1.Write a blog on Difference between HTTP1.1 VS HTTP2.

Differences between HTTP1.1 Vs HTTP2:

HTTP 1.1:

HTTP 1.1, the first version of HTTP, was introduced in 1997. It presented significant performance optimizations (over HTTP/0.9 and HTTP/1.0) and transformed the way requests and responses were exchanged between clients and servers.

Key Features of HTTP/1.1:

It was no longer required for each connection to be terminated immediately after every request was served with a response; instead, with the keep-alive header, it was possible to have persistent connections. It allowed multiple requests/responses per TCP connection.

The Upgrade header was used to indicate a preference from the client that made it possible to switch to a more preferred protocol if found appropriate by the server.

HTTP/1.1 provided support for chunk transfers that allowed streaming of content dynamically as chunks and for additional headers to be sent after the message body. This enhancement was particularly useful in cases where values of a field remained unknown until the content had been produced. For example, when the content had to be digitally signed, it was not possible to do so before the entire content gets generated.

Other features that reinforced its stability were introduced such as:

pipelining (the second request is sent before the response to the first is adequately served)

content negotiation (an exchange between client and server to determine the media type, it also provides the provision to serve different versions of a resource at the same URI)

cache control (used to specify caching policies in both requests and responses)Objects and its Internal representation in Javascript:

HTTP/2:

At the beginning of 2010, Google introduced an experimental protocol, SPDY, which supported multiplexing (multiple requests/responses sent and received asynchronously over a single TCP connection) but as it gained traction IETF’s HTTP Working Group came up with HTTP/2 in 2015, which is based on the SPDY protocol.

Key Features of HTTP/2:

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.

Introduces the concept of multiplexing that interleaves the requests and responses without head-of-line blocking and does so over a single TCP connection.

It is a binary protocol i.e. only binary commands in the form of 0s and 1s are transmitted over the wire. The binary framing layer divides the message into frames that are segregated based on their type – Data or Header. This feature greatly increases efficiency in terms of security, compression and multiplexing.

HTTP/2 uses HPACK header compression algorithm that is resilient to attacks like CRIME and utilizes static Huffman encoding.

2.Write a blog about objects and its internal representation in Javascript.

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 are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.

An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don’t actually store the value.

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

**Objects and Properties**

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot-notation**:**

**objectName.propertyName**

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let’s create an object named myCar and give it properties named make, model, and year as follows:

**var myCar = new Object();**

**myCar.make = 'Ford';**

**myCar.model = 'Mustang';**

**myCar.year = 1969;**

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)).

**myCar.color; // undefined**

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see [property accessors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_Accessors)). 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 myCar object as follows:

**myCar['make'] = 'Ford';**

**myCar['model'] = 'Mustang';**

**myCar['year'] = 1969;**

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';**

**console.log(myObj);**

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

**var propertyName = 'make';**

**myCar[propertyName] = 'Ford';propertyName = 'model';**

**myCar[propertyName] = 'Mustang';**

# **Creating Objects In JavaScript :**

# **Create JavaScript Object with Object Literal:**

One of easiest way to create a javascript object is object literal, simply define the property and values inside curly braces as shown below:

**let bike = {name: 'SuperSport', maker:'Ducati', engine:'937cc'};**

# **Create JavaScript Object with Constructor:**

Constructor is nothing but a function and with help of new keyword, constructor function allows to create multiple objects of same flavor as shown below:

**function Vehicle(name, maker) {**

**this.name = name;**

**this.maker = maker;**

**}**

**let car1 = new Vehicle(’Fiesta’, 'Ford’);**

**let car2 = new Vehicle(’Santa Fe’, 'Hyundai’)**

**console.log(car1.name); //Output: Fiesta**

**console.log(car2.name); //Output: Santa Fe**

# **Using the JavaScript Keyword new**

The following example also creates a new JavaScript object with four properties:

**Example**

**var person = new Object();**

**person.firstName = “John”;**

**person.lastName = “Doe”;**

**person.age = 50;**

**person.eyeColor = “blue”;**

# **Using the object.create method**

Objects can also be created using the [Object.create()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create) method. This method can be very useful, because it allows you to choose the prototype object for the object you want to create, without having to define a constructor function.

**//Animal properties and method encapsulation**

**var Animal = {**

**type: 'Invertebrates', // Default value of properties**

**displayType: function() { // Method which will display type of Animal**

**console.log(this.type);**

**}**

**};**

**// Create new animal type called animal1**

**var animal1 = Object.create(Animal);**

**animal1.displayType(); // Output:Invertebrates**

**// Create new animal type called Fishes**

**var fish = Object.create(Animal);**

**fish.type = 'Fishes';**

**fish.displayType();**

**// Output:Fishes**