

index.js + 3zymz2ccj NEW

```
1 // normal function
2 function f(a,b){
3     const sum = a + b
4     // sum = 3
5     return sum
6 }
7 console.log(f(2,4))
8
9
10 // Anonymous function
11 let f = function(a,b){
12     const sum = a + b
13     return sum
14 }
15 console.log(f(2,3))
16
17 // Immediate invoke
18 let f = function(a,b){
19     const sum = a + b
20     return sum
21 }(2,6)
22 console.log(f)
23
24 // Arrow function
25 let f = (a,b) => {
26     const sum = a + b
27     return sum
28 }
```

STDIN
Input for the program (Optional)

Output:
11

index.js + 3zywc39kh NEW

```
1 // Object is a collection of key value pairs
2
3 // Object Literals
4 var obj = {
5   name: "Chirag",
6   age: 18,
7
8   greet(name){
9     return "good morning " + name
10  }
11
12
13 }
14
15 console.log(obj.greet("Chirag"))
16
17
18 // objects within function
19 function Calc(val){
20   const obj = {
21     add(val1){
22       var a = val + val1
23       return a
24     }
25   }
26 }
27
28 }
```

STDIN
Input for the program (Optional)

Output:
good morning Chirag

index.js +

3zywc39kh

NEW

```
13 // }
14 // console.log(obj.greet("Chirag"))
15
16
17
18 // objects within function
19 function Calc(val){
20   const obj = {
21     add(val1){
22       var a = val + val1
23       return a
24     },
25     sub(val1){
26       var b = val - val1
27       return b
28     }
29   }
30   return obj
31 }
32
33
34 console.log(Calc(5).add(2))
35 console.log(Calc(5).sub(2))
36
37
38
39
```

STDIN

Input for the program (Optional)

Output:

7

3

index.js +

3zz2dk8fz

NEW

```
13 // }  
14 // console.log(obj.greet("Chirag"))  
15  
16  
17  
18 // objects within function  
19 function Calc(val){  
20  
21  
22     function add(val1){  
23         var a = val + val1  
24         return a  
25     }  
26  
27     function sub(val1){  
28         var b = val - val1  
29         return b  
30     }  
31     return obj = {  
32         add , sub  
33     }  
34 }  
35  
36  
37  
38 console.log(Calc(5).add(3))  
39 console.log(Calc(5).sub(5))
```

STDIN

Input for the program (Optional)

Output:

8

0

```
index.js 8:45
1 // var arr = new Array(3)
2 // arr[0] = 2
3 // arr[1] = 4
4 // arr[2] = 3
5 // console.log(arr)
6
7 // let arr = [{ 
8 //   fname: 'chirag',
9 //   age: 18
10 // }, {
11 //   fname: 'raj',
12 //   age: 19
13 // }]
14 // console.log(arr)
15
16 var num = [1,2,3,4,5]
17 for(let i =0 ; i <num.length ; i++){
18   console.log(num[i])
19 }
```

Learn With Chirag

0.00 KB/S NEW 4G 47

STDIN

Input for the program (Optional)

Output:

```
1
2
3
4
5
```

index.js + Learn With Chirag 

NEW JAVASCRIPT

```
15
16
17 // for(let i =0 ; i <num.length ; i++){
18 //   console.log(num[i])
19 // }
20
21 var num = [1,2,3,4,5]
22 var a = num.map((n)=>{
23   return n+1
24 })
25
26
27 // function fun(n){
28 //   return n+1
29 // }
30
31 console.log(a)
32 console.log(num)
33
34
35 array.map(function(currentValue, index, arr))
```

STDIN
Input for the program (Optional)

Output:
[2, 3, 4, 5, 6]
[1, 2, 3, 4, 5]

index.js



Learn With Chirag

NEW

JAVASCRIPT

```
35 // array.map(function(currentValue, index, arr))
36
37
38 // array.filter(function(currentValue, index, arr))
39
40 const ages = [19 , 12 , 23 , 21 , 5]
41 const result = ages.filter(checkAdult)
42
43 function checkAdult(age){
44     return age >=18
45 }
46
47 console.log(ages)
48 console.log(result)
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
```

STDIN

Input for the program (Optional)

Output:

```
[19, 12, 23, 21, 5]
[ 19, 23, 21 ]
```

10:38 index.js

```
1 Let arr = [1,2,3,4,5]
2 Let sum = arr.reduce(addNum)
3
4 function addNum(a,b){
5     console.log(a)
6     console.log(b)
7
8     console.log()
9
10    return a + b
11 }
12
13 console.log(sum)
14 // array.reduce(function(total, currentValue, currentIndex, arr))
```

Learn With Chirag



115
KB/S

NEW
ITEM

5G
88

JAVASCRIPT

STDIN

Input for the program (Optional)

Output:

1
2
3
3
6
4
10
5
15

6:13

```
index.js    +   
Learn With Chirag   
13 // function addsquare(a,b){  
14 //   return square(add(a,b))  
15 // }  
16  
17 // const result = add(2,3)  
18 // console.log(square(result))  
19  
20 // console.log(addsquare(3,4))  
21  
22 function compose(f1 , f2){  
23   return function(a,b){  
24     return f2(f1(a,b))  
25   }  
26 }  
27  
28 const composeTwo = (f1,f2) => (a,b) => f2(f1(a,b))  
29  
30 const result = compose(add , multTwo)  
31 console.log(result(2,3))  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42
```

3.00 KB/S NEW
LTE 5G

92



5G



92



5G



92

STDIN

Input for the program (Optional)

Output:

10

6:14 ind

Learn With Chirag

16.0 KB/S 5G 92%

```
1 function add(a , b){  
2     return a +b  
3 }  
4  
5 function multwo(val){  
6     return val*2  
7 }  
8  
9 function square(val){  
10    return val*val  
11 }  
12  
13 const result = add(2,3)  
14 console.log(square(result))
```

STDIN

Input for the program (Optional)

Output:

25

index.js + Learn With Chirag 

NEW JAVASCRIPT

```
27 // const composeTwo = (f1,f2, f3) => (a,b) => f3(f2(f1(a,b)))
28
29
30 // const result = composeTwo(add , multTwo , square)
31 // console.log(result(4,5))
32
33 function composeAll(...funcs){
34   return function(...values){
35     return funcs.reduce((val , fn) => fn(val) , values)
36   }
37 }
38
39 const composeAll =
40   (...funcs) =>
41   (...values) =>
42     funcs.reduce((val , fn) => fn(val) , values)
43
44 const result = composeAll(add , multTwo , square)
45 console.log(result(4,6))
46
47
48
49
50
51
52
53
54
55
56
```

STDIN
Input for the program (Optional)

Output:
400

index.js + Learn With Chirag 

STDIN
Input for the program (Optional)

Output:

```
1 // memoization
2
3 function square(n){
4     return n*n
5 }
6
7 function memoize(func){
8     let cache = {}
9     return function(...args){
10        let n = args[0]
11        if(n in cache){
12            return cache[n]
13        }else{
14            let result = func(n)
15            cache[n] = result
16            return result
17        }
18    }
19 }
20
21 console.time()
22 // console.log(square(5))
23 let effResult = memoize(square)
24 console.log(effResult(5))
25 console.timeEnd()
26
27 console.time()
28 // console.log(square(5))
29 console.log(effResult(5))
30 console.timeEnd()
```

□ ⚙️ Add Two Promises □ Run

index.js x +

index.js > ...

```
1 // synchronous = Executes line by line consecutively in a sequential manner.  
2 //           Code that waits for an operation to complete.  
3  
4 // asynchronous = Allows multiple operations to be performed concurrently  
5 //           without waiting.  
6 //           Doesn't block the execution flow and allows the program to  
7 //           continue.  
8 //           (I/O operations, network requests , fetching data)  
9 //           Handled with Callbacks, Promises , Async/Await  
10  
11 setTimeout(function(){  
12   console.log("Task-1"), 3000  
13 } )  
14 console.log("Task-2")  
15 console.log("Task-3")
```

Console 28c

Run

Task-1
Task-2
Task-3

Shell 78c

Run

Task-2
Task-3
Task-1

AI {JavaScript}

Ln 11, Col 10 (6 chars) • Spaces: 2 History

A screenshot of a code editor interface, likely Visual Studio Code, displaying a file named `index.js`. The code implements a promise that logs "async Task" to the console after a two-second delay, and then logs "Promise Resolved". The editor shows line numbers from 1 to 14. A green "Run" button is visible in the top right. The bottom status bar indicates the code is in JavaScript mode, has 131 characters, and 2 spaces.

```
// Promises
const myPromise = new Promise(function(resolve , reject){
    setTimeout(function(){
        console.log("async Task")
        resolve()
    },2000)
})
myPromise.then(function(){
    console.log("Promise Resolved")
})

```

The right side of the interface shows the `Console` tab with the output:

```
async Task
Promise Resolved
```

index.js

```
8 // })
9
10 // myPromise.then(function( ){
11 //   console.log("Promise Resolved")
12 // })
13
14 const myPromise = new Promise(function(resolve , reject){
15   let fileLoader = false
16   if(fileLoader){
17     resolve("File Loaded")
18   }else{
19     reject("File Not Loaded")
20   }
21 })
22
23 myPromise.then(function(value){
24   console.log(value)
25 }).catch(error => console.log(error))
26
27
```

▶ Run

Console Shell

File Loaded

Run

node:internal/process/promises:289
triggerUncaughtException(
mPromise */);
^

[UnhandledPromiseRejection: This error is thrown by default when a promise is rejected without being handled by throwing inside of an async function, catching it in a catch block, or by rejecting a promise which was not handled with .catch(). The promise rejection reason "File Not Loaded".] {
code: 'ERR_UNHANDLED_REJECTION'
}

Node.js v20.10.0

Run

File Not Loaded

The screenshot shows a code editor interface with a dark theme. On the left, the code editor displays a file named `index.js`. The code contains several lines of JavaScript, including promises and a catch block. A yellow warning icon is visible near the rejection part of the promise chain. The code editor has a status bar at the bottom indicating "Ln 19, Col 30 • Spaces: 2 History". On the right, there is a terminal window showing the execution results. The terminal output includes "File Loaded", an error message about unhandled promise rejection, and "Node.js v20.10.0". The error message details the rejection reason as "File Not Loaded".

```
// })
// myPromise.then(function(){
//   console.log("Promise Resolved")
// })
new Promise((resolve , reject) => {
  let fileLoader = false
  if(fileLoader){
    resolve("File Loaded")
  }else{
    reject("File Not Loaded")
  }
})
myPromise.then(value => console.log(value))
  .catch(error => console.log(error))
```

File Loaded

node:internal/process/promises:289
triggerUncaughtException(
mPromise */);
[UnhandledPromiseRejection: This error
was triggered in an unhandled promise
rejection. If you are not handling a
promise rejection in a catch block, or by
rejecting a promise that was not
handled with .catch(). The promise
was rejected because its reason
"File Not Loaded".] {
 code: 'ERR_UNHANDLED_REJECTION'
}

Node.js v20.10.0

File Not Loaded

□ ⚙️ Add Two Promises □

▶ Run

index.js x +

index.js > ...

```
1 // Async/Await = Async = makes a function return a promise
2 //           Await = makes an async function wait for a promise
3
4
5 v function loadFile(){
6 v   return new Promise((resolve, reject) => {
7   let fileLoader = false
8 v   if (fileLoader) {
9     resolve("File Loaded")
10 v } else {
11   reject("File Not Loaded")
12 }
13 })
14 }
15
16 v async function myFunction(){
17 v   try{
18     const value = await loadFile()
19     console.log(value)
20 v } catch(error){
21   console.log(error)
}
```

AI { } JavaScript

Ln 28, Col 1 • Spaces: 2 History

>_ Console x Shell x +

Run 501

File Loaded

Run 423

node:internal/process/promises:289
triggerUncaughtException(
mPromise *);
^

[UnhandledPromiseRejection: This error
was triggered because it did not have
a handler assigned to it in the promise
chain. If you intended to handle this
error, either assign a catch() block
to it, or by rejecting a promise
that was not handled with .catch(). The promise
that triggered this error was rejected
with reason "File Not Loaded".] {
code: 'ERR_UNHANDLED_REJECTION'
}

Node.js v20.10.0

Run 324

File Not Loaded

A screenshot of the Visual Studio Code (VS Code) interface. The left side shows a code editor with a file named "index.js". The code uses the `setInterval` function to log "Hello Everyone" to the console every 3000 milliseconds. The right side shows a terminal window with three tabs: "Console", "Shell", and another "Console" tab. The "Console" tab shows the output "Hello Everyone" twice, indicating the code is running.

```
1 // setTimeout() = function in JavaScript that allows you to schedule the
2 // execution of a function after an amount
3 // of time (milliseconds)
4
5 // setTimeOut(callback, delay)
6
7 // clearTimeout() = can cancel a timeout before it triggers
8
9
10 const timeoutID = setTimeout(function(){
11     console.log("Hello Everyone")
12 }, 3000)
13
14 clearTimeout(timeoutID)
```

Console Output:

```
Hello Everyone
Hello Everyone
```

The screenshot shows a code editor interface with a dark theme. At the top, there's a toolbar with icons for file operations, a dropdown labeled "Add Two Promises", and a "Run" button. Below the toolbar, there are tabs for "index.js" and "index.js > ...". The main area contains the following code:

```
1 // setTimeout() = function in JavaScript that allows you to schedule the
2 // execution of a function after an amount
3 // of time (milliseconds)
4
5 // setTimeOut(callback, delay)
6
7 function greet(){
8     console.log("Hello Everyone")
9 }
10
11
12 setTimeout(greet, 3000)
```

The code defines a function named "greet" which logs "Hello Everyone" to the console. It then uses the `setTimeout` function to call "greet" after a delay of 3000 milliseconds (3 seconds). The code editor has a status bar at the bottom indicating "Results of your code will appear here".

A screenshot of a code editor (VS Code) showing a JavaScript file named `index.js`. The code uses `setInterval` to log "Learn With Chirag" to the console every second, and `setTimeout` to clear this interval after 5 seconds.

```
// setInterval
function test(){
    console.log("Learn With Chirag")
}
const stop = setInterval(test,1000)
setTimeout(()=>{
    clearInterval(stop)
},5000)
```

The code editor interface includes tabs for `index.js`, `Console`, and `Shell`. The `Console` tab shows the output of the code, which consists of seven lines of "Learn With Chirag" followed by four more lines after the interval is cleared.

```
new Promise(executor)
```

state : "pending"
result : undefined

state : "fulfilled"
result : value

resolve(value)

reject(error)

state : "rejected"
result : error

The screenshot shows a code editor interface with a dark theme. On the left, there is a file tree with two files: 'index.js' and 'index.js > myMap'. The main editor area contains the following code:

```
1 let myMap = new Map([
2   ['name', 'Chirag'],
3   [true, 'Boolkey'],
4   [1, 'Numberkey'],
5 ]
6
7 myMap.set('age', 20)
8 // console.log(myMap)
9
10 // console.log(myMap.get('name'))
11 // console.log(myMap.has('gender'))
12 myMap.delete('name')
13 console.log(myMap)
```

The code uses ES6 syntax to create a Map object with three entries: a string key 'name' with value 'Chirag', a boolean key true with value 'Boolkey', and a number key 1 with value 'Numberkey'. It then sets an 'age' entry with value 20, and finally deletes the 'name' entry and logs the resulting Map object.

On the right side of the interface, there is a 'Run' button and a 'Console' tab. The 'Console' tab displays the output of the code execution:

```
Chirag
false
```

Below the console, there is another 'Run' button and a log entry:

```
Map(3) { true => 'Boolkey', 1 => 'Numberkey', age => 20}
```

8:00 Learn With Chirag

1 Let text = |{
2 "browsers": {
3 "firefox": {
4 "name": "Firefox",
5 "pref_url": "about:config",
6 "releases": {
7 "1": {
8 "release_date": "2004-11-09",
9 "status": "retired",
10 "engine": "Gecko",
11 "engine_version": "1.7"
12 }
13 }
14 }
15 }
16 }
17
18 Let json = JSON.parse(text)
19 console.log(json)
20
21 console.log(JSON.stringify(json))

STDIN
Input for the program (Optional)

Output:
{
 browsers: {
 firefox: { name: 'Firefox', pref_url:
 }
 }
 {"browsers":{"firefox":{"name":"Firefox",

HelloWorld.js

+

Learn With Chirag 

NEW

```
1 // Lodash's _.chunk
2
3 // Lodash - Lodash is a utility library for JavaScript,
4 // providing functions to work with
5 // arrays, objects, strings, etc.
6
7 // // Syntax -
8 //_.chunk(array, size)
9 I
10
11 // Example
12
13 const abc = require("lodash")
14 let arr = [1, 2, 3, 4, 5, 6]
15
16 // Making chunks of size 1
17 console.log(abc.chunk(arr, 1))
```

STDIN

Input for the program (Optional)

Output:

[[1], [2], [3], [4], [5], [6]]

The screenshot shows a code editor interface with two main panes. The left pane displays a code editor with dark mode styling, showing a file named 'ArrayWrapper.js'. The right pane shows a preview of the code execution results.

Code Editor (Left):

```
// Define a Person class
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log(`Hello, my name is ${this.name}!`);
  }
}

// Create some Person objects
let john = new Person('John', 30);
let alice = new Person('Alice', 25);

// Both 'john' and 'alice' can use the 'greet' method
john.greet(); // Output: Hello, my name is John!
alice.greet(); // Output: Hello, my name is Alice!
```

Preview (Right):

115 KB/S 97

Editorial Solutions Submissions

/y Wrapper

es

iyWrapper that accepts an array of integers in its constructor. have two features:

ences of this class are added together with the + operator, the sum of all the elements in both arrays.

g() function is called on the instance, it will return a comma separated string surrounded by brackets. For example, [1, 2, 3].

```
= [[1,2],[3,4]], operation = "Add"

new ArrayWrapper([1,2]);
new ArrayWrapper([3,4]);
// 10

= [[23,98,42,70]], operation = "String"
[23,98,42,70]

new ArrayWrapper([23,98,42,70]);
```

Code

JavaScript Auto

1 // Define a Person class
2 class Person {
3 constructor(name, age) {
4 this.name = name;
5 this.age = age;
6 }
7
8 greet() {
9 console.log(`Hello, my name is \${this.name}!`);
10 }
11 }
12
13 // Create some Person objects
14 let john = new Person('John', 30);
15 let alice = new Person('Alice', 25);
16
17 // Both 'john' and 'alice' can use the 'greet' method
18 john.greet(); // Output: Hello, my name is John!
19 alice.greet(); // Output: Hello, my name is Alice!

Saved

[Editorial](#) | [Solutions](#) | [Submissions](#)

Wrapper

es

iyWrapper that accepts an array of integers in its constructor. It has two features:

ences of this class are added together with the `+` operator, the result will be the sum of all the elements in both arrays.

`g()` function is called on the instance, it will return a comma-separated string surrounded by brackets. For example, `[1, 2, 3]`.

```
= [[1,2],[3,4]], operation = "Add"
```

```
: new ArrayWrapper([1,2]);
: new ArrayWrapper([3,4]);
// 10
```

```
= [[23,98,42,70]], operation = "String"
[[23,98,42,70]]
```

```
: new ArrayWrapper([23,98,42,70]);
```

Code

JavaScript ▾ Auto

```
22 // Let's create a constructor function for creating Person objects
23 function Person(name, age) {
24     this.name = name;
25     this.age = age;
26 }
27
28 // Now, let's add a method to the Person prototype
29 Person.prototype.greet = function() {
30     console.log('Hello, my name is ' + this.name + '!');
31 };
32
33 // Now, let's create some Person objects
34 var john = new Person('John', 30);
35 var alice = new Person('Alice', 25);
36
37 // Both 'john' and 'alice' can use the 'greet' method
38 john.greet(); // Output: Hello, my name is John!
39 alice.greet(); // Output: Hello, my name is Alice!
```

Saved