JS OOP Summery   
---------------------------------

بسم الله الرحمن الرحيم أول حاجه ده شرح لل OOP الخاص بال JS الى هو يعنى الجافا سكريبت ال Native من الاخر و هنشرح فيها من بدايه ما اشتغلوا فى اللغه و التحديثات الى حصلت فى ال ECMA Script

#######################################################################  
اذاى الجافا سكريبت بنعمل فيها اوبجيكت

----------------------------

أول حاجه الاوبجيكت ذى ما انت عارف ده اهم حاجه فى الجافا سكريبت و ده لان اللغه بتعتمد بشكل اساسى على الاوبجيكت ياباشا و ده لانه يعتبر فاريبول بيعبر عن مجموعه من الفاريبولس على شكل Key و value و بنقدر نشيل فيه فاريبولز و كمان ميثودز و ادينى هنعمل أوبجيكت اهه و نشوفه كمان

var  obj = {

    Name:"Mahmoud" ,

    Age : 25,

    addres : "Cairo"

}

console.log(obj);  //  {Name: "Mahmoud", Age: 25, addres: "Cairo"}

#######################################################################  
استخدام الاوبجيكت انى اعمل 100 نوع عربيه مثلا

----------------------------

var  Car1 = {

    Name:"Lancer" ,

    Model : 1991,

    color : "red"

}

console.log(Car1);

var  Car2 = {

    Name:"Marcedes" ,

    Model : 2000,

    color : "Black"

}

console.log(Car2);

انا كده هعمل 100 أوبجيكت عشان اعمل 100 عربيه لو همشى بالطريقه دى

فقالك لا ده انت تعمل فانكشن ياعم يبقى مهمه الفانكشن دى انك تديها اسم و نوع و لون العربيه تروح عملالك الاوبجكيت بتعها

#######################################################################  
عمل الاوبجيكتات بالفانكشن طريقه ال factory

----------------------------

كل الى بعمله انى بعمل فانكشن بتعمل ريتيرن ل obj و بملى الاوبجيكت جواها و أرجعه فى نتيجه الفانكشن

function CreateCar(Name, Model, color) {

    return {

        Name: Name,

        Model: Model,

        color: color

    }

}

const MercedesCar = CreateCar("Mercedes", 2000, "Black")

console.log(MercedesCar);

لاحظ انى و انا بعمل ال call للفانكشن بعت البراميتار و خزنت نتيجتها فى فاريبول و بعد كده قرأت النتيجه فى الكونسول (الطريقه دى بيسموها factory Design Pattern)

طبعا اصبح الطريقه دى انك تقدر تعمل مليون call للفانكشن و فى كل مره هيروح يعملك أوبجيكت من أول و جديد داخل الفانكشن (طبعا بالرغم من حلاوه الموضوع و انه خلانى معملش الاوبجيكت 100 مره الا انه فشخ الميمورى ) لان كل مره هنادى فيها الفانكشن بأرجيومنتس جديده او حتى هى هى هيروح يعملى اوبجيكت جديد خالص غير الى قبله و بالتالى ده هيفرتك البرفورمنس برضوا

#######################################################################  
عمل الاوبجيكتات بالفانكشن و جواه فانكشنز و نشوف هتشتغل اذاى

----------------------------

هنعملهم كانهم بروبرتى عاديه بس القيمه بتعتهم هى الفانكشن بقه و هتناديهم عن طريق الكونست الى انت خزنت فيه الاوبجيكت لان هو بيبقى مخزن كل الاوبجيكت جواه و بالتالى هتلاقى فيه البروبيرتيس و الميثودز

function CreateCar(Name, Model, color) {

    return {

        Name: Name,

        Model: Model,

        color: color,

        start: function () {

            console.log("start Engine")

        },

        Stop: () => {

            console.log("Stop Engine")

        }

    }

}

const MercedesCar = CreateCar("Mercedes", 2000, "Black")

console.log(MercedesCar);  // {Name: "Mercedes", Model: 2000, color: "Black", start: ƒ, Stop: ƒ}

MercedesCar.start(); //start Engine

MercedesCar.Stop(); //Stop Engine

#######################################################################  
عمل الاوبجيكتات بالفانكشن طريقه ال constractor

----------------------------

هى ذى الى فوق بس الفرق حاجتين أول حاجه ان الفانكشن دى مش هيبقى فيها ريتيرن لحاجه لأ دى البراميتار انا بخزنها فى البروبيرتى الخاصه بالفانكشن بشكل مباشر و لما باجى انادي الفانكشن الكونستراكتور دى عشان افهمه ان البروبارتيس الى جواها دى بتعبر عن الفانكشن الكنستراكتور باستخدم قبلها الكى ورد new

طيب تعالى نعمل الكلام ده و مش هنحط new و هنشوف ايه الى هيحصل و ليه ؟؟

function Car(Name, Model, color) {

        this.Name= Name;

        this.Model= Model;

        this.color= color;

        console.log(this);  // window Object

}

const MercedesCar = Car("Mercedes", 2000, "Black");

console.log(MercedesCar);  // undefined as you do not use new Key word before func

console.log(window.Name , window.Model,window.color);  // "Mercedes", 2000, "Black"

الى حصل ان احنا عشان مستخدمناش كلمه new و احنا بنعمل call للفانكشن Car راحت اعتبرت ان this الى جواها دى هى ال window Obj و بالتالى لما جينا نشغل الاوبجيكت بتاع ال MercedesCar مجبش حاجه لان متسجلش فيه حاجه اصلا

طيب تعالى بقه نستخدم كلمه new قبل ال call بتاع الفانكشن هتلاقى ان this اصبحت بتعبر عن الكوستراكتور الى اسمه Car كانه عمل instance منه و خزنه فى ال this و ابتدى ان اى بروبرتى بتكتبها بعد ال this بيخزنها جواه و يعرضها لما اجى اتعامل معاه

function Car(Name, Model, color) {

        this.Name= Name;

        this.Model= Model;

        this.color= color;

        console.log(this);  // Car Object {Name: "Mercedes", Model: 2000, color: "Black"}

}

const MercedesCar = **new** Car("Mercedes", 2000, "Black");

console.log(MercedesCar);  // Car Object {Name: "Mercedes", Model: 2000, color: "Black"}

console.log(window.Name , window.Model,window.color);  // undefined undefined undefined  as they did not saved in wondow Object

الفرق بقه بين طريقه الفاكتورى و الكنستراكتور ؟؟

الفرق كبير جدا

الفاكتورى انت كل مره بتنادى فيها الفانكشن بتروخ تكريت أوبجيكت جديد خالص و ده بيقتل الميمورى

بينما فى الكنستراكتور هتلاقى انه بيعمل كنستراكتور واحد و بيكريت منه الاوبجيكت و بعد كده كل اما هتناديله مش هيرجع يكريته من جديد لأ ده هيعمل منه instances تقدر تستخدمها عادى من غير ما تفشخ الميمورى و عشان كده لاحظ ان الى رجع من الفاكتورى الاوبجيكت بالشكل ده

{Name: "Mercedes", Model: 2000, color: "Black }

بينما الى رجع من طريقه الكونستراكتور هتلاحظ انه رجع أوبجيكت من نوع Car و كأن فانكشن الكونستراكتور دى كلاس بالظبط

**Car** {Name: "Mercedes", Model: 2000, color: "Black"}

#######################################################################  
عمل الاوبجيكتات بالفانكشن طريقه ال constractor و جواها فانكشنز

----------------------------

function Car(Name, Model, color) {

        this.Name= Name;

        this.Model= Model;

        this.color= color;

        this.start = ()=>{

            console.log(this);

            console.log("start Engine");

        }

}

const MercedesCar = new Car("Mercedes", 2000, "Black");

console.log(MercedesCar); // Car {Name: "Mercedes", Model: 2000, color: "Black", start: ƒ}

MercedesCar.start()

// Car {Name: "Mercedes", Model: 2000, color: "Black", start: ƒ}

//start Engine

const VernaCar = new Car("Verna", 2010, "Red");

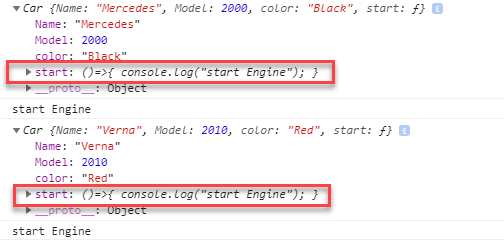
console.log(VernaCar);  // Car {Name: "Verna", Model: 2010, color: "Red", start: ƒ}

VernaCar.start()

// Car {Name: "Verna", Model: 2010, color: "Red", start: ƒ}

//start Engine

هتلاحظ ان كل اما بتنادى للاوبجيكت بيروح يعمل انستانس و داخل كل انستانس جواه بينشأ فانكشنز و لو الفانكشنز دى فيها بروسيسنج و حوارات كده انا بقه هفرتك الميمورى برضوا



كده انا عاوز حاجه تخبى الفانكشن دى خالص و ميتمش استخدمها الا وقت ما احتجها بس اعمل ايه ؟؟

الحاجه الى بتعمل كده و الى اللغه مستخدمها هى حاجه اسمها ال **Prototype** و هيتم ستخدمها كده يا كبير

function Car(Name, Model, color) {

        this.Name= Name;

        this.Model= Model;

        this.color= color;

}

//create  start function in Car Prototype **(Do not use Arrow with proto)**

Car.prototype.start   = function(){

    console.log("start Engine");

}

const MercedesCar = new Car("Mercedes", 2000, "Black");

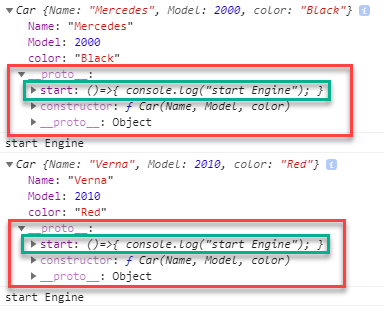
console.log(MercedesCar);

MercedesCar.start()

const VernaCar = new Car("Verna", 2010, "Red");

console.log(VernaCar);

VernaCar.start()



كده الفانكشن الى عملتها فى البروتوتايب اصبحت مش بتتكريت كل مره فى الاوبحكت لا ده بتتكريت فقط لما بناديها و طول منا منديتهاش فهى مستخبيه جوه ال prototype   
ملحوظه مهمه جدا أوعى تستخدم ال arrow function مع ال prototype و ده لان ال this هتجيلك على انها window obj و ده لان ال Arrow فانكشن بتاخد الاوبجيكت بتعها من الاسكوب الى هى فيه و بما انها فى الاسكوب بتاع ال window فهتاخد الاوبجيكت بتاع ال window

على عكس لو انت مستخدم فانكشن عاديه ال this جواها بتاخد اوبجيكت الحاجه الى بتناديها بيها فلو ناديتها بالويندوز هتاخد الويندوز و لو ناديتها بالكونستراكتور هتلاقى ان الthis بتعتها اخدت الكونستراكتور و لو ناديتها بالبتنجان هتلاقيها اخدت البتنجان

نيجى بقه لملحوظه تانيه و هى انى عشان اعرف الى جوه البروتو بعمل كونسول و اشوف الكونستراكتور معاه اكستنشن بيجيب البروتو اسمه protoype و الاوبجيكت معاه اكستنشان اسمه \_\_proto\_\_ برضوا هيعرفك ايه الى جوه البروتوتايب عشان لو محتاج تستخدمه يا اسطى :D طيب لو عاوز اشغل اى حاجه موجوده جواه بناديها عادى بالاوبجيكت كانها فانكشن عاديه خالص او بروبرتى عاديه خالص و هو بيدور عليها و يلاقيها و يجيبها

function Car(Name, Model, color) {

        this.Name= Name;

        this.Model= Model;

        this.color= color;

}

//create  start function in Car Prototype

Car.prototype.start   = function(){

    console.log(this);

    console.log("start Engine");

}

const MercedesCar = new Car("Mercedes", 2000, "Black");

console.log(Car.prototype);  // to know what is inside Proto by func

console.log(MercedesCar.\_\_proto\_\_); // to know what is inside Proto by Object

MercedesCar.start(); //  call some thing inside proto by Object

#######################################################################  
 حاجات بنستخدمها كل يوم و فيها بروتوتايب يا اسطى

----------------------------

Array

String

Boolean

Object

Function

و ادى هوريك الكونسولز بتعتهم عشان تتأكد بنفسك انهم بسيخدموا بروتوتايب كتير فشخ جواهم

console.log( String.prototype)

console.log( Number.prototype)

console.log( Array.prototype)

console.log( Object.prototype)

#######################################################################  
Inheritance in JS By Prototype

----------------------------

الوراثه فى الجافا سكريبت على الطريقه القديمه بتاعت البيور جافا سكريبت

فانكشن بتورث من فانكشن يا ولاد الهرمه و ده عن طريق استخدام ال prototype و ال Object.create

معروف اهميه البروتوتايب انها ذى الصندوق الى بخبى فيه الحاجات الى هستعمها من داخل الفانكشن بس مش عاوزها تشتغل الا لما انديها بس عشان احافظ على الميمورى

انما ايه فايده ال object.create() دى يا سيدى فانكشن مهمتها انها بحط فيها بروتوتايب الاب الى الابن هيورث منه و بروح مخزن بروتوتايب الاب كله بقه فى البروتوتايب بتاع الابن و بكده انا ورثت ل خصائض الاب فى الابن

و لان الفانكشنز بتاعت الاب هى الى انا اصلا عاوز اورثها و بالتالى كل فانكشنز الاب لازم تبقى محطوطه فى البروتوتايب لان الابن بيورث من الاب البروتوتايب بتاعه

و الكلام ده لا يلغى ان بيتم وراثه البروبرتيس العاديه و لكن الفانكشنز بقه هى الى لازم تتحط فى البروتوتايب عشان تتورث

و كان انا بقوله اعملى اوبجيكت من بروتوتايب الاب و حطهولى فى بروتوتايب الابن و بالتالى اصبح الاب جزء موجود داخل الابن خلاص

تعالى اوريك مثال للوراثه بطريقه الاوبجيكت كريت فانكشن دى

function Parent(Name) {

    this.Name= Name;

}

// adding ParentFunc to Parent Prototype

Parent.prototype.ParentFunc = function(...param){

    console.log(`Hello ${param[0]} from Function in Parent`);

}

function child(age) {

    this.age= age;

}

// inheretance from parent proto to child proto

child.prototype = Object.create(Parent.prototype);

const ParentObj = new  Parent("Mahmoud");

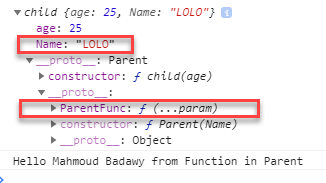
const ChildObj = new  child(25);

ChildObj.Name = "LOLO";

console.log(ChildObj)

//child use parent function now

ChildObj.ParentFunc("Mahmoud Badawy")

s

بس الطريقه الى فوق دى ناقصه و ده لان الكونستراكتور بتاع الابن اتمسح بعد ما ورث عليه و كده يبقى لازم ارجعه تانى و ده بيبقى انى بضيفه تانى للبروتوتايب بتاعه و هيبقى الكود بقه كده فى الاخر

// inheretance from parent proto to child proto

**child.prototype = Object.create(Parent.prototype);**

**child.prototype.constructor = child;**

**طيب انا عاوز مش كل مره اجى اعمل انهيرتنس اتفقع كده و اعد اكتب كتير فانا هعمل هيلبر فانكشن اكتب فيه الطريقه مره واحده و ابعتلها اسم الابن و الاب و هى تتعامل بقه فهيبقى شكل الكود كده بعد التعديل**

function Parent(Name) {

        this.Name= Name;

}

// adding ParentFunc to Parent Prototype

Parent.prototype.ParentFunc = function(...param){

    console.log(`Hello ${param[0]} from Function in Parent`);

}

function child(age) {

    this.age= age;

}

// inheretance from parent proto to child proto

function inheritance(Mychild,MyParent){

    Mychild.prototype = Object.create(MyParent.prototype);

    Mychild.prototype.constructor = Mychild;

}

inheritance(child,Parent);

const ParentObj = new  Parent("Mahmoud");

const ChildObj = new  child(25);

ChildObj.Name = "LOLO";

console.log(ChildObj)

//child use parent function now

ChildObj.ParentFunc("Mahmoud Badawy")

طيب ملحوظه مهمه جدا بقه الطريقه دى مش احسن حاجه تستخدمها و ناس كتير بتنصح ان الوراثه بالبروتوتايب فيها مشاكل كتير و نبطل نشتغل بيها احسن و نشتغل بال composite method احسن و دى هنشرحها بعدين

#######################################################################  
access Modifier in JS By Prototype

----------------------------

انت من المثال الى فات اخدت بالك اكيد ان البروبيرتيس كلها الى فى الاب Public عشان كده الابن ورثها عادى طب

function Parent(Name,Addres) {

       this.Name= Name; // Now Name became Public

        this.Addres = Addres;

}

// adding ParentFunc to Parent Prototype

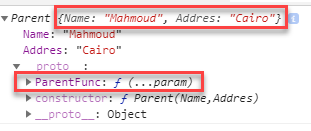
Parent.prototype.ParentFunc = function(...param){

    console.log(`Hello ${param[0]} from Function in Parent`);

}

const ParentObj = new  Parent("Mahmoud","Cairo");

console.log(ParentObj)



لو انت عاوز تعمل اى بروبرتى private عشان ميتورسش تعملها اذاى دى ؟؟

كده يا اسطى و نشرح بعد الكود

function Parent(Name,Addres) {

       let \_Name= Name; // Now Name became private

        this.Addres = Addres;

}

// adding ParentFunc to Parent Prototype

Parent.prototype.ParentFunc = function(...param){

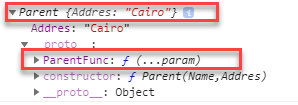
    console.log(`Hello ${param[0]} from Function in Parent`);

}

const ParentObj = new  Parent("Mahmoud","Cairo");

console.log(ParentObj)

كل الى اتعمل انى قولتله لأ متستخدمش this عشان ميضيفش البروبرتى ده للاوبجيكت خليته ينشأ ال Name بفاريبول عن طريق let و بالتالى كدهمتحزنش جوه Object فكده بقه Private و ميخرجش بره الاسكوب بتاعه خصوصا انى عامله let بص على الصوره هتلاحظ ان ال Name مش موجوده فى الاوبجيكت بتاع الاب اهه



#######################################################################  
setter and Getter in JS By Prototype

----------------------------

دلوقتى انت عملت ال Name بفاريبول و خليته كده private طب انا عاوز اقراه بره الفانكشن اقراه اذاى ؟؟

بص يا سيدى مفيش غير طريقه واحده هى الى بتعمل الحاجه دى و هى ال Setter , Getter هما عاملين ذى الكوبرى كده الى بتعدى قيمه الحاجات البرايفت بره الاسكوب بتعها و ده لانى باخد قيمه الحاجه البرايفت اسجلها فيها عن طريق ال Set وبعد كده اظهرها بره للناس الى باره بال get و ده لانهم بيبقوا معمولين انهم public

الفكره كأن بنت عاوزه تشترى حاجه بس لبسه لبس البيت و مش نافع ان حد يشوفها فهتبعت اخوها يكلم الناس و يشترى هو الحاجه دى فكده هو راح شاف الناس و الناس شافته و عمل المصلحه الى هيقضيها لاخته و رجعلها تانى اداها الحاجه الى هى عاوزه تشتريها بس كده

ملحوظه عشان تفهم ال Setter , Getter دول بيتم تلخصيهم فى كلمه واحده و هى ال Property

فانا هنا فى الجافا سكريبت هعمل من الاخر كده جوه الفانكشن بروبيرتى

تعالى اوريك جيتر و سيتر هيتكتبوا اذاى لل Name عشان يعرف يتعامل مع الحاجات الى بره الكلاس او الفانكشن او الاسكوب بتاعه هههههههههه

function Parent( Addres) {

    let \_Name =  "Mahmoud"; // Now Name became private

    this.Addres = Addres;

    Object.defineProperty(this, 'NameProp', {

        get: function () {

            return \_Name;

        }

    })

}

// adding ParentFunc to Parent Prototype

Parent.prototype.ParentFunc = function (...param) {

    console.log(`Hello ${param[0]} from Function in Parent`);

}

const ParentObj = new Parent("Cairo");

console.log(ParentObj);

ParentObj.NameProp  = "Mesho"; // Not seen as there is no Setter for It

console.log(ParentObj.NameProp)

ذى ما انت شايف ال getter ببتعمل اذاى و هو انى بقوله عارف ال Object هيقولى ايوه هقوله عرفلى فيها بروبرتى بقه داخل this الى هى تعود على الكلاس او الفانكشن الى انا جواها دى و سمى البروبرتى NameProp و هتبقى ال Get بترجع القيمه الى ال \_Name واخداها و ده لانى برجع قيمتها ف الريتيرن

قالك فى طريقه احسن من دى و More flexable و هو انى هستخدم الكونفيجريشن أوبجيكت و ده بيخلينى استخدم شويه خواص فى اخر براميتار فى الDefineProperty

function Parent( Addres) {

    this.Addres = Addres;

    Object.defineProperty(this, 'NameProp', {

        configurable : false,

        writable : false,

        enumerable:false,

        value : "Mahmoud"

    })

}

// adding ParentFunc to Parent Prototype

Parent.prototype.ParentFunc = function (...param) {

    console.log(`Hello ${param[0]} from Function in Parent`);

}

const ParentObj = new Parent("Cairo");

console.log(ParentObj);

ParentObj.NameProp  = "Mesho"; // Not seen as Prop is not Writable

console.log(ParentObj.NameProp)

ذى ما انت شايف مسحت فاريبول ال Name و خليت كل حاجه خاصه بالبروبيرتى الى اسمه Name جوه فانكشن الديفاين بروبيرتى بقوله عارف ال Object هيقولى ايوه هقوله عرفلى فيها بروبرتى بقه داخل this الى هى تعود على الكلاس او الفانكشن الى انا جواها دى و سمى البروبرتى NameProp و هىبقوا هو ده اسمه الى بناديه بيه فبترجع القيمه الى ال Value الى جوه الكونفيجريشن

شرح دول بقه

configurable : false,

        writable : false,

        enumerable:false,

        value : "Mahmoud"

Configurable : دى لو false ده بيمنع مسح البروبرتى بالطريقه دى

delete ParentObj.NameProp

و لو true هتتمسح

Writable : لو true يبقى ساعتها كانى عملتله Setter فيقدر يغير من قيمتها عادى

Value : دى القيمه المبدأيه او القيمه بشكل عام للبروبرتى دى   
enumerable : دى خاصه بعمليه ال Iteration يعنى لو انا بعمل لوب على الى جوه الاوبجيكت لو انا قايله false فالبروبرتى ده مش هيرجع ضمن الحاجات الى راجعه فى الفورلوب بيتما لو عاملها true هيبقى فى اللوب او الاتريشن على الاوبيجكت هيرجعلك البروبرتى مع حاجات الاوبجيكت عادى

طيب اذاى بعمل iteration بالفورلوب على الاوبجيكت الى فوق ده ؟؟

بص يا سيدى انت لو عاملته من غير hasOwnProperty هتلاقى اللوب مرجعلك كل الفيلدس و معاها البروبرتيس الى انت مشغلها ال enumerable و كمان هتلاقيه بيلف على الحاجات الى جوه البروتوتايب يجيبها فده طبعا عك السنين لازم ميجبش الا الحاجات الى بره البروتوتايب و عشان كده انا جوه الفور لوب المفروض اقوله مترجعش الا الى ليه بروبرتى بمعنى انى هقوله بص على كل الى راجع جواه فانكشن الهازاون بروبرتى الى تلاقيه ليه بروبرتىرجعه و الى ملوش ذى الفانكشن الى بتتعمل فى البروتوتايب يبقى مترجعهاش

function Parent( Addres) {

    this.Addres = Addres;

    Object.defineProperty(this, 'NameProp', {

        configurable : false,

        writable : false,

        enumerable:true,

        value : "Mahmoud"

    })

}

// adding ParentFunc to Parent Prototype

Parent.prototype.ParentFunc = function (...param) {

    console.log(`Hello ${param[0]} from Function in Parent`);

}

const ParentObj = new Parent("Cairo");

console.log(ParentObj);

for (let Key in ParentObj){

    if(ParentObj.hasOwnPropery(Key)){

        console.log(Key);

    }

}

جرب كده من غير فانكشن hasOwnProperty هتلاقى انه بيرجع اى حاجه فى وشه و طبعا ده مش صح

#######################################################################

Prototype inheritance VS Composition

----------------------------------------------

احنا فوق شوفنا الانهيريتانس بالبروتوتايب بس ملحظناش عيوبه

طيب ايه عيبوبه ؟؟

بص ياسيدى البروتوتايب انهيرتانس ده لو فى ريليشنز كتير فى الوراثه يبقى مينفعش خالص و ده لان مثلا بفرض انا عندى صفات فى ال Parent عاوز اديها ل child و مديهاش ل Child تانى فى الحاله دى مش هنعرف ننفذه بالانهيرتانس و ادى مثال يا اسطى انا عندى كلاس اسمه Animal فيه ال Name و فيه بقه 3 فانكشنز

Eat , Drink , Run

عندى بقه كلاس جديد اسمه Dog عاوزه يورث كل الفانكشنز الى هما Eat,Drink,Run فده عادى خالص مش هيقول لأ و هيأدى

بس انا عندى كلاس تانى اسمه Cat انا عاوزه يورث من كلاس ال animal 2 فانكشنز بس الى هما eat , drink

فى حاله ال prototype inheritance الدنيا كلاس القطه هتورث كله غصبا عنها و عشان احل المشكله دى هضطر اعمل كلاس اسمه AnimalWithoutRun و احط فيه الاتنين فانكشن بتوع eat , Drink و اروح اورث منه و هكذا بقه لو عاوز امنع الاكل اروح اعمل كلاس animalWithoutEat و هكذا بقه قمه البيض يعنى

فقالك بقه لأ فاكس من ال prototype inheritance لو بتفكر تورث حاجات لكلاس و حاجات لأ و نخلى الدنيا فليكسبول خالص عن طريق حاجه جديده اسمها   
Composition  
دى سا سيدى انت بتعمل behaivers يعنى مثلا انت دلوقتى عندك 3 فانكشنز اسمهم Eat,Drink , Run  
فانت بتعمل ايه بتروح تعملهم فانكشنز جديده اسمها CanEat,CanDrink,CanRun الفانكشنز دول مهمتهم انهم بيخزنوا جواهم الفانكشن eat,drink,run و لما تحب تدى واحده منهم لكلاس ما بتروح عامل assign للكلاس ده انه بيعرف ياكل و بيعرف يشرب و بيعرف يجرى او ممكن بيعرف ياكل و يشرب بس او بيعرف ياكل بس تعالى اوريلك الكومبوسيشن ككود بقه و اوريك اذاى ورث

بص ياعم ادى الكود بال prototype و معاه الصوره بتأكد كلامى

function animal( Name) {

    this.Name = Name;

}

animal.prototype.eat = function (...param) {

    console.log(`${param[0]}  can eat`);

}

animal.prototype.drink = function (...param) {

    console.log(`${param[0]} can drink`);

}

animal.prototype.run = function (...param) {

    console.log(`${param[0]} can run`);

}

function cat( Name) {

    this.Name = Name;

}

function dog( Name) {

    this.Name = Name;

}

function inheritance(Mychild,MyParent){

    Mychild.prototype = Object.create(MyParent.prototype);

    Mychild.prototype.constructor = Mychild;

}

inheritance(cat,animal);

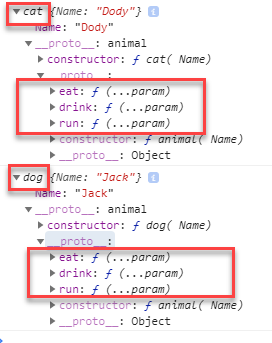
inheritance(dog,animal);

const CatObj = new cat("Dody");

const DogObj = new dog("Jack");

console.log(CatObj);

console.log(DogObj);



اما طريقه ال composition بقه أحسن 100 مره بص يا صديقى السهوله و الحلاوه و الطعامه فى الكود و كمان هى مش وراثه اصلا ده يعتبر بعمل الفانكشن و بروح اقول لكل كلاس انت بتعرف تعلم تاكل و تشرب يبقى خلاص انا هعمل Assign و اعرف البروتوتايب بتاعك انك بتعرف تاكل و تشرب غير بيعرف ياكل و يشرب و يجرى يبقى ابعتله فى ال assign ال 3 سلوك دول انه يعرف يعملهم و هكذا و عشان كده هتلافى انى معملتش هنا اصلا كلاس ال animal لان اصبحت مش محتاجه اصلا

// create Behavires By Refactoring Functions and save it in CanDo Const

const CanEat = function () {

    return {

        eat : function () {

            console.log(`${this.Name}  can eat`);

        }

    }

}

const CanDrink = function () {

    return {

        drink : function () {

            console.log(`${this.Name}  can drink`);

        }

    }

}

const CanRun = function () {

    return {

        run : function () {

            console.log(`${this.Name}  can run`);

        }

    }

}

function cat( Name) {

    this.Name = Name;

}

//adding Behaivers to Cat by Object.assign

Object.assign(cat.prototype,CanEat(),CanDrink())

function dog( Name) {

    this.Name = Name;

}

//adding Behaivers to Dog by Object.assign

Object.assign(dog.prototype,CanEat(),CanDrink(),CanRun())

const CatObj = new cat("Dody");

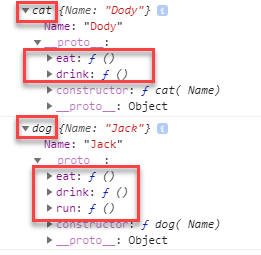
const DogObj = new dog("Jack");

console.log(CatObj);

console.log(DogObj);

اول حاجه بروح اعمل فانكشنز اسمها CanDrink و جواها بعمل return للفانكشن الى انا عاوز اضيفها بعد كده لاى كلاس من الاخر كده الفكره انى بعمل فانكشن اسمها CanDrink و احط فيها الريتيرن مش قيمه لأ ده الريتيرن بتعها عباره عن اوبجكت فيه فانكشن فلما انا اروح انادى ال CanDrink فبدل ما هترجع قيمه لأ هترجع فانكشن كامله و كأن CanDrink دى عباره عن فانكشن مخزن بخزن فيها الفانكشن الاساسيه لما انادى ل CanDrink كانى بقولى فاتحلى المخزن ياعم الحج و شغلى فانكشن ال Drink و اهميه الكلام ده انى لما اجى بعد كده عاوز اضيف الفانكشن دى لكلاس معينه فبروح اقوله ياباشا اعملى   
Object.assign و عرفه انى عاوز احط الميثود دى او السلوك ده داخل البروتوتايب بتاع الكلاس الفلانى و عشان كده انا قولتله مثلا اعملى اسيين و عرف كلاس القطه انه يعرف ياكل و يشرب بس و كمان روح عرف كلاس الكلب انه يقدر ياكل و يشرب و يجرى و طبعا انا قايله ان البيهيفير الى يقدر يعملها خزنهالى جوه البروتوتايب بتاع كل كلاس منهم مش داخل الكلاس نفسها لان الطبيعى انى بسجل اى مثود داخل البروتوتايب مش جوه الكلاس :D

و ادى صوره نتيجه الكود الى بتأكد كلامى



من الاخر ملخص الكومبوزيشين انه بيتعمل كده : كلاس حيوانات هيورث منه كلاس القطه فى سلوك انها تاكل  
بص افهم من الحته الصغيره دى كل حاجه بقه ههههههههههه

// create Behavires By Refactoring Functions and save it in CanDo Const

const CanEat = function () {

    return {

        eat : function () {

            console.log(`${this.Name}  can eat`);

        }

    };

}

function cat( Name) {

    this.Name = Name;

}

//adding Behaivers to Cat by Object.assign

Object.assign(cat.prototype,CanEat())

const CatObj = new cat("Dody");

console.log(CatObj);

CatObj.eat();

لو حابب تعمل ريفاكتور للكلاس cat تعمل اذاى ؟؟  
بص يا سيدى كل الى هتعمله هتحول الفانكشن الى عامله ذى الكونستراكتور دى لفاكتورى و ده عن طريق انك هتعمل الاسم جوه اوبجيكت و بيتملى بفاريبول الاسم برضوا و تخليه بيعمل ريتيرن فى حاجه اسمها MyCatObj و بعد كده تعمل الاسين بتاع الكومبوزيشن داخل الكلاس بقه و تدي للكلاس ال MyCatObj , Behaviors

بص على الكود الى تحت ده هتفهم ياعم الحج متصدعنيش

ملحوظه مهمه جدا جدا جدا جدا

الكومبزيشن ممكن نعمله للكلاس عادى خالص لان اصلا الفانكشن فى الجافا سكريبت هى هى الكلاس و بالتالى هتعامل مع الكلاس كانى بتعامل مع فانكشن بس مكتوبه بشكلها بعد التعديل

const CanEat = function () {

    return {

        eat : function () {

            console.log(`${this.Name}  can eat`);

        }

    }

}

class cat {

    constructor(Name){

        this.Name = Name;

    }

}

Object.assign(cat.prototype , CanEat())

const CatObj = new cat("Dody");

console.log(CatObj);

CatObj.eat();

ادى الكومبزيشن مع الكلاس يا باشا عاوز حاجه تانى تعالى بقه يا اسطى نخش فى الحاجات العنب بقه

#######################################################################  
inheritance in Functions By Call

----------------------------------------------

هنا بقه انا هشرح اذاى تعمل انهيرتانس من فانكشن لفانكشن تانيه من غير استخدام البروتوتايب و هيبقى الوراثه هنا عن طريق this و call بمعنى انى فى الابن هنادى الاب و ابعتله قيمه this بالاوبجيكت بتاع الابن وبالتالى اى اضافه للبروبرتيس على this الى انا بعتها هى اضافه فى اوبجيكت الابن و ادى الطريقه بالكود يا اسطى

        function Father(Name, Fav) {

            // this here is Father\_Obj

            this.Fathername = Name;

            this.Fatherfav = Fav;

            this.getInfo = function () {

            console.log(this.Fathername + "  likes " + this.Fatherfav);

            }

        }

        function Child(Name, Age) {

            // this here is Child\_Obj

            this.ChildName = Name;

            this.Childage = Age;

            // I will send it to Father to inhert from him

            Father.call(this, "Ahmed Badawy", "Basket ball");

            // Overloading After Inheritance

            this.getInfo = function (Name, Fav) {

                console.log("Hi From Child");

            }

        }

        var Child\_obj = new Child("Mahmoud", 25);

        var Father\_obj = new Father("Badawy", "Football");

        console.log(Father\_obj);

        console.log(Child\_obj);

        Father\_obj.getInfo();

        Child\_obj.getInfo();

ذى ما انت شايف فى فانكشن الابن عملنا فانكشن مهمتها انى ابعت فيها this بتاعت الابن للاب و معاها طبعا البراميتارز بتاعت الفانكشن بتاعت الاب و ده عشان املى بيهم خصائص الاب و انا بورثها و ذى ما انت شايف بقوله خش على فانكشن و ممكن نعتبره كلاس الاب father اعملها call و ابعتلى فيها ابوجيكت الابن فى this عشان لما اروح و املى this بتاعت الاب يبقى بتاعت الابن اخدت مكانها و بالتالى اى خاصيه قبلها this تتورث من الاب للابن على طول

و طبعا انت عارف الاوفرلودينج و هو ان لما الابن يورث فانكشن من الاب يقدر يعمل عليها اى تعديل و هنا بنسميها overloading و ده طبقته فى الجزئيه بتاعت getInfo

#######################################################################

OOP In ES6

----------------------------

كل الى فات ده كان ES5 و كل ده اصدارات قديمه و جه بقه البرنس الجديد ES6 حدث كل ده و عمل الشغل كما يجب ان يكون و سهل الدنيا خالص و من هنا هنشرح بقه كل الى فوق ده بقه بعد التحديث فى ملخص سريع جدا

شوفت الكلام الجميل الى فوق ده هوريهولك بقه بالكلاس هيبقى سهل جدا و هو عن طريق ال

Class – extends – constractors – super

class Father {

    Fathername;

    Fatherfav;

    constructor(Name, Fav) {

        this.Fathername = Name;

        this.Fatherfav = Fav;

    }

    getInfo = function () {

        console.log(this.Fathername + "  likes " + this.Fatherfav);

    }

}

class Child extends Father {

    ChildName;

    Childage;

    constructor(Name, Age, FthrName, Fav) {

        super(FthrName, Fav);

        this.ChildName = Name;

        this.Childage = Age;

    }

//Overloading

    getInfo = function () {

        console.log(`Hi Mesho My Fav is Coding`);

    }

}

var Father\_obj = new Father("Badawy", "Football");

var Child\_obj = new Child("Mahmoud", 25, "Boshkash", "TV");

console.log(Father\_obj);

console.log(Child\_obj);

Father\_obj.getInfo();

Child\_obj.getInfo();

ذى ما انت شايف كلاس دى عشان اعرف الكومبيلر ان دى كلاس و اكستيند دى عشان اورث من كلاس لكلاس اخر و داخل كل كلاس بقه بروباريتيس و فيه كونستراكتور و ده بملى فيه البروبارتيس و من الاخر كده بستخدمه فى نقل البيانات بين كلاس وكلاس و

ال super بقه دى بنحطها داخل كونستراكتور الابن الذى ورث و هو لازم تتحط و ببعت فيها اى حاجه عاوز ابعتها كانى بكلم كونستراكتور الاب تماما و ده لان سوبر دى اعتبرها كونستراكتور الاب الى هو الكلاس الى انا وارث منه و العمليه دى بتسمى هنا Dependency injection

#######################################################################

OOP In ES6

----------------------------