DAY-01 TASK

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

HTTP 2	HTTP 1.1
Multiplexing: Each request and response requires a separate connection, and they are processed sequentially.	**No Multiplexing:** Multiple requests and responses can be sent concurrently over a single connection, allowing for more efficient use of network resources
Header Compression: HTTP/2 uses header compression (HPACK) to reduce the overhead of sending headers with each request and response.	**No Header Compression:** HTTP/1.1 does not include header compression, so each request and response contain the full set of headers.
Binary Protocol: It uses a binary protocol. The binary format is more efficient to parse, resulting in faster and more compact communication between clients and servers.	**Text-Based Protocol:** HTTP/1.1 uses a text-based protocol, which is human-readable but can be less efficient in terms of parsing and transmission
Server Push: Servers in HTTP/2 can predict what you need and send it to you before you ask.	**No Server Push:** In HTTP/1.1, the server can't send things before you ask for them.
Stream Prioritization: It supports stream prioritization, enabling more important resources to be transmitted first.	**No Stream Prioritization:** HTTP/1.1 does not provide built-in mechanisms for stream prioritization. Requests are processed in the order they are received
Flow Control: It helps prevent data traffic jams by controlling how much information is sent and received at once.	**No Flow Control:** There's no built-in control to manage how much data is being sent or received.
Connection Multiplexing: It allows multiple streams of data to be sent and received over a single connection.	**No Connection Multiplexing:** Each connection in HTTP/1.1 is used for a single request and response pair. This can result in a higher number of connections
Header Deduplication: HTTP/2 reduces redundancy in headers by using header compression.	**No Header Deduplication:** It does not perform header deduplication, leading to the transmission of redundant header information.
TLS Usage Encouraged: TLS (Transport Layer Security) is optional and its usage is strongly encouraged	**Connection Keep-Alive:** It introduced the concept of connection keep-alive to reuse a connection for multiple requests,
Backward Compatibility: HTTP/2 is designed to work with older systems	**Upgrade to WebSocket:** It can be used for WebSocket communication, it requires an upgrade from HTTP to WebSocket.

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

Objects

- Objects, in JavaScript, is it's most important data-type and forms the building blocks for modern JavaScript.
- JavaScript. These objects are quite different from JavaScript's primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).
- Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.
- An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don't actually store the value.

Creating Objects

You can create objects in JavaScript using either object literals **{}** or the **Object** constructor. The literal syntax is more common and concise, making it the preferred method for most developers

Example : var person = {

name : "gomathy",

age : 20,

gender : "female"

state : "tamil nadu"

}

Create JavaScript Object with Constructor

Constructor is nothing but a function and with help of new keyword, constructor function allows to create multiple objects of same flavor as shown below

Example

```
function Vehicle(name, maker) {

this.name = name;

this.maker = maker;

}

let car1 = new Vehicle('Fiesta', 'Ford');

let car2 = new Vehicle('Santa Fe', 'Hyundai')
```

```
console.log(car1.name);    //Output: Fiesta
console.log(car2.name);    //Output: Santa Fe
```

Using the JavaScript Keyword new

The following example also creates a new JavaScript object with four properties:

Example:

```
var person = new Object();
person.firstName = “John”;
person.lastName = “Doe”;
person.age = 50;
person.eyeColor = “blue”
```

The syntax for adding a property to an object is :

```
ObjectName.ObjectProperty = propertyValue;
```

The syntax for deleting a property from an object is:

```
delete ObjectName.ObjectProperty;
```

The syntax to access a property from an object is:

```
objectName.property
//or
objectName["property”]
//or
objectName[expression]
```

The Internal Representation:

Properties and Methods:

Internally, JavaScript objects store properties and methods as key-value pairs. Properties hold data values, while methods are functions associated with the object. This distinction is vital for understanding an object's internal structure.

Prototypes and Inheritance:

JavaScript objects support prototypal inheritance, allowing them to inherit properties and methods from other objects. Each object has a prototype, which can be another object or null. This mechanism forms the basis of JavaScript's object-oriented programming.

Hidden Classes:

JavaScript engines use a concept called hidden classes to optimize object creation and property access. These hidden classes define the internal structure of objects, making property access more efficient by reducing the need for runtime checks.

Property Descriptors:

Properties in JavaScript objects have associated property descriptors, defining attributes such as whether a property is writable, enumerable, or configurable. The "Object.getOwnPropertyDescriptor()" method provides insight into these descriptors.

Example:

```
const person = { name: 'John Doe' };

const descriptor = Object.getOwnPropertyDescriptor(person, 'name');

console.log(descriptor);
```

Garbage Collection:

When an object is no longer referenced, it becomes eligible for garbage collection, freeing up memory resources.

Memory Leaks:

Improper handling of objects can lead to memory leaks, where unused objects persist in memory.