Object.create()

```
const createPlayerFuncs = {
        charge: function charge(){
            this.battery = 100;
        }
}

function createPlayer(name, battery){
    let newPlayer = Object.create(createPlayerFuncs)
    newPlayer.name = name;
    newPlayer.battery = battery;
    return newPlayer;
}
```

Object.create creates a link from newPlayer to createPlayerFuncs object.

So now newPlayer object that we created via Object.create() has access to createPlayerFuncs object through prototypal inheritance

If you console.log(newPlayer) you won't see charge function, but if you log newPlayer.__proto__ you can see it became newPlayer is pointing upto createPlayerFuncs in the prototypal chain.

Another alternative → Constructor functions

```
function createPlayer(name, battery) {
        this.name = name;
        this.battery = battery;
}

const player1 = new createPlayer('John', 20)
> player1.name
> John
```

Any function invoked with the new keyword is a constructor function.
We don't need to return the object from the constructor function, instead it is automatically created and returned

Convention : Always start your constructor function with capital letter.
So CreatePlayer instead of createPlayer

** when it comes to constructor functions, 'this' keyword dosent point to global (window) object

instead it points to the object we want to create. In this case player1.

So this.name = name; is equivalent to player1.name = name

That's how we get these properties in player1

So all of it happens because we're using new keyword

Now how do we actually attach charge function when using constructor functions.

When it comes to constructor functions, we can attach any new properties using prototype that every function has access to when created

```
createPlayer.prototype.charge = function(){
    this.battery = 100;
}
```