

TÀI LIỆU LẬP TRÌNH WEB

**JAVASCRIPT**



October 29, 2022

LÂM VŨ THÀNH TÀI

0787903125 - lamvuthanhtai14042002@gmail.com

**MENU**

[**1.** **Giới Thiệu JS** 4](#_Toc119615582)

[**2.** **Sử dụng JS trong file HTML** 4](#_Toc119615583)

[**3.** **Biến JS** 5](#_Toc119615584)

[**4.** **Comments** 5](#_Toc119615585)

[**5.** **Hàm Built-in** 5](#_Toc119615586)

[**5.1** **Alert** 6](#_Toc119615587)

[**5.2** **Console** 6](#_Toc119615588)

[**5.2.1** **console.log** 6](#_Toc119615589)

[**5.2.2** **console.warn** 7](#_Toc119615590)

[**5.2.3** **console.error** 7](#_Toc119615591)

[**5.4** **Prompt** 7](#_Toc119615592)

[**5.5** **Set timeout** 7](#_Toc119615593)

[**5.6** **Set interval** 8](#_Toc119615594)

[**6.** **Operator (Toán Tử)** 8](#_Toc119615595)

[**6.1** **Arithmetic (Số học)** 8](#_Toc119615596)

[**6.1.1** **Prefix and Postfix (Tiền tố và hậu tố)** 9](#_Toc119615597)

[**6.2** **Assignment (Gán)** 10](#_Toc119615598)

[**6.3** **Comparison (So sánh)** 10](#_Toc119615599)

[**6.4** **Logic (logical)** 11](#_Toc119615600)

[**6.5** **String (chuỗi)** 11](#_Toc119615601)

[**7.** **Câu lệnh điều kiện If- else** 11](#_Toc119615602)

[**8.** **Kiểu dữ liệu trong JS (Data types)** 14](#_Toc119615603)

[**8.1** **Kiểu dữ liệu nguyên thủy (Primary Data)** 14](#_Toc119615604)

[**8.1.1** **Number** 14](#_Toc119615605)

[**8.1.2** **BigInt** 15](#_Toc119615606)

[**8.1.3** **String** 15](#_Toc119615607)

[**8.1.4** **Boolean** 15](#_Toc119615608)

[**8.1.5** **Undefined** 15](#_Toc119615609)

[**8.1.6** **Null** 15](#_Toc119615610)

[**8.1.7** **Symbol** 15](#_Toc119615611)

[**8.2** **Kiểu dữ liệu phức tạp (Complex Data)** 16](#_Toc119615612)

[**8.2.1** **Function** 16](#_Toc119615613)

[**8.2.2** **Object** 16](#_Toc119615614)

[**8.2.3** **Array** 16](#_Toc119615615)

[**8.2.4** **Cách kiểm tra kiểu dữ liệu** 17](#_Toc119615616)

[**9.** **Chuỗi trong JS** 17](#_Toc119615617)

[**9.1** **Tạo chuỗi** 17](#_Toc119615618)

[**9.2** **Một số cách sử dụng backslash (“ \ “)** 18](#_Toc119615619)

[**9.3** **Xem độ dài chuỗi** 18](#_Toc119615620)

[**9.4** **Cách xuống dòng trong lúc viết code** 18](#_Toc119615621)

[**9.5** **Template string ES6** 18](#_Toc119615622)

[**9.6** **Work with string** 19](#_Toc119615623)

[**9.6.1** **Find index** 19](#_Toc119615624)

[**9.6.2** **Cut string (Cắt chuỗi)** 20](#_Toc119615625)

[**9.6.3** **Replace (Ghi đè chuỗi)** 21](#_Toc119615626)

[**9.6.4** **Convert to upper case** 21](#_Toc119615627)

[**9.6.5** **Convert to lower case** 22](#_Toc119615628)

[**9.6.6** **Trim (xóa khoản cách 2 bên chuỗi)** 22](#_Toc119615629)

[**9.6.7** **Split (string to array)** 22](#_Toc119615630)

[**9.6.8** **Get a character by index** 23](#_Toc119615631)

[**10.** **Number trong JS** 23](#_Toc119615632)

[**10.1** **Cách khởi tạo** 23](#_Toc119615633)

[**10.2** **Work with Number** 24](#_Toc119615634)

[**10.2.1** **To String** 24](#_Toc119615635)

[**10.2.2** **To Fixed** 25](#_Toc119615636)

[**11.** **Mảng (Array)** 25](#_Toc119615637)

[**11.1** **Tạo mảng** 25](#_Toc119615638)

[**11.2** **Truy xuất mảng** 26](#_Toc119615639)

[**11.2.1** **Độ dài mảng** 26](#_Toc119615640)

[**11.2.2** **Lấy phần tử theo index** 26](#_Toc119615641)

[**11.3** **Work with Array** 27](#_Toc119615642)

[**11.3.1** **To string** 27](#_Toc119615643)

[**11.3.2** **Join (array to string và lấy phần chung)** 27](#_Toc119615644)

[**11.3.3** **Pop (xóa phần tử ở cuối mảng)** 27](#_Toc119615645)

[**11.3.4** **Push (thêm 1-n phần tử vào cuối mảng)** 28](#_Toc119615646)

[**11.3.5** **Shift (xóa phần tử đầu mảng)** 28](#_Toc119615647)

[**11.3.6** **Unshift (thêm 1-n phần tử vào đầu mảng)** 29](#_Toc119615648)

[**11.3.7** **Splicing (thêm, xóa, sửa phần tử trong mảng)** 29](#_Toc119615649)

[**11.3.8** **Concat (hợp nhất 2 mảng lại với nhau)** 30](#_Toc119615650)

[**11.3.9** **Slicing (Cắt 1-n phần tử trong mảng)** 31](#_Toc119615651)

[**12.** **Hàm trong JS(Function)** 32](#_Toc119615652)

[**12.1** **Định nghĩa** 32](#_Toc119615653)

[**12.2** **Các loại hàm** 32](#_Toc119615654)

[**12.3** **Cách thực thi hàm** 32](#_Toc119615655)

[**12.4** **Hàm có tham số** 33](#_Toc119615656)

[**12.4.1** **Hàm nhiều tham số:** 33](#_Toc119615657)

[**12.4.2** **Agruments** 34](#_Toc119615658)

[**12.4.3** **Giới thiệu vòng lặp for** 35](#_Toc119615659)

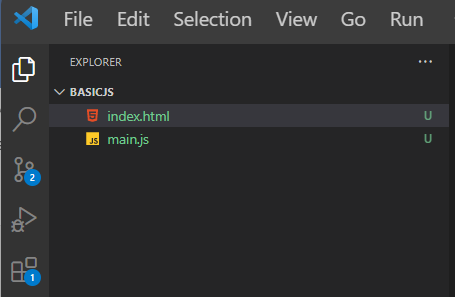
[**12.4.4** **Return trong funtion** 36](#_Toc119615660)

1. **Giới Thiệu JS**

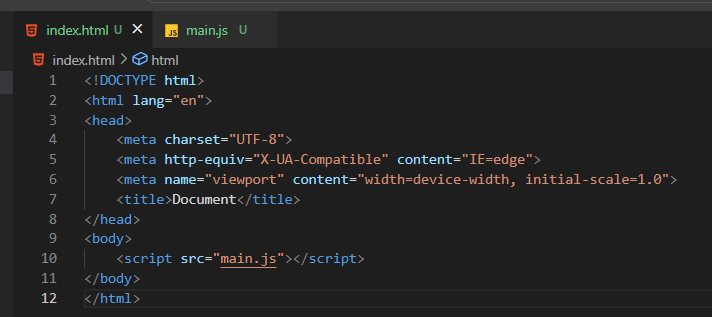
-Giúp tạo những hiệu ứng phức tạp và tạo những sự kiện, trải nghiệm người dùng tốt hơn, bên cạnh đó cũng giúp việc kết hợp giữa font end và back end dễ dàng hơn

1. **Sử dụng JS trong file HTML**

Bước 1: Tạo 1 file.js bên ngoài, có thể cùng cấp với file.html

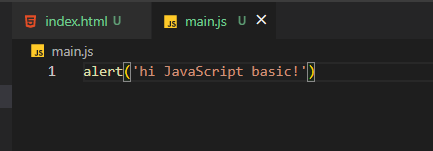
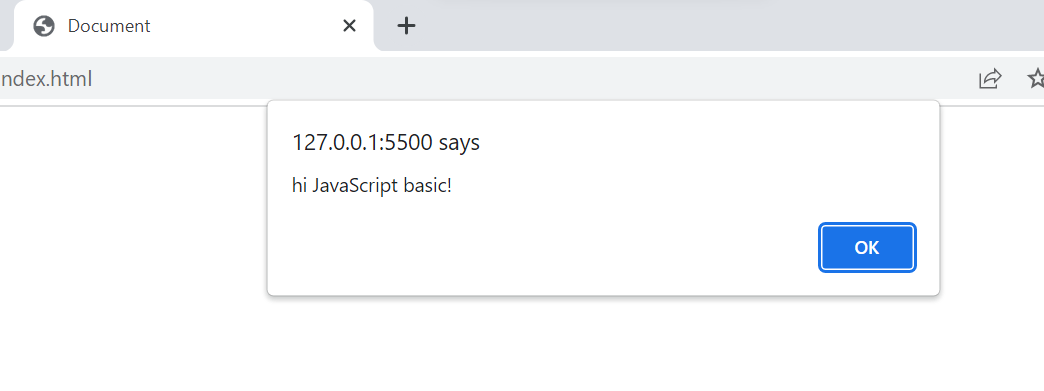


Bước 2: Trong phần body sử dụng thẻ <script></script> và link đến file.js vừa tạo

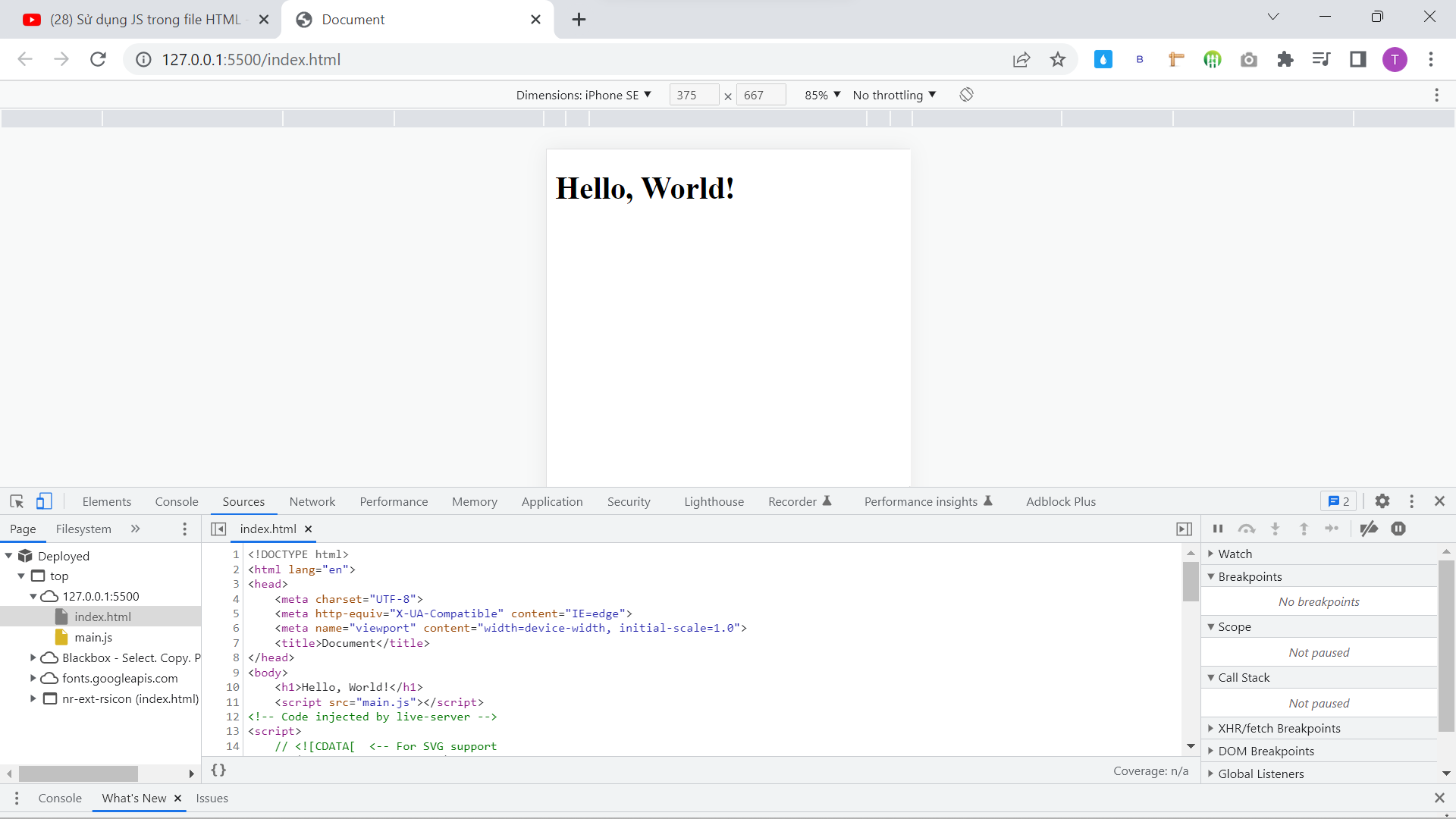


Bước 3: kiểm tra liên kết có thành công hay không

+Code vài dòng bên file.js sau đó chạy live sever check kết quả

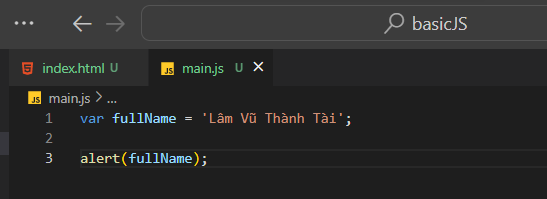
+Kiểm tra bằng Devtool



1. **Biến JS**

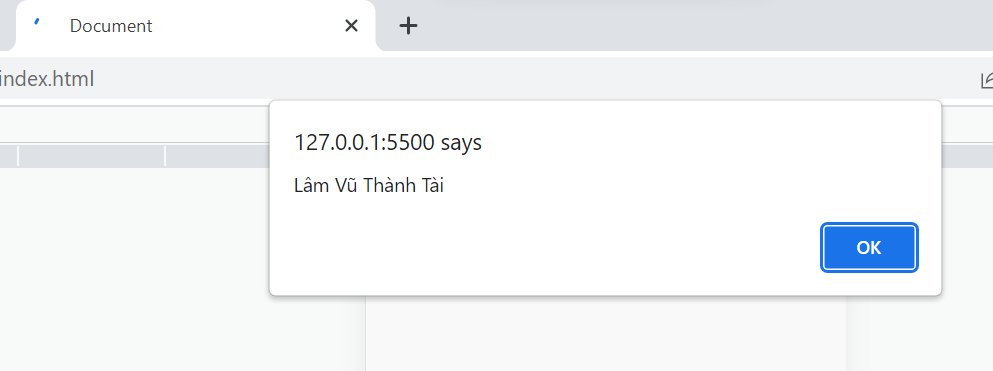
-Dùng **var** để khai báo biến:

var tênbiến = ‘Giá trị gán vào’



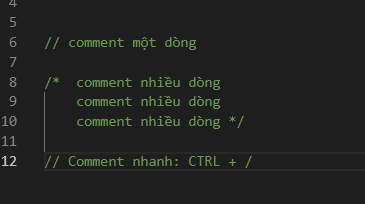
Dấu = dùng để gán giá trị cho biến

Kết quả:



1. **Comments**

-Giúp ghi chú, vô hiệu hóa hoặc giải thích code



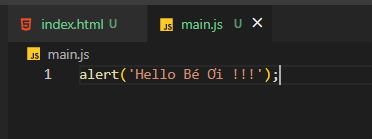
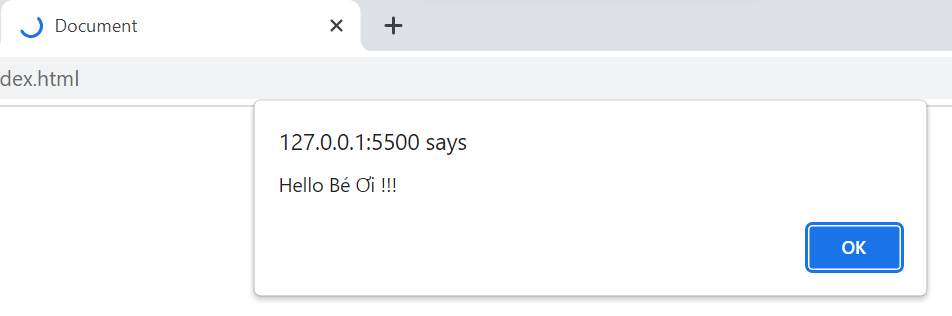
1. **Hàm Built-in**

-Là các hàm có sẵn trong JS, chỉ việc nhớ và gọi ra sài khi cần thôi !

* 1. **Alert**

-Dùng để hiển thị message (thông báo) trên web site.

-Cú pháp: alert(‘Nội dung message’);

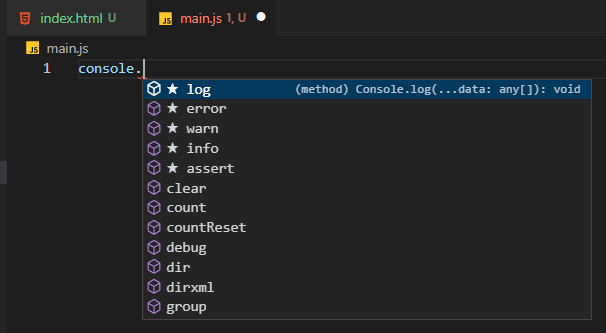
 

-Thông báo sẽ hiện cho đến khi chúng ta tắt (bấm OK).

* 1. **Console**

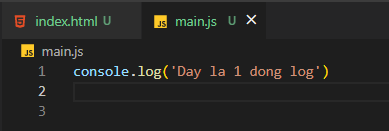
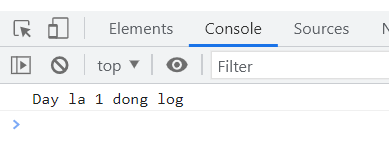
-Là một đối tượng có chứa rất nhiều các methods (phương thức).

-Cách gọi các phương thức của console ta dùng cú pháp: console + “dấu chấm”

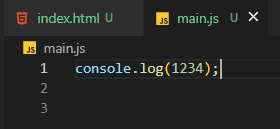
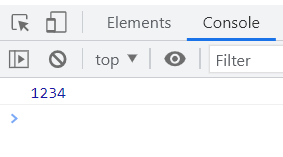


* + 1. **console.log**

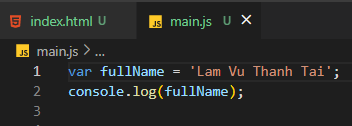
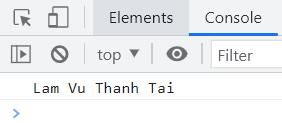
-Giúp hiển thị cảnh báo trên màn hình console (trong dev tool)

-Ngoài giá trị chuỗi ta có thể gán giá trị số

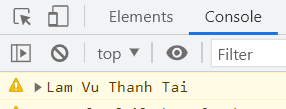
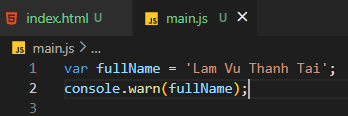
 

-Gán giá trị là biến JS

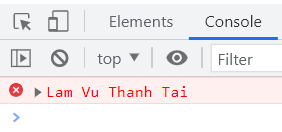
* + 1. **console.warn**

-Dùng để cảnh báo trên màng hình console

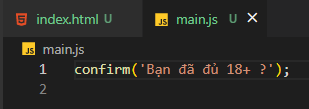
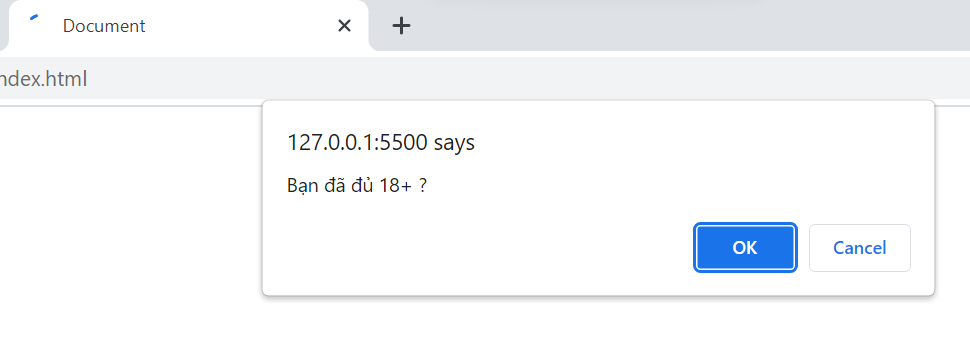


* + 1. **console.error**

-Dùng để báo lỗi trên màng hình console

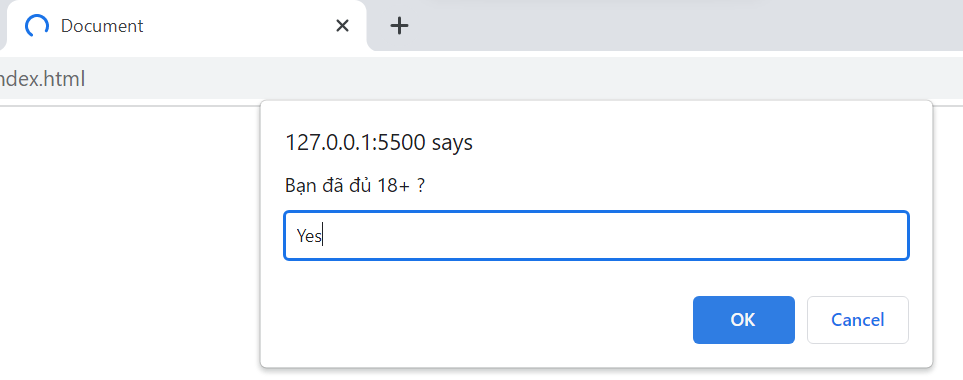
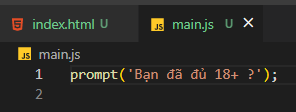
* 1. **Confirm**-Dùng để thông báo (message) và yêu cầu user xác nhận.

****

-Sẽ hiện hộp thông báo đến khi người dùng xác nhận.

* 1. **Prompt**

-Kết hợp Alert và Confirm, ngoài ra yêu cầu người dùng nhập input để xác nhận.



-Hộp thoại sẽ hiện đến khi user nhập input và click xác nhận.

* 1. **Set timeout**

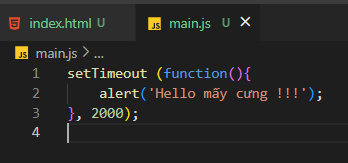
-Dùng để chạy một đoạn code trong một khoản thời gian (được tính là ms)

-Cú pháp:

setTimeout (function(){

//Code thực thi;

}, thời gian thực hiện);



Sau 2s thì sẽ hiện lên thống báo:



* 1. **Set interval**

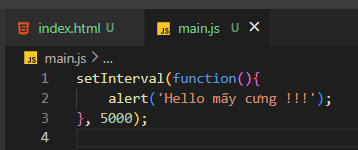
-Dùng để chạy một đoạn code trong một khoản thời gian (được tính là ms) nhưng khác với timout là nó sẽ lặp lại vô tận

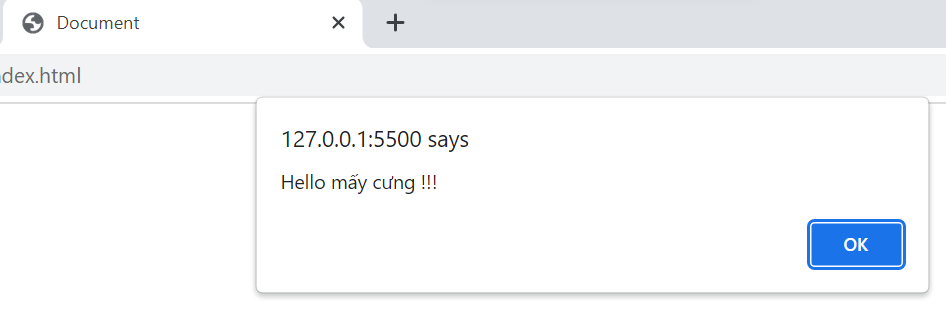
-Cú pháp:

setInterval (function(){

//Code thực thi;

}, thời gian thực hiện);

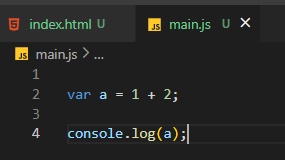
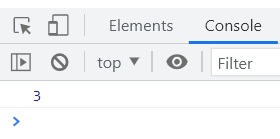
****



-Dù user có tắt thông báo thì cứ mỗi 5s sẽ hiển thị lại !

1. **Operator (Toán Tử)**
   1. **Arithmetic (Số học)**

- **Các toán tử số học** (toán tử đại số) này thực hiện trên các số - dữ liệu dạng số (cụ thể hoặc là biến). Chúng cũng gần tương tự như trong toán học thôi.

** **

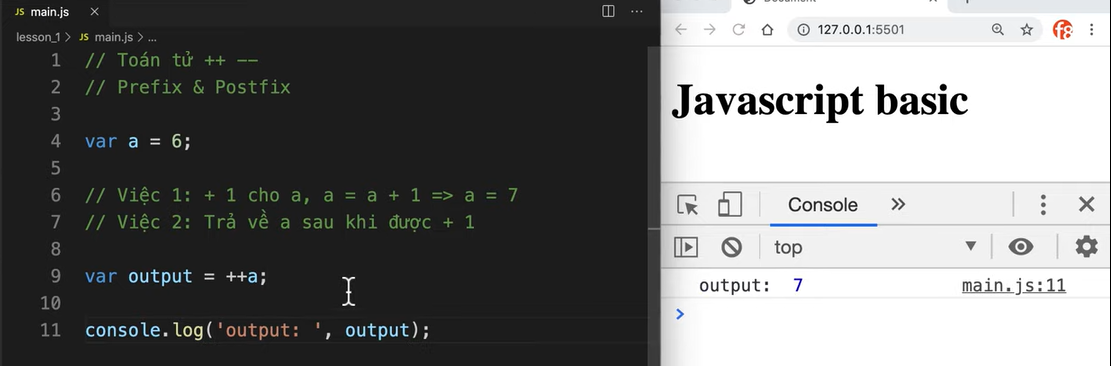
-Ngoài ra còn có các toán tử:



* + 1. **Prefix and Postfix (Tiền tố và hậu tố)**

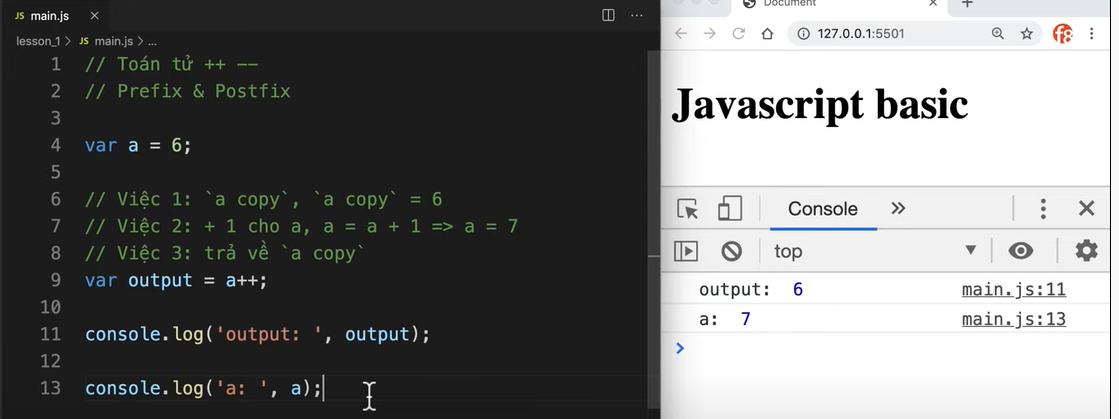
-Trong toán tử **++,** --

+Tiền tố:



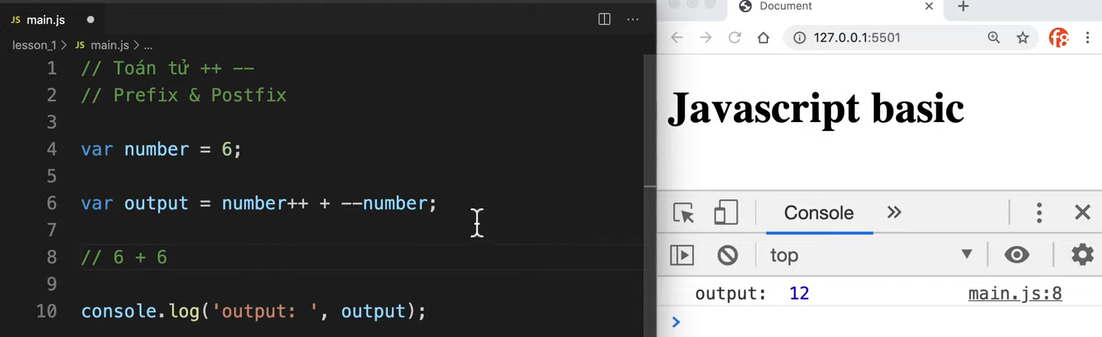
-Đối với --a; cũng giống v

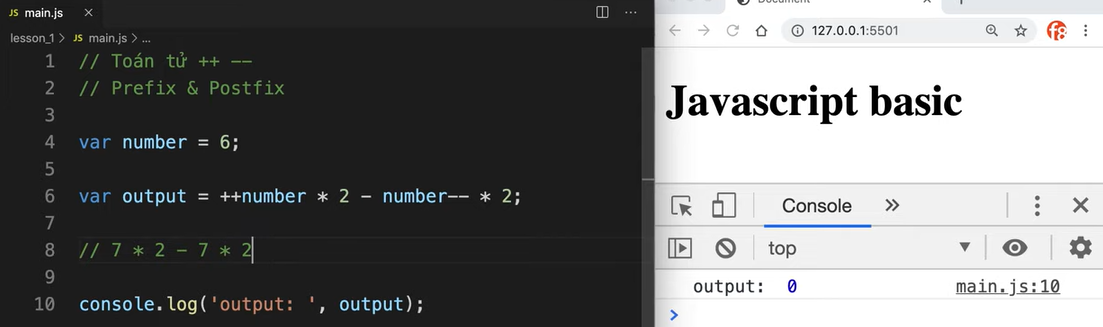
+Hậu tố:



-Đối với --a; cũng giống v

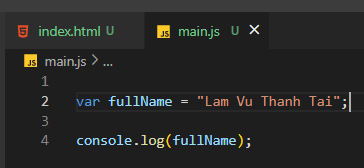
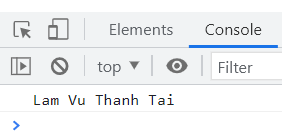
+VD:



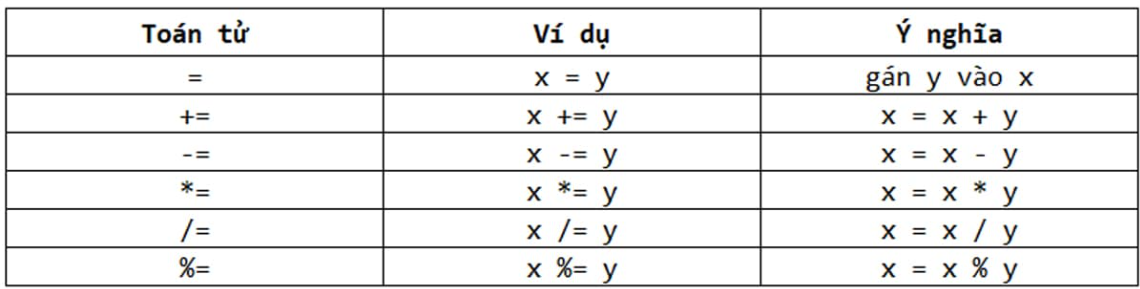


* 1. **Assignment (Gán)**

**- Toán tử gán** được dùng để **gán giá trị** ở bên phải toán tử vào biến ở bên trái toán tử.

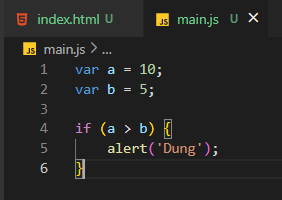
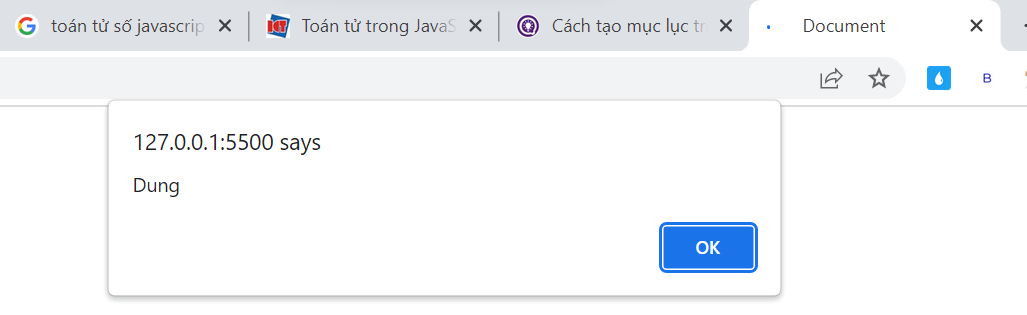
** **

-Dấu “ = ” là một trong những toán tử gán ! Ngoài ra còn:



* 1. **Comparison (So sánh)**

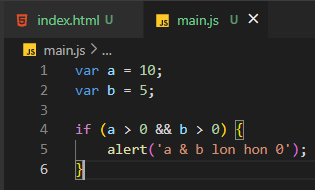
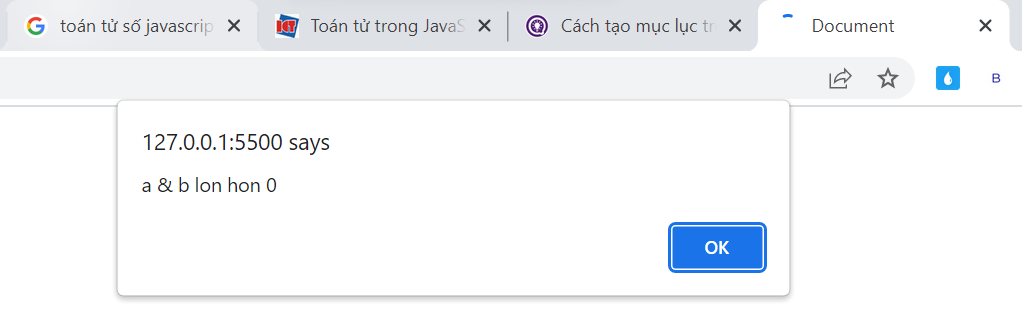
**- Toán tử so sánh trong JavaScript** sử dụng trong các biểu thức về logic để so sánh bằng, khác nhau, lớn hơn, nhỏ hơn.... Nó sẽ trả về giá trị true hoặc false.

**** ****

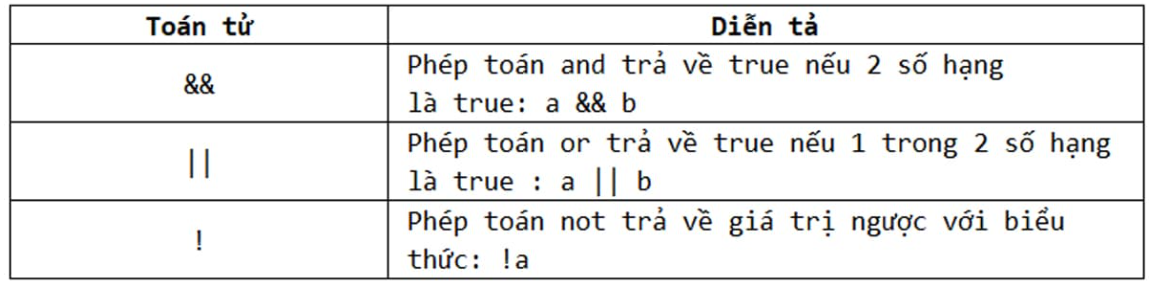
-Ngoài ra còn có các toán tử so sánh khác:

****

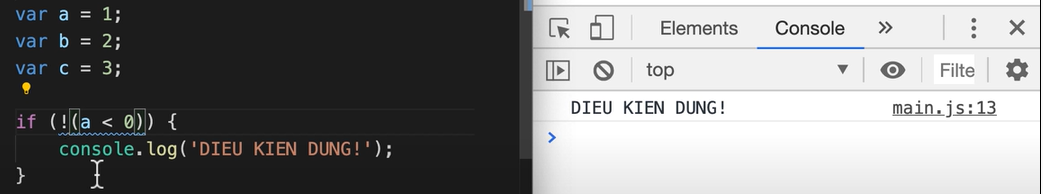
* 1. **Logic (logical)**

**** ****

**-** JavaScript cung cấp 3 loại **toán tử logic** gồm các phép toán: and, or, not.

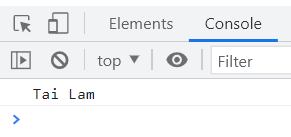
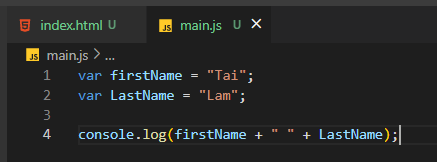
****

-Toán tử not: phủ định lại giá trị ban đầu.

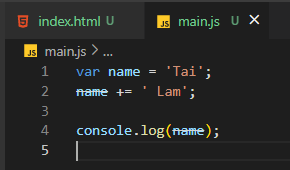
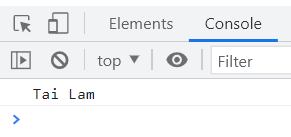
****

* 1. **String (chuỗi)**

**-**Trong JavaScript, chúng ta có thể sử dụng toán tử + để **nối hai chuỗi** lại với nhau.Nó có thể nối cả số vào chuỗi.

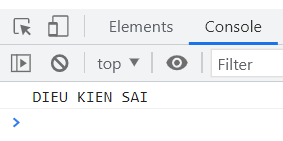
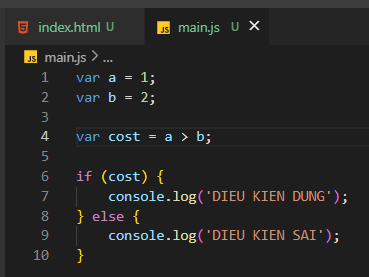
****

**-**Ngoài ra có thể kết hợp với toán tử gán.

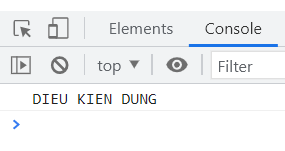
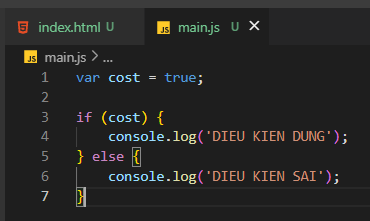
****

1. **Câu lệnh điều kiện If- else**

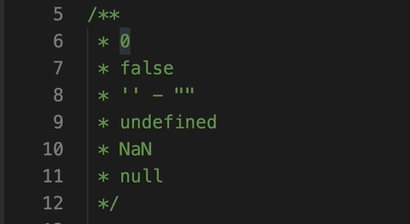
-Câu điều kiện Nếu – Thì, cấu trúc câu giống như ở các ngôn ngữ lập trình khác.

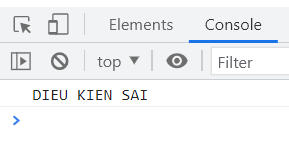
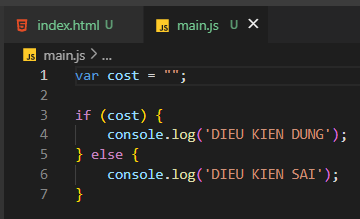
****

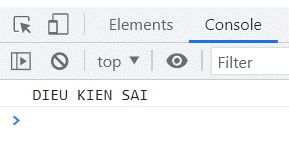
**-**Ngoài điều kiện là các toán tử ra thì còn nhận giá trị true, false.



-6 giá trị khi sử dụng JS sẽ chuyển đổi thành false.



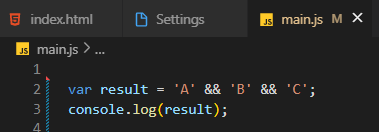


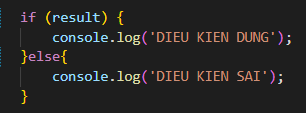
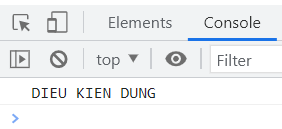


-Ngoài những giá trị này ra thì javascript sẽ convert (chuyển đổi) thành giá trị true.

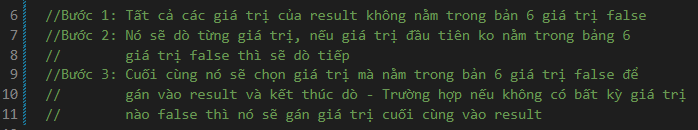
- Một số lưu ý về toán tử **And** và **OR** trong câu điều kiện if-else

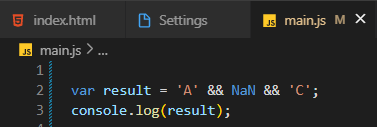
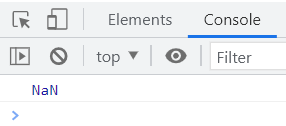
**+And**

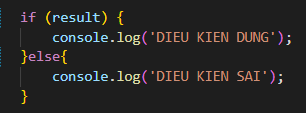
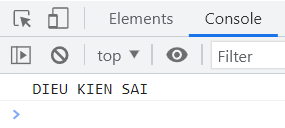
 

-Lý do in ra đáp án C là vì:



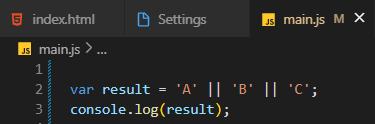
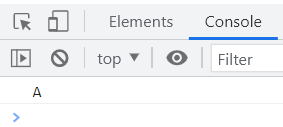
 

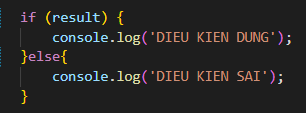
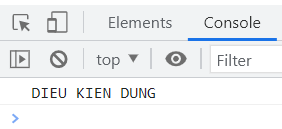
-Trường hợp có 1 giá trị nằm trong bảng 6 giá trị false thì nó sẽ gán giá trị đó cho “result” và không quan tâm vế sau.

**+OR**

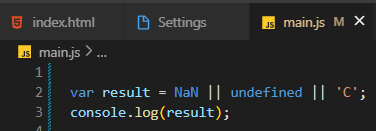
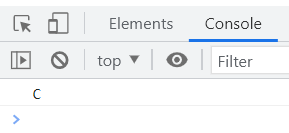
-Ngược với **And** – dò đến khi tìm giá trị false và gán vào biến

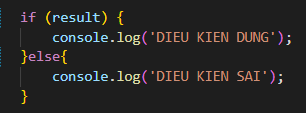
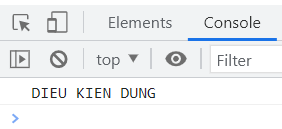
-Toán tử **OR** sẽ dò đến khi tìm được giá trị true và gán vào biến

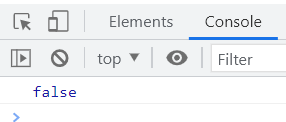
 

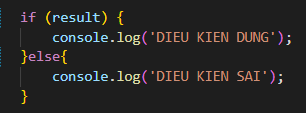
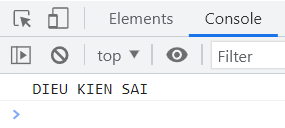
-Bởi vì A là giá trị true (không nằm trong bản 6 giá trị false) nên sẽ lấy giá trị A gán vào result và không quan tâm vế đằng sau.

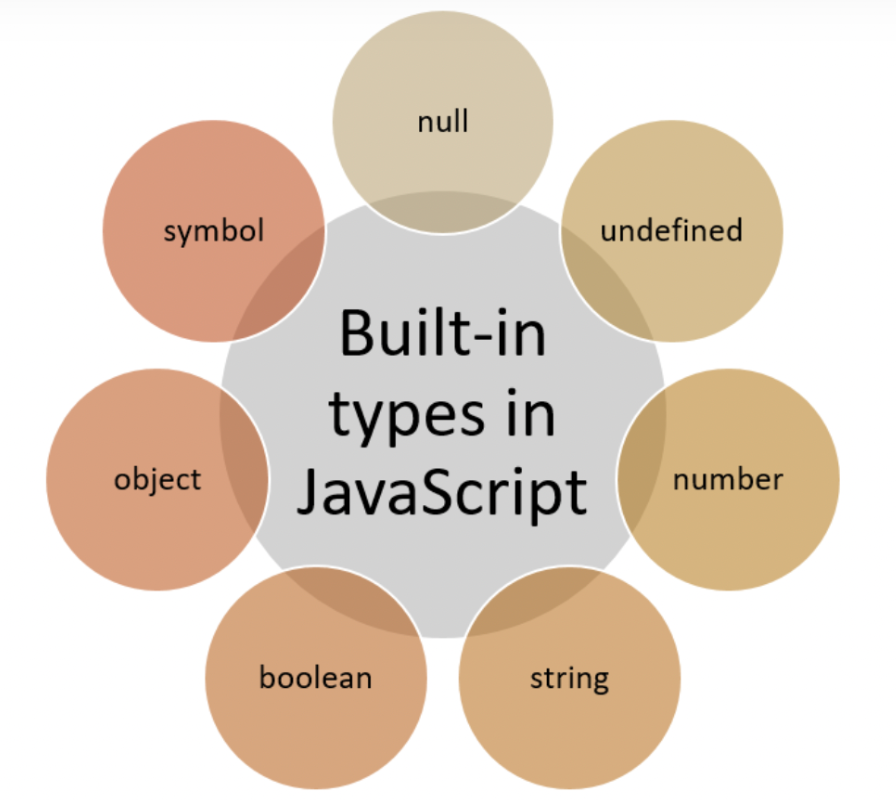
-Hai giá trị đầu là false nên sẽ được bỏ qua, đến giá trị C là giá trị true nên sẽ được gán vào result.

-Vì không có giá trị true nào nên sẽ gán false cho result

1. **Kiểu dữ liệu trong JS (Data types)**

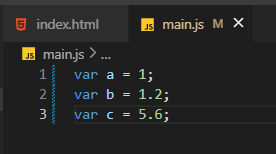
****

* 1. **Kiểu dữ liệu nguyên thủy (Primary Data)**

-Khi chúng ta gán một giá trị cho một biến thì trong RAM sẽ tạo ra một ô chứa dữ liệu của giá trị đó, nó sẽ không thể biến mất hoặc thay đổi dù chúng ta chỉnh sửa hay xóa.

* + 1. **Number**

-Gồm giá trị số nguyên, số thực.

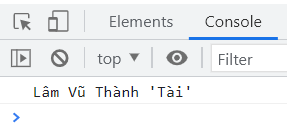
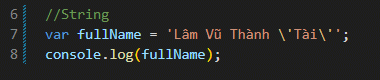
****

* + 1. **BigInt**
    2. **String**

-Là kiểu dữ liệu chứa giá trị là chuỗi, có thể điền số hoặc chữ đều đc, cách nhận biết là giá trị được bao quanh bằng dấu ngoặc đơn hoặc kép.

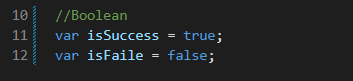
****

**-**Nếu muốn hiển thị dấu ngoặc đơn hoặc đơn hoặc kép trong một chuỗi ta sử dụng “ \ ” đặt phía trước.



* + 1. **Boolean**

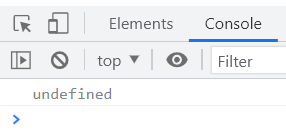
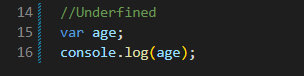
**-**Là kiểu dữ liệu chỉ biểu hiện tính “đúng” – “sai”, cách nhận biết đó là giá trị là “true” hoặc “false”.



-Thường kiểu dữ liệu Boolean người ta thường đặt biến và trước đó có chữ “is” nhằm tạo sự clean code và dễ đọc.

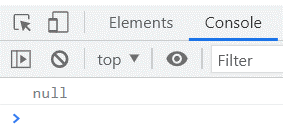
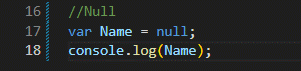
* + 1. **Undefined**

**-**Khi một biến mà không được gán bất kỳ giá trị nào, chỉ được khai báo tên biến thì ta gọi đó là undefined.



* + 1. **Null**

-Khi một biến được gán giá trị là “null” thì tức là biến đó không chứa bất kì dữ liệu gì.



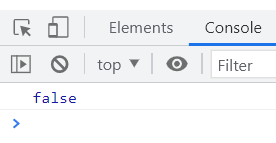
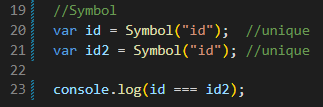
* + 1. **Symbol**

-Là kiểu dữ liệu mà dùng để mô tả một biến để biến đó là biến duy nhất

-Gán giá trị cho biến là “Symbol()”, bên trong Symbol() là phần mô tả biến đó

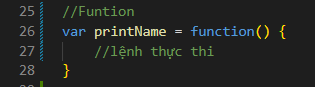


-Giá trị mô tả điền vào là duy nhất, cho dù có cùng tên biến

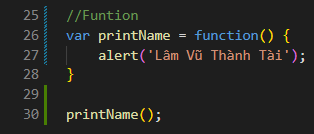
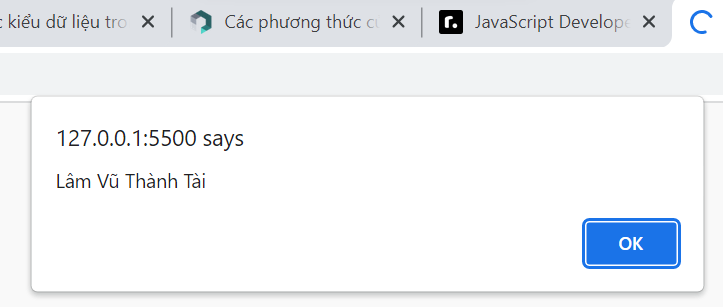


* 1. **Kiểu dữ liệu phức tạp (Complex Data)**
     1. **Function**

-Như tên gọi, là kiểu dữ liệu mà giá trị gán của biến là một hàm, bên trong hàm chứa những đoạn code thực thi một chức năng nào đó.



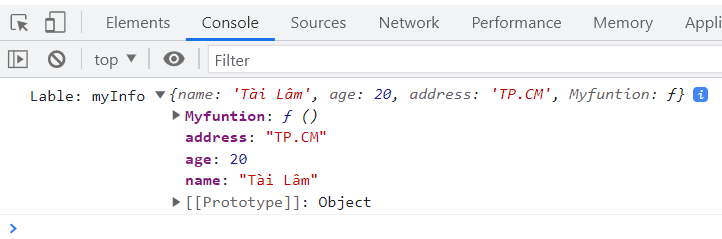
-Hàm sẽ không được chạy đến khi chúng ta gọi nó!

* + 1. **Object**

**-**Là kiểu dữ liệu mà giá trị được gán gồm “key” và “value”, trong một object thì chứa rất nhiều thứ như: giá trị (gồm nhiều kiểu dữ liệu), hàm..

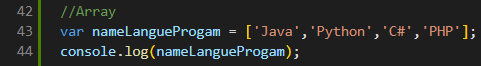
**-**Cách nhận biết đó là kiểu object là sử dụng dấu ngoặc nhọn có chấm phẩy ở cuối, ở mỗi cặp “key”-“value” cách nhau bởi dấu phẩy.

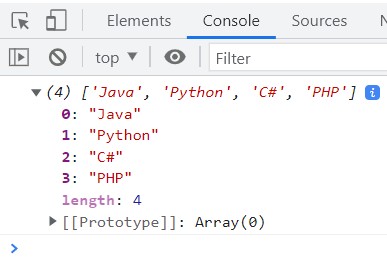
 

* + 1. **Array**

**-**Khi chúng ta cần khai báo giá trị dạng danh sách thì chúng ta sử dụng kiểu dữ liệu Array (mảng).

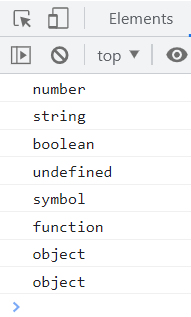
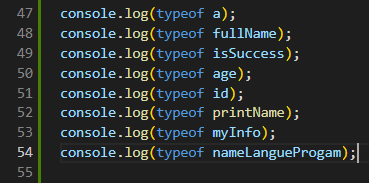
**-**Cách nhận biết kiểu Array đó dấu ngoặc vuông và các phần tử đặt trong nhấy đơn hoặc kép, bên trong ngoặc vuông có thể chứa kiểu dữ liệu số hoặc chuỗi.





* + 1. **Cách kiểm tra kiểu dữ liệu**

-Trong khi ta sử dụng hàm built-in là “console.log()” để kiểm tra thì thêm “typeof” bên trong sẽ giúp kiểm tra xem biến đó là theo kiểu dữ liệu gì

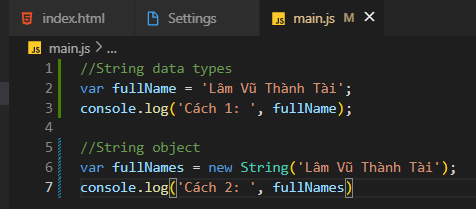


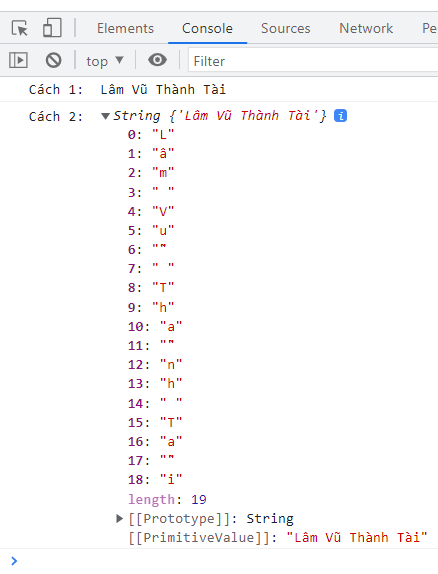
-Kiểu dữ liệu Null, Array khi hiển thị ra màn hình console thì là kiểu object (Hiện tại chưa có lời lý giải về hiện tượng này)

# **Chuỗi trong JS**

* 1. **Tạo chuỗi**

-Có 2 cách để tạo chuỗi trong JS là tạo chuỗi với kiểu dữ liệu String và tạo chuỗi bằng cách khởi tạo một đối tượng.

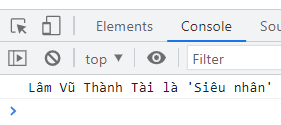




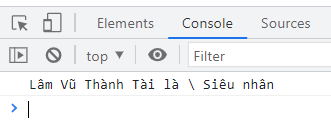
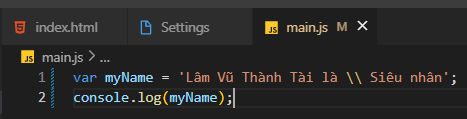
-Cách tạo chuỗi thông thường sẽ nhanh hơn là tạo đối tượng String

* 1. **Một số cách sử dụng backslash (“ \ “)**

-Để hiện dấu nháy đơn và kép

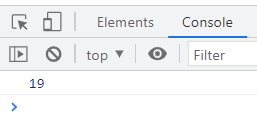
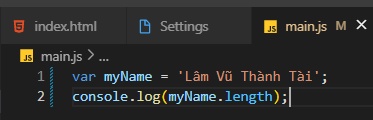
****

**-**Để hiện dấu “\” trong chuỗi



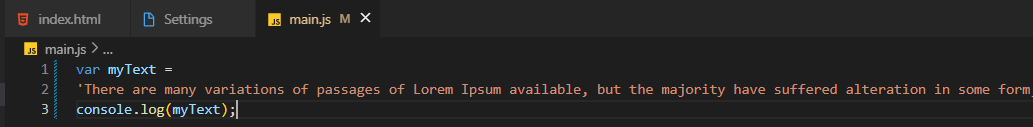
* 1. **Xem độ dài chuỗi**

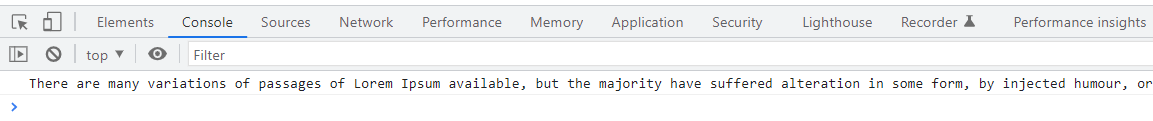
**-**Cú pháp <tên biến>.length



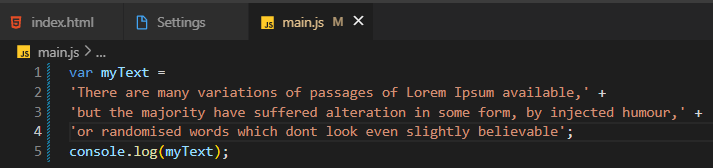
* 1. **Cách xuống dòng trong lúc viết code**

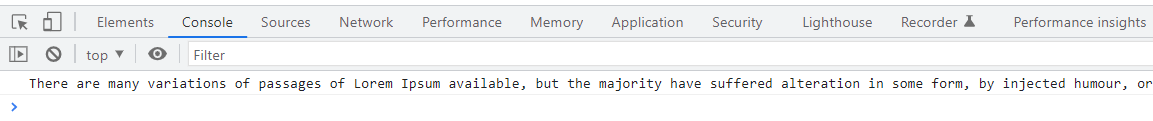
**-**Cách xuống hàng ngay sau toán tử gán





-Xuống dòng bằng cách nối các chuỗi bằng toán tử “ + “

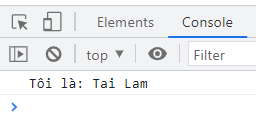
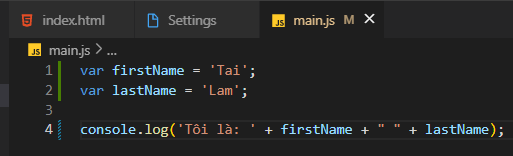




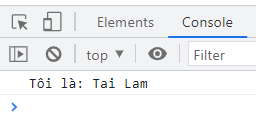
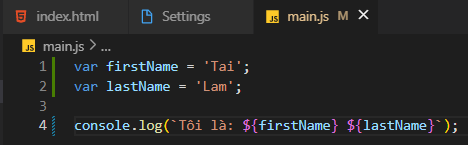
-Cả 2 cách đều mang kết quả giống nhau nhưng cách 2 sẽ khiến code trong gọn gàng và dễ nhìn hơn.

* 1. **Template string ES6**

**-**Bình thường chúng ta khai báo biến trong một chuỗi thì sẽ làm như sau.



-Cách này không có gì sai nhưng nhìn vào khá là rườm rà và không gọn.

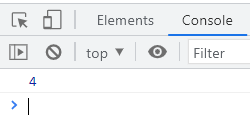
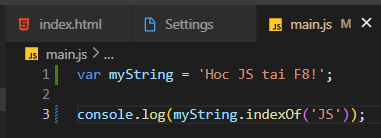


-Vì vậy template string ES6 ra đời giúp nhìn gọn hơn hẳn.

-Lưu ý giá trị đưa vào phải là string, còn ngoài ra không phải string thì nó sẽ tự convert sang string luôn ! Cú pháp là dấu `` và cách khái báo biến là ${}.

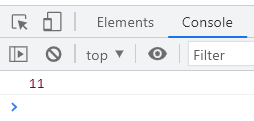
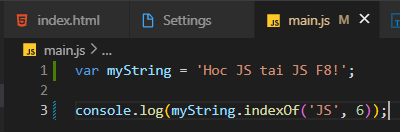
* 1. **Work with string**
     1. **Find index**

**-**Tìm vị trí của một ký tự nằm trong một chuỗi



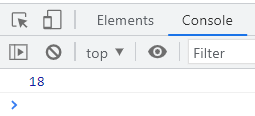
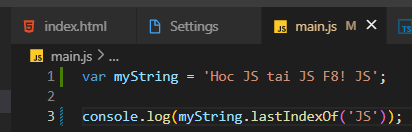
-Bắt đầu từ 0”H” 1”o” 2”c” 3” ” .

-Dùng hàm **indexOf(‘<giá trị muốn tìm>’)** một tham sốvà nhập giá trị mà mình muốn tìm ! Trường hợp này là ‘JS’ trong chuỗi myString và kết quả trả về là nằm ở vị trí số 4.



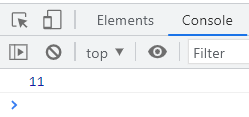
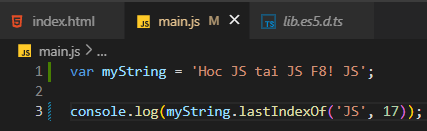
-Nếu ta có nhiều ký tự giống nhau mà buốn bỏ qua những ký tự đầu chỉ muốn biết vị trí của ký tự tiếp theo ta dùng hàm **indexOf(‘<giá trị muốn tìm>’, <vị trí>)** hai tham số và nhập vị trí lớn hơn các giá trị ban đầu.

-Trường hợp này ký tự ‘JS’ đầu tiên đang ở vị trí 4 nhưng chúng ta lại muốn biết vị trí của ký tự ‘JS’ sau vì vậy tự vị trí 4 ta đếm 5”J” 6”S” và miễn là giá trị lớn hơn hoặc bằng 6 ta sẽ tìm được vị trí của ký tự ‘JS’ tiếp theo.



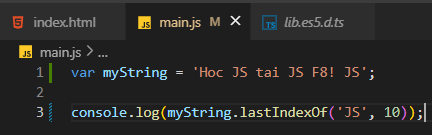
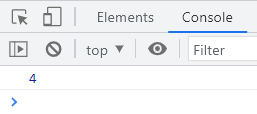
-Nếu muốn tìm vị trí của ký tự cuối cùng ta dùng hàm **lastIndexOf(‘<tên giá trị cần tìm>’)** một tham số và điền giá trị cần tìm vào

-Trường hợp này ‘JS’ ở cuối có vị trí là 18

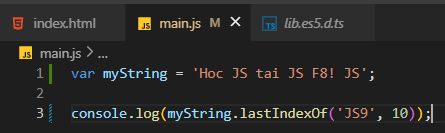
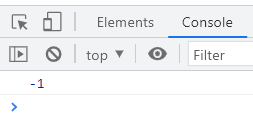


-Nếu có nhiều ký tự giống nhau thì chúng ta sử dung **lastIndexOf(‘<tên giá trị cần tìm>’, <vị trí>)** hai tham số.

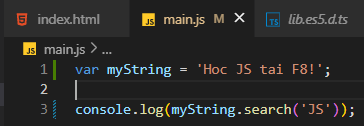
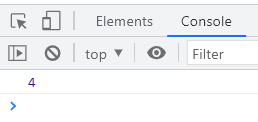
-Cách thức hoạt động là nó sẽ dựa vào vị trí mà chúng ta muốn tìm sau đó đếm ngược lại về 0 nếu ký tự nào có cùng giá trị cần tìm và gần với vị trí nhất sẽ được hiện ra.

-Trong trường hợp này chúng ta muốn tìm ký tự “JS” ở vị trí là 10, thì tại ngay vị trí số 10 nó sẽ đếm về không và phát hiện vị trí số 4 có cùng giá trị ký tự mà ta muốn tìm nên hiện ra màn hình

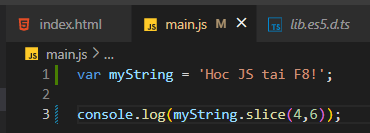
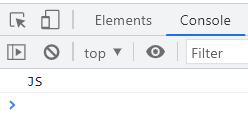
-Trường hợp ký tự mà chúng ta muốn tìm không nằm trong chuỗi thì sẽ thông báo là -1 kể cả **indexOf() và lastIndexOf().**

-Ngoài ra hàm **search()** cũng giống với **indexOf()** nhưng cách thức hoạt động lại khác nhau.

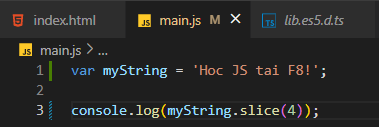
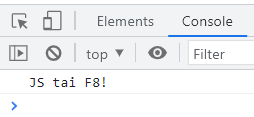
* + 1. **Cut string (Cắt chuỗi)**

**-**Cắt một phần chuỗi tại vị trí được quy định

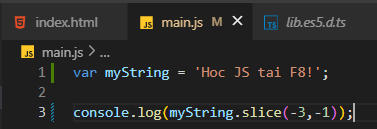
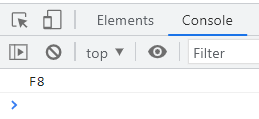
 

-Ta sử dụng hàm **slice(<vị trí đầu tiên>, <vị trí cuối cùng>)** để cắt một khoản trong chuỗi.

-Trường hợp này là cắt ở vị trí 4 và kết thúc ở vị trí 6 của chuỗi myString nên kết quả ra là “JS”.

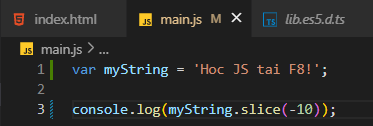
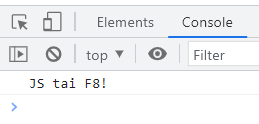
 

-Trường hợp ta sử dụng hàm **slice(<vị trí đầu tiên>)** một tham số thì sẽ cắt từ vị trí đầu tiên cho đến hết chuỗi.

-Ngoài cắt từ trái sang phải thì chúng ta còn có thể cắt từ phải sang trái bằng cách đổi vị trí đầu tiên và vị trí cuối cùng là số âm.

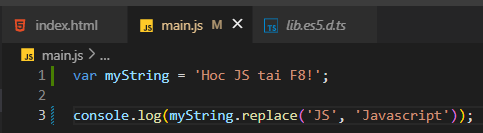
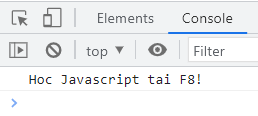
-Trong trường hợp này chúng ta muốn cắt ký tự ‘F8’ thì vị trí đầu tiên sẽ nằm sau chữ F tức là -3 (đếm từ ngoài vào 0, -1, -2, -3) và vị kí cuối cùng là nằm trước số 8 tức là -1.

-Trường hợp một tham số mà là số âm thì nó sẽ đếm từ phải sang trái và lấy vị trí cắt và phần sau của chuỗi.

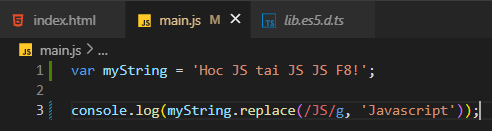
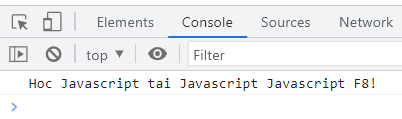
* + 1. **Replace (Ghi đè chuỗi)**

-Ghi đè ký tự mà chúng ta muốn thay đổi trong chuỗi.

-Chúng ta sử dụng hàm **replace(‘<ký tự muốn ghi đè>’, ‘<ký tự thay đổi>’)** để ghi đè lên chuỗi.

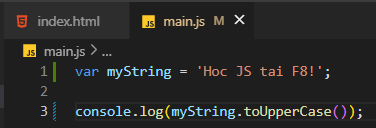
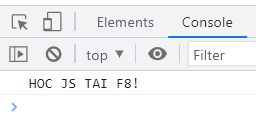
-Trong trường hợp này chúng ta đã ghi đè ký tự ‘JS’ thành ‘Javascript’.

-Trường hợp chúng ta muốn thay đổi toàn bộ ký tự có giá trị giống nhau thì sử dụng biểu thức chính quy **/<ký tự>/g.**

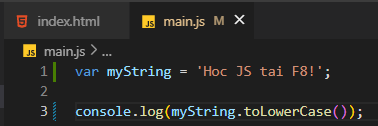
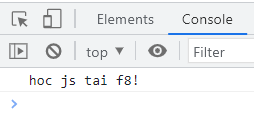
* + 1. **Convert to upper case**

-Thay đổi tất cả ký tự trong chuỗi thành chữ in hoa bằng hàm **toUpperCase().**

**** ****

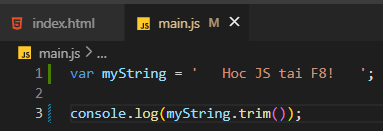
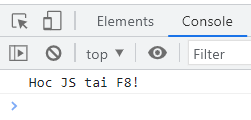
* + 1. **Convert to lower case**

**-**Thay đổi tất cả ký tự trong chuỗi thành chữ in thường bằng hàm **toLowerCase().**

**** ****

* + 1. **Trim (xóa khoản cách 2 bên chuỗi)**

**-**Loại bỏ khoản trắng, khoản cách hai bên trong input người dùng.

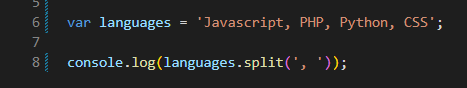
 

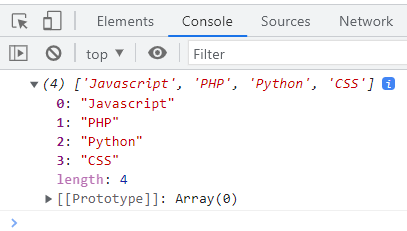
-Chúng ta cố tình ghi khoản cách hai bên của chuỗi, nhưng khi dùng hàm **trim()** nó sẽ loại bỏ toàn bộ khoản cách 2 bên đầu.

-Rất hữu ích cho sau này khi người dùng vô ý gõ khoản cách đầu dòng khi nhập password.

* + 1. **Split (string to array)**

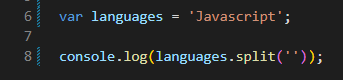
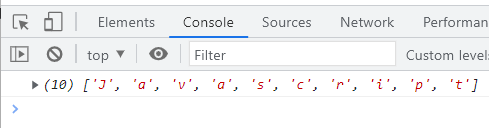
-Chuyển đổi một chuỗi thành một mảng (Array) dựa vào điểm chung.





-Chúng ta sử dụng hàm **split(‘<điểm chung>’)** để chuyển chuỗi thành mảng (Array).

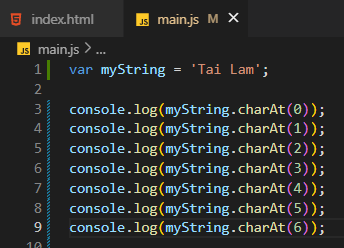
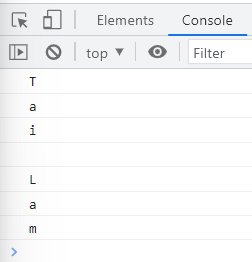
-Lưu ý là phải tìm điểm chung phù hợp để các phần tử trong chuỗi tách đều nhau ra và mảng của chúng ta nhìn gọn đẹp tránh lỗi sau này.

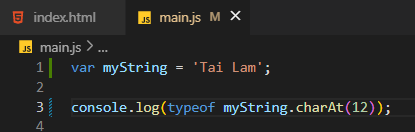
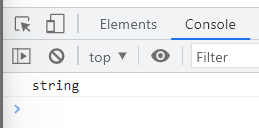
-Trường hợp chúng ta muốn mảng là những ký tự đơn thì điểm chung sẽ để trống.

* + 1. **Get a character by index**

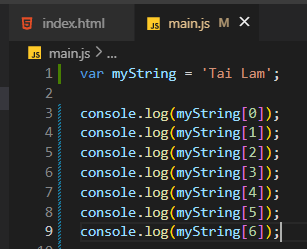
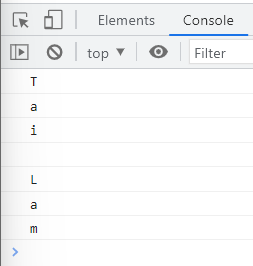
**-**Tìm ký tự tại vị trí index được chỉ định.

-Ta sử dụng hàm **charAt(<vị trí ký tự muốn lấy>)** để lấy ký tự tại vị trí mà chúng ta muốn.

-Trường hợp vị trí index mà quá lớn thì sẽ trã về một chuỗi rỗng.

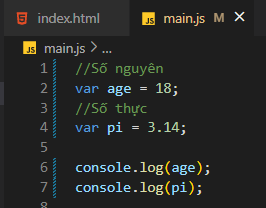
-Ngoài hàm **charAt()** ta cỏ thể sử dụng dấu **[]** để tìm vị trí ký tự.

1. **Number trong JS**

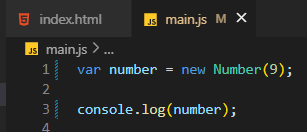
-Giới thiệu chi tiết về kiểu dữ liệu number trong JS và cách làm việc với nó.

* 1. **Cách khởi tạo**

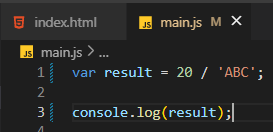
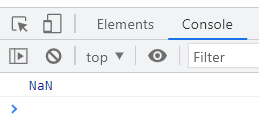
-Cách 1: khai báo theo kiểu dữ liệu number, cách nhận biết là số nguyên, số thực



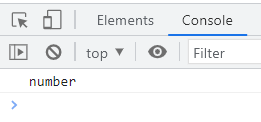
- Cách 2: Khai báo đối tượng **Number();**



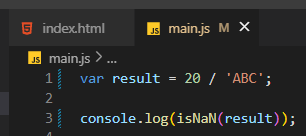
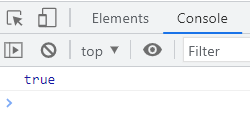
-Cả 2 cách đều có thể dùng để khai báo kiểu dữ liệu số tuy nhiên cách 1 sẽ nhanh và đúng hơn ! vì cách 2 khi tạo đối tượng nó không còn là kiểu dữ liệu number nữa mà là kiểu dữ liệu object.

-Trường hợp người dùng nhập sai giá trị như hình trên (lấy số chia chuỗi) thì JS sẽ hiểu result đó là một số không hợp lệ.

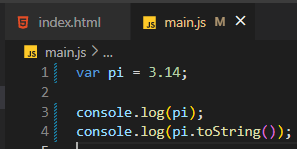
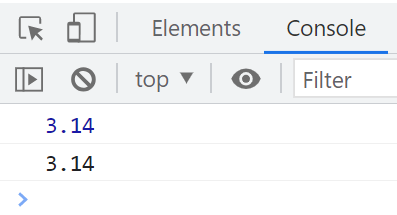
-Dù là nhập phép tính sai nhưng kiểu trả về là number

-Ngoài ra ta có thể dùng hàm **isNaN(<Tên biến hoặc giá trị>);** để xem kiểu number đó có phải là một số không hợp lệ không !

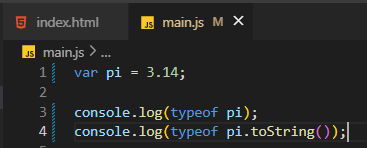
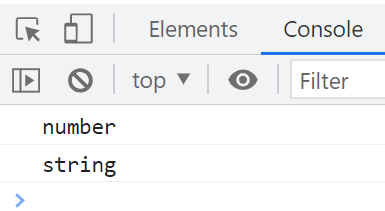
* 1. **Work with Number**
     1. **To String**

-Chuyển đổi kiểu number sang kiểu string.

-Để chuyển sang string ta dùng hàm **toString().**

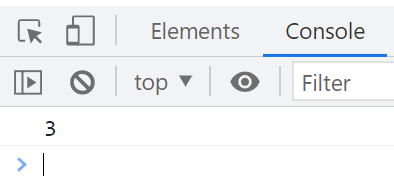
-Thường khi number chuyển sang string thì sẽ không có màu mè gì.

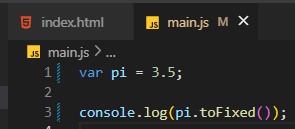
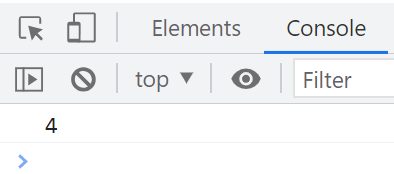
-Để chắc rằng đã chuyển kiểu number sang string ta dùng typeof để check.

* + 1. **To Fixed**

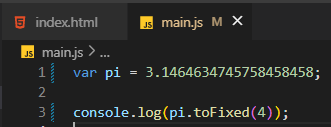
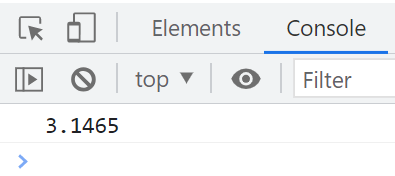
**-**Làm tròn số thập phân thành bao theo chỉ định

-Chúng ta sử dụng hàm **toFixed()** để có thế làm tròn số thập phân

-Lưu ý khi số thập phân phía sau nhỏ hơn 0.5 thì sẽ bỏ phần dư chỉ lấy phần nguyên, ngược lại nếu lớn hơn hoặc bằng 0.5 thì sẽ làm tròn cả số nguyên lên.

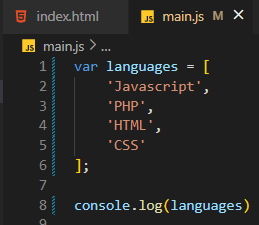
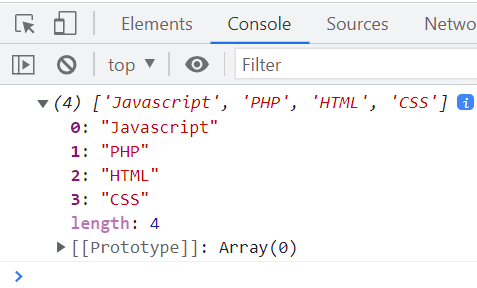
-Ngoài ra chúng ta có thể điều chỉnh bao nhiêu số dư phía sau tùy ý bằng cách gán giá trị vào hàm **toFixed().**

1. **Mảng (Array)**

-Khi muốn lưu trữ một danh sách có nhiều phần tử thì chúng ta không thể tạo ra từng biến một, như vậy rất tốn thời gian vì vậy Array sẽ giúp chúng ta việc đó.

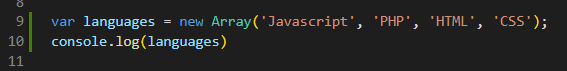
* 1. **Tạo mảng**

**-Cách 1:** Tạo mảng theo kiểu dữ liệu Array.

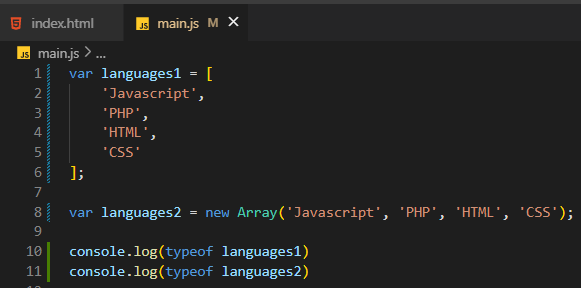
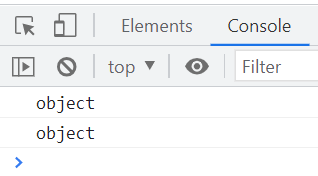
-Khi đặt tên biến có kiểu dữ liệu là Array ta nên dùng số nhiều (thêm s vào cuối) đễ dễ phân biệt đó là mảng.

-**Cách 2:** Tạo mảng bằng cách tạo đối tượng **Array()** mà JS đã cung cấp sẵn.

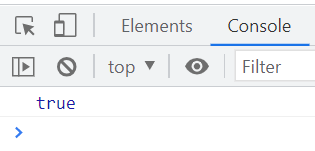
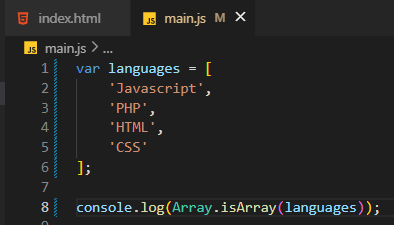




-Cả 2 cách đều có thể tạo mảng nhưng khuyến khích sài cách 1 hơn vì ít tốn tài nguyên và đơn giản code hơn.

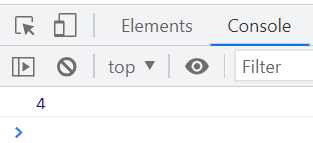
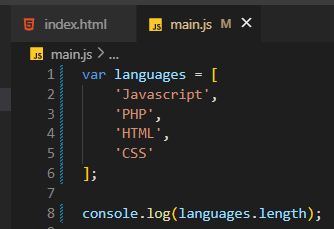
 

-Vì kiểu dữ liệu Array trong JS được hiểu là kiểu object vì vậy chúng ta hay thường nhầm lẫn 2 kiểu Array với object này với nhau trong lúc code.



-Tuy nhiên chúng ta có thể kiểm tra xem biến đó có phải là Array hay không bằng cách sử dụng hàm **isArray(<tên biến>)** nằm trong đối tượng Array có sẵn của JS.

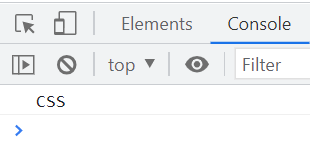
* 1. **Truy xuất mảng**
     1. **Độ dài mảng**

****

-Chúng ta có thể biết chiều dài của mảng là bao nhiêu khi sử dụng thuộc tính **length.**

**-**Trường hợp trên trong mảng đang có 4 phần tử nên kết qua in ra sẽ là 4.

* + 1. **Lấy phần tử theo index**

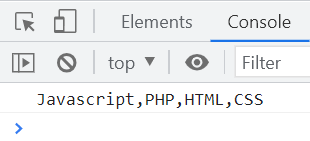
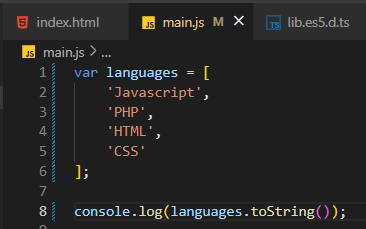
****

**-**Sử dụng tên biến + ngoặc vuông và điền giá trị tại vị trí phần tử sẽ trả về phần tử mà chúng ta cần tìm.

**-**Trong trường hợp này là tìm vị trí index = 3 trong mảng languages, bắt đầu từ 0 chúng ta đếm xuống vị trí 3 và kết quả trả về là CSS

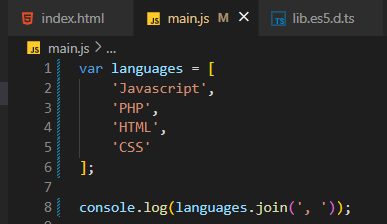
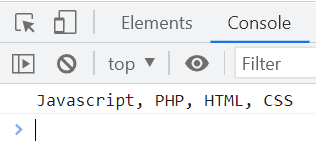
* 1. **Work with Array**
     1. **To string**

**-**Chuyển kiễu dữ liệu Array thành string

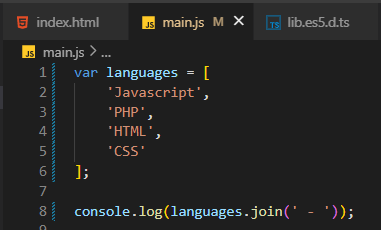
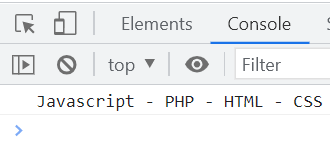


* + 1. **Join (array to string và lấy phần chung)**

-Giống như phương thức **split()** trong String, nhưng là chuyển đổi từ kiểu dữ liệu Array sang String

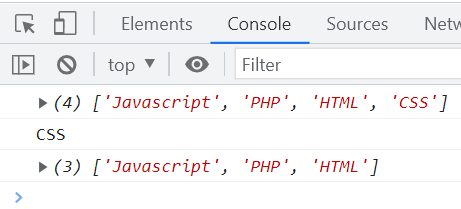
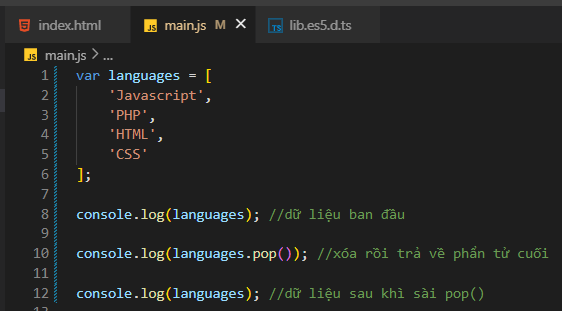
 

-Cũng phải chú ý lấy điểm chung của Array để code trong gọn gàng hơn.

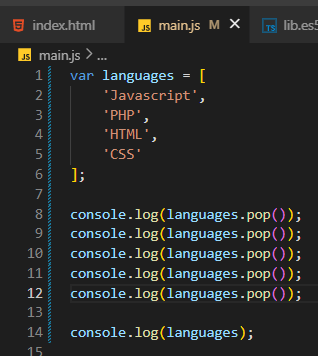
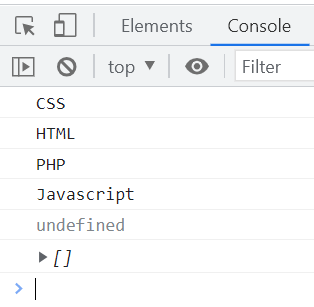
 

* + 1. **Pop (xóa phần tử ở cuối mảng)**

-Phương thức **pop()** dùng để xóa phần tử ở cuối mảng và trả về phần tử đã xóa.

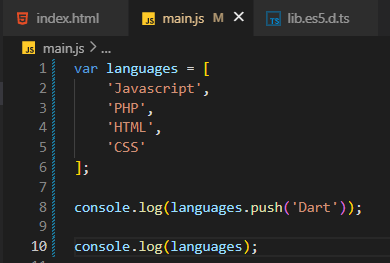
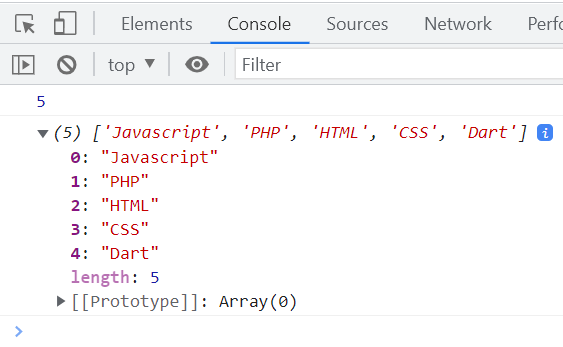


-Trường hợp không có phần tử nào trong mảng mà vẫn sài hàm **pop()** thì sẽ trả về là undefind và một mảng rỗng.

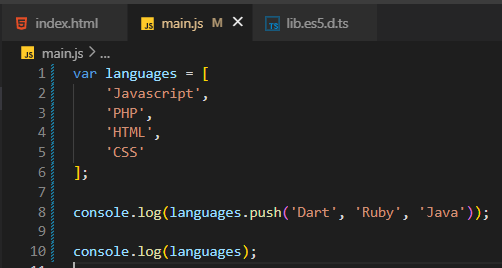
 

* + 1. **Push (thêm 1-n phần tử vào cuối mảng)**

**-**Thêm một hoặc nhiều phần tử vào cuối mảng và trả về độ dài mới của mảng.

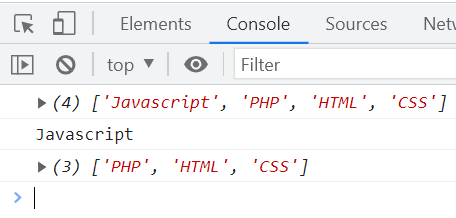
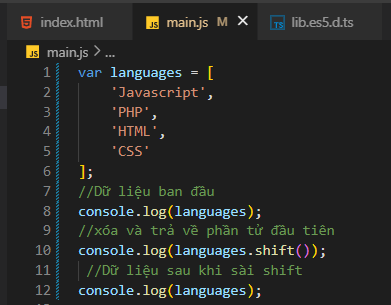
-Lúc đầu chỉ có 4 phần tử sau đó chúng ta sài hàm **push(<tên phần tử thêm>)** và thêm một phần tử ‘Dart’ vào mảng. Sau đó nó sẽ trả về độ dài mới của mảng bây giờ là 5 phần tử và phần tử đc thêm ở cuối mảng.

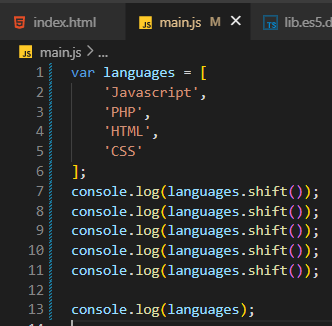
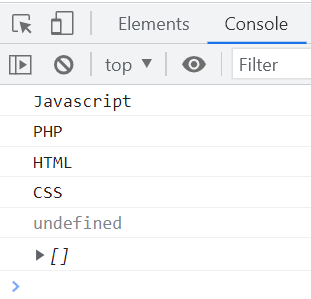
-Ngoài ra chúng ta có thể thêm nhiều phần tử cùng lúc vào hàm **push().**

* + 1. **Shift (xóa phần tử đầu mảng)**

-Phương thức **shift()** dùng để xóa phần tử ở đầu mảng và trả về phần tử đã xóa.

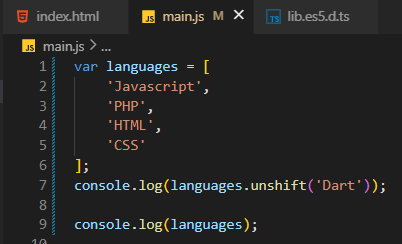
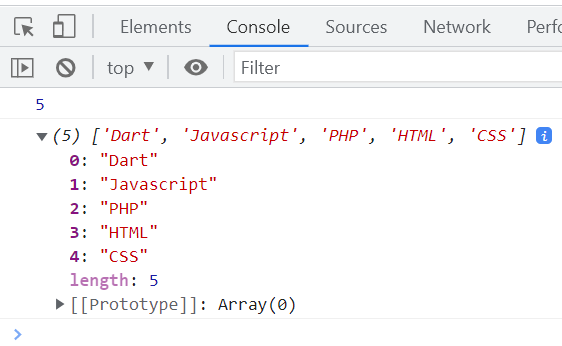


-Trường hợp không có phần tử nào trong mảng mà vẫn sài hàm **shift()** thì sẽ trả về là undefind và một mảng rỗng.

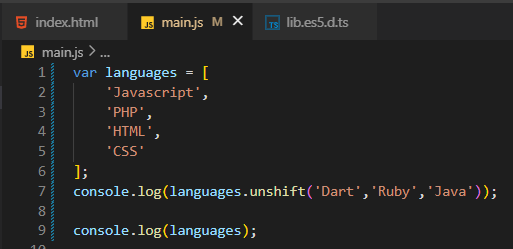
 

* + 1. **Unshift (thêm 1-n phần tử vào đầu mảng)**

**-**Cách thức hoạt động củng giống như hàm **push()** nhưng khác nhau là hàm **unshift()** thêm một hoặc nhiều phần tử ở đầu mảng và trả về độ dài mới của mảng.

-Lúc đầu chỉ có 4 phần tử những sau khi chúng ta sử dụng hàm **unshift()** và thêm một phần tử ‘Dart’ vào mảng, kết quả trả về là độ dài mới của mảng và phần tử được thêm trên đầu.

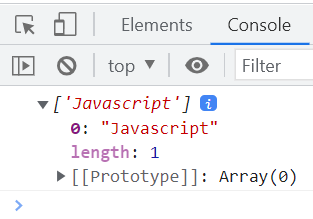
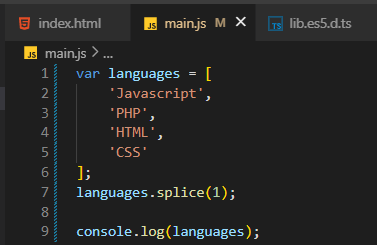


-Ngoài ra chúng ta có thể thêm nhiều phần tử cùng lúc trong hàm **unshift().**

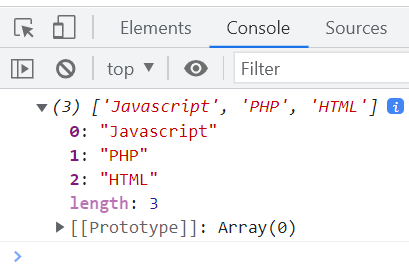
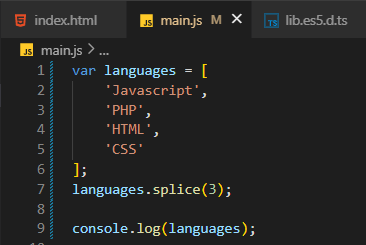
* + 1. **Splicing (thêm, xóa, sửa phần tử trong mảng)**

-Dùng hàm **splice()** để có thể thêm, xóa, sửa các phần tử trong mảng

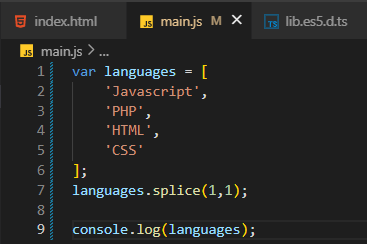
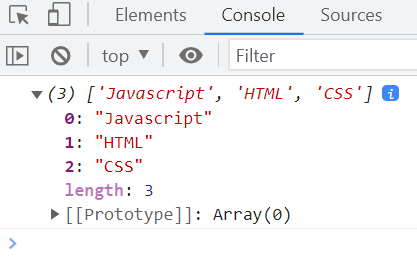
**+splice(< vị trí index>,<số phần tử muốn xóa>,<thêm phần tử mới>)**

****

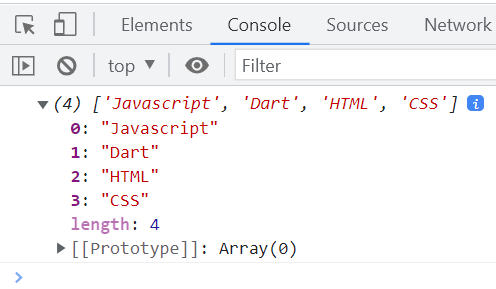
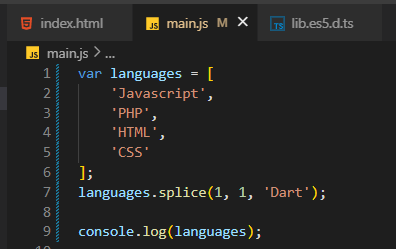
**-**Trả về một phần tử (phần tử đầu tiên) và xóa hết các phần tử phía sau



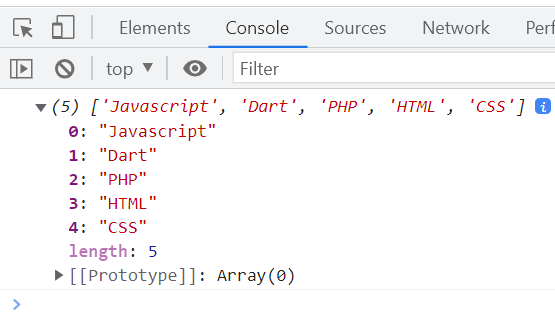
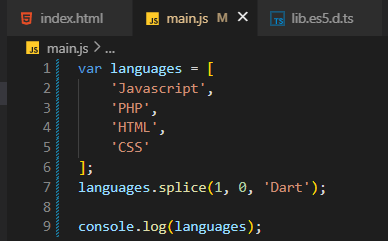
-Trả về 3 phần tử trong mảng (từ đầu đếm đến 3) và xóa hết các phần tử phía sau

** **

-Tại vị trí 1 xóa một phần tử ra khỏi mảng



-Tại vị trí 1 xóa một phần tử và thêm vào phần tử có giá trị ‘Dart’ vào mảng

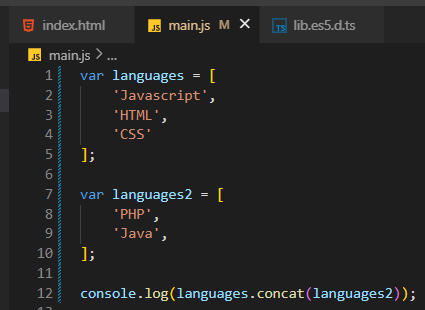
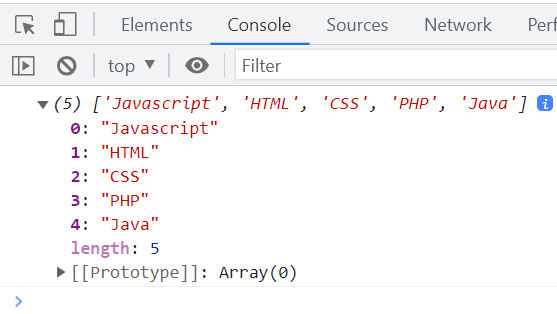


-Tại vị trí index=1, không xóa phần tử nào, thêm phẩn tử ‘Dart’ vào mảng.

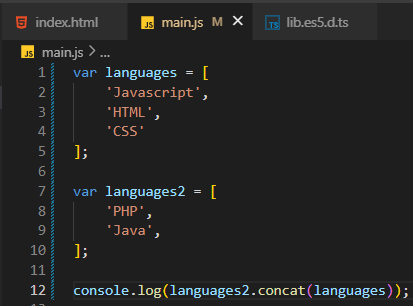
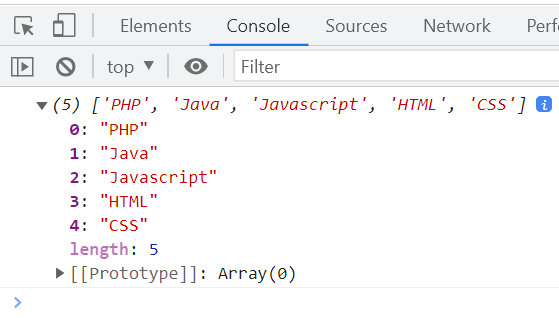
-Ngoài ra có thể thêm vào nhiều phần tử ở tham số thứ 3.

* + 1. **Concat (hợp nhất 2 mảng lại với nhau)**

-Dùng để nối 2 mảng lại với nhau thông qua hàm **concat().**

****  ****

-languages là mảng gốc, languages2 là mảng cần nối đến mảng gốc.

** **

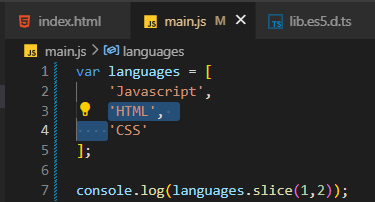
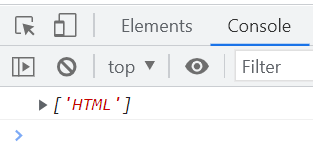
**-**Việc chọn mảng nào là gốc mảng nào là mảng cần nối sẽ ảnh hưởng đến vị trí của các phần tử.

**-**Trong trường hợp này languages2 là mảng gốc các phần tử sẽ ở đầu mảng.

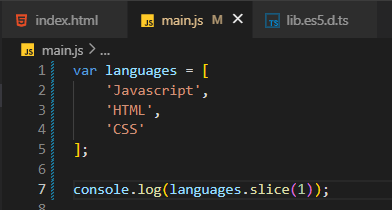
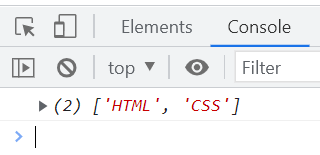
* + 1. **Slicing (Cắt 1-n phần tử trong mảng)**

**-**Dùng để cắt 1-n phần tử trong mảng thông qua hàm **slice().**

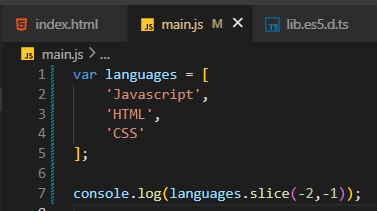
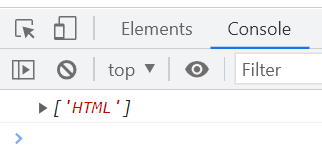
**-slice(<vị trí đầu tiên>,<vị trí cuối cùng>).**

**** ****

-Tại vị trí ban đầu = 1 và vị trí kết thúc = 2 ta cắt được phần tử ‘HTML’

**** ****

-Tại vị trí ban đầu = 1, tham số thứ 2 không có tức là cắt từ vị trí ban đầu = 1 đến cuối mảng.

**** ****

-Ngoài cắt từ trên xuống ta có thể cắt từ dưới lên bằng cách truyền tham số index = số âm.

-Cách sử dụng **slice()** của Array giống y chang String.

# **Hàm trong JS(Function)**

* 1. **Định nghĩa**

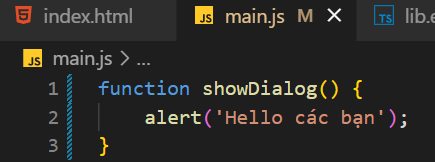
**-**Hàm là một khối mã (những dòng code được đặt trong ngoặc nhọn) và làm một việc cụ thể (hàm thường thực thi một chức năng duy nhất như: in ra màn hình, tính tổng, sắp xếp, thêm, xóa, sửa, tìm kiếm….).

* 1. **Các loại hàm**

**-**Trong JS hiện nay có 2 loại hàm hay sử dụng:

****

**+**Hàm Built-in (những hàm có sẵn trong JS).

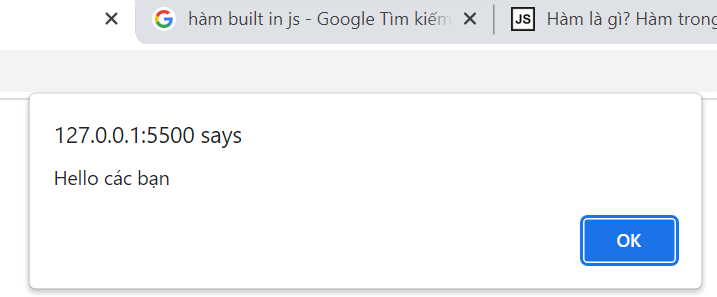


**+**Hàm tự định nghĩa (là những hàm mà do chính tay chúng ta những developer tự viết ra)

-Cách đặt tên hàm cũng giống như các ngôn ngữ lập trình khác.

* 1. **Cách thực thi hàm**

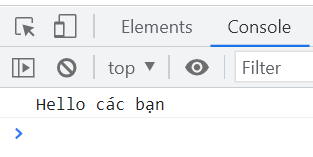
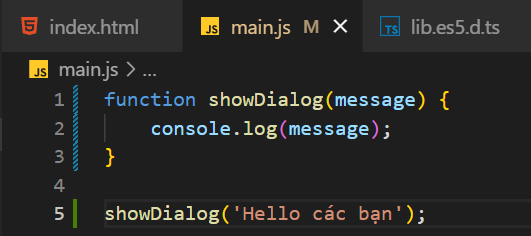
**-**Hàm sau khi được khởi tạo và viết code sử lý bên trong vẫn chưa có thể thực thi được.

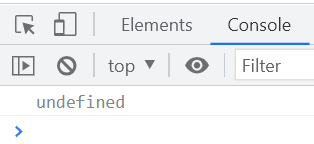
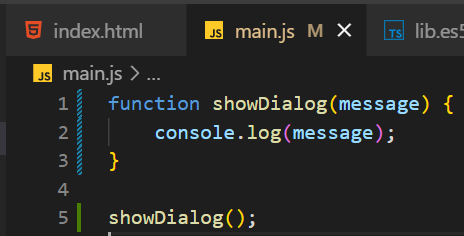
-Vì vậy chúng ta cần sử dụng toán tử **call()** – viết tắt là “ () “ để có thể gọi hàm sau đó hàm mới được thực thi.

* 1. **Hàm có tham số**

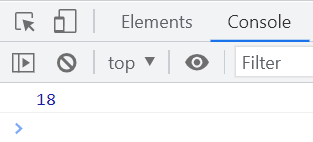
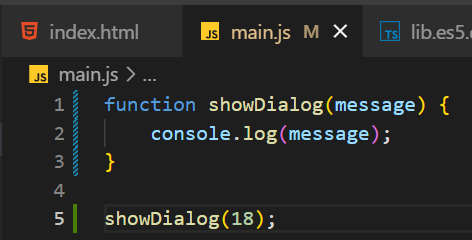
**-**Khác với hàm không có tham số (những hàm chỉ gọi ra và thực thi) thì hàm có tham số phải thêm một bước là nhập **đối số** mới thực thi được.

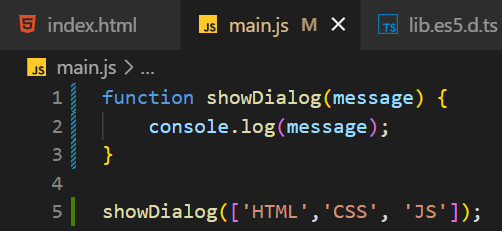
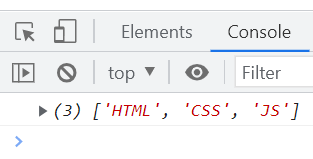


-Trong ví dụ trên “ message “ là **tham số** ‘Hello các bạn’ là **đối số.**



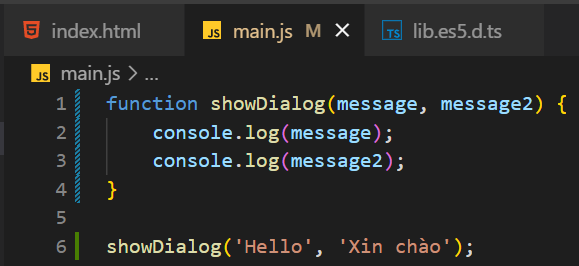
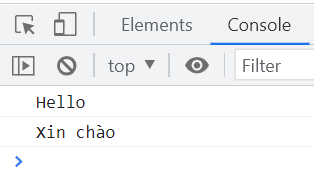
-Trường hợp hàm có tham số nhưng không nhập **đối số** thì sẽ trả về undefined (chưa được xác định).



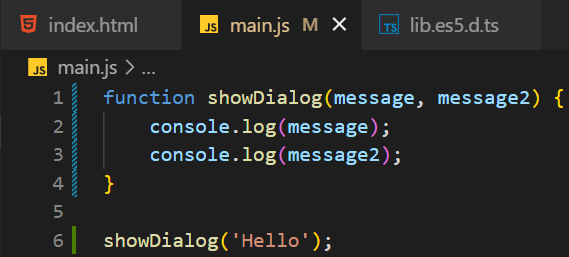
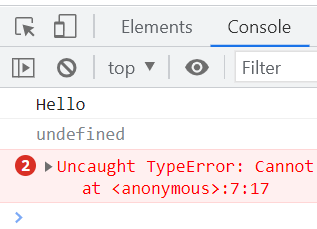
 

-Chúng ta có thể điền bất kỳ kiểu dữ liệu nào trong hàm có tham số.

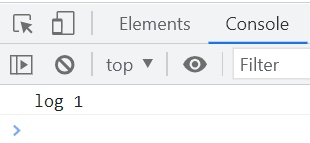
* + 1. **Hàm nhiều tham số:**

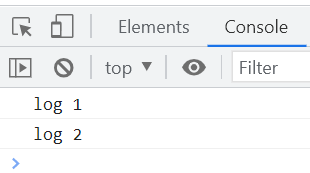
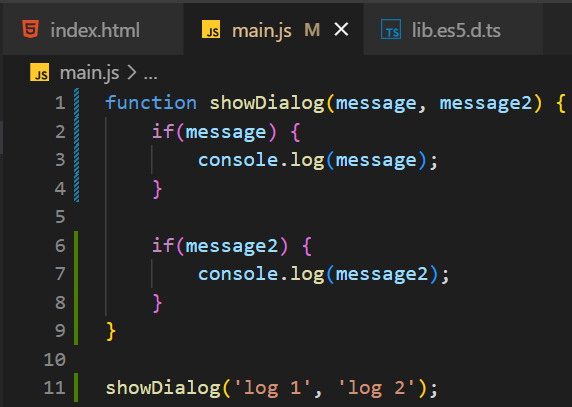
-Ngoài hàm 1 tham số thì chúng ta còn có thể truyền thêm n tham số cho hàm.

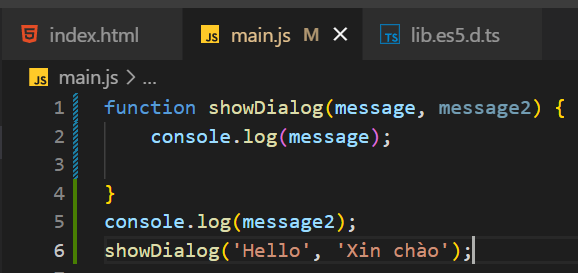
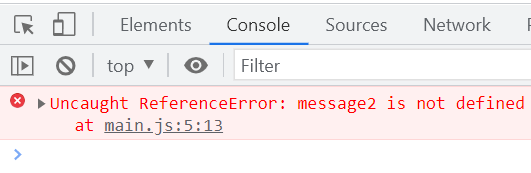
-Trường hợp không nhập hoặc nhập không đủ **đối số** sẽ trả về undefined và báo lỗi ngay.



-Hàm kết hợp với câu điều kiện if-else.

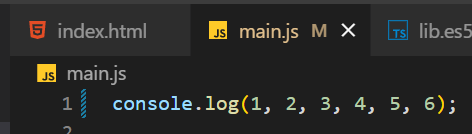
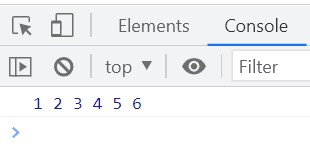


-Trường hợp nhập 1 đối số sẽ trả về màn hình console 1 đối số, còn nhập 2 sẽ trả về 2.

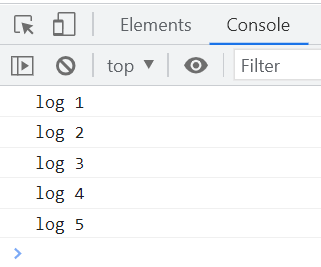
 

-Khi khai báo một biến (trường hợp ở đâu là tham số) trong một hàm thì phạm vi sử dụng biến chỉ ở trong hàm đó mà thôi ! Nếu đem ra ngoài hàm để chảy sẽ báo lỗi là chưa xác định.

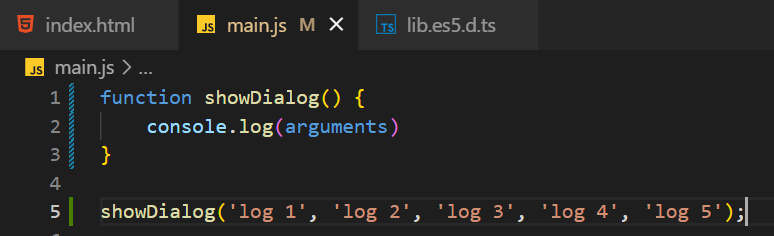
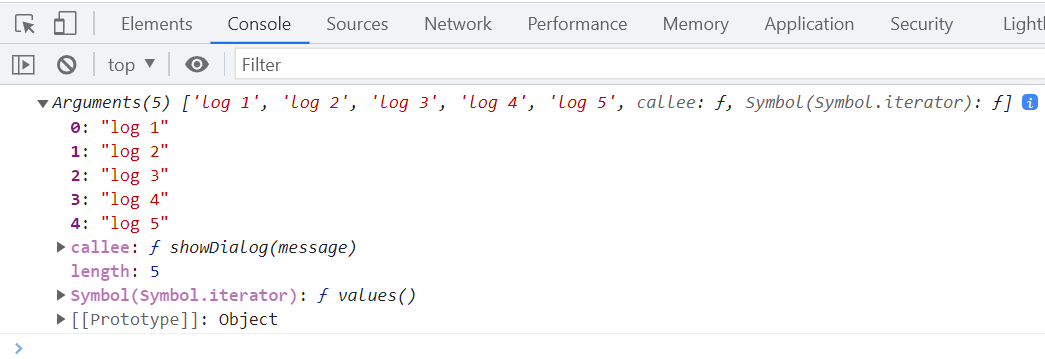
* + 1. **Agruments**

-Phân tích hàm **log();** Nó có thể thông báo bao nhiêu số cũng được tùy theo đối số mà ta nhập.

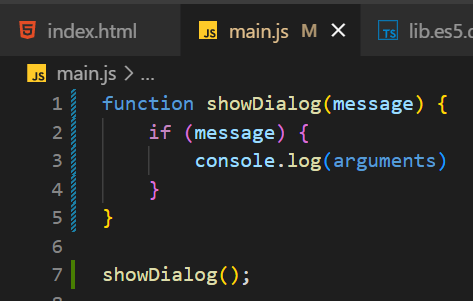
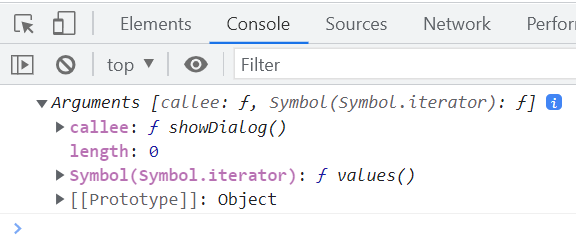


-Để làm giống như hàm **log();** của JS ta phải tạo thêm nhiều tham số và phải nhập nhiều đối số nữa => **Rất mệt mỏi để làm ra một hàm như log() của JS.**

** **

-Thực tế hàm **log();** của JS không hoạt động như vậy. Mà là truyền vào một đối tượng **arguments.**

**-**Trả về một đối tượng **Arrguments** chứa mảng có các đối số mà ta đã nhậpvào.

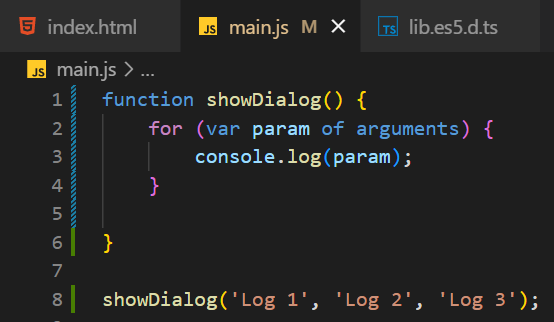
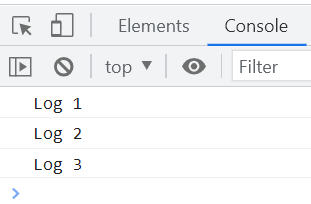
 

-Trường hợp không truyền đối số vào hàm thì sẽ trả về đối tượng **argument** có kiểu\_dữ liệu mảng là rỗng.

-Không cần phải khai báo tham số cho hàm,

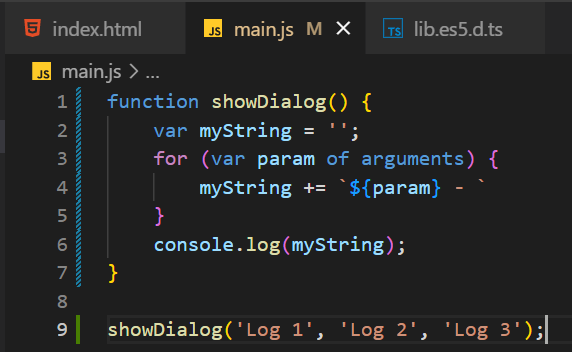
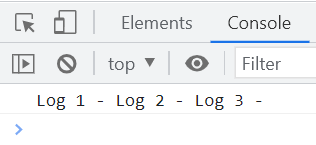
* + 1. **Giới thiệu vòng lặp for**

**-**Vòng lặp **for** dùng để thực hiện một việc gì đó lặp đi lặp lại.

-Giải thích ví dụ trên: vòng lặp **for** sẽ lấy các phần tử của arguments (ở đây còn có thể hiểu là đối số) gán vào biến ‘ param ’ sau đó thông báo bằng hàm **log().**

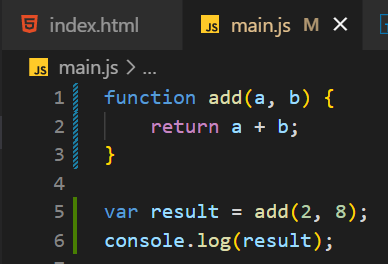
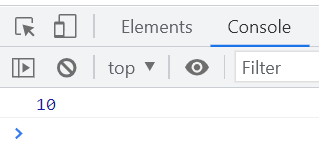
Tiếp theo quay lại vòng lặp **for** và lấy các phần tử còn lại…cho đến khi không còn phần tử nào trong **arguments.**

**** ****

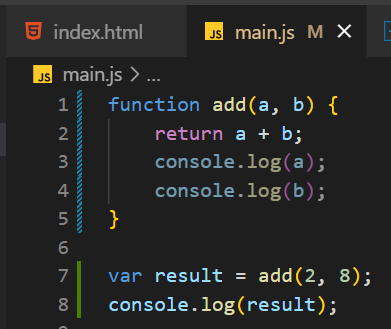
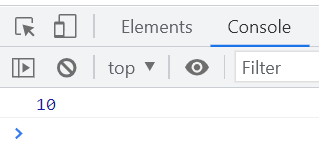
-Ngoài ra chúng ta có thể kết hợp với vòng lặp for với Template ES6 để trong gọn đẹp hơn.

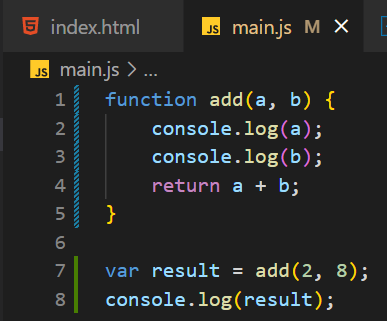
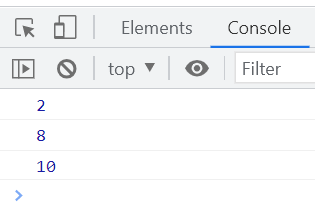
* + 1. **Return trong funtion**

**-**Dùng để trả về các giá trị như: kết quả tính toán, tên biến….

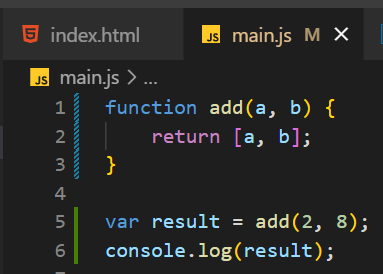
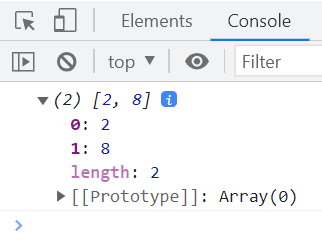
 

-Đúng như tên gọi **return** dùng để trả về mọi giá trị mà chúng ta muốn (chỉ sài ở hàm có tham số).

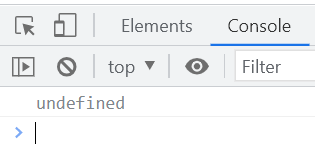
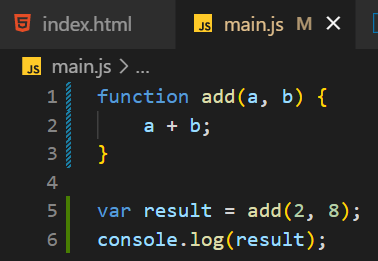
 

-Vì nó là bước cuối cùng của hàm nên cho dù có code thực thi gì sau **return** thì cũng không chạy.

-Ngoài ra **return** có thể trả về bất kì kiểu dữ liệu nào như: int, array, object, string…



-Trong một hàm nếu như không có **return** hoặc là không trả về bất kì giá trị nào thì hàm đó khi thực sẽ trả về không xác định.