

## Conceptos básicos de lógica

En tu proceso de aprendizaje, ten muy presente los siguientes conceptos para aplicarlos en los ejercicios de lógica de programación o desarrollo.

CONCEPTO	EJEMPLO
<b>Clase:</b> Es un modelo o plantilla que define propiedades (atributos) y comportamientos (métodos) de un objeto.	<pre>class <u>Circulo</u> {     constructor(<u>radio</u>) {         this.radio = <u>radio</u>; //         Propiedad pública         this.pi = 3.1416; // Valor         aproximado de <math>\pi</math>     }      // Método para calcular el área     calcularArea() {         return this.pi * (this.radio * this.radio);     }      // Método para calcular el     perímetro     calcularPerimetro() {         return 2 * this.pi * this.radio;     } }</pre>
<b>Instancia:</b> Es un objeto creado a partir de una clase. Representa un elemento concreto basado en el molde definido por la clase.	<pre>class <u>Perro</u> {     constructor(<u>nombre</u>, <u>raza</u>) {         this.nombre = <u>nombre</u>; //         Propiedad pública         this.raza = <u>raza</u>; // Propiedad         pública     }      // Método público</pre>

	<pre>     ladrar() {         return `\${this.nombre} está ladrando: ¡Guau guau!`;     } }  // Crear una instancia de la clase const miPerro = new Perro("Firulais", "Pastor Alemán");  // Usar las propiedades y métodos console.log(miPerro.nombre); // Accede a la propiedad: "Firulais" console.log(miPerro.raza); // Accede a la propiedad: "Pastor Alemán" console.log(miPerro.ladrar()); // Llama al método: "Firulais está ladrando: ¡Guau guau!" </pre>
<p><b>Propiedades y Métodos estáticos:</b></p> <p><b>Propiedades estáticas:</b> Son variables asociadas a la clase, no a sus objetos.</p> <p><b>Métodos estáticos:</b> Son funciones asociadas a la clase, no a sus objetos.</p>	<pre> class Persona {     // Propiedad estática     static contadorPersonas = 0;      // Propiedad no estática     constructor(nombre, edad) {         this.nombre = nombre; // Propiedad no estática         this.edad = edad;      // Propiedad no estática         Persona.contadorPersonas++; // Incrementa el contador de personas al crear una nueva instancia     }      // Método estático     static obtenerContador() {         return Persona.contadorPersonas; // Accede a la propiedad estática     } } </pre>

	<pre> // Método no estático mostrarInfo() {     console.log(`Nombre: \${this.nombre}, Edad: \${this.edad}`); }  // Crear instancias de la clase Persona const persona1 = new Persona("Juan", 30); const persona2 = new Persona("Ana", 25); const persona3 = new Persona("Carlos", 40);  // Mostrar información de las personas persona1.mostrarInfo(); // Muestra: Nombre: Juan, Edad: 30 persona2.mostrarInfo(); // Muestra: Nombre: Ana, Edad: 25 persona3.mostrarInfo(); // Muestra: Nombre: Carlos, Edad: 40  // Obtener el contador de personas (propiedad estática) console.log("Número total de personas:", Persona.obtenerContador()); // Muestra: 3 </pre>
<p><b>Modificadores de acceso:</b> Controlan el acceso a las propiedades y métodos</p> <p><b>Propiedad pública:</b> Se puede acceder desde cualquier parte del código.</p> <p><b>Propiedad privada:</b> Solo accesible desde dentro de la clase, usa # para definirla.</p> <p><b>Método público:</b> Puede ser llamado desde cualquier parte del código.</p> <p><b>Método privado:</b> Solo puede ser llamado dentro de</p>	<pre> class Persona {     // Propiedad pública     nombre;      // Propiedad privada     #edad; </pre>

la misma clase, usa # para definirlo.

```
constructor(nombre, edad) {
    this.nombre = nombre; //
Propiedad pública
    this.#edad = edad;      //
Propiedad privada
}

// Método público
mostrarNombre() {
    console.log(`Nombre:
${this.nombre}`);
}

// Método privado
#mostrarEdad() {
    console.log(`Edad:
${this.#edad}`);
}

// Método público para acceder al
método privado
mostrarInfo() {
    this.mostrarNombre();
    this.#mostrarEdad(); //
Llamada al método privado
}
}

const personal = new Persona("Juan",
30);
personal.mostrarNombre(); //
Muestra: Nombre: Juan
personal.mostrarInfo();    //
Muestra: Nombre: Juan, Edad: 30

// Intentando acceder a la propiedad o
método privado fuera de la clase
console.log(personal.#edad); //
Error: Private field '#edad' must be
declared in an enclosing class
personal.#mostrarEdad();    // Error:
Private method '#mostrarEdad' is not
```

**Encapsulación:** Los detalles internos del objeto se ocultan (ocultación de información) y se exponen solo las partes necesarias a través de interfaces públicas (por ejemplo, métodos getter y setter).

accessible outside class

```
class Persona {  
    // Propiedad privada  
    #nombre;  
  
    constructor(nombre) {  
        this.#nombre = nombre; //  
Inicializamos la propiedad privada  
    }  
  
    // Método público para acceder a la  
propiedad privada  
    getNombre() {  
        return this.#nombre;  
    }  
  
    // Método público para modificar la  
propiedad privada  
    setNombre(nombre) {  
        if (nombre.length > 0) {  
            this.#nombre = nombre; //  
Modificar la propiedad privada  
        } else {  
            console.log("El nombre no  
puede estar vacío.");  
        }  
    }  
}  
  
const personal = new Persona("Juan");  
// Accediendo a la propiedad privada a  
través de métodos públicos  
console.log(personal.getNombre()); //  
Muestra: Juan  
// Modificando la propiedad privada a  
través de un método público  
personal.setNombre("Carlos");  
console.log(personal.getNombre()); //  
Muestra: Carlos  
// Intentando acceder directamente a  
la propiedad privada (causa error)
```

	<pre>console.log(personal.#nombre); // Error: Private field '#nombre' must be declared in an enclosing class</pre>
<p><b>Herencia:</b> Permite crear nuevas clases basadas en clases existentes, heredando sus atributos y métodos.</p>	<pre>// Clase Padre class Animal {     constructor(nombre) {         this.nombre = nombre;     }      saludar() {         console.log(`Hola, soy un \${this.nombre}`);     } }  // Clase Hija que hereda de Animal class Perro extends Animal {     constructor(nombre, raza) {         super(nombre); // Llama al constructor de la clase Padre         this.raza = raza;     }      ladrar() {         console.log("¡Guau!");     } }  const miPerro = new Perro("Perro", "Bulldog"); miPerro.saludar(); // Muestra: Hola, soy un Perro miPerro.ladrar(); // Muestra: ¡Guau!</pre>
<p><b>Polimorfismo:</b> Permite que diferentes objetos respondan de distintas formas al mismo mensaje o método. Esto se logra mediante la sobrecarga de métodos (métodos con el mismo nombre, pero diferentes parámetros) o la sobrescritura (modificar el comportamiento de un método heredado)</p>	<pre>// Clase Padre class Animal {     saludar() {         console.log("Soy un animal");     } }  // Clase Hija que sobrescribe el</pre>

```
método saludar()
class Perro extends Animal {
    saludar() {
        console.log("¡Guau! Soy un
perro");
    }
}

// Otra clase Hija que también
sobrescribe el método saludar()
class Gato extends Animal {
    saludar() {
        console.log("¡Miau! Soy un
gato");
    }
}

// Instanciando objetos
const animal = new Animal();
const perro = new Perro();
const gato = new Gato();

// Llamando al método saludar() en
cada objeto
animal.saludar(); // Muestra: Soy un
animal
perro.saludar(); // Muestra: ¡Guau!
Soy un perro
gato.saludar(); // Muestra: ¡Miau!
Soy un gato
```