**¿Qué son los getters y setters?**

En esencia, son métodos especiales que se utilizan para obtener (getter) o establecer (setter) el valor de una propiedad de un objeto. Sin embargo, a diferencia de acceder directamente a una propiedad, los getters y setters te permiten realizar acciones adicionales antes o después de la asignación o lectura del valor.

**¿Por qué utilizarlos?**

* **Encapsulación:** Ocultan la implementación interna de un objeto, permitiendo modificar el comportamiento interno sin afectar al código externo que utiliza el objeto.
* **Validación:** Puedes validar los valores antes de asignarlos a una propiedad, asegurando que sean válidos y consistentes.
* **Cálculos:** Puedes realizar cálculos complejos o derivar valores a partir de otras propiedades del objeto.
* **Control de acceso:** Puedes restringir el acceso a ciertas propiedades o modificar su comportamiento en función del contexto.

**Ejemplo práctico:**

JavaScript

class Persona { constructor(nombre) { this.\_nombre = nombre; } get nombre() { return this.\_nombre.toUpperCase(); // Convertir el nombre a mayúsculas al obtenerlo } set nombre(nuevoNombre) { if (nuevoNombre.length > 0) { this.\_nombre = nuevoNombre; } else { console.error("El nombre no puede estar vacío"); } } } const persona1 = new Persona("Juan"); console.log(persona1.nombre); // Imprime: JUAN persona1.nombre = "Ana"; console.log(persona1.nombre); // Imprime: ANA persona1.nombre = ""; // No se asigna porque está vacío

Use code [with caution.](file:///C:/faq#coding)

**Explicación del ejemplo:**

* **\_nombre:** Es una propiedad privada que almacena el nombre real.
* **getter nombre:** Cuando se accede a persona1.nombre, se ejecuta este getter, convirtiendo el nombre a mayúsculas antes de devolverlo.
* **setter nombre:** Cuando se intenta asignar un nuevo valor a persona1.nombre, se ejecuta este setter, validando que el nuevo nombre no esté vacío. Si es válido, se asigna a la propiedad privada.

**Cuándo utilizarlos:**

* **Propiedades calculadas:** Cuando el valor de una propiedad se deriva de otras propiedades o cálculos.
* **Validación de datos:** Para asegurar la integridad de los datos antes de almacenarlos.
* **Control de acceso:** Para restringir el acceso a ciertas propiedades o modificar su comportamiento.
* **Encapsulación:** Para ocultar la implementación interna de un objeto.

**Consideraciones adicionales:**

* **Convenciones de nomenclatura:** Generalmente se utilizan nombres de propiedades sin guiones bajos para los getters y setters, y nombres con guiones bajos para las propiedades privadas.
* **Herramientas de desarrollo:** Los navegadores modernos y las herramientas de desarrollo te permiten inspeccionar los objetos y ver cómo funcionan los getters y setters.
* **JavaScript moderno:** Con ES6 y versiones posteriores, los getters y setters se han vuelto más fáciles de usar y son una parte integral de la programación orientada a objetos en JavaScript.