Optional Chaining

Optional chaining (`?.`) in ES6 allows you to safely access object properties, methods, or array elements without causing errors if they don't exist. If a property is `null` or `undefined`, it returns `undefined` instead of throwing an error.

```
const user = { profile: { name: "John" } };

console.log(user?.profile?.name); // Output: John
console.log(user?.profile?.age);  // Output: undefined (no error)
console.log(user?.account?.balance); // Output: undefined (no error)
```

Map()⇒ *Map Mainly perform a loop and its return as a array*

```
let numbers=[2,3,4,5,6,78];
let ans=numbers.map(value⇒value+1);
```

```
let ans=numbers.map(value⇒{
   let sum=value+2;
   return sum;
})
 console.log(ans);

let friends=['jobbar','poyzar','zeleka','ano','koysar','kuddus'];

let newfriend=friends.map((element,index)⇒{
   console.log(index, element);
})
```

forEach ⇒ is also a array method that does not return anything

```
let products=[
   {id:1, name:"Iphone", color:"Silver",price:1500, brand:"Apple"},
   {id:2, name:"Samsung", color:"Black",price:1200, brand:"sm"},
   {id:3, name:"Nokia", color:"White",price:800, brand:"Nokia"},
   {id:4, name:"Iphone", color:"gold",price:1200, brand:"Apple"},
   {id:5, name:"Samsung", color:"blue",price:1300, brand:"sm"},
]
products.forEach(product ⇒{
    console.log(product);
   if(product.price>=1000)
   {
       console.log(product);
   }
})
```

filter⇒ is a array method that returns a filtered array based on specific condition but if does not match any condition then it will return a empty array

```
let products=[
    {id:1, name:"Iphone", color:"Silver",price:1500, brand:"Apple"},
    {id:2, name:"Samsung", color:"Black",price:1200, brand:"sm"},
    {id:3, name:"Nokia", color:"White",price:800, brand:"Nokia"},
    {id:4, name:"Iphone", color:"gold",price:1200, brand:"Apple"},
    {id:5, name:"Samsung", color:"blue",price:1300, brand:"sm"},
];

let filterdProducts=products.filter(product => product.brand==="Apple");
console.log(filterdProducts);
```

find() is also a array method that only return a single value/object based on conditions.If condition does not match then it will show undefined. **Note: When we need multiple/all elements based on condition then we will use filter() method on the other-hand when we need just a single value then use find() method.**

```
let products=[
    {id:1, name:"Iphone", color:"Silver",price:1500, brand:"Apple"},
    {id:2, name:"Samsung", color:"Black",price:1200, brand:"sm"},
    {id:3, name:"Nokia", color:"White",price:800, brand:"Nokia"},
    {id:4, name:"Iphone", color:"gold",price:1200, brand:"Apple"},
    {id:5, name:"Samsung", color:"blue",price:1300, brand:"sm"},
];

let product=products.find(product=> product.brand==='sm');
console.log(product);
```

Class→Constructor→Inheritance→Super

```
class Vechile{
    constructor(name,price,brand)
    {
        this.name=name;
        this.price=price;
        this.brand=brand;
    }

    move()
    {
        console.log("Can move");
    }
}
class Bus extends Vechile{
    constructor(name,price,brand,seat)
    {
        super(name,price,brand);
        this.seat=seat;
    }
    route()
    {
        console.log("Dhaka to Rangpur")
    }
};
class Car extends Vechile{
    constructor(name,price,brand,year)
    {
        super(name,price,brand);
        this.year=year;
    }
    route()
    {
        console.log("Only Entire Dhaka city");
    }
};
```

```
let Hanif= new Bus("Hanif",12333,"HINO",50);
console.log(Hanif);
Hanif.route();

let car=new Car("Tesla",12300,"Tesla",2025);
console.log(car,car.route());
```

Encapsulation is a process to protect data that don’t allow to access outside of class

```
class Person{
    #isMarid;
    #sallary;
    constructor(name,isMarid,age,sallary)
    {
        this.name=name;
        this.#isMarid=isMarid;
        this.age=age;
        this.#sallary=sallary;
    }
};
let kuddus=new Person("Kuddus",false,25,23999);
console.log(kuddus);
```