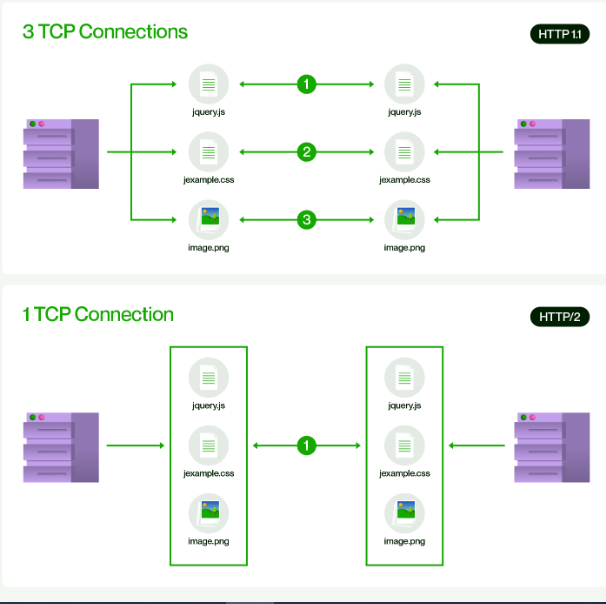
**Difference between HTTP1.1 vs HTTP2**



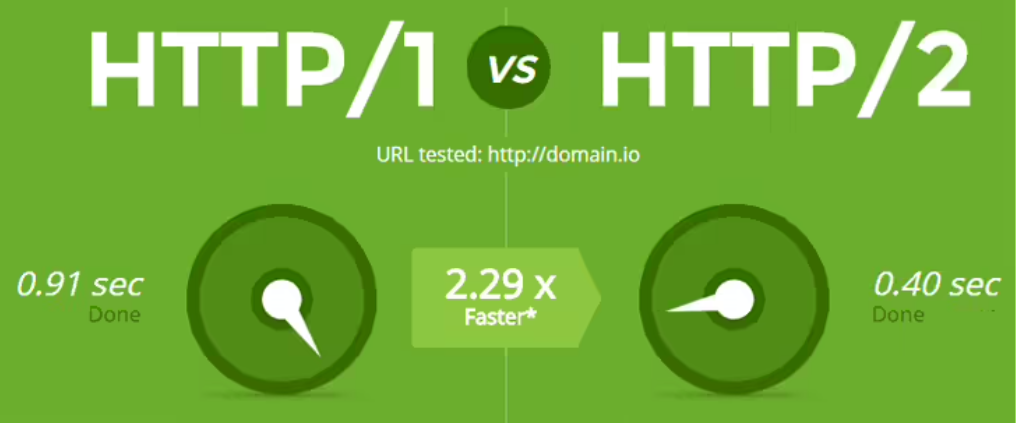
**HTTP** stands for **hypertext transfer protocol** & it is used in client-server communication. By using HTTP user sends the request to the server & the server sends the response to the user. There are several stages of development of HTTP but we will focus mainly on HTTP/1.1 which was created in 1997 & the new one is HTTP/2 which was created in 2015.

**HTTP/1.1:**

* For better understanding, let’s assume the situation when you make a request to the server for the example.html page & server responds to you as a resource example.html page.
* before sending the request and the response there is a TCP connection established between client & server. again you make a request to the server for image img.jpg & the server gives a response as an image img.jpg. the connection was not lost here after the first request because we add a keep-alive header which is the part of the request so there is an open connection between the server & client.
* there is a persistent connection which means several requests & responses are merged in a single connection.
* These are the drawbacks that lead to the creation of HTTP/2: The first problem is HTTP/1.1 transfer all the requests & responses in the plain text message form.
* The second one is head of line blocking in which TCP connection is blocked all other requests until the response does not receive.
* all the information related to the header file is repeated in every request.

**HTTP/2:**

* HTTP/2 was developed over the SPDY protocol.
* HTTP/2 works on the binary framing layer instead of textual that converts all the messages in binary format.
* it works on fully multiplexed that is one TCP connection is used for multiple requests.
* HTTP/2 uses HPACK which is used to split data from header.
* it compresses the header.
* The server sends all the other files like CSS & JS without the request of the client using the PUSH frame.



| **HTTP/1.1** | **HTTP/2** |
| --- | --- |
| It works on the textual format. | It works on the binary protocol. |
| There is head of line blocking that blocks all the requests behind it until it doesn’t get its all resources. | It allows multiplexing so one TCP connection is required for multiple requests. |
| It uses requests resource Inlining for use getting multiple pages | It uses PUSH frame by server that collects all multiple pages |
| It compresses data by itself. | It uses HPACK for data compression. |

## Object

* **In JavaScript, objects are king. If you understand objects, you understand JavaScript.**

In JavaScript, almost "everything" is an object.

* Booleans can be objects (if defined with the new keyword)
* Numbers can be objects (if defined with the new keyword)
* Strings can be objects (if defined with the new keyword)
* Dates are always objects
* Maths are always objects
* Regular expressions are always objects
* Arrays are always objects
* Functions are always objects
* Objects are always objects

JavaScript Primitives

A **primitive value** is a value that has no properties or methods.

**3.14** is a primitive value

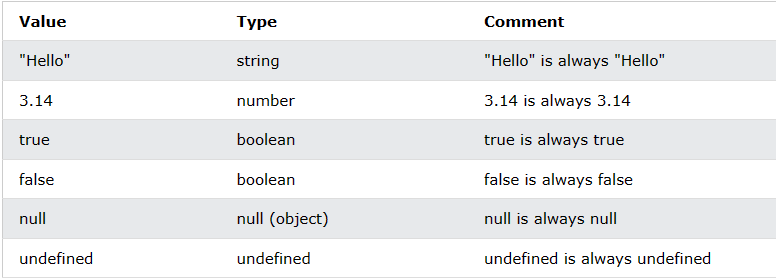
A **primitive data type** is data that has a primitive value.

JavaScript defines 7 types of primitive data types:

Examples

* string
* number
* boolean
* null
* undefined
* symbol
* bigint

Primitive values are immutable (they are hardcoded and cannot be changed).



Objects are variables too. But objects can contain many values.

Object values are written as **name : value** pairs (name and value separated by a colon).

A JavaScript object is a collection of **named values**

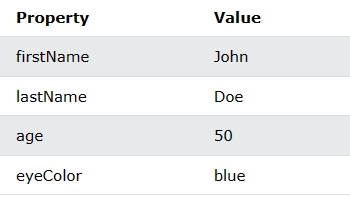
It is a common practice to declare objects with the const keyword.

Example:

const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

## Object Properties

The named values, in JavaScript objects, are called **properties**.

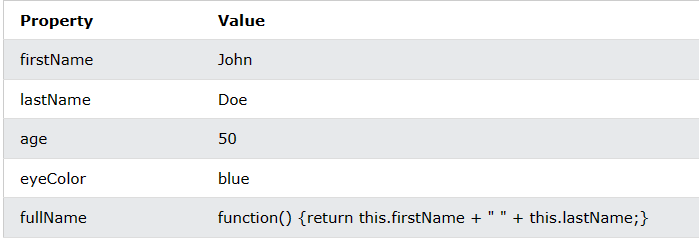


## Object Methods

Methods are **actions** that can be performed on objects.

Object properties can be both primitive values, other objects, and functions.

An **object method** is an object property containing a **function definition**.



JavaScript objects are containers for named values, called properties and methods.

## Creating a JavaScript Object

There are different ways to create new objects:

* Create a single object, using an object literal.
* Create a single object, with the keyword new.
* Define an object constructor, and then create objects of the constructed type.
* Create an object using Object.create().

### Example 1:

const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

### Example 2:

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};

### Example 3:

const person = {};  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";

### Example 4:

const person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";

But there is no need to use new Object().

For readability, simplicity and execution speed, use the object literal method(Example 1).

## JavaScript Objects are Mutable

Objects are mutable: They are addressed by reference, not by value.

If person is an object, the following statement will not create a copy of person:

const x = person;  // Will not create a copy of person.

The object x is **not a copy** of person. It **is** person. Both x and person are the same object.

Any changes to x will also change person, because x and person are the same object.

### Example :

const person = {  
  firstName:"John",  
  lastName:"Doe",  
  age:50, eyeColor:"blue"  
}  
  
const x = person;  
x.age = 10;

# JavaScript Object Properties

Properties are the most important part of any JavaScript object.

Properties can usually be changed, added, and deleted, but some are read only.

Accessing JavaScript Properties

*objectName.property*// person.age

or

*objectName*["*property*"]   // person["age"]

or

*objectName*[*expression*]   // x = "age"; person[x]

JavaScript for...in Loop

The JavaScript for...in statement loops through the properties of an object.

for (let *variable* in *object*) {  
*// code to be executed*  
}