91)OOP1:Factory Functions

See code 1.js. we have 2 elfs, we call functions on them.

Whats the benefit of what we have done? We moved towards OOP. We have something called encapsulation. We have grouped functionality togather. we have state and behaviour into these tiny containers called objects. Methods can modify or read data.

Whats the problem with this code. If we have same elf, we have duplicate same code again and again. We are not following DRY.

Lets go to step 2- factory function(2.js)-

We have factory function, that creates object for us. We have avoided repetitive code, we moved a step closer to OOP,but there is still a problem here. we are storing functions separately for each object, we can store them at a place and share them across all objects.

We will use prototypal inheritance to do this.

92. OOP2: Object.create()

Before we move to prototypal inheritance there is one way of doing this manually.

See 3.js.

But still there is lot of manual work. Lets improve it a bit, by using Object.create. code snippet-4.js.

This is better. what we are doing with Object.create is true prototypal inheritance it is meant to be used like this. however you wnt see this out in most code bases. I would’nt say this is very standard or accepted by js community as whole. What we do here is not necessarily object oriented programing yet and you will see why. Next we will see what programmers did before we had Object.create.

93. OOP3: Constructor Function

Before Object.create, we used to call function in constructor mode. code sinnpet-5.js.

Lot of people dnt like new keyword. Any function that is invoked using new is called constructor function. Function,Object, Array , they are all constructor fucntions.

When we do let arr = [1,2,3];, this variable is created by calling Array in constructor mode.

Lets see how to create elf using constructor function that comes natively in js. it is Function-

const Elf1 = new Function('name', 'weapon',

`this.name = name;

this.weapon = weapon`);

const sarah = new Elf1('Sarah', 'Fireworks');

console.log(sarah);

lets add function in this constructor approach. See cpde snippet- 6.js

94. More Constructor Function

Code snippet- 7.js

function Elf(name, weapon) {

this.name = name;

this.weapon = weapon;

}

const peter = new Elf('Peter', 'stones');

console.log(peter);

console.log(peter.\_\_proto\_\_);

output-

Elf {}

Cannot understand meaning of elf.

every object has prototype property but only constructor has use of it.

Common gotca with constrcurot fucntions-

See code snippet 8.js.

Solutions are call, bind , apply an store refrence of this.

So now re we at OOP programming nirvana(in code snippet 6)? Answer is no, bcoz prototype is kind of weird, hard to understand and ugly. For people not good at js , this can get very confusing. So peole dnt like this style. Althrough, in older code bases you can still this style.

Bt the problem is OOp is all about idea of classes and there is no classes there, in this code there are no classes, we have prototype thing, we have to use capital thing. That is why **Object.create** was added to js, to avoid all this headache and use pure prototypal inheritance.

But oop languages like java use new and this a lot. So this style of coding is closer to OOP. So Object.create is technical less OO than something like this. But prototype is ugly. So will improve it a lot. In next lectures we will add classes and go to s close a possible to OOP in js.

95)Funny thing about js

In js everything is a object, everything has a constructor function for it.

With the exception of null and undefined , we have constructor functions for everything, so that we have methods that we can use.

96)OOP4: ES Classes

So nobody liked using prototype. So with ES6 class keyword was introduced. class is OOP.

See code snippet 9.js. here even through we are adding methods inside the class, they are shared across all objects. So now things are simple for us as compared to using calling function in constructor mode.

9.js also looks familiar to people coming for c++ and java. This is beauty of OOP, we have one location where entire elf object is, when we want to update something, we can go here and update it.

We finally have classes, we finally have OOP in js, right? not really. All this is syntactic sugar, under the hood in js we are still using the prototypal inheritance. We are not using classes like class work in other languages. So under hood, it is still using new keyword with prototype.

Then in lecture it is explaned why we dnt introduced classes in js from beginning. It was for marketing reasons.in other languages classes are real thing but in js they are just objects.

In interview if they ask does js have classes, answer-yes it do. But it is syntactical sugar, class keyword is still just prototypal inheritance.Some people class this pseudo classical inheritance because it is not real inheritance.

See lecture.

One question is why we are not adding the function inside constructor? This is bcoz every time we us enew keyword and create or instantiate a class, constructor function is called, this is because variable need separate space for each object. adding function inside constructor would mean that for each instance, separate function is created.

97)Object.create() vs Claass

Some people love classes some hate it. What we doe with classes can be done with Object.create function , some would say, without creating such mess. We are able to create these prototypal chains without pretending that we have classes. some people call using Object.create **pure prototypal inheritance**. But most of the new code bases use class syntax.

99)Inheritnce

The core aspect of OOP is inheritance. See code snippet 10.js**, we have cloned ogre, but ogre does not have elf as base class**. Output-

**Elf { name: 'Fiona', weapon: 'stones' }**

**Elf {}**

**{ name: 'Fiona', weapon: 'stones' }**

**{}**

So we have prototypal inheritance chain.**if we try to call function on cloned object, we will get error.** So this is our problem, how can we extend Elf. This is where inheritance comes into picture. This used to very difficult before classes. We had ugly code with lot of prototype code. But now this is lot cleaner.

Lets see inheritance, code snippet-11.js. this is called subclasing in OOp. We have base class(character) and then a subclass(elf).

Now lets elf have a extra property)its own property). See code snippet- 12.js. if we have constructor in subclass, we need to call constructor of baseclass using super keyword. This super keyword is from OOP languages. We have to call super, of we are setiing any property in constructor of chid class using this keyword. Even if we use this keyword in constructor of child class before we call super, we will get eror.

class Elf extends Character {

constructor(name , weapon, type) {

console.log(this);

super(name,weapon);

this.type = type;

}

}

if we try to acess to some method or property on object of elf that does not exist o elf, it will look for that property in Character class.

Let add a new base class that also has some methods. See code snippet 13.js. so we have added base classes, ne have its new properties, one has it’s new method.

100)Inheritnce-2

In code snippet 13.js, we can see that new method of Ogre class is added to prototype objet of ogre function. you can even call it using that. But when we pirint prototype object, it is empty. So we cnt see fucntions there, but they are present there and we can call them.

In code snippet 14.js we checked the prototype chains and they are what we expected. Now there is easy way of checking these connections using **instanceOf**  keyword.

Now dolby is instance of both Elf and Character class. Bcoz character is super class of Elf. So in inheritance we are not copying and making things inefficient., we are using prototype chain, so that things are shared. We have objects inheriting from object, we have no classes.

In java and c++ we have actual classes. languages like java and c++ copy objects when do sometging like extend(althrough methods are shared). But in js we link and objects are refrenced. So there is bit of efficiency there.

101)Pubic vs private

In oop we have public and private variables. In js we do not have it. Before people used to add \_ in front of variable names to tell programs to not to change this variable. But programmeers can still change it. Code- 15.js. other languages have private and public.

In js we can solve it with weak maps, bt it is little hacky and we dnt want to get into it.

As of ow we do not have public and private in js.it is in proposal.

102)OOP in react

103) 4 pillars of OOP

Encapsulation, abstraction,inheritance, polymorphism.