1. **HTTP/1.1 VS HTTP/2.0**

**HTTP**

* Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989.
* Development of early HTTP Requests for Comments (RFCs) was a coordinated effort by the Internet Engineering Task Force (IETF) and the World Wide Web (WWW), with work later moving to the IETF.
* HTTP is a protocol which allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser.
* A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more.

**HTTP/1.1**

* **HTTP/1.1** was first documented in 1997.
* Caching
* Hierarchical proxy support
* Persistent connections
* Virtual host support
* TCP inefficiencies
* Authentication
* Issues with increasing size of content

**HTTP/2.0 or HTTP/2**

* **HTTP/2.0 or HTTP/2** is a major revision of the HTTP network protocol used by the World Wide Web. It was derived from the earlier experimental SPDY protocol, originally developed by Google.
* HTTP/2 was developed by the HTTP Working Group (also called httpbis, where "bis" means "second") of the Internet Engineering Task Force. HTTP/2 is the first new version of HTTP since HTTP 1.1, which was standardized in 1997.
* The Working Group presented HTTP/2 to IESG for consideration as a Proposed Standard in December 2014, and IESG approved it to publish as Proposed Standard on February 17, 2015. The HTTP/2 specification was published on May 14, 2015.
* Mobile network latency
* Residual TCP inefficiencies carried forward with HTTP/1.1
* The standardization effort was supported by Chrome, Opera, Firefox, Internet Explorer 11, Safari, Amazon Silk, and Edge browsers. Most major browsers had added HTTP/2 support by the end of 2015.
* HTTP/2 can send multiple requests for data in parallel over a single TCP connection. This is the most advanced feature of the HTTP/2 protocol because it allows you to download web files asynchronously from one server. Most modern browsers limit TCP connections to one server.
* HTTP/1.1 and HTTP/2 share semantics, ensuring that the requests and responses traveling between the server and client in both protocols reach their destinations as traditionally formatted messages with headers and bodies, using familiar methods like GET and POST.
* But while HTTP/1.1 transfers these in plain-text messages, HTTP/2 encodes these into binary, allowing for significantly different delivery model possibilities.

1. **Object and its internal representations**

**Objects:**

Objects in JavaScript, just as in many other programming languages, can be compared to objects in real life. The concept of objects in JavaScript can be understood with real life, tangible objects.

In JavaScript, an object is a standalone entity, with properties and type. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics.

**Internal Representation of object**

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.

An object can be created with figure brackets {…} with an optional list of properties. A property is a “key: value” pair, where a key is a string (also called a “property name”), and value can be anything.

To understand this rather abstract definition, let us look at an example of a JavaScript Object :

let school = {

name : “Vivekananda School”,

location : “Delhi”,

established : “1971”

}

In the above example “name”, “location”, “established” are all “keys” and “Vivekananda School”, “Delhi” and 1971 are values of these keys respectively.

Each of these keys is referred to as properties of the object. An object in JavaScript may also have a function as a member, in which case it will be known as a method of that object.

**Properties of JavaScript Object**

The property names can be strings or numbers. In case the property names are numbers, they must be accessed using the “bracket notation” like this :

Let us see such an example :

// javascript code demonstrating a simple object

let school = {

name: ‘Vivekananda School’,

location : ‘Delhi’,

established : ‘1971’,

displayInfo : function(){

console.log(${school.name} was established

in ${school.established} at ${school.location});

}

}

school.displayInfo();

Output:

Vivekananda School was established in 1971 at Delhi

Property names can also be strings with more than one space separated words. In which case, these property names must be enclosed in quotes :

let school = {

“school name” : “Vivekananda School”,

}

Like property names which are numbers, they must also be accessed using the bracket notation.

**Creating Objects**

There are several ways or syntax’s to create objects. One of which, known as the Object literal syntax, we have already used. Besides the object literal syntax, objects in JavaScript may also be created using the constructors, Object Constructor or the prototype pattern.

Using the Object literal syntax : Object literal syntax uses the {…} notation to initialize an object an its methods/properties directly.

Let us look at an example of creating objects using this method :

var obj = {

member1 : value1,

member2 : value2,

};

These members can be anything — strings, numbers, functions, arrays or even other objects. An object like this is referred to as an object literal. This is different from other methods of object creation which involve using constructors and classes or prototypes, which have been discussed below.

Object Constructor : Another way to create objects in JavaScript involves using the “Object” constructor. The Object constructor creates an object wrapper for the given value. This, used in conjunction with the “new” keyword allows us to initialize new objects.

**Example :**

const school = new Object();

school.name = ‘Vivekanada school’;

school.location = ‘Delhi’;

school.established = 1971;

school.displayInfo = function(){

console.log(${school.name} was established

in ${school.established} at ${school.location});

}

school.displayInfo();

**Constructors:** Constructors in JavaScript, like in most other OOP languages, provides a template for creation of objects. In other words, it defines a set of properties and methods that would be common to all objects initialized using the constructor.

Let us see an 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

It creates a new object and sets the constructor property of the object to schools (It is important to note that this property is a special default property that is not enumerable and cannot be changed by setting a “constructor: someFunction” property manually).

Then, it sets up the object to work with the Vehicle function’s prototype object ( Each function in JavaScript gets a prototype object, which is initially just an empty object but can be modified.The object, when instantiated inherits all properties from its constructor’s prototype object).

Then calls Vehicle() in the context of the new object, which means that when the “this” keyword is encountered in the constructor(vehicle()), it refers to the new object that was created in the first step.

Once this is finished, the newly created object is returned to car1 and car2(in the above example).

Inside classes, there can be special methods named constructor().

class people {

constructor()

{

this.name = “Adam”;

}

}

let person1 = new people();

// Output : Adam

console.log(person1.name);

**Prototypes :** Another way to create objects involves using prototypes. Every JavaScript function has a prototype object property by default(it is empty by default). Methods or properties may be attached to this property. A detailed description of prototypes is beyond the scope of this introduction to objects.

However you may familiarize yourself with the basic syntax used as below:

let obj = Object.create(prototype\_object, propertiesObject)

// the second propertiesObject argument is optional

An example of making use of the Object.create() method is:

let footballers = {

position: “Striker”

}

let footballer1 = Object.create(footballers);

// Output : Striker

console.log(footballer1.position);

All objects created in this way inherits all properties and methods from its prototype objects. Prototypes can have prototypes and those can have prototypes and so on. This is referred to as prototype chaining in JavaScript. This chain terminates with the Object.prototype which is the default prototype fallback for all objects. Javascript objects, by default, inherit properties and methods from Object.prototype but these may easily be overridden. It is also interesting to note that the default prototype is not always Object.prototype.For example Strings and Arrays have their own default prototypes — String.prototype and Array.prototype respectively.

**Accessing Object Members:**

Object members(properties or methods) can be accessed using the

dot notation :

(objectName.memberName)

let school = {

name : “Vivekanada”,

location : “Delhi”,

established : 1971,

20 : 1000,

displayinfo : function() {

console.log(${school.name} was established

in ${school.established} at ${school.location});

}

}

console.log(school.name);

// Output: 1000

console.log(school[‘20’]);