|  |
| --- |
| Hiểu được nguồn gốc ra đời, lịch sử phát triển các phiên bản cũng như đặc trưng của ngôn ngữ lập trình JavaScript |
| Hiểu khái niệm về biến và các kiểu dữ liệu trong JavaScriptĐịnh nghĩa và ví dụ về biến (2:05) - Các biến được dùng để lưu trữ các giá trị để tái sử dụng các giá trị nhiều lần.  - Để lưu trữ một giá trị vào một biến cần khai báo biến và gán giá trị vào cho biến đó.  Giá trị lưu trữ trong JS có 2 kiểu: Đối tượng (Object) và dữ liệu nguyên thủy (Primitive)   * + 7 kiểu dữ liệu nguyên thủy (0:48)   - Ba dạng dữ liệu cơ bản nhất : number, string, boolean  - Bốn dạng dữ liệu ít sử dụng hơn: undefined, null, symbol và bigint |
| Biết cách khai báo biến, phân biệt giữa let const và var Từ khóa let để khai báo các biến có thể thay đổi giá trị được  - Các biến này còn được gọi là mutable variable (biến có thể thay đổi)  - Từ khóa const để khai báo các biến chỉ gán giá trị một lần và không thể thay đổi giá trị sau đó  - Các biến này còn được gọi là immutable variable (biến bất biến) |
| Vận dụng được các toán tử cơ bản, độ ưu tiên giữa các toán tử - Các toán tử toán học cơ bản như : +, -, \*, /, \*\* (lũy thừa)   * + Phép nối các string (6:00)   - Phép nối chuỗi sử dụng toán tử cộng +   * + Phép gán (8:20)   - Toán tử gán =, += , \*= , .... Ví dụ: x += 10 // có nghĩa là x = x + 10  - Toán tử gán ++ , --. Ví dụ: x++ // có nghĩa là x = x + 1   * + Phép so sánh (12:30)   - Các toán tử so sánh quen thuộc như : ==, > , < , >=, <= |
| Hiểu rõ về câu lệnh if-else, switch if (scoreDolphins > scoreKoalas && scoreDolphins >= 100) {    console.log("Dolphins 🏆");  } else if (scoreDolphins < scoreKoalas && scoreKoalas >= 100) {    console.log("Koalas 🏆");  } else if (    scoreDolphins === scoreKoalas &&    scoreDolphins >= 100 &&    scoreKoalas >= 100  ) {    console.log("Both win the trophy! 🏆");  }  const language1 = prompt("Enter Language: ");  // switch (language1) {  //   case "Chinese":  //     console.log("MOST number of native speakers!");  //     break;  //   case "Mandarin":  //     console.log("MOST number of native speakers!");  //     break;  //   case "Spanish":  //     console.log("2nd place in number of native speakers");  //     break;  //   case "English":  //     console.log("3rd place");  //     break;  //   case "Hindi":  //     console.log("Number 4");  //     break;  //   case "Arabic":  //     console.log("5th most spoken language");  //     break;  //   default:  //     console.log("Great language too :D");  //     break;  // } |
| Biết cách khai báo hàm, cách gọi hàm và một số cơ chế trong hàm  * + Định nghĩa hàm (0:00)   - Hàm đơn giản là một đoạn code mà chúng ta có thể sử dụng lặp đi lặp lại trong code  - Hàm hơi giống biến nhưng nó là một tập các đoạn code, biến chứa giá trị còn hàm chưa một hoặc nhiều dòng code hoàn chỉnh  - Khai báo hàm bao gồm tên hàm, tham số truyền vào, thân hàm  // Functions  function logger() {    console.log('My name is Jonas');  }  // calling / running / invoking function  logger();   * + Câu lệnh **return** (3:25)   - Có thể truyền tham số đầu vào cho một hàm và hàm cũng có thể trả về một giá trị  - Sử dụng từ khóa return để trả về giá trị cho một hàm   * + Tham số/đối số đầu vào cho hàm (7:30)   - Các tham số truyền vào (đối số) có vai trò như các biến có phạm vi sử dụng trong hàm  function fruitProcessor(apples, oranges) {    const juice = Juicewith${app≤s}app≤sand${oran≥s}oran≥s.;    return juice;  }  const appleJuice = fruitProcessor(5, 0);  console.log(appleJuice);   * + Khai báo hàm (0:00)   Cú pháp khai báo hàm cơ bản:  function tên\_hàm(các\_tham\_số) {  // thân hàm  }   * + Biểu thức hàm (4:10)   Cú pháp tạo một biểu thức hàm khác khai báo hàm ở chỗ nó không có tên hàm và nó được gán cho một biến, tên biến đó chính là tên hàm  Ví dụ :  // Function expression  const calcAge2 = function (birthYeah) {    return 2037 - birthYeah;  }  const age2 = calcAge2(1991);   * + Khác biệt giữa khai báo hàm và biểu thức hàm về vị trí sử dụng (7:40)   - Khác biệt lớn nhất giữa khai báo hàm và biểu thức hàm là với khai báo hàm, ta có thể gọi các khai báo hàm trong code trước khi chúng được định nghĩa |
| Nắm được các khái niệm về Object trong JavaScript  * + Giới thiệu về object với key-value (1:35)   - Object quy định các phần tử theo cặp key-value  - Key giống như tên biến  - Value có thể thuộc bất kỳ kiểu dữ liệu nào, có thể gán value là một biểu thức. Mỗi cặp key-value trong object được gọi là thuộc tính của object   * + Ví dụ về object (2:05)   - Array:  https://firebasestorage.googleapis.com/v0/b/funix-way.appspot.com/o/xSeries%2FPRF192x%2FContent_Image%2FPRF192x_L5_H%C3%ACnh%201.PNG?alt=media&token=805f21ff-acd8-4ced-a72a-0fda493e143d  - Object:  https://firebasestorage.googleapis.com/v0/b/funix-way.appspot.com/o/xSeries%2FPRF192x%2FContent_Image%2FPRF192x_L5_H%C3%ACnh%202.PNG?alt=media&token=486e1cd7-4d77-4be3-bd51-45facfef8e67   * 2 cách lấy giá trị của thuộc tính: dấu chấm và ngoặc vuông (1:05)   Ví dụ:  Sử dụng dấu chấm: jonas.lastName  Sử dụng dấu ngoặc vuông: jonas['lastName']   * + Khi nào thì chúng ta nên sử dụng cách nào (5:20)   - Khi sử dụng ngoặc vuông ta có thể đặt bất kỳ biểu thức nào bên trong nó, linh hoạt trong một số trường hợp.  - Khi sử dụng dấu chấm ta phải biết chính xác tên thuộc tính.   * + Đưa thêm thuộc tính vào object (11:45)   Có thể sử dụng dấu chấm hoặc dấu ngoặc vuông để chỉ định thêm một thuộc tính cho object, đơn giản là gán giá trị cho thuộc tính đó.  Ví dụ:  https://firebasestorage.googleapis.com/v0/b/funix-way.appspot.com/o/xSeries%2FPRF192x%2FContent_Image%2FPRF192x_L5_H%C3%ACnh%204.PNG?alt=media&token=bdc0ebcb-8842-4646-982b-b36d506a901f   * + Cú pháp **this**trong object (6:00)   - Để truy cập vào giá trị của một thuộc tính ngay trong object thì sử dụng tử khóa this  - Từ khóa this hay biến this về cơ bản là object (đối tượng) hiện tại của phương thức được gọi   * + Tại sao lại cần **this**để trỏ vào object?(10.20)   - Cần this trỏ vào object để lấy ra được những thuộc tính cần để tính toán mà không cần phải truyền biến từ bên ngoài vào   Tạo một thuộc tính trong phương thức ta dùng từ khóa this.ten\_thuoc\_tinh và gán giá trị vào cho nó. Ví dụ:  https://firebasestorage.googleapis.com/v0/b/funix-way.appspot.com/o/xSeries%2FPRF192x%2FContent_Image%2FPRF192x_L5_H%C3%ACnh%206.PNG?alt=media&token=2a984cc6-5fb6-4ce8-8ffe-b2432b0411c1 |
| Sử dụng các vòng lặp for, while cũng với các từ khóa break và continue  * + Định nghĩa về vòng lặp (0:00)   - Vòng lặp là một cấu trúc điều khiển  - Vòng lặp cho phép tự động hóa các tác vụ lặp đi lặp lại   * + Cấu trúc vòng lặp for và ví dụ (2:25)   - Vòng lặp bắt đầu với từ khóa for, biểu thức vòng lặp và phần thân  - Biểu thức vòng lặp gốm 3 phần, cách nhau bởi dấu chấm phẩy:  1. Giá trị ban đầu của bộ đếm  2. Điều kiện logic được đánh giá trước mỗi lần lặp, nếu điều kiện là true thì vòng lặp sẽ tiếp tục, nếu điều kiện là false thì vòng lặp sẽ dừng lại  3. Tăng bộ đếm  Ví dụ:  https://firebasestorage.googleapis.com/v0/b/funix-way.appspot.com/o/xSeries%2FPRF192x%2FContent_Image%2FPRF192x_L5_H%C3%ACnh%207.PNG?alt=media&token=a5349608-3575-485f-b2b6-fea7edcd66d9   * + ử dụng vòng lặp for để duyệt các phần tử của array (1:00)   + Sử dụng vòng lặp for để duyệt qua các phần tử của mảng (array)   - Thứ tự của các phần tử trong array bắt đầu từ 0 nên khi duyệt bằng vòng lặp for, bộ đếm nên bắt đầu từ 0  Ví dụ sử dụng for để duyệt qua các phần tử trong mảng:  https://firebasestorage.googleapis.com/v0/b/funix-way.appspot.com/o/xSeries%2FPRF192x%2FContent_Image%2FPRF192x_L5_H%C3%ACnh%208.PNG?alt=media&token=fc0e982e-5f77-4ccd-a23e-d976df916696   * + Câu lệnh **continue** (16:40)   - Continue là việc bỏ qua việc lặp lại hiện tại của vòng lặp mà thực hiện lần lặp tiếp theo.  Ví dụ:  const arr4 = [1, 2, 3, "asd", 2, 8, "qwe","zxc"];  for (let index = 0; index < arr4.length; index++) {    if (typeof arr4[index] !== "string") {      continue;    }    console.log(arr4[index], typeof arr4[index]);  }   * + Câu lệnh **break**(19:55)   - Break được sử dụng để dừng lại hoàn toàn vòng lặp  Ví dụ:  const arr4 = [1, 2, 3, "asd", 2, 8, "qwe","zxc"];  for (let index = 0; index < arr4.length; index++) {    if (typeof arr4[index] === "string") {      break;    }    console.log(arr4[index], typeof arr4[index]);  }   * + So sánh vòng lặp while và for (1:10)   - Vòng lặp while cũng có các phần giống vòng lặp for, vẫn có biến đếm, một điều kiện để so sánh và tăng / giảm biến đếm bằng cách nào đó.  - Tuy nhiên vòng lặp while khác với for là ở biểu thức vòng lặp chỉ cần điều kiện lặp.  Ví dụ:  https://firebasestorage.googleapis.com/v0/b/funix-way.appspot.com/o/xSeries%2FPRF192x%2FContent_Image%2FPRF192x_L5_H%C3%ACnh%2012.PNG?alt=media&token=207411fe-f9f1-46f0-b0aa-79048c47b54b |
| HIểu khái niệm về mảng cũng với các toán tử trong mảng  * + Giới thiệu về array (0:00)   - Array (mảng) là một cấu trúc dữ liệu quan trọng trong JS, cho phép lưu trữ các giá trị khác nhau vào trong một biến duy nhất  Có 2 cách khai báo mảng:  1. const array = [ item1, item2 ]  2. const array = new Array(item1, item2)  [**Các phương thức cơ bản của array**](https://funix.udemy.com/course/the-complete-javascript-course/learn/lecture/22648255#overview)   * **push** – thêm một phần tử vào cuối array (0:50)   const friends = ['Michael', 'Steven', 'Peter'];  // Add elements  const newLength = friends.push('Jay'); // 4   * **unshift** – thêm một phần tử vào đầu array (3:55)   Ví dụ: const newLength = friends.unshift('John');   * **pop** – gỡ một phần tử khỏi đuôi array (5:00)   Ví dụ: const popped = friends.pop();   * **shift**– gỡ một phần tử khỏi đầu array (7:05)   Ví dụ: friends.shift(); // First  **indexof**- chỉ ra vị trí của 1 phần tử trong array (8:00)  Hàm index trả về vị trí của phần tử trong mảng và trả về -1 nếu không tìm thấy. Ví dụ:  const index = friends.indexOf('Steven') // 1  const index = friends.indexOf('Bob') // -1   * + **includes** – chỉ ra phần tử có xuất hiện trong array không (9:15)   Hàm includes trả về giá trị true/ false. Phương thức này sử dụng phép so sánh ===.  Ví dụ:  friends.includes('Steven') // true  friends.includes('Steven') // false |
| Hiểu được cách viết code tốt, các kỹ năng giải quyết vấn đề để trở thành một lập trình viên chuyên nghiệp |
| Thiết lập được các công cụ để hỗ trợ khi viết code JavaScript nhanh hơn và dễ đọc/bảo trì code hơn |
| Hiểu về sử dụng được các công cụ Debugging |
| [HIểu được khái về DOM và DOM Manipulation](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/962395c555594598b99d125b04c14e1c/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40962395c555594598b99d125b04c14e1c)  * + Định nghĩa DOM – document object model   - DOM (Document Object Model) là một biểu diễn có cấu trúc của tài liệu HTML  - DOM cho phép chúng ta sử dụng JS để truy cập vào các phần tử và style (CSS) để sử dụng chúng. Nói các khác, DOM là một điểm kết nối giữa tài liệu HTML và code JS  - DOM được trình duyệt tạo tự động ngay sau khi tải trang HTML, và được lưu trữ dưới dạng cây DOM  Trong cây DOM, mỗi phần tử HTML là một object hay còn gọi là một node. Document là object đầu tiên (root) của cây DOM, nó là điểm truy cập của cây DOM. Phần tử con tiếp theo thường là html và nó tiếp tục có 2 phần tử con là head và body (2 phần tử này là 2 phần tử liền kề, anh em). Đi tiếp xuống dưới sẽ thấy nhiều các phần tử html lồng nhau và chúng được biểu diễn trong cây DOM.  - Cây DOM không chỉ có các node phần tử mà còn có các node cho text, command,... Tất cả những thứ có trong tài liệu HTML đều được biểu diễn trong DOM   * + (4:20) Quan hệ giữa DOM với JS và trình duyệt web   - DOM và các phương thức của phần tử DOM là một phần của WEB API (là các thư viện viết bằng JS mà trình duyệt triển khai mà chúng ta có thể truy cập từ code JS) |
| [Thao tác được việc truy vấn các phần tử trên DOM Tree](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/962395c555594598b99d125b04c14e1c/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40962395c555594598b99d125b04c14e1c) |
| [Biết cách xử lý một số Event trong DOM](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/962395c555594598b99d125b04c14e1c/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40962395c555594598b99d125b04c14e1c)  * + Khái niệm Event và EventListener   - Event là những sự kiện xảy ra trên trang như click chuột, di chuột, bấm phím,...  - EventListener là việc lắng nghe sự kiện xảy ra và phản hồi nó   * + (2:05) Phương thức **addEventListener**   Phương thức addEventListener của một phần tử DOM nhận đầu vào là 2 tham số :  - Sự kiện  - Hàm xử lý sự kiện (biểu thức hàm). Hàm xử lý sự kiện chỉ được gọi khi sự kiện xảy ra  Ví dụ:  document.querySelector('.check').addEventListener('click', function () {     const guess = Number(document.querySelector('.guess').value);     console.log(guess, typeof guess);  }) |
| [Vận dụng được nguyên tắc Dry để tối ưu hóa mã nguồn](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/962395c555594598b99d125b04c14e1c/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40962395c555594598b99d125b04c14e1c) |
| Sử dụng JavaScript để thay đổi các thuộc tính trên DOM Tree DOM cũng bao gồm cả CSS style và chúng ta cũng có thể thay đổi style của trang HTML  Sử dụng **document.querySelector**để trỏ vào body   * + Sử dụng thuộc tính **style** với 2 thuộc tính **backgroundColor** và **width**   Lưu ý khi truyền giá trị vào cho style, style phải gọi tên theo camelCase và giá trị truyền vào luôn là string.  Ví dụ:  document.querySelector('body').style.backgroundColor = '#60b347';  document.querySelector('.number').style.width = '30rem';" |
| Nắm rõ được quá trình thực thi một chương trình Javascript |
| [Nắm được các khái niệm như JS engine. JS runtime, Web API, Callback queue](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/6cf62c9f160a4a0883d287029e06161f/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%406cf62c9f160a4a0883d287029e06161f)  * + Định nghĩa JS engine   - Javascript Engine là một chương trình máy tính thực thi code Javascript.  - Mỗi trình duyện hiện nay đều có Javascript Engine, công cụ được biết đến nhiều nhất là Google V   * + Cấu trúc JS engine   - Bất kỳ JS engine nào cũng chứa callstack và heap.  - CallStack là nơi code được thực thi bằng cách sử dụng Execution Context.  - Heap là một vùng nhớ (memory pool) không có cấu trúc, lưu trữ tất cả các đối tượng mà ứng dụng cần.   * + JS runtime   Javascript Runtime trong trình duyệt như một hôp, thùng chứa lớn chứa tất cả những thứ cần thiết để sử dụng Javascript. Bao gồm:  - JS Engine  - Web APIs  - Callback Queue   * + Web APIs   - JS Runtime bao gồm Web APIs (các chức năng được cung cấp cho engine).  - JS có quyền truy cập vào các API này thông qua đối tượng global window.   * + Callback Queue   - Callback Queue là cấu trúc dữ liệu có chứa tất cả các hàm callback sẵn sàng để thực thi.  Ví dụ: Hàm được truyền vào để xử lý sự kiện trên một phần từ DOM cũng là một hàm callback.  - Sau khi một event xảy ra, hàm callback được đặt vào Callback Queue, sau đó khi stack rỗng, hàm callback được đặt vào stack để nó thực thi, quá trình này xảy ra theo cơ chế event loop.   * + Event Loop   - Event Loop nhận các hàm callback từ Callback Queue và đưa chúng vào Call Stack để thực thi.   * + JS không chỉ chạy trên trình duyệt   - Tồn tại nhiều kiểu JS Runtime khác nhau và JS không chỉ chạy trên trình duyệt. |
| [Có kiến thức cơ bản về Execution Context, Call Stack trong JavaScript](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/6cf62c9f160a4a0883d287029e06161f/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%406cf62c9f160a4a0883d287029e06161f)  * + Quá trình thực thi chi tiết của 1 chương trình JS   Sau khi quá trình biên dịch hoàn tất, mã máy được đưa vào quá trình thực thi:  1. Tạo ra global excecution context (ngữ cảnh thực thi toàn cục) - dành cho những code thuộc top-level (code thuộc top-level là những code không nằm trong bất kỳ function nào)  - Execute context là ngữ cảnh thực thi định nghĩa là môi trường thực thi đoạn code JS, nó giống như một chiếc hộp lưu giữ tất cả các thông tin cần thiết để một code được thực thi như là các biến cục bộ hay các đối số truyền vào.  - Code JS luôn chạy trong một Execution Context.  - Trong bất kỳ project JS nào cũng chỉ có một global execution context là context mặc định để thực thi các code thuộc top level.  2. Thực thi top level code bên trong GC context  3. Thực thi các hàm và chờ callback  - Với mỗi hàm sẽ có một execution context.  - Các EC này kết hợp vs nhau tạo thành Call Stack.   * + (5:20) Bên trong ngữ cảnh thực thi (Execution Context)   Bên trong Execution Context bao gồm:  1. Variable Environment chứa các biến, khai báo hàm và cũng có một object argument đặc biệt (chứa tất cả các đối số được truyền vào hàm thuộc về EC hiện tại)  2. Scope Chain bao gồm các tham chiếu đến các biến nằm ngoài hàm hiện tại  3. Từ khóa this  Lưu ý: EC thuộc về hàm mũi tên không lấy được arguments và cũng không có this.   * + (8:10) Ví dụ về Execution Context khi thực thi 1 chương trình JS   + Call Stack trong JS   - Call Stack là nơi các Execution Context xếp chồng lên nhau, EC ở trên cùng của stack là thứ hiện đang chạy và khi chạy xong nó sẽ bị xóa khỏi stack.  - Việc thưc thi diễn ra theo nguyên lý của Stack - Last In First Out. |
| [Hiểu về Scope và 3 dạng Scope trong Javascript](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/6cf62c9f160a4a0883d287029e06161f/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%406cf62c9f160a4a0883d287029e06161f)  * + Các khái niệm về scope   - Scoping kiểm soát cách JS Engine sắp xếp và truy cập các biến trong chương trình.  - Lexical scoping là cách tổ chức và truy cập các biến hoàn toàn bị vị trí của các hàm và các khối trong code kiểm soát. Ví dụ một hàm được viết trong một hàm khác có quyền truy cập các biến của hàm cha. Scope bị ảnh hưởng bởi vị trí mà chúng ta đặc các hàm và khối code của mình.  - Scope là không gian hoặc môi trường khai báo một biến nhất định.  - Scope của một biến là toàn bộ khu vực code có thể truy cập vào biến đó.   * + Ba dạng scope trong JS   Trong JS có 3 loại scope: global scope, function scope và block scope:  - Global Scope là các top-level code, các biến được khai báo bên ngoài bất kỳ hàm hoặc khối nào. Các biến này có thể được truy cập ở mọi nơi trong chương trình.  - Function Scope: mỗi một hàm đều tạo một scope và các biến khai báo bên trong phạm vi của hàm đó chỉ có thể được truy cập bên trong hàm. Đây còn được gọi là local scope.  - Block Scope: Từ ES6, các khối block cũng tạo ra một scope, code block tất cả mọi thứ nằm trong dấu ngoặc nhọn {} (ví dụ: khối lệnh if, vòng lặp for). Giống function scope, tất cả các biến được khai báo trong block chỉ có thể được truy cập bên trong block đó, không thế truy cập từ bên ngoài.  Khác biệt giữa function scope và block scope là block scope chỉ áp dụng cho các biến được khai báo bằng let, const và var là function scope.  Ngoài ra, tất cả các scope đều có quyền truy cập vào các biến từ các scope bên ngoài của chúng. |
| [Áp dụng được từ khóa this](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/edefe8f7a18a4d35aa88edb3c0bdc39a/?child=first)  * + Định nghĩa **this**   - Biến this là một biến đặc biệt được tạo trong mọi ngữ cảnh thực thi (Execution Context) và cho mọi function nào.  - This trỏ tới giá trị của chủ sở hữu của hàm nơi mà this được gọi.  - This không phải static, nó phụ thuộc vào cách mà hàm được gọi - giá trị của this chỉ được gán khi hàm thực sự được gọi.  Có 4 kiểu hàm:  - Hàm là một phương thức của một đối tượng: Khi gọi this thì giá trị của this sẽ trỏ đến đối tượng mà phương thức của đối tượng đó được gọi.  const person = {          name: "asdasdqwe",          brithYear: 1991,          calcAge: function () {            return 2037 - this.brithYear;          },        };        const a = person.calcAge();        console.log(a);  - Hàm là một function bình thường, không gắn với một object, khi gọi this sẽ trả về undefined đối với strict-mode. Nếu không đặt trong chế độ strict-mode, this sẽ trả về global object trong trình duyệt là đối tượng window.        const calcAge1 = function (birthYear) {          console.log(2037 - birthYear);          console.log(this);//=>undefined        };        calcAge1(1991)  - Hàm là một arrow function (lexical this). Biến this sẽ có giá trị là giá trị của hàm bao bên ngoài nó, hàm parent.  const calcAge2 = (birthYear) => {          console.log(2037 - birthYear);          console.log(this); //=>Window {window: Window, self: Window, document: document, name: '', location: Location, …}        };        calcAge2(1991);  - Hàm là một Event Listener, từ khóa this sẽ luôn trỏ đến phần tử DOM mà hàm xử lý được gắn vào.  Ngoài các cách gọi này còn có call, new, apply, bind, ... (sẽ trình bày sau). |
| [Phân biệt được Arrow Function và Regular Function](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/edefe8f7a18a4d35aa88edb3c0bdc39a/?child=first)  * + **this**trong hàm mũi tên   - Biến this trong hàm mũi tên không trỏ đến đối tượng gọi nó mà nhận những giá trị từ hàm bao ngoài.  - Không nhầm lẫn giữa khai báo object với một block scope.  - Biến this ở global object luôn là window object. |
| [Phân biệt được Primitives với Objects](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/11340a391e3047ecba5b9df264c21e77/edefe8f7a18a4d35aa88edb3c0bdc39a/?child=first)  * + Primitive và object lưu trữ dữ liệu trong JS engine ra sao   - Kiểu dữ liệu nguyên thủy (primitive types): number, string, boolean, undefined, null, symbol, bigint; được lưu trữ trong các Execution Context trong CallStack.  - Kiểu dữ liệu object (reference types): object, arrays, function, ... được lưu trữ trong heap.  - Trong Call Stack, nơi các biến primitive được lưu trữ, giá trị của biến được tham chiếu trực tiếp đến địa chỉ nơi chứa giá trị của biến, khi gán giá trị cho một biến mới, nó sẽ yêu cầu cấp thêm một ô nhớ nữa.  - Khi một Object được tạo ra, nó được lưu trữ trong heap nhưng định danh của object không trỏ tới địa chỉ ô nhớ trong heap mà nó sẽ trỏ tới một ô nhớ mới trong Call Stack, và ô nhớ mới trong stack này sẽ trỏ tới giá trị object được lưu trong heap. Nói cách khác phần bộ nhớ trong call stack tham chiếu tới bộ nhớ heap, vì vậy ta gọi các object là các reference type. Js thực hiện theo cơ chế này vì object có thể quá lớn để lưu trữ trong call stack, thay vào đó nó được lưu vào heap - vùng nhớ không giới hạn. |
| Phân biệt được Pass by Value với và Pass by Reference  Truyền một biến primitive vào trong một hàm giống việc tạo một bản sao biến bên ngoài function.  - Khi truyền một object vào một function nó cũng giống như việc sao chép một object bên ngoài, đều trỏ tới một giá trị object trong bộ nhớ heap. Điều này dễ gây ra lỗi khi hai function cùng thao tác trên một object. |
| Hiểu rõ về First-Class Function và Higher-Order Function  * + First-Class function – gán một biến cho 1 hàm   - JS là ngôn ngữ có các first-class function,  có nghĩa là các function được coi như là các giá trị.  - Các function là một loại object đặc biệt trong JS, coi các function là các giá trị nên có thể:  1. lưu trữ function trong các biến hoặc các thuộc tính của object  2. truyền các function làm đối số cho các function khác (ví dụ addEventListener() trong DOM)  3. trả về một function từ một function khác  4. Gọi phương thức trong function   * + Higher-Order Function - lấy các hàm khác làm đối số hoặc trả về các hàm làm kết quả của chúng   - Higher Order Funtion là một function nhận một function khác như một đối số hoặc một function trả về một function mới.  1. Function là một đối số truyền vào function khác: Function này thường được gọi là callback function, callback function sẽ được gọi sau bởi higher order function  2. Function trả về một function khác (hướng dẫn ở những bài tới)  - First class function chỉ là một tính năng của một ngôn ngữ lập trình có hoặc không có, nó chỉ là một khái niệm.  - Higher order function tồn tại vì nó được xây dựng dựa trên các first class function. |
| Nắm được khái niệm về Callback và Closure trong JavaScript Tại sao hàm callback lại hữu ích và trở nên thông dụng  Giúp dễ dàng phân tách hoặc viết code thành nhiều phần có thể tái sử dụng và kết nối với nhau.  - Tạo ra sự trừu tượng, ẩn chi tiết của một số code được thực thi bởi vì chúng ta không thực sự quan tâm đến những chi tiết đó.  Hàm đầu tiên được gọi là greet('Hey'), hàm này trả về một hàm mới mà chúng ta lưu trữ nó vào biến greeterHey, sau đó gọi hàm như bình thường. Cũng có thể viết là greet('Hey')('Jonas').  - Javascript có thể làm được điều này là nhờ cơ chế closure (sẽ được học sau). |
| Vận dụng được các phương thức gọi hàm như call, apply và bind.  * + Sử dụng **call**   - Phương thức call là phương thức có sẵn của 1 hàm.  - Đối số đầu tiên của phương thức call là đối tượng mà chúng ta muốn từ khóa this trỏ đến.  - Các đối số sau là các đối số muốn truyền vào hàm gốc.        const lufthansa = {          airline: "Lufthanse",          iataCode: "LH",          books: [], //mang          //function          book(flightNumb, name) {            console.log(              `${name} booked a seat on ${this.airline} flight ${this.iataCode}${flightNumb}`            );            //them vao mang "books"            this.books.push({ flight: `${this.iataCode}${flightNumb}`, name });          },        };        lufthansa.book(3216541, "asdfasf");        lufthansa.book(3216542231, "asdasdasd");        console.log(lufthansa);        const eurowins = {          airline: "Eurowins",          iataCode: "EW",          books: [],        };        //su dung ham "call"        //khai bao bien chua "ham" cua "object" muon "call"        const book = lufthansa.book;        /\*        this: Function => la muon this tro vao ham do        \*/        book.call(eurowins, 123123, "xzczxzcxzv");        console.log(eurowins);   * + Sử dụng **apply**   - Phương thức apply gần giống phương thức call tuy nhiên nó không nhận được các đối số sau từ khóa this, thay vào đó nó nhận đối số là một array.  - Đối số đầu tiên của phương thức apply là đối tượng mà chúng ta muốn từ khóa this trỏ đến.  - Đối số thứ 2 là mảng dữ liệu lưu trữ các đối số theo thứ tự.  const lufthansa = {          airline: "Lufthanse",          iataCode: "LH",          books: [], //mang          //function          book(flightNumb, name) {            console.log(              `${name} booked a seat on ${this.airline} flight ${this.iataCode}${flightNumb}`            );            //them vao mang "books"            this.books.push({ flight: `${this.iataCode}${flightNumb}`, name });          },        };        lufthansa.book(3216541, "asdfasf");        lufthansa.book(3216542231, "asdasdasd");        console.log(lufthansa);        const eurowinsApply = {          airline: "eurowinsApply",          iataCode: "EA",          books: [],        };        //apply ta phai co 1 mang        const flightData = [5498, "lkjlklkj"];        //su dung ham "apply"        //khai bao bien chua "ham" cua "object" muon "call"        const bookapply = lufthansa.book;        //apply can truyen vao mot array        bookapply.apply(eurowinsApply, flightData);        console.log(eurowinsApply);   * + Định nghĩa và sự khác biệt của **bind**   - Giống như call, bind cũng cho phép chúng ta đặt từ khóa this theo cách thủ công cho bất kì lệnh gọi hàm nào.  - Sự khác biệt là bind không gọi hàm ngay, nó trả về một hàm mới nơi mà từ khóa this bị ràng buộc. Nó được đặt thành bất kỳ giá trị nào mà chúng ta truyền vào bind.   const bookEW =  book.bind(eurowins)       bookEW(190808,"ljojk")       console.log(eurowins); |
| Áp dụng được các Array Method trong JavaScript (slice, splice, reverse, concat, join, ...)  * + Phương thức **slice**   - Phương thức slice() này trả về một array mới có các phần từ được lấy từ một array gốc.  - Mảng gốc sẽ không bị sửa đổi.   * + (6:50) Sử dụng toán tử spread (dấu 3 chấm)   - Có 2 cách để tạo bản sao từ một array là sử dụng slice và toán tử spread.  const arr4 =[0,1,2,3,4,5,6,7,8]       const arr5 = arr4.slice(1,2)       console.log(arr5);   * + Phương thức **splice**   - Phương thức splice() thay đổi nội dung của một mảng bằng cách loại bỏ hoặc thay thế các phần tử hiện có và / hoặc thêm các phần tử mới vào vị trí.  - Khác với slice, mảng gốc bị sửa đổi.    const arr4 =[0,1,2,3,4,5,6,7,8]       const arr5 = arr4.splice(1,2) // bo di 1 2       console.log(arr4);   * + Phương thức **reverse**   - Phương thức reverse() là phương thức dùng để đảo ngược một mảng.  - Phương thức reverse làm thay đổi mảng gốc.  const arr6= arr4.reverse()       console.log(arr6);   * + Phương thức**concat**   - Phương thức concat () được sử dụng để hợp nhất hai hoặc nhiều mảng.  - Phương thức này không thay đổi các mảng hiện có mà thay vào đó trả về một mảng mới.  const arr4 =[0,1,2,3,4,5,6,7,8]       const arr4\_1 =[10,11]       const arr7= arr4.concat(arr4\_1)       console.log(arr7);   * + Phương thức **join**   - Phương thức join() tạo và trả về một chuỗi mới bằng cách nối tất cả các phần tử trong một mảng (hoặc một đối tượng giống mảng), được phân tách bằng dấu phẩy hoặc một chuỗi phân tách được chỉ định.   - Nếu mảng chỉ có một mục, thì mục đó sẽ được trả về mà không cần sử dụng dấu phân tách.  const arr4 =[0,1,2,3,4,5,6,7,8]       const arr7= arr4.join("+")       console.log(arr7); |
| Sử dụng được các Array Method để biến đổi mảng (map, reduce, filter, ...)  * + Giới thiệu phương thức map   - Phương thức map tạo ra một mảng mới dựa trên mảng ban đầu.  - Phương thức map nhận đầu vào là một array, lặp lại trên array đó và áp dụng hàm callback vào phần tử hiện tại, sau đó lưu giá trị mới vào mảng mới.   const arr = [0, 1, 2, 3, 4, 5, 6, 7, 8];        const arr1 = arr.map((item) => item + 1);        console.log(arr1);   * + (2:15) Giới thiệu phương thức filter   - Phương thức filter lọc một mảng để lấy ra các phần tử thỏa mãn một điều kiện nhất định.  - Tất cả các phần tử vượt qua bài kiểm tra sẽ được đặt trong một array mới.  const arr = [0, 1, 2, 3, 4, 5, 6, 7, 8];        const arr2 = arr.filter(item=>item%2===0)        console.log(arr2);   * + (3:00) Giới thiệu phương thức reduce   - Phương thức reduce tổng hợp các giá trị của các phần tử thành một giá trị duy nhất dựa trên phép toán được truyền vào.      const arr = [0, 1, 2, 3, 4, 5, 6, 7, 8];        const arr3 = arr.reduce((previousValue,currentValue)=>previousValue+currentValue)        console.log(arr3 ); |
| Vận dụng được các thao tác như truy vấn, tạo mới và xóa DOM Element <style>        .hide {          display: none;        }      </style>    </head>    <body>      <div class="test"></div>      <div id="testID">        Test        <p class="displayText hide">asdqweqaqqqqqqqqqqqqqqqsdqweasd</p>      </div>      <input type="text" id="input" />      <br />      <button id="submit">hide</button>      <button id="submitShow">Show</button>  //cau lenh chon        const btn = document.querySelector("#submit");        const btnShow = document.querySelector("#submitShow");        const testID = document.querySelector("#testID");        const displayText = document.querySelector(".displayText");        //su kien "click"        btn.addEventListener("click", function () {          displayText.classList.remove("hide");        });        btnShow.addEventListener("click", function () {          displayText.classList.add("hide");        }); |
| Sử dụng được JavaScript để thay đổi các Style, Attribute và Class trong DOM displayText.style.color="red" |
| Có kiến thức về Event Propagation và Event Delegation |
| Sử dụng được cơ chế DOM Traversing |
| [Hiểu được khái niệm về OOP và 4 đặc trung cơ bản của OOP trong JS](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/af29b39aa91a43f086d98199a9e3291e/f4caf7b56eb0428c9abf8ffae2ea0fbb/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40f4caf7b56eb0428c9abf8ffae2ea0fbb)  * + Lập trình hướng đối tượng (OOP) là gì?   - Lập trình hướng đối tượng là mô hình lập trình dựa trên khái niệm đối tượng. Mô hình lập trình sẽ quy định phong cách code, cách viết và tổ chức code.  - Sử dụng mô hình đối tượng để mô hình hóa, mô tả các khía cạnh trong thế giới thực  - Các object có thể chứa dữ liệu bao gồm thuộc tính (properties) và phương thức - method (code).  - Bằng cách sử dụng object chúng ta đóng gói tất cả dữ liệu và hành vi tương ứng thành một khối dễ dàng thao tác.  - Trong OOP, các đối tượng là các đoạn/khối mã độc lập; Các đối tượng là các khối xây dựng của ứng dụng và tương tác với nhau.  - Tương tác xảy ra thông qua giao diện công khai - public interface (API) -  các phương thức mà code bên ngoài đối tượng có thể truy cập và sử dụng để giao tiếp với đối tượng.  - OOP tồn tại để đáp ứng mục tiêu cấu trúc, tổ chức code linh hoạt hơn và dễ dàng bảo trì.  - Hiện nay OOP là mô hình lập trình được sử dụng rộng rãi nhất trong kỹ thuật lập trình.   * + (4:20) Class (lớp) và instance trong OOP truyền thống   - Chúng ta đã sử dụng object trong những buổi học trước nhưng chúng ta mới có những object lỏng lẻo, chưa tương tác với nhau.  - Trong OOP chúng ta cần một cách thống nhất để tạo ra các object mới từ code của mình.  - Class như một bản thiết kế chi tiết, sau đó tạo ra object mới dựa trên các quy tắc được quy tắc trong class.  - Các object được tạo từ class được gọi là instance của class, instance là một đối tượng thực có các giá trị thuộc tính và hành vi thực.   * + (8:35) 4 đặc tính cơ bản của OOP   - Trừu tượng: Sự trừu tượng cơ bản là bỏ đi hoặc ẩn chi tiết không quan trọng. Nó cho phép ta có cái nhìn tổng quan về những gì mà chúng ta triển khai.  - Đóng gói: Đóng gói là giữ một số thuộc tính và các phương thức private bên trong class để chúng không truy cập được từ bên ngoài class. Chúng ta luôn có mục tiêu là đóng gói hầu hết các trạng thái và phương thức và chỉ công khai các phương thức thiết yếu.  - Kế thừa: Trong OOP chúng ta có hai class có liên quan chặt chẽ với nhau, một class kế thừa class kia và chúng sẽ là class cha và class con. Class con kế thừa các thuộc tính và phương thức của class cha. Tính thừa kế làm cho tất cả các thuộc tính và phương thức của một class nhất định có sẵn cho class con, nó tạo thành một hệ thống phân cấp.  - Đa hình (nhiều hình dạng): Một class con có thể ghi đè một phương thức mà nó được thừa kế từ một class cha. |
| Sử dụng được constructor và toán tử new trong JavaScript const Person = function (firstName, birthYear) {  //tao thuoc tinh          this.firstName = firstName;          this.birthYear = birthYear;        };        const person2 = new Person("asdf",1992)        console.log(person2); |
| Hiểu rõ được định nghĩa về Prototype trong JavaScript Sử dụng thuộc tính **prototype** để khai báo phương thức thay cho phương pháp khai báo thẳng vào object  Person.prototype.calcAge = function () {          console.log(2037 -this.birthYear);         }         console.log(person2);         person2.calcAge() |
| Khai báo được ES6 Class class PerSonCL {          constructor(firstName, birthYear) {            //tao thuoc tinh            this.firstName = firstName;            this.birthYear = birthYear;          }          //tao phuong thuc trong class          calcAge(){            console.log(2037-this.birthYear);          }        }        //su dung phuong "new"        const personCL = new PerSonCL("sdfasdfas",1999)        console.log(personCL); |
| Áp dụng được các cách kế thừa Class       const Person = function (firstName, birthYear) {          this.firstName = firstName;          this.birthYear = birthYear;        };        const person2 = new Person("asdf", 1992);        console.log(person2);        //prototype        Person.prototype.calcAge = function () {          console.log(2037 - this.birthYear);        };        console.log(person2);        person2.calcAge();        //tao mot doi tuong moi        const Student = function (firstName,birthYear,coure) {          // PerSon(firstName,birthYear) goi ham thong thuong          //su dung phuong thuc "call"          Person.call(this,firstName,birthYear)          this.coure = coure         }         //tao linking prototypes de su dung cach ham        Student.prototype = Object.create(Person.prototype)         const student1 = new Student("asdfa",1992,"Computer")         console.log(student1);         student1.calcAge() |
| [Biết cách sử dụng hàm super()](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/af29b39aa91a43f086d98199a9e3291e/f4caf7b56eb0428c9abf8ffae2ea0fbb/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40f4caf7b56eb0428c9abf8ffae2ea0fbb)  class StudentCl extends PerSonCL{          constructor(firstName,birthYear,course){            super(firstName,birthYear),//Từ khóa super được sử dụng để truy cập các thuộc tính trên [[Nguyên mẫu]]            //của đối tượng hoặc của lớp, hoặc gọi phương thức khởi tạo của lớp cha.            this.course =course          }          //ghi de phuong thuc          calcAge(){            console.log(`I am ${this.firstName} ${2037-this.birthYear}`);          }        }        console.log(personCL);        const studentCl1 = new StudentCl("asdqwe",1998,"IT")        studentCl1.calcAge()        console.log(studentCl1); |
| [Hiểu được khái niệm về đồng bộ và bất đồng bộ](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/af29b39aa91a43f086d98199a9e3291e/358de685c2834b5aaed2daea7d54b421/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40358de685c2834b5aaed2daea7d54b421) Đồng bộ có nghĩa là code được thực thi từng dòng một theo thứ tự thực hiện chính xác mà chúng ta đã xác định trong code của mình.  Mỗi dòng code luôn đợi dòng trước thực thi xong rồi mới thực thi  Điều này dẫn đến vấn đề đối với một dòng code mất nhiều thời gian để chạy, việc thực thi dòng code đó sẽ chặn những dòng code khác thực thi   * + Bất đồng bộ và ví dụ về trường hợp bất đồng bộ   - Bất đồng bộ là code sẽ được thực thi sau khi một task chạy ở background kết thúc. ( Hàm setTimeout là bất đồng bộ vì nó sẽ được thực thi sau khi timer chạy trong BACKGROUND kết thúc)  - Code ở luồng chính không bị chặn  - Việc thực thi không phải đợi một task bất đồng bộ phải thực hiện xong |
| [Nắm được khái niệm về API và AJAX Call](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/af29b39aa91a43f086d98199a9e3291e/358de685c2834b5aaed2daea7d54b421/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40358de685c2834b5aaed2daea7d54b421)  * + Định nghĩa API   - API (Application Programming API) là một phần của phần mềm có thể được sử dụng bởi một phần mềm khác, cho phép các ứng dụng nói chuyện với nhau và trao đổi thông tin  - Trong JS và lập trình web có nhiều loại API: DOM Api, Geolocation Api, Online Api, ... và chúng ta cũng có thể tự triển khai các API của mình  - API như một phần mềm độc lập cho phép các phần mềm khác tương tác với chúng   * + Giới thiệu về AJAX call   - Ajax (Asynchronous Javascript And XML)  cho phép chúng ta giao tiếp với các máy chủ web từ xa theo cách không đồng bộ  - Với Ajax chúng ta có thể yêu cầu dữ liệu từ web server một cách linh động |
| [Áp dụng được XMLHttpRequest](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/af29b39aa91a43f086d98199a9e3291e/358de685c2834b5aaed2daea7d54b421/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40358de685c2834b5aaed2daea7d54b421) const request = new XMLHttpRequest();  request.open('GET', `URL`);   request.send();   * + Load dữ liệu thông qua EventListener   - Ví dụ: request.addEventListener('load', callback\_function); |
| [Sử dụng được Promises và Fetch API](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/af29b39aa91a43f086d98199a9e3291e/358de685c2834b5aaed2daea7d54b421/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40358de685c2834b5aaed2daea7d54b421) const getCountryData = function (country) {    fetch(`https://restcountries.com/v2/name/${country}`)      .then(        response => response.json() //=> trả về 1 promise      )      .then(data => {        renderCountry(data[0]);        //tạo biến chứa các quôc gia láng giềng        const neighbour = data[0].borders?.[0];        if (!neighbour) {          return;        }        //sau khi có dữ liệu ta nạp API tiếp theo để tìm "neighbour"        return fetch(`https://restcountries.com/v2/alpha/${neighbour}`)          .then(response => response.json())          .then(data => renderCountry(data, `neighbour`));      })      .catch(err => {       console.log(`${err} 💥💥💥💥`);          renderError(`Something went wrong 💥💥💥💥 ${err.message}`)      });  }; |
| [Vận dụng được Async/Await để xử lý bất đồng bộ](https://courses.funix.edu.vn/courses/course-v1:FUNiX+PRF192x_2.1-A_VN+2022_T3/courseware/af29b39aa91a43f086d98199a9e3291e/550889e5d94d43e5a1c59b0839a3f604/?activate_block_id=block-v1%3AFUNiX%2BPRF192x_2.1-A_VN%2B2022_T3%2Btype%40sequential%2Bblock%40550889e5d94d43e5a1c59b0839a3f604) const whereAmI = async function (country) {   const res = await fetch(`https://restcountries.com/v2/name/${country}`);   const data = await res.json()   console.log(data);   renderCountry(data[0])      const row = document.createElement("p")      row.innerHTML = data[0].name      test.appendChild(row)  };  whereAmI("USA") |