**1)Max consecutive 1 in an array**

function maxConsecutiveOnes(arr) {

let str = arr.join('');

let maxCount = Math.max(...str.split('0').map(x => x.length));

return maxCount;

}

// Test the function

let arr = [1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0];

O/P: 3

**2)How to remove the duplicates**

**Type 1**

var arr = ["apple", "mango", "apple",

"orange", "mango", "mango"];

function removeDuplicates(arr) {

return arr.filter((item,

index) => arr.indexOf(item) === index);

}

console.log(removeDuplicates(arr));

**Output:**

["apple", "mango", "orange"]

**Type 2:**

**Using a Set() Method: This method sets a new object type with ES6 (ES2015) that allows you to create collections of unique values.**

var arr = ["apple", "mango", "apple",

"orange", "mango", "mango"];

function removeDuplicates(arr) {

return [...new Set(arr)];

}

console.log(removeDuplicates(arr));

**Output:**

["apple", "mango", "orange"]

**Type 3:**

let arr1 = ["apple", "mango", "apple","orange", "mango", "mango"];

let final = arr1.reduce((acc,cur) =>{

if(!acc.includes(cur)){

acc.push(cur)

}

return acc

}, [])

console.log(final)

**Type 4:**

let arrFruit = ["apple", "mango", "apple",

"orange", "mango", "mango"];

let finalArray = [];

for (item in arrFruit){

if(!finalArray.includes(arrFruit[item])){

finalArray.push(arrFruit[item])

}

}

console.log(finalArray);

O/P: ['apple', 'mango', 'orange']

**3)String sort: (Ascending order)**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();

O/p:  ['Apple', 'Banana', 'Mango', 'Orange']

**String sort: (Descending order)**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

console.log(fruits.sort().reverse())

O/P: ['Orange', 'Mango', 'Banana', 'Apple']

**4) Reverse array:**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.reverse();

O/P: Mango,Apple,Orange,Banana

**5) Number sort: (Ascending)**

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a - b});

O/P: 1,5,10,25,40,100

**6) Number sort: (Descending)**

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b - a});

O/P: 100,40,25,10,5,1

**7) Splitting the object data based on key values:**

const fruits = {f1:"Banana",f2: "Orange", f3:"Apple", f4:"Mango"};

//console.log(Object.entries(fruits))

let arr1 = {} , arr2 ={}

for ([key,value] of Object.entries(fruits)){

if(key.substring(1)>2)

{

arr2 = {...arr2, [key]: value};

}else{

arr1 = {...arr1, [key]: value};

}

}

console.log("arr1:", arr1, "arr2:", arr2);

**Assign:**

assign is primarily used for objects. It copies enumerable own properties from one or more source objects to a target object. If properties have the same keys, the later source object's properties overwrite earlier ones. It modifies the target object directly (mutates).

const target = { a: 1, b: 2 };

const source1 = { b: 3, c: 4 };

const source2 = { c: 5, d: 6 };

Object.assign(target, source1, source2);

// target is now { a: 1, b: 3, c: 5, d: 6 }

**concat**

concat is used for arrays and strings. It creates a new array or string by joining existing ones. It does not modify the original arrays or strings (immutable).

const array1 = [1, 2];

const array2 = [3, 4];

const newArray = array1.concat(array2);

// newArray is [1, 2, 3, 4]

// array1 and array2 remain unchanged

const string1 = "hello";

const string2 = " world";

const newString = string1.concat(string2);

// newString is "hello world"

// string1 and string2 remain unchanged

**7) To flatten the array**

let arr = [[1, 2],[3, 4],[5, 6, 7, 8, 9],[10, 11, 12]];

console.log('concat:', [].concat.apply([], arr))

O/P:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

**8) To find the sum of the array elements (using reduce)**

let arr = [1,4,6,7]

console.log(arr.reduce((total,val)=> total = total +val))

O/P: 18

**9) To find the sum of the array of object (using reduce)**

let obj = [{name: 'aw', age:3},{name:'re', age:6}]

console.log(obj.reduce((total,item)=> total +=item.age,0)) -> the last “0” is initial value total variable

**10) To add an attribute to each array element**

**Output Problems:**

1. console.log(2 + '2' - 1); *// Output: 21 (string concatenation then subtraction)*
2. console.log([] == ![]); // Output: true (both sides coerce to 0)
3. setTimeout(() => console.log(1), 0);  
    console.log(2);

// Output: 2, 1 (due to event loop and macrotask queue)

**11)Skip few properties: (properties to skip should be configurable not fixed)**

**I/P:**

const Avengers = [

  {

    fname: "tony",

    lname: "stark",

    age: 30,

    gender: "M",

    powers: ["inteligence", "money"],

  },

  {

    fname: "natasha",

    lname: "romonov",

    age: 23,

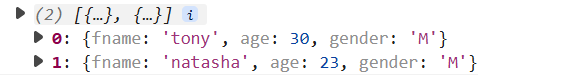
    gender: "M",

    powers: ["inteligence", "slow ageing"],

  },

];

**O/P:**

****

**Solution:**

const Avengers = [

{

fname: "tony",

lname: "stark",

age: 30,

gender: "M",

powers: ["inteligence", "money"],

},

{

fname: "natasha",

lname: "romonov",

age: 23,

gender: "M",

powers: ["inteligence", "slow ageing"],

},

];

const attributesRequired =(data, ...attributes)=>{

console.log('attri:', attributes);

let tempArr = [];

data.forEach((avenger)=>{

let temp = {}

console.log('aven:',avenger);

Object.keys(avenger).forEach((key) => {

console.log('key:', key);

if (attributes.includes(key)){

temp = {...temp, [key]: avenger[key]}

}

})

tempArr = [...tempArr, temp]

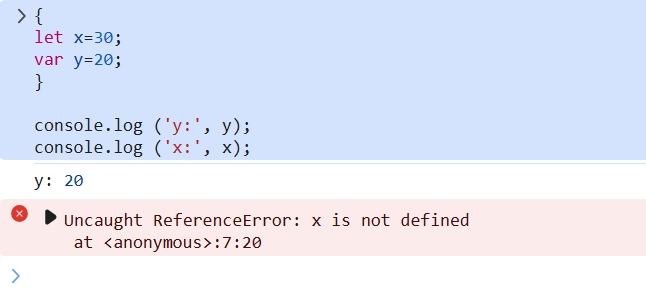
})

return tempArr;

}

console.log(attributesRequired(Avengers, 'fname','age','gender' ))

**What is the output?**



for(var i=0; i<3 ; i++){

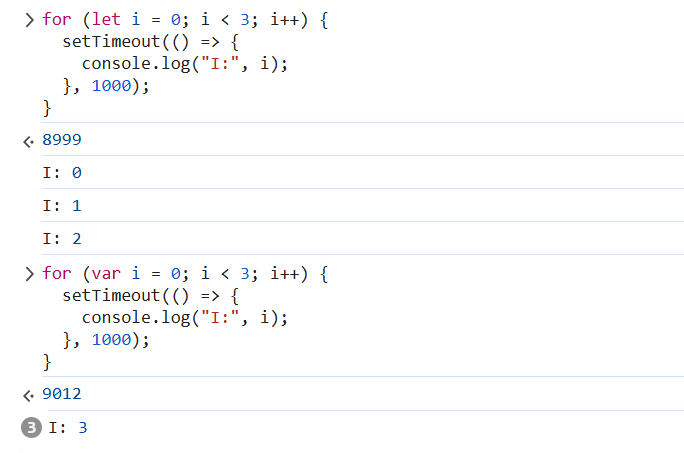
setTimeOut(()=>{

console.log(“I:”, i)

},1000)

}

**Output with var and let due to asynchronous call:**



**Reason for all the times 3 for “var”**

The classic issue of closures and asynchronous code!

The problem is that the setTimeout function is asynchronous, and by the time it executes, the loop has already finished, and the value of i is 3. So, all three timeouts will log "I: 3" to the console.

**To fix this, you can use an immediately invoked function expression (IIFE) to create a closure that captures the current value of i:**

for (var i = 0; i < 3; i++) {

(function(index) {

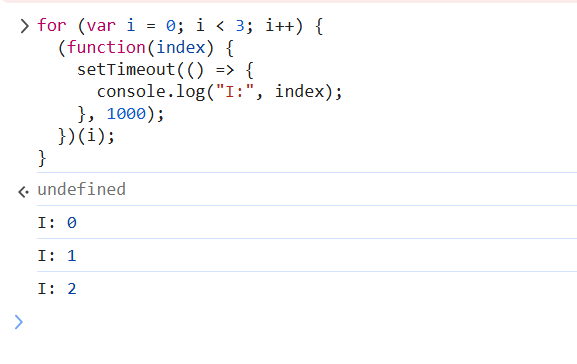
setTimeout(() => {

console.log("I:", index);

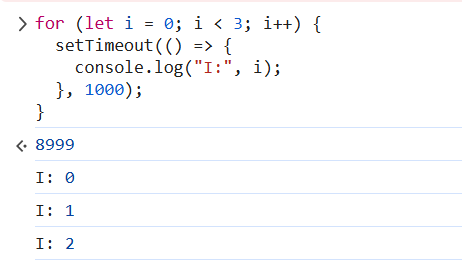
}, 1000);

})(i);

}



**Alternatively, you can use let instead of var to declare the loop variable, which will create a new scope for each iteration:**

****

**Debounce functionality custom function**

**Example: 1**

// Create a text input element

const input = document.createElement('input');

input.placeholder = 'Type something...';

document.body.appendChild(input);

// Create a paragraph element to display the debounced output

const output = document.createElement('p');

document.body.appendChild(output);

// Define the debounce function

function debounce(func, wait) {

let timeout;

return function() {

const context = this;

const args = arguments;

const later = function() {

timeout = null;

func.apply(context, args);

};

clearTimeout(timeout);

timeout = setTimeout(later, wait);

};

}

// Define the function to be debounced

function logInput(event) {

output.textContent = `You typed: ${event.target.value}`;

}

// Create a debounced version of the logInput function

const debouncedLogInput = debounce(logInput, 500);

// Add an event listener to the input element

input.addEventListener('input', debouncedLogInput);

**Explanation:**

This program creates a text input element and a paragraph element to display the debounced output. It defines the debounce function, which takes a function and a wait time as arguments, and returns a new function that debounces the original function.

The program then defines the logInput function, which logs the input value to the paragraph element. It creates a debounced version of this function using the debounce function, with a wait time of 500ms.

Finally, the program adds an event listener to the input element, which calls the debounced logInput function whenever the input value changes.

When you run this program and type something into the input element, you will see that the output is only updated after 500ms have passed since the last input change. This demonstrates the debounce functionality.

Here's a breakdown of how the program works:

The user types something into the input element.

The input event is triggered, and the event listener calls the debounced logInput function.

The debounced function checks if the wait time (500ms) has passed since the last call. If not, it does nothing.

If the wait time has passed, the debounced function calls the original logInput function, passing the current input value as an argument.

The logInput function updates the paragraph element with the current input value.

By debouncing the logInput function, we ensure that the output is only updated after a short delay, rather than on every keystroke. This can help improve performance and reduce the number of unnecessary updates.

**Example 2:**

import { useState } from 'react'

import reactLogo from './assets/react.svg'

import viteLogo from '/vite.svg'

import './App.css'

function App() {

  const [count, setCount] = useState(0)

  const [inputValue, setInputValue] = useState(0)

  const debounce = (fn, delay) => {

    console.log('debounce')

    let timeoutId;

    return (...args) => {

      if (timeoutId) {

        clearTimeout(timeoutId);

      }

      timeoutId = setTimeout(() => {

        fn(...args);

      }, delay);

    };

  }

  const handleInputChange = () => {

    //console.log('Input value changed to:', event.target.value)

    setInputValue((prev)=>prev+1)

    console.log('Input value changed to:', inputValue)

  }

  const debouncedHandleInputChange = debounce(handleInputChange, 1000)

  return (

    <>

      <div>

        <a href="https://vite.dev" target="\_blank">

          <img src={viteLogo} className="logo" alt="Vite logo" />

        </a>

        <a href="https://react.dev" target="\_blank">

          <img src={reactLogo} className="logo react" alt="React logo" />

        </a>

      </div>

      <h1>Vite + React</h1>

      <div className="card">

        <button onClick={() => setCount((count) => count + 1)}>

          count is {count}

        </button>

        <p>

          Edit <code>src/App.tsx</code> and save to test HMR

        </p>

        <p>Debounce Increment:{inputValue}</p>

        <button onClick={debouncedHandleInputChange}>Increment</button>

      </div>

      <p className="read-the-docs">

        Click on the Vite and React logos to learn more

      </p>

    </>

  )

}

export default App

**Example 3: (debounce using lodash.debounce) – a library**

**To install lodash.debounce:**

**npm install lodash.debounce**

(OR)

**npm i --save-dev @types/lodash.debounce**

**App.tsx**

import debounce from 'lodash.debounce'

const [inputValue1, setInputValue1] = useState(0)

const handleInputChange = (event) => {

setInputValue1(event.target.value)

console.log('Input value changed to:', event.target.value)

}

const debouncedHandleInputChange = debounce(handleInputChange, 1500)

<label htmlFor="input">Debounce Input</label>

<input type="text" id="input" value={inputValue1} onChange={debouncedHandleInputChange} />

**Promise:**

const success = true;

const promise1 = new Promise((resolve, reject) => {

if (true) {

resolve({ name: 'success', msg: 'done' });

} else {

reject({ name: 'failure', msg: 'failed' }); // added closing bracket

}

});

promise1.then((message)=>{

console.log(message.name + ' ' + message.msg);

}).catch((message)=>{console.log(message.name + '' + message.msg);})

**Promise.all:**

const success = true;

const promise1 = new Promise((resolve, reject) => {

  if (success) {

    resolve({ status: "Promise 1 done" });

  } else {

    reject("Promise 1 failed"); // added closing bracket

  }

});

const promise2 = new Promise((resolve, reject) => {

  resolve("Promise 2 done");

});

Promise.all([promise1, promise2])

  .then((message) => {

    console.log(message[0].status);

  })

  .catch((error) => {

    console.log(error);

  });

****

**O/P:**

****

For Promise.all(), the output will be an array, since in the first promise we are trying to access the object and so we are printing only the first promise message. If both promise output to be expected then just give as a string.

const success = true;

const promise1 = new Promise((resolve, reject) => {

  if (success) {

    resolve("Promise 1 done");

  } else {

    reject("Promise 1 failed"); // added closing bracket

  }

});

const promise2 = new Promise((resolve, reject) => {

  resolve("Promise 2 done");

});

Promise.all([promise1, promise2])

  .then((message) => {

    console.log(message);

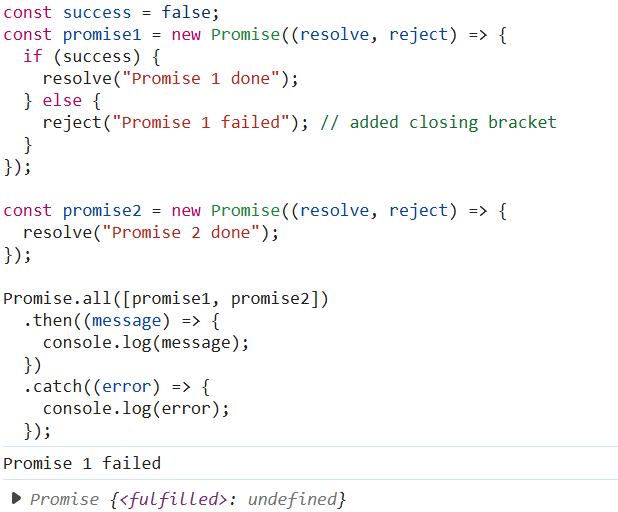
  })

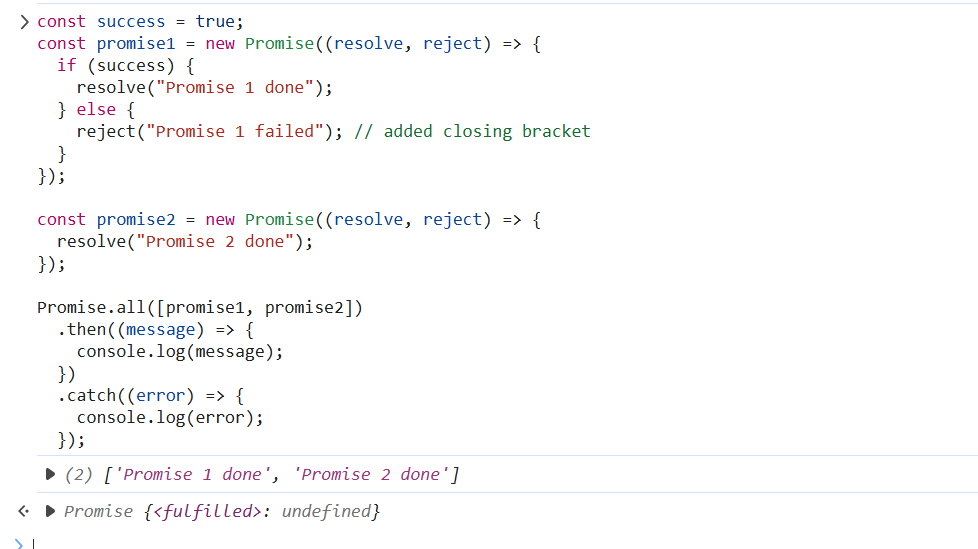
  .catch((error) => {

    console.log(error);

  });

Note: As one of the promise is failed, it came out





O/P:



**Promise.race (First success or failure is displayed)**

const success = true;

const promise1 = new Promise((resolve, reject) => {

  if (success) {

    resolve("Promise 1 done");

  } else {

    reject("Promise 1 failed"); // added closing bracket

  }

});

const promise2 = new Promise((resolve, reject) => {

  resolve("Promise 2 done");

});

Promise.race([promise1, promise2])

  .then((message) => {

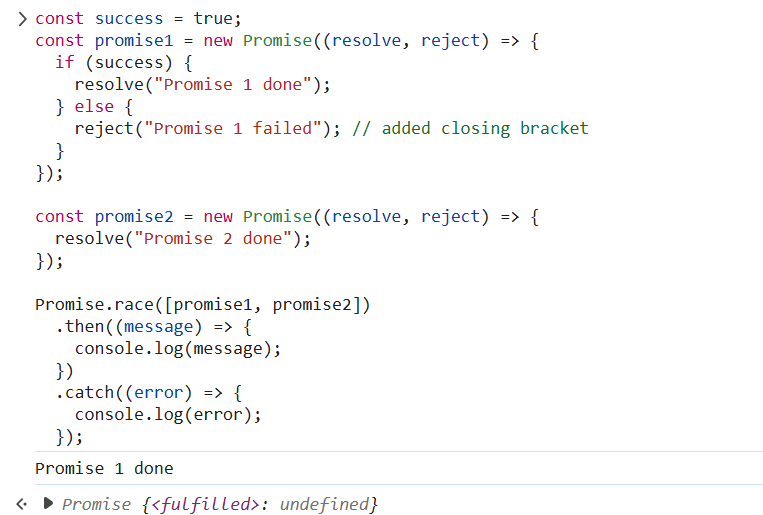
    console.log(message);

  })

  .catch((error) => {

    console.log(error);

  });

****

**How to write a listener in useEffect and clean it up when the component unmounts:**

import { useEffect } from 'react';

function MyComponent() {

useEffect(() => {

// Add the listener

window.addEventListener('scroll', handleScroll);

// Cleanup function to remove the listener when the component unmounts

return () => {

window.removeEventListener('scroll', handleScroll);

};

}, []);

const handleScroll = () => {

// Handle the scroll event

console.log('Scrolled!');

};

return <div>Hello World!</div>;

}

**Adding event listener to a button**

import { useEffect, useRef } from 'react';

function MyComponent() {

const buttonRef = useRef(null);

useEffect(() => {

const button = buttonRef.current;

if (button) {

button.addEventListener('click', handleButtonClick);

}

return () => {

if (button) {

button.removeEventListener('click', handleButtonClick);

}

};

}, []);

const handleButtonClick = () => {

console.log('Button clicked!');

};

return (

<div>

<button ref={buttonRef}>Click me!</button>

</div>

);

}

**React query:**

'use client';

import { QueryClient, QueryClientProvider } from 'react-query';

import '../globals.css';

import MainHeader from '@/components/main-header';

// export const metadata = {

// title: 'Next.js Page Routing & Rendering',

// description: 'Learn how to route to different pages.',

// }

export default function RootLayout({ children }) {

const queryClient = new QueryClient();

return (

<html lang="en">

<body>

<div id="page">

<MainHeader />

<QueryClientProvider client={queryClient}>

{children}

</QueryClientProvider>

</div>

</body>

</html>

)

}

**Page.js (news)**

'use client';

import "@/app/globals.css";

//import { DUMMY\_NEWS } from "@/dummy-news";

import NewsList from "@/components/news-list";

import { useEffect, useState } from "react";

import { useQuery } from "react-query";

import axios from "axios";

export default function NewsPage() {

    const [news, setNews] = useState([]);

    const {data, isLoading, error} = useQuery('news',async ()=>{

        const response = await axios.get('http://localhost:8080/news');

        return response.data;

    });

    useEffect(() => {

        if(data) {

            setNews(data);

        }

    }, [data]);

    if(isLoading) {

        return <p>Loading...</p>;

    }

    if(error) {

        return <p>{error.message}</p>;

    }

    console.log('news:',news);

    return (

        <div id="news">

            <h1>News</h1>

            <NewsList news={news} />

        </div>

    );

}

**To perform a POST action using Axios with React Query, you can use the useMutation hook from React Query. Here's an example:**

import { useMutation, useQueryClient } from 'react-query';

import axios from 'axios';

const postApiCall = async (data) => {

const response = await axios.post('/api/endpoint', data);

return response.data;

};

const { mutate, isLoading, error } = useMutation(postApiCall, {

onSuccess: () => {

// Handle success

},

onError: (error) => {

// Handle error

},

});

In this example, postApiCall is the function that makes the POST request using Axios. The useMutation hook is then used to wrap this function, providing options for handling the mutation's lifecycle.

You can then call the mutate function to trigger the POST request:

mutate({ /\* data to send in the request body \*/ });