**NỘI DUNG ÔN TẬP JAVASCRIPT**

1. Nguồn gốc ra đời, lịch sử phát triển các phiên bản cũng như đặc trưng của ngôn ngữ lập trình JavaScript
2. **Khái niệm về biến và các kiểu dữ liệu trong JavaScript**

Các biến được dùng để **lưu trữ các giá trị để tái sử dụng** các giá trị nhiều lần

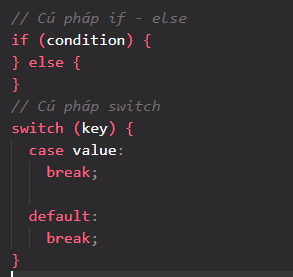
Javascript có 7 kiểu dữ liệu cơ bản:

* number : các số bất kỳ loại nào: số nguyên hoặc dấu phẩy động.
* string  : chuỗi. Chuỗi có thể có một hoặc nhiều ký tự, không có loại ký tự đơn riêng biệt.
* boolean  : true/ false.
* null  : các giá trị không xác định – một loại độc lập có một giá trị duy nhất null.
* undefined : các giá trị chưa được gán – một kiểu độc lập có một giá trị duy nhất undefined.
* Symbol
* bigInt

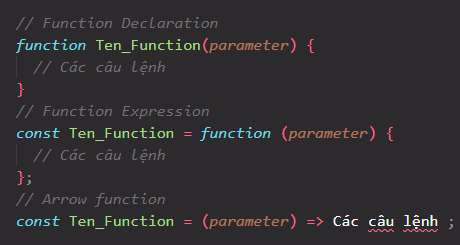
1. **Khai báo biến, phân biệt giữa let const và var**

|  |  |  |
| --- | --- | --- |
| **Var:**  Khai báo trên global scope, **có thể** khai báo lại cùng tên biến, **có thể** gán giá trị trước rồi mới khai báo | **Let**:  Khai báo trên blocked scope, không thể khai báo lại  **Có thể** gán lại giá trị | **Const:**  Khai báo trên blocked scope, không thể khai báo lại  Không thể gán lại giá trị |

1. Các toán tử cơ bản, độ ưu tiên giữa các toán tử
2. **câu lệnh if-else, switch**

****

1. **khai báo hàm, cách gọi hàm và một số cơ chế trong hàm**

****

1. **khái niệm về Object trong JavaScript**

Object quy định các phần tử theo cặp key-value

Key giống như tên biến

Value có thể thuộc bất kỳ kiểu dữ liệu nào, có thể gán value là một biểu thức. Mỗi cặp key-value trong object được gọi là thuộc tính (attribute) của object

2 cách lấy giá trị của thuộc tính: dấu chấm và ngoặc vuông

Sử dụng dấu chấm: jonas.lastName

Sử dụng dấu ngoặc vuông: jonas['lastName']

1. **Vòng lặp for, while cũng với các từ khóa break và continue**
2. **khái niệm về mảng cũng với các toán tử trong mảng**

- Array (mảng) là một cấu trúc dữ liệu quan trọng trong JS, cho phép lưu trữ các giá trị khác nhau vào trong một biến duy nhất.

**push** – thêm một phần tử vào cuối array

Ngoài việc thêm một phần tử vào array hàm push cũng trả về độ dài mới của array. Ví dụ:

const friends = ['Michael', 'Steven', 'Peter'];

// Add elements

const newLength = friends.push('Jay'); // 4

**unshift** – thêm một phần tử vào đầu array (3:55)

Ví dụ: const newLength = friends.unshift('John');

**pop** – gỡ một phần tử khỏi đuôi array (5:00)

Ví dụ: const popped = friends.pop();

**shift**– gỡ một phần tử khỏi đầu array (7:05)

Ví dụ: friends.shift(); // First

**indexof**- chỉ ra vị trí của 1 phần tử trong array (8:00)

Hàm index trả về vị trí của phần tử trong mảng và trả về -1 nếu không tìm thấy. Ví dụ:

const index = friends.indexOf('Steven') // 1

const index = friends.indexOf('Bob') // -1

* + **includes** – chỉ ra phần tử có xuất hiện trong array không (9:15)

Hàm includes trả về giá trị true/ false. Phương thức này sử dụng phép so sánh ===.  Ví dụ:

1. Cách viết code tốt, các kỹ năng giải quyết vấn đề để trở thành một lập trình viên chuyên nghiệp
2. Thiết lập được các công cụ để hỗ trợ khi viết code JavaScript nhanh hơn và dễ đọc/bảo trì code hơn
3. Sử dụng được các công cụ Debugging
4. **DOM và DOM Manipulation**

**DOM** là tên gọi viết tắt của (**D**ocument **O**bject **M**odel – tạm dịch Mô hình Các Đối tượng Tài liệu)

**DOM** được dùng để truy xuất và thao tác trên các tài liệu có cấu trúc dạng HTML hay XML bằng các ngôn ngữ lập trình thông dụng như Javascript, PHP…

Có 3 loại node chính:

* Node gốc (Document node): Chính là tài liệu HTML
* Node phần tử (Element node): Biểu diễn cho 1 phần tử của HTML
* Node văn bản (Text node): mỗi đoạn ký tự trong tài liệu HTML, bên trong 1 tag HTML đều là một node văn bản. Đó có thể là tên trang web trong thẻ <title>, tên heading 1 trong thẻ <h1>, hay một đoạn văn bản <p>.
* Ngoài ra còn có node thuộc tính (attribute node) và node chú thích (comment node).

1. **truy vấn các phần tử trên DOM Tree**
2. **cách xử lý một số Event trong DOM**
3. Nguyên tắc Dry để tối ưu hóa mã nguồn
4. **Sử dụng JavaScript để thay đổi các thuộc tính trên DOM Tree**
5. quá trình thực thi một chương trình Javascript
6. JS engine. JS runtime, Web API, Callback queue
7. kiến thức cơ bản về Execution Context, Call Stack trong JavaScript
8. Scope và 3 dạng Scope trong Javascript

- Scoping kiểm soát cách JS Engine sắp xếp và truy cập các biến trong chương trình.

- Lexical scoping là cách tổ chức và truy cập các biến hoàn toàn bị vị trí của các hàm và các khối trong code kiểm soát. Ví dụ một hàm được viết trong một hàm khác có quyền truy cập các biến của hàm cha. Scope bị ảnh hưởng bởi vị trí mà chúng ta đặc các hàm và khối code của mình.

- Scope là không gian hoặc môi trường khai báo một biến nhất định.

- Scope của một biến là toàn bộ khu vực code có thể truy cập vào biến đó.

Trong JS có 3 loại scope: global scope, function scope và block scope

Global Scope là các top-level code, các biến được khai báo bên ngoài bất kỳ hàm hoặc khối nào. Các biến này có thể được truy cập ở mọi nơi trong chương trình.

Function Scope: mỗi một hàm đều tạo một scope và các biến khai báo bên trong phạm vi của hàm đó chỉ có thể được truy cập bên trong hàm. Đây còn được gọi là local scope.

Block Scope: Từ **ES6**, các khối block cũng tạo ra một scope, code block tất cả mọi thứ nằm trong dấu ngoặc nhọn { } (ví dụ: khối lệnh if, vòng lặp for). Giống function scope, tất cả các biến được khai báo trong block chỉ có thể được truy cập bên trong block đó, không thế truy cập từ bên ngoài.

Khác biệt giữa function scope và block scope là block scope chỉ áp dụng cho các biến được khai báo bằng let, const và var là function scope hoặc global scope.

Ngoài ra, tất cả các scope đều có quyền truy cập vào các biến từ các scope bên ngoài của chúng.

1. Áp dụng được từ khóa this
2. Phân biệt được Arrow Function và Regular Function

-Khác cú pháp khai báo

- arrow functions sẽ không có this của nó

|  |  |
| --- | --- |
| ARROW FUNCTION | REGULAR FUNCTION |
| * không có this của nó * Không thể sử dụng argument. Vẫn có thể sử dụng rest parameters (…) để thay thế cho *arguments* * Không thể gọi bằng cách sử dụng từ khóa new * không phù hợp làm method cho object * không có thuộc tính prototype * không được hoisted: nghĩa là bạn phải định nghĩa arrow function trước khi sử dụng nó | * Có this của nó * Dùng được argument * Dùng được từ khóa new |

1. Phân biệt được Primitives với Objects

|  |  |
| --- | --- |
| PRIMITIVES | OBJECTS(Non-Primitive Data Types) |
| Các biến chúng ta khởi tạo. Lưu trữ trong callstackBiến trong giá trị nguyên thủy được lưu trữ vẫn có thể được gán lại một giá trị mới.Có 7 kiểu dữ liệu primitive (xem ở trên) | * Lưu trữ trong HEAP * Thay đổi giá trị của obj là thay đổi giá trị trong HEAP |

1. Phân biệt được Pass by Value với và Pass by Reference
2. First-Class Function và Higher-Order Function

First-Class Function là function chúng ta dùng bình thường

**Higher order functions** là hàm hoạt động trên các hàm khác , bằng cách lấy chúng làm tham số hoặc trả về chúng. Nói một cách đơn giản, một Higher-Order function là hàm nhận một hàm dưới dạng đối số hoặc trả về hàm dưới dạng đầu ra.

1. Khái niệm về Callback và Closure trong JavaScript
2. **phương thức gọi hàm như call, apply và bind**
3. **Array Method trong JavaScript (slice, splice, reverse, concat, join, ...)**

**Phương thức slice**

- Phương thức slice() này trả về một array mới có các phần từ được lấy từ một array gốc.

- Mảng gốc sẽ không bị sửa đổi.

**Phương thức splice**

- Phương thức splice() thay đổi nội dung của một mảng bằng cách loại bỏ hoặc thay thế các phần tử hiện có và / hoặc thêm các phần tử mới vào vị trí.

- Khác với slice, mảng gốc bị sửa đổi.

**Phương thức reverse**

- Phương thức reverse() là phương thức dùng để đảo ngược một mảng.

- Phương thức reverse làm thay đổi mảng gốc.

**Phương thức concat**

- Phương thức concat () được sử dụng để hợp nhất hai hoặc nhiều mảng.

- Phương thức này không thay đổi các mảng hiện có mà thay vào đó trả về một mảng mới.

**Phương thức join**

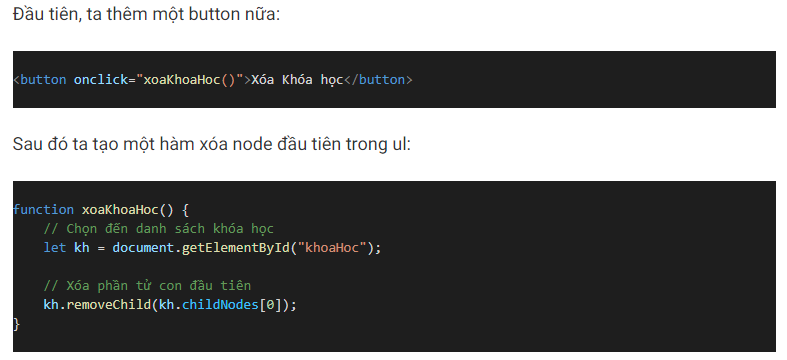
- Phương thức join() tạo và trả về một chuỗi mới bằng cách nối tất cả các phần tử trong một mảng (hoặc một đối tượng giống mảng), được phân tách bằng dấu phẩy hoặc một chuỗi phân tách được chỉ định.

- Nếu mảng chỉ có một mục, thì mục đó sẽ được trả về mà không cần sử dụng dấu phân tách.

1. **Array Method để biến đổi mảng (map, reduce, filter, ...)**
2. các thao tác như truy vấn, tạo mới và xóa DOM Element

* createElement(tagName): Tạo một node mới
* createTextNode(text): Tạo một text Node mới
* appendChild(node): Thêm một node con vào node hiện tại
* removeChild(): Xóa node





1. Sử dụng được JavaScript để thay đổi các Style, Attribute và Class trong DOM
2. Kiến thức về Event Propagation và Event Delegation
3. **Sử dụng được cơ chế DOM Traversing**
4. **Khái niệm về OOP và 4 đặc trung cơ bản của OOP trong JS**

**OOP (viết tắt của Object Oriented Programming)** - một phương pháp lập trình dựa trên khái niệm về Class và Object.

Mục tiêu của OOP là tối ưu việc quản lý source code, giúp tăng khả năng tái sử dụng và giúp tóm gọn các thủ tục đã biết trước tính chất thông qua việc sử dụng các đối tượng.

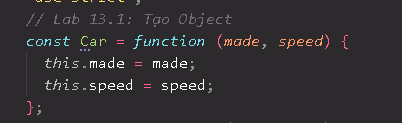
- Trừu tượng: Sự trừu tượng cơ bản là **bỏ đi hoặc ẩn chi tiết không quan trọng**. No cho phép ta có cái nhìn tổng quan về những gì mà chúng ta triển khai.

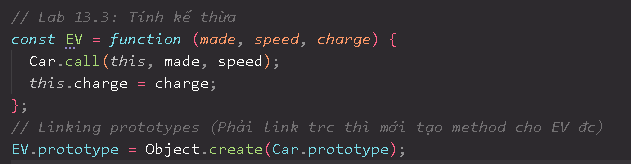
- Đóng gói: Đóng gói là **giữ một số thuộc tính và các phương thức private bên trong class để chúng không truy cập được từ bên ngoài class**. Chúng ta luôn có mục tiêu là đóng gói hầu hết các trạng thái và phương thức và chỉ công khai các phương thức thiết yếu.

- Kế thừa: Trong OOP chúng ta có hai class có liên quan chặt chẽ với nhau, **một class kế thừa class kia** và chúng sẽ là class cha và class con. Class con kế thừa các thuộc tính và phương thức của class cha. Tính thừa kế làm cho tất cả các thuộc tính và phương thức của một class nhất định có sẵn cho class con, nó tạo thành một hệ thống phân cấp.

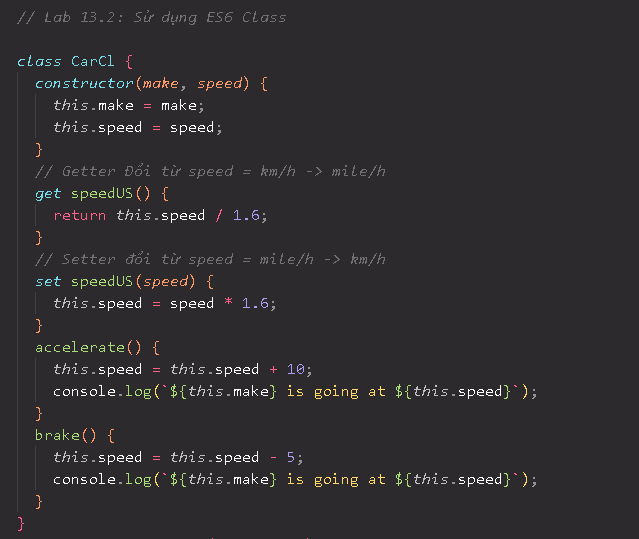
- Đa hình : (nhiều hình dạng) Một class con **có thể ghi đè một phương thức** mà nó được thừa kế từ một class cha.

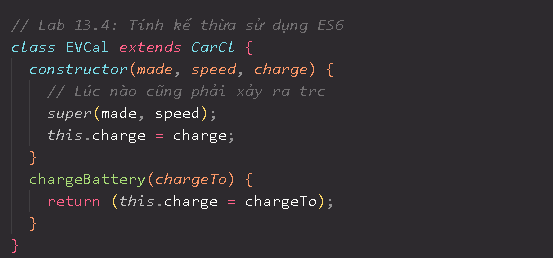
1. **Constructor và toán tử new trong JavaScript**
2. **định nghĩa về Prototype trong JavaScript**
3. **Khai báo được ES6 Class**
4. **Các cách kế thừa Class**

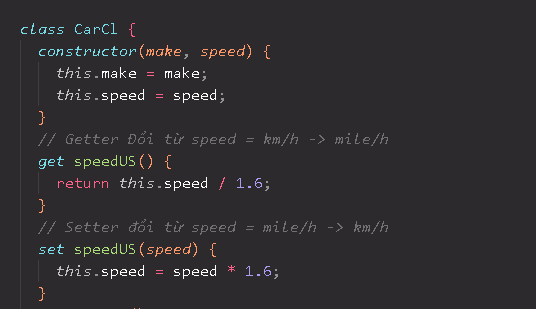
****

****

**ES6**

****

****

****

****

1. **hàm super()**
2. khái niệm về đồng bộ (schyronous) và bất đồng bộ (aschyronous)

* **Schyronous** : Đồng bộ có nghĩa là code được thực thi từng dòng một theo thứ tự thực hiện chính xác mà chúng ta đã xác định trong code của mình.

Mỗi dòng code luôn đợi dòng trước thực thi xong rồi mới thực thi

Điều này dẫn đến vấn đề đối với một dòng code mất nhiều thời gian để chạy, việc thực thi dòng code đó sẽ chặn những dòng code khác thực thi

* **Aschyronous**: Bất đồng bộ là code sẽ được thực thi sau khi một task chạy ở background kết thúc

Code ở luồng chính không bị chặn

Việc thực thi không phải đợi một task bất đồng bộ phải thực hiện xong

1. khái niệm về API và AJAX Call

- API (Application Programming Interface) là một phần của phần mềm có thể được sử dụng bởi một phần mềm khác, cho phép các ứng dụng giao tiếp và trao đổi thông tin.

- Ajax (Asynchronous Javascript And XML)  cho phép chúng ta giao tiếp với các máy chủ web từ xa theo cách không đồng bộ

- JSON là một đối tượng dạng JS nhưng được chuyển thành một chuỗi string

1. XMLHttpRequest
2. **Promises và Fetch API**

Với promises, chúng ta sẽ triển khai Ajax theo cách mới, không dùng XMLHttpRequest mà dùng Fetch API. Fetch Api sẽ trả về một **Promise**

- Promise là một object được sử dụng như một placeholder cho kết quả trong tương lai của hành động bất đồng bộ. SETTLED - khi task kết thúc, promise đã nhận được xử lý và nhận 1 trong 2 giá trị: FULFILLED và REJECTED

1. **Async/Await**

Async/await được cấu tạo từ 2 phần.

Phần đầu tiên là **function async**. Hàm này sẽ được tự động thực thi bất đồng bộ. Giá trị nó trả về là một Promise. Vì trả về Promise nên bạn sẽ phải sử dụng các handler của Promise để xử lý giá trị này.

Phần thứ hai của async/await là **operator await**. Operator này sẽ được dùng cùng với một Promise. Nó sẽ khiến cho function async tạm dừng cho đến khi Promise đó chạy xong. Ngay sau đó nó sẽ lấy gía trị của Promise mà cho function async tiếp tục chạy.

****

* Cú pháp thêm HTML vào

*let* html = "";

    data.articles.forEach((*article*) => {

      html += `

      <Đoạn code HTML>

      `;

      newsContainer.innerHTML = html;

🡺Chú ý sử dụng +=, sẽ thêm từng đoạn HTML kế tiếp dựa theo dữ liệu trong mảng