**Part A:**

**Problem 0**

var cat = {  
 name: ‘Fluffy’,  
 activities: [‘play’, ‘eat cat food’],  
 catFriends: [  
 {  
 name: ‘bar’,  
 activities: [‘be grumpy’, ‘eat bread omblet’],  
 weight: 8,  
 furcolor: ‘white’  
 },   
 {  
 name: ‘foo’,  
 activities: [‘sleep’, ‘pre-sleep naps’],  
 weight: 3  
 }  
 ]  
}console.log(cat);

**Solution:**

//Add height and weight to Fluffy

cat.height =3;

cat.weight =8;

//Fluffy name is spelled wrongly. Update it to Fluffyy

cat.name='Fluffyy';

//Update the fur color of bar

cat.catFriends[0].furcolor='black';

//List all the activities of Fluffyy’s catFriends

for(var i in cat.catFriends){

console.log(cat.catFriends[i].activities);

}

console.log('\t');

//Print the catFriends names

for(var k in cat.catFriends){

console.log(cat.catFriends[k].name);

}

console.log('\t');

//Print the total weight of catFriends

var sumOfWeight=0;

for(var j in cat.catFriends){

sumOfWeight=sumOfWeight+parseInt(cat.catFriends[j].weight);

}

console.log("The total weight of catFriends: "+sumOfWeight);

console.log('\t');

//Print the total activities of all cats

var count=0;

for(var l in cat.activities){

count++;

}

for( var m in cat.catFriends){

for(var n in cat.catFriends[m].activities)

count++;

}

console.log(count)

console.log('\t');

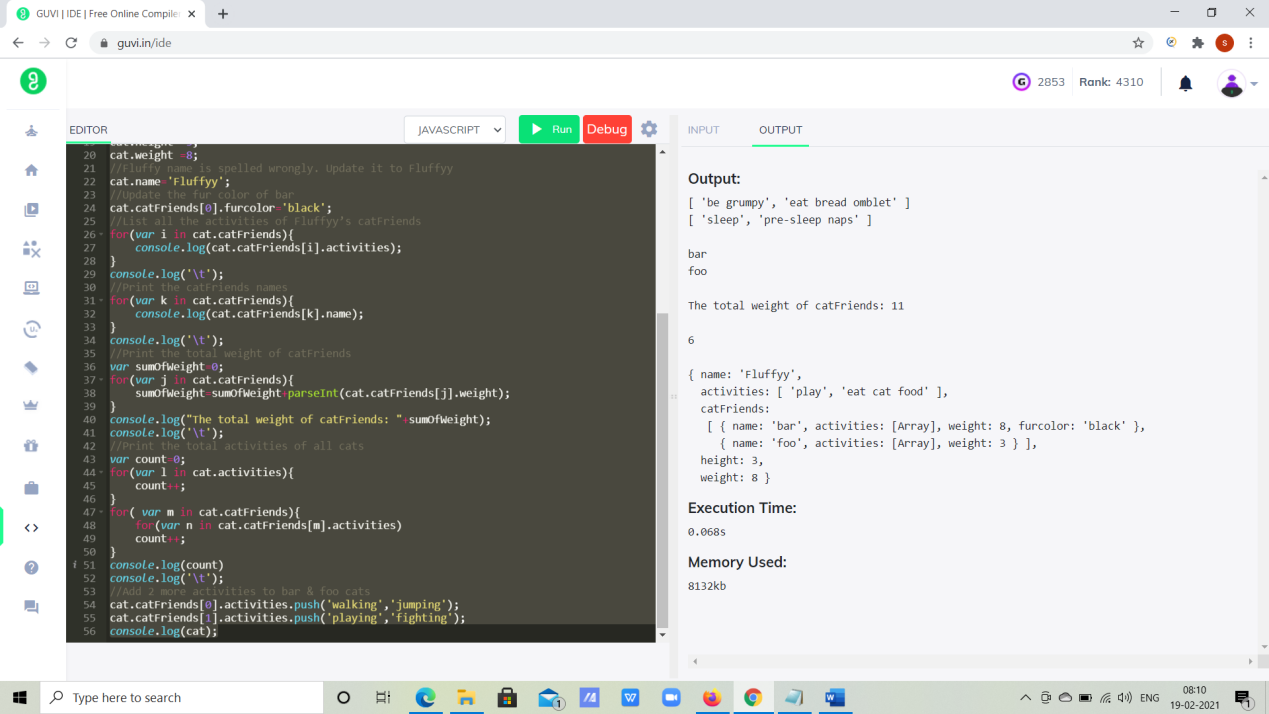
//Add 2 more activities to bar & foo cats

cat.catFriends[0].activities.push('walking','jumping');

cat.catFriends[1].activities.push('playing','fighting');

console.log(cat);

**ScreenPrint:**



**Part B:**

**Problem 0**

var myCar = {

make: 'Bugatti',

model: 'Bugatti La Voiture Noire',

year: 2019,

accidents: [

{

date: '3/15/2019',

damage\_points: '5000',

atFaultForAccident: true

},

{

date: '7/4/2022',

damage\_points: '2200',

atFaultForAccident: true

},

{

date: '6/22/2021',

damage\_points: '7900',

atFaultForAccident: true

}

]

}

**Solution:**

for(var i in myCar.accidents){

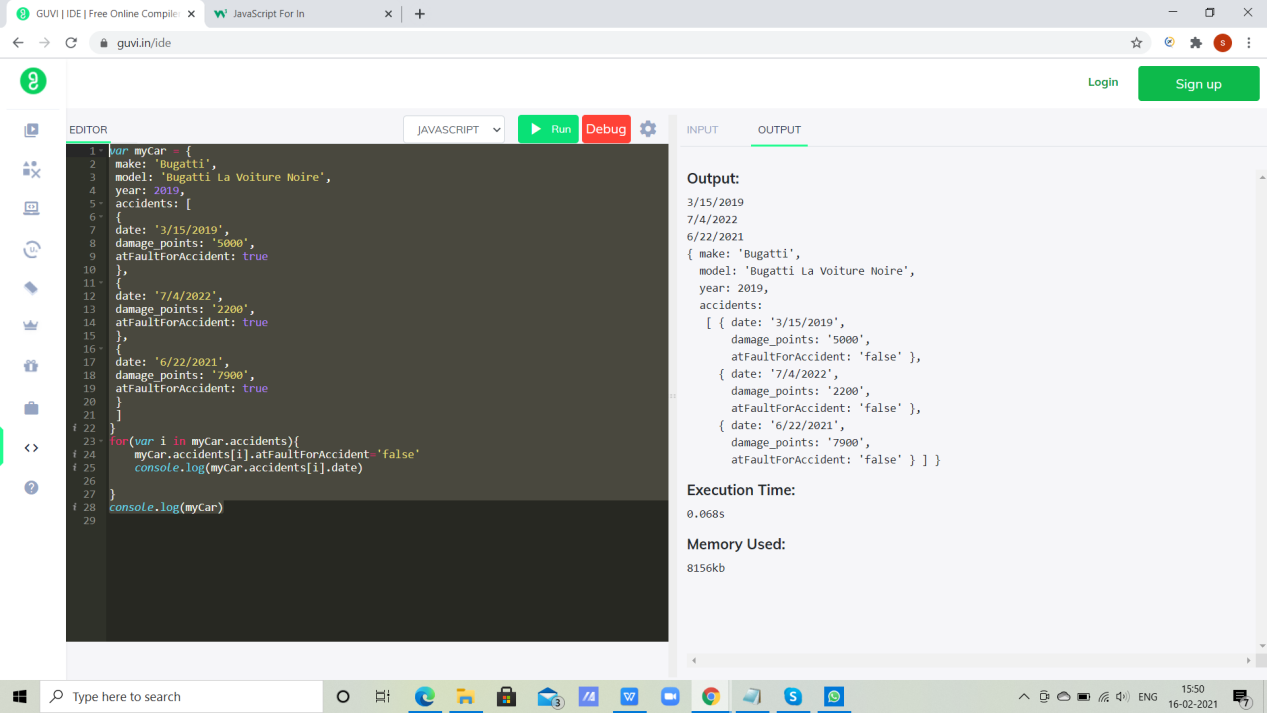
myCar.accidents[i].atFaultForAccident='false'

console.log(myCar.accidents[i].date)

}

console.log(myCar)

**ScreenPrint:**



**Problem 1:**

Write a function called “printAllValues” which returns an newArray of all the input object’s values.

Input (Object):

var object = {name: “RajiniKanth”, age: 33, hasPets : false};  
Output:

[“RajiniKanth”, 33, false]

**Solution:**

var obj = {name : "RajiniKanth", age : 33, hasPets : false};

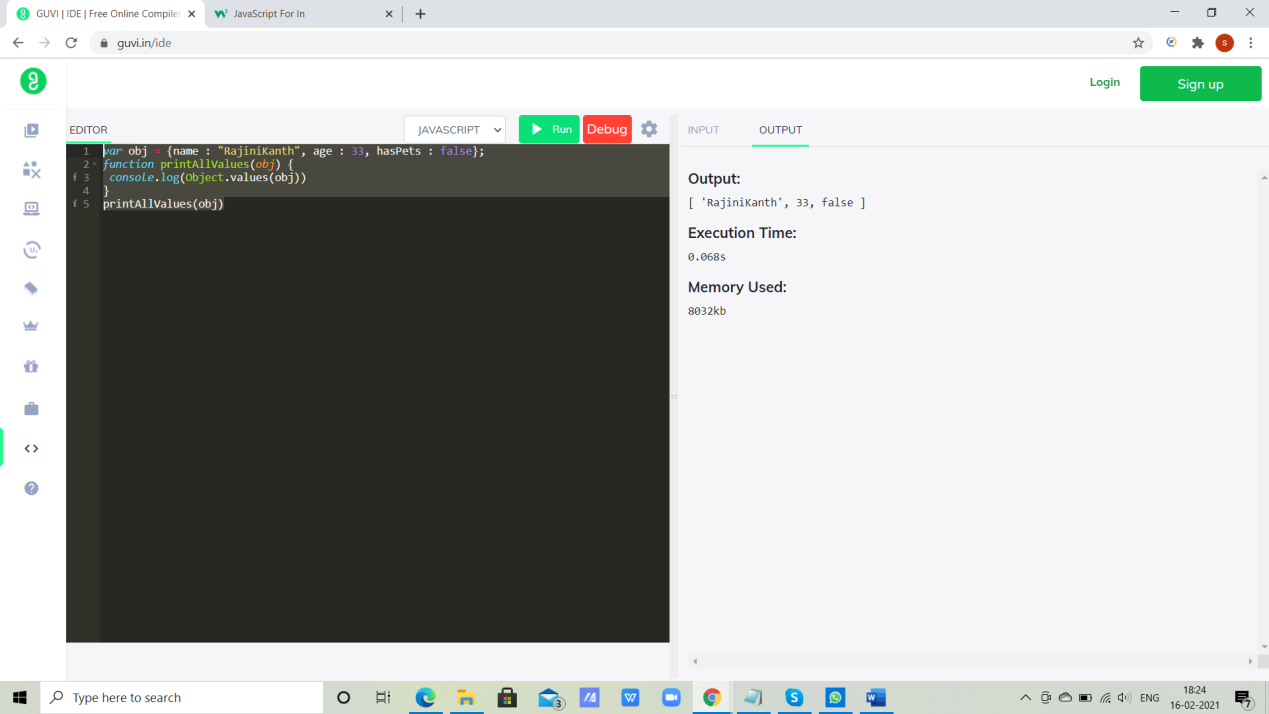
function printAllValues(obj) {

console.log(Object.values(obj))

}

printAllValues(obj)

**ScreenPrint:**



**Problem 2:**

Write a function called “printAllKeys” which returns an newArray of all the input object’s keys.

Example Input:  
{name : ‘RajiniKanth’, age : 25, hasPets : true}  
Example Output:  
[‘name’, ‘age’, ‘hasPets’]

**Solution:**

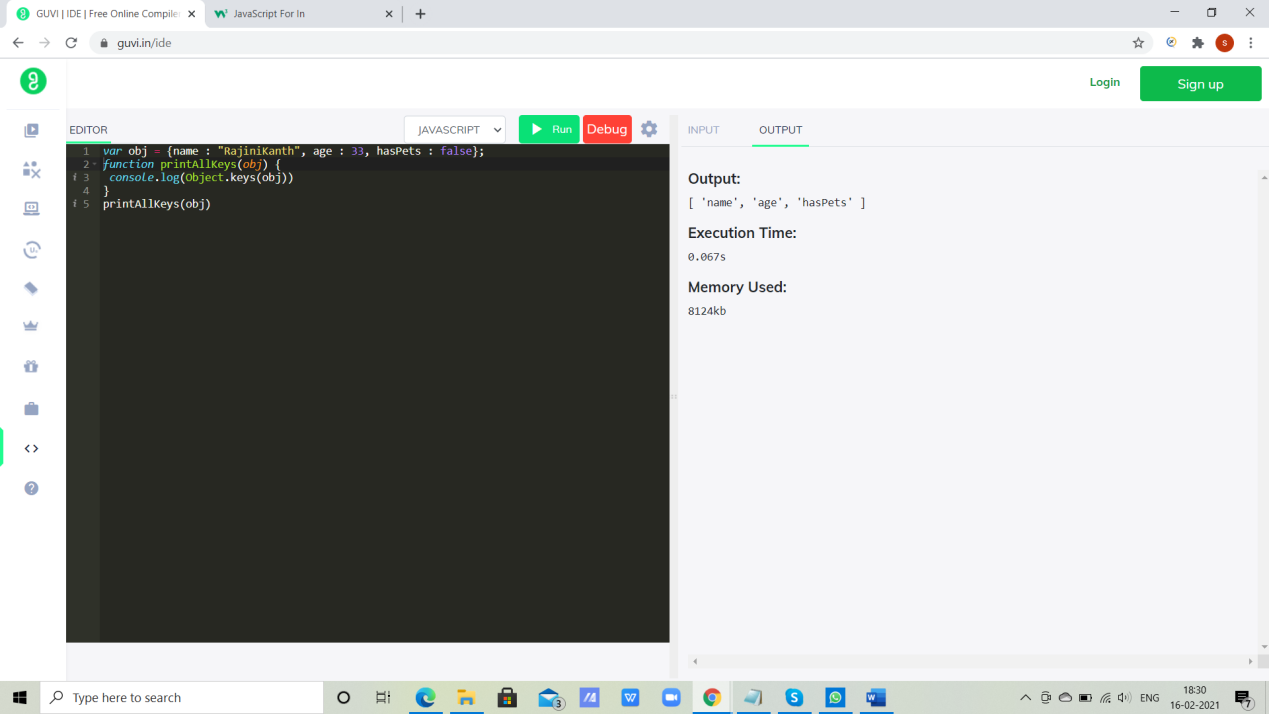
var obj = {name : "RajiniKanth", age : 33, hasPets : false};

function printAllKeys(obj) {

console.log(Object.keys(obj))

}

printAllKeys(obj)

**ScreenPrint:**

**Problem 3:**

Write a function called “convertObjectToList” which converts an object literal into an array of arrays.  
Input (Object):  
var object = {name: “ISRO”, age: 35, role: “Scientist”};  
Output:  
[[“name”, “ISRO”], [“age”, 35], [“role”, “Scientist”]]

**Solution:**

var obj = {name: "ISRO", age: 35, role: "Scientist"};

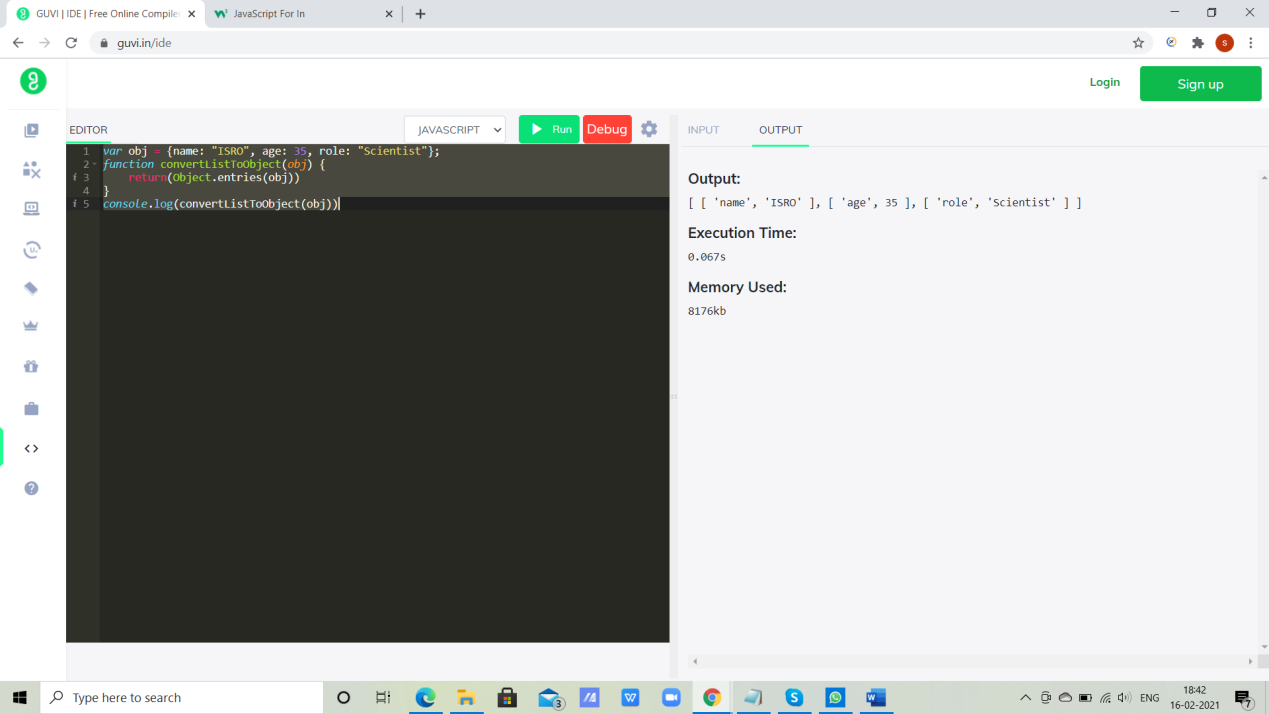
function convertListToObject(obj) {

return(Object.entries(obj))

}

console.log(convertListToObject(obj))

**ScreenPrint:**



**Problem 4:**

Write a function ‘transformFirstAndLast’ that takes in an array, and returns an object with:  
1) the first element of the array as the object’s key, and  
2) the last element of the array as that key’s value.  
Input (Array):  
var array = [“GUVI”, “I”, “am”, “Geek”];  
Output:  
var object = {  
GUVI : “Geek”  
}

**Solution:**

var arr = ["GUVI", "I", "am", "a geek"];

var newObject={}

function transformFirstAndLast(arr) {

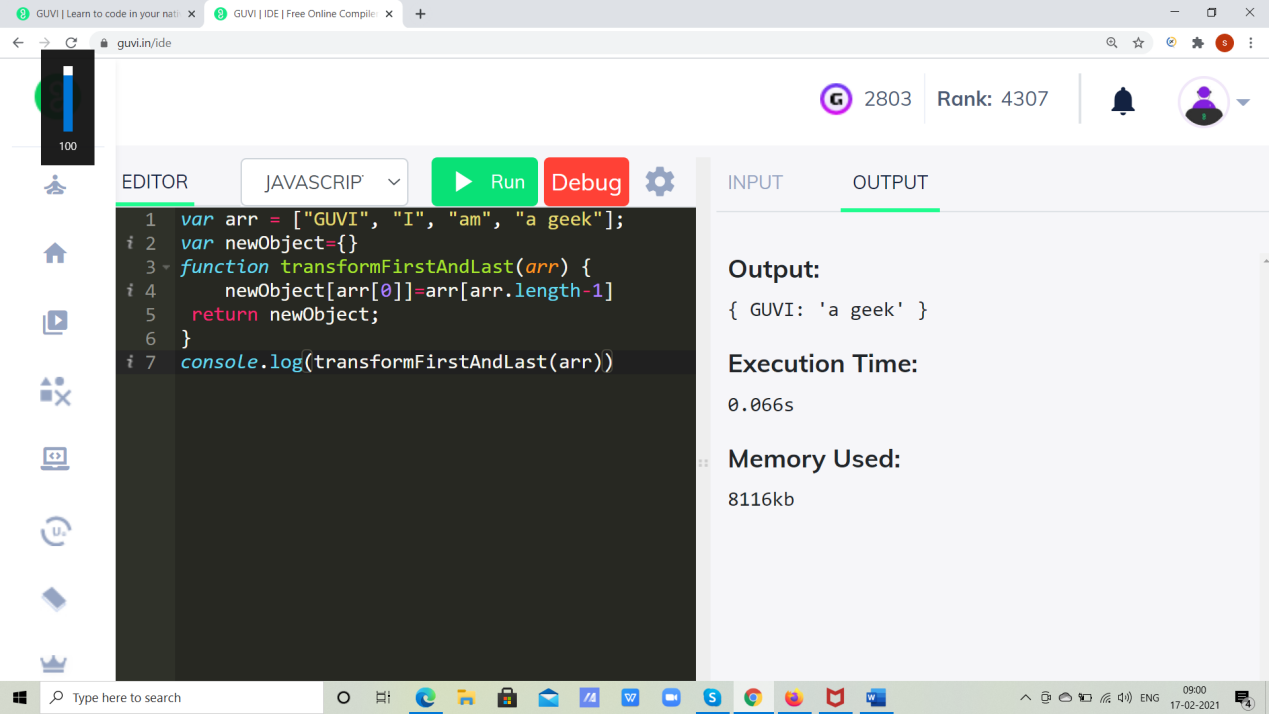
newObject[arr[0]]=arr[arr.length-1]

return newObject;

}

console.log(transformFirstAndLast(arr))

**ScreenPrint:**



**Problem 5:**

Write a function “fromListToObject” which takes in an array of arrays, and returns an object with each pair of elements in the array as a key-value pair.  
Input (Array):  
var array = [[“make”, “Ford”], [“model”, “Mustang”], [“year”, 1964]];  
Output:  
var object = {  
make : “Ford”  
model : “Mustang”,  
year : 1964  
}

**Solution:**

var arr = [["make", "Ford"], ["model", "Mustang"], ["year", 1964]];

function fromListToObject(arr) {

var newObject = {};

for (var i in arr){

var arr1=arr[i]

newObject[arr1[0]]=arr1[1]

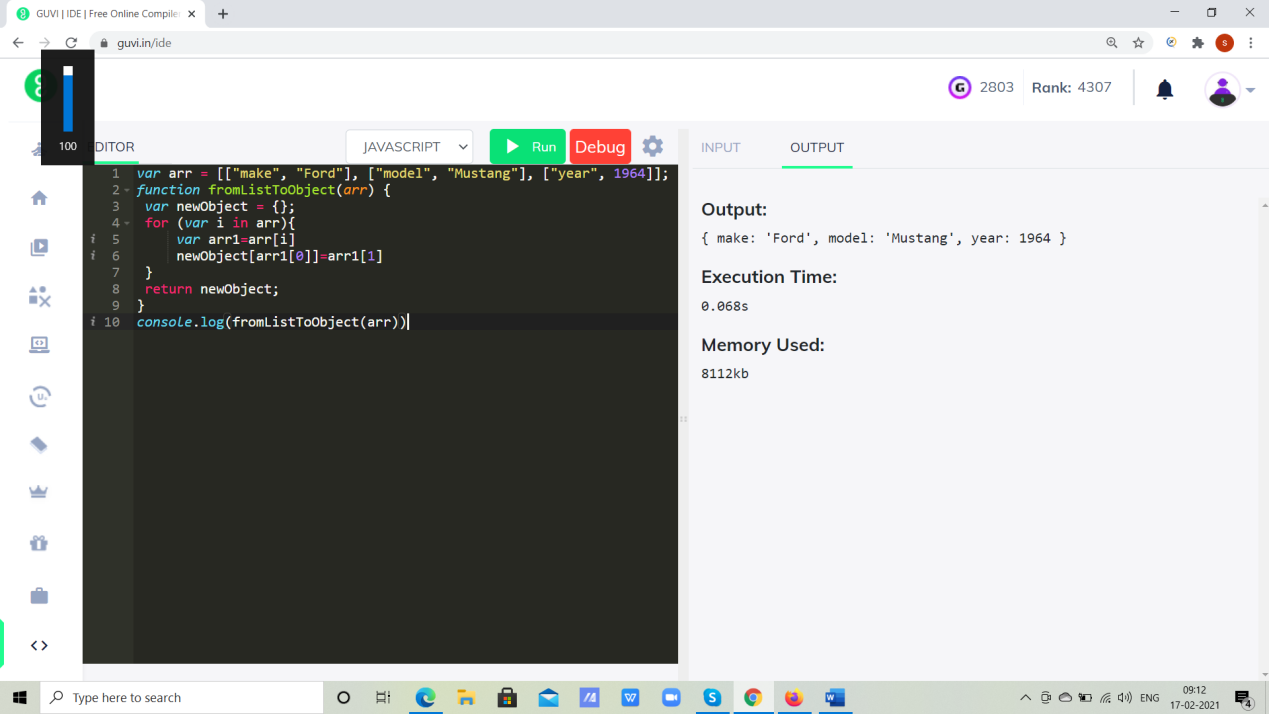
}

return newObject;

}

console.log(fromListToObject(arr))

**ScreenPrint:**



**Problem 6:**

Write a function called “transformGeekData” that transforms some set of data from one format to another.

Input (Array):  
var array = [[[“firstName”, “Vasanth”], [“lastName”, “Raja”], [“age”, 24], [“role”, “JSWizard”]], [[“firstName”, “Sri”], [“lastName”, “Devi”], [“age”, 28], [“role”, “Coder”]]];  
Output:  
[  
{firstName: “Vasanth”, lastName: “Raja”, age: 24, role: “JSWizard”},  
{firstName: “Sri”, lastName: “Devi”, age: 28, role: “Coder”}  
]

**Solution:**

var arr= [[["firstName", "Vasanth"], ["lastName", "Raja"], ["age", 24], ["role", "JSWizard"]], [["firstName", "Sri"], ["lastName", "Devi"], ["age", 28], ["role", "Coder"]]];

function transformEmployeeData(arr) {

var tranformEmployeeList = [];

for(var i=0;i<arr.length;i++){

let newobject={};

for(var j=0;j<arr[i].length;j++){

let key = arr[i]

let value =arr[j];

newobject[key]=value;

}

tranformEmployeeList.push(newobject);

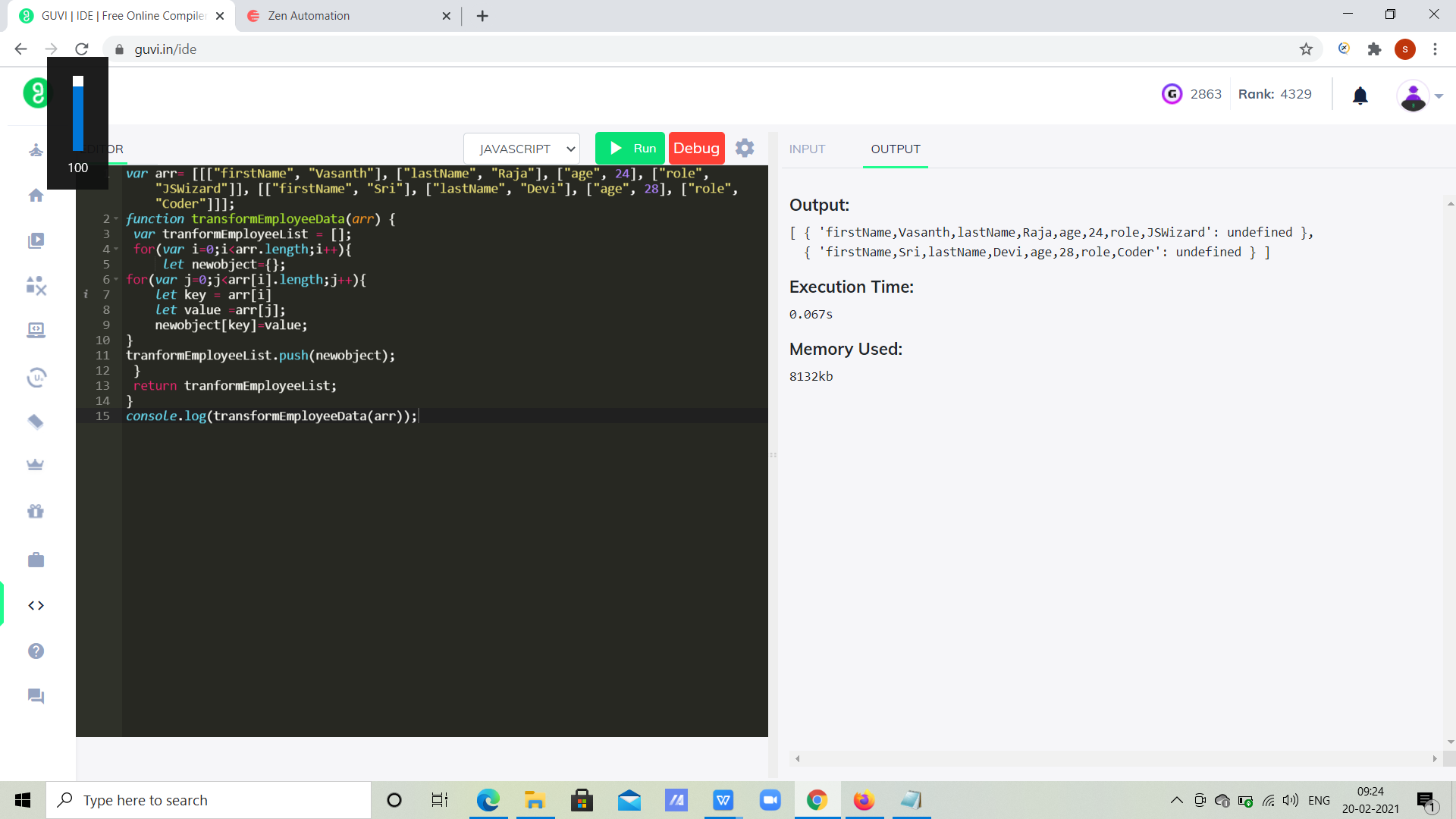
}

return tranformEmployeeList;

}

console.log(transformEmployeeData(arr));

**ScreenPrint:**



**Problem 7:**

Input:  
var expected = {foo: 5, bar: 6};  
var actual = {foo: 5, bar: 6}  
assertObjectsEqual(actual, expected, ‘detects that two objects are equal’);  
Output:  
Passed  
Failure Case:  
Input:var expected = {foo: 6, bar: 5};  
var actual = {foo: 5, bar: 6}  
assertObjectsEqual(actual, expected, ‘detects that two objects are equal’);  
Output:  
FAILED [my test] Expected {“foo”:6,”bar”:5}, but got {“foo”:5,”bar”:6}

**Solution:**

**//Input 1**

var expected = {foo: 5, bar: 6};

var actual = {foo: 5, bar: 6}

function assertObjectsEqual(actual, expected, testName){

var a = JSON.stringify(actual)

var b = JSON.stringify(expected)

console.log(a === b)

}

assertObjectsEqual(actual, expected, 'two objects are equal')

console.log('\t')

**//Input 2**

var expected1 = {foo: 6, bar: 5};

var actual1 = {foo: 5, bar: 6}

function assertObjectsEqual1(actual1, expected1, testName){

var c = JSON.stringify(actual1)

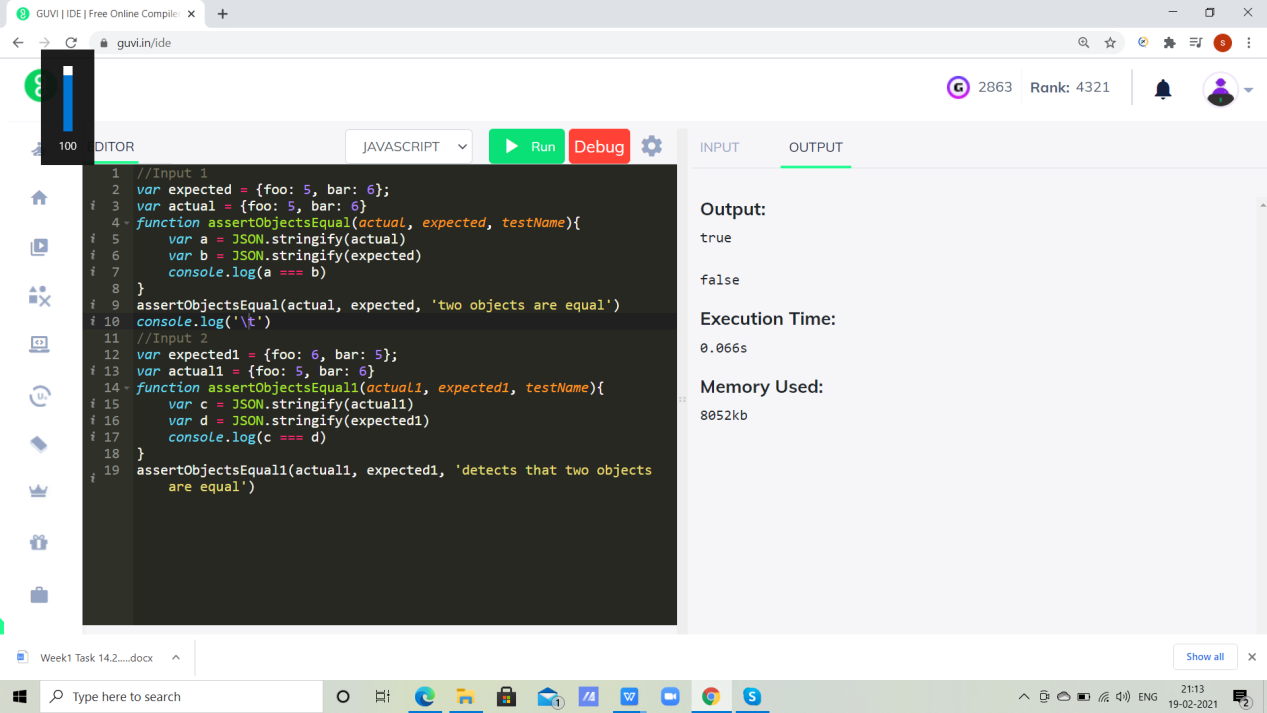
var d = JSON.stringify(expected1)

console.log(c === d)

}

assertObjectsEqual1(actual1, expected1, 'detects that two objects are equal')

**ScreenPrint:**



**Problem 8:**

var securityQuestions = [  
 {  
 question: “What was your first pet’s name?”,  
 expectedAnswer: “FlufferNutter”  
 },  
 {  
 question: “What was the model year of your first car?”,  
 expectedAnswer: “1985”  
 },  
 {  
 question: “What city were you born in?”,  
 expectedAnswer: “NYC”  
 }  
]function chksecurityQuestions(securityQuestions,question) {  
  
 // your code here return true or false;   
}//Test case1:var ques = “What was your first pet’s name?”;  
var ans = “FlufferNutter”;var status = chksecurityQuestions(securityQuestions, ques, ans);console.log(status); // true//Test case2:var ques = “What was your first pet’s name?”;  
var ans = “DufferNutter”;var status = chksecurityQuestions(securityQuestions, ques, ans);console.log(status); // false

**Solution:**

function chksecurityQuestions(securityQuestions,question,answer) {

if(securityQuestions[0].question===ques&& securityQuestions[0].expectedAnswer===ans){

return true

}

else{

return false

}

}

var securityQuestions = [

{

question: 'What was your first pet’s name?',

expectedAnswer: 'FlufferNutter'

},

{

question: 'What was the model year of your first car?',

expectedAnswer: '1985'

},

{

question: 'What city were you born in?',

expectedAnswer: 'NYC'

}

]

//Test case1:

var ques = 'What was your first pet’s name?';

var ans = 'FlufferNutter';

var returnStatus=chksecurityQuestions(securityQuestions, ques, ans);

console.log(returnStatus); // true

//Test case2:

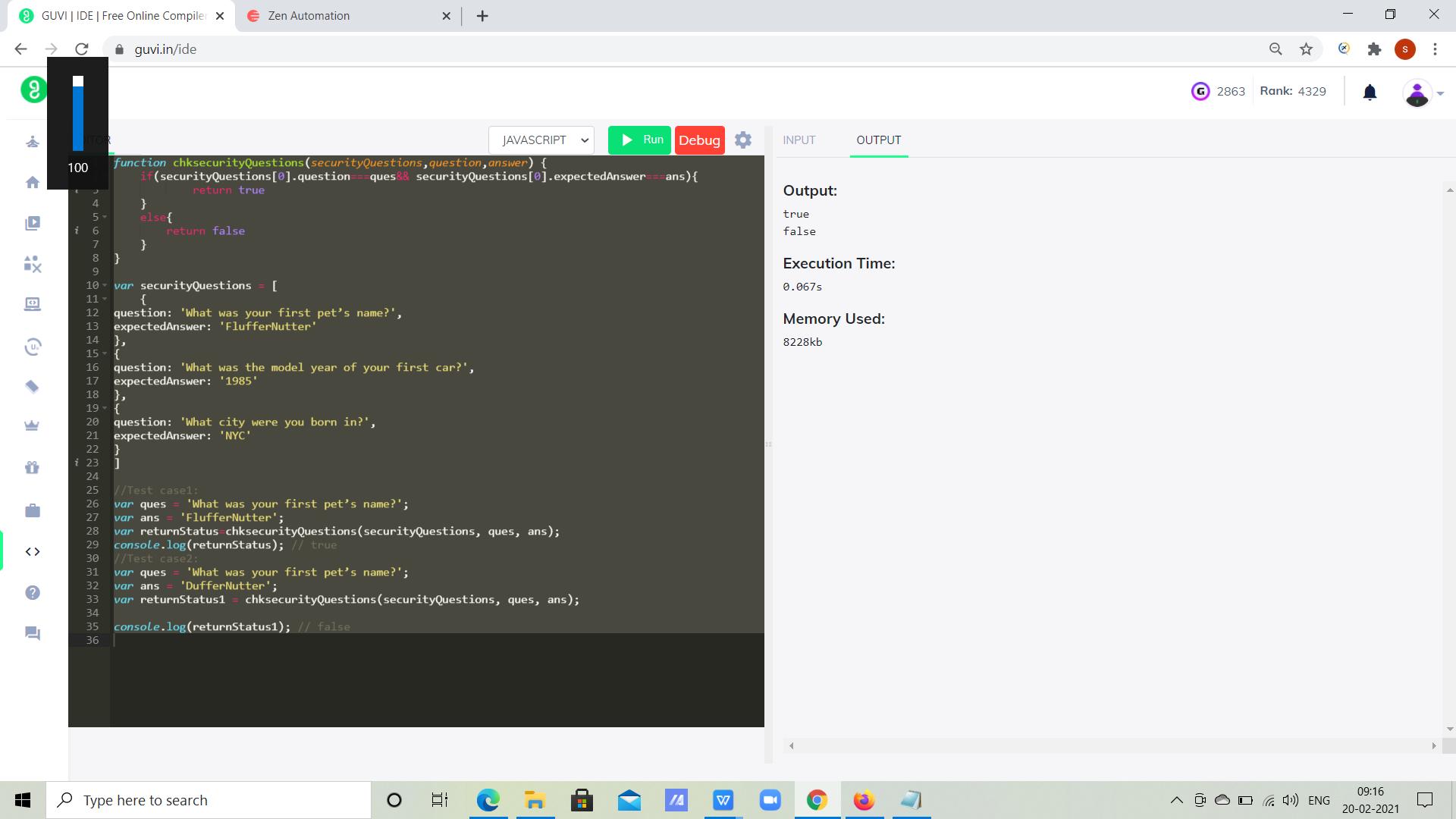
var ques = 'What was your first pet’s name?';

var ans = 'DufferNutter';

var returnStatus = chksecurityQuestions(securityQuestions, ques, ans);

console.log(returnStatus); // false

**ScreenPrint:**



**Problem 9:**

Write a function to return the list of characters below 20 age

var students = [  
{  
name: “Siddharth Abhimanyu”, age: 21}, { name: “Malar”, age: 25},  
{name: “Maari”,age: 18},{name: “Bhallala Deva”,age: 17},  
{name: “Baahubali”,age: 16},{name: “AAK chandran”,age: 23}, {name:“Gabbar Singh”,age: 33},{name: “Mogambo”,age: 53},  
{name: “Munnabhai”,age: 40},{name: “Sher Khan”,age: 20},  
{name: “Chulbul Pandey”,age: 19},{name: “Anthony”,age: 28},  
{name: “Devdas”,age: 56} ];

**Solution:**

function returnMinors(arr)

{

for(var i in students){

if(students[i].age<20){

console.log(students[i])

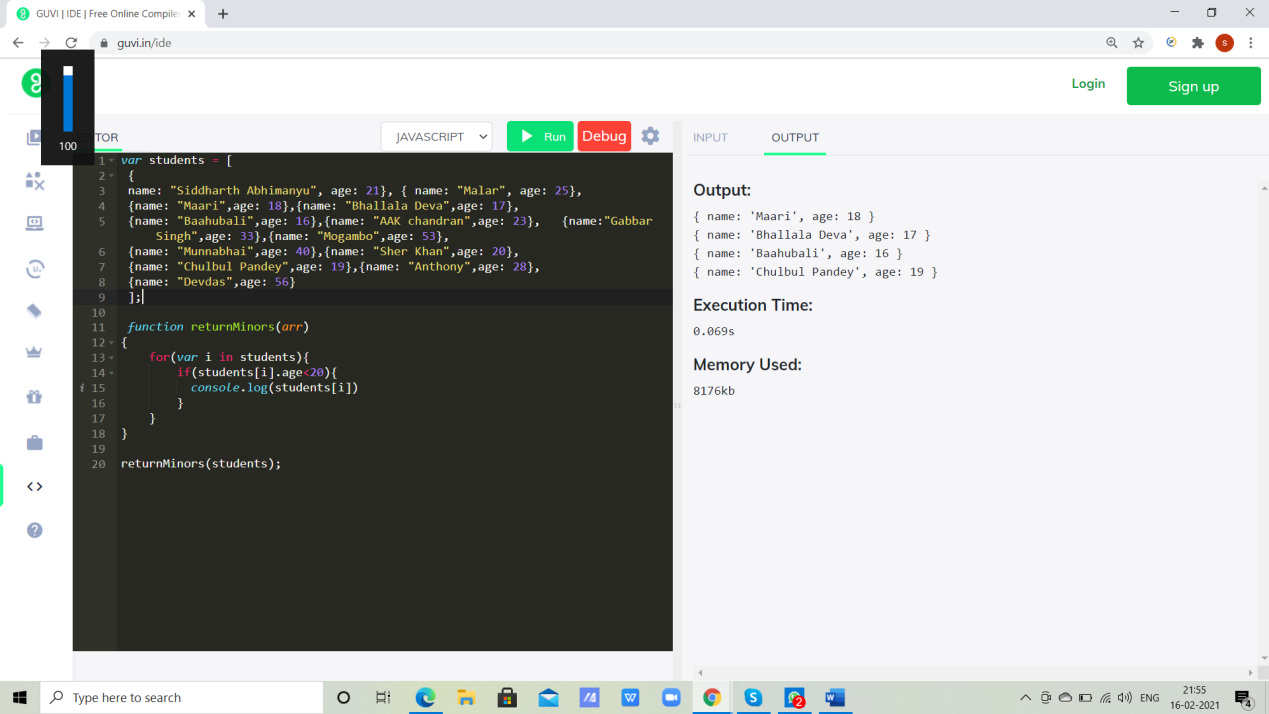
}

}

}

returnMinors(students);

**ScreenPrint:**



**Write Up Activity Solutions:**

The differences are quite significant:

* HTTP/2 is a **binary protocol**. This is potentially much more compressed than HTTP/1.1 and this fewer bytes on-the-wire. This means less bandwidth usage and faster load times - benefits which are amplified with many smaller requests.
* HTTP/2 is **multiplexed**. This means many requests can leverage an existing connection in parallel. Coming from HTTP/1.1’s one connection per request this is a huge leap in efficiency.
* HTTP/2 allows clients to make **multiple requests** over one TCP connection while HTTP/1.1 allows only **one request** (even with request pipelining)
* HTTP/2 helps to boost network performance (reduced total number of TCP connections, faster RTT(Round trip time), less resource consumptions) while still supporting all core features of HTTP/1.1.

**HTTP/1.0 Building Extensibility:**

* HTTP/0.9 was very limited and both browsers and servers quickly extended it to be more versatile:
* Versioning information is now sent within each request (HTTP/1.0 is appended to the GET line)
* A status code line is also sent at the beginning of the response, allowing the browser itself to understand the success or failure of the request and to adapt its behaviour in consequence (like in updated version compared to http/0.9)
* The notion of HTTP headers has been introduced, both for the requests and the responses, allowing metadata(record of communication that occurs in network) to be transmitted and making the protocol extremely flexible and extensible.
* With the help of the new HTTP headers, the ability to transmit other documents than plain HTML files has been added content is added to html file. Html file contains the image addition is extension of HTTP/0.9

**HTTP 1.1 is the Standardized Protocol:**

The first standardized version of HTTP, HTTP/1.1 was published in early 1997, only a few months after HTTP/1.0.

HTTP/1.1 clarified ambiguities and introduced numerous improvements:

* A connection can be reused, saving the time to reopen it numerous times to display the resources embedded into the single original document retrieved.
* Pipelining(multiple request) has been added, allowing to send a second request before the answer for the first one is fully transmitted, lowering the latency of the communication.
* Chunked responses are now also supported.
* Additional cache control mechanisms have been introduced.
* Content negotiation, including language, encoding, or type, has been introduced, and allows a client and a server to agree on the most adequate content to exchange.
* [Host](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Host) header, the ability to host different domains at the same IP address now allows server colocation.

**HTTP /2-A Protocol of Great Performance:**

The HTTP/2 protocol has several prime differences from the HTTP/1.1 version:

* It is a binary protocol rather than text. It can no longer be read and created manually. Despite this hurdle, improved optimization techniques can now be implemented.
* It is a multiplexed protocol. Parallel requests can be handled over the same connection, removing the order and blocking constraints of the (HTTP/1.0 or HTTP/1.1) protocol.
* It compresses headers. As these are often similar among a set of requests, this removes duplication and overhead of data transmitted.
* It allows a server to populate data in a client cache, in advance of it being required, through a mechanism called the server push.

It is officially standardized in May 2015 and it is used by 2016 worldwide by all the websites because it reduced High traffic by 68% and saving the data transfer considerably large.

**HTTP/3 Development Stage:**

* The next major version of HTTP, HTTP/3, will use QUIC (Quick UDP internet connection) is an experimental transport layer network protocol designed by Google. The overall goal is to reduce latency compared to that of TCP.

**Difference Between Browser JS and Node JS**

* **JavaScript is** a simple programming language which runs in any **browser JavaScript** Engine. Whereas **Node JS is** an interpreter or running environment for a **JavaScript** programming language which holds a lot of excesses require libraries which can easily be accessed from **JavaScript** programming for better use. Node JS runs in the V8 version for the Chrome browser. Here in the example below standalone Javascript Environment is Execution environment (Node JS environment).

In other word. JavaScript code is broken down to token this technique is called **Lexical analysis** and thus converted token is constructed like tree structure it is called as Abstract syntax Tree.

|  |  |
| --- | --- |
| **Node JS** | **Browser JS** |
| 1.Javascript Runtime environment.  Environment is with (V8)+(c/c++) | 2.In addition to JS engine it also have Rendering Engine for rendering the Page |
| 2.V8 engine is used in Node Js | It depends on the Browser V8 engine in Chrome ,Spider Monkey in Firefox, Edge it is Blink. |
| 3.it is used in Back End | It is used in Front end |
| 4.It act as Global Object | Here Browser it is Window |
| 5.Synchronous Activity is Single Threaded | Single Threaded |
| 6.Module System in Common JS | Module System is ES6 |

**3.What happens When You type URL in the address bar of the Browser?**

* When you enter the URL into the Web-Browser. Browser looks for the IP address for the Domain name in DNS(Domain Name System), then the browser sends request to the Server and Server Send back a HTTP response.
* The Browser Renders the HTML Parser and browser sends request for additional parsers like CSS parsers, JavaScript parsers this will be done until browser send request to server and server send back a HTTP response.

**4.Difference between copy by value and copy by Reference?**

**Copy by value**:

In a primitive data-type(string, number, Boolean, undefined and null) when a variable is assigned a value we can imagine that a box is created in the memory. This box has a sticker attached to it i.e. the variable name. Inside the box the value assigned to the variable is stored.

**Copy By reference:**

In case of a non-primitive data-type(object, Function and array) the values are not directly copied. When a non-primitive data-type is assigned a value a box is created with a sticker of the name of the data-type. However, the values it is assigned is not stored directly in the box. The language itself assigns a different memory location to store the data. The address of this memory location is stored in the box created.

**5.How do you copy by value a composite data type?**

There are 3 ways to copy by value for composite data types.

1. Using the spread (...) operator
2. Using the Object.assign() method
3. Using the JSON.stringify() and JSON.parse() methods.

**Spread** **Operator**:

**Spread operator** allows an iterable to expand in places where 0+ arguments are expected. It is mostly used in the variable array where there is more than 1 values are expected. It allows us the privilege to obtain a list of parameters from an array. Using spread will clone your object. Note this will be a shallow copy.

**Object.assign()** :

The **Object.assign()** method copies all enumerable own properties from one or more source objectsto atarget object. It returns the target object. Note this will be a shallow copy.[ ] this will not mutate(does not change nature or form) of Original value.

**JSON.parse() and JSON.stringify():**

The JSON object, available in all modern browsers, has two useful methods to deal with JSON-formatted content: **parse and stringify**.

**JSON.parse()** takes a JSON string and transforms it into a JavaScript object. **JSON.stringify()** takes a JavaScript object and transforms it into a JSON string. Using JSON.parse() and JSON.stringify() for copy performs deep copy.