**Câu hỏi**

1. **Hệ thống các kiến thức đã học ở Module1**
2. **Giải thuật/Thuật toán là gì?**

Thuật toán/giải thuật bao gồm các chỉ thị để giải quyết một vấn  
đề. Có thể sử dụng các cách khác nhau để mô tả thuật toán, các  
cách thông dụng là:

* Mã giả (pseudo-code)
* Lưu đồ (flowchart)
* Ngôn ngữ lập trình

1. **So sánh việc trình bày giải thuật bằng pseudo-code và flowchart?**

|  |  |  |
| --- | --- | --- |
|  | Pseudo-code | Flowchart |
|  | Liệt kê tuần tự các bước bằng ngôn ngữ tự nhiên để biểu diễn thuật toán | Lưu đồ mô tả giải thuật bằng các sơ đồ hình khối, mỗi khối qui  định một hành động. |
| Ưu điểm: | Đơn giản, không cần kiến thức về cách biểu diễn (lưu đồ, ngôn ngữ lập  trình) | Lưu đồ giúp người đọc dễ dàng hình dung được cấu trúc của thuật toán và hiểu được các bước thực hiện của nó |
| Nhược điểm: | - Dài dòng, không cấu trúc  - Đôi lúc khó hiểu, không diễn đạt được thuật toán | Cần kiến thức về cách biểu diễn |

1. **Tại sao lại cần trình bày giải thuật trước khi bắt tay vào code?**

Trình bày giải thuật trước khi bắt tay vào code giúp bạn hiểu rõ hơn về cách hoạt động của giải thuật, giúp tìm lỗi và điểm yếu của giải thuật trước khi code, và giúp trao đổi ý tưởng và thông tin với các thành viên trong nhóm.

1. **Tại sao lại nên sử dụng flowchart hơn pseudo-code khi thể hiện giải thuật của bài toán phức tạp?**

- Dễ hiểu hơn cho người mới học: Với những người mới học lập trình, việc đọc và hiểu flowchart sẽ dễ dàng hơn so với việc đọc và hiểu pseudo-code, bởi vì flowchart sử dụng biểu đồ và hình ảnh để thể hiện các quá trình trong giải thuật. Việc sử dụng hình ảnh giúp người mới học dễ dàng hình dung các bước thực hiện của giải thuật.

- Độ rõ ràng và cụ thể hơn: Flowchart sử dụng các hình ảnh, biểu đồ để biểu diễn quá trình thực hiện của giải thuật, từ đó giúp người đọc hiểu rõ ràng và cụ thể hơn về các bước thực hiện. Trong khi đó, pseudo-code sử dụng ngôn ngữ lập trình và các từ khóa, dễ gây nhầm lẫn và khó hiểu với người mới học.

- Giúp phát hiện lỗi dễ dàng hơn: Flowchart thể hiện các quá trình thực hiện của giải thuật bằng các hình ảnh và biểu đồ, từ đó giúp người đọc dễ dàng nhận ra các lỗi hoặc thiếu sót trong giải thuật. Nếu có lỗi, người đọc có thể dễ dàng tìm ra nguyên nhân và sửa chữa.

1. **Tại sao giải thuật lại cần hữu hạn các bước thực thi?**

Không thể thực hiện được: Nếu giải thuật không có hữu hạn các bước thực thi, máy tính hoặc người thực hiện công việc sẽ không thể thực hiện được giải thuật đó. Điều này có nghĩa là giải thuật không có giá trị thực tiễn và không thể được áp dụng để giải quyết các vấn đề thực tế.

Không thể kiểm soát: Nếu giải thuật không có hữu hạn các bước thực thi, không thể kiểm soát được quá trình thực hiện của nó. Điều này có thể dẫn đến việc không thể xác định được thời gian thực hiện và sử dụng tài nguyên như thế nào, gây ra lãng phí tài nguyên và hiệu suất kém.

Không thể tối ưu hóa: Nếu giải thuật không có hữu hạn các bước thực thi, không thể tối ưu hóa quá trình thực hiện của nó. Điều này có nghĩa là không thể tìm ra cách để giảm thiểu thời gian thực hiện, sử dụng ít tài nguyên hơn hoặc tăng hiệu suất của giải thuật.

Do đó, hữu hạn các bước thực thi là một yêu cầu cần thiết cho giải thuật để đảm bảo tính khả thi, kiểm soát được quá trình thực hiện và tối ưu hóa hiệu suất.

1. **Trình bày ý nghĩa các ký hiệu sử dụng khi vẽ flowchart?**

**-** Hình oval: Bắt đầu hay kết thúc chương trình.

**-** Hình chữ nhật: những bước tính toán

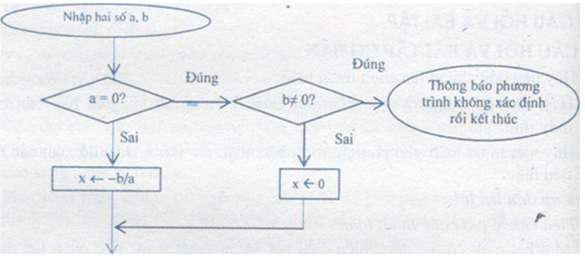
**-** Hình bình hành: các lệnh xuất hay nhập

**-** Hình thoi: các quyết định và rẽ nhánh

**-** Hình tròn: bộ nối hai phần trong chương trình

**-** Mũi tên: dòng chảy.

1. **Vẽ giải thuật giải phương trình bậc nhất?**

****

1. **Bạn biết gì về VCS – Version Control System (hệ thống quản lý phiên bản)?**

Version Control System (VCS) – Hệ thống quản lý phiên bản mã nguồn

là một phần mềm hỗ trợ:

- Khôi phục lại phiên bản cũ của các file

- Khôi phục lại phiên bản cũ của toàn bộ dự án

- Xem lại các thay đổi đã được thực hiện theo thời gian

- Xem ai là người thực hiện thay đổi cuối cùng có thể gây ra sự cố

- Khôi phục lại các file vô tình xoá mất

Các công cụ thông dụng:

• Subversion

• Git

• Mercurial

• Bazaar

• CVS

(Version Control System (VCS), hay còn gọi là hệ thống quản lý phiên bản, là một công cụ giúp quản lý và theo dõi sự thay đổi của các tệp tin và dự án phần mềm trong quá trình phát triển. Nó được sử dụng để lưu trữ, quản lý, và theo dõi các phiên bản của mã nguồn, tài liệu, hình ảnh, video và các tài nguyên khác liên quan đến một dự án phần mềm.

VCS cho phép các nhà phát triển làm việc cùng nhau trên cùng một dự án mà không gây ra xung đột vì các phiên bản khác nhau của các tệp tin được lưu trữ và theo dõi. Nó cũng cung cấp khả năng khôi phục lại phiên bản trước đó của mã nguồn hoặc dự án, trong trường hợp xảy ra lỗi hoặc sự cố.) - chatGPT

1. **Phân biệt hệ thống quản lý mã nguồn tập trung và hệ thống quản lý mã nguồn phân tán?**

Hệ thống quản lý mã nguồn tập trung (Centralized Version Control System - CVCS) là một loại VCS mà toàn bộ mã nguồn và lịch sử thay đổi được **lưu trữ trên một máy chủ tập trung**. Nhà phát triển sử dụng các phiên bản của mã nguồn từ máy chủ và có thể gửi các thay đổi của mình trở lại máy chủ để lưu trữ. Một số ví dụ về hệ thống quản lý mã nguồn tập trung bao gồm Subversion (SVN), Concurrent Versions System (CVS) và Perforce.

Trong khi đó, hệ thống quản lý mã nguồn phân tán (Distributed Version Control System - DVCS) cho phép các nhà phát triển tạo và **lưu trữ các bản sao độc lập** của kho mã nguồn trên các máy tính cá nhân của mình. Các bản sao này được đồng bộ hóa với nhau và có thể được chia sẻ với các nhà phát triển khác. Khi cần, các bản sao này có thể được đẩy lên máy chủ để chia sẻ với những người khác. Một số ví dụ về hệ thống quản lý mã nguồn phân tán bao gồm Git, Mercurial và Bitbucket.

Trong CVCS, việc truy cập mã nguồn phải được thực hiện thông qua máy chủ tập trung, điều này có thể dẫn đến một số vấn đề như hệ thống không thể sử dụng nếu máy chủ không hoạt động hoặc kết nối mạng bị gián đoạn. Tuy nhiên, CVCS có thể dễ dàng quản lý và theo dõi các thay đổi trên toàn bộ dự án.

Trong khi đó, DVCS cho phép các nhà phát triển làm việc độc lập và phát triển ở bất kỳ đâu, tăng tính linh hoạt và độc lập của các nhà phát triển. Việc sử dụng DVCS cũng giúp tránh được một số vấn đề về đồng bộ hóa và xung đột thay đổi giữa các nhà phát triển.

1. **Git là gì? Tại sao cần sử dụng git?**

Git là một hệ thống điều khiển phiên bản (version control system)

theo hình thức phân tán

Git được sử dụng để quản lý mã nguồn (source code) và ghi nhận các

thay đổi

1. **Git thuộc hệ thống quản lý mã nguồn tập trung hay phân tán?**
2. **Repository là gì? Phân biệt LocalRepository và RemoteRepository?**

Thường được gọi ngắn gọn là repo

• Repository là nơi chứa toàn bộ mã nguồn

• Repository bao gồm toàn bộ các file và lịch sử của các file đó

• Repository chứa tất cả các commit

• Có 2 loại repository:

• Local Repository: Ở trên máy của lập trình viên

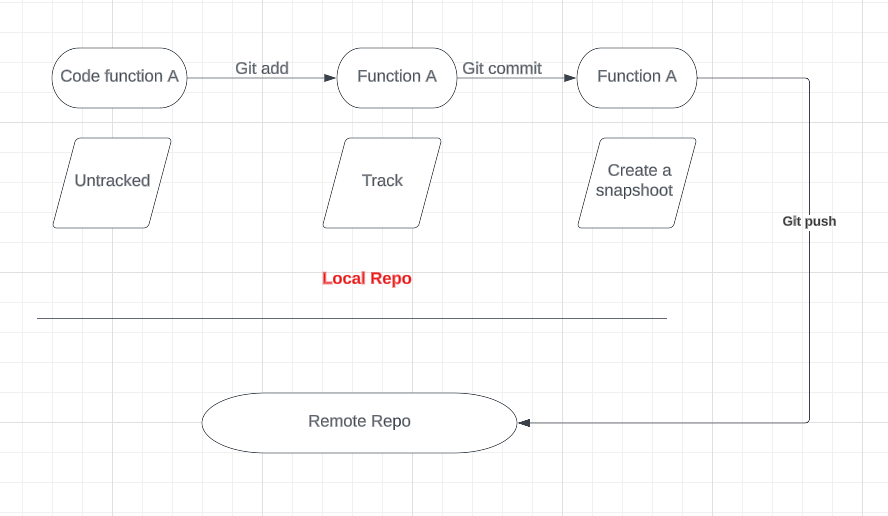
• Remote Repository: Ở trên một máy chủ chia sẻ (chẳng hạn như GitHub)  
Local Repository (Kho lưu trữ cục bộ): Là một bản sao đầy đủ của kho mã nguồn được lưu trữ trên máy tính cá nhân của một cá nhân hoặc một nhóm nhỏ các nhà phát triển. Local Repository được tạo ra khi một nhà phát triển sao chép toàn bộ kho mã nguồn từ Remote Repository xuống máy tính của mình. Khi thực hiện các thay đổi trên mã nguồn, các nhà phát triển thường làm việc trực tiếp trên Local Repository của mình và sau đó đẩy các thay đổi lên Remote Repository.

Remote Repository (Kho lưu trữ từ xa): Là một kho mã nguồn được lưu trữ trên một máy chủ từ xa. Đây là nơi chính thức lưu trữ mã nguồn của dự án và là nơi mà các nhà phát triển chia sẻ và đồng bộ hóa các thay đổi của mình. Remote Repository cung cấp khả năng cho các nhà phát triển làm việc độc lập, tạo ra các nhánh (branch) để phát triển đồng thời và gộp lại các thay đổi.

1. **Đồng bộ giữa các repository (local và remote) để làm gì?**

Quá trình đồng bộ này cần thiết để đảm bảo rằng tất cả các nhà phát triển đều làm việc trên phiên bản mới nhất của mã nguồn và có thể tiếp tục phát triển dự án một cách đồng bộ: chia sẻ dự án và phục hồi dự án, quản lý hiệu quả

1. **Thao tác push của git để làm gì? Cú pháp câu lệnh?**

****



**$git push**: Đưa các thay đổi từ local repository lên remote repository

1. **Thao tác pull của git để làm gì?**

Cập nhật mã nguồn từ một Remote Repository về Local Repository

1. **Thao tác clone của git để làm gì? Cú pháp câu lệnh?**

Sao chép một Remote Repository về máy của lập trình viên

$git clone <remote path>

1. **Nêu tên và ý nghĩa các câu lệnh cơ bản đã được học để làm việc với git?**

git clone: Sao chép một repository

git init: Khởi tạo một repository

git add: Đưa các file vào trong vùng staged

git commit: Ghi nhận các thay đổi

git push: Đưa các thay đổi từ local repository lên remote repo

git pull: Cập nhật mã nguồn từ một Remote Repository về Local Repository

1. **www là gì?**

**WWW** là viết tắt của cụm từ **World Wide Web**, dịch sang tiếng Việt là mạng lưới toàn cầu (gọi đơn giản là Web). Đây là một không gian thông tin chứa các tài liệu và nguồn tài nguyên khác của [website](https://vietnix.vn/website-la-gi/), được xác định bởi [URL](https://vietnix.vn/url-la-gi/) và liên kết với nhau bởi các siêu liên kết cũng như truy cập thông qua [Internet](https://vietnix.vn/internet-la-gi/). Hiểu một cách đơn giản hơn, WWW là thuật ngữ đề cập tới tất cả các trang [HTML](https://vietnix.vn/the-input-trong-html/) kết nối với nhau và có thể truy cập thông qua Internet. WWW là một tập hợp các trang web được tìm thấy trên mạng máy tính và trình duyệt web cần sử dụng internet để truy cập.

1. **HTML là gì?**

• HTML là ngôn ngữ được sử dụng để tạo ra các Webpage

• HTML là viết tắt của chữ Hypertext Markup

Language (Ngôn ngữ Đánh dấu Siêu Văn bản)

• Đánh dấu: sử dụng các thẻ

• Siêu văn bản: có thể đi đến văn bản khác thông qua

các liên kết (link)

1. **Nêu tên các thẻ làm việc với văn bản trong tài liệu html?**

**<h1-h6>, <p>, <span>**

1. **Thẻ div dùng để làm gì? <div> khác gì <span>**

Thẻ <div> được sử dụng để tạo các khối (block) trên trang web, có thể chứa nhiều phần tử khác nhau bên trong. Khi sử dụng thẻ <div>, ta có thể định dạng toàn bộ nội dung bên trong theo một kiểu nhất định, áp dụng các thuộc tính CSS cho toàn bộ phần tử, hoặc sử dụng JavaScript để thao tác với toàn bộ phần tử đó.

Thẻ <span>, åh tay, được sử dụng để định dạng các phần tử văn bản nhỏ hơn, chỉ chứa một số từ hoặc một phần của một từ trong một đoạn văn bản. Thẻ <span> không tạo ra một khối mới trên trang web và thường được sử dụng để áp dụng các thuộc tính CSS cho một phần của một đoạn văn bản.

1. **URL là gì? Phân biệt url tuyệt đối và url tương đối?**

URL là viết tắt của "Uniform Resource Locator" (định danh tài nguyên thống nhất), là một chuỗi ký tự được sử dụng để định danh một tài nguyên trên mạng Internet. URL cung cấp thông tin về giao thức truy cập, tên miền hoặc địa chỉ IP của máy chủ, đường dẫn đến tài nguyên cụ thể, và các tham số khác để truy cập tài nguyên đó.

URL được chia thành hai loại chính: URL tuyệt đối và URL tương đối.

URL tuyệt đối: Đây là URL hoàn chỉnh, chứa đầy đủ thông tin cần thiết để truy cập một tài nguyên trên mạng. Nó bao gồm tên miền hoặc địa chỉ IP của máy chủ, giao thức truy cập và đường dẫn đến tài nguyên cụ thể. Ví dụ: https://www.example.com/index.html là một URL tuyệt đối.

URL tương đối: Đây là URL được sử dụng để truy cập tài nguyên từ một vị trí tương đối trên mạng. Nó không chứa đầy đủ thông tin về giao thức, tên miền hoặc địa chỉ IP của máy chủ. Thay vào đó, nó chỉ cung cấp đường dẫn tới tài nguyên cần truy cập từ vị trí hiện tại của trình duyệt. URL tương đối thường được sử dụng để tạo liên kết nội bộ trong trang web. Ví dụ: nếu bạn đang ở trang web https://www.example.com/index.html và muốn liên kết đến một tài nguyên khác tại cùng một thư mục, bạn có thể sử dụng URL tương đối như "file2.html".

Tóm lại, URL là một chuỗi ký tự được sử dụng để định danh một tài nguyên trên mạng. URL tuyệt đối chứa đầy đủ thông tin để truy cập một tài nguyên, trong khi URL tương đối chỉ cung cấp đường dẫn tới tài nguyên từ vị trí hiện tại của trình duyệt.

1. **Siêu liên kết là gì?**

Siêu liên kết (Hyperlink) là một kỹ thuật trong HTML, cho phép tạo liên kết từ một trang web đến một trang web hoặc một tài nguyên khác trên mạng. Khi người dùng nhấp chuột vào siêu liên kết, trình duyệt sẽ chuyển hướng đến địa chỉ được liên kết.

Để tạo một siêu liên kết trong HTML, ta sử dụng thẻ <a> và cung cấp đường dẫn đến tài nguyên cần liên kết trong thuộc tính href.

1. **Muốn liên kết đến một phần trên cùng trang thì làm thế nào?**

Để liên kết đến một phần cụ thể trên cùng một trang web, ta sử dụng siêu liên kết kèm với thuộc tính **href** chứa giá trị của định danh phần tử (element ID) mà ta muốn liên kết tới.

Các bước thực hiện như sau:

Tạo một phần tử với định danh (ID) duy nhất tương ứng với phần trên trang mà bạn muốn liên kết đến. Ví dụ: <h2 id="section1">Phần đầu trang</h2>.

Tạo siêu liên kết với thuộc tính href chứa giá trị định danh của phần tử tương ứng. Ví dụ: <a href="#section1">Chuyển đến phần đầu trang</a>.

1. **Muốn liên kết đến một phần trên trang khác thì làm thế nào?**

Để liên kết đến một phần trên một trang khác, ta sử dụng siêu liên kết kèm với đường dẫn tuyệt đối hoặc tương đối đến trang đó, và cũng phải cung cấp định danh phần tử tương ứng.

Các bước thực hiện như sau:

Tạo một phần tử với định danh (ID) duy nhất tương ứng với phần trên trang cần liên kết đến.

Tạo siêu liên kết đến trang web cần liên kết đến, kèm theo đường dẫn tương đối hoặc tuyệt đối đến trang đó và cung cấp định danh phần tử tương ứng. Ví dụ: <a href="https://example.com/page2.html#section1">Chuyển đến phần đầu trang của trang khác</a>.

1. **Phân biệt internal link và external link?**

Internal link (liên kết nội bộ) là một loại liên kết trên trang web, mà khi được nhấp vào, sẽ dẫn đến một trang khác trong cùng một trang web. Ví dụ, khi bạn nhấp vào một liên kết trong trang chủ của một trang web để điều hướng đến một trang khác trong trang web đó, đó là một liên kết nội bộ.

External link (liên kết bên ngoài) là một loại liên kết trên trang web, mà khi được nhấp vào, sẽ dẫn đến một trang khác trên một trang web khác. Ví dụ, khi bạn nhấp vào một liên kết trên trang web của một công ty để truy cập vào trang web của đối tác, đó là một liên kết bên ngoài.

Sự khác biệt chính giữa liên kết nội bộ và liên kết bên ngoài là đích của liên kết. Liên kết nội bộ dẫn đến một trang khác trong cùng một trang web, trong khi liên kết bên ngoài dẫn đến một trang khác trên một trang web khác.

1. **Có mấy loại danh sách?**

Thẻ <ul> (unordered list - danh sách không có thứ tự): Thẻ này được sử dụng để tạo danh sách không có thứ tự, mà không có số hay chữ để chỉ thứ tự. Mỗi phần tử trong danh sách được đóng gói trong thẻ <li> (list item - mục trong danh sách).

Ví dụ:

<ul>

<li>Item 1</li>

<li>Item 2</li>

<li>Item 3</li>

</ul>

Thẻ <ol> (ordered list - danh sách có thứ tự): Thẻ này được sử dụng để tạo danh sách có thứ tự, mà có số hay chữ để chỉ thứ tự. Mỗi phần tử trong danh sách được đóng gói trong thẻ <li>.

Ví dụ:

<ol>

<li>First item</li>

<li>Second item</li>

<li>Third item</li>

</ol>

Thẻ <dl> (definition list - danh sách định nghĩa): Thẻ này được sử dụng để tạo danh sách định nghĩa, mà các thuật ngữ được định nghĩa trong thẻ <dt> (definition term - thuật ngữ định nghĩa), và các miêu tả định nghĩa được định nghĩa trong thẻ <dd> (definition description - miêu tả định nghĩa).

Ví dụ:

<dl>

<dt>HTML</dt>

<dd>HyperText Markup Language - ngôn ngữ đánh dấu siêu văn bản</dd>

<dt>CSS</dt>

<dd>Cascading Style Sheets - bộ khung dựng kiểu</dd>

</dl>

Ngoài ra, còn có thẻ <menu> được sử dụng để tạo danh sách menu, và thẻ <dir> để tạo danh sách thư mục, tuy nhiên cả hai thẻ này đã bị loại bỏ khỏi HTML5.

1. **Muốn hiển thị danh sách có thứ tự bắt đầu C. D. E… thì làm thế nào?**

<ol type="A" start="3">

<li>Coffee</li>

<li>Tea</li>

<li>Milk</li>

</ol>

1. **Form hay biểu mẫu dùng để làm gì?**

Biểu mẫu (form) là cơ chế cho phép người dùng nhập dữ liệu và gửi về server

• Chẳng hạn, chúng ta có form đăng nhập, form khảo sát, form tìm kiếm

1. **Thuộc tính action của form dùng để làm gì?**

Thuộc tính action có giá trị là một URL, quy định nơi dữ liệu được gửi đến

1. **Phân biệt 2 method là get và post?**

|  |  |
| --- | --- |
| **GET** | **POST** |
| Dữ liệu được gửi đi sẽ hiển thị trên thanh địa chỉ của trình duyệt | Dữ liệu được gửi đi KHÔNG hiển thị trên thanh địa  chỉ của trình duyệt |
| Không nên sử dụng để gửi các dữ liệu nhạy cảm(chẳng hạn như mật khẩu) | Được sử dụng để gửi các dữ liệu nhạy cảm |
| Thường được sử dụng để gửi những dữ liệu nhỏ | Không hạn chế dung lượng dữ liệu gửi đi. Thường được sử dụng để gửi dữ liệu lớn lên server. |

1. **Phân biệt các nút bấm có type là: button, submit, reset?**

<button> thường được sử dụng để tạo các nút tùy chỉnh và xử lý sự kiện bằng JavaScript.

<input type="submit"> được sử dụng để gửi form đến một URL được chỉ định.

<input type="reset"> được sử dụng để xóa giá trị của các trường trong form.

1. **Muốn submit form đến trang CodeGym.vn thì làm thế nào?**

Để submit form đến trang CodeGym.vn, bạn cần tạo một form trong HTML và thiết lập thuộc tính action của form đó để trỏ tới URL của trang CodeGym.vn.

Ví dụ: nếu bạn muốn submit form đến trang CodeGym.vn để đăng ký tài khoản:

<form action="https://codegym.vn/dang-ky-tai-khoan" method="post">

1. **Nếu muốn nối 3 ô trên cùng 1 hàng thì làm thế nào?**

Để nối 3 ô trong cùng 1 hàng của bảng HTML, ta sử dụng thuộc tính **colspan** của thẻ td hoặc th.

<table>

<tr>

<td colspan="3">Nội dung của ô 1, ô 2 và ô 3</td>

</tr>

<tr>

<td>Nội dung của ô 4</td>

<td>Nội dung của ô 5</td>

<td>Nội dung của ô 6</td>

</tr>

</table>

1. **Nếu muốn nối 2 ô trên cùng 1 cột thì làm thế nào?**

Để nối 2 ô trong cùng 1 cột của bảng HTML, ta sử dụng thuộc tính rowspan của thẻ td hoặc th.

<table>

<tr>

<td>Nội dung của ô 1</td>

<td rowspan="2">Nội dung của ô 2 và ô 3</td>

</tr>

<tr>

<td>Nội dung của ô 4</td>

</tr>

</table>

1. **Phân biệt thuộc tính cellpadding và thuộc tính cellspacing?**

cellpadding: thuộc tính này được sử dụng để thêm khoảng cách giữa nội dung của ô bảng và viền của ô đó.

cellspacing: thuộc tính này được sử dụng để thêm khoảng cách giữa các ô bảng.

<table cellpadding="10" cellspacing="5">

<tr>

<td>Ô bảng 1</td>

<td>Ô bảng 2</td>

</tr>

<tr>

<td>Ô bảng 3</td>

<td>Ô bảng 4</td>

</tr>

</table>

Ở ví dụ trên, khoảng cách giữa nội dung của ô bảng và viền của ô đó là 10px (cellpadding="10"), và khoảng cách giữa các ô bảng là 5px (cellspacing="5").

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

• Viết mã Javascript bên trong thẻ <script></script>

• Sử dụng file .js và nhúng vào trang web

• Sử dụng file .js từ bên ngoài

• Viết trực tiếp mã JavaScript trong thẻ html

1. **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?**
2. **Phát biểu “JavaScript phân biệt chữ hoa chữ thường ” đúng hay sai?**

**Đúng**

Trong JavaScript, chữ hoa và chữ thường được coi là khác nhau, vì vậy việc viết hoa hay viết thường trong tên biến, hàm, hoặc thuộc tính sẽ ảnh hưởng đến cách sử dụng của chúng trong mã. Ví dụ, biến "myVariable" và "MyVariable" là hai biến khác nhau trong JavaScript.

1. **Trình bày về quy tắc CamelCase và camelCase?**

Quy tắc CamelCase và camelCase là hai cách đặt tên biến trong lập trình. Cả hai cách đều kết hợp các từ bằng cách viết hoa chữ cái đầu của mỗi từ và xóa khoảng trắng. Sự khác biệt giữa hai cách là:

• Quy tắc CamelCase viết hoa chữ cái đầu của từ đầu tiên, ví dụ: UserLoginCount, CategoryClass, ThisIsMyVariable... Quy tắc này thường được sử dụng để đặt tên cho các lớp (class) trong nhiều ngôn ngữ lập trình.https://topdev.vn/blog/quy-chuan-dat-ten-trong-lap-trinh-camelcase-underscore-hay-pascalcase/https://levanphu.info/quy-chuan-dat-ten-ham-ten-bien-trong-lap-trinh-camelcase-underscore-hay-pascalcase

• Quy tắc camelCase viết thường chữ cái đầu của từ đầu tiên, ví dụ: userLoginCount, categoryClass, thisIsMyVariable... Quy tắc này thường được sử dụng để đặt tên cho các biến và phương thức (method) trong nhiều ngôn ngữ lập trình

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

Biến là một tên gọi được gắn cho một vùng nhớ chứa dữ liệu  
• Dữ liệu được lưu trữ trong vùng nhớ của biến được gọi là giá trị  
(value)

Biến cục bộ (local variable) là biến được khai báo trong phạm vi của một hàm hoặc một khối lệnh, chỉ có thể truy cập được từ bên trong phạm vi đó. Các biến cục bộ được tạo ra khi chương trình thực hiện hàm hoặc khối lệnh, và bị hủy khi hàm hoặc khối lệnh đó kết thúc.

Biến toàn cục (global variable) là biến được khai báo ở bên ngoài tất cả các hàm và khối lệnh, có thể truy cập được từ mọi nơi trong chương trình. Các biến toàn cục được tạo ra khi chương trình bắt đầu chạy và tồn tại cho đến khi chương trình kết thúc.

1. **Phân biệt: var, let, const?**

**var**: Là từ khóa được sử dụng để khai báo biến trong Javascript trước khi có sự xuất hiện của let và const. Biến được khai báo bằng var có phạm vi sử dụng là toàn cục hoặc phạm vi của hàm. Điều này có nghĩa là nếu biến được khai báo bên trong một hàm, nó chỉ có thể truy cập được từ bên trong hàm đó. Nếu biến được khai báo bên ngoài hàm, nó có thể truy cập được từ bất kỳ đâu trong chương trình. Biến được khai báo bằng var có thể gán lại giá trị mới bất cứ khi nào.

**>> Tái sử dụng vùng nhớ và phạm vi global**

**let**: Là từ khóa được sử dụng để khai báo biến có phạm vi sử dụng là phạm vi của khối lệnh. Biến được khai báo bằng let chỉ có thể truy cập được từ bên trong khối lệnh đó hoặc các khối lệnh con bên trong nó. Biến được khai báo bằng let cũng có thể gán lại giá trị mới bất cứ khi nào.

**>> Sinh ra vùng nhớ mới khi khai báo lại ở scope khác, phạm vi trong scope/block**

**const**: Là từ khóa được sử dụng để khai báo một hằng số. Biến được khai báo bằng const không thể gán lại giá trị mới sau khi được khởi tạo. Chúng ta có thể khởi tạo giá trị cho biến const khi khai báo hoặc trong một hàm khởi tạo, và giá trị này sẽ không thay đổi trong suốt quá trình chạy chương trình. Biến được khai báo bằng const cũng có phạm vi sử dụng là phạm vi của khối lệnh, tương tự như let.

**>> Phải khởi tạo giá trị ban đầu và không được gán lại**

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

JavaScript là một ngôn ngữ định kiểu yếu, có nghĩa là không cho phép người lập trình xác định kiểu biến. Một biến có thể lưu trữ bất kỳ kiểu dữ liệu nào trong thời gian chạy và các phép toán sẽ giả định kiểu của biến. Kết quả cũng có thể bị ép chuyển thành một kiểu dữ liệu khác. Ví dụ: một phép toán có thể trả về kết quả là chuỗi "5" thay vì số 5. Điều này có thể dẫn đến những sai lầm vô tình khi viết mã và lỗi trong mã do có lỗi về kiểu loại.

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

Number, String, Boolean, Null, Undefined, Kiểu ký tự đơn (Symbol)

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

Trả về một chuỗi thể hiện kiểu dữ liệu của biến hoặc giá trị đó:

typeof(x); //“number”

1. **Trình bày các cách đưa thông báo trong JavaScript?**

Alert: Alert là hộp thoại thông báo đơn giản hiển thị một tin nhắn cho người dùng. Cú pháp:

alert("Thông báo của bạn");

Prompt: Prompt là hộp thoại yêu cầu người dùng nhập dữ liệu vào. Cú pháp:

let result = prompt("Nhập thông tin của bạn", "Giá trị mặc định");

Trong đó, "Nhập thông tin của bạn" là thông báo hiển thị trên hộp thoại, "Giá trị mặc định" là giá trị mặc định để người dùng nhập vào (không bắt buộc).

Confirm: Confirm là hộp thoại yêu cầu người dùng xác nhận hoặc từ chối một tác vụ. Cú pháp:

let result = confirm("Bạn có chắc chắn muốn thực hiện tác vụ này?");

Trong đó, "Bạn có chắc chắn muốn thực hiện tác vụ này?" là thông báo hiển thị trên hộp thoại. Khi người dùng chọn "OK", kết quả trả về là true, và khi người dùng chọn "Cancel", kết quả trả về là false.

Console.log: Console.log là một phương thức trong đối tượng console, được sử dụng để hiển thị thông báo trên cửa sổ console của trình duyệt. Cú pháp:

console.log("Thông báo của bạn");

1. **Trình bày về các loại toán tử trong JavaScript?**

Toán tử số học: Bao gồm các toán tử + (cộng), - (trừ), \* (nhân), / (chia), % (chia lấy dư), ++ (tăng lên 1 đơn vị) và -- (giảm đi 1 đơn vị).

Toán tử gán: Bao gồm các toán tử = (gán), += (cộng và gán), -= (trừ và gán), \*= (nhân và gán), /= (chia và gán) và %= (chia lấy dư và gán).

Toán tử so sánh: Bao gồm các toán tử == (bằng), != (không bằng), === (tuyệt đối bằng), !== (tuyệt đối không bằng), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng) và <= (nhỏ hơn hoặc bằng).

Toán tử logic: Bao gồm các toán tử && (và), || (hoặc) và ! (phủ định).

Toán tử ba ngôi: Là toán tử duy nhất trong JavaScript có 3 toán hạng, bao gồm toán tử ? : (nếu đúng thì trả về giá trị đầu tiên, nếu sai thì trả về giá trị thứ hai).

Toán tử chuỗi: Bao gồm toán tử + (nối chuỗi).

Toán tử bit: Bao gồm các toán tử & (và bit), | (hoặc bit), ^ (hoặc biến đổi bit) và ~ (đảo bit).

Toán tử thao tác trên chuỗi: Bao gồm toán tử [] (truy xuất phần tử của chuỗi).

Ngoài ra, JavaScript còn có một số toán tử khác như toán tử typeof (xác định kiểu dữ liệu), toán tử instanceof (kiểm tra xem một đối tượng có phải là một thể hiện của một lớp hay không), toán tử delete (xóa một thuộc tính của một đối tượng) và toán tử new (tạo một đối tượng mới).

1. **Phân biệt giữa 2 toán tử == và ===?**

So sánh bằng giá trị và kiểu dữ liệu

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

Toán tử **&&** (và) sẽ trả về giá trị đầu tiên (false) nếu giá trị đó chứa false hoặc falsy.

console.log(false && true); // false

console.log(true && false); // false

console.log(5 && null); // null

Toán tử **||** (hoặc) sẽ trả về giá trị đầu tiên (true) nếu giá trị đó chứa true hoặc truthy.

console.log(true || true); // true

console.log(false || true); // true

console.log(true || false); // true

console.log(false || false); // false

console.log(0 || 1); // 1

console.log(1 || 0); // 1

console.log(1 || 2); // 1

console.log(null || 5); // 5

console.log(5 || null); // 5

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

**true**

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

Trong JavaScript, toán tử ++ và -- được dùng để tăng hoặc giảm giá trị của một biến số lên một đơn vị.

Toán tử ++ sẽ tăng giá trị của biến số lên một đơn vị và trả về giá trị mới của biến số đó. Toán tử -- sẽ giảm giá trị của biến số đi một đơn vị và trả về giá trị mới của biến số đó.

Cả hai toán tử này có thể được sử dụng trước hoặc sau biến số. Khi sử dụng trước biến số (ví dụ: ++i), giá trị của biến số sẽ được tăng lên trước khi giá trị mới được sử dụng trong biểu thức. Khi sử dụng sau biến số (ví dụ: i++), giá trị của biến số sẽ được sử dụng trong biểu thức trước khi tăng lên.

let x = 5;

let y = ++x;

console.log(x); // 6

console.log(y); // 6

let a = 10;

let b = a--;

console.log(a); // 9

console.log(b); // 10

let c = 2;

console.log(c++); // 2

console.log(c); // 3

console.log(++c); // 4

Toán tử - có thể được sử dụng để đảo ngược giá trị của một số. Nó sẽ trả về giá trị âm của số nếu số ban đầu là dương và ngược lại.

1. **Phân biệt ++a và a++?**
2. **Câu lệnh điều kiện/rẽ nhánh là gì?**

Câu lệnh điều kiện còn được gọi là câu lệnh ra quyết định (decision

making)

• Cho phép thay đổi luồng thực thi của chương trình

• Lựa chọn thực thi một khối lệnh dựa trên việc đánh giá một điều kiện

cho trước

• JavaScript hỗ trợ các câu lệnh điều kiện:

• if

• switch-case

1. **Có những loại câu lệnh điều kiện nào?**

If, switch-case

1. **Trình bày về câu lệnh if?**

Câu lệnh if là một câu lệnh điều kiện (conditional statement) trong JavaScript, được sử dụng để kiểm tra một điều kiện nào đó, nếu điều kiện đó đúng thì thực hiện một khối lệnh nào đó, nếu không thì bỏ qua.

1. **Trình bày về câu lệnh switch?**

switch-case là một cấu trúc điều kiện cho phép lựa chọn thực thi các khối lệnh khác nhau dựa trên kết quả của việc so sánh.

• switch-case so sánh giá trị của một biến với lần lượt từng giá trị một, nếu có giá trị phù hợp với biến thì khối lệnh tương ứng sẽ được thực thi

• switch-case không thể thay thế if-else trong tất cả các trường hợp

1. **Tại sao lại nói else là một tùy chọn (optional) của câu lệnh if?**

Câu lệnh else không phải là bắt buộc khi sử dụng câu lệnh if. Nếu không có câu lệnh else, thì khối lệnh trong câu lệnh if chỉ được thực hiện khi biểu thức là đúng, và không có khối lệnh nào được thực hiện khi biểu thức là sai.

1. **Khi nào nên sử dụng câu lệnh switch?**

Ta nên sử dụng câu lệnh switch khi có một biến hoặc biểu thức cần được kiểm tra đối với nhiều giá trị khác nhau và thực hiện các hành động tương ứng với từng giá trị. Một số ví dụ bao gồm:

Kiểm tra giá trị của một biến trạng thái (status) và thực hiện các hành động tương ứng với từng trạng thái, ví dụ như "mở" hoặc "đóng".

Kiểm tra giá trị của một biến tháng (month) và thực hiện các hành động tương ứng với từng tháng, ví dụ như hiển thị số ngày trong tháng đó.

Kiểm tra giá trị của một biến hướng (direction) và thực hiện các hành động tương ứng với từng hướng, ví dụ như di chuyển đối tượng sang trái, phải, lên hoặc xuống.

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

Giá trị của biểu thức sẽ được so sánh với từng trường hợp (case)

• Nếu có trường hợp bằng nhau thì khối lệnh tương ứng sẽ được thực thi

• Nếu gặp câu lệnh break thì sẽ kết thúc thực thi khối switch-case

• Nếu gặp trường hợp bằng nhau, nhưng sau đó

không có câu lệnh break thì tất cả những khối lệnh phía sau cũng được thực thi

1. **Mệnh đề default có bắt buộc trong câu lệnh switch?**

**Không,**

Nếu không có trường hợp nào bằng nhau thì khối lệnh trong default (nếu có) sẽ được thực thi.

1. **Trình bày về toán tử 3 ngôi?**

Biểu thức điều kiện đánh giá một biểu thức dựa vào một điều kiện cho trước

**condition** **?** **value1** **:** **value2**

Trong đó, condition là một biểu thức kiểm tra điều kiện, value1 là giá trị được trả về nếu condition đúng, và value2 là giá trị được trả về nếu condition sai.

1. **Trình bày về sự kiện và trình quản lý sự kiện trong javascript?**

Sự kiện (event) trong JavaScript là hành động của người dùng hoặc của trình duyệt, chẳng hạn như nhấp chuột, di chuột, nhấn phím, tải trang, và các hành động khác. Các sự kiện có thể được gọi là "trigger" (kích hoạt) để thực hiện một hành động nào đó trong trang web.

Trình quản lý sự kiện (event handler) là một hàm được gọi khi một sự kiện xảy ra trên một phần tử HTML. Trình quản lý sự kiện có thể được thêm vào bằng cách sử dụng các phương thức của đối tượng DOM như addEventListener() hoặc onclick.

1. **Sự kiện onchange phát sinh khi nào?**

Sự kiện onchange trong JavaScript phát sinh khi giá trị của một phần tửHTML thay đổi và mất trạng thái trỏ tới. Điều này xảy ra khi người dùng thay đổi giá trị của một phần tử, ví dụ như khi họ chọn một tùy chọn khác trong một danh sách thả xuống hoặc nhập một giá trị mới vào một trường văn bản.

1. **Sự kiện onclick phát sinh khi nào?**

Sự kiện onclick trong lập trình web phát sinh khi người dùng nhấn chuột vào một phần tử HTML như một nút bấm (<button>, <input type="button">,...), một liên kết (<a>), một hình ảnh (<img>), hoặc bất kỳ phần tử nào có thuộc tính onclick.

1. **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 một hộp thoại xác nhận với người dùng khi bấm vào nút xóa, chúng ta có thể sử dụng sự kiện onclick của nút đó và hàm confirm() của JavaScript.

1. **Phân biệt onmousevover và onmouseout, onmousemove?**

onmouseover: Sự kiện này được kích hoạt khi con trỏ chuột của người dùng di chuyển vào phần tử HTML. Nó cho phép xác định hành động nào sẽ được thực hiện khi con trỏ chuột được di chuyển vào một phần tử nào đó. Ví dụ: hiển thị một thông báo khi di chuột vào một liên kết hoặc thay đổi màu sắc của phần tử khi con trỏ chuột vào.

onmouseout: Sự kiện này được kích hoạt khi con trỏ chuột của người dùng di chuyển ra khỏi phần tử HTML. Nó cho phép xác định hành động nào sẽ được thực hiện khi con trỏ chuột di chuyển ra khỏi một phần tử nào đó. Ví dụ: ẩn thông báo khi di chuột ra khỏi một liên kết hoặc khôi phục màu sắc ban đầu của phần tử khi con trỏ chuột di chuyển ra khỏi.

onmousemove: Sự kiện này được kích hoạt khi con trỏ chuột di chuyển trên một phần tử HTML. Nó cho phép xác định hành động nào sẽ được thực hiện khi con trỏ chuột di chuyển trên một phần tử nào đó. Ví dụ: thay đổi vị trí của một hình ảnh khi con trỏ chuột di chuyển trên trang hoặc hiển thị thông tin chi tiết về một sản phẩm khi con trỏ chuột di chuyển trên hình ảnh sản phẩm.

1. **Phân biệt onkeyup, onkeydown và onkeypress?**

onkeyup: Sự kiện này được kích hoạt khi người dùng thả phím bàn phím mà họ đang giữ. Nó cho phép xác định hành động nào sẽ được thực hiện sau khi phím được thả ra. Ví dụ: tạo một chức năng tìm kiếm tức thời khi người dùng thả phím Enter.

onkeydown: Sự kiện này được kích hoạt khi người dùng nhấn phím bàn phím. Nó cho phép xác định hành động nào sẽ được thực hiện khi phím được nhấn xuống. Ví dụ: thay đổi màu sắc của một phần tử khi người dùng nhấn phím mũi tên.

onkeypress: Sự kiện này được kích hoạt khi người dùng nhấn phím bàn phím và giữ phím đó trong một khoảng thời gian nhất định. Nó cho phép xác định hành động nào sẽ được thực hiện khi người dùng nhấn phím và giữ phím đó trong một khoảng thời gian. Ví dụ: thực hiện tìm kiếm trên trang web khi người dùng nhập từ khóa vào một ô tìm kiếm.

1. **Phân biệt onfocus và onblur?**

onfocus: Sự kiện này được kích hoạt khi một phần tử HTML trở thành phần tử đang được chọn. Điều này có thể xảy ra khi người dùng nhấp chuột vào phần tử đó hoặc sử dụng phím tab để điều hướng đến phần tử đó. Sự kiện onfocus cho phép thực hiện các hành động cụ thể khi một phần tử được chọn. Ví dụ: hiển thị một tooltip hoặc thay đổi màu sắc của phần tử đó.

onblur: Sự kiện này được kích hoạt khi một phần tử HTML không còn là phần tử được chọn nữa. Điều này có thể xảy ra khi người dùng nhấp chuột vào một phần tử khác hoặc sử dụng phím tab để điều hướng đến một phần tử khác. Sự kiện onblur cho phép thực hiện các hành động cụ thể khi một phần tử không còn được chọn. Ví dụ: kiểm tra tính hợp lệ của dữ liệu trong phần tử đó trước khi người dùng điều hướng đến phần tử khác.

1. **Sự kiện onload phát sinh khi nào?**

Sự kiện onload trong lập trình web xảy ra khi một trang web hoặc một phần tử trên trang web đã hoàn thành việc tải dữ liệu và hiển thị trên trình duyệt của người dùng.

1. **Phân biệt innerHTML và innerText?**

Cả hai thuộc tính innerHTML và innerText đều là thuộc tính của các phần tử HTML và cho phép truy cập và thay đổi nội dung của phần tử đó.

Tuy nhiên, có một số khác biệt giữa chúng:

innerHTML: là thuộc tính được sử dụng để lấy hoặc đặt HTML bên trong một phần tử. Khi sử dụng innerHTML, nội dung được trả về là chuỗi chứa các thẻ HTML, bao gồm cả văn bản và các thẻ HTML khác.

Ví dụ:

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

Javascript code

var div = document.getElementById("myDiv");

console.log(div.innerHTML); // "This is some <strong>bold</strong> text."

innerText: là thuộc tính được sử dụng để lấy hoặc đặt văn bản bên trong một phần tử. Khi sử dụng innerText, nội dung được trả về là chuỗi chỉ chứa văn bản, các thẻ HTML không được bao gồm.

Ví dụ:

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

Javascript code

var div = document.getElementById("myDiv");

console.log(div.innerText); // "This is some bold text."

Vì innerHTML chứa cả các thẻ HTML, nên việc sử dụng nó có thể gây ra rủi ro bảo mật khi chèn nội dung không an toàn. Trong khi đó, innerText được coi là an toàn hơn và thường được sử dụng trong các trường hợp cần lấy hoặc đặt văn bản trong một phần tử.

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

Để vô hiệu hóa (disabled) một button khi click vào một checkbox, ta cần sử dụng JavaScript để bắt sự kiện click của checkbox và sau đó cập nhật thuộc tính disabled của button.

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

<button id="myButton">My Button</button>

<script>

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

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

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

if (checkbox.checked) {

button.disabled = true;

} else {

button.disabled = false;

}

});

</script>

1. **Phân biệt thuộc tính readonly và disabled của html?**

Thuộc tính readonly và disabled đều là thuộc tính của các phần tử HTML để kiểm soát trạng thái hoạt động của phần tử đó, nhưng có sự khác biệt như sau:

readonly: là thuộc tính cho phép người dùng chỉ xem dữ liệu của phần tử mà không thể chỉnh sửa giá trị của nó. Tức là người dùng không thể nhập giá trị vào phần tử, nhưng vẫn có thể sao chép hoặc cắt dữ liệu ra khỏi phần tử.

Ví dụ:

<input type="text" readonly value="This input is readonly">

disabled: là thuộc tính cho phép người dùng không thể tương tác với phần tử đó. Khi phần tử này được vô hiệu hóa, người dùng không thể nhập giá trị vào phần tử, không thể chọn hoặc tương tác với phần tử đó. Ngoài ra, các giá trị của phần tử cũng không được gửi đi khi gửi form.

Ví dụ:

<input type="text" disabled value="This input is disabled">

Vì vậy, readonly được sử dụng để ngăn người dùng chỉnh sửa giá trị của phần tử, trong khi disabled được sử dụng để ngăn người dùng tương tác hoàn toàn với phần tử đó.

1. **Vòng lặp là gì?**

Vòng lặp (Loop) là một cấu trúc điều khiển trong lập trình cho phép lặp đi lặp lại một khối mã một số lần hoặc đến khi một điều kiện được đáp ứng. Vòng lặp được sử dụng để giảm độ lặp lại của mã trong chương trình.

1. **Nêu các loại vòng lặp cơ bản trong JavaScript?**
2. **Phân biệt vòng lặp for,while,do…while?**

Có ba loại vòng lặp thông dụng trong các ngôn ngữ lập trình, bao gồm:

Vòng lặp for: Vòng lặp này được sử dụng khi số lần lặp cụ thể được biết trước. Vòng lặp for có ba phần chính là: khởi tạo biến đếm, điều kiện lặp và bước nhảy.

Ví dụ:

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

console.log(i);

}

Vòng lặp while: Vòng lặp này được sử dụng khi số lần lặp chưa biết trước và lặp cho đến khi một điều kiện được đáp ứng.

Ví dụ:

let i = 0;

while (i < 10) {

console.log(i);

i++;

}

Vòng lặp do-while: Tương tự như vòng lặp while, nhưng nó sẽ luôn được thực hiện ít nhất một lần trước khi kiểm tra điều kiện để lặp tiếp.

Ví dụ:

let i = 0;

do {

console.log(i);

i++;

} while (i < 10);

Mỗi loại vòng lặp đều có những ưu điểm và hạn chế riêng và phù hợp với từng trường hợp sử dụng khác nhau.

1. **Khi nào nên sử dụng vòng lặp do…while?**

Vòng lặp do...while trong các ngôn ngữ lập trình là một loại vòng lặp điều khiển lặp lại một khối mã lệnh ít nhất một lần trước khi kiểm tra điều kiện để xem liệu vòng lặp có nên tiếp tục thực hiện hay không. Vì vòng lặp do...while luôn thực hiện ít nhất một lần, nó thường được sử dụng để thực hiện một số công việc ban đầu trước khi kiểm tra điều kiện.

Các trường hợp nên sử dụng vòng lặp do...while bao gồm:

Cần thực hiện một khối lệnh ít nhất một lần trước khi kiểm tra điều kiện: Ví dụ, việc hiển thị một menu cho người dùng chọn sẽ cần phải thực hiện ít nhất một lần trước khi kiểm tra xem người dùng đã chọn một tùy chọn hay chưa.

Cần lặp lại một tác vụ đến khi điều kiện được đáp ứng: Ví dụ, yêu cầu người dùng nhập một giá trị hợp lệ từ bàn phím, vòng lặp do...while sẽ thực hiện việc yêu cầu nhập giá trị và kiểm tra giá trị đó đến khi người dùng nhập giá trị hợp lệ.

Tuy nhiên, vòng lặp do...while không phải là lựa chọn tốt nhất trong mọi trường hợp. Khi điều kiện lặp lại không cần thiết được kiểm tra trước khi vòng lặp bắt đầu, thì việc sử dụng vòng lặp while hoặc for sẽ là một lựa chọn tốt hơn.

1. **Khi nào nên sử dụng vòng lặp for?**

Vòng lặp for trong các ngôn ngữ lập trình là một loại vòng lặp điều khiển, được sử dụng để lặp lại một khối mã lệnh một số lần cụ thể, được xác định trước bởi một biểu thức số học. Các trường hợp nên sử dụng vòng lặp for bao gồm:

Lặp lại một số lần cụ thể: Vòng lặp for thường được sử dụng khi bạn biết trước số lần lặp cần thực hiện, ví dụ như lặp lại một khối mã lệnh cho mỗi phần tử trong một mảng.

Lặp lại theo một chuỗi giá trị cụ thể: Nếu bạn muốn lặp lại một khối mã lệnh với các giá trị tăng dần hoặc giảm dần, vòng lặp for cũng là một lựa chọn tốt.

Lặp lại các thao tác phức tạp: Vòng lặp for có thể được sử dụng để lặp lại các thao tác phức tạp hơn, ví dụ như tính toán các phép toán số học phức tạp hoặc đọc và ghi dữ liệu vào các tệp.

Ngoài ra, vòng lặp for còn cho phép bạn tùy chỉnh các bước lặp lại và giá trị ban đầu của biến điều khiển, điều này giúp bạn dễ dàng kiểm soát quá trình lặp lại và điều chỉnh theo nhu cầu của mình.

1. **Khi nào nên sử dụng vòng lặp while?**

Vòng lặp while trong các ngôn ngữ lập trình là một loại vòng lặp điều khiển, được sử dụng để lặp lại một khối mã lệnh cho đến khi một điều kiện nhất định được đáp ứng. Các trường hợp nên sử dụng vòng lặp while bao gồm:

Lặp lại cho đến khi một điều kiện được đáp ứng: Vòng lặp while thường được sử dụng khi bạn không biết trước số lần lặp lại cần thực hiện, mà chỉ biết điều kiện để dừng lại. Ví dụ, bạn có thể sử dụng vòng lặp while để đọc các dòng từ tệp đến khi đọc hết tệp.

Lặp lại cho đến khi người dùng nhập đúng dữ liệu: Vòng lặp while cũng thường được sử dụng để yêu cầu người dùng nhập dữ liệu từ bàn phím và lặp lại cho đến khi dữ liệu đó hợp lệ.

Lặp lại khi tình huống thay đổi động: Nếu điều kiện để dừng vòng lặp phụ thuộc vào các tình huống động, ví dụ như việc lắng nghe sự kiện từ chuột hoặc bàn phím, vòng lặp while cũng là lựa chọn tốt để lặp lại cho đến khi tình huống đó xảy ra.

Tuy nhiên, khi số lần lặp lại cần biết trước, vòng lặp for là lựa chọn tốt hơn. Và khi bạn cần thực hiện một khối lệnh ít nhất một lần trước khi kiểm tra điều kiện, vòng lặp do...while là lựa chọn phù hợp hơn.

1. **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?**

Nên sử dụng vòng lặp do…while: thực hiện hành động ít nhất 1 lần trước khi kiểm tra điều kiện

let number;

do {

number = parseInt(prompt("Enter an even number:"));

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

console.log("The even number you entered is: " + number);

1. **Yêu cầu người dùng tính tổng các số chẵn từ 10 đến 100 thì nên dùng vòng lặp loại nào?**

Nên dùng for:

let sum = 0;

for (let i = 10; i <= 100; i += 2) {

sum += i;

}

console.log("Tổng các số chẵn từ 10 đến 100 là: " + sum);

1. **Trình bày về vòng for thiếu?**
2. **Nếu muốn hiển thị các cặp số nguyên dương có tổng là 10 thì viết thế nào?**

for i in range(1, 10):

for j in range(1, 10):

if i + j == 10:

print(i, j)

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

let n = 5; // số dòng của hình tam giác

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

let row = '';

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

row += '\*';

}

console.log(row);

}

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

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

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

// code để xử lý tại mỗi vòng lặp

}

}

Duyệt giá trị i từ 0 đến n-1. Với mỗi giá trị i sẽ duyệt giá trị j từ 0 đến m -1

1. **Phân biệt giữa câu lệnh break và continue?**

**break**: Dừng vòng lặp và thoát khỏi vòng lặp ngay lập tức.

Câu lệnh break được sử dụng để kết thúc vòng lặp ngay lập tức khi một điều kiện được thỏa mãn. Nó thường được sử dụng để ngắt vòng lặp khi một điều kiện nhất định được đáp ứng hoặc khi đã đạt đến một điểm dừng nào đó.

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

if (i === 5) {

break;

}

console.log(i);

}

//1

//2

//3

//4

**continue**: Bỏ qua phần còn lại của lần lặp hiện tại và bắt đầu vòng lặp mới.

Câu lệnh continue được sử dụng để bỏ qua phần còn lại của lần lặp hiện tại và bắt đầu vòng lặp mới. Nó thường được sử dụng để bỏ qua các phần tử không cần thiết trong một mảng hoặc để loại bỏ các giá trị không hợp lệ.

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

if (i === 5) {

continue;

}

console.log(i);

}

1

2

3

4

6

7

8

9

10

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

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

if (i % 5 === 0) {

continue;

}

console.log(i);

}

Để hiển thị các số không chia hết cho 5 từ 1 đến 100 và sử dụng câu lệnh continue, ta có thể sử dụng vòng lặp for kết hợp với câu lệnh if để kiểm tra xem số hiện tại có chia hết cho 5 hay không. Nếu số hiện tại chia hết cho 5, ta sẽ sử dụng câu lệnh continue để bỏ qua số đó và tiếp tục vòng lặp. Nếu không, ta sẽ in ra số đó ra màn hình.

1. **Mảng là gì?**

Mảng là một loại biến đặc biệt, có thể lưu được nhiều giá trị thay vì chỉ một giá trị như các biến thông thường

•Mỗi giá trị trong mảng được gọi là một phần tử

•Các phần tử được lưu trữ ở các vị trí kế tiếp nhau trong bộ nhớ

1. **Trình bày cú pháp khai báo mảng trong JavaScript?**

- **C1**: let myArray = [item1, item2, ..., itemN];

- **C2**: var array = new Array[item1, itèm, …, itemN];

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

Chỉ số của mảng là một số nguyên không âm được sử dụng để truy cập các phần tử trong mảng. Chỉ số của mảng bắt đầu từ 0 trong hầu hết các ngôn ngữ lập trình, bao gồm JavaScript.

1. **Kích thước của mảng là gì?**

Trong JavaScript, kích thước của một mảng được xác định bằng thuộc tính "length". Thuộc tính này trả về số lượng phần tử trong mảng, và có thể được sử dụng để lấy kích thước của mảng.

1. **Phân biệt push() và unshift()?**

push(): Phương thức này được sử dụng để thêm một phần tử vào cuối mảng. Nghĩa là, phần tử được thêm vào vị trí sau cùng của mảng.

unshift(): Phương thức này được sử dụng để thêm một phần tử vào đầu mảng. Nghĩa là, phần tử được thêm vào vị trí đầu tiên của mảng.

1. **Phân biệt push() và pop()?**

push(): Phương thức này được sử dụng để thêm một phần tử vào cuối mảng. Nghĩa là, phần tử được thêm vào vị trí sau cùng của mảng.

pop(): Phương thức này được sử dụng để xóa phần tử cuối cùng của mảng. Nghĩa là, phần tử cuối cùng sẽ được xóa khỏi mảng và trả về giá trị của phần tử đó.

1. **Phân biệt shift() và unshift()?**

unshift(): Phương thức này được sử dụng để thêm một phần tử vào đầu mảng. Nghĩa là, phần tử được thêm vào vị trí đầu tiên của mảng.

shift(): Phương thức này được sử dụng để xóa phần tử đầu tiên của mảng. Nghĩa là, phần tử đầu tiên sẽ được xóa khỏi mảng và trả về giá trị của phần tử đó.

1. **Nếu muốn xóa phần tử cuối cùng của mảng dùng hàm nào?**

let arr = [1, 2, 3];

arr.pop();

console.log(arr); // đầu ra: [1, 2]

**Dùng phương thức .pop();**

1. **Nếu xóa phần tử đầu tiên của mảng dùng hàm nào?**

shift(): Phương thức này được sử dụng để xóa phần tử đầu tiên của mảng.

let arr = [1, 2, 3];

let removed = arr.shift();

console.log(removed); // đầu ra: 1

console.log(arr); // đầu ra: [2, 3]

1. **Muốn sắp xếp mảng thì dùng hàm nào?**

Dùng hàm sort();

Vd1: sắp xếp số

var numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5];

numbers.sort(function(a, b) {

return a - b;

});

Vd2: sắp xếp chuỗi

var fruits = ['banana', 'apple', 'orange', 'kiwi'];

fruits.sort();

1. **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();

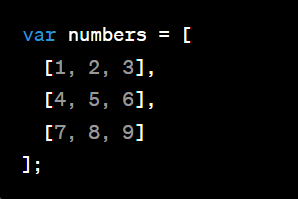
1. **Thao tác duyệt mảng là gì? Triển khai thao tác bằng code?**

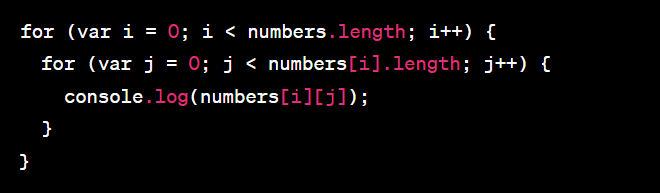
Thao tác duyệt mảng là quá trình truy cập và xử lý từng phần tử trong mảng một cách tuần tự.

Để duyệt mảng trong JavaScript, ta có thể sử dụng các vòng lặp như for, while, do...while, hoặc sử dụng các phương thức của mảng như forEach(), map(), filter().

1. **Thao tác duyệt mảng 2 chiều?**

Sử dụng 2 vòng lặp lồng nhau để duyệt

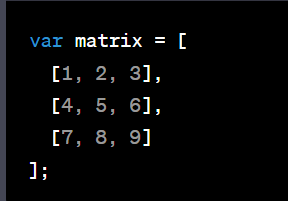


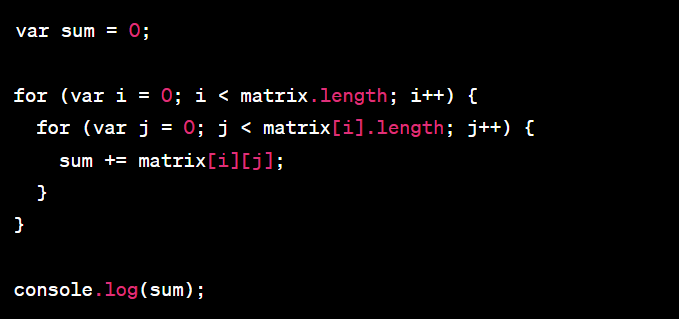


1. **Mảng 2 chiều cần mấy chỉ số để xác định duy nhất 1 phần tử?**

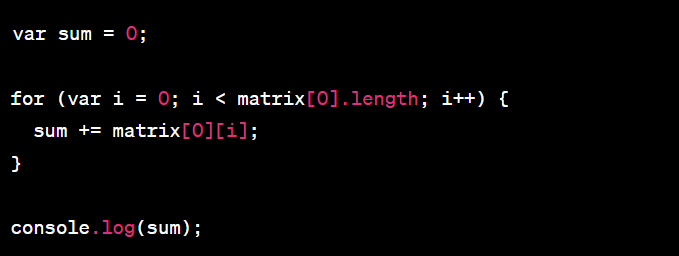
**Cần 2 chỉ số**

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

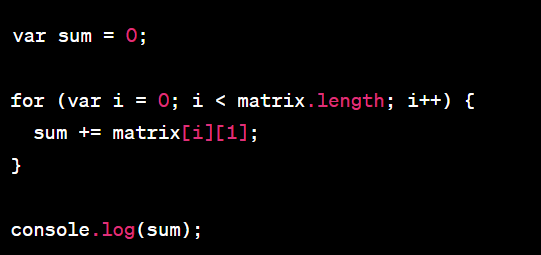
****

****

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

****

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

****

1. **Hàm (function) là gì?**

Hàm (function) là một nhóm các câu lệnh thực hiện một nhiệm vụ nhất định

• Hàm là thuật ngữ được sử dụng phổ biến trong Lập trình hướng đối tượng. Trong nhiều trường hợp khác, các tên gọi được sử dụng là phương thức (method) và thủ tục (procedure)

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

Tham số hình thức (formal arguments) là các tham số được định nghĩa trong khai báo hàm. Chúng là các biến được sử dụng để nhận giá trị được truyền vào từ tham số thực. Thông thường, tham số hình thức được khai báo trong cặp dấu ngoặc đơn của hàm và được liệt kê theo thứ tự, mỗi tham số cách nhau bởi dấu phẩy.

Tham số thực (actual arguments) là giá trị được truyền vào cho tham số hình thức khi gọi hàm. Chúng được truyền vào trong cặp dấu ngoặc đơn khi gọi hàm và phải tuân theo thứ tự và kiểu dữ liệu của các tham số hình thức.

-----------GPT---------

Tham số (còn được gọi đầy đủ là tham số hình thức – formal parameter) là các biến được khai báo trong phần header

• Khi gọi hàm thì giá trị của các biến này sẽ được truyền vào. Các giá trị này được gọi là tham số thực (actual parameter) hoặc đối số (argument)

1. **Gọi hàm (invoke) là gì?**

Gọi (call hoặc invoke) phương thức là cách để thực thi một hàm đã được định nghĩa trước đó:

• Khi gọi hàm thì cần truyền đối số vào

• Ví dụ, gọi hàm không có giá trị trả về:

console.log("Welcome to Java!");

• Ví dụ, gọi hàm có giá trị trả về:

var larger = max(3, 4);

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

Trong lập trình, hàm gọi (calling function) là một hàm được sử dụng để gọi một hàm khác, trong khi hàm được gọi (called function) là hàm được gọi bởi hàm gọi.

def called\_function(x):

return x \*\* 2

def calling\_function(y):

result = called\_function(y)

return result + 1

print(calling\_function(3))

1. **Phân biệt truyền tham trị và truyền tham chiếu?**

Trong Javascript, khi truyền đối số vào hàm, truyền tham trị (pass by value) và truyền tham chiếu (pass by reference) cũng được sử dụng tương tự như trong các ngôn ngữ lập trình khác.

Tuy nhiên, cách thức truyền tham chiếu trong Javascript có thể gây nhầm lẫn. Trong Javascript, các kiểu dữ liệu cơ bản (như string, number, boolean) được truyền theo cơ chế tham trị, còn các đối tượng (objects và arrays) được truyền theo cơ chế tham chiếu.

Ví dụ về truyền tham trị:

function square(x) {

x = x \*\* 2;

return x;

}

let num = 5;

console.log(square(num)); // 25

console.log(num); // 5

Ví dụ về truyền tham chiếu:

function appendZero(arr) {

arr.push(0);

}

let myArr = [1, 2, 3];

appendZero(myArr);

console.log(myArr); // [1, 2, 3, 0]

Trong ví dụ trên, biến myArr là một đối tượng (array), nên khi được truyền vào hàm appendZero, nó sẽ được truyền theo cơ chế tham chiếu. Hàm appendZero thêm số 0 vào cuối của mảng myArr, và mảng myArr được thay đổi bởi hàm.

Tuy nhiên, khi gán một đối tượng cho một biến khác, ví dụ như:

let arr1 = [1, 2, 3];

let arr2 = arr1;

arr2.push(4);

console.log(arr1); // [1, 2, 3, 4]

console.log(arr2); // [1, 2, 3, 4]

Thì các biến arr1 và arr2 đều trỏ đến cùng một vùng nhớ của mảng [1, 2, 3]. Khi thay đổi mảng thông qua biến arr2, mảng cũng thay đổi, do đó arr1 cũng trả về mảng [1, 2, 3, 4].

1. **Câu lệnh return dùng để làm gì?**

câu lệnh return được sử dụng để trả về một giá trị từ một hàm. Khi hàm được gọi và thực thi, nó sẽ thực hiện các lệnh trong hàm, và nếu gặp câu lệnh return, nó sẽ trả về giá trị tương ứng và kết thúc việc thực thi hàm.

1. **Có được phép viết “return;”?**

Có, trong JavaScript, ta có thể sử dụng câu lệnh return mà không cần trả về giá trị nào.

1. **Phân biệt return và break?**

**return**: Dùng để trả về một giá trị từ một hàm. Khi gặp lệnh return trong một hàm, hàm sẽ ngay lập tức dừng việc thực thi và trả về giá trị tương ứng.

**break**: Dùng để dừng vòng lặp đang thực thi và thoát khỏi vòng lặp đó. Khi gặp lệnh break trong một vòng lặp, vòng lặp sẽ dừng việc thực thi ngay lập tức và thoát khỏi vòng lặp.

1. **Hàm thực hiện việc kiểm tra nên trả về giá trị không? Nếu có, nên trả về gì?**

Hàm kiểm tra thường được thiết kế để kiểm tra điều kiện nào đó và trả về giá trị true hoặc false tùy thuộc vào kết quả của điều kiện. Do đó, hàm kiểm tra thường trả về giá trị boolean, có thể là true hoặc false.

Ví dụ, hàm kiểm tra xem một số có phải là số nguyên tố hay không có thể trả về giá trị boolean true nếu số đó là số nguyên tố và false nếu không phải.

Tuy nhiên, trong một số trường hợp đặc biệt, hàm kiểm tra cũng có thể trả về giá trị khác, chẳng hạn như một mã lỗi hoặc một thông báo lỗi để thông báo cho người dùng biết rằng việc kiểm tra đã thất bại và có lỗi xảy ra.

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

Hàm hoán vị `swap(a,b)` thực hiện hoán vị giá trị của hai biến a và b bằng cách tạo ra một bản sao của giá trị của biến a và gán giá trị của biến b cho biến a, sau đó gán giá trị của bản sao của giá trị của biến a cho biến b.

Vì vậy, khi chúng ta gọi hàm `swap(a,b)`, chúng ta thực sự đang hoán vị giá trị của hai biến a và b trong phạm vi của hàm. Tuy nhiên, giá trị của hai biến này không thực sự được hoán vị trong phạm vi của chương trình chứa hàm `swap(a,b)`.

Để thực sự hoán vị giá trị của hai biến trong phạm vi chương trình gốc, chúng ta cần gọi hàm `swap(a,b)` và truyền các biến a và b vào đó để hoán vị giá trị của chúng. Sau đó, khi hàm được thực thi, giá trị của các biến a và b sẽ được hoán vị trong phạm vi của chương trình gốc.

1. **Trình bày về OOP?**

OOP (Object-Oriented Programming) là một phương pháp lập trình phần mềm, tập trung vào việc thiết kế các đối tượng (objects) để thực hiện các chức năng trong chương trình. OOP được xây dựng trên ba nguyên tắc cơ bản:

1. Tính đóng gói (Encapsulation): Đóng gói dữ liệu và mã lệnh vào các đối tượng riêng biệt, giúp hạn chế tác động từ bên ngoài và bảo vệ tính riêng tư của dữ liệu.

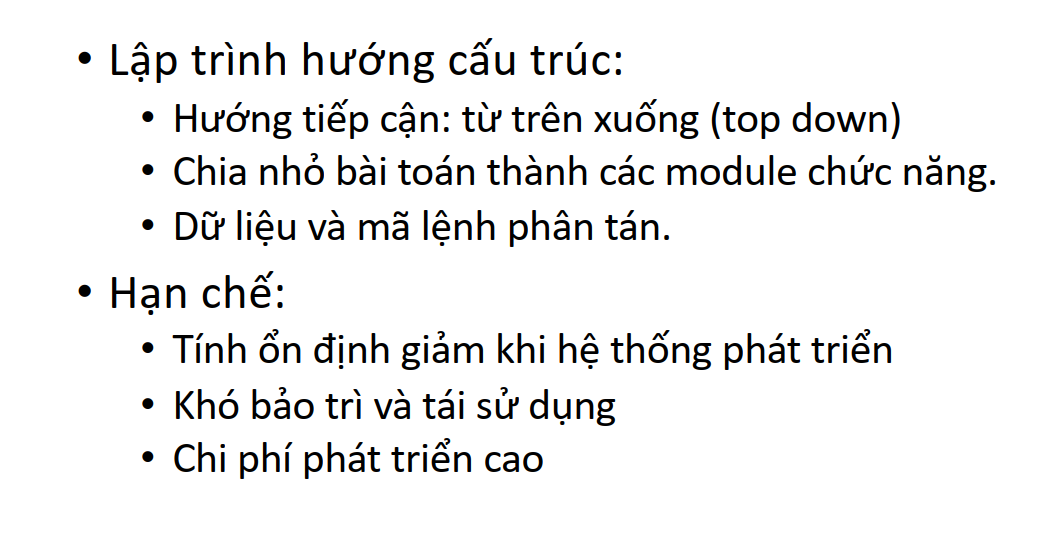
2. Kế thừa (Inheritance): Cho phép đối tượng mới được tạo ra bằng cách sử dụng các thuộc tính và phương thức của đối tượng cha. Kế thừa giúp giảm thiểu mã lệnh trùng lặp và tăng tính sáng tạo của chương trình.

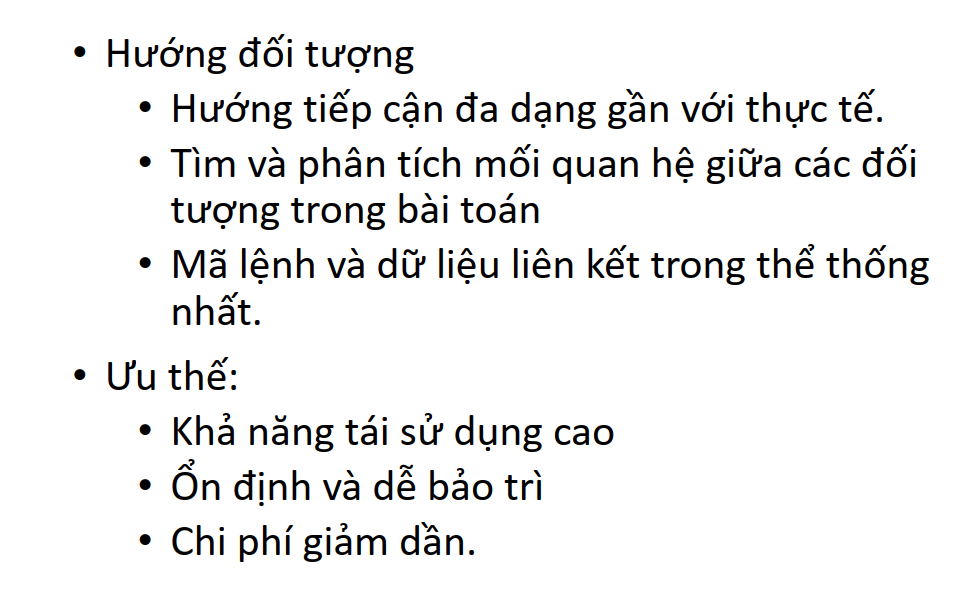
3. Đa hình (Polymorphism): Cho phép các đối tượng có thể thực hiện cùng một hành động khác nhau theo cách của riêng chúng. Đa hình giúp tăng tính linh hoạt và đơn giản hóa việc thiết kế chương trình.

Trong OOP, các đối tượng được tạo ra từ các lớp (classes) có cùng các thuộc tính và phương thức. Mỗi đối tượng là một thể hiện (instance) của lớp tương ứng, có thể thực hiện các phương thức để thực hiện các chức năng trong chương trình.

OOP giúp tăng tính tái sử dụng và bảo trì của mã lệnh, giúp tăng tính mở rộng và dễ dàng thay đổi chương trình khi cần thiết. Nó được sử dụng rộng rãi trong nhiều ngôn ngữ lập trình như Java, Python, C++, và C#.

1. **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 (procedural programming) là hai phương pháp lập trình khác nhau.

Lập trình hướng đối tượng tập trung vào việc thiết kế và sử dụng các đối tượng để thực hiện các chức năng trong chương trình. Các đối tượng là các thực thể độc lập trong chương trình, có thể có thuộc tính và phương thức riêng. Chương trình OOP thường được xây dựng dựa trên các lớp (classes), mỗi lớp định nghĩa các thuộc tính và phương thức của đối tượng tương ứng. OOP tập trung vào tính kế thừa, tính đóng gói và tính đa hình của các đối tượng.

Lập trình hướng thủ tục tập trung vào việc thiết kế và sử dụng các hàm hoặc thủ tục để thực hiện các chức năng trong chương trình. Các hàm hoặc thủ tục có thể nhận đầu vào và trả về giá trị, và thường được gọi để thực hiện các tác vụ cụ thể. Chương trình hướng thủ tục thường được tổ chức theo một chuỗi các hàm hoặc thủ tục được gọi theo một thứ tự nhất định để thực hiện các tác vụ cụ thể.

Tóm lại, sự khác biệt chính giữa lập trình hướng đối tượng và lập trình hướng thủ tục là phương pháp tiếp cận trong việc thiết kế và triển khai chương trình. Lập trình hướng đối tượng tập trung vào đối tượng, trong khi lập trình hướng thủ tục tập trung vào hàm hoặc thủ tục.

1. **Phân biệt giữa object và class?**

Trong lập trình hướng đối tượng, một class (lớp) là một bản thiết kế hoặc mẫu để tạo ra các object (đối tượng). Class định nghĩa các thuộc tính (attributes) và phương thức (methods) mà các object được tạo ra từ đó có thể sử dụng.

Một object là một thực thể cụ thể được tạo ra từ một class cụ thể, bằng cách sử dụng từ khóa "new" hoặc bằng các phương thức tạo mới (constructor) trong class đó. Mỗi object có thể có các thuộc tính và phương thức riêng của nó, và có thể thực hiện các hoạt động khác nhau dựa trên phương thức được cung cấp bởi class.

Ví dụ, một class "Car" có thể có các thuộc tính như "brand" (hãng xe), "model" (mẫu xe), "year" (năm sản xuất) và các phương thức như "start" (khởi động), "stop" (tắt máy), "accelerate" (tăng tốc) và "brake" (giảm tốc độ). Một object "car1" có thể được tạo ra từ class này với các thuộc tính như "Toyota", "Corolla", "2020" và có thể được điều khiển bằng các phương thức được cung cấp trong class "Car".

Tóm lại, class là một mẫu để tạo ra các object, trong khi object là một thực thể cụ thể được tạo ra từ class đó. Class định nghĩa các thuộc tính và phương thức, còn object có thể sử dụng các thuộc tính và phương thức đó để thực hiện các hoạt động cụ thể.

1. **Hai khái niệm “Môn học” và “Môn học JavaScript” thì đâu là object?**

"Môn học" là một khái niệm chung chung, không phải là một object trong lập trình hướng đối tượng. Trong khi đó, "Môn học JavaScript" có thể được coi là một object trong ngữ cảnh của lập trình hướng đối tượng, nếu nó được xem là một thực thể có thể có các thuộc tính và phương thức riêng của nó.

1. **Mô tả về sơ đồ lớp?**

Sơ đồ lớp mô tả những đặc điểm khái quát nhất về lớp bao gồm: Tên  
lớp, danh sách các thuộc tính (tên và loại dữ liệu) và các phương thức.

Sơ đồ lớp (Class Diagram) là một công cụ hữu ích để mô tả cấu trúc của một hệ thống hoặc một phần của hệ thống trong lập trình hướng đối tượng (OOP). Sơ đồ lớp biểu thị các class (lớp) và mối quan hệ giữa chúng, bao gồm các thuộc tính (attributes), phương thức (methods), interface, inheritance (kế thừa), association (liên kết) và các quan hệ khác.

Một sơ đồ lớp thường bao gồm các thành phần sau:

1. Class (Lớp): Lớp đại diện cho một loại đối tượng trong hệ thống, nó có tên, thuộc tính và phương thức. Class được định nghĩa bằng cách sử dụng các từ khóa như class, public, private, protected, abstract, và final.

2. Attribute (Thuộc tính): Thuộc tính đại diện cho các dữ liệu của class, bao gồm các biến (variable) và các hằng số (constant). Thuộc tính được mô tả bằng cách đặt tên và kiểu dữ liệu của chúng.

3. Method (Phương thức): Phương thức đại diện cho các hành động của class, bao gồm các hàm (function) và các thủ tục (procedure). Phương thức được mô tả bằng cách đặt tên, kiểu trả về và các tham số đầu vào của chúng.

4. Inheritance (Kế thừa): Kế thừa đại diện cho quan hệ "is-a" giữa các class, cho phép các class kế thừa các thuộc tính và phương thức của class cha.

5. Association (Liên kết): Liên kết đại diện cho quan hệ "has-a" giữa các class, cho phép một class có thể chứa các đối tượng của các class khác.

6. Interface: Interface đại diện cho các phương thức khai báo mà một class phải triển khai nếu nó muốn thực hiện interface đó.

Sơ đồ lớp cung cấp một cách tốt để hiểu cấu trúc của hệ thống và giúp cho việc thiết kế, phát triển và bảo trì hệ thống trở nên dễ dàng hơn.

1. **Vẽ sơ đồ lớp của lớp Học viên?**
2. **Thuộc tính là gì? Nên dùng từ loại nào để biểu diễn thuộc tính?**

Thuộc tính là những đặc điểm đặc trưng của đối tượng, thể hiện thông qua những giá trị cụ thể

1. **Phương thức là gì? Nên dùng từ loại nào để biểu diễn phương thức?**

Phương thức (Method)/Hành vi (Behavior)/Hành động (Action)/Khả năng (Capability): Hành vi là những cách thức mà qua đó đối tượng thể hiện sự hoạt động hay chức năng của chúng

1. **Cú pháp định nghĩa 1 class trong JavaScript?**
2. **Cú pháp tạo 1 đối tượng từ class?**
3. **Trình bày một số class có sẵn trong JavaScript mà em biết?**
4. **Cú pháp truy cập đến 1 thuộc tính của đối tượng?**
5. **Cú pháp truy cập đến 1 phương thức của đối tượng?**
6. **Từ khóa this dùng để làm gì?**
7. **Trình bày về DOM (Document Object Model – Mô hình đối tượng tài liệu)?**
8. **Thẻ <canvas> dùng để làm gì?**
9. **Vẽ hình chữ nhật trên canvas?**
10. **Vẽ hình tròn trên canvas?**
11. **Vẽ text trên canvas?**
12. **Nếu muốn di chuyển một hình tròn trên canvas cần làm gì?**