BÀI 5 – CƠ BẢN VỀ JAVASCRIPT

5.1. Nhập môn JavaScript

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

- Sử dụng các câu lệnh và các hàm trong cặp thẻ **<SCRIPT>.**
- Sử dụng các file nguồn JavaScript.
- Sử dụng một biểu thức JavaScript làm giá trị của một thuộc tính HTML.
- Sử dụng điều khiển 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 được sử dụng nhiều hơn cả.

Sử dụng thẻ SCRIPT: Script được 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 được tải và sẵn sàng trước khi phần còn lại của văn bản được tải. Với công nghệ hiện nay thẻ <SCRIPT> luôn được thiết lập thuộc tính LANGUAGE mặc định là JAVASCRIPT.

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

```
<SCRIPT>
    <!—Mã lệnh Javascript sẽ bị ẩn từ đây
    // Các lệnh Javascript
    // Đây là đoạn kết thúc ẩn mã lệnh -->
</SCRIPT>
```

Dòng cuối cùng của script cần có dấu // để trình duyệt không diễn dịch dòng này dưới dạng mã JavaScript.

5.1.1. Sử dụng một file nguồn JavaScript

Thuộc tính *SRC* của thẻ <*SCRIPT*> cho phép chỉ rõ file nguồn JavaScript được sử dụng.

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 nếu muốn đưa dòng lệnh sau vào giữa cặp thẻ <*SCRIPT SRC="..."*> và <*/SCRIPT*> có thể viết*:*

```
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://fit-haui.edu.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 được đuôi *.js* tới kiểu MIME application/x-javascript, Navigator sẽ tải file JavaScript được chỉ ra trong thuộc tính *SRC* về không đúng cách.

5.1.2. 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 được dịch như các dòng HTML khác và hiển thị trên trang.

Hơn nữa, JavaScript còn cho phép người lập trình tạo 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 trường 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 phương thức của một đối tượng như sau:

object_name.method_name

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

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

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

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

```
This text is plain
This text is bold.
This text is bold.
```

Hình 24. Phương thức write

```
// STOP HIDING FROM OTHER BROWSERS -->
</SCRIPT>
</PRE>
</BODY></HTML>
```

5.1.3. *Giao tiếp với người sử dụng*

JavaScript hỗ trợ khả năng cho phép người 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 hưởng đến việc xuất nội dung trang.

Cách đơn giản để làm việc đó là sử dụng cách thức **alert()**. Để sử dụng được cách thức này, bạn phải đưa vào một dòng text như khi sử dụng document.write() và document.writeln() trong phần trước. *Ví dụ*: alert("Nhấn vào OK để tiếp tục");

Hình 25. Hộp alert

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 vào form không hợp lệ.
- Kết quả sau khi tính toán không hợp lệ.
- Khi dịch vụ chưa sẵn sàng để truy nhập dữ liệu.

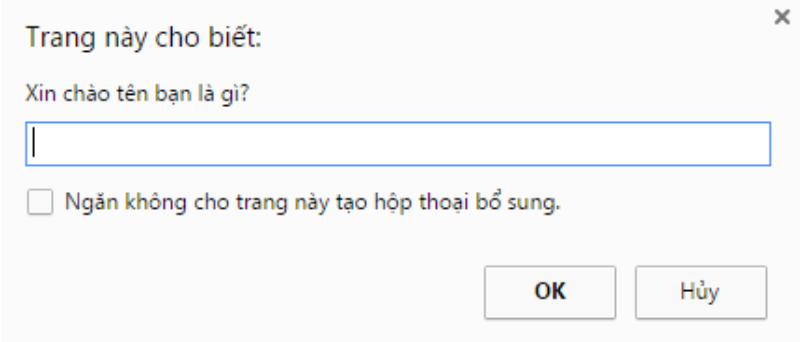
Tuy nhiên cách thức **alert**() mới chỉ cho phép thông báo với người sử dụng chứ chưa thực sự giao tiếp với người sử dụng. JavaScript cung cấp một cách thức khác để giao tiếp với người sử dụng là **prompt()**. Tương tự như **alert**(), **prompt**() tạo ra một hộp hội thoại với một dòng thông báo do bạn đưa vào, nhưng 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, 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 ngắn sử dụng tên mới đưa vào. Ví dụ được lưu vào file Hello.html

```
<HTML>
<HEAD>
    <TITLE>Ví dụ về JAVASCRIPT</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
        var name=window.prompt("Xin chào tên bạn là
        gì?","");
        document.write("Xin chào " + name + " ! Chúc bạn
        một ngày tốt lành ");
```

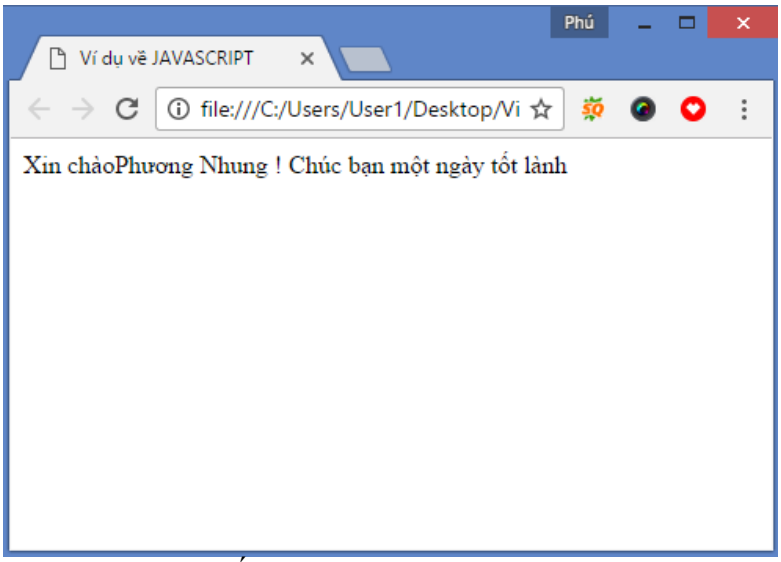
```
    </SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Khi duyệt có kết quả:



Hình 26. Hiển thị cửa sổ nhập tên

Ví dụ này hiển thị dấu nhắc nhập vào tên với phương thức **window.prompt**. Giá trị đạt được sẽ được ghi trong biến có tên là *name*. Biến *name* được kết hợp với các chuỗi khác và được hiển thị trong cửa sổ của trình duyệt nhờ phương thức **document.write**.



Hình 27. Kết hợp phương thức write và prompt

Hộp thoại **confirm** hỏi người dùng câu hỏi Yes – No và cung cấp tùy chọn cho người dùng trả lời qua 02 nút ấn OK và CANCEL. Hộp thoại **confirm** trả về một trong hai giá trị giá trị đúng (True) hoặc sai (False), kết quả trả về này do người dùng xác định cho đến khi hộp thoại được đóng lại.

Ví dụ:

```
var truthBeTold = window.confirm("Ấn OK để tiếp tục. Cancel
                                          để thoát");
        if (truthBeTold) {
            window.alert("Chào mừng bạn đến trang web của
                          chúng tôi!");
        }  else  window.alert("Chào tạm biệt!");
```

5.2. Biến trong Javascript

5.2.1. Biến và phân loại biến

Tên biến trong JavaScript phải bắt đầu bằng chữ hay dấu gạch dưới. Các chữ số không được sử dụng để mở đầu tên một biến nhưng có thể sử dụng sau ký tự đầu tiên. Phạm vi của biến có thể là một trong hai kiểu sau:

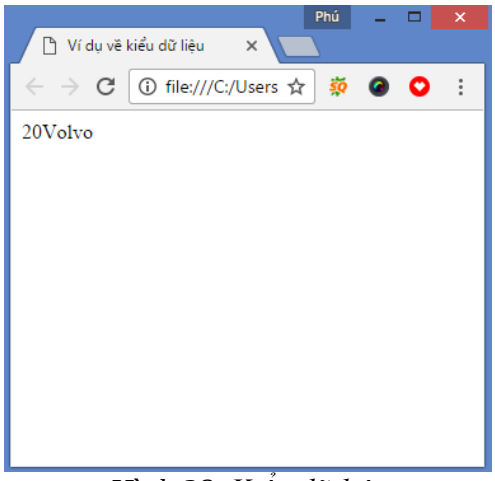
- *Biến toàn cục:* Có thể được truy cập từ bất kỳ đâu trong ứng dụng, được khai báo như sau: **x = 0;**
- *Biến cục bộ:* Chỉ được truy cập trong phạm vi chương trình mà nó khai báo. Biến cục bộ được khai báo trong một hàm với từ khoá ***var*** như sau: **var x = 0;**

5.2.2. 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 được tự động chuyển thành kiểu phù hợp khi cần thiết.

Ví dụ file javascript2.html:

```
<HTML><HEAD>
<TITLE> Ví dụ về kiểu dữ liệu
</TITLE>
<SCRIPT LANGUAGE= "JavaScript">
var x = 16 + 4 + "Volvo";
document.write(x);
</SCRIPT>
</HEAD><BODY></BODY></HTML>
```



Hình 28. Kiểu dữ liệu

Kiểu dữ liệu nguyên thủy là một giá trị mà trong đó không chứa phương thức và thuộc tính. Kiểu dữ liệu nguyên thủy sẽ mang giá trị nguyên thủy. Trong javascript định nghĩa gồm 5 kiểu dữ liệu nguyên thủy bao gồm : xâu ký tự (string), kiểu số (number), kiểu logic (boolean), kiểu dữ liệu trống (null) và kiểu dữ liệu không định nghĩa (undefined).

Kiểu số (Số nguyên và số thực): Số nguyên (Integer) có thể được biểu diễn theo ba cách: *Hệ cơ số 10* (hệ thập phân), *hệ cơ số 8* (hệ bát phân), *hệ cơ số 16* (hệ thập lục phân). Số thực (Floating Point): phần nguyên thập phân, dấu chấm thập phân (.), phần dư,

phần mũ. Để phân biệt kiểu dấu phảy động với kiểu số nguyên, phải có ít nhất một chữ số theo sau dấu chấm hay **E.** Ví dụ: -0.85E4.

Kiểu logic (Boolean): được sử dụng để chỉ hai điều kiện : đúng hoặc sai. Miền giá trị của kiểu này chỉ có hai giá trị: true, false.

Kiểu chuỗi (String): Một literal kiểu chuỗi được biểu diễn bởi không hay nhiều ký tự được đặt trong cặp dấu " ... " hay '... '. *Ví dụ:* "The dog ran up the tree".

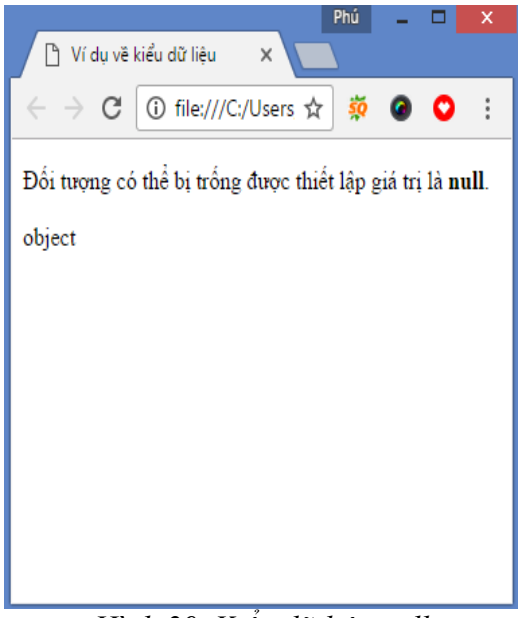
Để biểu diễn dấu nháy kép ("), trong chuỗi sử dụng (\"), *ví dụ:* document.write(" \"This text inside quotes.\" ");

Kiểu không định nghĩa (Undefined): là khi một biến không có giá trị, chứa giá trị không xác định.

Kiểu dữ liệu trống (NULL): trong javascript kiểu dữ liệu trống là không có gì (nothing) chứa bên trong. Kiểu dữ liệu của biến có giá trị là null thường là object.

Ví dụ:

```
<TITLE> Ví dụ về kiểu dữ liệu
</TITLE>
</HEAD><BODY>
<p>Đối tượng có thể bị trống được
thiết lập giá trị là
<b>null</b>.</p>
<p id="demo"></p>
<script>
var person = {firstName:"John",
lastName:"Doe", age:50,
eyeColor:"blue"};
var person = null;
document.getElementById("demo").inn
erHTML = typeof person;
</script>
</BODY></HTML>
```



Hình 29. Kiểu dữ liệu null

5.3. Xây dựng các biểu thức trong Javascript

5.3.1. Định nghĩa và 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:

- **Số học:** Nhằm để lượng giá trị số. Ví dụ *(3+4)+(84.5/3)* được đánh giá bằng *197.1666666667.*
- **Chuỗi:** Nhằm để đánh giá chuỗi. Ví dụ *"The dog barked" + barktone + "!"* là *The dog barked ferociously!.*
- **Logic:** Nhằm đánh giá giá trị logic. Ví dụ *temp>32* có thể nhận giá trị sai.

5.3.2. Các toán tử

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*.

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 thường	***Kiểu gán rút gọn***
x = x + y	x + = y
x = x – y	x - = y
x = x * y	x * = y
x = x / y	x / = y
x = x % y	x % = y

Toán tử 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.

Toán tử 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.
-	Toán tử phủ định, có giá trị phủ định toán hạng.
var ++	Toán tử này tăng var lên 1.
var --	Toán tử này giảm var đi 1.

Toán tử chuỗi: Khi được sử dụng với chuỗi, toán tử + được coi là kết hợp hai chuỗi, *ví dụ*: “abc” + “xyz” được “abcxyz”.

Toán tử Logic: JavaScript hỗ trợ các toán tử logic sau đây:

expr1 && expr2	Là toán tử logic AND, trả lại giá trị đúng nếu cả expr1 và expr2 cùng đúng.

Expr1 || expr2	Là toán tử logic OR, trả lại giá trị đúng nếu ít nhất một trong hai expr1 và expr2 đúng.
! expr	Là toán tử logic NOT phủ định giá trị của expr.

Toán tử Bitwise: Với các toán tử thao tác trên bit, đầu tiên giá trị được chuyển dưới dạng số nguyên 32 bit, sau đó lần lượt 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 trước 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 lượng 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 lượng 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 lượng bit bằng toán hạng phải. Bit dấu được dịch chuyển từ trái (giống >>). Những bit được dịch sang phải bị xoá đi. Ví dụ: -8>>>2 trở thành 1073741822 (bởi các bit dấu đã trở thành một phần của số). Tất nhiên với số dương kết quả của toán tử >> và >>> là giống nhau.

Có một số toán tử dịch chuyển bitwise rút gọn:

Kiểu bitwise thông thường	***Kiểu bitwise rút gọn***
x = x << y	x << = y
x = x >> y	x - >> y
x = x >>> y	x >>> = y
x = x & y	x & = y
x = x ^ y	x ^ = y
x = x | y	x | = y

- **Toán tử ?:** Cú pháp như sau:

(condition) ? valTrue : valFalse

Trong đó :

- 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 trường hợp ngược lại nó nhận giá trị "solid".

5.4. Các lệnh

5.4.1. 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** và **switch … case**

Câu lệnh 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 ( <biểu thức 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 **}** được sử dụng để tách các khối mã.*

Câu lệnh switch ... case: Switch so sánh một biểu thức nguyên với một danh sách giá trị các số nguyên, các hằng kí tự hoặc biểu thức hằng. Mỗi giá trị trong danh sách chính là một case label (nhãn trường hợp) trong khối mã lệnh của switch. Ngoài ra, trong khối mã lệnh của switch còn có thể có một default label (nhãn mặc định) - có thể có hoặc không. Mặt khác, trong mỗi label còn chứa các khối mã lệnh chờ được thực thi

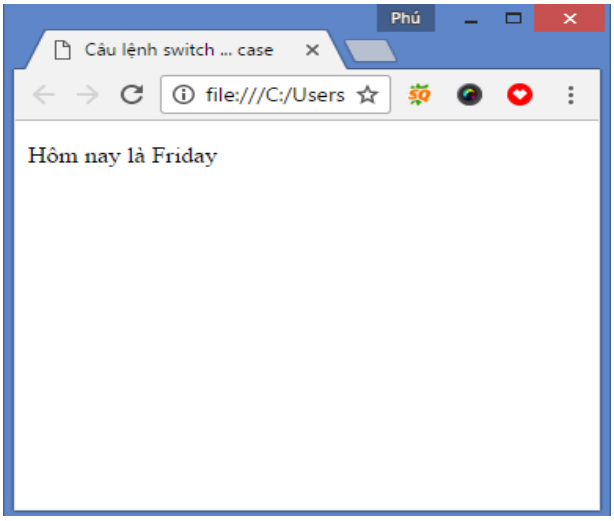
Cú pháp:

```
switch (<biến điều khiển>) {
case "nhãn 1":
     //Các câu lệnh với nhãn 1;
      break;
```

case nhãn 2:
//Các câu lệnh với nhãn 2;
break;
……
default:
//Các câu lệnh trong trường hợp mặc định;
}

Ví dụ:

```
<HTML><TITLE> Câu lệnh switch ... case </TITLE>
</HEAD>
<BODY>
<p id="demo"></p>
<script>
var day;
switch (new Date().getDay()) {
    case 0:day = "Sunday";break;
    case 1:day = "Monday";break;
    case 2:day = "Tuesday";break;
    case 3:day = "Wednesday";break;
    case 4:day = "Thursday";break;
    case 5:day = "Friday";break;
    case 6:day = "Saturday";
}
document.getElementById("demo").innerHTML = "Hôm nay là " +
day;
</script>
</BODY></HTML>
```



Hình 30. Câu lệnh switch ... case

5.4.2. Câu lệnh lặp

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

- for loop
- while loop
- do while

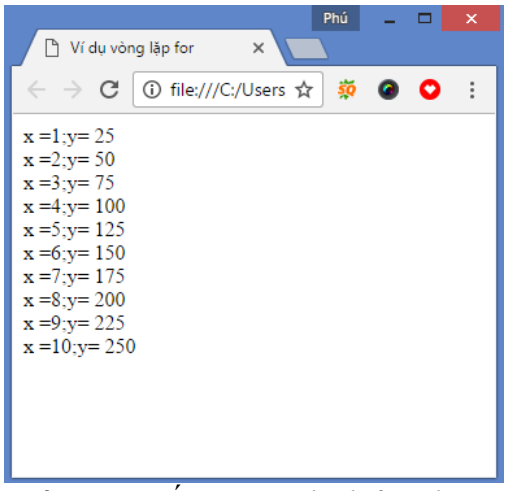
Vòng lặp for: thiết lập một biểu thức khởi đầu - initExpr, sau đó lặp một đoạn mã cho đến khi biểu thức *<điều kiện>* được đánh giá là đúng. Sau khi kết thúc mỗi vòng lặp, biểu thức *incrExpr* được đánh giá lại.

Cú pháp:

```
for (initExpr; <biểu thức điều kiện> ; incrExpr){
    //Các lệnh được 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>
```



Hình 31: Kết quả của lệnh for...loop

Câu lệnh while: Vòng lặp while lặp khối

lệnh chừng nào <điều kiện> còn được đánh giá là đúng.

Cú pháp:

while (<biểu thức 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 như ví dụ trước.

Câu lệnh do ... while: thực hiện giống như vòng lặp while nhưng khác là được thực hiện ít nhất 1 lần.

Cú pháp:

do
{
//Các câu lệnh thực hiện trong khi lặp
} while (<biểu thức điều kiện>);

Ví dụ:

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

Kết quả của ví dụ này giống như vòng lặp while.

Câu lệnh Break: dùng để kết thúc việc thực hiện của vòng lặp **for** hay **while**. Chương trình được 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 đưa vào vòng lặp nhỏ hơn 50, vòng lặp sẽ kết thúc.

Ví dụ:

```
while (x<100)
{
```

```
    if (x<50) break;
    x++;
}
```

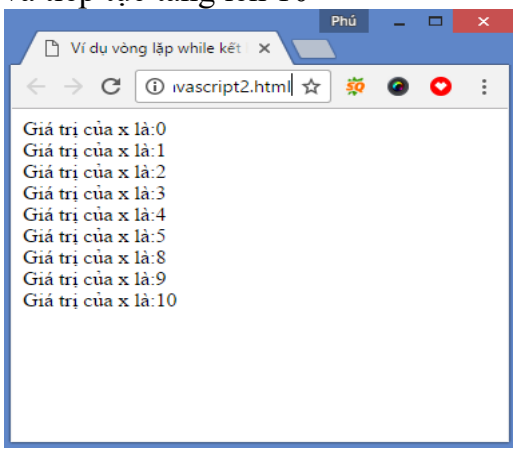
Câu lệnh continue: giống lệnh **break** nhưng khác ở chỗ việc lặp được kết thúc và bắt đầu vòng lặp mới với giá trị tiếp theo. Đối với vòng lặp **while**, lệnh **continue** điều khiển quay lại <điều kiện>; với **for,** lệnh **continue** điều khiển quay lại incrExpr.

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 là:"+ x+"<BR>");
if (x==5)
{
     x=8;
     continue;
}
x++;
}
```



Hình 32. Câu lệnh continue

5.4.3. Các câu lệnh thao tác trên đối tượng

Câu lệnh for...in: Câu lệnh này được sử dụng để lặp tất cả các thuộc tính (properties) của một đối tượng. Tên biến có thể là một giá trị bất kỳ, chỉ cần thiết khi bạn sử dụng các thuộc tính trong vòng lặp.

Cú pháp

for (<tên_biến> in <đối_tượng>)

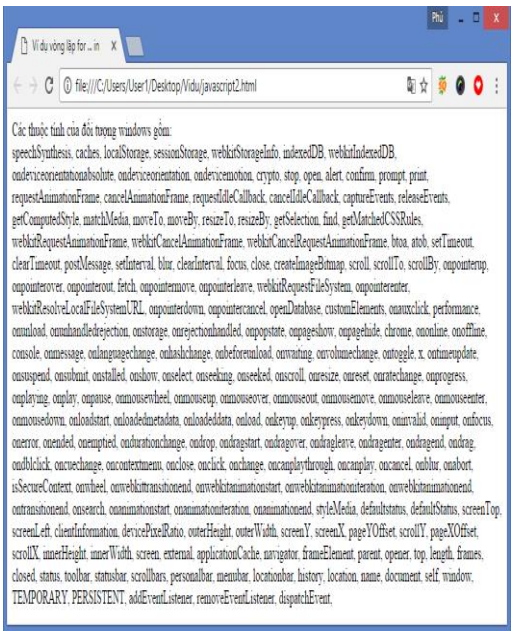
{

//Các câu lệnh

}

Ví dụ: Thực hiện lấy ra tất cả các thuộc tính của đối tượng Window và in ra tên của mỗi thuộc tính.

```
<HTML>
<HEAD>
<TITLE>For in Example </TITLE>
<SCRIPT LANGUAGE=
"JavaScript">
```



Hình 33 Câu lệnh for ... in

```
document.write("Các thuộc tính của đối tượng windows gồm:
<BR>");
for (var x in window)
     document.write("     "+ x + ", ");
</SCRIPT>
</HEAD>
<BODY>
</BODY></HTML>
```

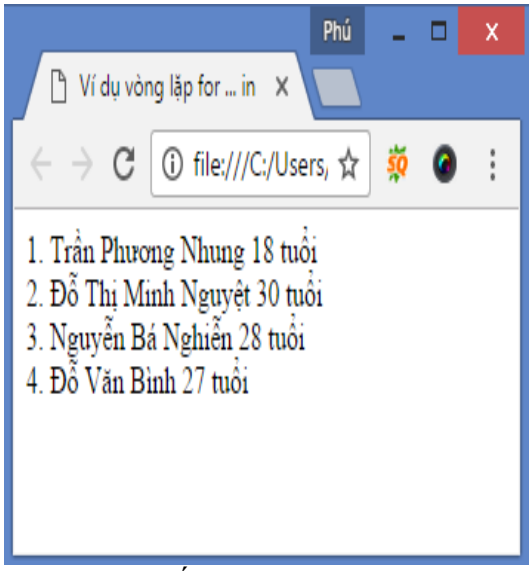
Câu lệnh new: được thực hiện để tạo ra một thể hiện mới của một đối tượng

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_nam
e;
     this.last_name=last_name;
     this.age=age;
     this.sex=sex;}
person1= new person("Nhung",
"Trần Phương", "18",
"Female");
person2= new person("Nguyệt",
"Đỗ Thị Minh", "30",
"Female");
person3= new person("Nghiễn",
"Nguyễn Bá", "28", "Male");
person4= new person("Bình",
"Đỗ Văn", "27", "Male");
```



Hình 34. Kết quả của ví dụ lệnh New

```
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>
```

Câu lệnh this: Từ khoá **this** được sử dụng để chỉ đối tượng hiện thời. Đối tượng được gọi thường là đối tượng hiện thời trong phương thức hoặc trong hàm.

Cú pháp:

this [.property]

Có thể xem ví dụ của lệnh new.

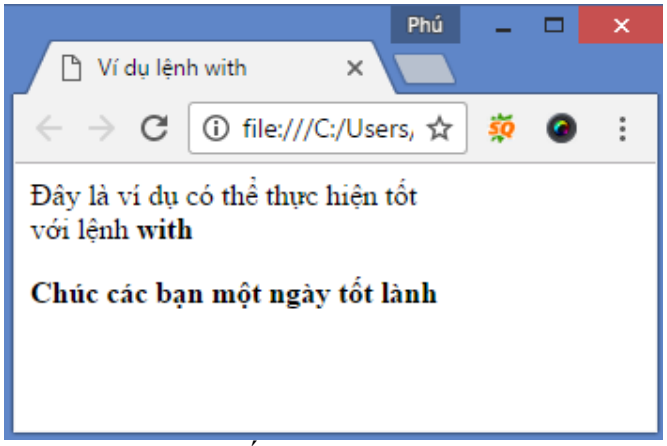
Câu lệnh with: Lệnh này được sử dụng để thiết lập đối tượng ngầm định cho một nhóm các lệnh, bạn có thể sử dụng các thuộc tính mà không đề cập đến đối tượng.

Cú pháp:

with (đối_tượng)
{
//Các câu lệnh thực hiện
}

Ví dụ: Chỉ ra cách sử dụng lệnh with để thiết lập đối tượng ngầm định là ***document*** và có thể sử dụng phương thức ***write*** mà không cần đề cập đến đối tượng document

```
<HTML>
<HEAD>
<TITLE>Ví dụ lệnh with</TITLE>
<SCRIPT LANGUAGE=
"JavaScript">
with (document){
write("Đây là ví dụ có
thể thực hiện tốt
<BR>");
write("với lệnh
<B>with<B> <P>");
write("Chúc các bạn một
ngày tốt lành");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
```



Hình 35: Kết quả của ví dụ lệnh with

```
</HTML>
```

5.5. Các hàm(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 dữ liệu 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])
{
//Các câu lệnh của hàm
}

Có thể xem lại ví dụ trong lệnh new

5.6. Các hàm có sẵn

JavaScript có một số hàm có sẵn, gắn trực tiếp vào chính ngôn ngữ và không nằm trong một đối tượng nào. Các hàm này bao gồm: eval, parseInt, parseFloat.

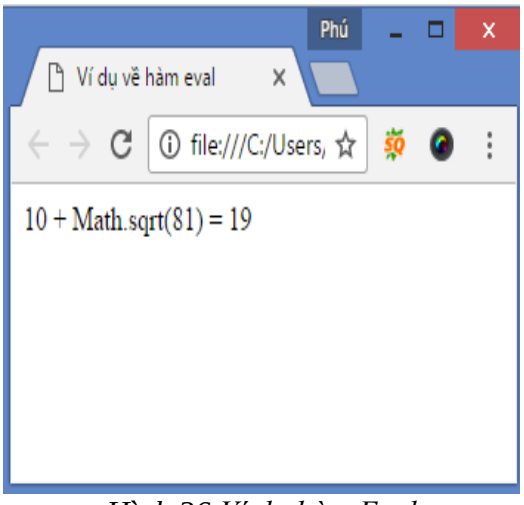
Hàm eval: Hàm này được sử dụng để đánh giá các biểu thức hay lệnh. Biểu thức, lệnh hay các đối tượng của thuộc tính đều có thể được đánh giá.

Cú pháp:

returnval=eval(bất kỳ biểu thức hay lệnh hợp lệ trong Javascript**)**

Ví dụ:

```
<HTML>
 <HEAD>
<TITLE>Ví dụ về hàm Eval
</TITLE>
<SCRIPT LANGUAGE=
"JavaScript">
     var string=”10+
Math.sqrt(81)”;
     document.write(string+
“=”+ eval(string));
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```



Hình 36 Ví dụ hàm Eval

Hàm 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

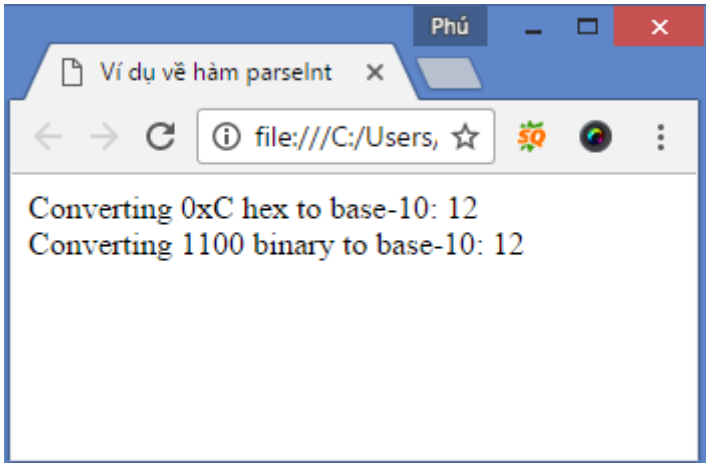
số nguyên sang cơ số 10 và đảm bảo rằng các dữ liệu được nhập dưới dạng ký tự được chuyển thành số trước khi tính toán. Trong trường hợp dữ liệu vào không hợp lệ, hàm parseInt sẽ đọc và chuyển dạng chuỗi đến vị trí nó tìm thấy ký tự không phải là số. Ngoài ra, hàm này còn cắt dấu phảy động.

Cú pháp:

parseInt (string, [, radix])

Ví dụ:

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



Hình 37: Ví dụ parseInt

Hàm parseFloat: Hàm này giống hàm parseInt nhưng nó chuyển chuỗi thành số biểu diễn dưới dạng dấu phảy động.

Cú pháp:

parseFloat (string)

Ví dụ: Mô tả cách thức xử lý của parseFloat với các kiểu chuỗi khác nhau.

```
<HTML> <HEAD>
<TITLE> perseFload Exemple </TITLE>
<SCRIPT LANGUAGE= "JavaScript">
document.write("This script will show how diffrent 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>
```

5.7. Mảng (Array)

Mảng là một biến đặc biệt để cung cấp nhiều giá trị trong cùng một thời điểm. Nếu muốn liệt kê ra các thành phần của mảng sẽ thấy việc lưu trữ giống như các biến độc lập. Ví dụ: var comp1="dell"; comp2="Toshiba".

Tuy nhiên, nếu muốn duyệt qua tất cả các biến comps thì ta cần làm gì? Và nếu như không phải chỉ có 2 biến mà là 200 biến thì sẽ thực hiện ra sao?

Việc giải quyết này sẽ được giải quyết ở mảng, mảng sẽ lưu trữ rất nhiều giá trị dưới cùng một tên và để lấy giá trị của mảng chỉ cần lấy thông qua chỉ số của mảng. Việc khởi tạo mảng thực hiện rất đơn giản theo cú pháp:

var biến_mảng = [thành_phần1, thành_phần2, ...];

Khi thực hiện khởi tạo có các khoảng trắng và giữa các dòng cũng không quá quan trọng vì việc khai báo có thể thực hiện trên nhiều dòng. Ví dụ:

```
var comps=[
        "dell",
        "toshiba",
        "lenovo"];
```

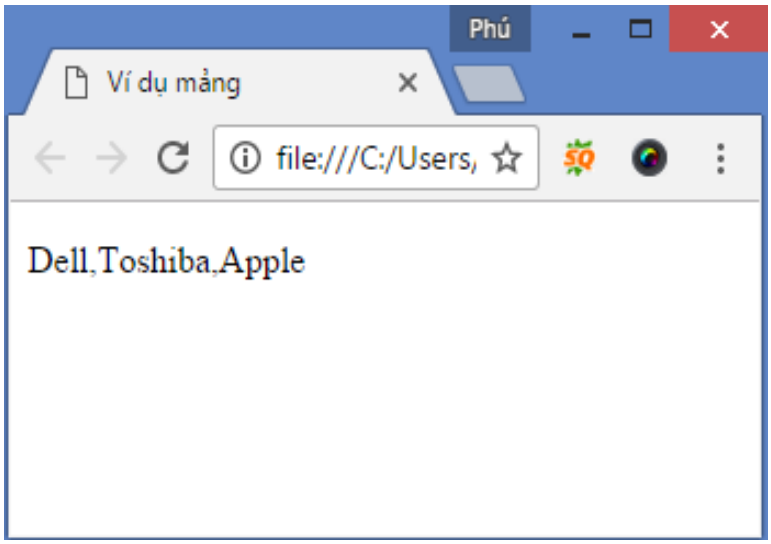
Ngoài ra, việc khai báo cũng có thể thực hiện thông qua câu lệnh **new**, việc thực hiện khai báo theo cú pháp sau:

var biến_mảng = new Array(thành_phần1, thành_phần2 ….);

Để truy xuất vào các thành phần thuộc mảng được thực hiện thông qua các chỉ số của mảng. Chỉ số đầu tiên được xác định trong mảng là 0, nếu muốn lấy giá trị của tất cả các thành phần trong mảng chỉ cần sử dụng tên biến mảng.

Ví dụ:

```
<HTML>
<TITLE> Ví dụ mảng </TITLE>
<body>
<p id="demo"></p>
<script>
var comps = ["Dell", "Toshiba", "Apple"];
document.getElementById("demo").innerHTML = comps;
</script>
</BODY></HTML>
```



Hình 38: Ví dụ mảng

Mảng cũng có thể là một kiểu đối tượng đặc biệt, toán tử typeof trong javascript sẽ trả về đối tượng kiểu mảng, tuy nhiên để tốt nhất nên thực hiện mô tả từ thành phần trong mảng thông qua tên.

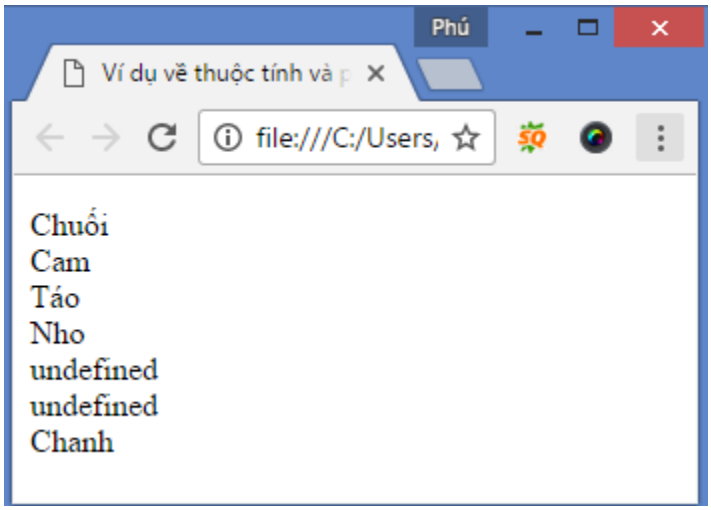
Ví dụ:

```
<HTML>
<TITLE> Ví dụ truy xuất thông qua tên thành phần của mảng
</TITLE>
<body>
<p id="demo"></p>
<script>
var person = {firstName:"Nhung", lastName:"Trần Phương",
age:18};
document.getElementById("demo").innerHTML =
person["firstName"];
</script>
```

Điểm đặc biệt của mảng đó chính là các phương thức và thuộc tính, chúng được xây dựng trong nội tại của mảng. Các phương thức và thuộc tính bao gồm: toString(), join(), pop(), push(), shift(), unshift(), delete, splice(), concat(), slice(), sort(), reverse(), length.

Ví dụ:

```
<HTML>
```



Hình 39. Ví dụ về thực hiện thuộc tính và phương thức của mảng

```
<TITLE> Ví dụ về thuộc tính và phương thức của mảng
</TITLE>
<body>
<p id="demo"></p>
<script>
var fruits, text, fLen, i;
fruits = ["Chuối", "Cam", "Táo", "Nho"];
fruits[6] = "Chanh";
fLen = fruits.length;
text = "";
for (i = 0; i < fLen; i++) {
    text += fruits[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script></BODY></HTML>
```

Chú ý: Việc thêm thành phần vào mảng trong đó có các thành phần trống ở giữa sẽ được nhận giá trị là *undefined*.