1. Difference between HTTP1.1 vs HTTP2

**HTTP/1.1:** Loads resources one after the other, so if one resource cannot be loaded, it blocks all the other resources behind it.

It loads the data slower.

**HTTP/2:** It is able to use a single TCP connection to send multiple streams of data at once so that no one resource blocks any other resource.

It loads the data faster than HTTP/1.1.

1. Objects and its internal representation in Javascript

## **What Are Objects in JavaScript?**

In JavaScript, objects are data structures that allow you to store and organize data in a flexible manner. They are collections of key-value pairs, where keys are strings (or symbols in modern JavaScript) that act as property names, and values can be of any data type, including other objects, functions, and more. Objects can represent a wide range of entities and concepts, making them a versatile tool for developers.

Here's a basic example of an object in JavaScript:

let person = {

name: "John",

age: 30,

sayHello: function () {

console.log("Hello!");

},

};

In this example, **person** is an object with three properties: **name**, **age**, and **sayHello**. The **name** and **age** properties store data, while **sayHello** is a method (a function stored as a property) that can be invoked.

## **Internal Representation of Objects**

To understand how JavaScript represents objects internally, let's explore some key concepts:

### **1. Properties and Methods**

Objects consist of properties and methods. Properties are key-value pairs, and methods are functions stored as properties of an object. Properties define the object's characteristics, while methods define its behavior.

### **2 .Object Representation**

Internally, JavaScript engines use efficient data structures to represent objects. While the exact details can vary among JavaScript engines (e.g., V8, SpiderMonkey), objects are often represented as dictionaries or hash maps. In these data structures, property names (keys) are mapped to their corresponding values.

Functions, which are objects themselves, have additional internal data, including the code to execute and a reference to their lexical environment (closures). This allows functions to capture and maintain their surrounding

* 3.Object creation

****Object.create()****: The **Object.create()** method allows you to create objects with a specified prototype. This is a more flexible way of creating objects, especially when dealing with inheritance.

// Creating an object using a constructor function

function Person(name, age) {

this.name = name;

this.age = age;

}

let person1 = new Person("Alice", 25);

// Creating an object using Object.create()

let car = Object.create(null);

car.make = "Toyota";

car.model = "Camry";

### **4. Dynamic Properties**

JavaScript allows you to dynamically add, modify, or delete properties from objects at runtime. This dynamic nature is one of the language's strengths and enables you to adapt objects to changing requirements.

let car = {};

car.make = "Toyota"; // Adding a property

car.model = "Camry";

delete car.make; // Deleting a property

### **5. Object Serialization**

JavaScript objects can be converted to JSON strings using **JSON.stringify()**. This is useful for data interchange between different systems or for saving data to a file.

let person = { name: "Alice", age: 25 };

let jsonPerson = JSON.stringify(person);

console.log(jsonPerson); // Outputs: '{"name":"Alice","age":25}'