Resumen UD5

1. Introducción a la POO en JavaScript

¿Qué es un objeto?

Un **objeto** es una entidad de un programa que tiene:

- **Propiedades** (atributos, características)
- **Comportamientos** (métodos, funciones)

Programación Orientada a Objetos (POO)

Modelo de programación que acerca el mundo real a un sistema informático, unificando en un mismo elemento (**objeto**) tanto los datos como las funciones que los manipulan.

Ejemplo conceptual:

```
ANIMAL (Prototipo)
   ↓
PERRO (Objeto)
   ├── ATRIBUTOS: Pancho, 4 años
   └── MÉTODOS: comer, caminar
```

Característica única de JavaScript

A diferencia de otros lenguajes, JavaScript está basado en **PROTOTIPOS**, no en clases.

Prototipo: Objeto que se utiliza como plantilla para obtener un conjunto inicial de propiedades de un nuevo objeto.

---

2. JSON (JavaScript Object Notation)

¿Qué es JSON?

Sintaxis para almacenar e intercambiar datos entre aplicaciones. Alternativa más sencilla que XML.

Características de JSON:

- ✅ Ampliamente aceptado en arquitectura cliente-servidor
- ✅ Fácil de usar, basado en texto, liviano y legible
- ✅ No es un lenguaje de programación, solo notación de datos
- ✅ Independiente de plataforma y lenguaje
- ✅ Extensión de archivo: `.json`
- ✅ Codificación por defecto: UTF-8

Estructuras JSON:

A. OBJETOS (entre llaves {})

```
{
   "nombre": "Cristian",
   "apellido": "Gonzalez",
   "edad": 25,
   "dni": 12345678,
   "direccion": "Av. Siempre Viva 123",
   "telefono": 12345678,
   "email": "cristiang@gmail.com"
}
```

B. ARRAYS (entre corchetes [])

```
[
   3,
   {
      "nombre": "Juan",
      "edad": 30,
      "pais": "España"
   },
   {
      "nombre": "Pedro",
      "edad": 35,
      "pais": "España"
   },
```

```
    true
]
```

Tipos de valores permitidos:

- **Strings** (entre comillas dobles)
- **Numbers** (enteros, flotantes, notación exponencial)
- **Booleans** (`true` o `false`)
- **null**
- **Arrays**
- **Objetos** (anidados)

Reglas importantes:

- ❌ Los strings deben ir entre comillas dobles
- ❌ No dejar vacía ni la clave ni el valor
- ❌ No incluye sintaxis para comentarios
- ❌ No poner coma después del último nombre/valor

Ventajas de JSON vs XML:

- ✅ Más corto y legible
- ✅ Más rápido de leer y escribir
- ✅ Puede utilizar arrays
- ✅ No necesita etiquetas de fin
- ✅ Se parsea con función JavaScript estándar

---

3. Métodos JSON en JavaScript

A. JSON.parse()

Convierte una cadena JSON en un objeto JavaScript.

```
const datos = `{
   "nombre": "Cristian",
```

```
    "apellido": "Gonzalez",
    "edad": 25
}`;

const persona = JSON.parse(datos);
console.log(persona.nombre);  // "Cristian"
console.log(persona.edad);    // 25
```

B. JSON.stringify()

Convierte un objeto JavaScript en una cadena JSON.

```
const usu = {
    usuario: "gustavo",
    clave: "123456"
};

console.log(typeof usu);  // object

const datos = JSON.stringify(usu);
console.log(typeof datos); // string
console.log(datos); // '{"usuario":"gustavo","clave":"123456"}'
```

---

4. Gestión de Objetos

Acceso a propiedades y métodos:

```
let vector = [4, 2, 7, 9];
console.log(vector.length);  // Propiedad
console.log(typeof vector);  // object

// Notaciones equivalentes:
objeto.propiedad
objeto["propiedad"]

objeto.método(parámetros)
objeto["método"](parámetros)
```

Operador instanceof

Comprueba el tipo/prototipo de un objeto.

```
let vector = [4, 2, 7, 9];
console.log(vector instanceof Array);  // true

let conjunto = new Set();
console.log(conjunto instanceof Map);  // false
```

Creación de objetos con new Object()

```
let notas = new Object();
notas.valores = [7, 5, 3, 2, 3, 9, 6];
notas.cantidad = notas.valores.length;
notas.media = notas.valores.reduce((a, b) => a + b, 0) / notas.cantidad;
notas.verMedia = function() {
   console.log(notas.media);
};

notas.verMedia();  // 5
```

Palabra clave THIS

Referencia al objeto actual dentro de un método.

```
let viaje = {
   origen: "Granada",
   destino: "El Cairo",
   dias: 8,
   precio: 750,
   mostrar: function() {
      console.log(`${this.origen}/${this.destino}`);
      console.log(`durante ${this.dias} dias: EUR${this.precio}`);
   }
};

viaje.mostrar();
```

5. Constructores y Clases

Constructores

Método especial para crear e inicializar objetos.

```
class Viaje {
    constructor(or, des, di, pre) {
        this.origen = or;
        this.destino = des;
        this.dias = di;
        this.precio = pre;
    }

    mostrar() {
        console.log(this.origen + '/' + this.destino);
        console.log('durante ' + this.dias + ' dias: ' + this.precio + ' EUR');
    }
}

let miViaje = new Viaje("Barcelona", "Ibiza", 2, 112);
miViaje.mostrar();
```

Ejemplo: Clase Teléfono Móvil

```
class TelefonoMovil {
    constructor(CPU, RAM, almacenamiento, ancho, alto, numCamaras) {
        this.CPU = CPU;
        this.RAM = RAM;
        this.almacenamiento = almacenamiento;
        this.ancho = ancho;
        this.alto = alto;
        this.numCamaras = numCamaras;
    }

    toString() {
        return `CPU: ${this.CPU}, RAM: ${this.RAM}, Almacenamiento: ${this.a
lmacenamiento}, Ancho: ${this.ancho}, Alto: ${this.alto}, Número de Cámar
```

```
as: ${this.numCamaras}`;
    }
}

let movil1 = new TelefonoMovil("Snapdragon 888", "8 GB", "256 GB", 6.5, 1
2.4, 4);
console.log(movil1.toString());
```

6. Herencia y Polimorfismo

Herencia

Mecanismo por el que una clase hereda características de otra.

```
class Miembro {
    constructor(nombre, alta, estado) {
        this.nombre = nombre;
        this.alta = alta;
        this.estado = estado;
    }

    cobrar() {
        document.write(`El Miembro ${this.nombre} ha cobrado <br>`);
    }
}

class Profesor extends Miembro {
    constructor(nombre, alta, estado, nAlumnos) {
        super(nombre, alta, estado);  // Llama al constructor padre
        this.nAlumnos = nAlumnos;
    }

    cobrar() {  // Sobrescribe el método padre
        document.write(`El Profesor ${this.nombre} ha cobrado <br>`);
    }
}

class Alumno extends Miembro {
```

```
    constructor(nombre, alta, estado, nAsignaturas) {
        super(nombre, alta, estado);
        this.nAsignaturas = nAsignaturas;
    }

    cobrar() {  // Sobrescribe el método padre
        document.write(`El Alumno ${this.nombre} ha cobrado <br>`);
    }
}
```

Polimorfismo

Diferentes objetos pueden tener métodos con el mismo nombre pero comportamiento diferente.

```
let unMiembro = new Miembro("Pepe Ruiz González", "12/02/2021", "finaliz
ado");
let unProfesor = new Profesor("Samuel Orta Pérez", "25/06/2021", "finaliza
do", 30);
let unAlumno = new Alumno("Elena Sánchez Sanz", "06/03/2021", "finaliza
do", 11);

unMiembro.cobrar();   // "El Miembro Pepe Ruiz González ha cobrado"
unProfesor.cobrar();  // "El Profesor Samuel Orta Pérez ha cobrado"
unAlumno.cobrar();    // "El Alumno Elena Sánchez Sanz ha cobrado"
```

---

7. Recorrido de Objetos

Bucle for...in

Recorre las propiedades de un objeto (incluyendo prototipos).

```
document.write('PROPIEDADES DE unProfesor:<br>');
for (let propiedad in unProfesor) {
    document.write(propiedad + ': ' + unProfesor[propiedad] + '<br>');
}
```

Bucle for...of con Object.values()

Recorre solo los valores de las propiedades propias del objeto.

```
document.write('PROPIEDADES DE unMiembro COMO ARRAY:<br>');
for (let valor of Object.values(unMiembro)) {
   document.write(valor + '<br>');
}
```

getOwnPropertyNames()

Obtiene solo las propiedades propias del objeto (no heredadas).

```
Object.getOwnPropertyNames(unObjeto);
```

---

8. Borrado de Propiedades

Operador delete

Elimina propiedades de un objeto.

```
let miViaje = new Viaje("Barcelona", "Ibiza", 2, 112);

// Eliminar propiedades
delete miViaje.precio;
delete miViaje.dias;

// Después del borrado, solo quedan origen y destino
for (elemento in miViaje) {
   console.log(elemento + ": " + miViaje[elemento]);
}
```

---

9. Prototipos en JavaScript

Concepto fundamental

JavaScript utiliza prototipos en lugar de clases. Todos los objetos proceden de un prototipo.

Modificación de prototipos

```
class Viaje {
    constructor(or, des, di, pre) {
        this.origen = or;
        this.destino = des;
        this.dias = di;
        this.precio = pre;
    }

    mostrar() {
        console.log(this.origen + '/' + this.destino + ': durante ' + this.dias + '
dias, ' + this.precio + ' EUR');
    }
}

// Añadir método al prototipo
Viaje.prototype.costeDiario = function() {
    return this.precio / this.dias;
};

// Añadir propiedad al prototipo
Viaje.prototype.descuento = '20%';

let miViaje = new Viaje("Barcelona", "Ibiza", 2, 112);
console.log(miViaje.costeDiario());  // 56
```

Cadena de prototipos

Cuando se accede a una propiedad/método:

1. Se busca en el objeto
2. Si no existe, se busca en su prototipo
3. Si no existe, se busca en el prototipo del prototipo
4. Continúa hasta encontrar la propiedad o llegar al final

---

10. Objetos Predefinidos

A. STRING - Manipulación de cadenas

Métodos más utilizados:

```
let cadena = " Bolonia ";

console.log(cadena.charAt(1));       // "B"
console.log(cadena.toUpperCase());   // " BOLONIA "
console.log(cadena.toLowerCase());   // " bolonia "
console.log(cadena.indexOf("n"));    // 5
console.log(cadena.lastIndexOf("o")); // 4
console.log(cadena.replace("B","C")); // " Colonia "
console.log(cadena.trim());          // "Bolonia"
console.log(cadena.slice(1,3));      // "Bo"
console.log(cadena.substr(1,3));     // "Bol"
console.log(cadena.split("o"));      // [" B", "l", "nia "]
```

Métodos estáticos:

```
String.fromCharCode(65, 66, 67);  // "ABC"
String.fromCodePoint(9731, 9733); // "☃★"
```

B. DATE - Manejo de fechas y horas

Creación de fechas:

```
let fechaSinParametros = new Date();
let fechaTodosParametros = new Date(2022, 8, 17, 13, 59, 49, 0);
let fechaTresParametros = new Date(2022, 8, 17);
let fechaUnParametro = new Date(1000);
```

Métodos más utilizados:

```
let fecha = new Date();

fecha.getDate();         // Día del mes (1-31)
fecha.getDay();          // Día de la semana (0-6)
fecha.getFullYear();     // Año (4 dígitos)
fecha.getHours();        // Hora (0-23)
fecha.getMonth();        // Mes (0-11)
```

```
fecha.getTime();          // Milisegundos desde 1/1/1970

fecha.toDateString();     // Formato legible
fecha.toLocaleString();   // Formato local
fecha.toJSON();           // Formato JSON
```

Métodos estáticos:

```
Date.now();               // Milisegundos actuales
Date.parse("2022-09-17"); // Convierte cadena a milisegundos
Date.UTC(2022, 8, 17);    // Fecha en UTC
```

C. MATH - Operaciones matemáticas

Constantes:

```
Math.E;       // 2.718 (Euler)
Math.PI;      // 3.14159
Math.SQRT2;   // 1.414 (Raíz de 2)
```

Métodos:

```
Math.abs(-5);          // 5 (Valor absoluto)
Math.ceil(4.3);        // 5 (Redondeo superior)
Math.floor(4.7);       // 4 (Redondeo inferior)
Math.round(4.5);       // 5 (Redondeo normal)
Math.max(10, 20, 5);   // 20
Math.min(10, 20, 5);   // 5
Math.pow(2, 3);        // 8 (2³)
Math.sqrt(16);         // 4
Math.random();         // Número aleatorio 0-1
Math.trunc(4.9);       // 4 (Parte entera)
```

D. BOOLEAN - Valores lógicos

⚠️ **Precaución importante:**

```
let logico1 = false;          // Primitivo booleano
let logico2 = new Boolean(false); // Objeto Boolean
```

```
console.log(logico1);         // false
console.log(logico2);         // Boolean {false}

if (logico1)                  // NO entra (false)
    console.log("logico1 entra");

if (logico2)                  // SÍ entra (objeto evaluado como true)
    console.log("logico2 entra");
```

Regla: Usar siempre booleanos primitivos, no objetos Boolean.

E. EXPRESIONES REGULARES (RegExp)

Creación de expresiones regulares:

```
// Forma literal
let erLiteral = /[0-9]/;

// Con constructor
let erObjeto = new RegExp('[0-9]');
```

Método test():

```
let er = new RegExp('[0-9]');
console.log(er.test("a"));       // false
console.log(er.test("almab6Mia")); // true
console.log(er.test("987"));     // true
```

Modificadores importantes:

I. Modificador i (case insensitive):

```
let er = /a/i;
console.log(er.test("pizza")); // true
console.log(er.test("TACO"));  // true
```

II. Modificador ^ (inicio de cadena):

```
let er = /^a/;
console.log(er.test("armario")); // true
console.log(er.test("pizza"));   // false
```

III. Modificador $ (fin de cadena):

```
let er = /pon$/;
console.log(er.test("tapon"));     // true
console.log(er.test("ponderado")); // false
```

IV. Modificador . (cualquier carácter):

```
let er = /ar.ón/;
console.log(er.test("arcón"));  // true
console.log(er.test("arpón"));  // true
```

V. Modificador [] (caracteres opcionales):

```
let er = /[aeiou]/;
console.log(er.test("SOS"));   // false
console.log(er.test("col"));   // true
```

VI. Modificador [^] (caracteres no permitidos):

```
let er = /[^0-9]/;
console.log(er.test("526"));     // false
console.log(er.test("cabo"));    // true
```

Cardinalidad (repeticiones):

```
/a?/      // 0 o 1 vez
/a*/      // 0 o más veces
/a+/      // 1 o más veces
/a{2}/    // exactamente 2 veces
/a{2,}/   // 2 o más veces
/a{2,4}/  // entre 2 y 4 veces
```

Método exec() (búsqueda avanzada):

```
let exreg = /sendero\\s(arenoso).+?noche/ig;
let res = exreg.exec('El sendero arenoso de noche puede ser peligroso');

// Resultado:
// [
//   "sendero arenoso de noche",
//   "arenoso",
//   index: 3,
//   input: 'El sendero arenoso de noche puede ser peligroso',
//   groups: undefined
// ]
```