1)find duplicate and same values in two array

```
var fullWordList = ['1','2','3','4','5'];
var wordsToRemove = ['1','2','3'];

// here we find duplicate values
var duplicates = fullWordList.filter(value => wordsToRemove.includes(value));

// Find same values (intersection)
var sameValues = fullWordList.filter(value => wordsToRemove.includes(value));

console.log("Duplicates:", duplicates);
console.log("Same values:", sameValues);
```

Code Explanation: This code uses the filter() method to iterate over the fullWordList array and check if each value is present in the wordsToRemove array using the includes() method. The duplicates array will contain all the values that are present in both arrays, while the sameValues array will contain the values that are present in both arrays without any duplicates.

2) const word = '00000111110101001111100001001';

```
// Function to find the longest chain of letters in a word
function findLongestChain(word) {
  let currentChain = 0;
  let longestChain = 0;

  for (let i = 0; i < word.length; i++) {
    if (word[i] === '1') {
      currentChain++;
      longestChain = Math.max(longestChain, currentChain);
    } else {
      currentChain = 0;
    }
  }

  return longestChain;
}

// Call the function and print the result
const longestChain = findLongestChain(word);
console.log("Longest chain of letters:", longestChain);
```

3)
```
let obj1 = { "greeting": "hello" };
let obj2 = obj1;
obj1["greeting"] = "Bye";
console.log(obj1);
console.log(obj2);
```

Ouput : { greeting: 'Bye' }
{ greeting: 'Bye' }

4) console.log("7" > 7) **output: false**
console.log("2">"21") **output: false**
console.log("KL">"S") **output: false**

5) function average(a, b) {
return a + b / 2;
}
console.log(average(2, 1));

Ans: 2.5

For Node.Js

Example for Promise.all, resolve , reject

```
const promise1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('Promise 1 resolved');
  }, 2000);
});

const promise2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject(new Error('Promise 2 rejected'));
  }, 1500);
});

const promise3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('Promise 3 resolved');
  }, 1000);
});

Promise.all([promise1, promise2, promise3])
  .then((results) => {
    console.log('All promises resolved:', results);
  })
  .catch((error) => {
    console.log('Error occurred:', error.message);
  });
```

For React :

A class and a function component example (please also use Typescript for function component: For example create interface for function Props)

// CLASS COMPONENT

```
import React, { Component } from 'react';

// Props interface
interface MyClassProps {
  name: string;
}

// State interface
interface MyClassState {
  count: number;
```

```
}

class MyClass extends Component<MyClassProps, MyClassState> {
  constructor(props: MyClassProps) {
    super(props);
    this.state = {
      count: 0,
    };
  }

  handleClick = () => {
    this.setState((prevState) => ({ count: prevState.count + 1 }));
  };

  render() {
    const { name } = this.props;
    const { count } = this.state;
    return (
      <div>
        <h1>Hello, {name}!</h1>
        <p>Count: {count}</p>
        <button onClick={this.handleClick}>Increment Count</button>
      </div>
    );
  }
}

export default MyClass;
```

// FUNCTION COMPONENT

```
import React, { useState } from 'react';

// Props interface
interface MyFunctionComponentProps {
  name: string;
}

const MyFunctionComponent: React.FC<MyFunctionComponentProps> = ({ name }) => {
  const [count, setCount] = useState(0);

  const handleClick = () => {
    setCount((prevCount) => prevCount + 1);
  };

  return (
    <div>
      <h1>Hello, {name}!</h1>
      <p>Count: {count}</p>
      <button onClick={handleClick}>Increment Count</button>
    </div>
  );
};
```

```
export default MyFunctionComponent;
```

Use hooks to get update regarding any state value into function component.

```
import React, { useState, useEffect } from 'react';

// Props interface
interface MyFunctionComponentProps {
  name: string;
}

const MyFunctionComponent: React.FC<MyFunctionComponentProps> = ({ name }) => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log(`Count has been updated: ${count}`);
  }, [count]);

  const handleClick = () => {
    setCount((prevCount) => prevCount + 1);
  };

  return (
    <div>
      <h1>Hello, {name}!</h1>
      <p>Count: {count}</p>
      <button onClick={handleClick}>Increment Count</button>
    </div>
  );
};

export default MyFunctionComponent;
```

Into function component show example for hooks (useState,UseEffect,UseCallback, useContext)

Using Typescript

```
import React, { useState, useEffect, useCallback, useContext } from 'react';

// Context
const MyContext = React.createContext('');

// Component using hooks
const MyFunctionComponent: React.FC = () => {
  // useState hook
  const [count, setCount] = useState(0);

  // useEffect hook
  useEffect(() => {
    console.log('Component mounted');
    return () => {
      console.log('Component unmounted');
    };
  }, []);
```

```
  // useCallback hook
  const handleClick = useCallback(() => {
    setCount(prevCount => prevCount + 1);
  }, []);

  // useContext hook
  const contextValue = useContext(MyContext);

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={handleClick}>Increment Count</button>
      <p>Context Value: {contextValue}</p>
    </div>
  );
};

// Parent component
const ParentComponent: React.FC = () => {
  return (
    <MyContext.Provider value="Hello, Context!">
      <MyFunctionComponent />
    </MyContext.Provider>
  );
};

export default ParentComponent;
```

Using React

```
import React, { useState, useEffect, useCallback, useContext } from 'react';

// Custom context
const MyContext = React.createContext();

// Component using hooks
const MyFunctionComponent = () => {
  // useState hook
  const [count, setCount] = useState(0);

  // useEffect hook
  useEffect(() => {
    console.log('Component mounted');
    return () => {
      console.log('Component unmounted');
    };
  }, []);

  // useCallback hook
  const handleClick = useCallback(() => {
    setCount((prevCount) => prevCount + 1);
  }, []);

  // useContext hook
  const value = useContext(MyContext);
```

```
  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={handleClick}>Increment Count</button>
      <p>Context Value: {value}</p>
    </div>
  );
};

// Usage of MyContext.Provider
const App = () => {
  const contextValue = 'Hello, useContext!';

  return (
    <MyContext.Provider value={contextValue}>
      <MyFunctionComponent />
    </MyContext.Provider>
  );
};

export default App;
```

Add indexing for Schema's

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  username: { type: String, required: true },
  email: { type: String, required: true },
  age: { type: Number, required: true },
  createdAt: { type: Date, default: Date.now }
});

// Add indexes to the schema
userSchema.index({ username: 1 });
userSchema.index({ email: 1, age: -1 });

const User = mongoose.model('User', userSchema);
```

ALL CODE ARE ATTACHED IN GITHUB