**HTTP 1.1 vs HTTP 2**

**HTTP:**

HTTP (Hypertext Transfer Protocol) is a set of rules that runs on top of the TCP/IP suite of protocols and defines how files are to be transferred between clients and servers on the world wide web.

| **HTTP 1.1** | **HTTP 2** |
| --- | --- |
| Introduced in the year **1997**. It’s the first usable form of HTTP. It’s an upgrade to HTTP 1.0 | Introduced in the year **2015** by Google as an update to HTTP 1.1. |
| It allowed multiple requests/responses per TCP connection.  The Upgrade header was used to indicate a preference from the client that made it possible to switch to a more preferred protocol if found appropriate by the server.  HTTP/1.1 provided support for chunk transfers that allowed streaming of content dynamically as chunks and for additional headers to be sent after the message body. This enhancement was particularly useful in cases where values of a field remained unknown until the content had been produced. For example, when the content had to be digitally signed, it was not possible to do so before the entire content gets generated.  Other features that reinforced its stability were introduced such as:   * **pipelining** (the second request is sent before the response to the first is adequately served) * **content negotiation** (an exchange between client and server to determine the media type, it also provides the provision to serve different versions of a resource at the same URI) * **cache control** (used to specify caching policies in both requests and responses) | In addition to the features of HTTP 1.1, HTTP 2 allows us to prioritize which part of the webpage has to be loaded first.  When a [client](https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/) makes a request for a webpage, the server sends several streams of data to the client at once, instead of sending one thing after another. This method of data delivery is known as **Multiplexing**. Developers can assign each of these data streams a different weighted value, and the value tells the client which data stream to render first. 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 webpages, 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. |

**Objects**

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.

Objects may contain any combination of the primitive data-types (number, string, undefined, boolean, null, symbol) 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.

**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 (object property shorthand). 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

**Creation of Objects:**

A new object can be created using curly braces, new operator, object constructor and Object.create() method.

* var myCar = new Object();
* var myCar = {};
* function Vehicle(name, maker) {

this.name = name;

this.maker = maker;

}

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

* var Animal = {

type: 'Invertebrates', displayType: function() { console.log(this.type);

}

};

var animal1 = Object.create(Animal);