**CÂU HỎI ĐỊNH HƯỚNG CHỦ ĐỀ HÀM (FUNCTION)**

**I. Tổng quan về hàm**

**1. Hàm là gì? Tại sao cần dùng hàm trong lập trình?**

Hàm là khối mã nhận tham số và trả về giá trị. Dùng Hàm trong lập trình để có thể tách nhỏ công việc và tái sử dụng.

**2. Sự khác nhau giữa hàm định nghĩa sẵn và hàm do người dùng tự định nghĩa là gì?**

Hàm định nghĩa có sẵn thì không cần khai báo và hàm do người dùng tự định nghĩa thì ngược lại

**3. Cú pháp khai báo một hàm trong JavaScript như thế nào?**

Function functionName(){

}

**4. Làm sao để gọi (sử dụng) một hàm đã định nghĩa?**

let ketQua = tenHam("doiSo1", "doiSo2");

console.log(ketQua); // In giá trị mà hàm trả về

**5. Hàm có thể trả về giá trị như thế nào? Dùng return để làm gì?**

Hàm trả về giá trị thế nào tùy thuộc vào định nghĩa hàm mà lập trình viên đã đặt trước.

Dùng return là để định nghĩa kết quả hàm trả về

**6. Hàm có thể không có tham số hoặc nhiều tham số không?**

Hàm có thể có nhiều hoặc không có tham số

**7. Điều gì xảy ra nếu gọi hàm mà truyền thiếu hoặc dư tham số?**

Nếu truyền thiếu tham số, tham số thiếu sẽ định nghĩa undefinded

Nếu truyền dư, các đối số dư sẽ không được gán giá trị

**II. Làm việc với tham số và giá trị trả về**

**8. Tham số là gì? Giá trị truyền vào hàm gọi là gì?**

Tham số là các giá trị của hàm, giá trị truyền vào hàm là đối số

**9. Làm sao để gán giá trị mặc định cho tham số?**

function chào(tên = "Khách", lờiChào = "Xin chào")

{ console.log(`${lờiChào}, ${tên}!`);

Chúng ta có thể gán giá trị mặc định trực tiếp trong Hàm

**10. Hàm có thể trả về nhiều giá trị không? Nếu không thì có cách nào “giả” để làm vậy?**

Hàm chỉ có thể trả về 1 giá trị đó là return mà lập trình viên đặt. Có cách làm giá điều vậy bằng cách return lại 1 mảng

**11. Sự khác biệt giữa console.log(...) và return ... trong hàm là gì?**

console.log(...) là để hiển thị kết quả trong console

return là trả về giá trị của hàm

**III. Các cách khai báo hàm trong JavaScript**

**12. Khác biệt giữa function declaration và function expression là gì?**

**13. Arrow function (=>) là gì? Khi nào nên dùng?**

// Function expression truyền thống

const cộng = function(a, b)

{ return a + b; }; // Arrow function tương đương

const cộngArrow = (a, b) => a + b;

console.log(cộng(5, 2)); // Output: 7

console.log(cộngArrow(5, 2)); // Output: 7

**14. Khi nào không nên dùng arrow function? (ví dụ khi dùng this)**

**15. Có thể gán hàm cho biến, truyền hàm như tham số, hay trả về hàm khác được không?**

// Function Declaration

function sayHello(name) {

return `Xin chào, ${name}!`; } // Gán hàm sayHello cho biến greetingFunction const greetingFunction = sayHello; // Gọi hàm thông qua biến

console.log(greetingFunction("Alice")); // Output: Xin chào, Alice!

**IV. Tư duy xử lý với hàm**

**16. Làm sao để chia nhỏ chương trình thành nhiều hàm đơn giản hơn?**

Để chia nhỏ từng hàm cần định nghĩa từng hàm với công việc nhỏ nhất, chuyên biệt và cụ thể

**17. Thế nào là một hàm làm một việc duy nhất (clean function)?**

Ví dụ như việc tính tổng. hàm làm duy nhất 1 việc đó là return a+b

**18. Hàm có thể gọi chính nó không? (đệ quy là gì?)**

có thể. Ví dụ cụ thể

function factorial(n) {

// Trường hợp cơ sở: nếu n là 0, trả về 1

if (n === 0) { return 1; }

// Bước đệ quy: gọi lại factorial với n-1 và nhân với n

else { return n \* factorial(n - 1); } }

console.log(factorial(5)); // Output: 120 (5 \* 4 \* 3 \* 2 \* 1)

**19. Khi nào nên dùng đệ quy thay vì vòng lặp?**

**Khi nào nên sử dụng đệ quy:**

* Khi bài toán có cấu trúc đệ quy tự nhiên.
* Khi giải pháp đệ quy dễ hiểu và dễ viết hơn so với giải pháp lặp.
* Khi độ sâu đệ quy được kiểm soát và không có nguy cơ tràn bộ nhớ ngăn xếp.

**Khi nào nên cân nhắc sử dụng vòng lặp thay vì đệ quy:**

* Khi hiệu suất là yếu tố quan trọng và giải pháp lặp có thể hiệu quả hơn.
* Khi có nguy cơ cao về tràn bộ nhớ ngăn xếp do độ sâu đệ quy lớn.
* Khi giải pháp lặp dễ hiểu và bảo trì hơn

**20. Làm sao để viết hàm tái sử dụng được nhiều lần?**

Hàm phải có nhiệm vụ cụ thể và giá trị return phải có công dụng chung

**V. Hàm nâng cao và kỹ thuật liên quan**

**21. Closure là gì? Khi nào một hàm tạo ra closure?**

 Một hàm được định nghĩa bên trong một hàm khác.

 Hàm bên trong này sau đó có quyền truy cập vào các biến của hàm bên ngoài (và các phạm vi xa hơn bên ngoài).

function outerFunction(outerVar) {

let innerVar = 'Hello from inner';

function innerFunction() {

console.log(outerVar);

console.log(innerVar); }

return innerFunction; }

const myClosure = outerFunction('Hello from outer');

myClosure(); // Output: Hello from outer // Hello from inner

**22. Callback là gì? Dùng callback để làm gì?**

Trong JavaScript, callback là một hàm được truyền vào một hàm khác như một đối số. Hàm callback này thường được thực thi tại một thời điểm sau đó, thường là sau khi hàm bên ngoài đã hoàn thành một tác vụ nào đó. Nói một cách đơn giản, callback là một hàm được "gọi lại" bởi một hàm khác

**23. Hàm setTimeout, setInterval có liên quan gì đến callback?**

**24. Higher-order function là gì?**

**25. Các hàm như map, filter, reduce có gì đặc biệt?**

**VI. Lỗi thường gặp và cách khắc phục**

**26. Điều gì xảy ra nếu bạn quên return trong hàm?**

Hàm sẽ không trả về giá trị nào

**27. Gọi hàm trước khi khai báo có được không?**

Trong JavaScript, việc gọi hàm trước khi khai báo có thể hoặc không thể thực hiện được, tùy thuộc vào cách bạn khai báo hàm đó:

**1. Function Declaration (Khai báo hàm):**

console.log(sayHello("Alice"));// Gọi hàm trước khi khai báo

function sayHello(name) {

return `Xin chào, ${name}!`; }

**2. Function Expression (Biểu thức hàm):**

// console.log(sayGoodbye("Bob")); // Lỗi: Cannot access 'sayGoodbye' before initialization (nếu dùng const/let) // hoặc undefined is not a function (nếu dùng var)

const sayGoodbye = function(name) {

return `Tạm biệt, ${name}!`; };

console.log(sayGoodbye("Bob")); // Gọi hàm sau khi đã gán giá trị

**28. Làm sao để debug một hàm hoạt động sai?**

Dùng console để xem bug đoạn nào

**29. Làm sao để kiểm thử (test) một hàm?**

Thêm đoạn cosole.log để kiểm tra đoạn code trả về giá trị nào

**30. Khi nào nên viết hàm lại thành nhỏ hơn, dễ hiểu hơn?**

Khi cần làm rõ từng thao tác của 1 công việc. chia nhỏ hàm để thực hiện