**DIFFERNCE BETWEEN HTTP1.1 AND HTTP 2**

Hypertext Transfer Protocol (HTTP) is a set of standards allowing internet users to exchange website information.

HTTP/1.1 was the third version of HTTP and the standard protocol for over 15 years. It introduced persistent connections for improved performance and laid the foundation for standard requests, such as GET, HEAD, PUT, and POST.

HTTP/2 was released in 2015 as a major revision to the HTTP/1.1 protocol.HTTP/2 improved on HTTP/1.1 in a number of ways that allowed for speedier content delivery and improved user experience.

**1.MULTIPLEXING:** In HTTP1.1, the server responds to the 1st client request with the help of one TCP connection and then accepts the second client request and responds to it with another TCP connection, that means a packet at the front of the line blocks others from being transmitted. While HTTP 2 is multiplexed i.e., it can initiate multiple requests in parallel over a single TCP connection. As a result, web pages containing several elements are delivered over one TCP connection.

**2. PROTOCOLS:** HTTP 1.1 uses textual protocols while HTTP 2 uses binary protocols that consume less bandwidth, are more efficiently parsed and are less error-prone than the textual protocols used by HTTP/1.1. Additionally, they can better handle elements such as whitespace, capitalization and line endings.

3.HEADER COMPRESSION: HTTP 2 uses header compression to reduce the overhead caused by TCP’s slow-start mechanism.

4.SERVER PUSH: HTTP 2 servers push likely-to-be-used resources into a browser’s cache, even before they’re requested. This allows browsers to display content without additional request cycles.

5.INCREASED SECURITY: Web browsers only support HTTP/2 via encrypted connections, increasing user and application security.

**OBJECT AND ITS 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 Eg. 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. 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;**

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';**

So, the function call **showProps(myCar, "myCar")** would return the following:

**myCar.make = Ford**

**myCar.model = Mustang**

**myCar.year = 1969**