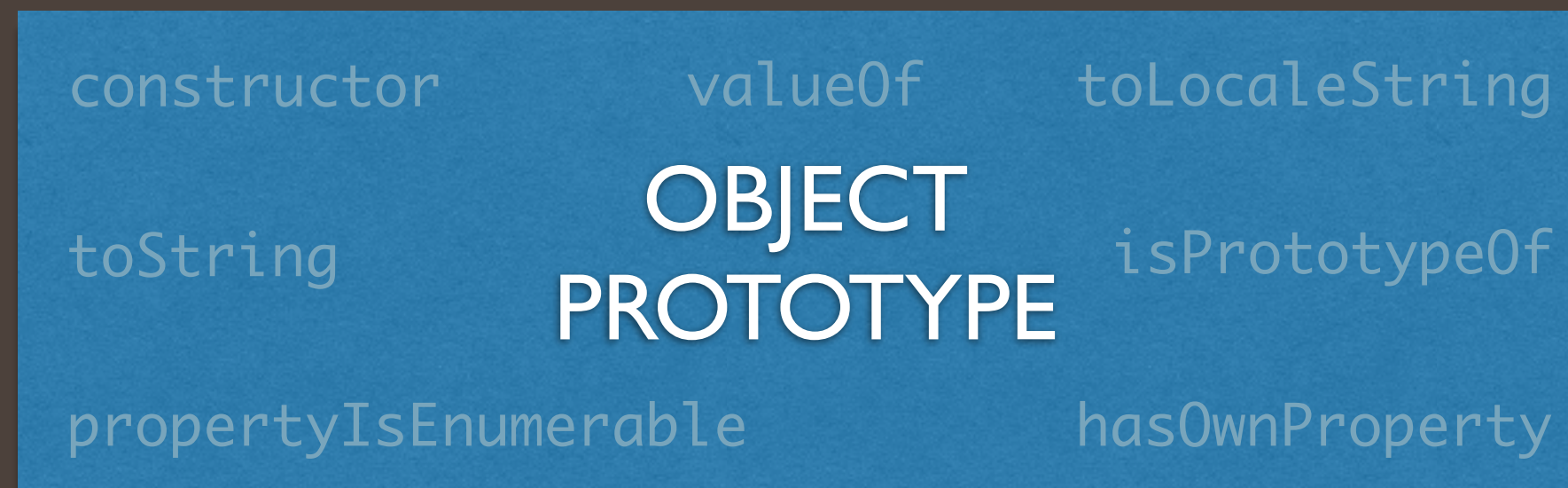


# THE OBJECT'S PARENT IS CALLED ITS “PROTOTYPE”

When a generic Object is created, its prototype passes it many important properties



```
var aquarium = {🐠, 🐡, 🏰, 🐉, addCriticter, takeOut, countFish,  
};
```

# THE OBJECT'S PARENT IS CALLED ITS “PROTOTYPE”

When a generic Object is created, its prototype passes it many important properties

*A Prototype is like a blueprint Object for the Object we are trying to create.*

OBJECT  
PROTOTYPE

A blue rectangular box with the text 'OBJECT PROTOTYPE' in white. Two grey arrows point downwards from the bottom of the box towards the code block below.

```
var aquarium = {🐠, 🐟, 🏰, 🐉, addCriticter, takeOut, countFish,  
  constructor, valueOf, toLocaleString, isPrototypeOf,  
  toString, propertyIsEnumerable, hasOwnProperty, ... };
```

# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

*A Prototype is like a blueprint Object for the Object we are trying to create.*

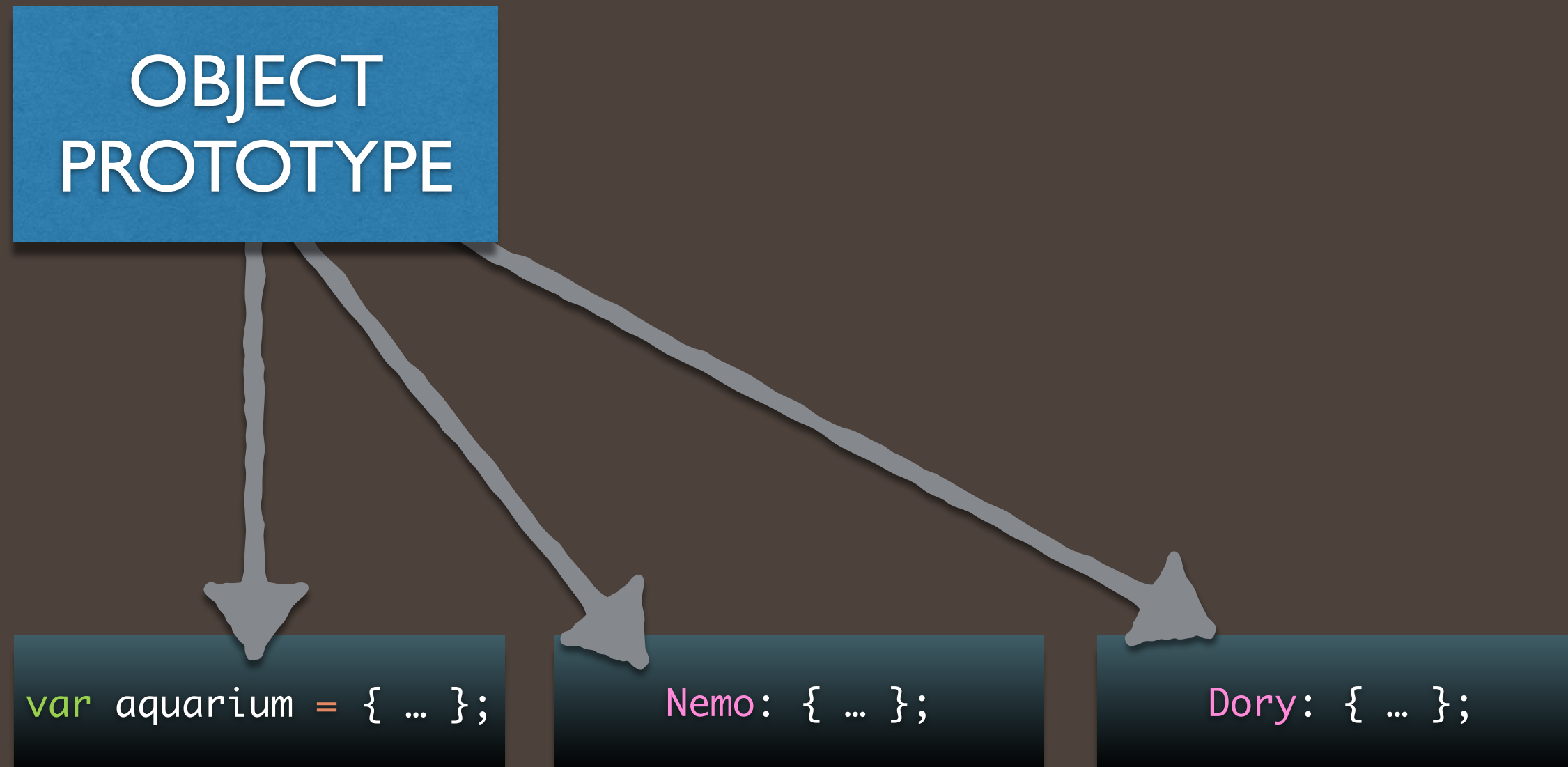
OBJECT  
PROTOTYPE

A blue rectangular box with the text 'OBJECT PROTOTYPE' in white. Two grey arrows point downwards from the bottom of the box towards the code block below.

```
var aquarium = { 🐠, 🐟, 🏰, 🐉, addCriticter, takeOut, countFish,  
  constructor, valueOf, toLocaleString, isPrototypeOf,  
  toString, propertyIsEnumerable, hasOwnProperty, ... };
```

# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.



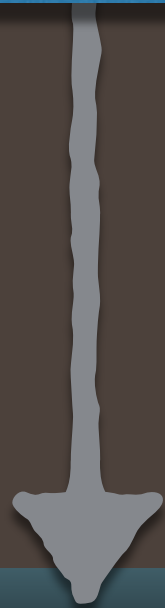
So far, all the Object literals we've made with `{ }` inherit directly from the highest level in the JS hierarchy, the Object prototype....



# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

OBJECT  
PROTOTYPE



```
var aquarium = { ... };
```

Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!

ARRAY  
Prototype

length  
pop()  
push()

shift()  
reverse()  
sort()

join()  
reduce()  
slice()



```
var myArray = [ "This", "array", "inherits", "properties",  
                "from", "the", "Array", "prototype!" ];
```

# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

## OBJECT PROTOTYPE



```
var aquarium = { ... };
```

Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!

## ARRAY Prototype



length  
pop()  
push()

shift()  
reverse()  
sort()

join()  
reduce()  
slice()

```
var myArray = [ "This", "array", "inherits", "properties",  
                "from", "the", "Array", "prototype!" ];
```

myArray.  
myArray.  
myArray.

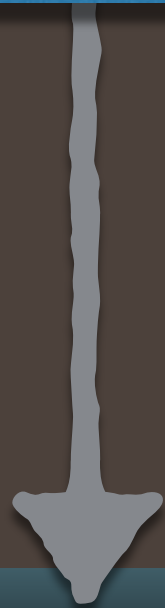
myArray.  
myArray.  
myArray.

myArray.  
myArray.  
myArray.

# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

## OBJECT PROTOTYPE



```
var aquarium = { ... };
```

*Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!*

## ARRAY Prototype



```
var myArray = [ "This", "array", "inherits", "properties",  
                "from", "the", "Array", "prototype!" ];
```

myArray.length

myArray.pop()

myArray.push()

myArray.shift()

myArray.reverse()

myArray.sort()

myArray.reduce()

myArray.join()

myArray.slice()

# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

OBJECT  
PROTOTYPE

Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!

ARRAY  
Prototype

STRING  
Prototype

length  
charAt()  
trim()

concat()  
indexOf()  
replace()

toUpperCase()  
toLowerCase()  
substring()

```
var aquarium = { ... };
```

```
var myString = "I am secretly a child of the String prototype."
```



# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

## OBJECT PROTOTYPE

Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!

### ARRAY Prototype

### STRING Prototype

length  
charAt()  
trim()

concat()  
indexOf()  
replace()

toUpperCase()  
toLowerCase()  
substring()

```
var aquarium = { ... };
```

```
var myString = "I am secretly a child of the String prototype."
```

myString.  
myString.  
myString.

myString.  
myString.  
myString.

myString.  
myString.  
myString.

# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

OBJECT  
PROTOTYPE

Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!

ARRAY  
Prototype

STRING  
Prototype

```
var aquarium = { ... };
```

```
var myString = "I am secretly a child of the String prototype."
```

myString.length  
myString.charAt()  
myString.trim()

myString.concat()  
myString.indexOf()  
myString.replace()

myString.toUpperCase()  
myString.toLowerCase()  
myString.substring()

# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

OBJECT  
PROTOTYPE

*Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!*

ARRAY  
Prototype

STRING  
Prototype

NUMBER  
Prototype

toFixed()  
toExponential()  
toPrecision()

```
var aquarium = { ... };
```

```
var myNumber = 6;
```

myNumber.

myNumber.

myNumber.

# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

OBJECT  
PROTOTYPE

*Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!*

ARRAY  
Prototype

STRING  
Prototype

NUMBER  
Prototype

```
var aquarium = { ... };
```

```
var myNumber = 6;
```

```
myNumber.toFixed()  
myNumber.toExponential()  
myNumber.toPrecision()
```



# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

OBJECT  
PROTOTYPE

Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!

ARRAY  
Prototype

STRING  
Prototype

NUMBER  
Prototype

FUNCTION  
Prototype

name      call()  
bind()    apply()

```
var aquarium = { ... };
```

```
function myFunction(){  
  return "Functions have secret properties too!";  
}
```

myFunction.  
myFunction.

myFunction.  
myFunction.

# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

OBJECT  
PROTOTYPE

*Turns out, all of the native JS data structures inherit all of their properties and methods from their very own prototypes!*

ARRAY  
Prototype

STRING  
Prototype

NUMBER  
Prototype

FUNCTION  
Prototype

```
var aquarium = { ... };
```

```
function myFunction(){  
  return "Functions have secret properties too!";  
}
```

myFunction.name

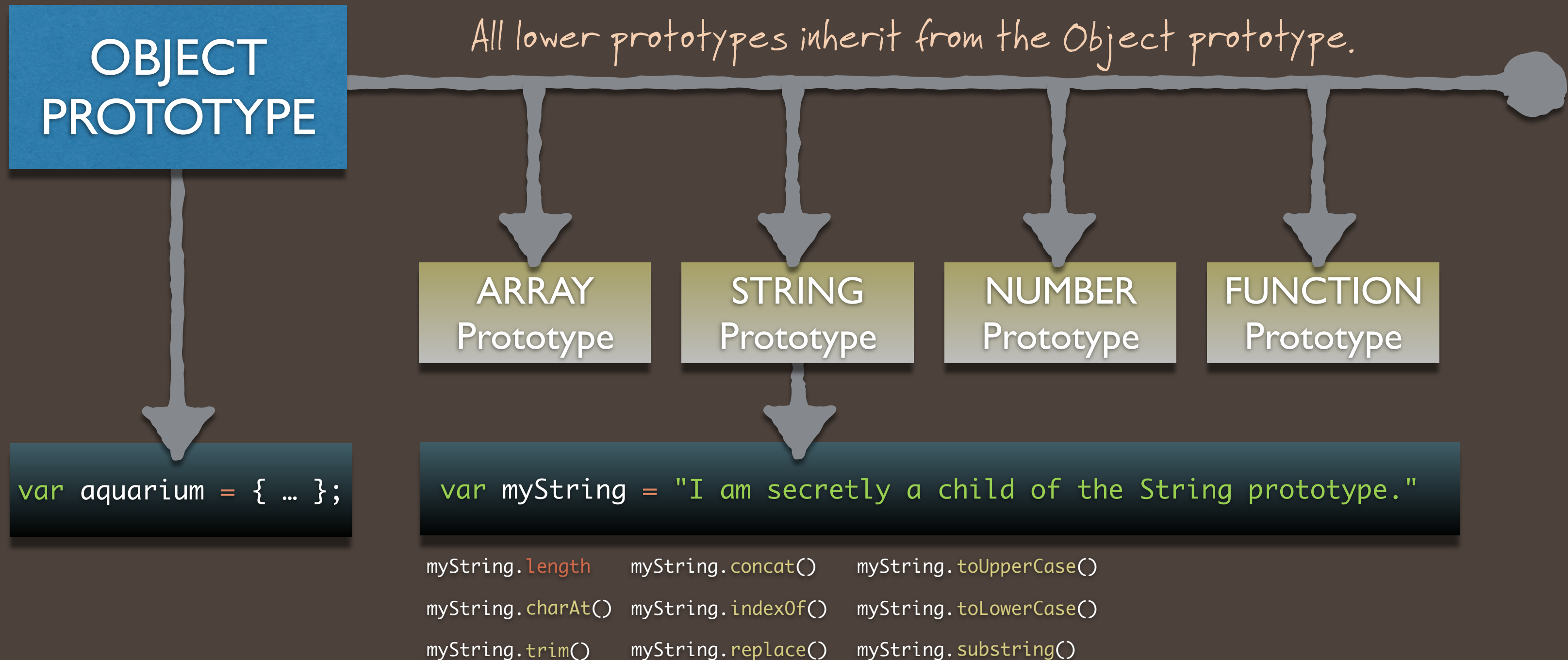
myFunction.bind()

myFunction.call()

myFunction.apply()

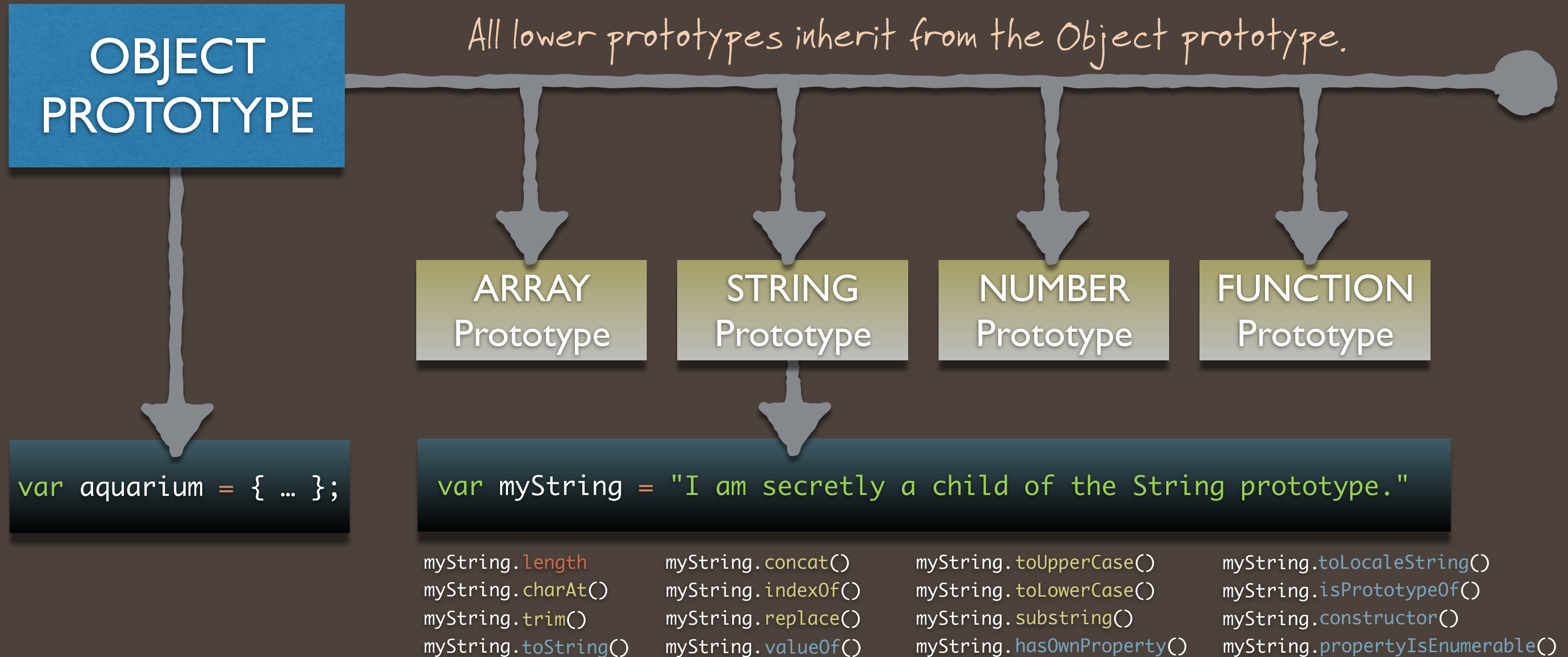
# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.



# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.





# PASSING DOWN PROPERTIES IS CALLED “INHERITANCE”

Inheritance helps avoid over-coding multiple properties and methods into similar objects.

OBJECT  
PROTOTYPE

```
graph TD; OP[OBJECT PROTOTYPE] --> SP[STRING Prototype]; SP --> MS["var myString = 'I am secretly a child of the String prototype.'"]
```

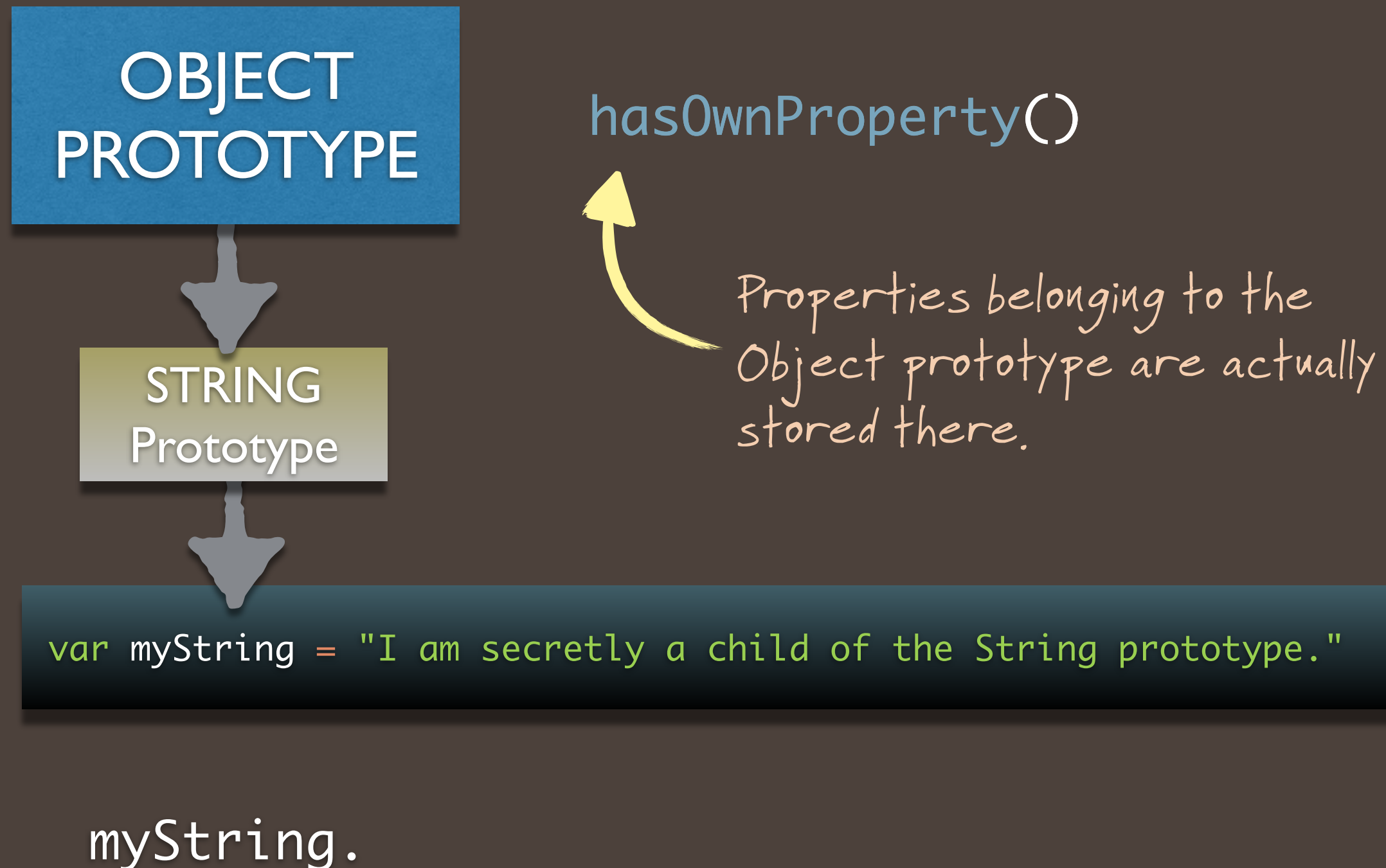
The diagram illustrates the concept of inheritance in a prototype-based system. It starts with a blue box labeled 'OBJECT PROTOTYPE' on the left. A horizontal line extends from this box to the right, ending in a small grey circle. A vertical arrow points down from the center of this line to a yellow box labeled 'STRING Prototype'. Another vertical arrow points down from the center of the yellow box to a dark blue box containing the code: `var myString = "I am secretly a child of the String prototype."`

STRING  
Prototype

```
var myString = "I am secretly a child of the String prototype."
```

# INHERITANCE AVOIDS DUPLICATE MEMORY STORAGE

Though properties are inherited, they are still “owned” by prototypes, not the inheriting Object



# INHERITANCE AVOIDS DUPLICATE MEMORY STORAGE

Though properties are inherited, they are still “owned” by prototypes, not the inheriting Object

OBJECT  
PROTOTYPE

`hasOwnProperty()`

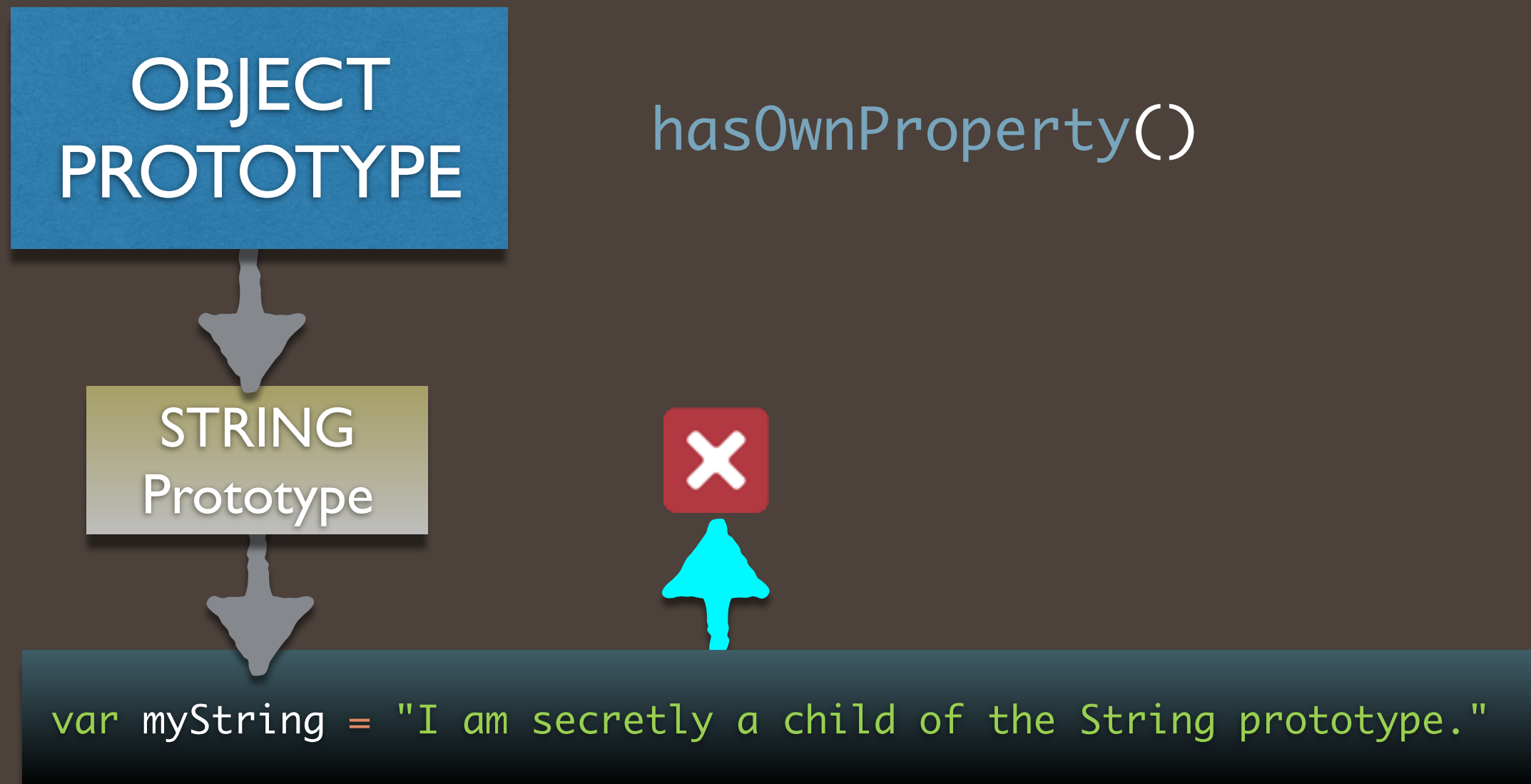
STRING  
Prototype

```
var myString = "I am secretly a child of the String prototype."
```

`myString.hasOwnProperty()`

# INHERITANCE AVOIDS DUPLICATE MEMORY STORAGE

Though properties are inherited, they are still “owned” by prototypes, not the inheriting Object



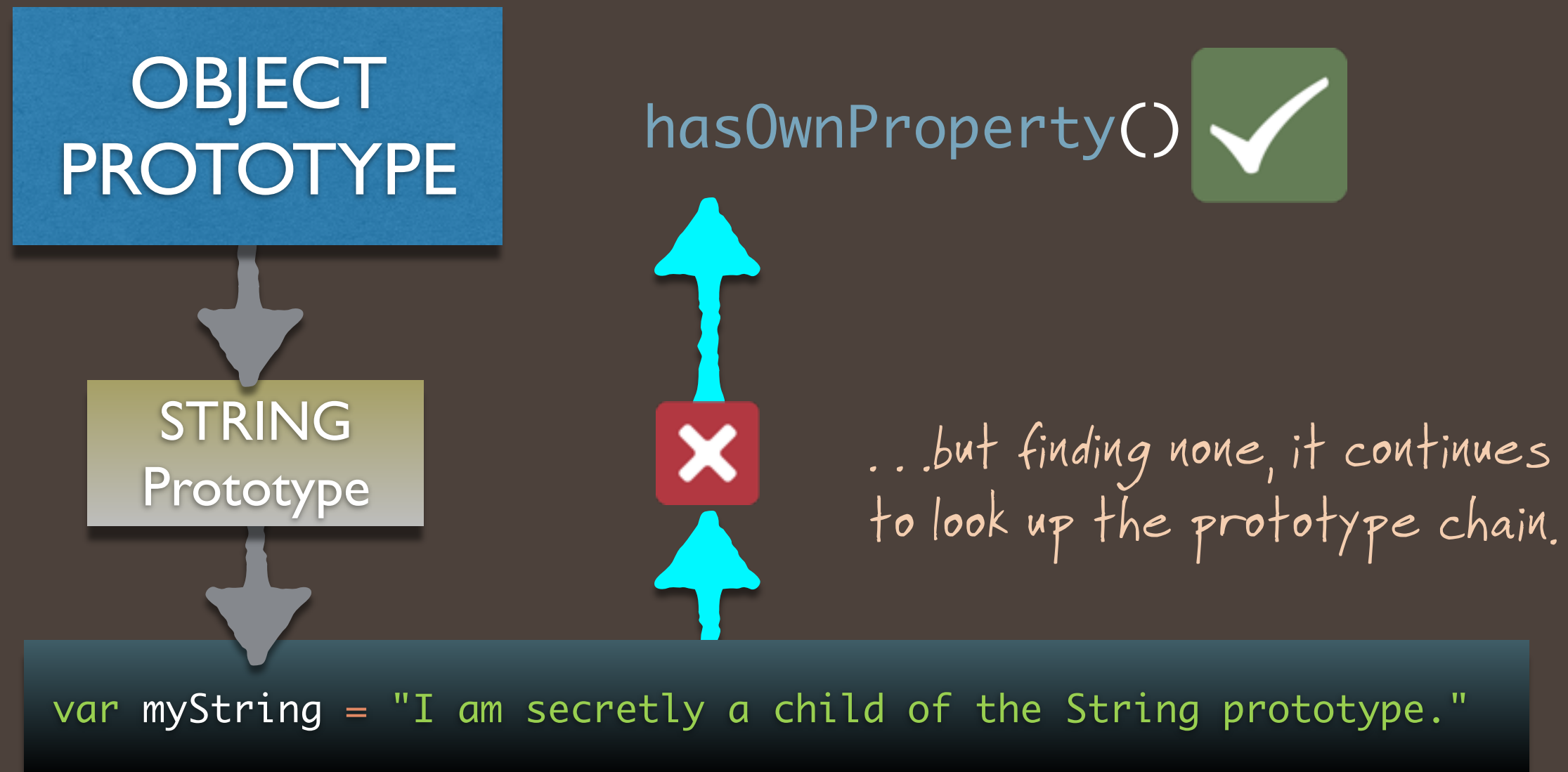
`myString.hasOwnProperty()`

When we call this function on a string, the string first looks up to the String prototype to find it...



# INHERITANCE AVOIDS DUPLICATE MEMORY STORAGE

Though properties are inherited, they are still “owned” by prototypes, not the inheriting Object



`myString.hasOwnProperty()`

# INHERITANCE AVOIDS DUPLICATE MEMORY STORAGE

Though properties are inherited, they are still “owned” by prototypes, not the inheriting Object

OBJECT  
PROTOTYPE



STRING  
Prototype



```
var myString = "I am secretly a child of the String prototype."
```

hasOwnProperty()



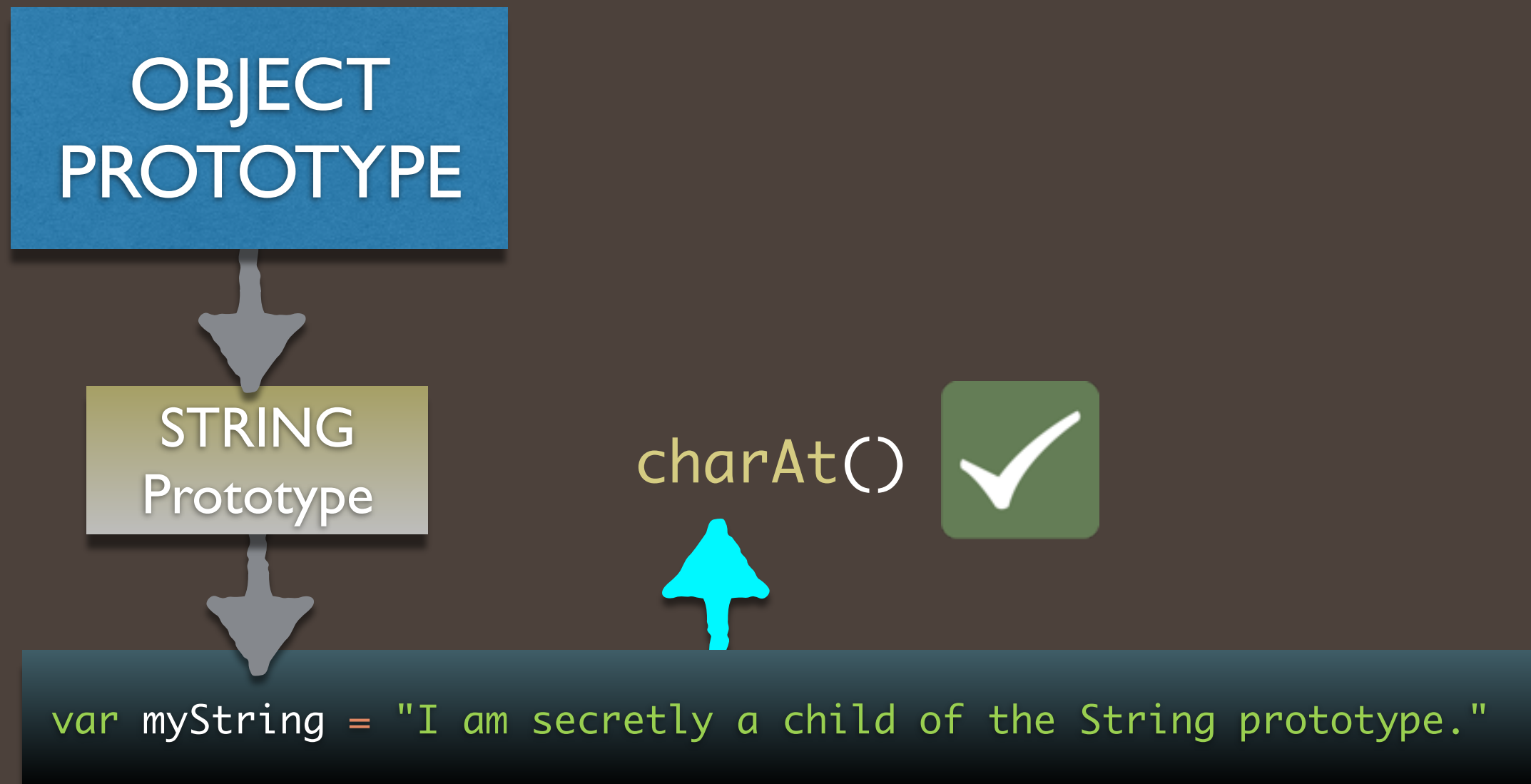
Success! The prototype provides access to the `hasOwnProperty` method without needing it be stored in `myString`.



`myString.hasOwnProperty()`

# INHERITANCE AVOIDS DUPLICATE MEMORY STORAGE

Though properties are inherited, they are still “owned” by prototypes, not the inheriting Object



myString.charAt()

Same goes with a common String property function like charAt...

# INHERITANCE AVOIDS DUPLICATE MEMORY STORAGE

Though properties are inherited, they are still “owned” by prototypes, not the inheriting Object

OBJECT  
PROTOTYPE



STRING  
Prototype



Since the String prototype owns `charAt`, the search stops there, and the String prototype grants `myString` access to the function. Woot, inheritance!

`charAt()`



```
var myString = "I am secretly a child of the String prototype."
```

`myString.charAt()`



# ADDING INHERITABLE PROPERTIES TO PROTOTYPES

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

# of  
A's

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

# ADDING INHERITABLE PROPERTIES TO PROTOTYPES

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

# of  
E's

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

# 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.";
```

```
function countAll ( string, letter ) { ... }
```

# 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.";
```

countAll

# 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



# 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

dorothy.countAll("h");

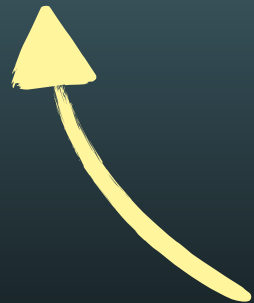
When `countAll` is part of the prototype, we'll be able to call it from any string! Let's add it in.

# 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



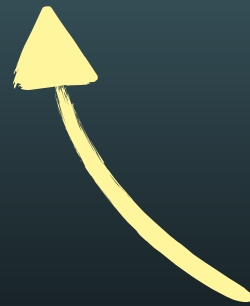
This dot notation finds the prototype  
for all String values everywhere.

# 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 =



To add our function to the String prototype object, we use another dot and name the property with our function's name. This will make it inheritable by all Strings as `countAll`.

# 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 ){
```



Since we are giving the function to the overarching String prototype, we won't need to pass the function a string...


```
};
```

# 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 ){
```



We need to make sure our function can accept a requested letter as a parameter, so that it will return a count for any letter we want.

```
};
```



# 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;
```



*We get a counter variable ready...*

```
};
```

# 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++) {
    }
};
```



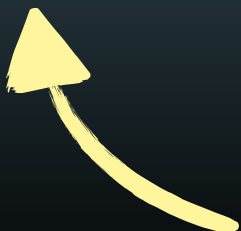
Since the string we're interested in will be calling `countAll` on itself, the function should look for its caller's length using `this`.

# 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++;
    }
  }
};
```



We look at the current character in this string...

# 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++;  
        }  
    }  
};
```

...and convert it to Upper Case to simplify our search. If it's already Upper case, it will stay upper case.

"bam!".toUpperCase()

→ BAM!




# 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() ) {
      } We compare the converted current
    } character to the converted letter
  } to see if we have a match!
};
```






# 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++;  
        }  
    }  
};
```



If we found a letter, we increment the counter.

# 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;
};
```

Lastly, the function  
returns the final amount.

