**TASK – 1**

**1) Write a blog on Difference between HTTP1.1 vs HTTP2**

|  |  |
| --- | --- |
| **HTTP/1.1** | **HTTP/2** |
| It works on the textual format. | It works on the binary protocol. |
| There is head of line blocking that blocks all the requests behind it until it doesn’t get its all resources. | There is head of line blocking that blocks all the requests behind it until it doesn’t get its all resources. |
| It uses requests resource Inlining for use getting multiple pages | It uses PUSH frame by server that collects all multiple pages |
| It compresses data by itself | It uses HPACK for data compression. |

**2) Objects and its internal representation in JavaScript**

Objects, in JavaScript, is it’s most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript’s primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of “key: value” pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

For Eg. If your object is a student, it will have properties like name, age, address, id, etc and methods like updateAddress, updateNam, etc.

**Representation**

let’s **create an object** named **myperonal** and give it properties named **name, year, and batch** as follows:

**var myperonal = new Object();  
myperonal.name = 'Priyanka';  
myperonal.year = 2023;  
myperonal.batch = ‘B48WET’;**

Unassigned properties of an object are [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) (and not [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null)).

**myperonal.age; // undefined**

Properties of JavaScript objects can also be accessed or set using a bracket notation. Objects are sometimes called *associative arrays*, since each property is associated with a string value that can be used to access it. So, for example, you could **access the properties of the myperonal object as follows:**

**myperonal[‘name’] = ‘Priyanka’;  
myperonal[‘year’] = 2023;  
myperonal[‘batch’] = ‘B48WET’;**

An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation. This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime). Examples are as follows:

**// four variables are created and assigned in a single go,   
// separated by commas  
var myObj = new Object(),  
str = 'myString',  
rand = Math.random(),  
obj = new Object();  
myObj.type = 'Dot syntax';  
myObj['date created'] = 'String with space';  
myObj[str] = 'String value';  
myObj[rand] = 'Random Number';  
myObj[obj] = 'Object';  
myObj[''] = 'Even an empty string'**

You can also access properties by using a string value that is stored in a variable:

**var propertyName = 'name';  
myCar[propertyName] = 'Priyanka';**

**propertyName = 'Batch';  
myCar[propertyName] = 'B48WET';**