Loops →

① For loop →

Syntax → for (let index = 0; index < array.length ;
index ++) {
const element = array[index];
}

eg → for (let i = 0; i <= 10 ; i++) {
console.log(i);
}

② while loop →

Syntax → while (condition) {
}

In while loop we have to increment or decrement the condition variable inside the code scope.

eg → let index = 0
while (index <= 10) {
console.log (`value of index is ${index}`);
index++;
}

③ do-while loop →

Syntax → do {
} while (condition);

eg → let score = 1
do {
console.log (`score is ${score}`);
score++;
} while (score <= 10);

This loop will always run atleast once

④ for of loop →

Syntax → for (const iterator of object) {
}

This loop is specially for the objects on which you can iterate. Here object is not JS object but element where you want to iterate.

eg → i) const arr = [1, 2, 3, 4, 5]
for (const num of arr) {
console.log(num);
}

ii) const greetings = "Hello world!"
for (const greet of greetings) {
console.log(`Each char is ${greet}`)
}

we can apply for of loop on map data structure in JS as well.

Map →
The Map object holds key-value pairs and remembers the original insertion order of the keys where as objects do not keep insertion order.
In Map the values will be unique, it can't hold duplicasy.

const map = new Map()
map.set('IN', "India")

```
map.set('USA', "United States of America")
map.set('Fr', "France")
map.set('IN', "India")

console.log(map)
```

Output→ Map(3) {
'IN' => 'India',
'USA' => 'United States of America',
'Fr' => 'France'
}

for of loop on map

(i)
```
for (const key of map){
    console.log(key)
}
```

Output→ ['IN', 'India']
['USA', 'United States of America']
['Fr', 'France']

(ii)
```
for (const [key, value] of map) {
    console.log(key, ':-', val)
}
```

This method is used when we want to get values seprately.

Output→ IN :- India
USA :- United states of America
Fr :- France

Note → you can't iterate objects using for of loop as you iterate maps, you will get error. There are different ways to iterate in objects.

⑤ for in loop →
This loop is used to iterate in objects.

Syntax → for (const key in object) {
}

eg → const myObject = {
js : 'javascript',
cpp : 'C++',
rb : 'ruby',
swift : 'swift by apple'
}
for (const key in myObject) {
console.log(key)
}

Output → js
cpp
rb
swift

To print value as well →
for (const key in myObject) {
console.log(myObject[key]);
}
Output → javascript
C++
ruby
swift by Apple

you can use the for in loop on array as well but if you directly print key you will get array key i.e 0, 1, 2 ... , but to get the value use the same syntax as in object i.e array [key]

you can't use for in loop for maps because they are not iterable.

⑥ For each loop →

This is basicall not a loop but a function that takes a call back function as a parameter. call back function is a function that don't have any name.

eg → const coding = ["js", "ruby", "java", "python"]

(i) coding.forEach (function (val) {
    console.log(val);
})

(ii) using arrow function →

function.forEach ((item) => {
    console.log(item);
})

(iii) we can do the same using predefined function but when calling it in forEach we pass reference

```
function printMe(item) {
    console.log(item);
}
coding.forEach(printMe)
```

(iv) This function has access to index and whole array as well

```
coding.forEach((item, index, arr) => {
    console.log(item, index, arr);
})
```

Important key points of for each loop →

① This loop is a higher order array loop.
② This is widely used when we have array of objects. This loop is generally used when we are working with databases as we get array of objects

eg →
```
const myCoding = [
    {
        languageName: "JavaScript",
        languageFileName: "js"
    },
    {
        languageName: "python",
        languageFileName: "py"
    }
]
```

```
myCoding.forEach ( (item) => {
        console.log (item.languageName);
})
```

Output→ javascript
python

→ Foreach loop do not return any value so if you try to store it in a variable or return it and then store you will still get undefined.

```
const values = coding.forEach ( (item) => {
        //console.log (item)
        return item
})
console.log (values);
```

Output→ undefined

⑦ filter function or loop for array→

The filter function is same as foreach but difference is that it returns a value and you can give condition if you want specific conditional values.

eg→ const myNums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```
const newNums = myNums.filter((num) =>
                                num > 4)
console.log(newNums)
```

Output → [5, 6, 7, 8, 9, 10]

OR

```
const newNums = myNums.filter((num) => {
    return num > 4
})
```

Doing same thing with forEach →

```
const newNums = []

myNums.forEach((num) => {
    if (num > 4) {
        newNums.push(num)
    }
})
console.log(newNums);
```

Output → [5, 6, 7, 8, 9, 10]

→ you can use filter for objects as well

⑧ map function →

This is same as filter function but main difference is that it will always return a value, so you can also perform operations and also it support chaining

eg→① const myNumbers = [1,2,3,4,5,6,7,8,9,10]

const newNums = myNumbers.map((num) =>
{ return num+10; })

Output → [11,12,13,14,15,16,17,18,19,20]

chaining →

② const newNums = myNumbers
.map((num) => num*10)
.map((num) => num+1)

Output → [11, 21, 31, 41, 51, 61, 71, 81, 91, 101]

③ you can also chain map with filter

const newNums = myNumbers
.map((num) => num*10)
.map((num) => num+1)
.filter((num) => num >=40)

console.log(newNums)

Output → [41, 51, 61, 71, 81, 91, 101]

⑨ Reduce function →

reduce() method parse and return value from calculation on the preceding element.
The final result of running the reducer accross all elements of array in single value.

in this we take accumulator and in the beginning it take a initial value and the execute rest, after getting first value accumulator hold the value of current value and cycle continues like this till we iterate over whole array.

eg → const myNums = [1,2,3]

```
const myTotal = myNums.reduce(function(acc, currval) {
    console.log(`acc : ${acc} and currval : ${currval}`)
    return acc + currval
}, 0)
```

This is the initial value that acc will take or accumulator will take

```
console.log(myTotal)
```

Output →
```
acc : 0  and currval: 1
acc : 1  and currval: 2
acc : 3  and currval: 3
6
```

Another way →

```
const myTotal = myNums.reduce((acc, curr) => acc+curr , 0)
console.log(myTotal)
```

Output → 6

Note → This reduce function is widely used when we have to add the prices in shopping cart and you already know that data generally comes in form of objects and we already know how to work with objects.