**¿QUE ES UN FRAMEWORK?**

Un framework es un conjunto de herramientas, bibliotecas y reglas que facilitan el desarrollo de software. Proporciona una estructura base sobre la cual los desarrolladores pueden construir sus aplicaciones sin tener que programar desde cero.

**Características de un framework:**

* Estructura predefinida: Establece una organización para el código, facilitando el mantenimiento y escalabilidad.
* Código reutilizable: Ofrece funciones y módulos listos para usar.
* Convención sobre configuración: Reduce la cantidad de decisiones que el desarrollador debe tomar, siguiendo patrones establecidos.
* Automatización: Agiliza tareas repetitivas, como la gestión de bases de datos, enrutamiento o seguridad.

**Ejemplos de frameworks:**

* **Para desarrollo web:** Laravel (PHP), Spring (Java), Django (Python), Angular (JavaScript).
* **Para aplicaciones de escritorio:** JavaFX (Java), Electron (JavaScript).
* **Para desarrollo móvil:** Flutter (Dart), React Native (JavaScript).

**¿QUE ES UNA LIBRERIA?**

Una librería es un conjunto de funciones, clases y módulos reutilizables que los desarrolladores pueden usar en sus proyectos para evitar escribir código desde cero. A diferencia de un framework, una librería no impone una estructura específica ni dicta cómo se debe organizar el código.

**Diferencias clave entre librería y framework:**

| **Característica** | **Librería** | **Framework** |
| --- | --- | --- |
| **Control del flujo** | El desarrollador decide cuándo y cómo usarla. | El framework controla el flujo del programa. |
| **Estructura** | No impone una forma de organizar el código. | Define una arquitectura específica. |
| **Flexibilidad** | Se puede usar en cualquier parte del código. | Es más restrictivo, sigue convenciones establecidas. |

Ejemplos de librerías:

* **Java:** JDBC (para bases de datos), JUnit (para pruebas).
* **Python:** NumPy (cálculo numérico), Pandas (manejo de datos).
* **JavaScript:** Lodash (utilidades), Axios (peticiones HTTP).

FRAMEWORK VS LIBRERIA

La diferencia clave entre un **framework** y una **librería** radica en el **control del flujo del programa** y el **grado de estructura** que imponen al desarrollo.

**DIFERENCIAS**

| **Característica** | **Framework** | **Librería** |
| --- | --- | --- |
| **Control del flujo** | El framework controla el flujo de la aplicación (Inversión de Control). | El desarrollador decide cuándo y cómo usar la librería. |
| **Estructura** | Impone una arquitectura específica que se debe seguir. | No impone una estructura, se usa libremente. |
| **Flexibilidad** | Menos flexible, ya que dicta cómo construir la aplicación. | Más flexible, se puede usar en cualquier parte del código. |
| **Ejemplo de uso** | Un framework web como Django o Spring dicta cómo organizar el código, manejar rutas y bases de datos. | Una librería como NumPy o jQuery proporciona funciones específicas sin imponer una estructura. |

E**jemplos de frameworks vs librerías**

| **Tecnología** | **Frameworks** | **Librerías** |
| --- | --- | --- |
| **Java** | Spring Boot, JavaFX | JDBC, JUnit |
| **Python** | Django, Flask | NumPy, Pandas |
| **JavaScript** | Angular, Next.js | jQuery, Lodash |
| **PHP** | Laravel, Symfony | PHPMailer, Guzzle |

**EJEMPLO EN CÓDIGO (JAVASCRIPT)**

**Uso de una librería (Axios para peticiones HTTP)**

import axios from 'axios';

axios.get('https://api.example.com/data')

    .then(response => console.log(response.data))

    .catch(error => console.error(error));

Uso de un framework (Express.js para un servidor web)

const express = require('express');

const app = express();

app.get('/', (req, res) => {

    res.send('¡Hola, mundo!');

});

app.listen(3000, () => console.log('Servidor corriendo en puerto 3000'));

Express **define** cómo estructurar el servidor y gestionar las rutas.

**¿Qué es ECMAScript (ES)?**

ECMAScript **(ES)** es un estándar de scripting que define cómo debe funcionar JavaScript. Fue desarrollado por **ECMA International** en 1997 bajo la especificación ECMA-262 para asegurar que todos los navegadores ejecuten JavaScript de manera uniforme.

**JavaScript** es un lenguaje basado en **ECMAScript**, al igual que otros como **ActionScript y JScript.**

**¿Qué es ES6 (ECMAScript 2015)?**

**ECMAScript 6 (ES6)**, también conocido como **ECMAScript 2015**, fue una de las actualizaciones más importantes de JavaScript, introduciendo nuevas características para mejorar la legibilidad y eficiencia del código.

🔹 **Lanzado en:** 2015  
🔹 **Propósito:** Modernizar JavaScript y hacerlo más fácil de escribir y mantener.

**Novedades principales de ES6**

| **Característica** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| **let y const** | Nuevas formas de declarar variables con mejor alcance. | let x = 10; const y = 20; |
| **Arrow functions (=>)** | Funciones más cortas y sin this propio. | const suma = (a, b) => a + b; |
| **Template literals** | Permiten interpolación de variables en strings. | `Hola, ${nombre}!` |
| **Destructuración** | Extrae valores de arrays u objetos fácilmente. | const { nombre, edad } = persona; |
| **Clases (class)** | Soporte oficial para programación orientada a objetos. | class Persona { constructor(nombre) { this.nombre = nombre; } } |
| **Módulos (import / export)** | Permite dividir el código en archivos reutilizables. | import { funcion } from './modulo.js'; |
| **Promesas (Promise)** | Manejo moderno de funciones asíncronas. | fetch('api.com').then(res => res.json()); |

**Versiones posteriores a ES6**

Después de ES6, ECMAScript se ha actualizado cada año. Algunas mejoras clave incluyen:

| **Versión** | **Año** | **Novedades** |
| --- | --- | --- |
| **ES7 (2016)** | 2016 | Array.prototype.includes(), Exponentiation Operator (\*\*) |
| **ES8 (2017)** | 2017 | async/await, Object.values(), Object.entries() |
| **ES9 (2018)** | 2018 | Rest/Spread Properties, Promise.prototype.finally() |
| **ES10 (2019)** | 2019 | flat(), flatMap(), Object.fromEntries() |
| **ES11 (2020)** | 2020 | BigInt, Nullish Coalescing (??), Optional Chaining (?.) |

**CONCLUSIÓN**

* **ECMAScript** es el estándar en el que se basa JavaScript.
* **ES6 (ECMAScript 2015)** fue una actualización importante con mejoras como let, const, arrow functions, clases y más.
* Nuevas versiones de **ECMAScript** continúan mejorando JavaScript cada año.

**¿Qué es una Clase en JavaScript?**

Una clase es una plantilla para crear objetos. Introducidas en ES6 (ECMAScript 2015), las clases en JavaScript facilitan la programación orientada a objetos (POO), permitiendo la creación de objetos con propiedades y métodos.

Antes de ES6, JavaScript usaba funciones constructoras y prototipos para definir objetos, pero con las clases, el código es más limpio y fácil de entender.

**🔹 Sintaxis de una Clase**

class Persona {

    constructor(nombre, edad) {

      this.nombre = nombre;

      this.edad = edad;

    }

    // Método de la clase

    saludar() {

      return `Hola, soy ${this.nombre} y tengo ${this.edad} años.`;

    }

  }

  // Crear un objeto de la clase Persona

  const persona1 = new Persona("Eder", 22);

  console.log(persona1.saludar());  // Hola, soy Eder y tengo 22 años.

**Características Principales de las Clases en JavaScript**

1. **Constructor (constructor)**
   * Es un método especial que se ejecuta al crear un nuevo objeto.
   * Define las propiedades iniciales de la instancia.
2. **Métodos**
   * Se definen dentro de la clase sin la palabra clave function.
   * Se pueden llamar desde los objetos creados con la clase.
3. **Instanciación (new)**
   * Para crear un nuevo objeto de una clase, se usa new NombreClase().

**Herencia (Extends y Super)**

Las clases pueden heredar propiedades y métodos de otra clase usando **extends.**

class Estudiante extends Persona {

    constructor(nombre, edad, carrera) {

      super(nombre, edad); // Llama al constructor de la clase padre

      this.carrera = carrera;

    }

    estudiar() {

      return `${this.nombre} está estudiando ${this.carrera}.`;

    }

  }

  // Crear un objeto de Estudiante

  const estudiante1 = new Estudiante("Eder", 22, "Ingeniería");

  console.log(estudiante1.saludar());  // Hola, soy Eder y tengo 22 años.

  console.log(estudiante1.estudiar()); // Eder está estudiando Ingeniería.

**Getters y Setters**

Permiten controlar cómo se acceden y modifican las propiedades.

class CuentaBancaria {

    constructor(saldo) {

      this.\_saldo = saldo; // Usamos \_ para indicar que es "privado"

    }

    get saldo() {

      return `Saldo actual: $${this.\_saldo}`;

    }

    set saldo(nuevoSaldo) {

      if (nuevoSaldo >= 0) {

        this.\_saldo = nuevoSaldo;

      } else {

        console.log("El saldo no puede ser negativo.");

      }

    }

  }

  const cuenta = new CuentaBancaria(1000);

  console.log(cuenta.saldo); // Saldo actual: $1000

  cuenta.saldo = 2000;

  console.log(cuenta.saldo); // Saldo actual: $2000

**Conclusión**

* Las clases en JavaScript permiten organizar mejor el código y facilitan la reutilización.
* Tienen constructores y métodos para definir el comportamiento de los objetos.
* Soportan herencia, lo que permite reutilizar código en subclases.
* Getters y Setters ayudan a controlar el acceso a las propiedades.