**CÂU HỎI AUDIT MODULE 1**

**BOOTCAMP PREPARATION 2.0**

|  |  |  |
| --- | --- | --- |
| **STT** | **Câu hỏi** | |
| 1 | | Repository là gì? có bao nhiêu loại repository?  Repository là một nơi lưu trữ các tập tin, mã nguồn và dữ liệu liên quan đến một dự án hoặc một phần mềm cụ thể. Nó cung cấp một cách tiện lợi và dễ dàng quản lý các tập tin liên quan đến dự án hoặc phần mềm.  Có hai loại repository chính:   * Local Repository: Đây là một nơi lưu trữ trên máy tính cá nhân của bạn và chỉ có bạn mới có quyền truy cập vào nó. * Remote Repository: Đây là một nơi lưu trữ được host trên một máy chủ công cộng và có thể truy cập từ bất kỳ đâu trên Internet. Nó được sử dụng để quản lý mã nguồn cho các dự án phức tạp và cho phép nhiều người cùng làm việc trên cùng một dự án.   Các nền tảng lưu trữ phổ biến như GitHub, GitLab và Bitbucket cung cấp các remote repository để lưu trữ và quản lý mã nguồn cho các dự án mã nguồn mở hoặc riêng. | |
| 2 | | Các câu lệnh căn bản để làm việc với git?   * git init: Sử dụng câu lệnh này để tạo một repository mới. * git clone [repository]: Sử dụng câu lệnh này để sao chép một repository từ máy chủ xa về máy tính cục bộ. * git add [file]: Sử dụng câu lệnh này để thêm tập tin hoặc thư mục vào stage. * git commit -m "[message]": Sử dụng câu lệnh này để lưu lại các thay đổi trong stage vào repository. * git push: Sử dụng câu lệnh này để gửi các thay đổi từ repository cục bộ lên repository xa. * git pull: Sử dụng câu lệnh này để tải về các thay đổi từ repository xa về repository cục bộ. * git status: Sử dụng câu lệnh này để xem trạng thái hiện tại của repository. * git log: Sử dụng câu lệnh này để xem lịch sử các commit trong repository. * git diff [file]: Sử dụng câu lệnh này để xem sự khác biệt giữa các phiên bản của tập tin. * git branch: Sử dụng câu lệnh này để xem danh sách các nhánh trong repository. | |
| 3 | | Giải thuật là gì? các cách để biểu diễn thuật toán?   * Giải thuật là một tập hợp các bước cụ thể và rõ ràng để giải quyết một vấn đề hoặc xử lý một tác vụ cụ thể. Nó cung cấp một cách dễ nhìn và dễ hiểu để giải quyết các vấn đề phức tạp. * Các cách để biểu diễn thuật toán bao gồm: * Sử dụng Pseudocode: Đây là một ngôn ngữ mô tả thuật toán mà không phải là mã máy tính cụ thể. Nó giống với mã, nhưng dễ hiểu hơn và dễ dàng chuyển đổi sang mã máy. * Sử dụng Flowchart: Đây là một hình vẽ biểu đồ dùng để biểu diễn các bước trong thuật toán. Các biểu tượng và cạnh của flowchart biểu thị các bước và các quan hệ giữa các bước. * Sử dụng mã máy tính: Đây là cách chuyển thuật toán thành mã máy tính cụ thể và chạy trên máy tính. Mã máy tính có thể viết bằng nhiều ngôn ngữ lập trình khác nhau như C, Python, Java, vv. | |
| 4 | | Mô tả thuật toán tìm kiếm 1 phần tử trong mảng bằng mã giả?  Begin  for each item in the array do  if item equals target then  return the index of item  end if  end for  return “Không tìm thấy”  End  Giải thích:   1. Thuật toán nhận đầu vào là một mảng (array) và giá trị cần tìm kiếm (target). 2. Tiến hành lặp qua từng phần tử trong mảng. 3. Kiểm tra nếu giá trị của phần tử đó bằng giá trị cần tìm kiếm thì trả về chỉ số của phần tử đó. 4. Nếu không tìm thấy, trả về -1.   Thuật toán tìm kiếm này được gọi là tìm kiếm tuyến tính (linear search). Thuật toán này có độ phức tạp thời gian O(n), trong đó n là số phần tử trong mảng. | |
| 5 | | Mô tả thuật toán sắp xếp mảng số nguyên bằng mã giả?  Dưới đây là mã giả Pseudo của thuật toán sắp xếp mảng số nguyên theo thuật toán sắp xếp chọn (selection sort):  Begin  for i from 0 to n-1 do  minIndex = i  for j from i+1 to n do  if array[j] < array[minIndex] then  minIndex = j  end if  end for  if minIndex != i then  swap array[minIndex] and array[i]  end if  end for  End | |
| 6 | | Trình bày ý tưởng của 1 thuật toán bất kì?  thuật toán tính tổng các phần tử trong một mảng:  Begin  procedure sumArray(array)  sum = 0 // khởi tạo tổng ban đầu là 0  for i from 0 to n-1 do // lặp qua từng phần tử trong mảng  sum = sum + array[i] // cộng giá trị của từng phần tử vào tổng  end for  return sum // trả về tổng của các phần tử trong mảng  End  Thuật toán trên có ý tưởng chính như sau:   * Thuật toán nhận đầu vào là một mảng (array). * Khởi tạo tổng ban đầu (sum) là 0. * Duyệt qua từng phần tử trong mảng bắt đầu từ đầu mảng đến hết mảng. * Tại mỗi vòng lặp, cộng giá trị của phần tử hiện tại vào tổng. * Trả về giá trị của tổng sau khi đã tính toán xong tất cả các phần tử trong mảng. | |
| 7 | | Nêu các thẻ để tạo danh sách ? Sự khác nhau giữa <ul> và <ol> ?  Có hai thẻ chính để tạo danh sách trong HTML:   * **<ul>** (Unordered List): Tạo danh sách không có thứ tự. Mỗi mục trong danh sách được bao bọc bởi thẻ **<li>** (List Item). * **<ol>** (Ordered List): Tạo danh sách có thứ tự. Tương tự như **<ul>**, mỗi mục trong danh sách được bao bọc bởi thẻ **<li>**.   Sự khác nhau giữa **<ul>** và **<ol>** là cách thức hiển thị thứ tự của các mục trong danh sách. Trong danh sách không có thứ tự (**<ul>**), mỗi mục được hiển thị bằng một dấu bullet, còn trong danh sách có thứ tự (**<ol>**), mỗi mục được hiển thị bằng một số thứ tự tăng dần. | |
| 8 | | Các thẻ tạo nên 1 table?   1. Thẻ <table> : Là thẻ gốc của bảng và bao gồm toàn bộ nội dung của bảng. 2. Thẻ <tr> (Table Row) : Là thẻ dùng để tạo một hàng trong bảng. 3. Thẻ <td> (Table Data) : Là thẻ dùng để tạo một ô dữ liệu trong một hàng. 4. Thẻ <th> (Table Header) : Là thẻ dùng để tạo một tiêu đề cho hàng hoặc cột trong bảng.   Phân biệt th và td?   * **<th>** được sử dụng để tạo tiêu đề cho hàng hoặc cột trong bảng, thông thường được in đậm và có nền màu khác so với các ô dữ liệu. * **<td>** được sử dụng để tạo một ô dữ liệu trong một hàng, chứa các giá trị thực tế của bảng | |
| 9 | | Phân biệt innerHTML và innerText?  **innerHTML** và **innerText** là hai thuộc tính của đối tượng DOM (Document Object Model) trong JavaScript, dùng để truy cập hoặc thay đổi nội dung của một thẻ HTML.  **innerHTML** truy cập hoặc thay đổi toàn bộ nội dung bao gồm cả các thẻ HTML trong thẻ đó. | |
| 10 | | Nêu một số thẻ HTML cơ bản mà bạn biết   * 1. **<html>**: Thẻ HTML gốc, bao gồm toàn bộ nội dung trang web   2. **<head>**: Thẻ head chứa thông tin về trang web như tiêu đề, tập tin CSS và JavaScript   3. **<body>**: Thẻ body chứa nội dung của trang web   4. **<header>**: Thẻ header chứa thông tin đầu trang như logo, tiêu đề và menu   5. **<nav>**: Thẻ nav chứa các liên kết đến các trang khác trong trang web   6. **<main>**: Thẻ main chứa nội dung chính của trang web   7. **<section>**: Thẻ section chứa nội dung theo chủ đề khác nhau   8. **<article>**: Thẻ article chứa nội dung tự do như bài viết, tin tức   9. **<aside>**: Thẻ aside chứa nội dung bổ sung như menu bên, quảng cáo   10. **<footer>**: Thẻ footer chứa thông tin cuối trang như địa chỉ, số điện thoại và bản quyền.   Còn các thẻ như **<p>**, **<h1>** - **<h6>**, **<a>**, **<img>**, **<ul>**, **<ol>**, **<li>**, **<table>**, **<tr>**, **<th>**, **<td>**, **<form>**, **<input>**, **<label>**, **<select>**, **<option>**, **<textarea>** cũng là những thẻ HTML quan trọng. | |
| 11 | | Các thuộc tính cơ bản của thẻ form   * 1. action: Xác định URL mà dữ liệu được gửi đến khi người dùng nhấn nút submit.   2. method: Xác định phương thức gửi dữ liệu (GET hoặc POST).   3. enctype: Xác định kiểu mã hóa dữ liệu được gửi đến máy chủ.   4. target: Xác định khung trình duyệt mà kết quả sẽ được hiển thị.   5. name: Tên của form, dùng để xác định form trong JavaScript.   6. autocomplete: Xác định xem form có cho phép tự động hoàn tất hay không.   7. novalidate: Không kiểm tra form trước khi gửi.   8. accept-charset: Xác định bảng ký tự sẽ được sử dụng khi gửi dữ liệu đến máy chủ. | |
| 12 | | Phân biệt phương thức get và post trong thẻ form.   * Phương thức GET: Là phương thức mặc định khi gửi một form. Nó sẽ gửi dữ liệu từ form lên máy chủ theo đường dẫn URL của trang. Dữ liệu sẽ được gửi dưới dạng tham số trong URL, vì thế nếu bạn sử dụng GET, các dữ liệu sẽ được hiển thị trong URL. * Phương thức POST: Là phương thức gửi dữ liệu từ form lên máy chủ mà không hiển thị trong URL. Điều này có nghĩa là dữ liệu gửi lên sẽ được giữ an toàn và không bị hiển thị cho bất kỳ ai. Nếu bạn cần gửi dữ liệu mật mã hoặc bảo mật, hãy sử dụng phương thức POST.   Chung chung, phương thức GET dùng cho việc lấy dữ liệu và phương thức POST dùng cho việc gửi dữ liệu. | |
| 13 | | Cách khai báo biến bằng từ khóa let và var khác nhau như thế nào ?. Phạm vi của biến?  Khai báo biến bằng từ khóa **let** và **var** khác nhau trong JavaScript như sau:   * Khai báo biến bằng **var**: * Biến được khai báo bằng **var** có thể được khai báo lại và gán giá trị mới. * Biến được khai báo bằng **var** có phạm vi toàn cục. * Khai báo biến bằng **let**: * Biến được khai báo bằng **let** chỉ có thể được khai báo một lần và không thể khai báo lại. * Biến được khai báo bằng **let** có phạm vi block-level.   Phạm vi của biến là khu vực trong chương trình mà biến đó có thể được truy cập và sử dụng. Với **var**, biến có phạm vi toàn cục, nghĩa là biến có thể được truy cập và sử dụng trong bất kỳ đoạn mã nào trong chương trình. Còn với **let**, biến chỉ có phạm vi trong block-level, nghĩa là biến chỉ có thể được truy cập và sử dụng trong block của nó được khai báo. | |
| 14 | | Trong Javascript có bao nhiêu loại kiểu dữ liệu ?. Làm thế nào để xác định được biến có kiểu dữ liệu gì?  Trong Javascript có 7 kiểu dữ liệu chính:   1. Number: Kiểu dữ liệu cho các số, bao gồm số thực và số nguyên. 2. String: Kiểu dữ liệu cho các chuỗi ký tự, bao gồm cả ký tự đơn và ký tự đặc biệt. 3. Boolean: Kiểu dữ liệu cho các giá trị true hoặc false. 4. Null: Kiểu dữ liệu cho giá trị null. 5. Undefined: Kiểu dữ liệu cho giá trị undefined. 6. Object: Kiểu dữ liệu cho các đối tượng, có thể là các đối tượng cụ thể hoặc chung. 7. Symbol: Kiểu dữ liệu mới trong ES6, cho biết một giá trị duy nhất.   Bạn có thể sử dụng typeof operator để xác định kiểu dữ liệu của một biến. Ví dụ:  **let x = 123;** **console.log(typeof x); // Output: number**  **let y = "hello";** **console.log(typeof y); // Output: string**  **let z = true;** **console.log(typeof z); // Output: boolean** | |
| 15 | | Các cách tạo chuỗi chứa dấu nháy ?  **Trong JavaScript, có ba cách để tạo một chuỗi chứa dấu nháy:**   1. Sử dụng dấu nháy đôi: Bạn có thể tạo một chuỗi chứa dấu nháy đôi bằng cách sử dụng dấu nháy đôi để gói chuỗi. Ví dụ: "This is a string with quotes." 2. Sử dụng dấu nháy đơn: Bạn có thể tạo một chuỗi chứa dấu nháy đơn bằng cách sử dụng dấu nháy đơn để gói chuỗi. Ví dụ: 'This is a string with quotes.' 3. Sử dụng ký tự escape: Bạn có thể tạo một chuỗi chứa dấu nháy bằng cách sử dụng ký tự escape. Ví dụ: "This is a string with "quotes"."   **Lưu ý rằng, bạn phải chọn loại dấu nháy phù hợp với yêu cầu và tùy chọn của mình.** | |
| 16 | | NaN là gì? NaN === NaN có đúng không?Vì sao?  NaN là viết tắt của "Not-a-Number" (Không phải là một số) và là một giá trị kiểu số trong JavaScript. Nó được sử dụng để biểu thị kết quả của một phép tính không hợp lệ hoặc bị lỗi, ví dụ như chia cho 0 hoặc chuyển đổi một chuỗi sang kiểu số mà không thể chuyển đổi được.  NaN === NaN là sai, vì đây là một đặc điểm đặc biệt của NaN. Hàm **isNaN()** được sử dụng để kiểm tra xem một giá trị có phải là NaN hay không. | |
| 17 | | Cho 1 bài toán liên quan đến ++ trước và ++ sau. Yêu cầu dự đoán kết quả, và tại sao lại có kết quả như vậy? | |
| 18 | | Các loại toán tử trong js? Cho biết độ ưu tiên của các toán tử trong một biểu thức?  **Trong JavaScript, có nhiều loại toán tử khác nhau, bao gồm:**   1. Toán tử tăng giảm (++, --) 2. Toán tử gán (=, +=, -=, \*=, /=, %=) 3. Toán tử so sánh (==, ===, !=, !==, >, <, >=, <=) 4. Toán tử logic (&&, ||, !) 5. Toán tử bit (&, |, ^, ~, <<, >>, >>>) 6. Toán tử ternary (? :) 7. Toán tử instanceof 8. Toán tử typeof 9. Toán tử void 10. Toán tử delete   Độ ưu tiên của các toán tử trong một biểu thức được xác định bởi các quy tắc toán học, và một số toán tử có độ ưu tiên cao hơn so với toán tử khác. Ví dụ, toán tử tăng giảm và toán tử gán có độ ưu tiên cao hơn so với toán tử so sánh. Toán tử logic và toán tử ternary có độ ưu tiên thấp hơn so với toán tử tăng giảm và toán tử gán. | |
| 19 | | Toán tử ba ngôi là gì? Cú pháp?  Toán tử ba ngôi (ternary operator) là một toán tử đặc biệt trong JavaScript, cho phép bạn thực hiện một biểu thức điều kiện theo cú pháp ngắn gọn. Nó giống như if-else, nhưng viết trong một dòng và cấu trúc ngắn gọn hơn.  Cú pháp của toán tử ba ngôi là:  **condition ? expression1 : expression2**   * **condition** là biểu thức điều kiện được kiểm tra. * **expression1** là biểu thức sẽ được thực hiện nếu điều kiện là **true**. * **expression2** là biểu thức sẽ được thực hiện nếu điều kiện là false. | |
| 20 | | So sánh null và rỗng  **null** và **undefined** là hai giá trị khác nhau trong JavaScript.   * **null** là một giá trị cho biến chỉ để chú thích rằng biến đó không có giá trị hoặc không được gán bất kỳ giá trị nào. * **undefined** là một giá trị đặc trưng cho biến chưa được gán hoặc chưa được xác định giá trị.   Còn với **""** (hoặc một chuỗi rỗng), nó là một chuỗi giá trị cho biến mà không chứa bất kỳ ký tự nào.  Về mặt so sánh, **null** và **undefined** đều là giá trị rỗng trong JavaScript. Tuy nhiên, **null** được sử dụng để gán cho biến mục đích đặc biệt (như chú thích rằng biến đó không có giá trị), trong khi **undefined** được sử dụng để xác định rằng một biến hoặc một giá trị đã được xác định nhưng chưa được gán bất kỳ giá trị nào. | |
| 21 | | Toán tử == và === khác và giống nhau như thế nào  Toán tử **==** và **===** trong Javascript có một số khác biệt về cách so sánh giá trị của hai biến.   * Toán tử **==** (so sánh bằng với chuyển đổi kiểu) sẽ thực hiện việc chuyển đổi kiểu dữ liệu của hai giá trị trước khi so sánh chúng. Ví dụ, **"5" == 5** sẽ trả về **true**, vì chuỗi "5" được chuyển đổi thành số 5 trước khi so sánh. * Toán tử **===** (so sánh bằng với không chuyển đổi kiểu) sẽ so sánh hai giá trị mà không thực hiện việc chuyển đổi kiểu. Ví dụ, **"5" === 5** sẽ trả về **false**, vì hai giá trị là kiểu dữ liệu khác nhau.   Tổng quan, việc sử dụng toán tử **===** luôn được ưu tiên hơn so với toán tử **==** vì nó luôn luôn chính xác trong việc so sánh giá trị và kiểu dữ liệu, tránh việc xảy ra các lỗi so sánh khi sử dụng toán tử **==** | |
| 22 | | Phân biệt giữa 2 toán tử && và || trong JavaScript?  Toán tử && (AND) và || (OR) là 2 toán tử logic trong JavaScript, có mục đích phân tích 2 giá trị và trả về một giá trị dựa trên kết quả.   * Toán tử && (AND) sẽ trả về giá trị true nếu cả 2 giá trị đầu vào là true, trong trường hợp còn lại nó sẽ trả về false. * Toán tử || (OR) sẽ trả về giá trị true nếu bất kỳ một trong 2 giá trị đầu vào là true, trong trường hợp còn lại nó sẽ trả về false.   let a = true, b = false;  console.log(a && b); // false  console.log(a || b); // true | |
| 23 | | Các hàm dùng để tạo thông báo trong Javascript ?  Trong JavaScript có hai hàm chính được sử dụng để tạo thông báo:   1. alert(): Hàm alert() được sử dụng để hiển thị một thông báo dạng hộp thoại. Bạn có thể chứa một chuỗi, biến hoặc biểu thức trong nó. 2. prompt(): Hàm prompt() cung cấp một cửa sổ hộp thoại để nhập dữ liệu từ người dùng. Nó cũng hiển thị một thông báo và có một trường nhập văn bản. Kết quả được trả về là giá trị nhập vào bởi người dùng | |
| 24 | | Có bao nhiêu Statement control(câu lệnh điều khiển) trong Java Script ?  Trong JavaScript có 4 loại câu lệnh điều khiển chính:   1. if...else: điều khiển chỉ có hai nhánh và dùng để xác định một điều kiện. 2. switch: cho phép bạn chọn một trong nhiều nhánh của một điều kiện. 3. for: dùng để lặp một mảng hoặc tập hợp dữ liệu cụ thể. 4. while: dùng để lặp cho đến khi một điều kiện được đáp ứng. | |
| 25 | | So sánh sự khác nhau giữa if và switch case?.  **if** và **switch case** đều là các câu lệnh điều khiển trong JavaScript, nhưng có một số sự khác nhau giữa hai kiểu điều khiển này.   1. Cú Pháp: Cú pháp cho **if** là if (điều kiện) { ... } trong khi cú pháp cho **switch case** là switch (biến) { case giá trị: ...; break; } 2. Áp dụng: **if** thường được sử dụng để xác định và thực hiện hành động theo một điều kiện đặc biệt, trong khi **switch case** thường được sử dụng khi cần xác định nhiều điều kiện và các hành động tương ứng với mỗi điều kiện. 3. Tốc độ chạy: **if** thường chạy nhanh hơn **switch case** trong một số trường hợp, nhưng nếu số lượng điều kiện lớn, **switch case** có thể chạy nhanh hơn **if**.   Tổng kết, tùy thuộc vào tình huống cụ thể và yêu cầu, bạn có thể chọn sử dụng **if** hoặc **switch case** hoặc cả hai để hoàn thành tác vụ điều khiển. | |
| 26 | | Switch case so sánh == hay ===. Đặt ra trường hợp là so sánh bằng thì khi nào sử dụng if bậc thang? Khi nào sử dụng switch case   * Trong JavaScript, toán tử so sánh bằng '==' sẽ so sánh giá trị của 2 biến mà không quan tâm đến kiểu dữ liệu của chúng, trong khi đó toán tử so sánh bằng '===' sẽ so sánh cả giá trị và kiểu dữ liệu của 2 biến. * Về việc sử dụng if bậc thang hay switch case, ta có thể sử dụng if bậc thang khi có nhiều điều kiện cần kiểm tra và có thể sử dụng toán tử so sánh bằng '==' hoặc '===' trong mỗi điều kiện. Nếu có quá nhiều điều kiện, thì sử dụng switch case có thể là cách tốt hơn để tăng tính dễ đọc và hiệu quả của mã. | |
| 27 | | Các biểu thức và luồng thực thi của for. Nếu thiếu 1 hoặc tất cả các biểu thức thì vòng for sẽ chạy như thế nào?  Trong JavaScript, cấu trúc vòng lặp for có 3 phần tử:   * Khởi tạo biến đếm và gán giá trị ban đầu cho biến đó. * Điều kiện để tiếp tục hoặc kết thúc vòng lặp. * Cập nhật giá trị của biến đếm sau mỗi lần lặp.   Cú pháp của vòng lặp for như sau:  for (khởi tạo biến đếm; điều kiện; cập nhật giá trị biến đếm) {  // thực hiện các lệnh trong vòng lặp  }  Khi một trong ba phần tử không được cung cấp, vòng lặp for vẫn có thể hoạt động.  Nếu thiếu phần tử khởi tạo biến đếm, biến đếm sẽ được định nghĩa trước đó (hoặc sử dụng biến đếm đã được khai báo ở bên ngoài vòng lặp). Ví dụ:  let i = 0;  for (; i < 5; i++) {  console.log(i);  }  Nếu thiếu phần tử điều kiện, vòng lặp sẽ được coi là luôn luôn đúng, do đó sẽ trở thành một vòng lặp vô hạn. Ví dụ:  for (let i = 0;; i++) {  console.log(i);  }  Nếu thiếu phần tử cập nhật giá trị biến đếm, bạn có thể cập nhật giá trị biến đếm bên trong thân vòng lặp. Ví dụ:  for (let i = 0; i < 5;) {  console.log(i);  i++;  }  Nếu tất cả các phần tử đều bị thiếu, vòng lặp sẽ trở thành một vòng lặp vô hạn:  for (;;) {  console.log("This is an infinite loop");  }  Vì vậy, khi sử dụng vòng lặp for, chúng ta cần chắc chắn cung cấp đầy đủ các phần tử để tránh các lỗi không mong muốn. | |
| 28 | | Đặt ra 1 bài toán. Xác định bài toán cần sử dụng vòng lặp nào?  Giả sử ta có một danh sách các số nguyên và muốn tìm tổng của tất cả các số đó. Để giải quyết bài toán này, ta cần sử dụng một vòng lặp để lặp qua từng phần tử trong danh sách và tính tổng.  Ví dụ mã giả Pseudo để tính tổng các số trong một danh sách:  Begin  sum = 0 // khởi tạo tổng ban đầu là 0  for i from 0 to length(list) - 1 do // lặp qua từng phần tử trong danh sách  sum = sum + list[i] // cộng giá trị của từng phần tử vào tổng  end for  return sum // trả về tổng của danh sách  End  Ở đây, ta sử dụng vòng lặp **for** để lặp qua từng phần tử trong danh sách và tính tổng. Vòng lặp **for** có thể được sử dụng để lặp qua một tập hợp các phần tử với một số lần lặp được xác định trước (trong trường hợp này là độ dài của danh sách).  Vì vậy, khi gặp các bài toán tương tự, chúng ta có thể sử dụng vòng lặp **for** để lặp qua từng phần tử trong danh sách và thực hiện các thao tác tính toán hoặc xử lý các phần tử đó. | |
| 29 | | So sánh giống và khác nhau giữa for, while và do..while  Cả ba câu lệnh lặp for, while và do..while được sử dụng để thực hiện lặp lại một khối mã nhiều lần. Tuy nhiên, chúng có một số khác nhau về cú pháp, cách thức thực hiện lặp, và điều kiện kiểm tra để dừng lặp.   1. Cú pháp:  * **for**:   for (initialization; condition; increment/decrement) {  // code to be executed  }  **while**:  while (condition) {  // code to be executed  }  **do..while**:  do {  // code to be executed  } while (condition);   1. Cách thức thực hiện lặp:  * **for**:   + Thực hiện khởi tạo một lần trước khi lặp bắt đầu   + Thực hiện kiểm tra điều kiện lặp trước mỗi lần lặp lại   + Thực hiện các câu lệnh trong khối lệnh sau đó thực hiện bước tăng/giảm trước khi kiểm tra điều kiện lặp. * **while**:   + Thực hiện kiểm tra điều kiện lặp trước mỗi lần lặp lại.   + Nếu điều kiện lặp đúng, thực hiện các câu lệnh trong khối lệnh. * **do..while**:   + Thực hiện các câu lệnh trong khối lệnh một lần đầu tiên.   + Thực hiện kiểm tra điều kiện lặp trước mỗi lần lặp lại. Nếu điều kiện lặp đúng, thực hiện lại các câu lệnh trong khối lệnh.  1. Điều kiện kiểm tra để dừng lặp:  * **for**:   + Lặp lại cho đến khi điều kiện lặp sai. * **while**:   + Lặp lại cho đến khi điều kiện lặp sai. * **do..while**:   + Lặp lại cho đến khi điều kiện lặp sai. | |
| 30 | | So sánh break và continue  Cả hai lệnh **break** và **continue** đều được sử dụng để kiểm soát lưu thông của vòng lặp. Tuy nhiên, chúng có một số khác nhau về cách thức hoạt động và tác động tới vòng lặp.   1. **break**:  * **break** dùng để dừng một vòng lặp hoặc một câu lệnh switch. * Khi vòng lặp đang thực hiện, nếu **break** được gọi, vòng lặp sẽ dừng ngay lập tức và chương trình sẽ tiếp tục thực hiện các câu lệnh sau vòng lặp. * **break** thường được sử dụng để kiểm tra một điều kiện và dừng vòng lặp nếu điều kiện đó được thỏa mãn.   Ví dụ:  for (var i = 0; i < 10; i++) {  if (i === 5) {  break;  }  console.log(i);  }  Khi i = 5, vòng lặp sẽ dừng lại và giá trị của i sẽ không được in ra.   1. **continue**:  * **continue** dùng để bỏ qua một lần lặp của vòng lặp và tiếp tục lặp tiếp theo. * Khi vòng lặp đang thực hiện, nếu **continue** được gọi, câu lệnh trong vòng lặp sẽ không được thực hiện cho lần lặp hiện tại và chương trình sẽ tiếp tục với lần lặp tiếp theo. * **continue** thường được sử dụng để kiểm tra một điều kiện và bỏ qua một lần lặp nếu điều kiện đó được thỏa mãn.   Ví dụ:  for (var i = 0; i < 10; i++) {  if (i === 5) {  continue;  }  console.log(i);  }  Khi i = 5, lần lặp hiện tại sẽ bị bỏ qua và giá trị của i sẽ không được in ra.  Tóm lại, **break** và **continue** là hai lệnh quan trọng được sử dụng để kiểm soát vòng lặp. **break** được sử dụng để dừng một vòng lặp hoặc một câu lệnh switch, trong khi **continue** được sử dụng để bỏ qua một lần lặp và tiếp tục với lần lặp tiếp theo. | |
| 31 | | Đặc điểm mảng một chiều trong Javascript  Mảng một chiều trong JavaScript là một tập hợp các phần tử có cùng kiểu dữ liệu được lưu trữ trong một biến có tên được đặt trước đó. Mảng được đánh số bắt đầu từ 0, do đó phần tử đầu tiên có chỉ số là 0, phần tử thứ hai có chỉ số là 1 và cứ như vậy.  Một số đặc điểm của mảng một chiều trong JavaScript bao gồm:   1. Kiểu dữ liệu đa dạng: Mảng một chiều trong JavaScript có thể lưu trữ nhiều kiểu dữ liệu khác nhau, bao gồm số, chuỗi, đối tượng và cả hàm. 2. Độ dài có thể thay đổi: Khi một mảng được tạo ra, nó có thể được chỉ định với một kích thước cụ thể. Tuy nhiên, độ dài của mảng có thể thay đổi trong quá trình chạy chương trình bằng cách thêm hoặc xoá các phần tử. 3. Truy cập vào phần tử: Phần tử của mảng có thể được truy cập bằng cách sử dụng chỉ số của phần tử đó. Chỉ số này có thể được sử dụng để truy cập vào giá trị của phần tử, hoặc để thay đổi giá trị của phần tử. 4. Lặp lại các phần tử của mảng: Các phần tử của mảng có thể được lặp lại bằng cách sử dụng các vòng lặp như for, while hoặc do-while. 5. Phương thức tích hợp: Mảng trong JavaScript đi kèm với các phương thức tích hợp như push, pop, shift, unshift, splice và slice để thêm, xoá hoặc cắt các phần tử của mảng.   Tóm lại, mảng một chiều trong JavaScript là một cấu trúc dữ liệu đơn giản nhưng rất quan trọng để lưu trữ và truy cập các phần tử dữ liệu theo thứ tự được chỉ định. | |
| 32 | | Các cách khởi tạo một mảng kiểu String trong JavaScript  Có thể khởi tạo một mảng kiểu String trong JavaScript bằng nhiều cách khác nhau. Dưới đây là một số cách phổ biến để khởi tạo mảng kiểu String trong JavaScript:   1. Sử dụng cú pháp **[]** để tạo một mảng rỗng, sau đó thêm các phần tử kiểu String vào mảng bằng cách sử dụng phương thức push():   var arr = [];  arr.push("Hello");  arr.push("World");   1. Khởi tạo mảng với giá trị mặc định bằng cách sử dụng cú pháp khởi tạo mảng:   var arr = ["Hello", "World"];   1. Sử dụng cú pháp **new Array()** để tạo một mảng, sau đó thêm các phần tử kiểu String vào mảng bằng cách sử dụng chỉ mục:   var arr = new Array();  arr[0] = "Hello";  arr[1] = "World";   1. Sử dụng cú pháp **split()** để chuyển đổi một chuỗi thành một mảng kiểu String. Phương thức này chia chuỗi thành các phần tử tại vị trí của dấu phân cách được chỉ định (thường là khoảng trắng) và trả về một mảng các phần tử kiểu String:   var str = "Hello World";  var arr = str.split(" ");   1. Sử dụng cú pháp **{}** để tạo một object với các thuộc tính kiểu String, sau đó lấy giá trị của các thuộc tính đó để tạo một mảng kiểu String:   var obj = {0: "Hello", 1: "World", length: 2};  var arr = [];  for (var i = 0; i < obj.length; i++) {  arr.push(obj[i]);  } | |
| 33 | | - 1 số hàm thao tác với mảng?   1. push(): Thêm một phần tử vào cuối mảng. 2. pop(): Xóa phần tử cuối cùng của mảng và trả về giá trị của phần tử đó. 3. shift(): Xóa phần tử đầu tiên của mảng và trả về giá trị của phần tử đó. 4. unshift(): Thêm một phần tử vào đầu mảng. 5. splice(): Xóa hoặc thay thế các phần tử của mảng. 6. concat(): Ghép nối hai hoặc nhiều mảng lại với nhau để tạo ra một mảng mới. 7. slice(): Sao chép một phần của mảng thành một mảng mới. 8. indexOf(): Tìm kiếm vị trí đầu tiên của một phần tử trong mảng. 9. reverse(): Đảo ngược thứ tự các phần tử trong mảng. 10. sort(): Sắp xếp các phần tử của mảng theo thứ tự tăng dần hoặc giảm dần.   - Phân biệt push() và unshift()?   1. Hàm push() dùng để thêm một phần tử vào cuối mảng, trong khi hàm unshift() dùng để thêm một phần tử vào đầu mảng. Ví dụ:   var arr = [1, 2, 3];  arr.push(4); // sau khi thực thi, arr sẽ trở thành [1, 2, 3, 4]  arr.unshift(0); // sau khi thực thi, arr sẽ trở thành [0, 1, 2, 3, 4] - Phân biệt push() và pop()?   1. Hàm push() dùng để thêm một phần tử vào cuối mảng, trong khi hàm pop() dùng để xóa phần tử cuối cùng của mảng và trả về giá trị của phần tử đó. Ví dụ:   var arr = [1, 2, 3];  arr.push(4); // sau khi thực thi, arr sẽ trở thành [1, 2, 3, 4]  var lastElement = arr.pop(); // sau khi thực thi, lastElement sẽ có giá trị là 4 và arr sẽ trở thành [1, 2, 3] - Phân biệt shift() và unshift()?   1. Hàm shift() dùng để xóa phần tử đầu tiên của mảng và trả về giá trị của phần tử đó, trong khi hàm unshift() dùng để thêm một phần tử vào đầu mảng. Ví dụ:   var arr = [1, 2, 3];  arr.shift(); // sau khi thực thi, arr sẽ trở thành [2, 3]  arr.unshift(0); // sau khi thực thi, arr sẽ trở thành [0, 2, 3] | |
| 34 | | Phân biệt tham trị và tham chiếu trong Javascript   * Tham trị (primitive types): đây là các kiểu dữ liệu cơ bản như số, chuỗi, boolean, null và undefined. Khi truyền tham trị vào một hàm, giá trị được truyền vào sẽ được sao chép và hàm sẽ làm việc trên một bản sao của giá trị này. Sau khi hàm thực thi xong, giá trị ban đầu không bị thay đổi.   Ví dụ:  function changeValue(x) {  x = 5;  }  var a = 10;  changeValue(a);  console.log(a); // a vẫn là 10   * Tham chiếu (reference types): đây là các kiểu dữ liệu đối tượng như mảng, đối tượng, hàm, v.v. Khi truyền tham chiếu vào một hàm, địa chỉ bộ nhớ của giá trị được truyền vào sẽ được sao chép. Hàm sẽ thao tác trực tiếp trên giá trị ban đầu, vì vậy nếu giá trị này được thay đổi trong hàm, thì giá trị ban đầu cũng sẽ bị thay đổi.   Ví dụ:  function addToArray(arr) {  arr.push(4);  }  var myArray = [1, 2, 3];  addToArray(myArray);  console.log(myArray); // myArray bây giờ là [1, 2, 3, 4]  Khi ta truyền một đối tượng vào hàm, hàm đó có thể sửa đổi đối tượng đó bên trong, vì vậy nó được gọi là "tham chiếu". Ngược lại, khi ta truyền vào một giá trị nguyên thủy (số, chuỗi, boolean, v.v.), hàm không thể làm gì với giá trị đó bên trong, vì vậy nó được gọi là "tham trị". | |
| 35 | | Phân biệt giữa Hàm có return và hàm không có return   * Hàm có **return**: Hàm có **return** sẽ trả về một giá trị khi được gọi, và giá trị này có thể được sử dụng trong chương trình. Hàm có thể trả về một giá trị bất kỳ nào, bao gồm cả giá trị đối tượng và giá trị nguyên thủy.   Ví dụ:  function sum(a, b) {  return a + b;  }  var result = sum(3, 4);  console.log(result); // Output: 7   * Hàm không có **return**: Hàm không có **return** sẽ thực hiện các công việc nhất định mà không trả về bất kỳ giá trị nào. Khi được gọi, hàm sẽ chỉ thực hiện các lệnh bên trong và sau đó trở lại điểm gọi của nó.   Ví dụ:  function sayHello(name) {  console.log("Hello, " + name);  }  sayHello("John"); // Output: Hello, John  Trong trường hợp không có câu lệnh **return** trong hàm, giá trị được trả về sẽ là **undefined**. | |
| 36 | | Lập trình hướng đối tượng là gì  Lập trình hướng đối tượng (OOP) là phương pháp lập trình tập trung vào các đối tượng và mối quan hệ giữa chúng. Các đối tượng có các thuộc tính và hành vi riêng, được đại diện bởi các biến và phương thức. OOP giúp dễ bảo trì, tái sử dụng mã, mở rộng và tường minh hơn. | |
| 37 | | Phân biệt class và object  Trong lập trình hướng đối tượng, một class là một bản thiết kế hoặc mẫu để tạo ra các đối tượng. Nó chứa các đặc điểm chung của các đối tượng cùng loại, bao gồm các thuộc tính và phương thức. Các đối tượng được tạo ra từ class được gọi là các instance của class.  Một object, hoặc đối tượng, là một thực thể cụ thể được tạo ra từ một class. Nó có thể có các giá trị riêng lẻ cho các thuộc tính của nó và có thể thực hiện các hành động riêng lẻ bằng cách sử dụng các phương thức của class.  Ví dụ, class "Car" có thể chứa các thuộc tính như "màu sắc", "năm sản xuất" và "hãng sản xuất", cùng với các phương thức như "chạy xe" và "dừng lại". Một object được tạo ra từ class "Car" có thể có các giá trị riêng lẻ cho các thuộc tính của nó như "màu đỏ", "năm sản xuất 2021" và "hãng sản xuất Toyota", và có thể thực hiện các hành động riêng lẻ bằng cách sử dụng các phương thức của class "Car" như "chạy xe" và "dừng lại". | |
| 38 | | Các đặc điểm trong Lập trình hướng đối tượng. Đưa ra 1 ví dụ 1 trong 4 tính chất. (lưu ý cần phải trình bày được 4 tính chất bằng tiếng anh)  Có bốn đặc điểm chính trong lập trình hướng đối tượng, gồm:   1. Encapsulation (Đóng gói): đảm bảo rằng dữ liệu và phương thức được đóng gói trong một đối tượng và không thể truy cập trực tiếp từ bên ngoài đối tượng. 2. Inheritance (Kế thừa): cho phép một class kế thừa các thuộc tính và phương thức của một class khác, tạo ra một sự tương đồng giữa các class. 3. Polymorphism (Đa hình): cho phép một phương thức có thể được triển khai bởi nhiều class khác nhau, với các hành vi khác nhau cho mỗi class đó. 4. Abstraction (Trừu tượng): giúp che giấu các chi tiết cài đặt phức tạp của một đối tượng và tập trung vào các chức năng cơ bản của đối tượng đó.   Ví dụ về tính chất Inheritance (Kế thừa) là:  Một class "Animal" có thể có các thuộc tính và phương thức chung của tất cả các loại động vật, chẳng hạn như "loài", "tuổi", "cân nặng" và "ăn".  Một class "Cat" có thể kế thừa các thuộc tính và phương thức của class "Animal" và có thêm các thuộc tính và phương thức đặc biệt cho loài mèo như "hình dạng tai" và "chỉ số IQ".  Tương tự, một class "Dog" cũng có thể kế thừa các thuộc tính và phương thức của class "Animal" và có thêm các thuộc tính và phương thức đặc biệt cho loài chó như "hình dạng mõm" và "khả năng chăm sóc gia đình". | |
| 39 | | Constructor là gì? Trong 1 class có nhiều hơn 1 contrustor được hay không?  Trong lập trình hướng đối tượng, constructor là một phương thức đặc biệt trong class được sử dụng để khởi tạo các đối tượng của class đó. Khi một đối tượng mới được tạo, constructor sẽ được gọi tự động để khởi tạo các thuộc tính của đối tượng đó.  Constructor có tên giống với tên của class và được khai báo bằng từ khóa **constructor**. Constructor có thể có hoặc không có tham số, tùy thuộc vào cách sử dụng.  Một class chỉ có thể có một constructor. Tuy nhiên, trong một constructor, có thể sử dụng từ khóa **this** để gọi các phương thức khác của class hoặc khởi tạo các thuộc tính của đối tượng bằng các giá trị mặc định được định nghĩa trước.  Ví dụ:  class Person {  constructor(name, age) {  this.name = name;  this.age = age;  }  sayHello() {  console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);  }  }  let person1 = new Person("John", 25);  person1.sayHello(); // output: "Hello, my name is John and I am 25 years old."  Trong ví dụ này, **Person** là một class có một constructor nhận hai tham số là **name** và **age**. Constructor này sử dụng từ khóa **this** để gán giá trị cho thuộc tính **name** và **age** của đối tượng được tạo. Phương thức **sayHello** được sử dụng để in ra màn hình thông tin của đối tượng được tạo. Cuối cùng, một đối tượng mới **person1** được tạo bằng cách gọi constructor của **Person** và được sử dụng để gọi phương thức **sayHello()**. | |
| 40 | | Những phương thức nào cho phép tương tác với chuỗi  Trong JavaScript, có nhiều phương thức cho phép tương tác với chuỗi, bao gồm:   1. **charAt()**: trả về ký tự ở vị trí chỉ định trong chuỗi. 2. **concat()**: kết hợp hai hoặc nhiều chuỗi lại với nhau và trả về một chuỗi mới. 3. **indexOf()**: trả về vị trí đầu tiên của ký tự hoặc chuỗi con được tìm thấy trong chuỗi ban đầu. 4. **lastIndexOf()**: trả về vị trí cuối cùng của ký tự hoặc chuỗi con được tìm thấy trong chuỗi ban đầu. 5. **slice()**: trả về một phần của chuỗi bằng cách xác định vị trí bắt đầu và kết thúc. 6. **substring()**: trả về một phần của chuỗi bằng cách xác định vị trí bắt đầu và kết thúc, tuy nhiên phương thức này không chấp nhận các vị trí âm. 7. **replace()**: thay thế ký tự hoặc chuỗi con đầu tiên được tìm thấy trong chuỗi ban đầu bằng chuỗi mới được chỉ định. 8. **replaceAll()**: thay thế tất cả các ký tự hoặc chuỗi con được tìm thấy trong chuỗi ban đầu bằng chuỗi mới được chỉ định. 9. **toLowerCase()**: chuyển đổi chuỗi thành dạng chữ thường. 10. **toUpperCase()**: chuyển đổi chuỗi thành dạng chữ hoa. 11. **trim()**: loại bỏ khoảng trắng ở đầu và cuối chuỗi.   Và còn nhiều phương thức khác nữa. | |
| 41 | | Các câu lệnh để vẽ 1 hình tròn. Giải thích từng câu lệnh?  Để vẽ một hình tròn bằng canvas, chúng ta có thể sử dụng các câu lệnh sau đây trong HTML và JavaScript:   1. **var canvas = document.getElementById("myCanvas");**: Lấy đối tượng HTML Canvas thông qua ID của nó. 2. **var context = canvas.getContext("2d");**: Lấy đối tượng CanvasRenderingContext2D để vẽ hình tròn trên Canvas. 3. **context.beginPath();**: Bắt đầu một đường mới hoặc đường viền cho hình tròn. 4. **context.arc(x, y, radius, startAngle, endAngle, anticlockwise);**: Vẽ hình tròn tại tọa độ (x, y) với bán kính là **radius**. **startAngle** và **endAngle** định nghĩa góc bắt đầu và góc kết thúc của hình tròn (tính theo radian). **anticlockwise** là một giá trị boolean xác định hướng vẽ của hình tròn (chiều kim đồng hồ hoặc ngược chiều kim đồng hồ). 5. **context.fillStyle = color;**: Thiết lập màu sắc của hình tròn. 6. **context.fill();**: Đổ màu vào hình tròn.   Ví dụ đầy đủ để vẽ một hình tròn bằng canvas như sau:  <canvas id="myCanvas" width="200" height="200"></canvas>  <script>  var canvas = document.getElementById("myCanvas");  var context = canvas.getContext("2d");  var x = canvas.width / 2;  var y = canvas.height / 2;  var radius = 50;    context.beginPath();  context.arc(x, y, radius, 0, 2 \* Math.PI, false);  context.fillStyle = "red";  context.fill();  </script> | |