Working with Maps

↓

Some like objects but has an ability of iteration & can Key can be of any types, method of adding elements inside is

```
Const mapVar = new Map()
mapVar.set (Key, value)
Console.log(mapVar.get(Key))
for (let i of mapVar) {
    Console.log(i)}
Console.log(mapVar.keys())
```

→ [Key1, Value1], [Key2, Value2], [Key3, Value3]

→ all the Keys.

another way of adding elements into the map

↓

```
mapVar = new Map(
    [[Key, value], [Key, value],
     [Key, value]])
```

Optimum way of getting Keys & values from loop

```
for [Key, value] of mapVar {
    Console.out('Key is', Key)
    Console.out('value is', value)
```

① Cloning,

```
Const Var={ name:"name"}
Const VarCopy = Obj.assign
    ({}, Var)
```

②

Optional Chaining

↓

helps code to not returning an error when data type exists Keys are not present

↓

```
let Var:={ Key : "data of Key"}
Console.log(Var?.Key)
```

→ suppose if we don't have Key & Var Variables over JS code will return Undefine but not return an error.

? Will check if var exist or not

This Keyword

↓

always select the entire calling object

"use strict"

always select the entire calling object

```
const obj1 = {
    name : "name",
    Marks : "marks"
    }

const obj2 = {
    name : "name",
    Marks : "marks"
    }

const obj3 = {
    name : "name" , }
    Marks : "marks"
    }
```

```
function getInfo (obj) {
    console.log(`Your value is ${this.name} &
    Your marks are (this.marks)`)
```

imp

Call , apply , bind → helps you to store your function with this inside to a new variable

Call: Help you to call a function which is having this with it

apply: Same but optional parameters will be passed as an array

eg

```
function thisInside ( ) {
    console.log(`name is ${this.name} & marks is ${this.marks}`)

info1 = { name: "name1",
          marks: "marks1" }

info2 = { name: "name2",
          marks: "marks2" }

info3 = { name: "name3",
          marks: "marks3" }
```

Call ⇒ thisInside.call(info2)

apply ⇒ thisInside.apply(info2, [any other parameters in array])

Bind ⇒ const objbind = thisInside.bind(info3)

This in arrow function

this is arrow will always refer to the object one level above the normal level.

Proto → best technique for memory management in JS.

Your new object will also contain a reference of an old or another object

So you will have an infinite chain of multiple objects inside a single object.

Creating a chain that will help you to get the property of obj that you have stored in diff object using ob_2 = Obj. Create (Proto object)

eg → const older_obj = { "name" : "name", "course" : "course" }

new_obj = Object.new (older_obj)

new_obj ['marks'] = 100

new_obj ['attendance'] = 100

Console.log (new_obj ['name']) o/p = name

Prototype → a property of a function that have unlimited storage for objects and functions.

function protoose () { }

protoose.prototype.name = "name"

protoose.prototype.info = { class : "class", sub : "sub" }

protoose.prototype.ret_license = function (age) { return age > 18 }

Console.log (protoose.prototype.name)

→ Console.log (protoose.prototype.info)

Console.log (protoose.prototype.ret_license(21))