

Using Prototype Objects

This lesson teaches us how to add properties and methods to an object prototype.

WE'LL COVER THE FOLLOWING ^

- Adding Properties
 - Syntax
 - Example
- Adding Methods
 - Syntax
 - Example

Adding Properties

We already discussed why properties should be added to the *prototype* object of a *constructor* function. Now let's look at the method to do so.

Syntax

Here is how *properties* can be added to a *constructor function* using the *prototype* property:

```
ConstructorFunctionName.prototype.PropertyName = PropertyValue
```



Syntax for adding properties using Prototype object

Since the property is being defined on the *prototype* of the *constructor function*, the double dot notation has to be used to set property values. First, the prototype property is accessed using the dot notation, then the property is defined on it using the dot operator.

Similarly, the double dot notation has to be used to get the property values defined on the prototype object as well.



```
//accessing the constructor property of a Prototype Object  
ConstructorFunctionName.prototype.constructor
```

Accessing "constructor" property of a Prototype Object

Example

Let's take a look at an example below:

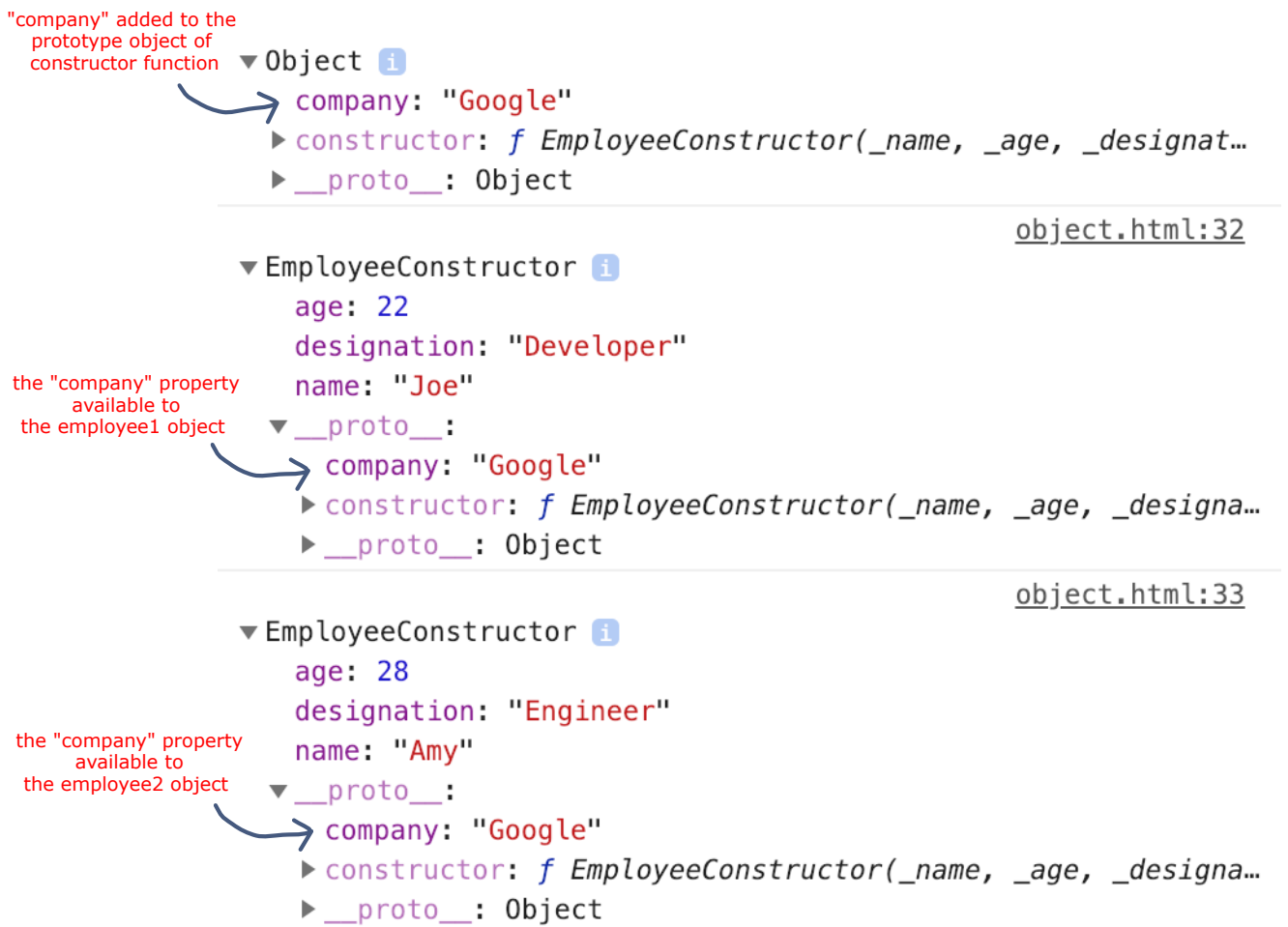


```
//constructor function called EmployeeConstructor  
function EmployeeConstructor(_name, _age, _designation){  
  this.name = _name  
  this.age = _age  
  this.designation = _designation  
}  
  
//adding a property company to the constructor  
EmployeeConstructor.prototype.company = 'Google'  
  
//creating an object called employeeObj1  
var employeeObj1 = new EmployeeConstructor('Joe', 22, 'Developer')  
  
//displaying properties of employeeObj1  
console.log("Name of employee:",employeeObj1.name)  
console.log("Age of employee:",employeeObj1.age)  
console.log("Designation of employee:",employeeObj1.designation)  
console.log("Employee works in the company:",employeeObj1.company)  
  
//creating another object called employeeObj2  
var employeeObj2 = new EmployeeConstructor('Amy', 28, 'Engineer')  
  
//displaying properties of employeeObj2  
console.log("Name of employee:",employeeObj2.name)  
console.log("Age of employee:",employeeObj2.age)  
console.log("Designation of employee:",employeeObj2.designation)  
console.log("Employee works in the company:",employeeObj2.company)
```



Creating Object Instances

The property **company** is added to the prototype object of the **EmployeeConstructor** *constructor function*. Due to this, it automatically becomes available to both **employeeObj1** and **employeeObj2** when they are created. Hence, it can directly be accessed by both object instances.



EmployeeConstructor's Prototype object shared by both "employeeObj1" and "employeeObj1"

Adding Methods

Just like properties, *methods* can also be added to a *constructor function's* *prototype* object.

Syntax

Here is how *methods* can be added to a *constructor function* using the *prototype* property:

```
ConstructorFunctionName.prototype.MethodName = function () {  
    //function body  
}
```



Example

Let's take a look at an example below:

```
//constructor function called Employee
function EmployeeConstructor(_name, _age, _designation){
  this.name = _name
  this.age = _age
  this.designation = _designation
}

//adding a property company to the constructor
EmployeeConstructor.prototype.displayName = function () {
  return this.name
}

//creating an object called employeeObj1
var employeeObj1 = new EmployeeConstructor('Joe', 22, 'Developer')

//calling the function for employeeObj1
console.log("Name of employee is:",employeeObj1.displayName())

//creating another object called employeeObj2
var employeeObj2 = new EmployeeConstructor('Amy', 28, 'Engineer')

//calling the function for employeeObj2
console.log("Name of employee is:",employeeObj2.displayName())
```



Creating Object Instances

The method `displayName` is added to the prototype object of the `EmployeeConstructor` *constructor* function. Due to this, it automatically becomes available to both `employeeObj1` and `employeeObj2` when they are created. Now, the method can directly be accessed by object instances.

"displayName" method added
to the prototype object of
constructor function

```
▼ Object ⓘ  
  ► displayName: f ()  
  ► constructor: f EmployeeConstructor(_name, _age, _designat...  
  ► __proto__: Object
```

[object.html:31](#)

the "displayName"
method available to
the employee1 object

```
▼ EmployeeConstructor ⓘ  
  age: 22  
  designation: "Developer"  
  name: "Joe"  
  ▼ __proto__:  
    ► displayName: f ()  
    ► constructor: f EmployeeConstructor(_name, _age, _designa...  
    ► __proto__: Object
```

[object.html:32](#)

the "displayName"
method available to
the employee2 object

```
▼ EmployeeConstructor ⓘ  
  age: 28  
  designation: "Engineer"  
  name: "Amy"  
  ▼ __proto__:  
    ► displayName: f ()  
    ► constructor: f EmployeeConstructor(_name, _age, _designa...  
    ► __proto__: Object
```

EmployeeConstructor's Prototype object shared by both "employeeObj1" and "employeeObj2"

So far, whether we used object literals or constructor functions to create objects, all their properties were accessible outside them. This brings us to the question: is it possible to protect or hide the properties to prevent unauthorized access? Let's discuss this in the next lesson!