1. Introduction

We've been using properties and methods all up to this point, and in the next section you'll bring your knowledge of object-oriented programming to another level of sophistication by creating classes (i.e., categories) of objects!

1. Constructor Functions

First letter should be capitalized.

function SoftwareDeveloper() {

this.favoriteLanguage = 'JavaScript';

}

Constructor is different than a normal object.

function Sandwich(bread, meat, vegetables) {

this.bread = bread;

this.meat = meat;

this.vegetables = vegetables;

}

Typeof instanceof

1. The `this` Keyword

The value of this is actually not assigned to anything until an object calls the method where this is used. In other words, the value assigned to this is based on the object that invokes the method where this is defined.

1. Setting Our Own `this`

Call(), apply(), bind()

multiply.call(window, 3, 4);

apply() takes the function's arguments in an array.

multiply.apply(window, [3, 4]);

introduce.call(andrew, 'JavaScript');

Similar to call() and apply(), the bind() method allows us to directly define a value for this. bind() is a method that is also called \_on\_ a function, but unlike call() or apply(), which both invoke the function right away -- bind() returns a new function that, when called, has this set to the value we give it.

driver.displayName.bind(car);

1. Prototypal Inheritance

Looks up the chain of inheritance for properties and methods.

// (B) preferred as it saves memory

function Dalmatian (name) {

this.name = name;}

Dalmatian.prototype.bark = function() {

console.log(`${this.name} barks!`);};

The hasOwnProperty() allows you to find the origin of a particular property.

Objects also have access to the isPrototypeOf() method, which checks whether or not an object exists in another object'sconsole.log(board.constructor);

// function Longboard() {

// this.material = 'bamboo';

// } prototype chain.

1. Prototypal Inheritance: Subclasses

One of the benefits of implementing inheritance is that it allows you to reuse existing code. By establishing inheritance, we can subclass, that is, have a "child" object take on most or all of a "parent" object's properties while retaining unique properties of its own.

const bear = {

claws: true,

diet: 'carnivore'

};

function PolarBear() {

// ...

}

PolarBear.prototype = bear;

const snowball = new PolarBear();

snowball.color = 'white';

snowball.favoriteDrink = 'cola';

When the new instance of GuineaPig is created, the special property waffle.\_\_proto\_\_ is set to GuineaPig.prototype.

First, even though \_\_proto\_\_ can access the prototype of the object it is called on, using it in any code you write is not good practice.

1. Lesson Summary

Great Job!

1. Course Outro