Đối số truyền vào hàm được đặt trong cặp ngoặc đơn của định nghĩa hàm. Dưới đây là ví dụ về cách dùng đối số:

function myFunction(doiSo1, doiSo2, ...., doiSoN){

}

Ví dụ tiếp theo có hai đối số:

function myFunction (doiSo1, doiSo2){

   // viết mã ở đây

}

Thao tác gọi hàm đơn giản như sau:

myFunction(val1, val2);

Một trong những điểm khác biệt giữa Javascript và các ngôn ngữ khác là trong JavaScript bạn không cần chỉ ra số lượng đối số truyền vào hàm và số lượng đối số được truyền vào không bắt buộc phải khớp với danh sách đối số đã định nghĩa của hàm. Khi được gọi, hàm nhận vào một đối tượng (hoạt động như một mảng) có tên là arguments. Arguments chứa các đối số đã truyền vào hàm, giúp bạn biết số lượng đối số được truyền vào. Sau đây là ví dụ minh họa cách thức hoạt động của đối số.

function myFunction(){

     let firstArg = arguments[0];

     let secondArg =  arguments[1];

}

Cách tốt hơn là bạn lấy ra độ dài của đối tượng arguments và chạy vòng lặp qua từng đối số như sau:

function myFunction(){

      // truy xuất độ dài của đối tượng arguments

      let argLength = arguments.length;

      for(var i = 0; i < argLength; i++) {

             // Xử lý từng đối số i

      }

}

Sau đây là ví dụ hoàn chỉnh hơn minh họa tác dụng của việc sử dụng đối tượng arguments

<!doctype html>

<html>

<head>

    <title>Arguments Array</title>

</head>

<body>

<script type="text/javascript">

function myFunction() {

    let firstArg = arguments[0];

    let secondArg = arguments[1];

    alert("firstArg is: " + firstArg);

    alert("secondArg is: " + secondArg);

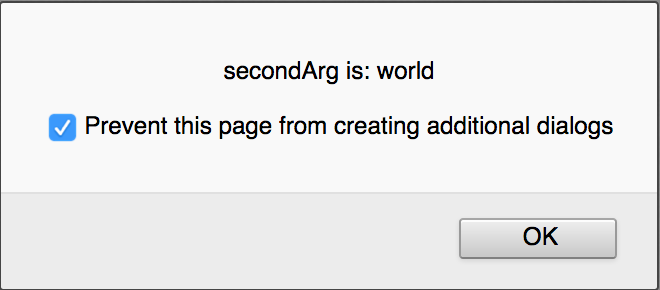
}

myFunction("hello","world");

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

Khi đoạn mã chạy, hai hộp thoại thông báo sẽ xuất hiện,





Sử dụng đối tượng arguments theo cách này, bạn có thể truy xuất bất cứ đối số nào, chứ không chỉ hai đối số kể trên.

Phạm vi biến

Đối số của hàm thường là tên biến và chúng ta không nên đặt tên đối số giống tên biến của lời gọi hàm. Tôi chủ định dùng từ *không nên* thay cho *không được* vì bạn có thể sử dụng cùng một tên cho biến trong hàm và biến ở lời gọi hàm, nhưng làm như vậy có thể gây nhầm lẫn trong đoạn mã và phạm vi của biến.

Phạm vi biến bao gồm phạm vi biến trong và ngoài hàm. Sau đây là đoạn mã về phạm vi biến:

<!doctype html>

<html>

<head>

    <title>Scoping Example</title>

    <script type="text/javascript">

       let aNewVariable = "is global.";

       function doSomething(incomingBits) {

           alert("Global variable within the function: " + aNewVariable);

           alert("Local variable within the function: " + incomingBits);

       }

    </script>

</head>

<body>

<script type="text/javascript">

    doSomething("is a local variable");

    alert("Global var outside the function: " + aNewVariable);

    alert("Local var outside the function: " + incomingBits);

</script>

</body>

</html>

Ví dụ này cho thấy bạn có thể khai báo và xác định phạm vi biến toàn cục và cục bộ bên trong và ngoài một hàm. Tuy nhiên, ví dụ này giữ cho các biến tách biệt về logic ở điểm đoạn mã không dùng cùng tên cho biến và sau đó chỉ thay đổi giá trị biến. Tiếp theo là ví dụ trong đó việc dùng cùng tên biến gây ra sự nhầm lẫn. Đây là đoạn tôi viết nhiều năm trước và nó đủ rắc rối dù không có các vấn đề về phạm vi, vì vậy tránh viết như sau:

function addNumbers(){

    firstNum = 4;

    secondNum = 8;

    result = firstNum + secondNum;

    return result;

}

result = 0;

sum = addNumbers();

Có lẽ bạn đã nhìn ra vấn đề trong đoạn mã này. Đoạn mã thiếu từ khóa var. Dù đoạn mã có khởi tạo biến result với giá trị 0 bên ngoài hàm, biến này lại được thay đổi sau lời gọi hàm addNumber(). Hàm addNumber() thay đối giá trị biến result thành 12, kết quả của phép cộng 4 và 8 bên trong hàm.

Nếu bạn thêm dòng lệnh alert để hiển thị giá trị biến result ngay sau khi khởi tạo, hộp thoại sẽ cho ra kết quả 0. Và nếu bạn thêm một lệnh alert khác để hiển thị giá trị biến result sau khi gọi hàm addNumbers(), thông báo sẽ hiện thị là 12.