31. Functions:

Some predefined functions:

```
var a = [                    ];
• var a.push( );
• a . pop( );
• a.shift( );
• a. unshift( );
• console.log(a.join(" - "));
• console.log(a.indexOf(" "));
• a. sort( );
• a. reverse( );
```

- In above we pass arguments.

Passing function as argument in predefined functions:

```
var age = ["h1", "h2", "h3", "h5", "h6"];
var f = age.find(myFunc);
function myFunc(value, index, array){
        return index >1;
}
console.log(f);
```

O/p: h3

Objects Methods:

```
var obj = {65:"A", 66:"B", 67:"C", 68:"D"};
console.log(obj);
console.log(Object.keys(obj));      //["65", "66", "67", "68"
console.log(Object.values(obj));    //["A", "B", "C", "D"]
console.log(Object.entries(obj));   // shows in dimension
```

O/p: Start This is normal string End

34. ES6 DESTRUCTURING:

- let:

allows to declare a variable with block scope

```
var x = 10;
  // Here x is 10.
{
 let x=2;
   // Here x is 2.
}
// Here x is 10.
```

- Using const is safer than using var, because a function expression is always a constant value.

- Default value

```
let person = {
      firstname: "Nikhil",
      lastname: "Agrawal",
      age: 20,
    }

let {age , firstname, lastname , middlename=" "}=person;
console.log(age);
console.log ( firstname );
console.log( lastname );
console.log (middlename);
```

O/P: 20
  Nikhil
  Agrawal .

- Arrow function:

M1.

```
function add (x,y)
{
  return x+y'
}
```

M.2.

```
let add = function (x,y){
        return x+y;
}
console.log(add(10,20))
```

35. ES6 - Map, filter, Promise :-

. Maps :-

- A Map hold key value pairs where the keys can be any datatype.
- A map remembers the original insertion order of the keys.
- A map has a property that represents the size of map.

Map Methods:

new Map() - creates a new Map object
set() - sets the value for a key in a Map
get() - gets the value for a key in a Map
has() - returns true if a key exists in a Map

Ex.

```
let ranks = [1, 2, 3];
console.log (ranks.map ((index, e) => { return (e) }))
                         a      b
```

a → returns each element
b → returns each index

can perfor
operations a

★ Map returns a new array whereas forEach doesnot.

Difference between map and filter:
- Under map, if a condition doesnot satisfy, the return statement should show undefined where a in filter it will skip.

For example:

```
let ranks = [1, 2, 3, 4, 5, 6];
let new maparr = ranks.map ((e, index) => {
  if (e % 2 == 0)
     return e
```

```
let  newfilterarr = ranks. filter((e, index)=> {
        if( e%2 ==0 )
            return e
})
```

O/P: (3) [2,4,6]

Promise keyword :

Producing code is code that can take some time.
Consuming code is code that must wait for the result.
A promise javascript object is one that links producing code and consuming code.

myPromise. state	myPromise . result
pending	undefined
fulfilled	a result value
rejected	an error object

- Any function inside a object is a method.

Async-await :

It is promise with easy syntax.

Ex.

```
        variable                    object
let attendConcert = new Promise( function(resolve,
setTimeout () => {                     ↓        reject){
                                  executive        ↓
      if (concert) {   → to make  function
                         it asynchronous task
          resolve ("BOB ATTENDED");              parametric
      }                                          function
      else {
          reject ("BOB FAILED");
      }
    }, 2000 ),
});
console.log (attendConcert)
```

- await is combination of both .then and .catch

Now,

```
attendconcert.then (data) => console.data( data) . catch(error)
 => console.log (error)
```

is equivalent to

```
async function afunc() {
 let result = await attendconcert;
 console.log (result)
  return result;
 }
```

Try and catch for error handling:

M.1.

```
async function asyfunc() {
try {
    let result = await attendconcert ;
    console.log(result)
 }
 catch(e) {
    console.log(e)
  }
 }
 asyfunc()
```

```
import React from 'react';
function MyComponent(props){
    return <div>Hello, {props.name}</div>';
}
```

```
import ReactDOM from
    'react-dom';
function App () {
  return (
   <div>
     <MyComponent name=
      "John" />
   </div>
  );
}
```

```
ReactDOM.render(<App />, document.
getElementById('root'));
```

PROPS AND STATE:-
- used to manage data within a component
- Props (pass data (parent → child))
- read-only, cannot be modified by a child component
- Example

State:
- Used to manage data within a component that can change over time.

```
class MyComponent extends React.Component {
    constructor (props) {
    super(props);
    this.state = { count: 0 };
  }
  incrementCount(){
     this.setState ( { count : this.state.count +1 }));
  }
  render(){
    return (){
      <div>
        <p>Count : { this . state . count }</p>
        <button onClick ={ () => this.incrementCount()} >
        Increment</ button>
      </div>
     );
  }
}
ReactDOM.render (< MyComponent />, document .getElement
Id ('root'));
```

Hooks: allow us to use state and other React features in functional components.

Previously → Only used in class components

• useState ( ) :-
- allows add state fun. comp.
- takes initial state value as an argument and returns an array containing the current state value and a function to update the state.

• use Content ( )
- content object as an argument and returns current value of that content.

• use Effect ( )
- add side effects (fet. data or updating DOM) to fun. comp.
- argument → function
- called after the component has rendered.

• use Reducer ( )
- Hook that is used for state management
- alternative to useState
- useState is built using useReducer

Redux: State management library for Javascript applications

- powerful and flexible way to manage state in React application making it easier to manage complex data flows and ensuring that your UI stays in sync with the underlying data.

Provides a way to pass data or state through the component tree without having to pass props down manually through each nested component.

Reduce vs useReducer ( )

reduce in Javascript	useReducer in React
(i) array.reducer(reducer, initialValue)	(i) useReducer (reducer, initialState)
(ii) single value = reducer (accumullator, itemValue)	(ii) new state = reducer (currentState, action)
(iii) reduce method returns a single value	(iii) useReducer returns a pair of values. [newState, dispatch]