**JavaScript Factory Functions with ES6+**

A factory function is any function which is not a class or constructor that returns a (presumably new) object. In JavaScript, any function can return an object. When it does so without the new keyword, it’s a factory function.

Factory functions have always been attractive in JavaScript because they offer the ability to easily produce object instances without diving into the complexities of classes and the new keyword.

For Example:

let person = {firstName:"John", lastName:"Doe", age:50};

console.log(person.firstName) // John

We can do the same like this

const firstName = "John";

const lastName = "Doe";

const age = "50";

var person = { firstName, lastName, age};

console.log(person) // {firstName:"John", lastName:"Doe", age:50}

**Prototypes**

const food = {

init: function(type){

this.type = type;

},

eat: function(){

console.log('You ate '+ this.type);

}

}

food.init('Waffle');

food.eat();

If we want to create this type of more objects we need to use Object.create() method for creating new objects from the existing one.

//food.init('Waffle');

//food.eat();

const waffle = Object.create(food);

waffle.init('Waffle');

waffle.eat();

const carrot = Object.create(food);

carrot.init('Carrot');

carrot.eat();

In the above example, I have seen that we have created two new objects instead of calling directly the method of the object. We also learn that how to create new objects from the existing one.

One thing you need to remember that is when we create a new object using new keyword actually it does not create a copy of the object. It just creates an empty object and inherits their properties and methods as prototype. When we call a method it first looks into its own method if not found then it looks into its prototype. In our case “food” object is called a prototype object.

const food = {

init: function(type){

this.type = type;

},

eat: function(){

console.log('You ate '+ this.type);

}

}

const carrot = Object.create(food);

const waffle = Object.create(food);

food.eat = function(){

console.log("YOU TOTALY ATE THE " + this.type.toUpperCase())};

waffle.init('Waffle');

waffle.eat();

carrot.init('Carrot');

carrot.eat();

Output:

YOU TOTALY ATE THE WAFFLE

YOU TOTALY ATE THE CARROT

From the above code, we can see that food.eat() method signature is effecting on carrot and waffle objects although we have created first before assigning the signature of the food.eat() method.

**Prototype Inheritance**

Understanding Prototypal inheritance in JavaScript.

\_ What is constructor in javaScript.

- How to add properties to prototype.

- How to create classes

\_ ECMA6, ECMA2015

- The root object in javascript

In Javascript, by default every function has a property called prototype and this property object is by default empty so we can add properties and methods in the prototype property of the function. When we create an object from the function the object inherits the prototype.

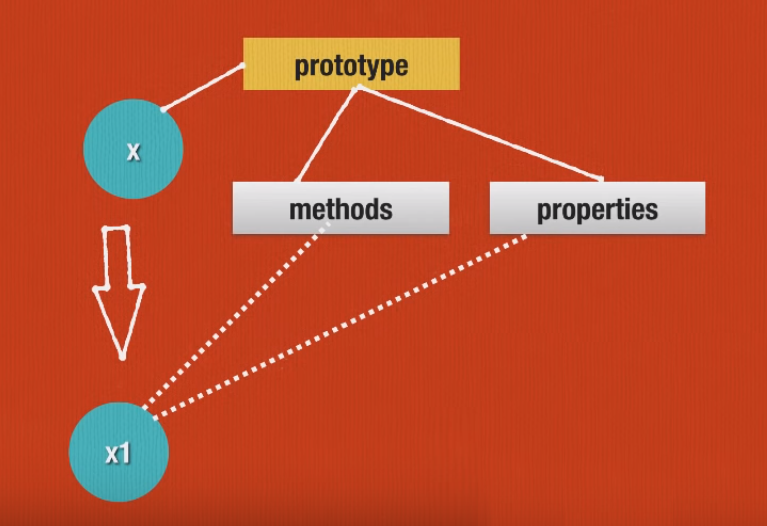
For example:

let Person = function (){

//

}

Person.prototype.propertyName;



**Constructor**:

Normally OOP Class and Javascript Class are not same actually the class in javascript is a constructor and it has prototype based inheritance. Every function expression is a constructor in javascript but we should use capital letter for separating constructor from the normal function.

For example:

let Person = function(age)

{

this.firstName="John";

this. lastName="Doe";

this. age=age;

this.fullName = function(){

return this.firstName +” ”+ this.lastName;

}

}

let p1 = new Person(50);

let p2 = new Person(40);

console.log(p1.fullName);

In the above constructor we can set prototype property like this later when an object will be created that will be inherited.

For example 02:

let Person = function(age)

{

this.firstName="John";

this. lastName="Doe";

this. age=age;

}

Person.prototype.fullName = function(){

return this.firstName +” ”+ this.lastName;

}

let p1 = new Person(50);

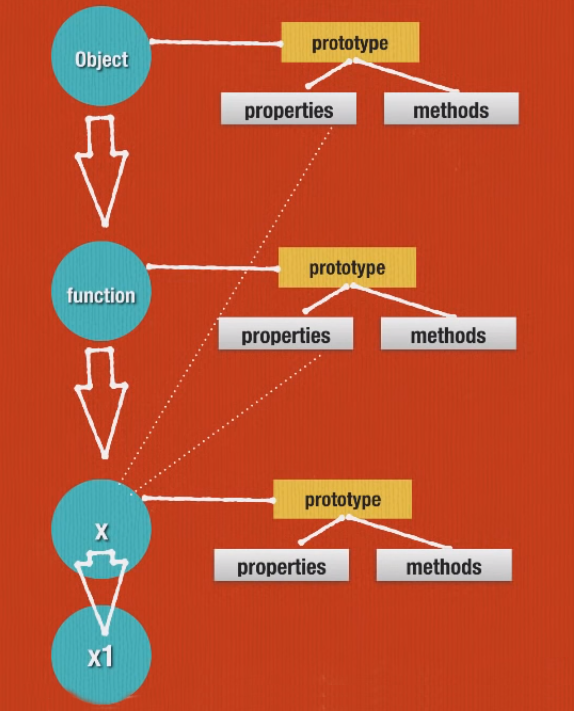
let p2 = new Person(40);

console.log(p1.fullName);

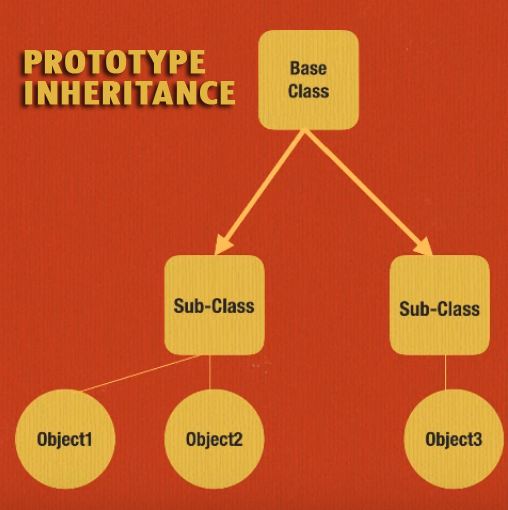
**Prototype Chain:**

console.dir(Person);

Then it will show details about prototype chain.



If we create an object from x then all of the properties and methods from the prototype chain will be inherited to that object like this

New topics to learn:

1. Creating Sub Class (Sub Constructor)
2. Overriding in Prototype Chain
3. Adding prototype to Master Object

**Base Class:**

//Base class constructor

let Job = function ()

{

this.pays = true;

}

// Job Class’s Prototype Method

Job.prototype.print = function()

{

console.log(this.pays ? 'Please hire me' : 'no thank you');

}

// subclass constructor

let TechJob = function(title,pays)

{

Job.call(this);

this.titile = title;

this.pays = pays;

}

// inheritance

TechJob.prototype = Object.create(Job.prototype);

TechJob.prototype.constructor = TechJob;

// method overriding

Job.prototype.print = function()

{

console.log(this.pays ? 'Great Job. Please hire me' : 'No thank you');

}

// Adding Method to Master Object’s Prototype

Object.prototype.print = function()

{

console.log("Hi, From Master");

}

var softwarePosition = new TechJob('Javascript Programmer', true);

var softwarePosition2 = new TechJob('PHP Programmer', false);

console.log(softwarePosition.print());

console.log(softwarePosition2.print());

**N.B**, "Object" is called the master or root object. All objects in Javascript are created from "Object". The Object's prototype methods and properties are available to all objects.

**Summary**:

When we are calling the print method using softwarePosition object if it is the own method of softwarePosition then ok otherwise it will look for its prototype if not found it will look for its parent class or constructor if not found it will search for master object if not found the method then javascript interpreter will throw an error message.