**JAVASCRIPT**

En JavaScript, un objeto es una estructura de datos que agrupa valores relacionados y/o funcionalidades, y se define mediante una colección de pares clave-valor (llamados "propiedades"). Las propiedades pueden ser de distintos tipos, como valores primitivos (números, strings, booleanos, etc.), funciones o incluso otros objetos.

Por ejemplo, podemos definir un objeto "persona" con propiedades como "nombre", "edad" y "profesión", de la siguiente manera:

let persona = {

nombre: "Juan",

edad: 30,

profesion: "ingeniero"

};

Podemos acceder a las propiedades de un objeto mediante la notación de punto o la notación de corchetes, por ejemplo:

console.log(persona.nombre); // "Juan"

console.log(persona["edad"]); // 30

También podemos añadir o modificar propiedades en un objeto en cualquier momento:

persona.edad = 31;

persona.ciudad = "Madrid";

En resumen, los objetos en JavaScript son una forma de representar y organizar datos y funcionalidades de manera flexible y eficiente.

* **Clases**: son plantillas o modelos que definen las características y comportamientos de los objetos que se van a crear. En JavaScript, las clases se definen mediante la palabra reservada **class**. Una vez que se ha definido una clase, se pueden crear objetos basados en esa clase.
* **Objetos**: son instancias de una clase. Un objeto se crea a partir de una clase y hereda todas las características y comportamientos definidos en ella. En JavaScript, los objetos se crean mediante la palabra reservada **new**.
* **Atributos**: son variables que definen las características o propiedades de un objeto. En JavaScript, los atributos se definen dentro de una clase utilizando la palabra reservada **this**.
* **Métodos**: son funciones que definen el comportamiento de un objeto. En JavaScript, los métodos se definen dentro de una clase utilizando la palabra reservada **function**.
* **Herencia**: es un concepto que permite crear una clase a partir de otra ya existente, heredando todas sus características y comportamientos. En JavaScript, se puede utilizar la palabra reservada **extends** para crear una clase hija a partir de una clase padre.
* **Encapsulamiento**: es un principio de programación que consiste en ocultar la complejidad y la implementación interna de un objeto, y exponer solo una interfaz pública para interactuar con él. En JavaScript, se puede simular el encapsulamiento utilizando el concepto de propiedades privadas y públicas.
* **Polimorfismo**: es un concepto que permite que un objeto se comporte de diferentes maneras según el contexto en el que se utiliza. En JavaScript, se puede implementar el polimorfismo utilizando diferentes métodos con el mismo nombre en diferentes clases.

En resumen, en JavaScript las clases definen las características y comportamientos de los objetos, los objetos son instancias de una clase que heredan sus características y comportamientos, los atributos son variables que definen las características de un objeto, los métodos son funciones que definen su comportamiento, la herencia permite crear clases hijas a partir de clases padres, el encapsulamiento permite ocultar la complejidad interna de un objeto y el polimorfismo permite que un objeto se comporte de diferentes maneras según el contexto.

**Ejemplo de clases**

En este ejemplo, se define una clase **Persona** con dos atributos (**nombre** y **edad**) y un método (**saludar**) que muestra un mensaje por consola.

class Persona {

constructor(nombre, edad) {

this.nombre = nombre;

this.edad = edad;

}

saludar() {

console.log(`Hola, mi nombre es ${this.nombre} y tengo ${this.edad} años`);

}

}

**Ejemplo de objetos**

En este ejemplo, se crean dos objetos (**persona1** y **persona2**) a partir de la clase **Persona** que definimos previamente.

const persona1 = new Persona('Juan', 25);

const persona2 = new Persona('María', 30);

**Ejemplo de atributos**

En el ejemplo anterior, los atributos **nombre** y **edad** de la clase **Persona** son **nombre** y **edad**. Estos atributos se definen en el constructor de la clase.

**Ejemplo de métodos**

En el ejemplo anterior, el método **saludar** de la clase **Persona** muestra un mensaje por consola utilizando los atributos **nombre** y **edad**. Este método se define dentro de la clase utilizando la palabra reservada **function**.

**Ejemplo de herencia**

En este ejemplo, se define una clase **Estudiante** que hereda de la clase **Persona** y agrega un nuevo atributo (**curso**) y un nuevo método (**presentarse**).

class Estudiante extends Persona { constructor(nombre, edad, curso) { super(nombre, edad); this.curso = curso; } presentarse() { console.log(`Hola, soy ${this.nombre} y curso ${this.curso}`); } }

**Ejemplo de encapsulamiento**

En JavaScript no existe un concepto formal de propiedades privadas, pero se puede simular utilizando convenciones. En este ejemplo, se define un atributo **#saldo** que se considera privado y solo puede ser accedido y modificado a través de métodos públicos.

javascriptCopy code

class CuentaBancaria { constructor(saldoInicial) { this.#saldo = saldoInicial; } depositar(monto) { this.#saldo += monto; } retirar(monto) { if (this.#saldo >= monto) { this.#saldo -= monto; } else { console.log('Saldo insuficiente'); } } consultarSaldo() { console.log(`Saldo actual: $${this.#saldo}`); } #saldo; }

**Ejemplo de polimorfismo**

En este ejemplo, se definen dos clases (**Perro** y **Gato**) que comparten un método (**hacerSonido**), pero cada clase implementa este método de manera diferente.

javascriptCopy code

class Perro { hacerSonido() { console.log('¡Guau!'); } } class Gato { hacerSonido() { console.log('¡Miau!'); } }

Estos son solo ejemplos sencillos para ilustrar los diferentes conceptos en JavaScript. En la práctica, estos conceptos pueden ser mucho más complejos y se utilizan en combinación para crear aplicaciones robustas y escalables.