

Object Instances

This lesson teaches us how to create object instances using constructor functions by explaining the syntax and giving an example.

We'll cover the following ^

- Creating an Object Instance
- Syntax
 - Explanation
- Example
 - Explanation

In the [previous](#) lesson, we learned how to create a *constructor function* and how all objects created using this will contain the properties defined inside the constructor. Now let's discuss how to create objects using the *constructor function*.

```
1 function Employee(_name, _age,  
2   this.name = _name  
3   this.age = _age  
4   this.designation = _designat  
5 }
```



Constructor Function for Employee

Creating an Object Instance

Every time a *new object* is created, it is referred to as a new *instance*. As discussed earlier, multiple object instances can be generated using constructor functions.

Syntax

Let's take a look at the syntax for creating an object:

```
1 var ObjectName = new Constructo
```



Syntax for creating an object instance

Explanation

- The *keyword* `new` is used to create a new object.
- That is followed by the *constructor function* being called with the required *arguments* passed into it. This is why the code in the *constructor function* executes every time a new object is instantiated.
- An object will then be returned which will be stored inside a variable, that is, `ObjectName` in the above case.
- Each *new* object created will store the argument values passed into the constructor function.

Example

Now let's create new objects using the `Employee` constructor function:

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

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

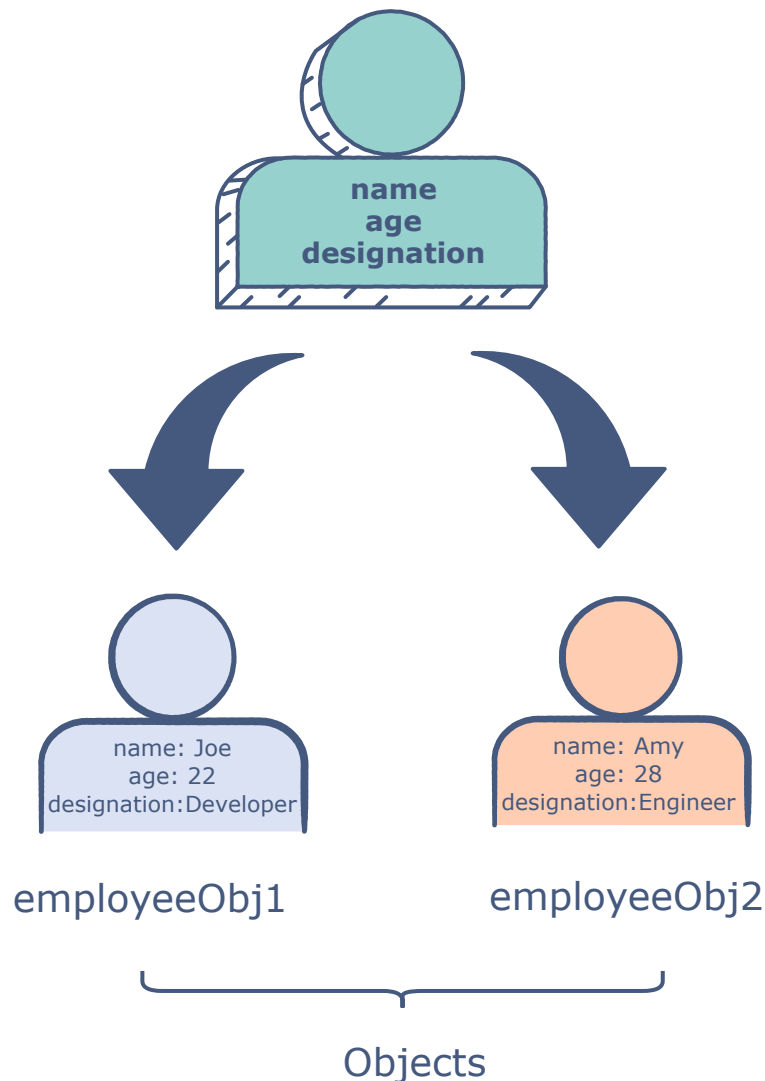
//displaying properties of employeeObj1
console.log("Name of employee:",employeeObj1.name)
console.log("Age of employee:",employeeObj1.age)
console.log("Designation of employee:",employeeObj1.designation)

//creating another object called employeeObj2
var employeeObj2 = new Employee('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)
```



Employee Blueprint



Explanation

- In **line 9** and **line 17** two new objects are created.
- Specific arguments for both the objects are passed into the constructor function, `Employee`.
- In **line 9** `Joe`, `22` and `Developer` are being passed as `name`, `age` and `designation` for `employeeObj1`.
- In **line 17** `Amy`, `28` and `Engineer` are being passed as `name`, `age` and `designation` for `employeeObj2`.
- When the `Employee` constructor function executes the object properties for both `employeeObj1` and `employeeObj2`, store the arguments passed into the `Employee` constructor in **lines 9** and **17**.
- The *properties* of an object are accessed by making a call to the object

The properties of an object are accessed by making a call to the object using that object's name, `employeeObj1` and `employeeObj2` in our example, followed by the property name. This can be seen in **lines 12-14** for `employeeObj1` and **lines 20-22** for `employeeObj2`.

Even though both `employeeObj1` and `employeeObj2` contain the same properties, both objects are created separately with their own specific arguments passed into the constructor, where `this` is used to assign them to the relevant properties. Hence, they are independent of each other, and upon access, they display their own property values, as discussed in the [previous](#) lesson.

In the next lesson, let's learn how to add new properties to a constructor function.