

Evidencias objetos lo básico

```
index.html > html
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Prácticas del Curso de JavaScript - Objetos</title>
7 <style>
8   body {
9     font-family: Arial, sans-serif;
10    background-color: #f9f9f9;
11    margin: 20px;
12    color: #333;
13  }
14  h1 {
15    color: #e83e8c; /* Rosa similar al de tu imagen */
16    text-align: center;
17    margin-bottom: 30px;
18  }
19  .section-container {
20    background-color: #ffe6f2; /* Fondo rosa claro para las secciones */
21    border-radius: 8px;
22    padding: 20px;
23    margin-bottom: 20px;
24    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
25  }
26  h2 {
27    color: #d63384; /* Rosa más oscuro para los títulos de sección */
28    margin-top: 0;
29    border-bottom: 1px solid #f0b3cc;
30    padding-bottom: 10px;
31    margin-bottom: 15px;
32  }
33  h3 {
34    color: #c2185b; /* Un tono más fuerte para los subtítulos */
35    margin-top: 20px;
36  }
```

```
index.html > html
2 <html lang="es">
3 <body>
4 <div class="section-container" id="tema-4-3">
5   <h2>4.3 Recolección de basura - 24/10/2025
6   <p>Este tema es principalmente conceptual. JavaScript tiene un recolector de basura
7   que automáticamente libera la memoria ocupada por objetos que ya no son accesibles
8   o referenciados por ninguna parte del programa. Esto evita fugas de memoria y
9   simplifica la gestión de la memoria para el desarrollador.
10  <p>No hay un código directo para "mostrar" la recolección de basura, ya que es un proceso interno.
11  <p>Aprendimos que JS gestiona la memoria automáticamente cuando un objeto se vuelve "inalcanzable".
12  <code>console.log("\n--- TEMA 4.3: Recolección de basura ---");
13  let obj1 = { data: "datos importantes" };
14  let obj2 = obj1; // obj2 referencia a obj1
15
16  obj1 = null; // obj1 ya no referencia al objeto, pero obj2 sí
17  <code>console.log("obj2 todavía tiene los datos:", obj2.data);
18
19  obj2 = null; // Ahora el objeto original ya no tiene referencias, será elegible para recolección
20  <code>console.log("El objeto original ahora es inalcanzable y será recolectado.");
21  <code>document.getElementById('tema-4-3').getElementsByTagName('h3')[0].insertAdjacentHTML('afterend',
22  </div>
23 <div class="section-container" id="tema-4-4">
24   <h2>4.4 Métodos de objeto, "this" - 24/10/2025
25   <p>Este bloque introduce los métodos (funciones dentro de objetos) y cómo usar la palabra clave 'this'.
26   <p>'this' se refiere al objeto actual donde se ejecuta el método.
27   <p>Aprendimos cómo definir funciones como propiedades de objetos y el contexto de 'this'.
28   <code>console.log("\n--- TEMA 4.4: Métodos de objeto, 'this' ---");
29   let persona = {
30     nombre: "santos",
31     saludar: function() {
32       // 'this' se refiere al objeto 'persona'
33       <code>console.log("Hola, mi nombre es " + this.nombre);
34     },
35   };
36   <code>persona.saludar();
37 </div>
38 <div class="section-container" id="tema-4-5">
39   <h2>4.5 Constructor, operador "new" - 24/10/2025
40   <p>Este bloque explica cómo crear funciones constructoras para generar múltiples objetos con la misma estructura.
41   <p>El operador 'new' se usa para invocar la función constructora, la cual:
42   1. crea un nuevo objeto vacío.
43   2. Asigna 'this' a ese nuevo objeto.
44   3. Ejecuta el cuerpo de la función.
45   4. Retorna 'this'.
46   <p>Aprendimos a estandarizar la creación de objetos similares.
47   <code>console.log("\n--- TEMA 4.5: Constructor, operador 'new' ---");
48   // Función constructora
49   function Usuario(nombre, edad) {
50     this.nombre = nombre;
51     this.edad = edad;
52     this.saludar = function() {
53       <code>console.log("Hola, soy " + this.nombre + " y tengo " + this.edad + " años.");
54     };
55   }
56   // Crear instancias de Usuario usando 'new'
57   let usuario1 = new Usuario("linda", 15);
58   let usuario2 = new Usuario("sofia", 35);
59   <code>console.log("Usuario 1:", usuario1);
60   <code>usuario1.saludar();
61   <code>console.log("Usuario 2:", usuario2);
62   <code>usuario2.saludar();
63   // Es importante usar 'new'. Sin 'new', 'this' apuntaría al objeto global (window)
64   // let usuarioSinNew = Usuario("Error", 10); // Esto intentaría modificar window.nombre y window.edad
65   // <code>console.log("Usuario sin new:", window.nombre); // Solo visible si no estás en modo estricto
66 </div>
67 <div class="section-container" id="tema-4-6">
68   <h2>4.6 Encadenamiento opcional "?." - 24/10/2025
69   <p>El operador de encadenamiento opcional '?.' permite acceder a propiedades anidadas de objetos sin tener que comprobar explícitamente si cada nivel de la cadena de propiedades existe. Si un intermedio es 'null' o 'undefined', la expresión se detiene y devuelve 'undefined' en lugar de lanzar un error.
70   <p>Aprendimos a escribir código más seguro y conciso al acceder a propiedades profundas.
71   <code>console.log("\n--- TEMA 4.6: Encadenamiento opcional '?.' ---");
72   let userProfile = {
73     name: "galeano",
74     address: {
75       street: "Calle Falsa 123",
76       city: "Springfield"
77     },
78     contact: null // Esta propiedad es nula
79   };
80   // Acceso normal a propiedades existentes
81   <code>console.log("Ciudad:", userProfile.address.city); // Springfield
82   // Intento de acceder a una propiedad de una propiedad nula (provoca error sin '?.')
83   // <code>console.log("Teléfono:", userProfile.contact.phone); // TypeError: Cannot read properties of null
84   // Con encadenamiento opcional, evita el error
85   <code>console.log("Teléfono (con ?.):", userProfile.contact?.phone); // undefined
86   <code>console.log("Código postal (con ?.):", userProfile.address?.zipCode); // undefined (la propiedad no existe)
87   // Se puede usar también con llamadas a métodos
88   let adminUser = {
89     name: "Boss",
90     greet() {
91       <code>console.log("Hola, soy el jefe.");
92     }
93   };
94   <code>adminUser.greet();
95 </div>
96 <div class="section-container" id="tema-4-7">
97   <h2>4.7 Tipo de símbolo </div>
```

```
73 console.log("Edad del user original:", user.age); // Sigue siendo 26
74 console.log("Edad del clon3:", clon3.age); // Es 50
75
76 document.getElementById('output-4-2').innerText = "Resultados en la consola. user.age: " + user.age;
77 // Fin TEMA 4.2
78
79 // TEMA: 4.3 Recolección de basura - 24/10/2025
80 // Este tema es principalmente conceptual. JavaScript tiene un recolector de basura
81 // que automáticamente libera la memoria ocupada por objetos que ya no son accesibles
82 // o referenciados por ninguna parte del programa. Esto evita fugas de memoria y
83 // simplifica la gestión de la memoria para el desarrollador.
84 // No hay un código directo para "mostrar" la recolección de basura, ya que es un proceso interno.
85 // Aprendimos que JS gestiona la memoria automáticamente cuando un objeto se vuelve "inalcanzable".
86 console.log("\n--- TEMA 4.3: Recolección de basura ---");
87 let obj1 = { data: "datos importantes" };
88 let obj2 = obj1; // obj2 referencia a obj1
89
90 obj1 = null; // obj1 ya no referencia al objeto, pero obj2 sí
91 console.log("obj2 todavía tiene los datos:", obj2.data);
92
93 obj2 = null; // Ahora el objeto original ya no tiene referencias, será elegible para recolección
94 console.log("El objeto original ahora es inalcanzable y será recolectado.");
95 document.getElementById('tema-4-3').getElementsByTagName('h3')[0].insertAdjacentHTML('afterend',
96 // Fin TEMA 4.3
97
98 // TEMA: 4.4 Métodos de objeto, "this" - 24/10/2025
99 // Este bloque introduce los métodos (funciones dentro de objetos) y cómo usar la palabra clave 'this'.
100 // 'this' se refiere al objeto actual donde se ejecuta el método.
101 // Aprendimos cómo definir funciones como propiedades de objetos y el contexto de 'this'.
102 console.log("\n--- TEMA 4.4: Métodos de objeto, 'this' ---");
103 let persona = {
104   nombre: "santos",
105   saludar: function() {
106     // 'this' se refiere al objeto 'persona'
107     console.log("Hola, mi nombre es " + this.nombre);
108   },
109 }
```

```
main.js > userProfile > name
133
134 document.getElementById('output-4-4').innerText = "Resultados en la consola. Ver los saludos y edades de los usuarios";
135 // Fin TEMA 4.4
136
137 // TEMA: 4.5 Constructor, operador "new" - 24/10/2025
138 // Este bloque explica cómo crear funciones constructoras para generar múltiples objetos con la misma estructura.
139 // El operador 'new' se usa para invocar la función constructora, la cual:
140 1. crea un nuevo objeto vacío.
141 2. Asigna 'this' a ese nuevo objeto.
142 3. Ejecuta el cuerpo de la función.
143 4. Retorna 'this'.
144 Aprendimos a estandarizar la creación de objetos similares.
145 console.log("\n--- TEMA 4.5: Constructor, operador 'new' ---");
146
147 // Función constructora
148 function Usuario(nombre, edad) {
149   this.nombre = nombre;
150   this.edad = edad;
151   this.saludar = function() {
152     console.log("Hola, soy " + this.nombre + " y tengo " + this.edad + " años.");
153   };
154 }
155
156 // Crear instancias de Usuario usando 'new'
157 let usuario1 = new Usuario("linda", 15);
158 let usuario2 = new Usuario("sofia", 35);
159
160 console.log("Usuario 1:", usuario1);
161 usuario1.saludar();
162
163 console.log("Usuario 2:", usuario2);
164 usuario2.saludar();
165
166 // Es importante usar 'new'. Sin 'new', 'this' apuntaría al objeto global (window)
167 // let usuarioSinNew = Usuario("Error", 10); // Esto intentaría modificar window.nombre y window.edad
168 // console.log("Usuario sin new:", window.nombre); // Solo visible si no estás en modo estricto
169
```

```
main.js > userProfile > name
172
173 // TEMA: 4.6 Encadenamiento opcional "?." - 24/10/2025
174 // El operador de encadenamiento opcional '?.' permite acceder a propiedades anidadas de objetos sin tener que comprobar explícitamente si cada nivel de la cadena de propiedades existe. Si un intermedio es 'null' o 'undefined', la expresión se detiene y devuelve 'undefined' en lugar de lanzar un error.
175 // Aprendimos a escribir código más seguro y conciso al acceder a propiedades profundas.
176 console.log("\n--- TEMA 4.6: Encadenamiento opcional '?.' ---");
177
178 let userProfile = {
179   name: "galeano",
180   address: {
181     street: "Calle Falsa 123",
182     city: "Springfield"
183   },
184   contact: null // Esta propiedad es nula
185 };
186
187 // Acceso normal a propiedades existentes
188 console.log("Ciudad:", userProfile.address.city); // Springfield
189
190 // Intento de acceder a una propiedad de una propiedad nula (provoca error sin '?.')
191 // console.log("Teléfono:", userProfile.contact.phone); // TypeError: Cannot read properties of null
192
193 // Con encadenamiento opcional, evita el error
194 console.log("Teléfono (con ?.):", userProfile.contact?.phone); // undefined
195 console.log("Código postal (con ?.):", userProfile.address?.zipCode); // undefined (la propiedad no existe)
196
197 // Se puede usar también con llamadas a métodos
198 let adminUser = {
199   name: "Boss",
200   greet() {
201     console.log("Hola, soy el jefe.");
202   }
203 };
204 adminUser.greet();
205
```

Prácticas del Curso de JavaScript - Objetos: Lo Básico

4.1 Objetos

Aquí se exploran los fundamentos de los objetos en JavaScript, cómo se crean y cómo acceder a sus propiedades.

Resultado en Consola / HTML:

Ver la consola (F12) para los resultados de la creación y acceso a propiedades de objetos.
Resultados en la consola. Nombre: Linda, Email: juan@example.com

4.2 Referencias de objetos y copia

Este apartado muestra cómo los objetos se copian por referencia y las diferencias con los tipos primitivos.

Resultado en Consola / HTML:

Ver la consola (F12) para entender cómo funcionan las referencias y copias de objetos.
Resultados en la consola. user.age: 26, clon2.name: Clonado

4.3 Recolección de basura