* Prototypes are like classes , they reveal the parent of an object
* Prototypes are like characteristics that every object possesses
* Do not use arrow functions (anonymous) inside of an object definition while prototyping, because it does not have its own ‘this’.
* Always use normal functions inside of object definition while prototyping
* Classes are somewhat like prototypes.
* You can add properties and methods to prototypes just like myFather.prototype.nationaliy= “English”, if this property does not exist on the prototype , it will create it for the myFather object only.

We just thoroughly explored classes and prototypes in JavaScript. Many of the concepts we discussed span even beyond the realm of ES6 and JS, so these lessons will translate to most programming languages you come across.

We covered a lot of ground though. It may feel nice to take an optional coding break. Grab some coffee, tea, or whatever you need - the usual!

Ok, let’s review the important topics we went over:

* Classes in JavaScript construct structures of data based off of the state and behavior of real world objects and introduce a system of inheritance.
* The **constructor**keyword initializes an object for a class.
* The **extends**keyword creates subclasses and children of parent classes.
* Static methods in classes can be called even outside the context of class.
* Object-oriented programming models objects to create programs centered around the interactions of these objects with each other. Major programming languages like C, Java, and Ruby contain heavy support for object-oriented programming.
* JavaScript is not based on object-oriented programming, but a **prototypal-inheritance model.**
* A **prototype**is a characteristic in every JavaScript object that reveals its parent and the properties that it inherits.
* All JavaScript objects contain a prototype and can trace their chain of prototypal inheritance all the way back to the base level Object prototype.
* Arrow functions don’t create their own local ‘this’ object like a normal function prototype, but instead refer to the ‘this’ tied to its outer scope.

Classes and prototypes appear everywhere in JavaScript. And every ES6 programmer needs to grasp these fundamental concepts to truly grasp how the language works. Luckily, once you understand that classes are simply prototypes, and prototypes are simply references to an object’s parent, it becomes less abstract.

* Sets , always contain unique values. On adding a same value to set, it wont throw an error but wont place it in set.
* set.has() funtion to check whether it contains something
* if you want to pass a whole array in to the set then let a=[1,2,3,4,5];
* let newSet = new Set(a);
* newSet.values() returns all values in a set
* an enhanced for loop for ( let element of newSet.values()){}
* Data Structure Maps is identical to Objects for example having the same key and value structure.
* Maps have a size property