Write a blog on Difference between HTTP1.1 vs HTTP2

HTTP, or Hypertext Transfer Protocol, is the backbone of the World Wide Web.

HTTP/1.1: The Old Reliable

HTTP/1.1 has been around since 1999 and served as the foundation for the modern web. It is a text-based protocol, and communication between the client (usually a web browser) and the server is done using a series of request-response messages.

Limitations of HTTP/1.1:

Head-of-line blocking: In HTTP/1.1, each resource (like an image, stylesheet, or script) is requested sequentially.

Multiple connections: To overcome head-of-line blocking, browsers often open multiple connections to a server (usually six).

Redundant headers: In each request, HTTP/1.1 sends a lot of redundant information (headers) which increases overhead and wastes bandwidth.

No built-in compression: HTTP/1.1 doesn't natively support data compression, so resources must be manually compressed before transmission.

No support for multiplexing: Multiplexing allows multiple requests and responses to be processed in parallel over a single connection, improving efficiency.

HTTP/2: The Modern Web Standard

HTTP/2, developed by the Internet Engineering Task Force (IETF) and released in 2015, was designed to overcome the limitations of HTTP/1.1 and provide a more efficient web communication protocol.

Key Features of HTTP/2:

Binary Protocol: HTTP/2 uses a binary protocol instead of a text-based one. This reduces the size of headers and makes data transmission more efficient.

Header Compression: HTTP/2 supports header compression, reducing overhead. This is particularly beneficial for requests with a large number of headers.

Multiplexing: HTTP/2 allows multiple requests and responses to be multiplexed over a single connection. This means that resources can be loaded in parallel, improving page load times significantly.

Stream Prioritization: It enables the server to prioritize which resources to send first, ensuring that critical resources are delivered faster.

Server Push: HTTP/2 allows servers to push resources to the client before they are requested.

Reduced Latency: By addressing head-of-line blocking and minimizing the number of connections needed, HTTP/2 reduces latency and speeds up page loading times.

Write a blog about objects and its internal representation in Javascript

In JavaScript, Objects are referred to as associative arrays, dictionaries, or hash maps in other programming languages. They consist of key-value pairs, where each key is a string that uniquely identifies a property, and each value can be of any data type, including other objects.

```
const person = {
  firstName: "John",
  lastName: "Peter",
  age: 24,
  },
  Hi: function() {
    console.log(`Hi I am ${this.firstName} ${this.lastName}.`);
  }
};
```

In this example, a person is an object with various properties, including strings, numbers, an array, nested objects, and even a function.

Internal Representation of Objects

To understand how objects are internally represented in JavaScript, it's helpful to know about a few key concepts:

1. Object Properties and Property Descriptors

Each property in an object is associated with a property descriptor, which defines the characteristics of the property.The property descriptor can be accessed using the Object.getOwnPropertyDescriptor() method.

2. Prototypes and Inheritance

JavaScript objects are built upon a prototype-based inheritance model. This means that an object can inherit properties and methods from another object, referred to as its prototype.

3. Memory Management

JavaScript engines have garbage collectors that automatically free up memory used by objects that are no longer in use.

Accessing Object Properties

To access properties of an object in JavaScript, you can use dot notation or bracket notation. Here's how you can access properties from the person object defined earlier:

```
console.log(person.firstName); // Using dot notation
console.log(person['lastName']); // Using bracket notation
```

Object Methods

In addition to properties, objects can also contain methods, which are functions defined as object properties. You can invoke object methods like this:

```
person.Hi(); // Calling the method
```