□ Tema 4: Programación Orientada a Objetos (POO) y Objetos Nativos en JavaScript

□ Resumen del contenido

1. Programación Orientada a Objetos (POO)

- Aunque **JavaScript no es un lenguaje puramente orientado a objetos**, permite el uso de clases desde **ES2015 (ES6)**.
- Los objetos se basan en **prototipos**, no en clases estrictas como en Java o PHP.
- Se puede crear un objeto directamente sin definir una clase previa.

1.1 Propiedades de un objeto

Se accede mediante `objeto.propiedad` o `objeto["propiedad"]`.

Se puede iterar con `for..in`:

```
for (const propiedad in casa) {
  console.log(propiedad + ": " + casa[propiedad]);
}
```

1.2 Definición de clases

```
class Casa {
  constructor(metros, direccion, ciudad) {
    this.metros = metros;
    this.direccion = direccion;
    this.ciudad = ciudad;
  }
  getDireccion() {
    return "Calle " + this.direccion + " " + this.ciudad;
  }
}
let casaPlaya = new Casa(65, "Miramar 5 4A", "Benidorm");
console.log(casaPlaya.getDireccion());
```

1.3 Variable this

Dentro de una función, `this` hace referencia al contexto actual. Puede cambiar según cómo se invoque la función (especialmente en eventos).

1.4 Herencia

```
class CasaIndividual extends Casa {
  constructor(metros, direccion, ciudad, metrosJardin, piscina) {
    super(metros, direccion, ciudad);
    this.metrosJardin = metrosJardin;
    this.piscina = piscina;
  }
  getDireccion() {
    return super.getDireccion() + " y con " + this.metrosJardin + " m² de jardín";
  }
}
```

1.5 Métodos estáticos

Se llaman directamente desde la clase y no acceden a `this`.

```
class Casa {
  static getTiposCalefaccion() {
    return ["gas natural", "eléctrica", "pellets"];
  }
}
console.log(Casa.getTiposCalefaccion());
```

1.6 toString()

Se ejecuta automáticamente al convertir un objeto a texto:

```
class Casa {
  toString() {
    return `Casa de ${this.metros} m² en ${this.ciudad}`;
  }
}
```

---

2. Funciones globales útiles

- `parseInt(valor)` → convierte a entero.

- `parseFloat(valor)` → convierte a decimal.
- `Number(valor)` → convierte a número (más estricto).
- `String(valor)` → convierte a cadena.
- `isNaN(valor)` → comprueba si no es número.
- `isFinite(valor)` → comprueba si es finito.

---

3. Objetos nativos de JavaScript

3.1 Number

- `toFixed(n)` → redondea a *n* decimales.
- `toLocaleString()` → aplica formato local (por ejemplo, `23.76` → `23,76`).

3.2 String

Métodos más usados:

- `.length`, `.charAt()`, `.substring()`, `.substr()`
- `.replaceAll()`, `.toLocaleLowerCase()`, `.toLocaleUpperCase()`
- `.trim()`, `.startsWith()`, `.endsWith()`, `.repeat()`
- `.localeCompare()` para comparar alfabéticamente.

3.3 Boolean

Convierte valores con `Boolean(valor)` o `!!valor`.

3.4 Math

Constantes y métodos:

- `Math.PI`, `Math.SQRT2`
- `Math.round()`, `Math.floor()`, `Math.ceil()`
- `Math.min()`, `Math.max()`, `Math.pow()`, `Math.abs()`, `Math.random()`, `Math.sqrt()`.

3.5 Date

Permite trabajar con fechas y horas:

```
let fecha = new Date(); // fecha actual
let fecha2 = new Date("2020-07-30");
```

Métodos principales:

- `getFullYear()`, `getMonth()`, `getDate()`, `getDay()`
- `getTime()` → milisegundos desde Epoch

- `setFullYear()`, `setDate()` ...
- `toString()`, `toLocaleString()`, `toDateString()`

Nota: al copiar un objeto `Date`, ambas variables apuntan al mismo valor. Para copiar correctamente:

```
let otraFecha = new Date(fecha.getTime());
```

---

4. Template literals

Permiten incluir variables dentro de cadenas usando backticks (`):

```
let edad = 25;
console.log(`El usuario tiene ${edad} años.`);
```

□ Ejercicios integradores

□ Ejercicio 1 (Básico)

Crea una clase `Persona` con las propiedades `nombre`, `edad` y un método `saludar()` que devuelva un mensaje:

```
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
  }
  saludar() {
    return `Hola, soy ${this.nombre} y tengo ${this.edad} años.`;
  }
}
let p1 = new Persona("Lucía", 22);
console.log(p1.saludar());
```

□ Ejercicio 2 (Intermedio)

Crea una clase `Rectangulo` con `base` y `altura` y métodos para calcular área y perímetro. Añade un método estático que devuelva una descripción del tipo de figura:

```
class Rectangulo {
  constructor(base, altura) {
    this.base = base;
    this.altura = altura;
  }
  area() { return this.base * this.altura; }
  perimetro() { return 2 * (this.base + this.altura); }
  static descripcion() { return "Figura geométrica con cuatro lados."; }
}
let r = new Rectangulo(4, 7);
console.log(Rectangulo.descripcion());
console.log("Área:", r.area(), "Perímetro:", r.perimetro());
```

□ Ejercicio 3 (Avanzado y completo)

Crea una clase `Alumno` que:

1. Tenga nombre y un array de notas.
2. Calcule la nota media usando `reduce()`.
3. Use métodos de `Math` para redondear.
4. Muestre la fecha actual del informe con `Date` y un texto formateado con *template literals*.

```
class Alumno {
  constructor(nombre, notas) {
    this.nombre = nombre;
    this.notas = notas;
  }
  media() {
    return this.notas.reduce((a, b) => a + b, 0) / this.notas.length;
  }
  informe() {
    let mediaRedondeada = Math.round(this.media() * 100) / 100;
    let fecha = new Date().toLocaleDateString();
    return `□ Informe de ${this.nombre}
Media: ${mediaRedondeada}
Fecha: ${fecha}`;
  }
}
let alumno = new Alumno("Carlos", [7.5, 8.2, 9.1, 6.8]);
console.log(alumno.informe());
```