**Q.1) To read:**

🡺 I have read all the given links.

**Q.2) Write a blog on Difference between HTTP1.1 vs HTTP2**

Blog link: https://medium.com/@arjunlaxane/difference-between-http-1-1-vs-http-2-8b5a7013577e

Launch of HTTP/1.1 in 1989 had made it de facto standard of communication for web communications. But gradually as the webpages started getting more complex and resource demanding. It compelled developers to strive for new features to overcome HTTP/1.1 limitations. Thus, came in 2015 HTTP/2. HTTP/2 aims to be faster and more efficient than HTTP/1.1.

Let’s have look at difference between HTTP/1.1 and HTTP/2:

**1.Binary framing layer:**

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, while still maintaining HTTP semantics, such as verbs, methods, and headers.

The conversion of messages into binary allows HTTP/2 to try new approaches to data delivery not available in HTTP/1.1, a contrast that is at the root of the practical differences between the two protocols.

**2.HOL and TCP connection:**

As opposed to HTTP/1.1, which must make use of multiple TCP connections to lessen the effect of HOL (head-of-line) blocking, HTTP/2 establishes a single connection object between the two machines. Within this connection there are multiple streams of data. Each stream consists of multiple messages in the familiar request/response format. Finally, each of these messages split into smaller units called frames.

**3.Prioritization:**

HTTP/2 allows for detailed control over which page resources will load first. E.g. which images or videos will load first after the page has been loaded. This helps to reduce webpage load. Thus, making downloading fast.

**4.Multiplexing:**

In HTTP/1.1 resources are loaded in sequence, thus if one resource is blocked then all resources behind it will get blocked. Let’s take an example of lane between cities X and Y.

On the other hand, HTTP/2 sets up multiple streams of data. For e.g. multilane highway between cities X and Y.

Multiplexing resolves the head-of-line blocking issue in HTTP/1.1 by ensuring that no message has to wait for another to finish. This also means that servers and clients can send concurrent requests and responses, allowing for greater control and more efficient connection management.

Since multiplexing allows the client to construct multiple streams in parallel, these streams only need to make use of a single TCP connection. Having a single persistent connection per origin improves upon HTTP/1.1 by reducing the memory and processing footprint throughout the network. This results in better network and bandwidth utilization and thus decreases the overall operational cost.

**5.Buffer Overflow:**

As HTTP/1.1 relies on the transport layer to avoid buffer overflow, each new TCP connection requires a separate flow control mechanism. HTTP/2, however, multiplexes streams within a single TCP connection, and will have to implement flow control in a different manner.

**5.Compression:**

HTTP/2 uses a compression technique called HPACK, which eliminates redundancy from HTTP headers leading to fewer packets per webpage. This webpage loads quickly.

**5.Resource request prediction:**

In HTTP/1.1, if the developer knows in advance which additional resources the client machine will need to render the page, they can use a technique called resource in-lining.To include the required resource directly within the HTML document that the server sends in response to the initial GET request.

Since HTTP/2 enables multiple concurrent responses to a client’s initial GET request, a server can send a resource to a client along with the requested HTML page, providing the resource before the client asks for it. This process is called server push. In this way, an HTTP/2 connection can accomplish the same goal of resource in-lining while maintaining the separation between the pushed resource and the document. This means that the client can decide to cache or decline the pushed resource separate from the main HTML document, fixing the major drawback of resource in-lining.

**Q.3) Write a blog about objects and its internal representation in Javascript**

Blog link: https://medium.com/@arjunlaxane/objects-and-its-internal-representation-in-javascript-7f22f00cb3ed

> A JavaScript object is a collection of named values having state and behavior (properties and method).

> Objects has keys and values like key: value

Syntax:

Const <Object-Name> = {key1: value1, key2: value2….};

>Thus, JavaScript objects are like containers for values.

> In real life , a car is an object. A car has properties like weight, colour, model and methods like start, stop, on and off

>All cars have the same properties, but the property values differ from car to car. All cars have the same methods, but the methods are performed at different times.

> Javascript variables are containers for data value.

> Objects are variable too. But object can contain many values.

> Spaces and line breaks aren’t important. An object definition can span multiple lines.

Objects:

The following code assigns a **simple value** (Rolls Royce wraith) to a **variable** named rolls\_Royce\_wraith.

Objects are variables too. But objects can contain many values.

Const rolls\_Royce\_wraith ={

model: “2013 model”,

ARAI\_Mileage: “10.2 kmph”,

max\_power: “591 bph 5000–5500rpm”,

engine\_displacement= 6592,

body\_type : “coupe”,

fuel\_type = “petrol”,

transmission\_type = “Automatic”,

fuel\_tank\_capacity = 83.0,

ground\_clearance\_unladen = “150mm”;

number\_of\_cylinder = 12

} ;

Here, key : values pairs are called properties.

From the above object, let’s have a look what falls under property and property value:

Properties: model, ARAI\_Mileage, max\_power, engine\_displacement, body\_type, fuel\_type, transmission\_type, fuel\_tank\_capacity, ground\_clearance\_unladen, number\_of\_cylinder.

Properties value: 10.2 kmph, 591 bph 5000–5500rpm”, 6592, coupe, petrol, Automatic, 83.0, 150mm, 12.

> The object properties can be different primitive values, other objects and functions.

> Properties can usually be changed, added, and deleted, but some are read only.

**Accessing object properties:**

1) rolls\_Royce\_wraith.body\_type; //coupe

2) rolls\_Royce\_wraith[“body\_type”] // coupe

**Object method:**

> Object can also have methods

> Methods are action that can be performed on objects.

> Methods are stored in properties as function definitions.

Const rolls\_Royce\_wraith ={

model: “2013 model”,

ARAI\_Mileage: “10.2 kmph”,

max\_power: “591 bph 5000–5500rpm”,

engine\_displacement:6592,

body\_type : “coupe”,

fuel\_type : “petrol”,

transmission\_type : “Automatic”,

fuel\_tank\_capacity : 83.0,

ground\_clearance\_unladen : “150mm”;

number\_of\_cylinder : 12,

body\_And\_Fuel: function(){

return this.body\_type+” “+this.fuel\_type;

}

} ;

Here, this refers to rolls\_royce\_wraith object.

this.body\_type means body\_type property of this.

this.fuel\_type means fuel\_type property of this.

# What is this?

In JS, this keyword refers to an object.

This keyword refers to different object depending on how it is used.

In an object method, this refers to the **object**.

Alone, this refers to the **global object**.

In a function, this refers to the **global object**.

In a function, in strict mode, this is undefined.

In an event, this refers to the **element** that received the event.

Methods like call(), apply(), and bind() can refer this to **any object**.

Adding new property to object:

rolls\_Royce\_wraith.ignition = “on”;

After consoling object rolls\_Royce\_wraith you will have, properties sequenced as alphabetically:

{

ARAI\_Mileage: “10.2 kmph”,

body\_type : “coupe”,

body\_And\_Fuel: function(){

return this.body\_type+” “+this.fuel\_type;

},

fuel\_type : “petrol”,

fuel\_tank\_capacity : 83.0,

ground\_clearance\_unladen : “150mm”,

ignition: “on”,

max\_power: “591 bph 5000–5500rpm”,

model: “2013 model”,

number\_of\_cylinder : 12,

transmission\_type : “Automatic”

}

Do not declare strings, numbers, and Booleans as object

When a JavaScript variable is declared with the keyword “new", the variable is created as an object:

x = new String(); // Declares x as a String object  
y = new Number(); // Declares y as a Number object  
z = new Boolean(); // Declares z as a Boolean object

Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

**Q.4) Codekata practice**

🡺 I have started practice on codekata