Ôn tập

1. Giải thuật và thuật toán là : - Giải thuật là : Là một tập hợp các quy trình hoặc lời giải để giải quyết một vấn đề - Thuật toán là : Là một tập hợp các bước của quy trình được thiết kế để giải quyết một vấn đề

2. So sánh trình bày pseudo-code và flowchart : - Mã giả : Ngôn ngữ tự nhiên , dễ hiểu , khó hiểu với những người ko cùng ngôn ngữ - Lưu đồ : Sử dụng hình vẽ , có tính chặt chẽ , nếu ko có chuyên môn thì sẽ ko hiểu được

3. Tại sao cần trình bày giải thuật trước khi vào code : - Giảm thiểu lỗi : khắc phục những sai sót trước khi thực hiện  - Tiết kiệm thời gian :

4. Tại sao lại nên sử dụng flowchart hơn pseudo-code khi thể hiện giải thuật của bài toán phức tạp?  - Lưu đồ cho chúng ta một biểu đồ rõ ràng dễ nhìn hơn , dễ sửa lỗi hơn vì sử dụng hình khối và mũi tên

5. Tại sao giải thuật lại cần hữu hạn các bước thực thi : - Để đảm bảo tính khả thi và hiệu quả của giải thuật

6. Trình bày ý nghĩa các ký hiệu sử dụng khi vẽ flowchart: - Hình tròn : bắt đầu , kết thúc  - Hình chữ nhật : tính toán  - Hình bình hành : xuấp nhập  - Hình thoi : rẽ nhánh

7. Vẽ giải thuật giải phương trình bậc nhất :  - Nếu a = 0 và b = 0, phương trình vô số nghiệm. - Nếu a = 0 và b ≠ 0, phương trình vô nghiệm. - Nếu a ≠ 0, ta chia bởi a vào cả hai vế để được phương trình x + c = 0, với c = b/a. - Giải phương trình x + c = 0 để tìm nghiệm của x.

8. Bạn biết gì về VCS – Version Control System (hệ thống quản lý phiên bản) : - Là một hệ thống lưu giữ các phiên bản của mã nguồn của sản phẩm phần mềm, giúp các lập trình viên có thể dễ dàng lấy lại phiên bản mong muốn.

9. Phân biệt hệ thống quản lý mã nguồn tập trung và hệ thống quản lý mã nguồn phân tán:  - Tính độc lập của các phiên bản - Trong CVCS, một thành viên chỉ có thể làm việc với phiên bản mới nhất được lưu trữ trong kho chung  -Trong khi đó, trong DVCS, các thành viên có thể checkout và commit phiên bản của riêng mình mà không cần phải phụ thuộc vào các phiên bản khác.

10. Git là gì? Tại sao cần sử dụng git:  - Là một hệ thống quản lý mã nguồn phân tán  - Khi có thay đổi mới Git sẽ tạo ra một phiên bản mới để lưu trữ  -Theo dõi được lịch sử của mã nguồn và tìm kiếm các phiên bản cũ để khắc phục hoặc theo dõi  - Dùng được trên nhiều hệ điều hành khác nhau

11. Git thuộc hệ thống quản lý mã nguồn tập trung hay phân tán? - Hệ thống quản lý mã nguồn phân tán

12. Repository là gì? Phân biệt LocalRepository và RemoteRepository:  - Là một kho lưu trữ chứa mã nguồn, tài liệu và các thành phần khác của dự án phần mềm. - Remote dùng giữa nhiều người , Local là dùng cho máy tính của bản thân , dành cho 1 người dùng

13. Đồng bộ giữa các repository (local và remote) để làm gì : -

Là quá trình đảm bảo rằng tất cả các phiên bản của mã nguồn được lưu trữ ở cả local và remote đều giống nhau

14. Thao tác push của git để làm gì? Cú pháp câu lệnh :  -

Thao tác push của git được sử dụng để đẩy các thay đổi từ local repository lên remote repository, làm cho các thay đổi được phản ánh và cập nhật trên server  - Git push origin master.

15. Thao tác pull của git để làm gì:  -

Thao tác pull của Git được sử dụng để lấy các thay đổi mới nhất từ remote repository và cập nhật chúng vào local repository.

16. Thao tác clone của git để làm gì? Cú pháp câu lệnh :  -

Thao tác clone của Git được sử dụng để tải về một bản sao đầy đủ của repository từ remote repository về local repository  -git clone https://github.com/username/repository.git

17. Nêu tên và ý nghĩa các câu lệnh cơ bản đã được học để làm việc với git : -

git init: Khởi tạo một repository mới trong thư mục hiện tại. - git commit -m "[commit message]": Commit các thay đổi vào local repository với message miêu tả nội dung của commit. - git push [remote] [branch]: Đẩy các thay đổi từ local repository lên remote repository.

18. www là gì :

 - tên miền

19. HTML là gì :

- là một ngôn ngữ đánh dấu được sử dụng để xây dựng các trang web

20. Nêu tên các thẻ làm việc với văn bản trong tài liệu html :

 - Thẻ tiêu đề h1 > h6  - Thẻ văn bản <p> - Thẻ chèn ảnh <img> - Thẻ danh sách <ul> <ol> <li> - Thẻ liên kết <a> - Thẻ phân đoạn <div>

21. Thẻ div dùng để làm gì?

<div> khác gì <span>  - Thẻ div để phân đoạn chia khu vực khác nhau của trang web - Thẻ div sử dụng khu vực lớn hơn trên trang web  -Thẻ <span> được áp dụng cho một phần của nội dung

22. URL là gì? Phân biệt url tuyệt đối và url tương đối :

 - Là một chuỗi ký tự được sử dụng để định vị tài nguyên trên Internet - URL tương đối là đường dẫn tới tài nguyên liên quan đến thư mục hiện tại hoặc trang web, trong khi URL tuyệt đối đưa ra đầy đủ thông tin về tên miền, đường dẫn và tên tài nguyên

23. Siêu liên kết là gì :

Là một đoạn văn bản hoặc hình ảnh trên trang web có thể được nhấp chuột để chuyển đến một tài nguyên khác trên cùng trang web hoặc một trang web khác trên Internet.

24. Muốn liên kết đến một phần trên cùng trang thì làm thế nào :

 - Sử dụng thuộc tính href của thẻ <a> và đặt giá trị của nó bằng # kèm theo ID của phần muốn liên kết đến.

25. Muốn liên kết đến một phần trên trang khác thì làm thế nào -

Để liên kết đến một phần trên trang khác, bạn cần biết địa chỉ URL của trang đó và ID của phần muốn liên kết đến .

26. Phân biệt internal link và external link? -

internal link là một liên kết giữa các trang web khác nhau trong cùng một trang web, trong khi external link là một liên kết giữa hai trang web khác nhau trên Internet.

28. có mấy loại danh sách ?

\_2 loại : ul-(unordered list). ol-(ordered list)

ul : là danh sách đc sắp xếp ko theo thứ tự nhất định

ol : là loại danh sách được sắp xếp theo thứ tự số hoặc chữ cái

29.hiển thị danh sách theo thứ tự bắt buộc C D E

30.Form hay biểu mẫu dùng để làm gì?

\_form: dùng để thu thập thông tin người dùng, cho phép người dùng nhập liệu và gửi thông tin lên máy chủ web để xử lý

\_biểu mẫu : Biểu mẫu có thể được sử dụng để tạo các tính năng như đăng ký thành viên, đăng nhập, tìm kiếm, thanh toán trực tuyến, liên hệ với quản trị viên, đăng bài viết và nhiều chức năng khác

31.thuộc tính action của form dùng để làm gì?

\_Thuộc tính "action" của form được sử dụng để chỉ định URL của trang web hoặc tài nguyên mà dữ liệu trong form sẽ được gửi đến sau khi người dùng bấm nút "submit". Điều này cho phép máy chủ nhận dữ liệu và thực hiện các xử lý phù hợp với nó

32.phân biệt 2 method là get và post?

1,GET:

.Dữ liệu được gửi đi sẽ hiển thị trên thanh địa chỉ của trình duyệt

.không sử dụng để gửi các dữ liệu nhạy cảm

. thường sủ dụng để gửi những giữ liệu nhỏ

2,POST:

.Dữ liệu được gửi đi không hiển thị trên thanh địa chỉ của trình duyệt.Dữ liệu được gửi trong phần body của yêu cầu HTTP.

.Được sử dụng để gửi các dữ liệu nhạy cảm.

.Không hạn chế dung lượng dữ liệu gửi đi. Thường được sử dụng để gửi dữ liệu lớn lên server..

34. muốn submit form đến trang codeym làm nào ?

- sử dụng thuộc tính action

vd : <form action="https://www.codeym.com>

35. nối 3 ô trên cùng 1 hàng ?

dùng thuộc tính : <th colspan = "3" ></th>

36. nối 3 ô trên cùng 1 cột

dùng thuộc tính : <td rowspan="2"></td>

37 .Phân biệt thuộc tính cellpadding và thuộc tính cellspacing?

cellpadding: Được sử dụng để định nghĩa khoảng cách giữa nội dung của mỗi ô và lề (margin) xung quanh của ô đó. Khoảng cách này có thể là một giá trị số hoặc là một phần trăm của kích thước của ô. Ví dụ: cellpadding="5" hoặc cellpadding="10%". Nếu không được chỉ định, giá trị mặc định của cellpadding là 1 pixel.

cellspacing: Được sử dụng để định nghĩa khoảng cách giữa các ô trong bảng. Khoảng cách này cũng có thể là một giá trị số hoặc là một phần trăm của kích thước của bảng. Ví dụ: cellspacing="5" hoặc cellspacing="10%". Nếu không được chỉ định, giá trị mặc định của cellspacing là 2 pixel.

38 .Trình bày các cách nhúng javascript vào 1 tài liệu html?

- dùng cặp thẻ <script></script> trong phần body của tài liệu Html

\_ dùng tập tin name.js riêng , dùng thuộc tính "src " link tới file name.js

\_dùng sự kiện để gắn đoạn mã js vào các phần tử html như ' onclick'

39.Phát biểu: “JavaScript là một ngôn ngữ script dựa trên đối tượng nhằm phát triển các ứng dụng Internet dựa trên client và server” là đúng hay sai? Giải thích?

\_ Đúng .

40.Phát biểu “JavaScript phân biệt chữ hoa chữ thường ” đúng hay sai?

\_ Đúng. ví dụ đặt 1 biên là myVariable và cố gắng truy cập biến bằng tên myvariable thì sẽ xảy ra lỗi ko tìm thấy biến.

41.Trình bày về quy tắc CamelCase và camelCase?

\_ CamelCase : là quy tắc đặt tên từ đầu tiên của từ đầu tiên được viết hoa và các từ tiếp theo được viết liền kề nhau mà không có dấu cách và từ đầu tiên của mỗi từ sau được viết hoa.\_camelCase :là quy tắc đặt tên từ đầu tiên của từ đầu tiên được viết thường và từ đầu tiên của các từ sau được viết hoa. Các từ được viết liền kề nhau mà không có dấu cách.

42.Biến là gì? Phân biệt biến cục bộ (local) và biến toàn cục (global)?

\_biến là một vùng bộ nhớ được sử dụng để lưu trữ giá trị hoặc tham chiếu đến đối tượng. Biến có thể thay đổi giá trị của nó trong suốt quá trình thực thi của chương trình.

\_Biến cục bộ (local variable) là biến được khai báo và chỉ được truy cập trong phạm vi của một hàm hoặc một khối mã. Khi hàm kết thúc hoặc khi khối mã kết thúc. biến đc khai báo trong hàm hoặc bên trong 1 khối mã và ko thể truy cập được từ bên ngoài phạm vi khối mã đó.

\_Biến toàn cục (global variable) là biến được khai báo bên ngoài bất kỳ hàm hoặc khối mã nào và có thể truy cập ở bất kỳ nơi nào trong chương trình. Biến toàn cục sẽ không bị giải phóng khỏi bộ nhớ cho đến khi chương trình kết thúc.

43.Phân biệt: var, let, const?

\_var được sử dụng để khai báo biến toàn cục hoặc cục bộ trong một hàm. Tuy nhiên, khi sử dụng var, biến có thể được truy cập và ghi đè từ mọi phần của chương trình, dẫn đến nguy cơ xảy ra lỗi do việc trùng tên biến hoặc không kiểm soát được giá trị của biến.

\_let và const được giới thiệu trong phiên bản ES6 của JavaScript nhằm giải quyết vấn đề này. let được sử dụng để khai báo biến cục bộ trong một khối mã (block scope) và có thể được ghi đè lại giá trị của nó trong cùng khối mã. Ngoài ra, let không cho phép khai báo lại biến trong cùng một khối mã.

Trong khi đó, const cũng được sử dụng để khai báo biến cục bộ trong một khối mã, nhưng giá trị của nó không thể thay đổi sau khi được gán. Điều này đảm bảo rằng các giá trị được gán cho biến const sẽ không bị thay đổi, đồng thời giúp cho mã của bạn trở nên an toàn hơn và dễ đọc hơn.

44.Phát biểu “JavaScript là ngôn ngữ định kiểu yếu hay động” nghĩa là gì? Trình bày?

\_Phát biểu “JavaScript là ngôn ngữ định kiểu yếu hay động” có nghĩa là JavaScript không yêu cầu khai báo rõ ràng kiểu dữ liệu của một biến trước khi sử dụng nó. Điều này cho phép bạn gán các giá trị khác nhau cho cùng một biến trong cùng một chương trình.

\_Tuy nhiên, điều này cũng tạo ra một số vấn đề khi ta sử dụng JavaScript. Vì JavaScript là một ngôn ngữ định kiểu yếu, nó không thể kiểm tra được kiểu dữ liệu của biến khi viết mã, và do đó dễ xảy ra lỗi trong quá trình thực thi

\_Mặt khác, vì JavaScript là một ngôn ngữ định kiểu động, nó cho phép các biến có thể thay đổi kiểu dữ liệu của chúng trong quá trình thực thi. Điều này giúp cho việc lập trình trở nên linh hoạt hơn, tuy nhiên cũng có thể dẫn đến khó khăn trong việc debug và tìm lỗi.

45.Nêu tên các kiểu dữ liệu nguyên thủy trong JavaScript?

1.kiểu number

2.kiểu chuỗi(string)

3.kiểu bolean : đại diện cho giá trị true or false

4.kiểu null : ko có giá trị

5.kiểu underfined :đại diện cho một biến chưa được khởi tạo giá trị hoặc một thuộc tính không tồn tại trong một đối tượng.

46.Trình bày về toán tử typeof? Khi khai báo “let x=10;” thì x có kiểu là gì?

\_Toán tử typeof là một toán tử trong JavaScript, được sử dụng để trả về kiểu dữ liệu của một biến hoặc một giá trị

\_let x= 10 : là kiểu dữ liệu number

47.Trình bày các cách đưa thông báo trong JavaScript?

- alert : hiện thị thông báo hộp thoại

\_confirm : hiện thị thông báo xác nhận hoặc từ chối hoạt động có thêm 2 nút 'ok' và 'cencel'

\_prompt : hộp thoại nhập liệu

48.Trình bày về các loại toán tử trong JavaScript?

1. toán tử số học : + , - , \* , / , % , lũy thừa \*\*

2. toán tử gán : = ; += ; -= ; \*= ; /=

3. toán tử so sánh == , >, < , =>, <=

4. toán tử logic && , \\ , ! (phủ định)

5.Toán tử ba ngôi: (condition) ? value1 : value2;

Toán tử ba ngôi là một toán tử có ba phần, được sử dụng để kiểm tra một điều kiện và trả về một giá trị nếu điều kiện đúng và một giá trị khác nếu điều kiện sai

49.Phân biệt giữa 2 toán tử == và ===?

(==) được gọi là toán tử so sánh trừu tượng

(===) được gọi là toán tử so sánh cân bằng nghiêm ngặt.

50.Phân biệt giữa 2 toán tử && và || trong JavaScript?

Toán tử && (AND) trả về giá trị true nếu cả hai biểu thức đều đúng (true)

Toán tử || (OR) trả về true nếu một trong hai biểu thức đúng,

51.Kết quả biểu thức +0===-0 là true hay false?

\_ kết quả: true

\_gthich: Trong JavaScript, +0 và -0 là hai giá trị khác nhau, tuy nhiên chúng được coi là bằng nhau theo quy ước của chuẩn IEEE 754. Vì vậy, khi so sánh hai giá trị này bằng toán tử ===, kết quả trả về là true.

52 .Trình bày về toán tử ++,-- và – trong JavaScript?

x++ tăng giá trị biến lên 1 và trả về giá trị trước khi tăng

++x tăng giá trị biến lên 1 và trả về giá trị sau khi tăng

x-- giảm giá trị biến xuống 1 và trả về giá trị trước khi giảm

--x giảm giá trị biến xuống 1 và trả về giá trị sau khi giảmm-x: Phủ định giá trị của biến Boolean x. Nếu x là true, kết quả là false và ngược lại.

53.Phân biệt ++a và a++?

++a : tăng giá trị biến a lên 1 và trả về giá trị sau khi tăng

a++ : tăng giá trị biến a lên 1 và trả về giá trị trước khi tăng

54.Câu lệnh điều kiện/rẽ nhánh là gì?

\_Câu lệnh này được sử dụng để kiểm tra một giá trị hoặc biểu thức, và thực thi một số hành động tùy thuộc vào kết quả của kiểm tra đó.

55.Có những loại câu lệnh điều kiện nào?

\_ 2 loại chính

1.Câu lệnh "if": Đây là câu lệnh dùng để thực hiện một hành động nếu điều kiện được kiểm tra là đúng và một hành động khác nếu điều kiện sai

2. Câu lệnh "switch"- "case": Câu lệnh switch cũng được sử dụng để kiểm tra điều kiện, nhưng cho phép nhiều giá trị được so sánh với một biến duy nhất và thực hiện các hành động khác nhau cho từng giá trị đó.

Câu 56: Trình bày về câu lệnh iF

"Câu lệnh if" là một câu lệnh điều kiện trong các ngôn ngữ lập trình. Nó cho phép chương trình thực hiện một hành động nếu một điều kiện được xác định là đúng, và một hành động khác nếu điều kiện đó là sai.

Cú pháp của câu lệnh if thường được viết như sau:

Copy Code

if (điều kiện) {

//Thực hiện hành động nếu điều kiện đúng

} else {

//Thực hiện hành động nếu điều kiện sai

}

Câu 57: Trình bày về câu lệnh Switch

Câu lệnh switch" là một câu lệnh điều kiện trong các ngôn ngữ lập trình. Nó cho phép chương trình thực hiện một hành động tùy thuộc vào giá trị của biến được xác định.

Cú pháp của câu lệnh switch thường được viết như sau:

Copy Code

switch (biến) {

case giá\_trị\_1:

//Thực hiện hành động nếu biến có giá trị bằng giá\_trị\_1

break;

case giá\_trị\_2:

//Thực hiện hành động nếu biến có giá trị bằng giá\_trị\_2

break;

...

default:

//Thực hiện hành động mặc định nếu biến không bằng bất kỳ giá trị nào khác

break;

}

Khi chạy, chương trình sẽ kiểm tra giá trị của biến và thực hiện hành động tương ứng với giá trị đó. Nếu không có giá trị nào khớp với giá trị của biến, chương trình sẽ thực hiện hành động mặc định.

Câu 58: Tại sao lại nói else là một tùy chọn (optional) của câu lệnh if?

Câu lệnh if được sử dụng để kiểm tra điều kiện và thực hiện một hành động nếu điều kiện đó đúng. Tuy nhiên, không phải lúc nào cũng cần thực hiện một hành động khác nếu điều kiện đó sai.

Do đó, else được coi là một tùy chọn trong câu lệnh if, vì nếu không có else, chương trình sẽ chỉ thực hiện hành động trong if khi điều kiện đúng và không làm gì cả khi điều kiện sai.

Ví dụ, ta có thể sử dụng câu lệnh if để kiểm tra xem một số có phải là số chẵn hay không, và chỉ in ra thông báo "So 6 la so chan" nếu số đó là số chẵn:

Copy Code

int so = 6;

if (so % 2 == 0) {

printf("So %d la so chan", so);

}

Trong ví dụ này, chúng ta không cần sử dụng else vì chỉ cần in ra thông báo khi điều kiện đúng, và không cần làm gì cả khi điều kiện sai.

Câu 59 Khi nào nên sử dụng câu lệnh switch?

Trong câu lệnh switch, "break" được sử dụng để kết thúc một case cụ thể và thoát khỏi câu lệnh switch. Khi chương trình gặp phải lệnh break trong một case, nó sẽ thoát khỏi switch và tiếp tục thực hiện các lệnh sau switch.

Nếu một case không có lệnh break, chương trình sẽ tiếp tục thực hiện các case tiếp theo cho đến khi gặp lệnh break hoặc kết thúc câu lệnh switch

Câu 60 Câu lệnh break dùng để làm gì? Nếu một case mà không có break thì chương trình chạy thế nào?

Câu lệnh break trong câu lệnh switch được sử dụng để kết thúc một khối code và thoát ra khỏi câu lệnh switch. Khi chương trình gặp phải câu lệnh break, nó sẽ không tiếp tục kiểm tra các case khác mà chỉ thoát khỏi câu lệnh switch.

Nếu một case trong câu lệnh switch không có break, chương trình sẽ tiếp tục kiểm tra các case khác mà không quan tâm đến giá trị của biểu thức điều kiện. Điều này gọi là "fall-through behavior". Nếu không có break ở case cuối cùng, chương trình sẽ thực hiện mã lệnh ở khối default, hoặc kết thúc khối câu lệnh switch nếu không có default.

Vì vậy, việc sử dụng break là rất quan trọng để đảm bảo rằng chương trình sẽ thực hiện đúng logic được mong muốn.

Câu 61Mệnh đề default có bắt buộc trong câu lệnh switch?

Trong câu lệnh switch, mệnh đề default không bắt buộc. Mệnh đề này được sử dụng khi không có bất kỳ trường hợp nào trong các điều kiện case đúng với giá trị của biểu thức được kiểm tra. Nếu mệnh đề default không được sử dụng và không có bất kỳ trường hợp nào khớp với giá trị được kiểm tra, thì câu lệnh switch sẽ không thực hiện bất kỳ hành động nào và thoát khỏi switch.

Câu 62 Trình bày về toán tử 3 ngôi?

Toán tử 3 ngôi, còn được gọi là toán tử có điều kiện, là một loại toán tử trong các ngôn ngữ lập trình. Toán tử này gồm bốn thành phần: một biểu thức điều kiện và hai biểu thức kết quả.

Cú pháp của toán tử 3 ngôi như sau:

Copy Code

*(condition)* ? *(value\_if\_true)* : *(value\_if\_false)*

Khi giá trị của biểu thức điều kiện đúng, toán tử sẽ trả về giá trị của biểu thức kết quả đầu tiên (value\_if\_true), ngược lại nếu biểu thức điều kiện sai thì trả về giá trị của biểu thức kết quả thứ hai (value\_if\_false).

Ví dụ, ta có thể sử dụng toán tử 3 ngôi để kiểm tra một số nguyên là số chẵn hay không:

javascriptCopy Code

var number = 7;

var result = (number % 2 === 0) ? 'even' : 'odd'; *// If number is even then return 'even', otherwise return 'odd'*

console.log(result); *// Output: odd*

Trong ví dụ trên, điều kiện kiểm tra xem number có chia hết cho 2 không. Nếu chia hết, toán tử 3 ngôi trả về chuỗi "even", nếu không thì trả về chuỗi "odd". Kết quả khi chạy đoạn mã trên là "odd", vì số 7 không chia hết cho 2.

Câu 63 Trình bày về sự kiện và trình quản lý sự kiện trong javascript?

Sự kiện (event) trong JavaScript là các hành động của người dùng hoặc trình duyệt mà trang web có thể đáp ứng. Ví dụ như click chuột, nhập liệu, cuộn trang, v.v.

Trình quản lý sự kiện (event listener) là một phần mềm được cài đặt bằng JavaScript để theo dõi sự kiện và thực hiện một hành động cụ thể khi sự kiện xảy ra.

Để gắn một trình quản lý sự kiện cho một phần tử HTML, ta sử dụng phương thức addEventListener() của đối tượng phần tử đó.

Cú pháp chung của addEventListener() như sau:

javascriptCopy Code

element.addEventListener(eventType, eventHandler);

Trong đó:

* element là đối tượng phần tử HTML.
* eventType là tên của sự kiện muốn gắn trình quản lý (ví dụ: "click", "keydown", "submit", v.v.).
* eventHandler là hàm xử lý sự kiện (hàm JavaScript được thực thi khi sự kiện xảy ra).

Ví dụ, để gắn một trình quản lý sự kiện cho một nút button khi được click, ta có thể viết code như sau:

javascriptCopy Code

*// Lấy tham chiếu đến nút button*

var button = document.getElementById("myButton");

*// Gắn trình quản lý sự kiện cho nút button*

button.addEventListener("click", function() {

alert("Button clicked!");

});

Câu 64 Sự kiện onchange phát sinh khi nào?

Sự kiện onchange được phát sinh khi giá trị của một phần tử HTML (như input, select, textarea) được thay đổi . Nghĩa là khi người dùng nhập liệu hoặc chọn các giá trị trong phần tử đó và sau đó di chuyển focus ra khỏi phần tử đó, sự kiện onchange sẽ được kích hoạt.

Câu 65

Sự kiện onclick trong Javascript phát sinh khi người dùng click chuột lên một phần tử DOM (Document Object Model) trên trang web, ví dụ như một button hoặc một đường link.

Câu 66 Muốn hiển thị 1 hộp thoại xác nhận “Bạn chắc chắn xóa không?” với người dùng khi bấm vào nút delete thì làm thế nào?

Để hiển thị hộp thoại xác nhận "Bạn chắc chắn xóa không?" với người dùng khi bấm vào nút delete, bạn có thể sử dụng hàm confirm của Javascript.

Ví dụ, nếu bạn có một button với id là "delete-btn", bạn có thể sử dụng đoạn mã sau để hiển thị hộp thoại xác nhận khi người dùng click vào button:

javascriptCopy Code

document.getElementById("delete-btn").addEventListener("click", function() {

if (confirm("Bạn chắc chắn xóa không?")) {

*// Logic xóa khi người dùng đồng ý*

} else {

*// Logic khi người dùng không đồng ý*

}

});

Trong đó, hàm confirm sẽ trả về giá trị true nếu người dùng click vào nút "OK" và trả về giá trị false nếu người dùng click vào nút "Cancel". Bạn có thể sử dụng giá trị trả về này để xử lý logic tiếp theo sau khi người dùng xác nhận hoặc từ chối xóa.

Câu 67 Phân biệt onmousevover và onmouseout, onmousemove?

onmouseover, onmouseout và onmousemove là các sự kiện (events) trong Javascript liên quan đến chuột.

* onmouseover: Sự kiện này xảy ra khi con trỏ chuột di chuyển vào một phần tử HTML. Nó có thể được sử dụng để thực hiện các hành động như hiển thị thông tin chi tiết của một sản phẩm, hoặc hiển thị một hình ảnh lớn hơn của một sản phẩm trong trang web.
* onmouseout: Sự kiện này xảy ra khi con trỏ chuột di chuyển ra khỏi phần tử HTML. Nó có thể được sử dụng để ẩn bớt thông tin chi tiết của một sản phẩm hoặc hình ảnh lớn hơn của một sản phẩm.
* onmousemove: Sự kiện này xảy ra khi con trỏ chuột di chuyển trên một phần tử HTML. Nó có thể được sử dụng để theo dõi vị trí của con trỏ chuột trên trang web, từ đó có thể thực hiện một số hành động tùy theo vị trí của chuột, ví dụ như hiển thị tooltip.

Vì vậy, để phân biệt giữa các sự kiện này, ta cần xác định rõ được chức năng của từng sự kiện và cách sử dụng chúng để áp dụng cho các tình huống cụ thể trong trang web của mình. Nếu bạn muốn hiển thị thông tin hoặc hình ảnh lớn của một sản phẩm khi người dùng di chuột vào, bạn có thể sử dụng onmouseover, còn nếu muốn ẩn thông tin hoặc hình ảnh đó khi người dùng di chuột ra khỏi phần tử thì sử dụng onmouseout. Nếu muốn theo dõi vị trí của con trỏ chuột trên trang web, bạn có thể sử dụng onmousemove.

Giả sử bạn có một ảnh sản phẩm trong trang web và bạn muốn hiển thị một hình ảnh lớn của sản phẩm đó khi người dùng di chuột vào ảnh nhỏ.

Ví dụ sử dụng onmouseover:

htmlCopy Code

<img src="small-product-image.jpg" onmouseover="displayLargeImage('large-product-image.jpg')">

Trong đó, chúng ta sử dụng sự kiện onmouseover để gọi hàm displayLargeImage() khi người dùng di chuột vào ảnh nhỏ. Hàm này sẽ hiển thị hình ảnh lớn của sản phẩm tương ứng.

Ví dụ sử dụng onmouseout:

htmlCopy Code

<img src="small-product-image.jpg" onmouseover="displayLargeImage('large-product-image.jpg')" onmouseout="hideLargeImage()">

Trong đó, chúng ta sử dụng sự kiện onmouseout để gọi hàm hideLargeImage() khi người dùng di chuột ra khỏi ảnh nhỏ. Hàm này sẽ ẩn đi hình ảnh lớn của sản phẩm.

Ví dụ sử dụng onmousemove:

htmlCopy Code

<div onmousemove="displayTooltip(event)">

Hover over me to see a tooltip

</div>

Trong đó, chúng ta sử dụng sự kiện onmousemove để gọi hàm displayTooltip() khi người dùng di chuột trên phần tử DIV. Hàm này sẽ hiển thị một tooltip theo vị trí của con trỏ chuột. Chúng ta truyền tham số event vào hàm để lấy được thông tin về vị trí của con trỏ chuột trên trang web.

Câu 68 Phân biệt onkeyup, onkeydown và onkeypress?

* onkeyup: Sự kiện này xảy ra khi bạn nhả phím bàn phím. Nó thường được sử dụng để xử lý các hành động người dùng sau khi họ nhập dữ liệu vào một trường văn bản hoặc vào một form.
* onkeydown: Sự kiện này xảy ra khi bạn bấm xuống một phím bàn phím. Nó có thể được sử dụng để xử lý các tác vụ như điều hướng trang web, gõ chữ hoặc thực hiện các thao tác khác.
* onkeypress: Sự kiện này xảy ra khi bạn giữ nút bấm xuống và giữ cho đến khi nó in ra ký tự tương ứng lên màn hình. Nó thường được sử dụng để thực hiện các hành động liên quan đến nhập liệu của người dùng.

Câu 69 Phân biệt onfocus và onblur?

* onfocus: Sự kiện này xảy ra khi một phần tử HTML được tập trung vào, tức là khi nó trở thành phần tử đang được điều khiển bởi người dùng. Nó thường được sử dụng để xử lý các tác vụ như hiển thị thông tin cụ thể hay thay đổi giao diện của phần tử khi nó được tập trung vào.
* onblur: Sự kiện này xảy ra khi một phần tử HTML không còn được tập trung vào nữa, tức là khi người dùng chuyển sang tập trung vào một phần tử khác hoặc click chuột vào bất kỳ nơi nào khác trên trang web. Nó thường được sử dụng để xử lý các tác vụ như kiểm tra tính hợp lệ của đầu vào hoặc ẩn đi một số thông tin chi tiết.

Ví dụ, bạn có thể sử dụng sự kiện onfocus để thay đổi màu nền của một phần tử khi nó được tập trung vào:

htmlCopy Code

<input type="text" id="myInput" onfocus="this.style.backgroundColor = 'yellow'">

Trong đó, khi phần tử <input> được tập trung vào, sự kiện onfocus sẽ được kích hoạt và thay đổi màu nền của phần tử thành màu vàng.

Tương tự, bạn có thể sử dụng sự kiện onblur để kiểm tra tính hợp lệ của đầu vào khi người dùng chuyển sang phần tử khác:

htmlCopy Code

<input type="text" id="myInput" onblur="validateInput()">

Trong đó, khi phần tử <input> không còn được tập trung vào( tức là đã điền xong ), sự kiện onblur sẽ được kích hoạt và gọi đến hàm validateInput() để kiểm tra tính hợp lệ của đầu vào.

Câu 70 Sự kiện onload phát sinh khi nào?

innerHTML và innerText là hai thuộc tính của đối tượng HTML Element trong JavaScript.

innerHTML trả về hoặc thiết lập nội dung HTML bên trong một phần tử. Nó cho phép bạn thêm hoặc xóa HTML bên trong phần tử đó. Ví dụ:

htmlCopy Code

<div id="example">Hello <span>world</span></div>

javascriptCopy Code

const element = document.getElementById("example");

console.log(element.innerHTML); *// "Hello <span>world</span>"*

element.innerHTML = "Hi <strong>there</strong>";

console.log(element.innerHTML); *// "Hi <strong>there</strong>"*

innerText cũng trả về hoặc thiết lập nội dung bên trong một phần tử, tuy nhiên nó sẽ loại bỏ tất cả các thẻ HTML và trả về chỉ các văn bản. Ví dụ:

htmlCopy Code

<div id="example">Hello <span>world</span></div>

javascriptCopy Code

const element = document.getElementById("example");

console.log(element.innerText); *// "Hello world"*

element.innerText = "Hi there";

console.log(element.innerText); *// "Hi there"*

Vì innerHTML sử dụng HTML, nó có thể gây ra lỗi bảo mật nếu không kiểm soát được nguồn dữ liệu đầu vào. Trong khi đó, innerText an toàn hơn để sử dụng vì nó chỉ hiển thị văn bản thuần túy và không chứa mã độc.

Câu 71 Phân biệt innerHTML và innerText?

innerHTML được sử dụng để thay đổi hoặc truy cập vào toàn bộ nội dung của một phần tử, bao gồm cả các thẻ HTML bên trong và văn bản. Nó trả về một chuỗi chứa mã HTML và tất cả các nội dung được hiển thị trên trang web.

Ví dụ:

htmlCopy Code

<div id="myDiv">This is <b>bold</b> text.</div>

javascriptCopy Code

var content = document.getElementById("myDiv").innerHTML;

console.log(content); *// "This is <b>bold</b> text."*

Trong khi đó, innerText chỉ trả về nội dung văn bản của phần tử, loại bỏ các thẻ HTML bên trong. Điều này có nghĩa là innerText không trả về các thẻ HTML, chỉ trả về văn bản được hiển thị trên trang web.

Ví dụ:

htmlCopy Code

<div id="myDiv">This is <b>bold</b> text.</div>

javascriptCopy Code

var text = document.getElementById("myDiv").innerText;

console.log(text); *// "This is bold text."*

Tóm lại, innerHTML trả về toàn bộ nội dung HTML của phần tử, bao gồm cả các thẻ HTML bên trong, trong khi innerText chỉ trả về nội dung văn bản của phần tử và loại bỏ các thẻ HTML bên trong.

Câu 72 Nếu muốn vô hiệu hóa (disabled) một button khi click vào 1 checkbox thì làm thế nào?

Để vô hiệu hóa một button khi click vào một checkbox, bạn có thể sử dụng các sự kiện (event) để lắng nghe sự kiện click của checkbox và sau đó đặt thuộc tính disabled của button thành true.

Ví dụ, giả sử bạn có một checkbox với id là myCheckbox và một button với id là myButton, bạn có thể sử dụng mã JavaScript sau đây để vô hiệu hóa button khi checkbox được chọn:

htmlCopy Code

*<!-- Checkbox và button -->*

<input type="checkbox" id="myCheckbox">

<button id="myButton">Click me</button>

<script>

const myCheckbox = document.getElementById("myCheckbox");

const myButton = document.getElementById("myButton");

*// Lắng nghe sự kiện click của checkbox*

myCheckbox.addEventListener("click", function() {

*// Nếu checkbox được chọn, vô hiệu hóa button*

if (myCheckbox.checked) {

myButton.disabled = true;

} else {

*// Nếu checkbox không được chọn, kích hoạt lại button*

myButton.disabled = false;

}

});

</script>

Trong ví dụ trên, khi checkbox được chọn, thuộc tính disabled của button sẽ được đặt thành true, làm cho button trở thành không hoạt động. Khi checkbox không được chọn, thuộc tính disabled sẽ được đặt lại thành false, cho phép button hoạt động lại bình thường.

Câu 73 Phân biệt thuộc tính readonly và disabled của html?

Cả hai thuộc tính readonly và disabled đều được sử dụng trong HTML để giới hạn hoạt động của các phần tử như input, select và textarea. Tuy nhiên, chúng có một số khác biệt quan trọng:

1. Thuộc tính readonly chỉ cho phép người dùng xem giá trị của phần tử và không thể chỉnh sửa nó, trong khi đó thuộc tính disabled vô hiệu hóa phần tử và không cho phép người dùng tương tác với nó.
2. Khi một phần tử có thuộc tính readonly, giá trị của phần tử có thể được gửi lại máy chủ khi form được gửi đi, trong khi đó khi một phần tử có thuộc tính disabled, giá trị của phần tử sẽ không được gửi về máy chủ.
3. Một phần tử có thể có cả hai thuộc tính readonly và disabled cùng lúc. Trong trường hợp này, thuộc tính disabled sẽ được ưu tiên hơn, nghĩa là phần tử sẽ bị vô hiệu hóa và không cho phép người dùng tương tác với nó.

Ví dụ:

htmlCopy Code

<input type="text" readonly value="Giá trị chỉ đọc"><br>

<input type="text" disabled value="Phần tử bị vô hiệu hóa">

Trong ví dụ trên, phần tử đầu tiên có thuộc tính readonly và người dùng chỉ có thể xem giá trị của nó nhưng không thể chỉnh sửa. Phần tử thứ hai có thuộc tính disabled, làm cho nó bị vô hiệu hóa và người dùng không thể tương tác với nó.

Câu 74 : Vòng lặp là gì?

Vòng lặp (hay còn gọi là loop) là cho phép thực hiện một khối lệnh nhiều lần theo một số điều kiện . Có hai loại vòng lặp chính là vòng lặp for và vòng lặp while. Vòng lặp for được sử dụng khi số lần lặp cần được biết trước, trong khi đó vòng lặp while được sử dụng khi số lần lặp không được biết trước và phụ thuộc vào một điều kiện nào đó

Câu 75 + 76 Nêu các loại vòng lặp cơ bản trong JavaScript?

Trong JavaScript, có ba loại vòng lặp cơ bản gồm:

1. Vòng lặp for: được sử dụng để lặp lại một khối lệnh cho một khoảng giá trị nhất định. Cú pháp:

Copy Code

for (biểu thức khởi tạo; điều kiện; biểu thức cập nhật) {

//khối lệnh muốn lặp lại

}

1. Vòng lặp while: được sử dụng để lặp lại một khối lệnh trong khi một điều kiện nhất định là đúng. Cú pháp:

Copy Code

while (điều kiện) {

//khối lệnh muốn lặp lại

}

1. Vòng lặp do...while: Tương tự với vòng lặp while, chỉ khác ở chỗ nó sẽ thực hiện ít nhất một lần ngay cả khi điều kiện không đúng từ đầu. Cú pháp:

Copy Code

do {

//khối lệnh muốn lặp lại

}

while (điều kiện);

Câu 77 +78+79 Khi nào nên dùng For, While, Do…while

* Vòng lặp do...while nên được sử dụng khi bạn muốn thực hiện một khối lệnh ít nhất một lần và sau đó kiểm tra điều kiện; trong trường hợp điều kiện không đúng, vòng lặp sẽ thoát. Ví dụ: khi bạn cần yêu cầu người dùng nhập ít nhất một giá trị, bạn có thể sử dụng vòng lặp do...while để đảm bảo rằng giá trị được nhập vào.
* Vòng lặp for nên được sử dụng khi số lần lặp lại là biết trước và tập giá trị cần lặp lại nằm trong một khoảng xác định. Ví dụ: khi bạn muốn lặp lại một số lần để thực hiện một tác vụ như in ra các phần tử trong một mảng.
* Vòng lặp while nên được sử dụng khi số lần lặp lại không biết trước và phụ thuộc vào một điều kiện nào đó. Ví dụ: khi bạn muốn lặp lại một tác vụ cho đến khi người dùng thực hiện một hành động nhất định như nhập vào một giá trị hợp lệ.

Câu 80 Yêu cầu người dùng nhập đi nhập lại giá chị của một số cho đến khi số đó là số chẵn thì có nên sử dụng vòng lặp không? Nếu có, nên sử dụng loại vòng lặp nào?

Có, để yêu cầu người dùng nhập đi nhập lại giá trị của một số cho đến khi số đó là số chẵn thì có thể sử dụng vòng lặp while. Vì số lần lặp không biết trước và phụ thuộc vào điều kiện số đó là số chẵn hay không.

Cụ thể, mã JavaScript sử dụng vòng lặp while để thực hiện yêu cầu này như sau:

Copy Code

let num;

do {

num = prompt("Nhập một số chẵn: ");

} while (num % 2 !== 0);

console.log(`Số bạn vừa nhập là ${num}.`);

Câu 81Yêu cầu người dùng tính tổng các số chẵn từ 10 đến 100 thì nên dùng vòng lặp loại nào

Câu 82 Trình bày về vòng for thiếu?

Vòng lặp for thiếu (hay còn gọi là for-in) là một dạng vòng lặp trong JavaScript được sử dụng để duyệt qua các thuộc tính của đối tượng. Vòng lặp này chỉ hoạt động trên các đối tượng có các thuộc tính đếm được như mảng hoặc đối tượng với thuộc tính là số nguyên.

Cú pháp của vòng lặp for thiếu như sau:

Copy Code

for (variable in object) {

*// code to be executed*

}

Trong đó:

* variable: tên biến tham chiếu đến từng thuộc tính của đối tượng.
* object: đối tượng muốn duyệt qua các thuộc tính.

Mỗi lần lặp, biến variable sẽ được gán giá trị của một thuộc tính của đối tượng. Các câu lệnh bên trong khối lệnh của vòng lặp sẽ được thực hiện cho mỗi thuộc tính của đối tượng.

Ví dụ, ta có thể sử dụng vòng lặp for thiếu để duyệt qua các thuộc tính của một đối tượng như sau:

Copy Code

const person = {

name: 'John',

age: 30,

gender: 'male'

};

for (let prop in person) {

console.log(`${prop}: ${person[prop]}`);

}

Kết quả khi chạy đoạn mã trên sẽ là:

Copy Code

name: John

age: 30

gender: male

Trong đó, vòng lặp for thiếu được sử dụng để duyệt qua các thuộc tính của đối tượng person. Mỗi thuộc tính của đối tượng person được gán vào biến prop, và giá trị của thuộc tính được in ra trên console bằng cách sử dụng person[prop].

Câu 83 Nếu muốn hiển thị các cặp số nguyên dương có tổng là 10 thì viết thế nào?

Để hiển thị các cặp số nguyên dương có tổng là 10, ta có thể sử dụng hai vòng lặp for lồng nhau để duyệt qua tất cả các cặp số có thể. Trong mỗi lần lặp, ta kiểm tra tổng của hai số và nếu bằng 10 thì in ra cặp số đó.

Ví dụ, mã JavaScript để hiển thị các cặp số nguyên dương có tổng là 10 như sau:

Copy Code

for (let i = 1; i <= 9; i++) {

for (let j = i + 1; j <= 10 - i; j++) {

if (i + j === 10) {

console.log(`(${i}, ${j})`);

}

}

}

84 Nếu muốn hiển thị hình tam giác vuông \* thì nên viết thế nào?

.để vẽ hình tam giác vuông ta dùng 2 vòng for lồng nhau.vòng for đầu tiên chạy từ 1 đến n trong đó n là số dòng muốn hiển thị. vòng for thứ 2 chạy từ 1 đến i mỗi lần lặp in ra kí tự \*, tạo một biến để lưu kết quả và in ra biến đó.

for (var i = 1; i <= n; i++) {

var row = '';

for (var j = 1; j <= i; j++) {

row += '\*';

}

console.log(row);

}

85. Trình bày về cách chạy của vòng for lồng nhau?

Cách chạy vòng for lồng nhau: vòng đầu tiên chạy từ i đến n mỗi lần vòng đầu chạy vòng thứ 2 sẽ chạy từ j đến i tức mỗi lần vòng i chạy 1 lần thì vòng 2 sẽ lặp i lần

  for(let i = 0; i< n; i++){

            for(let j = 0; j<i ; j++){

            }

        }

86.Phân biệt break và continue:

Break : dùng để kết thúc vòng lặp ngay lập tức khi điều kiện thỏa mãn

Continue: không thoát ra khỏi vòng lặp mà chỉ bỏ qua các lần lặp tiếp theo của vòng lặp và tiếp tục với vòng lặp kế tiếp.

87. Hiển thị các số không chia hết cho 5 từ 1-100 có sử dụng continue thì làm thế nào?

Để hiển thị các số không chia hết cho 5 từ 1-100 và sử dụng lệnh "continue", ta có thể sử dụng một vòng lặp for để lặp qua các số từ 1 đến 100, kiểm tra xem nếu số đó chia hết cho 5 thì bỏ qua và tiếp tục với lần lặp kế tiếp, ngược lại thì in ra giá trị của số đó. Ví dụ:

javascriptCopy Code

for (let i = 1; i <= 100; i++) {

if (i % 5 === 0) {

continue;

}

console.log(i);

}

Đoạn code trên sẽ in ra tất cả các số từ 1 đến 100 không chia hết cho 5. Khi gặp một số chia hết cho 5, lệnh "continue" sẽ được thực thi và bỏ qua các câu lệnh phía sau trong lần lặp đó, tiếp tục với lần lặp kế tiếp trong vòng lặp.

88. Mảng là gì?

Mảng là một kiểu dữ liệu được sử dụng để lưu trữ và quản lý các giá trị có cùng kiểu dữ liệu.

89. Trình bày cú pháp khai báo mảng trong JavaScript?

Cú pháp khai báo mảng

Let arr = [ ];

90 Chỉ số của mảng là gì? Chỉ số của mảng xuất phát từ 0 hay 1?

. chỉ số của mảng là giá trị để xác định vị trí của phần tử trong mảng. Chỉ số của mảng bắt đầu từ 0.

91. Kích thước của mảng là gì?

Kích thước của một mảng là số lượng phần tử trong mảng.

92. Phân biệt push() và unshift():

Push là thêm 1 phần tử cuối mảng

Unshift thêm một phần tử đầu mảng.

93.phân biệt giữa push() và pop()

Push () thêm 1 phần tử vào cuối mảng

Pop() xóa 1 phần tử cuối mảng

94. phân biệt giữa shift và unshift

Shitf () xóa 1 phần tử đầu mảng

Unshift() thêm 1 phần tử vào đầu mảng

95. Nếu muốn xóa phần tử cuối cùng của mảng dùng hàm nào

xóa phần tử cuối dùng hàm pop();

96. Muốn sắp xếp mảng thì dùng hàm nào

muốn sắp xếp mảng dùng hàm sort()

Vd: arr.sort()

97. Có một mảng gồm tên của các bạn học viên. Nếu muốn sắp xếp tên này theo thứ tự alphabet làm thế nào?

Dùng hàm sort

Vd. Arr.sort()

98. Thao tác duyệt mảng trong JavaScript là việc lặp qua từng phần tử của một mảng để thực hiện một số hoạt động trên từng phần tử đó

Vd triển khai

for (let i = 0; i < arr.length; i++) {

console.log(arr[i]);

}

99. Thao tác duyệt mảng 2 chiều trong JavaScript là việc lặp qua từng phần tử của một mảng có cấu trúc theo hàng và cột để thực hiện các hoạt động trên từng phần tử đó. Một mảng 2 chiều là một mảng chứa các mảng con, mỗi mảng con đại diện cho một hàng trong mảng 2 chiều.

100.

Vd thao tác duyệt mảng 2 chiều:

javascriptCopy Code

for (let i = 0; i < arr.length; i++) {

for (let j = 0; j < arr[i].length; j++) {

console.log(scores[i][j]);

}

}

101. Mảng 2 chiều cần mấy chỉ số để xác định duy nhất 1 phần tử

Cần 2 chỉ số là hàng và cột

102. Viết code tính tổng các phần tử của ma trận số (ý tưởng)?

chúng ta khởi tạo biến sum với giá trị ban đầu là 0. Sau đó, chúng ta sử dụng hai vòng lặp for để duyệt qua từng phần tử của ma trận. Với mỗi phần tử, giá trị của nó được cộng vào biến sum.

Vd let sum = 0;

for (let i = 0; i < arr.length; i++) {

for (let j = 0; j < arr[i].length; j++) {

sum += matrix[i][j];

}

}

console.log(sum);

103. Viết code tính tổng các ô trên cùng hàng của ma trận số (ý tưởng)?

chúng ta khởi tạo biến row để chỉ định hàng cần tính tổng. Sau đó, chúng ta khởi tạo biến sum với giá trị ban đầu là 0. Tiếp theo, chúng ta duyệt qua từng phần tử trong hàng được chỉ định và cộng giá trị của nó vào biến sum.

let row = 0;

let sum = 0;

for (let i = 0; i < arr[row].length; i++) {

sum += arr[row][i];

}

console.log(sum)

104. Viết code tính tổng các ô trên cùng cột của ma trận số (ý tưởng)?

húng ta khởi tạo biến column để chỉ định cột cần tính tổng. Sau đó, chúng ta khởi tạo biến sum với giá trị ban đầu là 0. Tiếp theo, chúng ta duyệt qua từng hàng trong ma trận và cộng giá trị của ô ở cột được chỉ định vào biến sum

let column = 0;

let sum = 0;

for (let i = 0; i < matrix.length; i++) {

sum += matrix[i][column];

}

console.log(sum);

105. Hàm (function) là gì?

Hàm (function) trong JavaScript là một khối mã độc lập, có thể được gọi bất cứ khi nào cần thiết. Hàm được sử dụng để thực hiện một tác vụ cụ thể và trả về một giá trị hoặc thực hiện một hành động như in ra một thông báo.

106. Phân biệt giữa tham số hình thức (formal arguments) và tham số thực (actual arguments)?

* Tham số hình thức là một danh sách các biến được khai báo trong định nghĩa của hàm. Chúng ta sử dụng tham số hình thức để truyền thông tin từ bên ngoài vào trong hàm để thực hiện các tính toán. Khi khai báo một hàm, chúng ta có thể chỉ định một hoặc nhiều tham số hình thức trong cặp ngoặc đơn () của hàm. Ví dụ:

javascriptCopy Code

function sum(a, b) {

return a + b;

}

Trong đoạn mã trên, a và b là tham số hình thức trong khai báo của hàm sum.

* Tham số thực là giá trị thực tế được truyền vào khi gọi hàm. Chúng ta sử dụng tham số thực để truyền giá trị từ bên ngoài vào trong hàm. Khi gọi một hàm, chúng ta có thể truyền giá trị cho các tham số hình thức đã được khai báo trong định nghĩa của hàm. Ví dụ:

javascriptCopy Code

let result = sum(2, 3);

Trong đoạn mã trên, 2 và 3 là tham số thực được truyền vào hàm sum khi gọi nó.

Tóm lại, tham số hình thức là các biến được khai báo trong định nghĩa của hàm để truyền thông tin từ bên ngoài vào trong hàm, còn tham số thực là giá trị thực tế được truyền vào khi gọi hàm để truyền giá trị từ bên ngoài vào trong hàm.

107. Gọi hàm (invoke) là gì?

Gọi hàm (invoke) trong JavaScript có nghĩa là thực thi một hàm đã được định nghĩa trước đó. Khi một hàm được gọi, các lệnh bên trong nó sẽ được thực thi, và kết quả trả về (nếu có) sẽ được sử dụng cho các mục đích khác.

108. Phân biệt hàm gọi (calling function) và hàm được goi (called function)?

* Hàm gọi (calling function) là một hàm mà trong đó có lời gọi một hoặc nhiều hàm khác. Khi hàm gọi được thực thi, các lệnh bên trong nó sẽ được thực thi theo thứ tự đã được xác định, và khi đến lời gọi hàm, nó sẽ tạm dừng thực thi và chuyển sang thực hiện các lệnh bên trong hàm được gọi. Sau khi hàm được gọi hoàn thành thực thi, hàm gọi sẽ tiếp tục thực thi các lệnh còn lại của nó. Ví dụ:

javascriptCopy Code

function callingFunction() {

console.log('Start calling function...');

calledFunction(); *// Gọi hàm được gọi*

console.log('End calling function...');

}

function calledFunction() {

console.log('Called function is executing...');

}

callingFunction(); *// Output: "Start calling functi*

* Hàm được gọi (called function) là một hàm được lời gọi từ một hàm khác. Khi hàm được gọi được thực thi, các lệnh bên trong nó sẽ được thực thi theo thứ tự đã được xác định, và kết quả trả về (nếu có) sẽ được truyền cho hàm gọi. Sau khi hàm được gọi hoàn thành thực thi, chương trình sẽ quay lại thực thi các lệnh của hàm gọi. Ví dụ:

javascriptCopy Code

function sum(a, b) {

return a + b;

}

let result = sum(2, 3);

console.log(result); *// Output: 5*

109. Phân biệt truyền tham trị và truyền tham chiếu?

* Truyền tham trị là cách truyền đối số bằng cách sao chép giá trị của biến được truyền vào hàm. Khi một hàm được gọi và truyền đối số theo cách này, bản sao của giá trị được tạo ra và sử dụng trong thân hàm. Giá trị của biến ban đầu không bị ảnh hưởng bởi các thao tác bên trong hàm. Ví dụ:

javascriptCopy Code

function changeValue(a) {

a = 2;

}

let x = 1;

changeValue(x);

console.log(x); *// Output: 1*

* Truyền tham chiếu là cách truyền đối số bằng cách truyền tham chiếu đến vị trí của biến được truyền vào hàm. Khi một hàm được gọi và truyền đối số theo cách này, các thao tác bên trong hàm sẽ ảnh hưởng trực tiếp đến giá trị của biến ban đầu. Ví dụ:

javascriptCopy Code

function changeValue(a) {

a[0] = 2;

}

let x = [1];

changeValue(x);

console.log(x); *// Output: [2]*

Tóm lại, truyền tham trị và truyền tham chiếu là hai cách khác nhau để truyền đối số vào hàm. Cách nào được sử dụng phụ thuộc vào mục đích của chương trình và loại dữ liệu được truyền

110. Câu lệnh return dùng để làm gì?

Câu lệnh return trong JavaScript được sử dụng để trả về giá trị từ một hàm và kết thúc việc thực thi của hàm. Khi một hàm được gọi và chạy đến câu lệnh return, các lệnh bên trong hàm sẽ dừng lại và giá trị được chỉ định trong câu lệnh return sẽ được trả về cho hàm gọi.

111. Có được phép viết “return;”?

Không.

**Câu 112 : Phân biệt return và break?**

function add(a, b) {

return a + b;

}

const sum = add(2, 3); // sum sẽ chứa giá trị 5

Trong khi đó, break được sử dụng để kết thúc vòng lặp (loop) hoặc switch case. Khi break được thực thi trong một vòng lặp hoặc switch case, chương trình sẽ thoát khỏi vòng lặp hoặc switch case đó. Ví dụ:

for (let i = 0; i < 10; i++) {

if (i === 5) {

break; // Khi i = 5, vòng lặp for sẽ kết thúc

}

console.log(i);

}

Kết quả: 0->4

Tóm lại, return được sử dụng để trả về giá trị của một hàm và kết thúc việc thực thi của hàm, trong khi break được sử dụng để kết thúc vòng lặp hoặc switch case.

**Câu 113 : Hàm thực hiện việc kiểm tra nên trả về giá trị không? Nếu có, nên trả về gì?**

Tại sao hàm hoán vị swap(a,b) lại không thực hiện được việc hoán vị của các tham số gốc?

Câu hỏi thứ nhất: Hàm thực hiện việc kiểm tra nên trả về giá trị không? Nếu có, nên trả về gì?

Trong phần lớn các trường hợp, hàm được thiết kế để thực hiện một tác vụ nhất định, chẳng hạn như tính toán một giá trị hoặc thay đổi trạng thái của ứng dụng. Trong các trường hợp như vậy, hàm không cần phải trả về giá trị.

Tuy nhiên, trong một số trường hợp, hàm có thể được thiết kế để thực hiện kiểm tra và trả về kết quả của kiểm tra đó. Ví dụ, một hàm kiểm tra xem một chuỗi có chứa một từ khóa hay không có thể trả về một giá trị boolean - true nếu chuỗi chứa từ khóa và false nếu không. Trong các trường hợp như vậy, hàm cần phải trả về giá trị để cho phép mã gọi hàm sử dụng kết quả kiểm tra đó.

**Câu 114 : Tại sao hàm hoán vị swap(a,b) lại không thực hiện được việc hoán vị của các tham số gốc?**

Trong JavaScript, các tham số được truyền vào hàm được truyền theo giá trị (pass by value), chứ không phải theo tham chiếu (pass by reference). Điều này có nghĩa là khi một biến được truyền vào một hàm, thì một bản sao của giá trị của biến đó sẽ được tạo ra và được sử dụng trong hàm đó. Khi hàm kết thúc, các thay đổi được thực hiện trên bản sao này sẽ không ảnh hưởng đến giá trị của biến gốc.

Vì vậy, trong hàm swap(a,b), khi ta truyền hai tham số vào hàm này, thì hai bản sao của giá trị của hai biến này sẽ được tạo ra và được sử dụng trong hàm. Mặc dù hàm swap sẽ hoán đổi giá trị của hai bản sao này, nhưng giá trị của hai biến gốc sẽ không bị thay đổi. Do đó, khi ta sử dụng hàm swap để hoán vị hai biến, ta cần lưu lại giá trị mới của hai bản sao này và gán chúng cho hai biến gốc tương ứng.

**Câu 115 : Trình bày về OOP?**

OOP (Object-Oriented Programming), hay lập trình hướng đối tượng, là một phương pháp thiết kế và phát triển phần mềm dựa trên các khái niệm về đối tượng.

Trong lập trình hướng đối tượng, đối tượng là một thực thể có thuộc tính và phương thức. Thuộc tính là các đặc điểm của đối tượng cụ thể, trong khi phương thức là các hành động mà đối tượng đó có thể thực hiện. Ví dụ, một đối tượng "học sinh" có thể có các thuộc tính như "tên", "tuổi", "giới tính", "địa chỉ" và các phương thức như "đăng ký khóa học", "nộp bài tập", "xem điểm số".

Trong lập trình hướng đối tượng, các đối tượng được sử dụng để đóng gói các thành phần của chương trình lại với nhau. Điều này giúp cho việc quản lý mã nguồn và tái sử dụng mã nguồn trở nên dễ dàng hơn. Thay vì phải viết lại các đoạn mã tương tự nhau nhiều lần, các đối tượng có thể được sử dụng lại trong các phần của chương trình khác nhau.

Một số khái niệm chính trong lập trình hướng đối tượng bao gồm:

1. Lớp (Class): Lớp là một bản thiết kế hay mẫu của đối tượng, mô tả các thuộc tính và phương thức của đối tượng.
2. Đối tượng (Object): Đối tượng là một thực thể được tạo ra từ lớp, có các thuộc tính và phương thức riêng biệt.
3. Tính kế thừa (Inheritance): Tính kế thừa cho phép một lớp con được tạo ra từ một lớp cha, kế thừa các thuộc tính và phương thức của lớp cha.
4. Đóng gói (Encapsulation): Đóng gói cho phép các thuộc tính và phương thức của đối tượng không được truy cập từ bên ngoài đối tượng.
5. Đa hình (Polymorphism): Đa hình cho phép một đối tượng có thể có nhiều hình dạng hoặc hành vi, tùy thuộc vào ngữ cảnh sử dụng.

Với OOP, việc thiết kế và phát triển phần mềm trở nên dễ dàng hơn, code được tái sử dụng và dễ bảo trì, giúp cho ứng dụng có thể phát triển một cách nhanh chóng và hiệu quả.

**Câu 116 : Phân biệt giữa lập trình hướng đối tượng và lập trình hướng thủ tục/chức năng?**

Lập trình hướng đối tượng (OOP) và lập trình hướng thủ tục/chức năng là hai phương pháp tiếp cận khác nhau để thiết kế và phát triển phần mềm.

Trong lập trình hướng đối tượng, chương trình được phát triển bằng cách xác định các đối tượng trong hệ thống và các mối quan hệ giữa chúng. Các đối tượng có thể chứa dữ liệu và các phương thức để thực hiện các hoạt động trên dữ liệu đó. Lập trình hướng đối tượng tập trung vào đối tượng, tránh việc sử dụng biến toàn cục và hàm.

Còn trong lập trình hướng thủ tục/chức năng, chương trình được viết theo từng bước thực hiện, các hàm và biến được sử dụng rất nhiều. Phương pháp này tập trung vào tác vụ cần thực hiện và các hàm được sử dụng để xử lý các tác vụ đó.

Một số điểm khác biệt giữa OOP và lập trình hướng thủ tục/chức năng:

1. Đối tượng: Trong OOP, chương trình được thiết kế bằng cách định nghĩa các đối tượng và các phương thức để thao tác với chúng. Trong khi đó, trong lập trình hướng thủ tục/chức năng, chương trình được xây dựng bằng cách định nghĩa các hàm và xử lý tác vụ theo tuần tự.
2. Tính tái sử dụng: OOP giúp cho việc tái sử dụng code trở nên dễ dàng hơn nhờ vào tính kế thừa và đa hình. Tuy nhiên, trong lập trình hướng thủ tục/chức năng, các hàm được viết lại nhiều lần để thực hiện các tác vụ tương tự nhau.
3. Quản lý mã nguồn: Trong OOP, việc quản lý mã nguồn được dễ dàng hơn nhờ vào tính đóng gói của đối tượng. Trong khi đó, trong lập trình hướng thủ tục/chức năng, các biến và hàm toàn cục có thể khó kiểm soát và gây ra lỗi nếu không quản lý tốt.
4. Phân chia chức năng: OOP giúp phân chia chức năng của chương trình thành các đối tượng khác nhau và từ đó giúp cho việc phát triển chương trình trở nên dễ dàng hơn. Trong khi đó, trong lập trình hướng thủ tục/chức năng, các chức năng của chương trình có thể được phân chia thành các hàm hoặc module nhỏ để dễ quản lý hơn.

Tóm lại, lập trình hướng đối tượng và lập trình hướng thủ tục/chức năng là hai phương pháp tiếp cận khác nhau trong thiết kế và phát triển phần mềm, với những ưu điểm và hạn chế khác nhau.

**Câu 117 : Phân biệt giữa object và class?**

Trong lập trình hướng đối tượng, object và class là hai khái niệm quan trọng.

Object là một thực thể hoặc một đối tượng cụ thể trong chương trình, ví dụ như một đối tượng "hình chữ nhật" sẽ có thuộc tính "chiều dài" và "chiều rộng". Đối tượng này cũng có thể có các phương thức để thực hiện các hành động trên nó. Object được xem là một thực thể cụ thể và có thể được tạo ra từ một class.

Class là một bản thiết kế hay một mẫu của đối tượng, được sử dụng để tạo ra các instance (thực thể) của đối tượng. Trong class, ta định nghĩa các thuộc tính và phương thức của đối tượng. Class hỗ trợ tính kế thừa, cho phép class con được tạo ra từ class cha, kế thừa các thuộc tính và phương thức của class cha. Các đối tượng được tạo ra từ các class đã được định nghĩa.

Ví dụ:

*// Định nghĩa class Rectangle*

class Rectangle {

constructor(height, width) {

this.height = height;

this.width = width;

}

*// Phương thức*

get area() {

return this.calcArea();

}

calcArea() {

return this.height \* this.width;

}

}

*// Tạo đối tượng từ class Rectangle*

const square = new Rectangle(10, 10);

console.log(square.area); *// Output: 100*

Ở ví dụ trên, ta định nghĩa một class "Rectangle" với hai thuộc tính là "height" và "width". Ta cũng định nghĩa phương thức "calcArea()" để tính diện tích của hình chữ nhật. Sau khi đã định nghĩa class, ta có thể tạo ra một đối tượng mới từ class bằng cách sử dụng từ khóa "new", như trong ví dụ ta tạo ra đối tượng "square". Ta có thể truy cập thuộc tính và phương thức của đối tượng này thông qua dấu chấm (.), ví dụ như "square.area" để tính diện tích của đối tượng.

**Câu 118 : Hai khái niệm “Môn học” và “Môn học JavaScript” thì đâu là object?**

"Môn học" và "Môn học JavaScript" là hai khái niệm, không phải là object trong lập trình hướng đối tượng.

Trong lập trình hướng đối tượng, object thường được coi như một thực thể cụ thể có thuộc tính và phương thức. Ví dụ, nếu ta có một đối tượng "học sinh", thì đối tượng đó sẽ có các thuộc tính như "tên", "tuổi", "địa chỉ", và các phương thức như "đăng ký khóa học", "nộp bài tập".

"Môn học" và "Môn học JavaScript" chỉ là các khái niệm, chúng không có thuộc tính hoặc phương thức được xác định để đối tượng "Môn học" hay "Môn học JavaScript" có thể thực hiện các hành động gì.

Tuy nhiên, trong lập trình JavaScript, "Môn học" và "Môn học JavaScript" có thể được biểu diễn bằng các chuỗi (string) hoặc đối tượng (object) tùy vào ngữ cảnh sử dụng. Chẳng hạn, ta có thể tạo ra một đối tượng "Môn học" với các thuộc tính như "tên môn học", "mô tả", "giáo viên",... và một trong số các thuộc tính này có thể là "JavaScript". Tuy nhiên, việc đó không liên quan đến khái niệm "Môn học" hay "Môn học JavaScript" mà chỉ là cách biểu diễn chúng trong ngữ cảnh lập trình.

**Câu 119 : Mô tả về sơ đồ lớp?**

Sơ đồ lớp (Class diagram) là một trong những loại sơ đồ UML (Unified Modeling Language) phổ biến được sử dụng để thiết kế và mô hình hoá các hệ thống phần mềm, bao gồm cả trong lập trình JavaScript.

Sơ đồ lớp thể hiện các lớp của hệ thống, các thuộc tính và phương thức của mỗi lớp và mối quan hệ giữa các lớp. Các lớp được vẽ dưới dạng hình chữ nhật với tên lớp ở trên cùng, các thuộc tính ở giữa và các phương thức ở dưới cùng. Mối quan hệ giữa các lớp có thể được biểu diễn bằng các mũi tên, ví dụ như quan hệ kế thừa, quan hệ sử dụng, quan hệ liên kết,...

Ví dụ, ta có thể vẽ sơ đồ lớp cho một hệ thống quản lý sinh viên trong JavaScript như sau:



Trong ví dụ này, ta có ba lớp là "Person", "Student" và "Teacher". Lớp "Person" là lớp cha của lớp "Student" và "Teacher", có các thuộc tính chung như "name" và "address". Lớp "Student" và "Teacher" có các thuộc tính riêng của mình như "studentId", "courses" và "teacherId", "department" tương ứng. Cả hai lớp con đều kế thừa phương thức và thuộc tính từ lớp cha "Person". Ngoài ra, lớp "Student" sử dụng một số phương thức của lớp "Course" (phụ thuộc), và lớp "Teacher" có quan hệ liên kết với lớp "Department".

Sơ đồ lớp giúp cho việc thiết kế và phát triển hệ thống trở nên dễ dàng hơn bằng cách giúp ta hiểu rõ hơn các thành phần của hệ thống và mối quan hệ giữa chúng.

**Câu 120 :**

Dưới đây là ví dụ về sơ đồ lớp cho lớp "Học viên" (Student) trong JavaScript:



Lớp "Học viên" (Student) được biểu diễn bằng hình chữ nhật với tên lớp ở trên cùng. Lớp này có các thuộc tính như "tên" (name), "tuổi" (age), "địa chỉ" (address), "số điện thoại" (phoneNumber), "email", "điểm số" (scores). Ngoài ra, lớp "Học viên" còn có các phương thức như "thêm điểm" (addScore), "tính điểm trung bình" (calculateAverageScore),...

Trong sơ đồ lớp này, ta không thể thấy được mối quan hệ của lớp "Student" với các lớp khác trong hệ thống. Để hiểu rõ hơn về mối quan hệ giữa các lớp, ta cần xem xét toàn bộ hệ thống và vẽ sơ đồ lớp cho các lớp khác trong hệ thống.

**Câu 121 : Thuộc tính là gì? Nên dùng từ loại nào để biểu diễn thuộc tính?**

Thuộc tính (property) là một biến được liên kết với một đối tượng và lưu trữ thông tin về đối tượng đó. Thuộc tính có thể là một giá trị đơn như số, chuỗi hoặc boolean, hoặc là một đối tượng khác.

Trong lập trình JavaScript, ta có hai cách để biểu diễn thuộc tính:

1. Sử dụng biến bên trong đối tượng: Trong JavaScript, ta có thể định nghĩa các biến bên trong một đối tượng để biểu diễn thuộc tính của đối tượng. Ta có thể truy cập thuộc tính của đối tượng bằng cách sử dụng dấu chấm (.) sau tên đối tượng và trước tên thuộc tính. Ví dụ:

var student = {

name: "John",

age: 18,

grades: [8, 9, 7],

address: {

street: "123 Main St",

city: "New York"

}

};

console.log(student.name); *// Output: "John"*

console.log(student.grades[0]); *// Output: 8*

console.log(student.address.city); *// Output: "New York"*

1. Sử dụng phương thức getter và setter: Trong JavaScript, ta có thể sử dụng phương thức getter và setter để định nghĩa các thuộc tính của đối tượng. Phương thức getter được sử dụng để truy cập thuộc tính, trong khi phương thức setter được sử dụng để thiết lập giá trị cho thuộc tính. Ví dụ:

var student = {

name: "John",

age: 18,

grades: [8, 9, 7],

get averageGrade() {

var total = 0;

for (var i = 0; i < this.grades.length; i++) {

total += this.grades[i];

}

return total / this.grades.length;

},

set nickname(nick) {

this.name = nick;

}

};

console.log(student.averageGrade); *// Output: 8*

student.nickname = "Johnny";

console.log(student.name); *// Output: "Johnny"*

Trong ví dụ này, ta định nghĩa một đối tượng "student" với các thuộc tính là "name", "age" và "grades". Ta cũng định nghĩa hai phương thức getter và setter cho thuộc tính "averageGrade" và "nickname". Phương thức getter tính trung bình điểm của học sinh, trong khi phương thức setter thiết lập biến "name" của đối tượng "student" thành một biến đầu vào là "nick".

**Câu 122 : Phương thức là gì? Nên dùng từ loại nào để biểu diễn phương thức?**

hương thức (method) là một hành động được thực hiện bởi một đối tượng trong lập trình hướng đối tượng. Phương thức có thể thực hiện các hoạt động trên thuộc tính của đối tượng và/hoặc trả về kết quả.

Trong lập trình JavaScript, ta có thể biểu diễn phương thức bằng cách định nghĩa một hàm bên trong đối tượng. Cú pháp định nghĩa phương thức như sau:

var object = {

method: function() {

*// code here*

}

};

Ở đây, "object" là tên của đối tượng, "method" là tên của phương thức và "function()" là mã lệnh chứa các hành động của phương thức.

Ví dụ, ta có thể định nghĩa đối tượng "calculator" và các phương thức "add", "subtract", "multiply" và "divide" như sau:

javascriptCopy Code

var calculator = {

add: function(x, y) {

return x + y;

},

subtract: function(x, y) {

return x - y;

},

multiply: function(x, y) {

return x \* y;

},

divide: function(x, y) {

if (y === 0) {

throw new Error("Cannot divide by zero");

}

return x / y;

}

};

Trong ví dụ này, ta định nghĩa đối tượng "calculator" với bốn phương thức "add", "subtract", "multiply" và "divide" để thực hiện các phép tính cộng, trừ, nhân và chia. Các phương thức này được định nghĩa bằng cách sử dụng từ khóa "function()". Để gọi một phương thức của đối tượng, ta sử dụng dấu chấm "." theo sau tên đối tượng và tên phương thức, ví dụ như "calculator.add(2, 3)" để tính tổng hai số 2 và 3.

**Câu 123 : Cú pháp định nghĩa 1 class trong JavaScript?**

Trong JavaScript ES6, ta có thể sử dụng cú pháp class để định nghĩa một lớp (class) như trong các ngôn ngữ lập trình khác. Cú pháp định nghĩa một class trong JavaScript như sau:

javascriptCopy Code

class ClassName {

*// thuộc tính và phương thức của class*

}

Trong đó, "ClassName" là tên của lớp (class), và các thuộc tính và phương thức được định nghĩa bên trong dấu ngoặc nhọn " {} ".

Ví dụ: Ta có thể định nghĩa một lớp "Rectangle" với hai thuộc tính là chiều dài và chiều rộng, và hai phương thức là tính diện tích và chu vi như sau:

javascriptCopy Code

class Rectangle {

constructor(length, width) {

this.length = length;

this.width = width;

}

getArea() {

return this.length \* this.width;

}

getPerimeter() {

return 2 \* (this.length + this.width);

}

}

Trong ví dụ này, ta định nghĩa lớp "Rectangle" với hai thuộc tính là "length" và "width" được thiết lập bằng phương thức khởi tạo constructor(), và hai phương thức là "getArea()" và "getPerimeter()" để tính diện tích và chu vi của hình chữ nhật. Phương thức constructor() được gọi khi một đối tượng "Rectangle" được tạo ra, và phương thức này thiết lập giá trị cho các thuộc tính của đối tượng.

**Câu 124 : Cú pháp tạo 1 đối tượng từ class?**

Để tạo một đối tượng từ một lớp (class) trong JavaScript, ta sử dụng từ khóa new kết hợp với phương thức khởi tạo (constructor) của lớp đó. Cú pháp để tạo một đối tượng từ class như sau:

javascriptCopy Code

var objectName = new ClassName(arguments);

Trong đó, "objectName" là tên của đối tượng được tạo ra, "ClassName" là tên của lớp và "arguments" là các tham số truyền vào phương thức khởi tạo.

Ví dụ: Ta có lớp Person định nghĩa các thuộc tính name và age cùng với phương thức khởi tạo và phương thức introduce() để giới thiệu bản thân. Để tạo một đối tượng person1 từ lớp này, ta sử dụng cú pháp sau:

javascriptCopy Code

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

introduce() {

console.log(`My name is ${this.name} and I am ${this.age} years old.`);

}

}

var person1 = new Person('John', 30);

person1.introduce(); *// Output: "My name is John and I am 30 years old."*

Trong ví dụ này, ta đã tạo một đối tượng person1 từ lớp Person bằng cách sử dụng từ khóa new kết hợp với phương thức khởi tạo constructor và truyền vào các đối số "John" và 30. Sau đó, ta gọi phương thức introduce() để giới thiệu bản thân của đối tượng này.

**Câu 125 : Trình bày một số class có sẵn trong JavaScript mà em biết?**

JavaScript cung cấp một số lượng lớn các lớp (class) có sẵn (built-in classes) mà ta có thể sử dụng để giải quyết các vấn đề trong lập trình. Dưới đây là một số lớp có sẵn phổ biến trong JavaScript:

1. Array: Lớp Array đại diện cho một mảng các giá trị và cung cấp nhiều phương thức để xử lý các phần tử trong mảng, ví dụ như push(), pop(), shift(), unshift() và splice().
2. String: Lớp String đại diện cho một chuỗi ký tự và cung cấp nhiều phương thức để xử lý các chuỗi, ví dụ như charAt(), indexOf(), replace(), slice() và toUpperCase().
3. Number: Lớp Number đại diện cho một số và cung cấp nhiều phương thức để chuyển đổi và xử lý số, ví dụ như toFixed(), toPrecision(), toString() và valueOf().

**Câu 126 : Cú pháp truy cập đến 1 thuộc tính của đối tượng?**

Để truy cập đến một thuộc tính của đối tượng trong lập trình hướng đối tượng, ta sử dụng ký hiệu dấu chấm (.) giữa tên đối tượng và tên thuộc tính. Cú pháp như sau:

objectName.propertyName

Ví dụ, nếu ta có một đối tượng "học sinh" với thuộc tính "tên", ta có thể truy cập thuộc tính "tên" bằng cách sử dụng cú pháp như sau:

var student = {

name: "John",

age: 18,

grade: 12

};

console.log(student.name); *// Output: John*

Ở ví dụ trên, đối tượng "học sinh" được khai báo bằng cách tạo một biến "student" với ba thuộc tính là "name", "age" và "grade". Để truy cập thuộc tính "name" của đối tượng này, ta sử dụng cú pháp "[student.name](http://student.name/)".

**Câu 127 : Cú pháp truy cập đến 1 phương thức của đối tượng?**

Để truy cập đến một phương thức (method) của một đối tượng trong JavaScript, ta sử dụng dấu chấm "." (dot notation) để kết nối giữa tên đối tượng và tên phương thức. Cú pháp như sau:

javascriptCopy Code

objectName.methodName(arguments);

Trong đó, "objectName" là tên của đối tượng, "methodName" là tên của phương thức và "arguments" là các tham số truyền vào cho phương thức (nếu có).

Ví dụ: Ta có đối tượng "person" với hai thuộc tính "name" và "age", và một phương thức "introduce()" để giới thiệu bản thân. Để gọi phương thức "introduce()" của đối tượng này, ta sử dụng cú pháp sau:

javascriptCopy Code

var person = {

name: "John",

age: 30,

introduce: function() {

console.log("My name is " + this.name + " and I am " + this.age + " years old.");

}

};

person.introduce(); *// Output: "My name is John and I am 30 years old."*

Trong ví dụ này, ta sử dụng dấu chấm "." để kết nối giữa tên đối tượng "person" và tên phương thức "introduce()" nhằm gọi phương thức này. Kết quả trả về sẽ là chuỗi "My name is John and I am 30 years old." do phương thức "introduce()" in ra màn hình.

**Câu 128 : Từ khóa this dùng để làm gì ?**

Trong JavaScript, từ khóa this được sử dụng để truy cập đến đối tượng hiện tại (current object). Từ khóa this có thể được sử dụng bên trong một phương thức của một đối tượng để tham chiếu đến chính đối tượng đó.

Từ khóa this cho phép ta truy cập và thay đổi các thuộc tính và phương thức của đối tượng, hoặc truyền đối tượng này vào các hàm khác làm tham số. Khi gọi phương thức của đối tượng với từ khóa this, nó sẽ tham chiếu đến chính đối tượng đó để thực hiện các hoạt động liên quan đến đối tượng đó.

Ví dụ: Ta có đối tượng "person" với hai thuộc tính "name" và "age", và một phương thức "introduce()" để giới thiệu bản thân. Trong phương thức "introduce()", ta sử dụng từ khóa this để truy cập đến thuộc tính của đối tượng hiện tại.

javascriptCopy Code

var person = {

name: "John",

age: 30,

introduce: function() {

console.log("My name is " + this.name + " and I am " + this.age + " years old.");

}

};

person.introduce(); *// Output: "My name is John and I am 30 years old."*

Trong ví dụ này, ta sử dụng từ khóa this để truy cập đến thuộc tính "name" và "age" của đối tượng hiện tại trong phương thức "introduce()". Khi phương thức này được gọi, nó sẽ in ra câu giới thiệu bản thân chứa thông tin về tên và tuổi của đối tượng.

Từ khóa this trong JavaScript rất quan trọng và thường được sử dụng trong lập trình hướng đối tượng. Nó giúp ta truy cập và thao tác các thuộc tính và phương thức của đối tượng một cách dễ dàng và linh hoạt.

**Câu 129 : Trình bày về DOM (Document Object Model – Mô hình đối tượng tài liệu)?**

DOM (Document Object Model) là một mô hình đối tượng cho phép truy cập và thay đổi nội dung của các tài liệu HTML, XHTML và XML. DOM được cung cấp bởi trình duyệt web và được trình bày dưới dạng một cây đối tượng, trong đó mỗi nút đại diện cho một phần tử của tài liệu.

Các phần tử trong tài liệu được biểu diễn dưới dạng các nút trong cây đối tượng DOM, và các thuộc tính của phần tử được biểu diễn dưới dạng các thuộc tính của các đối tượng trong cây này. Nhờ vậy, ta có thể sử dụng JavaScript để truy cập và thao tác các phần tử của tài liệu như thêm, xóa, sửa đổi nội dung, kiểm tra hiển thị của phần tử... một cách dễ dàng.

Một số đặc điểm của DOM:

* DOM được trình bày dưới dạng một cây đối tượng, trong đó mỗi phần tử trong tài liệu được biểu diễn dưới dạng một nút.
* Mỗi nút đều có một loại (element node, text node...) và một số thuộc tính (id, class, style...) được xác định bởi các thuộc tính của phần tử trong tài liệu.
* DOM được hỗ trợ bởi các trình duyệt web thông qua API (Application Programming Interface) để cho phép truy cập và thay đổi nội dung của tài liệu HTML, XHTML và XML.
* Sử dụng JavaScript, ta có thể truy cập và thao tác các nút trong cây DOM bằng cách sử dụng các phương thức và thuộc tính được cung cấp bởi DOM API.

DOM là một công cụ quan trọng trong việc xây dựng các ứng dụng web động và tương tác với người dùng. Với DOM, ta có thể dễ dàng thao tác và thay đổi nội dung của tài liệu, giúp cho việc tạo ra các trang web linh hoạt và đáp ứng tốt với yêu cầu của người dùng.

**Câu 130 : Thẻ <canvas> dùng để làm gì?**

Thẻ <canvas> trong HTML được sử dụng để tạo ra các đồ họa và hoạt ảnh trong trang web. Nó cho phép tạo và điều khiển các hình ảnh, đồ thị, biểu đồ, các hiệu ứng động và nhiều hơn nữa bằng cách sử dụng JavaScript.

Để sử dụng thẻ <canvas>, ta cần định nghĩa kích thước của canvas thông qua thuộc tính width và height. Sau đó, ta có thể sử dụng JavaScript để tạo ra các hình ảnh và đồ thị trên canvas.

Ví dụ, ta có thể vẽ một hình chữ nhật đơn giản trên canvas bằng cách sử dụng JavaScript như sau:

htmlCopy Code

<canvas id="myCanvas" width="200" height="100"></canvas>

javascriptCopy Code

var canvas = document.getElementById("myCanvas");

var ctx = canvas.getContext("2d");

ctx.fillStyle = "#FF0000";

ctx.fillRect(0, 0, 150, 75);

Trong ví dụ này, ta đã định nghĩa một canvas với chiều rộng 200 và chiều cao 100 pixel. Sau đó, ta lấy đối tượng canvas đó thông qua phương thức getElementById() và sử dụng phương thức getContext() để lấy đối tượng vẽ trên canvas (CanvasRenderingContext2D). Cuối cùng, ta sử dụng phương thức fillRect() để vẽ một hình chữ nhật đỏ lên canvas.

Thẻ <canvas> là một công cụ mạnh mẽ trong việc tạo ra các đồ họa và hoạt ảnh trên trang web. Nó cho phép ta tạo ra các hiệu ứng động phong phú, tăng tính tương tác của trang web và cải thiện trải nghiệm người dùng.

**Câu 131 : Vẽ hình chữ nhật trên canvas?**

Để vẽ một hình chữ nhật trên canvas, ta sử dụng phương thức fillRect() hoặc strokeRect() của đối tượng CanvasRenderingContext2D. Cả hai phương thức này đều có cú pháp giống nhau và nhận vào 4 tham số, lần lượt là tọa độ x, y của góc trên bên trái của hình chữ nhật, chiều rộng và chiều cao của hình chữ nhật.

Ví dụ, để vẽ một hình chữ nhật đỏ với chiều rộng 100px và chiều cao 50px tại vị trí (10, 10), ta sẽ làm như sau:

htmlCopy Code

<canvas id="myCanvas" width="400" height="200"></canvas>

javascriptCopy Code

var canvas = document.getElementById("myCanvas");

var context = canvas.getContext("2d");

context.fillStyle = "red";

context.fillRect(10, 10, 100, 50);

Trong ví dụ này, đầu tiên ta lấy đối tượng canvas thông qua phương thức getElementById(). Tiếp theo, ta lấy đối tượng CanvasRenderingContext2D thông qua phương thức getContext("2d"). Sau đó, ta thiết lập màu fill cho hình chữ nhật bằng cách sử dụng thuộc tính fillStyle. Cuối cùng, ta sử dụng phương thức fillRect() để vẽ hình chữ nhật với tọa độ x, y là 10 và 10 và chiều rộng, cao lần lượt là 100px và 50px.

Ngoài ra, ta cũng có thể sử dụng phương thức strokeRect() để vẽ hình chữ nhật với viền. Cú pháp và tham số của strokeRect() giống với fillRect(). Ví dụ:

javascriptCopy Code

var canvas = document.getElementById("myCanvas");

var context = canvas.getContext("2d");

context.strokeStyle = "blue";

context.strokeRect(10, 10, 100, 50);

Trong ví dụ này, ta đã thiết lập màu viền cho hình chữ nhật bằng cách sử dụng thuộc tính strokeStyle và sử dụng phương thức strokeRect() để vẽ hình chữ nhật với viền màu xanh.

**Câu 132 : Vẽ hình tròn trên canvas?**

Để vẽ một hình tròn trên canvas, ta sử dụng phương thức arc() của đối tượng CanvasRenderingContext2D. Phương thức này nhận vào 6 tham số:

1. Tọa độ x của tâm hình tròn.
2. Tọa độ y của tâm hình tròn.
3. Bán kính của hình tròn.
4. Góc bắt đầu của hình tròn (theo radian).
5. Góc kết thúc của hình tròn (theo radian).
6. Hướng vẽ của hình tròn (chiều kim đồng hồ hoặc ngược chiều kim đồng hồ).

Ví dụ, để vẽ một hình tròn đỏ tại tọa độ (50, 50) và có bán kính là 30px, ta sẽ làm như sau:

htmlCopy Code

<canvas id="myCanvas" width="400" height="200"></canvas>

javascriptCopy Code

var canvas = document.getElementById("myCanvas");

var context = canvas.getContext("2d");

context.beginPath();

context.arc(50, 50, 30, 0, 2 \* Math.PI);

context.fillStyle = "red";

context.fill();

Trong ví dụ này, đầu tiên ta lấy đối tượng canvas thông qua phương thức getElementById(). Tiếp theo, ta lấy đối tượng CanvasRenderingContext2D thông qua phương thức getContext("2d"). Sau đó, ta bắt đầu một đường vẽ mới bằng phương thức beginPath(). Tiếp theo, ta sử dụng phương thức arc() để vẽ hình tròn với thông số tâm là (50, 50), bán kính là 30px và góc bắt đầu từ 0 đến 2π (toàn bộ hình tròn) và chiều kim đồng hồ. Cuối cùng, ta thiết lập màu fill cho hình tròn bằng cách sử dụng thuộc tính fillStyle và sử dụng phương thức fill() để điền màu đỏ vào hình tròn.

Lưu ý rằng, khi sử dụng phương thức arc(), ta cần phải bắt đầu một đường vẽ mới bằng phương thức beginPath() trước khi vẽ hình tròn.

**Câu 133 : Vẽ text trên canvas?**

Để vẽ chữ lên canvas, ta sử dụng phương thức fillText() hoặc strokeText() của đối tượng CanvasRenderingContext2D. Cả hai phương thức này đều có cú pháp giống nhau và nhận vào 3 tham số:

1. Chuỗi ký tự cần vẽ.
2. Tọa độ x của điểm bắt đầu của chuỗi.
3. Tọa độ y của điểm bắt đầu của chuỗi.

Ví dụ, để vẽ một chuỗi "Hello, world!" với font kích thước 20px, màu đỏ tại vị trí (50, 50), ta sẽ làm như sau:

htmlCopy Code

<canvas id="myCanvas" width="400" height="200"></canvas>

javascriptCopy Code

var canvas = document.getElementById("myCanvas");

var context = canvas.getContext("2d");

context.font = "20px Arial";

context.fillStyle = "red";

context.fillText("Hello, world!", 50, 50);

Trong ví dụ này, đầu tiên ta lấy đối tượng canvas thông qua phương thức getElementById(). Tiếp theo, ta lấy đối tượng CanvasRenderingContext2D thông qua phương thức getContext("2d"). Sau đó, ta thiết lập font cho văn bản bằng thuộc tính font, màu fill cho văn bản bằng cách sử dụng thuộc tính fillStyle. Cuối cùng, ta sử dụng phương thức fillText() để vẽ chuỗi "Hello, world!" tại tọa độ (50, 50).

Ngoài ra, ta cũng có thể sử dụng phương thức strokeText() để vẽ chữ với viền. Cú pháp và tham số của strokeText() giống với fillText(). Ví dụ:

javascriptCopy Code

var canvas = document.getElementById("myCanvas");

var context = canvas.getContext("2d");

context.font = "20px Arial";

context.strokeStyle = "blue";

context.strokeText("Hello, world!", 50, 50);

Trong ví dụ này, ta đã thiết lập màu viền cho văn bản bằng cách sử dụng thuộc tính strokeStyle và sử dụng phương thức strokeText() để vẽ chuỗi "Hello, world!" với viền màu xanh.

**Câu 134 : Nếu muốn di chuyển một hình tròn trên canvas cần làm gì?**

htmlCopy Code

<canvas id="myCanvas" width="400" height="200"></canvas>

javascriptCopy Code

var canvas = document.getElementById("myCanvas");

var context = canvas.getContext("2d");

*// Vẽ hình tròn ban đầu*

var x = 50;

var y = 50;

var radius = 30;

context.beginPath();

context.arc(x, y, radius, 0, 2 \* Math.PI);

context.fillStyle = "red";

context.fill();

*// Di chuyển hình tròn*

var targetX = 200;

var targetY = 100;

var speed = 3;

setInterval(function() {

*// Xóa hình tròn ở vị trí cũ*

context.clearRect(x - radius - 1, y - radius - 1, radius \* 2 + 2, radius \* 2 + 2);

*// Tính toán tọa độ mới*

var deltaX = targetX - x;

var deltaY = targetY - y;

var distance = Math.sqrt(deltaX \* deltaX + deltaY \* deltaY);

if (distance > speed) {

x += speed \* deltaX / distance;

y += speed \* deltaY / distance;

} else {

x = targetX;

y = targetY;

}

*// Vẽ lại hình tròn ở vị trí mới*

context.beginPath();

context.arc(x, y, radius, 0, 2 \* Math.PI);

context.fillStyle = "red";

context.fill();

}, 30);

Ở ví dụ này, ta đã vẽ hình tròn ban đầu tại vị trí (50, 50) và sau đó sử dụng phương thức setInterval() để di chuyển hình tròn tới vị trí (200, 100). Trong hàm di chuyển hình tròn, ta tính toán tọa độ mới của hình tròn bằng cách sử dụng một số công thức toán học và sử dụng phương thức clearRect() để xóa hình tròn ở vị trí cũ. Cuối cùng, ta sử dụng phương thức arc() để vẽ lại hình tròn ở vị trí mới.