JavaScript

Buổi 3: Làm quen với mảng (Array). Làm việc với giá trị chuỗi.

Nội dung buổi 3

- Giới thiệu về mảng (Array).
- Phương thức cơ bản của mảng.
- Vòng lặp for.
- Vòng lặp while, do/while.
- Scoping - Hoisting.
- Đệ quy (Recursive).
- Phương thức cơ bản của chuỗi.

Giới thiệu về mảng (Array)

Giới thiệu về mảng (Array)

- Mảng (Array) là một dạng dữ liệu có thể chứa nhiều giá trị cùng một lúc. Các giá trị của mảng nằm trong cặp ngoặc [ ] và giữa các giá trị (lúc này được gọi là phần tử của mảng) ngăn cách nhau bởi dấu ",".
- Mỗi một phần từ trong mảng được gắn với một giá trị "index" là một số được đánh dấu theo thứ tự tự tăng dần và phần tử đầu tiên sẽ có số "index" là 0. Có thể sử dụng giá trị "index" làm vị trí để truy cập đến một phần tử cụ thể trong mảng.

Lấy giá trị từ mảng:

```
const students = ["Mai", "Phong", "Tuấn"];

students[0]; // "Mai"
students[1]; // "Phong"
students[2]; // "Tuấn"

students.length; // 3 -> Độ dài của mảng
```

Cập nhật giá trị trong mảng:

```
const students = ["Mai", "Phong", "Tuấn"];
students[0] = "Lan";
students[2] = "Minh";

console.log(students);
// => ["Lan", "Phong", "Minh"]
```

Phương thức cơ bản của mảng

Phương thức cơ bản của mảng

push: Thêm phần tử mới vào vị trí cuối cùng của mảng.

```
const students = ["Mai", "Phong", "Tuấn"];
students.push("Linh");
students; // ["Mai", "Phong", "Tuấn", "Linh"]
```

pop: Bỏ đi phần tử ở vị trí cuối cùng của mảng. Trả về phần tử bị bỏ đi đó.

```
const students = ["Mai", "Phong", "Tuấn"];
students.pop(); // "Tuấn"
students; // ["Mai", "Phong"]
```

shift: Bỏ đi phần tử ở vị trí đầu tiên của mảng. Trả về phần tử bị bỏ đi đó.

```
const students = ["Mai", "Phong", "Tuấn"];
students.shift(); // "Mai"
students; // ["Phong", "Tuấn"]
```

unshift: Thêm phần tử mới vào vị trí đầu tiên của mảng.

```
const students = ["Mai", "Phong", "Tuấn"];
students.unshift("Linh");
students; // ["Linh", "Mai", "Phong", "Tuấn"]
```

Phương thức cơ bản của mảng

indexOf: Trả về vị trí index của tham số truyền vào. Nếu không có trả về -1.

```
const students = ["Mai", "Phong", "Tuấn"];
students.indexOf("Mai"); // 0
students.indexOf("Phong"); // 1
students.indexOf("Tuấn"); // 2
students.indexOf("Linh"); // -1
```

includes: Kiểm tra một giá trị có phải là phần tử của một mảng.

```
const students = ["Mai", "Phong", "Tuấn"];
students.includes("Mai"); // true
students.includes("Linh"); // false
```

join: Nối các phần tử trong một mảng thành một chuỗi. Tham số truyền vào là ký tự ngăn cách giữa các phần tử

```
const students = ["Mai", "Phong", "Tuấn"];
students.join(); // "MaiPhongTuấn"
students.join(" "); // "Mai Phong Tuấn"
students.join("-"); // "Mai-Phong-Tuấn"
students.join(", "); // "Mai, Phong, Tuấn"
```

Vòng lặp là gì?

Vòng lặp là gì?

- Vòng lặp (loop) được áp dụng để thực hiện một khối code lặp đi lặp lại với các điều kiện được quy ước trước.
- Vòng lặp thường được sử dụng với các dạng dữ liệu như mảng (Array), Set, Map để lặp qua từng phần tử của các dạng dữ liệu này.
- Trong JavaScript, các kiểu dữ liệu trên được gọi là Iterator (Kiểu dữ liệu có thể lặp) và chúng có các phương thức đi kèm để tạo vòng lặp và trả về giá trị theo mong muốn của người lập trình. Ví dụ như, forEach, some, every, filter, map...
- Chú ý, khi sử dụng vòng lặp luôn phải đặt điều kiện để kết thúc vòng lặp (trừ các phương thức có sẵn của mảng) nếu không vòng lặp sẽ chạy vô tận và làm quá tải dung lượng dẫn tới treo ứng dụng.

Vòng lặp for

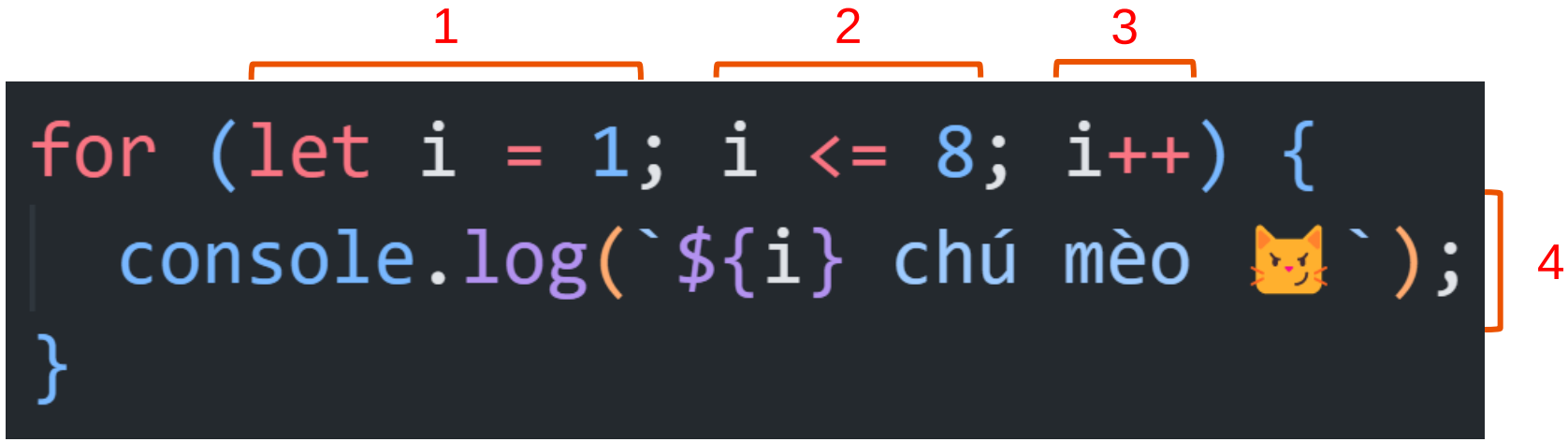
Vòng lặp for

Bài toán: Cần in ra cửa sổ console một chuỗi có dạng “1 chú mèo 🐱” từ 1 đến 8.

```
console.log("1 chú mèo 🐱");
console.log("2 chú mèo 🐱");
console.log("3 chú mèo 🐱");
console.log("4 chú mèo 🐱");
console.log("5 chú mèo 🐱");
console.log("6 chú mèo 🐱");
console.log("7 chú mèo 🐱");
console.log("8 chú mèo 🐱");
```

```
for (let i = 1; i <= 8; i++) {
  console.log(`${i} chú mèo 🐱`);
}
```

Vòng lặp for



Bước 1: Khai báo một biến i (không bắt buộc phải đặt là i) làm điểm mốc xuất phát (1).

Bước 2: Kiểm tra điều thỏa mãn của biến (2). Nếu không thỏa mãn, bỏ qua khối code trong vòng lặp và tiếp tục thực thi các khối tiếp theo.

Bước 3: Thực thi khối code (4).

Bước 4: Sau khi kết thúc khối code ở bước 3, gán giá trị mới cho biến i (3). Lặp lại từ **Bước 2**.

Vòng lặp for - Ngược chiều, Áp dụng với mảng

Bài toán: Vẫn là bài toán in ra chuỗi các chú mèo nhưng kết quả từ 8 - 1.

```
for (let i = 8; i >= 1; i--) {
  console.log(`${i} chú mèo 😾`);
}
```

Bài toán: Từ một mảng, in ra cửa sổ console tên và vị trí của các phần tử theo dạng “Bạn số 1 là Mai”...

```
const students = ["Mai", "Phong", "Tuấn"];

for (let i = 0; i < students.length; i++) {
  const studentName = students[i];
  const order = i + 1;
  console.log(`Bạn số ${order} là ${studentName}`);
}
```

Bài thực hành số 1

Bài thực hành số 1

Bài 1: Viết một hàm nhận tham số là một mảng gồm các giá trị số. Hàm trả về một mảng mới gồm các giá trị gấp đôi giá trị của mảng tham số truyền vào.

Bài 2: Viết một hàm nhận tham số là một mảng gồm các giá trị số. Hàm trả về một mảng mới gồm các giá trị số chẵn có trang mảng tham số truyền vào.

Bài 3: Viết một hàm nhận tham số là một số. Hàm trả về một mảng có các giá trị từ 0 cho đến giá trị tham số truyền vào. Ví dụ tham số truyền vào là 3 hàm trả về mảng [0, 1, 2, 3].

Bài 4: Viết một hàm tương tự như bài 3 kết quả trả về theo giá trị ngược lại, từ giá trị tham số đến 0.

Bài 5: Viết một hàm nhận tham số là một mảng gồm các giá trị số. Hàm kiểm tra xem trong mảng truyền vào có giá trị nào bằng tổng của các giá trị còn lại hay không. Nếu có trả về giá trị đó, nếu không có trả về null.

Vòng lặp while, do/while

Vòng lặp do/while

Bài toán: Đếm và in ra số lần lặp để tìm ra số 7 trong khoảng ngẫu nhiên từ 1 - 10.

do:

```
let count = 0;
let randomNumber;

while (randomNumber !== 7) {
  count++;
  randomNumber = Math.ceil(Math.random() * 10);
  console.log(`Lần ${count}: ${randomNumber}`);
}
```

do/while:

```
let count = 0;
let randomNumber;

do {
  count++;
  randomNumber = Math.ceil(Math.random() * 10);
  console.log(`Lần ${count}: ${randomNumber}`);
} while (randomNumber !== 7);
```

Bài thực hành số 2

Bài thực hành số 2

Bài 1: Viết lại bài 2 và bài 3 trong bài thực hành số 1 theo vòng lặp while.

Bài 2: Viết một hàm nhận tham số là một mảng gồm các số. Hàm trả về giá trị là số lớn nhất trong mảng truyền vào.

Bài 3: Viết một hàm nhận tham số là một mảng gồm các chuỗi là tên của người hoặc đồ vật... Hàm trả về một chuỗi có dạng như sau: Ví dụ tham số truyền vào là [“Mai”, “Vân”, “Tùng”, “Khánh”] thì kết quả trả về là “Mai, Vân, Tùng và Khánh”.

continue, break

continue, break

Bài toán: Sử dụng vòng lặp để tìm và trả về số chẵn đầu tiên tìm thấy được trong một mảng.

```
const numbers = [1, 2, 3, 4, 5, 6];
let result;

for (let i = 0; i < numbers.length; i++) {
  if (numbers[i] % 2 === 0) {
    result = numbers[i];
    break; → Kết thúc vòng lặp
  }
}
console.log(result); // 2
```

continue, break

Bài toán: Tạo ra một mảng mới chứa các giá trị là số chẵn từ mảng cho trước.

```
const numbers = [1, 2, 3, 4, 5, 6];
const evenNumbers = [];

for (let i = 0; i < numbers.length; i++) {
  if (numbers[i] % 2 !== 0) {
    continue;
  }

  evenNumbers.push(numbers[i]);
}
console.log(evenNumbers); // [2, 4, 6]
```

Bắt đầu vòng lặp tiếp theo. Không thực thi khối code bên dưới.

Scoping - Hoisting

Scoping

- Trong JavaScript, có 2 kiểu biến là *global variables* (*biến môi trường* hay *biến toàn cục*) và *local variables* (*biến địa phương* hay *biến cục bộ*). Biến global là biến được khai báo ngoài block code và biến local là biến khai báo bên trong block code.
- Trong block code có thể gọi và sử dụng biến global nhưng từ ở bên ngoài block code không thể gọi được biến local trong block code.
- Các biến global hoặc các biến local trong cùng một block code không được đặt trùng tên. Các biến local ở khác block code hoặc giữa biến global và biến local có thể đặt trùng tên. Tuy nhiên, không khuyến khích cách đặt trùng tên biến.
- Nếu trong một block code khai báo một biến local trùng tên với biến global khác thì trong block đó sẽ sử dụng biến theo biến local.

Scoping

```
const bookCat = "novel";

if (true) {
  const publisher = "Alpha Books";
  console.log(bookCat); // "novel"
}

if (true) {
  console.log(bookCat); // "novel"
  console.log(publisher);
  // ERROR: publisher is undefined
}

console.log(publisher);
// ERROR: pulisher is undefined
```

```
const bookCat = "novel";

if (true) {
  const bookCat = "rom-com";
  console.log(bookCat); // "rom-com"
}

if (true) {
  const bookCat = "detective";
  console.log(bookCat); // "detective"
}

console.log(bookCat); // "novel"
```

Chú ý: *Không khuyến khích đặt trùng tên biến.*

Hoisting

Khái niệm hoisting trong JavaScript được hiểu là các hành vi cho phép sử dụng biến và hàm trong các trường hợp sau sau:

- Trước khi biến và hàm đó được khai báo.
- Khi biến và hàm đó nằm trong một block code khác.

Hoisting được áp dụng với cú pháp khai báo biến *var* và hàm viết theo cú pháp *declarations.*

```
console.log(message); // "Hello"
var message = "Hello";

console.log(sayHi()); // "Hi"
function sayHi() {
  return "Hi";
}
```

JavaScript phiên bản mới đã bỏ tính năng này

Chú ý: *Khái niệm về hoisting chỉ sử dụng để nắm được hành vi của JavaScript và sửa lỗi.* ***Không sử dụng var*** *để khai báo biến,* ***không đặt biến trùng tên*** *và* ***không sử dụng hàm khai báo trong block code ở ngoài môi trường global****.*

```
if (true) {
  var message = "Hello";
}
console.log(message); // "Hello"

if (true) {
  function sayHi() {
    return "Hi";
  }
}
console.log(sayHi()); // "Hi"
```

Ứng dụng của Hoisting

```
const doActionA = () => {
  // Action A...
};

const doActionB = () => {
  // Action B...
};

const doSomething = () => {
  doActionA();
  doActionB();
};
```


```
function doSomething() {
  doActionA();
  doActionB();
}

function doActionA() {
  // Action A...
}

function doActionB() {
  // Action B...
}
```

Đệ quy (Recursive)

Đệ quy (Recursive)

- Đệ quy xảy ra khi một sự vật được định nghĩa theo chính nó hoặc thuộc loại của nó.
- Khi sử dụng đệ quy cần biết được điểm kết thúc nếu không sẽ bị gặp trường hợp gọi hàm vô tận dẫn đến tràn dung lượng.

Bài toán: Viết một hàm nhận tham số là một số. Hàm trả về giá trị là giai thừa của số đó.

Sử dụng vòng lặp for:

```
function calcFactorial(number) {
  let result = 1;

  for (let i = 2; i <= number; i++) {
    result = result * i;
  }

  return result;
}
```

Sử dụng đệ quy:

```
function calcFactorial(number) {
  if (number === 1) {
    return 1;
  }

  return number * calcFactorial(number - 1);
}
```

Ứng dụng đệ quy

Bài toán: Làm phẳng mảng.

```
function flatenArray(array) {
  // 1) Tạo biến để lưu kết quả
  let result = [];
  // 2) Tiến hành lặp mảng
  for (let i = 0; i < array.length; i++) {
    // 3) Kiểm tra từng giá trị trong mảng
    if (!Array.isArray(array[i])) {
      // 4) Nếu giá trị không phải là mảng, thêm vào biến "result"
      result.push(array[i]);
    } else {
      // 5) Nếu giá trị là mảng, truyền giá trị vào hàm "flatenArray"
      // để làm phẳng mảng
      const newArr = flatenArray(array[i]);
      // 6) Nối mảng đã làm phẳng vào biến "result"
      result.concat(newArr);
    }
  }
  return result;
}
flatenArray([1, [2, [3, 4], 5], 6]); // [1, 2, 3, 4, 5, 6]
```

Phương thức cơ bản của chuỗi

Phương thức cơ bản của chuỗi

indexOf, lastIndexOf, charAt:

```
"JavaScript".indexOf("J"); // 0
"JavaScript".indexOf("a"); // 1
"JavaScript".indexOf("t"); // 9
"JavaScript".lastIndexOf("a"); // 3
"JavaScript".charAt(0); // "J"
"JavaScript".charAt(3); // "a"
```

slice, length:

```
const word = "JavaScript";
const wordLength = word.length; // 10
word.slice(0, 4); // "Java"
word.slice(4); // "Script"
```

toLowerCase, toUpperCase:

```
"JavaScript".toLowerCase(); // "javascript"
"JavaScript".toUpperCase(); // "JAVASCRIPT"
```

Phương thức cơ bản của chuỗi

indexOf, lastIndexOf, charAt:

```
"JavaScript is Amazing".replace("a", ".");
// => "J.vaScript is Amazing"
"JavaScript is Amazing".replace(/a/g, ".");
// => "J.v.Script is Am.zing"
```

trim:

```
"abc   ".trim(); // "abc"
"    abc".trim(); // "abc"
"    abc   ".trim(); // "abc"
```

includes:

```
"abc".includes("a"); // true
"abc".includes("b"); // true
"abc".includes("g"); // false
```

Phương thức cơ bản của chuỗi

split:

```
"I am John".split();
// => ['I', ' ', 'a', 'm', ' ', 'J', 'o', 'h', 'n']
"I am John".split("");
// => ['I', ' ', 'a', 'm', ' ', 'J', 'o', 'h', 'n']
"I am John".split(" ");
// => ['I', 'am', 'John']
"I-am-John".split("-");
// => ['I', 'am', 'John']
```

includes:

```
"1".padStart(4, "0"); // "0001"
"12".padStart(4, "0"); // "0012"
"123".padStart(4, "0"); // "0123"
"1".padEnd(4, "-"); // "1---"
"12".padEnd(4, "-"); // "12--"
"123".padEnd(4, "-"); // "123-"
```

Bài thực hành số 3

Bài thực hành số 3

Bài 1: Viết một hàm nhận tham số là một chuỗi và trả về kết quả là true nếu chuỗi đó có các ký tự là “a”, “an”, “the”.

Bài 2: Viết một hàm nhận tham số là một mảng gồm các giá trị chuỗi. Hàm trả về một mảng có các giá trị là giá trị của mảng tham số truyền vào với kiểu chữ viết thường. Ví dụ tham số là [“The”, “GOOD”, “stOry”], kết quả trả về [“the”, “good”, “story”].

Bài 3: Viết một hàm nhận tham số là một chuỗi là một địa chỉ email. Hàm trả về số ký tự có trong địa chỉ email đó, không tính khoảng trắng.

Bài 4: Viết một hàm nhận tham số là một chuỗi là một đoạn văn. Hàm trả về chuỗi nhận được từ tham số với tất cả các chữ cái đầu của các từ viết hoa. Ví dụ tham số là “It is a good day”, kết quả trả về “It Is A Good Day”.

Bài 5: Viết một hàm nhận tham số là một mảng gồm 5 giá trị số, trong đó giá trị đầu tiên đại diện số giờ 0 (0 - 23), giá trị thứ 2 của mảng đại diện số phút (0 - 59), giá trị thứ 3 đại diện cho ngày (1 - 31), giá trị thứ 4 đại diện cho tháng (1 - 12) và giá trị cuối cùng đại diện cho số năm (> 0). Viết hàm trả về giá trị theo mẫu sau: “05:10 08/10/2022 AM”. Kiểm tra điều kiện hợp lệ của các giá trị trong mảng ở tham số, nếu có giá trị không hợp lệ hàm trả về undefined.

Hoàn thành JavaScript – Buổi 3

Good job!