DESTRUCTURING :-

Destructuring is the process of extracting array elements and storing them inside individual variables. and extracting object properties & storing them inside individual variables.

There are 2 types of destructuring.

(I) Array Destructuring.
(II) Object Destructuring.

(I) Array Destructuring :-

→ It is the process of ~~des~~ extracting array element and store them inside individual variable.

→ It avoids using array name again and again when accessing array elements.

⇒ When we do array destructuring elements will be stored inside the variable in sequencial order.

⇒ If you want to skip the element or don't want to store the elements, just pass coma (,) at the time of destructuring.

Eg :- let colors = ['red', 'green', 'blue', 'yellow'];
clg (colors [0])
clg (colors [1]).

or let r = colors [0]. // red
let g = colors [1]. // green.
clg (r, g, b)

```
let [r, g, b, y] = colors;
clg (r, g, b, y); // red, green, blue, yellow.
let [r, g, , y] = colors;
clg (r, g, y). // red, green, yellow
let [r, g, ....p] = colors;
clg (r, g, p) // red, green, ▶['blue', 'yellow']
```

```
let [r, ...p, c] = colors
clg (r, p, c). //error: Rest parameter must be last element
```

element or

```
let colors = ['red', 'blue', 'green', 'yellow']
let getColors = ([r, b, g, y]) =>
{
  console.log (r, b, g, y)
}
getColors (colors)

let getNumbers = () =>
{
  return [2, 7];
}
let res = getNumber ();
clg (res);
or
let [a, b] = getNumbers ();
clg (a, b)
```

(11) Object Destructuring :-

Object Destructuring is the process of extracting object properties and storing them inside individual variables.

⇒ Object Destructuring allows us to avoid using object name again and again when we access object properties

⇒ In Object Destructuring the variable names must be same as key names.

⇒ If we want to skip or don't want to store the

property, just ignore the keyname at the time of destructuring.

Eg:-

```
let student = { sname: "sara, age: 31, dept: "CSE",
                gender: "male", marks: 650 }

console.log (student)

let sname = student.sname;
console.log (sname).
```

without destructuring

destructuring.

1. `let { sname, age, dept, genden, mark } = student;`
   `clg ( sname, age, dept, gender, mark);`
   `// Sara 31 CSE male 650`
2. `let { gender, age, marks } = students`
   `clg (age, gender, marks)`
   `// 31 male 650.`
3. `let getstudentData = ({ sname, age, dept, gender, mark })`
   `=> {`
   `clg (sname, age, dept, mark, gender).`
   `}`
   `getStudent Data (Student).`
   `// Sara, 31 CSE 650 male.`
4. `let getStudent Data = () =>`
   `{`
   `return { sname: "Sara", dept: "CSE", gender: "mal`
   `marks : 650" }`
   `}`

5. `let { sname, age, mark } = get Student Data()`
   `clg ( sname, age, mark); // Sara 31 650`

5. let { gender, age, Marks, ...detail } = student
clg (age, gender, Marks, detail) // 31, Male 650 Mark
{ sara, CSE }

Nested Object Destructuring :-

```
let Student =
{
    sname : "Manisha",
    age : 22.,
    Marks : 560
    address :{
        home :{ city :"Bhawanipatna", state :"Odisha"}
        work : { city : "Bangalore"}
    }.
}
clg (student).
let { sname, address} = student.
let { home, work} = address.
let { city, state} = home
let { city : new city } = Work.
```

(or) → Renaming

```
let { sname, address : { home : {city, state},
work : { city: newcity } } } = student;
clg ( sname, city , newCity , state)
// Manisha Bhawanipatna Bangalore Odisha.
```

Object in an Array destructuring :-

Eg :-
```
let students = [
    {
      sname : "Syed"
      age : 31
      Marks : 650
    },
    {
      sname : "John"
      age : 36
      Marks : 780
    }
]
```

Case-1:
```
for(let {sname, age, Mark} of students)
{
  clg (sname, age, Mark) // Syed 31 560
}                              John 26 460
```

Case-2:
```
let res = student.map ({sname}) =>
{
  return sname.toUpperCase()
})
clg(res); // [SYED, JOHN]
```

Object Method

Eg:-

```
let Student = {
    sname : "Manisha",
    age : 21
    Marks : 650
}
clg (Student);
```

```
let employee {
    Sname : "Manisha"
    age : 21
    Marks : 650
}.
clg (employee).
```

```
clg (object. keys (Student)) // [sname, age, Marks]
clg (object. values (Student))
clg (object. entries. (Student))
clg (object. keys (Student). length)
clg (object. has own property ("Sname")); // true.
clg (object. has own property ("details")); // False.
object. freeze (student);
object. seal (student);
student. dept = "CSE";
student. Mark = 320;
delete student. age;
clg (object. isfrozen (Student)); // true
clg (object. isSealed (student)); // true.
```

```
let worker = Student
console.log (Student)
clg (object. is (student, Student) // true
clg (object. is (student, employee) // false
clg (object. is (student, worker) // true
```

```
let newobj = Object.create(student).
console.log(newObj)
clg(newObj.sname)
clg(Object.is(student, newObj)) // false.
```

↳ It is used to create a new object and ~~xxxx~~ copies the same object with different reference.

```
let newobj = Object.assign(student, employee);
clg(newobj)
clg(student)
```

↳ It will merge the employee object to student object.

```
let newobj = Object.assign({}, student, employee);
clg(newobj)
clg(student === newObj)
clg(student === employee)
```

Moment.js → Library

Date and time Methods ():-

```
let d = new Date().
console.log(d);
console.log(d.getFullYear())
console.log(d.getDate())
console.log(d.getMonth())
clg(d.getHours())
clg(d.getDay())
clg(d.getminutes())
clg(d.getseconds())
```