Prototypes & Inheritance

Understanding Prototypes

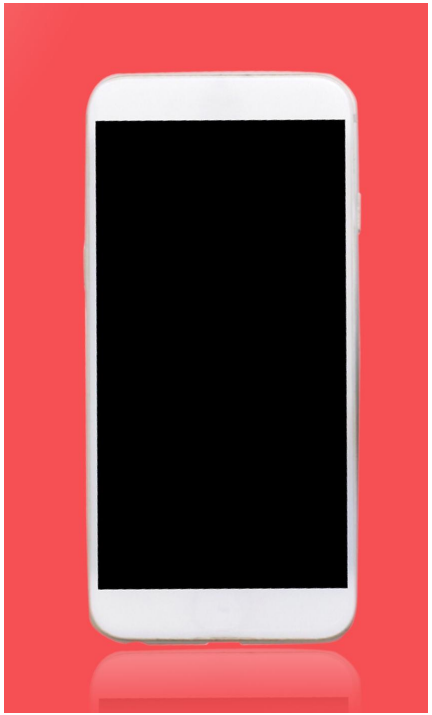
- JavaScript is Object Oriented
- Functions are very powerful in JavaScript
- JavaScript uses prototypical inheritance instead of classical inheritance

iPhone Example : Making the Prototype

Suppose you get a new iPhone. Every iPhone is manufactured the same way and come with the same features.

Make a new HTML file, add a script tag and try this:

```
function iPhone(){} // constructor

iPhone.prototype.faceId = function () {
    console.log("unlock my phone with my face!");
}

iPhone.prototype.video = function () {
    console.log("take 4k video with my iPhone");
}

iPhone.prototype.stocksApp = function () {
    console.log("find out what the stock market is doing");
}
```



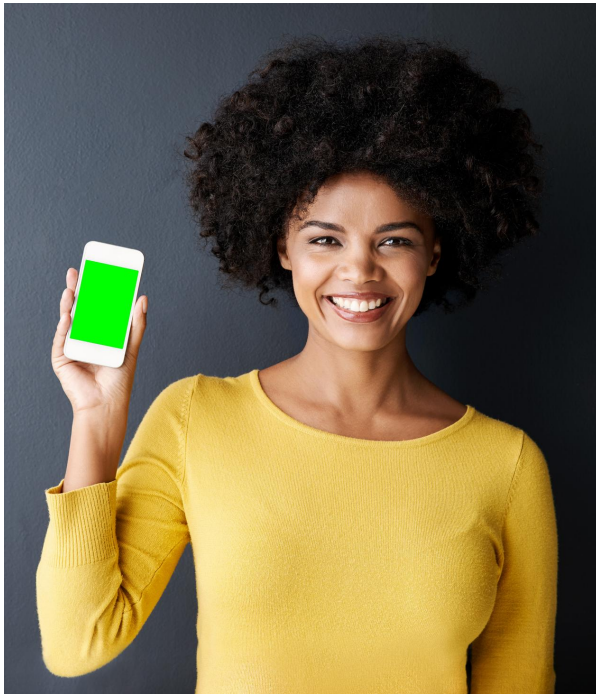
iPhone Example : Your Phone

Add this to the script:

```
let myPhone = new iPhone();

myPhone.faceId();
myPhone.video();
myPhone.stocksApp();
```

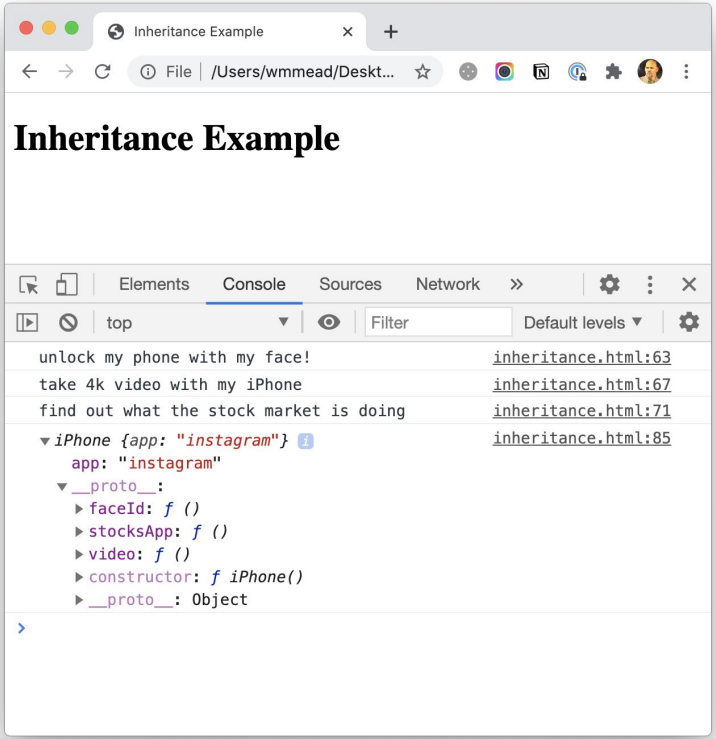
Then customize your phone by adding:

```
myPhone.app = "instagram";
```



iPhone Example : Seeing the Properties

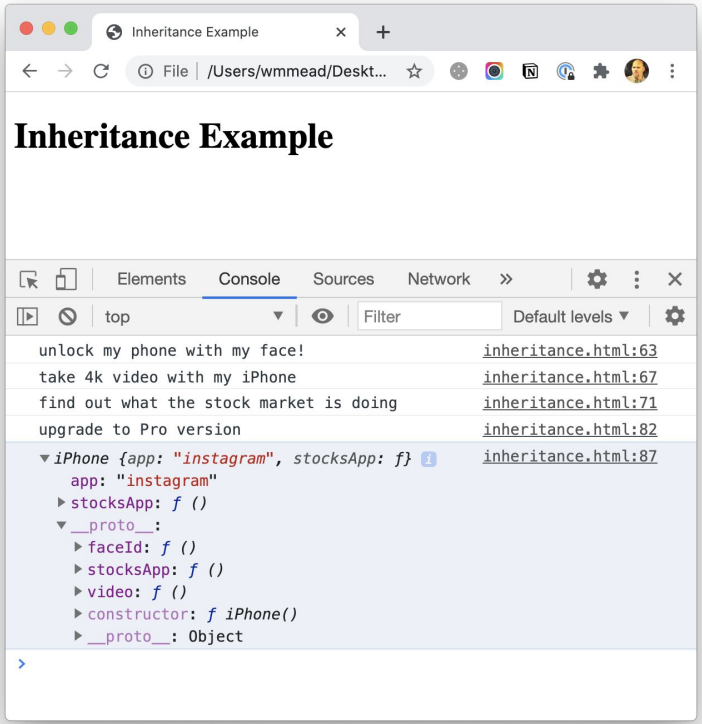
Type `console.log(myPhone);` in the console, with the page loaded.



iPhone Example : Overriding Inheritance

Suppose you have the ability to upgrade the stocks app on your phone to a pro version.

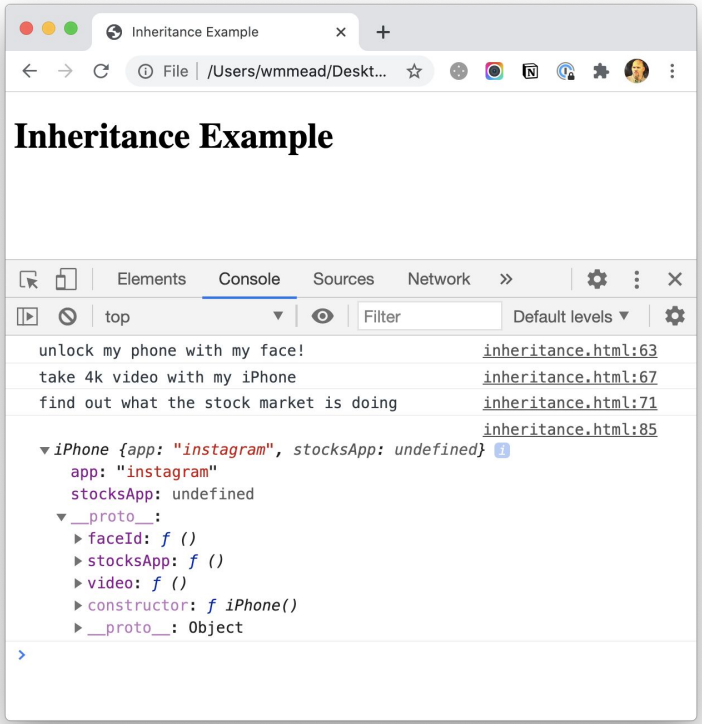
```
myPhone.stocksApp = function () {
    console.log("upgrade to Pro version");
}

myPhone.stocksApp();

console.log(myPhone);
```



iPhone Example : Deleting a Feature

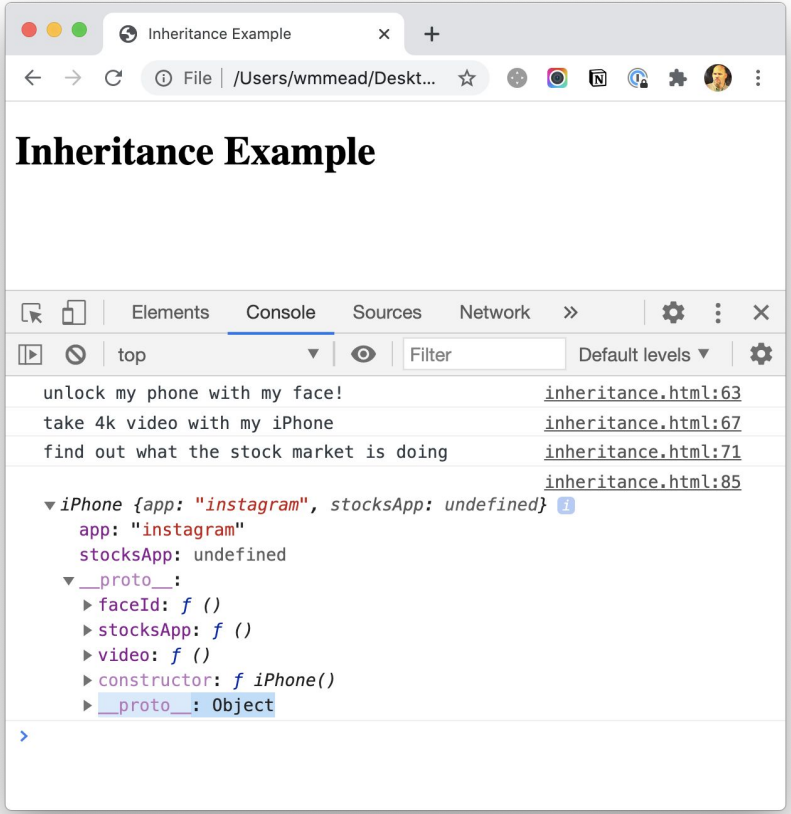
Delete the stocksApp...

```
myPhone.stocksApp = undefined;

myPhone.stocksApp;

console.log(myPhone);
```



What is __proto__?

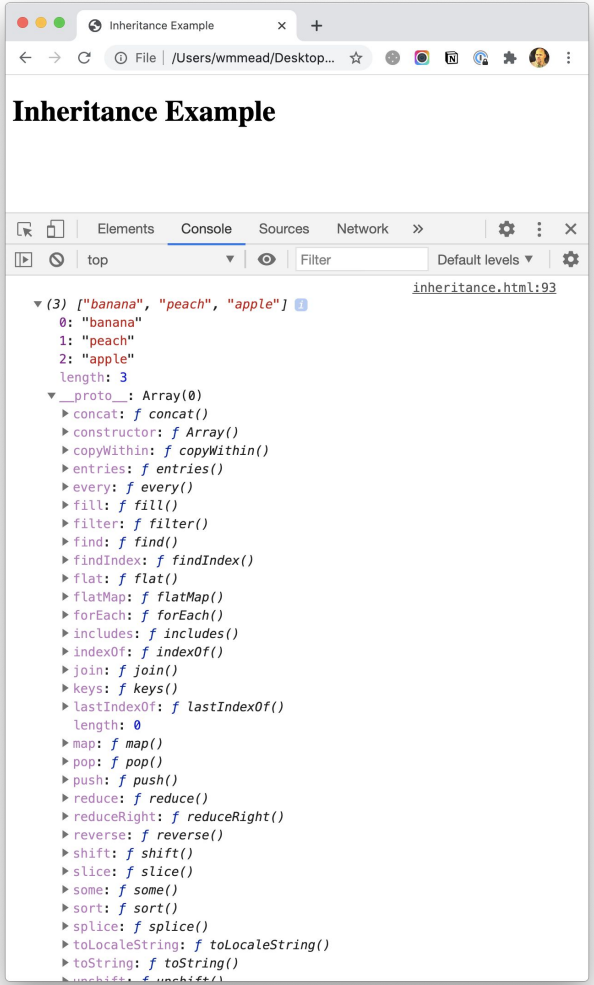
It is essentially this instance's pointer to the object it inherited from.



A Closer Look at an Array

Add this to the script, so you can see more about how prototypical inheritance works in JavaScript.

```
const myArray = ["banana", "peach", "apple"];

console.log(myArray);
```



Another Inheritance Example

```
const carManufacturer = {
    name: "Honda",
    model: "Accord",
    trimLevel: "ex",
    baseFeatures: ["wheels", "engine", "seats"],
    entertainment: "crappy audio system"
}

const dealership = Object.create(carManufacturer);
```

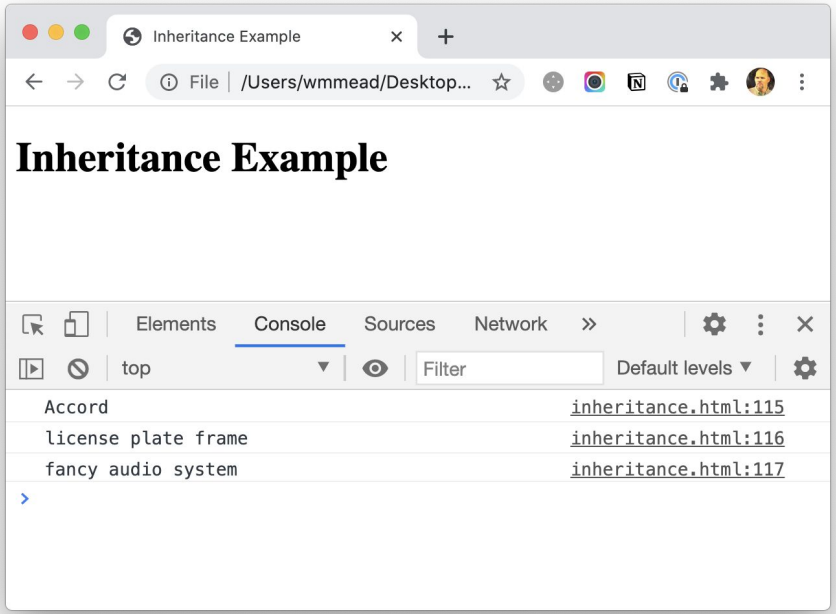
Adding to the Base

```
dealership.equipment = "roof rack";
dealership.warranty = "extended";
dealership.branding = "license plate frame";

const myCar = Object.create(dealership);

myCar.custom = "after market trailer hitch";
myCar.bumperSticker = "Save the Llamas!";
myCar.entertainment = "fancy audio system";
```

Accessing the Members of myCar

You can see that you can access members of the object from up and down the prototypical chain.

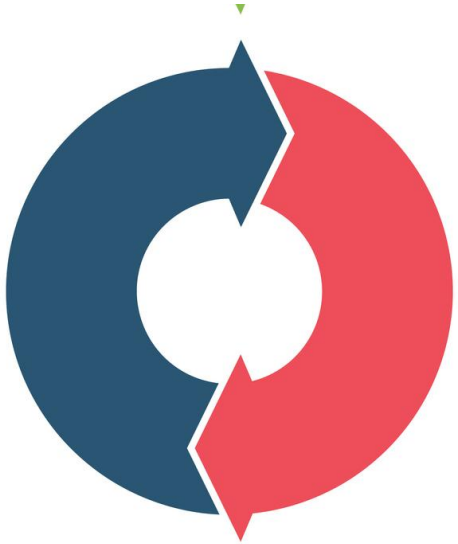
```
console.log(myCar.model);
console.log(myCar.branding);
console.log(myCar.entertainment);
```



The JavaScript For ... In Loop

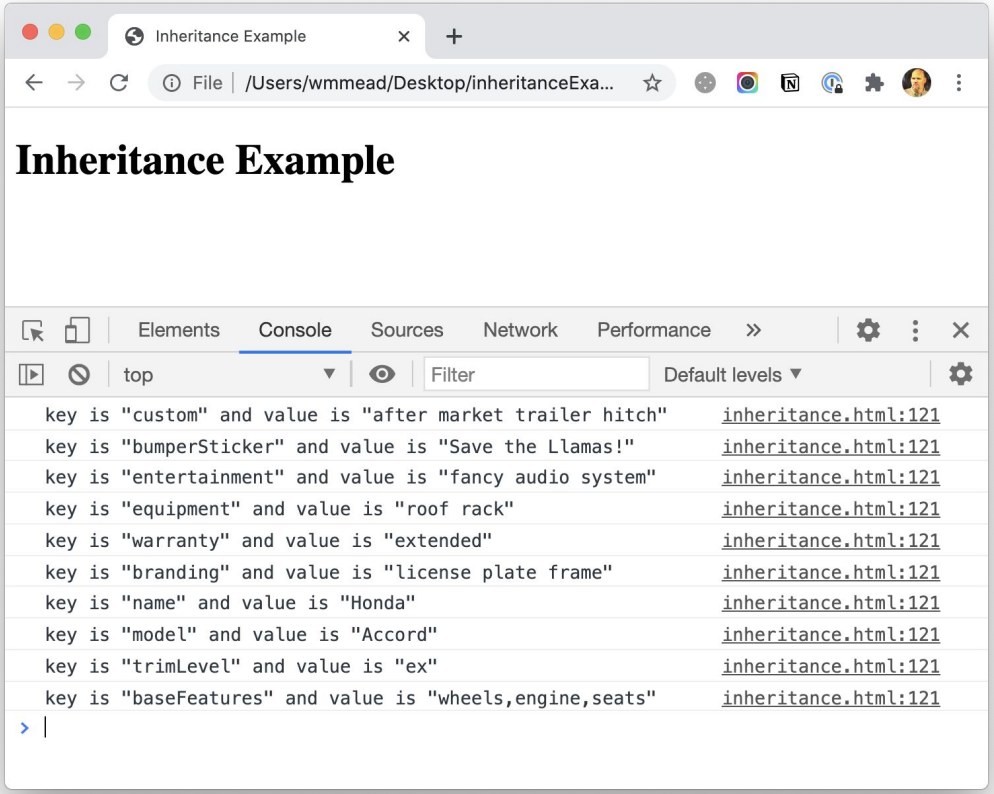
ES2015 (ES6) added a handy for ... in loop that is for iterating over objects. It looks like this:

```
for (const key in myCar) {
    const val = myCar[key];
    console.log(`key is "${key}" and value is "${val}"`);
}
```



For ... In Results

Here are the results...



Object Method: hasOwnProperty()

The object method, hasOwnProperty() can be used to filter out any unwanted properties that might come down the prototype chain.

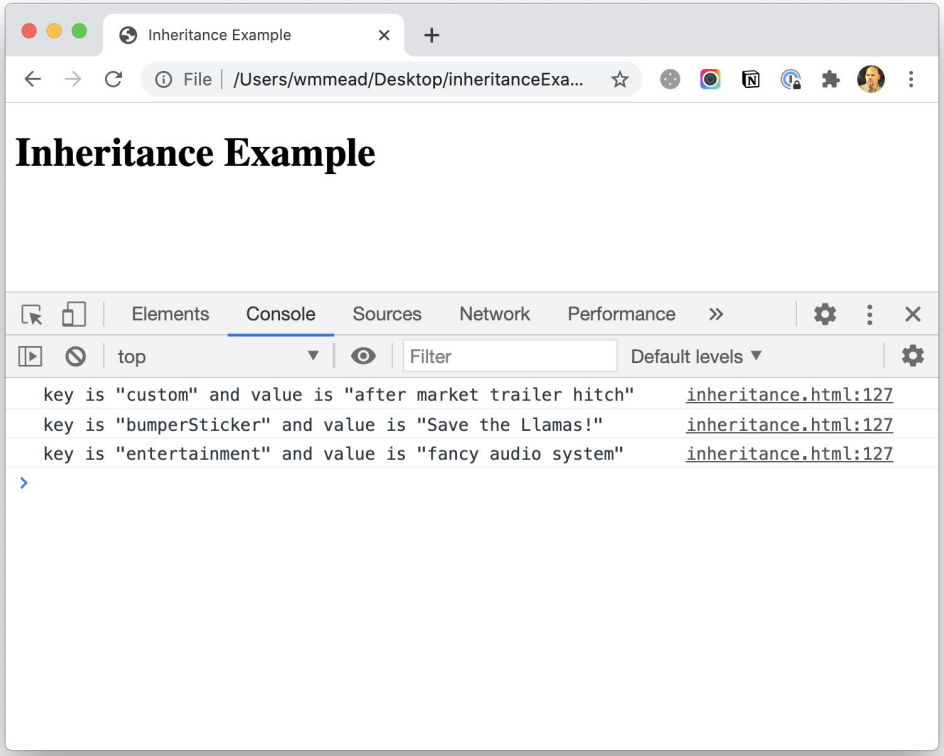
It is considered best practice to use this:

```
for (const key in myCar) {
    if (myCar.hasOwnProperty(key)) {
        const val = myCar[key];
        console.log(`key is "${key}" and value is "${val}"`);
    }
}
```

The results are shown on the next slide.

Final Result

Here is the result with the hasOwnProperty if statement filtering out anything inherited.



Summary

There is a lot to learn about prototypes and inheritance in JavaScript. It's different from other languages, but if you can get your head around it, it is a very powerful feature of this particular language.