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

# **1.MULTIPLEXING AND CONCURRENT REQUEST:**

# HTTP/1.1: Under the HTTP/1.1 protocol, each request necessitates a separate TCP connection. This serial nature leads to inefficiencies, as subsequent requests must await the completion of preceding ones

# HTTP/2: In contrast, HTTP/2 introduces multiplexing, enabling multiple requests and responses to traverse a single TCP connection concurrently. This enhancement minimizes latency and optimizes resource utilization, particularly for websites abundant in assets.

**2.HEADER COMPRESSION:**

* HTTP/1.1: Headers accompanying HTTP/1.1 requests and responses lack compression, resulting in redundant data transmission. This inefficiency contributes to increased overhead and slower page loading times.
* HTTP/2: By employing header compression mechanisms like HPACK, HTTP/2 reduces the size of headers transmitted between client and server. This streamlined approach diminishes bandwidth consumption and accelerates the delivery of web content.

**3.SERVER PUSH CAPABILITY:**

* HTTP/1.1: Traditional HTTP/1.1 communication relies on the client's explicit requests to retrieve necessary resources, leading to suboptimal performance for complex web pages.
* HTTP/2: HTTP/2 introduces server push functionality, empowering servers to proactively dispatch resources to clients without awaiting solicitation. This preemptive approach bolsters page load times by preemptively delivering assets, bypassing latency-inducing round trips.

**4.BINARY PROTOCOL:**

* HTTP/1.1: HTTP/1.1 communicates using plain text for both requests and responses, which can be inefficient for transmitting large volumes of data.
* HTTP/2: Embracing a binary framing layer, HTTP/2 employs a compact representation for its protocol, enhancing parsing efficiency and reducing processing overhead. This binary protocol facilitates swifter data transmission and minimizes latency.

**5.BACKWARD COMPATIBILITY AND ADOPTION:**

* HTTP/1.1: As the incumbent protocol, HTTP/1.1 boasts widespread compatibility with existing web infrastructure and systems, ensuring seamless operation across diverse environments.
* HTTP/2: Although HTTP/2 offers compelling performance benefits, its adoption necessitates support from both client and server implementations. Nevertheless, the growing adoption of HTTP/2 among modern browsers and servers underscores its trajectory towards becoming the standard for web communication.

**2.Write a blog about objects and its internal representation in Javascript?**

**1.OBJECT BASICS:**

* In JavaScript, objects are collections of key-value pairs, where keys are strings (or symbols) and values can be of any data type, including other objects, functions, or primitive types. Objects can be created using object literals, constructor functions, or the Object. create() method.

**2.INTERNAL REPRESENTATION:**

* Under the hood, JavaScript engines typically use two main approaches for representing objects: dictionary-based and class-based representations.

* Dictionary-based representation involves storing object properties in a hashmap-like structure, where each property name maps to its corresponding value. This approach provides flexibility but may result in slower property access.

**3.PROPERTIES & PROTOTYPES:**

* JavaScript objects can have own properties, which are directly defined on the object itself, as well as prototype properties, which are inherited from a prototype object.
* Prototypal inheritance allows objects to inherit properties and methods from their prototype chain, providing a powerful mechanism for code reuse and abstraction.

**4.PROPERTY ACCESS & PERFORMANCE:**

* Property access in JavaScript involves a process called property resolution, where the engine traverses the prototype chain to find the property.
* Accessing own properties is generally faster than accessing prototype properties due to optimizations in modern JavaScript engines.
* Minimizing property access and avoiding unnecessary property lookups can improve performance, especially in performance-critical code.

**5.BEST PRACTICES:**

* When defining objects, consider the access patterns and use cases to choose the most suitable representation.
* Use object literals or factory functions for simple objects with a fixed set of properties.
* For objects with shared behavior, consider using constructor functions or classes to leverage prototypal inheritance.
* Profile and optimize performance-critical code by minimizing property access and leveraging engine optimizations