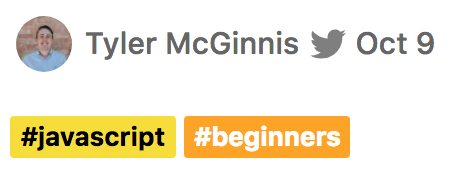
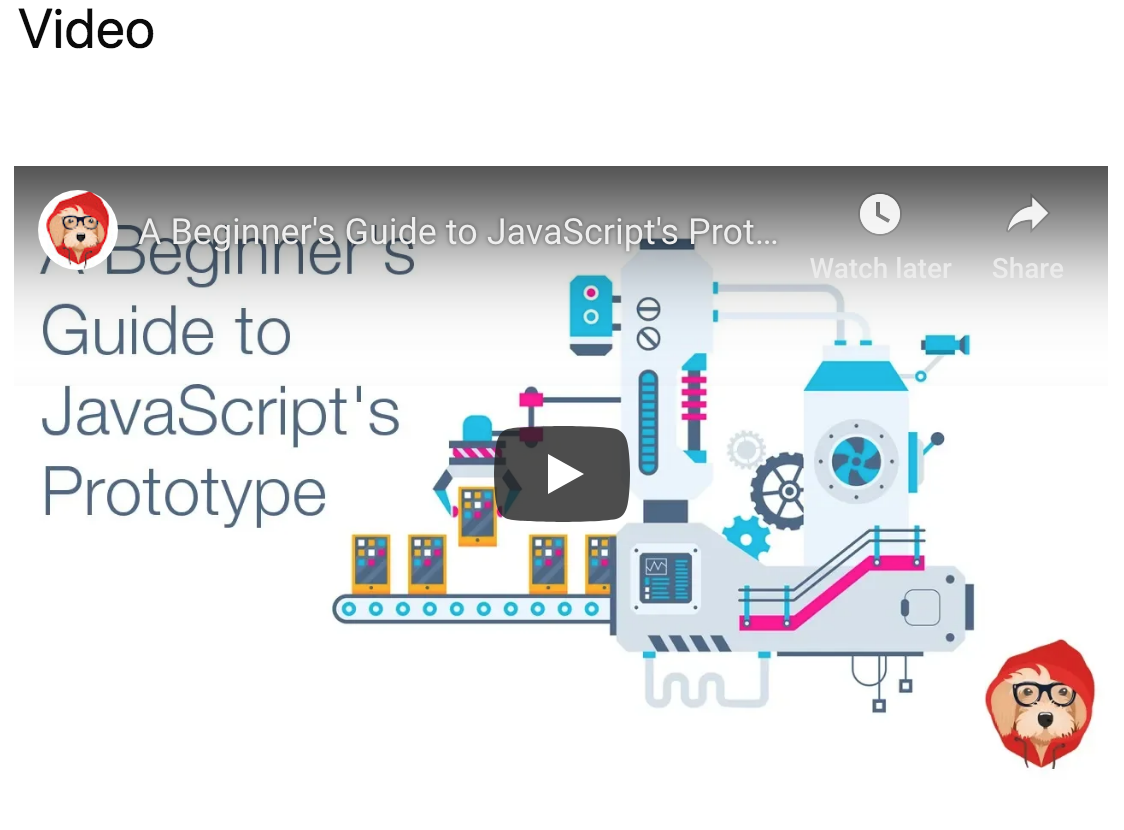
Hướng dẫn cho ngừoi mới bắt đầu đến với JavaScript's Prototype



Bạn không thể tiến xa trong Javascript mà không biết tới objects (đối tượng). Objects là nền tàng của hầu hết các ngôn ngữ lập trình trong JavaScript. Trong bài viết này, bạn sẽ học về sự đa dạng của patterns (các mẫu) trong việc tạo objects (đối tượng) mới, và để làm được như thế, bạn sẽ dần dần được dẫn đến sự hiểu biết tường tận về prototype (cha của các object) trong JavaScript.

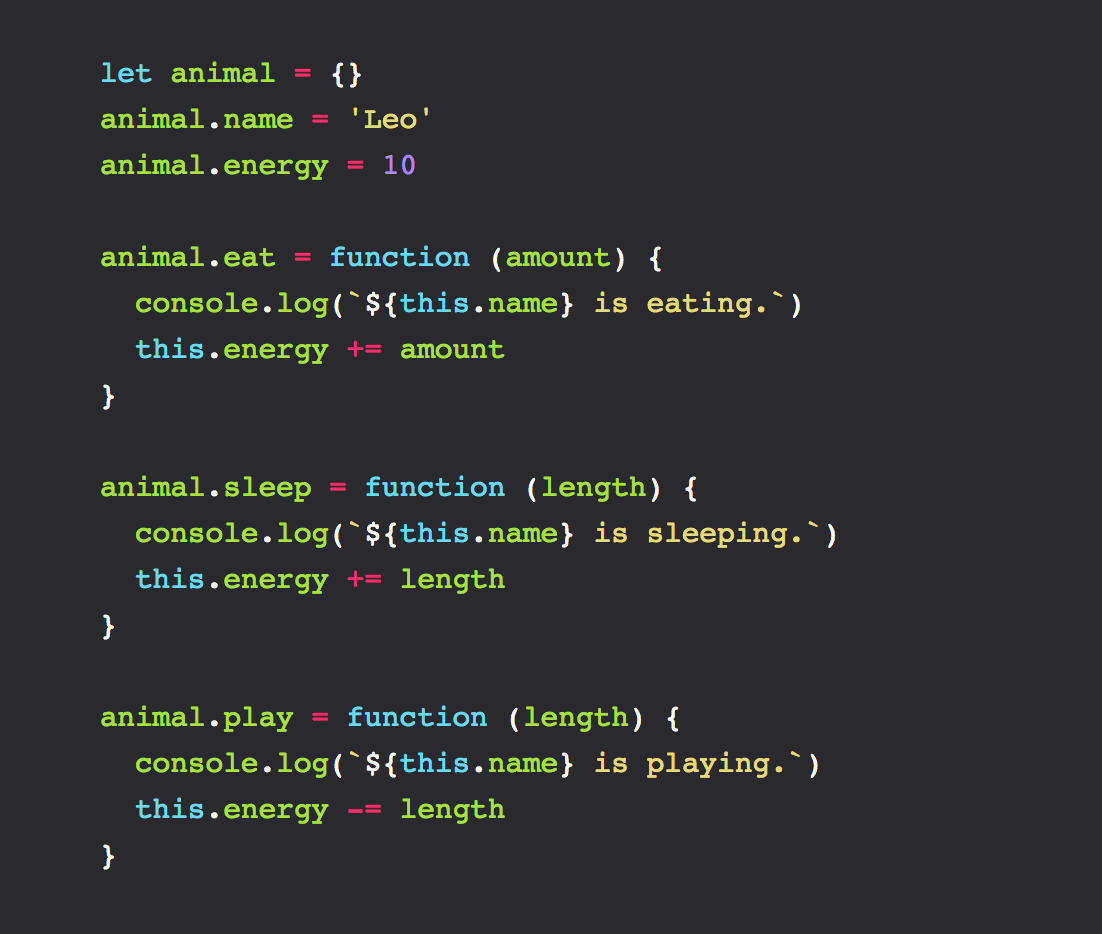
Đây là một phần của khoá [học nâng cao JavaScript](https://tylermcginnis.com/courses/advanced-javascript/). Nếu bạn thích bài viết này, thì thử trải nghiệm qua.



**Post**

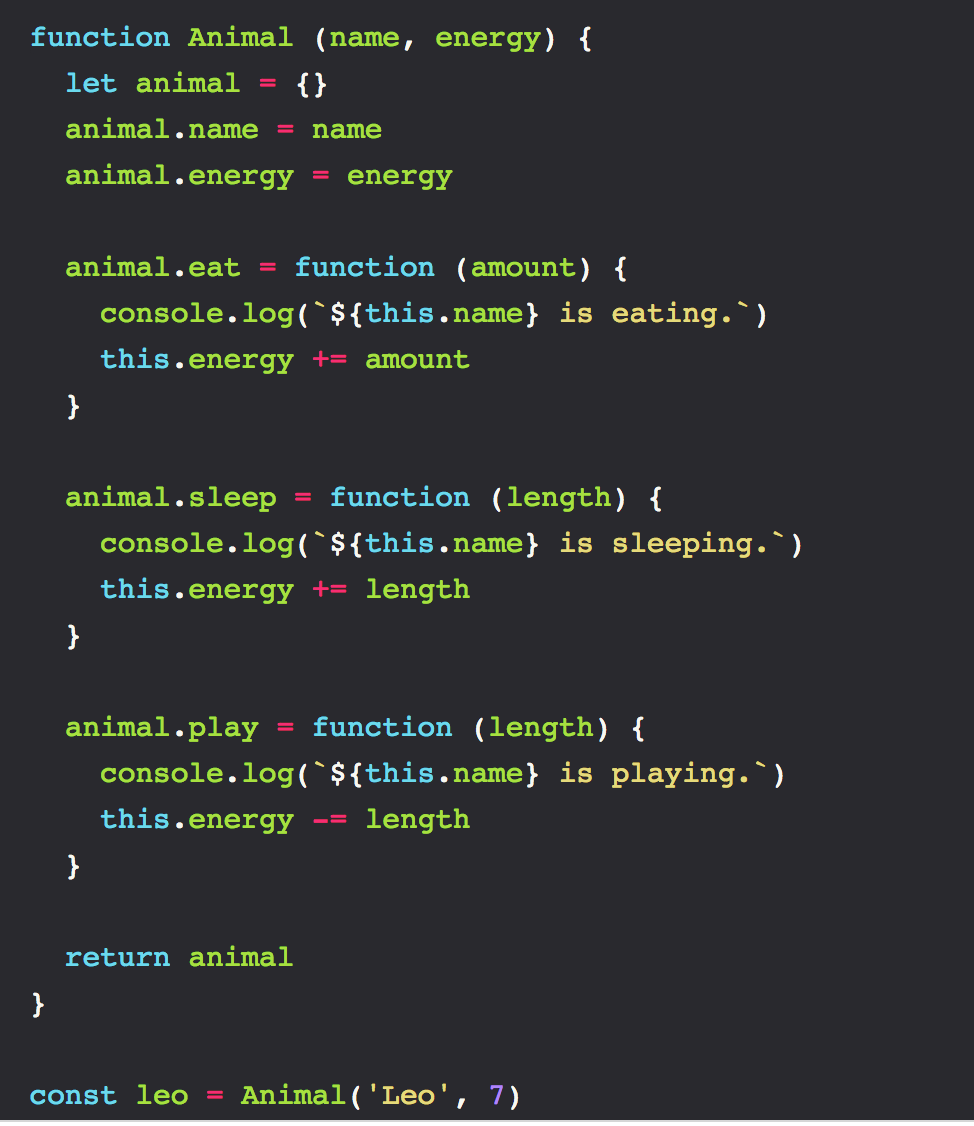
Bạn không thể tiến xa trong Javascript mà không biết tới objects (đối tượng). Objects là nền tàng của hầu hết các ngôn ngữ lập trình trong JavaScript. Thực tế, học để biết tạo một objects có lẽ là thứ đầu tiên bạn đã học khi mà bạn mới bắt đầu.

Objects là cặp key/ value ( khoá/ giá trị). Cách hữu hiệu nhất để tạo 1 object là dùng ngoặc nhọn { }, và bạn có thể thêm properties( giá trị) và methods( phương thức) vào object sử dụng dấu chấm (.).



Bây giờ, odds ( sự chênh lệch) xuất hiện trong ứng dụng của chúng ta nên chúng ta cần tạo hơn là 1 animal ( ở đây là tên của object trong dòng code bên trên). Bước tiếp theo là chúng ta cần gói gọn logic nằm bên trong function ( hàm), cái mà có thể gọi ra khi mà chúng ta cần object với tên là animal. Chúng ta gọi pattern ( mẫu) này là **Functional Instantiation,** và chúng ta gọi bản thân nó là 1 "hàm tạo" ("constructor function"), bởi vì nó có trách nhiệm tạo ra 1 object (đối tượng) mới.

**Functional Instantiation**

****

Bây giờ, khi mà chúng ta cần tạo một object tên là animal mới (hay là 1 ví dụ mới), thì chúng ta chỉ phải gọi function ( hàm) Animal, chúng ta còn có thể chuyền vào giá trị cho name (ten con vật) và energy (năng lượng). Cái này vừa hay mà lại đơn giản có phải không.

Tuy nhiên, bạn có thấy được điểm yếu của pattern ( mẫu) này không? Điểm yếu lớn nhất mà chúng ta có thể sửa được liên quan đến - eat, sleep, và play. Ba cái methods ( phương thức) đấy có quá nhiều điểm chung. Không có một lí do nào mà lại phải tạo lại những cái methods đó mỗi khi chúng ta tạo một animal mới. Chúng ta đang lãng phí bộ và làm cho animal object to ra một cách không cần thiết. Vậy bạn đã nghĩ ra một giải pháp nào chưa? Thay vì tạo lại những methods mỗi lần tạo một animal, chúng ta có thể chuyển những methods đó đến object riêng của nó, sau đó mỗi con vật animal sẽ có đường dẫn đến object của nó. Chúng ta có thể gọi mẫu (pattern) này là **Functional Instantiation with Shared Methods** (hàm dẫn với phương thức chung có thể chia sẻ cho nhau), hơi dài nhưng dễ hiểu hơn.

**Functional Instantiation with Shared Methods**



Bằng việc dẫn những cái phương thức chia sẻ được cho nhau vào object rieng của nó và dẫn thẳng cái object vào trong hàm animal (function animal), chúng ta đã giải quyết được vấn đề của những object animal với kích cỡ quá lớn.

#### Object.create

Củng cố ví dụ của chúng ta một lần nữa bằng việc sử dụng object.create .Object.create cho phép bạn tạo một object ngay cả khi mà có lỗi trong việc tìm property (giá trị) của object đó, nó có thể " hỏi" một object khác xem có property đó không (gia trị) đó không. Nhiều lý thuyết quá phải không :)))). Thế thì xem một chú code nha.



Trong ví dụ bên trên, child được tạo với **Object.create(parent),** khi mà cố lỗi tìm thẻ con **child**, **JavaScript** sẽ chuyển quyền đó qua thẻ cha **parent object**. có nghĩa là thẻ con **child** không có giá trị (property) **heritage** ,thẻ cha **parent** sẽ hiện cho thẻ **child** với giá trị **heritage** là *Irish*

Giờ với Object.create, Làm thế nào để làm đơn giản đi code của function Animal trên kia? Thay vì cho thêm tất cả shared methods ( phương thức chung) từng cái vào một như chúng ta làm trước kia, Ta có thể sử dụng Object.create để nhượng quyền cho animalMethods object. Để nghe hay hay hơn, chúng ta có thể gọi hàm mới là **Functional Instantiation with Shared Methods and Object.create 🙃** (hàm dẫn với phương thức chung có thể chia sẻ cho nhau và Object.create).

#### Functional Instantiation with Shared Methods and Object.create

📈 giờ mỗi lần chúng ta gọi leo.eat, JavaScript sẽ tìm eat method trong leo object. Cái tìm kiếm đó sẽ lỗi, và, vì Object.create, nó sẽ chuyển quyền cho animalMethods object là cái mà nó sẽ tìm object eat.

Có vài thứ chúng ta có thể làm cho nó tốt hơn. Nghe có vẻ khó để quán lý object riêng biệt (animalMethods) để chia sẻ methods giữa các ví dụ. Nghe như đó là cái đặc điểm chung mà bạn muốn thjwc hiện giữa các ngôn ngữ, và đó là lí do tại sao bạn lại đang ở đây - prototype.

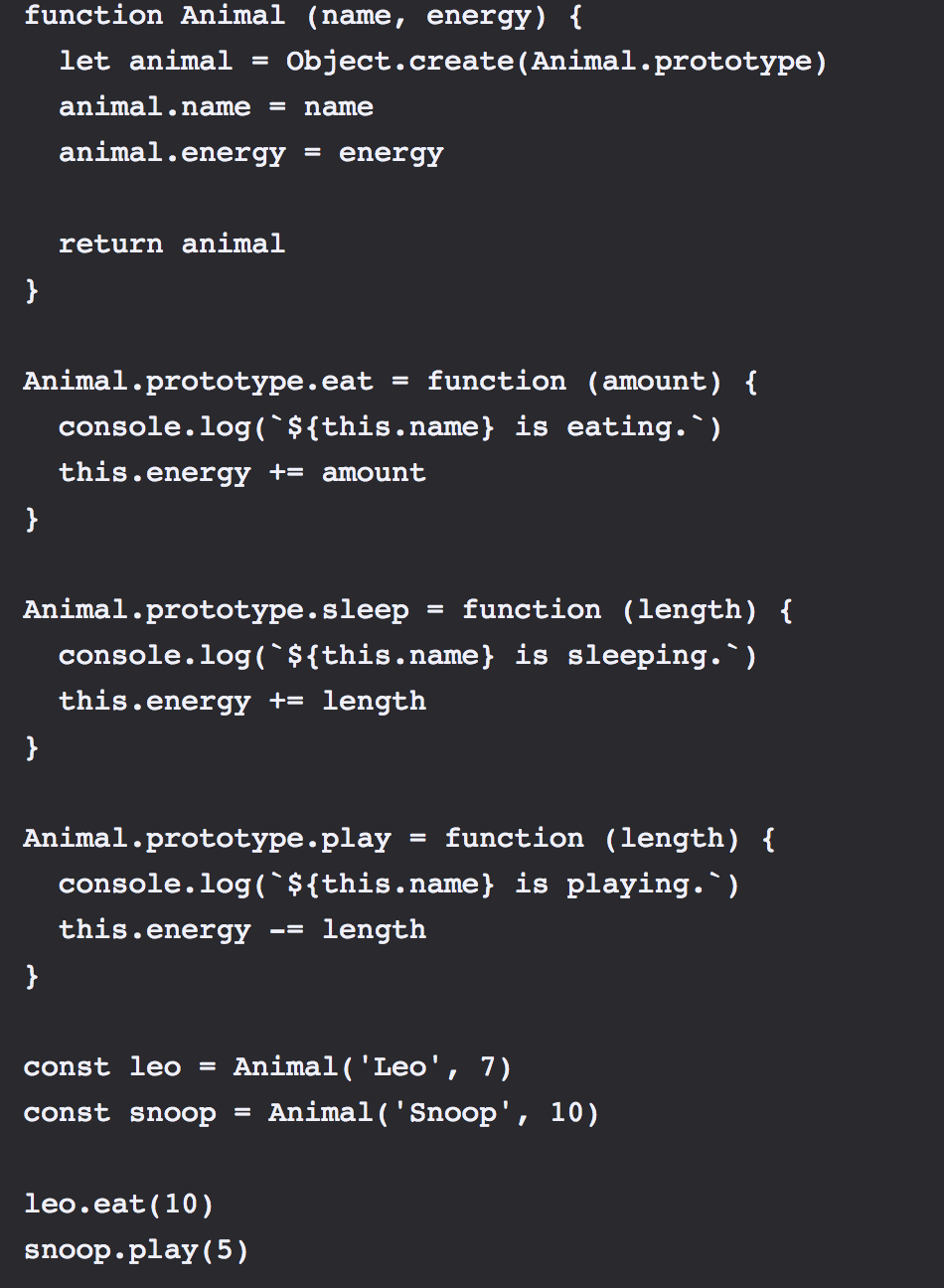
**Thế Chính xác** prototype trong JavaScript là gì ? Đơn giản thì mỗi hàm trong JavaScript thì có 1 giá trị prototype property cái mà dẫn đến một object. Vậy bạn tự trải nghiệm thử xem.

#### 

#### (animalsMethods

Thay vì tạo một object riêng để quán lý mehtods ( như cái mà đã dùng vớianimalMethods), chúng ta chỉ cần cho mỗi methods vào hàm của Animal function's prototype? và tat cả chúng ta cần làm là thay vì dùng Object.create để chuyển quyền cho animalMethods, chúng ta có thể sử dụng nó để chuyền quyền cho Animal.prototype. chúng ta gọi nó là pattern ( những cái làm sẵn) Prototypal Instantiation.

#### Prototypal Instantiation



👏👏👏 prototype  là một giá trị mà tất cả các function trong JavaScript đều có, và cũng như chúng ta thấy ở trên kia, nó cho phép chúng ta chia sẻ methods giữa các. Tất cả các tính năng vẫn được giữ nguyên nhưng bây giờ thay vì phải quản lý một object riêng cho tất cả các methods, chúng ta có thể dùng một object khác mà có sẵn bên trong Animal function, Animal.prototype.