### ADDING INHERITABLE PROPERTIES TO PROTOTYPES

What if we wanted to add some base values or functionality to ALL objects of a similar type?

```
var witch = "I'll get you, my pretty...and your little dog, too!";
var scarecrow = "Well, some people without brains do an awful lot of talking don't they?";
var glinda = "Be gone! Before someone drops a house on you!";
var dorothy = "There's no place like home.";
var lion = "Come on, get up and fight, you shivering junkyard!";
var wizard = "Do not arouse the wrath of the great and powerful Oz!";
var tinman = "Now I know I have a heart, because it's breaking.";
```

```
String.prototype.countAll = function ( letter ){
   var letterCount = 0;
   for (var i = 0; i<this.length; i++) {
      if ( this.charAt(i).toUpperCase() == letter.toUpperCase() ) {
        letterCount++;
      }
   }
   return letterCount;
};</pre>
```

### ADDING INHERITABLE PROPERTIES TO PROTOTYPES

What if we wanted to add some base values or functionality to ALL objects of a similar type?

```
var witch = "I'll get you, my pretty...and your little dog, too!";
var scarecrow = "Well, some people without brains do an awful lot of talking don't they?";
var glinda = "Be gone! Before someone drops a house on you!";
var dorothy = "There's no place like home.";
var lion = "Come on, get up and fight, you shivering junkyard!";
var wizard = "Do not arouse the wrath of the great and powerful Oz!";
var tinman = "Now I know I have a heart, because it's breaking.";
```

```
String.prototype.countAll = function ( letter ){
   var letterCount = 0;
   for (var i = 0; i<this.length; i++) {
      if ( this.charAt(i).toUpperCase() == letter.toUpperCase() ) {
        letterCount++;
      }
   }
   return letterCount;
};</pre>
```

```
lion.countAll("k");

→ 1

tinman.countAll("N");

→ 3
```



Welcome to

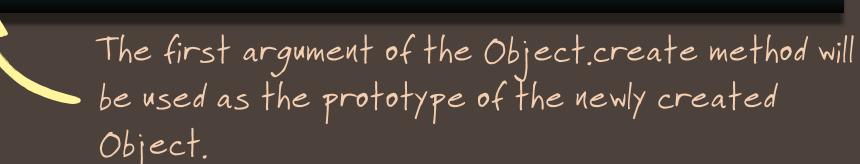
# THE PROTOTYPE PLAINS

## A SECOND WAY TO BUILD OBJECTS USING OBJECT.CREATED

Using inheritance, we can create new Objects with our existing Objects as prototypes

```
var shoe = { size: 6, gender: "women", construction: "slipper"};
```

var magicShoe = Object.create( shoe );



console.log( magicShoe );

```
Object {size: 6, gender: "women", construction: "slipper"}

The new Object magicShoe inherited all of its

properties from shoe, just like we'd expect from a

prototype.
```

## A SECOND WAY TO BUILD OBJECTS USING OBJECT. CREATED

Using inheritance, we can create new Objects with our existing Objects as prototypes

```
var shoe = { size: 6, gender: "women", construction: "slipper"};
var magicShoe = Object.create( shoe );
magicShoe.jewels = "ruby";
magicShoe.travelAction = "click heels";
magicShoe.actionsRequired = 3;
console.log( magicShoe );
                     Object {jewels: "ruby", travelAction: "click heels",
                                  actionsRequired: 3, size: 6, gender: "women",
```

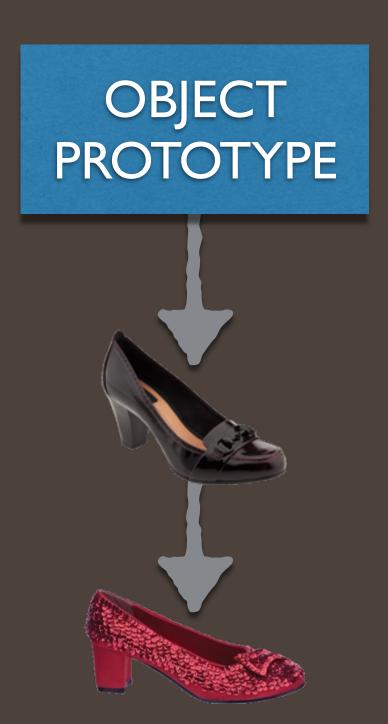
construction: "slipper"}

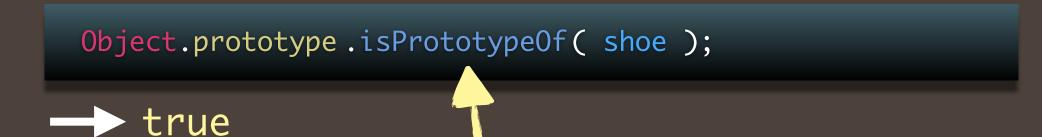
### A SECOND WAY TO BUILD OBJECTS USING OBJECT.CREATED

Using inheritance, we can create new Objects with our existing Objects as prototypes

```
var shoe = { size: 6, gender: "women", construction: "slipper"};
var magicShoe = Object.create( shoe );
magicShoe.jewels = "ruby";
magicShoe.travelAction = "click heels";
magicShoe.actionsRequired = 3;
console.log(
              shoe
                   );
             Object {size: 6, gender: "women", construction: "slipper"}
```

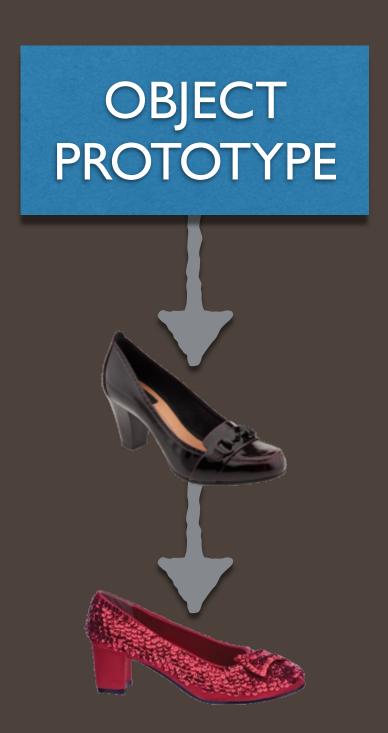
We can use an inherited method to demonstrate our newly created prototype chain





Remember this property that all JS Objects inherit from the Object prototype? We can use it to find out if any specific Object is a prototype of another.

We can use an inherited method to demonstrate our newly created prototype chain

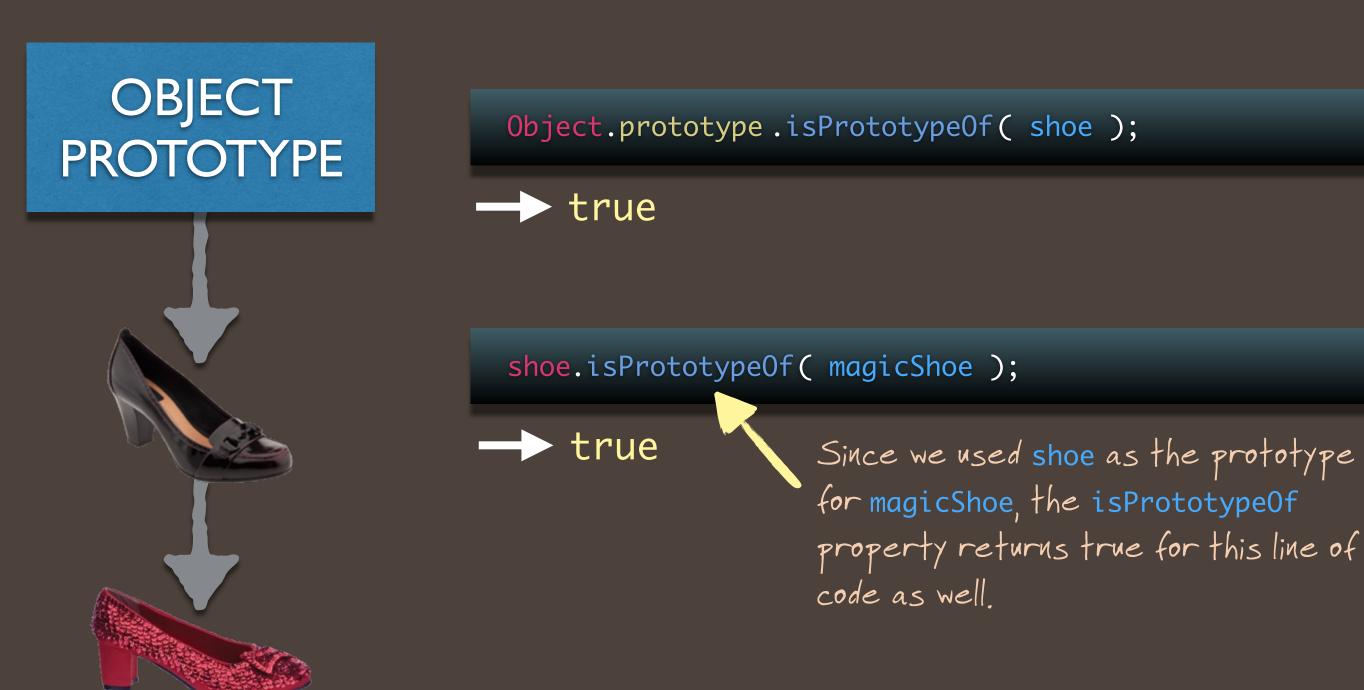


Object.prototype.isPrototypeOf( shoe );

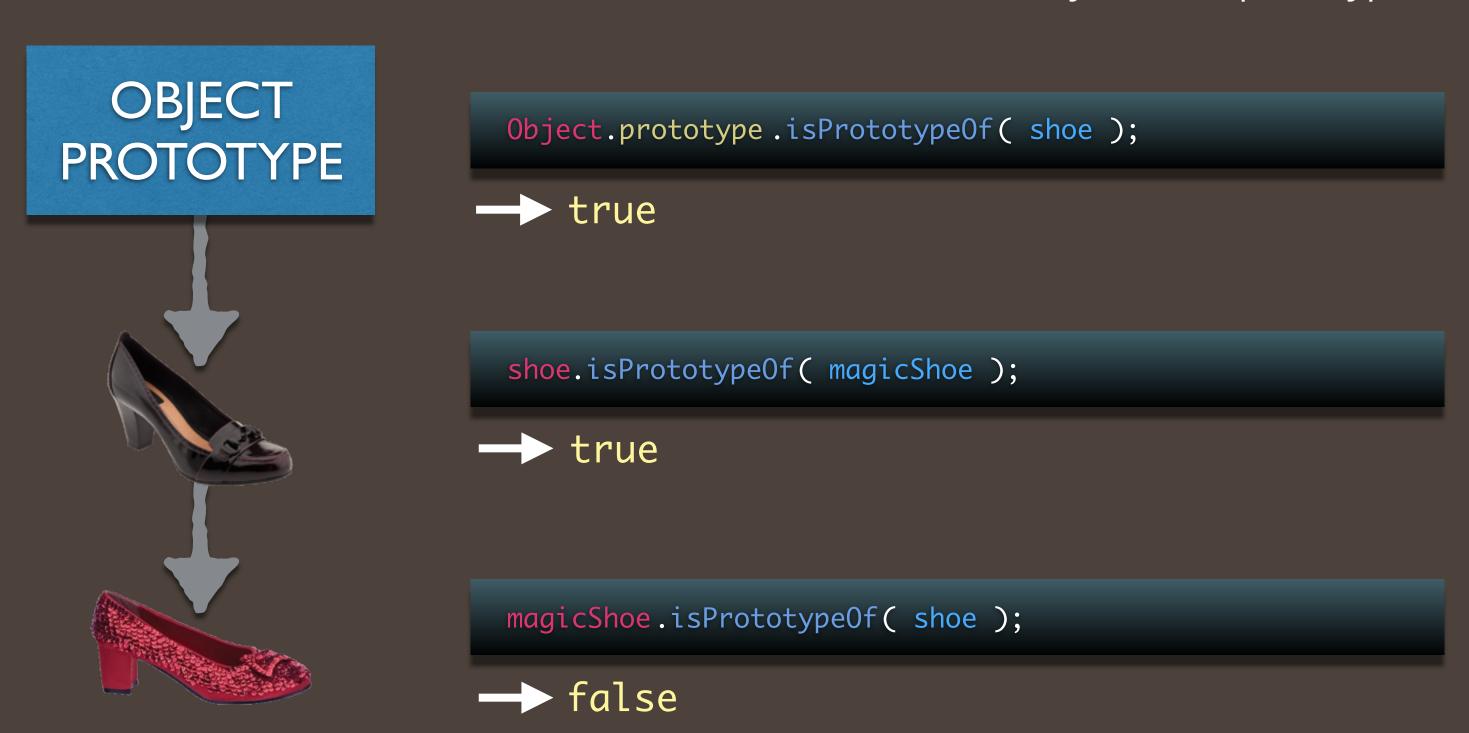
True

shoe

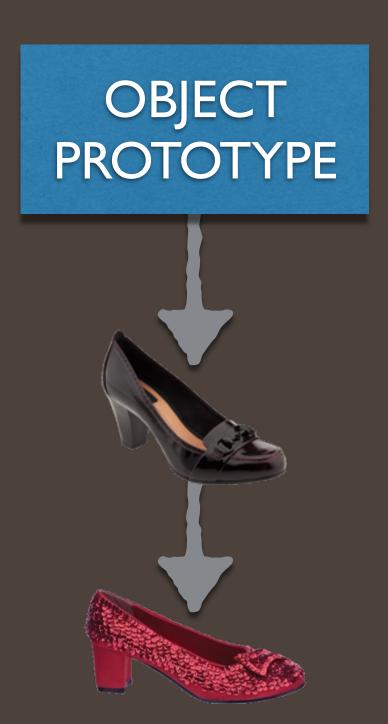
We can use an inherited method to demonstrate our newly created prototype chain



We can use an inherited method to demonstrate our newly created prototype chain



We can use an inherited method to demonstrate our newly created prototype chain





The isPrototypeOf method will look upward through the entire hierarchy (the prototype "chain") to see whether the Object.prototype Object is a prototypical "ancestor" of magicShoe.

### WHAT IF THERE WERE OTHER KINDS OF SHOES?

Could we use the same prototype to create boots, sneakers, sandals, and...uh...?



### WHAT IF THERE WERE OTHER KINDS OF SHOES?

Could we use the same prototype to create boots, sneakers, sandals, and...uh...?



## WHAT IF THERE WERE OTHER KINDS OF SHOES?

Could we use the same prototype to create boots, sneakers, sandals, and...uh...?











### WE MIGHT BUILD A PROTOTYPE WITH EMPTY PROPERTIES...

With a generic "shoe", we could build all of our shoes, and assign property values later.











```
var shoe = { size: undefined, gender: undefined, construction: undefined };

All this object has is a bunch of property names with no values. Now what?
```

```
var mensBoot = Object.create( shoe );
```

```
mensBoot.size = 12;
mensBoot.gender = "men";
mensBoot.construction = "boot";
```

```
var flipFlop = Object.create( shoe );
```

```
flipFlop.size = 5;
flipFlop.gender = "women";
flipFlop.construction = "flipflop";
```

A class is a set of Objects that all share and inherit from the same basic prototype.











All Shoes

Some Shoes

size



A class is a set of Objects that all share and inherit from the same basic prototype.











All Shoes

size

Some Shoes

color



A class is a set of Objects that all share and inherit from the same basic prototype.











All Shoes

size color Some Shoes

gender



A class is a set of Objects that all share and inherit from the same basic prototype.











All Shoes

size color gender Some Shoes

construction



A class is a set of Objects that all share and inherit from the same basic prototype.











All Shoes

size color gender construction Some Shoes

laceColor



A class is a set of Objects that all share and inherit from the same basic prototype.











All Shoes

size color gender construction



Some Shoes

laceColor



A class is a set of Objects that all share and inherit from the same basic prototype.











All Shoes

size color gender construction

jewels

Some Shoes

laceColor
laceUp()



A class is a set of Objects that all share and inherit from the same basic prototype.











#### All Shoes

size color gender construction

bowPosition

#### Some Shoes

laceColor
laceUp()
jewels



A class is a set of Objects that all share and inherit from the same basic prototype.











#### All Shoes

size color gender construction



#### Some Shoes

laceColor
laceUp()
 jewels
bowPosition



A class is a set of Objects that all share and inherit from the same basic prototype.











#### All Shoes

size color gender construction putOn()

dimensionalTravel()

#### Some Shoes

laceColor
laceUp()
 jewels
bowPosition



A class is a set of Objects that all share and inherit from the same basic prototype.











#### All Shoes

size color gender construction putOn()

takeOff()

#### Some Shoes

laceColor
laceUp()
jewels
bowPosition
dimensionalTravel()

A class is a set of Objects that all share and inherit from the same basic prototype.











#### All Shoes



With a good set of common properties we can expect ALL shoes to have, we're ready to build a Constructor for our class.

Since not all shoes have these properties, they shouldn't go in the prototype

#### Some Shoes

laceColor
 laceUp()
 jewels
 bowPosition
dimensionalTravel()

A constructor allows us to set up inheritance while also assigning specific property values.











#### All Shoes

size color gender construction putOn() takeOff() function Shoe (shoeSize, shoeColor, forGender, constructStyle) {

Caritalizina this function's name.

Capitalizing this function's name distinguishes it as a maker of an entire "Class" of Objects... a constructor.

}

A constructor allows us to set up inheritance while also assigning specific property values.











#### All Shoes

size color gender construction putOn() takeOff() function Shoe (shoeSize, shoeColor, forGender, constructStyle) {

Each of these parameters will be specific values for a specific kind of Shoe. The constructor function will "construct" a new "instance" of a Shoe and assign these values to it.

A constructor allows us to set up inheritance while also assigning specific property values.











#### All Shoes

size color gender construction putOn() takeOff()

```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {

this.size = shoeSize;
this.color = shoeColor;
this.gender = forGender;
this.construction = constructStyle;

The this keyword inside a constructor will automatically refer to the new instance of the class that is being made.
```

A constructor allows us to set up inheritance while also assigning specific property values.











#### All Shoes

size color gender construction putOn() takeOff()

A constructor allows us to set up inheritance while also assigning specific property values.











```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
   this.size = shoeSize;
   this.color = shoeColor;
   this.gender = forGender;
   this.construction = constructStyle;

   this.putOn = function () { alert("Shoe's on!"); };
   this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

A constructor allows us to set up inheritance while also assigning specific property values.











```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
  this.size = shoeSize;
  this.color = shoeColor;
  this.gender = forGender;
  this.construction = constructStyle;

  this.putOn = function () { alert("Shoe's on!"); };
  this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

JavaScript's 'new' keyword produces a new Object of the class, or "instantiates" the class.











```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
   this.size = shoeSize;
   this.color = shoeColor;
   this.gender = forGender;
   this.construction = constructStyle;
   this.putOn = function () { alert("Shoe's on!"); };
   this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

The new keyword asks to build a new instance of something. What something? A Shoe, in this case.

var beachShoe = new Shoe( 10, "blue", "women", "flipflop" );

JavaScript's 'new' keyword produces a new Object of the class, or "instantiates" the class.







```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
   this.size = shoeSize;
   this.color = shoeColor;
   this.gender = forGender;
   this.construction = constructStyle;
   this.putOn = function () { alert("Shoe's on!"); };
   this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

```
var beachShoe = new Shoe( 10, "blue", "women", "flipflop" );
```





JavaScript's 'new' keyword produces a new Object of the class, or "instantiates" the class.











```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
   this.size = shoeSize;
   this.color = shoeColor;
   this.gender = forGender;
   this.construction = constructStyle;
   this.putOn = function () { alert("Shoe's on!"); };
   this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

```
var beachShoe = new Shoe( 10, "blue", "women", "flipflop" );
console.log( beachShoe );
```

JavaScript's 'new' keyword produces a new Object of the class, or "instantiates" the class.





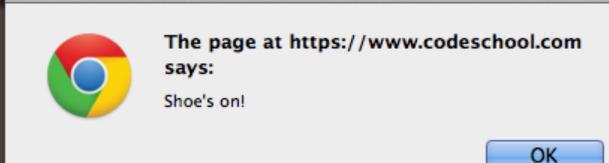






```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
   this.size = shoeSize;
   this.color = shoeColor;
   this.gender = forGender;
   this.construction = constructStyle;
   this.putOn = function () { alert("Shoe's on!"); };
   this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

var beachShoe = new Shoe( 10, "blue", "women", "flipflop" );
beachShoe.putOn();



JavaScript's 'new' keyword produces a new Object of the class, or "instantiates" the class.











```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
   this.size = shoeSize;
   this.color = shoeColor;
   this.gender = forGender;
   this.construction = constructStyle;
   this.putOn = function () { alert("Shoe's on!"); };
   this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

```
var beachShoe = new Shoe( 10, "blue", "women", "flipflop" );
beachShoe.straps = 2;
```

Later, we could add properties that are more shoespecific.

JavaScript's 'new' keyword produces a new Object of the class, or "instantiates" the class.











```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
  this.size = shoeSize;
  this.color = shoeColor;
  this.gender = forGender;
  this.construction = constructStyle;
  this.putOn = function () { alert("Shoe's on!"); };
  this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

```
var beachShoe = new Shoe( 10, "blue", "women", "flipflop" );
beachShoe.straps = 2;
```

Hold on, where's my efficient inheritance?



JavaScript's 'new' keyword produces a new Object of the class, or "instantiates" the class.











```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
   this.size = shoeSize;
   this.color = shoeColor;
   this.gender = forGender;
   this.construction = constructStyle;
   this.putOn = function () { alert("Shoe's on!"); };
   this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

Since these functions don't change between any Shoe, we should put them in a Shoe prototype so that they are stored efficiently in only one location that all Shoes can access.

### **ASSIGNING A PROTOTYPE TO A CONSTRUCTOR**

By setting a constructor's prototype property, every new instance will refer to it for extra properties!











```
function Shoe (shoeSize, shoeColor, forGender, constructStyle) {
   this.size = shoeSize;
   this.color = shoeColor;
   this.gender = forGender;
   this.construction = constructStyle;
   this.putOn = function () { alert("Shoe's on!"); };
   this.takeOff = function () { alert("Uh, what's that smell?"); };
}
```

Shoe.prototype = {

We build a new, secret Object within the constructor

function's prototype property! This will tell every

created Shoe to inherit from that Object.

Array.prototype



Object.prototype