bằng cách viết một hàm xây dựng.
2. Tạo ra một cá thể của đối tượng đó bằng toán tử **new**

Để định nghĩa một kiểu đối tượng, ta phải tạo ra một hàm để chỉ định rõ tên, các thuộc tính và các cách thức của kiểu đối tượng đó. Ví dụ giả sử bạn muốn tạo một kiểu đối tượng ô tô với tên là **car**, có các thuộc tính **make**, **model**, **year** và **color**, để thực hiện việc này có thể viết một hàm như sau:

```
function car(make, model, year ){
    this.make = make
    this.model = model
    this.year = year
}
```

Chú ý việc sử dụng toán tử **this** để gán giá trị cho các thuộc tính của đối tượng phải thông qua các tham số của hàm.

Ví dụ, bạn có thể tạo một đối tượng mới kiểu **car** như sau:

```
mycar = new car("Eagle", "Talon TSi", 1993)
```

Câu lệnh này sẽ tạo ra đối tượng mycar và liên kết các giá trị được đưa vào với các thuộc tính. Khi đó giá trị của **mycar.make** là "Eagle", giá trị của **mycar.model** là "Talon TSi", và **mycar.year** là một số nguyên 1993....Cứ như vậy bạn có thể tạo ra nhiều đối tượng kiểu **car**.

Một đối tượng cũng có thể có những thuộc tính mà bản thân nó cũng là một đối tượng. Ví dụ bạn định nghĩa thêm một đối tượng khác là **person** như sau:

```
function person(name, age, sex){
    this.name=name
    this.age=age
    this.sex=sex
}
```

Và sau đó ta tạo ra hai ng-ời mới:

```
rank = new person("Rank McKinnon", 33, "M")
ken = new person("Ken John", 39, "M")
```

Bây giờ bạn định nghĩa lại hàm xây dựng **car** nh- sau:

```
function car(make, model, year, owner) {
    this.make = make
    this.model = model
    this.year = year
    this.owner = owner
}
```

Nh- vậy bạn có thể tạo đối t-ợng kiểu **car** mới:

```
car1 = new car("Eagle", "Talon TSi", 1993, rank)
car2 = new car("Nissan", "300ZX", 1992, ken)
```

Nh- vậy, thay vì phải qua một xâu ký tự hay một giá trị số khi tạo đối t-ợng, ta chỉ cần đ-ưa hai đối t-ợng đã đ-ợc tạo ở câu lệnh trên vào dòng tham số của đối t-ợng mới tạo. Ta cũng có thể lấy đ-ợc thuộc tính của đối t-ợng owner bằng câu lệnh sau:

```
car2.owner.name
```

Chú ý rằng bạn cũng có thể tạo ra một thuộc tính mới cho đối t-ợng tr-ớc khi định nghĩa nó, ví dụ:

```
car1.color="black"
```

Nh- vậy, thuộc tính **color** của đối t-ợng **car1** đ-ợc gán là "**black**". Tuy nhiên, nó sẽ không gây tác động tới bất kỳ một đối t-ợng kiểu **car** nào khác. Nếu muốn thêm thuộc tính cho tất cả các đối t-ợng thì phải định nghĩa lại hàm xây dựng đối t-ợng.

5.1.3. Lập mục lục cho các thuộc tính của đối t-ợng

Trong Navigator 2.0, bạn có thể gọi thuộc tính của một đối t-ợng bằng tên thuộc tính hoặc bằng số thứ tự của nó. Tuy nhiên từ Navigator 3.0 trở đi, nếu ban đầu bạn định nghĩa một thuộc tính bằng tên của nó, bạn sẽ luôn luôn phải gọi nó bằng tên, và nếu bạn

định nghĩa một thuộc tính bằng chỉ số thì bạn cũng luôn luôn phải gọi tới nó bằng chỉ số.

Điều này ứng dụng khi bạn tạo một đối t-ợng với những thuộc tính của chúng bằng hàm xây dựng (nh- ví dụ về kiểu đối t-ợng **car** ở phần tr-ớc) và khi bạn định nghĩa những thuộc tính của riêng một đối t-ợng (nh- `mycar.color="red"`). Vì vậy nếu bạn định nghĩa các thuộc tính của đối t-ợng ngay từ đầu bằng chỉ số nh- `mycar[5]="25 mpg"`, bạn có thể lần l-ợt gọi tới các thuộc tính khác nh- `mycar[5]`.

Tuy nhiên điều này là không đúng đối với những đối t-ợng t-ơng ứng của HTML nh- mảng form. Bạn có thể gọi tới các đối t-ợng trong mảng bởi số thứ tự hoặc tên của chúng. Ví dụ thẻ `<FORM>` thứ hai trong một document có thuộc tính NAME là "myform" thì bạn có thể gọi tới form đó bằng `document.form[1]` hoặc `document.form["myForm"]` hoặc `document.myForm`

5.1.4. Định nghĩa thêm các thuộc tính cho một kiểu đối tượng

Bạn có thể thêm thuộc tính cho một kiểu đối tượng đã được định nghĩa trước bằng cách sử dụng thuộc tính `property`. Thuộc tính được định nghĩa này không chỉ có tác dụng đối với một đối tượng mà có tác dụng đối với tất cả các đối tượng khác cùng kiểu. Ví dụ sau thực hiện thêm thuộc tính **color** cho tất cả các đối tượng kiểu **car**, sau đó gán một giá trị màu cho thuộc tính **color** của đối tượng **car1**:

```
car.prototype.color=null
car1.color="red"
```

5.1.5. Định nghĩa các cách thức

Một cách thức là một hàm được liên kết với một đối tượng. Bạn định nghĩa một cách thức cũng có nghĩa là bạn định nghĩa một hàm chuẩn. Bạn có thể sử dụng cú pháp sau để gán một hàm cho một đối tượng đang tồn tại:

```
object.methodname = function_name
```

Trong đó **object** là đối tượng đang tồn tại, **methodname** là tên cách thức và **function_name** là tên hàm

Bạn có thể gọi cách thức này từ đối tượng như sau:

```
object.methodname(<tham số>)
```

Bạn có thể định nghĩa cách thức cho một kiểu đối tượng bằng cách đưa cách thức đó vào trong hàm xây dựng đối tượng. Ví dụ bạn có thể định nghĩa một hàm có thể định dạng và hiển thị các thuộc tính của các đối tượng kiểu **car** đã xây dựng ở phần trước:

```
function displayCar () {
    var result = "Abeautiful"+this.year+ " " + this.make +
                " " + this.model
    document.write(result)
}
```

Bạn có thể thêm cách thức này vào cho đối tượng **car** bằng cách thêm dòng lệnh sau vào hàm định nghĩa đối tượng

```
this.displayCar= displayCar;
```

Như vậy có thể định nghĩa lại đối tượng **car** như sau:

```
function car(make, model, year,owner ){
    this.make = make
    this.model = model
    this.year = year
    this.owner = owner
    this.displayCar= displayCar
}
```

Sau đó, bạn có thể gọi cách thức `displayCar` đối với mỗi đối tượng:

```
car1.displayCar()
car2.displayCar()
```

5.1.6. Sử dụng cho các tham chiếu đối tượng (Object References)

JavaScript có một từ khoá đặc biệt là **this** mà bạn có thể sử dụng nó cùng với một cách thức để gọi tới đối tượng hiện thời. Ví dụ, giả sử bạn có một hàm **validate** dùng để xác nhận giá trị thuộc tính của một đối tượng nằm trong một khoảng nào đó:

```
function validate(obj, lowval, hival){
    if ( (obj.value<lowval)|| (obj.value>hival) )
        alert("Invalid value!")
}
```

Sau đó bạn có thể gọi hàm **validate** từ mỗi thẻ sự kiện **onChange**:

```
<INPUT TYPE="TEXT" NAME="AGE" SIZE=3
      onChange="validate(this,18,99)" >
```

Khi liên kết với một thuộc tính form, từ khoá **this** có thể gọi tới form cha của đối tượng hiện thời. Trong ví dụ sau, **myForm** có chứa đối tượng **Text** và một nút bấm. Khi người sử dụng kích vào nút bấm, trường text sẽ hiển thị tên form. Thẻ sự kiện **onClick** của nút bấm sử dụng **this.form** để gọi tới form cha là **myForm**.

```
<FORM NAME="myForm">
    Form name:<INPUT TYPE="text" NAME="text1"
                VALUE="Beluga">
<P>
    <INPUT TYPE="button" NAME="button1"
          value="Show Form Name"
          onClick="this.form.text1.value=this.form.name">
</FORM>
```

5.1.7. Xóa đối tượng

Trong JavaScript cho Navigator 2.0, bạn không thể xóa các đối tượng. Chúng vẫn tồn tại trong khi bạn đã rời khỏi trang đó. Trong khi JavaScript cho Navigator 3.0 cho phép bạn có thể xóa một đối tượng bằng cách đặt cho nó trở về giá trị Null (nếu như đó là

lần cuối cùng gọi tới đối tượng). JavaScript sẽ đóng đối tượng đó ngay lập tức thông qua biểu thức gán.

6. BẢNG TỔNG KẾT CÁC TỪ KHOÁ

Sau đây là các từ được định nghĩa là một phần trong ngôn ngữ JavaScript và không được sử dụng là tên biến:

abstract	eval	int	static
boolean	extends	interface	super
break	false	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	null	throw
char	for	package	throws
class	function	parseFloat	transient
const	goto	parseInt	true
continue	if	private	try
default	implements	protected	var
do	import	public	void
double	in	return	while
else	instanceof	short	with

7. TỔNG KẾT

Như vậy, tài liệu không những đã giới thiệu sơ qua về JavaScript, mà nó còn là sách tham khảo hết sức hữu ích để phát triển ứng dụng của bạn.

Bạn có thể tham khảo toàn diện JavaScript trong quyển *Teach Yourself JavaScript in 14 Days*, hoặc *JavaScript Guide*

Do JavaScript là ngôn ngữ còn mới và có sự thay đổi nhanh chóng, bạn nên đến với trang Web của hãng Netscape (<http://www.netscape.com>) để có các thông tin mới nhất về ngôn ngữ này.