**Tip#1:**

Ta có đoạn code sau:

var a = 33;

var myFunc = function(){

    var a =44;

    console.log("Inside a function");

    console.log(a);

    console.log("Inside an if statement");

    if(a>0)

    {

        var a = 333;

        console.log(a);

    }

    console.log("Inside a function after change:");

    console.log(a);

}

console.log("Before calling the method");

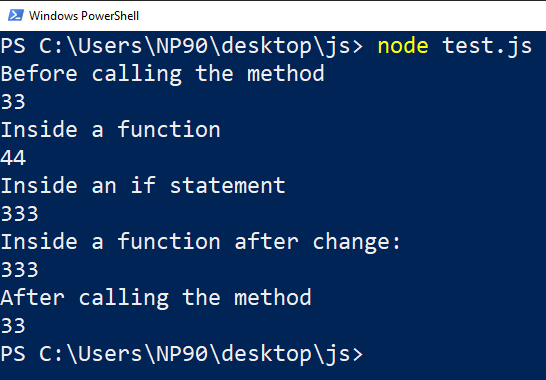
console.log(a);

myFunc();

console.log("After calling the method");

console.log(a);

Kết quả in ra của đoạn code trên là:



Ta thaays rằng ở đây có 2 phạm vi là bên ngoài function (global) và bên trong function. Ta thấy rằng bên trong function nó là một scope khác với bên ngoài function, nên biến a trong function khác với biến a ngoài function. Mặt khác, bên trong function thì biến a là một thể hiện dù ta có thực hiện khai báo lại bằng từ khóa var thì nó là một instant trong phạm vi bên trong hàm.

**Tip#2**

Ta sét đoạn code sau:

function sum(x, y){

result = x+y;

return result;

}

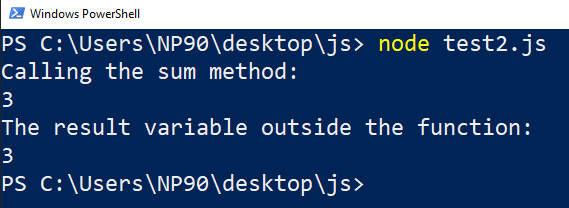
console.log("Calling the sum method: ");

console.log(sum(1,2));

console.log("The result variable outside the function:");

console.log(result);

Khi thực thi đoạn code trên ta được kết quả như sau:



Ta thấy rằng bên trong hàm sum, ta thực hiện sử dụng biến result mà không khai báo, do vậy trình dịch hiểu là nó sẽ là phạm vi global, do vậy khi thực thi hàm sum thì biến result vẫn còn giá trị. Trong thực tế điều này rất nguy hiểu nếu ở đâu đó đã khai báo một biến global result thì biến đó sẽ bị ta override giá trị lại.

Cách khắc phục ta thường làm như sau: (khai báo thêm từ khóa var).

function sum(x, y){

var result = x+y;

return result;

}

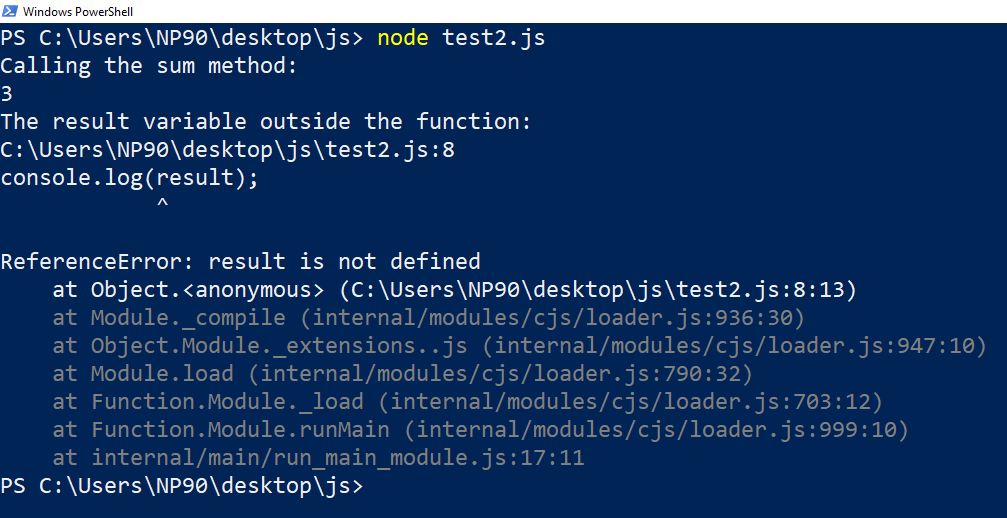
console.log("Calling the sum method: ");

console.log(sum(1,2));

console.log("The result variable outside the function:");

console.log(result);

Kết quả



**Tip#3**

Ta xem xét đoạn code sau:

function myFunction(){

var a=b=0;

console.log("First variable in the function:");

console.log(a);

console.log("Second variable in the function");

console.log(b);

}

console.log("Call the function");

myFunction();

console.log("Display values outside the function");

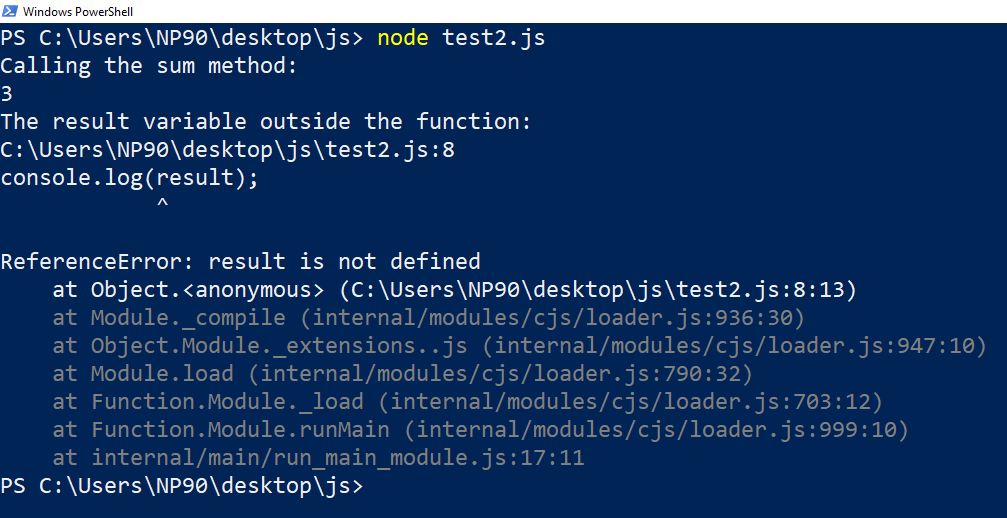
console.log("Second variable of the function");

console.log(b);

console.log("First variable of the function");

console.log(a);

Ta có kết quả như sau:



Ta thấy rằng với kiểu khai báo như trên thì complier hiểu rằng a là biến local bên trong hàm, còn b là biến global, do vậy kết thúc lời gọi hàm ta còn có thể truy cấp được biến a.

**Lý do:** Vì complier đánh giá toán tử từ phải sang trái,do vậy khai báo trên có thể được viết lại một cách dễ hiểu như sau: **var a = (b=0);**

**Giải pháp:**

function myFunction(){

var a, b;

a = b = 0;

console.log("First variable in the function:");

console.log(a);

console.log("Second variable in the function");

console.log(b);

}

console.log("Call the function");

myFunction();

console.log("Display values outside the function");

console.log("Second variable of the function");

console.log(b);

console.log("First variable of the function");

console.log(a);

**Tip#4**

Ta xét đoạn code bên dưới

var glVariable1 = 1;

glVariable2 = 2;

function testGlobalVariable(){

    glVariable3 = 3;

}

testGlobalVariable();

// Delete variables

delete glVariable1;

delete glVariable2;

delete glVariable3;

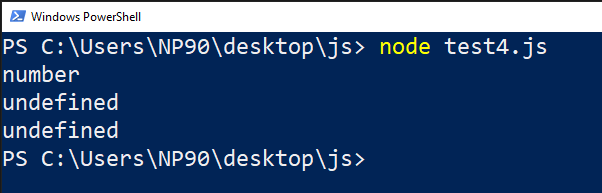
// Test delete variables

console.log(typeof glVariable1);

console.log(typeof glVariable2);

console.log(typeof glVariable3);

Ta được kết quả như sau:



Ta có kết luận như sau:

Tất cả các biến toàn cục khai báo tường minh với từ khóa var thì không thể dùng toán tử delete.

Tất cả các biến toàn cục hai báo không tường minh (không có từ khóa var) thì có thể bị delete.

**Tip#5**

Ta xét đoạn code sau:

var myName = "Tim Nguyen";

function DisplayName(){

    console.log(myName);

    var myName = "Testing";

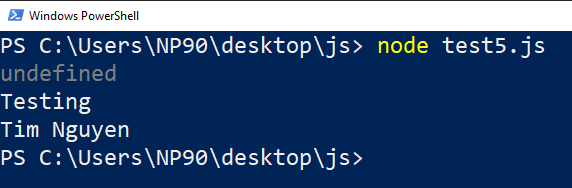
    console.log(myName);

}

DisplayName();

console.log(myName);

Ta có kết quả như sau:



Ta có kết luận như sau: do trong javascript có hoisting, hiểu một cách nôm na la nó put hết tất cả khai báo lên trên, do vậy ta sẽ thấy hiện tượng như trên.

Ta xét đoạn code sau:

function function1(){

    console.log("Global function 1");

}

function function2(){

    console.log("Global function 2");

}

function myFunction(){

    console.log(typeof function1);

    console.log(typeof function2);

    function1();

    function2();

    function function1(){

        console.log("Local function1");

    }

    function function2(){

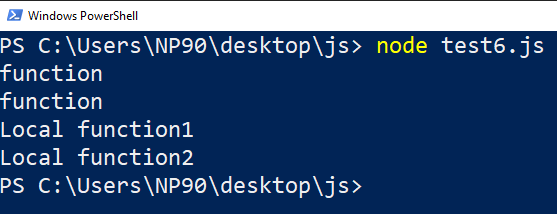
        console.log("Local function2");

    }

}

myFunction();

Kết quả chạy:



Ta thấy rằng những thằng function local phía sau nó sẽ chồng lên những thằng function mức global, nói một cách khách do hiệu ứng hoisting gây ra nó sẽ đem hết khai báo của mấy thằng functions đó lên đầu.

Bây giờ ta thay đổi đoạn code chút ít.

function function1(){

    console.log("Global function 1");

}

function function2(){

    console.log("Global function 2");

}

function myFunction(){

    console.log(typeof function1);

    console.log(typeof function2);

    function1();

    function2();

    function function1(){

        console.log("Local function1");

    }

    var function2 = function (){

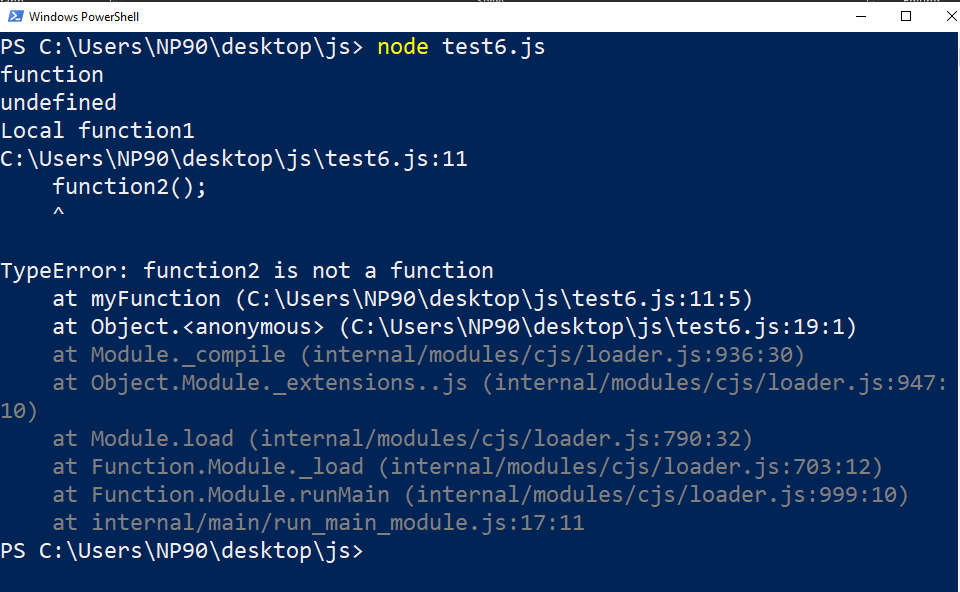
        console.log("Local function2");

    }

}

myFunction();

Ta sẽ được kết quả như sau:



Lý do của việc này là đối với function1, thì hoisting nó đem được hết định nghĩa của function đó lên phần khai báo, trong khi thằng function2 thì nó chỉ đem cái khai báo var function2 lên còn phần định nghĩa nó không, do vậy khi mình gọi đến function2() nó sẽ văng ra cục lỗi.

**Tip#6**

**Callback trong javascript.**

// Đối số của javascrip ngoài kiểu truyền thống như string, int, true/false, object thì javascript cho phép truyền vào một function.

// function as a parameter (callback)

var coffeMachine = {

    makeCoffee: function(onFinish){

*console*.log('Making coffee...');

        // Thực hiện một gì đó sau khi pha cafe

        // Validate hàm callback.

        if(typeof(onFinish!=='undefined') && typeof(onFinish)==='function')

        {

            onFinish();

        }

    }

};

var beep = function(){

*console*.log('tit tit');

}

// Truyền hàm callback vào sau khi hàm chính xong thì gọi đến hàm callback.

coffeMachine.makeCoffee(beep);

// Call back trong javascript voi tham so dau vao cho ham callback

var test = function(test2, a, b){

*console*.log('Begining');

    test2(*arguments*[1],*arguments*[2]);

};

test(function(a,b){

*console*.log(a);

*console*.log(b);

}, 'Testing', 'Testing 2');

// Kết quả là

// Begining

// Testing

// Testing 2

**Tip#6 array.map**

// array.map

// Đầu vào sẽ là một array và kết quả sẽ là một array

// ứng với mỗi phần tử của array ban đầu, map sẽ dùng function mà ta implement để xử lý với phần tử đó

var numbers = [1,2,3,4,5];

var squaredNumbers = numbers.map(function(*x*){

    return *x*\**x*;

});

*console*.log(squaredNumbers);

**Tip#7 array.filter**

// Đầu vào là một array

// nó sẽ dùng bộ lọc mà ta implement để cho ra kết quả thỏa điều kiện của bộ lọc đó.

var numbers = [1, 2, 3, 4, 5, 6];

var evenNumber = numbers.filter(function(*x*){

    // x là đại diện của từng phần tử mảng ban đầu

    return *x*%2===0;

});

*console*.log(evenNumber);

**Tip#8 array.find**

//array.find

// nó sẽ tìm và trả về phần tử đầu tiên nó tìm thấy

var numbers = [1,2,3,4];

var result = numbers.find(function(*number*){

    return *number*%2===0;

});

*console*.log(result);

// Kết quả là: 2

**Tip#9 array.reduce (chapter 1)**

//array.reduce

// liên tưởng đến dãy fibonacy

// dùng đầu ra của thằng trước làm đầu vào của thằng sau

// Tính tổng các phần tử trong mảng

var numbers = [1,2,3,4];

var result = numbers.reduce(function(*a*,*b*){

*console*.log(*a*, *b*);

    return *a*+*b*;

});

*console*.log(result);

// kết quả là

// 1, 2

// 3, 3

// 6, 4

// 10

**Tip#9 array.reduce (chapter 2)**

//array.reduce

// nếu có giá trị init thì thay vì lấy phần tử đầu tiên trong lần lặp đầu tiên thì nó sẽ lấy giá trị của thằng init thay vì phần tử đầu tiên.

var numbers = [1,2,3,4,5];

var sum = numbers.reduce(function(*a*,*b*){

*console*.log(*a*, *b*);

    return *a* + *b*;

},0);

*console*.log(sum);

// kết quả thực thi

// 0, 1

// 1, 2

// 3, 3

// 6, 4

// 10

var products = [

    {name: 'A', quantity: 2, unitPrice: 10},

    {name: 'A', quantity: 3, unitPrice: 20},

    {name: 'A', quantity: 4, unitPrice: 30}

];

var total = products.reduce(function(*currentTotal*, *product*){

    return *currentTotal* + *product*.quantity \* *product*.unitPrice;

},0);

*console*.log(total);

var items = ['Tom', 'Bill', 'Kim']

// Use reduce to make this result

// '<Tom><Bill><kim>'

**Tip#9 array.sort**

// array.sort(function(a,b){});

// => return a sorted array

// Quy tắc cần nhớ.

// if sort function

    // return value <0

    //  a will come before B

    // return value >0

    //  a will come after b

    // return 0

    //  a and b will stay unchanged.

var numbers = [2, 9, 3, 4, 1];

// => [1,2,3,4,9]

var ascNumbers = numbers.sort(function(*a*, *b*){

    // assume: a = 1, b=4

    return *a* - *b*;

});

*console*.log(ascNumbers);

var dscNumbers = numbers.sort(function(*a*, *b*){

    // assume: a = 3, b= 4;

    // expected: a comes after b

    return *b* - *a*;

});

*console*.log(dscNumbers);

**Tip#10 object methods trong javascript.**

// object methods

// Các phương thức của object.

var myDog = {

    weight: 5,

    name: "Pull",

    age: 1,

    bark: function(){ // anonymous function

*console*.log('Gau Gau');

        // trong một function nếu nó không return gì cả

        // thì nó sẽ return về undefined.

    }

};

myDog.bark();

**Tip #11 New keyword in javascript.**

// Cách khai báo theo kiểu thủ công object literal.

var mouse = {

    weight: 0.2,

    getWeight: function(){

        return *this*.weight;

    }

};

// Gọi ra

*console*.log(mouse.getWeight());

// constructor function

function Mouse(*color*){

*this*.type = 'mouse';

*this*.color = color;

}

var mouse0 = {type: 'mouse'}

*console*.log(mouse0);

// Khi khai báo theo một cách thông thường

// thì khi ta xuất ra nó chi là một object mà không có kiểu.

var mouse1 = **new** *Mouse*();

*console*.log(mouse1);

// Khi ta khai báo với từ khóa new thì khi ta xuất ra

// nó sẽ cho ta biết kiểu của nó là gì luôn.

var tom = {

    name: 'Tom',

    stomach: [],

    eat: function(*mouse*){

*this*.stomach.push(mouse);

        // return this ở đây để dùng cho việc method chaining.

        return *this*;

    }

};

var m1 = { name: 'm1'};

var m2 = {name: 'm2'};

var m3 = {name: 'm3'}

// Gọi method eat nhiều lần bằng cú pháp method chanining.

tom.eat(m1).eat(m2).eat(m3);

*console*.log(tom);

**Tip #12 prototypes trong javascript**

// Phân biệt giữa constructor function và normal function

// Constructor function

// - Được dùng với từ khóa new để tạo ra một object mới.

// - Được đặt tên theo danh từ

// - Chữ cái đầu tiên viết hoa.

// Normal function

// - Được dùng để làm một business gì đó, một việc gì đó.

// - Được đặt tên theo động từ

// - Chữ đầu tiên viết thường, chữ thứ 2 viết hoa (camelCase)

// Prototypes

// Tất cả các function điều có một thằng property là prototype.

// Trong prototype lại có môt thằng constructor function chỉ về thằng chứa prototype.

function Mouse(*color*, *weight*){

*this*.type = 'mouse';

*this*.color = color;

*this*.weight = weight;

}

*console*.log(*Mouse*.prototype);

// Kết quả là: Mouse {}

*console*.log(*Mouse*.prototype.constructor);

// Kết quả là: [Function: Mouse]

*console*.log(*Mouse*.prototype.constructor===Mouse);

// Kể quả là true, chứng tỏ rằng Mouse và thằng constructor trong prototype là 1 đứa.

// => Prototype là cái mà được chia sẻ giữa tất cả mấy thằng mà được tạo ra

// bằng từ khóa new.

// Bây giờ ta xem xét đoạn chương trình sau:

function Mouse(*color*, *weight*){

*this*.type = 'mouse';

*this*.color = color;

*this*.weight = weight;

};

*Mouse*.prototype.sleep =function(){

*console*.log('Sleeping...');

}

var jerry = **new** *Mouse*('Orange', 20);

var mickey = **new** *Mouse*('White',10);

*console*.log(jerry.sleep()===mickey.sleep());

// Kết quả là: True

// Ta thấy rằng nếu ta khai báo một method bằng prototype thì khi ta new các

// đối tượng chứa prototype đó thì nó sẽ giống nhau cái method được khai báo bằng

// prototype đó.

// Trong prototype ta cũng có thể truy cập đến các thuộc tính của đối tượng thông qua từ khóa this.

// Đại loại cũng có thể hiểu nó là một static vậy.

// Dùng prototype thì nó sẽ tiết kiệm bộ nhớ hơn.

// Nếu ta không khai báo một method bằng prototype mà để method định khai báo

// bằng prototype thành một propertiy của object đó

// Thi khi ta tạo đối tượng đó thì mỗi đối tượng sẽ có một property riêng (function định khai báo bằng prototype.)

**Tip #13 Callback hell trong javascript.**

// callback hell

var fs = require('fs');

// Read file theo kiểu đồng bộ sync

var song1 = fs.readFileSync('song1.txt',{encoding:'utf8'});

var song2 = fs.readFileSync('song2.txt',{encoding:'utf8'});

var song3 = fs.readFileSync('song3.txt',{encoding:'utf8'});

*console*.log(song1, song2, song3);

// Kết quả theo kiểu synchonous.

// Song 1 song 2 song 3

// Read file theo kiểu bất đồng bộ.

// NHƯNG VẪN CHO KẾT QUẢ TRẢ VỀ LÀ THEO KIỂU ĐỒNG BỘ.

fs.readFile('song1.txt',{encoding:'utf8'},function(*err*, *song1*){

    // function này được thực thi sau khi việc đọc file đã xong.

    // xuất kết quả của file song1.txt.

*console*.log(song1);

    // thực hiện việc đọc file thứ 2

    fs.readFile('song2.txt', {encoding:'utf8'}, function(*err2*, *song2*){

        // Xuất kết quả của file song2.txt.

*console*.log(song2);

        // Thực hiện việc đọc file song3.txt

        fs.readFile('song3.txt',{encoding:'utf8'},function(*err3*, *song3*){

            // Xuất kết quả của file song 3

*console*.log(song3);

        });

    });

});

// Kết quả trả về của đoạn chương trình trên như sau:

// Vì ta dùng callback để gọi lồng nhau nhau do vậy nó sẽ makes sure thực hiện đúng trình

// tự mà ta mong muốn, nếu ta không gọi lồng như vầy, mà gọi riêng từng cái (readfile)

// thì kết quả trả về sẽ không không như ta mong đợi.

// song 1

// song 2

// song 3

// ĐOẠN CHƯƠNG TRÌNH PHÍA TRÊN ĐƯỢC GỌI LÀ CALLBACK HELL, CÓ NGHĨA LÀ GỌI CÁC HÀM

// LỒNG LỒNG VÀO NHAU, ĐỂ KHẮC PHỤC ĐIỀU NÀY THÌ TA CÓ PROMISE RA ĐỜI!!!