1. Trong JavaScript, có ba cách chính để khai báo biến: var, let, và const. Mỗi cách có phạm vi và hành vi riêng trong mã.

**var:** var là cách cũ nhất để khai báo biến trong JavaScript. Biến được khai báo bằng var có phạm vi của hàm hoặc toàn cục, có nghĩa là chúng có thể truy cập trong phạm vi của hàm hoặc toàn cục trong toàn bộ mã.

**let:** let được giới thiệu trong phiên bản JavaScript ES6 và giúp giải quyết một số vấn đề liên quan đến var. Biến được khai báo bằng let có phạm vi block, chỉ có thể truy cập trong cùng một block nơi chúng được khai báo.

**const:** const cũng được giới thiệu trong JavaScript ES6 và được sử dụng để khai báo một hằng số. Biến được khai báo bằng const không thể được gán lại giá trị mới sau khi khởi tạo và có phạm vi block tương tự như let.

function example() {

var x = 10;

let h= 10

if (true) {

var y = 20;

let z = 20;

let t = 20;

console.log(x); // 10

console.log(y); // 20

console.log(z); // 20

console.log(t); // 20

console.log(h); // 20 // const cũng vậy

}

console.log(y); // 20 (vẫn có thể truy cập được)

console.log(z); // Lỗi: z không được xác định

console.log(t); // Lỗi: t không được xác định

}

**2. Kiểu dữ liệu**

JavaScript có các kiểu dữ liệu cơ bản và kiểu dữ liệu phức tạp. Dưới đây là một số kiểu dữ liệu phổ biến trong JavaScript:

* **Kiểu dữ liệu cơ bản (Primitive data types):**

1. Chuỗi (String): Đại diện cho các dãy ký tự. Ví dụ: "Hello", 'World'.
2. Số (Number): Đại diện cho các số. Ví dụ: 10, 3.14.
3. Boolean: Đại diện cho giá trị true (đúng) hoặc false (sai).
4. Undefined: Đại diện cho một biến chưa được khai báo hoặc không có giá trị.
5. Null: Đại diện cho một giá trị không tồn tại hoặc rỗng.

* **Kiểu dữ liệu phức tạp:**

1. Object: Đại diện cho một đối tượng, bao gồm một tập hợp các thuộc tính và phương thức. Đây có thể là đối tượng được xây dựng sẵn như đối tượng Date hoặc Math, hoặc là đối tượng tự định nghĩa.
2. Mảng (Array): Đại diện cho một tập hợp các giá trị trong một mảng. Mảng là một loại đặc biệt của đối tượng trong JavaScript, nơi các giá trị được sắp xếp theo thứ tự.

* **Chuyển đổi kiểu dữ liệu và ép kiểu (Type Conversion and Coercion):**

Trong JavaScript, bạn có thể chuyển đổi kiểu dữ liệu từ một kiểu sang kiểu khác. Một số phép chuyển đổi kiểu tự động cũng xảy ra trong quá trình gọi là ép kiểu (coercion). Ví dụ:

* Chuyển đổi kiểu dữ liệu:
* Sử dụng các hàm như Number(), String(), Boolean() để chuyển đổi thành số, chuỗi, boolean tương ứng.
* Sử dụng các phép toán như +, -, \*, / để chuyển đổi tự động các kiểu dữ liệu trong các biểu thức.
* var x = 10; // Kiểu dữ liệu: số (number)
* var y = "5"; // Kiểu dữ liệu: chuỗi (string)
* var sum = x + y; // Kết quả: "105" (ép kiểu y thành chuỗi và thực hiện nối chuỗi)
* var difference = x - y; // Kết quả: 5 (ép kiểu y thành số và thực hiện phép trừ)
* var product = x \* y; // Kết quả: 50 (ép kiểu y thành số và thực hiện phép nhân)
* var quotient = x / y; // Kết quả: 2 (ép kiểu y thành số và thực hiện phép chia)
* Ép kiểu tự động (Implicit Coercion):
* JavaScript thực hiện ép kiểu tự động trong một số tình huống. Ví dụ:

var x = 10;

var y = "5";

var result = x + y; // Kết quả: "105" (ép kiểu y thành chuỗi và thực hiện nối chuỗi)

* Ép kiểu tường minh (Explicit Coercion):
* Bạn cũng có thể ép kiểu tường minh bằng cách sử dụng các hàm như parseInt(), parseFloat(), toString(), Number(), String(), Boolean() để chuyển đổi kiểu dữ liệu một cách rõ ràng.

1. Phạm vi (Scope)

Trong JavaScript, phạm vi xác định vị trí và thời gian tồn tại của các biến trong mã. Hai phạm vi chính là phạm vi toàn cục (global scope) và phạm vi block (block scope).

* Phạm vi toàn cục (global scope): Biến được khai báo bên ngoài bất kỳ hàm hoặc khối mã nào có phạm vi toàn cục. Biến toàn cục có thể được truy cập từ bất kỳ nơi nào trong mã JavaScript.
* Phạm vi block (block scope): Biến được khai báo trong một khối mã (block) như trong vòng lặp, câu điều kiện if, hoặc khối mã bất kỳ có phạm vi chỉ trong khối mã đó.

Phạm vi block giới hạn sự truy cập và sự tồn tại của biến chỉ trong khối mã đó, và không có sự xung đột với các biến cùng tên trong các khối mã khác.

> và < là hai toán tử so sánh trong JavaScript. Toán tử > được sử dụng để kiểm tra xem giá trị bên trái có lớn hơn giá trị bên phải hay không. Toán tử < được sử dụng để kiểm tra xem giá trị bên trái có nhỏ hơn giá trị bên phải hay không. Kết quả của mỗi toán tử là một giá trị boolean (true hoặc false) tùy thuộc vào kết quả của phép so sánh.

4. Toán tử (Operators)

Trong JavaScript, có nhiều loại toán tử khác nhau để thực hiện các phép tính, so sánh và kiểm tra các điều kiện. Dưới đây là một số loại toán tử phổ biến trong JavaScript:

* Toán tử số học (Arithmetic operators):
* + (Cộng), - (Trừ), \* (Nhân), / (Chia), % (Chia lấy dư)
* ++ (Tăng giá trị một đơn vị), -- (Giảm giá trị một đơn vị)
* Toán tử gán (The assignment operator):
* = (Gán giá trị), += (Cộng và gán), -= (Trừ và gán), \*= (Nhân và gán), /= (Chia và gán), %= (Chia lấy dư và gán)
* Toán tử so sánh (The comparison operators):
* == (So sánh giá trị, không kiểm tra kiểu dữ liệu), === (So sánh giá trị và kiểu dữ liệu),
* != (Không bằng, không kiểm tra kiểu dữ liệu), !== (Không bằng, kiểm tra kiểu dữ liệu),
* < (Nhỏ hơn), > (Lớn hơn), <= (Nhỏ hơn hoặc bằng), >= (Lớn hơn hoặc bằng)
* Toán tử logic (Logical operators):
* && (Và), || (Hoặc), ! (Phủ định)
* Toán tử ba ngôi (Ternary operator):
* condition ? expression1 : expression2

Nếu điều kiện là đúng, thì expression1 được thực hiện, ngược lại, expression2 được thực hiện.

* Toán tử typeof:
* typeof operand
* Trả về kiểu dữ liệu của toán hạng.

var a = 5;

var b = 10;

var c = a + b; // 15 (toán tử cộng)

var d = a - b; // -5 (toán tử trừ)

var e = a \* b; // 50 (toán tử nhân)

var f = b / a; // 2 (toán tử chia)

var g = b % a; // 0 (toán tử chia lấy dư)

var x = 20;

x++; // 21 (tăng giá trị x lên 1 đơn vị)

var y = 15;

y--; // 14 (giảm giá trị y xuống 1 đơn vị)

var name = "John";

var greeting = "Hello, " + name; // "Hello, John" (toán tử nối chuỗi)

var isActive = true;

var isLoggedOut = !isActive; // false (toán tử phủ định)

var result = (a > b) ? "A is greater than B" : "A is less than or equal to B";

// "A is less than or equal to B" (toán tử ba ngôi)

console.log(typeof name); // "string" (toán tử typeof)

**5. Câu lệnh điều kiện (Conditional statements)**

Trong JavaScript, có hai loại câu lệnh điều kiện phổ biến: câu lệnh if và câu lệnh switch. Chúng cho phép bạn thực hiện các hành động khác nhau dựa trên điều kiện được xác định.

a) Câu lệnh if + else if + else:

Cú pháp:

if (condition1) {

// Thực hiện hành động 1 nếu điều kiện 1 đúng

} else if (condition2) {

// Thực hiện hành động 2 nếu điều kiện 2 đúng

} else {

// Thực hiện hành động mặc định nếu không có điều kiện nào đúng

}

var age = 18;

if (age < 18) {

console.log("Bạn chưa đủ tuổi.");

} else if (age >= 18 && age <= 65) {

console.log("Bạn là người trưởng thành.");

} else {

console.log("Bạn là người già.");

}

b) Câu lệnh switch + case + default và break:

Cú pháp:

switch (expression) {

case value1:

// Thực hiện hành động 1 nếu expression có giá trị value1

break;

case value2:

// Thực hiện hành động 2 nếu expression có giá trị value2

break;

default:

// Thực hiện hành động mặc định nếu không có giá trị nào khớp

break;

}

var day = 3;

var dayName;

switch (day) {

case 1:

dayName = "Thứ Hai";

break;

case 2:

dayName = "Thứ Ba";

break;

case 3:

dayName = "Thứ Tư";

break;

case 4:

dayName = "Thứ Năm";

break;

case 5:

dayName = "Thứ Sáu";

break;

default:

dayName = "Cuối tuần";

break;

}

console.log("Hôm nay là " + dayName);

6. Câu lệnh lặp (Looping statements)

Trong JavaScript, có hai loại câu lệnh lặp phổ biến: vòng lặp for và vòng lặp while. Chúng cho phép bạn thực hiện một khối mã lặp đi lặp lại một số lần hoặc cho đến khi một điều kiện được đáp ứng.

a) Vòng lặp for:

Cú pháp:

for (initialization; condition; increment/decrement) {

// Thực hiện một khối mã

}

for (var i = 0; i < 5; i++) {

console.log(i);

}

b) Vòng lặp while:

Cú pháp:

while (condition) {

// Thực hiện một khối mã

}

var i = 0;

while (i < 5) {

console.log(i);

i++;

}

7. Hàm (Function)

Trong JavaScript, hàm là một khối mã có thể được gọi để thực hiện một tác vụ cụ thể. Hàm giúp tái sử dụng mã, tăng tính tổ chức và giúp tách biệt các phần mã khác nhau trong ứng dụng.

Cú pháp:

function functionName(parameter1, parameter2, ...) {

// Khối mã

return value; // (Nếu cần)

}

function greet(name) {

return "Hello, " + name + "!";

}

var greeting = greet("John");

console.log(greeting); // "Hello, John!"

8. Làm việc với mảng (Working with arrays)

Trong JavaScript, có một số phương thức hữu ích để làm việc với mảng. Dưới đây là một số phương thức phổ biến:

a) map(): Phương thức map() tạo ra một mảng mới bằng cách thực hiện một hàm cho từng phần tử của mảng ban đầu.

var numbers = [1, 2, 3, 4, 5];

var doubledNumbers = numbers.map(function(number) {

return number \* 2;

});

console.log(doubledNumbers); // [2, 4, 6, 8, 10]

b) filter(): Phương thức filter() tạo ra một mảng mới chứa các phần tử của mảng ban đầu mà thỏa mãn một điều kiện được xác định bởi hàm.

var numbers = [1, 2, 3, 4, 5];

var evenNumbers = numbers.filter(function(number) {

return number % 2 === 0;

});

console.log(evenNumbers); // [2, 4]

c) includes(): Phương thức includes() kiểm tra xem một giá trị có tồn tại trong mảng hay không và trả về true hoặc false.

var numbers = [1, 2, 3, 4, 5];

console.log(numbers.includes(3)); // true

console.log(numbers.includes(6)); // false

d) forEach(): Phương thức forEach() thực hiện một hàm cho từng phần tử của mảng.

var numbers = [1, 2, 3, 4, 5];

numbers.forEach(function(number) {

console.log(number);

});

e) find(): Phương thức find() trả về giá trị của phần tử đầu tiên trong mảng thỏa mãn một điều kiện được xác định bởi hàm.

var numbers = [1, 2, 3, 4, 5];

var foundNumber = numbers.find(function(number) {

return number > 3;

});

console.log(foundNumber); // 4

f) some(): Phương thức some() kiểm tra xem có ít nhất một phần tử trong mảng thỏa mãn một điều kiện được xác định bởi hàm và trả về true hoặc false.

var numbers = [1, 2, 3, 4, 5];

console.log(numbers.some(function(number) {

return number > 3;

})); // true

console.log(numbers.some(function(number) {

return number > 5;

})); // false

g) every(): Phương thức every() kiểm tra xem tất cả các phần tử trong mảng có thỏa mãn một điều kiện được xác định bởi hàm và trả về true hoặc false.

var numbers = [1, 2, 3, 4, 5];

console.log(numbers.every(function(number) {

return number > 0;

})); // true

console.log(numbers.every(function(number) {

return number > 3;

})); // false

9. Làm việc với đối tượng (Working with objects)

Trong JavaScript, có một số phương thức hữu ích để làm việc với đối tượng. Dưới đây là một số phương thức phổ biến:

a) Object.keys(): Phương thức Object.keys() trả về một mảng chứa tất cả các khóa của đối tượng.

var person = {

name: "John",

age: 30,

city: "New York"

};

var keys = Object.keys(person);

console.log(keys); // ["name", "age", "city"]

b) Object.values(): Phương thức Object.values() trả về một mảng chứa tất cả các giá trị của đối tượng.

Ví dụ:

var person = {

name: "John",

age: 30,

city: "New York"

};

var values = Object.values(person);

console.log(values); // ["John", 30, "New York"]

c) Object.entries(): Phương thức Object.entries() trả về một mảng chứa các cặp khóa-giá trị của đối tượng dưới dạng mảng con.

Ví dụ:

var person = {

name: "John",

age: 30,

city: "New York"

};

var entries = Object.entries(person);

console.log(entries);

// [["name", "John"], ["age", 30], ["city", "New York"]]