**[Bài đọc] Tổng quan về ES6**

1. **Khái niệm**

* ECMAScript 6 (ES6), còn được gọi là ECMAScript 2015, là một phiên bản của ngôn ngữ lập trình JavaScript.
* ES6 đã đem lại nhiều cải tiến quan trọng cho ngôn ngữ JavaScript, làm tăng tính hiệu quả, linh hoạt và độ đáng đọc của mã nguồn. Các tính năng mới của ES6 không chỉ giúp làm cho mã nguồn JavaScript dễ hiểu hơn mà còn tăng cường khả năng lập trình và duy trì mã nguồn.

1. **Các tính năng mới của ES6**
2. **Từ khóa let**

* Có thể gán lại giá trị cho biến
* Không thể khai báo lại
* Phạm vi nằm trong một block code
* Cú pháp khai báo:

**let userName = "Nguyễn Văn A";**

**userName = "Nguyễn Văn B"; // Có thể gán lại**

**let userName = "Nguyễn Văn C"; // Gây ra lỗi**

1. **Từ khóa const**

* Được dùng để khai báo những hằng số có giá trị không bị thay đổi như: số PI,...
* Không thể khai báo và gán lại giá trị cho biến
* Phạm vi nằm trong một block code
* Cú pháp khai báo:

**const userName = "Nguyễn Văn A";**

**// Không thể gán lại**

**userName = "Nguyễn Văn B";**

**// không thể khai báo lại**

**const userName = "Nguyễn Văn C";**

1. **Arrow function**

* Arrow function (hàm mũi tên) là một cú pháp định nghĩa hàm ngắn gọn được giới thiệu trong ECMAScript 6 (ES6).
* Arrow function có cú pháp ngắn gọn hơn so với cú pháp hàm truyền thống (function)
* Không tạo ra bản sao mới của this, arguments, super, và new.target, giúp giảm nguy cơ nhầm lẫn.
* Cú pháp:

**const functionName = (parameters) => {**

**// Statements**

**};**

* Trong đó:
  + functionName: Tên của hàm (có thể bỏ qua nếu bạn muốn một hàm vô danh).
  + parameters: Các tham số của hàm.
  + => : Toán tử mũi tên, được sử dụng để định nghĩa arrow function.
  + Statements: Mã nguồn bên trong hàm, nơi bạn định nghĩa logic của hàm.
  + return result: Trả về một giá trị từ hàm (nếu cần).
* Ví dụ: Xây dựng hàm tính tổng hai số sử dụng arrow function

**const sum = (a, b) => {**

**return a + b;**

**};**

**console.log(sum(10, 20)); // 30**

**console.log(sum(20, 30)); // 50**

1. **Template Literals**

* Template literals (còn được gọi là template strings) cho phép bạn tạo các chuỗi có dạng mẫu (template) dễ đọc hơn và mạnh mẽ hơn bằng cách nhúng biểu thức JavaScript vào bên trong chuỗi.
* Template literals sử dụng ký tự backticks (``) để định nghĩa chuỗi mẫu.
* Trong template strings, chúng ta có thể chèn code JavaScript để hiển thị và thực hiện các logic bên trong.
* Để chèn được code JavaScript, chúng ta sử dụng cú pháp sau:

**let userName = "Nguyễn Văn Nam";**

**console.log(`Tên của tôi là: ${userName}`);**

1. **Default Parameters**

* Cho phép đặt giá trị mặc định cho các tham số của hàm
* Cú pháp:

**function greet(name = "Guest") {**

**console.log(`Hello, ${name}!`);**

**}**

**// Trong ví dụ trên, tham số tham được gán cho giá trị mặc định là "Guest"**

1. **Destructuring**

* Destructuring cho phép bạn trích xuất giá trị từ cấu trúc dữ liệu như mảng và đối tượng một cách tiện lợi.
* Destructuring giúp làm mã nguồn ngắn gọn hơn và tạo ra mã đọc dễ dàng hơn bằng cách gán các giá trị từ cấu trúc dữ liệu vào các biến một cách nhanh chóng.
* Cú pháp trích xuất từ mảng:

**const colors = ['red', 'green', 'blue'];**

**const [firstColor, secondColor, thirdColor] = colors;**

**console.log(firstColor); // 'red'**

**console.log(secondColor); // 'green'**

**console.log(thirdColor); // 'blue'**

* Cú pháp trích xuất từ đối tượng:

**const person = { userName: "John", age: 30 };**

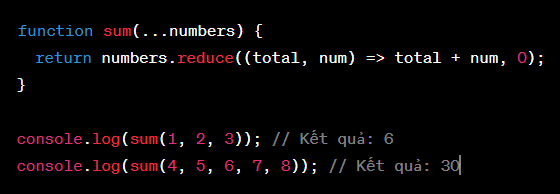
**const { name, age} = person;**

**console.log(userName); // 'John'**

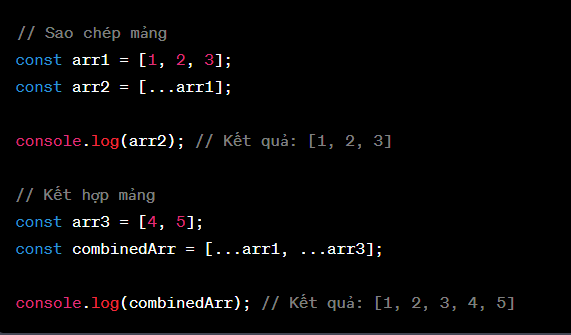
**console.log(age); // 30**

1. **Rest parameter và Spread Operators**

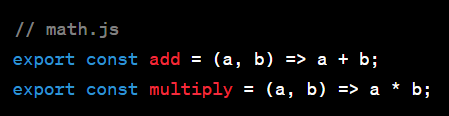
* **Rest Parameter** cho phép gom một số lượng không xác định các tham số của hàm vào một mảng. Điều này thường được sử dụng khi bạn muốn viết một hàm có thể nhận một số lượng biến tham số tùy ý.



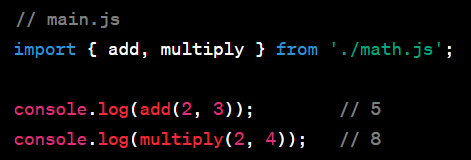
* Trong ví dụ trên, ...numbers là Rest Parameter, cho phép hàm sum nhận một số lượng biến tham số không giới hạn và gom chúng thành một mảng numbers.
* Spread Operator cho phép trải một mảng hoặc đối tượng thành các phần tử hoặc thuộc tính riêng lẻ. Điều này thường được sử dụng để sao chép mảng hoặc đối tượng, hoặc kết hợp nhiều mảng hoặc đối tượng lại với nhau.
* Ví dụ: Hợp nhất hai mảng số nguyên thành một mảng duy nhất



1. **Modules**
   * Trong ECMAScript 6 (ES6), cụm từ "module" được sử dụng để đề cập đến các thành phần mã JavaScript chia sẻ dữ liệu, hàm và lớp giữa các tệp.
   * ES6 đã đưa ra một cách tiêu chuẩn để tạo và sử dụng các module, giúp quản lý mã nguồn dễ dàng hơn và tạo sự sắp xếp, sáng sủa cho dự án JavaScript lớn.
   * Trong Modules có 2 khái niệm cần quan tâm là: import và export
     + **export** được sử dụng để xuất các biến, hàm hoặc lớp từ một module
     + Ví dụ: Trong file math.js, chúng ta xuất các biến ra bên ngoài:



* + - **import** được sử dụng để nhập các biến, hàm hoặc lớp từ một module khác vào module hiện tại.
    - Ví dụ: Trong file main.js, import các biến ở trong file math.js



* Module trong ES6 được xác định bởi <script type="module">: Để sử dụng các module ES6, bạn cần chỉ định <script type="module"> trong tệp HTML của mình:

