TASK 1

1 DIFFERENCE BETWEEN HTTP1.1 AND HTTP2.

- *The Hypertext Transfer Protocol, or HTTP, is an application protocol that has been the de facto standard for communication on the World Wide Web since its invention in 1989. From the release of HTTP/1.1 in 1997 until recently, there have been few revisions to the protocol. But in 2015, a reimagined version called HTTP/2 came into use, which offered several methods to decrease latency, especially when dealing with mobile platforms and server-intensive graphics and videos.*

- *HTTP/2 has since become increasingly popular, with some estimates suggesting that around a third of all websites in the world support it. In this changing landscape, web developers can benefit from understanding the technical differences between HTTP/1.1 and HTTP/2, allowing them to make informed and efficient decisions about evolving best practices.*

1.1 HTTP1

- HTTP is a top-level application protocol that exchanges information between a client computer and a local or remote web server. In this process, a client sends a text-based request to a server by calling a *method* like `GET` or `POST`. In response, the server sends a resource like an HTML page back to the client.

  For Example :

  GET /index.html HTTP /1.1

  Host : www.example.com.
- HTTP/1.1 relies on the transport layer to avoid buffer overflow, each new TCP connection requires a separate flow control mechanism

- The first response that a client receives on an HTTP `GET` request, contains links to additional resources needed by the requested page. Because of this, the client will have to make additional requests to retrieve these resources. In HTTP/1.0, the client had to break and remake the TCP connection with every new request, a costly affair in terms of both time and resources.
- It uses technique called *resource inlining* to include the required resource directly within the HTML document that the server sends in response to the initial `GET` request. For example, if a client needs a specific CSS file to render a page, inlining that CSS file will provide the client with the needed resource before it asks for it, reducing the total number of requests that the client must send.

- It uses programs like gzip to compress the data sent in HTTP messages, especially to decrease the size of CSS and JavaScript files.The header component of a message, however, is always sent as plain text , which creates this uncompress data weighs heavier and heavier on the connection as more requests are made.

1.2 HTTP2

- HTTP/2 began as the SPDY protocol, developed primarily at Google with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization.

- As opposed to HTTP/1.1, which keeps all requests and responses in plain text format, HTTP/2 uses the binary framing layer to encapsulate all messages in binary format, while still maintaining HTTP semantics, such as verbs, methods, and headers. An application level API would still create messages in the conventional HTTP formats, but the underlying layer would then convert these messages into binary.

- In HTTP/2, the binary framing layer encodes requests/responses and cuts them up into smaller packets of information, greatly increasing the flexibility of data transfer.

- It allows the client server to implement their own flow controls, rather than relying on the transport layer. The application layer communicates the available buffer space, allowing the client and server to set the receive window on the level of the multiplexed streams. This fine-scale flow control can be modified or maintained after the initial connection via a `WINDOW_UPDATE` frame.

- It uses a process called *server push.* In this way, an HTTP/2 connection can accomplish the same goal of resource inlining while maintaining the separation between the pushed resource and the document. This means that the client can decide to cache or decline the pushed resource separate from the main HTML document, fixing the major drawback of resource inlining.

- HTTP/2 can split headers from their data, resulting in a header frame and a data frame. The HTTP/2-specific compression program HPACK can then compress this header frame. This algorithm can encode the header metadata using Huffman coding, thereby greatly decreasing its size.

2 HTTP VERSION HISTORY.

- The original HTTP proposal by Tim Berners-Lee was designed with *simplicity in mind* as to help with the adoption of his other nascent idea: the World Wide Web.

A simple prototype was built, which implemented a small subset of the proposed functionality:

- Client request is a single ASCII character string.
- Client request is terminated by a carriage return (CRLF).
- Server response is an ASCII character stream.
- Server response is a hypertext markup language (HTML).

- Connection is terminated after the document transfer is complete.

The preceding exchange is not an exhaustive list of HTTP/1.0 capabilities, but it does illustrate some of the key protocol changes:

- Request may consist of multiple newline separated header fields.
- Response object is prefixed with a response status line.
- Response object has its own set of newline separated header fields.
- Response object is not limited to hypertext.
- The connection between server and client is closed after every request.

The work on turning HTTP into an official IETF Internet standard proceeded in parallel with the documentation effort around HTTP/1.0 and happened over a period of roughly four years: between 1995 and 1999.

The HTTP/1.1 standard resolved a lot of the protocol ambiguities found in earlier versions and introduced a number of critical performance optimizations: keepalive connections, chunked encoding transfers, byte-range requests, additional caching mechanisms, transfer encodings, and request pipelining.

The simplicity of the HTTP protocol is what enabled its original adoption and rapid growth. In fact, it is now not unusual to find embedded devices—sensors, actuators, and coffee pots alike—using HTTP as their primary control and data protocols.

But under the weight of its own success and as we increasingly continue to migrate our everyday interactions to the Web—social, email, news, and video, and increasingly our entire personal and job workspaces—it has also begun to show signs of stress. Users and web developers alike are now demanding near real-time responsiveness and protocol performance from HTTP/1.1, which it simply cannot meet without some modifications.

To meet these new challenges, HTTP must continue to evolve, and hence the HTTPbis working group announced a new initiative for HTTP/2 in early 2012.

The primary focus of HTTP/2 is on improving transport performance and enabling both lower latency and higher throughput. The major version increment sounds like a big step, which it is and will be as far as performance is concerned, but it is important to note that none of the high-level protocol semantics are affected: all HTTP headers, values, and use cases are the same.

3 DIFFERENCE BETWEEN JAVASCRIPT VC NODEJS.

JavaScript	Node JS

i.	JavaScript is a simple programming language which runs in any browser JavaScript Engine	i.	NodeJS is a Javascript runtime environment
ii.	Javascript can only be run in the browsers.	ii.	NodeJS code can be run outside the browser.
iii.	Javascript is capable enough to add HTML and play with the DOM.	iii.	Nodejs does not have capability to add HTML tags.
iv.	Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox.	iv.	Nodejs can only run in V8 engine of google chrome.
v.	Some of the javascript frameworks are RamdaJS, TypedJS, etc.	v.	Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm.

4 WHAT HAPPENS WHEN YOU TYPE A URL IN THE ADDRESS BAR IN THE BROWSER?

Browser checks cache for DNS entry to find the corresponding IP address of website.
It looks for following cache. If not found in one, then continues checking to the next until found.

- Browser Cache
- Operating Systems Cache
- Router Cache
- ISP Cache

1. DNS server initiates a DNS query to find IP address of server that hosts the domain name.

Browser initiates a TCP (Transfer Control Protocol) connection with the server using synchronize(SYN) and acknowledge(ACK) messages.
Browser sends an HTTP request to the web server. GET or POST request.
Server on the host computer handles that request and sends back a response. It assembles a response in some format like JSON, XML and HTML.

2. Server sends out an HTTP response along with the status of response.

Browser displays HTML content
Finally, Done.

5 DIFFERENCE BETWEEN COPY-BY-VALUE AND COPY-BY-REFERENCE.

Copy-by-value.	Copy-by-reference.
1. This method is applicable on primitive data type, namely numbers and strings.	1. This method is applicable on composite data types, namely arrays,objects and function.
2. 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:	2. 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
 	
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.
	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 ‘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.

6 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

7 JSON TASK HTTPS://MEDIUM.COM/@REACH2ARUNPRAKASH/GUVI-ZEN-CODE-SPRINT-JAVASCRIPT-PRACTICE-PROBLEMS-INJSON-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.activitie
s);
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
 },
 {
```

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.

8 EXAMPLE INPUT:

{name : ‘RajiniKanth’, age : 25, hasPets : true}

9 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.

10 INPUT (OBJECT):

var object = {name: “ISRO”, age: 35, role: “Scientist”};

11 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.

12 INPUT (ARRAY):

var array = [“GUVI”, “I”, “am”, “Geek”];

13 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.

14 INPUT (ARRAY):

var array = [[“make”, “Ford”], [“model”, “Mustang”], [“year”, 1964]];

15 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.

16 INPUT (ARRAY):

var array = [[[“firstName”, “Vasanth”], [“lastName”, “Raja”], [“age”, 24], [“role”, “JSWizard”]], [[“firstName”, “Sri”], [“lastName”, “Devi”], [“age”, 28], [“role”, “Coder”]]];

17 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.

18 SUCCESS CASE:

Input:

```
var expected = {foo: 5, bar: 6};
var actual = {foo: 5, bar: 6}
assertObjectsEqual(actual, expected, ‘detects that two objects are equal’);
```

19 OUTPUT:

Passed

20 FAILURE CASE:

Input:

```
var expected = {foo: 6, bar: 5};
var actual = {foo: 5, bar: 6}
assertObjectsEqual(actual, expected, ‘detects that two objects are equal’);
```

21

22 OUTPUT:

```
FAILED [my test] Expected {“foo”:6,”bar”:5}, but got {“foo”:5,”bar”:6}
```

23 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”

 }
```

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:“Gabbar
 {name: “Maari”,age: 18},{name: “Bhallala Deva”,age: 17},
 {name: “Baahubali”,age: 16},{name: “AAK chandran”,age: 23},
Singh”,age: 33},{name: “Mogambo”,age: 53},
 {name: “Munnabhai”,age: 40},{name: “Sher Khan”,age: 20},
```

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' ]
```