Chương 3

Cú pháp và câu lệnh trong JavaScript

Sau khi đọc xong chương này, bạn có thể:

- Nắm được các nguyên tắc cơ bản khi sử dụng ngôn ngữ lập trình JavaScript.
- Đặt mã JavaScript trong một trang web.
- Nhận biết câu lệnh JavaScript.
- Nhận biết các từ khóa trong JavaScript.

Một vài công việc lặt vặt

Phần còn lại của cuốn sách sẽ đề cập cụ thể hơn về những đặc trưng của JavaScript và làm thế nào để thực hiện các nhiệm vụ cụ thể. Tuy nhiên, bạn phải học đi trước khi học chạy. Vì thế, trước khi tìm hiểu sâu về JavaScript, bạn cần nắm được một số *cấu trúc từ vựng* của nó – đó là các quy tắc hay còn được gọi là *quy tắc cú pháp* của ngôn ngữ này.

Phân biệt chữ hoa chữ thường

JavaScript phân biệt chữ hoa và chữ thường. Cần chú ý điều này khi đặt tên cho biến và sử dụng từ khóa của ngôn ngữ. Một biến được đặt tên là `remote` sẽ khác biến có tên là `Remote` hay `REMOTE`. Tương tự, từ khóa cho vòng lặp là `while`, nhưng nếu viết thành `WHILE` hoặc `While` thì sẽ gây ra lỗi.

Các từ khóa được viết bằng chữ thường, còn biến có thể kết hợp tùy ý cả chữ hoa và chữ thường miễn là bạn có thể tự kiểm soát được. Ví dụ: Các tên biến được liệt kê dưới đây đều là tên biến hợp lệ trong JavaScript.

```
buttonOne

txt1

a

C
```

Mách nhỏ Bạn sẽ thấy JavaScript thường sử dụng chữ thường trong mã, ngoại trừ một vài trường hợp cần thiết, ví dụ như hàm *isNaN()*, dùng để xác định xem giá trị đầu vào có phải là số hay không (*NaN* trong tên hàm có nghĩa là Not a Number - không phải số). Bạn sẽ học về hàm này trong Chương 4 "Làm việc với biến và kiểu dữ liệu"

Chương 4 sẽ cung cấp thêm kiến thức về biến và quy ước đặt tên, còn bây giờ, bạn hãy chú ý đến chữ hoa, chữ thường khi viết tên biến trong JavaScript.

Ký tự trắng

Trong hầu hết trường hợp, JavaScript bỏ qua *ký tự trắng* nằm giữa các câu lệnh. Bạn có thể thêm ký tự trắng, lùi đầu dòng hoặc viết mã theo bất cứ quy ước mã nào để đoạn mã JavaScript rõ ràng và dễ đọc hơn. Tuy nhiên, có một vài ngoại lệ trong quy tắc này. Một số từ khóa, như `return` có thể bị thông dịch sai bởi trình thông dịch JavaScript khi được đặt cùng dòng với từ khóa `return` khác. Phần sau của chương sẽ trình bày ví dụ về vấn đề này.

Làm cho chương trình dễ đọc là một lý do để sử dụng ký tự trắng. Ví dụ như đoạn mã dưới đây dùng rất ít ký tự trắng và khoảng lùi đầu dòng:

```
function cubeme(incomingNum) { // Hàm tính lập phương
if (incomingNum == 1) {
return "Bạn đang làm gì đấy?";
} else {
return Math.pow(incomingNum,3);
}
}
var theNum = 2;
var finalNum = cubeme(theNum);
if (isNaN(finalNum)) {
alert("Bạn nên nhớ rằng 1 lũy thừa bao nhiêu vẫn là 1.");
} else {
alert("Lập phương của " + theNum + " là " + finalNum);
}
```

Bây giờ chúng ta xem xét lại đoạn mã trên có sử dụng thêm những khoảng lùi đầu dòng. (Bạn có thể tìm thấy đoạn mã này trong file example1.txt, thư mục mã nguồn mẫu Chương 3 của phần Tài nguyên đi kèm.)

```
function cubeme(incomingNum) { // Hàm tính lập phương
    if (incomingNum == 1) {
        return "Bạn đang làm gì đấy?";
    } else {
        return Math.pow(incomingNum,3);
    }
}

var theNum = 2;
var finalNum = cubeme(theNum);

if (isNaN(finalNum)) {
     alert("Bạn nên nhớ rằng 1 lũy thừa bao nhiêu vẫn là 1.");
} else {
     alert("Lập phương của " + theNum + " là " + finalNum);
}
```

Đoạn mã thứ hai thực thi chức năng giống như đoạn mã đầu nhưng dễ đọc và dễ theo dõi hơn! Thực tế là có thể bạn chỉ mất một chút thời gian để viết mã, nhưng bạn sẽ phải làm việc

với nó trong nhiều năm liền. Như trường hợp của tôi khi đọc lại đoạn mã sau một năm viết, tôi đã rất vui mừng vì trước đó mình đã viết mã một cách dễ đọc và dễ theo dõi.

Chú thích

Với việc viết mã và duy trì chúng trong một thời gian dài, chú thích là người bạn đồng hành tốt. Những đoạn mã trông hết sức rõ ràng nhưng lần tới đọc lại chúng sẽ không được như thế nữa, đặc biệt khi bạn viết từ quá lâu. Chú thích có thể được đặt trong đoạn mã JavaScript theo hai cách: chú thích nhiều dòng và chú thích một dòng.

Bạn sẽ quen với chú thích nhiều dòng trong JavaScript nếu như bạn từng sử dụng ngôn ngữ lập trình C. Chú thích *nhiều dòng* bắt đầu bằng ký tự /* và kết thúc bằng ký tự */ như dưới đây:

```
/* Đây là chú thích nhiều dòng trong JavaScript
Giống như chú thích trong ngôn ngữ C, đoạn chú thích này có thể kéo
dài nhiều dòng trước khi được đóng lại */
```

Chú thích *một dòng* bắt đầu với hai dấu gạch chéo (/ /) và không cần ký tự kết thúc vì nó chỉ ở trong một dòng. Dưới đây là ví dụ:

```
// Đây là chú thích một dòng.
```

Bạn có thể sử dụng nhiều chú thích một dòng. Với những khối chú thích ngắn, bạn có thể sử dụng nhiều chú thích một dòng thay vì dạng chú thích nhiều dòng như đã trình bày ở trên. Ví dụ như, bạn hãy xem khối chú thích dưới đây:

```
// Đây là một khối chú thích khác.
// Khối chú thích này có nhiều dòng.
// Trước mỗi dòng có hai dấu gạch chéo.
```

Mách nhỏ Với những chú thích ngắn, chỉ dài một hoặc hai dòng, bạn sẽ thao tác nhanh hơn nếu sử dụng cách chú thích với hai dấu gạch chéo. Với những chú thích dài hơn, ví dụ những chú thích ở đầu chương trình hay đầu đoạn mã, cách chú thích nhiều dòng là lựa chọn tốt hơn vì nó giúp chúng ta có thể dễ dàng thêm hoặc xóa thông tin.

Dấu chấm phẩy

Trong JavaScript, dấu chấm phẩy được sử dụng để phân tách các biểu thức. Về mặt kỹ thuật, dấu chấm phẩy là không bắt buộc trong hầu hết các câu lệnh và biểu thức. Tuy nhiên, những vấn đề khó lường bạn gặp phải khi không sử dụng dấu chấm phẩy có thể gây ra những lỗi không đúng do đó làm tốn thời gian tìm và gỡ lỗi. Trong một số trường hợp, trình thông dịch JavaScript có thể tự động chèn thêm dấu chấm phẩy khi bạn không muốn. Ví dụ như câu lệnh dưới đây:

```
return
(varName);
```

Thông thường, bạn nên viết là:

```
return(varName);
```

Nhưng JavaScript tự ý thêm một dấu chấm phẩy vào câu lệnh *return*, làm cho đoạn mã trở thành như dưới đây trong trình thông dịch JavaScript:

```
return;
(varName);
```

Đoạn mã này không hoạt động, trình thông dịch đã hiểu nhầm ý của bạn. Nếu bạn dùng đoạn mã này trong một hàm, nó sẽ trả về giá trị *undefined* (không xác định) cho hàm gọi, và đó không phải là điều bạn mong muốn. Đây là một ví dụ cho thấy chúng ta không nên tùy tiện sử dụng ký tự trắng - bạn cũng không thể sử dụng dấu ngắt dòng (nội dung này sẽ được trình bày cụ thể ở phần kế tiếp) để tách biệt từ khóa *return* và giá trị trả về.

Việc lập trình JavaScrip sẽ trở nên dễ dàng hơn nếu bạn sử dụng dấu chấm phẩy như là một quy tắc thay vì phải nhớ xem không cần dùng nó ở đâu.

Tuy nhiên, bạn nhất thiết không được dùng dấu chấm phẩy trong trường hợp sau: sử dụng vòng lặp và các điều kiện:

```
if (a == 4)
{
     // đoạn mã được đặt ở đây
}
```

Trong trường hợp này, bạn sẽ không sử dụng dấu chấm phẩy ở cuối câu lệnh *if* vì câu lệnh hoặc các khối câu lệnh bên trong cặp dấu ngoặc đi sau điều kiện là một phần của câu lệnh điều kiện, trong trường hợp này là câu lệnh *if*. Giả sử bạn đặt dấu chấm phẩy cuối câu lệnh *if*, phần đầu của câu lệnh *if* sẽ bị tách khỏi toàn bộ phần còn lại. Ví dụ, đoạn mã ở dưới đây là sai. (Đoạn mã trong cặp dấu ngoặc nhọn được thực thi bất kể giá trị *a* có bằng 4 hay không).

```
if (a == 4);
{
     // đoạn mã được đặt ở đây
}
```

Mách nhỏ Không sử dụng dấu chấm phẩy khi viết vòng lặp hoặc khai báo hàm.

Dấu ngắt dòng

Liên quan chặt chẽ tới ký tự trắng và thậm chí dấu chấm phẩy trong JavaScript là dấu ngắt dòng, đôi khi còn được gọi là *ký tự trở về đầu dòng*. Trong chuẩn ECMA-262, dấu ngắt dòng hay còn được gọi là "dấu kết thúc dòng" được dùng để phân biệt một dòng mã với dòng mã tiếp theo. Cũng như dấu chấm phẩy, vị trí đặt dấu ngắt dòng cũng cần phải được lưu ý. Có thể thấy từ ví dụ của phần trước, đặt sai vị trí dấu ngắt dòng có thể dẫn đến các lỗi khó lường.

Không có gì ngạc nhiên khi lợi ích thường gặp nhất của dấu ngắt dòng là để phân biệt các dòng mã khác nhau cho dễ đọc. Bạn cũng có thể làm cho các dòng mã dài dễ hiểu hơn bằng cách xuống dòng. Tuy nhiên, khi làm như vậy, bạn phải cẩn thận với những lỗi như kết quả trả về từ câu lệnh *return* đã nêu ở trên, trong đó dấu ngắt dòng thêm vào có thể gây ra những hiệu ứng không mong muốn, làm thay đổi ý nghĩa của đoạn mã.

Đặt đoạn mã JavaScript đúng vị trí

Bạn có thể đặt mã JavaScript ở nhiều vị trí trong trang HTML: trong phần *<head> </head>*, hoặc giữa cặp thẻ *<body>* </body>. Vị trí phổ biến của đoạn mã JavaScript là ở giữa cặp thẻ *<head> </head>* ở gần đầu trang. Tuy nhiên, cách đặt đoạn mã *<script>* trong phần *<body>* đang trở nên phổ biến hơn. Bạn phải khai báo trước loại script (mã kịch bản) mà bạn sử dụng. Vì đây là cuốn sách về JavaScript nên chúng ta sẽ khai báo như sau trong thẻ *<script>*:

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

Một vấn đề quan trọng cần lưu ý khi bạn sử dụng JavaScript với các trang được khai báo dạng XHTML là trong các trang này, thẻ *<script>* cần phải được khai báo trong phần CDATA. Nếu không, XHTML sẽ phân tích thẻ *<script>* như một thẻ XML và đoạn mã trong đó có thể không chạy như bạn mong đợi. Do đó, để sử dụng JavaScript trong XHTML cần khai báo như sau:

```
<script type="text/javascript">
<![CDATA[

     // Đoạn mã JavaScript được đặt ở đây

]]>
</script>
```

Các phiên bản trình duyệt cũ có thể không phân tích chính xác CDATA. Vấn đề này có thể được giải quyết bằng cách đặt dòng mở đầu và kết thúc của CDATA bên trong dấu chú thích của JavaScript như ví dụ dưới đây:

```
<script type="text/javascript">
//<![CDATA[

     // Đoạn mã JavaScript được đặt ở đây

//]]>
</script>
```

Khi đặt đoạn mã JavaScript trong một file ngoài (như bạn đã đọc ở Chương 2 "Lập trình với JavaScript"), bạn không cần dùng phần CDATA nữa. Bạn sẽ thấy rằng ngoại trừ những đoạn JavaScript ngắn nhất, sẽ tốt hơn nếu bạn định nghĩa JavaScript trong các file ngoài - thường là với đuôi mở rộng .js - và sau đó liên kết các file trong trang. Chương 2 đã mô tả cụ thể vấn đề này, nhưng phần này sẽ nói đến cách thức liên kết đến một file sử dụng thuộc tính *src* trong thẻ *<script>*:

```
<script type="text/javascript" src="myscript.js">
```

Đặt đoạn JavaScript vào file ngoài có nhiều lợi ích, ví dụ như:

- **Phân tách đoạn mã ra khỏi các thẻ đánh dấu:** Lưu mã JavaScript vào file ngoài khiến việc bảo trì đoạn mã HTML dễ hơn và giúp giữ cấu trúc của HTML mà không phải sử dụng phần CDATA cho XHTML.

- **Dễ bảo trì:** Với việc đặt mã JavaScript vào một file ngoài, bạn có thể thay đổi file đó mà không làm ảnh hưởng đến các file khác của website.
- **Lưu trữ tạm thời (Caching):** Nếu sử dụng file JavaScript ngoài, trình duyệt web có thể lưu trữ tạm thời file đó và nhờ thế làm tăng tốc độ tải trang web cho người dùng.

Câu lệnh trong JavaScript

Cũng như các chương trình được viết bằng ngôn ngữ khác, chương trình được viết bằng JavaScript cũng bao gồm các câu lệnh được sắp xếp để trình thông dịch JavaScript có thể thực hiện một hay nhiều tác vụ. Và cũng như các ngôn ngữ khác, câu lệnh trong JavaScript có thể đơn giản hoặc phức tạp. Phần này sẽ mô tả ngắn gọn mẫu câu lệnh JavaScript, với giả định rằng bạn đã xem qua một số ví dụ ở các chương trước cũng như ở các chương sau của cuốn sách.

Câu lệnh JavaScript có những phần tử gì?

Như đã trình bày trong Chương 1 "Hiểu hơn về JavaScript", câu lệnh JavaScript, hay biểu thức, là một tập hợp các từ khóa, toán tử hoặc chuỗi định danh được đặt với nhau để tạo thành một chương trình mà trình thông dịch JavaScript có thể hiểu được. Câu lệnh thường kết thúc với dấu chấm phẩy, ngoại trừ trong một vài trường hợp đặc biệt như lệnh khai báo cấu trúc rẽ nhánh và vòng lặp như *`if`*, *`while`*, và *`for`* được trình bày cụ thể hơn trong Chương 5 "Sử dụng toán tử và biểu thức".

Dưới đây là một vài ví dụ về câu lệnh cơ bản trong JavaScript:

```
var x = 4;
var y = x * 4;
alert("Xin chào");
```

Hai loại câu lệnh JavaScript

Câu lệnh JavaScript có hai loại cơ bản: đơn và phức hợp. Bạn không cần mất nhiều thời gian với khái niệm câu lệnh. Tuy vậy, bạn nên hiểu sự khác nhau giữa câu lệnh đơn và câu lệnh phức hợp.

Câu lệnh đơn là một câu lệnh như dưới đây:

```
var x = 4;
```

Câu lệnh phức hợp kết hợp nhiều cấp độ logic với nhau. Một câu lệnh điều kiện *`if/then/else`* như dưới đây là ví dụ điển hình cho câu lệnh phức hợp:

```
if (something == 1) {
  // đoạn mã đặt ở đây
} else {
  // đoạn mã khác được đặt ở đây
}
```

Từ khóa trong JavaScript

Một số từ trong JavaScript là *từ khóa*, nghĩa là bạn không thể sử dụng nó làm biến, chuỗi định danh hoặc hằng số trong chương trình của bạn. Nếu bạn làm vậy, đoạn mã sẽ gặp những kết quả không mong đợi như bị lỗi. Ví dụ, bạn đã thấy từ khóa *var* trong một vài ví dụ trước đây. Sử dụng từ khóa *var* ngoài mục đích để khai báo biến có thể gây ra lỗi hoặc các hiệu ứng không mong muốn khác tùy vào trình duyệt. Ví dụ như câu lệnh dưới đây:

```
// Không được viết như thế này!
var var = 4;
```

Đoạn mã ví dụ này không dẫn đến lỗi trực tiếp trên trình duyệt, nhưng nó sẽ không hoạt động như bạn mong muốn.

Danh sách dưới đây bao gồm những từ được dùng làm từ khóa theo chuẩn ECMA-262:

break	delete	if	this	while
case	do	in	throw	with
catch	else	instanceof	try	
continue	finally	new	typeof	
debugger	for	return	var	
default	function	switch	void	

Một số từ khóa khác (được liệt kê ở danh sách dưới đây) được dành cho các mục đích sử dụng trong tương lai và do đó bạn không nên sử dụng chúng trong chương trình của mình.

class	enum	extends	super	
const	export	import		

Danh sách các từ khóa dưới đây được dành để sử dụng trong chế độ strict:

implements	let	private	public	yield
interface	package	protected	static	

Sơ lược về hàm

Bạn đã được xem nhiều ví dụ về hàm ở các chương trước. JavaScript có nhiều *hàm có sẵn* hoặc hàm do bản thân ngôn ngữ này định nghĩa. Chúng ta đã đề cập đến hàm *`alert()`* ngoài ra còn nhiều hàm khác nữa. Tùy theo phiên bản ngôn ngữ mà bạn đang sử dụng, sẽ có các hàm có sẵn khác nhau. Nhiều hàm chỉ có ở các phiên bản gần đây của JavaScript – những phiên bản chỉ được hỗ trợ bởi một số trình duyệt. Tìm ra các hàm (và các đối tượng) có sẵn trên trình duyệt là một công việc quan trọng để xác định xem trình duyệt của người dùng có thể chạy mã JavaScript mà bạn tạo trên trang web của mình không. Nội dung này sẽ được đề cập ở Chương 11 "Các sự kiện trong JavaScript và làm việc với trình duyệt".

Mách nhỏ Một nguồn thông tin rất hữu ích về sự tương thích trình duyệt là trang web QuirksMode: *(http://www.quirksmode.org/compatibility.html)*.

Tương tự các ngôn ngữ lập trình khác, JavaScript cũng chấp nhận các hàm người dùng định nghĩa. Ví dụ trước đấy trong chương này đã định nghĩa một hàm có tên *cubeme()* để tính lập phương của một số. Đoạn mã đó cũng cho thấy cách dùng JavaScript ở cả thẻ *<head>* và *<body>* của trang web.

Đặt hàm người dùng định nghĩa trong JavaScript

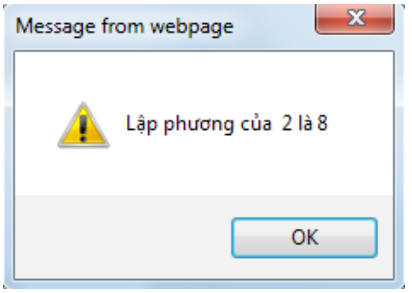
1. Dùng Visual Studio, Eclipse, hoặc một trình soạn thảo khác, thay đổi file example1.htm trong thư mục mã nguồn mẫu của Chương 3.
2. Trong trang web, thay thế dòng chú thích TODO bằng đoạn mã in đậm dưới đây:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<script type="text/javascript">
function cubeme(incomingNum) {
     if (incomingNum == 1) {
          return "Bạn đang làm gì đấy?";
     } else {
          return Math.pow(incomingNum,3);
     }
}
</script>
     <title>Ví dụ Chương 3</title>
</head>

<body>
<script type="text/javascript">
var theNum = 2;
var finalNum = cubeme(theNum);
if (isNaN(finalNum)) {
     alert("Bạn nên nhớ rằng 1 lũy thừa bao nhiêu vẫn là 1.";
} else {
     alert("Lập phương của " + theNum + " là " + finalNum);
}
</script>

</body>
</html>
```

3. Lưu file này lại, sau đó chạy đoạn mã hoặc sử dụng trình duyệt để mở trang web. Bạn sẽ thấy một thông báo như sau:



Đoạn mã trong ví dụ này kết hợp đoạn mã từ ví dụ trước đó để tạo thành một trang HTML hoàn chỉnh, bao gồm cả khai báo DOCTYPE. Đoạn mã này khai báo một hàm là *cubeme()* trong thẻ *<head>* của tài liệu, như sau:

```
function cubeme(incomingNum) {
     if (incomingNum == 1) {
          return "Bạn đang làm gì đấy?";
     } else {
          return Math.pow(incomingNum,3);
     }
}
```

Hàm này chấp nhận một đối số là *incomingNum*. Câu lệnh điều kiện *if/then* là trọng tâm của hàm. Khi *incomingNum* bằng 1, hàm sẽ trả về chuỗi ký tự "Bạn đang làm gì đấy?". Và ngược lại, khi *incomingNum* khác 1, phương thức *Math.pow* được gọi với *incomingNum* và số nguyên 3 là đối số. Việc gọi hàm *Math.pow* đã tăng giá trị của *incomingNum* lên lũy thừa bậc 3 và giá trị này được trả về hàm gọi. Cách thức gọi hàm sẽ được đề cập tiếp ở Chương 4.

Toàn bộ đoạn mã trước đều được đặt trong thẻ *<head>* của tài liệu, do đó nó có thể được các đoạn mã khác gọi đến; đây chính là việc mà chúng ta sắp thực hiện. Trình duyệt khi đó sẽ hiển thị phần *<body>* của tài liệu, bao gồm một vài đoạn mã JavaScript khác. Và đoạn mã tiếp theo này sẽ gán cho biến *theNum* giá trị bằng 2.

```
var theNum = 2;
```

Sau đó, đoạn mã sẽ gọi hàm được định nghĩa trước đó là *cubeme()*, sử dụng biến *theNum* làm đối số. Cần chú ý rằng biến finalNum sẽ nhận giá trị trả về từ hàm *cubeme()*, như dưới đây:

```
var finalNum = cubeme(theNum);
```

Đoạn mã JavaScript cuối cùng trong trang là tập hợp các điều kiện *if/then*. Đoạn mã này kiểm tra để xác định xem giá trị trả về được lưu trong biến finalNum có là số hay không, bằng cách sử dụng hàm *isNaN()*. Nếu giá trị trả về không phải là số, hộp thoại thông báo sẽ được hiển thị với nội dung cho thấy *1* đã được dùng làm đối số (Tất nhiên có nhiều lý do khiến giá trị trả về không phải là số, tuy nhiên chúng ta hãy tạm thời chấp nhận ví dụ đã nêu). Nếu như giá trị trả về là số, số đó được hiển thị, và bạn có thể thấy ở hộp thoại của hàm *alert()* hiển thị giá trị như ở bước 3 bên trên.

Chế độ strict của JavaScript

ECMA-262 phiên bản 5 giới thiệu một biến thể nghiêm ngặt, thường được gọi là *chế độ strict*, trong đó có cải tiến chức năng kiểm tra lỗi và bảo mật. Ví dụ, để chặn nguy cơ gõ nhầm tên biến, việc khai báo biến bắt buộc phải sử dụng từ khóa `var`. Thêm vào đó, những thay đổi hàm `eval()` và các khía cạnh khác cũng giúp chúng ta nâng cao chất lượng mã JavaScript.

Chế độ strict được kích hoạt bằng cú pháp rất giống với cú pháp dùng trong Perl:

```
"use strict";
```

Chế độ strict có phạm vi cục bộ tại khối mà nó được sử dụng vì thế bạn có thể kích hoạt toàn cục bằng cách đặt dòng `use strict` ở đầu đoạn mã JavaScript hoặc bạn có thể kích hoạt nó trong một hàm, bằng cách đặt dòng `use strict` trong chính hàm đó ví dụ như:

```
function doSomething() {
     "use strict";
     //Đoạn mã của hàm được đặt ở đây.
}
```

Một cải tiến khác trong chế độ strict có thể bắt được lỗi đánh máy là tính năng ngăn chặn các biến chưa được khai báo. Tất cả các biến trong chế độ strict cần được khởi tạo trước khi sử dụng. Ví dụ như câu lệnh dưới đây:

```
"use strict";
x = 4; // Tạo ra lỗi cú pháp
```

Đoạn mã này sẽ gây lỗi vì biến `x` vẫn chưa được khai báo trước với từ khóa `var`, như dưới đây:

```
"use strict";
var x = 4; // Cú pháp này là đúng.
```

Một cải tiến đáng chú ý về bảo mật mà chế độ strict mang lại là thay đổi cách thức vận dụng hàm `eval()`. Trong chế độ strict, `eval()` không thể khởi tạo một biến hoặc hàm mới để dùng bên ngoài câu lệnh `eval()`. Ví dụ:

```
"use strict";
eval("var testVar = 2;");
alert(testVar); // Tạo ra lỗi cú pháp.
```

Trong đoạn mã ví dụ, sẽ có một lỗi cú pháp vì chế độ strict được kích hoạt làm cho biến `testVar` không truy xuất được bên ngoài câu lệnh `eval()`.

Chế độ strict cũng ngăn chặn việc trùng tên biến trong một đối tượng hoặc một lời gọi hàm:

```
"use strict";
var myObject ={
```

```
testVar: 1,
testVar: 2
};
```

Đoạn mã trên sẽ gây ra lỗi ở chế độ strict vì biến *testVar* được thiết lập hai lần trong đoạn mã định nghĩa đối tượng.

Cũng như các tính năng khác của ECMA-262 phiên bản 5, chế độ strict có thể không khả dụng ở tất cả các trình duyệt và nhiều khả năng không chạy được trên các trình duyệt cũ. Bạn có thể tìm hiểu thêm thông tin về ECMA-262 phiên bản 5 và ví dụ đầy đủ tại đây: *http://besen.sourceforge.net/*.

Bài tập

1. Câu lệnh JavaScript nào dưới đây hợp lệ? (Chọn tất cả các đáp án có thể)

   a. `if (var == 4) { // Viết mã ở đây }`

   b. `var testVar = 10;`

   c. `if (a == b) { // Viết mã ở đây }`

   d. `testVar = 10;`

   e. `var case = "Yes";`

2. Đúng hay sai: Bạn bắt buộc phải dùng dấu chấm phẩy để kết thúc toàn bộ các câu lệnh JavaScript?

3. Kiểm tra đoạn JavaScipt dưới đây. Kết quả sẽ như thế nào? (Giả định rằng khai báo JavaScript đã có và đoạn mã này được đặt hợp lệ trong phần <*head*> của trang).

```
var orderTotal = 0;
function collectOrder(numOrdered) {
     if (numOrdered > 0) {
          alert("Bạn đã đặt " + orderTotal);
          orderTotal = numOrdered * 5;
     }
return orderTotal;
}
```