CANTHO UNIVERSITY

TRƯỜNG ĐẠI HỌC CẦN THƠ

KHOA CÔNG NGHỆ THÔNG TIN & TT

BÀI GIẢNG

CT188

NHẬP MÔN LẬP TRÌNH WEB

Nhóm tác giả: Bộ môn THUD

Chương 4
JavaScript (JS)

Nội dung

- Giới thiệu
- JavaScript Engine
- DataTypes
- Statements
- CallBack Function
- DOM

JavaScript là gì?

- JavaScript (JS) là một ngôn ngữ lập trình thông dịch (interpreted programming language)
- Là một trong những công nghệ cốt lõi của Word Wide Web
- Được sử dụng để xây dựng các chức năng cho website có khả năng tương tác tốt với người sử dụng phía client
- Mã lệnh được nhúng vào các trang web và được thực thi bởi trình duyệt web
- Các trình duyệt được tích hợp một cỗ máy để thực thi mã lệnh JS được gọi là JavaScript Engine

Lịch sử ra đời

- Được phát triển bởi Brendan Eich, nhân viên của cty Netscape năm 1995
- Phiên bản đầu tiên có tên LiveScript được phát hành như một phần của trình duyệt web Netscape Navigator
- Năm 1996 Microsoft phát hành Internet Explorer (IE) cạnh tranh với Netscape và ra mắt ngôn ngữ tương tự JS với tên Jscript
- Khác biệt giữa Jscript và JS đã gây khó khăn cho các nhà phát triển website hoạt động tốt trên cả 2 trình duyệt IE và Netscape

Lịch sử ra đời

- Netscape đã gởi đặc tả tiêu chuẩn lên ECMA* yêu cầu các nhà phát triển trình duyệt tuân thủ các tiêu chuẩn chung
- Điều này dẫn đến việc chính thức phát hành bản đặc tả ngôn ngữ ECMAScript (ES) đầu tiên vào tháng 6 năm 1997
- Quá trình tiêu chuẩn hóa tiếp tục trong những năm tiếp theo
  - Phát hành ES2 vào tháng 6 năm 1998
  - ES3 vào tháng 12 năm 1999.
- Công việc trên ES4 bắt đầu vào năm 2000.

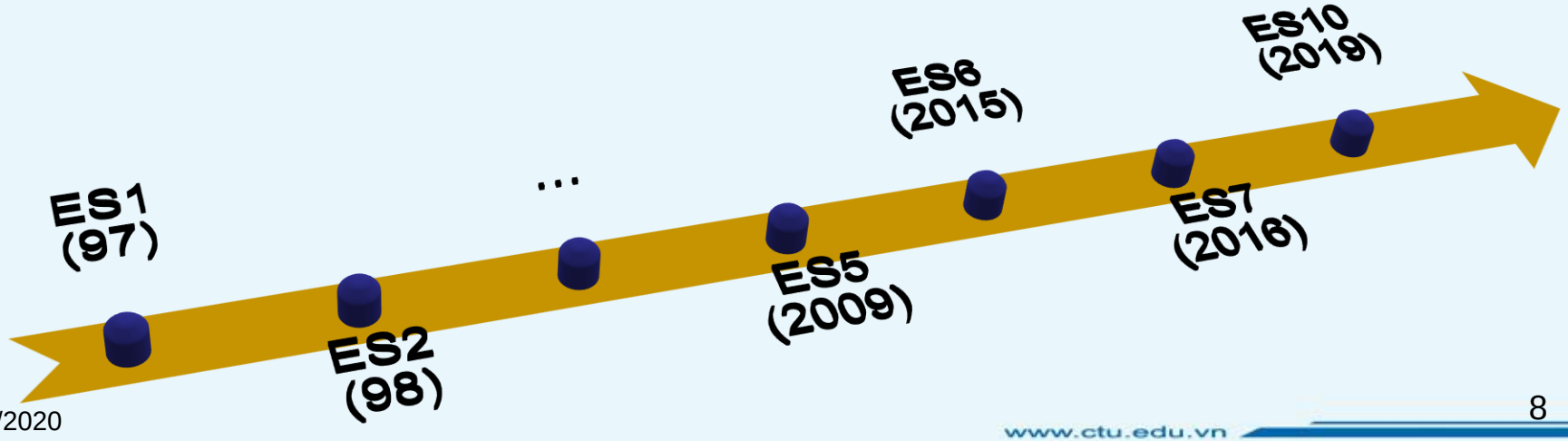
** European Computer Manufacturers Association là một tổ chức đặt ra các tiêu chuẩn cho các hệ thống thông tin và truyền thông*

Lịch sử ra đời

- Trong khi đó, Microsoft ngày càng chiếm được vị thế thống trị trên thị trường trình duyệt sau khi ra mắt Internet Explorer (IE).
- Đến đầu những năm 2000, thị phần của IE đạt 95%.
- Điều này dẫn đến JScript trở thành tiêu chuẩn cho ngôn ngữ kịch bản phía máy khách.
- Ban đầu Microsoft đã tham gia vào quá trình tiêu chuẩn hóa và thực hiện một số đề xuất bằng ngôn ngữ JScript của mình, nhưng cuối cùng họ đã ngừng cộng tác trong công việc ECMA. Do đó, ES4 đã bị hủy bỏ.
- Thời kỳ này ngôn ngữ kịch bản đã bị đình trệ mãi cho đến 2004 khi Mozilla, người kế nhiệm của Netscape phát hành Firefox

Lịch sử ra đời

- Năm 2005, Mozilla gia nhập ECMA International nhằm cố gắng hoàn chỉnh ES4 nhưng không thành công khi không có sự cộng tác từ Microsoft
- Năm 2008 Google ra mắt trình duyệt Chrome với Engine V8 nhanh hơn đối thủ đã nâng JS lên tầm cao mới
- ES5 đã được phát hành cuối năm 2009 sau khi đã đạt được sự thống nhất từ các công ty, tổ chức
- ES6 được phát hành năm 2015 với nhiều tiêu chuẩn mới được thêm vào

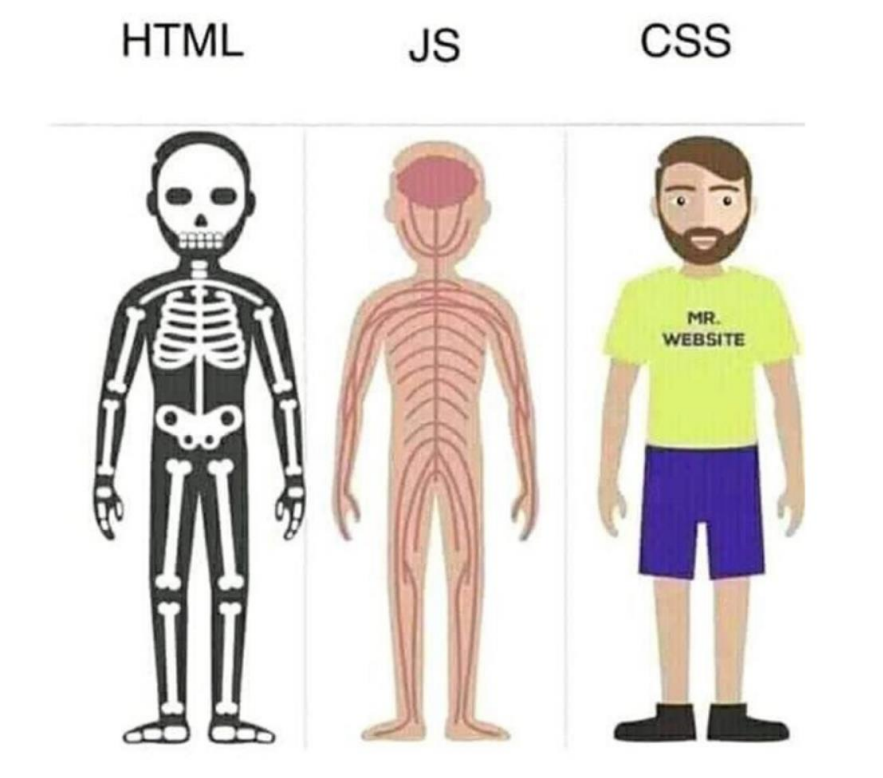


JavaScript Engine là gì?

- Là một chương trình nhằm phân tích cú pháp và thực thi mã lệnh JS được tích hợp trong các trình duyệt
- Phiên bản đầu tiên được phát triển bởi Brendan Eich với tên **SpiderMonkey**
- **SpiderMonkey** được sử dụng bởi **Firefox Browser**
- **V8 engine** (2008) được sử dụng trong **Google Chrome** được đánh giá hiệu năng tốt hơn các engine trước đây
- **Webkit engine** được phát triển bởi **Apple** và được sử dụng trong trình duyệt Safari
- **Carakan** được sử dụng trong **Opera**, sau này chuyển sang V8
- **Chakra** (JScript9): sử dụng bởi **IE, Microsoft Edge**

Tại sao phải sử dụng JavaScript?

- HTML mô tả cấu trúc và nội dung page
- CSS làm cho page hiển thị đẹp hơn
- JS giúp cho page có thể tương tác với người sử dụng



webpage

JS có thể làm được gì?

- Làm cho giao diện web linh động hơn
  - Các hiệu ứng đẹp mắt
  - Thêm/bớt/thay đổi nội dung trên trang web
- Form validation: Kiểm tra dữ liệu nhập trên form
  - Kiểm tra dữ liệu nhập: không được rỗng, nhập số, kiểm tra độ mạnh yếu của mật khẩu, vv…
- Xử lý các sự kiện của người dùng
  - Thực thi các lệnh khi người dùng rê chuột, nhấn nút, gõ phím, …
- Đọc/ghi lịch sử truy cập trang của người dùng (cookie)
- Giao tiếp với server để gởi/nhận dữ liệu (Ajax)

Làm sao để sử dụng JS?

- Mã JS có thể được đặt trực tiếp trong file HTML (inline script) hay nhúng từ tập tin ngoài (external file script)
- Cách 1: Inline script, code đặt trong thẻ **<script>**

```
<script language="javascript" type="text/javascript">
	document.write("Hello World!");
	function sayHi() {
		alert("Hi!");
	}
</script>
```

  - Các thuộc tính **language** và **type** cần phải khai báo để các trình duyệt hiểu, tuy nhiên thuộc tính language có thể bỏ qua vì hầu hết các trình duyệt không quan tâm đến thuộc tính này.

Làm sao để sử dụng JS?

- Cách 2: External file script, code đặt trong tập tin riêng biệt

```
<script src="path/to/scrip.js"></script>
```

Ví dụ 1.1: Nhúng mã js trong file trên cùng host

```
<script src="js/myscript.js"></script>
```

Ví dụ 1.2: Nhúng mã js trong file khác host host

```
<script src="https://ajax.googleapis.com/jquery/3.5.1/jquery.min.js">
        // không được viết thêm code ở đây!
</script>
```

Lưu ý: Không chèn code js vào trong thẻ <script></script> ***khi nhúng code từ tập tin***

Code JS được đặt ở đâu?

- Mã lệnh JS được đặt ở bất kỳ đâu trong trang HTML, sau thẻ **<html>**
- Trong thực tế, mã JS thường được đặt trong thẻ <head> của trang
- Nếu mã JS có tương tác với các phần tử trên trang, nó phải được đặt sau phần tử đó, thông thường sẽ được đặt trước thẻ **</body>** để đảm bảo không gây lỗi load trang
- Thứ tự của các file JS là quan trọng, trong trường hợp file1 chứa thư viện mà file2 sử dụng thì file1 phải được đặt trước file2.

Code JS được đặt ở đâu?

- Ví dụ 2. Cách đặt mã JS trong trang HTML

```
<!DOCTYPE html>
<html>

<head>
<title>Example HTML Page</title>
</head>

<body>
    <!-- content here -->
    <script type=”text/javascript” src=”example1.js”></script>
    <script type=”text/javascript” src=”example2.js”></script>
</body>

</html>
```

Thứ tự thực thi mã lệnh JS

- Mã lệnh JS thực thi theo tứ tự từ trên xuống
- Đối với inline script, mã lênh sẽ được thực thi ngay khi web browser load đến vị trí script trên page
- Đối với external file cũng tương tự inline script, nội dung sẽ được download và thực thi ngay khi browser load đến vị trí nhúng file
- Thuộc tính **`defer`** (chỉ áp dụng cho external script) được sử dụng trong tình huống **bắt buộc** script thực thi **sau khi browser đã load xong nội dung của page** (đến khi trình duyệt nhận được thẻ `</html>`)
- Thuộc tính `defer` chỉ có tác dụng trên các trình duyệt phiên bản mới IE, Firefox, Safari và Chrome, các trình duyệt khác hầu như bỏ qua thuộc tính này

Thứ tự thực thi mã lệnh JS

- Ví dụ 3: bắt buộc mã JS thực thi sau khi load xong page

```
<!DOCTYPE html>
<html>

<head>
  <title>Example HTML Page</title>
  <script type=”text/javascript” defer src=”example1.js”></script>
  <script type=”text/javascript” defer src=”example2.js”></script>
</head>

<body>
        <!-- content here -->
</body>

</html>
```

Thứ tự thực thi mã lệnh JS

- Trong trường hợp external script quá lớn thì:
  - thời gian download script chậm
  - ảnh hưởng đến tốc độ load trang do browser đợi nạp xong file script và thực thi xong mới tiếp tục load nội dung
- Để khắc phục tình trạng này, thuộc tính async được sử dụng để ngăn quá trình trì hoãn load nội dung trang
- Thuộc tính **async** qui định:
  - Browser tiếp tục load nội dung của page, không cần chờ đợi external script download xong
  - Script sẽ thực thi ngay sau khi nó vừa download xong

  → thứ tự thực thi sẽ không theo thứ tự chèn script

Thứ tự thực thi mã lệnh JS

- Ví dụ 4: defer vs async

```
<!DOCTYPE html>
<html>

<head>
  <title>Example HTML Page</title>
  <script type=”text/javascript” defer src=”example1.js”></script>
  <script type=”text/javascript” defer src=”example2.js”></script>

  <script type=”text/javascript” defer src=”example3.js”></script>
  <script type=”text/javascript” defer src=”example4.js”></script>
</head>

<body>
        <!-- content here -->
</body>

</html>
```

Cho biết thứ tự thực thi của các script?

Syntax

- Cú pháp tương tự C, Java
- Case-sensitive: JS phân biệt ký tự HOA thường
- Identifiers:
  - Định danh bắt đầu bằng 1 ký tự, gạch dưới (_) hoặc dấu $, ví dụ **`$n = 20;`**
  - Các ký tự tiếp theo có thể là chữ, số, hay _
  - Định danh có thể sử dụng ký tự Unicode nhưng không khuyến cáo, ví dụ **`chuỗi = "Xin chào"`**
  - Các từ khóa không được sử dụng để đặt tên định danh (biến, hàm,..), ví dụ `true, false, null, for, if, else, catch, continue, return, function, finally, instanceof, typeof, do, break, case, var, void, while, with, this, throw, default,..`

Syntax

- Comments: các chú thích được sử dụng tương tự như C

```
// single line comment

/*
* This is a multi-line
* comment
*/
```

- Statements: các lệnh được viết trên 1 dòng, được kết thúc bằng dấu chấm phẩy (;)
  - Dấu chấm phẩy không bắt buộc phải có, tuy nhiên không khuyến cáo

```
var sum = a + b //valid even without a semicolon - not recommended
var diff = a - b; //valid - preferred
```

Strict Mode

- Chế độ **strict mode** được đặt tả trong EMAScript v5
- Từ khóa **"use strict"** được sử dụng để thông báo cho JS engine thực thi code trong chế độ "strict mode"
- Chế độ strict mode thực thi mã JS một cách an toàn, ví dụ thông báo lỗi khi sử dụng tên biến hoặc object chưa được khai báo, không cho phép tham số của hàm trùng tên,…
- Từ khóa **"use strict"** được đặt ở đầu tập tin JS
- Nếu **"use strict"** được đặt trong block code thì chỉ có tác dụng trong block đó

Strict Mode

- Ví dụ 4: strict mode

```
<script>
    "use strict";
    myFunction();

    function myFunction() {
        y = 3.14; // lỗi ở đây do y chưa được khai báo
    }
</script>
```

```
<script>
    x = 3.14; // Hợp lệ vì strict mode không được bật.
    myFunction();

    function myFunction() {
        "use strict";
        y = 3.14; // Lỗi do đã bật strict mode
    }
</script>
```

Variable

- Biến trong JS được khai báo không cần chỉ định kiểu, được xác định kiểu khi gán trị
- Biến được khai báo thông qua từ khóa **`var`** hoặc **`let`** hoặc **`const`**
- Trong **`strict mode`**, biến bắt buộc phải được khai báo trước khi sử dụng
- Trong **`non-strict mode`**, biến được sử dụng không cần khai báo trước, **có phạm vi global**
- Nếu khai báo nhiều tên bến cùng lúc, các biến cách nhau bởi dấu phẩy
  - Ví dụ `var n = 2, i = 4, k=6;`
- **`var`** có phạm vi trong hàm (function scope) và **`let`** có phạm vi trong khối (block scope)
- **`const`** dùng khai báo hằng, tương tự như `let`

Variable

- Ví dụ 5: Sự khác nhau giữa **var** và **let**

```
function test() {
    var x = 5;
     console.log(x); // output: 5

     let y = 10;
     console.log(y); // output: 10

     console.log(z); // output: undefined
     var z = 2;

     console.log(a); // output: error a is not defined
     let a = 3;
}

test();
console.log(x); // output: error x is not defined
```

Giá trị undefined có nghĩa rằng biến đã được khai báo nhưng chưa gán trị

Scope

- Trong JS có 4 phạm vi là **`global`**, **`local`**, **`lexical`** và **`module`**
- **`global scope`**: khi biến được khai báo bên ngoài function, có phạm vi toàn cục
- **`local scope`**: biến khai báo bên trong function có 2 loại
  - **`function scope`**: phạm vi trong hàm
  - **`block scope`**: phạm vi trong khối
- **`lexical scope`**: chỉ đến khả năng truy cập biến bên trong hàm con (inner function) được khai báo ở phạm vi bên ngoài (outer function)
- module scope: biến chỉ có thể truy cập bên trong module, ngoại trừ việc export biến

Scope

- Ví dụ 6: phạm vi biến

```
<script>
      // global variables
      var counter = 1;
      let found = true;
      const PI = 3.14;

      if (found == true) {
            var fruit1 = "Apple"; // global scope
            let fruit2 = "Banana"; // block scope
            const fruit3 = "Cherry"; // block scope
      }

      console.log(fruit1); // Apple
      console.log(fruit2); // Error! fruit2 is not defined
      console.log(fruit3); // Error! fruit3 is not defined

      function outerFunc() {

            let outerVar = 'I am from outside!';
            function innerFunc() {
               console.log(outerVar); // 'I am from outside!'
               console.log(counter); // 1
               console.log(found); // true
               console.log(PI); // 3.14
            }

            return innerFunc();
       }
    outerFunc();
    console.log(outerVar); // ERROR! outerVar is not defined

</script>
```

Data types

- Trong đặc tả ES có 5 kiểu dữ liệu cơ bản
  - undefined
  - null
  - boolean (true/false, không dùng ~~True/False~~ or ~~TRUE/FALSE~~)
  - number (float, int, decimal)
  - string
- Ngoài ra còn có thêm 2 dữ liệu kiểu khác là object và function
- Toán tử **typeof** trả về kiểu dữ liệu của biến

```
<script>
    var message = "some string";
    var  fish = null;
    alert(typeof message);  //"string"
    alert(typeof(message)); //"string"
    alert(typeof 95);       //"number"
    alert(typeof fish);       //"objec"
</script>
```

- *toán tử typeof có thể sử dụng như hàm typeof(var)*
- *typeof(null) là object!*

undefined

- Khi một biến được khai báo nhưng chưa được gán giá trị sẽ có kiểu là undefined
- Khi truy cập biến trước khi nó được khai báo bởi **var** sẽ trả về undefine
- Truy cập chỉ số mảng hoặc thuộc tính của đối tượng không tồn tại sẽ trả về undefined

Ví dụ 7

```
<script>

    function test1(){
        var myVar;
        alert(myVar); // undefined
    }

    function test2(){
        alert(myVar); // undefined
        var myVar = "hello";
    }

    function test3(){
        alert(myVar); // ERROR: Cannot access
        'myVar'
                                before initialization
        let myVar = "hello";
    }

    var arr = ["Apple", "Banana"];
    alert(arr[2]); // undefined

    var person = {firstname: "Dang", lastname: "Vo"};
    alert(person.age); // undefined

    test3();
</script>
```

Number type

- JS phân biệt 2 kiểu dữ liệu số là **integer** và **float**
- Kiểu **integer** có thể được biểu diễn ở dạng số bát phân (octal) và thập lục phân (hexa)
  - octal: `070; // 56`
  - hexa: `0xA; // 10`
- JS sẽ xem biến có kiểu dữ liệu là `float` khi được gán giá trị số thập phân có phần lẻ
- Khi phần lẻ thập phân bằng 0, JS sẽ ép kiểu sang `integer`
  - `var f = 1.5; // float`
  - `var n = 1.0; // → 1 integer`
- Miền giá trị số được lưu trữ trong thuộc tính:
  - `Number.MIN_VALUE // 5e-324`
  - `Number.MAX_VALUE //  1.7976931348623157e+308`
- Kết quả tính toán vượt miền giá trị sẽ trả về **Infinity**

Number type - casting

- JS hỗ trợ 3 hàm ép kiểu **`Number()`, `parseInt()`** và **`parseFloat()`**
- Number() sử dụng các quy tắc ép kiểu:
  - Ép kiểu từ **`Boolean`**: **`true`** → 1; **`false`** → 0
  - Ép kiểu từ **`null`** trả về 0
  - Ép kiểu từ **`undefined`**, trả về **`NaN`** (Not-A-Number)
  - Ép kiểu từ **`String`**:
    - "1" → 1; "012" → 12;
    - "1.1" → 1.1 (float)
    - "0xF" → 15
    - "" → 0
    - "hello" → NaN

Number type - casting

- **parseInt()** dùng để ép kiểu sang **Interger** với các quy tắc sau:
  - **null** / chuỗi rỗng "" trả về **NaN**
  - nếu ký tự đầu là ký tự → **NaN**
  - nếu ký tự đầu là ký số
    - chuyển sang số
    - lặp lại ở ký tự tiếp theo cho đến ký tự cuối cùng
    - nếu gặp phải ký tự không phải là số nó sẽ trả về kết quả
    - ví dụ: parseInt("123abc"); // 123
  - chuyển từ kiểu **float** sẽ mất phần lẻ thập phân
  - chuyển từ kiểu octal và hexa sẽ trả về giá trị **Integer**
  - tham số thứ 2 của hàm là tham số tùy chọn, chuyển kiểu sang hệ nào (nhị phân, bát phân, thập lục phân)

Number type - casting

```
<script>
    var num1 = parseInt("1234blue"); //1234
    var num2 = parseInt(""); //NaN
    var num3 = parseInt("0xA"); //10 - hexadecimal
    var num4 = parseInt(22.5); //22
    var num5 = parseInt("70"); //70 - decimal
    var num6 = parseInt("0xf"); //15 - hexadecimal
    var num7 = parseInt("10", 2); //2 - parsed as binary
    var num8 = parseInt("10", 8); //8 - parsed as octal
    var num9 = parseInt("10", 10); //10 - parsed as decimal
    var num10 = parseInt("11", 16); //11 –parsed as hexadecimal
    var num11 = parseInt("AF", 16); //175
    var num12 = parseInt("AF"); //NaN
</script>
```

String

- Chuỗi được đặt trong cặp dấu “” hoặc ‘’
- ký tự theo sau \ không được xem là ký tự đặc biệt
- Toán tử **+** dùng để ghép 2 chuỗi
- Ví dụ 9.1: Khai báo chuỗi

```
<script>
	var firstName = "Dang";
	var lastName = 'Vo';
	var fullName = "Dang H Vo'; // syntax error
	document.write("He said, \"hey.\""); // He said, "hey."
	document.write("Java" + "Script"); //JavaScript
</script>
```

String

- Các phương thức (method) và thuộc tính (property) xử lý chuỗi
  - `length`: trả về độ dài của chuỗi
  - `indexOf("string")`: trả về vị trí xuất hiện của chuỗi, trả về -1 nếu không tìm thấy
  - `lastIndexOf("string")`: tương tự `indexOf` trả về vị trí xuất hiện cuối cùng
  - `slice(start, end)`: trích chuỗi con tại `start` đến `end`, nếu `start` và `end` là số âm thì vị trí đếm từ phải sang trái
  - `substring(start, end)`: tương tự `slice` nhưng không chấp nhận tham số có giá trị âm
  - `substr(start, length)`: tương tự `slice` nhưng tham số thứ 2 là độ dài của chuỗi con cần cắt

String

- Các phương thức (method) và thuộc tính (property) xử lý chuỗi
  - `toUperCase()`: đổi sang chữ HOA
  - `toLowerCase()`: đổi sang chữ thường
  - `trim()`: loại bỏ whitespace trước và sau chuỗi
  - `charAt(index)`: trả về ký tự tại index
  - `charCodeAt(index)`: trả về mã Unicode của ký tự
  - `spilt("char")`: tách chuỗi thành mảng ký tự, phân cách bởi char
  - `replace("str1", "str2")`: thay thế str1 bởi str2

String

- Ví dụ 9.2: các thuộc tính và phương thức chuỗi

```
<script>
    var str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    console.log("length: " + str.length); // length: 26
    console.log(str.indexOf("C")); // 2
    console.log("ABACA".lastIndexOf("A")); // 4
    console.log("Hello World".slice(6,11)); // World
    console.log("Hello World".substr(6,5)); // World
    console.log("Apple, Banana, Kiwi".slice(-12,-6)); // Banana
    console.log("ABC".toLocaleLowerCase()); // abc
    console.log("ABC".charCodeAt(2)); // 66 (C)
    console.log('A,B,C'.split(',')); // ["A", "B", "C"]
</script>
```

if statement

- if (condition) statment1 else statement 2

```
if (i > 25) {
      alert("Greater than 25.");
} else if (i < 0) {
      alert("Less than 0.");
} else {
      alert("Between 0 and 25, inclusive.");
}
```

do-while statement

- lặp lại đến khi điều kiện không còn đúng nữa

```
do {

        statement

} while (expression);
```

```
var i = 0;

do {

        i += 2;

} while (i < 10);
```

while statement

- Kiểm tra điều kiện trước khi thực thi statement, lặp lại đến khi điều kiện không còn đúng nữa

```
while(expression) {

      statement

}
```

```
var i = 0;

while (i < 10) {

      i += 2;

}
```

for statement

- for (initialization; expression; post-loop-expression) statement
- for (property in expression) statement
  - for-in dùng để duyệt qua danh sách các object

```
var count = 10;
for (var i = 0; i < count; i++) {
        console.log(i);
}
```

```
for (var propName in window) {
    document.write(propName);
    document.write("<br/>");
}
```

forEach statement

- **`collection.forEach(function(item, index))`**
  - duyệt qua các phần tử trong tập hợp
  - tham số là 1 function (callback function)
  - item: là phần tử đang duyệt đến
  - index: chỉ số phần tử đang duyệt trong collection

```
<script>
    var fruits = ["apple", "orange", "cherry"];

    fruits.forEach(function (e, i) {
        console.log(i + "-" + e);
    });

    // OUTPUT
    // 0-apple
    // 1-orange
    // 2-cherry
</script>
```

switch statement

```
switch (expression) {
        case value: statement
                break;
        case value: statement
                break;
        case value: statement
                break;
        case value: statement
                break;
        default: statement
}
```

```
var num = 25;
switch (num) {
  case num < 0:
    alert("Less than 0.");
    break;
  case num >= 0 && num <= 10:
    alert("Between 0 and 10.");
    break;
  case num > 10 && num <= 20:
    alert("Between 10 and 20.");
    break;
  default:
    alert("More than 20.");
}
```

function

- Cú pháp chung

```
function fn(arg1, arg2, ... argn) {
        // statements
}
```

- Hàm có thể trả về kết quả bởi từ khóa `return`, hoặc không trả về kết quả nào
- Các dòng lệnh sau từ khóa `return` sẽ không được thực thi
- Có thể có nhiều từ khóa `return` trong hàm, chỉ có 1 được thực thi

function

```
function sayHi(name) {
      alert(("Hi");
      return;
      alert("Hello " + name); //never called
}
```

```
function diff(num1, num2) {
   if (num1 < num2) {
      return num2 - num1;
   } else {
      return num1 - num2;
   }
}
```

function - arguments

- các tham số của hàm không xác định kiểu
- → không thể overload hàm như C hay Java

```
<script>
    function addSomeNumber(num) {
        return num + 100;
    }
    function addSomeNumber(num) {
        return num + 200;
    }
    var result = addSomeNumber(100); //300
</script>
```

function - arguments

- có thể truy cập tham số hàm thông qua tên hoặc object `arguments`

```
function logger(msg) {
	console.logger(msg);
	console.logger(arguments[0]);
}

logger("hello");
```

function - arguments

- Tham số mặc định

```
function multiply(a, b) {
   return a * b
}
multiply(5, 2) // 10
multiply(5) // NaN !
```

```
function multiply(a, b) {
        b = (typeof b !== 'undefined') ? b : 1
        return a * b
}
multiply(5, 2) // 10
multiply(5) // 5
```

tham số bị khuyết có giá trị undefined

```
function multiply(a, b = 1) {
        return a * b
}
multiply(5, 2) // 10
multiply(5) // 5
multiply(5, undefined) // 5
```

tham số bị khuyết có giá trị mặc định

function - arguments

- Hàm không tham số
- ví dụ 10: truy cập tham số qua object arguments

```
function sum() {
        let total = 0;
        for(let i = 0; i < arguments.length; i++) {
                total += arguments[i];
        }
        return total;
}
var t;
t = sum(2, 2, 3); // 7
t = sum(2, 2, 3, 2, 2, 4); //15
```

tham số được truy cập thông qua object arguments

function - arguments

- Hàm không giới hạn tham số (ES6)
- Ví dụ 11: truy cập danh sách các tham số

```
function sum(...args){
  let total = 0;
      for(let i in args) {
            total += args[i];
      }
  return total;
}
var t;
t = sum(2, 2, 3); // 7
t = sum(2, 2, 3, 2, 2, 4); //15
```

các tham số được truy cập thông qua tên mảng tham số

Thứ tự thực thi hàm trong JS

- JS thực thi các hàm không theo thứ tự
- Xét ví dụ bên
- Hàm fn2() thực thi trước hàm fn1() mặc dù nó được gọi sau do fn1 bị delay 500ms
- ➔ fn2() cho ra kết quả trước
- Làm thế nào để fn2() phải thực thi sau fn1()?

```
function fn1() {
        // đợi 500ms
        setTimeout(function () {
                console.log(1);
        }, 500);
}

function fn2(){
        console.log(2);
}

fn1();
fn2();
// output
// 2
// 1
```

Callback function

- Callback function là một kiểu định nghĩa hàm với tham số là một hàm khác
- Các hàm được gọi theo thứ tự đã định nghĩa
- Ví dụ 12: Callback function

```
function myFunc(str) {
        console.log("hi " + str);
}

function test(msg, myFunc) {
    console.log(msg);
    myFunc(msg);
}

test("Dang", myFunc);
// output
// Dang
// hi Dang   <-- called by myFunc
```

Callback function

- Anonymous function là một hàm được định nghĩa không có tên
- Ví dụ 13: Anonymous Callback function

```
function test(msg, myFunc) {
        console.log(msg);
        myFunc(msg);
}

test("Dang", function(str){
        console.log("Hello " + str);
});

// output
// Dang
// Hello Dang <-- called by anonymous function
```


JavaScript DOM
JavaScript

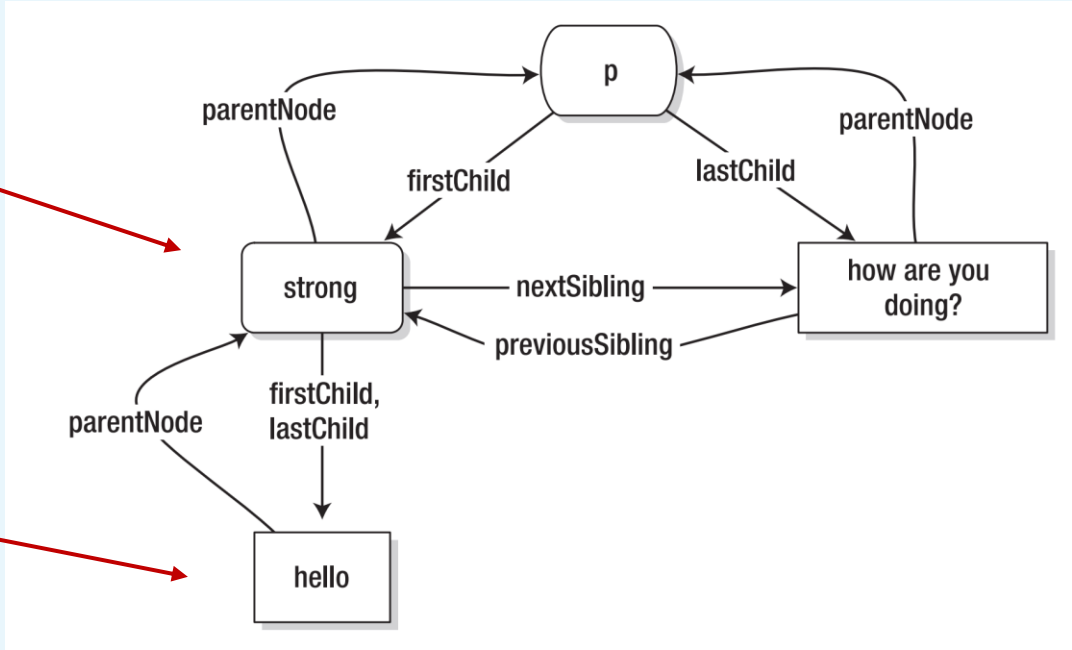
Document Object Model (DOM)

- Trong trang HTML, mỗi element được xem như là một object
- Element chứa các element → tạo nên một cấu trúc phân cấp cha – con
- Cấu trúc phân cấp này được gọi là cây, có nút gốc là window
- Các object có thể là element, text, document, comment, tag được gọi là các nút (node)
- Ví dụ

```
<p><strong>Hello</strong> how are you doing?</p>
```

element node

text node



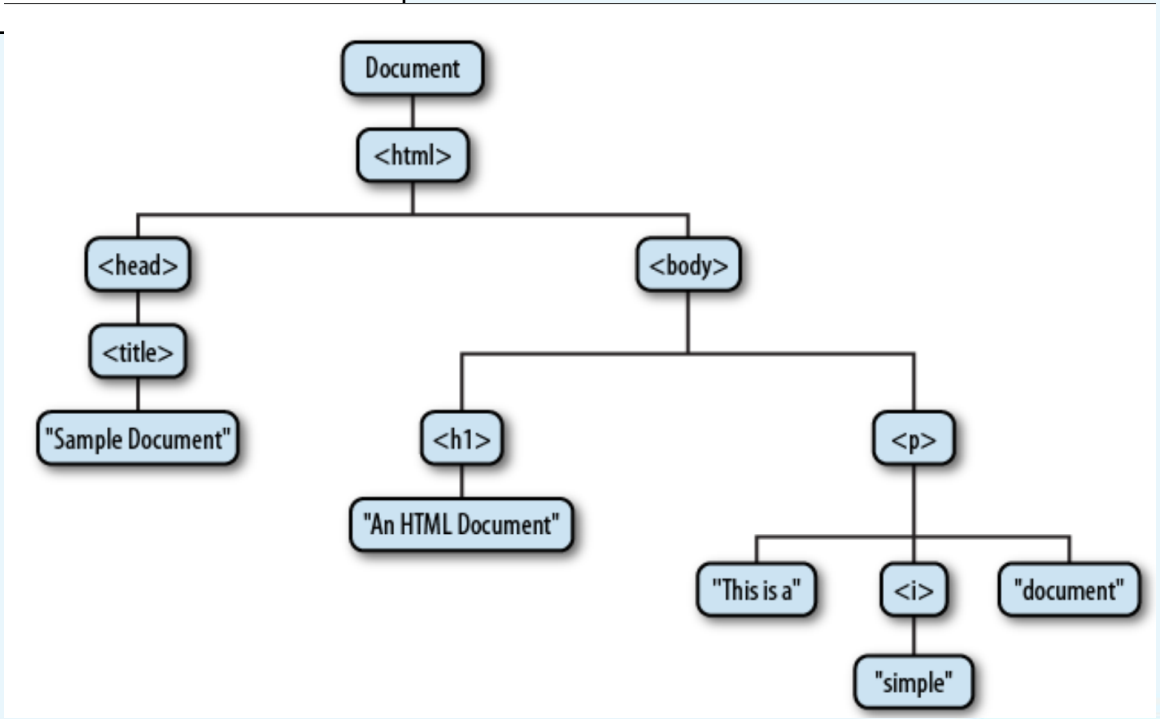
Document Object Model (DOM)

- Ví dụ 14: Cấu trúc DOM của trang HTML

```
<html>

<head>
<title>Sample Document</title>
</head>

<body>
     <h1>An HTML Document</h1>
     <p>This is a <i>simple</i> document.
</html>
```



Document Object Model (DOM)

- JS cung cấp các phương thức cho phép thao tác trên DOM bao gồm
  - Duyệt DOM
  - Thay đổi thuộc tính, giá trị, định dạng của các element trên trang
  - Tạo mới phần tử
  - Chèn thêm phần tử vào node trên trang HTML
  - Loại bỏ phần tử trên trang HTML
  - Tìm kiếm các phần tử với nhiều điều kiện khác nhau

Các thuộc tính truy xuất DOM

Property	Description
document.baseURI	Trả về URL của trang html
document.body	Trả về phần tử <body>
document.cookie	Trả về cookie
document.doctype	Trả về phần tử doctype
document.documentElement	Trả về phần tử <html> của trang
document.domain	Trả về tên miền, ví dụ yoursite.com
document.forms	Trả về tất cả các phần tử form

Các thuộc tính truy xuất DOM

Property	Description
document.head	Trả về phần tử <head>
document.images	Trả về tất cả phần tử <img>
document.inputEncoding	Tả về dạng mã hóa của trang (character set)
document.links	Trả về tất cả phần tử <area> và <a> có thuộc tính href
document.readyState	Trả về trạng thái (loading) của trang
document.scripts	Trả về tất cả phần tử <script>
document.title	Trả về phần tử <title>

Chọn phần tử DOM

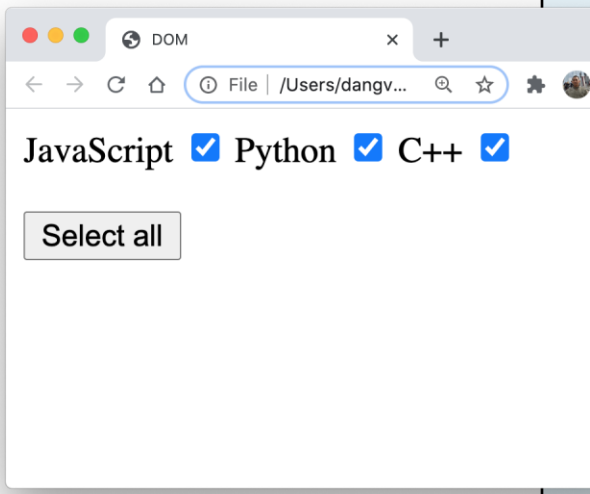
1. `document.getElementById("e_id")`: chọn phần tử có thuộc tính `id="e_id"`
2. `document.getElementByName("str")`: chọn phần tử có thuộc tính `name="str"`
3. `document.getElementByTagName("tag")`: chọn phần tử theo loại, ví dụ `<input>, <form>, <span>,...`
4. `document.getElementByClassName("cls")`: chọn phần tử có thuộc tính `class="cls"`
5. `document.querySelectorAll(CSS_selectors)`: chọn các phần tử khớp với bộ chọn CSS

Lưu ý: (2),(3),(4), (5) trả về kết quả là 1 danh sách các node thỏa mãn điều kiện

Chọn phần tử DOM

- Ví dụ 15: Chọn và thay đổi thuộc tính của phần tử

```
<html>
<head>
<title>DOM</title>
<script>
    function checkall() {
        var x = document.getElementsByName("course");
        for (let i = 0; i < x.length; i++) {
                if (x[i].type == "checkbox") {
                        x[i].checked = true;
                }
        }
    }
</script>
</head>
<body>
    JavaScript <input name="course" type="checkbox" value="js">
    Python <input name="course" type="checkbox" value="py">
    C++ <input name="course" type="checkbox" value="cpp">
    <br />
    <button onclick="checkall()">Select all</button>
</html>
```



Chọn phần tử DOM

- Ví dụ 16: Chọn phần tử với phương thức querySelectorAll()

```
<script>
 // chọn tất cả các phần tử thuộc lớp content
 var x = document.querySelectorAll(".content");

 // chọn tất cả các phần tử p có thuộc tính class="content"
 var x = document.querySelectorAll("p.content");

 // chọn tất cả các phần tử p là con của phần tử div
 var x = document.querySelectorAll("div > p");

// chọn phần tử h2 và phần tử div và span
 var x = document.querySelectorAll("h2, div, span");

 // chọn tất cả các phần tử input có thuộc tính checked
 var x = document.querySelectorAll("input[checked]")

</script>
```

Duyệt phần tử DOM

- `node.childNodes`: trả về danh sách các node con, bao gồm text node và element node
- `node.children`: trả về danh sách **element node** con
- `node.firstChild`: trả về node con đầu tiên của node, nếu node không có node con trả về null
- `node.firstElementChild`: trả về element node con đầu tiền
- `node.lastChild`: trả về node con cuối cùng của node
- `node.lastElementChild`: trả về element node con cuối cùng

Duyệt phần tử DOM

- `node.parentNode`: trả về node cha
- `node.parentElement`: trả về element node cha
- `node.closest(ancestor)`: tìm node tổ tiên ancestor của node hiện tại
- `node.nextSibling`: trả về node anh em ruột phải
- `node.nextElementSibling`: trả về element node ruột phải
- `node.previousSibling`: trả về node anh em ruột trái
- `node.previousElementSibling`: trả về element node anh em ruột trái

Duyệt phần tử DOM

- Ví dụ 17: Lấy tất cả element nodes con

```
<div id="main">
        <p>This is an element node</p>
        This is a text node
        <h3>This is heading element</h3>
</div>
```

```
<script>
        var root = document.getElementById("main");

        var elements = root.children;
        for(let i = 0; i < elements.length; i++) {
                console.log(elements[i].textContent);
        }

// output
// This is an element node
// This is heading element
</script>
```

Duyệt phần tử DOM

- Ví dụ 18: Lấy tất cả nodes con

```
<div id="main">
    <p>This is an element node</p>
    This is a text node
    <h3>This is heading element</h3>
</div>
```

```
<script>
    var root = document.getElementById("main");

    var elements = root.childNodes;

    for(let i = 0; i < elements.length; i++) {
        console.log(elements[i].textContent);
    }

// output
// This is an element node
// This is a text node
// This is heading element
</script>
```

Bao gồm cả text node

Các phương thức và thuộc tính

- **`document.write(data):`** ghi giá trị trực tiếp trên trang HTML
- **`node.attribute:`** nhận / gán giá trị cho thuộc tính node
- **`node.getAttribute(attrib):`** nhận giá trị của thuộc tính **`attrib`**
- **`node.setAttribute(attrib, val):`** gán val cho thuộc tính **`attrib`**
- **`node.style.property:`** nhận / thay đổi giá trị định dạng của **`node`**

Các phương thức và thuộc tính

- **`node.innerText`**: Trả về nội dung của **`node`**, không bao gồm mã HTML, khoảng trắng và nội dung có thuộc tính CSS ẩn (ví dụ display:none, hidden)
- **`node.textContent:`** tương tự **`innerText`** nhưng bao gồm cả khoảng trắng, nội dung có thuộc tính CSS ẩn
- **`node.innerHTML:`** Trả về toàn bộ text, mã html và khoảng trắng

Các phương thức và thuộc tính

- Ví dụ 19: sự khác nha giữa các thuộc tính

```
<p id="demo">  This element has extra spacing and contains <span>a span element</span>.</p>
```

```
var x = document.getElementById("demo");

alert(x.innerText);
// "This element has extra spacing and contains a span element."

alert(x.textContent);
// " This element has extra spacing and contains a span element."

alert(x.innerHTML);
// " This element has extra spacing and contains <span>a span element</span>.”

// assign new value
x.innerHTML = "New <b>content</b>";
x.style.color = "red";
```

Thêm, sửa, xóa node

- **`document.createDocumentFragment()`:** tạo một tài liệu rỗng, mục đích dùng làm container để thêm các node con vào trước khi chèn vào trang HTML
- Ví dụ 20: chèn phần tử <li> và danh sách

```
<ul id="list"> </ul>
<script>
    var element = document.getElementById('list');
    var fragment = document.createDocumentFragment();
    var browsers = ['Firefox', 'Chrome', 'Opera',
            'Safari', 'Internet Explorer'];

    browsers.forEach(function (browser) {
        var li = document.createElement('li');
        li.textContent = browser;
        fragment.appendChild(li);
    });

    element.appendChild(fragment);
</script>
```

- Firefox
- Chrome
- Opera
- Safari
- Internet Explorer

chèn phần tử vào tài liệu

Thêm, sửa, xóa node

- **node.createElement(name):** tạo mới **element** tên **name** (ví dụ span, div, h1, table, td,…)
- **node.createTextNode(string):** tạo mới **text node**
- **document.createAttribute(name):** tạo mới **attribute** cho element
- **node.setAttributeNode(attrOjb):** gán **attribute** cho **node** với **attrObj** là đối tượng tạo ra bởi **createAttribute()**
- **node.cloneNode(bool)**
  - **true**: nhân bản **node** bao gồm **attribute** và tất cả các **node** con bên trong
  - **false**: chỉ nhân bản **node** và **attribute**

Thêm, sửa, xóa node

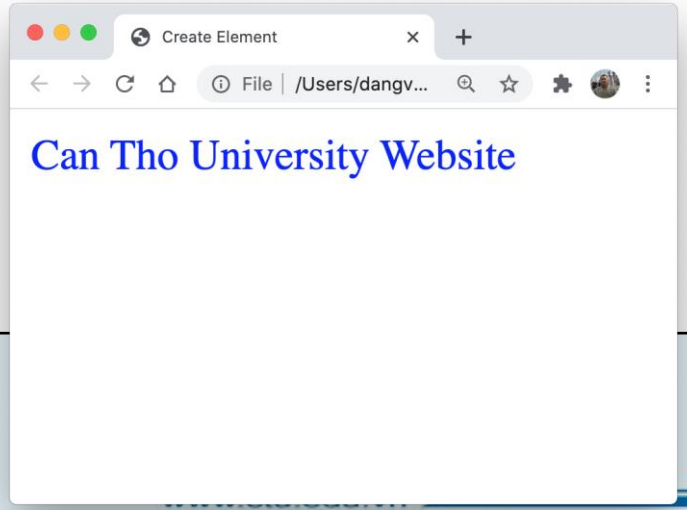
- Ví dụ 21: Tạo mới liên kết và chèn vào tài liệu

```
<body>
<div id="main"></div>

<script>
    var main = document.getElementById("main");
    var a = document.createElement("a");
    var href = document.createAttribute("href");

    href.value = "http://www.ctu.edu.vn";
    a.setAttributeNode(href);
    a.innerText = "Can Tho University Website";
     a.style.textDecoration = "none";
     a.style.color = "blue";

    main.appendChild(a);
</script>
</body>
```



Thêm, sửa, xóa node

- **node.appendChild(n)**: **n** trở thành con của **node**, n sẽ nằm ở vị trí cuối cùng trong danh sách con **node**
- **node.insertBefore(new,child):** chèn **new** như là nút con của **node**, ngay trước nút **child**
- **node.remove():** xóa bỏ node khỏi document
- **node.removeAttribute("name"):** xóa **attribute** ra khỏi **node**, không trả về kết quả
- **node.removeAttributeNode(attrOjb):** xóa bỏ thuộc tính là **object**, trả về object vừa xóa
- **node.removeChild(n):** xóa bỏ nút n là con của **node**, trả về nút đã xóa nếu xóa thành công, trả về null nếu nút con không tồn tại
- **node.replaceChild(new,old):** thay thế nút con **old** của **node** bằng **new**, trả về nút **old** đã thay thế

Thêm, sửa, xóa node

- Ví dụ 22: Chèn, xóa phần tử

```
<ul id="list">
  <li>Python</li>
  <li>JavaScript</li>
  <li>C++</li>
</ul>

<script>
    var l = document.getElementById("list");
    var item = document.createElement("li");
    var cpp = l.lastElementChild;

   // xóa phần tử C++
    cpp.remove();

    item.innerText = "Java";
    l.insertBefore(item, l.firstElementChild);

</script>
```

- Java
- Python
- JavaScript