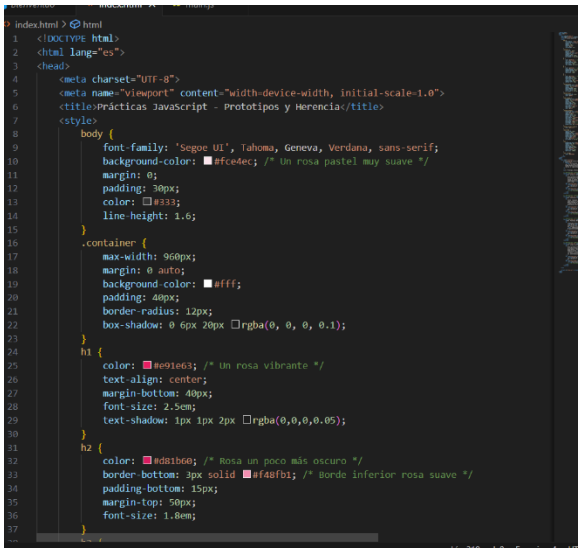
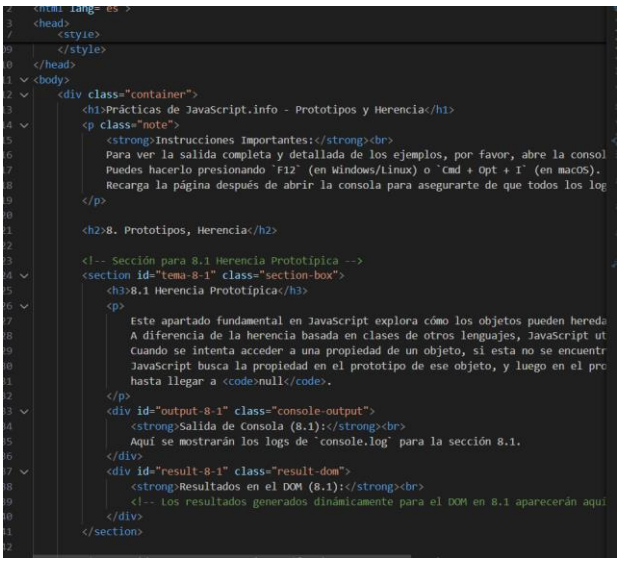
# Evidencias de prototipo, herencia



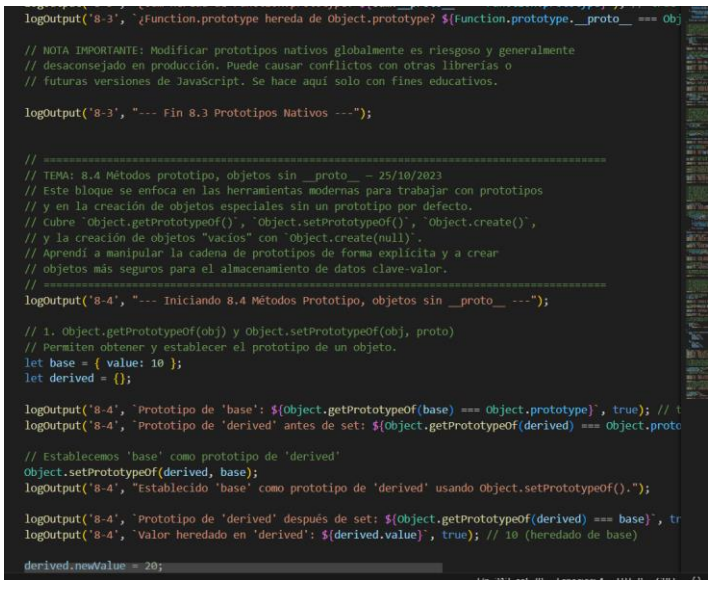
```
index.html > html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prácticas JavaScript - Prototipos y Herencia</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background-color: #fce4ec; /* Un rosa pastel muy suave */
            margin: 0;
            padding: 30px;
            color: #333;
            line-height: 1.6;
        }
        .container {
            max-width: 960px;
            margin: 0 auto;
            background-color: #fff;
            padding: 40px;
            border-radius: 12px;
            box-shadow: 0 6px 20px rgba(0, 0, 0, 0.1);
        }
        h1 {
            color: #e91e63; /* Un rosa vibrante */
            text-align: center;
            margin-bottom: 40px;
            font-size: 2.5em;
            text-shadow: 1px 1px 2px rgba(0,0,0,0.05);
        }
        h2 {
            color: #d81b60; /* Rosa un poco más oscuro */
            border-bottom: 3px solid #f48fb1; /* Borde inferior rosa suave */
            padding-bottom: 15px;
            margin-top: 50px;
            font-size: 1.8em;
        }
```



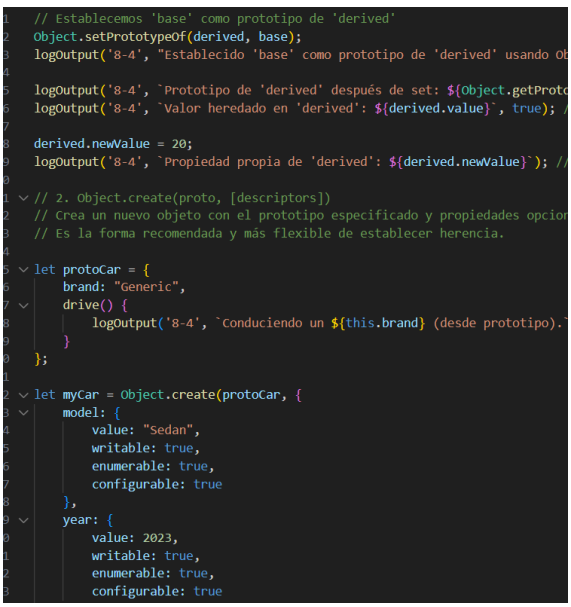
```
<html lang="es">
<head>
    <style>
    </style>
</head>
<body>
    <div class="container">
        <h1>Prácticas de JavaScript.info - Prototipos y Herencia</h1>
        <p class="note">
            <strong>Instrucciones Importantes:</strong><br>
            Para ver la salida completa y detallada de los ejemplos, por favor, abre la consol
            Puedes hacerlo presionando `F12` (en Windows/Linux) o `Cmd + Opt + I` (en macOS).
            Recarga la página después de abrir la consola para asegurarte de que todos los log
        </p>

        <h2>8. Prototipos, Herencia</h2>

        <!-- Sección para 8.1 Herencia Prototípica -->
        <section id="tema-8-1" class="section-box">
            <h3>8.1 Herencia Prototípica</h3>
            <p>
                Este apartado fundamental en JavaScript explora cómo los objetos pueden hereda
                A diferencia de la herencia basada en clases de otros lenguajes, JavaScript ut
                Cuando se intenta acceder a una propiedad de un objeto, si esta no se encuentr
                JavaScript busca la propiedad en el prototipo de ese objeto, y luego en el pro
                hasta llegar a <code>null</code>.
            </p>
            <div id="output-8-1" class="console-output">
                <strong>Salida de Consola (8.1):</strong><br>
                Aquí se mostrarán los logs de `console.log` para la sección 8.1.
            </div>
            <div id="result-8-1" class="result-dom">
                <strong>Resultados en el DOM (8.1):</strong><br>
                <!-- Los resultados generados dinámicamente para el DOM en 8.1 aparecerán aqui
            </div>
        </section>
```

```
// Autor: [Tu Nombre Completo]
// Fecha de elaboración: [Fecha de hoy, ej. 25/10/2023]

// Función auxiliar para registrar en la consola y el DOM
// Esto nos ayuda a enviar mensajes a las cajas de salida correspondientes en el HTM
function logOutput(sectionId, message, isDOMResult = false) {
    const outputElement = document.getElementById(`output-${sectionId}`);
    const resultElement = document.getElementById(`result-${sectionId}`);

    if (outputElement) {
        if (!outputElement.innerHTML.includes('<strong>Salida de Consola')) {
            outputElement.innerHTML = `<strong>Salida de Consola (${sectionId}):</st
        }
        outputElement.innerHTML += `<p>${message.replace(/\n/g, '<br>')}</p>`; // Re
        outputElement.scrollTop = outputElement.scrollHeight; // Scroll automático a
    }
    if (isDOMResult && resultElement) {
        if (!resultElement.innerHTML.includes('<strong>Resultados en el DOM')) {
            resultElement.innerHTML = `<strong>Resultados en el DOM (${sectionId}):
        }
        resultElement.innerHTML += `<p>${message}</p>`;
        resultElement.scrollTop = resultElement.scrollHeight;
    }
    console.log(`[${sectionId}] ${message}`); // Siempre loguear en la consola del n
}

// ========================================================================
// TEMA: 8.1 Herencia Prototípica — 25/10/2023
// Este bloque demuestra cómo funciona la herencia prototípica en JavaScript.
// Es un concepto clave donde un objeto puede tener otro objeto como su prototipo,
// heredando sus propiedades y métodos. Si una propiedad no se encuentra en el objet
// JavaScript la busca en su prototipo, y así sucesivamente en la cadena prototípica
// Aprendí que esta es la forma fundamental de herencia en JavaScript antes de ES6 c
// ========================================================================
logOutput('8-1', "--- Iniciando 8.1 Herencia Prototípica ---");
```

```
Person.prototype.species = "Homo Sapiens"; // También podemos añadir propiedades

logOutput('8-2', "Función constructora 'Person' y su prototipo definidos.");
logOutput('8-2', `Person.prototype: ${JSON.stringify(Person.prototype)} (contiene sayHi y species)`);

// 3. Creamos instancias de 'Person'
let john = new Person("John");
let anna = new Person("Anna");

logOutput('8-2', "Instancias 'john' y 'anna' creadas:", true);
logOutput('8-2', `John: ${JSON.stringify(john)}`);
logOutput('8-2', `Anna: ${JSON.stringify(anna)}`);


// 4. Accedemos a métodos y propiedades heredadas
john.sayHi(); // "Hola, soy John desde el prototipo."
anna.sayHi(); // "Hola, soy Anna desde el prototipo."

logOutput('8-2', `Especie de John (heredada): ${john.species}`, true); // Homo Sapiens
logOutput('8-2', `Especie de Anna (heredada): ${anna.species}`, true); // Homo Sapiens

// 5. Demostrando la cadena prototípica con funciones constructoras
// La instancia (john) tiene como prototipo a Person.prototype
logOutput('8-2', `Prototipo de 'john' (john.__proto__ === Person.prototype): ${john.__proto__ === Per
// Person.prototype tiene como prototipo a Object.prototype
logOutput('8-2', `Prototipo de Person.prototype (Person.prototype.__proto__ === Object.prototype): $

// 6. Añadiendo una propiedad propia que oculta la del prototipo
john.species = "Super Sapiens"; // 'species' ahora es una propiedad propia de 'john'
logOutput('8-2', `Nueva especie de John (propia): ${john.species}`, true); // Super Sapiens
logOutput('8-2', `Especie de Anna (sigue heredando): ${anna.species}`, true); // Homo Sapiens
logOutput('8-2', `¿John tiene su propia propiedad 'species'? ${john.hasOwnProperty('species')}`); //
```

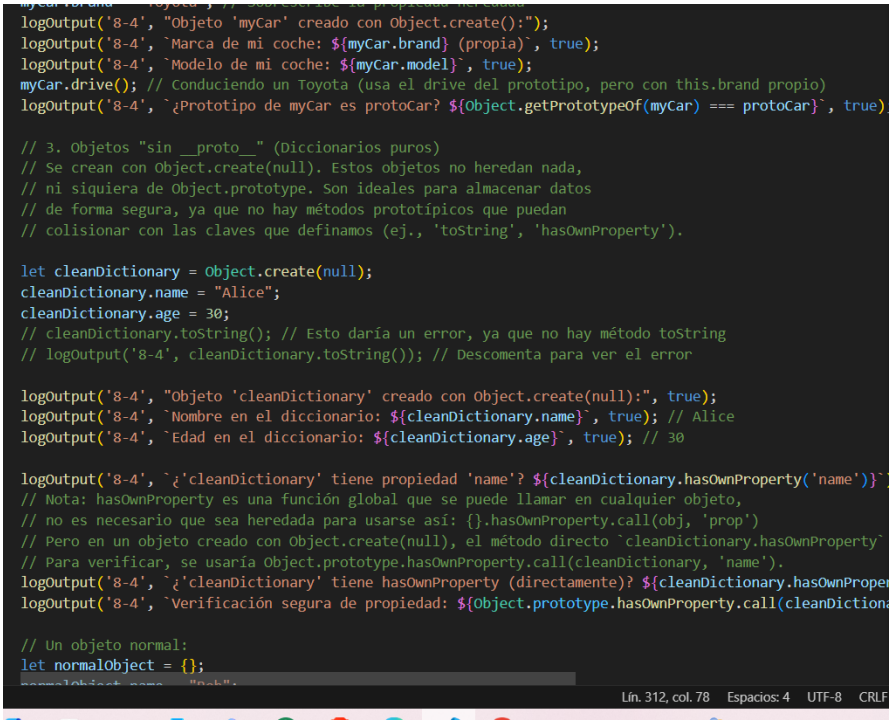


```
logOutput('8-3', `¿Function.prototype hereda de Object.prototype? ${Function.prototype.__proto__ === Obj

// NOTA IMPORTANTE: Modificar prototipos nativos globalmente es riesgoso y generalmente
// desaconsejado en producción. Puede causar conflictos con otras librerías o
// futuras versiones de JavaScript. Se hace aquí solo con fines educativos.

logOutput('8-3', "--- Fin 8.3 Prototipos Nativos ---");


// ========================================================================
// TEMA: 8.4 Métodos prototipo, objetos sin __proto__ — 25/10/2023
// Este bloque se enfoca en las herramientas modernas para trabajar con prototipos
// y en la creación de objetos especiales sin un prototipo por defecto.
// Cubre `Object.getPrototypeOf()`, `Object.setPrototypeOf()`, `Object.create()`,
// y la creación de objetos "vacíos" con `Object.create(null)`.
// Aprendí a manipular la cadena de prototipos de forma explícita y a crear
// objetos más seguros para el almacenamiento de datos clave-valor.
// ========================================================================
logOutput('8-4', "--- Iniciando 8.4 Métodos Prototipo, objetos sin __proto__ ---");

// 1. Object.getPrototypeOf(obj) y Object.setPrototypeOf(obj, proto)
// Permiten obtener y establecer el prototipo de un objeto.
let base = { value: 10 };
let derived = {};

logOutput('8-4', `Prototipo de 'base': ${Object.getPrototypeOf(base) === Object.prototype}`, true); // t
logOutput('8-4', `Prototipo de 'derived' antes de set: ${Object.getPrototypeOf(derived) === Object.proto

// Establecemos 'base' como prototipo de 'derived'
Object.setPrototypeOf(derived, base);
logOutput('8-4', "Establecido 'base' como prototipo de 'derived' usando Object.setPrototypeOf().");

logOutput('8-4', `Prototipo de 'derived' después de set: ${Object.getPrototypeOf(derived) === base}`, tr
logOutput('8-4', `Valor heredado en 'derived': ${derived.value}`, true); // 10 (heredado de base)

derived.newValue = 20;
```



```
// Establecemos 'base' como prototipo de 'derived'
Object.setPrototypeOf(derived, base);
logOutput('8-4', "Establecido 'base' como prototipo de 'derived' usando Ob

logOutput('8-4', `Prototipo de 'derived' después de set: ${Object.getProto
logOutput('8-4', `Valor heredado en 'derived': ${derived.value}`, true);

derived.newValue = 20;
logOutput('8-4', `Propiedad propia de 'derived': ${derived.newValue}`);

// 2. Object.create(proto, [descriptors])
// Crea un nuevo objeto con el prototipo especificado y propiedades opcion
// Es la forma recomendada y más flexible de establecer herencia.

let protoCar = {
    brand: "Generic",
    drive() {
        logOutput('8-4', `Conduciendo un ${this.brand} (desde prototipo).`
    }
};

let myCar = Object.create(protoCar, {
    model: {
        value: "Sedan",
        writable: true,
        enumerable: true,
        configurable: true
    },
    year: {
        value: 2023,
        writable: true,
        enumerable: true,
        configurable: true
```

# Evidencias de prototipo, herencia



```
logOutput('8-4', "Objeto 'myCar' creado con Object.create():");
logOutput('8-4', `Marca de mi coche: ${myCar.brand} (propia)`, true);
logOutput('8-4', `Modelo de mi coche: ${myCar.model}`, true);
myCar.drive(); // Conduciendo un Toyota (usa el drive del prototipo, pero con this.brand propio)
logOutput('8-4', `¿Prototipo de myCar es protoCar? ${Object.getPrototypeOf(myCar) === protoCar}`, true)

// 3. Objetos "sin __proto__" (Diccionarios puros)
// Se crean con Object.create(null). Estos objetos no heredan nada,
// ni siquiera de Object.prototype. Son ideales para almacenar datos
// de forma segura, ya que no hay métodos prototípicos que puedan
// colisionar con las claves que definamos (ej., 'toString', 'hasOwnProperty').

let cleanDictionary = Object.create(null);
cleanDictionary.name = "Alice";
cleanDictionary.age = 30;
// cleanDictionary.toString(); // Esto daría un error, ya que no hay método toString
// logOutput('8-4', cleanDictionary.toString()); // Descomenta para ver el error

logOutput('8-4', "Objeto 'cleanDictionary' creado con Object.create(null):", true);
logOutput('8-4', `Nombre en el diccionario: ${cleanDictionary.name}`, true); // Alice
logOutput('8-4', `Edad en el diccionario: ${cleanDictionary.age}`, true); // 30

logOutput('8-4', `¿'cleanDictionary' tiene propiedad 'name'? ${cleanDictionary.hasOwnProperty('name')}`
// Nota: hasOwnProperty es una función global que se puede llamar en cualquier objeto,
// no es necesario que sea heredada para usarse así: {}.hasOwnProperty.call(obj, 'prop')
// Pero en un objeto creado con Object.create(null), el método directo `cleanDictionary.hasOwnProperty`
// Para verificar, se usaría Object.prototype.hasOwnProperty.call(cleanDictionary, 'name').
logOutput('8-4', `¿'cleanDictionary' tiene hasOwnProperty (directamente)? ${cleanDictionary.hasOwnProper
logOutput('8-4', `Verificación segura de propiedad: ${Object.prototype.hasOwnProperty.call(cleanDictiona

// Un objeto normal:
let normalObject = {};
```

Lín. 312, col. 78  Espacios: 4  UTF-8  CRLF

## Prácticas de JavaScript.info - Prototipos y Herencia

**Instrucciones Importantes:**
Para ver la salida completa y detallada de los ejemplos, por favor, abre la consola de tu navegador. Puedes hacerlo presionando `F12` (en Windows/Linux) o `Cmd + Opt + I` (en macOS). Recarga la página después de abrir la consola para asegurarte de que todos los logs se capturen correctamente.

### 8. Prototipos, Herencia

#### 8.1 Herencia Prototípica

Este apartado fundamental en JavaScript explora cómo los objetos pueden heredar propiedades y métodos de otros objetos. A diferencia de la herencia basada en clases de otros lenguajes, JavaScript utiliza un mecanismo de "cadena de prototipos". Cuando se intenta acceder a una propiedad de un objeto, si esta no se encuentra directamente en él, JavaScript busca la propiedad en el prototipo de ese objeto, y luego en el prototipo del prototipo, y así sucesivamente, hasta llegar a `null`.

```
Salida de Consola (8.1):

Aquí se mostrarán los logs de `console.log` para la sección 8.1.
```