[Bài đọc] Thiết kế trong Lập trình hướng đối tượng

Chúng ta sẽ tìm hiểu các vấn đề cơ bản bằng việc tập trung vào Hướng đối tượng trên JavaScript (OOJS) - bài viết này sẽ trình bày căn bản về Lập trình Hướng Đối tượng (OOP), sau đó chúng ta sẽ tìm hiểu cách JavaScript mô phỏng các lớp với các hàm tạo (constructor), và cách để tạo ra một đối tượng.

|  |  |
| --- | --- |
| **Điều kiện tiên quyết:** | Có khả năng sử dụng máy tính căn bản, hiểu biết cơ bản về HTML và CSS, đã làm quen với các khái niệm căn bản trong JavaScript (Xem lại các First Step và Building block) và căn bản về OOJS (xem bài viết [Introduction to objects](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Object-oriented/Introduction)). |
| **Mục tiêu:** | Hiểu các lý thuyết căn bản đứng sau lập trình Hướng Đối tượng, và việc thể hiện nó trong JavaScript ("mọi thứ đều là đối tượng"), cách để định nghĩa các constructor và tạo ra đối tượng. |

**Lập trình Hướng Đối tượng từ khoảng cách hàng nghìn mét**

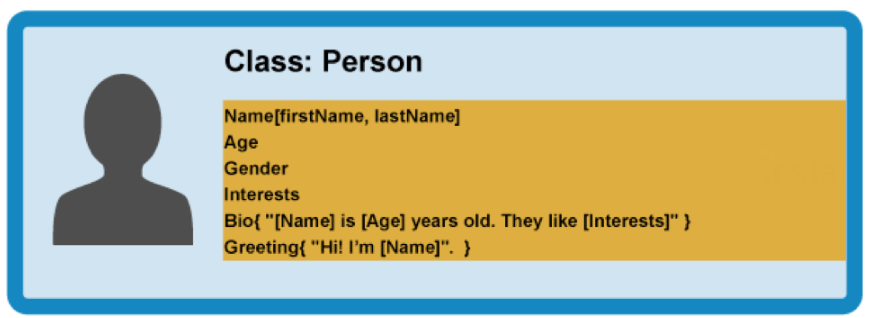
Để bắt đầu, chúng tôi sẽ cho bạn thấy một cái nhìn đơn giản, ở mức khái quát về lập trình hướng đối tượng (OOP). Chúng tôi nói đơn giản, bởi vì bạn sẽ nhanh chóng thấy sự phức tạp của OOP, và chuẩn bị sẵn các phương thức để "điều trị" trước khi nó làm khó cho bạn. Ý tưởng cơ bản của OOP đó là chúng ta mô phỏng các đối tượng trong thế giới thực vào trong các chương trình, và\hoặc cung cấp cách thức đơn giản để truy xuất vào các chức năng mà trên thực tế khó hoặc không thể sử dụng được.

Các đối tượng có thể chứa các dữ liệu và mã liên quan, chúng thể hiện thông tin về thứ mà bạn đang muốn mô hình hóa, và chức năng hay hành vi mà bạn muốn chúng có. Dữ liệu đối tượng (và thường cả các chức năng) có thể được lưu trữ một cách gọn gàng (thường được gọi là **bao gói** - encapsulated) bên trong một đối tượng (thường đặt tên cụ thể, đôi khi gọi là **namespace**), điều này giúp nó dễ dàng được cấu trúc và truy xuất; các đối tượng cũng thường được sử dụng cho việc lưu trữ dữ liệu để dễ dàng gửi qua mạng.

**Định nghĩa một bản mẫu cho đối tượng**

Hãy xem xét một chương trình đơn giản để hiển thị thông tin về các sinh viên và giáo viên của một trường học. Chúng ta sẽ thấy ở đó các khái niệm tổng quan về OOP, không chỉ trong ngữ cảnh của ngôn ngữ lập trình.

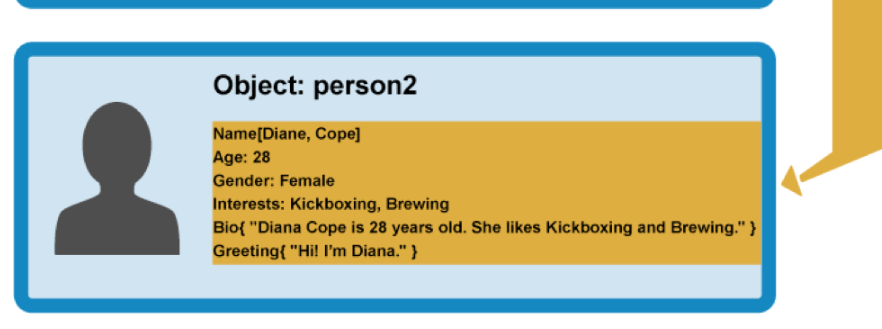
Để bắt đầu điều này chúng ta có thể trở lại với loại đối tượng Person trong bài viết trước đó về [các đối tượng trong JavaScript](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics), Person định nghĩa các dữ liệu và chức năng nói chung của một người. Có nhiều thử bạn có thể biết về một con người (địa chỉ, chiều cao, cỡ giày, DNA, số hộ chiếu, tính cách, ...), nhưng trong trường hợp này chúng ta chỉ quan tâm đến việc hiển thị tên, tuổi, giới tính, sở thích, và chúng ta cũng cũng muốn viết một giới thiệu ngắn về họ dựa trên các dữ liệu này, cuối cùng yêu cầu đối tượng này nói lời chào. Điều vừa mô tả được gọi là sự trừu tượng (**abstraction**) - tạo một mẫu đơn giản từ một thứ phức tạp hơn bằng cách trình bày những khía cạnh quan trọng nhất của nó theo một cách mà chương trình của chúng ta dễ dàng xử lý.



Trong một số ngôn ngữ lập trình OOP, việc định nghĩa ra một loại đối tượng được gọi là một **lớp**(class) (JavaScript sử dụng một cơ chế và thuật ngữ khác các ngôn ngữ khác, bạn sẽ thấy ở phần sau) - nó không phải là một đối tượng mà chỉ là một khuôn mẫu dùng để định nghĩa những đặc tính cho một đối tượng được tạo ra từ khuôn mẫu này.

### Tạo các đối tượng

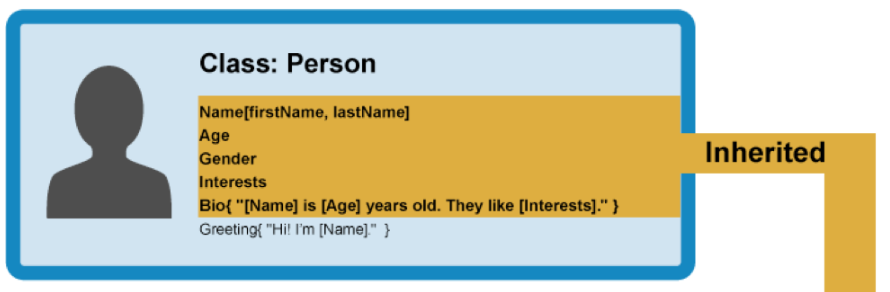
Từ lớp chúng ta có thể tạo ra các **đối tượng**(object instance) - các đối tượng sẽ có các dữ liệu và chức năng được định nghĩa bởi lớp. Từ lớp Person chúng ta có thể tạo ra đối tượng người:

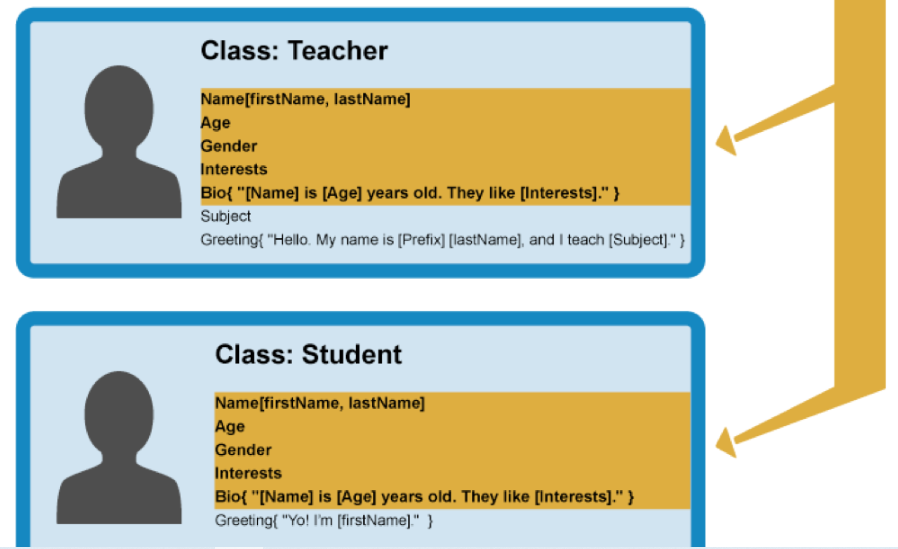


Khi tạo một đối tượng từ lớp, **hàm tạo** (constructor) sẽ chạy để thực hiện việc tạo đối tượng. Quá trình này sẽ tạo ra một đối tượng từ lớp và được gọi là **sự khởi tạo**- một đối tượng được **khởi tạo**từ lớp.

### Các lớp chuyên biệt

Trong trường hợp này chúng ta không muốn mô tả con người nói chung - chúng ta muốn những phân loại con người cụ thể như giáo viên, sinh viên. Trong OOP, chúng ta có thể tạo ra các lớp mới dựa trên những lớp khác - những **lớp con** mới có thể **kế thừa**(inherit) dữ liệu và các tính năng được viết trong **lớp cha**, vì thế bạn có thể sử dụng lại các chức năng cho tất cả các loại đối tượng đó thay vì phải viết lại chúng. Trong trường hợp chức năng khác nhau giữa các lớp bạn có thể trực tiếp định nghĩa các chức năng chuyên biệt cho lớp đó nếu cần.

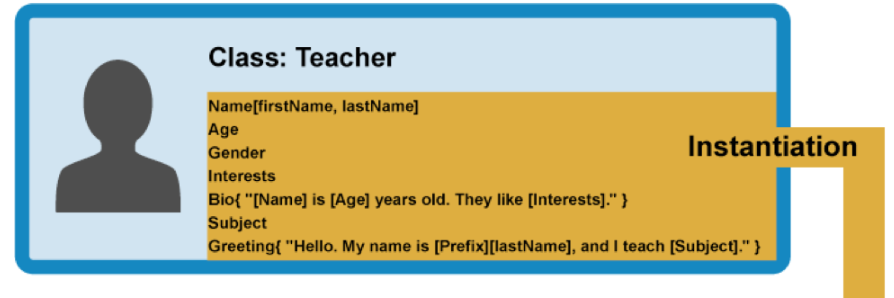




Điều này thực sự hữu ích - các giáo viên và sinh viên chia sẻ những đặc tính chung như tên, giới tính, và tuổi, vì vậy nó thuận lợi cho việc chỉ phải định nghĩa những thứ đó một lần. Bạn cũng có thể định nghĩa các đặc tính tương tự riêng cho các lớp khác nhau, khi đó mỗi định nghĩa về đặc tính đó sẽ nằm trong một namespace khác nhau. Ví dụ, một sinh viên sẽ chào là "Yo, I'm [firstName]" (VD: Yo, I'm Sam), trong khi một giáo viên lại chào theo hình thức khác, kiểu như "Hello, my name is [Prefix] [lastName], and I teach [Subject].", (VD: Hello, my name is Mr. Griffiths, and I teach Chemistry).

Lưu ý: Thuật ngữ dùng để diễn đạt cho việc nhiều loại đối tượng khác nhau cùng triển khai chức năng như nhau là đa hình (polymorphism). Phòng trường hợp bạn còn thắc mắc.

Bây giờ bạn đã có thể tạo các đối tượng từ các lớp con của mình. VD:





Trong phần còn lại của bài viết chúng ta sẽ bắt đầu xem xét OOP được triển khai trong JavaScript như thế nào.

**Hàm tạo (constructor) và đối tượng**

Một số người tranh luận rằng JavaScript không thực sự là một ngôn ngữ lập trình hướng đối tượng - ví dụ, nó không dùng từ khóa class để định nghĩa các lớp giống nhưng những ngôn ngữ OOP khác. Thay vào đó JavaScript dùng các hàm đặc biệt gọi là các hàm tạo (constructor) để định nghĩa đối tượng và các đặc tính của nó. Điều đó khá tiện ích bởi bạn thường gặp phải tình huống mà bạn không biết sẽ phải tạo ra bao nhiêu đối tượng; các hàm tạo cung cấp phương tiện hiệu quả để bạn tạo ra nhiều đối tượng khi cần, bổ sung dữ liệu và các chức năng cho chúng khi được yêu cầu.

Khi một đối tượng mới được tạo ra từ một hàm tạo, các chức năng không phải được sao chép vào đối tượng mới giống như các ngôn ngữ OOP "cổ điển" - thay vào đó nó liên kết thông qua một chuỗi tham chiếu được gọi là chuỗi nguyên mẫu (prototype) (xem phần [Object prototypes](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes)). Vì vậy nói đúng ra đây không thực sự là sự khởi tạo - JavaScript sử dụng một kỹ thuật khác để chia sẻ chức năng giữa các đối tượng.

Lưu ý: Không giống "Ngôn ngữ OOP cổ điển" không nhất thiết là thứ gì đó không tốt; như đã đề cập ở trên, OOP có thể có những thứ khá phức tạp, và JavaScript có những cách tốt hơn để tận dụng các đặc tính OOP mà không cần phải đi sâu vào nó.

Hãy xem xét việc tạo các lớp bằng các hàm tạo và tạo các đối tượng trong JavaScript. Trước hết, bạn hãy sao chép tệp [oojs.html](https://github.com/mdn/learning-area/blob/master/javascript/oojs/introduction/oojs.html) mà chúng ta đã thấy trong bài viết đầu tiên về các đối tượng.

**Một ví dụ đơn giản**

1. Hãy bắt đầu bằng việc quan sát cách bạn có thể định nghĩa một người với một hàm thông thường. Thêm vào mã hiện có của bạn hàm như sau:

function createNewPerson(name) {  
   var obj = {};  
   obj.name = name;  
   obj.greeting = function() {  
     alert('Hi! I\'m ' + this.name + '.');  
   };  
   return obj;  
 }

2. Bây giờ bạn có thể tạo ra một người bằng cách gọi hàm này - hãy thử gõ dòng mã sau vào màn hình console JavaScript của trình duyệt:

var salva = createNewPerson('Salva');  
 salva.name;  
 salva.greeting();

3. Điều này hoạt động khá tốt, nhưng nó khá dài dòng; nếu chúng ta biết chúng ta muốn tạo ra một đối tượng, tại sao chúng ta cần phải tạo ra một đối tượng rỗng và trả lại nó? Thật may mắn, JavaScript cung cấp cho chúng ta một cách thức ngắn gọn hơn, dưới dạng các hàm tạo - hãy thực hiện nó ngay bây giờ nhỉ!

4. Thay thể hàm mà bạn đã viết ở trên bằng đoạn mã sau:

function Person(name) {  
   this.name = name;  
   this.greeting = function() {  
     alert('Hi! I\'m ' + this.name + '.');  
   };  
 }

Hàm tạo là phiên bản của một lớp trong JavaScript. Bạn chú ý rằng nó có tất cả các đặc tính mà bạn trông đợi trong một hàm, mặc dù nó không trả lại bất cứ thứ gì hoặc tạo ra một đối tượng rõ ràng - căn bản nó chỉ định nghĩa các thuộc tính và phương thức. Bạn thấy từ khóa this đã được sử dụng ở đây - nó chỉ để nói rằng đó là nơi một đối tượng được tạo ra, thuộc tính name sẽ bằng với giá trị của biến name được truyền vào khi hàm tạo được gọi, và phương thức greeting() cũng sẽ sử dụng giá trị name được truyền vào hàm tạo.

Chú ý: Tên của hàm tạo thường bắt đầu với một chữ cái viết hoa - quy ước này thường được sử dụng để dễ dàng nhận biết ra các hàm tạo trong mã của bạn.

Vậy, làm thế nào chúng ta gọi một hàm tạo để tạo ra một số đối tượng?

1. Thêm những dòng mã sau vào đoạn mã của bạn:

var person1 = new Person('Bob');  
 var person2 = new Person('Sarah');

2. Lưu tệp tin và tải lại nó trên trình duyệt, và thử nhập các dòng sau vào ô văn bản trên trang của bạn:

person1.name  
 person1.greeting()  
 person2.name  
 person2.greeting()

Tuyệt cú mèo! Bây giờ bạn đã thấy rằng có hai đối tượng mới trên trang, mỗi đối tượng lưu dưới một namespace khác nhau - khi bạn truy xuất các thuộc tính và phương thức của chúng, bạn phải bắt đầu với person1 hoặc person2; chúng được đóng gói vì vậy chúng không xung đột chức năng với nhau. Dĩ nhiên chúng đều có thuộc tính name và phương thức greeting(). Lưu ý rằng chúng sử dụng giá trị name của riêng mình đã được gán khi tạo đối tượng; đây là lý do tại sao việc sử dụng từ khóa this là rất quan trọng, nó giúp đối tượng sử dụng các giá trị của chính mình chứ không phải các giá trị khác.

Hãy quan sát lại việc gọi hàm tạo:

var person1 = new Person('Bob');  
 var person2 = new Person('Sarah');

Trong mỗi trường hợp từ khóa new được sử dụng để nói với trình duyệt rẳng chúng ta muốn tạo một đối tượng mới, trong cặp ngoặc tròn của hàm là tham số và kết quả được lưu giữ vào một biến - khá giống cách thức mà một hàm chuẩn được gọi. Mỗi đối tượng được tạo theo định nghĩa sau:

function Person(name) {  
   this.name = name;  
   this.greeting = function() {  
     alert('Hi! I\'m ' + this.name + '.');  
   };  
 }

Sau khi các đối tượng mới được tạo ra, biến person1 và person2 tham chiếu đến các đối tượng như sau:

{  
   name: 'Bob',  
   greeting: function() {  
     alert('Hi! I\'m ' + this.name + '.');  
   }  
 }  
   
 {  
   name: 'Sarah',  
   greeting: function() {  
     alert('Hi! I\'m ' + this.name + '.');  
   }  
 }

Lưu ý rằng mỗi khi chúng ta gọi hàm tạo, chúng ta lại định nghĩa phương thức greeting(), điều này lý tưởng lắm. Để tránh chúng ta có thể định nghĩa các hàm với nguyên mẫu (prototype), điều này sẽ được xem xét này.

**Xây dựng hoàn chỉnh hàm tạo**

Ở trên chúng ta đã thấy một ví dụ đơn giản để giúp chúng ta khởi đầu. Bây giờ hãy tiếp tục và xây dựng hoàn chính hàm tạo Person().

1. Thay thế mã trước đây của bạn với mã sau - đây chính xác là khá giống các ví dụ trước, chỉ thêm một chút để nó phức tạp hơn:

function Person(first, last, age, gender, interests) {  
   this.name = {  
     first,  
     last  
   };  
   this.age = age;  
   this.gender = gender;  
   this.interests = interests;  
   this.bio = function() {  
     alert(this.name.first + ' ' + this.name.last + ' is ' + this.age + ' years old. He likes ' + this.interests[0] + ' and ' + this.interests[1] + '.');  
   };  
   this.greeting = function() {  
     alert('Hi! I\'m ' + this.name.first + '.');  
   };  
 };

2. Bây giờ thêm dòng mã dưới đây vào sau đoạn mã trên để tạo mới đối tượng:

var person1 = new Person('Bob', 'Smith', 32, 'male', ['music', 'skiing']);

Bạn sẽ thấy rằng bạn có thể truy xuất các thuộc tính và phương thức giống như đã làm với đối tượng đầu tiên mà chúng ta đã định nghĩa:

person1['age']  
 person1.interests[1]  
 person1.bio()  
 *// etc.*

Chú ý: Nếu bạn có vấn đề khi chạy đoạn mã này, hãy so sánh và xem lại mã của bạn với mã của chúng tôi trong trang [oojs-class-finished.html](https://github.com/mdn/learning-area/blob/master/javascript/oojs/introduction/oojs-class-finished.html) (và quan sát mã thực tế hoạt động [ở đây](http://mdn.github.io/learning-area/javascript/oojs/introduction/oojs-class-finished.html)).

**Những bài tập khác**

Để bắt đầu hãy thử thêm các dòng mã để tạo ra một cặp đối tượng của bạn, và thử lấy ra, thiết lập các thành phần của các đối tượng đó.

Ngoài ra, có một số vấn đề với phương thức bio() - chuỗi đầu ra luốn bao gồm đại từ "He", thậm chí ngay cả khi giới tính của người đó là nữ (female), hoặc phân loại giới tính khác. Bio cũng chỉ có hai sở thích, thậm chí ngay cả khi sở thích là một mảng. Bạn có thể xử lý được điều này bằng cách định nghĩa lại hàm tạo không? Bạn có thể bổ sung mã chó hàm tạo (bạn cần sử dụng thêm cấu trúc điều kiện và cấu trúc lặp).

Lưu ý: Nếu bạn gặp khó khăn, bạn có thể tham khảo giải pháp của chúng tôi [tại đây](https://github.com/mdn/learning-area/blob/master/javascript/oojs/introduction/oojs-class-further-exercises.html) - nhưng trước tiên hãy thử theo cách của bạn đã nhé.

**Các cách khác để tạo ra đối tượng**

Vậy là chúng ta đã có hai cách khác nhau để tạo ra một đối tượng - [mô tả một hằng đối tượng](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#Object_basics) và sử dụng hàm tạo.

Những điều này đều có ý nghĩa nhưng vẫn còn có nhưng cách khác - chúng tôi muốn bạn làm quen với những điều này trong trường hợp bạn gặp phải chúng trên một trang web nào đó.

**Hàm tạo Object()**

Trước tiên, bạn có thể sử dụng hàm tạo Object() để tạo một đối tượng mới. Vâng, ngay cả các đối tượng chung chung cũng có một hàm tạo, nó tạo ra một đối tượng rỗng.

1. Thử nhập dòng mã sau vào JavaScript console trên trình duyệt:

var person1 = new Object();

2. Điều này tạo ra một đối tượng rỗng và gán cho biến person1. Sau đó bạn có thể bổ sung các thuộc tính cho đối tượng sử dụng dấu chấm hoặc cặp ngoặc vuông để mô tả; hãy thử theo VD sau:

person1.name = 'Chris';  
 person1['age'] = 38;  
 person1.greeting = function() {  
   alert('Hi! I\'m ' + this.name + '.');  
 };

3. Bạn cũng có thể truyền một hằng đối tượng như một tham số cho hàm tạo Object(), để cung cấp cho nó các thuộc tính và phương thức. VD:

var person1 = new Object({  
   name: 'Chris',  
   age: 38,  
   greeting: function() {  
     alert('Hi! I\'m ' + this.name + '.');  
   }  
 });

**Sử dụng phương thức create()**

JavaScript có một hàm dựng sẵn là [create()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create), hàm này cho phép bạn tạo ra một đối tượng mới từ một đối tượng đang tồn tại.

1. Thử nhập dòng lệnh sau vào JavaScript console:

var person2 = Object.create(person1);

2. Tiếp theo là hai dòng lệnh:

person2.name  
 person2.greeting()

Bạn sẽ thấy person2 được tạo ra dựa trên person1 - nó có cùng thuộc tính và phương thức. Điều này khá hưu ích vì nó cho phép bạn tạo mới một đối tượng mà khồn cần đến hàm tạo. Ngoài ra, các hàm tạo đòi hỏi thêm mã - bạn có thể tạo các hàm tạo của mình ở một nơi, sau đó tạo các đối tượng khi cần, và chỉ rõ nó nó đến từ đâu.

Nhược điểm là  rằng phương thức create() không được hỗ trợ bởi IE8. Do đó các hàm tạo cso thể là cách nếu bạn muốn hỗ trợ những trình duyệt cũ, hoặc bạn chỉ cần một cặp bản sao của một đối tượng. Điều này tùy thuộc vào sở thích của bạn. Một số người tìm đến phương thức create() bởi nó dễ hiểu và dễ sử dụng.

Chúng ta sẽ khám phá hiệu quả của creat() chi tiết trong các bài viết sau.

**Tổng kết**

Bài viết này đã cung cấp một cái nhìn đơn giản về khái niệm hướng-đối tượng - nó không hoàn toàn đầy đủ nhưng đã cho bạn một ý tưởng về những gì chúng tôi đang trình bày ở đây. Ngoài ra, chúng ta đã bắt đầu xem xét cách JavaScript thể hiện và những khác biệt so với "ngôn ngữ OOP cổ điển", cách để triển khai các lớp trong JavaScript với hàm tạo và các cách khác nhau để tạo ra các đối tượng.

Trong bài tiếp theo chúng ta sẽ tìm hiểu về các nguyên mẫu đối tượng (object prototype) trong JavaScript.

<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS>