javascript

1. Cài đặt CSCode Extention
2. Javascript là gì ?
3. Cách tạo và sử dụng Javascript đơn giản
4. Variables: const, let, naming convention
5. Hoisting
6. Temporal Dead Zone
7. Data Types: String, Number, Boolean, Object, Array

1. Cài đặt CSCode Extention

- Bracket Pair Colorizer 2: hightlight cặp ngoặc tương ứng.
- Javascript (ES6) code snippets: gõ tắt trong javascript.
- ESLint: tìm bắt lỗi javascript.
- Prettier: Format code.
- Code Runner: thực thi code trực tiếp.
- Live Server: Live preview static web.
- Material Icon Theme: Bộ icon đẹp cho folders.

2. javascript là gì:

Ngôn ngữ lập trình là gì:

Ngôn ngữ lập trình bao gồm những thành phần nào

- synctax: bộ quy tắc hình thànhnên ngôn ngữ.
- variable: cách khai báo biến.
- function: cách khai báo và sử dụng function.
- control structures: cấu trúc điều khiển.
- data structure: cấu trúc dữ liệu (cấu trúc ngôn ngữ đó hổ trợ kiểu dử liệu nào, mổi kiểu data có gì đặc biệt, và làm được gì với kiểu dữ liệu đó).
- tools: những phần mềm hỗ trợ code hiệu quả (visualstudio).

Sự ra đời của javascript:

https://www.educative.io/blog/javascript-versions-history

Javascript không liên quan tới java vì thời điểm đó java đang thống lĩnh thị trường nên đặt tên để ké frame.

ECMAScript: là đặc tả ngôn ngữ tài liệu nó không phải là ngôn ngữ. Còn js là 1 ngôn ngữ được cài đặt theo đặc tả của ECMAScript

Babel chuyển code ES6 -> ES5.

Javascript làm được gì:

- FE: reactjs, angular, vuejs, svelte, remix,...

- Mobile: React Native.
- BE: nodejs + express, nestjs,...
- DataBase: mongoDB,...

Thực thi code js:

cài đặt nodejs.

node index.js

code runner.

3. "use strict mode":

- từ ES5 có nhiều thay đổi cà conflicts với code trước đó, nên cần có cơ chế để quyết định sử dụng code mới hay cũ.
- "use strict" là cái flag để biết mình muốn dùng code mới (tức từ ES5 trờ về sau).
- hiện thì nếu ko nhắc gì thêm thì mặc định sữ có dùng "use strict".

```
// đặt use strict ở đầu file để apply strict mocde cho toàn bộ file js này
x = 5; // bị lỗi thì có strict mode, còn trước đó vẫn chạy được
```

- khi đã bật: "use strict" thì không có cách nào cancel được.
- lưu ý là "use strict" phải được nằm ở đầu file.
- còn nếu chỉ muốn bật strict mode cho 1 hàm thì đặt nó ở đầu hàm, tuy nhiên hầy như k dùng cách này.
- khi bạn có sử dụng class hoặc module thì strict mode sẽ được bật tự động mà không cần phải thêm "use strict"

> tham khảo thêm https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

4. làm quen với synctax trong js:

Làm quen với cú pháp (syntax) trong JavaScript là bước quan trọng đầu tiên khi học ngôn ngữ lập trình này. Dưới đây là một số khái niệm cơ bản:

1. Biến (Variables): Sử dụng để lưu trữ dữ liệu.

```
var x = 10;
let y = 20;
const z = 30;
```

2. Kiểu dữ liệu (Data Types): Bao gồm số, chuỗi, boolean, mảng, đối tượng, và null/undefined.

```
let num = 10;
let str = "Hello";
let isTrue = true;
```

```
let arr = [1, 2, 3];
let obj = { name: "John", age: 30 };
let empty = null;
z = 30;
```

3. Cấu trúc điều kiện (Conditional Statements): Sử dụng để thực thi các khối mã dựa trên điều kiện.

```
if (x > 10) {
    console.log("x lớn hơn 10");
} else {
    console.log("x nhỏ hơn hoặc bằng 10");
}
```

4. Vòng lặp (Loops): Sử dụng để lặp qua một tập hợp các giá trị.

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}

while (condition) {
    // do something
}
```

5. Hàm (Functions): Một khối mã được gói lại để thực thi một công việc cụ thể.

```
function add(a, b) {
    return a + b;
}
```

6. Phương thức (Methods): Hàm được gắn liền với một đối tượng.

```
let person = {
    name: "John",
    greet: function() {
        console.log("Hello, " + this.name);
    }
};
```

7. Sự kiện (Events): Các hành động người dùng khởi tạo trên trang web.

```
button.addEventListener('click', function() {
    console.log("Button clicked!");
});
```

5. cách đặt tên biến:

Đặt tên biến, hàm và các đối tượng trong JavaScript là một phần quan trọng của việc viết mã. Điều này không chỉ làm cho mã của bạn dễ đọc hơn mà còn giúp bạn và những người khác hiểu rõ mục đích của các phần mã đó. Dưới đây là một số quy tắc và gợi ý khi đặt tên trong JavaScript:

1. Sử dụng tên có ý nghĩa: Tên biến và hàm nên mô tả mục đích của chúng. Điều này giúp dễ dàng hiểu mã của bạn hơn.

```
// Không tốt
let a = 10;

// Tốt
let age = 10;
```

2. Sử dụng camelCase: CamelCase là quy ước phổ biến cho việc đặt tên trong JavaScript, trong đó mỗi từ bắt đầu bằng chữ cái viết hoa, ngoại trừ từ đầu tiên.

```
let myVariableName = "value";
function myFunctionName() {
    // code here
}
```

3. Tránh sử dụng tên ngắn gọn quá: Đặt tên ngắn gọn có thể làm cho mã của bạn khó hiểu. Hãy sử dụng tên mô tả và dài hơn nếu cần thiết để làm cho mã rõ ràng hơn.

```
// Không tốt
let x = 10;
// Tốt
let numberOfStudents = 10;
```

4. Tránh sử dụng tên giống từ khóa của JavaScript: Đừng đặt tên cho biến hoặc hàm trùng với các từ khóa như var, function, if, else, v.v.

```
// Không tốt
let function = () => {
    // code here
};
```

```
// Tốt
let myFunction = () => {
    // code here
};
```

5. Sử dụng tiền tố cho biến đối tượng: Đôi khi, việc thêm tiền tố cho biến đối tượng có thể giúp dễ dàng phân biệt chúng với các biến khác.

```
let personName = "John";
let carModel = "Toyota";
```

6. Tuân thủ quy ước của dự án hoặc chuẩn mã hóa: Trong một dự án cụ thể hoặc trong một nhóm phát triển, có thể có các quy ước đặt tên cụ thể. Tuân thủ các quy ước này giúp giữ cho mã được viết một cách nhất quán.

Nhớ rằng, quan trọng nhất là làm cho mã của bạn dễ đọc và dễ hiểu cho người khác.

6. biến

Trong JavaScript, let, const, và var đều được sử dụng để khai báo biến, nhưng chúng có những đặc điểm và cách hoạt động khác nhau.

1. var:

var đã tồn tại từ phiên bản cũ của JavaScript và có phạm vi hoạt động là phạm vi hàm (function scope). Biến được khai báo bằng var có thể được truy cập từ bất kỳ đâu trong hàm mà nó được khai báo. Biến được khai báo bằng var có thể được khai báo lại và gán lại giá trị mà không gây ra lỗi.

```
var x = 10;
console.log(x); // 10

if (true) {
    var y = 20;
}

console.log(y); // 20
```

2. let:

let được giới thiệu trong phiên bản ECMAScript 6 (ES6) và có phạm vi hoạt động là phạm vi khối (block scope). Biến được khai báo bằng let chỉ có thể truy cập được trong cùng một khối lệnh (block) mà nó được khai báo. Biến được khai báo bằng let không thể được khai báo lại trong cùng một phạm vi.

```
let x = 10;
console.log(x); // 10
```

```
if (true) {
    let y = 20;
    console.log(y); // 20
}

console.log(y); // Lỗi: y is not defined
```

3. const:

const cũng được giới thiệu trong ES6 và có các đặc điểm giống như let, nhưng giá trị của biến được khai báo bằng const không thể được thay đổi sau khi được gán. Biến được khai báo bằng const phải được gán một giá trị trong quá trình khai báo và không thể được khai báo lại hoặc gán lại giá trị.

```
const x = 10;
console.log(x); // 10

x = 20; // Lỗi: Assignment to constant variable.

const y; // Lỗi: Missing initializer in const declaration
```

Tóm lại, khi viết mã trong JavaScript, nên sử dụng const khi giá trị biến không thay đổi, và sử dụng let khi cần phải thay đổi giá trị của biến. Tránh sử dụng var nếu có thể để tránh các vấn đề về phạm vi và làm cho mã của bạn dễ bảo trì hơn.

7. Temporal Dead Zone

Trong JavaScript, "Temporal Dead Zone" (TDZ) là một khái niệm liên quan đến phạm vi và hoisting của biến được khai báo bằng let và const. Khi một biến được khai báo bằng let hoặc const, nó sẽ tồn tại trong phạm vi của block mà nó được khai báo, nhưng sẽ không thể truy cập vào giá trị của biến trước khi nó được khai báo.

TDZ diễn ra từ thời điểm khai báo biến đến thời điểm nó thực sự được gán giá trị. Trong khoảng thời gian này, bất kỳ cố gắng truy cập biến sẽ gây ra một lỗi. Điều này giúp tránh các lỗi liên quan đến sử dụng biến trước khi nó được khởi tạo, làm tăng tính dự đoán và đảm bảo an toàn trong mã JavaScript.

Ví dụ:

```
console.log(x); // ReferenceError: Cannot access 'x' before initialization
let x = 10;
```

Trong ví dụ này, biến x tồn tại trong TDZ từ thời điểm khai báo đến thời điểm gán giá trị. Việc truy cập x trước khi nó được khởi tạo sẽ dẫn đến lỗi.

8. Hoisting.

Trong JavaScript, "hoisting" là một quy tắc trong quá trình biên dịch mà khai báo biến và hàm được di chuyển lên đầu của phạm vi của chúng trước khi mã JavaScript thực sự được thực thi. Tuy nhiên, chỉ có phần khai báo

được hoisted, còn phần gán giá trị sẽ vẫn được giữ lại ở vị trí gốc trong mã.

Khi một biến được khai báo, JavaScript sẽ "nâng cao" phần khai báo lên trên đầu của phạm vi của biến đó. Điều này có nghĩa là bạn có thể truy cập biến trước khi nó được khai báo trong mã, mặc dù giá trị của nó sẽ là undefined nếu bạn cố gắng truy cập trước khi gán giá trị.

Ví dụ:

```
console.log(x); // undefined
var x = 10;
```

Sau khi mã trên được biên dịch, JavaScript sẽ xử lý như sau:

```
var x;
console.log(x); // undefined
x = 10;
```

Hãy nhớ rằng hoisting chỉ áp dụng cho phần khai báo, không phải phần gán giá trị. Điều này đối với cả biến và hàm.

9. kiểu dữ liệu cơ bản:

Trong JavaScript, có một số kiểu dữ liệu cơ bản, bao gồm:

String: Kiểu dữ liệu này được sử dụng để biểu diễn các chuỗi ký tự. Ví dụ:

```
var message = "Hello, world!";
```

Number: Kiểu dữ liệu này được sử dụng để biểu diễn các số. Cả số nguyên và số thập phân đều thuộc kiểu dữ liệu này. Ví dụ:

```
var num = 10;
var pi = 3.14;
```

Boolean: Kiểu dữ liệu này chỉ có hai giá trị: true hoặc false. Nó được sử dụng để biểu diễn các giá trị logic. Ví dụ:

```
var isTrue = true;
var isFalse = false;
```

Object: Kiểu dữ liệu này cho phép bạn tạo ra các đối tượng, mỗi đối tượng có thể chứa nhiều thuộc tính và phương thức. Ví dụ:

```
var person = {
    name: "John",
    age: 30,
    isStudent: false
};
```

Array: Kiểu dữ liệu này cho phép bạn tạo ra các mảng, mỗi mảng có thể chứa nhiều phần tử có cùng hoặc khác kiểu dữ liệu. Ví dụ:

```
var numbers = [1, 2, 3, 4, 5];
var fruits = ["apple", "banana", "orange"];
```

Các kiểu dữ liệu này cùng với các cấu trúc đi kèm như object và array là những khái niệm quan trọng trong JavaScript, giúp bạn xây dựng và làm việc với dữ liệu một cách linh hoạt và mạnh mẽ.