**Blog 1: Difference between HTTP1.1 vs HTTP2**

**HTTP/1.1**: It was developed in 1989 as a communication standard for world wide web. In this, 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, let’s say you are visiting a website at the domain www.example.com. When you navigate to this URL, the web browser on your computer sends an HTTP request in the form of a text-based message, similar to the one shown here:

GET /index.html HTTP/1.1

Host: www.example.com

Not all of the resources are returned to the client in the first call for data. The requests and responses will go back and forth between the server and client until the web browser has received all the resources necessary to render the contents of the HTML page on your screen. HTTP resides on top of the internet protocol stack i.e., application layer. HTTP/1.1 introduced persistent connections, HTTP/1.1 assumes that a TCP connection should be kept open unless directly told to close. This allows the client to send multiple requests along the same connection without waiting for a response to each, greatly improving the performance of HTTP/1.1 over HTTP/1.0.

Head-of-line (HOL) blocking was a significant problem with optimizing connection efficiency in HTTP/1.1. HOL is a situation in which a request at the head of the queue that cannot retrieve its required resource will block all the requests behind it. Adding separate, parallel TCP connections could alleviate this issue, but there are limits to the number of concurrent TCP connections possible between a client and server, and each new connection requires significant resources.

**HTTP/2:** It was developed in 2015 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. The conversion of messages into binary allows HTTP/2 to try new approaches to data delivery not available in HTTP/1.1

🡪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 supports multiplexing, which allows the client to construct multiple streams in parallel, these streams only need to make use of a single TCP connection. Having a single persistent connection per origin improves upon HTTP/1.1 by reducing the memory and processing footprint throughout the network. This results in better network and bandwidth utilization and thus decreases the overall operational cost.

🡪It also supports stream prioritization, When a client sends concurrent requests to a server, it can prioritize the responses it is requesting by assigning a weight between 1 and 256 to each stream. The higher number indicates higher priority. In addition to this, the client also states each stream’s dependency on another stream by specifying the ID of the stream on which it depends. If the parent identifier is omitted, the stream is considered to be dependent on the root stream. The server uses the information to create a dependency tree, which allows the server to determine the order in which the requests will retrieve their data.

🡪It introduces the concept of a server push where the server anticipates the resources that will be required by the client and pushes them prior to the client making requests. The client retains the authority to deny the server push; however, in most cases, this feature adds a lot of efficiency to the process.

The key differences HTTP/2 has to HTTP/1.x are as follows:

* It is binary, instead of textual
* It is fully multiplexed, instead of ordered and blocking.
* It can use one connection for parallelism.
* It uses header compression to reduce overhead.
* It allows Server Pushing to add responses proactively into the Browser cache.

**Blog 2: Objects and it’s internal representation in JavaScript.**

In any language, objects are the representation of real-world entities by defining it by properties and values. For example, in real world- table is an object having properties- length, width and height. In JavaScript, it is in key: value pair. An Object can have a number of data types inside it as values. It can have **numbers, string, arrays** and even**another Object**inside it.

Ways to create an object in javascript are as follows-

1. Object literal: object literal is a comma-separated list of key-value pairs wrapped in curly braces.

*var car={id:1 , name:’abc’ , display:function() }*

Property values can be of any data type, including array literals, functions, nested object literals, or primitive data type.

1. **Object.create():** This method creates a new object, using an existing object as the prototype of the newly created object.

var car2 = Object.create(car);

1. **Object constructor**:This is used when we require to create multiple objects of similar type.

Constructor function:

function Student(name, age, eye) {  
this.Name = name;  
this.age = age;  
this.eyeColor = eye;  
}

creating objects using constructor-

var a1= new Student(“John”, 15, “blue”);  
var a2= new Student(“Shally”, 17, “green”);

## Object.assign(): It is used to copy the values and properties from one or more source objects to a target object. Example- *Input* : var obj1 = { a: 10 }; var obj2 = { b: 20 }; var obj3 = { c: 30 }; var new\_obj = Object.assign(obj1, obj2, obj3); console.log(new\_obj); *Output :* Object { a: 10, b: 20, c: 30 }

## Object.fromentries: This method converts list of key-value pairs into an object. For example:

## const entries = new car([ [‘id’, 4], [‘color’, ‘blue’] ]);

const car1= Object.fromEntries(entries);

console.log(car1);  
*output:* Object { id: 4, color: ‘blue’}

*Note: JavaScript is an object-oriented language. It does allow you to create hierarchy of objects and to have inheritance of properties and their values by mainly constructor functions instead of classes. We don’t have classes in JS.*