# DAY 2 – Javascript Object and Prototype chaining

## Introducing JavaScript objects

In JavaScript, most things are objects, from core JavaScript features like strings and arrays to the browser [APIs](https://developer.mozilla.org/en-US/docs/Glossary/API) built on top of JavaScript. You can even create your own objects to encapsulate related functions and variables into efficient packages, and act as handy data containers. The object-based nature of JavaScript is important to understand if you want to go further with your knowledge of the language, therefore we've provided this module to help you. Here we teach object theory and syntax in detail, then look at how to create your own objects.

## Object basics

An object is a collection of related data and/or functionality (which usually consists of several variables and functions — which are called properties and methods when they are inside objects.)

const person = {

name: ['Bob', 'Smith'],

age: 32,

gender: 'male',

interests: ['music', 'skiing'],

bio: function() {

alert(this.name[0] + ' ' + this.name[1] + ' is ' + this.age + ' years old. He likes ' + this.interests[0] + ' and ' + this.interests[1] + '.');

},

greeting: function() {

alert('Hi! I\'m ' + this.name[0] + '.');

}

};

What is "this"?

The this keyword refers to the current object the code is being written inside — so in this case this is equivalent to person.

const person1 = {

name: 'Chris',

greeting: function() {

alert('Hi! I\'m ' + this.name + '.');

}

}

const person2 = {

name: 'Deepti',

greeting: function() {

alert('Hi! I\'m ' + this.name + '.');

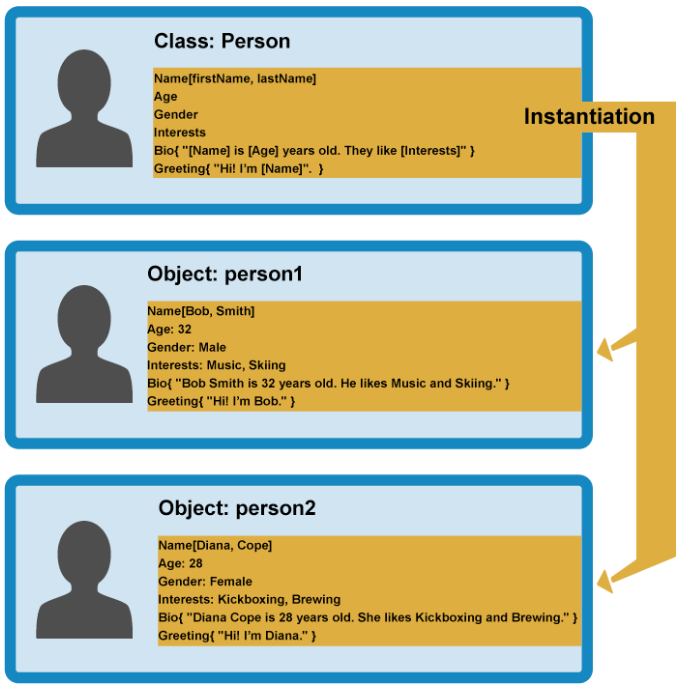
}

}

## Object-oriented JavaScript

The basic idea of OOP is that we use objects to model real world things that we want to represent inside our programs, and/or provide a simple way to access functionality that would otherwise be hard or impossible to make use of.

Let's consider a simple program that displays information about the students and teachers at a school.



When an object instance is created from a class, the class's **constructor function** is run to create it. This process of creating an object instance from a class is called **instantiation** — the object instance is **instantiated** from the class.

## Constructors and object instances

JavaScript uses special functions called **constructor functions** to define and initialize objects and their features.

function Person(name) {

this.name = name;

this.greeting = function() {

alert('Hi! I\'m ' + this.name + '.');

};

}

The constructor function is JavaScript's version of a class. Notice that it has all the features you'd expect in a function, although it doesn't return anything or explicitly create an object — it basically just defines properties and methods. Notice also the this keyword being used here as well — it is basically saying that whenever one of these object instances is created, the object's name property will be equal to the name value passed to the constructor call, and the greeting() method will use the name value passed to the constructor call too.

Now how to create object-

let person1 = new Person('Bob');

let person2 = new Person('Sarah');

In each case, the new keyword is used to tell the browser we want to create a new object instance, followed by the function name with its required parameters contained in parentheses, and the result is stored in a variable .

// See the objects in console

let person1 = new Object({

name: 'Chris',

age: 38,

greeting: function() {

alert('Hi! I\'m ' + this.name + '.');

}

});

Using the create() method

JavaScript has a built-in method called [create()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create) With it, you can create a new object based on any existing object.

let person2 = Object.create(person1);

## Object prototypes

Prototypes are the mechanism by which JavaScript objects inherit features from one another.

An object's prototype object may also have a prototype object, which it inherits methods and properties from, and so on. This is often referred to as a **prototype chain**, and explains why different objects have properties and methods defined on other objects available to them.

 the properties and methods are defined on the prototype property on the Objects' constructor functions, not the object instances themselves.

In JavaScript, a link is made between the object instance and its prototype (its \_\_proto\_\_ property, which is derived from the prototype property on the constructor), and the properties and methods are found by walking up the chain of prototypes.

Lets start with previous example again.

function Person(first, last, age, gender) {

// property and method definitions

this.name = {

'first': first,

'last' : last

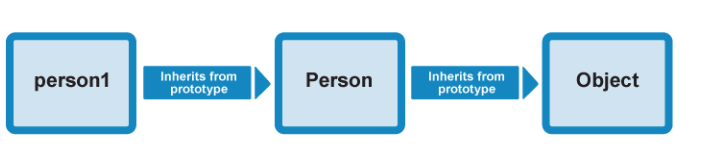
};

this.age = age;

this.gender = gender;

}

let person1 = new Person('Bob', 'Smith', 32, 'male');



See the property available on person1 object. // console or ide

Lets call

person1.valueOf()

This method — [Object.valueOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/valueOf) is inherited by person1 because its constructor is Person(), and Person()'s prototype is Object(). valueOf() returns the value of the object it is called on — try it and see! In this case, what happens is:

* The browser initially checks to see if the person1 object has a valueOf() method available on it, as defined on its constructor, Person().
* It doesn't, so the browser then checks to see if the Person() constructor's prototype object (Object()) has a valueOf() method available on it. It does, so it is called, and all is good!

1. You can check out existing prototype properties for yourself — go back to our previous example and try entering the following into the JavaScript console:

Person.prototype

1. The output won't show you very much because we haven't defined anything on our custom constructor's prototype! By default, a constructor's prototype always starts empty. Now try the following:

Object.prototype

You'll see a large number of methods defined on Object's prototype property, which are then available on objects that inherit from Object, as shown earlier.

You'll see other examples of prototype chain inheritance all over JavaScript — try looking for the methods and properties defined on the prototype of the [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), [Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date), [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), and [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) global objects

Earlier on we showed how the [Object.create()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create) method can be used to create a new object instance.

1. For example, try this in your previous example's JavaScript console:

let person2 = Object.create(person1);

1. What create() actually does is to create a new object from a specified prototype object. Here, person2 is being created using person1 as a prototype object. You can check this by entering the following in the console:

person2.\_\_proto\_\_

This will return the person1 object.

person1.constructor

person2.constructor

These should both return the Person() constructor, as it contains the original definition of these instances.

## Modifying prototypes

Person.prototype.farewell = function() {

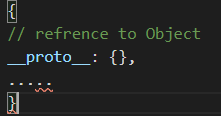
alert(this.name.first + ' has left the building. Bye for now!');

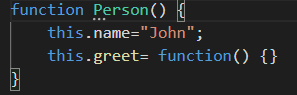
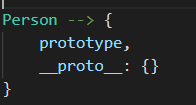
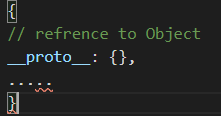
};

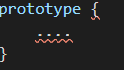
person1.farewell();

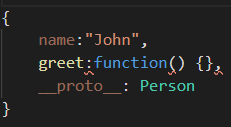
**You will see a alert,**

This new method will be available on all instance derived from person.

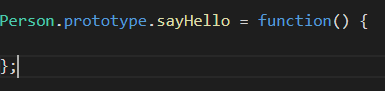
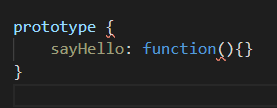
 



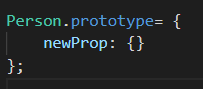
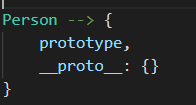
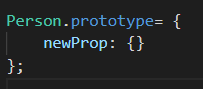
 

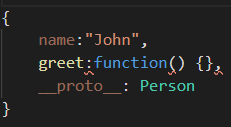
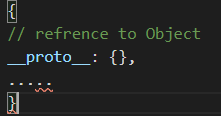
Updated to



When we de reference the Person prototype object



When we dereferenced the prototype of function is now pointing to new prototype object and for new instances \_\_proto\_\_ property will point to new prototype (object).

But for previously created objects they are still pointing to old prototype Object they will have access to previous version of prototype