1. **Write a blog on Difference between HTTP1.1 vs HTTP2**

**Ans:**

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 geeksforgeeks.html page & server responds to you as a resource geeksforgeeks.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.

**Difference between HTTP/1.1 and HTTP/2 are:**

| **HTTP/1.1** | **HTTP/2** |
| --- | --- |
| It uses 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. |

1. Write a blog about objects and its internal representation in JavaScript

**Ans:**

**Object**

The **Object** type represents one of [JavaScript's data types](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures). It is used to store various keyed collections and more complex entities. Objects can be created using the [Object()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/Object) constructor or the [object initializer / literal syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer).

[**Description**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#description)

Nearly all [objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#objects) in JavaScript are instances of [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object); a typical object inherits properties (including methods) from Object.prototype, although these properties may be shadowed (a.k.a. overridden). The only objects that don't inherit from Object.prototype are those with [null prototype](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#null-prototype_objects), or descended from other null prototype objects.

Changes to the Object.prototype object are seen by **all** objects through prototype chaining, unless the properties and methods subject to those changes are overridden further along the prototype chain. This provides a very powerful although potentially dangerous mechanism to override or extend object behavior. To make it more secure, Object.prototype is the only object in the core JavaScript language that has [immutable prototype](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/setPrototypeOf#description) — the prototype of Object.prototype is always null and not changeable.

[**Object prototype properties**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#object_prototype_properties)

You should avoid calling any Object.prototype method, especially those that are not intended to be polymorphic (i.e. only its initial behavior makes sense and no descending object could override it in a meaningful way). All objects descending from Object.prototype may define a custom own property that has the same name, but with entirely different semantics from what you expect. Furthermore, these properties are not inherited by [null-prototype objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#null-prototype_objects). All modern JavaScript utilities for working with objects are [static](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#static_methods). More specifically:

* [valueOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/valueOf), [toString()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString), and [toLocaleString()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toLocaleString) exist to be polymorphic and you should expect the object to define its own implementation with sensible behaviors, so you can call them as instance methods. However, valueOf() and toString() are usually implicitly called through [type conversion](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#type_coercion) and you don't need to call them yourself in your code.
* [\_\_defineGetter\_\_()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/__defineGetter__), [\_\_defineSetter\_\_()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/__defineSetter__), [\_\_lookupGetter\_\_()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/__lookupGetter__), and [\_\_lookupSetter\_\_()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/__lookupSetter__) are deprecated and should not be used. Use the static alternatives [Object.defineProperty()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty) and [Object.getOwnPropertyDescriptor()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyDescriptor) instead.
* The [\_\_proto\_\_](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/proto) property is deprecated and should not be used. The [Object.getPrototypeOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getPrototypeOf) and [Object.setPrototypeOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/setPrototypeOf) alternatives are static methods.
* The [propertyIsEnumerable()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/propertyIsEnumerable) and [hasOwnProperty()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwnProperty) methods can be replaced with the [Object.getOwnPropertyDescriptor()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyDescriptor) and [Object.hasOwn()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwn) static methods, respectively.
* The [isPrototypeOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/isPrototypeOf) method can usually be replaced with [instanceof](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/instanceof), if you are checking the prototype property of a constructor.

In case where a semantically equivalent static method doesn't exist, or if you really want to use the Object.prototype method, you should directly [call()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call) the Object.prototype method on your target object instead, to prevent the object from having an overriding property that produces unexpected results.

const obj = {

foo: 1,

// You should not define such a method on your own object,

// but you may not be able to prevent it from happening if

// you are receiving the object from external input

propertyIsEnumerable() {

return false;

},

};

obj.propertyIsEnumerable("foo"); // false; unexpected result

Object.prototype.propertyIsEnumerable.call(obj, "foo"); // true; expected result

Copy to Clipboard

[**Deleting a property from an object**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#deleting_a_property_from_an_object)

There isn't any method in an Object itself to delete its own properties (such as [Map.prototype.delete()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/delete)). To do so, one must use the [delete operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete).

[**null-prototype objects**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#null-prototype_objects)

Almost all objects in JavaScript ultimately inherit from Object.prototype (see [inheritance and the prototype chain](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain)). However, you may create null-prototype objects using [Object.create(null)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create) or the [object initializer syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer) with \_\_proto\_\_: null (note: the \_\_proto\_\_ key in object literals is different from the deprecated [Object.prototype.\_\_proto\_\_](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/proto) property). You can also change the prototype of an existing object to null by calling [Object.setPrototypeOf(obj, null)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/setPrototypeOf).

const obj = Object.create(null);

const obj2 = { \_\_proto\_\_: null };

Copy to Clipboard

An object with a null prototype can behave in unexpected ways, because it doesn't inherit any object methods from Object.prototype. This is especially true when debugging, since common object-property converting/detecting utility functions may generate errors, or lose information (especially if using silent error-traps that ignore errors).

For example, the lack of [Object.prototype.toString()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString) often makes debugging intractable:

const normalObj = {}; // create a normal object

const nullProtoObj = Object.create(null); // create an object with "null" prototype

console.log(`normalObj is: ${normalObj}`); // shows "normalObj is: [object Object]"

console.log(`nullProtoObj is: ${nullProtoObj}`); // throws error: Cannot convert object to primitive value

alert(normalObj); // shows [object Object]

alert(nullProtoObj); // throws error: Cannot convert object to primitive value

Copy to Clipboard

Other methods will fail as well.

normalObj.valueOf(); // shows {}

nullProtoObj.valueOf(); // throws error: nullProtoObj.valueOf is not a function

normalObj.hasOwnProperty("p"); // shows "true"

nullProtoObj.hasOwnProperty("p"); // throws error: nullProtoObj.hasOwnProperty is not a function

normalObj.constructor; // shows "Object() { [native code] }"

nullProtoObj.constructor; // shows "undefined"

Copy to Clipboard

We can add the toString method back to the null-prototype object by assigning it one:

nullProtoObj.toString = Object.prototype.toString; // since new object lacks toString, add the original generic one back

console.log(nullProtoObj.toString()); // shows "[object Object]"

console.log(`nullProtoObj is: ${nullProtoObj}`); // shows "nullProtoObj is: [object Object]"

Copy to Clipboard

Unlike normal objects, in which toString() is on the object's prototype, the toString() method here is an own property of nullProtoObj. This is because nullProtoObj has no (null) prototype.

In practice, objects with null prototype are usually used as a cheap substitute for [maps](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map). The presence of Object.prototype properties will cause some bugs:

const ages = { alice: 18, bob: 27 };

function hasPerson(name) {

return name in ages;

}

function getAge(name) {

return ages[name];

}

hasPerson("hasOwnProperty"); // true

getAge("toString"); // [Function: toString]

Copy to Clipboard

Using a null-prototype object removes this hazard without introducing too much complexity to the hasPerson and getAge functions:

const ages = Object.create(null, {

alice: { value: 18, enumerable: true },

bob: { value: 27, enumerable: true },

});

hasPerson("hasOwnProperty"); // false

getAge("toString"); // undefined

Copy to Clipboard

In such case, the addition of any method should be done cautiously, as they can be confused with the other key-value pairs stored as data.

Making your object not inherit from Object.prototype also prevents prototype pollution attacks. If a malicious script adds a property to Object.prototype, it will be accessible on every object in your program, except objects that have null prototype.

const user = {};

// A malicious script:

Object.prototype.authenticated = true;

// Unexpectedly allowing unauthenticated user to pass through

if (user.authenticated) {

// access confidential data

}

Copy to Clipboard

JavaScript also has built-in APIs that produce null-prototype objects, especially those that use objects as ad hoc key-value collections. For example:

* The return value of [Array.prototype.group()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/group)
* The groups and indices.groups properties of the result of [RegExp.prototype.exec()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/exec)
* [Array.prototype[@@unscopables]](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/@@unscopables) (all @@unscopables objects should have null-prototype)
* [import.meta](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/import.meta)
* Module namespace objects, obtained through [import \* as ns from "module";](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import#namespace_import) or [import()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/import)

The term "null-prototype object" often also includes any object without Object.prototype in its prototype chain. Such objects can be created with [extends null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/extends#extending_null) when using classes.

[**Object coercion**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#object_coercion)

Many built-in operations that expect objects first coerce their arguments to objects. [The operation](https://tc39.es/ecma262/#sec-toobject) can be summarized as follows:

* Objects are returned as-is.
* [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) and [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/null) throw a [TypeError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypeError).
* [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean), [Symbol](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol), [BigInt](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt) primitives are wrapped into their corresponding object wrappers.

The best way to achieve the same effect in JavaScript is through the [Object()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/Object) constructor. Object(x) converts x to an object, and for undefined or null, it returns a plain object instead of throwing a [TypeError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypeError).

Places that use object coercion include:

* The object parameter of [for...in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in) loops.
* The this value of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) methods.
* Parameters of Object methods such as [Object.keys()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys).
* Auto-boxing when a property is accessed on a primitive value, since primitives do not have properties.
* The [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this) value when calling a non-strict function. Primitives are boxed while null and undefined are replaced with the [global object](https://developer.mozilla.org/en-US/docs/Glossary/Global_object).

Unlike [conversion to primitives](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#primitive_coercion), the object coercion process itself is not observable in any way, since it doesn't invoke custom code like toString or valueOf methods.

[**Constructor**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#constructor)

[Object()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/Object)

Turns the input into an object.

[**Static methods**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#static_methods)

[Object.assign()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign)

Copies the values of all enumerable own properties from one or more source objects to a target object.

[Object.create()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create)

Creates a new object with the specified prototype object and properties.

[Object.defineProperty()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty)

Adds the named property described by a given descriptor to an object.

[Object.defineProperties()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperties)

Adds the named properties described by the given descriptors to an object.

[Object.entries()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/entries)

Returns an array containing all of the [key, value] pairs of a given object's **own** enumerable string properties.

[Object.freeze()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze)

Freezes an object. Other code cannot delete or change its properties.

[Object.fromEntries()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/fromEntries)

Returns a new object from an iterable of [key, value] pairs. (This is the reverse of [Object.entries](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/entries)).

[Object.getOwnPropertyDescriptor()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyDescriptor)

Returns a property descriptor for a named property on an object.

[Object.getOwnPropertyDescriptors()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyDescriptors)

Returns an object containing all own property descriptors for an object.

[Object.getOwnPropertyNames()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyNames)

Returns an array containing the names of all of the given object's **own** enumerable and non-enumerable properties.

[Object.getOwnPropertySymbols()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertySymbols)

Returns an array of all symbol properties found directly upon a given object.

[Object.getPrototypeOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getPrototypeOf)

Returns the prototype (internal [[Prototype]] property) of the specified object.

[Object.is()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/is)

Compares if two values are the same value. Equates all NaN values (which differs from both IsLooselyEqual used by [==](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Equality) and IsStrictlyEqual used by [===](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Strict_equality)).

[Object.isExtensible()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/isExtensible)

Determines if extending of an object is allowed.

[Object.isFrozen()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/isFrozen)

Determines if an object was frozen.

[Object.isSealed()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/isSealed)

Determines if an object is sealed.

[Object.keys()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys)

Returns an array containing the names of all of the given object's **own** enumerable string properties.

[Object.preventExtensions()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/preventExtensions)

Prevents any extensions of an object.

[Object.seal()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/seal)

Prevents other code from deleting properties of an object.

[Object.setPrototypeOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/setPrototypeOf)

Sets the object's prototype (its internal [[Prototype]] property).

[Object.values()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/values)

Returns an array containing the values that correspond to all of a given object's **own** enumerable string properties.

[**Instance properties**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#instance_properties)

[Object.prototype.constructor](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/constructor)

Specifies the function that creates an object's prototype.

[Object.prototype.\_\_proto\_\_](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/proto) Deprecated

Points to the object which was used as prototype when the object was instantiated.

[**Instance methods**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#instance_methods)

[Object.prototype.\_\_defineGetter\_\_()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/__defineGetter__)

Associates a function with a property that, when accessed, executes that function and returns its return value.

[Object.prototype.\_\_defineSetter\_\_()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/__defineSetter__)

Associates a function with a property that, when set, executes that function which modifies the property.

[Object.prototype.\_\_lookupGetter\_\_()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/__lookupGetter__)

Returns the function bound as a getter to the specified property.

[Object.prototype.\_\_lookupSetter\_\_()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/__lookupSetter__)

Returns the function bound as a setter to the specified property.

[Object.prototype.hasOwnProperty()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwnProperty)

Returns a boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain.

[Object.prototype.isPrototypeOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/isPrototypeOf)

Returns a boolean indicating whether the object this method is called upon is in the prototype chain of the specified object.

[Object.prototype.propertyIsEnumerable()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/propertyIsEnumerable)

Returns a boolean indicating whether the specified property is the object's [enumerable own](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Enumerability_and_ownership_of_properties) property.

[Object.prototype.toLocaleString()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toLocaleString)

Calls [toString()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString).

[Object.prototype.toString()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString)

Returns a string representation of the object.

[Object.prototype.valueOf()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/valueOf)

Returns the primitive value of the specified object.

[**Examples**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#examples)

[**Constructing empty objects**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#constructing_empty_objects)

The following examples store an empty Object object in o:

const o1 = new Object();

const o2 = new Object(undefined);

const o3 = new Object(null);

Copy to Clipboard

[**Using Object to create Boolean objects**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#using_object_to_create_boolean_objects)

The following examples store [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean) objects in o:

// equivalent to const o = new Boolean(true)

const o = new Object(true);

Copy to Clipboard

// equivalent to const o = new Boolean(false)

const o = new Object(Boolean());

Copy to Clipboard

[**Object prototypes**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#object_prototypes)

When altering the behavior of existing Object.prototype methods, consider injecting code by wrapping your extension before or after the existing logic. For example, this (untested) code will pre-conditionally execute custom logic before the built-in logic or someone else's extension is executed.

When modifying prototypes with hooks, pass this and the arguments (the call state) to the current behavior by calling apply() on the function. This pattern can be used for any prototype, such as Node.prototype, Function.prototype, etc.

const current = Object.prototype.valueOf;

// Since my property "-prop-value" is cross-cutting and isn't always

// on the same prototype chain, I want to modify Object.prototype:

Object.prototype.valueOf = function (...args) {

if (Object.hasOwn(this, "-prop-value")) {

return this["-prop-value"];

} else {

// It doesn't look like one of my objects, so let's fall back on

// the default behavior by reproducing the current behavior as best we can.

// The apply behaves like "super" in some other languages.

// Even though valueOf() doesn't take arguments, some other hook may.

return current.apply(this, args);

}

};