# Constructor

It is defined as normal function and only difference is what it returns and how we invoke it.

In constructors we mainly use ‘this’ keyword which refers to current object. ‘’this’ keyword is important as we use constructor to create multiple objects using ‘new’ keyword.

// constructor function definition

function Employee(name){

// ‘this’ keyword is used to refer to current object.

// since we use construct to crate multiple object ‘this’ is important

    this.name = name;

    this.empid = function(){ return this.name + 'abcd123' }

}

// create new instance of constructor using ‘new’

const emp = new Employee('Ashpak')

console.log(emp.empid());

## object prototype

All objects in JavaScript inherits from ‘prototype’

if we have to add one more property to constructor, we can **NOT** add it directly

// can not directly add property to constructor like below. This wont

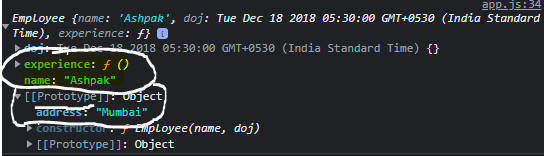
// give error but when property is access it will have value as 'undefined'

Employee.address = 'Pune';

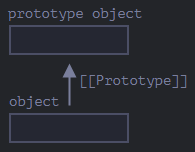
We can use prototype to add new property on constructor

Employee.prototype.address = 'Mumbai'; // This works

If we see properties added to prototype in console, they are shown not directly under object, but shown under ‘prototype’.



In JavaScript, objects have a special hidden property [[Prototype]] as shown above, that is either null or references another object. That object is called “a prototype”.



When we read a property from object, and it’s missing, JavaScript automatically takes it from the prototype. In programming, this is called “prototypal inheritance”

function animal(){

    this.eats= function(){ return true};

};

animal.prototype.breath  = function(){return true;}

function rabbit(){

    this.jumps= function(){ return true};

};

// set rabbits prototype same as animal prototype so that is any of missing

// property is referred in rabbit then it will look into its prototype which is same as animal prototype

rabbit.prototype = animal.prototype;

let an = new animal();

let rab = new rabbit();

// we can find both properties in rabbit now:

console.log(rab.jumps());

console.log(rab.breath());

## Object.create()

This is ES5 syntax of inheritance. ES6 makes these things easy using classes which is just a syntactical sugar, as Javascript doesn’t have classes.

JavaScript is object-oriented language, it isn't a [class-based language](http://en.wikipedia.org/wiki/Class-based_programming)—it's a [prototype-based language](http://en.wikipedia.org/wiki/Prototype-based_programming).

This creates a new object, using an existing object as the prototype of the newly created object.

/// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROTOTYPE INHERITANCE  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const person = {

    age: 20,

    printIntroduction: function() {

      console.log(`My name is ${this.name}. I am ${this.age} years old`);

    }

  };

  // creating new object and inheriting all properties in prototype

  const me = Object.create(person);

  me.name = 'Ashpak'; // "name" is a property set on "me", but not on "person"

  me.age= 35; // inherited properties can be overwritten

  me.printIntroduction();

  //==> My name is Ashpak. I am 35 years old

# CLASSES

Classes are syntactical sugar on top of prototype-based approach we have seen.

class Person{

        //constructor to initialize properties

        constructor(name,age){

            this.name = name;

            this.age = age;

        }

        //class methods which can be access by objects.

        // THESE METHODS WILL BE ADDED TO OBJECTS PROTOTYPE SAME AS PROTOTYPE

        // INHERITNACE WHICH WE HAVE SEEN ABOVE

        printIntroduction(){

            console.log(`My name is ${this.name}. I am ${this.age} years old`);

        }

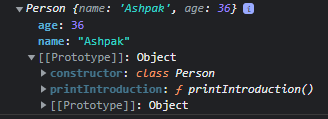
}

const newMe = new Person('Ashpak', 36);

newMe.printIntroduction();

//==> My name is Ashpak. I am 35 years old

Class methods are added to prototype of newly created object, so under the hood classes works on prototype inheritance since Java doesn’t have classes.



# Static Method

When we need some methods like helper method which doesn’t needs to access class variable or doesn’t need class object to be instantiated then we can use static method.

In above example if we need some method which returns full gender like ‘Male’ or ‘Female’ depending on shortform then we can add it as static method and accept gender shortform as param.

    static getFullGender(gender){

        if (gender === 'm') return 'Male';

        if (gender === 'f') return 'Female';

    }

}

const newMe = new Person('Ashpak', 36);

newMe.printIntroduction();

//==> My name is Ashpak. I am 35 years old

// while calling static method we don’t need to initialize the object of class we just need to access method directly using class.

console.log(Person.getFullGender('m'));

//==> output Male

# Subclass

Subclass can access parent class methods and static methods from parent class. Parent class can’t access methods from subclass.

// create a subclass by extending parent class

class Customer extends Person{

    constructor(name, age, phone, type){

        // need to call constructor from parent class

        super(name,age);

        // setting own class properties

        this.phone = phone;

        this.type = type;

    }

    printCustomerInfo(){

        console.log(`I am ${this.name} a ${this.type} customer. My phone is ${this.phone}.`);

    }

}

const newCust = new Customer('Ashpak',37,99999999,'premium');

//call own class method

newCust.printCustomerInfo();

//call parent class method

newCust.printIntroduction();