# Prototypal inheritance

* Every function has a property called prototype, an empty object derived from *Object.prototype* where methods and properties can be added or overwritten. It can be seen as a fallback source of properties.
* When a function x creates x1, x1 will inherit the methods and properties in prototype. (In this case x can be seen as a constructor)
* Javascript is unlike Java, where it is class-inheritance based.
* Though Javascript has the *class* keyword in ES6, it is still prototypal inheritance.
* All instances(objects) of a class should share the same prototypes.

To create an instance of a given class, the object has to be derived from the correct prototype.

*protoRabbit* contains a list of methods and properties that all rabbits share. With that, we can make a constructor

function makeRabbit(type) {

let rabbit = Object.create(protoRabbit);

rabbit.type = type;

return rabbit;

}

|  |  |
| --- | --- |
| Before 2015 | After 2015 |
| function makeRabbit(type) {  let rabbit = Object.create(protoRabbit);  rabbit.type = type;  } | class Rabbit {  //constructor  constructor(type) {  this.type = type;  }  //methods shared by all instances/objects  speak(line) {  console.log(‘The **$(**type**)** rabbit says **‘$(**line**)’’**)  }  } |

Every function expression is essentially a constructor in Javascript.

Below shows a simple example of how constructor is created with a function, and how objects can be instantiated from it:

var x = function(**j**) {

this.i = 0;

this.**j** = **j**;

this.getJ = function() {

return this.j;

}

}

var x1 = new x(1) ;

var x2 = new x(2);

x1 and x2 are instances of x, which means they are distinct objects.

x1 and x2 inherits all the methods in x. So x is technically parent class.

Every object will have the getJ method:

this.getJ = function() {

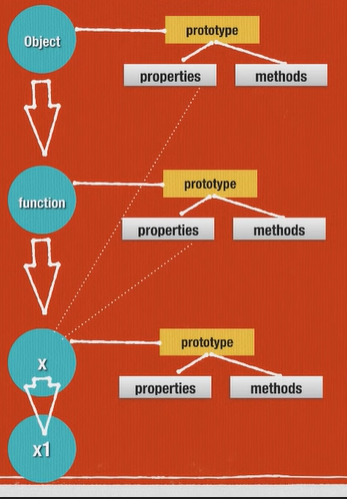
return this.j;

}

,which **takes up space** (unnecessarily) when the object is created.

A better solution will be to use “x.prototype.getJ = function() {}”. So that whenever getJ is called in x1 or x2 or x100 for that matter, the method will found in the parent (x) rather than from the individual objects.

Object.create() can be used to create an object with a specific prototype.



Polymorphism is one of the tenets of Object Oriented Programming (OOP). It is the practice of designing objects to share behaviors and to be able to override shared behaviors with specific ones. Polymorphism takes advantage of inheritance in order to make this happen.