**Prototypes**

All objects in JavaScript are linked to a prototype object.

**Prototypal Inheritance/delegation** - Prototype object contains methods and properties that all objects linked to it can access and use

Objects delegate behaviour to the linked prototype object (**why the arrow points up)**

* Objects inherit methods and properties from the prototype
* Diagram

  Description automatically generatedIn this context, this is an **instance** inheriting from a **prototype** (**not** like classic OOP, **class** inheritingfrom another **class**)
* num can use .map() method because of prototypal inheritance
* Array prototype is the **prototype of all array objects** we create in JS, so **all arrays have access to the .map() method**
* .map() is a method on the Array.prototype object
* The prototype object contains all the array methods
* Since Array.prototype is the prototype of the num array, it means that num is linked to that prototype -> has access to **all the methods that are defined on the array.prototype object**
* Array inherits the map method and therefore map method is not defined on the num array itself

**3 ways of implementing Prototypal Inheritance in JS**

**Constructor Functions:** Used since inception of JS, technique which is used to create objects from a function

**ES6 Classes:** Introduced with ES6 as an alternative to Constructor Functions. Under the hood it works exactly the same as Constructor Functions, there’s just an easier to use syntax. Does **not** behave like “classical OOP” classes.

**Object.create():** Easiest way to link an object to a prototype object

Graphical user interface, text, application, email

Description automatically generated