JAVASCRIPT FULL STACK DEVELOPER

TASK 1

1) Difference between HTTP1.1 vs HTTP2 HTTP version history

HTTP stands for hypertext transfer protocol, is the method computers and servers use to request and send information. The first usable version of HTTP was created in 1997. Because it went through several stages of development, this first version of HTTP was called HTTP/1.1. This version is still in use on the web. In 2015, a new version of HTTP called HTTP/2 was created.

HTTP/2 solves several problems that the creators of HTTP/1.1 did not anticipate. In particular, HTTP/2 is much faster and more efficient than HTTP/1.1.

Few performance differences between HTTP/1.1 and HTTP/2 are:

a) Prioritization:

In the context of web performance, prioritization refers to the order in which pieces of content are loaded.
In HTTP/2, developers have hands-on, detailed control over prioritization. This allows them to maximize perceived and actual page load speed to a degree that was not possible in HTTP/1.1.

HTTP/2 offers a feature called weighted prioritization. This allows developers to decide which page resources will load first, every time. In HTTP/2, when a client makes a request for a webpage, the server sends several streams of data to the client at once, instead of sending one thing after another. This method of data delivery is known as multiplexing. Developers can assign each of these data streams a different weighted value, and the value tells the client which data stream to render first.

b) Multiplexing:

HTTP/1.1 loads resources one after the other, so if one resource cannot be loaded, it blocks all the other resources behind it. In contrast, HTTP/2 is able to use a single TCP connection to send multiple streams of data at once so that no one resource blocks any other resource. HTTP/2 does this by splitting data into binary-code messages and numbering these messages so that the client knows which stream each binary message belongs to.

c) Server push:

Typically, a server only serves content to a client device if the client asks for it. However, this approach is not always practical for modern webpages, which often involve several dozen separate resources that the client must request. HTTP/2 solves this problem by allowing a server to "push" content to a client before the client asks for it.

d) Header compression:

Small files load more quickly than large ones. To speed up web performance, both HTTP/1.1 and HTTP/2 compress HTTP messages to make them smaller. However, HTTP/2 uses a more advanced compression method called HPACK that eliminates redundant information in HTTP header packets. This eliminates a few bytes from every HTTP packet. Given the volume of HTTP packets involved in loading even a single webpage, those bytes add up quickly, resulting in faster loading.

2) List 5 differences between Browser JS vs Node Js.

S.No	Javascript	NodeJS
1.	Javascript is a programming language that is used for writing scripts on the website.	NodeJS is a Javascript runtime environment.
2.	Javascript can only be run in the browsers.	NodeJS code can be run outside the browser.
3.	It is basically used on the client-side.	It is mostly used on the server-side.
4.	Javascript is capable enough to add HTML and play with the DOM.	Nodejs does not have capability to add HTML tags.
5.	Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox.	Nodejs can only run in V8 engine of google chrome.
6.	Javascript is used in frontend development.	Nodejs is used in server-side development.

3) What happens when you type a URL in the address bar in the browser?

The following process occurs every time URL is typed in the address bar of the browser:

1. URL is entered into a web browser
2. The browser looks up the IP address for the domain name via DNS
3. The browser sends a HTTP request to the server
4. The server sends back a HTTP response
5. The browser begins rendering the HTML
6. The browser sends requests for additional objects embedded in HTML (images, css, JavaScript) and repeats steps 3-5.
7. Once the page is loaded, the browser sends further async requests as needed.

4) Write a write up on Difference between copy by value and copy by reference.

Copy-by-value	Copy-by-reference
This method is applicable on primitive data types, namely numbers and strings.	This method is applicable on composite data types, namely array, objects and functions.
Memory allocation : In a primitive data-type 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. Considering an example:	**Memory allocation :** In composite data-type the values are not directly copied. When a composite 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. Considering an example:

Program:
var x = 2;
var y =9;
var a=x;
var b=y;
var x= “aba”
var y= “cbc”
Console.log(x,y,a,b) // → aba , cbc , 2 , 9
	Program:
	let stu= {name: “Sid ”}
	let new=stu;
	new.name= “Ram ” // value changed
	alert(stu.name) // Name changed to Ram

From the above example both ‘x’ and ‘a’ contains the value 2. Both ‘y’ and ‘b’ contain the value ‘9. However, even though ‘x’ and ‘a’ as well as ‘y’ and ‘b’ contain the same value they are not connected to each other. It is so because the values are directly copied into the new variables.	From the above example both ‘stu and ‘new are storing the address of the memory location. And when one changes the values in the allocated memory it is reflected in the other as well.

5) How to copy by value a composite data type (array+objects).

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

6) JSON task https://medium.com/@reach2arunprakash/guvi-zen-code-sprint-javascript-practice-problems-in-json-objects-and-list-49ac3356a8a5

Problem 0 : Part A

```
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);
```

1. **Add height and weight to Fluffy**

```
cat.height = 3;
cat.weight = 5;
```

2. **Fluffy name is spelled wrongly. Update it to Fluffyy**

```
cat.name ="Fluffyy" ;
```

3. **List all the activities of Fluffyy's catFriends.**

```
for (var i in cat.catFriends)
{
console.log(cat.catFriends[i].activities);
}
```

4. **Print the catFriends names.**

```
for (var i in cat.catFriends)
{
console.log("Name of cat friends is " + cat.catFriends[i].name);
}
```

5. **Print the total weight of catFriends**

```
let total=0
for (var i in cat.catFriends)
{
   let weight = cat.catFriends[i].weight;
   total+=weight;
}
```

6. **Print the total activities of all cats (op:6)**

```
var act=[];
act.push(cat.activities);
act.push((cat.catFriends[0].activities));
act.push((cat.catFriends[1].activities));
console.log("All activities of cats are " + act.join(", "));
```

7. **Add 2 more activities to bar & foo cats**

```
cat.catFriends[0].actvities = ["be grumpy" , "eat bread omblet" ," Chase other cats" , " Eat rats"]
cat.catFriends[1].actvities = ["sleep", "pre-sleep naps" ," Jump" , " Eat fish "]
```

8. **Update the fur color of bar**

```
cat.catFriends[0].color = "Black "
```

Final Output:

```
['be grumpy', 'eat bread omblet']
['sleep', 'pre-sleep naps']

Name of cat friends is bar
Name of cat friends is foo

Total weight of cat friends = 11

All activities of cats are play,eat cat food, be grumpy,eat bread omblet, sleep,pre-sleep
naps

{ name: 'Fluffyy',
  activities: [ 'play', 'eat cat food' ],
  catFriends:
   [ { name: 'bar',
       activities: [Array],
       weight: 8,
       furcolor: 'white',
       actvities: [Array],
       color: 'Black ' },
     { name: 'foo',
       activities: [Array],
       weight: 3,
       actvities: [Array] } ],
  height: 3,
  weight: 5 }
```

Problem 0 : Part B

```
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
 }
 ]
}
```

1. **Loop over the accidents array. Change atFaultForAccident from true to false.**

```
for(var i in myCar.accidents)
{
myCar.accidents[i].atFaultForAccident= false;
}
```

2. **Print the dated of my accidents**

```
for(var i in myCar.accidents)
{
   console.log(" The date of accident is " + myCar.accidents[i].date)
}
console.log(myCar)
```

Final Output:

```
The date of accident is 3/15/2019
 The date of accident is 7/4/2022
 The date of accident is 6/22/2021

{ make: 'Bugatti',
  model: 'Bugatti La Voiture Noire',
  year: 2019,
  accidents:
   [ { date: '3/15/2019',
       damage_points: 5000,
       atFaultForAccident: false },
     { date: '7/4/2022',
       damage_points: 2200,
       atFaultForAccident: false },
     { date: '6/22/2021',
       damage_points: 7900,
       atFaultForAccident: false } ] }
```

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};

Expected Output:

[“RajiniKanth”, 33, false]

Code:

```
var obj = {name : "RajiniKanth", age : 33, hasPets : false};
function printAllValues(obj)
{
var data=[];
for (var i in obj)
{
  data.push(obj[i]);
}
return data;
}
console.log(printAllValues(obj));
```

Output:

```
[ 'RajiniKanth', 33, false ]
```

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’]

Code:

```
var obj = {name : "RajiniKanth", age : 25, hasPets : true};
function printAllKeys(obj)
{
 var key =  Object.keys(obj);
return key
}
console.log(printAllKeys(obj));
```

Output:

```
[ 'name', 'age', 'hasPets' ]
```

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”]]

Code:

```
var obj = {name: "ISRO" , age: 35, role: "Scientist" };
function convertListToObject(obj)
{
let list= (Object.entries(obj))
return list;
}
 console.log(convertListToObject(obj) )
```

Output:

```
[ [ 'name', 'ISRO' ], [ 'age', 35 ], [ 'role', 'Scientist' ] ]
```

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”

}

Code:

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

function transformFirstAndLast(arr) {
 var obj={};
 obj[arr[0]]=arr[3];
 return obj
}
console.log(transformFirstAndLast(arr))
```

Output:

```
{ GUVI: 'a geek' }
```

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
}

Code:

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

function fromListToObject(arr)
{
 var newObject = {};
 for ( var i=0; i<arr.length;i++)
 {
    newObject[arr[i][0]]=arr[i][1];
 }
 return newObject;
}
console.log(fromListToObject(arr))
```

Output:

```
{ make: 'Ford', model: 'Mustang', year: 1964 }
```

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”}
]

Code:

```
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++)
 {
  tranformEmployeeList[i]={};
  for (var j=0;j<arr[i].length;j++)
  {
     tranformEmployeeList[i][arr[i][j][0]]=arr[i][j][1];
  }
 }
 return tranformEmployeeList;
}
console.log(transformEmployeeData(arr))
```

Output:

```
[ { firstName: 'Vasanth',
    lastName: 'Raja',
    age: 24,
    role: 'JSWizard' },
  { firstName: 'Sri', lastName: 'Devi', age: 28, role: 'Coder' } ]
```

Problem 7:

Write an "assertObjectsEqual" function from scratch.
Assume that the objects in question contain only scalar values (i.e., simple values like strings or numbers).
It is OK to use JSON.stringify().
Note: The examples below represent different use cases for the same test. In practice, you should never have multiple tests with the same name.

Success Case:

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}

Code:

```
function assertObjectsEqual(actual, expected,testName)
{
var strexp=JSON.stringify(expected);
var stract=JSON.stringify(actual);
var out;
      if(strexp === stract)
      {
         out = "Passed”;
      }
      else
      {
         out= ( "FAILED "  + testName + " Expected " + strexp + " , but got " + stract )
      }
return out
}
console.log(assertObjectsEqual(actual, expected,"Mytest"))
```

a)

Input:

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

Output:

```
FAILED Mytest Expected {"foo":5,"bar":6} , but got {"foo":6,"bar":6}
```

b)

Input:

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

Output:

```
Passed
```

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); // flase
```

Code:

```
function chksecurityQuestions(securityQuestions, ques, ans) {
var ques,ans;
var x = "false"

for ( var i in securityQuestions)
{
   if (( securityQuestions[i].question === ques) && (securityQuestions[i].expectedAnswer === ans))
   {
      x = "true";
   }
}
 return x;
}
```

a)

Input:

```
ques = "What was your first pet’s name?";
ans  =  "FlufferNutter";
var status = chksecurityQuestions(securityQuestions, ques, ans);
console.log(status);
```

Output:

```
True
```

b)

Input:

```
var ques = "What was your first pet’s name?";
var ans  =  "DufferNutter";
var status = chksecurityQuestions(securityQuestions, ques, ans);
console.log(status);
```

Output:

```
false
```

Problem 9:

```
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}
 ];function returnMinors(arr)
{}console.log(returnMinors(students));
```

Code:

```
function returnMinors(arr)
{
   var listofnames=[];
 for (var i=0;i<students.length;i++)
 {
    if(students[i].age < 20 )
    {
       listofnames.push(students[i].name)
    }
 }
return listofnames;
}
console.log(returnMinors(students));
```

Output:

```
[ 'Maari', 'Bhallala Deva', 'Baahubali', 'Chulbul Pandey' ]
```