***Object oriented programming:***

It is based on classes and objects , whereas classes are the blue prints of objects.

Objects with building of resubale code like functions all are wrapped up under a class.

In javascript , classes are itself wrapped up in special function in some sort , this means they are an alternate of defining objects in js using a constructor function.

Making up classes in javascript is way different than classical oop programming in java or .net

It lets you to inherit(copy or use) properties and functions of one class to another class easily.

Oop consist of 4 paradigms :

* Classes and objects
* Inheritance
* Prototypes
* Classes

**Objects :** are the collection of properties and methods. The object ‘person’ have properties like : ‘name’ , ‘age’ , ‘height’ etc.

//object

let dog = { dogName: "JavaScript",  // object properties

 weight: 2.4,

 color: "brown",

 breed: "chihuahua"

 };

**Classes :** encapsulate data and function that are the part of particular class.

 class Guy {  // class name

    constructor(age,covid){  // constructor and its parameters passed

        this.age = 56

        this.covid = false

    }

 }

 let obj = new Guy("age","covid"); // object creation , arguments

 console.log(obj);

remember we said classes are also just special function as well.

Example :

//classes as spacial functions

function Ship(size,manfactyear,floors,type,route){

    this.size=size;

    this.manfactyear=manfactyear;

    this.floors=floors;

    this.type=type;

    this.route=route;

}

let ship = new Ship("210000 Tons",2008,4,"84CB","atlantic ocean");

console.log(ship);

as actual class:

example:

//above function can also be turned into actual class syntax likee this

class Shippy{

    constructor(size1,manfactyear1,floors1,type1,route1){

        this.size1=size1;

        this.manfactyear1=manfactyear1;

        this.floors1=floors1;

        this.type1=type1;

        this.route1=route1;

    }

}

let ship1 = new Shippy("56000 Tons",2000,2,"21.33b","pacific ocean");

console.log(ship1);

console.log("the ship of class : ",ship1.type1," follows the pathway of ",ship1.route1);

**classes :** as if functions and object does the same thing as classes why do we need classes ? because classes leverage us to do less typing , and prevent typo mistakes.

In objects we need to target and initialize each and every property again and again whenever we need the same properties in other functions or things.

We can reuse properties of objects in every function as more we want.

**Constructor :** it is a type of special method that we need to initialize at the top of the class.

Constructor contains properties of objects and this properties will be set and initiating the class.

class Shippy{

    constructor(size1,manfactyear1,floors1,type1,route1){

        this.size1=size1;

        this.manfactyear1=manfactyear1;

        this.floors1=floors1;

        this.type1=type1;

        this.route1=route1;

    }

}

Graphical user interface, text, application

Description automatically generated

**Methods :**  inside a class we cant call a function (function) instead of this we call function a method which can eventually use properties of objects initialized inside constructor.

We don’t need to use keyword function inside a class , but directly define a method and use it later on.

// a class with a method

class Person{

    constructor(fname,lastname){

        this.fname=fname;

        this.lastname=lastname;

    }

    greet(){

        console.log("hey there ! my name is ",this.fname," and my father name is : ",this.lastname);

    }

}

let obj3 = new Person("Ahmed Ali Ansari","Asghar Ali Ansari");

obj3.greet();

Graphical user interface, text, application, email

Description automatically generated

**Properties:** they hold the data inside a class also called sometimes fields , usually writtern inside a constructor.

class Person{

    constructor(fname,lastname){

        this.fname=fname; // property

        this.lastname=lastname; // property

    }

    greet(){

        console.log("hey there ! my name is ",this.fname," and my father name is : ",this.lastname);

    }

}

**Lock properties :**

We can restrict usage of properties inside a class body. This is done by placing “#” with property. See the below example :

// locking properties

class Person2 {

    #firstname2;

    #lastname2;

    constructor(firstname2, lastname2) {

    this.#firstname2 = firstname2;

    this.#lastname2 = lastname2;

    }

   }

Now if we try to access it through object outside of paranthese of class body it says undefined.



// locking properties

class Person2 {

    #firstname2;

    #lastname2;

    constructor(firstname2, lastname2) {

    this.#firstname2 = firstname2;

    this.#lastname2 = lastname2;

    }

   }

let res = new Person2("ahmed","ali");

console.log(res.firstname2); // undefined.

To access them at all cost this is where getter and setters comes into play.

**Getter and setter:** in order to use restricted properties accession outside class boundary we use getter and setter , to make fields private and wont allow modifications in properties by other classes we use restriction # , this is also called **encapsulation.** But to use it precisely we use getter and setter methods.

Getter and setter are look like functions but in reality they are called accessors with keywords getter and setter.

// get set

class Gumnaam{

    #badnaam;

    #Khabar;

    constructor(badnaam,Khabar){

        this.#badnaam = badnaam;

        this.#Khabar = Khabar;

    }

    get badnaam(){

        return this.#badnaam;

    }

    set badnaam(badnaam){

        this.#badnaam = badnaam;

    }

    get Khabar(){

        return this.#Khabar;

    }

    set Khabar(Khabar){

        this.#Khabar = Khabar;

    }

}

let num = new Gumnaam("mien","nahi");

console.log(num.Khabar); // nahi

we can also update properties value , because there is a set that can sets new value into a property and get returns the value for us.

num.badnaam = "tum";

console.log(num.badnaam); // mein updated to tum

**Inheritance :** it is one of the key concepts of oop programming. It lets you use property of one class into another classes usually a parent class properties and methods can be used by child classes.

// Inhertiance

class Vehicle{

    constructor(type1,typ2,typ3,minspeed,maxspeed,){

        this.type1 = "two wheeler";

        this.typ2 = " 4 wheeler";

        this.typ3 = "Tempo";

        this.maxspeed = "78kmph";

        this.minspeed = "35kmph";

    }

    move(){

        console.log("the ",this.typ2," is moving at the speed of ",this.minspeed);

    }

}

class Motorcylce extends Vehicle{

    constructor(type1,maxspeed,fuel,company,fuel2){

           super(type1,maxspeed);

           this.fuel = "12 liters";

           this.company = "Honda VXR";

           this.fuel2 = fuel2;

    }

    classify(){

        console.log("look at the ",this.company," of ",this.type1,

        " moving at the speed of ",this.maxspeed," with fuel of ",this.fuel);

    }

    another(adjust){

        this.fuel = adjust;

    }

}

let val = new Motorcylce();

val.classify();

//val.fuel2("20 liters");

val.another("20 liters");

val.classify();

**Prototype:** we can add functions , properties of objects into class instance , outside of class using prototype keyword like below example:

// prototype

class Student{

    constructor(sem,id){

        this.sem = sem;

        this.id = id;

    }

    introduce(){

        return "hey there from : "+this.sem+" of id : "+this.id;

    }

}

Student.prototype.introduce = function(){    // added new function through prototype

    console.log("new function added to the class Student !");

};

Student.prototype.Gpa = 3.52;        // added new property of object into constructor through prototype

let showcase = new Student("Ahmed","4th");

console.log(showcase,showcase.Gpa);