index.js + 3zymz2ccj NEW

```
1
2  // normal function
3  function f(a,b){
4    const sum = a + b
5    // sum = 3
6    return sum
7  }
8  console.log(f(2,4))
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

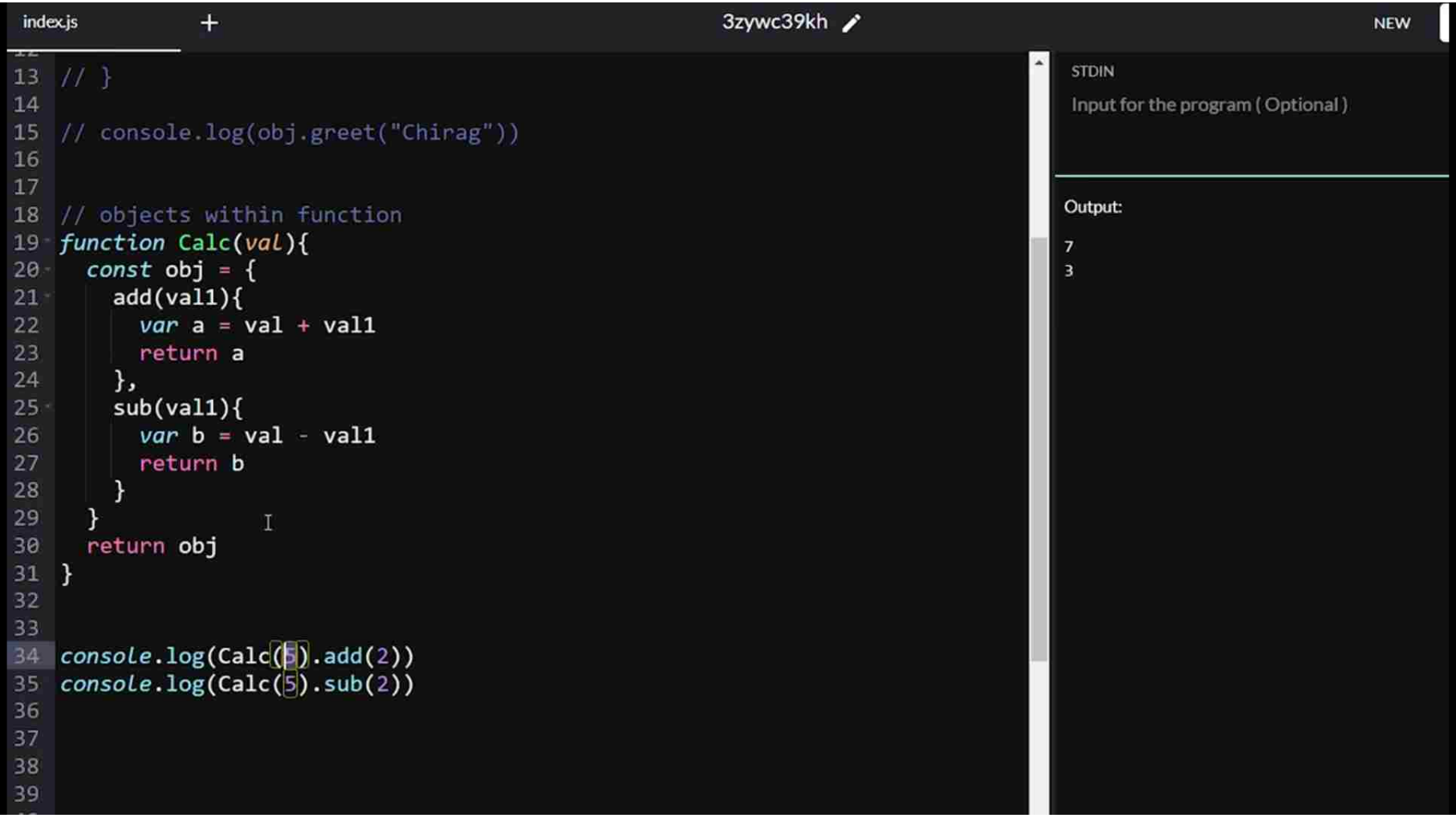
```
1  // Object is a collection of key value pairs
2
3  // Object Literals
4  var obj = {
5    name: "Chirag",
6    age: 18,
7
8    greet(name){
9      return "good morning " + name
10   }
11
12
13 }
14
15 console.log(obj.greet("Chirag"))
16
17
18 // objects within function
19 function Calc(val){
20   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
// }
// console.log(obj.greet("Chirag"))
// objects within function
function Calc(val){
const obj = {
add(val1){
var a = val + val1
return a
},
sub(val1){
var b = val - val1
return b
}
}
return obj
}
console.log(Calc(5).add(2))
console.log(Calc(5).sub(2))
STDIN
Input for the program (Optional)
Output:
7
3

index.js

3zz2dk8fz

NEW

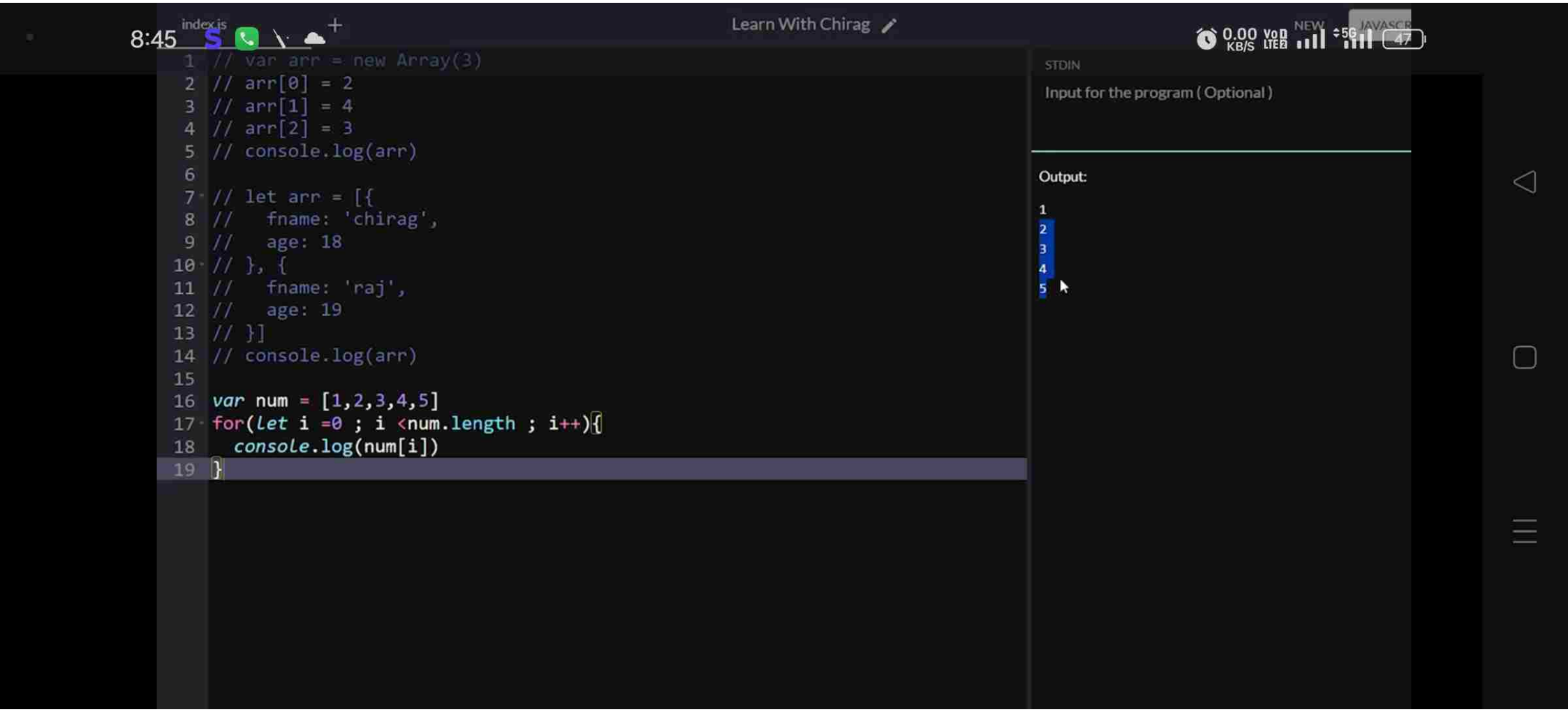
```
// }

// console.log(obj.greet("Chirag"))


// objects within function
function Calc(val){


   function add(val1){
      var a = val + val1
      return a
    }

   function sub(val1){
      var b = val - val1
      return b
    }
    return obj = {
      add , sub
    }

}


console.log(Calc(5).add(3))
console.log(Calc(5).sub(5))
```

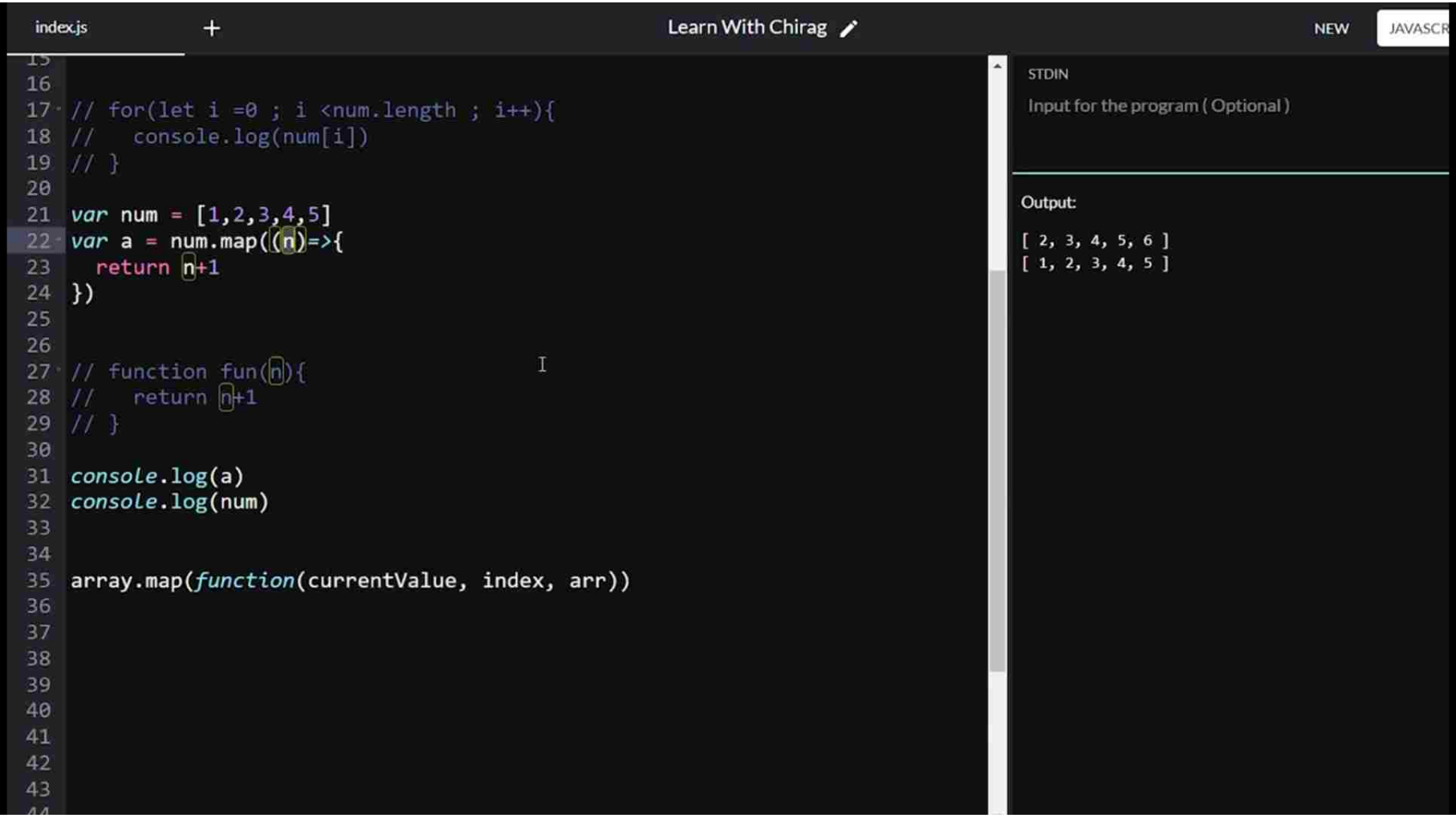
STDIN

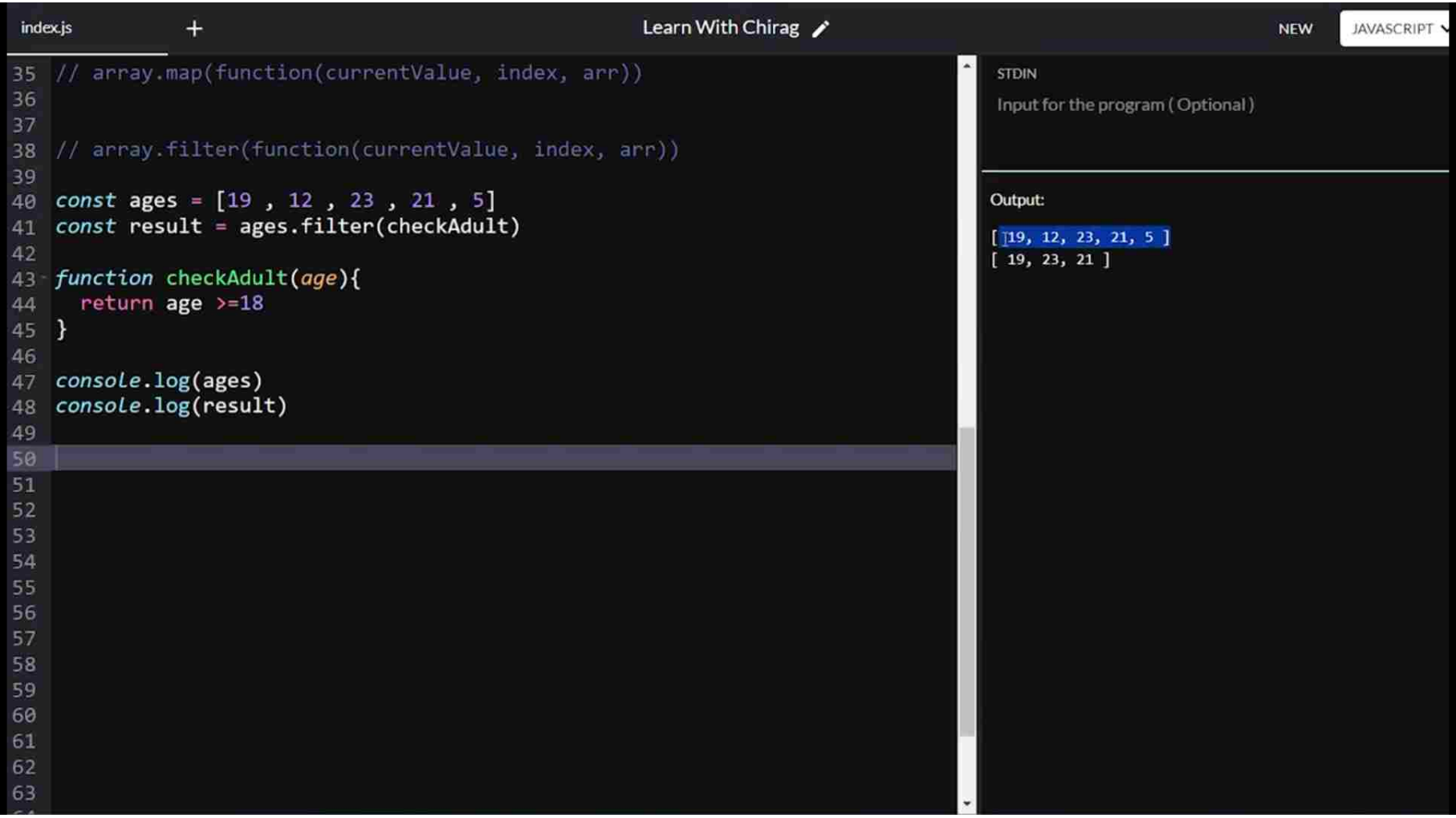
Input for the program (Optional)

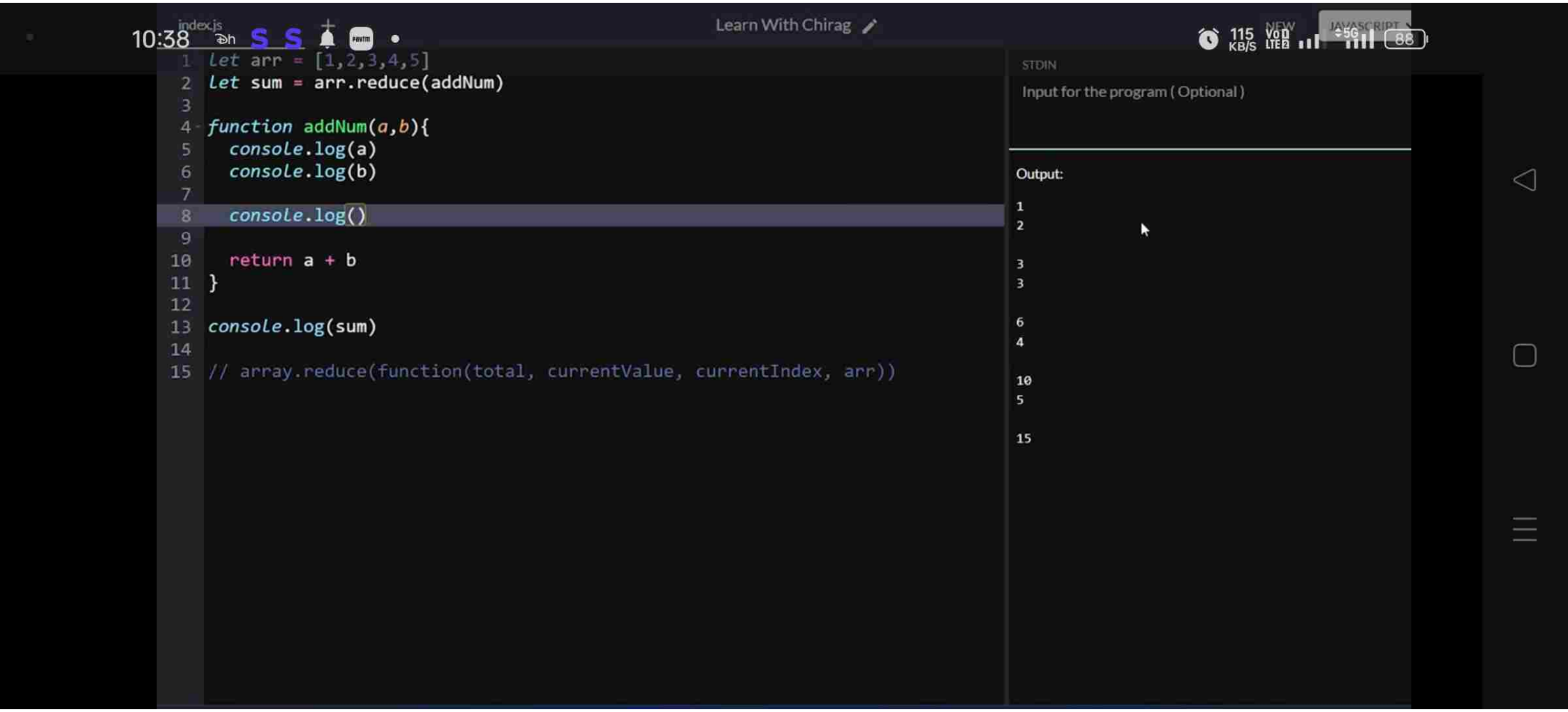
Output:

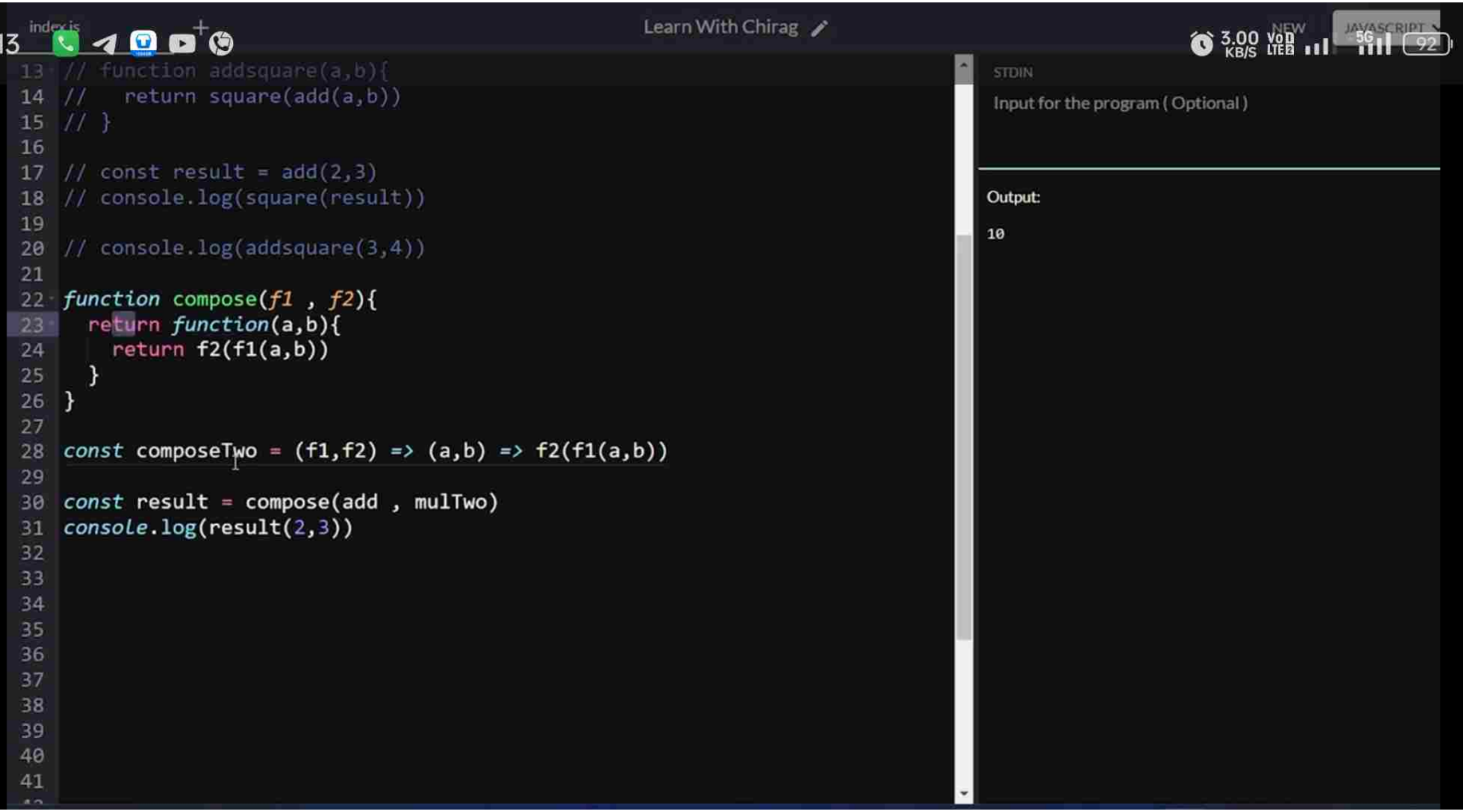
```
8
0
```

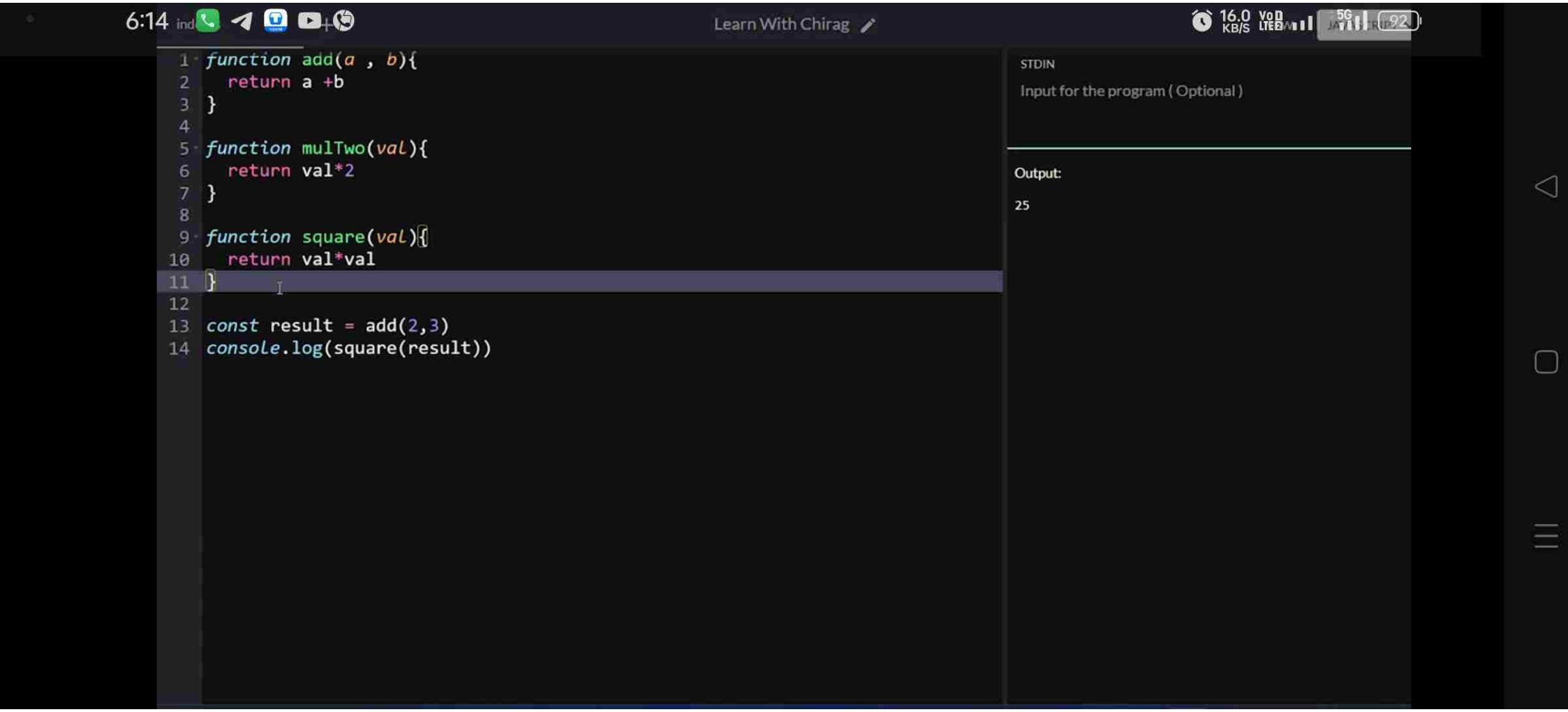

8:45
index.js
Learn With Chirag
0.00 KB/S
NEW
47
// var arr = new Array(3)
// arr[0] = 2
// arr[1] = 4
// arr[2] = 3
// console.log(arr)
// let arr = [{
// fname: 'chirag',
// age: 18
// }, {
// fname: 'raj',
// age: 19
// }]
// console.log(arr)
var num = [1,2,3,4,5]
for(let i =0 ; i <num.length ; i++){
console.log(num[i])
}
STDIN
Input for the program (Optional)
Output:
1
2
3
4
5

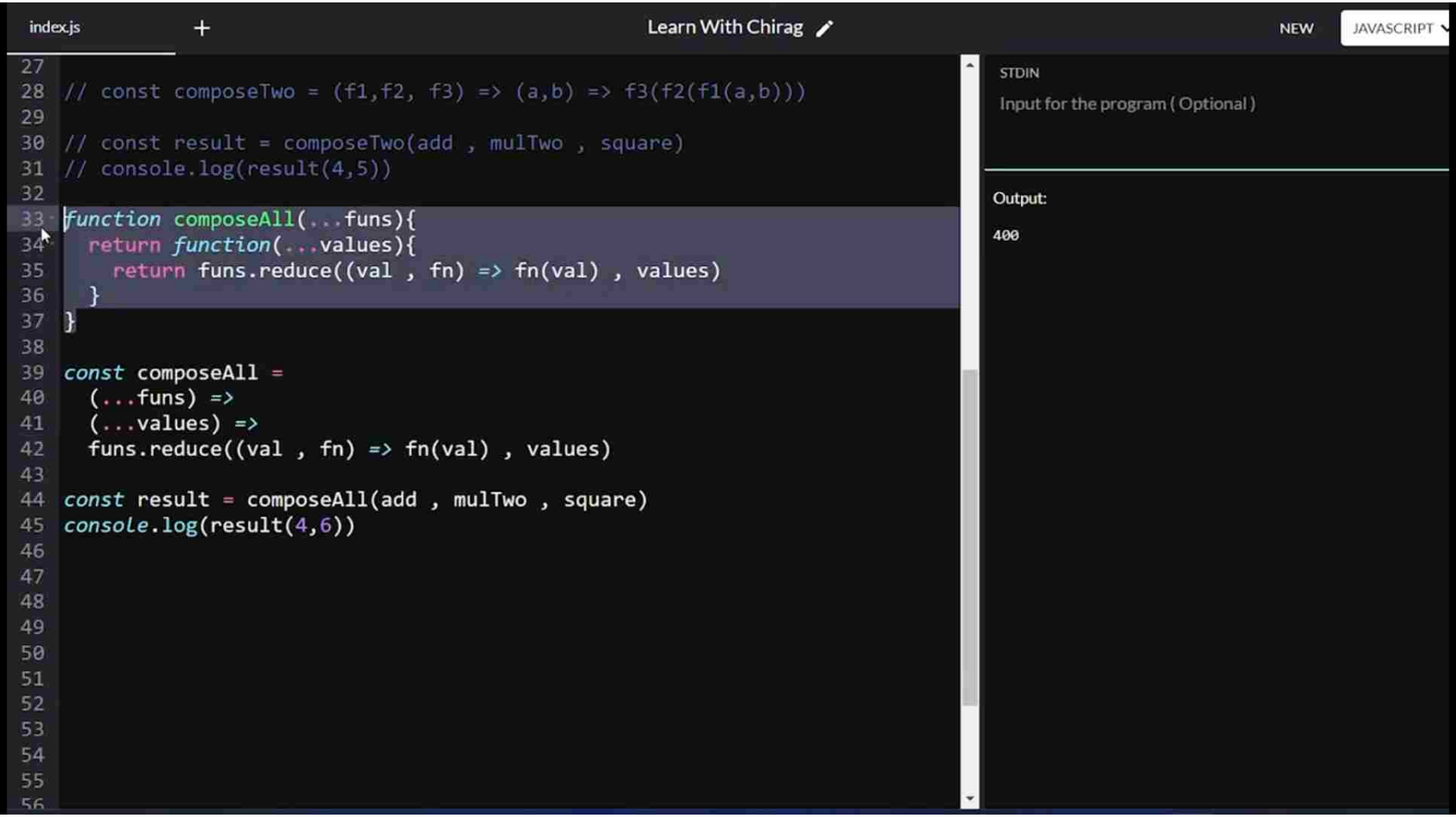

index.js
Learn With Chirag
NEW
JAVASCR
// for(let i =0 ; i <num.length ; i++){
//   console.log(num[i])
// }
var num = [1,2,3,4,5]
var a = num.map((n)=>{
  return n+1
})
// function fun(n){
//   return n+1
// }
console.log(a)
console.log(num)
array.map(function(currentValue, index, arr))
STDIN
Input for the program (Optional)
Output:
[2, 3, 4, 5, 6]
[1, 2, 3, 4, 5]

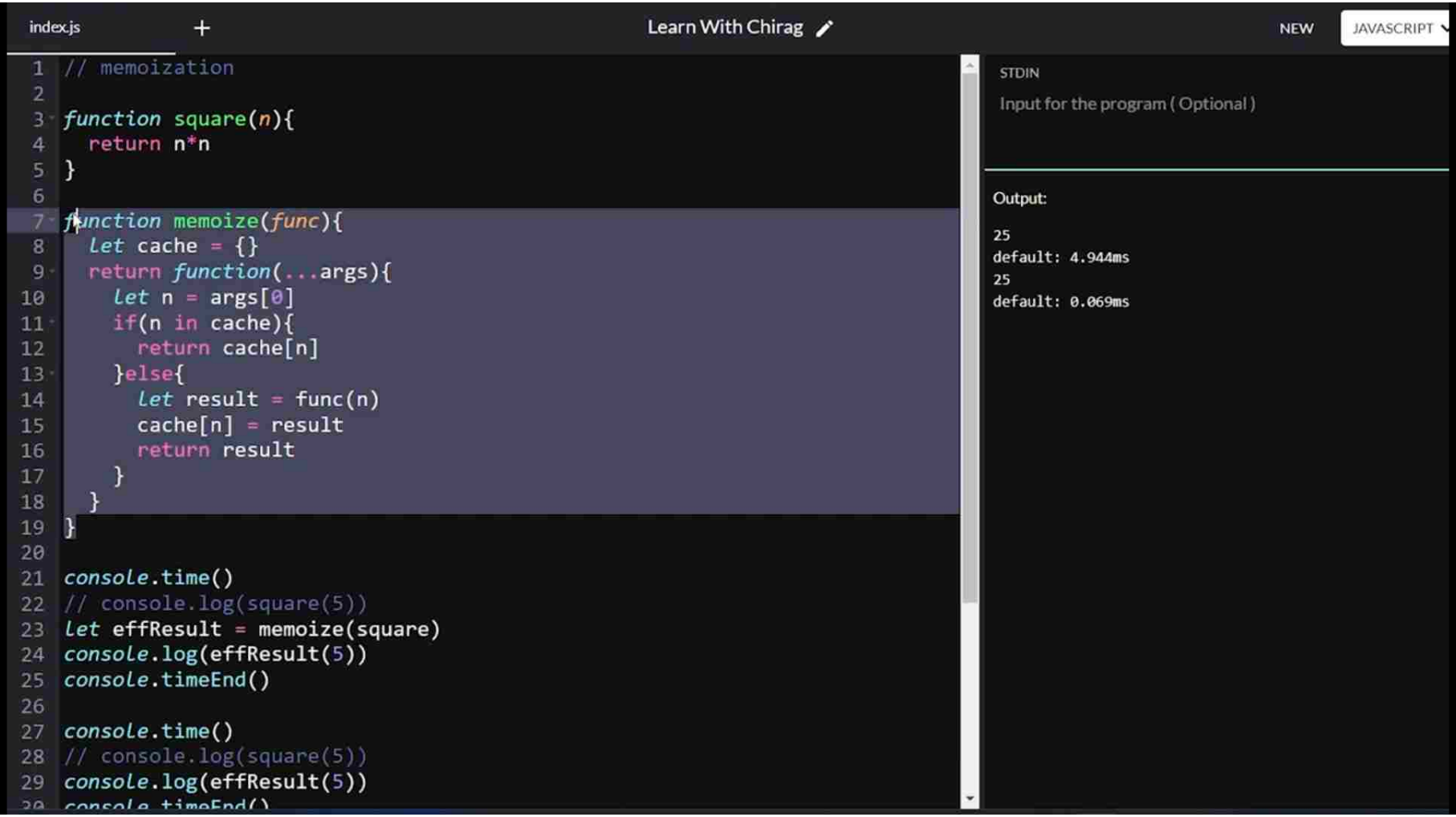

index.js
Learn With Chirag
NEW
JAVASCRIPT
// array.map(function(currentValue, index, arr))
// array.filter(function(currentValue, index, arr))
const ages = [19 , 12 , 23 , 21 , 5]
const result = ages.filter(checkAdult)
function checkAdult(age){
return age >=18
}
console.log(ages)
console.log(result)
STDIN
Input for the program (Optional)
Output:
[19, 12, 23, 21, 5]
[19, 23, 21]

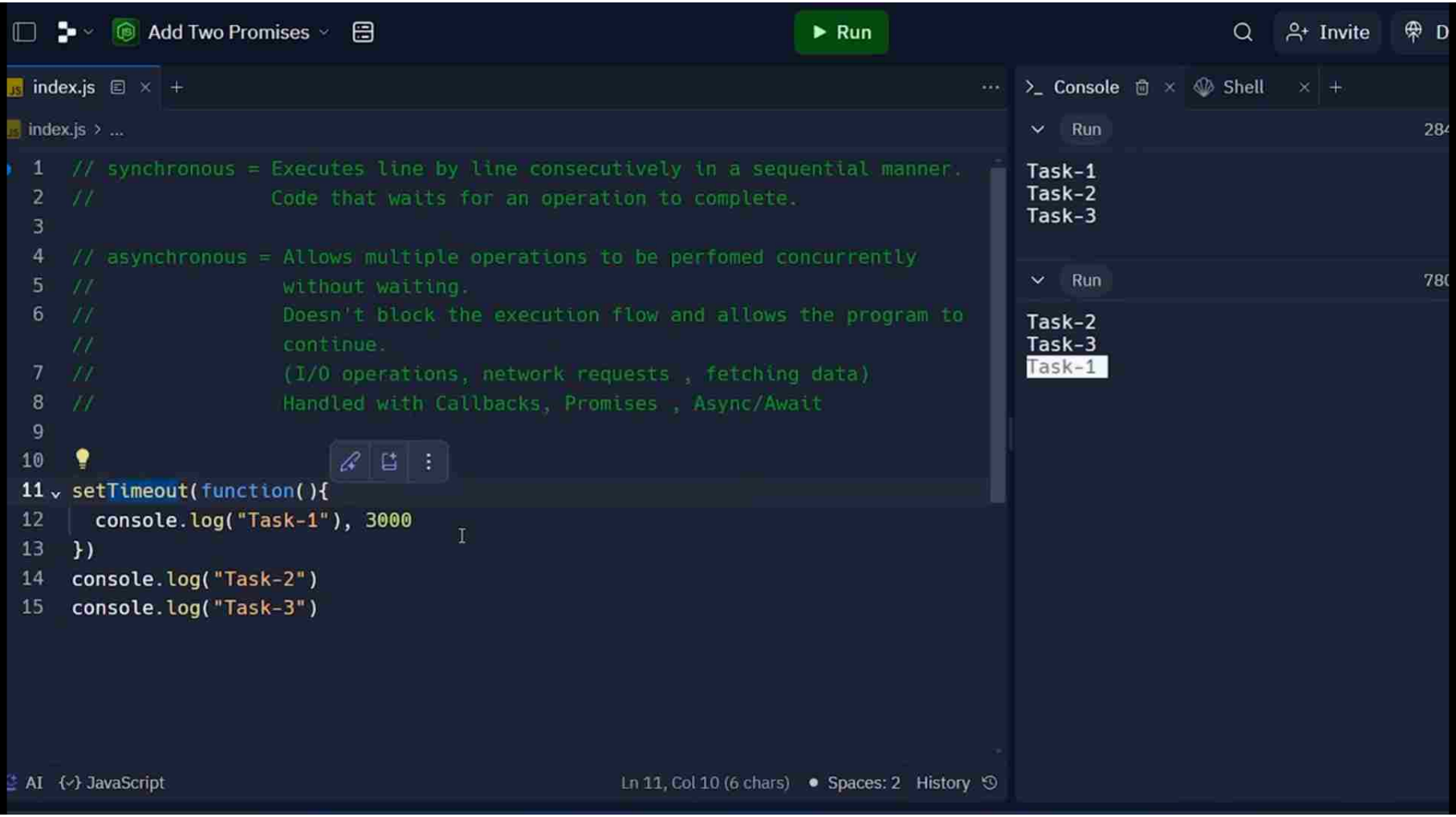

10:38
index.js
Learn With Chirag
115 KB/S
NEW
JAVASCRIPT
88
let arr = [1,2,3,4,5]
let sum = arr.reduce(addNum)
function addNum(a,b){
console.log(a)
console.log(b)
console.log()
return a + b
}
console.log(sum)
// array.reduce(function(total, currentValue, currentIndex, arr))
STDIN
Input for the program (Optional)
Output:
1
2
3
3
6
4
10
5
15

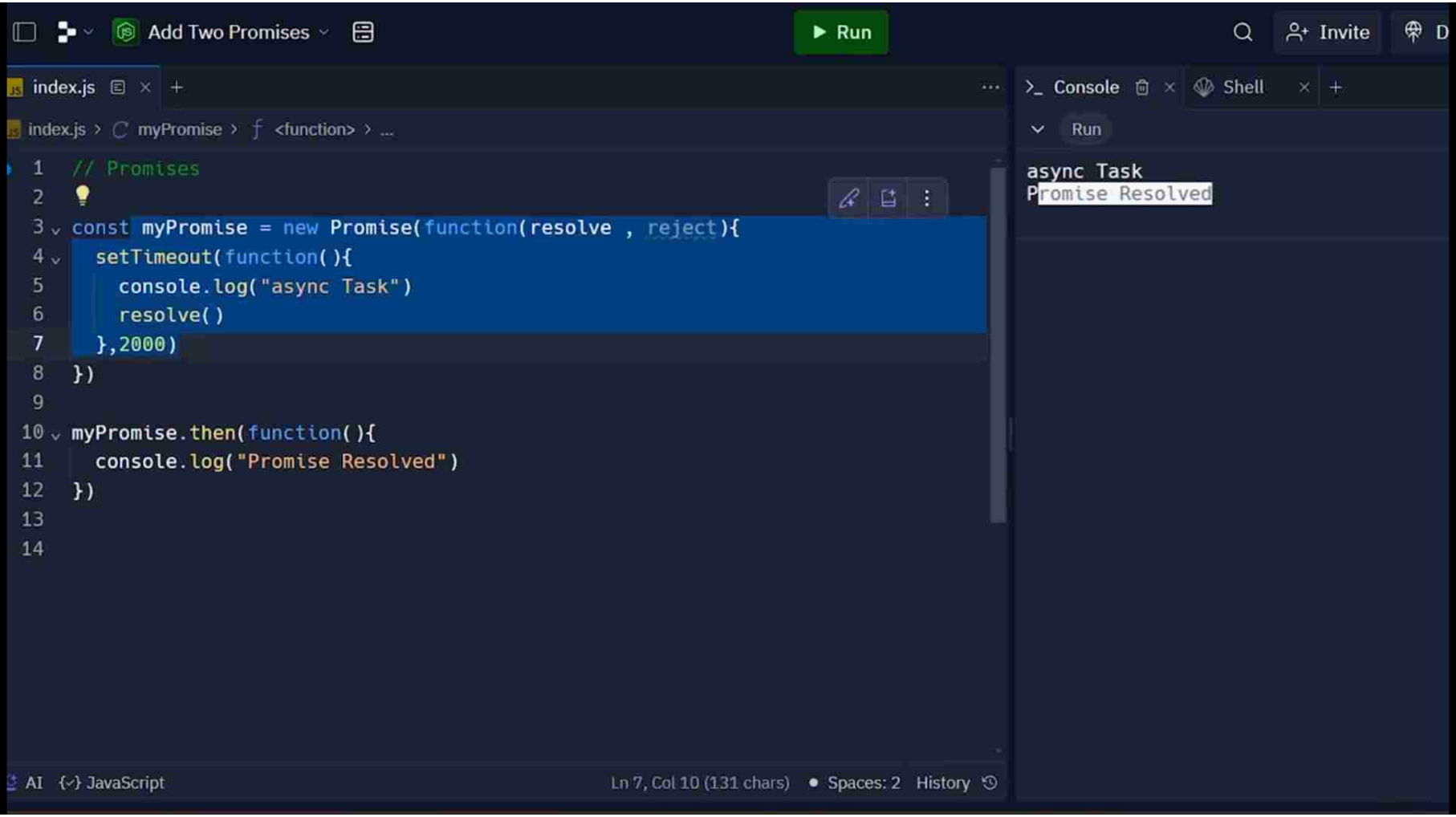

index.js
Learn With Chirag
JAVASCRIPT
// function addsquare(a,b){
//   return square(add(a,b))
// }
// const result = add(2,3)
// console.log(square(result))
// console.log(addsquare(3,4))
function compose(f1 , f2){
return function(a,b){
return f2(f1(a,b))
}
}
const composeTwo = (f1,f2) => (a,b) => f2(f1(a,b))
const result = compose(add , mulTwo)
console.log(result(2,3))
STDIN
Input for the program (Optional)
Output:
10

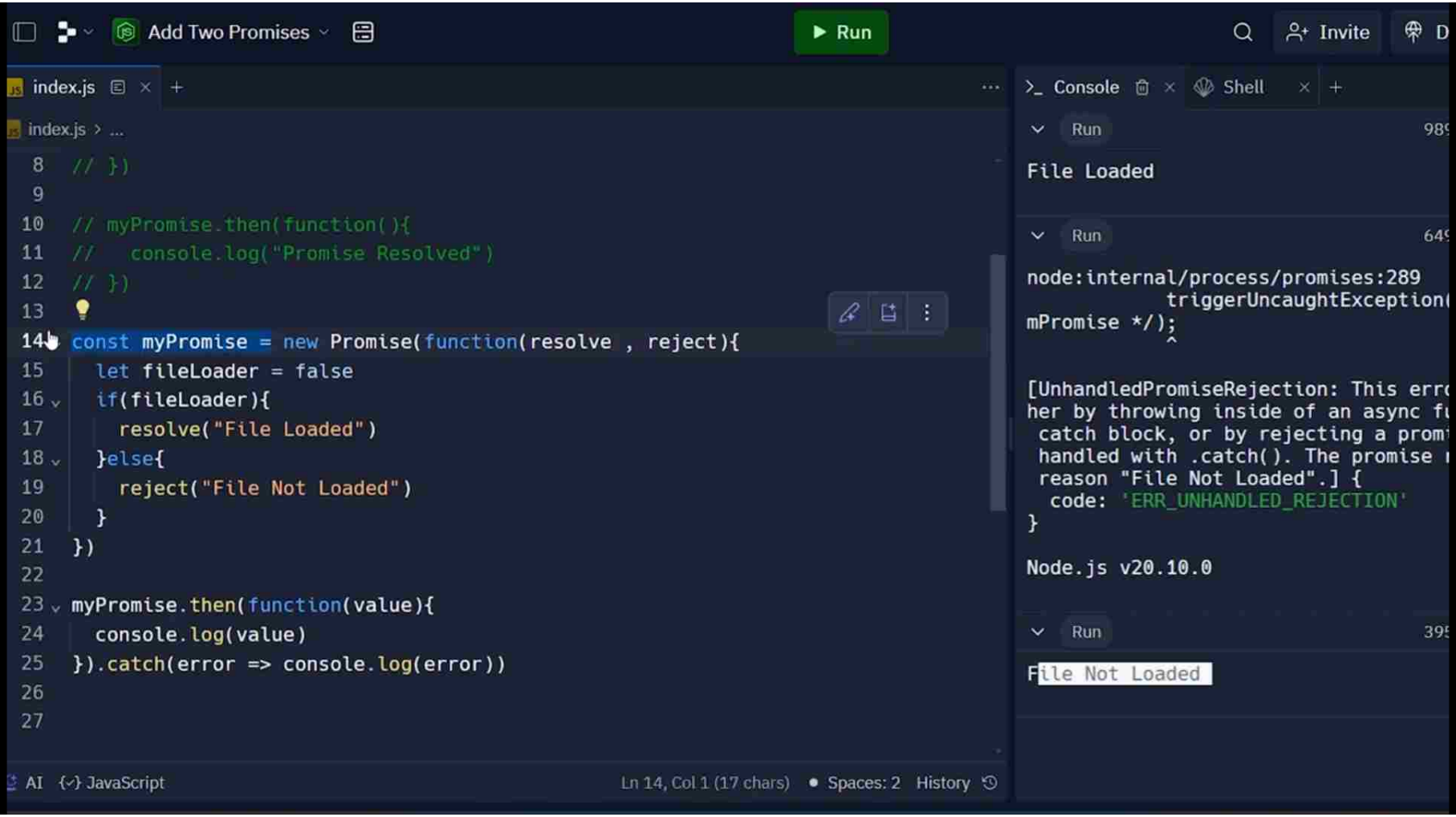

6:14
Learn With Chirag
16.0 KB/S
92
function add(a , b){
return a +b
}
function mulTwo(val){
return val*2
}
function square(val){
return val*val
}
const result = add(2,3)
console.log(square(result))
STDIN
Input for the program (Optional)
Output:
25

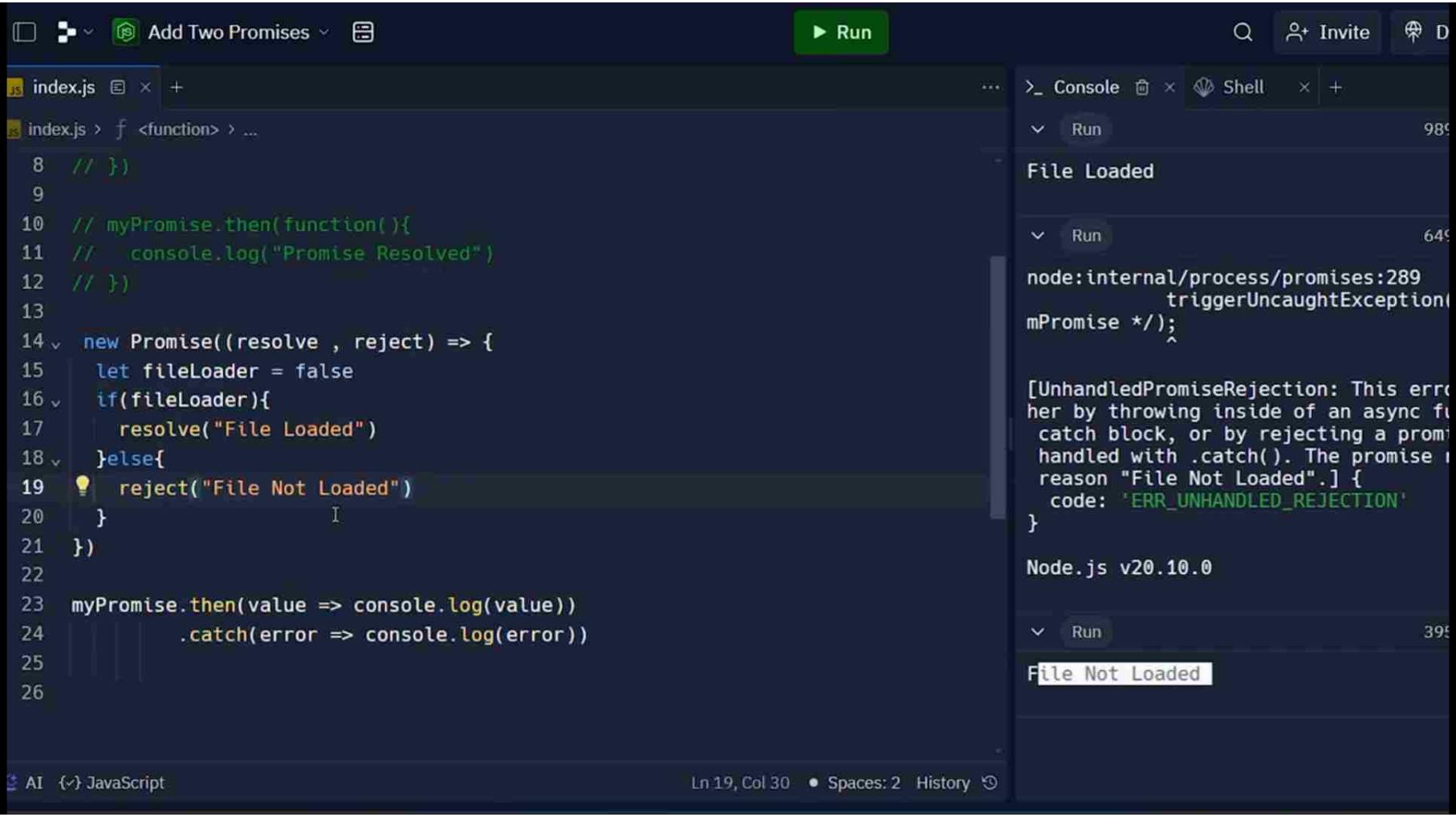

index.js
Learn With Chirag
NEW
JAVASCRIPT
// const composeTwo = (f1,f2, f3) => (a,b) => f3(f2(f1(a,b)))
// const result = composeTwo(add , mulTwo , square)
// console.log(result(4,5))
function composeAll(...funs){
return function(...values){
return funs.reduce((val , fn) => fn(val) , values)
}
}
const composeAll =
(...funs) =>
(...values) =>
funs.reduce((val , fn) => fn(val) , values)
const result = composeAll(add , mulTwo , square)
console.log(result(4,6))
STDIN
Input for the program (Optional)
Output:
400

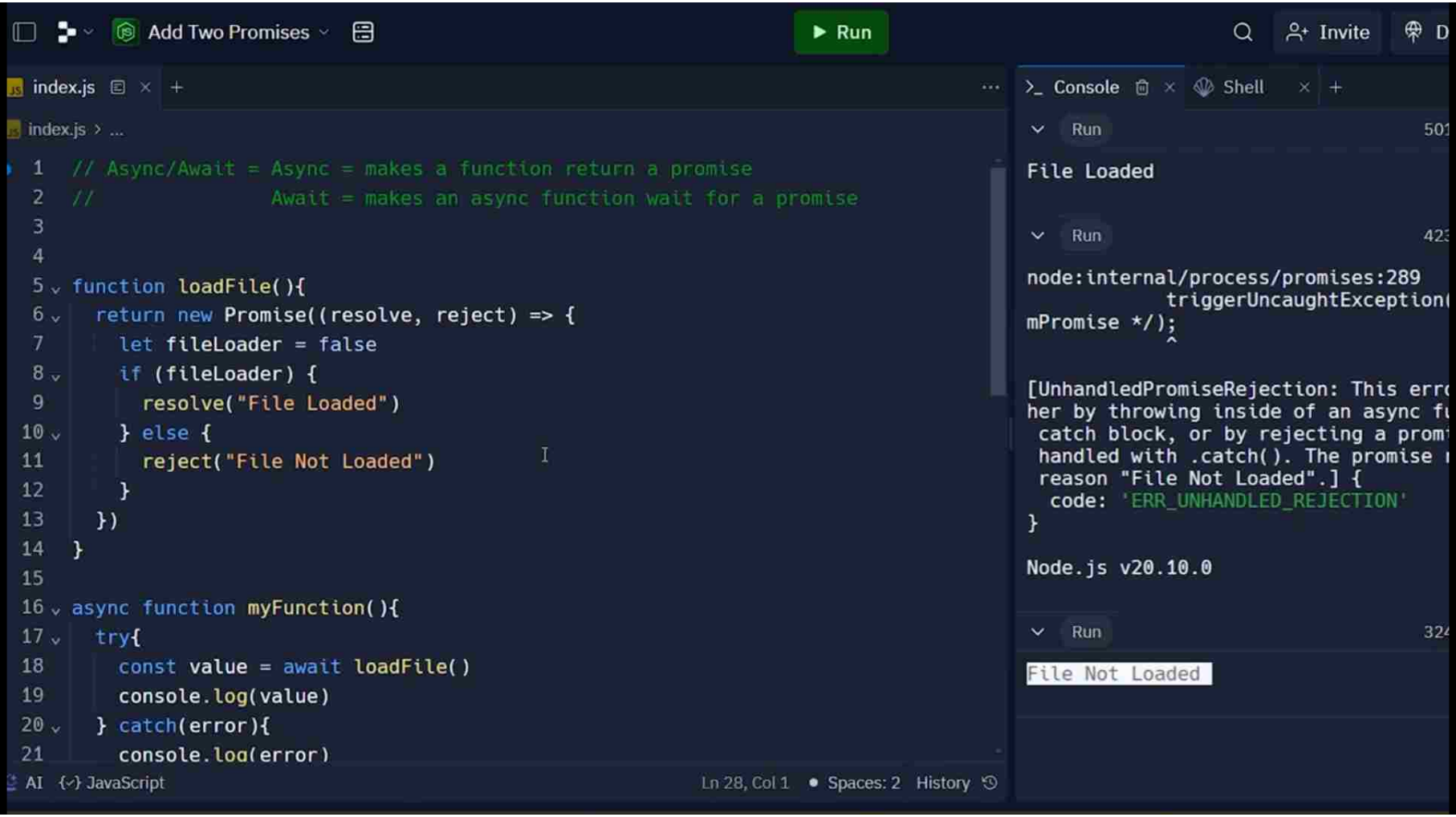

index.js
Learn With Chirag
NEW
JAVASCRIPT
// memoization
function square(n){
return n*n
}
function memoize(func){
let cache = {}
return function(...args){
let n = args[0]
if(n in cache){
return cache[n]
}else{
let result = func(n)
cache[n] = result
return result
}
}
}
console.time()
// console.log(square(5))
let effResult = memoize(square)
console.log(effResult(5))
console.timeEnd()
console.time()
// console.log(square(5))
console.log(effResult(5))
STDIN
Input for the program (Optional)
Output:
25
default: 4.944ms
25
default: 0.069ms

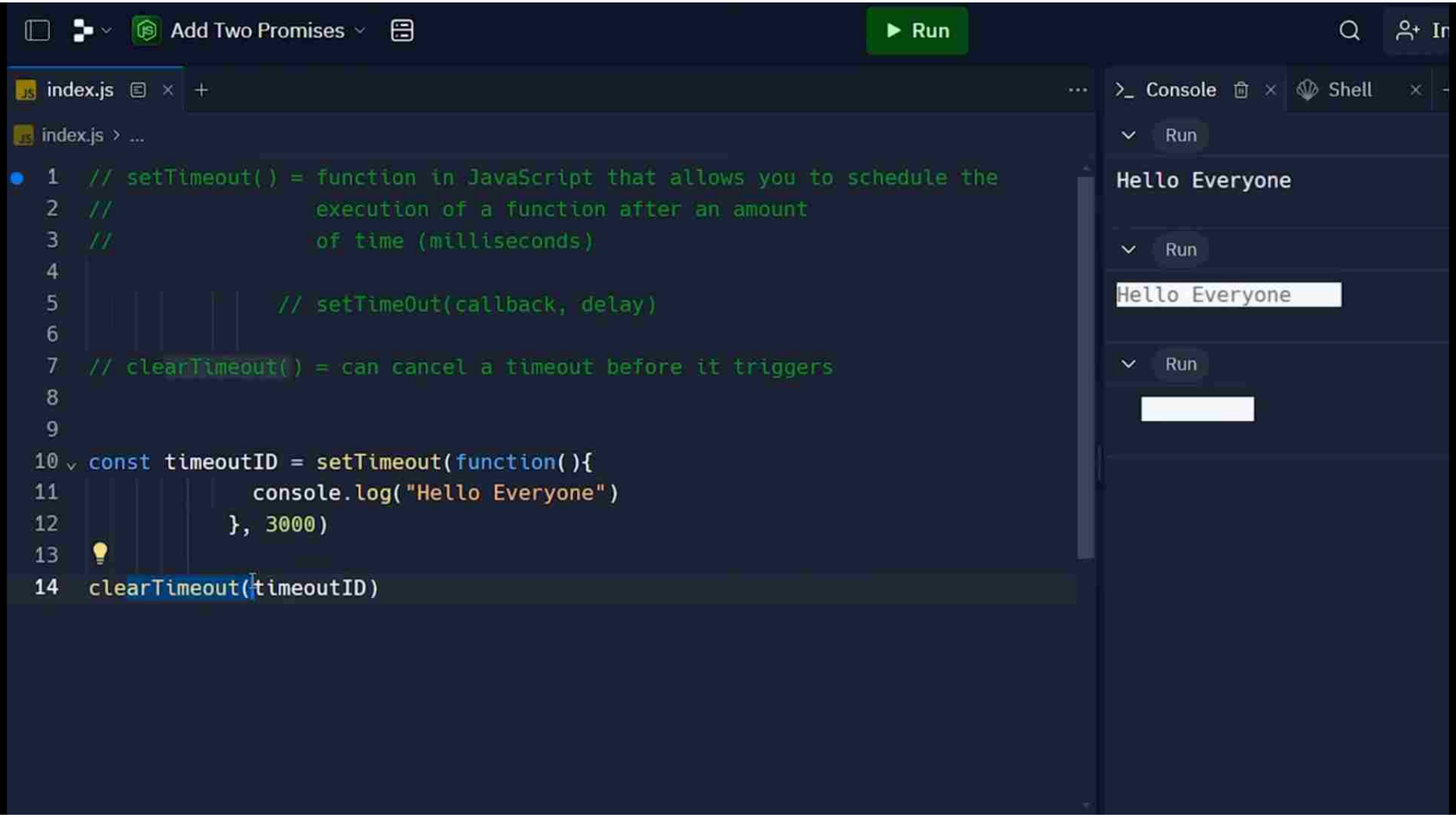

Add Two Promises
Run
Invite
index.js
index.js > ...
// synchronous = Executes line by line consecutively in a sequential manner.
// Code that waits for an operation to complete.
// asynchronous = Allows multiple operations to be perfomed concurrently
// without waiting.
// Doesn't block the execution flow and allows the program to
// continue.
// (I/O operations, network requests , fetching data)
// Handled with Callbacks, Promises , Async/Await
setTimeout(function(){
console.log("Task-1"), 3000
})
console.log("Task-2")
console.log("Task-3")
Console
Shell
Run
Task-1
Task-2
Task-3
Run
Task-2
Task-3
Task-1
AI
JavaScript
Ln 11, Col 10 (6 chars)
Spaces: 2
History

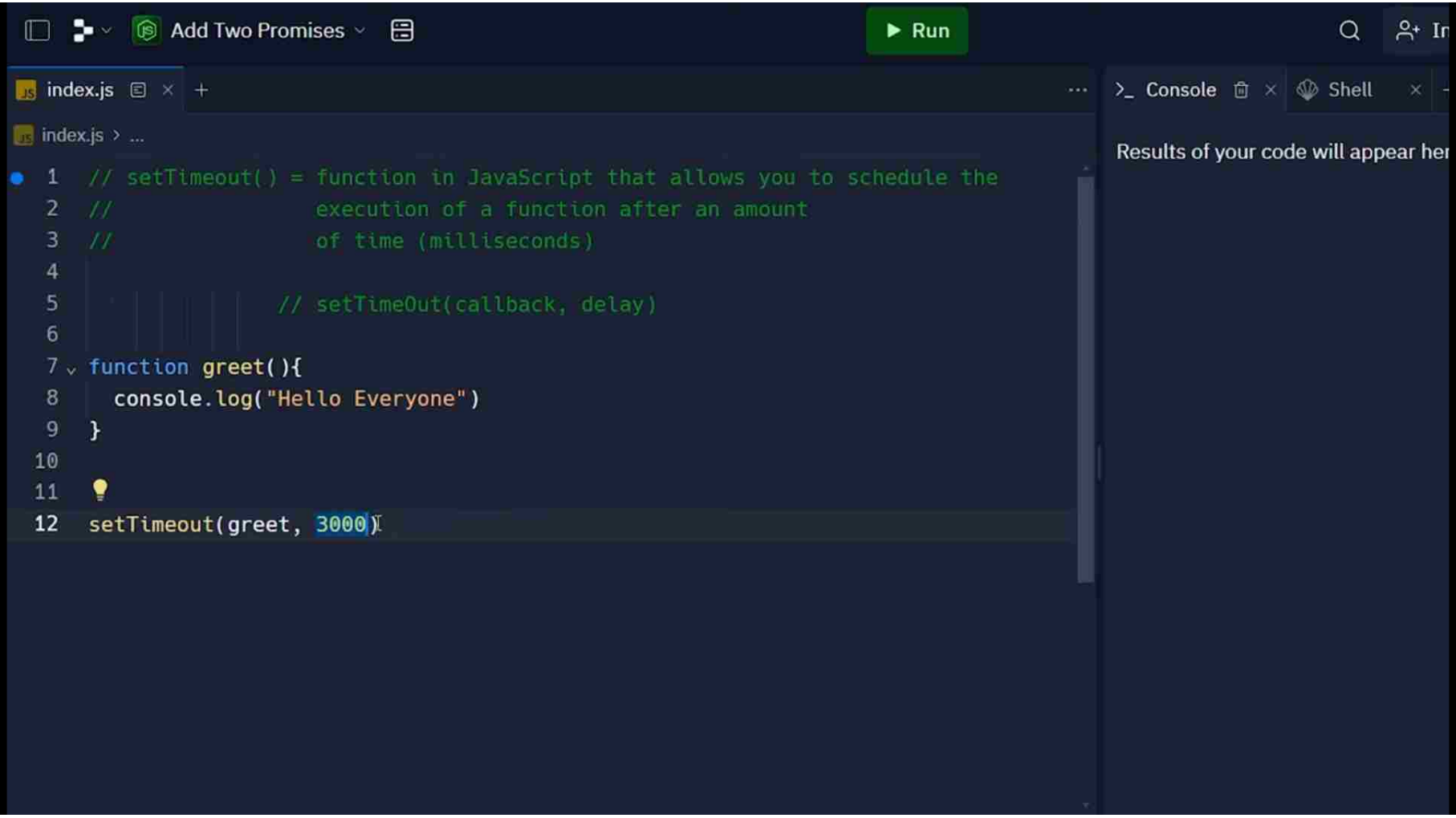

Add Two Promises
Run
Invite
index.js
index.js › myPromise › <function> › ...
1 // Promises
2
3 const myPromise = new Promise(function(resolve , reject){
4 setTimeout(function(){
5 console.log("async Task")
6 resolve()
7 },2000)
8 })
9
10 myPromise.then(function(){
11 console.log("Promise Resolved")
12 })
13
14
Console
Shell
Run
async Task
Promise Resolved
AI
JavaScript
Ln 7, Col 10 (131 chars)
Spaces: 2
History

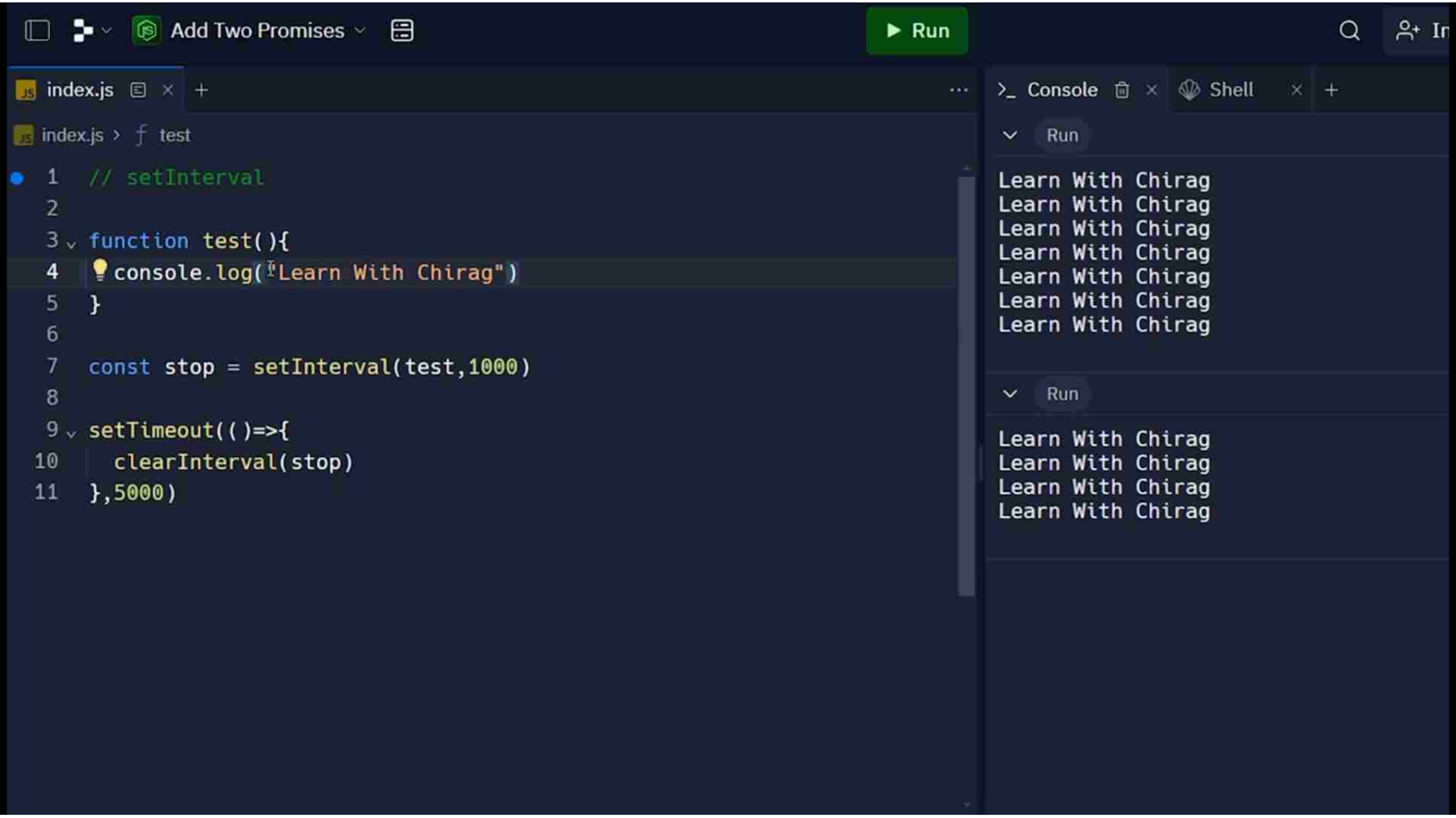

Add Two Promises
Run
Invite
index.js
index.js > ...
// })
// myPromise.then(function(){
// console.log("Promise Resolved")
// })
const myPromise = new Promise(function(resolve , reject){
let fileLoader = false
if(fileLoader){
resolve("File Loaded")
}else{
reject("File Not Loaded")
}
})
myPromise.then(function(value){
console.log(value)
}).catch(error => console.log(error))
AI
JavaScript
Ln 14, Col 1 (17 chars)
Spaces: 2
History
Console
Shell
Run
File Loaded
Run
node:internal/process/promises:289
triggerUncaughtException(
mPromise */);
^
[UnhandledPromiseRejection: This erro
her by throwing inside of an async fu
catch block, or by rejecting a promi
handled with .catch(). The promise r
reason "File Not Loaded".] {
code: 'ERR_UNHANDLED_REJECTION'
}
Node.js v20.10.0
Run
File Not Loaded

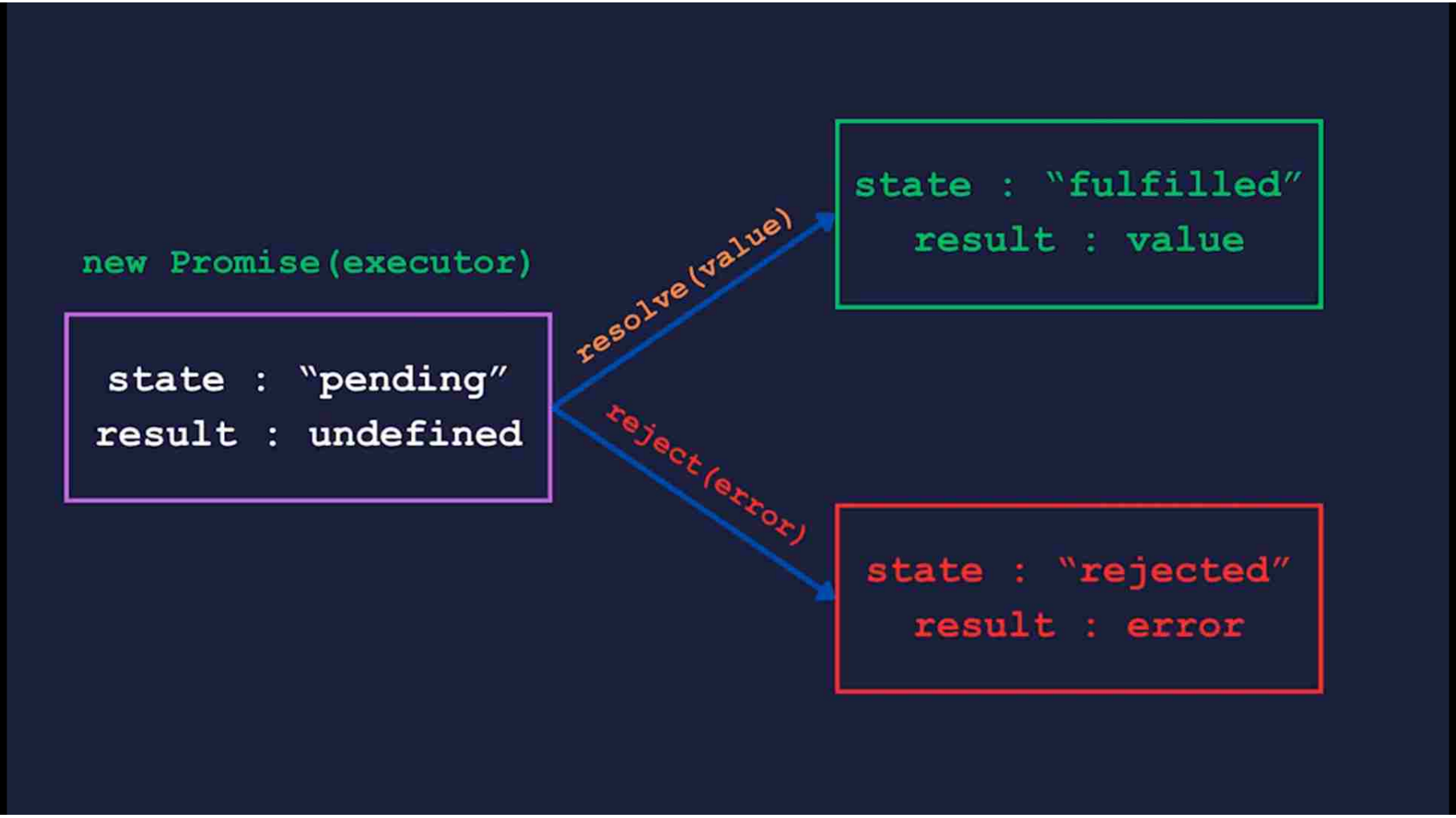

Add Two Promises
Run
Invite
index.js
index.js > <function> > ...
// })
// myPromise.then(function(){
// console.log("Promise Resolved")
// })
new Promise((resolve , reject) => {
let fileLoader = false
if(fileLoader){
resolve("File Loaded")
}else{
reject("File Not Loaded")
}
})
myPromise.then(value => console.log(value))
.catch(error => console.log(error))
AI
JavaScript
Ln 19, Col 30
Spaces: 2
History
Console
Shell
Run
File Loaded
Run
node:internal/process/promises:289
triggerUncaughtException(
mPromise */);
^
[UnhandledPromiseRejection: This erro
her by throwing inside of an async fu
catch block, or by rejecting a promi
handled with .catch(). The promise r
reason "File Not Loaded".] {
code: 'ERR_UNHANDLED_REJECTION'
}
Node.js v20.10.0
Run
File Not Loaded

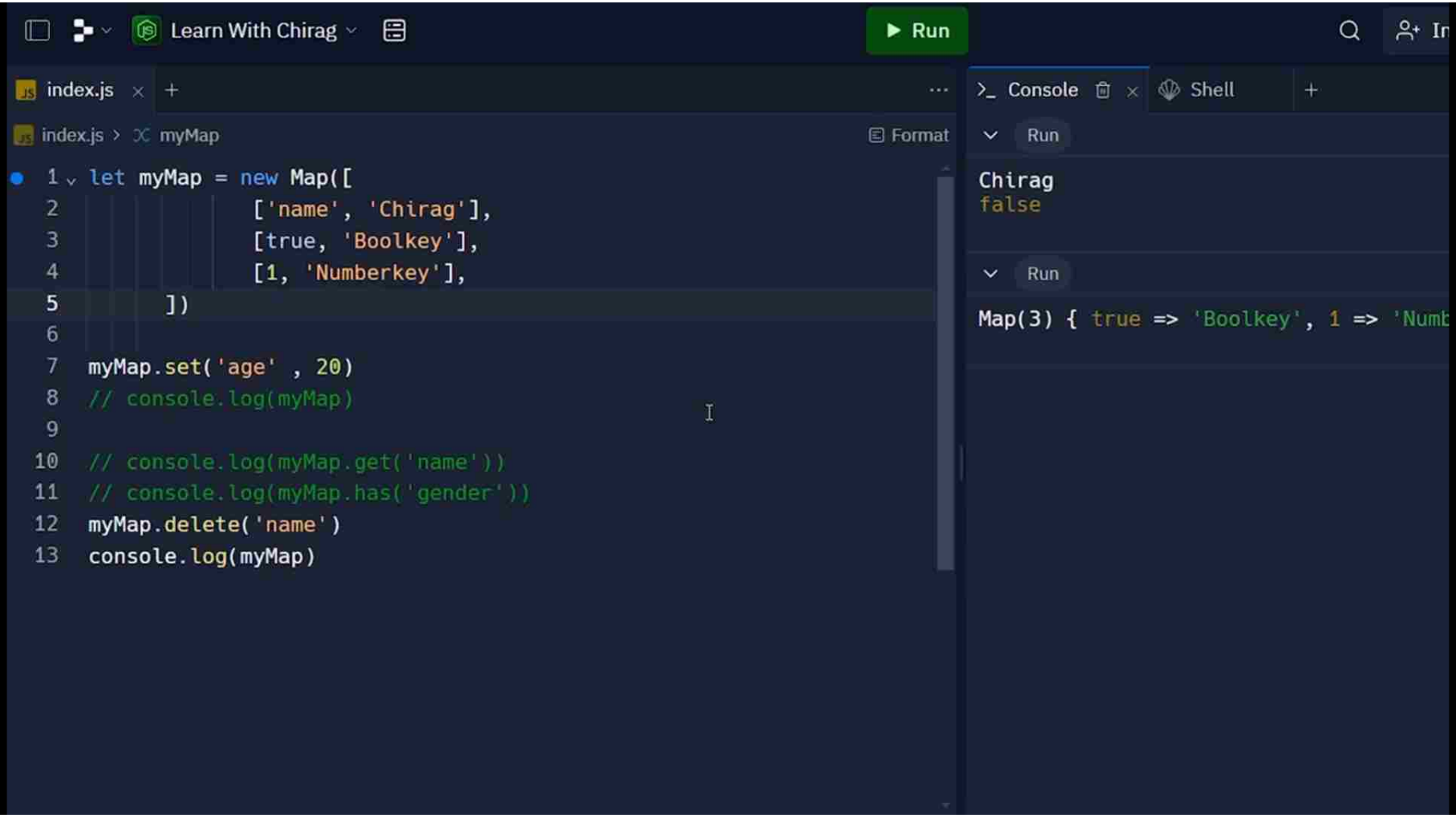

Add Two Promises
Run
Invite
index.js
index.js › ...
// Async/Await = Async = makes a function return a promise
//   Await = makes an async function wait for a promise
function loadFile(){
return new Promise((resolve, reject) => {
let fileLoader = false
if (fileLoader) {
resolve("File Loaded")
} else {
reject("File Not Loaded")
}
})
}
async function myFunction(){
try{
const value = await loadFile()
console.log(value)
} catch(error){
console.log(error)
AI
{-} JavaScript
Ln 28, Col 1
Spaces: 2
History
Console
Shell
Run
File Loaded
Run
node:internal/process/promises:289
triggerUncaughtException(
mPromise */);
^
[UnhandledPromiseRejection: This erro
her by throwing inside of an async fu
catch block, or by rejecting a promi
handled with .catch(). The promise r
reason "File Not Loaded".] {
code: 'ERR_UNHANDLED_REJECTION'
}
Node.js v20.10.0
Run
File Not Loaded

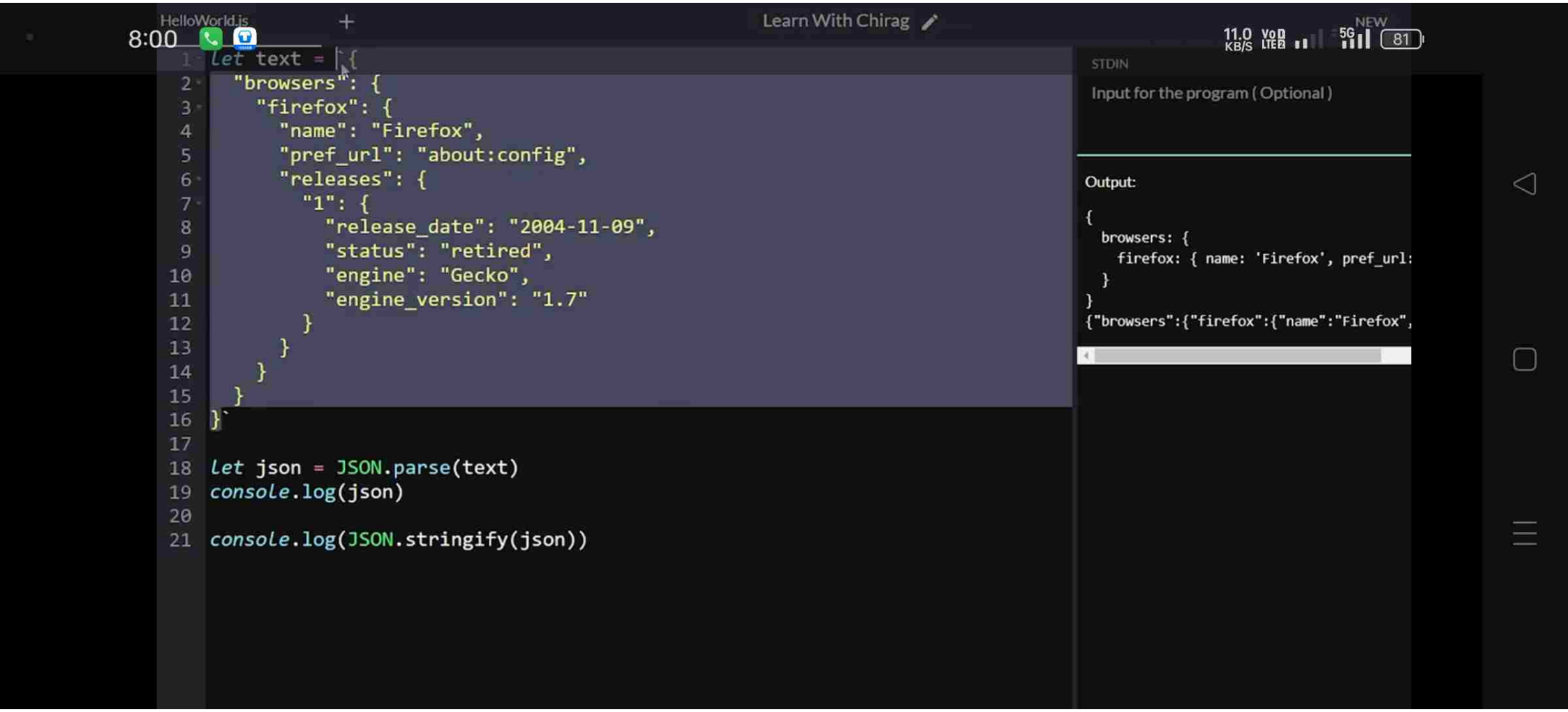

Add Two Promises
Run
index.js
index.js > ...
1 // setTimeout() = function in JavaScript that allows you to schedule the
2 // execution of a function after an amount
3 // of time (milliseconds)
4
5 // setTimeOut(callback, delay)
6
7 // clearTimeout() = can cancel a timeout before it triggers
8
9
10 const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

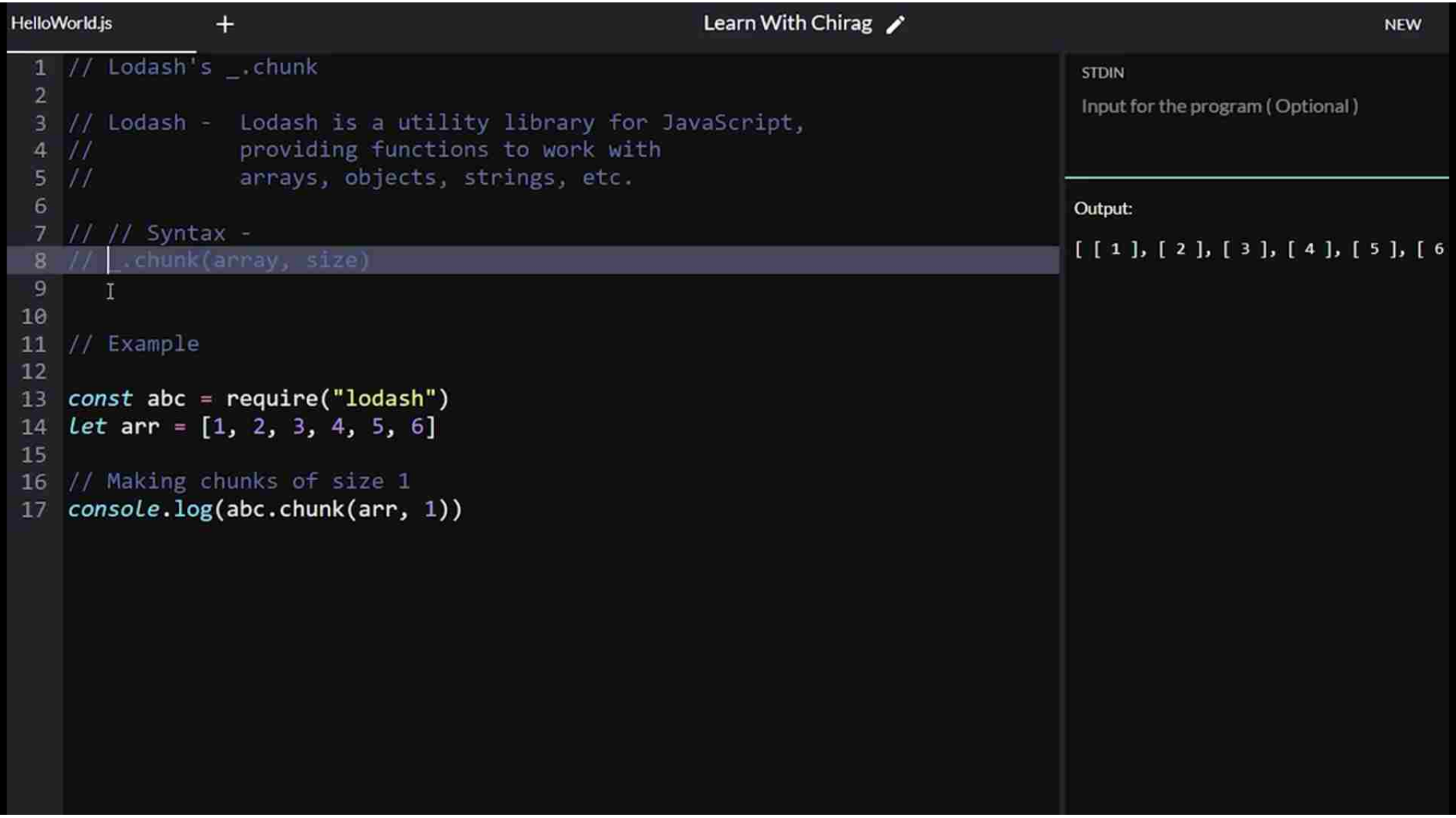

Add Two Promises
Run
index.js
index.js > ...
// setTimeout( ) = function in JavaScript that allows you to schedule the
// execution of a function after an amount
// of time (milliseconds)
// setTimeOut(callback, delay)
function greet(){
console.log("Hello Everyone")
}
setTimeout(greet, 3000)
Console
Shell
Results of your code will appear her

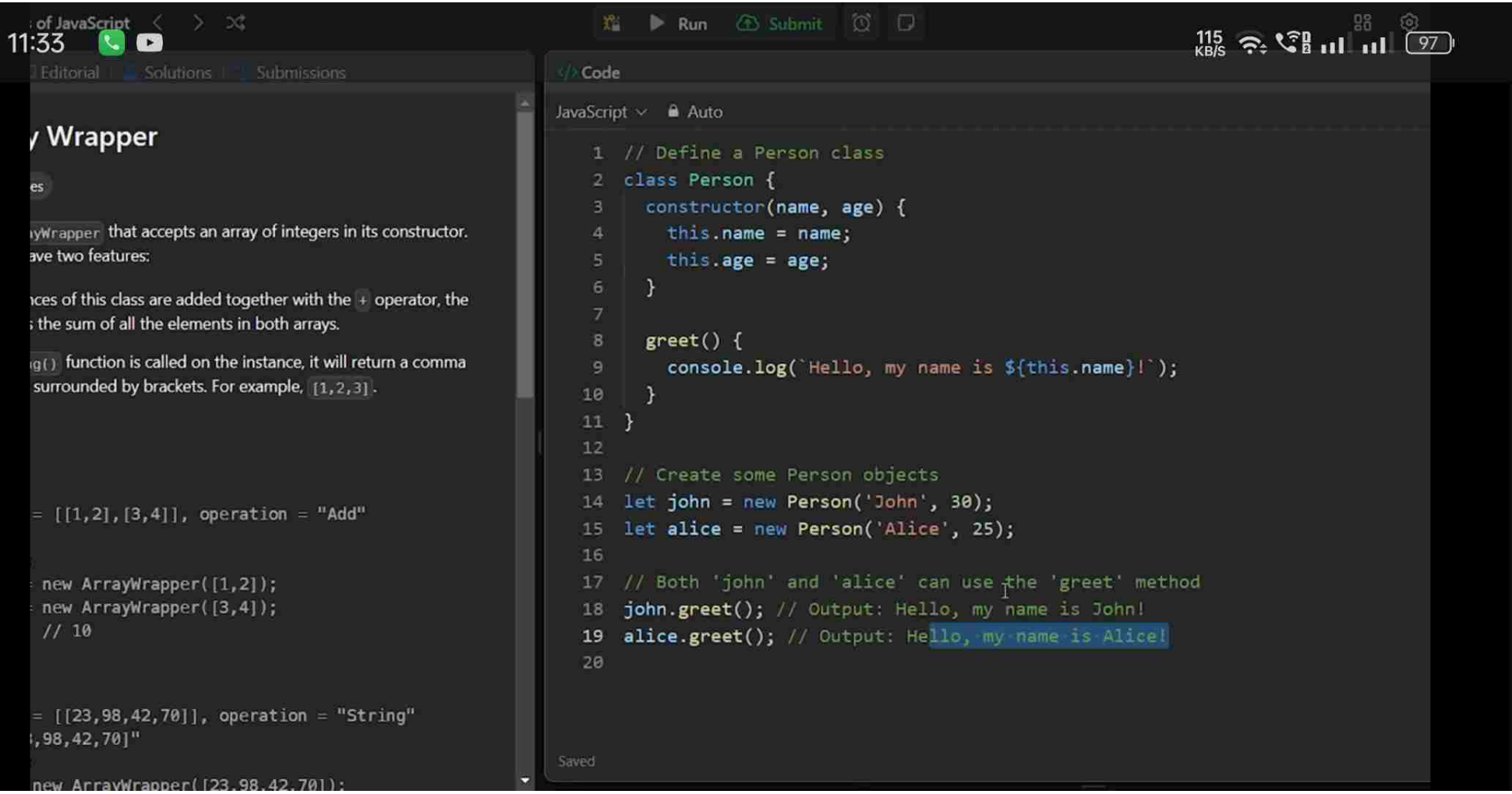

Add Two Promises
Run
index.js
index.js › test
// setInterval
function test(){
console.log("Learn With Chirag")
}
const stop = setInterval(test,1000)
setTimeout(()=>{
clearInterval(stop)
},5000)
Console
Shell
Run
Learn With Chirag
Learn With Chirag
Learn With Chirag
Learn With Chirag
Learn With Chirag
Learn With Chirag
Learn With Chirag
Run
Learn With Chirag
Learn With Chirag
Learn With Chirag
Learn With Chirag

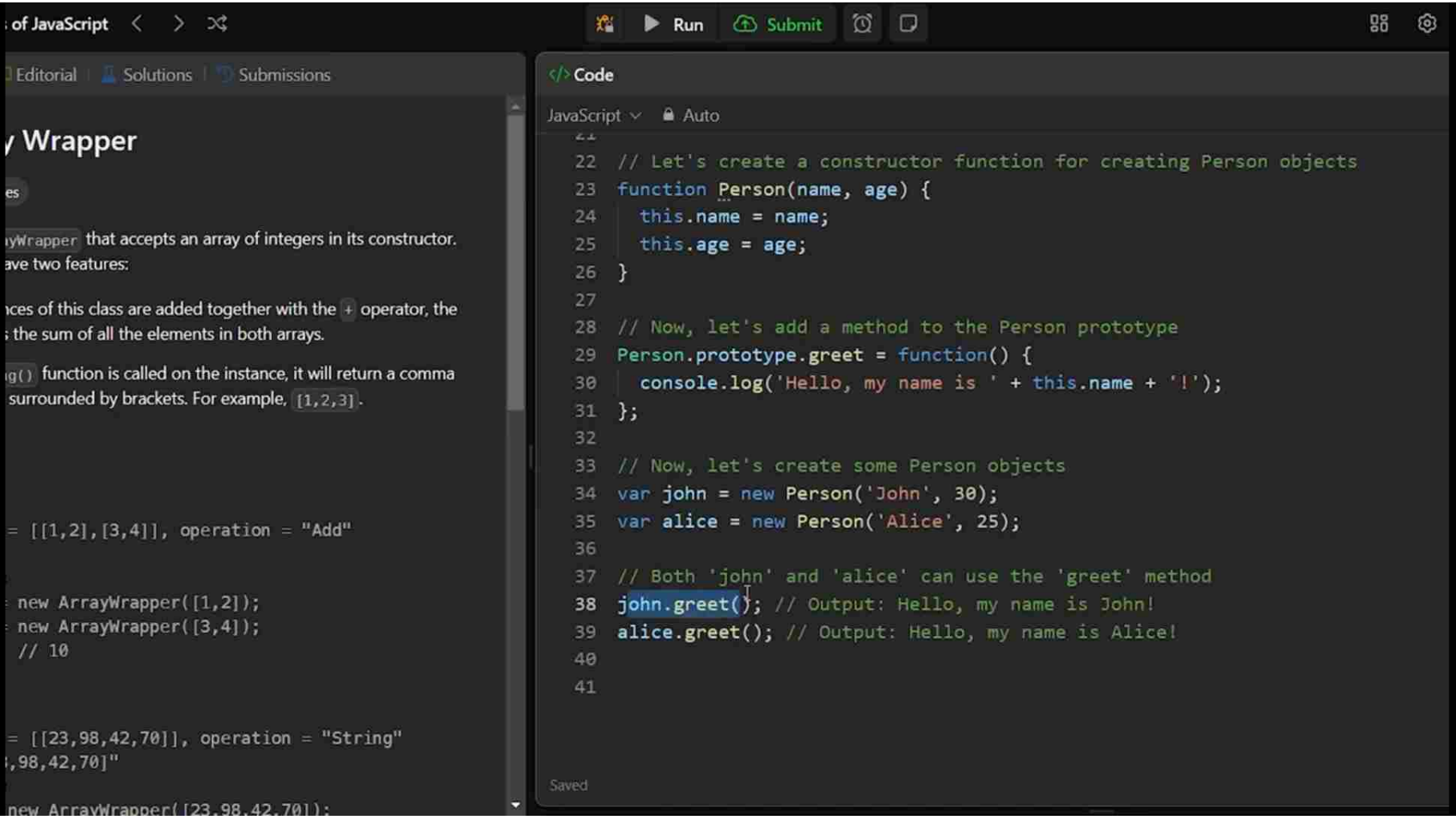

new Promise(executor)
state : "pending"
result : undefined
resolve(value)
reject(error)
state : "fulfilled"
result : value
state : "rejected"
result : error


Learn With Chirag
Run
index.js
index.js > myMap
Format
let myMap = new Map([
['name', 'Chirag'],
[true, 'Boolkey'],
[1, 'Numberkey'],
])
myMap.set('age' , 20)
// console.log(myMap)
// console.log(myMap.get('name'))
// console.log(myMap.has('gender'))
myMap.delete('name')
console.log(myMap)
Console
Shell
Run
Chirag
false
Run
Map(3) { true => 'Boolkey', 1 => 'Numb


HelloWorld.js
Learn With Chirag
8:00
11.0 KB/S
NEW
81
let text = `{
"browsers": {
"firefox": {
"name": "Firefox",
"pref_url": "about:config",
"releases": {
"1": {
"release_date": "2004-11-09",
"status": "retired",
"engine": "Gecko",
"engine_version": "1.7"
}
}
}
}
}`
let json = JSON.parse(text)
console.log(json)
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
// Lodash's _.chunk
// Lodash - Lodash is a utility library for JavaScript,
// providing functions to work with
// arrays, objects, strings, etc.
// // Syntax -
// _.chunk(array, size)
// Example
const abc = require("lodash")
let arr = [1, 2, 3, 4, 5, 6]
// Making chunks of size 1
console.log(abc.chunk(arr, 1))
STDIN
Input for the program (Optional)
Output:
[[1], [2], [3], [4], [5], [6


11:33
of JavaScript
Run
Submit
115 KB/S
97
Editorial
Solutions
Submissions
Wrapper
that accepts an array of integers in its constructor.
ave two features:
nces of this class are added together with the + operator, the
the sum of all the elements in both arrays.
g() function is called on the instance, it will return a comma
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"
,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!
20
Saved


of JavaScript
Run
Submit
Editorial
Solutions
Submissions
Wrapper
that accepts an array of integers in its constructor.
ave two features:
nces of this class are added together with the + operator, the
the sum of all the elements in both arrays.
g() function is called on the instance, it will return a comma
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"
,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!
40
41
Saved