Name: Aswin Babu M J

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

HTTP/1.1

The first usable version of HTTP was created in 1997. Because it went through several stages of development, this first version of HTTP was called HTTP/1.1. This version is still in use on the web.

HTTP/2

In 2015, a new version of HTTP called HTTP/2 was created. HTTP/2 solved several problems that the creators of HTTP/1.1 did not anticipate. In particular, HTTP/2 is much faster and more efficient than HTTP/1.1. One of the ways in which HTTP/2 is faster is in how it prioritises content during the loading process.

The other differences between HTTP/2 and HTTP/1.1 that impact performance

Multiplexing: HTTP/1.1 loads resources one after the other, so if one resource cannot be loaded, it blocks all the other resources behind it. In contrast, HTTP/2 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. HTTP/2 does this by splitting data into binary-code messages and numbering these messages so that the client knows which stream each binary message belongs to.

Server push: Typically, a server only serves content to a client device if the client asks for it. However, this approach is not always practical for modern web pages, which often involve several dozen separate resources that the client must request. HTTP/2 solves this problem by allowing a server to "push" content to a client before the client asks for it. The server also sends a message letting the client know what pushed content to expect – like if Bob had sent Alice a Table of Contents of his novel before sending the whole thing.

Header compression: Small files load more quickly than large ones. To speed up web performance, both HTTP/1.1 and HTTP/2 compress HTTP messages to make them smaller. However, HTTP/2 uses a more advanced compression method called HPACK that eliminates redundant information in HTTP header packets. This eliminates a few bytes from every HTTP packet. Given the volume of HTTP packets involved in loading even a single webpage, those bytes add up quickly, resulting in faster loading.

2. Objects and internal representation in Javascript

Objects, in JavaScript, is it's most important data-type and forms the building blocks for modern 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.
Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of "key: value" pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.
For Eg. If your object is a student, it will have properties like name, age, address, id, etc and methods like updateAddress, updateNam, etc.
Objects and properties
A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot-notation:
objectName.propertyName
Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let's create an object named myCar and give it properties named make, model, and year as follows:

```
var myCar = new Object();
myCar.make = 'Ford';
```

myCar.model = 'Mustang';
myCar.year = 1969;
Unassigned properties of an object are undefined (and not null).
myCar.color; // undefined
Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see property accessors). Objects are sometimes called associative arrays, since each property is associated with a string value that can be used to access it. So, for example, you could access the properties of the myCar object as follows:
myCar['make'] = 'Ford';
myCar['model'] = 'Mustang';
myCar['year'] = 1969;
An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation. This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime). Examples are as follows:
// four variables are created and assigned in a single go,
// separated by commas
var myObj = new Object(),
  str = 'myString',
  rand = Math.random(),
  obj = new Object();
myObj.type   = 'Dot syntax';
myObj['date created']   = 'String with space';
myObj[str]   = 'String value';
myObj[rand]   = 'Random Number';
myObj[obj]   = 'Object';
myObj['']   = 'Even an empty string';console.log(myObj);
You can also access properties by using a string value that is stored in a variable:
var propertyName = 'make';
myCar[propertyName] = 'Ford';propertyName = 'model';
myCar[propertyName] = 'Mustang';
You can use the bracket notation with for...in to iterate over all the enumerable properties of an object. To illustrate how this works, the following function displays the properties of the object when you pass the object and the object's name as arguments to the function:
function showProps(obj, objName) {
 var result = ``;
 for (var i in obj) {
  // obj.hasOwnProperty() is used to filter out properties from the object's prototype chain

```
  if (obj.hasOwnProperty(i)) {
    result += `${objName}.${i} = ${obj[i]}\n`;
  }
 }
 return result;
}
```

So, the function call showProps(myCar, "myCar") would return the following:

myCar.make = Ford
myCar.model = Mustang
myCar.year = 1969

Creating Objects In JavaScript :

Create JavaScript Object with Object Literal

One of easiest way to create a javascript object is object literal, simply define the property and values inside curly braces as shown below

```
let bike = {name: 'SuperSport', maker:'Ducati', engine:'937cc'};
```

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

```
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";
```

Using the Object.create method

Objects can also be created using the Object.create() method. This method can be very useful, because it allows you to choose the prototype object for the object you want to create, without having to define a constructor function.

```
// Animal properties and method encapsulation
var Animal = {
 type: 'Invertebrates', // Default value of properties
 displayType: function() {  // Method which will display type of Animal
   console.log(this.type);
```

```
 }
};
// Create new animal type called animal1
var animal1 = Object.create(Animal);
animal1.displayType(); // Output:Invertebrates
// Create new animal type called Fishes
var fish = Object.create(Animal);
fish.type = 'Fishes';
fish.displayType();
// Output:Fishes
```