**Các lỗi phổ biến trong JavaScript**

Bài viết này sẽ nói về một số lỗi JavaScript thường gặp.

**Vô tình sử dụng các toán tử gán**

Các chương trình JavaScript có thể tạo ra nhiều kết quả không mong đợi nếu lập tình viên vô tình sử dụng toán tử gán không đúng mục đích (=), thay vì đáng ra phải sử dụng toán tử so sánh bằng (==) trong câu lệnh điều kiện if.

Câu lệnh **if** dưới đây trả về kết quả **false** (như mong đợi) bởi vì x không bằng 10:

*var x = 0;*  
*if (x == 10)*

Câu lệnh **if** dưới đây trả về kết quả **true** (không như mong đợi) bởi vì 10 là đúng:

*var x = 0;*  
*if (x = 10)*

Câu lệnh **if**dưới đây trả về kết quả **false** (có thể không như mong đợi), bởi vì 0 là false:

*var x = 0;*  
*if (x = 0)*

Một phép gán luôn luôn trả về giá trị của phép gán.

**So sánh lỏng lẻo không như mong đợi**

Trong phép so sánh thông thường, kiểu dữ liệu không quan trọng. Câu lệnh if dưới đây sẽ trả về kết quả true:

*var x = 10;*  
*var y = “10”;*  
*if (x == y)*

Trong phép so sánh chặt, kiểu dữ liệu là quan trọng. Câu lệnh if dưới đây trả về kết quả false:

*var x = 10;*  
*var y = “10”;*  
*if (x === y)*

Một lỗi phổ biến là quên rằng câu lệnh switch sử dụng so sánh chặt:

Trong trường hợp này switch luôn luôn hiển thị thông báo:

*var x = 10;*  
*switch(x) {*  
*case 10: alert(“Hello”);*  
*}*

Trong trường hợp này switch sẽ không hiển thị thông báo:

*var x = 10; // 10 này là number*  
*switch(x) {*  
*case “10”: alert(“Hello”); // còn “10” này là string do vậy không thể bằng nhau ở phép so sánh chặt*  
*}*

**Nhầm lẫn giữa cộng & nối**

**Phép cộng**chỉ phép tính cộng giữa các **con Số**.

**Nối** chỉ phép nối chuỗi giữa **các Chuỗi** với nhau.

Trong JavaScript cả hai phép tính đều sử dụng cùng toán tử +

Vì lý đó, việc cộng số với số sẽ cho kết quả khác với việc cộng thêm số với chuỗi:

*var x = 10 + 5;          // cho kết quả là số 15*  
*var x = 10 + “5”;        // cho kết quả là chuỗi “105”*

Khi cộng hai biến, sẽ có khó khăn để dự đoán kết quả cũng như nhận biết sai lầm đang ở chỗ nào:

*var x = 10;*  
*var y = 5;*  
*var z = x + y;           // kết quả của z là 15*

*var x = 10;*  
*var y = “5”;*  
*var z = x + y;           // kết quả của z là “105”*

**Hiểu lầm về số Thực**

Tất cả số trong JavaScript được lưu giữ dưới dạng 64-bits số thực dấu chấm động.

Tất cả các ngôn ngữ lập trình, bao gồm cả JavaScript sẽ gặp khó khăn với việc chắn chắn về giá trị của số thực dấu chấm động:

var x = 0.1;  
var y = 0.2;  
var z = x + y            // kết quả của z không phải là 0.3  
if (z == 0.3)            // câu lệnh if sẽ trả về kết quả fail

Để giải quyết rắc rối trên, mẹo là nhân lên rồi chia ra:

**Ví dụ**

var z = (x \* 10 + y \* 10) / 10;       // z sẽ có kết quả 0.3

**Bẻ một chuỗi JavaScript**

JavaScript cho phép bạn bẻ câu lệnh thành hai dòng:

**Ví dụ 1**

var x =  
“Xin Chào!”;

Nhưng nếu bẻ câu lệnh ở giữa chuỗi, nó sẽ không làm việc:

**Ví dụ 2**

var x = “Xin  
Chào!”;

Bạn phải sử dụng dấu gạch ngược nếu bạn bẻ một câu lệnh trong chuỗi:

**Ví dụ 3**

var x = “Hello \  
World!”;

**Dấu chẩm phẩy đặt sai**

Bởi vì dấu chẩm phẩy đặt sai, khối code sẽ thực thi mà không tính đến giá trị của x:

if (x == 19);  
{  
// khối code  
}

**Bẻ một câu lệnh trả về (return)**

Là một hành vi mặc định trong JavaScript khi đóng một câu lệnh tự động tại vị trí cuối cùng của một dòng.

Vì lý do đó, hai ví dụ dưới đây sẽ trả về cùng một kết quả:

**Ví dụ 1**

*function myFunction(a) {*  
*var power = 10*  
*return a \* power*  
*}*

**Ví dụ 2**

*function myFunction(a) {*  
*var power = 10;*  
*return a \* power;*  
*}*

JavaScript cũng cho phép bạn bẻ câu lệnh thành hai dòng.

Vì lý do đó, ví dụ 3 sẽ trả về cùng một kết quả:

**Ví dụ 3**

*function myFunction(a) {*  
*var*  
*power = 10;*  
*return a \* power;*  
*}*

Nhưng điều gì sẽ xảy ra nếu bản bẻ câu lệnh return trong hai dòng giống như thế này:

**Ví dụ 4**

*function myFunction(a) {*  
*var*  
*power = 10;*  
*return*  
*a \* power;*  
*}*

Hàm sẽ trả về kết quả là không xác định!

Tại sao? Bởi vì JavaScript nghĩ ý bạn là như này:

**Ví dụ 5**

*function myFunction(a) {*  
*var*  
*power = 10;*  
*return;*  
*a \* power;*  
*}*

**Giải thích**

Nếu một câu lệnh không hoàn chỉnh như thế này:

*var*

JavaScript sẽ cố gắng hoàn thành câu lệnh bằng cách đọc dòng kết tiếp:

*power = 10;*

Nhưng câu lệnh này là hoàn chỉnh:

*return*

JavaScript sẽ tự động đóng nó giống như thế này:

*return;*

Chuyện này xảy ra bởi vì câu lệnh cuối cùng với dâu chấm phẩy là tuỳ chọn trong JavaScript.

JavaScript sẽ đóng câu lệnh return tại vị trí cuối cùng của dòng, bởi vì nó là câu lệnh hoàn chỉnh.

Không bao giờ được bẻ câu lệnh return – nó sẽ cho bạn các kết quả không như mong đợi.

**Truy cập mảng với chỉ mục Tên**

Rất nhiều ngôn ngữ lập trình hỗ trợ mảng với **chỉ mục tên**.

Mảng với chỉ mục tên được gọi là **mảng liên kết** (hoặc băm).

JavaScript không hỗ trợ mảng với chỉ mục tên.

Trong JavaScript, **mảng** sử dụng **chỉ mục số**:

**Ví dụ**

var person = [];  
person[0] = “Đức Anh”;  
person[1] = “Nguyễn”;  
person[2] = 30;  
var x = person.length;         // person.length sẽ trả về kết quả 3  
var y = person[0];             // person[0] sẽ trả về kết quả “Đức Anh”

Trong JavaScript, **đối tượng** sử **chỉ mục tên**.

Nếu bạn sử dụng chỉ mục tên, khi truy cập một mảng, JavaScript sẽ định nghĩa mảng thành đối tượng tiêu chuẩn.

Sau khi tự động định nghĩa lại, các phương thức và thuộc tính của mảng sẽ tạo ra các kết quả vô nghĩa hoặc không chính xác:

**Ví dụ:**

var person = [];  
person[“firstName”] = “Đức Anh”;  
person[“lastName”] = “Nguyễn”;  
person[“age”] = 30;  
var x = person.length;         // person.length sẽ trả về 0  
var y = person[0];             // person[0] sẽ trả về undefined

**Kết thúc định nghĩa mảng với một dấu phẩy**

**Không chính xác:**

points = [40, 100, 1, 5, 25, 10,];

Một số công cụ jSON và JavaScript sẽ lỗi, hoặc có hành vi không như mong đợi.

**Chính xác:**

points = [40, 100, 1, 5, 25, 10];

**Kết thúc định nghĩa Đối tượng với một dấu phẩy**

**Không chính xác:**

person = {firstName:”Đức Anh”, lastName:”Nguyễn”, age:30,}

Một số công cụ jSON và JavaScript sẽ lỗi, hoặc có hành vi không như mong đợi.

**Chính xác:**

person = {firstName:”Đức Anh”, lastName:”Nguyễn”, age:30}

**Undefined Không phải là Null**

Với JavaScript, **null** là đối tượng, **undefined**là cho biến, thuộc tính và đối tượng.

Là null, một đối tượng đã được xác định, nếu không nó sẽ là undefined.

Nếu bạn muốn kiểm tra một đối tượng tồn tại, điều này sẽ ném ra một lỗi nếu đối tượng là không xác định:

**Không chính xác:**

if (myObj !== null && typeof myObj !== “undefined”)

Vì lý do đó, bạn phải kiểm tra typeof() trước:

**Chính xác:**

if (typeof myObj !== “undefined” && myObj !== null)

**Expecting Block Level Scope**

JavaScript **không tạo**phạm vi mới cho mỗi khối code.

Nó là đúng trong nhiều ngôn ngữ lập trình, nhưng **không đúng** trong JavaScript.

Đây là một lỗi phổ biến trong các nhà phát triển JavaScript, tin rằng đoạn code dưới đây trả về undefined:

**Ví dụ**

for (var i = 0; i < 10; i++) {  
// some code  
}  
return i;