**Objetos en JavaScript**

Un objeto en JavaScript es una colección de propiedades, donde cada propiedad es una asociación entre una clave (o nombre de propiedad) y un valor. Los valores de las propiedades pueden ser de cualquier tipo de datos, incluidos otros objetos, funciones, y más.

**Características de los Objetos**

* **Propiedades:** Son pares clave-valor donde la clave es una cadena (string) o un símbolo (symbol), y el valor puede ser de cualquier tipo.
* **Métodos:** Los valores de las propiedades pueden ser funciones, que se denominan métodos cuando están asociados con un objeto.
* **No Ordenados:** A diferencia de los arreglos, los objetos no tienen un orden garantizado para sus propiedades.
* **Accesibles por Clave:** Las propiedades se acceden mediante notación de punto (object.property) o notación de corchetes (object['property']).
* **Herencia:** Los objetos en JavaScript heredan de un prototipo, lo que les permite tener propiedades y métodos heredados.

**Prototypes de Atributos:** Prototypes (o prototipos) son un mecanismo por el cual los objetos pueden heredar propiedades y métodos de otros objetos. Cada objeto en JavaScript tiene un prototipo, que actúa como un "objeto padre" del que puede heredar características. Cuando hablamos de "prototypes de atributos", nos referimos a las propiedades y métodos que un objeto puede heredar de su prototipo.

**[[Prototype]] interno:** Cada objeto en JavaScript tiene una referencia interna llamada [[Prototype]], que apunta a otro objeto o a null. Este objeto es el prototipo del objeto original. Si intentas acceder a una propiedad o método en un objeto y no se encuentra en ese objeto, JavaScript buscará la propiedad en su prototipo, y luego en el prototipo de su prototipo, hasta que encuentre la propiedad o alcance null.

**Object.prototype:** Todos los objetos en JavaScript descienden de Object.prototype en última instancia. Esto significa que los métodos y propiedades disponibles en Object.prototype están disponibles en casi todos los objetos.

**La Clase Object:** La clase Object es la raíz de todos los objetos. Todos los objetos en JavaScript son instancias de Object o heredan de él, ya sea directa o indirectamente. Esto significa que cualquier objeto que creas, ya sea un objeto literal, una instancia de una clase o un objeto creado con una función constructora, hereda propiedades y métodos de Object.

* **Métodos heredados:** Todos los objetos heredan métodos de “Object.prototype”, como “.toString()”, “.valueOf()”, y “.hasOwnProperty()”.
* **Personalización de métodos:** Puedes sobrescribir los métodos heredados en tus propios objetos para personalizar su comportamiento.
* **Funciones utilitarias:** “Object” también proporciona funciones estáticas útiles, como “Object.keys()”, “Object.values()”, “Object.assign()”, y “Object.create()”.

Texto

Descripción generada automáticamente

**Formas de Declarar Objetos**

1. **Literal de Objeto:** La forma más común y sencilla de crear un objeto es usando la notación literal de objeto. Solo en los objetos literales tenemos problemas con el contexto this.

Texto

Descripción generada automáticamente

1. **Constructor de Objeto:** Puedes crear un objeto utilizando el constructor Object. Esta forma es menos común para la creación simple de objetos, pero puede ser útil en ciertos contextos.

**Texto

Descripción generada automáticamente**

1. **Constructor de Función:** Puedes definir un constructor de función y usarlo para crear múltiples instancias de objetos con las mismas propiedades y métodos.

**Texto

Descripción generada automáticamente**

Cuando usas una función constructora, es común que quieras que todas las instancias compartan ciertos métodos, en lugar de definir esos métodos dentro de la función constructora misma, lo cual sería ineficiente y desperdiciaría memoria. Para lograr esto, los métodos se definen en el “prototype” de la función constructora.

Texto

Descripción generada automáticamente con confianza baja

**Encapsulamiento y Privacidad:** Las funciones constructoras en JavaScript no ofrecen encapsulamiento o privacidad de manera nativa como en otros lenguajes de programación. Sin embargo, se pueden utilizar

variables locales dentro de la función constructora que no estén adjuntas al objeto “this”para simular la privacidad.

1. **Método Object.create():** Puedes crear un nuevo objeto que herede de un objeto existente utilizando el método Object.create(). Esto es útil para establecer la herencia prototípica.

**Texto

Descripción generada automáticamente**

1. **Objetos con class:** Las clases proporcionan una sintaxis más moderna y limpia para crear objetos y manejar herencia.

**Texto

Descripción generada automáticamente**

**Métodos en Objetos**

Los objetos en JavaScript pueden contener métodos. Un método es una función que está asociada con un objeto. Los métodos permiten que los objetos realicen acciones utilizando sus propios datos. Un **método** es una función que está asociada con un objeto. En otras palabras, un método es una propiedad de un objeto cuyo valor es una función.

Texto

Descripción generada automáticamente

En el ejemplo anterior, start y displayInfo son métodos del objeto car. Los métodos pueden acceder a las propiedades del objeto utilizando this.

**Objetos Globales:** JavaScript proporciona varios objetos globales que están disponibles en todos los entornos de ejecución (navegadores, Node.js, etc.). Algunos ejemplos son:

**Object:** La clase base para todos los objetos en JavaScript.

**Array:** Para trabajar con colecciones de datos indexados.

**Function:** Para crear y manejar funciones.

**Math:** Para realizar operaciones matemáticas.

**Date:** Para manejar fechas y horas.

**JSON:** Para trabajar con datos JSON.

**window** (en navegadores): Representa la ventana del navegador y contiene propiedades y métodos relacionados con la ventana.

**Clases en JavaScript:** Las clases en JavaScript son una forma más reciente de trabajar con la orientación a objetos, introducidas en ECMAScript 2015 (ES6). Aunque son una abstracción sintáctica sobre el sistema de prototipos existente, ofrecen una sintaxis más familiar para los desarrolladores que vienen de otros lenguajes orientados a objetos. Vamos a profundizar en los aspectos clave de las clases, incluyendo atributos privados, y el uso de métodos “get” y “set”.

**Definición Básica de Clases:** Una clase en JavaScript se define usando la palabra clave “class”. Dentro de una clase, se pueden definir un constructor (para inicializar objetos) y métodos que operan sobre esos objetos.

**Texto

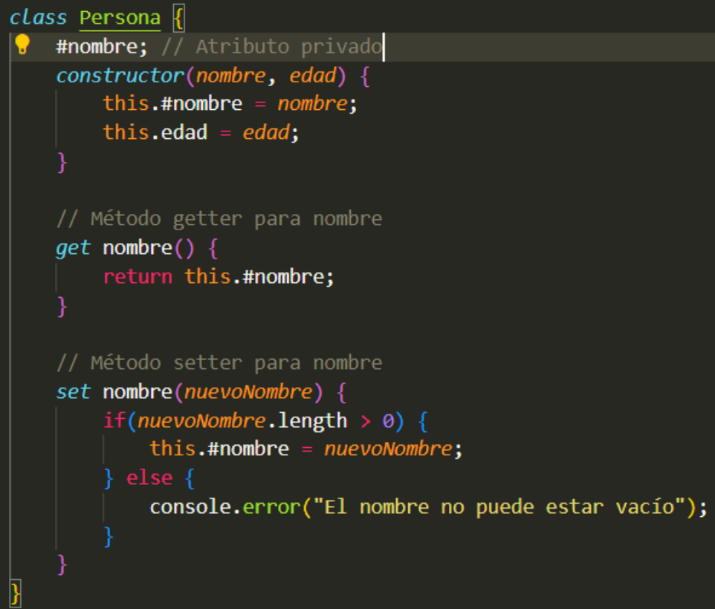
Descripción generada automáticamente**

**Atributos Privados:** Tradicionalmente, JavaScript no tenía soporte nativo para atributos privados, lo que significaba que todas las propiedades de un objeto eran públicas por defecto. Sin embargo, con la introducción de ECMAScript 2020, se introdujeron los campos privadosen las clases.

**Texto

Descripción generada automáticamente**

La propiedad “saldo” es completamente privada y no se puede acceder directamente fuera de la clase. Solo los métodos definidos dentro de la clase pueden interactuar con esta propiedad.

**Métodos “get” y “set”:** Los métodos “get” y “set” permiten controlar cómo se accede y modifica el valor de una propiedad. Son especialmente útiles para encapsular la lógica de acceso a propiedades y para validar o transformar los valores antes de asignarlos o devolverlos.

**Agregar una nueva propiedad a un Objeto**

* **Agregar Propiedad Directamente:** Solo afecta a la instancia específica del objeto.

**Imagen de la pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media**

* **Agregar Propiedad al Prototipo:** Afecta a todas las instancias (existentes y futuras) que comparten el mismo prototipo.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

**Eliminar la propiedad de un Objeto**

* **Usar el Operador delete:** El operador delete es la forma más directa de eliminar una propiedad de un objeto. El operador delete elimina una propiedad de un objeto y devuelve true si la propiedad ha sido eliminada con éxito.

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

* **Usar Reflect.deleteProperty():** La función Reflect.deleteProperty() es parte de la API de Reflect y proporciona una forma similar al operador delete para eliminar propiedades, pero con manejo de errores más explícito.

Texto

Descripción generada automáticamente