1. Write a blog on Difference between HTTP1.1 vs HTTP2

HTTP/1.1 and HTTP/2 are both protocols used for transmitting data over the internet, but they differ significantly in terms of performance, efficiency, and features. Here’s a concise breakdown of their key disparities:

1. Multiplexing:

• HTTP/1.1 uses a series of sequential connections to transfer resources, which can lead to head-of-line blocking issues where subsequent requests are delayed by earlier ones.

• HTTP/2 supports multiplexing, allowing multiple requests and responses to be interleaved over a single TCP connection. This reduces latency and improves efficiency by eliminating the need for multiple connections.

2. Binary Protocol:

• HTTP/1.1 relies on text-based headers, which can be verbose and inefficient.

• HTTP/2 employs a binary framing layer for more efficient data transfer. This simplifies parsing and reduces overhead, leading to faster processing and lower latency.

3. Header Compression:

• HTTP/1.1 headers are sent as plain text, resulting in redundant information and increased bandwidth usage.

• HTTP/2 introduces header compression mechanisms, reducing the size of header fields and minimizing bandwidth consumption. This improves performance, especially for requests with large headers.

4. Server Push:

• HTTP/1.1 requires the client to explicitly request each resource, leading to additional round trips and delays.

• HTTP/2 supports server push, allowing the server to push resources to the client proactively without waiting for explicit requests. This optimizes resource delivery and can significantly reduce page load times.

Overall, HTTP/2 offers substantial improvements over HTTP/1.1 in terms of performance, efficiency, and features, making it the preferred protocol for modern web communication.

1. Write a blog about objects and its internal representation in Javascript.

Objects are fundamental to JavaScript, serving as the backbone of its dynamic and flexible nature. While they are often used intuitively, understanding their internal representation is crucial for harnessing their full potential. In this blog, we delve into the intricacies of objects in JavaScript, exploring how they are internally represented and manipulated.

Anatomy of Objects:

At its core, an object in JavaScript is a collection of key-value pairs, where each key is a unique identifier (or property) associated with a value. Properties can hold various data types, including primitive values, functions, or even other objects, making them incredibly versatile.

Internal Representation:

JavaScript engines typically use a variety of techniques to internally represent objects efficiently. While the exact implementation may vary across engines, common strategies include:

1. Dictionary: Objects are often implemented as dictionaries (or hash maps), where property names are stored as keys, and their corresponding values are stored as associated values. This allows for fast property access and dynamic addition/removal of properties.

2. Hidden Classes: JavaScript engines may use hidden classes (also known as shapes or prototypes) to optimize property access and memory usage. These hidden classes define the layout of objects and their properties, enabling efficient property lookup and storage.

3. Property Descriptors: Each property in JavaScript is associated with a property descriptor, which defines attributes such as value, writability, and configurability. These descriptors are used internally by the engine to control property behavior and access.