

Trong tài liệu thực hành này, chúng ta sẽ sử dụng Node.js làm công cụ biên dịch code javascript.

Dưới đây là các bước chi tiết để cài đặt và sử dụng Node.js:

Cài đặt Node.js

Bước 1: Tải và cài đặt Node.js

- Chúng ta có thể tải phiên bản Node.js từ trang web chính thức: <https://nodejs.org>
- Trang web này cung cấp hai phiên bản chính:
 - LTS (Long Term Support): Phiên bản này là bản ổn định, được hỗ trợ lâu dài. Khuyên dùng cho các dự án sản phẩm thực tế.
 - Current: Phiên bản mới nhất với các tính năng mới nhất nhưng không ổn định như LTS.

Bước 2: Cài đặt Node.js trên hệ điều hành

→ Windows:

- Sau khi tải file .msi, mở file và làm theo các bước hướng dẫn để cài đặt.
- Trong quá trình cài đặt, Chúng ta có thể chọn thêm Node.js vào biến môi trường PATH (mặc định là có).

→ macOS:

- Tải file .pkg và mở để cài đặt như các ứng dụng thông thường trên macOS.
- Ngoài ra, Chúng ta có thể sử dụng Homebrew để cài đặt từ terminal:

brew install node

Bước 3: Kiểm tra cài đặt

Sau khi cài đặt xong, mở Terminal (hoặc Command Prompt trên Windows) và nhập lệnh sau để kiểm tra phiên bản Node.js và npm:



The screenshot shows a terminal window with several tabs at the top: PROBLEMS, OUTPUT, TERMINAL (which is underlined), PORTS, and POSTMAN CONSOLE. Below the tabs, there are three entries in the terminal history:

- (base) lamtanphuong@192 ~ % node -v
v21.7.1
- (base) lamtanphuong@192 ~ % npm -v
10.5.0
- (base) lamtanphuong@192 ~ % [empty box]

Nếu Chúng ta thấy các phiên bản hiển thị, nghĩa là Node.js đã được cài đặt thành công.

Bài thực hành số 01

Câu lệnh điều kiện

1. Kiểm tra số dương

Bài tập: Viết một chương trình kiểm tra xem một số có phải là số dương hay không.

Gợi ý:

```
const number = 5;  
if (number > 0) {  
    console.log("Số dương");  
} else {  
    console.log("Không phải số dương");  
}
```

2. Tìm số lớn nhất

Bài tập: Nhập vào hai số và tìm số lớn nhất.

Gợi ý:

```
const a = 3, b = 7;  
if (a > b) {  
    console.log("Số lớn nhất là:", a);  
} else {  
    console.log("Số lớn nhất là:", b);  
}
```

3. Phân loại tuổi

Bài tập: Nhập tuổi và in ra nhóm tuổi: trẻ em (dưới 18), người lớn (18-60), người già (>60).

Gợi ý:

```
const age = 25;  
if (age < 18) {  
    console.log("Trẻ em");  
} else if (age <= 60) {  
    console.log("Người lớn");  
} else {  
    console.log("Người già");  
}
```

4. Kiểm tra số chẵn lẻ

Bài tập: Viết chương trình kiểm tra xem một số có phải là số chẵn hay không.

Gợi ý:

```
const number = 4;  
if (number % 2 === 0) {  
    console.log("Số chẵn");  
} else {  
    console.log("Số lẻ");  
}
```

5. Kiểm tra năm nhuận

Bài tập: Kiểm tra xem một năm có phải là năm nhuận hay không.

Gợi ý:

```
const year = 2024;
if ((year % 4 === 0 && year % 100 !== 0) || year % 400 === 0) {
    console.log("Năm nhuận");
} else {
    console.log("Không phải năm nhuận");
}
```

6. Xếp loại điểm

Bài tập: Dựa trên điểm số, xếp loại học sinh:

- ≥ 90: Xuất sắc
- 80-89: Giỏi
- 65-79: Khá
- < 65: Trung bình

Gợi ý:

```
const score = 85;
if (score >= 90) {
    console.log("Xuất sắc");
} else if (score >= 80) {
    console.log("Giỏi");
} else if (score >= 65) {
    console.log("Khá");
} else {
    console.log("Trung bình");
}
```

7. Tính giá trị tuyệt đối

Bài tập: Tìm giá trị tuyệt đối của một số.

Gợi ý:

```
const number = -10;
const absolute = number >= 0 ? number : -number;
console.log("Giá trị tuyệt đối:", absolute);
```

8. Kiểm tra ký tự nguyên âm

Bài tập: Nhập một ký tự và kiểm tra xem đó có phải là nguyên âm hay không.

Gợi ý:

```
const char = 'a';
if ('aeiou'.includes(char.toLowerCase())) {
    console.log("Nguyên âm");
} else {
    console.log("Phụ âm");
}
```

9. Sử dụng switch-case

Bài tập: Nhập vào một số từ 1 đến 7, in ra ngày trong tuần.

Gợi ý:

```
const day = 3;
switch (day) {
```

```
case 1: console.log("Thứ Hai"); break;
case 2: console.log("Thứ Ba"); break;
case 3: console.log("Thứ Tư"); break;
case 4: console.log("Thứ Năm"); break;
case 5: console.log("Thứ Sáu"); break;
case 6: console.log("Thứ Bảy"); break;
case 7: console.log("Chủ Nhật"); break;
default: console.log("Số không hợp lệ");
}
```

10. Kiểm tra chữ cái viết hoa

Bài tập: Nhập một ký tự và kiểm tra xem nó có phải chữ cái viết hoa hay không.

Gợi ý:

```
const char = 'A';
if (char >= 'A' && char <= 'Z') {
    console.log("Chữ cái viết hoa");
} else {
    console.log("Không phải chữ cái viết hoa");
}
```

Vòng lặp

11. In số từ 1 đến 10

Bài tập: Sử dụng vòng lặp for để in các số từ 1 đến 10.

Gợi ý:

```
for (let i = 1; i <= 10; i++) {
    console.log(i);
}
```

12. Tính tổng các số từ 1 đến n

Bài tập: Nhập một số n và tính tổng các số từ 1 đến n.

Gợi ý:

```
const n = 5;
let sum = 0;
for (let i = 1; i <= n; i++) {
    sum += i;
}
console.log("Tổng là:", sum);
```

13. In số chẵn từ 1 đến n

Bài tập: Sử dụng vòng lặp for để in tất cả các số chẵn từ 1 đến n.

Gợi ý:

```
const n = 10;
for (let i = 1; i <= n; i++) {
    if (i % 2 === 0) {
        console.log(i);
    }
}
```

14. Đếm số chữ số

Bài tập: Đếm số chữ số của một số nguyên dương n.

Gợi ý:

```
const n = 12345;
let count = 0;
let temp = n;
while (temp > 0) {
    count++;
    temp = Math.floor(temp / 10);
}
console.log("Số chữ số:", count);
```

15. Kiểm tra số nguyên tố

Bài tập: Kiểm tra xem một số n có phải là số nguyên tố hay không.

Gợi ý:

```
const n = 7;
let isPrime = true;
if (n < 2) isPrime = false;
for (let i = 2; i <= Math.sqrt(n); i++) {
    if (n % i === 0) {
        isPrime = false;
        break;
    }
}
console.log(isPrime ? "Số nguyên tố" : "Không phải số nguyên tố");
```

16. Tính giai thừa

Bài tập: Tính giai thừa của một số n.

Gợi ý:

```
const n = 5;
let factorial = 1;
for (let i = 1; i <= n; i++) {
    factorial *= i;
}
console.log("Giai thừa là:", factorial);
```

17. Tính Fibonacci

Bài tập: Tính số Fibonacci thứ n.

Gợi ý:

```
const n = 7;
let a = 0, b = 1;
for (let i = 2; i <= n; i++) {
    const next = a + b;
    a = b;
    b = next;
}
console.log("Fibonacci thứ n:", b);
```

18. Tìm số lớn nhất trong mảng

Bài tập: Tìm số lớn nhất trong một mảng số nguyên.

Gợi ý:

```
const arr = [1, 5, 3, 9, 2];
let max = arr[0];
for (const num of arr) {
    if (num > max) max = num;
}
```

```
console.log("Số lớn nhất:", max);
```

19. In bảng cửu chương

Bài tập: In bảng cửu chương từ 1 đến 10.

Gợi ý:

```
for (let i = 1; i <= 10; i++) {  
    for (let j = 1; j <= 10; j++) {  
        console.log(` ${i} x ${j} = ${i * j}`);  
    }  
    console.log("---");  
}
```

20. Đảo ngược chuỗi

Bài tập: Nhập một chuỗi và đảo ngược chuỗi đó.

Gợi ý:

```
const str = "hello";  
let reversed = "";  
for (let i = str.length - 1; i >= 0; i--) {  
    reversed += str[i];  
}  
console.log("Chuỗi đảo ngược:", reversed);
```

Kiểu dữ liệu (Data Type)

21. Kiểm tra kiểu dữ liệu

Bài tập: Viết một chương trình kiểm tra kiểu dữ liệu của một biến và in ra kết quả.

Gợi ý:

```
const value = 42;  
console.log(typeof value); // Output: "number"
```

22. Chuyển đổi kiểu dữ liệu

Bài tập: Chuyển đổi một chuỗi "123" thành số và một số 456 thành chuỗi.

Gợi ý:

```
const str = "123";  
const num = 456;  
console.log(Number(str)); // Output: 123  
console.log(String(num)); // Output: "456"
```

23. So sánh giá trị và kiểu dữ liệu

Bài tập: So sánh giá trị và kiểu dữ liệu của hai biến sử dụng == và ===.

Gợi ý:

```
const a = "5";  
const b = 5;  
console.log(a == b); // Output: true (chỉ so sánh giá trị)  
console.log(a === b); // Output: false (so sánh cả giá trị và kiểu dữ liệu)
```

24. Kiểm tra NaN

Bài tập: Viết chương trình kiểm tra xem một giá trị có phải là NaN hay không.

Gợi ý:

```
const value = NaN;
```

```
console.log(Number.isNaN(value)); // Output: true
```

25. Làm tròn số

Bài tập: Làm tròn một số thực đến số nguyên gần nhất, lên hoặc xuống.

Gợi ý:

```
const num = 5.7;  
console.log(Math.round(num)); // Output: 6 (làm tròn gần nhất)  
console.log(Math.floor(num)); // Output: 5 (làm tròn xuống)  
console.log(Math.ceil(num)); // Output: 6 (làm tròn lên)
```

26. Chuyển đổi mảng sang chuỗi

Bài tập: Chuyển một mảng thành chuỗi, ngăn cách bởi dấu phẩy hoặc ký tự khác.

Gợi ý:

```
const arr = [1, 2, 3, 4];  
console.log(arr.join(", ")); // Output: "1,2,3,4"  
console.log(arr.join("-")); // Output: "1-2-3-4"
```

27. Kiểm tra kiểu dữ liệu Object

Bài tập: Kiểm tra xem một giá trị có phải là một Object hay không.

Gợi ý:

```
const value = { name: "Alice" };  
console.log(typeof value === "object" && value !== null); // Output: true
```

28. Làm việc với Boolean

Bài tập: Chuyển một giá trị bất kỳ thành kiểu Boolean và kiểm tra kết quả.

Gợi ý:

```
const value = 0;  
console.log(Boolean(value)); // Output: false (0 là falsy)  
const value2 = "hello";  
console.log(Boolean(value2)); // Output: true (chuỗi không rỗng là truthy)
```

29. Truy cập phần tử trong mảng

Bài tập: Lấy phần tử đầu tiên và phần tử cuối cùng của một mảng.

Gợi ý:

```
const arr = [10, 20, 30, 40];  
console.log(arr[0]); // Output: 10  
console.log(arr[arr.length - 1]); // Output: 40
```

30. Tách ký tự từ chuỗi

Bài tập: Lấy ký tự đầu tiên, ký tự cuối cùng và chiều dài của một chuỗi.

Gợi ý:

```
const str = "";  
console.log(str[0]); // Output: "J" (ký tự đầu tiên)  
console.log(str[str.length - 1]); // Output: "t" (ký tự cuối cùng)  
console.log(str.length); // Output: 10 (chiều dài chuỗi)
```

Bài thực hành số 02

Hàm cơ bản

1. Tạo hàm cơ bản

Bài tập: Viết một hàm in ra "Hello, World!"

Gợi ý:

```
function sayHello() {  
    console.log("Hello, World!");  
}  
sayHello(); // Gọi hàm
```

2. Hàm có tham số

Bài tập: Viết một hàm nhận một tên và in ra "Hello, [name]!"

Gợi ý:

```
function greet(name) {  
    console.log(`Hello, ${name}!`);  
}  
greet("Alice");
```

3. Hàm trả về giá trị

Bài tập: Viết một hàm nhận hai số và trả về tổng của chúng.

Gợi ý:

```
function add(a, b) {  
    return a + b;  
}  
console.log(add(5, 3));
```

4. Kiểm tra số chẵn lẻ

Bài tập: Viết một hàm kiểm tra xem một số có phải là số chẵn hay không.

Gợi ý:

```
function isEven(number) {  
    return number % 2 === 0;  
}  
console.log(isEven(4)); // true  
console.log(isEven(7)); // false
```

5. Hàm tính giai thừa

Bài tập: Viết một hàm tính giai thừa của một số.

Gợi ý:

```
function factorial(n) {  
    let result = 1;  
    for (let i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}  
console.log(factorial(5)); // 120
```

6. Tính trung bình

Bài tập: Viết hàm tính trung bình của một mảng số.

Gợi ý:

```
function average(arr) {  
    const sum = arr.reduce((total, num) => total + num, 0);  
    return sum / arr.length;  
}  
console.log(average([1, 2, 3, 4, 5])); // 3
```

7. Kiểm tra số nguyên tố

Bài tập: Viết một hàm kiểm tra xem một số có phải là số nguyên tố hay không.

Gợi ý:

```
function isPrime(n) {  
    if (n < 2) return false;  
    for (let i = 2; i <= Math.sqrt(n); i++) {  
        if (n % i === 0) return false;  
    }  
    return true;  
}  
console.log(isPrime(7)); // true  
console.log(isPrime(10)); // false
```

8. Tìm số lớn nhất

Bài tập: Viết một hàm nhận ba số và trả về số lớn nhất.

Gợi ý:

```
function max(a, b, c) {  
    return Math.max(a, b, c);  
}  
console.log(max(3, 7, 5)); // 7
```

9. Hàm mặc định

Bài tập: Viết một hàm nhận hai số, nếu không truyền số thứ hai thì mặc định là 10.

Gợi ý:

```
function multiply(a, b = 10) {  
    return a * b;  
}  
console.log(multiply(5)); // 50
```

10. Tính chu vi hình tròn

Bài tập: Viết hàm tính chu vi hình tròn với bán kính r.

Gợi ý:

```
function circleCircumference(r) {  
    return 2 * Math.PI * r;  
}  
console.log(circleCircumference(5)); // 31.4159
```

11. Đảo ngược chuỗi

Bài tập: Viết một hàm đảo ngược chuỗi.

Gợi ý:

```
function reverseString(str) {  
    return str.split("").reverse().join("");  
}
```

```
console.log(reverseString("hello")); // "olleh"
```

12. Kiểm tra chuỗi đối xứng

Bài tập: Viết một hàm kiểm tra xem chuỗi có đối xứng (palindrome) hay không.

Gợi ý:

```
function isPalindrome(str) {  
    const reversed = str.split("").reverse().join("");  
    return str === reversed;  
}  
console.log(isPalindrome("radar")); // true  
console.log(isPalindrome("hello")); // false
```

13. Tính tổng các số trong mảng

Bài tập: Viết hàm tính tổng các số trong một mảng.

Gợi ý:

```
function sumArray(arr) {  
    return arr.reduce((total, num) => total + num, 0);  
}  
console.log(sumArray([1, 2, 3, 4])); // 10
```

14. Lọc số chẵn

Bài tập: Viết hàm trả về các số chẵn từ một mảng.

Gợi ý:

```
function filterEven(arr) {  
    return arr.filter(num => num % 2 === 0);  
}  
console.log(filterEven([1, 2, 3, 4])); // [2, 4]
```

15. Đếm ký tự trong chuỗi

Bài tập: Viết hàm đếm số lần xuất hiện của một ký tự trong chuỗi.

Gợi ý:

```
function countChar(str, char) {  
    return str.split(char).length - 1;  
}  
console.log(countChar("hello", "l")); // 2
```

16. Sắp xếp mảng

Bài tập: Viết hàm sắp xếp mảng theo thứ tự tăng dần.

Gợi ý:

```
function sortArray(arr) {  
    return arr.sort((a, b) => a - b);  
}  
console.log(sortArray([3, 1, 4, 2])); // [1, 2, 3, 4]
```

17. Tìm giá trị lớn nhất trong mảng

Bài tập: Viết hàm tìm giá trị lớn nhất trong mảng.

Gợi ý:

```
function findMax(arr) {  
    return Math.max(...arr);  
}  
console.log(findMax([3, 7, 1, 5])); // 7
```

18. Hàm đệ quy

Bài tập: Viết hàm đệ quy tính tổng từ 1 đến n.

Gợi ý:

```
function sum(n) {  
    if (n === 1) return 1;  
    return n + sum(n - 1);  
}  
console.log(sum(5)); // 15
```

19. Hàm callback

Bài tập: Viết hàm sử dụng callback để xử lý mảng.

Gợi ý:

```
function processArray(arr, callback) {  
    return arr.map(callback);  
}  
console.log(processArray([1, 2, 3], num => num * 2)); // [2, 4, 6]
```

20. Tính số Fibonacci

Bài tập: Viết hàm tính số Fibonacci thứ n.

Gợi ý:

```
function fibonacci(n) {  
    if (n <= 1) return n;  
    return fibonacci(n - 1) + fibonacci(n - 2);  
}  
console.log(fibonacci(7)); // 13
```

21. Đếm số nguyên tố trong mảng

Bài tập: Viết hàm đếm số nguyên tố trong mảng.

Gợi ý:

```
function countPrimes(arr) {  
    const isPrime = num => {  
        if (num < 2) return false;  
        for (let i = 2; i <= Math.sqrt(num); i++) {  
            if (num % i === 0) return false;  
        }  
        return true;  
    };  
    return arr.filter(isPrime).length;  
}  
console.log(countPrimes([1, 2, 3, 4, 5])); // 3
```

22. Viết hàm tính khoảng cách giữa hai điểm trong không gian 2D

Bài tập: Viết một hàm nhận vào tọa độ hai điểm (x_1, y_1) và (x_2, y_2) và trả về khoảng cách giữa chúng.

Gợi ý:

```
function distance(x1, y1, x2, y2) {  
    return Math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2);  
}  
console.log(distance(0, 0, 3, 4)); // 5
```

23. Viết hàm kiểm tra mảng con

Bài tập: Viết hàm kiểm tra xem một mảng có phải là mảng con của một mảng khác không.

Gợi ý:

```
function isSubArray(arr, subArr) {  
    return subArr.every(val => arr.includes(val));  
}  
console.log(isSubArray([1, 2, 3, 4], [2, 3])); // true  
console.log(isSubArray([1, 2, 3, 4], [5])); // false
```

24. Viết hàm lấy ngẫu nhiên phần tử từ mảng

Bài tập: Viết một hàm trả về một phần tử ngẫu nhiên từ một mảng.

Gợi ý:

```
function getRandomElement(arr) {  
    const index = Math.floor(Math.random() * arr.length);  
    return arr[index];  
}  
console.log(getRandomElement([1, 2, 3, 4, 5])); // Ví dụ: 3
```

25. Viết hàm lọc các phần tử duy nhất trong mảng

Bài tập: Viết một hàm loại bỏ các phần tử trùng lặp trong mảng.

Gợi ý:

```
function uniqueElements(arr) {  
    return [...new Set(arr)];  
}  
console.log(uniqueElements([1, 2, 2, 3, 4, 4])); // [1, 2, 3, 4]
```

26. Viết hàm nhóm các phần tử mảng theo một tiêu chí

Bài tập: Viết một hàm nhận vào một mảng và một callback để nhóm các phần tử theo tiêu chí.

Gợi ý:

```
function groupBy(arr, callback) {  
    return arr.reduce((result, item) => {  
        const key = callback(item);  
        if (!result[key]) result[key] = [];  
        result[key].push(item);  
        return result;  
    }, {});  
}  
console.log(groupBy([6.1, 4.2, 6.3], Math.floor));  
// { '4': [4.2], '6': [6.1, 6.3] }
```

27. Viết hàm tính tổng các số lớn hơn giá trị cho trước

Bài tập: Viết hàm nhận vào một mảng số và một giá trị n, trả về tổng các số lớn hơn n.

Gợi ý:

```
function sumGreaterThan(arr, n) {  
    return arr.filter(num => num > n).reduce((sum, num) => sum + num, 0);  
}  
console.log(sumGreaterThan([1, 5, 8, 10], 5)); // 18 (8 + 10)
```

28. Viết hàm tìm tất cả các tổ hợp có thể của một mảng

Bài tập: Viết một hàm trả về tất cả các tổ hợp có thể của một mảng.

Gợi ý:

```
function getCombinations(arr) {  
    const result = [];  
    const generate = (combo, index) => {  
        result.push(combo);  
        for (let i = index; i < arr.length; i++) {  
            generate([...combo, arr[i]], i + 1);  
        }  
    };  
    generate([], 0);  
    return result;  
}  
console.log(getCombinations([1, 2, 3]));  
// [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]
```

29. Viết hàm so khớp mẫu chuỗi

Bài tập: Viết hàm kiểm tra xem một chuỗi con có xuất hiện trong một chuỗi lớn hơn không, không phân biệt chữ hoa/chữ thường.

Gợi ý:

```
function isSubstring(str, substr) {  
    return str.toLowerCase().includes(substr.toLowerCase());  
}  
console.log(isSubstring(" is awesome", "AWESOME")); // true
```

30. Viết hàm tìm giá trị gần nhất

Bài tập: Viết hàm tìm số trong mảng gần nhất với một giá trị cho trước.

Gợi ý:

```
function findClosest(arr, target) {  
    return arr.reduce((closest, num) =>  
        Math.abs(num - target) < Math.abs(closest - target) ? num : closest  
    );  
}  
console.log(findClosest([1, 5, 9, 15], 10)); // 9
```

31. Viết hàm đệ quy để tính tổng chữ số

Bài tập: Viết một hàm đệ quy tính tổng các chữ số của một số.

Gợi ý:

```
function sumDigits(num) {  
    if (num < 10) return num;  
    return (num % 10) + sumDigits(Math.floor(num / 10));  
}  
console.log(sumDigits(1234)); // 10 (1 + 2 + 3 + 4)
```

32. Viết hàm tính thời gian từ số giây

Bài tập: Viết một hàm nhận vào số giây và trả về chuỗi thời gian dạng HH:MM:SS.

Gợi ý:

```
function formatTime(seconds) {  
    const hours = Math.floor(seconds / 3600);  
    const minutes = Math.floor((seconds % 3600) / 60);  
    const secs = seconds % 60;  
    return `${hours.toString().padStart(2, "0")}:${minutes  
        .toString()  
        .padStart(2, "0")}:${secs.toString().padStart(2, "0")}`;  
}  
console.log(formatTime(3661)); // "01:01:01"
```

33. Viết hàm kiểm tra số Fibonacci

Bài tập: Viết hàm kiểm tra xem một số có phải là số Fibonacci hay không.

Gợi ý:

```
function isFibonacci(num) {  
    const isPerfectSquare = x => Math.sqrt(x) % 1 === 0;  
    return (  
        isPerfectSquare(5 * num ** 2 + 4) || isPerfectSquare(5 * num ** 2 - 4)  
    );  
}  
console.log(isFibonacci(8)); // true  
console.log(isFibonacci(10)); // false
```

arrow function

Thực hiện lại các bài tập từ 10 đến 25 theo kiểu arrow function

callback function

1. Callback đơn giản

Bài tập: Viết một hàm nhận vào một callback và gọi callback đó.

Gợi ý:

```
function executeCallback(callback) {  
    callback();  
}  
executeCallback(() => console.log("Callback executed!"));
```

2. Tính toán với callback

Bài tập: Viết một hàm thực hiện phép toán giữa hai số dựa trên callback.

Gợi ý:

```
function calculate(a, b, operation) {  
    return operation(a, b);  
}
```

```
console.log(calculate(5, 3, (x, y) => x + y)); // 8
console.log(calculate(5, 3, (x, y) => x * y)); // 15
```

3. Lọc mảng với callback

Bài tập: Viết hàm lọc mảng dựa trên điều kiện được cung cấp qua callback.

Gợi ý:

```
function filterArray(arr, condition) {
  return arr.filter(condition);
}
console.log(filterArray([1, 2, 3, 4], num => num % 2 === 0)); // [2, 4]
```

4. Sắp xếp mảng với callback

Bài tập: Viết hàm sắp xếp mảng dựa trên callback.

Gợi ý:

```
function sortArray(arr, compare) {
  return arr.sort(compare);
}
console.log(sortArray([3, 1, 4, 2], (a, b) => a - b)); // [1, 2, 3, 4]
```

5. Duyệt mảng với callback

Bài tập: Viết hàm duyệt qua từng phần tử của mảng và gọi callback.

Gợi ý:

```
function forEachElement(arr, callback) {
  arr.forEach(callback);
}
forEachElement([1, 2, 3], num => console.log(num)); // 1, 2, 3
```

6. Biến đổi mảng với callback

Bài tập: Viết hàm biến đổi mảng dựa trên callback.

Gợi ý:

```
function mapArray(arr, transform) {
  return arr.map(transform);
}
console.log(mapArray([1, 2, 3], num => num * 2)); // [2, 4, 6]
```

7. Callback trong hàm đệ quy

Bài tập: Viết hàm đệ quy nhận một callback để xử lý từng phần tử.

Gợi ý:

```
function recursiveProcess(arr, callback, index = 0) {
  if (index < arr.length) {
    callback(arr[index]);
```

```
        recursiveProcess(arr, callback, index + 1);
    }
}
recursiveProcess([1, 2, 3], num => console.log(num * 2)); // 2, 4, 6
```

8. Gọi callback khi điều kiện thỏa mãn

Bài tập: Viết hàm chỉ gọi callback khi một điều kiện được đáp ứng.

Gợi ý:

```
function conditionalCallback(value, condition, callback) {
    if (condition(value)) {
        callback(value);
    }
}
conditionalCallback(10, x => x > 5, x => console.log(`\$x} is greater than 5`)); // "10 is greater than 5"
```

9. Truyền nhiều callback

Bài tập: Viết hàm nhận vào hai callback và gọi chúng theo thứ tự.

Gợi ý:

```
function executeCallbacks(cb1, cb2) {
    cb1();
    cb2();
}
executeCallbacks(
    () => console.log("First callback"),
    () => console.log("Second callback")
);
```

10. Kiểm tra tất cả các phần tử với callback

Bài tập: Viết hàm kiểm tra xem tất cả phần tử trong mảng có thỏa mãn điều kiện không.

Gợi ý:

```
function allMatch(arr, condition) {
    return arr.every(condition);
}
console.log(allMatch([2, 4, 6], num => num % 2 === 0)); // true
console.log(allMatch([2, 3, 6], num => num % 2 === 0)); // false
```

11. Tìm phần tử đầu tiên với callback

Bài tập: Viết hàm tìm phần tử đầu tiên trong mảng thỏa mãn điều kiện qua callback.

Gợi ý:

```
function findElement(arr, condition) {
    return arr.find(condition);
}
console.log(findElement([1, 3, 5, 8], num => num % 2 === 0)); // 8
```

12. Thực hiện callback sau một khoảng thời gian

Bài tập: Viết hàm thực hiện callback sau 2 giây.

Gợi ý:

```
function delayedCallback(callback) {  
    setTimeout(callback, 2000);  
}  
delayedCallback(() => console.log("Executed after 2 seconds"));
```

13. Lồng nhiều callback

Bài tập: Viết hàm lồng callback để thực hiện các tác vụ theo thứ tự.

Gợi ý:

```
function nestedCallbacks(cb1, cb2) {  
    cb1(() => cb2());  
}  
nestedCallbacks(  
    next => {  
        console.log("First task");  
        next();  
    },  
    () => console.log("Second task")  
);
```

14. Callback với lỗi

Bài tập: Viết hàm nhận callback xử lý lỗi hoặc thành công.

Gợi ý:

```
function processValue(value, callback) {  
    if (value < 0) {  
        callback("Error: Value cannot be negative", null);  
    } else {  
        callback(null, value * 2);  
    }  
}  
processValue(-1, (err, result) => {  
    if (err) console.log(err);  
    else console.log(result);  
});
```

15. Tính toán bất đồng bộ

Bài tập: Viết hàm tính tổng các số trong mảng một cách bất đồng bộ qua callback.

Gợi ý:

```
function asyncSum(arr, callback) {  
    setTimeout(() => {  
        const sum = arr.reduce((a, b) => a + b, 0);  
        callback(sum);  
    }, 1000);
```

```
}
```

```
asyncSum([1, 2, 3], result => console.log(result)); // 6
```

16. Thực hiện callback tuần tự

Bài tập: Viết hàm thực hiện danh sách các callback theo thứ tự.

Gợi ý:

```
function sequentialCallbacks(callbacks) {
  callbacks.forEach(cb => cb());
}

sequentialCallbacks([
  () => console.log("Task 1"),
  () => console.log("Task 2"),
  () => console.log("Task 3")
]);
```

17. Thực hiện callback với mỗi phần tử

Bài tập: Viết hàm gọi callback cho từng phần tử trong mảng và in kết quả.

Gợi ý:

```
function processElements(arr, callback) {
  arr.forEach(el => console.log(callback(el)));
}

processElements([1, 2, 3], num => num * num); // 1, 4, 9
```

18. Gọi callback theo điều kiện

Bài tập: Viết hàm gọi callback nếu giá trị trong mảng lớn hơn 5.

Gợi ý:

```
function processIfGreaterThan(arr, threshold, callback) {
  arr.forEach(num => {
    if (num > threshold) callback(num);
  });
}

processIfGreaterThan([2, 7, 4, 9], 5, num => console.log(num)); // 7, 9
```

19. Callback kiểm tra mảng rỗng

Bài tập: Viết hàm kiểm tra mảng có rỗng hay không và gọi callback tương ứng.

Gợi ý:

```
function checkArray(arr, cbIsEmpty, cbNotEmpty) {
  if (arr.length === 0) {
    cbIsEmpty();
  } else {
    cbNotEmpty();
  }
}

checkArray([], () => console.log("Array is empty"), () => console.log("Array is not empty"));
```

20. Tạo callback tùy chỉnh

Bài tập: Viết một hàm tạo callback tùy chỉnh dựa trên điều kiện đầu vào.

Gợi ý:

```
function createCallback(condition) {  
  if (condition === "success") {  
    return () => console.log("Success callback executed");  
  } else {  
    return () => console.log("Failure callback executed");  
  }  
}  
const cb = createCallback("success");  
cb(); // "Success callback executed"
```

Object & Array

1. Tạo một Object đơn giản

Bài tập: Tạo một Object đại diện cho một người với các thuộc tính name, age, và job.

Gợi ý:

```
const person = {  
  name: "John",  
  age: 30,  
  job: "Developer"  
};  
console.log(person);
```

1.1. Truy cập thuộc tính Object: Truy cập và in ra giá trị của các thuộc tính name và age trong Object.

Gợi ý:

```
console.log(person.name); // "John"  
console.log(person["age"]); // 30
```

1.2. Thêm và xóa thuộc tính: Thêm thuộc tính address và xóa thuộc tính job trong Object.

Gợi ý:

```
person.address = "New York";  
delete person.job;  
console.log(person);
```

1.3. Lặp qua các thuộc tính của Object: Sử dụng vòng lặp for...in để in ra tất cả các thuộc tính và giá trị của Object.

Gợi ý:

```
for (let key in person) {  
  console.log(`$key: ${person[key]}`);  
}
```

1.4. Kiểm tra thuộc tính có tồn tại: Kiểm tra xem thuộc tính age có tồn tại trong Object không.

Gợi ý:

```
console.log("age" in person); // true
```

2. Tạo một mảng và thêm phần tử

Bài tập: Tạo một mảng chứa các số từ 1 đến 5 và thêm số 6 vào cuối mảng.

Gợi ý:

```
const numbers = [1, 2, 3, 4, 5];
numbers.push(6);
console.log(numbers);
```

2.1. Xóa phần tử cuối cùng trong mảng: Xóa phần tử cuối cùng trong mảng và in ra mảng.

Gợi ý:

```
numbers.pop();
console.log(numbers);
```

2.2. Truy cập phần tử mảng: Truy cập phần tử đầu tiên và phần tử cuối cùng trong mảng.

Gợi ý:

```
console.log(numbers[0]); // 1
console.log(numbers[numbers.length - 1]); // 5
```

2.3. Lặp qua mảng: Sử dụng vòng lặp for để in ra từng phần tử trong mảng.

Gợi ý:

```
for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
}
```

2.4. Lọc mảng: Viết hàm lọc ra các số chẵn từ mảng.

Gợi ý:

```
const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // [2, 4]
```

2.5 Biến đổi mảng: Tạo một mảng mới chứa bình phương của các phần tử trong mảng ban đầu.

Gợi ý:

```
const squaredNumbers = numbers.map(num => num ** 2);
console.log(squaredNumbers); // [1, 4, 9, 16, 25]
```

2.6. Tìm phần tử trong mảng: Tìm phần tử đầu tiên lớn hơn 3 trong mảng.

Gợi ý:

```
const found = numbers.find(num => num > 3);
console.log(found); // 4
```

2.7. Sắp xếp mảng: Sắp xếp mảng theo thứ tự giảm dần.

Gợi ý:

```
numbers.sort((a, b) => b - a);
console.log(numbers);
```

2.8. Tính tổng các phần tử: Tính tổng tất cả các phần tử trong mảng.

Gợi ý:

```
const sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum);
```

3. Lồng Object trong mảng

Tạo một mảng chứa các Object đại diện cho sinh viên với các thuộc tính name và score.

Gợi ý:

Tài liệu thực hành Chuyên đề ngôn ngữ lập trình

```
const students = [
  { name: "Alice", score: 85 },
  { name: "Bob", score: 92 },
  { name: "Charlie", score: 78 }
];
console.log(students);
```

3.1 Lọc sinh viên có điểm cao: Lọc ra các sinh viên có điểm lớn hơn 80.

Gợi ý:

```
const highScorers = students.filter(student => student.score > 80);
console.log(highScorers);
```

3.2. Thêm thuộc tính mới vào các Object trong mảng: Thêm thuộc tính passed (true/false) vào từng sinh viên dựa trên điểm số.

Gợi ý:

```
students.forEach(student => {
  student.passed = student.score >= 80;
});
console.log(students);
```

3.3. Nhóm sinh viên theo kết quả: Nhóm sinh viên thành hai nhóm passed và failed.

Gợi ý:

```
const grouped = students.reduce(
  (result, student) => {
    student.passed ? result.passed.push(student) :
    result.failed.push(student);
    return result;
  },
  { passed: [], failed: [] }
);
console.log(grouped);
```

3.4. Tạo Object từ mảng: Chuyển một mảng các cặp [key, value] thành Object.

Gợi ý:

```
const entries = [["name", "Alice"], ["age", 25], ["job", "Engineer"]];
const obj = Object.fromEntries(entries);
console.log(obj);
```

3.5. Tách Object thành mảng: Chuyển một Object thành mảng các cặp [key, value].

Gợi ý:

```
const object = { name: "Alice", age: 25, job: "Engineer" };
const pairs = Object.entries(object);
console.log(pairs);
```

Bài thực hành số 03

DOM

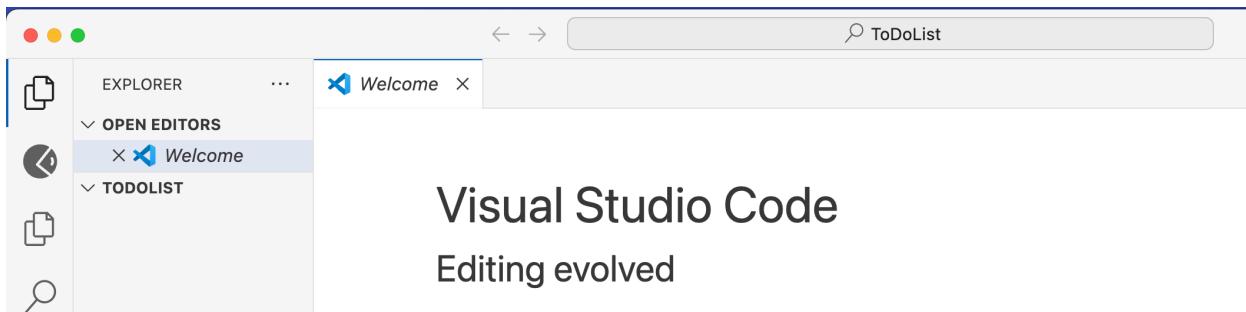
1. Quản lý danh sách công việc (To-Do List)

Mô tả:

Tạo một ứng dụng To-Do List đơn giản cho phép người dùng:

1. Thêm công việc vào danh sách.
2. Xóa công việc khỏi danh sách.
3. Đánh dấu công việc là đã hoàn thành.

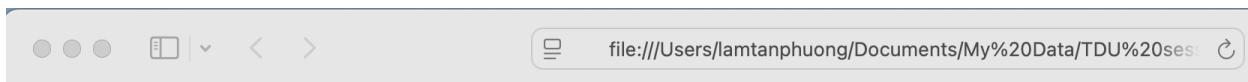
Hướng dẫn: Tạo một folder ToDoList trong visualCode như hình sau:



Bước 01: tạo index.html như hình sau:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>To-Do List</title>
<style>
body {
    font-family: Arial, sans-serif;
}
.completed {
    text-decoration: line-through;
    color: gray;
}
</style>
</head>
<body>
<h1>To-Do List</h1>
<input type="text" id="taskInput" placeholder="Enter a new task">
<button id="addTaskButton">Add Task</button>
<ul id="taskList"></ul>
<script src="script.js"></script>
</body>
</html>
```

kết quả bước 01:



To-Do List

Enter a new task Add Task

Bước 02: Truy cập và thao tác DOM

Tạo file script.js để thêm logic JavaScript.

Hướng dẫn thực hiện từng bước:

S1: Truy cập các phần tử DOM:

Sử dụng document.getElementById hoặc document.querySelector để truy cập các phần tử input, button, và ul.

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there are two expanded sections: 'OPEN EDITOR...' containing 'Welcome' (HTML), 'index.html' (HTML), and 'script.js' (JavaScript); and 'TODOLIST' containing 'index.html' and 'script.js'. The 'script.js' file is currently selected and open in the main editor area. The code in the editor is as follows:

```
1 // Truy cập các phần tử
2 const taskInput = document.getElementById("taskInput");
3 const addTaskButton = document.getElementById("addTaskButton");
4 const taskList = document.getElementById("taskList");
5 
```

S2: Thêm sự kiện để thêm công việc mới:

Tiếp tục với file script.js

Lắng nghe sự kiện click trên nút "Add Task".

Khi người dùng nhấn nút, tạo một thẻ mới, thêm nội dung và chèn vào danh sách.

```
1 // Truy cập các phần tử
2 const taskInput = document.getElementById("taskInput");
3 const addTaskButton = document.getElementById("addTaskButton");
4 const taskList = document.getElementById("taskList");
5
6 // Xử lý sự kiện thêm công việc
7 addTaskButton.addEventListener("click", function () {
8     const taskText = taskInput.value.trim(); // Lấy giá trị từ ô input
9     if (taskText === "") {
10         alert("Please enter a task!");
11         return;
12     }
13
14     // Tạo một thẻ <li> mới
15     const listItem = document.createElement("li");
16     listItem.textContent = taskText;
17
18     // Thêm vào danh sách
19     taskList.appendChild(listItem);
20
21     // Xóa nội dung trong ô input
22     taskInput.value = "";
23 });

S3: Thêm chức năng xóa công việc:
```

Tiếp tục với file script.js

Mỗi thẻ có thể được nhấp để xóa công việc.

```
24
25     // Xóa công việc khi nhấp vào
26     taskList.addEventListener("click", function (event) {
27         const clickedItem = event.target; // Xác định mục được nhấp
28         if (clickedItem.tagName === "LI") {
29             taskList.removeChild(clickedItem);
30         }
31     });
32 
```

S4: Thêm chức năng đánh dấu hoàn thành:

Tiếp tục với file script.js

Khi nhấn đúp chuột vào một mục, thay đổi trạng thái "hoàn thành" bằng cách thêm/lấy lớp CSS.

```
32 // Đánh dấu công việc là hoàn thành khi nhấn đúp chuột
33 taskList.addEventListener("dblclick", function (event) {
34     const clickedItem = event.target;
35     if (clickedItem.tagName === "LI") {
36         clickedItem.classList.toggle("completed"); // Thêm/xóa lớp "completed"
37     }
38 });
39
40
```

Tính năng hoàn chỉnh:

Nhập công việc vào ô input và nhấn nút "Add Task" để thêm công việc.

Nhấp vào một mục để xóa công việc.

Nhấn đúp vào một mục để đánh dấu hoặc bỏ đánh dấu hoàn thành.

Yêu cầu tự thực hiện:

Thêm nút "X" bên cạnh mỗi công việc để xóa trực tiếp.

Lưu danh sách công việc vào localStorage để không mất dữ liệu khi tải lại trang.

Thêm tính năng chỉnh sửa nội dung công việc.

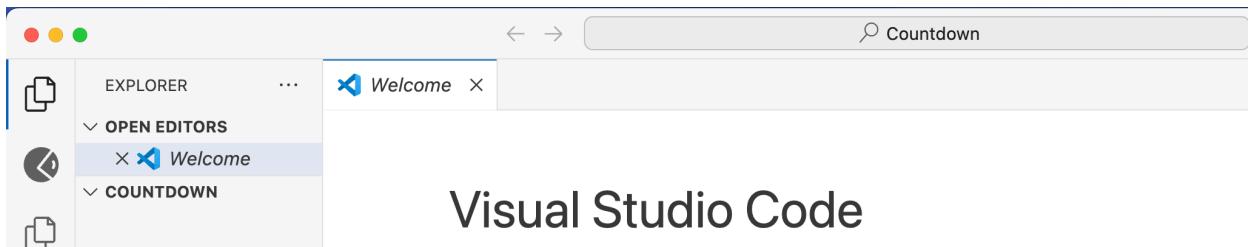
2. Tạo đồng hồ đếm ngược (Countdown Timer)

Mô tả:

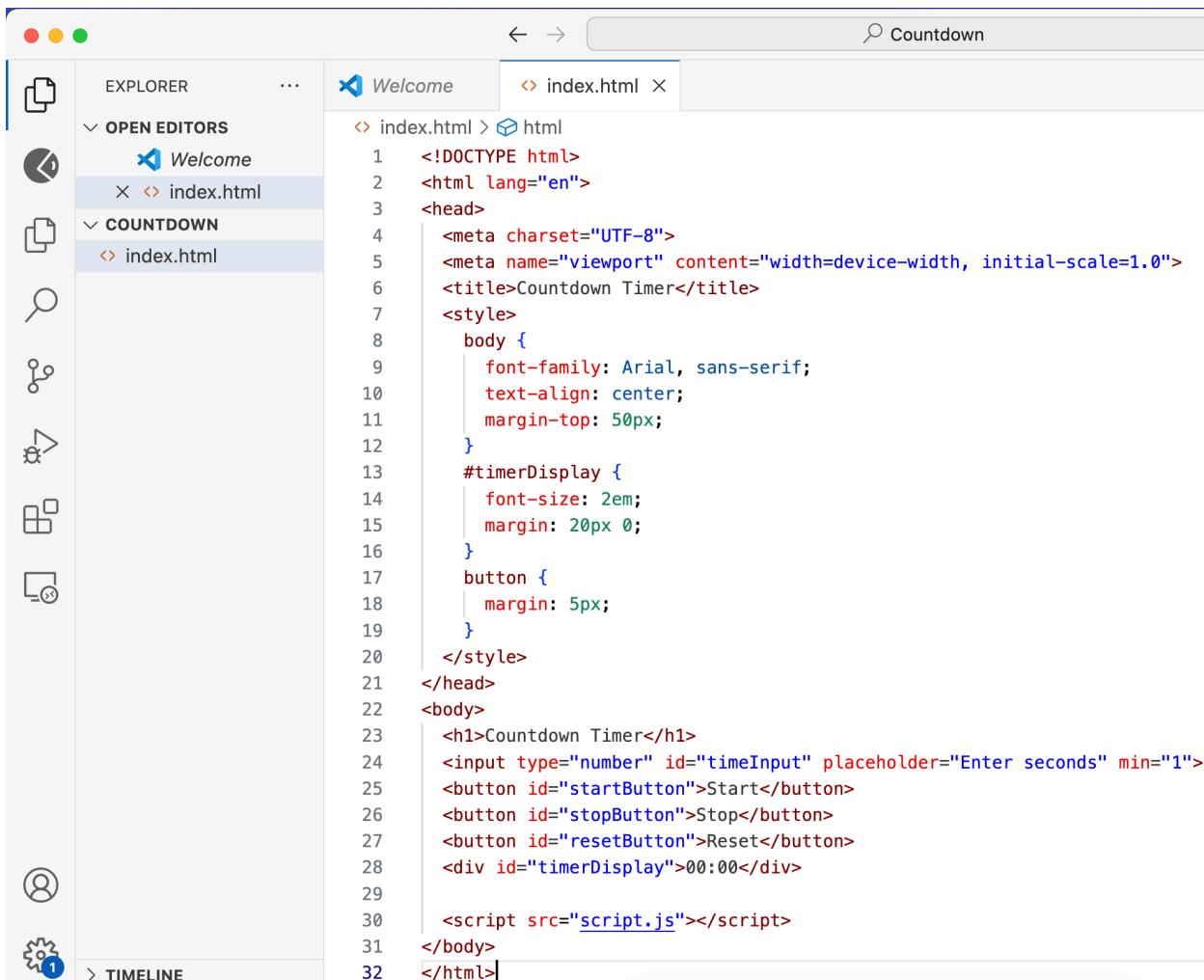
Tạo một ứng dụng đồng hồ đếm ngược cho phép người dùng:

1. Nhập số giây muốn đếm ngược.
2. Nhấn nút "Start" để bắt đầu đếm ngược.
3. Hiển thị thời gian còn lại.
4. Dừng hoặc đặt lại đồng hồ khi cần.

Hướng dẫn: Tạo folder Countdown trong visualCode như hình:

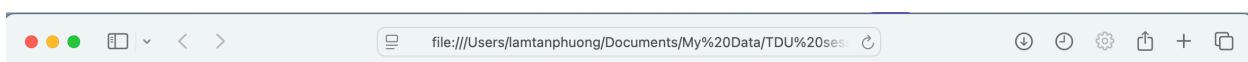


Bước 01: tạo index.html như hình sau:



The screenshot shows the Microsoft Edge browser window. The title bar says "Countdown". The address bar shows the URL "file:///Users/lamtanphuong/Documents/My%20Data/TDU%20ses...". The main content area displays a web page titled "Countdown Timer". The page has a heading "Countdown Timer", an input field for "Enter seconds", and three buttons: "Start", "Stop", and "Reset". Below the input field is a div with the ID "timerDisplay" containing the text "00:00". At the bottom of the page is a script tag pointing to "script.js". On the left side of the browser, there is a sidebar with various icons and a "TIMELINE" section.

kết quả bước 01:



Countdown Timer

Enter seconds

00:00

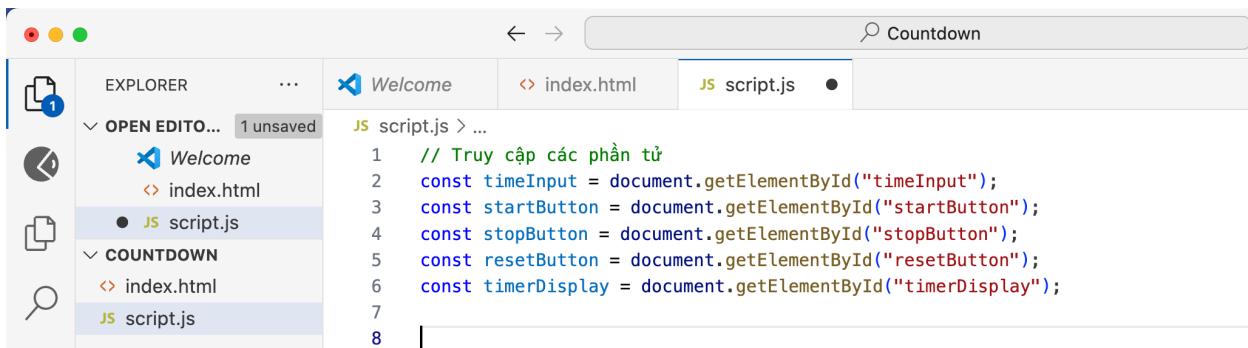
Bước 02: Truy cập và thao tác DOM

Tạo file script.js để thêm logic JavaScript.

Hướng dẫn thực hiện từng bước:

S1: Truy cập các phần tử DOM:

Sử dụng document.getElementById để truy cập các phần tử input, button, và div.



The screenshot shows a code editor interface with the following details:

- Explorer Bar:** Shows a folder structure with "OPEN EDITOR..." containing "Welcome" and "index.html", and a "COUNTDOWN" folder containing "index.html" and "script.js".
- Search Bar:** Displays the text "Countdown".
- File Script.js:** Contains the following JavaScript code:

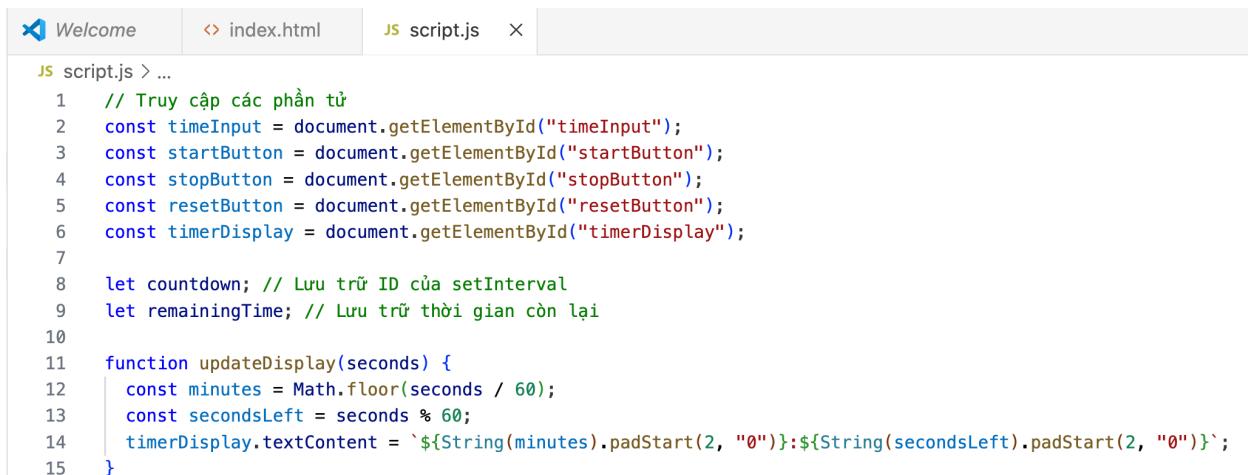
```
1 // Truy cập các phần tử
2 const timeInput = document.getElementById("timeInput");
3 const startButton = document.getElementById("startButton");
4 const stopButton = document.getElementById("stopButton");
5 const resetButton = document.getElementById("resetButton");
6 const timerDisplay = document.getElementById("timerDisplay");
```

S2: Xử lý logic đếm ngược:

Tiếp tục với file script.js

Sử dụng setInterval để giảm thời gian mỗi giây.

Hiển thị thời gian còn lại theo định dạng MM:SS.



The screenshot shows the continuation of the script.js file with the following code:

```
1 // Truy cập các phần tử
2 const timeInput = document.getElementById("timeInput");
3 const startButton = document.getElementById("startButton");
4 const stopButton = document.getElementById("stopButton");
5 const resetButton = document.getElementById("resetButton");
6 const timerDisplay = document.getElementById("timerDisplay");
7
8 let countdown; // Lưu trữ ID của setInterval
9 let remainingTime; // Lưu trữ thời gian còn lại
10
11 function updateDisplay(seconds) {
12     const minutes = Math.floor(seconds / 60);
13     const secondsLeft = seconds % 60;
14     timerDisplay.textContent = `${String(minutes).padStart(2, "0")}:${String(secondsLeft).padStart(2, "0")}`;
15 }
```

S3: Thêm sự kiện cho nút "Start":

Tiếp tục với file script.js

Lấy số giây từ input và bắt đầu đếm ngược.

Vô hiệu hóa nút "Start" khi đồng hồ đang chạy.

```
16
17  startButton.addEventListener("click", function () {
18      const inputTime = parseInt(timeInput.value, 10);
19
20      if (isNaN(inputTime) || inputTime <= 0) {
21          alert("Please enter a valid number of seconds.");
22          return;
23      }
24
25      remainingTime = inputTime;
26      updateDisplay(remainingTime);
27
28      startButton.disabled = true; // Vô hiệu hóa nút Start
29
30      countdown = setInterval(function () {
31          remainingTime--;
32          updateDisplay(remainingTime);
33
34          if (remainingTime <= 0) {
35              clearInterval(countdown);
36              alert("Time's up!");
37              startButton.disabled = false; // Kích hoạt lại nút Start
38          }
39      }, 1000);
40  });

S4: Thêm sự kiện cho nút "Stop":
```

Tiếp tục với file script.js

Dừng đồng hồ đếm ngược khi nhấn nút.

```
41
42  stopButton.addEventListener("click", function () {
43      clearInterval(countdown);
44      startButton.disabled = false; // Kích hoạt lại nút Start
45  });

S5: Thêm sự kiện cho nút "Reset":
```

Tiếp tục với file script.js

Dừng đồng hồ và đặt lại hiển thị.

```
46
47     resetButton.addEventListener("click", function () {
48         clearInterval(countdown);
49         remainingTime = 0;
50         updateDisplay(remainingTime);
51         startButton.disabled = false; // Kích hoạt lại nút Start
52         timeInput.value = ""; // Xóa nội dung input
53     });

```

Tính năng hoàn chỉnh:

Nhập số giây vào ô input và nhấn nút "Start" để bắt đầu đếm ngược.

Nhấn "Stop" để tạm dừng đếm ngược.

Nhấn "Reset" để đặt lại đồng hồ.

Yêu cầu tự thực hiện:

Thêm thông báo âm thanh khi đồng hồ kết thúc.

Thêm giao diện hiển thị màu sắc thay đổi khi thời gian gần hết (ví dụ: đổi màu đỏ khi còn 10 giây).

Lưu thời gian còn lại vào localStorage để tiếp tục đếm ngược khi tải lại trang.

Lập trình bất đồng bộ (Asynchronous)

1. Sử dụng setTimeout

Bài tập: In ra dòng chữ "Hello, Async!" sau 2 giây.

Gợi ý:

```
setTimeout(() => {
    console.log("Hello, Async!");
}, 2000);
```

2. Sử dụng setInterval

Bài tập: In ra số đếm từ 1 đến 5, mỗi giây một lần.

Gợi ý:

```
let count = 1;
const interval = setInterval(() => {
    console.log(count);
    if (count === 5) clearInterval(interval);
    count++;
}, 1000);
```

3. Giả lập bất đồng bộ với callback

Bài tập: Tạo hàm fakeAsync nhận callback và thực thi sau 3 giây.

Gợi ý:

```
function fakeAsync(callback) {
    setTimeout(() => {
        callback("Done!");
    }, 3000);
}
```

```
fakeAsync(result => console.log(result)); // "Done!"
```

4. Callback Hell

Bài tập: Tạo 3 hàm bất đồng bộ gọi nhau tuần tự bằng callback.

Gợi ý:

```
setTimeout(() => {
  console.log("Task 1");
  setTimeout(() => {
    console.log("Task 2");
    setTimeout(() => {
      console.log("Task 3");
    }, 1000);
  }, 1000);
}, 1000);
```

5. Sử dụng Promise

Bài tập: Tạo một Promise trả về kết quả sau 2 giây.

Gợi ý:

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => resolve("Success!"), 2000);
});
myPromise.then(result => console.log(result));
```

6. Xử lý lỗi trong Promise

Bài tập: Tạo một Promise bị từ chối và xử lý lỗi.

Gợi ý:

```
const errorPromise = new Promise((resolve, reject) => {
  setTimeout(() => reject("Error occurred!"), 2000);
});
errorPromise
  .then(result => console.log(result))
  .catch(error => console.log(error));
```

7. Chuỗi Promise

Bài tập: Kết nối 3 Promise liên tiếp.

Gợi ý:

```
new Promise(resolve => {
  setTimeout(() => resolve("Task 1"), 1000);
})
  .then(result => {
    console.log(result);
    return new Promise(resolve => setTimeout(() => resolve("Task 2"), 1000));
  })
  .then(result => {
    console.log(result);
    return new Promise(resolve => setTimeout(() => resolve("Task 3"), 1000));
  })
  .then(result => console.log(result));
```

8. Sử dụng async/await

Bài tập: Viết hàm bất đồng bộ sử dụng async/await để chờ kết quả của một Promise.

Gợi ý:

```
async function asyncFunction() {
    const result = await new Promise(resolve => setTimeout(() =>
    resolve("Done!"), 2000));
    console.log(result);
}
asyncFunction();
```

9. Xử lý lỗi trong async/await

Bài tập: Viết hàm bắt đồng bộ xử lý lỗi bằng try...catch.

Gợi ý:

```
async function asyncError() {
    try {
        const result = await new Promise(_, reject) => setTimeout(() =>
        reject("Error!"), 2000));
        console.log(result);
    } catch (error) {
        console.log(error);
    }
}
asyncError();
```

10. Sử dụng Promise.all

Bài tập: Chạy song song 3 Promise và in ra kết quả khi tất cả hoàn thành.

Gợi ý:

```
const promise1 = new Promise(resolve => setTimeout(() => resolve("One"),
1000));
const promise2 = new Promise(resolve => setTimeout(() => resolve("Two"),
2000));
const promise3 = new Promise(resolve => setTimeout(() => resolve("Three"),
3000));

Promise.all([promise1, promise2, promise3]).then(results =>
console.log(results));
```

11. Sử dụng Promise.race

Bài tập: Chạy song song 3 Promise và in ra kết quả của Promise hoàn thành đầu tiên.

Gợi ý:

```
Promise.race([promise1, promise2, promise3]).then(result =>
console.log(result));
```

12. Chờ nhiều await trong hàm

Bài tập: Viết hàm sử dụng nhiều lần await.

Gợi ý:

```
async function multipleAwait() {
    const first = await new Promise(resolve => setTimeout(() =>
    resolve("First"), 1000));
    console.log(first);
    const second = await new Promise(resolve => setTimeout(() =>
    resolve("Second"), 2000));
    console.log(second);
}
multipleAwait();
```

13. Sử dụng setTimeout với Promise

Bài tập: Tạo hàm trả về Promise giả lập thời gian chờ.

Gợi ý:

```
function delay(ms) {  
    return new Promise(resolve => setTimeout(resolve, ms));  
}  
delay(2000).then(() => console.log("2 seconds later"));
```

14. Tải dữ liệu giả lập

Bài tập: Giả lập tải dữ liệu bằng Promise.

Gợi ý:

```
function fetchData() {  
    return new Promise(resolve => setTimeout(() => resolve("Data fetched!"),  
3000));  
}  
fetchData().then(data => console.log(data));
```

15. Bất đồng bộ với fetch

Bài tập: Sử dụng fetch để lấy dữ liệu từ một API.

Gợi ý:

```
fetch("https://jsonplaceholder.typicode.com/posts/1")  
.then(response => response.json())  
.then(data => console.log(data));
```

16. Sử dụng async/await với fetch

Bài tập: Lấy dữ liệu từ API bằng async/await.

Gợi ý:

```
async function getData() {  
    const response = await  
fetch("https://jsonplaceholder.typicode.com/posts/1");  
    const data = await response.json();  
    console.log(data);  
}  
getData();
```

17. Xử lý lỗi API

Bài tập: Xử lý lỗi khi gọi API.

Gợi ý:

```
async function getDataWithError() {  
    try {  
        const response = await  
fetch("https://jsonplaceholder.typicode.com/invalid-url");  
        if (!response.ok) throw new Error("API Error");  
        const data = await response.json();  
        console.log(data);  
    } catch (error) {  
        console.log(error.message);  
    }  
}  
getDataWithError();
```

18. Thực hiện tuần tự với Promise

Bài tập: Thực hiện 3 tác vụ tuần tự bằng Promise.

Gợi ý:

```
function task(message, delay) {
    return new Promise(resolve => setTimeout(() => resolve(message), delay));
}

task("Task 1", 1000)
    .then(result => {
        console.log(result);
        return task("Task 2", 2000);
    })
    .then(result => {
        console.log(result);
        return task("Task 3", 1000);
    })
    .then(result => console.log(result));
```

19. Giả lập API đa tầng

Bài tập: Gọi 2 API giả lập, API thứ 2 phụ thuộc kết quả của API thứ 1.

Gợi ý:

```
function fetchUser() {
    return new Promise(resolve => setTimeout(() => resolve({ id: 1, name: "John" }), 2000));
}

function fetchPosts(userId) {
    return new Promise(resolve => setTimeout(() => resolve(["Post1", "Post2"])), 2000));
}

fetchUser()
    .then(user => {
        console.log(user);
        return fetchPosts(user.id);
    })
    .then(posts => console.log(posts));
```

20. Giả lập tiến trình tải

Bài tập: Tạo hàm giả lập tiến trình tải với Promise.

Gợi ý:

```
function loadProgress() {
    return new Promise(resolve => {
        let progress = 0;
        const interval = setInterval(() => {
            progress += 20;
            console.log(`Loading: ${progress}%`);
            if (progress === 100) {
                clearInterval(interval);
                resolve("Load complete!");
            }
        }, 500);
    });
}
loadProgress().then(message => console.log(message));
```

Bài thực hành số 04

Destructuring

1. Destructuring với Object lồng nhau

Bài tập: Cho Object sau, sử dụng destructuring để lấy ra giá trị city và zipcode.

```
const person = {  
    name: "Alice",  
    address: {  
        city: "New York",  
        zipcode: "10001",  
        country: "USA"  
    }  
};
```

Gợi ý:

Có thể truy cập trực tiếp thuộc tính của Object lồng nhau bằng cú pháp destructuring.

```
const { address: { city, zipcode } } = person;  
console.log(city); // "New York"  
console.log(zipcode); // "10001"
```

2. Đổi tên biến khi destructuring

Bài tập: Cho Object sau, sử dụng destructuring để lấy giá trị name và age, nhưng đặt tên biến là fullName và yearsOld.

```
const user = {  
    name: "Bob",  
    age: 25  
};
```

Gợi ý:

Sử dụng cú pháp : newVariableName để đổi tên biến khi destructuring.

```
const { name: fullName, age: yearsOld } = user;  
console.log(fullName); // "Bob"  
console.log(yearsOld); // 25
```

3. Destructuring với giá trị mặc định

Bài tập: Cho Object sau, sử dụng destructuring để lấy name và country. Nếu country không tồn tại, gán giá trị mặc định là "Unknown".

```
const person = {  
    name: "Charlie"  
};
```

Gợi ý:

Sử dụng cú pháp key = defaultValue để gán giá trị mặc định.

```
const { name, country = "Unknown" } = person;  
console.log(name); // "Charlie"  
console.log(country); // "Unknown"
```

4. Destructuring với Array

Bài tập: Cho mảng sau, sử dụng destructuring để lấy giá trị đầu tiên, giá trị cuối cùng và các giá trị còn lại.

```
const numbers = [10, 20, 30, 40, 50];
```

Gợi ý:

Dùng cú pháp ...rest để lấy phần còn lại của mảng.

```
const [first, , , last] = numbers;
console.log(first); // 10
console.log(last); // 50

const [firstNum, ...rest] = numbers;
console.log(rest); // [20, 30, 40, 50]
```

5. Kết hợp Destructuring Object và Array

Bài tập: Cho mảng sau, mỗi phần tử là một Object, sử dụng destructuring để lấy name của sinh viên đầu tiên và điểm của sinh viên thứ hai.

```
const students = [
  { name: "Alice", score: 85 },
  { name: "Bob", score: 92 },
  { name: "Charlie", score: 78 }
];
```

Gợi ý:

Bạn có thể kết hợp destructuring Object và Array trong một câu lệnh.

```
const [ { name: firstName }, { score: secondScore } ] = students;
console.log(firstName); // "Alice"
console.log(secondScore); // 92
```

Spread Operator

1. Sao chép và thêm phần tử vào mảng

Bài tập: Cho mảng sau, sử dụng Spread Operator để sao chép mảng và thêm phần tử mới vào cuối mảng mà không làm thay đổi mảng gốc.

```
const originalArray = [1, 2, 3];
```

Gợi ý:

Spread Operator (...) có thể được dùng để sao chép mảng.

```
const newArray = [...originalArray, 4, 5];
console.log(originalArray); // [1, 2, 3]
console.log(newArray); // [1, 2, 3, 4, 5]
```

2. Gộp nhiều mảng thành một

Bài tập: Cho hai mảng sau, sử dụng Spread Operator để gộp chúng thành một mảng duy nhất.

```
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
```

Gợi ý:

Spread Operator giúp gộp nhiều mảng mà không cần dùng hàm concat().

```
const mergedArray = [...array1, ...array2];
```

```
console.log(mergedArray); // [1, 2, 3, 4, 5, 6]
```

3. Sao chép và chỉnh sửa Object

Bài tập: Cho Object sau, sử dụng Spread Operator để sao chép Object và cập nhật giá trị age thành 30.

```
const person = {  
    name: "Alice",  
    age: 25,  
    city: "New York"  
};
```

Gợi ý:

Spread Operator có thể sao chép và chỉnh sửa Object mà không làm thay đổi Object gốc.

```
const updatedPerson = { ...person, age: 30 };  
console.log(person); // { name: "Alice", age: 25, city: "New York" }  
console.log(updatedPerson); // { name: "Alice", age: 30, city: "New York" }
```

4. Thêm thuộc tính mới vào Object

Bài tập: Cho Object sau, sử dụng Spread Operator để thêm thuộc tính country với giá trị "USA" mà không thay đổi Object gốc.

```
const user = {  
    username: "john_doe",  
    email: "john@example.com"  
};
```

Gợi ý:

Spread Operator cho phép thêm thuộc tính mới vào Object.

```
const updatedUser = { ...user, country: "USA" };  
console.log(user); // { username: "john_doe", email:  
"john@example.com" }  
console.log(updatedUser); // { username: "john_doe", email:  
"john@example.com", country: "USA" }
```

5. Kết hợp Spread Operator với destructuring

Bài tập: Cho Object sau, sử dụng Spread Operator và destructuring để tách thuộc tính name và gộp các thuộc tính còn lại vào một Object khác.

```
const student = {  
    name: "Bob",  
    age: 20,  
    grade: "A",  
    major: "Computer Science"  
};
```

Gợi ý:

Sử dụng destructuring để tách một số thuộc tính, và Spread Operator để gộp các thuộc tính còn lại.

```
const { name, ...details } = student;  
console.log(name); // "Bob"  
console.log(details); // { age: 20, grade: "A", major: "Computer Science" }
```

Rest

1. Thu thập các tham số còn lại trong hàm

Bài tập: Viết một hàm sum nhận nhiều tham số và trả về tổng của chúng.

Gợi ý:

Sử dụng Rest Parameter (...args) để gom các tham số còn lại thành một mảng.

```
function sum(...numbers) {  
    return numbers.reduce((total, num) => total + num, 0);  
}  
console.log(sum(1, 2, 3, 4)); // 10  
console.log(sum(5, 10)); // 15
```

2. Phân tách tham số đầu tiên và các tham số còn lại

Bài tập: Viết một hàm getFirstAndRest để in tham số đầu tiên và mảng chứa các tham số còn lại.

Gợi ý:

Dùng destructuring để lấy phần tử đầu tiên và phần còn lại.

```
function getFirstAndRest(first, ...rest) {  
    console.log("First:", first);  
    console.log("Rest:", rest);  
}  
getFirstAndRest(10, 20, 30, 40);  
// First: 10  
// Rest: [20, 30, 40]
```

3. Rest với destructuring Object

Bài tập: Cho Object sau, sử dụng Rest để lấy thuộc tính name và gộp các thuộc tính còn lại vào một Object khác.

```
const user = {  
    name: "Alice",  
    age: 25,  
    city: "New York",  
    country: "USA"  
};
```

Gợi ý:

Sử dụng Rest Operator trong destructuring Object.

```
const { name, ...otherDetails } = user;  
console.log(name); // "Alice"  
console.log(otherDetails); // { age: 25, city: "New York", country: "USA" }
```

4. Rest với destructuring Array

Bài tập: Cho mảng sau, sử dụng Rest để lấy hai phần tử đầu tiên và gộp các phần tử còn lại vào một mảng khác.

```
const numbers = [10, 20, 30, 40, 50];
```

Gợi ý:

Sử dụng Rest Operator trong destructuring Array.

```
const [first, second, ...rest] = numbers;  
console.log(first); // 10  
console.log(second); // 20
```

```
console.log(rest); // [30, 40, 50]
```

5. Kết hợp Rest Parameter với các tham số khác trong hàm

Bài tập: Viết một hàm greet nhận tham số đầu tiên là greeting, tham số thứ hai là name, và các tham số còn lại là extras để hiển thị thông điệp đầy đủ.

Gợi ý:

Đặt Rest Parameter sau các tham số bắt buộc.

```
function greet(greeting, name, ...extras) {
  console.log(` ${greeting}, ${name}! `);
  if (extras.length > 0) {
    console.log("Extras:", extras.join(", "));
  }
}
greet("Hello", "Alice", "Have a great day!", "See you soon!");
// Hello, Alice!
// Extras: Have a great day!, See you soon!
```

Classes

1. Tạo lớp cơ bản và sử dụng

Bài tập: Tạo một lớp Rectangle có các thuộc tính width và height. Viết phương thức getArea để trả về diện tích hình chữ nhật.

Gợi ý:

Sử dụng từ khóa class để định nghĩa lớp.

```
class Rectangle {
  constructor(width, height) {
    this.width = width;
    this.height = height;
  }

  getArea() {
    return this.width * this.height;
  }
}

const rect = new Rectangle(10, 5);
console.log(rect.getArea()); // 50
```

2. Kế thừa lớp

Bài tập: Tạo lớp Square kế thừa từ lớp Rectangle, và đảm bảo rằng Square chỉ cần một thuộc tính side.

Gợi ý:

Sử dụng từ khóa extends để kế thừa. Gọi phương thức super() trong constructor.

```
class Square extends Rectangle {
  constructor(side) {
    super(side, side);
  }
}

const square = new Square(4);
```

```
console.log(square.getArea()); // 16
```

3. Sử dụng phương thức tĩnh (static)

Bài tập: Thêm một phương thức tĩnh compareAreas vào lớp Rectangle để so sánh diện tích của hai hình chữ nhật.

Gợi ý:

Dùng từ khóa static để định nghĩa phương thức tĩnh.

```
class Rectangle {  
    constructor(width, height) {  
        this.width = width;  
        this.height = height;  
    }  
  
    getArea() {  
        return this.width * this.height;  
    }  
  
    static compareAreas(rect1, rect2) {  
        return rect1.getArea() - rect2.getArea();  
    }  
}  
  
const rect1 = new Rectangle(10, 5);  
const rect2 = new Rectangle(6, 6);  
console.log(Rectangle.compareAreas(rect1, rect2)); // 14
```

4. Tạo thuộc tính và phương thức riêng tư

Bài tập: Tạo lớp BankAccount với thuộc tính riêng tư _balance. Thêm các phương thức deposit và withdraw để cập nhật số dư và getBalance để lấy số dư hiện tại.

Gợi ý:

Sử dụng ký hiệu # để định nghĩa thuộc tính và phương thức riêng tư.

```
class BankAccount {  
    #balance;  
  
    constructor(initialBalance) {  
        this.#balance = initialBalance;  
    }  
  
    deposit(amount) {  
        if (amount > 0) this.#balance += amount;  
    }  
  
    withdraw(amount) {  
        if (amount > 0 && amount <= this.#balance) this.#balance -= amount;  
    }  
  
    getBalance() {  
        return this.#balance;  
    }  
}  
  
const account = new BankAccount(1000);  
account.deposit(500);  
account.withdraw(300);
```

```
console.log(account.getBalance()); // 1200
```

5. Overriding phương thức

Bài tập: Tạo lớp Employee với phương thức getDetails trả về tên và tuổi của nhân viên. Tạo lớp Manager kế thừa từ Employee và override phương thức getDetails để thêm thông tin về vai trò.

Gợi ý:

Sử dụng phương thức super() để gọi phương thức của lớp cha khi cần.

```
class Employee {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    getDetails() {  
        return `Name: ${this.name}, Age: ${this.age}`;  
    }  
}  
  
class Manager extends Employee {  
    constructor(name, age, role) {  
        super(name, age);  
        this.role = role;  
    }  
  
    getDetails() {  
        return `${super.getDetails()}, Role: ${this.role}`;  
    }  
}  
  
const manager = new Manager("Alice", 30, "Team Lead");  
console.log(manager.getDetails());  
// Name: Alice, Age: 30, Role: Team Lead
```

6. Sử dụng getter và setter

Bài tập: Tạo lớp Product với thuộc tính price. Thêm getter và setter cho price để đảm bảo giá trị không âm.

Gợi ý:

Dùng từ khóa get và set để định nghĩa getter và setter.

```
class Product {  
    #price;  
  
    constructor(price) {  
        this.#price = price > 0 ? price : 0;  
    }  
  
    get price() {  
        return this.#price;  
    }  
  
    set price(value) {  
        if (value > 0) this.#price = value;  
    }  
}
```

```
const product = new Product(100);
product.price = 150; // cập nhật giá
console.log(product.price); // 150
product.price = -50; // không cập nhật giá vì âm
console.log(product.price); // 150
```

Closure

1. Bộ đếm (Counter)

Bài tập: Viết một hàm createCounter trả về một hàm khác. Hàm này sẽ tăng giá trị đếm mỗi lần được gọi.

Gợi ý:

Closure cho phép một hàm "ghi nhớ" biến trong phạm vi của nó, ngay cả sau khi hàm cha đã thực thi xong.

```
function createCounter() {
  let count = 0; // Biến được giữ bởi closure
  return function () {
    count++;
    return count;
  };
}

const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2
console.log(counter()); // 3
```

2. Bộ nhớ cache cho hàm

Bài tập: Viết một hàm memoize nhận vào một hàm fn và trả về một phiên bản mới của fn với khả năng ghi nhớ kết quả cho các đầu vào đã tính trước đó.

Gợi ý:

Sử dụng closure để lưu trữ kết quả trong một object.

```
function memoize(fn) {
  const cache = {}; // Bộ nhớ cache
  return function (...args) {
    const key = JSON.stringify(args);
    if (key in cache) {
      console.log("Fetching from cache:", key);
      return cache[key];
    }
    const result = fn(...args);
    cache[key] = result;
    console.log("Calculating result:", key);
    return result;
  };
}

const add = (a, b) => a + b;
const memoizedAdd = memoize(add);

console.log(memoizedAdd(2, 3)); // Calculating result: [2,3] => 5
console.log(memoizedAdd(2, 3)); // Fetching from cache: [2,3] => 5
```

3. Tạo hàm với tham số cố định

Bài tập: Viết một hàm createMultiplier nhận một số x và trả về một hàm nhân số đó với một số khác.

Gợi ý:

Closure giữ phép "nhớ" giá trị của tham số ban đầu.

```
function createMultiplier(x) {
  return function (y) {
    return x * y;
  };
}

const multiplyBy2 = createMultiplier(2);
console.log(multiplyBy2(5)); // 10
console.log(multiplyBy2(10)); // 20

const multiplyBy3 = createMultiplier(3);
console.log(multiplyBy3(5)); // 15
```

4. Quản lý trạng thái với Closure

Bài tập: Viết một hàm createState nhận vào một giá trị ban đầu và trả về hai hàm: getState để lấy giá trị hiện tại và setState để cập nhật giá trị.

Gợi ý:

Closure giúp giữ trạng thái và cung cấp các phương thức để truy cập hoặc thay đổi nó.

```
function createState(initialValue) {
  let state = initialValue; // Biến được lưu trữ trong closure
  return {
    getState: function () {
      return state;
    },
    setState: function (newValue) {
      state = newValue;
    }
  };
}

const stateManager = createState(10);
console.log(stateManager.getState()); // 10
stateManager.setState(20);
console.log(stateManager.getState()); // 20
```

5. Tạo một hàm đếm thời gian chờ

Bài tập: Viết một hàm createDelay nhận vào một khoảng thời gian delay (ms) và trả về một hàm khác, hàm này sẽ in ra thông báo sau khoảng thời gian chờ đã định.

Gợi ý:

Closure giữ giá trị delay và sử dụng nó trong hàm được trả về.

```
function createDelay(delay) {
  return function (message) {
    setTimeout(() => {
      console.log(message);
    }, delay);
  };
}
```

```
}
```

```
const delay1000 = createDelay(1000);
delay1000("This message appears after 1 second.");

const delay2000 = createDelay(2000);
delay2000("This message appears after 2 seconds.");
```

Higher-order Functions

1. Lọc danh sách với filter

Bài tập: Viết một hàm filterByLength nhận vào một mảng các chuỗi và một số length. Hàm trả về một mảng chỉ chứa các chuỗi có độ dài lớn hơn length.

Gợi ý:

filter là một Higher-order Function nhận vào một hàm callback để xác định các phần tử cần giữ lại.

```
function filterByLength(strings, length) {
  return strings.filter(str => str.length > length);
}

const words = ["apple", "banana", "kiwi", "cherry", "fig"];
console.log(filterByLength(words, 4)); // ["apple", "banana", "cherry"]
```

2. Biến đổi mảng với map

Bài tập: Viết một hàm capitalizeWords nhận vào một mảng các chuỗi và trả về một mảng mới với mỗi chuỗi được viết hoa chữ cái đầu.

Gợi ý:

map là một Higher-order Function dùng để biến đổi từng phần tử trong mảng.

```
function capitalizeWords(words) {
  return words.map(word => word.charAt(0).toUpperCase() + word.slice(1));
}

const words = ["apple", "banana", "cherry"];
console.log(capitalizeWords(words)); // ["Apple", "Banana", "Cherry"]
```

3. Tổng hợp giá trị với reduce

Bài tập: Viết một hàm sumEvenNumbers nhận vào một mảng số và trả về tổng các số chẵn trong mảng.

Gợi ý:

reduce là một Higher-order Function để tổng hợp giá trị từ mảng.

```
function sumEvenNumbers(numbers) {
  return numbers.reduce((sum, num) => num % 2 === 0 ? sum + num : sum, 0);
}

const numbers = [1, 2, 3, 4, 5, 6];
console.log(sumEvenNumbers(numbers)); // 12
```

4. Kết hợp các Higher-order Functions

Bài tập: Viết một hàm processNumbers nhận vào một mảng số. Hàm lọc các số chẵn, nhân đôi giá trị của chúng, và trả về tổng các số đã xử lý.

Gợi ý:

Kết hợp filter, map, và reduce để xử lý mảng.

```
function processNumbers(numbers) {  
    return numbers  
        .filter(num => num % 2 === 0) // Lọc số chẵn  
        .map(num => num * 2) // Nhân đôi giá trị  
        .reduce((sum, num) => sum + num, 0); // Tính tổng  
}  
  
const numbers = [1, 2, 3, 4, 5, 6];  
console.log(processNumbers(numbers)); // 24
```

5. Tự viết Higher-order Function

Bài tập: Viết một hàm Higher-order Function applyOperation nhận vào một mảng số và một hàm operation. Hàm applyOperation sẽ áp dụng operation lên từng phần tử của mảng và trả về mảng mới.

Gợi ý:

Một Higher-order Function nhận vào hàm khác như tham số.

```
function applyOperation(numbers, operation) {  
    return numbers.map(operation);  
}  
  
const numbers = [1, 2, 3, 4, 5];  
const square = num => num * num; // Hàm tính bình phương  
const double = num => num * 2; // Hàm nhân đôi  
  
console.log(applyOperation(numbers, square)); // [1, 4, 9, 16, 25]  
console.log(applyOperation(numbers, double));
```

Currying

1. Tạo hàm Currying cơ bản

Bài tập: Viết một hàm add có thể được gọi dưới dạng add(2)(3) để trả về tổng của 2 và 3.

Gợi ý:

Sử dụng closure để tạo hàm Currying.

```
function add(a) {  
    return function (b) {  
        return a + b;  
    };  
}  
  
console.log(add(2)(3)); // 5  
console.log(add(5)(10)); // 15
```

2. Chuyển hàm nhiều tham số thành Currying

Bài tập: Viết một hàm curry để chuyển một hàm nhận nhiều tham số thành dạng Currying.

Gợi ý:

Trả về một hàm mới nhận từng tham số một.

```
function curry(fn) {  
    return function curried(...args) {  
        if (args.length >= fn.length) {  
            return fn(...args);  
        }  
        return function (...nextArgs) {  
            return curried(...args, ...nextArgs);  
        };  
    };  
}  
  
function multiply(a, b, c) {  
    return a * b * c;  
}  
  
const curriedMultiply = curry(multiply);  
console.log(curriedMultiply(2)(3)(4)); // 24  
console.log(curriedMultiply(2, 3)(4)); // 24
```

3. Tạo hàm Currying linh hoạt

Bài tập: Viết một hàm currySum để tính tổng của bất kỳ số lượng tham số nào, được gọi dưới dạng Currying.

Gợi ý:

Sử dụng closure và hàm tự gọi để tiếp tục thêm tham số.

```
function currySum(a) {  
    return function (b) {  
        if (b !== undefined) {  
            return currySum(a + b);  
        }  
        return a;  
    };  
}  
  
console.log(currySum(1)(2)(3)()); // 6  
console.log(currySum(10)(20)(30)(40)()); // 100
```

4. Sử dụng Currying trong hàm lọc

Bài tập: Viết một hàm filterWith sử dụng Currying để tạo hàm lọc dựa trên điều kiện.

Gợi ý:

Currying giúp tạo ra các hàm lọc có thể tái sử dụng.

```
function filterWith(predicate) {  
    return function (array) {  
        return array.filter(predicate);  
    };  
}  
  
const isEven = num => num % 2 === 0;  
const filterEven = filterWith(isEven);  
  
console.log(filterEven([1, 2, 3, 4, 5])); // [2, 4]
```

5. Currying cho hàm so sánh

Bài tập: Viết một hàm compare sử dụng Currying để so sánh hai giá trị với toán tử tùy chọn ($>$, $<$, $==$).

Gợi ý:

Sử dụng Currying để cấu hình toán tử trước khi thực hiện so sánh.

```
function compare(operator) {  
    return function (a) {  
        return function (b) {  
            switch (operator) {  
                case ">":  
                    return a > b;  
                case "<":  
                    return a < b;  
                case "===":  
                    return a === b;  
                default:  
                    throw new Error("Invalid operator");  
            }  
        };  
    };  
}  
  
const greaterThan = compare(">");  
console.log(greaterThan(5)(3)); // true  
  
const lessThan = compare("<");  
console.log(lessThan(5)(10)); // true  
  
const equals = compare("==");  
console.log(equals(5)(5)); // true
```

Partial Application

1. Tạo hàm Partial Application cơ bản

Bài tập: Viết một hàm **partial** nhận vào một hàm và một số tham số ban đầu. Trả về một hàm mới nhận các tham số còn lại. (lưu ý hàm này sẽ được sử dụng lại cho các bài tập 2, 3, 4, 5)

Gợi ý:

Sử dụng closure để lưu các tham số ban đầu và kết hợp với tham số sau này.

```
function partial(fn, ...args) {  
    return function (...newArgs) {  
        return fn(...args, ...newArgs);  
    };  
}  
  
function multiply(a, b, c) {  
    return a * b * c;  
}  
  
const partialMultiply = partial(multiply, 2);  
console.log(partialMultiply(3, 4)); // 24
```

2. Partial Application với hàm in chuỗi

Bài tập: Viết một hàm partialGreet nhận một hàm greet và một tham số greeting. Trả về một hàm nhận tên và in ra lời chào đầy đủ.

Gợi ý:

Dùng partial để "cố định" tham số greeting.

```
function greet(greeting, name) {  
    return `${greeting}, ${name}!`;  
}  
  
const partialGreet = partial(greet, "Hello");  
console.log(partialGreet("Alice")); // "Hello, Alice!"  
console.log(partialGreet("Bob")); // "Hello, Bob!"
```

3. Partial Application trong hàm xử lý mảng

Bài tập: Viết một hàm partialFilter sử dụng Partial Application để tạo ra hàm lọc dựa trên điều kiện cố định.

Gợi ý:

Dùng partial để "đóng gói" điều kiện lọc.

```
function filterArray(array, predicate) {  
    return array.filter(predicate);  
}  
  
const isEven = num => num % 2 === 0;  
const partialFilter = partial(filterArray, undefined, isEven);  
  
console.log(partialFilter([1, 2, 3, 4, 5])); // [2, 4]
```

4. Partial Application với toán tử

Bài tập: Viết một hàm partialCompare để tạo các hàm so sánh giá trị cố định.

Gợi ý:

Partial Application giúp tạo các hàm so sánh như "lớn hơn 10", "nhỏ hơn 5",...

```
function compare(a, b) {  
    return a > b;  
}  
  
const greaterThan10 = partial(compare, 10);  
console.log(greaterThan10(5)); // false  
console.log(greaterThan10(15)); // true
```

5. Tính tổng động với Partial Application

Bài tập: Viết một hàm partialSum nhận vào một số tham số cố định và trả về một hàm nhận các tham số còn lại để tính tổng.

Gợi ý:

Partial Application giữ các giá trị cố định và bổ sung giá trị mới khi cần.

```
function sum(...numbers) {  
    return numbers.reduce((total, num) => total + num, 0);  
}  
  
const partialSum = partial(sum, 5, 10);  
console.log(partialSum(15, 20)); // 50  
console.log(partialSum(30, 40)); // 85
```


Bài thực hành số 05

JSON

1. Parse JSON

Bài tập: Parse chuỗi JSON `{"name": "Alice", "age": 25}` thành một object JavaScript.

Gợi ý:

Sử dụng `JSON.parse()` để chuyển đổi từ chuỗi JSON thành object.

```
const jsonString = '{"name": "Alice", "age": 25}';  
const parsedObject = JSON.parse(jsonString);  
console.log(parsedObject); // { name: "Alice", age: 25 }
```

2. Stringify Object

Bài tập: Chuyển object `{name: "Bob", age: 30}` thành chuỗi JSON.

Gợi ý:

Sử dụng `JSON.stringify()` để chuyển đổi từ object thành chuỗi JSON.

```
const obj = { name: "Bob", age: 30 };  
const jsonString = JSON.stringify(obj);  
console.log(jsonString); // '{"name": "Bob", "age": 30}'
```

3. Trích xuất giá trị từ JSON

Bài tập: Lấy giá trị của key "name" từ chuỗi JSON `{"name": "Charlie", "age": 35}`.

Gợi ý:

Parse chuỗi JSON trước, sau đó truy cập key.

```
const jsonString = '{"name": "Charlie", "age": 35}';  
const obj = JSON.parse(jsonString);  
console.log(obj.name); // "Charlie"
```

4. Thêm thuộc tính vào JSON

Bài tập: Thêm key "country": "USA" vào object JSON `{name: "Alice", age: 25}`.

Gợi ý:

Parse JSON, chỉnh sửa object, sau đó stringify lại nếu cần.

```
let jsonString = '{"name": "Alice", "age": 25}';  
let obj = JSON.parse(jsonString);  
obj.country = "USA";  
jsonString = JSON.stringify(obj);  
console.log(jsonString); // '{"name": "Alice", "age": 25, "country": "USA"}'
```

5. Lặp qua các key của JSON

Bài tập: In ra tất cả các key trong object JSON `{name: "Alice", age: 25}`.

Gợi ý:

Dùng vòng lặp `for...in` để duyệt qua các key.

```
const obj = { name: "Alice", age: 25 };  
for (const key in obj) {  
    console.log(key); // "name", "age"  
}
```

Fetch API

1. Gửi GET Request

Bài tập: Gửi GET request đến <https://jsonplaceholder.typicode.com/posts/1> và in kết quả ra console.

Gợi ý:

Sử dụng fetch để gửi yêu cầu.

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

2. Gửi POST Request

Bài tập: Gửi POST request đến <https://jsonplaceholder.typicode.com/posts> với body {title: "foo", body: "bar", userId: 1}.

Gợi ý:

Sử dụng fetch với method "POST".

```
fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ title: "foo", body: "bar", userId: 1 })
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

3. Xử lý lỗi API

Bài tập: Thêm xử lý lỗi khi gọi GET request đến một API không tồn tại.

Gợi ý:

Kiểm tra response.ok trước khi parse JSON.

```
fetch('https://jsonplaceholder.typicode.com/nonexistent')
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

4. Xử lý trạng thái tải

Bài tập: Hiển thị thông báo "Loading..." trong khi chờ kết quả từ API.

Gợi ý:

Sử dụng một biến để theo dõi trạng thái tải.

```
console.log("Loading...");
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => response.json())
```

```
.then(data => {
  console.log(data);
  console.log("Finished loading.");
})
.catch(error => console.error('Error:', error));
```

5. Xóa dữ liệu qua API

Bài tập: Gửi DELETE request đến <https://jsonplaceholder.typicode.com/posts/1>.

Gợi ý:

Sử dụng method "DELETE" trong fetch.

```
fetch('https://jsonplaceholder.typicode.com/posts/1', {
  method: 'DELETE'
})
.then(response => {
  if (response.ok) {
    console.log("Deleted successfully.");
  } else {
    throw new Error("Failed to delete.");
  }
})
.catch(error => console.error('Error:', error));
```

Kết hợp API và JSON

1. Kết hợp nhiều API

Bài tập: Gửi hai GET request đến hai API khác nhau và kết hợp kết quả.

Gợi ý:

Sử dụng Promise.all để xử lý song song.

```
Promise.all([
  fetch('https://jsonplaceholder.typicode.com/posts/1').then(res =>
res.json()),
  fetch('https://jsonplaceholder.typicode.com/users/1').then(res =>
res.json())
])
.then(([post, user]) => {
  console.log('Post:', post);
  console.log('User:', user);
})
.catch(error => console.error('Error:', error));
```

2. Lưu dữ liệu API vào localStorage

Bài tập: Gửi GET request đến API và lưu kết quả vào localStorage.

Gợi ý:

Dùng localStorage.setItem để lưu dữ liệu dưới dạng JSON.

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
.then(response => response.json())
.then(data => {
  localStorage.setItem('post', JSON.stringify(data));
  console.log('Data saved to localStorage.');
})
.catch(error => console.error('Error:', error));
```

3. Xử lý dữ liệu API với map

Tài liệu thực hành Chuyên đề ngôn ngữ lập trình

Bài tập: Gửi GET request đến <https://jsonplaceholder.typicode.com/posts> và chỉ in ra title của mỗi bài viết.

Gợi ý:

Sử dụng map để xử lý mảng dữ liệu.

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json())
  .then(posts => {
    const titles = posts.map(post => post.title);
    console.log(titles);
  })
  .catch(error => console.error('Error:', error));
```

4. Tìm kiếm dữ liệu API

Bài tập: Gửi GET request đến API và tìm bài viết có id = 5.

Gợi ý:

Sử dụng find để tìm kiếm trong mảng.

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json())
  .then(posts => {
    const post = posts.find(post => post.id === 5);
    console.log(post);
  })
  .catch(error => console.error('Error:', error));
```

5. Hiển thị dữ liệu API lên giao diện

Bài tập: Gửi GET request đến API và hiển thị danh sách bài viết trên trang HTML.

Gợi ý:

Tạo phần tử HTML và thêm nội dung từ API.

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json())
  .then(posts => {
    const container = document.getElementById('posts');
    posts.forEach(post => {
      const postElement = document.createElement('div');
      postElement.textContent = post.title;
      container.appendChild(postElement);
    });
  })
  .catch(error => console.error('Error:', error));
```

Axios

1. Gửi một GET Request cơ bản

Bài tập: Sử dụng Axios để gửi một GET request tới API sau và in kết quả:
API: <https://jsonplaceholder.typicode.com/posts/1>

Gợi ý:

Sử dụng phương thức axios.get.

```
const axios = require('axios');
```

```
axios.get('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error:', error.message);
  });
});
```

2. Gửi một POST Request

Bài tập: Gửi một POST request với dữ liệu:

```
{
  "title": "foo",
  "body": "bar",
  "userId": 1
}
```

API: <https://jsonplaceholder.typicode.com/posts>

Gợi ý:

Sử dụng axios.post với dữ liệu JSON.

```
const axios = require('axios');

axios.post('https://jsonplaceholder.typicode.com/posts', {
  title: 'foo',
  body: 'bar',
  userId: 1
})
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error:', error.message);
  });
});
```

3. Thêm Headers vào Request

Bài tập: Gửi một GET request kèm headers:

Headers:

```
{
  "Authorization": "Bearer example_token"
}
```

Gợi ý:

Sử dụng đối số thứ hai của axios.get để thêm headers.

```
const axios = require('axios');

axios.get('https://jsonplaceholder.typicode.com/posts/1', {
  headers: {
    Authorization: 'Bearer example_token'
  }
})
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error:', error.message);
  });
});
```

```
});
```

4. Xử lý nhiều request cùng lúc

Bài tập: Gửi hai GET request đồng thời tới:

<https://jsonplaceholder.typicode.com/posts/1>

<https://jsonplaceholder.typicode.com/posts/2>

Gợi ý:

Sử dụng axios.all hoặc Promise.all.

```
const axios = require('axios');

Promise.all([
  axios.get('https://jsonplaceholder.typicode.com/posts/1'),
  axios.get('https://jsonplaceholder.typicode.com/posts/2')
])
  .then(responses => {
    console.log('Post 1:', responses[0].data);
    console.log('Post 2:', responses[1].data);
  })
  .catch(error => {
    console.error('Error:', error.message);
  });
});
```

5. Sử dụng Axios với Async/Await

Bài tập: Viết lại bài tập 1 sử dụng async/await.

Gợi ý:

Định nghĩa một hàm async và sử dụng await.

```
const axios = require('axios');

async function fetchData() {
  try {
    const response = await
    axios.get('https://jsonplaceholder.typicode.com/posts/1');
    console.log(response.data);
  } catch (error) {
    console.error('Error:', error.message);
  }
}

fetchData();
```

6. Xử lý lỗi với Axios

Bài tập: Thử gửi request tới một URL sai (<https://jsonplaceholder.typicode.com/posts/invalid>) và xử lý lỗi trả về.

Gợi ý:

Sử dụng .catch hoặc try...catch với error.response.

```
const axios = require('axios');

axios.get('https://jsonplaceholder.typicode.com/posts/invalid')
```

```
.then(response => {
  console.log(response.data);
})
.catch(error => {
  if (error.response) {
    console.error('Error Status:', error.response.status);
    console.error('Error Data:', error.response.data);
  } else {
    console.error('Error:', error.message);
  }
});
```

7. Cấu hình Axios với Base URL

Bài tập: Thiết lập axios với baseURL để tránh lặp lại URL gốc.

Gợi ý:

Sử dụng axios.create.

```
const axios = require('axios');

const api = axios.create({
  baseURL: 'https://jsonplaceholder.typicode.com'
});

api.get('/posts/1')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error:', error.message);
  });
});
```

8. Gửi một PUT Request

Bài tập: Gửi một PUT request để cập nhật bài viết:

API: <https://jsonplaceholder.typicode.com/posts/1>

Dữ liệu:

```
{
  "id": 1,
  "title": "updated title",
  "body": "updated body",
  "userId": 1
}
```

Gợi ý:

Sử dụng axios.put.

```
const axios = require('axios');

axios.put('https://jsonplaceholder.typicode.com/posts/1', {
  id: 1,
  title: 'updated title',
  body: 'updated body',
  userId: 1
})
  .then(response => {
    console.log(response.data);
  })
});
```

```
.catch(error => {
  console.error('Error:', error.message);
});
```

9. Hủy một Request

Bài tập: Hủy một request đang chạy bằng CancelToken.

Gợi ý:

Sử dụng axios.CancelToken.

```
const axios = require('axios');

const CancelToken = axios.CancelToken;
const source = CancelToken.source();

axios.get('https://jsonplaceholder.typicode.com/posts/1', {
  cancelToken: source.token
})
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    if (axios.isCancel(error)) {
      console.error('Request canceled:', error.message);
    } else {
      console.error('Error:', error.message);
    }
  });
}

source.cancel('Operation canceled by the user.');
```

10. Sử dụng Interceptors

Bài tập: Thêm interceptor để log thông tin trước và sau khi gửi request.

Gợi ý:

Sử dụng axios.interceptors.

```
const axios = require('axios');

// Add a request interceptor
axios.interceptors.request.use(config => {
  console.log('Request Sent:', config);
  return config;
}, error => {
  return Promise.reject(error);
});

// Add a response interceptor
axios.interceptors.response.use(response => {
  console.log('Response Received:', response);
  return response;
}, error => {
  return Promise.reject(error);
});

axios.get('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    console.log('Data:', response.data);
```

```
})
.catch(error => {
  console.error('Error:', error.message);
});
```

Bài thực hành số 06

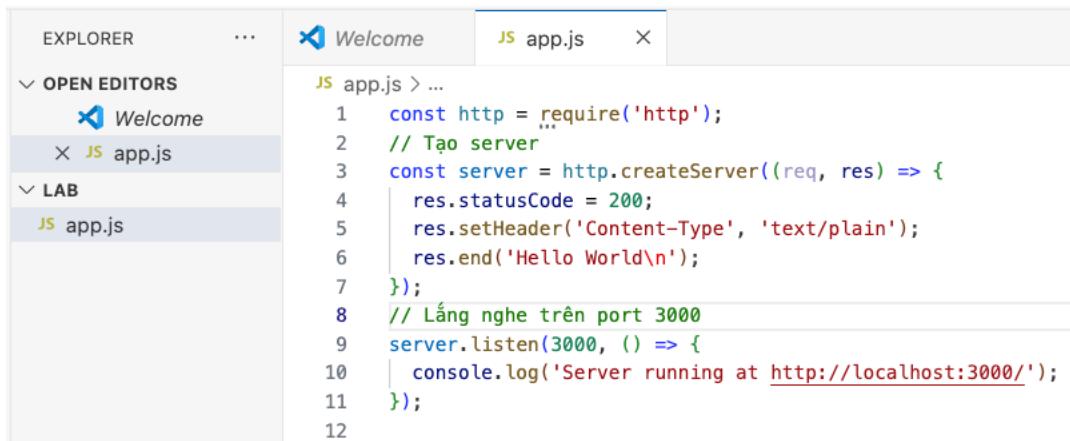
Node.js

Sử dụng Node.js

Bước 1: Tạo file Node.js đầu tiên

Mở trình soạn thảo code (như VS Code)

- Tạo một thư mục có tên lab và add vào VS code
- Tạo một file mới với tên app.js.
- Trong file app.js, viết đoạn code sau để tạo một server đơn giản:



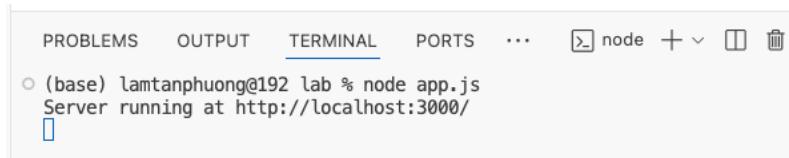
```
EXPLORER ... JS Welcome JS app.js X
OPEN EDITORS
  Welcome
  JS app.js
LAB
  JS app.js

JS app.js > ...
1 const http = require('http');
2 // Tạo server
3 const server = http.createServer((req, res) => {
4   res.statusCode = 200;
5   res.setHeader('Content-Type', 'text/plain');
6   res.end('Hello World\n');
7 });
8 // Lắng nghe trên port 3000
9 server.listen(3000, () => {
10   console.log('Server running at http://localhost:3000/');
11 });
12
```

Bước 2: Chạy file Node.js

Mở terminal và điều hướng đến thư mục chứa file app.js, sau đó chạy lệnh:

Nếu mọi thứ hoạt động tốt, Chúng ta sẽ thấy thông báo:



```
PROBLEMS OUTPUT TERMINAL PORTS ... node + v
(base) lamtanphuong@192 lab % node app.js
Server running at http://localhost:3000/
```

Bước 3: Truy cập ứng dụng

Mở trình duyệt và truy cập <http://localhost:3000/>. Chúng ta sẽ thấy dòng chữ Hello World trên trình duyệt.



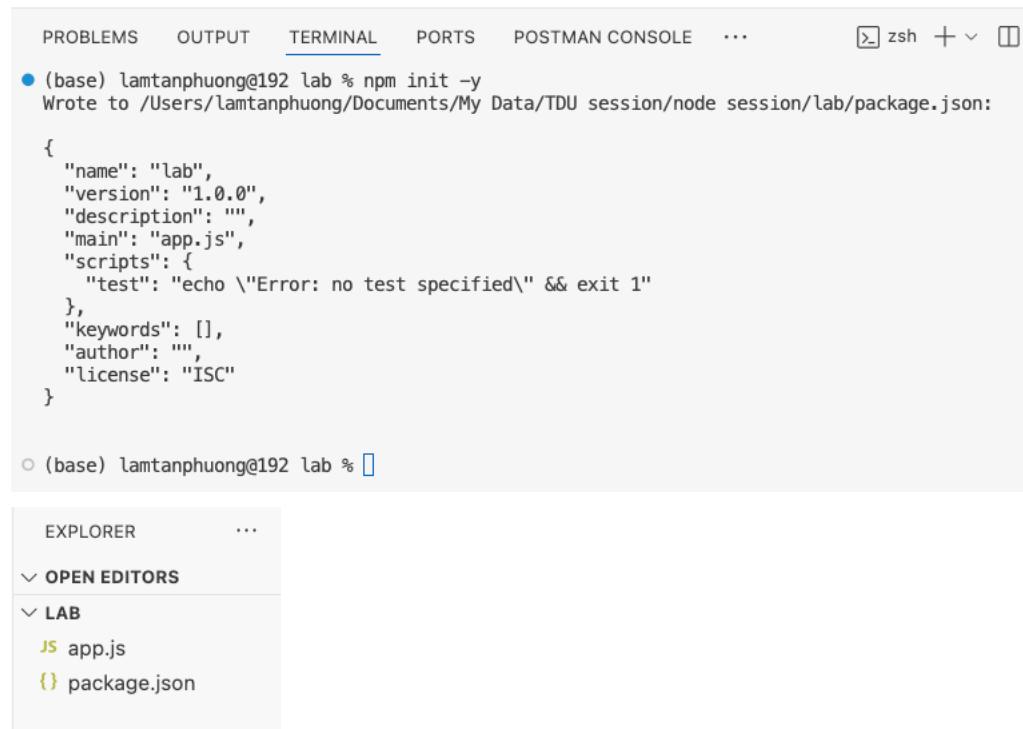
3. Quản lý các package với npm

Node.js đi kèm với npm (Node Package Manager), giúp Chúng ta cài đặt các package (thư viện) để mở rộng tính năng cho ứng dụng.

Sử dụng thư mục lab cho dự án

Bước 1: Khởi tạo dự án với npm

Chạy lệnh “npm init -y” để khởi tạo dự án và tạo file package.json:



The screenshot shows the VS Code interface. The terminal tab is active, displaying the command "npm init -y" being run in a terminal window titled "zsh". The output shows the creation of a package.json file with basic configuration. Below the terminal, the Explorer sidebar shows a new "LAB" folder containing "app.js" and "package.json".

```
(base) lamtanphuong@192 lab % npm init -y
Wrote to /Users/lamtanphuong/Documents/My Data/TDU session/node session/lab/package.json:

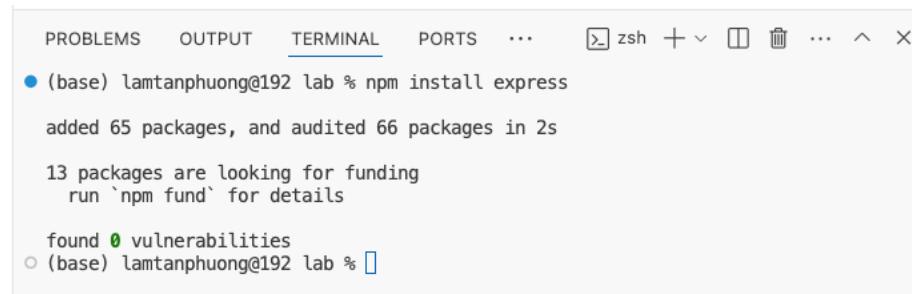
{
  "name": "lab",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

○ (base) lamtanphuong@192 lab %
```

```
EXPLORER      ...
▽ OPEN EDITORS
▽ LAB
  JS app.js
  {} package.json
```

Bước 2: Cài đặt package

Ví dụ, cài đặt thư viện Express để tạo ứng dụng web dễ dàng hơn:



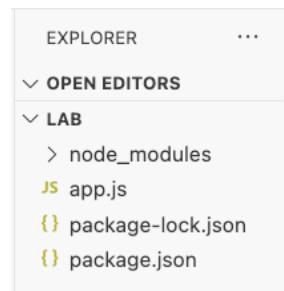
The screenshot shows the VS Code interface with the terminal tab active. The command "npm install express" is run, which installs the Express package and its dependencies. The terminal shows the progress and audit results.

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  ...
● (base) lamtanphuong@192 lab % npm install express
added 65 packages, and audited 66 packages in 2s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
○ (base) lamtanphuong@192 lab %
```

Thao tác này sẽ cài đặt express và tạo thư mục node_modules chứa tất cả các package của dự án.



The screenshot shows the VS Code interface with the Explorer tab active. It displays the file structure of the "LAB" folder, which now includes a "node_modules" directory along with "app.js", "package-lock.json", and "package.json".

```
EXPLORER      ...
▽ OPEN EDITORS
▽ LAB
  > node_modules
  JS app.js
  {} package-lock.json
  {} package.json
```

Bước 3: Sử dụng package

Sau khi cài đặt express, Chúng ta có thể sử dụng nó trong ứng dụng của mình.

Tài liệu thực hành Chuyên đề ngôn ngữ lập trình

Tạo file server.js với nội dung sau:

The screenshot shows the VS Code interface. On the left, the Explorer sidebar lists files: 'OPEN EDITORS' contains 'server.js'; 'LAB' contains 'node_modules', 'app.js', 'package-lock.json', 'package.json', and 'server.js'. The 'server.js' file in the LAB folder is selected and shown in the main editor area. The code in 'server.js' is:

```
1 const express = require('express');
2 const app = express();
3 // Route chính
4 app.get('/', (req, res) => {
5 | res.send('Hello from Express!');
6 });
7 // Khởi động server
8 app.listen(3000, () => {
9 | console.log('Server is running on port 3000');
10});
```

Chạy lệnh node:

The screenshot shows the VS Code terminal tab. It displays the command 'node server.js' being run and its output: 'Server is running on port 3000'.

```
PROBLEMS OUTPUT TERMINAL PORTS ...
○ (base) lamtanphuong@192 lab % node server.js
Server is running on port 3000
```

Mở trình duyệt và truy cập <http://localhost:3000/> để xem kết quả.

Để chạy bằng lệnh npm, chúng ta mở file package.json và thay đổi giá trị của "scripts" như sau:

The screenshot shows the VS Code editor with 'package.json' open. The 'scripts' section is highlighted, and the 'start' key is set to 'node server.js'. The rest of the package.json content is as follows:

```
{
  "name": "lab",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.21.0"
  }
}
```

Bây giờ chúng ta có thể chạy dự án bằng lệnh npm:

The screenshot shows the VS Code terminal tab. It displays the command 'npm start' being run and its output: 'Server is running on port 3000'.

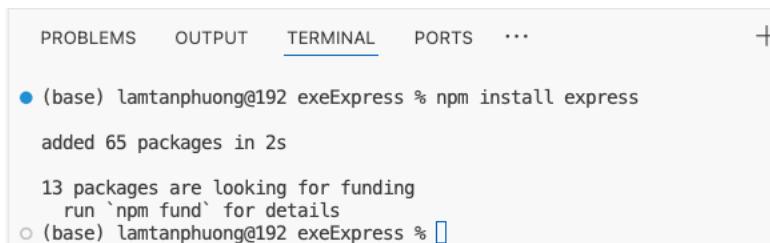
```
PROBLEMS OUTPUT TERMINAL PORTS ...
✖ (base) lamtanphuong@192 lab % node server.js
Server is running on port 3000
^C
○ (base) lamtanphuong@192 lab % npm start
> lab@1.0.0 start
> node server.js

Server is running on port 3000
```

Express module

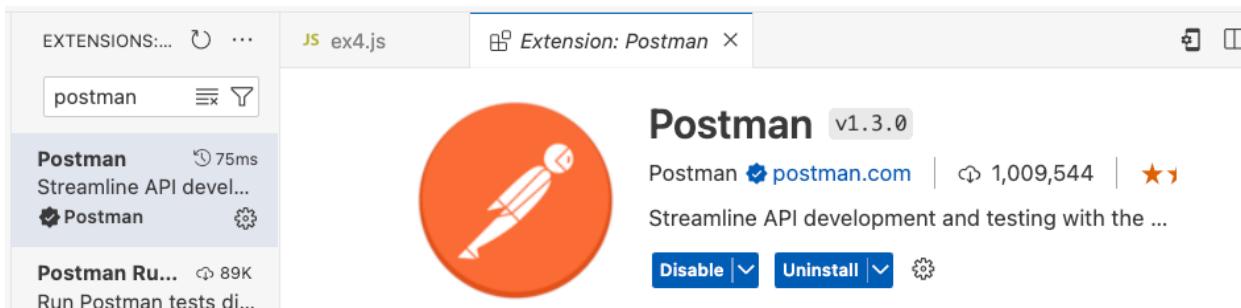
Thực hiện tạo một thư mục có tên exeExpress và add vào VS Code để thực hiện các bài tập sau:

Cài module Express: npm install express



```
PROBLEMS OUTPUT TERMINAL PORTS ... +  
● (base) lamtanphuong@192 exeExpress % npm install express  
added 65 packages in 2s  
13 packages are looking for funding  
  run `npm fund` for details  
○ (base) lamtanphuong@192 exeExpress %
```

Cài đặt thêm extension Postman cho visual code:



1. Bài tập 1: Tạo server đơn giản với Express

Mô tả: Tạo một ứng dụng Express đơn giản để trả về "Hello World!" khi người dùng truy cập vào trang chủ.

Hướng dẫn:



```
JS ex1.js ●  
exeExpress > JS ex1.js > ...  
1  const express = require('express');  
2  const app = express();  
3  const port = 3000;  
4  
5  app.get('/', (req, res) => {  
6    |  res.send('Hello World!');  
7  });  
8  
9  app.listen(port, () => {  
10   |  console.log(`Server is running on http://localhost:${port}`);  
11});
```

Yêu cầu khảo sát:

- Tạo một ứng dụng Express.
- Lắng nghe trên cổng 3000 và trả về "Hello World!" khi người dùng truy cập vào trang chủ.

2. Bài tập 2: Tạo các route khác nhau trong Express

Mô tả: Thêm các route khác nhau để trả về các thông điệp khác nhau dựa trên đường dẫn URL.

Hướng dẫn:



```
JS ex1.js JS ex2.js X
exeExpress > JS ex2.js > ...
1 const express = require('express');
2 const app = express();
3 const port = 3000;
4
5 app.get('/', (req, res) => {
6   res.send('This is the homepage');
7 });
8
9 app.get('/about', (req, res) => {
10   res.send('This is the about page');
11 });
12
13 app.get('/contact', (req, res) => {
14   res.send('This is the contact page');
15 });
16
17 app.listen(port, () => {
18   console.log(`Server is running on http://localhost:${port}`);
19 });
```

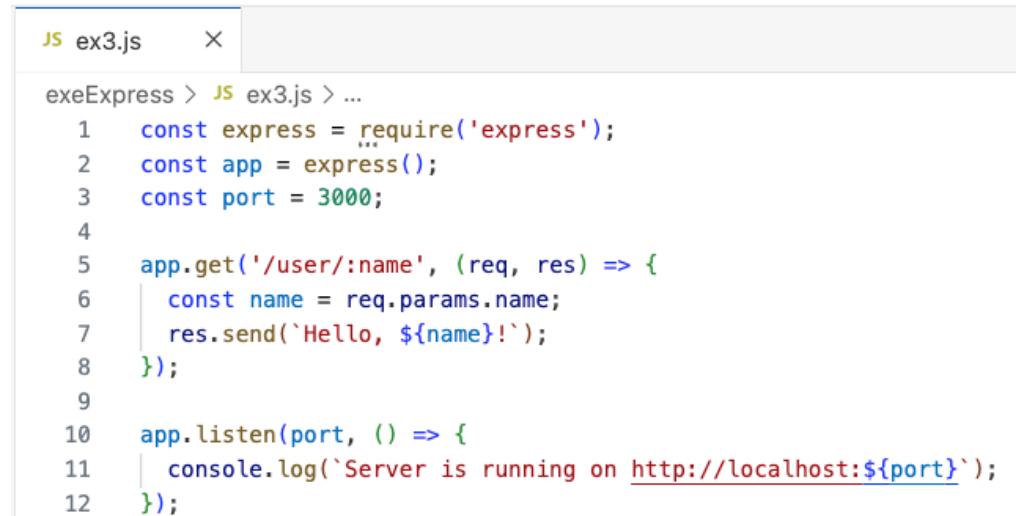
Yêu cầu khảo sát:

- Tạo các route: /, /about, và /contact.
- Mỗi route trả về một thông điệp khác nhau khi người dùng truy cập.

3. Bài tập 3: Sử dụng req.params để xử lý URL động

Mô tả: Tạo một route động để lấy thông tin từ URL và trả về kết quả dựa trên thông tin đó.

Hướng dẫn:



```
JS ex3.js X
exeExpress > JS ex3.js > ...
1 const express = require('express');
2 const app = express();
3 const port = 3000;
4
5 app.get('/user/:name', (req, res) => {
6   const name = req.params.name;
7   res.send(`Hello, ${name}!`);
8 });
9
10 app.listen(port, () => {
11   console.log(`Server is running on http://localhost:${port}`);
12 });
```

Yêu cầu khảo sát:

- Tạo một route động /user/:name, trong đó :name là một tham số.
- Trả về một thông điệp chào dàu trên tên người dùng được truyền vào URL.

4. Bài tập 4: Xử lý dữ liệu từ form với req.body

Mô tả: Tạo một route POST để xử lý dữ liệu từ form gửi lên.

Hướng dẫn:



```
JS ex4.js    X HTTP http://localhost:3000
exeExpress > JS ex4.js > ...
1  const express = require('express');
2  const bodyParser = require('body-parser'); 486.7k (gzipped: 212k)
3  const app = express();
4  const port = 3000;
5
6  app.use(bodyParser.urlencoded({ extended: true }));
7
8  app.post('/submit', (req, res) => {
9    const name = req.body.name;
10   const email = req.body.email;
11   res.send(`Received form data: Name - ${name}, Email - ${email}`);
12 });
13
14  app.listen(port, () => {
15    console.log(`Server is running on http://localhost:${port}`);
16  });
17
```

Yêu cầu:

- Cài đặt và sử dụng body-parser để xử lý dữ liệu từ form.
- Tạo một form HTML đơn giản và gửi dữ liệu POST đến server.
- Trả về thông tin tên và email từ form.

Có thể sử dụng Postman để thực thi như hình sau:

The screenshot shows the Postman application interface. At the top, there's a header with tabs for 'JS ex4.js' and 'HTTP http://localhost:3000/submit'. Below the header, there's a toolbar with icons for saving, environment, and more. The main area has a 'POST' method selected and the URL 'http://localhost:3000/submit'. A large blue 'Send' button is on the right. Underneath, there are tabs for 'Body' (selected), 'Code', and 'Cookies'. The 'Body' tab shows options for 'none', 'form-data', 'x-www-form-urlencoded' (selected), 'raw', 'binary', and 'GraphQL'. Below these options is a table with two rows. The first row has 'Key' and 'Value' columns. The second row has 'name' with value 'ken' and 'email' with value 'ken@gmail.com'. The third row is empty with 'Key' and 'Value' columns. At the bottom of the body section, there are tabs for 'Pretty', 'Raw', 'Preview', and 'HTML' (selected). To the right of these tabs are icons for copy, search, and refresh. The status bar at the bottom shows 'Status: 200 OK Time: 9 ms Size: 281 B'.

5. Bài tập 5: Sử dụng Middleware trong Express

Mô tả: Sử dụng Middleware trong Express để xử lý yêu cầu trước khi gửi phản hồi.

Hướng dẫn:

The screenshot shows a code editor window with the file 'ex5.js' open. The code is written in JavaScript and uses the Express framework. It starts by requiring 'express', creating an app, and setting the port to 3000. It then adds a middleware to log requests to the console. Finally, it defines a route for the homepage ('/') that sends back a simple message. The code is numbered from 1 to 18.

```
exeExpress > JS ex5.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3000;
4
5  // Middleware để log thông tin request
6  app.use((req, res, next) => {
7    |   console.log(`Request Method: ${req.method}, Request URL: ${req.url}`);
8    |   next();
9  });
10
11 // Route chính
12 app.get('/', (req, res) => {
13 |   res.send('This is the homepage');
14 });
15
16 app.listen(port, () => {
17 |   console.log(`Server is running on http://localhost:${port}`);
18 });
```

Yêu cầu khảo sát:

- Tạo một middleware để log thông tin về phương thức HTTP và URL của mỗi yêu cầu.
- Sử dụng middleware này cho mọi route trong ứng dụng.

6. Bài tập 6: Xử lý lỗi trong Express

Mô tả: Thêm chức năng xử lý lỗi trong ứng dụng Express.

Hướng dẫn:

```
JS ex6.js    X
exeExpress > JS ex6.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3000;
4
5  app.get('/', (req, res) => {
6      res.send('This is the homepage');
7  });
8
9  // Route tạo lỗi giả định
10 app.get('/error', (req, res, next) => {
11     const err = new Error('Something went wrong!');
12     next(err);
13 });
14
15 // Middleware xử lý lỗi
16 app.use((err, req, res, next) => {
17     console.error(err.message);
18     res.status(500).send('Internal Server Error');
19 });
20
21 app.listen(port, () => {
22     console.log(`Server is running on http://localhost:${port}`);
23 });
```

Yêu cầu :

- Tạo một route /error giả lập một lỗi.
- Sử dụng middleware để bắt và xử lý lỗi, sau đó trả về thông báo lỗi cho người dùng.

7. Bài tập 7: Triển khai ứng dụng Express với ejs

Cài đặt ejs:

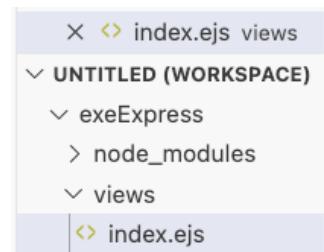
PROBLEMS OUTPUT TERMINAL PORTS POSTMAN CONSOLE

```
● (base) lamtanphuong@192 ~ % npm install ejs
added 16 packages, and audited 82 packages in 1s
15 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
○ (base) lamtanphuong@192 ~ %
```

Mô tả: Sử dụng ejs để tạo các view động trong ứng dụng Express.

Hướng dẫn:

Tạo thư mục views, trong đó có file index.ejs như sau:

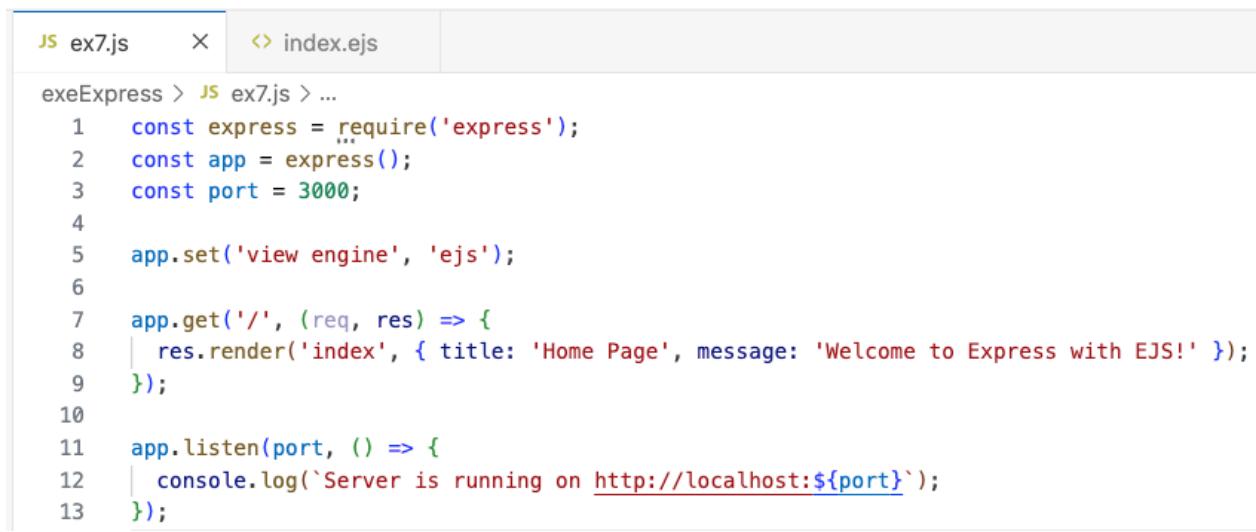


Index.ejs

The image shows the code editor with a tab for 'index.ejs'. The code is an EJS template for an HTML page:

```
exeExpress > views > index.ejs > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title><%= title %></title>
7  </head>
8  <body>
9    <h1><%= message %></h1>
10 </body>
11 </html>
```

Tạo server để thực hiện render



```
JS ex7.js      X  <> index.ejs
exeExpress > JS ex7.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3000;
4
5  app.set('view engine', 'ejs');
6
7  app.get('/', (req, res) => {
8    | res.render('index', { title: 'Home Page', message: 'Welcome to Express with EJS!' });
9  });
10
11 app.listen(port, () => {
12   | console.log(`Server is running on http://localhost:\${port}`);
13 });


```

Yêu cầu khảo sát:

- Cài đặt ejs và cấu hình nó trong ứng dụng Express.
- Tạo một file index.ejs và hiển thị dữ liệu động từ server.

Phần mở rộng

Đây là phần mở rộng, sinh viên tự làm thêm

My SQL module

Yêu cầu chuẩn bị:

- Có 1 server MySQL (có thể sử dụng xampp)
- Tạo sẵn 1 CSDL và trong đó có sẵn 1 bảng dữ liệu như hình sau:

The screenshot shows the phpMyAdmin interface. At the top, there's a navigation bar with tabs for Structure, SQL, Search, Query, Export, Import, Operations, Privileges, Routines, and Events. Below the navigation bar, the main area shows a database structure with a tree view. Under the 'db_test' database, there is a 'users' table. A summary table for the 'users' table is displayed, showing 4 rows, InnoDB type, utf8mb4_general_ci collation, and sizes of 16.0 KiB and 0 B. Below this, a detailed table structure for the 'users' table is shown with columns: #, Name, Type, Collation, Attributes, Null, Default, Comments, Extra, and Action. The 'id' column is defined as int(11) with AUTO_INCREMENT, 'username' as varchar(50) with utf8mb4_general_ci collation, and 'email' as varchar(50) with utf8mb4_general_ci collation. The 'Action' column includes links for Change, Drop, and More. At the bottom, a list of five records from the 'users' table is shown with columns: id, username, and email. Each record has edit, copy, and delete options.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	username	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
3	email	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More

			id	username	email	
<input type="checkbox"/>	Edit	Copy	Delete	1	Ken	ken@gmail.com
<input type="checkbox"/>	Edit	Copy	Delete	3	Peter	peter@email.com
<input type="checkbox"/>	Edit	Copy	Delete	4	Mary	mary@abc.com
<input type="checkbox"/>	Edit	Copy	Delete	5	Pitty	pitty@gmail.com

Cài đặt module mysql:

The terminal window shows the following output:

```
● (base) lamtanphuong@192 ~ % npm install mysql
added 11 packages in 1s
○ (base) lamtanphuong@192 ~ %
```

1. Bài tập 1: Kết nối đến cơ sở dữ liệu MySQL

Mô tả: Tạo một ứng dụng Node.js để kết nối đến cơ sở dữ liệu MySQL sử dụng module mysql.

Hướng dẫn:

Yêu cầu:

The code editor shows a file named `ex1.js` with the following content:

```
JS ex1.js X
exemysql > JS ex1.js > [?] connection > 🔍 password
1 const mysql = require('mysql'); 271.9k (gzipped: 88.7k)
2
3 // Tạo kết nối đến MySQL
4 const connection = mysql.createConnection({
5   host: 'localhost',
6   user: 'root',
7   password: '',
8   database: 'db_test'
9 });
10
11 connection.connect((err) => {
12   if (err) {
13     console.error('Error connecting: ' + err.stack);
14     return;
15   }
16   console.log('Connected as id ' + connection.threadId);
17 });
18
19 connection.end();
```

Yêu cầu khảo sát:

Kết nối đến cơ sở dữ liệu MySQL sử dụng thông tin đăng nhập như host, user, password, và database.

In ra thông báo nếu kết nối thành công hoặc lỗi nếu kết nối thất bại.

2. Bài tập 2: Thực hiện truy vấn SELECT

Mô tả: Truy vấn dữ liệu từ một bảng trong cơ sở dữ liệu MySQL và hiển thị kết quả.

Hướng dẫn:

```
JS ex2.js      X  
exemysql > JS ex2.js > ...  
1  const mysql = ...require('mysql');  271.9k (gzipped: 88.7k)  
2  
3  const connection = mysql.createConnection({  
4    host: 'localhost',  
5    user: 'root',  
6    password: '',  
7    database: 'db_test'  
8  );  
9  
10 connection.connect();  
11  
12 connection.query('SELECT * FROM users', (err, results, fields) => {  
13   if (err) throw err;  
14  
15   console.log('Data from users table:');  
16   console.log(results);  
17 });  
18  
19 connection.end();
```

Yêu cầu:

Kết nối đến cơ sở dữ liệu.

Thực hiện truy vấn SELECT * FROM users và in kết quả ra màn hình.

Ngắt kết nối sau khi truy vấn thành công.

3. Bài tập 3: Chèn dữ liệu vào bảng (INSERT)

Mô tả: Sử dụng truy vấn INSERT để thêm một bản ghi mới vào bảng trong cơ sở dữ liệu.

Hướng dẫn:

Yêu cầu:

```
JS ex3.js    X
exemysql > JS ex3.js > ...
1   const mysql = ...require('mysql');  271.9k (gzipped: 88.7k)
2
3   const connection = mysql.createConnection({
4     host: 'localhost',
5     user: 'root',
6     password: '',
7     database: 'db_test'
8   );
9
10  connection.connect();
11
12  const user = {username: 'John Lam', email: 'lam@gmail.com' };
13  connection.query('INSERT INTO users(username, email) VALUES(?)',
14    |           |           |           |           user, (err, results) => {
15      if (err) throw err;
16
17      console.log('Inserted data with ID:', results.insertId);
18    });
19
20  connection.end();
```

Yêu cầu khảo sát:

- Thêm một người dùng mới với username và email vào bảng users.
- Hiển thị ID của bản ghi vừa được thêm.

4. Bài tập 4: Cập nhật dữ liệu trong bảng (UPDATE)

Mô tả: Sử dụng truy vấn UPDATE để cập nhật thông tin của một bản ghi trong bảng.

Hướng dẫn:

Yêu cầu:

```
JS ex3.js JS ex4.js X
exemysql > JS ex4.js > ...
1  const mysql = ...require('mysql');  271.9k (gzipped: 88.7k)
2
3  const connection = mysql.createConnection({
4      host: 'localhost',
5      user: 'root',
6      password: '',
7      database: 'db_test'
8  });
9
10 connection.connect();
11
12 const id = 1;
13 const newname = 'Ricardo';
14 connection.query('UPDATE users SET username = ? WHERE id = ?',
15                   [newname, id], (err, results) => {
16     if (err) throw err;
17
18     console.log('Changed rows:', results.affectedRows);
19
20
21 connection.end();
```

Yêu cầu khảo sát:

- Cập nhật username của người dùng có id = 1 trong bảng users.
- Hiển thị số lượng dòng đã bị ảnh hưởng bởi truy vấn UPDATE.

5. Bài tập 5: Xóa dữ liệu khỏi bảng (DELETE)

Mô tả: Sử dụng truy vấn DELETE để xóa một bản ghi khỏi bảng.

Hướng dẫn:

```
JS ex5.js    X  
exemysql > JS ex5.js > ...  
1  const mysql = require('mysql');  271.9k (gzipped: 88.7k)  
2  
3  const connection = mysql.createConnection({  
4      host: 'localhost',  
5      user: 'root',  
6      password: '',  
7      database: 'db_test'  
8  });  
9  
10 connection.connect();  
11  
12 const userId = 1;  
13 connection.query('DELETE FROM users WHERE id = ?',[userId], (err, results) => {  
14     if (err) throw err;  
15  
16     console.log('Deleted rows:', results.affectedRows);  
17 } );  
18  
19  
20 connection.end();
```

Yêu cầu khảo sát:

- Xóa người dùng có id = 1 khỏi bảng users.
- Hiển thị số lượng dòng đã bị xóa.

6. Bài tập 6: Sử dụng Promise để thực hiện truy vấn

Mô tả: Thay vì sử dụng callback, hãy sử dụng Promise để thực hiện truy vấn MySQL.

Hướng dẫn:

```
JS ex5.js      JS ex6.js      X
exemysql > JS ex6.js > ...
1  const mysql = ...require('mysql');  271.9k (gzipped: 88.7k)
2
3  const connection = mysql.createConnection({
4    host: 'localhost',
5    user: 'root',
6    password: '',
7    database: 'db_test'
8  );
9
10 connection.connect();
11
12 const queryPromise = (query, params) => {
13   return new Promise((resolve, reject) => {
14     connection.query(query, params, (err, results) => {
15       if (err) return reject(err);
16       resolve(results);
17     });
18   });
19 };
20
21 queryPromise('SELECT * FROM users')
22   .then(results => {
23     console.log('Data:', results);
24   })
25   .catch(err => {
26     console.error('Error:', err);
27   })
28   .finally(() => {
29     connection.end();
30   });

```

Yêu cầu khảo sát:

- Tạo một hàm queryPromise để thực hiện truy vấn MySQL và trả về một Promise.
- Thực hiện truy vấn SELECT * FROM users và xử lý kết quả sử dụng Promise.

7. Bài tập 7: Sử dụng MySQL Pool

Mô tả: Tạo một pool kết nối đến MySQL và thực hiện truy vấn.

Hướng dẫn:

Yêu cầu:



```
JS ex7.js X
exemysql > JS ex7.js > ...
1  const mysql = require('mysql');  271.9k (gzipped: 88.7k)
2
3  const pool = mysql.createPool({
4    connectionLimit: 10,
5    host: 'localhost',
6    user: 'root',
7    password: '',
8    database: 'db_test'
9  });
10
11 pool.query('SELECT * FROM users', (err, results) => {
12   if (err) throw err;
13
14   console.log('Data from users table:');
15   console.log(results);
16
17   pool.end();
18 });
```

Yêu cầu khảo sát:

- Sử dụng mysql.createPool() để tạo một pool kết nối MySQL.
- Thực hiện truy vấn SELECT * FROM users thông qua pool.

8. Bài tập 8: Sử dụng MySQL với async/await

Mô tả: Sử dụng cú pháp async/await để thực hiện truy vấn MySQL.

Hướng dẫn:

```
JS ex8.js      X

exemysql > JS ex8.js > [?] connection > 🔒 password
1  const mysql = require('mysql');  271.9k (gzipped: 88.7k)
2  const util = require('util');
3
4  const connection = mysql.createConnection({
5    host: 'localhost',
6    user: 'root',
7    password: '',
8    database: 'db_test'
9  );
10
11 connection.connect();
12
13 //chuyển đổi hàm query của đối tượng connection từ dạng callback
14 //sang dạng Promise để có thể sử dụng cú pháp async/await
15 const query = util.promisify(connection.query).bind(connection);
16
17 (async () => {
18   try {
19     const results = await query('SELECT * FROM users');
20     console.log('Data:', results);
21   } catch (err) {
22     console.error('Error:', err);
23   } finally {
24     connection.end();
25   }
26 })();
```

Yêu cầu khảo sát:

- Chuyển đổi hàm connection.query thành dạng Promise sử dụng util.promisify.
- Sử dụng async/await để thực hiện truy vấn MySQL và xử lý kết quả.

RESTful API

Bài tập: Tạo một RESTful API quản lý người dùng

Yêu cầu:

GET /users: Lấy danh sách tất cả người dùng.

GET /users/:id: Lấy thông tin chi tiết của một người dùng dựa trên ID.

POST /users: Thêm một người dùng mới.

PUT /users/:id: Cập nhật thông tin của người dùng dựa trên ID.

DELETE /users/:id: Xóa một người dùng dựa trên ID.

Bước 1: Cài đặt môi trường

Trước tiên, chúng ta phải cần cài đặt các gói cần thiết:

- npm init -y

```
● (base) lamtanphuong@192 exeRESTfulApi % npm init -y
Wrote to /Users/lamtanphuong/Documents/My Data/TDU session/node session/exeRESTfulApi/package.json:

{
  "name": "exerestfulapi",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

○ (base) lamtanphuong@192 exeRESTfulApi %
```

- npm install express body-parser

```
● (base) lamtanphuong@192 exeRESTfulApi % npm install express body-parser
added 65 packages, and audited 66 packages in 2s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
○ (base) lamtanphuong@192 exeRESTfulApi %
```

- npm install mysql

```
● (base) lamtanphuong@192 exeRESTfulApi % npm install mysql
added 12 packages, and audited 78 packages in 997ms

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
○ (base) lamtanphuong@192 exeRESTfulApi %
```

→ express: Framework dùng để tạo server.

→ body-parser: Middleware dùng để phân tích dữ liệu từ POST request.

- Sử dụng lại CSDL db_test:

Tài liệu thực hành Chuyên đề ngôn ngữ lập trình

The screenshot shows the phpMyAdmin interface for the 'db_test' database. The left sidebar lists databases like 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', and 'qlsanpham'. The main area shows the 'Structure' tab for the 'users' table, which has three columns: 'id' (int(11)), 'username' (varchar(50)), and 'email' (varchar(50)). The table contains five rows of data: 1 (Ken, ken@gmail.com), 3 (Peter, peter@email.com), 4 (Mary, mary@abc.com), and 5 (Pitty, pitty@gmail.com). Below the table, there is a list of users with edit, copy, and delete options.

- Cài đặt module mysql:

```
PROBLEMS OUTPUT TERMINAL PORTS ... zsh + 
● (base) lamtanphuong@192: ~ % npm install mysql
added 11 packages in 1s
○ (base) lamtanphuong@192: ~ %
```

Bước 2: Cấu hình API trong Express

Hướng dẫn:

```
JS exapi.js ×
exeRESTfulApi > JS exapi.js > [o] connection > database
1 const express = require('express');
2 const bodyParser = require('body-parser'); 486.7k (gzipped: 212k)
3 const mysql = require('mysql'); 271.9k (gzipped: 88.7k)
4 const util = require('util');
5 const app = express();
6 const port = 3000;
7
8 // Sử dụng bodyParser để xử lý dữ liệu JSON từ request body
9 app.use(bodyParser.json());
10
11 //Khởi tạo kết nối đến CSDL
12 const connection = mysql.createConnection({
13   host: 'localhost',
14   user: 'root',
15   password: '',
16   database: 'db_test'
17 });
18
19 connection.connect();
20 //chuyển đổi hàm query sang dạng promises để sử dụng async/await
21 const query = util.promisify(connection.query).bind(connection);
22
23 // 1. GET /users - Lấy danh sách tất cả người dùng
24 app.get('/users', (req, res) => {
25   (async ()=>{
26     try{
27       const users = await query('SELECT * FROM users');
28       res.json(users);
29     }catch(err){
30       return res.status(500).send('No data select');
31     }finally{
32       connection.end();
33     }
34   })();
35
36 });

```

```

JS exapi.js  X
exeRESTfulApi > JS exapi.js > [?] connection > ⚡ database

37 // 2. GET /users/:id - Lấy thông tin chi tiết của một người dùng theo ID
38 app.get('/users/:id', (req, res) => {
39   (async ()=>{
40     try{
41       const iduser = req.params.id; // nhận giá trị id
42       const users = await query('SELECT * FROM users where id = ?', [iduser]);
43       res.json(users);
44     }catch(err){
45       return res.status(500).send('No data select');
46     }
47   })();
48 });
49
50
51 // 3. POST /users - Thêm người dùng mới
52 app.post('/users', (req, res) => {
53   // nhận dữ liệu
54   const username = req.body.username;
55   const email = req.body.email;
56
57   //xử lý và trả kết quả
58   (async ()=>{
59     try{
60       const result = await query('INSERT INTO users(username, email) VALUES(?,?)',
61       | | [username, email]);
62       res.json(result);
63     }catch(err){
64       return res.status(500).send('No data select');
65     }
66   })();
67 });

```

```

JS exapi.js  X
exeRESTfulApi > JS exapi.js > [?] connection > ⚡ database

68
69 // 4. PUT /users/:id - Cập nhật thông tin người dùng theo ID
70 app.put('/users/:id', (req, res) => {
71   //nhận dữ liệu
72   const userId = req.params.id;
73   const username = req.body.username;
74   const email = req.body.email;
75   //xử lý và trả kết quả
76   (async ()=>{
77     try{
78       const result = await query('UPDATE users SET username = ?, email = ? WHERE id = ?',
79       | | | [username, email, userId]);
80       res.json(result);
81     }catch(err){
82       return res.status(500).send('No data select');
83     }
84   })();
85 });

```

```
JS exapi.js X
exeRESTfulApi > JS exapi.js > ...
86
87 // 5. DELETE /users/:id - Xóa người dùng theo ID
88 app.delete('/users/:id', (req, res) => {
89     //nhận dữ liệu
90     const userId = req.params.id;
91     //xử lý và trả kết quả
92     (async ()=>{
93         try{
94             const result = await query('DELETE FROM users WHERE id = ?', [userId]);
95             res.json(result);
96         }catch(err){
97             return res.status(500).send('No data select');
98         }
99     })();
100 });
101
102 // Khởi động server
103 app.listen(port, () => {
104     console.log(`Server is running on http://localhost:${port}`);
105 });


```

Bước 3: Giải thích

GET /users: Trả về danh sách tất cả người dùng. Dữ liệu là một mảng các đối tượng người dùng được lấy từ CSDL với các thông tin như id, name, và email.

GET /users/:id: Lấy thông tin chi tiết của một người dùng được lấy từ CSDL dựa trên ID.

POST /users: Thêm một người dùng mới. Dữ liệu người dùng được gửi qua req.body, bao gồm name và email. ID của người dùng mới được tự động tạo và thêm vào CSDL.

PUT /users/:id: Cập nhật thông tin người dùng vào CSDL dựa trên ID. Thông tin có thể bao gồm name hoặc email.

DELETE /users/:id: Xóa người dùng dựa trên ID.

Bước 4: Kiểm tra API

Bạn có thể sử dụng công cụ như Postman hoặc cURL để kiểm tra các route của API:

GET: Kiểm tra bằng cách truy cập URL <http://localhost:3000/users>.

POST: Thêm người dùng mới bằng cách gửi yêu cầu POST với dữ liệu JSON tới <http://localhost:3000/users>.

PUT: Cập nhật thông tin người dùng bằng cách gửi yêu cầu PUT tới <http://localhost:3000/users/:id>.

DELETE: Xóa người dùng bằng cách gửi yêu cầu DELETE tới <http://localhost:3000/users/:id>.

Tài liệu thực hành Chuyên đề ngôn ngữ lập trình

```
● (base) lamtanphuong@192 node session % mkdir chat-room
● (base) lamtanphuong@192 node session % cd chat-room
● (base) lamtanphuong@192 chat-room % npm init -y
Wrote to /Users/lamtanphuong/Documents/My Data/TDU session/node session/chat-room/package.json:

{
  "name": "chat-room",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

○ (base) lamtanphuong@192 chat-room %

PROBLEMS   OUTPUT   TERMINAL   PORTS   ...
zsh - chat-room + ▾  ⌂  ... <
● (base) lamtanphuong@192 chat-room % npm install express
added 65 packages, and audited 66 packages in 2s
13 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
● (base) lamtanphuong@192 chat-room % npm install socket.io
added 23 packages, and audited 89 packages in 1s
13 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
○ (base) lamtanphuong@192 chat-room %
```