Mục lục

Giới thiệu JavaScript

1. Giới thiệu JavaScript

- JavaScript, là ngôn ngữ lập trình kịch bản dựa trên nền HTML

- JavaScript tương tác với các phần tử HTML trên trang web nhằm làm cho trang web trở nên động, có khả năng phản hồi, giao tiếp với người dùng.

- JavaScript, có khả năng thay đổi nội dung trang web, các thuộc tính của các phần tử HTML.

- JavaScript, có khả năng thay đổi thuộc tính CSS trên trang Web.

- JavaScript, có khả năng làm việc với dữ liệu trên trang web.

2. Mã JavaScript viết ở vị trí nào?

Mã JavaScript được đặt trong thẻ <body>, thẻ <head> của trang HTML.

Mã JavaScript phải được viết trong thẻ <script> giữa thẻ mở <script> và thẻ đóng </script>.

Ví dụ:

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

2.1. Hàm và sự kiện JavaScript

Một hàm JavaScript là một khối mã lệnhJavaScript, mà có thể thi hành khi được yêu cầu.

Ví dụ, một hàm có thể được thi hành khi một sự kiện xảy ra, như là có một cú bấm chuột vào một nút bấm

2.2. Mã JavaScript trong thẻ <head> hoặc <body>

Có thể đặt mã JavaScript trong thẻ <head> hoặc <body>. Và nên để tập trung một chỗ cho dễ quản lý.

a) Mã JavaScript trong thẻ <head>

Trong ví dụ dưới, một hàm JavaScript được đặt trong thẻ <head>. Hàm được kích hoạt khi một nút bấm được bấm.

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>
<h1>My Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

b) JavaScript trong thẻ <body>

Trong ví dụ dưới, một hàm JavaScript được đặt trong thẻ <body>. Hàm được kích hoạt khi một nút bấm được bấm.

```
<!DOCTYPE html>
<html>
<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

c) Mã JavaScript có thể để trong file riêng

Mã JavaScript cũng có thể để trong một file riêng biệt, như vậy nó thể được dùng chung trong nhiều trang web khác nhau. File chứa mã JavaScript cần có tên mở rộng là ***.js***.

Để sử dụng mã JavaScript từ file ngoài, hãy khai báo tên file chứa mã trong thuộc tính ***src*** của thẻ <script>:

```
<!DOCTYPE html>
<html>
<body>
<script src="myScript.js"></script>
</body>
</html>
```

Để sử dụng mã JavaScript từ file ngoài, hãy khai báo tên file chứa mã trong thuộc tính ***src*** của thẻ <script>:

3. Javascript Output

JavaScript không có hàm chuyên phụ trách việc in dữ liệu ra.

- Hiển thị trong hộp cảnh báo (alert box), bằng cách sử dụng lệnh ***window.alert()***.
- Hiển thị qua mã HTML bằng cách sử dụng lệnh ***document.write()***.
- Hiển thị qua phần tử HTML, bằng lệnh ***innerHTML***.
- Hiển thị trong cửa sổ console, bằng lệnh *console.log()*.

3.1. Sử dụng lệnh window.alert()

Ví dụ:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
```

```
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

3.2. Sử dụng lệnh document.write()

Ví dụ:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>
</body>
</html>
```

Sử dụng document.*write()*đi sau mã HTML sẽ hóa hoàn toàn các mã HTML:

Ví dụ:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<button onclick="document.write(5 + 6)">Try it</button>
</body>
</html>
```

Chú ý: phương thức.*write()*chỉ sử dụng trong khi gỡ lỗi.

3.3. Sử dụng lệnh innerHTML

To access an HTML element, JavaScript can use the document.getElementById(id) method.

The id attribute defines the HTML element. The innerHTML property defines the HTML content:

Để truy cập một phần tử HTML, JavaScript có thể sử dụng phương thức *getElementById(id)*.

Thuộc tính *id* xác định phần tử HTML. Thuộc tính *innerHTML* xác định nội dung HTML:

Ví dụ:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
```

```
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

Chú ý: Để hiện thị dữ liệu trong HTML, trong hầu hết các trường hợp bạn sẽ gán giá trị cho thuộc tính *innerHTML*.

3.4. Sử dụng lệnh console.log()

Trong trình duyệt của bạn, bạn có thể sử dụng phương thức *console.log()* để hiển thị dữ liệu.

Kích hoạt giao diện console của trình duyệt bằng phím F12, và chọn "Console" trong menu.

Ví dụ:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

4. Cú pháp JavaScript

4.1. Chương trình JavaScript

Một chương trình máy tính là một danh sách các "lệnh" để máy tính "thực hiện". JavaScript là một ngôn ngữ lập trình. Các lệnh JavaScript được phân cách bằng dấu chấm phẩy.

Ví dụ:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Statements</h1>
<p>Statements are separated by semicolons.</p>
<p>The variables x, y, and z are assigned the values 5, 6, and 11:</p>
<p id="demo"></p>
<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>
</body>
</html>
```

Chú ý: Trong trang HTML, chương trình JavaScript dc thi hành bởi trình duyệt.

4.2. Lệnh, hằng, biến, phép toán, biểu thức trongJavaScript

Lệnh: được tạo thành từ giá trị, phép toán, biểu thức, từ khoá, và chú thích.

JavaScript định nghĩa hai loại giá trị: giá trị cố định và giá trị biến đổi. Giá trị cố định được gọi là hằng. Giá trị biến đổi được gọi là biến.

Hằng:

- Hằng số được ghi có hoặc không có số thập phân: 10.50, 1003.
- Hằng chuỗi là văn bản được viết trong nháy kép hoặc nháy đơn: "John Doe", 'John Doe'

Biến: Trong một ngôn ngữ lập trình, các biến dc sử dụng để lưu giá trị dữ liệu. JavaScript sử dụng từ khóa *var*để định nghĩa biến. Dấu = được sử dụng để gán giá trị cho biến. Ví dụ:

```
var x;
x = 6;
```

Phép toán:

- Phép toán = để gán giá trị cho biến:

```
var x = 5;
var y = 6;
```

- Các phép toán số học: + - * / để tính toán số học:

```
(5 + 6) * 10
```

Biểu thức:

Một biểu thức là một sự kết hợp các giá trị, các biến và các phép toán, và nó sẽ trả ra một giá trị. Ví dụ: 5 * 10 sẽ trả về 50.

Biểu thức có thể chứa giá trị biến: x * 10

Các giá trị có thể thuộc các kiểu khác nhau như là số và chuỗi. Ví dụ, "*John" + " " + "Doe*", trả ra "*John Doe*".

Từ khóa:

Từ khóa JavaScript được dùng xác định hành vi được thi hành. Từ khóa var cho trình duyệt biết cần tạo ra một biến mới.

```
var x = 5 + 6;
var y = x * 10;
```

Chú thích trong JavaScript:

Tất các các lệnh JavaScript đều được thi hành. Các mã viết sau hai dấu sổ chéo // hoặc viết giữa cặp /* và */ là các văn bản chú thích. Các chú thích sẽ bị trình duyệt bỏ qua và không thi hành.

```
var x = 5;   // I will be executed
// var x = 6;   I will NOT be executed
```

Tên trong JavaScript:

Trong JavaScript, các tên được sử dụng để đặt cho các biến (và từ khóa, hàm, nhãn). Quy tắc đặt tên tương tự như nhiều ngôn ngữ lập trình phổ diến khác.

Trong JavaScript, ký tự đầu phải là chữ cái, dấu gạch nối hoặc dấu dollar ($).

JavaScript là ngôn ngữ phân biệt hao thưởng: như vậy hai biến dưới đây là khác nhau:

```
lastName = "Doe";
lastname = "Peterson";
```

Từ khóa *var* chỉ gồm các chữ cái in thường.

Trong lịch sử lập trình, các lập trình viên thường sử dụng 3 cách sau để nối các từ trong một tên biến:

Dấu trừ:first-name, last-name, master-card, inter-city.

Dấu gạch nối:first_name, last_name, master_card, inter_city.

Camel Case:FirstName, LastName, MasterCard, InterCity.

Với JavaScript, người ta thương sử dụng quy tắc "camel case"bắt đầu tên là một chữ cái in thường:firstName, lastName, masterCard, interCity.

Chú ý: Quy tắc dấu trừ không được dùng trong JavaScript. Nó sẽ bị hiểu là phép - số học.

JavaScript sử dụng bộ ký tự Unicode. Nó hỗ trợ hầu hết các ngôn ngữ trên thế giới.

5. Lệnh và chương trình JavaScript

Trong HTML, các lệnh JavaScript được thi hành bởi trình duyệt.

5.1. Lệnh JavaScript

Lệnh sau yêu cầu trình duyệt viết chuỗi "*Hello Dolly*." Trong phần tử HTML có *id = "demo"*:

Ví dụ:

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

5.2. Chương trình JavaScript

Hầu hết các chương trình JavaScript chứa nhiều lệnh JavaScript. Các lệnh được thi hành, từ lệnh một theo thứ tự chúng được viết trong chương trình. Ví dụ: x, y, và z là các giá trị cho trước, và cuối cungff z được hiển thị;:

```
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML = z;
```

Chương trình JavaScript (và lệnh JavaScript) thường được gọi là mã JavaScript.

5.3. Dấu chấm phảy;

Dấu chấm phảy được sử dụng để phân tách các lệnh JavaScript. Hãy thêm một dấu chấm phảy vào cuối mỗi lệnh thi hành:

```
a = 5;
b = 6;
c = a + b;
```

Khi được phân cách bằng dấu chấm phảy, nhiều lệnh được phép viết trên một dòng:

```
a = 5; b = 6; c = a + b;
```

Chý ý: Trên trang web, bạn có thể nhìn thấy các ví dụ không dùng dấu chấm phảy. Việc viết dấu chấm phảy ở cuối mỗi lệnh là không bắt buộc, chỉ là khuyến khích.

5.4. JavaScript và dấu cách

JavaScript bỏ qua các khoảng trống. Bạn có thể thêm các khoảng trống vào mã của mình để làm cho nó dễ đọc. Ví dụ:

```
var person = "Hege";
```

```
var person="Hege";
```

Nên sử dụng các dấu cách để phân tách các phép toán = + - * /. Ví dụ: *var x = y + z*;

5.4. Độ dài các dòng và dấu ngắt dòng trong JavaScript

Để dễ đọc nhất, các lập trình viên thường tránh các dòng mã nhiều hơn 80 ký tự. Nếu một lệnh JavaScript không vừa một dòng, cách tốt nhất là ngắt chúng thành nhiều dòng.

Ví dụ:

```
document.getElementById("demo").innerHTML =
"Hello Dolly.";
```

5.5. Khối lệnh JavaScript

Lệnh JavaScript có thể được nhóm lại với nhau thành khối mã lệnh, bên trong cặp dấu ngoặc nhọn {...}. Mục đích của khối mã là để xác định các lệnh cần được thực hiện cùng nhau.

Một trong những nơi bạn sẽ thấy câu lệnh được nhóm lại với nhau thành khối, là trong hàm JavaScript. Ví dụ:

```
function myFunction() {
    document.getElementById("demo").innerHTML = "Hello Dolly.";
    document.getElementById("myDIV").innerHTML = "How are you?";
}
```

5.6. Từ khóa JavaScript

Các lệnh JavaScript thường bắt đầu với một từ khóa để xác định hành động JavaScript phải được thực hiện. Dưới đây là một danh sách của một số các từ khóa mà bạn sẽ học

Từ khóa	Ý nghĩa
`break`	Kết thúc một lệnh *switch* hoặc một vòng lặp
`continue`	Nhảy ra khỏi một bước lặp và bắt đầu bước lặp mới
`debugger`	Dừng việc thực hiện JavaScript, và gọi (nếu có) hàm gỡ lỗi
`do ... while`	Thực thi và lặp lại khối lệnh, khi nào điều kiện là đúng
`for`	Thực thi và lặp lại khối lệnh, khi nào điều kiện là đúng
`function`	Khai báo một hàm
`if ... else`	Thực thi khối lệnh theo điều kiện
`return`	Thoát khỏi một hàm
`switch`	Thực thi khối lệnh, theo từng trường hợp khác nhau
`try ... catch`	Thực hiện và gỡ rối một khối lệnh
`var`	Khai báo một biến

Chú ý: Từ khóa trong JavaScript là dành riêng keywords không được sử dụng cho mục đích khác.

6. Chú thích trong JavaScript

Chú thích trong JavaScript có thể sử dụng để giải thích mã JavaScript, và để nó dễ đọc hơn.

Chú thích trong JavaScript cũng có thể sử dụng để ngăn chặn việc thực thi, khi kiểm tra mã thay thế.

6.1. Chú thích dòng đơn

Các chú thích theo từng dòng đơn bắt đầu bằng ký hiệu //.

Bất kỳ văn bản nào trên một dòng và đi sau //, sẽ bị bỏ qua (không được thực hiện).

Ví dụ dưới sử dụng một dòng chú thích duy nhất trước mỗi dòng, để giải thích mã:

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Ví dụ dưới sử dụng một bình luận duy nhất ở cuối mỗi dòng, để giải thích mã:

```
var x = 5;      // Declare x, give it the value of 5
var y = x + 2;  // Declare y, give it the value of x + 2
```

6.2. Chú thích nhiều dòng

Các chú thích nhiều dòng nằm giữa cặp /* và */. Ví dụ:

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Chú ý: Chủ yếu là sử dụng chú thích dòng đơn. Khối chú thích thường được sử dụng cho các tài liệu chính thức. Ví dụ:

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Chú thích nhiều dòng dưới đây để vô hiệu hóa một số lệnh:

```
/*
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
*/
```

7. Biến trong JavaScript

7.1. Tên biến

Biến JavaScript là các thùng chứa để lưu trữ các giá trị dữ liệu. Trong ví dụ này, *x, y, z*, là các biến:

```
var x = 5;
var y = 6;
var z = x + y;
```

Tất cả các biến JavaScript phải được xác định với những cái tên khác nhau. Tên biến có thể ngắn (như *x* và *y*), hoặc dài hơn (*age, sum, totalVolume*).

Các quy tắc chung để khai báo biến là:

- Tên có thể chứa các chữ cái, chữ số, dấu gạch dưới và dấu hiệu đô la.
- Tên phải bắt đầu bằng một chữ cái
- Tên cũng có thể bắt đầu với $ và _ (nhưng chúng ta không sử dụng nó trong bài học này)

- Tên là nhạy cảm (y và Y là các biến khác nhau)
- Các từ dành riêng (như các từ khóa JavaScript) không thể được sử dụng để khai báo biến

7.2. Lệnh gán

Trong JavaScript, dấu bằng (=) là một lệnh gán, không phải phép toán so sánh bằng. Điều này là khác nhau với phép toán số học. Lệnh sau đây không có ý nghĩa trong số học: x = x + 5

Tuy nhiên, trong JavaScript, nó nó gán giá trị của x + 5 cho x.

(Nó tính toán giá trị của x + 5 và đặt kết quả vào biến x. Giá trị của x được tăng 5 đơn vị)

Phép toán so sánh bằng được viết là == trong JavaScript.

Phép gán	Ví dụ	Tương đương với
=	x = y	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

7.3. Kiểu dữ liệu

Biến JavaScript có thể lưu giá trị số như 100, và giá trị văn bản như "*John Doe*". Trong chương trình, giá trị văn bản được gọi là chuỗi văn bản.

JavaScript có thể xử lý nhiều loại dữ liệu, nhưng bây giờ, ta quan tâm đến số và chuỗi.

Chuỗi được viết bên trong dấu nháy kép hoặc nháy đơn. Các số được viết mà không có dấu nháy. Nếu bạn đặt dấu nháy kép quanh một số, nó sẽ được coi như là một chuỗi văn bản. Ví dụ:

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

Các biến JavaScript có thể chứa nhiều kiểu dữ liệu: số, chuỗi, mảng, đối tượng, ...:

```
var length = 16;                               // Number
var lastName = "Johnson";                      // String
var cars = ["Saab", "Volvo", "BMW"];           // Array
var x = {firstName:"John", lastName:"Doe"};    // Object
```

Quan niệm về kiểu dữ liệu

Trong lập trình, kiểu dữ liệu rất quan trọng. Để có thể tính toán trên các biến, ta cần xác định được kiểu của nó. Không có kiểu dữ liệu, máy tính không thể giải quyết hoàn toàn bài toán:

```
var x = 16 + "Volvo";
```

Kết quả của x là bao nhiêu? Có lỗi không? JavaScript sẽ đối xử với ví dụ trên là:

```
var x = "16" + "Volvo";
```

JavaScript tính toán giá trị biểu thức từ trái sang phải. Ví dụ:

```
var x = 16 + 4 + "Volvo"; // x = 20Volvo
```

Và với biểu thức dưới đây:

```
var x = "Volvo" + 16 + 4; // x = Volvo164
```

JavaScript chấp nhận xử liệu động

JavaScript có kiểu dữ liệu động. Cùng một biến có thể chứa nhiều kiểu dữ liệu khác nhau:

```
var x;               // Now x is undefined
var x = 5;           // Now x is a Number
var x = "John";      // Now x is a String
```

Chuỗi trong JavaScript

Một chuỗi là một dãy các ký tự như "John Doe". Các chuỗi được viết trong cặp dấu nháy kép. Bạn có thể dùng cặp nháy đơn. Ví dụ:

```
var carName = "Volvo XC60";   // Dùng nháy kép
var carName = 'Volvo XC60';   // Dùng nháy đơn
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Bạn có thể sử dụng dấu nháy kép bên trong một chuỗi, miễn là nó giống với dấu nháy bao bọc chuỗi. Nếu bao bọc chuỗi bằng nháy đơn thì bạn có thể chứa nháy kép bên trong và ngược lại.

Ví dụ:

```
var answer = "It's alright";
var answer = "He is called 'Johnny'";
var answer = 'He is called "Johnny"';
```

Kiểu số trong JavaScript

JavaScript chỉ có một kiểu số. Các số có thể được viết với dấu thập phân hoặc không. Ví dụ:

```
var x1 = 34.00;     // có dấu thập phân
var x2 = 34;        // không có dấu thập phân
```

Các số đều có thể viết dưới dạng dấu chấm động:

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```

Kiểu Boolean trong JavaScript

Kiểu boolean chỉ có hai giá trị: *true*hoặc *false*.

Ví dụ:

```
var x = true;
var y = false;
```

Kiểu boolean thường chỉ được dùng đển kiểm tra điều kiện.

Một giá trị boolean biểu diễn cho một trong hai giá trị: *true* hoặc *false*. Thông thường, trong lập trình, bạn sẽ cần một kiểu dữ liệu mà chỉ có thể có một trong hai giá trị, như: YES / NO; ON / OFF; TRUE / FALSE. JavaScript có kiểu dữ liệu Boolean để biểu diễn chúng.

- Hàm *Boolean()*:

Bạn có thể sử dụng hàm *Boolean()*để xác định một biểu thức (hoặc một biến) là đúng.

```
Boolean(10 > 9)        // returns true
```

Thậm chí dễ hơn:

```
(10 > 9)              // also returns true
10 > 9                // also returns true
```

- So sánh và điều kiện: sẽ có bài học chi tiết hơn, dưới đây là một số ví dụ:

Phép toán	**Mô tả**	**Ví dụ**
==	equal to	if (day == "Monday")
>	greater than	if (salary > 9000)
<	less than	if (age < 18)

Chú ý: Giá trị Boolean của một biểu thức là điều cơ bản để so sánh và làm điều kiện so sánh. Tất cả mọi thứ với một giá trị "Thực" là *true*. Ví dụ:

```
100
3.14
-15
"Hello"
"false"
7 + 1 + 3.14
5 < 6
```

Tất cả mọi thứ không có thực là False: giá trị 0, -0, chuỗi rỗng "", *null*, *NaN* là *false*.

Mảng trong JavaScript

Mảng trong JavaScript được viết với dấu ngoặc vuông. Các phần tử của mảng được phân cách bằng dấu phẩy. Mã sau đây khai báo một mảng gọi là cars, chứa ba mục (tên xe). Ví dụ:

```
var cars = ["Saab", "Volvo", "BMW"];
```

Chỉ số mảng được bắt đầu từ 0, phần tử đầu tiên là [0], phần tử thứ hai [1], v.v...

Đối tượng trong JavaScript

Đối tượng JavaScript được viết với dấu ngoặc nhọn. Các thuộc tính của đối tượng được viết kiểu cặp *name:value* , cách nhau bởi dấu phẩy. Ví dụ:

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

Đối tượng *person*trong ví dụ trên có 4 thuộc tính: *firstName, lastName, age, and eyeColor*.

Phép toán typeof

Bạn có thể sử dụng phép toán typeof để xác định kiểu của biến JavaScript. Ví dụ:

```
typeof "John"                 // Returns string
typeof 3.14                   // Returns number
typeof false                  // Returns boolean
typeof [1,2,3,4]              // Returns object
typeof {name:'John', age:34} // Returns object
```

Chú ý: Trong JavaScript, mảng là kiểu đối tượng đặc biệt. Như vậy *typeof [1,2,3,4]*trả về là *object*.

Kiểu Undefined

Trong JavaScript, một biến không có giá trị là biến có kiểu không xác định. Và *typeof*của nó cũng là *undefined*. Ví dụ:

```
var person;          // giá trị và kiểu là không xác định
```

Một biến có thể được gán kiểu *undefined*. Và kiểu của nó sẽ là không xác định. Ví dụ:

```
person = undefined; // giá trị và kiểu là không xác định
```

Giá trị rỗng

Một giá trị rỗng khác với không xác định. Một biến chuỗi rỗng có giá trị và có kiểu.

```
var car = ""; // Giá trị là "", và typeof là string
```

Giá trị Null

Trong JavaScript có giá trị "nothing". Nó là một cái gì đó không tồn tại. Kiểu của *null*là *object*. Hãy gán cho một đối tượng rỗng giá trị *null.* Ví dụ:

```
var person = null; // Giá trị null, kiểu là object
```

Bạn cũng có thể làm rỗng một biến đối tượng bằng cách gán cho nó giá trị *undefined*.

```
var person = undefined; // Giá trị là undefined, kiểu là undefined
```

Sự khác nhau giữa Undefined và Null

```
typeof undefined              // undefined
typeof null                   // object
null === undefined            // false
null == undefined             // true
```

7.4. Khai báo biến trong JavaScript

Tạo một biến trong JavaScript được gọi là khai báo biến. Bạn khai báo một biến JavaScript với từ khóa *var*:

```
var carName;
```

Sau khi khai báo, biến không có giá trị. (Về mặt kỹ thuật nó có giá trị không xác định)

Để gán một giá trị cho biến, ta sử dụng dấu bằng. Trong ví dụ dưới đây, chúng ta tạo ra một biến gọi là carName và gán giá trị "Volvo" với nó. Sau đó chúng ta "in ra" giá trị bên trong một phần tử HTML với *id = "demo":*

Ví dụ:

```
<p id = "demo"></ p>
<script>
var carName = "Volvo";
document.getElementById ("demo") innerHTML = carName.
</ script>
```

Chú ý: Tốt nhất là khai báo tất cả các biến ở đầu của một kịch bản.

7.5. Khai báo cùng lúc nhiều biến

Có thể khai báo nhiều biến trong một lệnh. Bắt đầu lệnh với *var* và tách các biến bằng dấu phảy:

```
var person = "John Doe", carName = "Volvo", giá = 200;
```

Một lệnh có thể sử dụng nhiều dòng:

```
var person = "John Doe",
carName = "Volvo",
price = 200;
```

7.6. Biến có giá trị không xác định

Trong chương trình máy tính, các biến thường được khai báo không có giá trị. Giá trị có thể là cái gì đó đã được tính toán, hoặc cái gì đó sẽ được cung cấp sau, như người dùng nhập vào.

Một biến được khai báo mà không có giá trị sẽ có giá trị không xác định. Biến *carName* có giá trị không xác định sau khi thực hiện câu lệnh:

```
var carName;
```

7.7. Khai báo lại biến

Nếu bạn khai báo lại một biến JavaScript, nó sẽ không mất đi giá trị đang có. Biến *carName* vẫn sẽ có giá trị "*Volvo*" sau khi thực hiện các lệnh sau:

```
var carName = "Volvo";
```

```
var carName;
```

7.8. Bài tập

1. Tạo biến *carName*, gán giá trị "*Volvo*" cho nó và in nó ra màn hình

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
// Create the variable here
</script>
</body>
</html>
```

2. In ra tổng 5 + 10, sử dụng hai biến x, y:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
// Create the variables here
</script>
</body>
</html>
```

3. Khai báo biến z = x + y và in ra giá trị của z?

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
var x = 5;
var y = 10;
</script>
</body>
</html>
```

4. Dùng một từ khóa *var* tạo ra 3 biến với giá trị: firstName = "John" , lastName = "Doe" , age = 35

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
// Create the variables here
document.getElementById("demo").innerHTML =
firstName + " " + lastName + " is " + age;
</script>
```

```
</body>
</html>
```

8. Số học trong JavaScript

Bạn có thể làm phép tính với các biến JavaScript, sử dụng phép toán như = và +. Ví dụ:

```
var x = 5 + 2 + 3;
```

Bạn cũng có thể cộng chuỗi, nhưng chuỗi sẽ được nối thêm. Ví dụ:

```
var x = "John" + "" + "Doe";
```

Ví dụ khác về cộng chuỗi:

var x = "5" + 2 + 3; // *x = "523"*

Chú ý: Nếu bạn cộng thêm một số vào một chuỗi, số đó sẽ được đối xử như chuỗi, và được nối vào cuối chuỗi.

Phép toán	Mô tả
+	Cộng
-	Trừ
*	Nhân
/	Chia
%	Lấy dư
++	Tự tăng
--	Tự giảm

8.1.Các phép toán so sánh

Phép toán	Ý nghĩa
==	so sánh bằng
===	so sánh bằng giá trị và bằng kiểu
!=	so sánh khác
!==	không bằng giá trị hoặc không bằng kiểu
>	lớn hơn
<	nhỏ hơn
>=	lớn hơn hoặc bằng
<=	nhỏ hơn hoặc bằng

8.2. Chi tiết về số học trong JavaScript

a) Số trong JavaScript luôn là số dấu chấm động 64-bit

Không giống như nhiều ngôn ngữ lập trình khác, JavaScript không xác định các loại số khác nhau như số nguyên ngắn, dài, dấu chấm động, v.v...

Số JavaScript luôn được lưu trữ như số dấu chấm động có độ chính xác kép, theo tiêu chuẩn IEEE 754 quốc tế.

Các số đều có độ lớn là 64 bit, trong đó phần số được lưu trữ trong các bit 0-51, số mũ trong bit 52-62, và dấu trong bit 63:

Giá trị	Mũ	Dấu
52 bits (0 - 51)	11 bits (52 - 62)	1 bit (63)

b) Độ chính xác của số

- Độ chính xác của số nguyên lên đến 15 chữ số.

```
var x = 999999999999999;   // x will be 999999999999999
```

```
var y = 9999999999999999;  // y will be 10000000000000000
```

- Số thập phân có số lượng số thập phân tối đa là 17, nhưng số dấu chấm động không luôn chính xác 100%. Ví dụ:

```
var x = 0.2 + 0.1;         // x xấp xỉ 0.30000000000000004
```

Để giải quyết vấn đề trên, ta làm như sau:

```
var x = (0.2 * 10 + 0.1 * 10) / 10;   // x là 0.3
```

c) Cơ số 16:

JavaScript coi một số trong cơ số 16 nếu nói số đó đi theo sau *0x*. Ví dụ:

```
var x = 0xFF;             // x will be 255
```

Chú ý: Không bao giờ viết số 0 ở đầu (như là 07). Một số phiên bản JavaScript sẽ coi số đó là cơ số 8. Mặc định, Javascript hiển thị các số ở cơ số 10. Bạn có thể sử dụng phương thức *toString()*để hiển thị số ở cơ số 16 (hex), cơ số 8 (octal), hoặccơ số 2 (binary). Ví dụ:

```
var myNumber = 128;
myNumber.toString(16);     // returns 80
myNumber.toString(8);      // returns 200
myNumber.toString(2);      // returns 10000000
```

d) Giá trị vô cực và âm vô cực

Giá trị vô cực và âm vô cực là giá trị JavaScript sẽ trả về nếu bạn tính toán một số mà kết quả vượt quá khả năng lưu trữ của nó. Ví dụ:

```
var myNumber = 2;
while (myNumber != Infinity) {          // Execute until Infinity
    myNumber = myNumber * myNumber;
}
```

Phép chia cho 0 cũng sinh ra giá trị vô cực:

```
var x =  2 / 0;          // x will be Infinity
var y = -2 / 0;          // y will be -Infinity
```

Vô cực là một số, vì vậy *typeOf Infinity* sẽ trả về là *number*.

e) NaN không phải một số

NaN là từ dành riêng trong JavaScript cho biết rằng một giá trị không phải số. Việc cố gắng tính toán số học với một chuỗi không phải số sẽ trả về một kết quả *NaN* (Not a Number). Ví dụ:

```
var x = 100 / "Apple";  // x will be NaN (Not a Number)
```

Tuy nhiên, nếu chuỗi chứa một giá trị số thì kết quả sẽ là số. Ví dụ:

```
var x = 100 / "10";     // x will be 10
```

Bạn có thể dùng hàm toàn cục *isNaN()*để kiểm tra một giá trị có là số hay không. Ví dụ:

```
var x = 100 / "Apple";
isNaN(x);     // returns true because x is Not a Number
```

Nếu bạn dùng NaN trong một phép toán toán học, kết quả trả về sẽ là NaN. Ví dụ:

```
var x = NaN;
var y = 5;
var z = x + y;          // z will be NaN
```

Hoặc kết quả có thể là phép nối chuỗi. Ví dụ:

```
var x = NaN;
var y = "5";
```

```
var z = x + y;          // z will be NaN5
```

Kiểu của *NaN*là *number*.

```
typeof NaN;              // returns "number"
```

f) Số có thể là đối tượng

Thông thường số trong JavaScript là giá trị nguyên thủy được tạo ra từ các chữ: *var x = 123*. Nhưng số cũng có thể được định nghĩa là các đối tượng với các từ khóa *new*: *var y = new Number (123)*. Ví dụ:

```
var x = 123;
var y = new Number(123);
// typeof x returns number
// typeof y returns object
```

Chú ý: Không tạo các đối tượng Number. Chúng làm chậm tốc độ thực thi, và các tác dụng phụ khó chịu. Như là khi, sử dụng các toán tử so sánh bằng ==, các số bằng nhau là giốn nhau. Ví dụ:

```
var x = 500;
var y = new Number(500);
// (x == y) bằng nhau
```

Khi dùng phép so sánh bằng ===, thì chúng không bằng nhau vì kiểu của chúng khác nhau.

```
var x = 500;
var y = new Number(500);
// (x === y) khác nhau
```

Thậm chú các đối tượng là không thể so sánh được:

```
var x = new Number(500);
var y = new Number(500);
// (x == y) là false vì không thể so sánh đối tượng.
```

g) Thuộc tính và phương thức của kiểu số

Giá trị nguyên thủy (như 3.14 hay 2014), không thể có các thuộc tính và phương thức (vì chúng không phải là đối tượng). Nhưng với JavaScript, phương thức và thuộc tính có sẵn cho giá trị nguyên thủy, vì JavaScript xử lý giá trị nguyên thủy như các đối tượng khi thực hiện phương thức và thuộc tính của chúng.

Thuộc tính	Mô tả
MAX_VALUE	Trả về giá trị số lớn nhất
MIN_VALUE	Trả về giá trị số nhỏ nhất
NEGATIVE_INFINITY	Biểu diễn số âm vô vực
NaN	Biểu diễn giá trị không phải số
POSITIVE_INFINITY	Biểu diễn số vô cực

```
var x = Number.MAX_VALUE;
```

Thuộc tính số thuộc về đối tượng số của JavaScript được gọi là *Number*. Các thuộc tính này chỉ có thể được truy cập như *Number.MAX_VALUE*. Việc sử dụng *myNumber.MAX_VALUE*, trong đó *myNumber* là một biến, biểu thức, hoặc giá trị, sẽ trả về giá trị không xác định. Ví dụ:

```
var x = 6;
var y = x.MAX_VALUE;    // y becomes undefined
```

8.3. Các phương thức số phổ biến

8.3.1. Các phương thức toàn cục

Các hàm JavaScript toàn cục có thể được sử dụng trên tất cả các kiểu dữ liệu

Phương thức	Mô tả
Number()	Trả về một số bằng cách chuyển đổi từ đối số của hàm
parseFloat()	Phân tích đối số và trả về một số dấu chấm động.
parseInt()	Phân tích đối số và trả về một số nguyên

8.3.2. Các phương thức số

Các phương thức số trong JavaScript có thể sử dụng với mọi số:

Phương thức	Mô tả
toString()	Trả về một chuỗi từ đối số là số
toExponential()	Trả về một chuỗi, với số chữ số thập phân được làm tròn và viết theo kiểu dấu chấm động.
toFixed()	Trả về một chuỗi, với số chữ số thập phân được làm tròn và viết theo kiểu dấu chấm tĩnh.
toPrecision()	Trả về một chuỗi, với số chữ số được xác định.
valueOf()	Trả về một số từ một số hoặc một biểu thức số

Ví dụ:

```
var x = 123;
x.toString();            // returns 123 from variable x
(123).toString();        // returns 123 from literal 123
(100 + 23).toString();   // returns 123 from expression 100 + 23
var x = 9.656;
x.toExponential(2);     // returns 9.66e+0
x.toExponential(4);     // returns 9.6560e+0
x.toExponential(6);     // returns 9.656000e+0
x.toFixed(0);           // returns 10
x.toFixed(2);           // returns 9.66
x.toFixed(4);           // returns 9.6560
x.toFixed(6);           // returns 9.656000
x.toPrecision();        // returns 9.656
x.toPrecision(2);       // returns 9.7
x.toPrecision(4);       // returns 9.656
x.toPrecision(6);       // returns 9.65600
x = true;
Number(x);        // returns 1
x = false;
Number(x);        // returns 0
x = new Date();
Number(x);        // returns 1404568027739
x = "10"
Number(x);        // returns 10
x = "10 20"
Number(x);        // returns NaN
```

```
parseInt("10");          // returns 10
parseInt("10.33");       // returns 10
parseInt("10 20 30");    // returns 10
parseInt("10 years");    // returns 10
parseInt("years 10");    // returns NaN
parseFloat("10");         // returns 10
parseFloat("10.33");      // returns 10.33
parseFloat("10 20 30");   // returns 10
parseFloat("10 years");   // returns 10
parseFloat("years 10");   // returns NaN
var x = 123;
x.valueOf();             // returns 123 from variable x
(123).valueOf();         // returns 123 from literal 123
(100 + 23).valueOf();    // returns 123 from expression 100 + 23
```

8.4. Bài tập

1. Tạo biến *myNumber*, gán giá trị 50 cho nó và in nó ra.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
// Create the variable here
</script>
</body>
</html>
```

2. Giá trị của z phải là 11, nhưng vì sao nó không in ra như vậy, hãy sửa chữa nó

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var x = 5;
var y = "6";
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>
</body>
</html>
```

3. Hãy chia x cho 3.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
```

```
var x = 50;
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

4. Hãy in ra tích 8 * 5, sử dụng hai biến x và y.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
// Create the variables here
</script>
</body>
</html>
```

9. Hàm trong JavaScript

Một hàm JavaScript là một khối mã được thiết kế để thực hiện một nhiệm vụ cụ thể. Hàm Javascript được thực hiện khi "cái gì đó" gọi nó. Ví dụ:

```
function myFunction(p1, p2) {
    return p1 * p2;
}
```

9.1. Cú pháp hàm JavaScript

Một hàm JavaScript được định nghĩa với từ khóa *function*, tiếp theo là một tên, theo sau là cặp dấu ngoặc ().

Tên hàm có thể chứa các chữ cái, chữ số, dấu gạch dưới và ký hiệu đô la (tương tự như quy tắc tên biến).

Dấu ngoặc đơn có thể gồm tên tham số cách nhau bằng dấu phẩy: (*argument1, argument2, ...*)

Mã lệnh được thực thi bởi hàm, được đặt bên trong dấu ngoặc nhọn: {}

```
function name(parameter1, parameter2, parameter3) {
mã lệnh thi hành;
}
```

a) Tham số của hàm

Các tham số hàm là tên được liệt kê trong định nghĩa hàm. Tham số hàm là những giá trị thực tế hàm nhận được khi nó được gọi. Bên trong hàm, các tham số được cư xử như các biến địa phương. Lưu ý, hàm giống như thủ tục hoặc chương trình con, trong ngôn ngữ lập trình khác.

b) Gọi hàm

Các mã bên trong hàm sẽ được thực hiện khi "cái gì đó" gọi hàm:

- Khi một sự kiện xảy ra (ví dụ, người dùng nhấp chuột vào một nút)
- Khi nó được gọi (gọi là) từ một lệnh JavaScript khác
- Tự động (tự gọi)

c) Giá trị trả về của hàm

Khi JavaScript gặp lệnh *return*, hàm dừng thực hiện. Nếu hàm được gọi từ một lệnh, JavaScript sẽ "trở về" để thực thi mã sau mã gọi nó.

Hàm thường tính toán một giá trị trả về. Giá trị trả về là "trả lại" cho "cái gọi nó". Ví dụ:

```
var x = myFunction(4, 3); // Gọi hàm và trả giá trị về cho x là 12
function myFunction(a, b) {
    return a * b;     // trả về giá trị a*b
}
```

9.2. Vì sao phải dùng hàm?

Bạn có thể sử dụng lại mã: Viết mã một lần, và sử dụng nó nhiều lần. Bạn có thể sử dụng mã nhiều với các đối số khác nhau, để tạo ra kết quả khác nhau. Ví dụ: Đổi độ F^0 sang độ C^0

```
function toC(f) {
    return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toC(77);
```

a) Phép toán () gọi hàm

Trong ví dụ trên, *toCelsius* trỏ đến đối tượng hàm, và *toCelsius()* trỏ đến đến kết quả hàm. Ví dụ, truy cập vào một hàm mà không có () sẽ trả về định nghĩa hàm:

```
function toCelsius(fahrenheit){
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

b) Sử dụng hàm như biến

Trong JavaScript, bạn có thể sử dụng các hàm như cách bạn sử dụng các biến. Ví dụ

```
var text = "The temperature is " + toCelsius(77) + " Celsius";
```

Thay cho:

```
var x = toCelsius(32);
var text = "The temperature is " + x + " Celsius";
```

9.3. Bài tập

1. Hãy gọi hàm

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Hello World!";
}
// Gọi hàm ở đây
</script>
</body>
</html>
```

2. Tìm ra những gì là sai với hàm, sửa chữa nó và chạy nó!

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
func myFunc {
    document.getElementById("demo").innerHTML = "Hello World!";
}
myFunction();
</script>
</body>
</html>
```

3. Sử dụng hàm để hiển thị giá trị 5 * 5.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
function myFunction() {
    // viết mã vào đây
}
document.getElementById("demo").innerHTML = myFunction();
</script>
</body>
</html>
```

4. Sử dụng hàm để hiển thị chuỗi "*Hello John*".

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Hiển thị kết quả ở đây.</p>
<script>
function myFunction(name) {
    return "Hello " + name;
}
</script>
</body>
</html>
```

5. Định nghĩa hàm "*myFunction*", và làm cho nó in ra chuỗi "*Hello World!*" trong phần tử <p>.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"> Hiển thị kết quả ở đây.</p>
<script>
// định nghĩa và gọi hàm ở đây
```

```
</script>
</body>
</html>
```

10. Đối tượng trong JavaScript

Các đối tượng trong đời thực luôn có thuộc tính và hành vi. Ví dụ, một chiếc xe hơi ở ngoài đời là một đối tượng. Xe có các thuộc tính và hành vi:

Thuộc tính của đối tượng	Hành vi của đối tượng
car.name = Fiat	car.start()
car.model = 500	car.drive()
car.weight = 850kg	car.brake()
car.color = white	car.stop()

Tất cả các xe ô tô có thuộc tính tương tự, nhưng giá trị thuộc tính khác nhau. Tất cả các xe có các hành vi tương tự, nhưng các hành vi này được thực hiện vào những thời điểm khác nhau.

10.1. Đối tượng trong JavaScript

Bạn đã biết các biến JavaScript là thùng chứa cho giá trị dữ liệu. Mã này gán một giá trị đơn giản (*Fiat*) cho một biến có tên *car*:

```
var car = "Fiat";
```

Các đối tượng có nhiều biến hơn và có thể chứa nhiều giá trị hơn. Mã sau gán nhiều giá trị cho biến có tên *car*:

```
var car = {type:"Fiat", model:500, color:"white"};
```

Các giá trị được viết theo cặp *name:value*cách nhau dấu hai chấm. Các đối tượng JavaScript objects là các thùng chứa các giá trị có tên.

10.1.1. Các thuộc tính của đối tượng

Các cặp*name:values*trong đối tượng JavaScript được gọi là các thuộc tính.

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

Thuộc tính	Giá trị thuộc tính
firstName	John
lastName	Doe
age	50
eyeColor	blue

10.1.2. Các phương thức của đối tượng

Các hành vi (phương thức) là những hành động có thể được thực hiện trên đối tượng. Các phương thức được lưu trữ trong các thuộc tính như các định nghĩa hàm.

Thuộc tính	Giá trị thuộc tính
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Chú ý: Các đối tượng JavaScript là các thùng chứa các thuộc tính và phương thức.

10.1.3. Định nghĩa đối tượng

Bạn định nghĩa một đối tượng JavaScript như một hằng đối tượng. Ví dụ:

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

Các dấu cách, dòng trống không quan trọng.Một định nghĩa đối tượng có thể gồm nhiều dòng. *Ví dụ*:

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

10.1.4. Truy xuất thuộc tính đối tượng

Bạn có thể truy xuất thuộc tính đối tượng theo 2 cách:

```
objectName.propertyName
```

hoặc

```
objectName["propertyName"]
```

Ví dụ 1:

```
person.lastName;
```

Ví dụ 2:

```
person["lastName"];
```

10.1.5. Truy xuất hành vi đối tượng

Bạn có thể truy xuất hành vi đối tượng như sau:

```
objectName.methodName()
```

Ví dụ:

```
name = person.fullName();
```

Nếu bạn truy xuất thuộc tính *fullName*, không có (), nó sẽ trả về định nghĩa hàm:

Ví dụ:

```
name = person.fullName;
```

Chú ý: Không khai báo Strings, Numbers, và Booleans như các đối tượng!

Khi một biến JavaScript được khai báo với từ khóa *new*, biến đó sẽ được tạo ra như một đối tượng:

```
var x = new String();        // Declares x as a String object
var y = new Number();        // Declares y as a Number object
var z = new Boolean();       //Declares z as a Boolean object
```

Nên tránh các đối tượng String, Number, và Boolean. Chúng sẽ được biên dịch và làm giảm tốc độ thi hành mã của bạn.

10.2. Đối tượng Math

- Các đối tượng Math cho phép bạn thực hiện các nhiệm vụ toán học trên các kiểu số.
- Các đối tượng Math bao gồm một số các phương thức toán học.

10.2.1. Tạo số ngẫu nhiên, lấy cực đại, cực tiểu

```
Math.random();       // returns a random number
Math.min() and Math.max()
```

Math.min() và *Math.max()*có thể dùng để trả về giá trị lớn nhất hoặc nhỏ nhất trong danh sách đối số. Ví dụ:

```
Math.min(0, 150, 30, 20, -8, -200);      // returns -200
Math.max(0, 150, 30, 20, -8, -200);      // returns 150
Math.random()
```

*Math.random()*trả về số ngẫu nhiên trong nửa đoạn [0 ,1).

10.2.2. Làm tròn

Gồm các hàm *Math.round(), Math.ceil(), Math.floor()*

```
Math.round(4.7);            // returns 5
Math.round(4.4);            // returns 4
```

- *Math.ceil()* làm tròn lên số nguyên gần nhất.

```
Math.ceil(4.4);             // returns 5
```

- *Math.floor()*làm tròn xuống đến số nguyên gần nhất.

```
Math.floor(4.7);            // returns 4
```

*Math.floor() và Math.random()*có thể dùng cùng nhau để trả về số nguyên trong khoảng 0 và 10, như sau:

```
Math.floor(Math.random() * 11);
```

10.2.3. Hằng toán học

```
Math.E          // returns cơ số của logarit tự nhiên
Math.PI         // returns PI
Math.SQRT2      // returns the square root of 2
Math.SQRT1_2    // returns the square root of 1/2
Math.LN2        // returns the natural logarithm of 2
Math.LN10       // returns the natural logarithm of 10
Math.LOG2E      // returns base 2 logarithm of E
Math.LOG10E     // returns base 10 logarithm of E
```

10.2.4. Các phương thức của đối tượng Math

Phương thức	Mô tả
`abs(x)`	Returns the absolute value of x
`acos(x)`	Returns the arccosine of x, in radians
`asin(x)`	Returns the arcsine of x, in radians
`atan(x)`	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
`atan2(y,x)`	Returns the arctangent of the quotient of its arguments
`ceil(x)`	Returns x, rounded upwards to the nearest integer
`cos(x)`	Returns the cosine of x (x is in radians)
`exp(x)`	Returns the value of Ex
`floor(x)`	Returns x, rounded downwards to the nearest integer
`log(x)`	Returns the natural logarithm (base E) of x
`max(x,y,z,...,n)`	Returns the number with the highest value
`min(x,y,z,...,n)`	Returns the number with the lowest value
`pow(x,y)`	Returns the value of x to the power of y
`random()`	Returns a random number between 0 and 1
`round(x)`	Rounds x to the nearest integer

`sin(x)`	Returns the sine of x (x is in radians)
`sqrt(x)`	Returns the square root of x
`tan(x)`	Returns the tangent of an angle

10.3. Bài tập

1. Hiển thị chuỗi "John" bằng cách lấy thông tin từ đối tượng *person*.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
var person = {firstName:"John", lastName:"Doe"};
</script>
</body>
</html>
```

2. Thêm thuộc tính và giá trị sau đây vào đối tượng person: *country: USA*

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var person = {firstName:"John", lastName:"Doe"};
document.getElementById("demo").innerHTML = person.country;
</script>
</body>
</html>
```

3. Tạo một đối tượng person với name = John, age = 50. Sau đó, truy xuất đối tượng để hiển thị "*John is 50 years old*".

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
// Create the object here
</script>
</body>
</html>
```

4. Sử dụng phương thức *random()* để in ra một số ngẫu nhiên

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
// Insert the Math method here
</script>
</body>
```

```
</html>
```

5. Sửa mã sau để hiển thị giá trị lớn nhất trong danh sách các số đã cho.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
document.getElementById("demo").innerHTML = Math(0, 150, 30, 20, -8);
</script>
</body>
</html>
```

6. Làm tròn "5.3" đến số nguyên gần nhất và in nó ra

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
document.getElementById("demo").innerHTML = Math.round(5.3);
</script>
</body>
</html>
```

7. In ra giá trị căn bậc hai của "9".

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
// Insert the Math method here
</script>
</body>
</html>
```

11. Phạm vi trong JavaScript

Phạm vi là tập các biến mà bạn có thể truy xuất

Trong JavaScript, đối tượng và hàm cũng là các biến. Phạm vi là tập các biến, đối tượng, và các hàm bạn có thể truy cập. JavaScript có phạm vi hàm: Phạm vi thay đổi trong các hàm.

11.1. Các biến cục bộ

Các biến được khai báo trong một hàm JavaScript, trở thành cục bộ với hàm đó. Biến cục bộ có phạm vi địa phương: Chúng chỉ có thể được truy cập trong hàm.

Ví dụ:

```
function myFunction() {
    var carName = "Volvo";
    // mã ở đây có thể sử dụng carName
}
```

Vì các biến cục bộ chỉ được nhìn thấy bên trong hàm của nó, các biến có cùng tên có thể được sử dụng trong các hàm khác nhau.

Biến cục bộ được tạo ra khi một hàm bắt đầu, và bị xóa khi hàm kết thúc công việc.

11.2. Biến toàn cục

Một biến được khai báo bên ngoài một hàm, trở thành toàn cục. Một biến toàn cục có phạm vi toàn cục: Tất cả các kịch bản và các hàm trong một trang web có thể truy cập nó. Ví dụ:

```
var carName = " Volvo";
// mã ở đây có thể dùng carName
function myFunction() {
    // mã ở đây có thể dùng carName
}
```

11.3. Biến toàn cục tự động

Nếu bạn gán một giá trị cho một biến mà đã không được khai báo, nó sẽ tự động trở thành một biến toàn cục. Ví dụ mã dưới sẽ khai báo *carName* như một biến toàn cục, thậm chí nếu nó được thực hiện bên trong một hàm. Ví dụ:

```
// code here can use carName
function myFunction() {
    carName = "Volvo";
    // code here can use carName
}
```

11.4. Tuổi đời của biến

Tuổi đời của một biến JavaScript bắt đầu khi nó được khai báo. Biến cục bộ được xóa khi hàm hoàn thành nhiệm vụ. Biến toàn cục sẽ bị xóa khi bạn đóng trang.

11.5. Đối số hàm

Đối số hàm (tham số) làm việc như là các biến cục bộ bên trong các hàm.

11.6. Biến toàn cục trong HTML

Với JavaScript, phạm vi toàn cục là hoàn toàn trong môi trường JavaScript.Trong HTML, phạm vi toàn cục là các đối tượng cửa sổ: Tất cả các biến toàn cục thuộc về đối tượng cửa sổ.

Ví dụ

```
function myFunction() {
    carName = "Volvo";
}
```

12. Sự kiện JavaScript

Các sự kiện HTML "thứ" xảy ra với các phần tử HTML. Khi JavaScript được sử dụng trong các trang HTML, JavaScript có thể "phản ứng" với những sự kiện này.

12.1. Sự kiện HTML

Một sự kiện HTML có thể là một điều gì đó mà trình duyệt thực hiện, hoặc một cái gì đó mà người dùng thực hiện. Dưới đây là một số ví dụ về các sự kiện HTML:

- Một trang web HTML đã tải xong
- Một trường nhập dữ liệu HTML đã được thay đổi
- Một nút HTML được nhấp

Thông thường, khi sự kiện xảy ra, bạn có thể muốn làm một cái gì đó. JavaScript cho phép bạn thực thi mã khi sự kiện này được phát hiện. HTML cho phép xử lý thuộc tính sự kiện, với mã JavaScript, có thể được thêm vào các phần tử HTML.

Với dấu nháy đơn:

```
<some-HTML-element some-event='some JavaScript'>
```

Với dấu nháy kép:

```
<some-HTML-element some-event="some JavaScript">
```

Ví dụ dưới, một thuộc tính onclick (với mã), được thêm vào phần tử *button*:

```
<button onclick='getElementById("demo").innerHTML=Date()'> The time is?
</button>
```

Trong ví dụ trên, mã JavaScript thay đổi nội dung của phần tử với *id = "demo"*. Trong ví dụ sau, mã thay đổi nội dung của chính phần tử đó (bằng cách sử dụng *this.innerHTML*):

```
<button onclick="this.innerHTML=Date()"> The time is?</button>
```

Chú ý: mã JavaScript thường có vài dòng, thường có dạng phổ biến sau:

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display the date.</p>
<button onclick="displayDate()">The time is?</button>
<script>
function displayDate() {
    document.getElementById("demo").innerHTML = Date();
}
</script>
<p id="demo"></p>
</body>
</html>
```

12.2. Các sự kiện HTML phổ biến

Sự kiện	Mô tả
onchange	Một phần tử HTML bị thay đổi
onclick	Người dùng bấm chuột vào một phần tử HTML
onmouseover	Người dùng rê chuột qua một phần tử HTML
onmouseout	Người dùng rê chuột ra khỏi một phần tử HTML
onkeydown	Người dùng nhấn một phím
onload	Trình duyệt nạp xong một trang.

12.3. Mã JavaScript có thể làm gì?

Điều khiển sự kiện có thể được sử dụng để xử lý, và xác minh, người dụng nhập liệu, hành động của người dùng, và các hành động trình duyệt:

- Những điều đó nên được thực hiện mỗi khi tải trang

- Những điều đó nên được thực hiện khi trang được đóng lại
- Hành động cần được thực hiện khi người dùng nhấp chuột vào một nút
- Nội dung đó cần được xác nhận khi một dữ liệu được nhập vào bởi người sử dụng
- Và hơn thế nữa ...

Nhiều phương pháp khác nhau có thể được sử dụng với JavaScript và các sự kiện:

- Thuộc tính sự kiện HTML có thể thực thi mã JavaScript trực tiếp
- Thuộc tính sự kiện HTML có thể gọi hàm JavaScript
- Bạn có thể gán các hàm xử lý sự kiện của riêng bạn cho các phần tử HTML
- Bạn có thể ngăn chặn các sự kiện được gửi đi hoặc bị xử lý
- Và hơn thế nữa ...

12.4. Bài tập

1. Phần tử <p> cần thực hiện điều gì đó khi có cú bấm chuột vào nó. Hãy sửa mà cho nó hoạt động.

```
<!DOCTYPE html>
<html>
<body>
<p someevent="this.innerHTML='GOOD JOB!'">Click me.</p>
</body>
</html>
```

2. Khi nút bấm nhận được cú bấm chuột, hãy kích hoạt hàm *myFunction()* bằng một sự kiện.

```
<!DOCTYPE html>
<html>
<body>
<button>Click Me</button>
<p id="demo"></p>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Hello World";
}
</script>
</body>
</html>
```

3. Phần tử cần thực hiện việc gì đó khi có ai đó rê chuột qua nó. Hãy làm cho nó hoạt động.

```
<!DOCTYPE html>
<html>
<body>
<span someevent="this.style.color='red'">Mouse over me!</span>
</body>
</html>
```

13. Chuỗi trong JavaScript

Chuỗi JavaScript được dùng để lưu và thao tác với văn bản.

13.1. Chuỗi JavaScript

Một chuỗi JavaScript chỉ đơn giản là lưu trữ một dãy các ký tự như "John Doe".Một chuỗi có thể chứa được văn bản bất kỳ bên trong dấu nháy kép. Bạn cũng có thể sử dụng dấu nháy đơn. Ví dụ:

```
var carname = "Volvo XC60";
var carname = 'Volvo XC60';
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:
Example

Bạn có thể sử dụng dấu nháy kép bên trong một chuỗi, miễn là chúng không giống với dấu nháy bao quanh chuỗi. Ví dụ:

```
var answer = "It's alright";
var answer = "He is called 'Johnny'";
var answer = 'He is called "Johnny"';
```

13.2. Độ dài chuỗi

Độ dài chuỗi được tính bằng thuộc tính length của chuỗi. Ví dụ:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

13.3. Ký tự đặc biệt

Vì chuỗi ký tự phải được lưu trong cặp dấu nháy kép, JavaScript sẽ không hiểu chuỗi:

```
var y = "We are the so-called "Vikings" from the north."
```

Chuỗi y sẽ chỉ nhântj được "We are the so-called ".

Giải pháp cho vấn đề này là dùng kèm ký tự \

Ký tự \ trả về cho chuỗi ký tự đặc biệt. Ví dụ:

```
var x = 'It\'s alright';
var y = "We are the so-called \"Vikings\" from the north."
```

Ký tự đặc biệt (\) cũng có thể được dùng để chèn các ký tự đặc biệt khác trong xâu.

Mã	Kết quả
\'	một dấu nháy đơn
\"	một dấu nháy kép
\\	Một dấu chéo
\n	Một ký tự sang dòng mới
\r	Một ký tự xuống dòng
\t	Ký tự tab
\b	Ký tự xóa lùi
\f	Ký tự ngắt trang

13.4. Ngắt dòng mã dài

Để dễ đọc nhất, lập trình viên thường muốn tránh dòng code dài hơn 80 ký tự. Nếu một lệnh JavaScript quá dài trên một dòng, nơi tốt nhất để ngắt nó là sau một phép toán. Ví dụ:

```
document.getElementById("demo").innerHTML =
"Hello Dolly.";
```

Bạn cũng có thể ngắt giữa chuỗi văn bản bằng dấu chéo \:

```
document.getElementById("demo").innerHTML = "Hello \
```

```
Dolly!";
```

Chú ý: Các sử dụng dấu \ không hợp tiêu chuẩn ECMAScript. Một số trình duyệt không chấp nhận các ký tự đứng sau \. Để an toàn hơn hãy ngắt như cách trên.

13.5. Chuỗi có thể là đối tượng

Thông thường, chuỗi JavaScript là giá trị nguyên thủy, tạo ra từ các chữ: *var firstName = "John"*. Nhưng chuỗi cũng có thể được định nghĩa là đối tượng với từ khóa *new*: *var firstName = new String ("John")*. Ví dụ:

```
var x = "John";
var y = new String("John");
// typeof x will return string
// typeof y will return object
```

Chú ý: Tuy nhiên, không tạo chuỗi như đối tượng, nó làm giảm tốc độ xử lý của mã.

Khi sử dụng phép toán so sánh bằng ==, ta thấy:

```
var x = "John";
var y = new String("John");
// (x == y) is true because x and y have equal values
```

Khi sử dụng phép toán === thì không phải vậy.

```
var x = "John";
var y = new String("John");
// (x === y) is false because x and y have different types (string and
object)
```

Hoặc tình huống xấu.Các đối tượng không thể so sánh:

```
var x = new String("John");
var y = new String("John");
// (x == y) is false because x and y are different objects
// (x == x) is true because both are the same object
```

Chú ý: Không thể so sánh các đối tượng JavaScript.

13.6. Thuộc tính và phương thức của chuỗi

Giá trị nguyên thủy, như "John Doe", không thể có thuộc tính hoặc phương thức (vì chúng không phải là đối tượng).

Nhưng với JavaScript, phương thức và thuộc tính cũng có sẵn cho giá trị nguyên thủy, vì JavaScript xử lý giá trị nguyên thủy như các đối tượng khi thực hiện phương thức và thuộc tính.

a) Các thuộc tính của chuỗi:

Thuộc tính	Mô tả
constructor	Tạo và trả về một nguyên mẫu đối tượng chuỗi
length	Trả về độ dài chuỗi
prototype	Cho phép bạn thêm thuộc tính và phương thức cho đối tượng

b) Các phương thức của String:

Phương thức	Mô tả
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings

fromCharCode()	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches
replace()	Searches a string for a value and returns a new string with the value replaced
search()	Searches a string for a value and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts a part of a string from a start position through a number of characters
substring()	Extracts a part of a string between two specified positions
toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

c) Các phương thức String phổ biến

Các phương thức chuỗi giúp bạn làm việc với chuỗi.

1) Tìm kiếm chuỗi

- indexOf() trả về vị trí đầu tiên xuất hiện của một chuỗi trong một chuỗi khác

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate"); // pos == 7
```

- *lastIndexOf()*trả xuất hiện lần cuối của một chuỗi trong một chuỗi khác

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");// pos == 21
```

Cả hai phương thức: *indexOf()*và *lastIndexOf()*đều trả về -1 nếu không tìm thấy.

Chú ý: Chỉ số chuỗi của JavaScript bắt đầu từ 0.

Cả hai phương thức đều chấp nhận tham số thứ 2 như là vị trí bắt đầu tìm kiếm.

- *search()* tìm kiếm một chuỗi trong một chuỗi

Phương thức *search()* tương tự *indexOf()*:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate"); //pos == 7
```

Vậy hai phương thức, *indexOf()* và *search()* giống nhau. Chúng chấp nhận cùng một đối số (tham số), và chúng trả về cùng một giá trị.

Hai phương pháp đều như nhau, nhưng *search()* mạnh mẽ hơn nhiều. Bạn sẽ tìm hiểu thêm về giá trị tìm kiếm mạnh mẽ trong các chương sau.

2) Trích một phần của chuỗi

Có 3 phương thức trích lấy một phần của một chuỗi:*slice(start, end);substring(start, end)*;*substr(start, length)*;

- Phương thức *slice()*trích và trả một phần của chuỗi trong một chuỗi mới. Phương thức này nhận hai tham số *start* và *end* ứng với vị trí bắt đầu và kết thúc của chuỗi cần trích. Ví dụ:

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

Nếu một tham số là âm, thì vị trí tìm được tính từ cuối chuỗi ngược về đầu. Ví dụ:

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12,-6);// tính từ 12 về 6, res == "Banana"
```

Nếu bạn bỏ qua tham số thứ hai, phương thức sẽ lấy vị trí *end* là cuối chuỗi.

```
var res = str.slice(7); //res == "Banana, Wiki"
var res = str.slice(-12); //res == "Banana, Wiki"
```

Chú ý: Vị trí âm không hoạt động trên Internet Explorer 8 hoặc cũ hơn.

- *substring()* cũng tương tự như *slice()*. Điểm khác là *substring()*không chấp nhận chỉ số âm.

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

- *substr()*tương tự *slice()*. Điểm khác là, tham số thứ hai là độ dài chuỗi cần trích. Ví dụ:

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

Nếu tham số đầu tiên là âm, vị trí tính từ vị trí kết thúc của chuỗi.Tham số thứ hai không thể âm, vì nó xác định độ dài.Nếu bạn bỏ qua tham số thứ hai, *substr()* sẽ trích ra phần còn lại của chuỗi.

3) Thay thế chuỗi

- Phương thức *replace()*thay thế giá trị được chỉ định bằng giá trị khác trong chuỗi. Ví dụ:

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft","W3Schools");
//str == " Please visit W3Schools!"
```

4) Chuyển chuỗi in hoa - in thường

- Một chuỗi được chuyển sang dạng in hoa bằng phương thức *toUpperCase()*:

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2=="HELLO WORLD!"
```

- Một chuỗi được chuyển sang dạng in hoa bằng phương thức *toLowerCase()*:

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();  // text2=="hello world!"
```

5) Nối chuỗi

Phương thức *concat()*nối hai hoặc nhiều chuỗi thành một chuỗi.

```
var text1 = "Hello";
var text2 = "World";
text3 = text1.concat("  ",text2); //text3 =="Hello World"
```

Có thể dùng phép toán + để thay thế *concat()*:

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ","World!");
```

Lưu ý Tất cả các phương thức chuỗi trả về một chuỗi mới. Chúng không sửa đổi chuỗi ban đầu.

6) Trích ký tự trong chuỗi

Có 2 phương thức an toàn để trích các ký tự:*charAt(position); charCodeAt(position)*.

- Phương thức *charAt()* trả về ký tự tại vị trí chỉ định trong chuỗi.

```
var str = "HELLO WORLD";
str.charAt(0); // returns H
```

- Phương thức *charCodeAt()*trả về mã của ký tự unicode tại vị trí được chỉ định trong chuỗi.

```
var str = "HELLO WORLD";
str.charCodeAt(0);         //  returns 72
```

Truy xuất một chuỗi như một mảng là không an toàn. Có thể bạn đã nhìn thấy mã kiểu này:

```
var str = "HELLO WORLD";
str[0];                   // returns H
```

Nó không an toàn và không đoán trước được điều gì sẽ xảy ra:

- Nó không hoạt động trên tất cả các trình duyệt (như IE5, IE6, IE7)
- Nếu bạn muốn làm việc với một chuỗi như một mảng thì hãy chuyển nó sang mảng trước.

7) Chuyển một chuỗi sang một mảng

Một chuỗi có thể chuyển sang một mảng với phương thức *split()*. Ví dụ:

```
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split on commas
txt.split(" ");          // Split on spaces
txt.split("|");          // Split on pipe
```

- Nếu các dấu phân cách là bỏ qua, mảng kết quả sẽ chứa toàn bộ chuỗi trong chỉ số [0].
- Nếu các dấu phân cách là "", mảng kết quả sẽ là một mảng kí tự.

```
var txt = "Hello";  // String
txt.split("");      // Split in characters
```

13.7. Bài tập

1. Cho biết vị trí chuỗi "*World*" trong chuỗi *txt*.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var txt = "Hello World";
document.getElementById("demo").innerHTML = txt;
</script>
```

```
</body>
</html>
```

2. Dùng phương thức *slice()*hiển thị chuỗi con "*Banana,Kiwi*".

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var str = "Apple,Banana,Kiwi";
document.getElementById("demo").innerHTML = str;
</script>
</body>
</html>
```

3. Dùng phương thức *replace()* thay thế "*World*" bằng "*Universe*".

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var txt = "Hello World";
document.getElementById("demo").innerHTML = txt;
</script>
</body>
</html>
```

4. Chuyển chuỗi *txt* thành in hoa

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var txt = "Hello World";
document.getElementById("demo").innerHTML = txt;
</script>
</body>
</html>
```

5. Chuyển chuỗi *txt* thành in thường.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var txt = "Hello World";
document.getElementById("demo").innerHTML = txt;
</script>
</body>
```

```
</html>
```

6. Dùng phương thức *concat()* nối hai chuỗi: *str1* và *str2*.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
var str1 = "Hello ";
var str2 = "World!";
</script>
</body>
</html>
```

14. Ngày tháng trong JavaScript

Đối tượng Date cho phép bạn làm việc với kiểu ngày tháng (*years, months, days, hours, minutes, seconds, và milliseconds*)

14.1. Định dạng ngày tháng

Giá trị ngày tháng trong JavaScript có thể được viết dưới dạng chuỗi:

```
Tue Nov 03 2015 11:03:52 GMT+0700 (SE Asia Standard Time)
```

Hoặc dạng số:

```
1446523432319
```

Ngày tháng như là một số, tính bằng mili giây trôi qua tính từ *January 1, 1970, 00:00:00*.

14.1.1. Chuỗi ngày tháng hợp lệ

Có 4 kiểu định dạng ngày tháng hợp lệ:

- ISO Dates: (YYYY-MM-DD), ví dụ:

```
var d1 = new Date("2015-03-25");
// Wed Mar 25 2015 07:00:00 GMT+0700 (SE Asia Standard Time)
var d2 = new Date("2015-03");
// Sun Mar 01 2015 07:00:00 GMT+0700 (SE Asia Standard Time)
var d3 = new Date("2015");
// Thu Jan 01 2015 07:00:00 GMT+0700 (SE Asia Standard Time)
var d4 = new Date("2015-03-25T12:00:00");
// Wed Mar 25 2015 19:00:00 GMT+0700 (SE Asia Standard Time)
```

Chú ý: UTC (Universal Time Coordinated) bằng giờ GMT (Greenwich Mean Time).

- Long Dates: "MMM DD YYYY"

```
var d1 = new Date("Mar 25 2015");
// Wed Mar 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
var d2 = new Date("25 Mar 2015");
// Wed Mar 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
var d3 = new Date("2015 Mar 25");
// Wed Mar 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
var d4 = new Date("January 25 2015");
```

```
// Sun Jan 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
var d5 = new Date("Jan 25 2015");
// Sun Jan 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
var d6 = new Date("2015, JANUARY, 25");
// Sun Jan 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
```

- Short Dates: "MM/DD/YYYY"

```
var d1 = new Date("Jan 25 2015");
// Wed Mar 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
var d2 = new Date("2015, JANUARY, 25");
// Wed Mar 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
var d3 = new Date("2015/03/25");
// Wed Mar 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
var d = new Date("03-25-2015");
// Wed Mar 25 2015 00:00:00 GMT+0700 (SE Asia Standard Time)
```

- Full Format:

```
var d = new Date("Wed Mar 25 2015 09:56:24 GMT+0100 (W. Europe Standard
Time)");
// Wed Mar 25 2015 15:56:24 GMT+0700 (SE Asia Standard Time)
var d = new Date("Fri Mar 25 2015 09:56:24 GMT+0100 (Tokyo Time)");
// Wed Mar 25 2015 15:56:24 GMT+0700 (SE Asia Standard Time)
```

14.2. Hiển thị ngày tháng

Trong ví dụ dưới đây, ta in ra ngày tháng tròn phần tử <*p*> với *id = "demo"* :

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Date();
</script>
```

Trong ví dụ trên, ta gán giá trị hàm *Date()*vào nội dung (*innerHTML*) của phần tử có *id="demo*".

14.3. Tạo đối tượng ngày tháng

Đối tượng *Date* cho phép chúng ta làm việc với kiểu ngày tháng. Một kiểu ngày tháng gồm năm, tháng, ngày, giờ, phút, giây, và mili giây. Đối tượng ngày tháng được tạo ra bằng phương thức *new Date()*. Có 4 cách để khởi đầu một ngày:

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

- Dùng *new Date()*, tạo ra một đối tượng ngày tháng với giá trị thời gian hiện tại:

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

- Dùng *new Date(dateString)*, tạo ra một đối tượng ngày tháng từ một thời gian định trước, ghi trong biến chuỗi *dateString*.

```
<script>
var d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
</script>
```

Chú ý: Chuỗi đối số phải theo định dạng đã nêu ở trên.

- Dùng *new Date(number)*, tạo ra một đối tượng ngày tháng từ một thời gian định trước, ghi trong biến kiểu số *number*. Số 0 ứng với giá trị *01 January 1970 00:00:00 UTC*. Số được tính theo milliseconds.

```
<script>
var d = new Date(86400000);
document.getElementById("demo").innerHTML = d;
</script>
```

Chú ý: Ngày tháng trong JavaScript được tính theo milliseconds từ: 01 January, 1970 00:00:00 Universal Time (UTC). Một ngày gồm 86,400,000 millisecond.

- Dùng *new Date(danh sách 7 số)*, tạo ra một đối tượng ngày tháng theo giá trị 7 đối số.

```
<script>
var d = new Date(99,5,24,11,33,30,0);
document.getElementById("demo").innerHTML = d;
// Thu Jun 24 1999 11:33:30 GMT+0700 (SE Asia Standard Time)
</script>
```

Một cách viết khác, bỏ qua 4 đối số cuối.

```
<script>
var d = new Date(99,5,24);
document.getElementById("demo").innerHTML = d;
// Thu Jun 24 1999 00:00:00 GMT+0700 (SE Asia Standard Time)
</script>
```

Chú ý: Giá trị tháng từ 0..11, tháng 1 là 0, tháng 12 là 11.

14.4. Các phương thức ngày tháng

14.4.1. Phương thức Date()

Khi một đối tượng *Date* được tạo ra, một số phương thức cho phép bạn để thực hiện trên nó. Phương thức *Date* cho phép bạn gán năm, tháng, ngày, giờ, phút, giây, và mili giây của đối tượng, sử dụng thời gian địa phương hoặc UTC (hoặc GMT). Lưu ý phương thức *Date* được đề cập đến trong chương sau.

14.4.2. Hiển thị ngày tháng

Khi bạn hiển thị một đối tượng ngày tháng trong HTML, nó sẽ tự động được chuyển thành một chuỗi, với phương thức *toString()*. Ví dụ:

```
<p id="demo"></p>
<script>
d = new Date();
document.getElementById("demo").innerHTML = d;
```

```
</script>
```

Hoặc như dưới đây cho kết quả như nhau:

```
<p id="demo"></p>
<script>
d = new Date();
document.getElementById("demo").innerHTML = d.toString();
</script>
```

- Phương thức *toUTCString()*chuyển một giá trị ngày tháng sang chuỗi UTC (một chuẩn hiển thị ngày tháng). Ví dụ:

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.toUTCString();
// Tue, 03 Nov 2015 04:28:00 GMT
</script>
```

- Phương thức *toDateString()* chuyển một giá trị ngày tháng sang định dạng khác:

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.toDateString();
// Tue Nov 03 2015
</script>
```

14.4.3. Các phương thức ngày tháng thông dụng

Phương thức *Date* cho phép bạn nhận và thiết lập các giá trị ngày tháng (năm, tháng, ngày, giờ, phút, giây, mili giây)

a) Các phương thức lấy ngày tháng

Các phương pháp lấy ngày thán được sử dụng để nhận được một phần của một ngày. Dưới đây là các phương thức phổ biến nhất (theo thứ tự abc):

Phương thức	**Mô tả**
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

b) Các phương thức gán ngày tháng

Các phương thức gán ngày tháng được dùng để gán một phần giá trị cho một đối tượng ngày tháng nào đó:

Phương thức	Mô tả
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)

setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

c) Nhập và phân tách giá trị ngày tháng

Nếu bạn có một chuỗi ngày tháng hợp lệ, bạn có thể sử dụng phương thức *Date.parse()* để chuyển đổi nó về mili giây. *Date.parse()* trả về giá trị mili giây giữa ngày được nhập và 01 tháng 1 năm 1970.

```
<script>
var msec = Date.parse("March 21, 2012");
document.getElementById("demo").innerHTML = msec;
</script>
```

Bạn có thể dùng giá trị mili giây để chuyển đổi nó về đối tượng ngày tháng.

```
<script>
var msec = Date.parse("March 21, 2012");
var d = new Date(msec);
document.getElementById("demo").innerHTML = d;
</script>
```

d) So sánh ngày tháng

Dễ dàng so sánh kiểu ngày tháng. Ví dụ:

```
var today, someday, text;
today = new Date();
someday = new Date();
someday.setFullYear(2100, 0, 14);
if (someday > today) {
    text = "Today is before January 14, 2100.";
} else {
    text = "Today is after January 14, 2100.";
}
document.getElementById("demo").innerHTML = text;
```

14.6. Bài tập

1. Tạo đối tượng *Date* để hiển thị ngày tháng hiện tại

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Display the result here.</p>
<script>
// Add some code here
</script>
</body>
</html>
```

2. Sử dụng đối tượng *Date* với giá trị "*January 10, 2015 10:00:00*".

```
<!DOCTYPE html>
<html>
```

```
<body>
<p id="demo">Display the result here.</p>
<script>
// Add some code here
</script>
</body>
</html>
```

3. Sử dụng phương thức *toDateString()* để chuyển đổi hiển ngày tháng sang dạng khác.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = new Date();
</script>
</body>
</html>
```

15. Mảng trong JavaScript

Mảng trong JavaScript được dùng để lưu nhiều giá trị trong một biến.

15.1. Hiển thị mảng

Ví dụ dưới đây hiển thị giá trị của một mảng trong một phần tử <p> với id = "demo".

```
<p id="demo"></p>
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
```

Dòng đầu tiên (trong kịch bản) tạo ra một mảng có tên là *cars*. Dòng thứ hai hiện phần tử với *id = "demo"*, và hiện các phần tử mảng trong "innerHTML" của nó.

Mảng là một biến đặc biệt, nó lưu giữ nhiều hơn một giá trị tại một thời điểm. Nếu bạn có một danh sách các phần tử (như danh sách các tên xe), mảng sẽ lưu trữ các phần tử như sau:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

Tuy nhiên nếu bạn muốn duyệt qua danh sách xe và tìm một tên nào đó? Và nếu bạn có nhiều hơn 3 chiếc xe thì sao? Giải pháp là: mảng dùng một tên chung cho các phần tử, và bạn có thể truy xuất các giá trị dựa vào chỉ số của nó.

15.2. Tạo một mảng

- Sử dụng một mảng giá trị là cách tạo mảng dễ nhất. Cú pháp là:

```
var array-name = [item1, item2, ...];
```

Ví dụ:

```
var cars = ["Saab", "Volvo", "BMW"];
```

- Sử dụng từ khóa *new*

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Chú ý: Hai ví dụ trên có hiệu quả như nhau. Không cần thiết sử dụng *new Array()*. Để đơn giản, dễ đọc và thực hiện nhanh, hãy sử dụng cách đầu tiên.

15.3. Truy xuất phần tử mảng

You refer to an array element by referring to the index number.
This statement accesses the value of the first element in cars:

Bạn tham chiếu một phần tử mảng bằng cách tham chiếu chỉ số của nó.

- Lệnh này truy xuất giá trị của phần tử đầu tiên trong danh sách xe:

```
var name = cars[0];
```

- Lệnh này thay đổi giá trị của phần tử đầu tiên:

```
cars[0] = "Opel";
```

Chú ý: [0] là phần tử đầu tiên trong mảng. [1] là phần tử thứ hai. Chỉ số mảng được tính từ 0.

Bạn có thể lưu các đối tượng khác nhau trong một mảng. Các biến JavaScript có thể là đối tượng. Mảng là loại đặc biệt của đối tượng. Vì điều này, bạn có thể có các biến khác kiểu trong cùng một mảng.

Bạn có thể có các đối tượng trong một mảng. Bạn có thể có hàm trong một mảng. Bạn có thể có các mảng trong một mảng.

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

15.4. Mảng là các đối tượng

Mảng là một loại đặc biệt của đối tượng. Toán tử typeof trong JavaScript trả về "object" với một mảng. Nhưng, các mảng JavaScript được mô tả tốt nhất như các mảng.

Mảng sử dụng chỉ số thoại để truy cập vào "phần tử" của nó. Trong ví dụ này, *person[0]* trả về là *John*.

```
var person = ["John", "Doe", 46];
```

Objects use names to access its "members". In this example, person.firstName returns John:

Các đối tượng sử dụng tên để truy cập "thành viên" của nó. Trong ví dụ này, *person.firstName* trả về là *John*:

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

15.5. Thuộc tính và phương thức của Array

15.5.1. Thuộc tính và phương thức mảng

Sức mạnh thực sự của mảng JavaScript được xây dựng trong tính mảng và phương pháp:

```
var x = cars.length; // trả về số phần tử mảng
var y = cars.sort();// phương thức sort() sắp xếp các phần tử
```

- Thuộc tính *length*: Trả về số phần tử mảng

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length; // là 4
```

- Thêm phần tử mới vào cuối mảng bằng phương thức *push()*.

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

- Thêm phần tử mới vào cuối mảng bằng cách gán trực tiếp:

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

- Thêm phần tử với chỉ số vượt ra ngoài phạm vi hiện tại của mảng có thể tạo ra các lỗ thủng có kiểu không xác định trong mảng.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[10] = "Lemon";
// các phần tử 4..9 là không xác định
```

15.5.2. Các phương thức mảng thông dụng

Sức mạnh của mảng trong JavaScript nằm ở các phương thức:

a) Chuyển mảng thành xâu

In JavaScript, all objects have the valueOf() and toString() methods.
The valueOf() method is the default behavior for an array. It returns an array as a string:

Trong JavaScript, tất cả các đối tượng có các phương thức *valueOf()* và *toString()*. Phương thức *valueOf()* là hành vi mặc định cho một mảng. Nó trả về một mảng như là một chuỗi.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.valueOf();
// Banana,Orange,Apple,Mango
```

Với mảng trong JavaScript, *valueOf()*và *toString()*là như nhau.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

b) Nối tất cả phần tử mảng thành một chuỗi

Phương thức *join()*nối tất cả các phần tử mảng thành một chuỗi

```
<p id="demo"></p>
<script>
var fruits = ["Banana", "Orange","Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
// Banana * Orange * Apple * Mango
</script>
```

c) Thêm và xóa phần tử mảng

- Phương thức *pop()*xóa phần tử cuối của mảng

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

Phương thức *pop()*trả về phần tử bị xóa.

- Phương thức *push()*thêm phần tử mới vào cuối mảng.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");// thêm "Kiwi" vào mảng, x = "Kiwi"
```

Phương thức *push()* trả về phần tử mới.

d) Dịch chuyển các phần tử

Chuyển dịch tương đương với xóa, thực hiện với phần tử đầu tiên. Phương thức *shift()* loại bỏ phần tử đầu mảng, và dịch các phần tử phía sau lên phía trước 1 vị trí. Ví dụ:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();// Removes the first element "Banana" from fruits
```

Phương thức*unshift()* thêm một phần tử mới vào đầu mảng, và đẩy các phần tử cũ ra phía sau một vị trí. Ví dụ:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");// Thêm "Lemon" vào vị trí đầu
```

Phương thức *shift()* trả về phần tử đã bị xóa.

Phương pháp *unshift()* trả về độ dài mảng mới.

e) Thay đổi các phần tử

- Các phần tử mảng được truy cập bằng cách sử dụng số chỉ mục của chúng. Ví dụ:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi"; // thay đổi ptử đầu thành "Kiwi"
```

- Thuộc tính *length*cung cấp một cách thêm mới phần tử vào cuối mảng. Ví dụ:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi"; // Thêm "Kiwi" vào cuối mảng
```

f) Xóa các phần tử

Vì mảng JavaScript là các đối tượng, có thể xóa các phần tử bằng cách sử dụng phép toán *delete*. Ví dụ:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0]; // Xóa ptử đầuthành undefined
```

Chú ý: Sử dụng phép toán *delete*trên mảng để lại một phần tử không kiểu trong mảng. Nếu bạn muốn xóa hẳn thì sử dụng *pop()*hoặc *shift()*.

g) Chèn thêm một dãy giá trị hoặc mảng vào mảng hiện tại

Phương thức *splice()*được sử dụng để thêm các phần tử mới vào một mảng. Ví dụ:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

- Tham số đầu tiên (2) xác định vị trí nơi mà các phần tử mới cần được thêm vào.
- Tham số thứ hai (0) xác định có bao nhiêu phần tử cần được loại bỏ.
- Các tham số còn lại ("Lemon", "Kiwi") xác định các phần tử mới được thêm vào.

Sử dụng *splice()*có thể xóa bỏ các phần tử mà không tạo ra các "lỗ thủng" trong mảng. Ví dụ:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1); // Xóa ptử đầu tiên
```

- Tham số đầu tiên (0) xác định vị trí nơi mà các phần tử mới được thêm vào.
- Tham số thứ hai (1) xác định có bao nhiêu phần tử cần loại bỏ.
- Phần còn lại của các tham số được bỏ qua. Không có phần tử mới nào được thêm vào.

h) Sắp xếp mảng

Phương thức *sort()*cho phép sắp xếp mảng theo trật tự alphabetically. Ví dụ:

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

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

i) Lật ngược mảng

Phương thức *reverse()*lật ngược thứ tự các phần tử mảng. Có thể sử dụng nó để sắp xếp mảng theo chiều ngược lại (giảm dần). Ví dụ:

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

k) Sắp xếp các số

Theo mặc định, hàm *sort()*sắp xếp các giá trị như chuỗi.Tuy nhiên, nếu số được sắp xếp như chuỗi, thì "25" là lớn hơn "100", bởi vì "2" là lớn hơn "1". Có thể khắc phục bằng cách cung cấp một hàm so sánh. Ví dụ:

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
```

Và cũng bằng cách này ta có thể sắp xếp giảm dần dãy số:

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b-a});
```

- *Hàm so sánh*: Mục đích của hàm so sánh là để xác định một thứ tự sắp xếp thay thế. Hàm so sánh phải trả lại một giá trị âm, 0, hoặc giá trị dương, tùy thuộc vào đối số:

```
function(a, b) {return a-b}
```

Khi hàm *sort()*so sánh hai giá trị, nó sẽ gửi các giá trị cho hàm so sánh, và sắp xếp các giá trị căn cứ theo giá trị hàm so sánh trả về. Ví dụ:

Khi so sánh 40 và 100, phương thức *sort()*gọi hàm so sánh (40, 100). Hàm so sánh tính toán 40-100, và trả về -60 (giá trị âm). Phương thức *sort()*sẽ sắp xếp 40 là giá trị đứng trước 100.

- *Tìm giá trị cao nhất (hoặc thấp nhất)*:

Làm thế nào để tìm giá trị cao nhất trong một mảng?Ví dụ:

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b-a});
// points[0] chứa giá trị cao nhất
```

Và thấp nhất? Ví dụ:

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
// points[0] chứa giá trị thấp nhất
```

l) Nối các mảng

Phương thức *concat()*tạo ra một mảng mới chứa hai mảng thành phần:

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias","Linus"];
var myChildren = myGirls.concat(myBoys);
// Nối myGirls và myBoys
```

Phương thức *concat()*cũng có thể thực hiện với số lượng mảng bất kỳ:

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias","Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);
```

```
// Nối arr1 với arr2 và arr3
```

m) Phân chia một mảng

Phương thức *slice(start, end)* lấy ra một phần của một mảng và tạo thành một mảng mới:

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1, 3);
```

Phương thức *slice()* chọn các phần tử từ vị trí *start* và kết thúc vị trí trước vị trí *end*(nửa đoạn).

Nếu đối số cuối cùng bị bỏ qua, *slice()* lấy ra phần còn lại của mảng, tính từ vị trí *start*.

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
```

15.6. Duyệt các phần tử mảng

Các tốt nhất là sử dụng vòng lặp *for*. Ví dụ:

```
var index;
var fruits = ["Banana", "Orange", "Apple", "Mango"];
for  (index = 0; index < fruits.length; index++) {
    text += fruits[index];
}
```

15.7. Mảng kết hợp

Nhiều ngôn ngữ lập trình hỗ trợ các mảng với chỉ số được đặt tên. Mảng với chỉ số có tên gọi là mảng kết hợp (hoặc mảng băm). JavaScript không hỗ trợ mảng với chỉ số được đặt tên. Trong JavaScript, mảng luôn luôn sử dụng các chỉ số là số. Ví dụ:

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length;  // 3
var y = person[0]; // "John"
```

Chú ý: nếu bạn dùng chỉ số được đặt tên, JavaScript sẽ định nghĩa lại mảng về đối tượng chuẩn. Sau đó, tất cả các phương thức thuộc tính mảng sẽ trả ra các kết quả sai.

```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
var x = person.length;  // 0
var y = person[0];      // undefined
```

15.8. Sự khác nhau giữa mảng và đối tượng

- Trong JavaScript, mảng sử dụng chỉ số là số.
- Trong JavaScript, các đối tượng sử dụng chỉ số tên.
- JavaScript không hỗ trợ mảng kết hợp
- Bạn nên sử dụng đối tượng khi bạn muốn đặt tên các phần tử.
- Bạn nên sử dụng mảng khi bạn muốn đặt tên các phần tử là chỉ số.

- Tránh dùng *new Array(),* hãy dùng [] thay thế

```
var points = new Array();  // Bad
var points = [];           // Good
```

Hai cách khác nhau để tạo mảng có 6 phần tử.

```
var points = new Array(40, 100, 1, 5, 25, 10)  // Bad
var points = [40, 100, 1, 5, 25, 10];          // Good
```

từ khóa *new*đôi khi gây rắc rối cho bạn:

```
var points = new Array(40, 100);
```

tạo mảng với 2 phần tử 40 và 100. Còn lệnh dưới đây tạo ra mảng với 40 phần tử không kiểu.

```
var points = new Array(40);
```

15.9. Tổ chức mảng thế nào

Một câu hỏi chung là: Làm thế nào để biết một biến là một mảng? Vấn đề là toán tử *typeof* trả về "object":

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
typeof fruits; // typeof returns object
```

Để giải quyết vấn đề này bạn có thể tạo hàm isArray()của riêng bạn:

```
function isArray(myArray) {
    return myArray.constructor.toString().indexOf("Array") > -1;
}
```

Hàm này luôn trả về giá trị true nếu đối số là mảng.

15.10. Bài tập

1. Dùngphương thức *pop()*để loại bỏ mục cuối cùng từ mảng*fruits*.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var fruits = ["Banana", "Orange", "Apple"];
document.getElementById("demo").innerHTML = fruits;
</script>
</body>
</html>
```

2. Dùng phương thức *push()*để thêm phần tử mới *Kiwi*vào *fruits*

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var fruits = ["Banana", "Orange", "Apple"];
document.getElementById("demo").innerHTML = fruits;
</script>
</body>
```

```
</html>
```

3. Dùng phương thức *splice()*để loại bỏ "*Orange*" và "*Apple*" trong *fruits*.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var fruits = ["Banana", "Orange", "Apple", "Kiwi"];
document.getElementById("demo").innerHTML = fruits;
</script>
</body>
</html>
```

4. Dùng phương thức *sort()*để sắp xếp mảng theo trật tự alphabetically.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var fruits = ["Banana", "Orange", "Apple", "Kiwi"];
document.getElementById("demo").innerHTML = fruits;
</script>
</body>
</html>
```

5. Dùng phương thức *concat()*để nối *girls*và *boys*.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var girls = ["Cecilie", "Lone"];
var boys = ["Emil", "Tobias", "Linus"];
var children = // add code here
document.getElementById("demo").innerHTML = children;
</script>
</body>
</html>
```

16. Cấu trúc điều kiện

Như các ngôn ngữ lập trình khác, JavaScript cũng có các câu lệnh điều kiện ***if*** và ***if ..else***:

16.1. Cấu trúc if

Cú pháp:

```
if (điều kiện) {
    Lệnh
```

```
}
```

Ví dụ:

```
if (hour < 18) {
    greeting = "Good day";
}
```

16.2. lệnh else

Cú pháp:

```
if (Điều kiện) {
    lệnh 1;
} else {
     Lệnh 2;
}
```

Ví dụ:

```
if (hour < 18) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

16.3. Lệnh else if

Cú pháp:

```
if (điều kiện 1) {
    lệnh 1;
} else if (điều kiện 2) {
    Lệnh 2;
} else {
    Lệnh 3;
}
```

Ví dụ:

```
if (time < 10) {
    greeting = "Good morning";
} else if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

16.4. Lệnh switch

Sử dụng lệnh switch để lựa chọn thực thi một trong nhiều khối lệnh.

Cú pháp:

```
switch(expression) {
    case n:
        code block
```

```
        break;
    case n:
        code block
        break;
    default:
        default code block
}
```

Trong cấu trúc lệnh switch, thành phần biểu thức <*expression*> là một biểu thức giá trị. Giá trị của biểu thức này được so sánh với giá trị của mỗi trường hợp trong các nhánh *case*.

Nếu giá trị được so khớp, thì khối lệnh kết hợp với trường hợp này được thi hành.

Ví dụ:

Phương thức *getDay()* trả về ngày trong tuần như một số nguyên trong đoạn 0..6 (Sunday=0, Monday=1, Tuesday=2 ..)

Trong ví dụ dưới ta đổi dạng số sang dạng tên của ngày trong tuần:

```
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";
        break;
}
```

Từ khóa break

Khi thực hiện các lệnh của chương trình, gặp từ khóa break, máy tính sẽ dừng xử lý các lệnh tiếp theo trong khối lệnh hiện tại và thoát khỏi khối lệnh này.

Chú ý: Khi so sánh giá trị biểu thức của lệnh *switch* khớp với giá trị kết hợp với khối lệnh trong *switch*, thì các lệnh trong khối lệnh này được thi hành cho đến khi gặp *break*.

Từ khóa default

Từ khóa *default* chỉ định khối mã lệnh được thị hành khi không có trường hợp nhánh case nào trong *switch* được so khớp với giá trị biểu thức của *switch*:

Ví dụ: Phương thức *getDay()* trả về ngày trong tuần ứng với giá trị thuộc đoạn 0 và 6.

Nếu hôm nay không phải Saturday (6) hoặc Sunday (0), thì *default* sẽ đưa ra thông báo:

```
switch (new Date().getDay()) {
    case 6:
        text = "Today is Saturday";
        break;
    case 0:
        text = "Today is Sunday";
        break;
    default:
        text = "Looking forward to the Weekend";
}
```

Nhiều nhánh case trong switch sử dụng chung khối lệnh

Đôi khi, bạn muốn nhiều trường hợp giá trị dùng chung một khối lệnh trong *switch*. Ví dụ:

```
switch (new Date().getDay()) {
    case 1:
    case 2:
    case 3:
    default:
        text = "Looking forward to the Weekend";
        break;
    case 4:
    case 5:
       text = "Soon it is Weekend";
        break;
    case 0:
    case 6:
       text = "It is Weekend";
}
```

Các trường hợp giá trị ngày trong tuần là 1, 2, 3 và *defaul* dùng chung một thông báo. Còn các giá trị 4, 5 dùng chung một thông báo khác.

17. Cấu trúc lặp

17.1. Khái niệm về vòng lặp

Vòng lặp thường được dùng trong trường hợp, bạn muốn thực hiện cùng một mã nhiều lần, mỗi lần với một giá trị khác nhau. Thường là trường hợp khi làm việc với mảng:

Thay vì viết:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
```

```
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

Bạn có thể viết:

```
for (i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
}
```

Các loại cấu trúc lặp. JavaScript hỗ trợ các cấu trúc lặp sau:

- Cấu trúc lặp *for* thực thi một khối lệnh, một số lần.
- Cấu trúc lặp *for/in* duyệt qua các thuộc tính của một đối tượng.
- Cấu trúc lặp *while* thực thi khối lệnh chừng nào điều kiện lặp còn đúng.
- Cấu trúc lặp *do/while* thực thi khối lệnh cho đến khi điều kiện lặp còn đúng.

17.2. Cấu trúc For

Cấu trúc lặp *for* thường là công cụ mà bạn sẽ sử dụng khi bạn muốn tạo ra một vòng lặp.

Cấu trúc *for* có cú pháp như sau:

```
for (Lệnh 1; Lệnh 2; Lệnh 3) {
    Khối lệnh cần thi hành
}
```

- *Lệnh 1* được thực hiện trước khi *khối lệnh* bắt đầu.
- *Lệnh 2* định nghĩa điều kiện để chạy *khối lệnh*.
- *Lệnh 3* được thực hiện mỗi lần sau khi *khối lệnh* đã được thực hiện.

Ví dụ:

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

Trong ví dụ trên, bạn có thể hiểu:

- Lệnh 1 gán giá trị biến i = 0 trước khi vòng lặp bắt đầu.
- Lệnh 2 định nghĩa điều kiện lặp (i < 5).
- Lệnh 3 làm tăng giá trị (i++) mỗi lần khối mã trong vòng lặp được thực thi.

+ ***Với lệnh 1***: bạn sẽ sử dụng lệnh 1 để khởi tạo biến đếm sử dụng trong vòng lặp (i = 0). Không bắt buộc phải luôn như vậy, JavaScript không quan tâm. Lệnh 1 là tùy chọn. Bạn có thể khởi tạo nhiều giá trị trong lệnh 1 (cách nhau bằng dấu phẩy). Ví dụ:

```
for (i = 0, len = cars.length, text = ""; i < len; i++) {
    text += cars[i] + "<br>";
}
```

Bạn có thể bỏ qua lệnh 1. Ví dụ:

```
var i = 2;
var len = cars.length;
var text = "";
for (; i < len; i++) {
    text += cars[i] + "<br>";
```

```
}
```

+ ***Lệnh 2***: Thường được sử dụng để đánh giá điều kiện của biến ban đầu. Không nhất thiết luôn như vậy, JavaScript không quan tâm. Lệnh 2 cũng là tùy chọn.

Nếu lệnh 2 trả về *true*, vòng lặp thực hiện lại một lần nữa, nếu nó trả về *false*, vòng lặp sẽ kết thúc.

Chú ý: Nếu bạn bỏ qua lệnh 2, bạn phải có một lệnh *break* trong vòng lặp. Nếu không vòng lặp sẽ không bao giờ kết thúc. Điều này sẽ làm sụp đổ trình duyệt của bạn. Bạn sẽ tìm hiểu về *break* trong phần sau.

+ ***Lệnh 3***: Thường làm tăng giá trị biến ban đầu. Cũng không phải là bắt buộc, JavaScript không quan tâm, và lệnh 3 là tùy chọn. Lệnh 3 có thể làm bất cứ điều gì giống như giảm (i--), tăng (i = i + 15), hoặc bất cứ điều gì khác.

Lệnh 3 cũng có thể được bỏ qua (như khi đó bạn phải tự tăng giá trị của bạn trong vòng lặp). Ví dụ:

```
var i = 0;
var len = cars.length;
for (; i < len; ) {
    text += cars[i] + "<br>";
    i++;
}
```

17.3. Cấu trúc For/In

Cấu trúc lặp *for/in* duyệt qua các thuộc tính của một đối tượng. Ví dụ:

```
var person = {fname:"John", lname:"Doe", age:25};
var text = "";
var x;
for (x in person) {
    text += person[x];
}
```

17.4. Cấu trúc While

Cấu trúc lặp while duyệt qua một khối mã khi nào điều kiện lặp còn bằng *true*. Cú pháp:

```
while (điều kiện) {
     khối mã sẽ được thực thi
}
```

Ví dụ: mã trong vòng lặp sẽ còn chạy khi nào biến i < 10:

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

Chú ý: Nếu bạn quên thay đổi giá trị biến trong biểu thức điều kiện lặp, bạn có thể làm vòng lặp không bao giờ dừng.

17.5. The Do/While Loop

Cấu trúc lặp *do/while* là một biến thể của cấu trúc lặp *while*. Cấu trúc lặp này sẽ thực hiện khối mã một lần, trước khi kiểm tra điều kiện lặp. Nếu điều kiện lặp còn bằng *true*, nó sẽ lặp lại vòng lặp lần nữa, ngược lại nó sẽ dừng vòng lặp. Cú pháp:

```
do {
    code block to be executed
}
while (condition);
```

Ví dụ: Vòng lặp *do/while* dưới đây. Vòng lặp sẽ thi hành tối thiểu một lần, thậm chí cả khi điều kiện lặp = false, vì khối mã được thi hành trước khi điều kiện lặp được kiểm tra:

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

Chú ý: Nếu bạn quên thay đổi giá trị biến trong biểu thức điều kiện lặp, bạn có thể làm vòng lặp không bao giờ dừng.

So sánh cấu trúc For và While

Nếu bạn đã đọc về vòng lặp for, bạn sẽ thấy rằng vòng lặp while có nhiều điểm giống vòng lặp for, với lệnh 1 và lệnh 3 bị bỏ qua.

Ví dụ dưới sử dụng một vòng lặp for để thu thập các tên xe từ mảng *cars*:

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";
for (;cars[i];) {
    text += cars[i] + "<br>";
    i++;
}
```

Cũng với mục đích trên, nếu dùng vòng lặp while, ta có mã sau:

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";
while (cars[i]) {
    text += cars[i] + "<br>";
    i++;
}
```

17.6. Lệnh break, continue và label

- Lệnh *break* dùng để nhảy ra khỏi vòng lặp.
- Lệnh *continue* để bỏ qua một bước lặp.

17.6.1. Lệnh break

Bạn đã thấy lệnh *break* được sử dụng trong phần trước. *Break* được sử dụng để "nhảy ra" khỏi một lệnh *switch()*.

Câu lệnh *break* cũng có thể được sử dụng để nhảy ra khỏi một vòng lặp.

Câu lệnh *break* ngắt vòng lặp và tiếp tục thực hiện các mã lệnh sau vòng lặp (nếu có):

Ví dụ:

```
for (i = 0; i < 10; i++) {
    if (i === 3) { break; }
    text += "The number is " + i + "<br>";
}
```

17.6.2. The Continue Statement

Lệnh *continue* bỏ qua một bước lặp (trong vòng lặp), nếu một điều kiện xác định xảy ra, và vòng lặp vẫn tiếp tục với bước lặp kế tiếp. Ví dụ dưới sẽ bỏ qua giá trị 3:

```
for (i = 0; i < 10; i++) {
    if (i === 3) { continue; }
    text += "The number is " + i + "<br>";
}
```

17.6.3. JavaScript Labels

Lệnh label nhằm gán một nhãn để qua nó có thể phá vỡ cấu trúc lệnh lặp hoặc khối lệnh.

Cú pháp:

```
break labelname;
continue labelname;
```

Lệnh continue (có hoặc không có nhãn) chỉ có thể được sử dụng để bỏ qua một bước lặp.

Lệnh break, không có nhãn, chỉ có thể được sử dụng để nhảy ra khỏi một vòng lặp hoặc switch.

Với một nhãn, lệnh break có thể được sử dụng để nhảy ra khỏi khối mã bất kỳ. Ví dụ:

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
list: {
    text += cars[0] + "<br>";
    text += cars[1] + "<br>";
    text += cars[2] + "<br>";
    text += cars[3] + "<br>";
    break list;
    text += cars[4] + "<br>";
    text += cars[5] + "<br>";
}
```

Chú ý: Một khối lệnh luôn nằm giữa cặp ngoặc nhọn { và }.

17.7. Bài tập

1. Trong vòng lặp for, thay *num1* = 0 và *num2* = 10, rồi chạy thử đoạn mã.

```
<!DOCTYPE html>
<html>
```

```
<body>
<p id="demo"></p>
<script>
var i;
for (i = num1; i < num2; i++) {
    document.getElementById("demo").innerHTML += i + "<br>";
}
</script>
</body>
</html>
```

2. Làm cho vòng lặp bắt đầu đếm từ 5 đến 50, với mỗi bước nhảy là 5:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var i;
for (i = 0; i < 10; i++) {
    document.getElementById("demo").innerHTML += i + "<br>";
}
</script>
</body>
</html>
```

3. Làm cho vòng lặp đếm lùi từ 10 về 1.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var i;
for (i = 0; i < 10; i++) {
    document.getElementById("demo").innerHTML += i + "<br>";
}
</script>
</body>
</html>
```

4. Bên trong vòng lặp for: gán biến đếm i = 0. Chạy vòng lặp khi nào i còn nhỏ hơn độ dài của mảng *food*. Mỗi bước lặp tăng i lên 1 đơn vị.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var text = "";
```

```
var food = ["Pizza", "Beans", "Tacos", "Fish", "Chicken"];
var i;
for () {
    text += "I love " + food[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

5. Trong bài tập này, bạn sẽ tạo ra một vòng lặp từ đầu. Chúng tôi đã tạo biến đếm i cho bạn, và một biến text để in kết quả ra.

Nhiệm vụ của bạn là ra một vòng lặp *for* in ra các số từ 1 3 5 7 9 với ngắt dòng giữa mỗi số.

Chú ý: Hãy nhớ tăng biến được sử dụng để tránh vòng lặp vô hạn.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var text = "";
var i;
// add for loop here
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

6. Trong vòng lặp while, thay *num1* = 0 và *num2* = 10, rồi chạy thử đoạn mã.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var i = num1;
while (i < num2) {
    document.getElementById("demo").innerHTML += i + "<br>";
    i++;
}
</script>
</body>
</html>
```

7. Làm cho vòng lặp bắt đầu đếm từ 5 đến 50, với mỗi bước nhảy là 5:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
```

```
<script>
var i = 0;
while (i < 10) {
    document.getElementById("demo").innerHTML += i + "<br>";
    i++;
}
</script>
</body>
</html>
```

8. Làm cho vòng lặp trong bài 7 đếm lùi từ 10 về 1.

9. Sửa bài 7 để dùng cấu trúc lặp do/while

10. Dùng break trong bài 1 để dừng vòng lặp tại i = 5.

11. Dùng continue trong bài 1 để bỏ qua các giá trị i chẵn.

18.