**Write a blog on differences between HTTP 1.1 vs. HTTP 2.**

HTTP stands for Hyper Text Transfer Protocol. It is a set of rules that runs on top of the TCP/IP suite of protocols and defines how files are to be transferred between clients and servers on the World Wide Web.

|  |  |
| --- | --- |
| **HTTP 1.1** | **HTTP 2** |
| HTTP 1.1 supports connection reuse i.e. for every TCP connection there could be multiple requests and responses, and pipelining where the client can request several resources from the server at once. However, pipelining was hard to implement due to issues such as head-of-line blocking and was not a feasible solution. | HTTP 2 uses multiplexing, where over a single TCP connection resources to be delivered are interleaved and arrive at the client almost at the same time. It is done using streams which can be prioritized, can have dependencies and individual flow control. It also provides a feature called server push that allows the server to send data that the client will need but has not yet requested. |
| Introduces a warning header field to carry additional information about the status of a message. Can define 24 status codes, error reporting is quicker and more efficient. | Underlying semantics of HTTP such as headers, status codes remains the same. |
| It is relatively secure since it uses digest authentication, NTLM authentication. | Security concerns from previous versions will continue to be seen in HTTP 2. However, it is better equipped to deal with them due to new TLS features like connection error of type inadequate security. |
| HTTP 1.1 was developed in the year 1997 | HTTP 2 was developed in the year 2015 |
| Expands on the caching support by using additional headers like cache-control, conditional headers like If-Match and by using entity tags. | HTTP/2 does not change much in terms of caching. With the server push feature if the client finds the resources are already present in the cache, it can cancel the pushed stream. |
| HTTP/1.1 provides faster delivery of web pages and reduces web traffic as compared to HTTP/1.0. However, TCP starts slowly and with domain sharing (resources can be downloaded simultaneously by using multiple domains), connection reuse and pipelining, there is an increased risk of network congestion. | HTTP/2 utilizes multiplexing and server push to effectively reduce the page load time by a greater margin along with being less sensitive to network delays. |

**Write a blog about objects and its internal representation in JavaScript.**

In JavaScript, an object is a standalone entity, with properties and type. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics.

**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 names 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 (and not null).

myCar.color; // undefined

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see 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).

Please note that all keys in the square bracket notation are converted to string unless they’re Symbols, since JavaScript object property names (keys) can only be strings or Symbols (at some point, private names will also be added as the class fields proposal progresses, but you won’t use them with [] form). When the key obj is added to the myObj, JavaScript will call the [obj.toString()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString" \t "_blank) method, and use this result string as the new key.

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';

You can use the bracket notation with for…in to iterate over all the enumerable properties of an object. To illustrate how this works, the following function displays the properties of the object when you pass the object and the object's name as arguments to the function.

Starting with ECMAScript 5, there are three native ways to list/traverse object properties:

* for...in loops  
  This method traverses all enumerable properties of an object and its prototype chain
* Object.keys(o)  
  This method returns an array with all the own (not in the prototype chain) enumerable properties' names ("keys") of an object o.
* Object.getOwnPropertyNames(o)  
  This method returns an array containing all own properties' names (enumerable or not) of an object o.