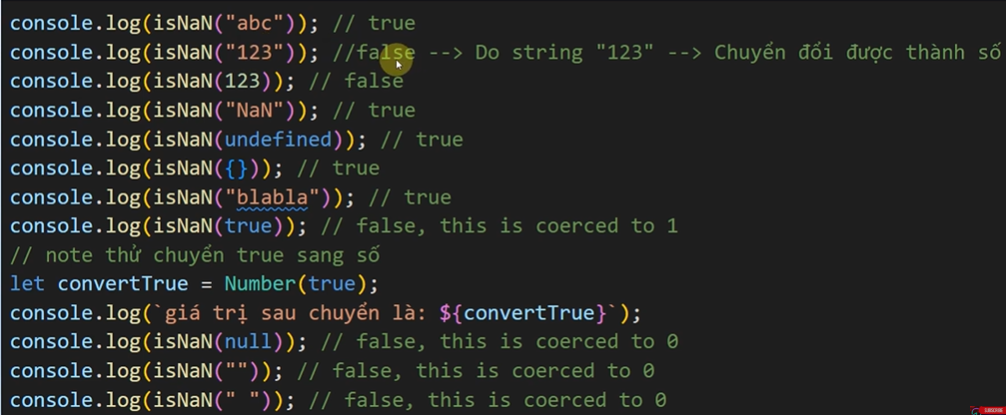
JAVASCRIPT

* Sự khác nhau giữa == và ===:

== : không quan tâm kiểu dữ liệu “1” = 1 🡪 true

=== : có quan tâm kiểu dữ liệu “1” = 1 🡪 false

* Một số lưu ý khi ghép số và chuỗi: chỉ có phép cộng giữa số và chuỗi thì sẽ chuyển thành chuỗi: 6 + “1” 🡪 chuỗi “61”, nhưng các phép - \* / thì sẽ tự động ép chuỗi sang kiểu số và thực hiện phép tính



HÀM

Cấu trúc: function tenham(){

Các câu lệnh

}

let ten\_bien = function (parameter){

}

* Arrow function

Let ten\_bien = (a,b) => a\*b //trả về kết quả a\*b

Javascript hoisting

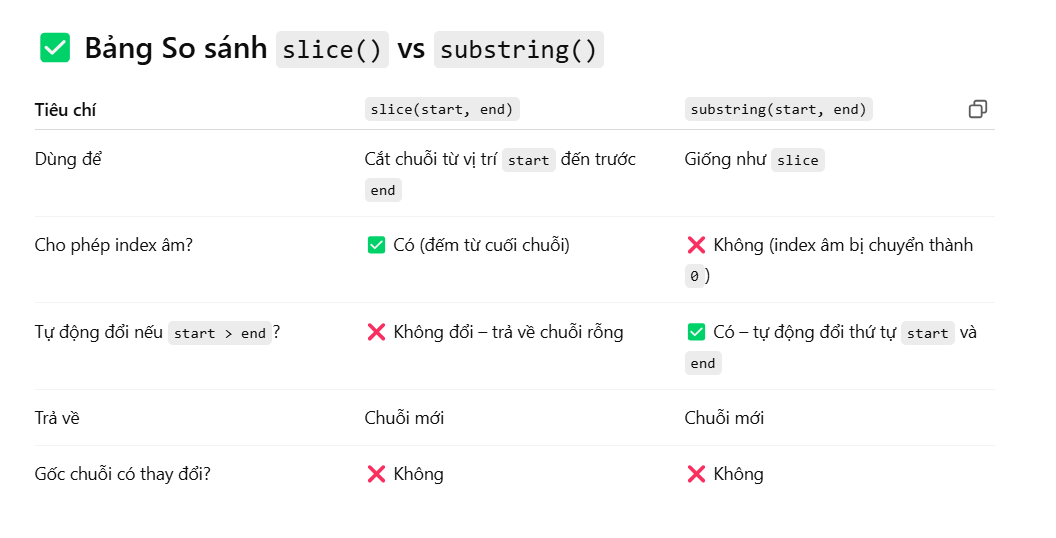
Console.log(c);

Var c = 9;

* In ra undefined🡪đẩy biến được khai báo lên đầu, biến có thể khai báo sau nhưng khi in ra chỉ in lấy biến chứ kh lấy được giá trị
* Biến var có phạm vi là function-scope: tức là nếu khai báo var 1 biến trong 1 khối thì bên ngoài khối và nằm trong hàm vẫn có thể sử dụng được còn let và const có phạm vi là block-scope
* Biến var có thể khai báo lại 1 biến đã khai báo mà kh gây ra lỗi
* Khai báo function: function tenham(){} thì có thể gọi hàm trước khi hàm được định nghĩa

Xử lý chuỗi

* Chuỗi có thể nằm trong “”, ‘’, ``.



Date and time

* Timestamp là 1 đại diện cho 1 thời điểm cụ thể trong thời gian, tính bằng milisecond từ móc 0: 0h0m0s 1/1/1970. Có phương thức getTime
* Các cách đặt ngày tháng theo ý muốn:

+ new Date(2025,1, 20) index của tháng sẽ bắt đầu = 0 🡺 ngày tháng hiển thị là 20/2/2025

+ new Date(2025-2-20T12:30:45) 🡺 T là ký tự phân tách

+ sử dụng các pt setfullyear, setmonth,… 🡪 index của tháng sẽ bdau chạy từ 0, nếu truyền vào 1 thì tháng sẽ là 2

* setTimeout được sử đụng để thực hiện 1 hàm sau 1 khoảng thời gian delay nhất định

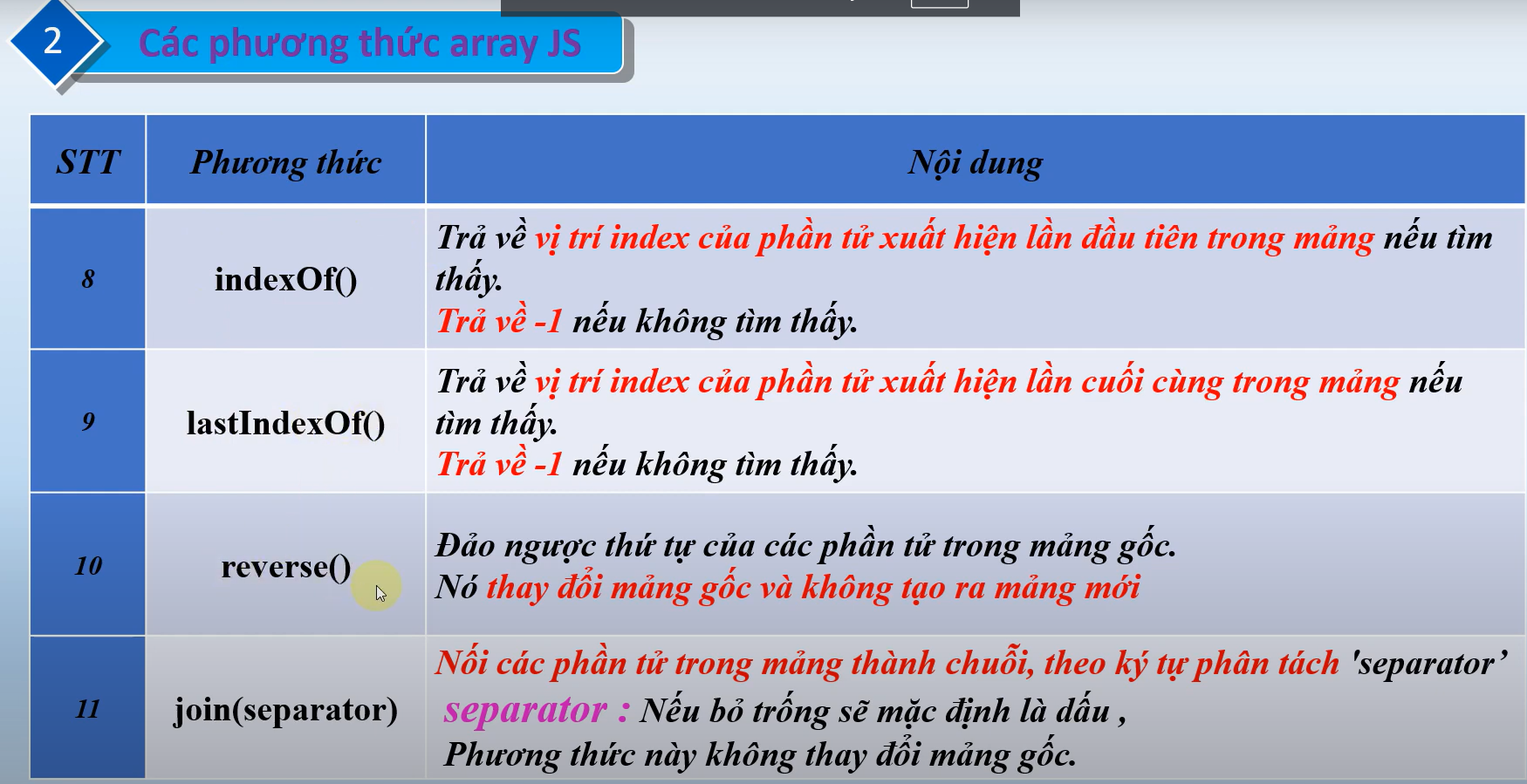
setTimeout(function, milisecond, đối số 1, đối số 2,..)

* setInterval giúp thực hiện function lặp đi lặp lại sau 1 khoảng thời gian cố định. Cú pháp tương tự setTimeout

Mảng

* khai báo: let a = []; , let a = new array;
* có thể có nhiều kiểu dữ liệu của các phần tử trong cùng 1 mảng
* duyệt mảng: for(let i of arr)🡪 nếu duyệt như này thfi ko chỉnh sửa được giá trị của các phần tử





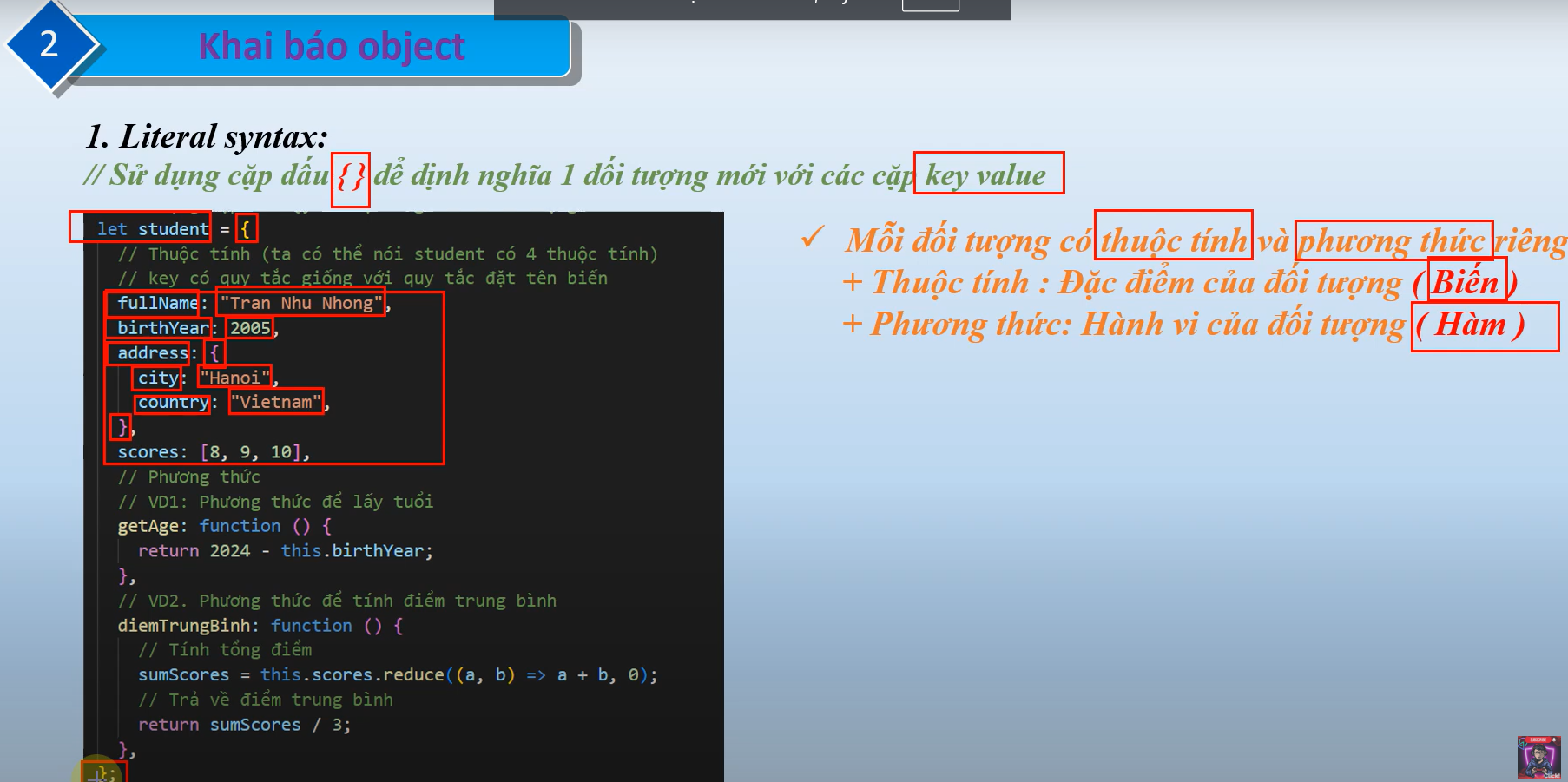
* phương thức splice(start,deleteCount,item1,item2,…) : loại bỏ deletecount phần từ bắt đầu từ start. Item1, item2,… sẽ là các phần tử được thay thế vào từ vị trí start. Nếu start là số âm thì nó sẽ được tính từ cuối mảng
* toán tử spread (…) :

let m1 = [1,2,3]

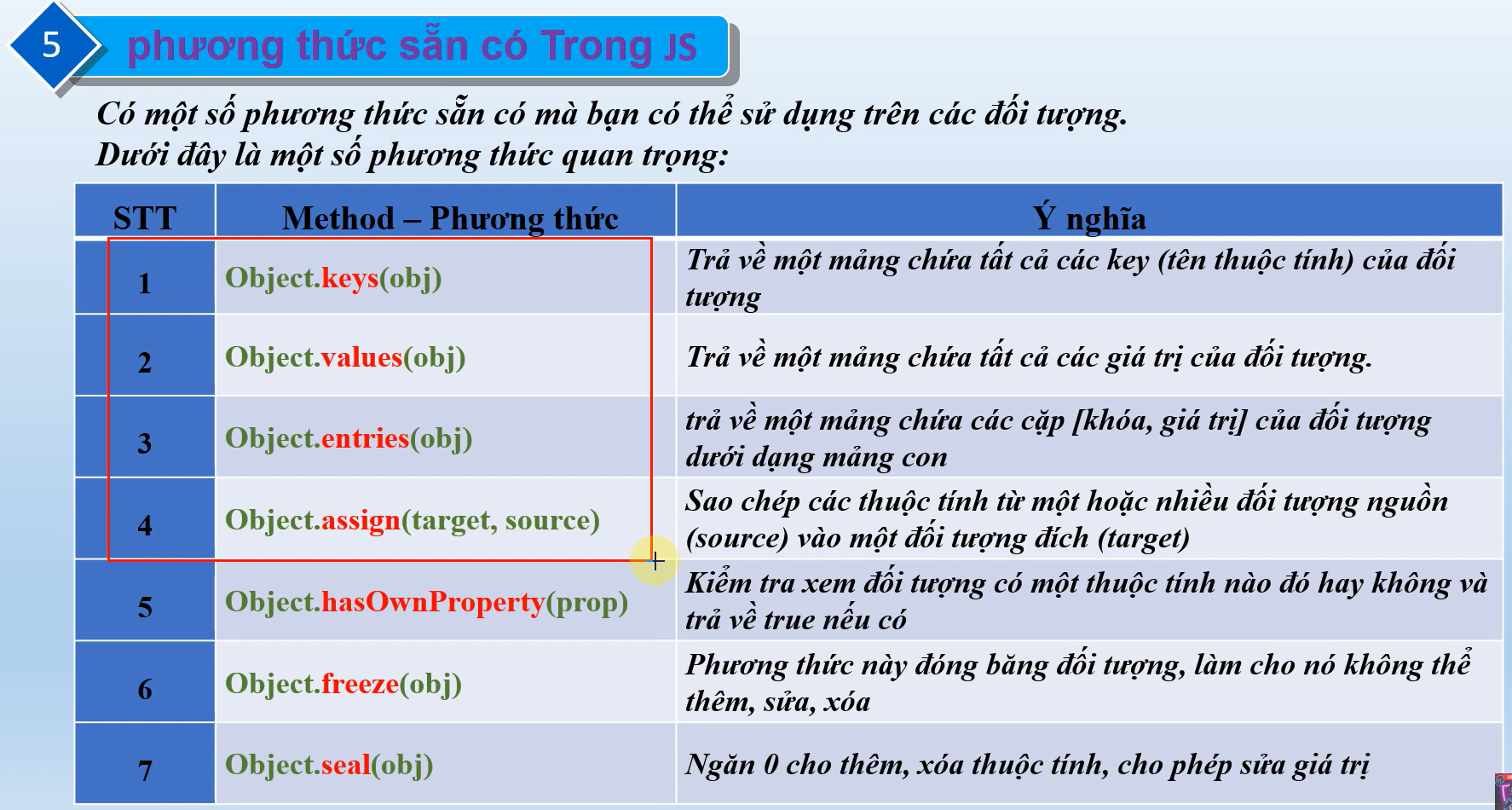
let m2 = […m1]

+ spread cho phép duyệt qua lần lượt các phần tử và qua đó: tạo bản sao (clone) của mảng, truyền đối số vào hàm, nối mảng, tạo mảng mới với thêm phần tử

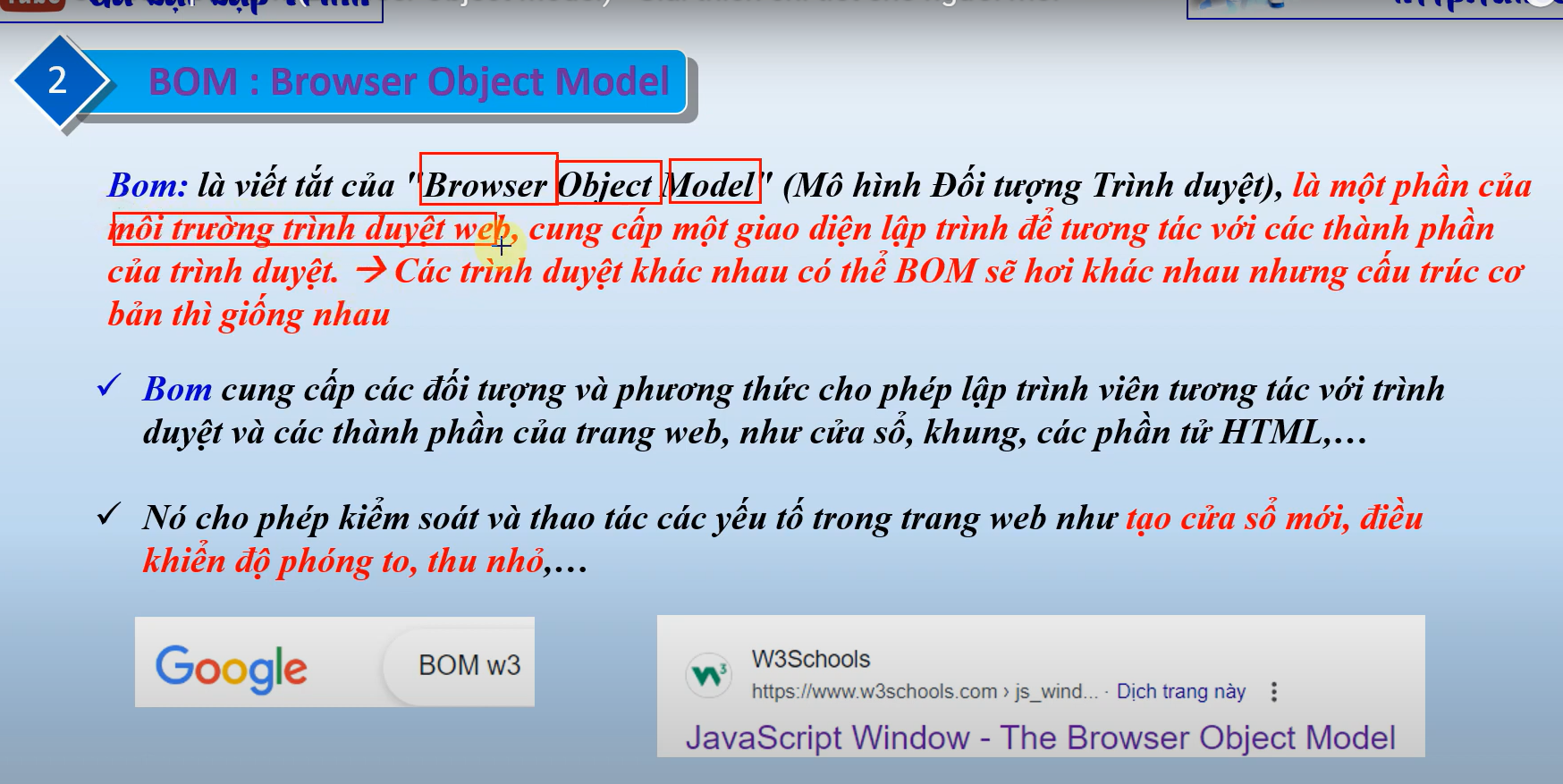
Objects

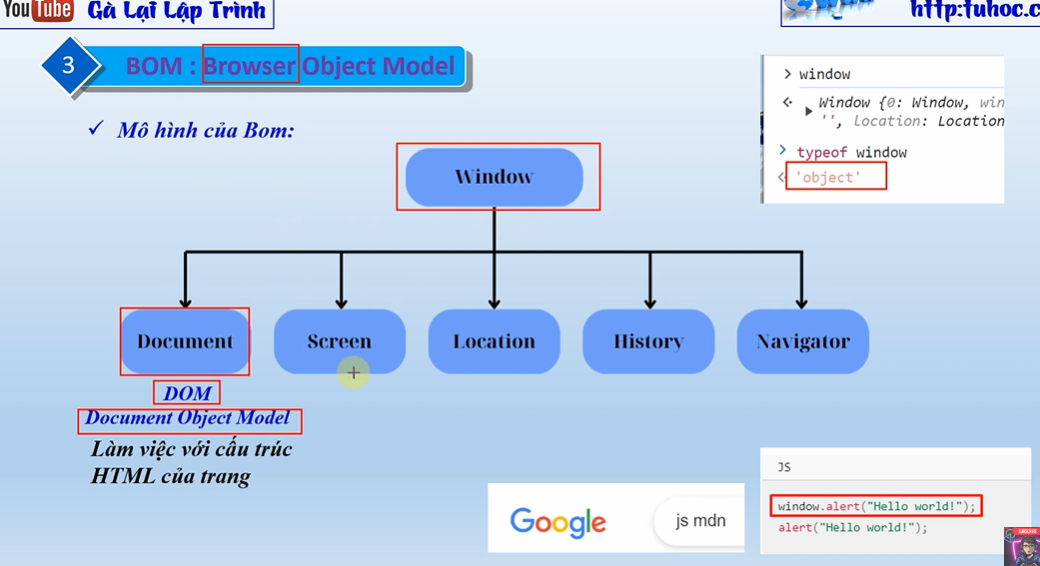


* để thêm thuộc tính mới ta lấy đối tượng . thuộc tính mới. vd: student.age = …
* xóa : delete student.age

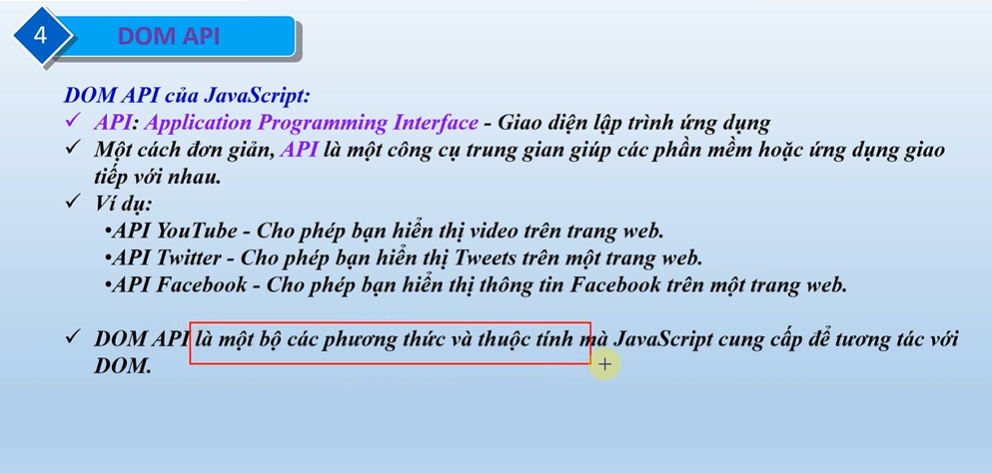


**BOM**

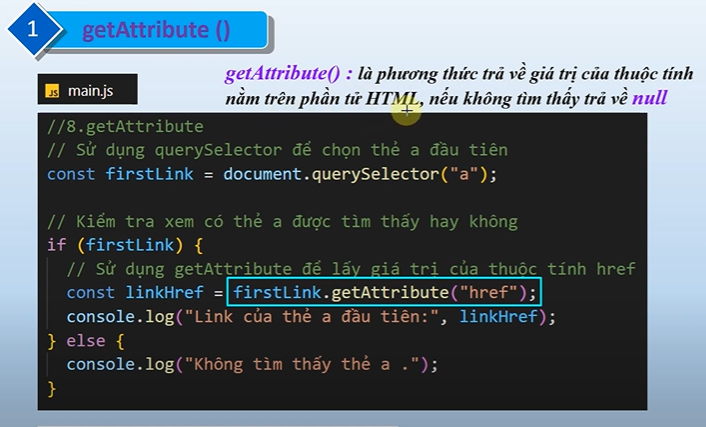




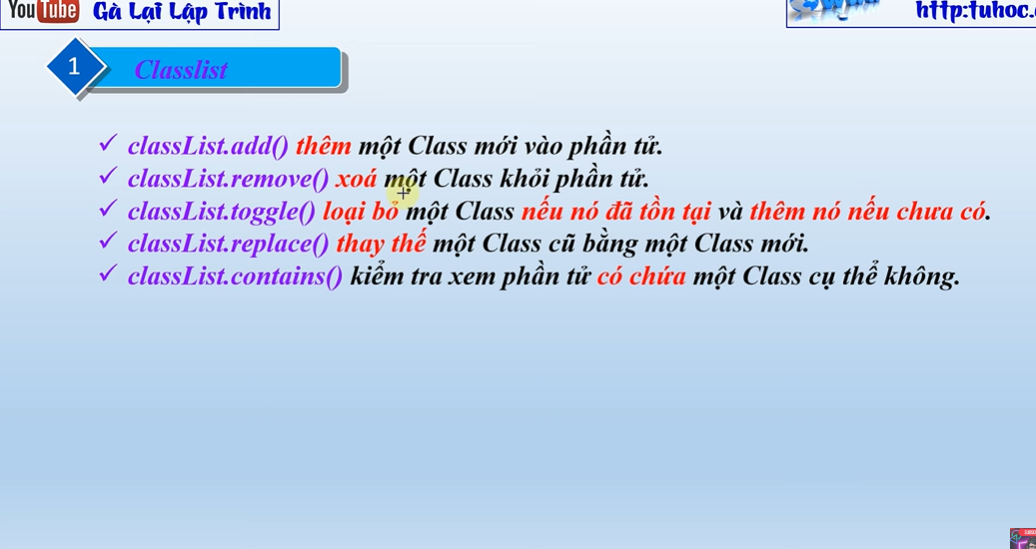
DOM



* getAttribute và setAttribute:



* innerText: chỉ lấy ra các giá trị được hiển thị trên trình duyệt
* TextContent: lấy các nội dung có trong thẻ đó
* innerHTML: lấy toàn bộ các thẻ con và nội dung có trong thẻ đó



Element.classList.add(“..”);



* tạo ra 1 phần tử mới: const mewImg = document.createElement(“img”)
* xóa phần tử con : lấy phần tử cha và phần tử con ra, sau đó ptucha.removeChild(ptucon)
* xóa phần tử: lấy phần tử ra 🡪 ptu.remove();

Event



**Event Handling**

* gọi hàm cho trang web: window.onload = tên hàm;
* addEventListener có thể thêm cùng 1 sự kiện nhiều lần trên cùng 1 phần tử

Event Object and keyboard events

* keydown : khi nhấn 1 phím
* keyup : khi thả phím ra

- preventDefault: hàm ngăn chặn hành vi mặc định của sự kiện