DIFFERENCE BETWEEN HTTP1.1 AND HTTP2

HTTP 1.1

* Developed by Timothy Berners-Lee in 1989 as a communication standard for the World Wide Web
* HTTP is a top-level application protocol that exchanges information between a client computer and a local or remote web server
* In this process, 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
* In response to this request, the example.com web server returns an HTML page to the requesting client, in addition to any images, stylesheets, or other resources called for in the HTML.
* Note that 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.
* You can think of this exchange of requests and responses as a single *application layer* of the internet protocol stack, sitting on top of the *transfer layer* (usually using the Transmission Control Protocol, or TCP) and *networking layers* (using the Internet Protocol, or IP)

HTTP 2

* HTTP/2 began as the SPDY protocol, developed primarily at Google with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization in May 2015.
* From a technical point of view, one of the most significant features that distinguishes HTTP/1.1 and HTTP/2 is the binary framing layer, which can be thought of as a part of the application layer in the internet protocol stack.
* 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.
* An application level API would still create messages in the conventional HTTP formats, but the underlying layer would then convert these messages into binary.
* This ensures that web applications created before HTTP/2 can continue functioning as normal when interacting with the new protocol.
* The conversion of messages into binary allows HTTP/2 to try new approaches to data delivery not available in HTTP/1.1

Objects and its internal representation in Javascript

* Objects are accessed using dot notation or brackets either
* 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

* Objects are represented in key and value pairs
* Objects fall under reference data types and follows copy by reference that

is while assigning a the value to the other object the address of the object

is stored in the memory location rather than the value of the object

* 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)).
* Objects are created using
  + Object Literals
  + New keyword
  + Constructors
* One of easiest way to create an object is object literal, simply define the property

and values inside curly braces as shown below

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

* Constructors is special way of assigning values to the objects
* Objects can also be created using new keywords. The new keyword allocates the

memory to the objects

var person = new Object();

person.firstName = “John”;

person.lastName = “Doe”;

person.age = 50;

person.eyeColor = “blue”;

* Object.keys returns us the array of the keys defined in the object whereas

Object.value returns us the array of the values defined in the object

* Object.assign assigns the values of the created object to the other defined object

rather than storing its address or pointing to the same address