Prototype Objects

This lesson explains prototype objects, what they are, how they are accessed, and why they are used to add properties.

WE'LL COVER THE FOLLOWING

In the previous lesson, we asked the question if there was a simpler approach for adding new methods/properties to a constructor function. This introduces us to the major concept of **"Prototype Objects"** in JavaScript.

Before we get into how it can be used, let's discuss what a *prototype* is.

Prototype Properties in Objects

Instead of jumping directly into the definition, let's begin by looking at the code below:

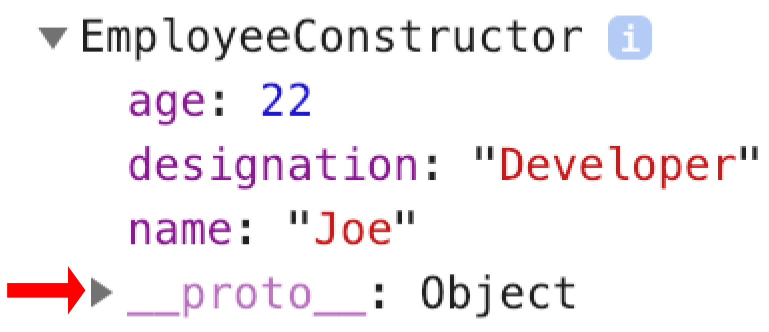
```
//creating an constructor function  named EmployeeConstructor
function EmployeeConstructor(_name,_age,_designation) {
   this.name = _name,
   this.age = _age,
   this.designation = _designation
}
//creating an object from EmployeeConstructor using "new" keyword
//The object created is stored in the variable employeeObj
var employeeObj = new EmployeeConstructor('Joe',22,'Developer')
console.log(employeeObj)
```

As seen above, if we display `employeeObj`, the three properties `name`, `age` and

As seen above, if we display `employeeObj`, the three properties, `name`, `age` and `designation` are displayed along with their values, encapsulated within the

`EmployeeConstructor` object, which was created using `new`.

If the same code is run locally on the browser, the browser console will display the following:



```
▼ EmployeeConstructor
    age: 22
    designation: "Developer"
    name: "Joe"
  ▶ __proto__: Object
```

Prototype Object

Notice anything different? Apart from `age`, `designation` and `name`, there is an additional property created inside the `EmployeeConstructor` object. This is known as the `[[Prototype]]` property of that object. It is a hidden property that is present inside every object created from a constructor function.

Prototype Object

So far, we know the following:

- Any object instance created from a *constructor function* has a hidden property.
- This hidden property is called the `[[Prototype]]` property of the object.

Just like objects, whenever a *constructor function* is initialized, it also gets a `[[Prototype]]` property assigned to it.

The `[[Prototype]]` property of a *constructor function* is an **object** itself containing further properties; hence, it is also known as a **Prototype Object**.

At this point, you should know that the `[[Prototype]]` property of an object instance, `employeeObj` in our case, always points to the **Prototype Object** of the *constructor function* from which it was created, `EmployeeConstructor` in our case.

> **Note:** The prototype property in **Prototype Object** points to `null` .

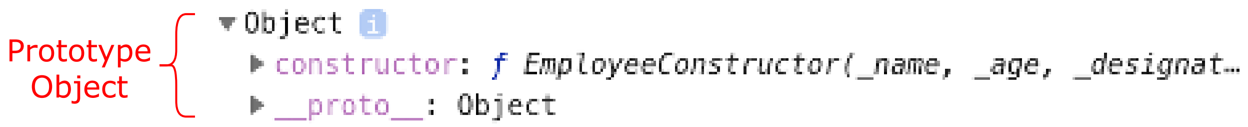
Accessing a Prototype Object

The method for accessing the `[[Prototype]]` property of an object instance will be discussed later in this course. In this lesson, we will only discuss how to access a **Prototype Object**.

In order to access the **Prototype Object** of the `EmployeeConstructor` *constructor function* the following syntax is used:

```
//accessing the prototype property of "Employee"
EmployeeConstructor.prototype
```

When the `EmployeeConstructor`'s `[[Prototype]]` property is accessed, the browser console displays the following:



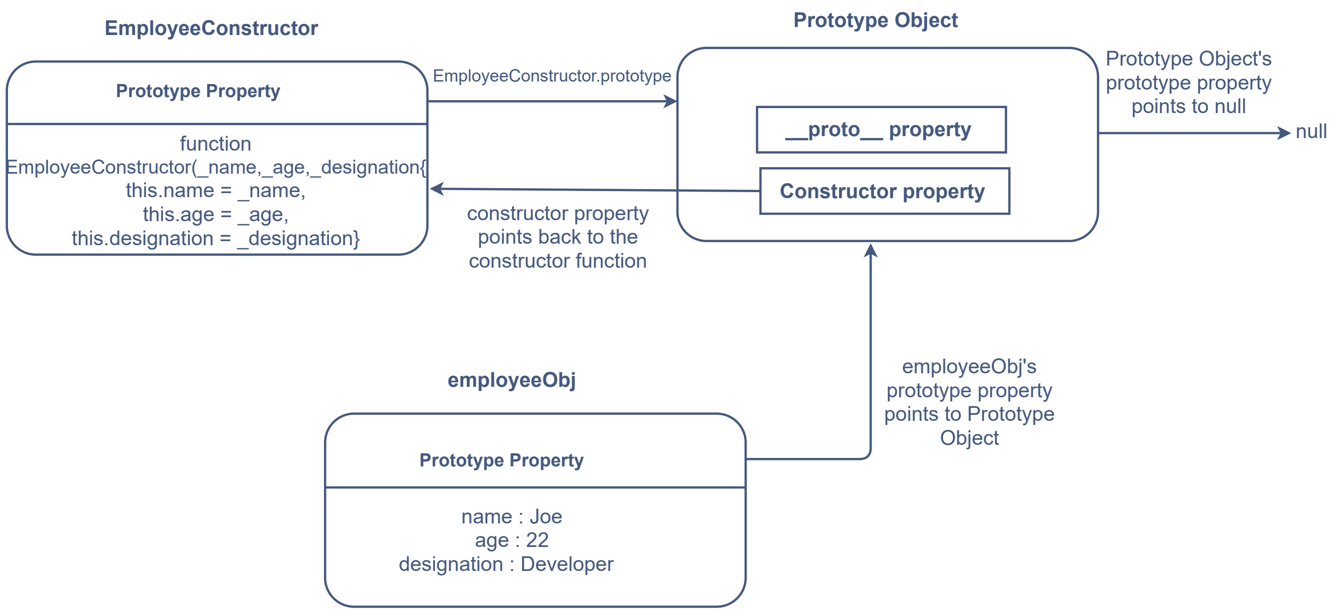
```
▼Object ℹ
  ▶constructor: ƒ EmployeeConstructor(_name, _age, _designat…
  ▶__proto__: Object
```

Constructor Function's Prototype Object

The **Prototype Object** for `EmployeeConstructor` consists of two further properties:

- The `constructor` which points back to the `EmployeeConstructor` *constructor function*.
- The `__proto__` property. You can ignore `__proto__` for now as it will be discussed later.

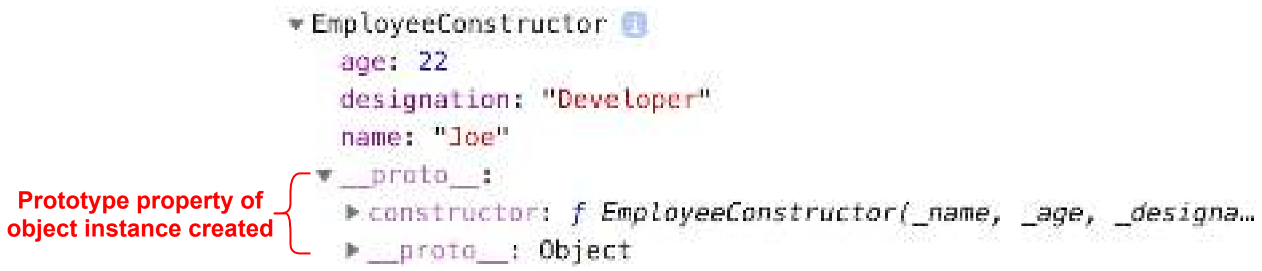
Take a look at the illustration below to visualize the concepts explained above:


EmployeeConstructor
Prototype Property
function EmployeeConstructor(_name,_age,_designation{
this.name = _name,
this.age = _age,
this.designation = _designation}
EmployeeConstructor.prototype
constructor property points back to the constructor function
Prototype Object
__proto__ property
Constructor property
Prototype Object's prototype property points to null
null
employeeObj's prototype property points to Prototype Object
employeeObj
Prototype Property
name : Joe
age : 22
designation : Developer

Why use Prototype Objects for Additions?

You might be wondering about the purpose of all this information and why a prototype object should be used to add properties or methods directly into the constructor.

At the start of this lesson, we created an object which was stored in `employeeObj`. Its properties were then displayed in the browser console. Take a look at the properties displayed again:



```
▼EmployeeConstructor
   age: 22
   designation: "Developer"
   name: "Joe"
  ▼__proto__:
   ▶constructor: ƒ EmployeeConstructor(_name, _age, _designa…
   ▶__proto__: Object
```

From the above image, you can see that the object instance's `[[Prototype]]` property points to the **Prototype object** of the *constructor function* `EmployeeConstructor`.

This brings us to a conclusion:

All objects created from a *constructor function* share its *Prototype Object*.

Due to this, if any properties/methods are added to the *prototype* object of the *constructor function*, they can be accessed by the objects created from it. This is why prototypes should be used for additions in the constructor functions.

Now that it’s clear what prototype objects are and why they’re used for adding properties, let’s discuss the method to do so in the next lesson!