class inheritance (subclasses)

- A classes which inherit functionality from another class also have their own additional properties or methods too are called as subclasses of class inheritance.

- in other words, a subclass extends another class.

```
- class User {
      constructor(username, email) {
          this.username = username;
          this.email = email;
      }
      login() {
          console.log(`${this.username} hello`);
          return this;
      }
      logout() {
          console.log(`${this.username} Good`);
          return this;
      }
}
class Admin extends User {

}
const userone = new User('mario', 'hello@gmail.com');
const userTwo = new User('hello', 'good@gmail.com);
const userThree = new Admin('shaun', 'shaun@gmail.com');
console.log(userThree);
```

↓

▶ Admin {username: "shaun", email: "shaun@gmail.com"}
and when we extend the admin i.e. when we click on ▶ we get following result

email: "shaun@gmail.com"
username: "shaun"
▼ -proto- : User
▶ constructor : class Admin
▼ -proto- :
▶ constructor : class User
▶ login : f login()
▶ logout : f logout()
▶ -proto- : Object

So here we have used the concept of class inheritance that is why in above example we have created new class called admin and to admin class we have all methods which our User class have.
Here we have created the admin class which extends the User class and therefore we get all properties and methods inherited inside admin class

And Now we can define extra methods and properties in admin class if we want to that only admins have
so for this consider below example :-

```
class Usee {
    constructor(useename, email) {
        this.useename = useename;
        this.email = email;
    }
    login() {
        console.log(`${this.username} logged in`);
    }
}
class Admin extends Usee {
    deleteUsee(usee) {
        usees = usees.filter((u) => {
            return u.useename !== usee.useename;
        })
    }
}
const useeOne = new Usee('mario', 'mario@gmail.com');
const useeThree = new Admin('hello', 'hello@gmail.com');

let usees = [useeOne, useeThree];
console.log(usees);
useeThree.deleteUsee(useeOne);
console.log(usees);
```

explanation on next page (1)

explanation next page (2)

▶ (2) [Usee, Admin] ← here we have both mario and hello along with their email and login method

▶ (1) [Admin]

here now we only have one hello along with email and login method

here, we have deleted 'useeOne' i.e. mario with email and method is deleted

① If this admin class does not have a constructor inside it then automatically it will look to class that ___ extends from and then it will call that constructor

② users = users.filter((u) => {
return u.username !== user.username;
});

↑ This can be written as below

↓

users = user.filter(u => u.username !== user.username);

↓ Just we have shortened our code

So now we have here a new class "Admin" which extends the "user", it inherits all of the methods and properties that class "User" has but here now only "Admin" has the "userDelete" method
And when try to use that userDelete method in class User we will get error.
i.e. In short only admins can use the "userDelete" method.

Now what if class "Admin", to class "Admin" how should we add some extra properties as well as methods, so for that we need to have Admin class constructor and also inherit the methods and properties of another constructor i.e. in previous example "User" has its own constructor
so to do this consider next

super()

- Now what if, add additional properties to only class "Admin" and we don't want those properties to be in class "User", we only want "Admin" to have these additional properties
so for these we need to define our own constructor inside the "Admin" class because that is where properties are defined inside a class.

- when we call super() inside a constructor then it looks for the parent class and it looks inside that class constructor.

```
class Usee {
    constructor(useename, email){
        this.useename = useename;
        this.email = email;
    }
    login(){
        console.log(`${this.useename} logged in`);
        return this;
    }
    logout(){
        console.log(`${this.useename} logged out`);
        return this;
    }
}
class Admin extends Usee {
    constructor(useename, email, title){
        super(useename, email); ← explanation on next page, next page ①
        this.title = title;
    }
    deleteUsee(usee) {
        usees = usees.filter(u => u.useename !== usee.username);
    }
}
const userOne = new User('hello', 'hello@gmail.com');
const userTwo = new Usee('no', 'nono@gmail.com');
const userThree = new Admin('yes', 'yes@gmail.com, 'ola');
console.log(userThree);
```

↓

▼ Admin {username: "yes", email: "yes@gmail.com", title: "ola"}
email: "yes@gmail.com"
title: "ola"
username: "yes"
▶ __proto__ : User

Now here only class "Admin" have property called "title" Regular "User" class won't.

Here class "User" has two properties and two methods but class "Admin" has now its own one property called "title" because we have used their constructor and in constructor we have also used super() so its meaning is along with one property "title", "Admin" class also has two properties of class "User" so in total class "Admin" has three properties.
Also "Admin" class has one method called "deleteUser" class "User" has two methods login() and logout() but no "deleteUser" but in class "Admin" we have access to login() and logout() so total "Admin" class has three methods too.

↖ Why access to login() and logout() due to "extends"

Explanation of ① in code we

when will not use this below line in our code then

```
super(username, email);
```

↓

if we will not use this then we will get following error.

▶ Uncaught ReferenceError : Must call super constructor in derived class before accessing 'this' or returning from derived constructor

Constructors (under the hood)

Before 'class' was introduced we would write code as following

older way

```
function User(username, email) {
    this.username = username;
    this.email = email;
}
```

← capital first letter and it distinguishes this function as a constructor

```
class User {
    constructor(username, email) {
        this.username = username;
        this.email = email;
    }
}
```

↑ the constructor is responsible for setting up the new properties

Both codes are exactly the same upperone is when class was not introduced. Both are giving the same output result.

- Now what if we want to add methods using oldee way for that consider below code

this first letter is capital so it indicate that this function is a constructor

```
function User(username, email) {
    this.username = username;
    this.email = email;
    this.login = function() {
        console.log(`${this.username} logged in`);
    }
}
const useeOne = new User('hello', 'hello@gmail.com');
useeOne.login();
```

hello logged in

we can do this because "this" eefees to the object instance inside constructor.

Heee we have added method called "login()"

we can add functions oe methods like we have done above but there is a bettee way to add methods to oue objects not inside the consteuctoe and that is where prototype comes into "picture"