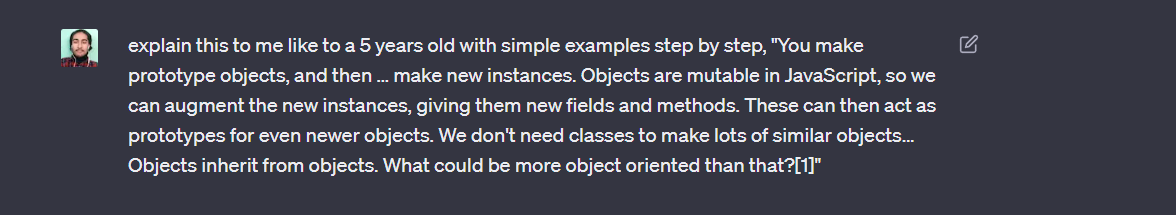
Prototype based Inheritance / Prototype based Object creation



Imagine you have a toy car. This toy car is like an object in JavaScript. It has certain properties like its color, size, and weight. It can also perform actions like moving forward and backward.

Now, let's say you want to create more toy cars that are similar to the first one. Instead of creating each car from scratch, you can make a prototype object. Think of the prototype object as a blueprint or a template for creating new cars.

So, you make a prototype object for the toy car, which has all the basic properties and actions. This prototype object can be thought of as a model for creating new cars.

Now, using that prototype, you can make new instances or copies of the toy car. Each new instance is like a brand new toy car that looks and behaves just like the prototype, but you can also customize it.

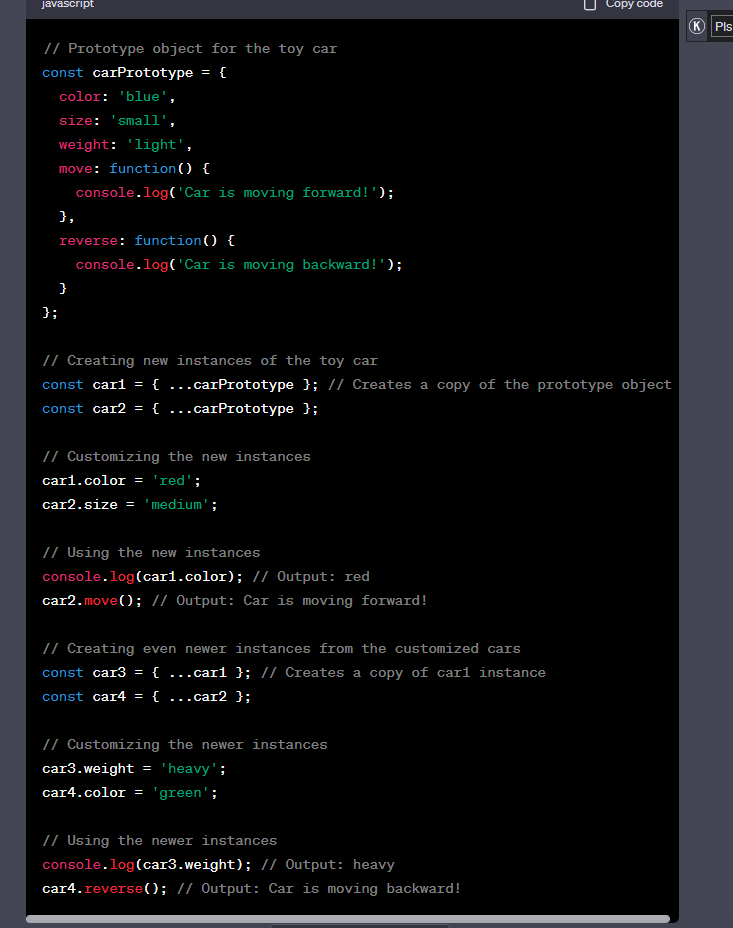
For example, you can take one instance of the toy car and paint it a different color or add stickers to it. This doesn't change the prototype itself, but it adds new fields (like the color or stickers) to that specific car.

But here's the cool part: the new instances you create can also act as prototypes for even newer cars! This means you can use one of the customized cars you made as a template to create even more cars with similar characteristics.

And guess what? We don't need classes to do all of this. In JavaScript, objects can inherit properties and behaviors from other objects. This means that one object can take on the characteristics of another object and build upon them.

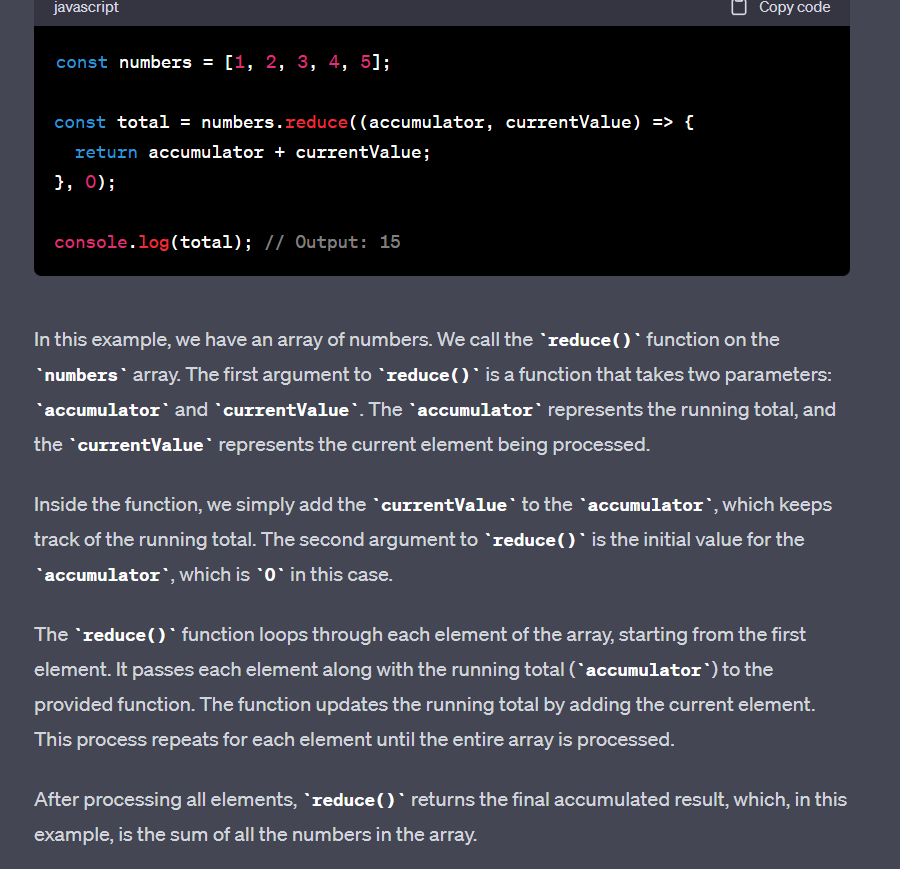
So, in our toy car example, the new instances inherit the properties and actions from the prototype object, but they can also have their own unique properties. And if you make new instances from those customized cars, the cycle continues, and you can keep creating more and more unique toy cars.

So, you see, JavaScript is a language where objects can inherit from other objects and build upon them. This makes it really flexible and object-oriented because you can create lots of similar objects without needing classes.



Remember the … spread operator always performs the shallow copy not deep copy

How Array.prototype.Reduce( ) works



If no any initial value is provided than the array[0] item is taken as the initial value for the currentValue

Arrow function do not have the arguments object

