**Câu 1: Phân biệt var – let – const ?**

|  |  |  |
| --- | --- | --- |
| Var | Let | Const |
| Là từ khóa định nghĩa 1 biến có phạm vi truy cập trong 1 function  + Có thể khai báo đa dạng kiểu number, string , boolean  + Có thêm tính chất hoisting => dù khai báo ở đâu cũng được đem lên đầu scope trước khi code thực hiện | Là từ khóa định nghĩa 1 biến có phạm vi trong 1 block-khối code {}. Biến let có phạm vi trong 1 cặp dấu ngoặc bao quanh nó.  + Let cho chúng ta cập nhật lại giá trị biến chứ không cho chúng ta khai báo lại biến đó  + Cũng có tính chất hosting nhưng khác ở chỗ giá trị khai báo var là undefined còn let thì không có giá trị nào => khi sử dụng sẽ gặp giá trị “Reference error”. | Tương tự như biến let nhưng nếu trường hợp primitive(string, number, null và undefined) thì không thể khai báo hay cập nhật giá trị trước đó. Đó với biến reference (bao gồm Object, array và funciton) thì không thể cập nhật giá trị biến nhưng cũng có thể cập nhật thuộc tính của biến.  + Dùng const khi muốn khai báo hằng số và giúp cho chúng ta biết được khi nào đột biến về giá trị => từ đó hạn chế về giá trị,=> với yêu cầu rõ ràng => từ khóa var không phù hợp. |

**Câu 2: So sánh “==” và “===” ?**

* “==”: toán tử so sánh trừu tượng
* “===”: toán tử so sánh cân bằng nghiêm ngặt.

=> So sánh “===” sẽ thêm điều kiện so sánh về kiểu dữ liệu (so sánh loại dữ liệu)

**Câu 3: CallBack là gì ? Tại sao phải dùng CallBack ?**

+ Gọi lại 1 function sẽ thực thi sau khi một function khác được thực thi xong.

EX: Bỏ funciton A vào funciton B, chạy xong funciton B thì nhớ gọi lại funciton A.

+ Sử dụng callBack vì: Javascipt là ngôn ngữ xây dựng theo mô hình event-loop, là ngôn ngữ bất đồng bộ (Non-blocking). Javascript sẽ tiếp tục thực thi các lệnh tiếp theo, đồng thời phải chờ đợi phản hồi từ các sự kiện khác. Để đảm bảo đúng tuần tự định sẵn, ta cần sử dụng khái niệm callback funciton. CallBack đảm bảo code không hoạt động trước khi các code khác được thực thi.

**Câu 4: Promise là gì? Tại sao phải dùng CallBack ?**

+ Là một cơ chế trong javascript giúp bạn thực thi các tác vụ bất đồng bộ mà không rơi vào callback hell hay pyramid of Dom

\* CallBack hell là trường hợp các callback lồng nhau quá nhiều tầng.

+ Tạo Promise: const p = new Promise(function(resolve,reject))

+ Resolve: là hàm được gọi đến khi Promise hoàn thành.

+ Reject: là hàm được gọi khi có lỗi xảy ra.

+ Chúng ta truy suất kết quả trả về bằng phương thức **then.** Phương thức ‘then’ nhận vào 2 hàm onSuccess và onError.

* OnSucces: khi Promise hoàn thành
* onError: khi Promise thất bại
* Có thể trả ra giá trị đồng bộ chẳng hạn như giá trị số, chuỗi, null, undefined, array, object hay promise object cho phép kết nối nhiều Promise khác.

**Câu 5: Async/Await là gì ?**

+ Là một tính năng của JS, giúp chúng ta là việc với bất đồng bộ theo cách dễ dàng hơn.

* Async/await là cách mới để viết code bất đồng bộ. Các phương pháp làm việc với code bất đồng bộ trước đây là sử dụng callback và promise.
* Async/await là khái niệm được xây dựng ở tầng trên promise. Do đó nó không thể sử dụng với callback thuần.
* Async/await cũng giống như promise, là non-blocking.
* Async/await làm cho code bất đồng bộ nhìn và chạy gần giống như code đồng bộ.

+ Async:

* Tự động biến đổi một hàm thông thường thành một Promise.
* Khi gọi tới hàm async nó sẽ xử lý mọi thứ và được trả về kết quả trong hàm của nó.
* Async cho phép sử dụng Await.

+ Await:

* Khi được đặt trước một Promise, nó sẽ đợi cho đến khi Promise kết thúc và trả về kết quả.
* Await chỉ làm việc với Promises, nó không hoạt động với callbacks.
* Await chỉ có thể được sử dụng bên trong các function async.