FLUTTER / DART

PARADIGMA REACTIVO

Equipo 4
Padilla Valencia Emanuel
Martinez Valdez Clara Andrea
Armas Diaz Erick Hidekio

FUNDAMENTOS

Dart

Dart es un lenguaje de programación de propósito general, de código abierto y orientado a objetos, desarrollado por Google.
Se basa en conceptos como clases, objetos y herencia, y está optimizado para crear aplicaciones rápidas y eficientes.
Su principal fortaleza es que permite desarrollar aplicaciones para móviles, web y escritorio usando una sola base de código, especialmente cuando se combina con Flutter.

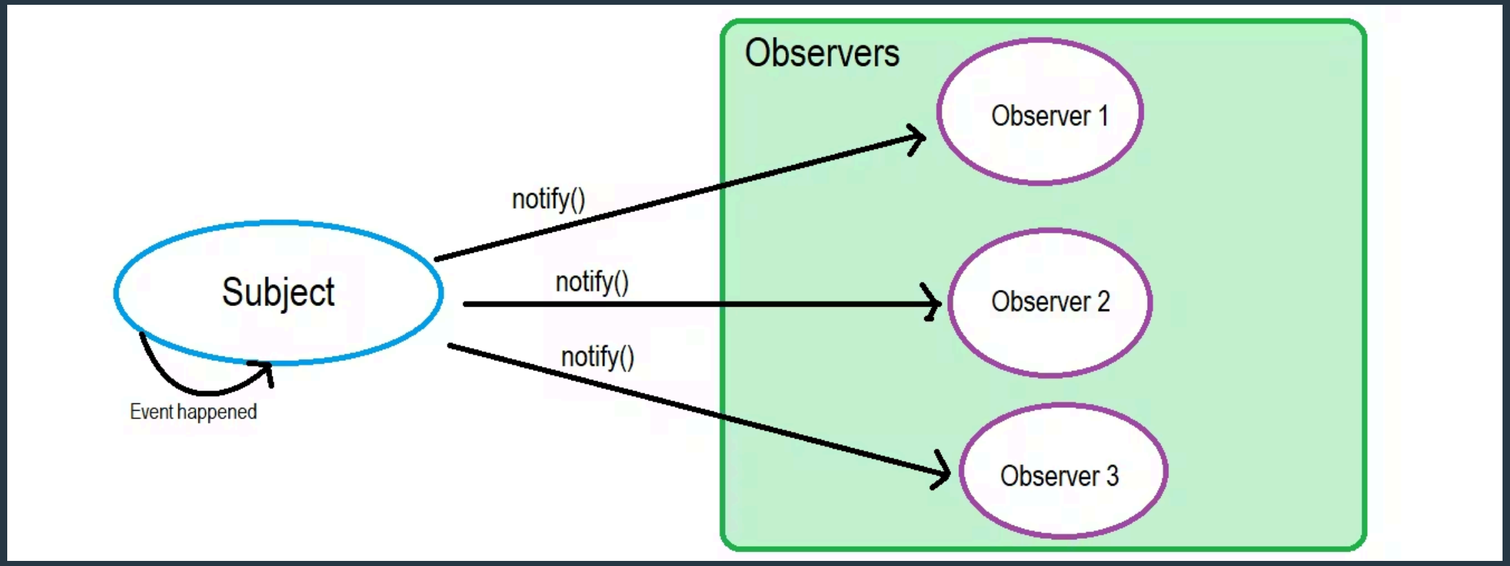
Flutter

Flutter es un framework creado por Google que se usa para desarrollar aplicaciones multiplataforma.
Permite crear apps para móviles, web y escritorio usando un solo lenguaje, Dart.
Además, usa su propio motor gráfico y herramientas que hacen que las aplicaciones sean rápidas, visuales y eficientes.

PARADIGMA REACTIVO

La programación reactiva es un paradigma decalrativo centrado en el manejo de flujos de datos (streams) y la propagación automatica del cambio.

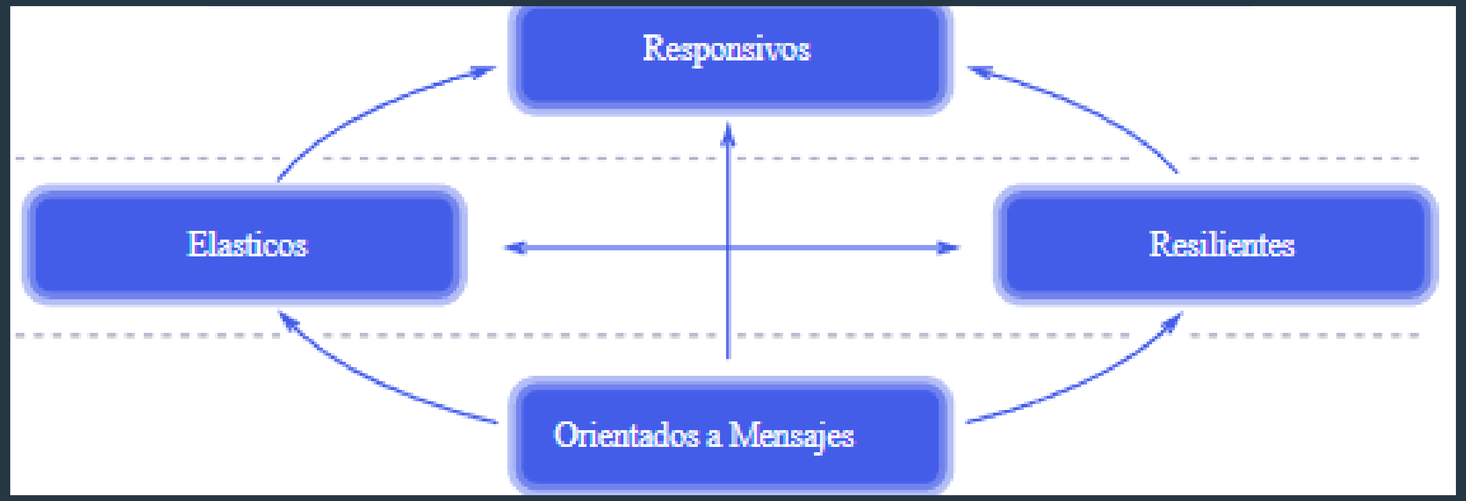
En lugar de preguntar constantemente por actualizaciones, el sistema reacciona automáticamente a eventos y cambios de datos.



MANIFIESTO REACTIVO

Es un documento que define las bases para construir sistemas modernos que sean capaces de lidiar con las exigencias actuales:

En lugar de centrarse en cómo programar, se centra en cómo debe comportarse el sistema.



1. Responsivo: Un sistema reactivo responde a tiempo. El objetivo es asegurar que la aplicación responda de forma rápida y constante
2. Resiliente: El sistema no se desmorona si algo falla. Si un componente se rompe, el resto de la aplicación debe seguir funcionando.
3. Elástico: El sistema se adapta a la carga de trabajo. Además de ser escalable, debe ser eficiente con los recursos.
4. Orientado a Mensajes: Los componentes se comunican mediante el envío de mensajes asíncronos.

PARADIGMA REACTIVO EN FLUTTER

- Streams de eventos (Flujos): Secuencias de datos que llegan a lo largo del tiempo.
- Propagación automática: Cuando el dato cambia, todos los elementos que lo "escuchan" se actualizan solos.
- Observabilidad: Los componentes están en constante espera de nuevos eventos o estados.

CARACTERÍSTICAS GENERALES

DART - Lenguaje:

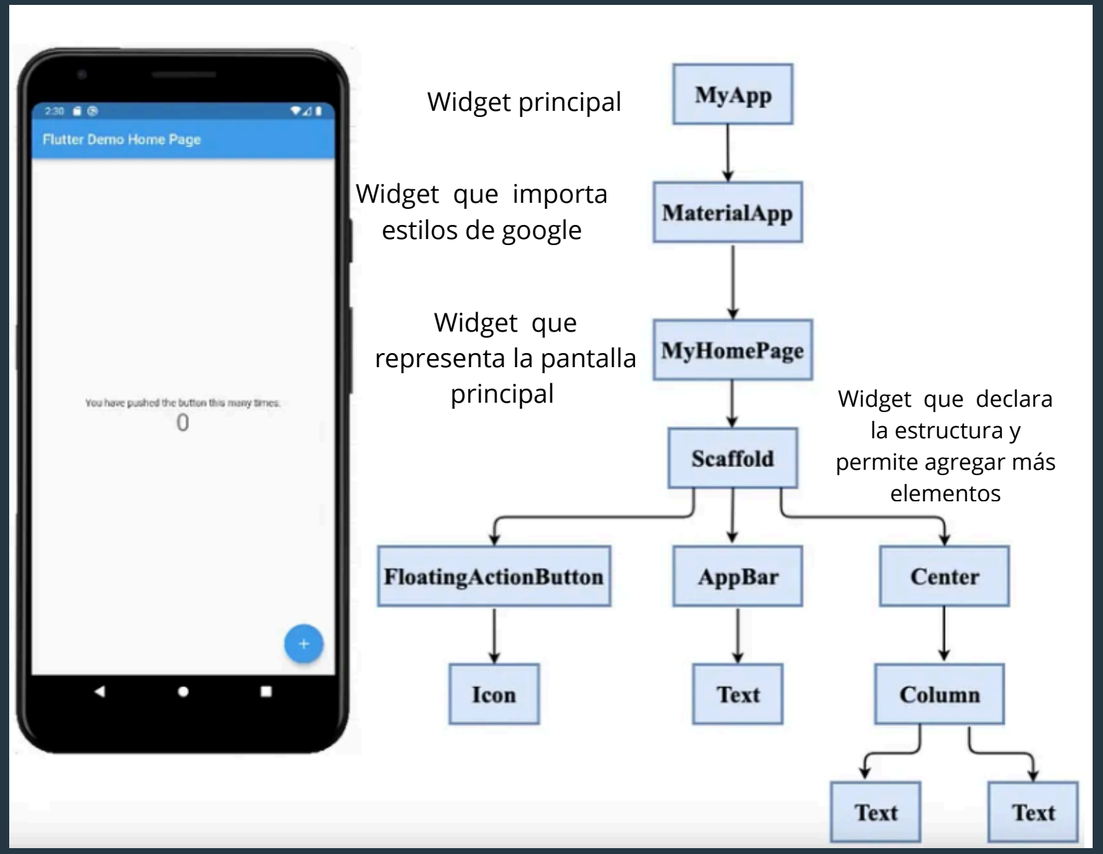
- Orientado a objetos (por ende del tipo *imperativo*)
- Cuenta con compilación JIT (Just-In-Time):un proceso de compilación en el cual el código se traduce desde una representación intermedia o un lenguaje de alto nivel a código de máquina durante la ejecución, en lugar de hacerlo antes de que el programa inicie, como en la complación AOT (Ahead-Of-Time).
- El Hot Reload es su mayor fuerte; permite ver cambios en el código en menos de un segundo sin perder el estado de la app.

FLUTTER - Framework:

- UI Declarativa
- Todo es un widget
- Motor de renderizado propio (Skia/Impeller)

Conceptos Combinados:

- Paradigma Mixto: Dart imperativo + Flutter declarativo



DISCPLINA DE TIPOS

Tipado Estático

Dart verifica los tipos durante la compilación. Si se declara un int de 64 bits, el compilador no permitirá asignarle un String

Tipado Fuerte

El lenguaje es estricto con las reglas de tipos. Por ejemplo:
Un int siempre ocupará sus 8 bytes y no puede ser usado donde se espera un bool (Booleano).
No puedes realizar operaciones matemáticas entre tipos incompatibles (como sumar un número a un texto)

Inferencia de Tipos

Dart puede inferir qué tipo ded variable es, por ejemplo:
var edad = 25;
Dart infiere automáticamente que es un int de 64 bits.

LANZAMIENTO

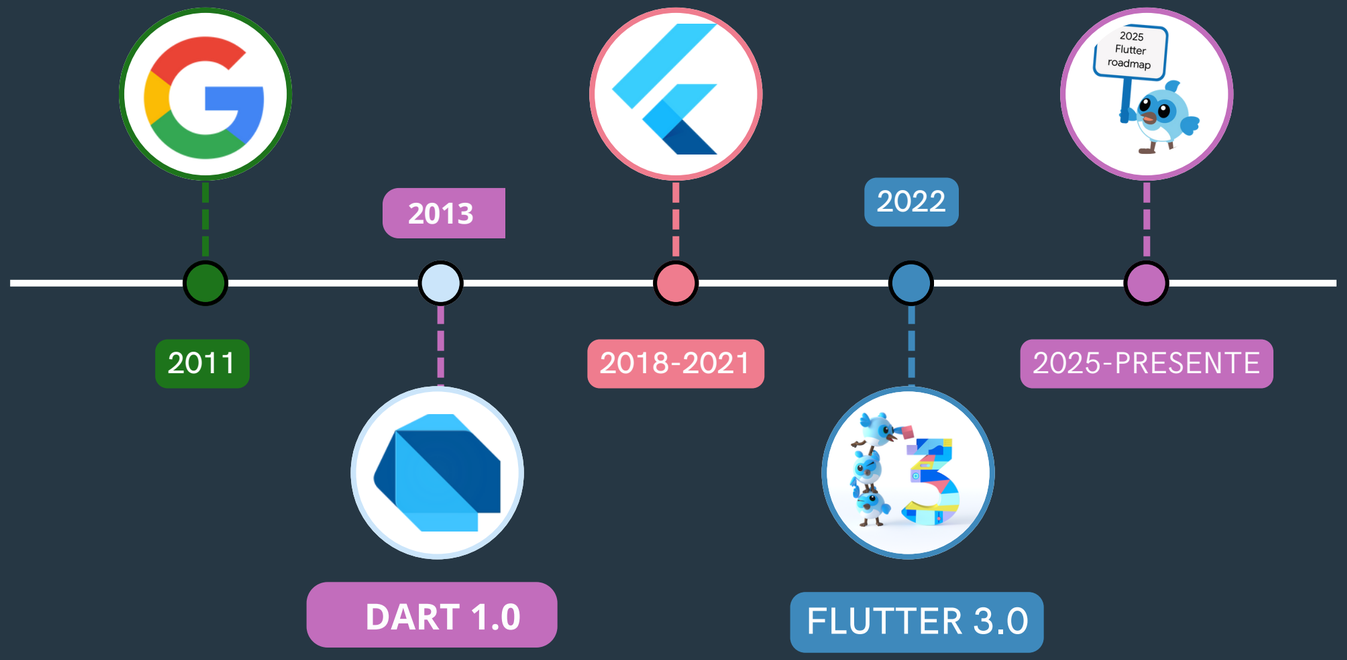
Google presenta Dart, con el propósito de reemplazar a JavaScript

CRECIMIENTO

Se lanza Flutter 1.0. Comienza el soporte para Android y iOS. Incorpora el tipado fuerte , crece en popularidad alzanzando 500,000 apps publicadas y 4 millones de desarrolladores

CONSOLIDACIÓN

Flutter es un líder indiscutible en desarrollo multiplataforma Competencia con React Native, .NET MAUI, Kotlin Multiplatform



Se lanza oficialmente la primera versión, pero JavaScript sigue siendo la opción popular

Alcanza soporte estable para las 6 plataformas: iOS, Android, Web, Windows, macOS, Linux. Consigue 5 millones de desarrolladores en Git

TIPOS DE DATOS EN DART

Objeto

num (Supertipo Numérico)
Descripción: Tipo padre de int y double
Uso: Cuando una variable puede ser entero o decimal
Tamaño: 64 bits

bool (Booleano)
Descripción: Valor lógico verdadero o falso
Tamaño: 1 bit de información (pero ocupa más en memoria por alineación)
Valores: Solo true o false

String (Cadena de texto)
Descripción: Secuencia de caracteres
Codificación: UTF-16
Tamaño: Variable (depende de la longitud)
Memoria: ~2 bytes por carácter + overhead

int (Entero)
Descripción: Números enteros sin decimales
Tamaño: 64 bits (8 bytes)
Rango: -2^{63} a $2^{63}-1$
Mínimo: -9,223,372,036,854,775,808
Máximo: 9,223,372,036,854,775,807

double (Punto Flotante)
Descripción: Números con decimales
Tamaño: 64 bits (8 bytes)
Estándar: IEEE 7
precisión: ~15-17 dígitos decimales
Rango: Aproximadamente $\pm 1.7976931348623157 \times 10^{308}$

int (num)	42
double (num)	3.14159
bool	true/false
String	‘Hello’ + “World”

Tipos Compuestos (Colecciones)

1. List (Listas)
Es la colección más común. Es un grupo ordenado de objetos donde cada elemento tiene un índice (empezando desde 0).

```
List<String> frutas = ['Manzana', 'Pera', 'Manzana'];
```

2. Set (Conjuntos)
Es una colección de elementos donde cada objeto debe ser único. No permite duplicados.

- Uso: Cuando necesitas asegurar que no haya elementos repetidos

```
Set<String> invitados = {'Juan', 'Maria', 'Pedro', 'Juan'};
```

3. Map (Mapas / Diccionarios)
Es una colección de pares llave-valor. Cada llave es única y se usa para acceder a su valor correspondiente.

- Uso: Para representar datos estructurados, como la información de un usuario.

```
Map<String, dynamic> usuario = {
  'nombre': 'Juan',
  'edad': 25,
  'esPremium': true
};
```

4.Tipos Especiales
dynamic (Tipo Dinámico)
Descripción: Variable que puede ser de cualquier tipo
Type checking: Deshabilitado en tiempo de compilación
Verificación: En tiempo de ejecución
Cuándo usar: Cuando el tipo es realmente desconocido (raro)
Riesgo: Errores en tiempo de ejecución

```
dynamic variableLoca = 'Hola';
variableLoca = 10;     // No hay error
variableLoca = true;   // Sigue sin haber error
```

5.Null (Ausencia de valor)
Descripción: Representa ausencia de valor
Null Safety: Sistema para prevenir errores de null
Tipos Nullable: Se marcan con ?
Tipos Non-nullable: Por defecto, no aceptan null
Ejemplos:

```
String nombre = "Gemini"; // No puede ser null
String? apellido;         // SÍ puede ser null (empieza siendo null)
```

Estructuras de Control

CICLOS

For

```
for (int i = 0; i < 5; i++) {
  print(i);
}
```

For-in

```
for (var item in lista) {
  print(item);
}
```

While

```
while (condicion) {
  // código
}
```

Do-While

```
do {
  // código
} while (condicion);
```

CONDICIONALES

if

```
if (condicion) {
 // código si es verdadero
}
else if (otraCondicion) {
// código si la segunda es
verdadera
}
else {
// código si ninguna se
cumple
}
```

Switch Case

```
switch (variable) {
  case valor1:
    // código
    break;
  default:
    // código por defecto
}
```

VENTAJAS

- Multiplataforma Real: Un solo código para iOS, Android, Web y Escritorio.
- UI Consistente: Al usar su propio motor, las animaciones son fluidas y la app se ve idéntica en cualquier dispositivo.
- Respaldo y Actualizaciones: Google mantiene un ciclo de actualizaciones muy activo.
- Curva de Aprendizaje: Si sabes Java, C# o JavaScript, aprender Dart es muy rápido

DESVENTAJAS

- Peso de la App: Las apps de Flutter son 4-7MB más pesadas que las nativas puras.
- Consumo de Memoria: Al incluir su propio motor gráfico y entorno de ejecución (runtime), el consumo de RAM es más elevado que en apps nativas.
- Ecosistema muy específico: Aunque pub.dev es muy bueno y organizado, se queda corto frente al ecosistema de JavaScript
- Aislamiento de Dart: Fuera de Flutter, Dart tiene muy poco uso profesional. No es un lenguaje “general” como JavaScript o Python.

Uso ideal y limitaciones

- Desarrollo multiplataforma con una sola base de código para Android, iOS, Web y Escritorio
- Proyectos con poco tiempo o presupuesto que necesitan salir rápido al mercado
- Creación de MVPs para validar una idea en varias plataformas al mismo tiempo
- Equipos que buscan reducir costos de desarrollo y mantenimiento
- Mantiene consistencia visual y funcional entre plataformas
- Acelera el desarrollo y simplifica el mantenimiento

Usos principales

- Aplicaciones multiplataforma
- Interfaces dinámicas con cambios frecuentes como redes sociales o chats
- Dashboards para visualización de métricas en tiempo real (ejemplo: pantalla de terminales de aeropuertos)

Limitaciones de Dart / Flutter

- Ecosistema más limitado frente a Java, Python o C++
- Menor disponibilidad de librerías y herramientas especializadas
- Integración más compleja con sistemas legacy
- No ideal para aplicaciones de una sola plataforma
- No recomendado para alto rendimiento crítico como juegos 3D complejos o procesamiento intensivo en tiempo real

Característica	Flutter / Dart (Reactivo)	Java (Orientado a Objetos)	Python (Funcional/Scripting)
Paradigma Principal	**Reactivo / Declarativo**: La UI reacciona a cambios de estado.	**Orientado a Objetos (POO)**: Basado estrictamente en clases y objetos.	**Multiparadigma (énfasis Funcional)**: Destaca por su flexibilidad y funciones.
Unidad de Código	**Widget**: Todo es un componente visual que se redibuja.	**Clase**:El molde que define cómo es y qué hace un objeto.	**Función / Script**: Bloques de código que procesan datos.
Manejo de Datos	**Flujos (Streams)**: Datos que llegan de forma asíncrona y continua.	**Estado del Objeto**: Datos encapsulados que mutan dentro de la instancia.	**Inmutabilidad**: Se prefiere procesar datos sin modificarlos directamente.
Sintaxis / Estilo	Declarativo (Dices *qué* quieres ver en pantalla).	Imperativo (Dices *cómo* realizar cada paso secuencial).	Multiparadigma Dices qué y cómo hacer las cosas.
Uso Ideal	Interfaces de usuario fluidas y aplicaciones móviles.	Sistemas empresariales masivos y aplicaciones Android nativas.	Ciencia de datos, IA, scripts rápidos y automatización.

Un StatefulWidget en Flutter es un widget con estado mutable
La clase Selector... hereda de la clase Stateful Widget

@override: Indica que sobrescribe un método de la clase padre.

State: Es la clase que contiene la lógica y las variables que sí pueden cambiar (como variables rojo, verde y azul).

createState() : Crea y devuelve la clase que manejará los cambios del widget.

build(): Método que construye/reconstruye la UI cuando cambia el estado.

return Scaffold: Widget que proporciona la estructura visual básica (el esqueleto)

```dart
void main() => runApp(const MaterialApp(home: SelectorColorReactivo()));

// Definición de un widget con estado (Stateful), necesario porque los colores cambiarán dinámicamente.
class SelectorColorReactivo extends StatefulWidget {
  const SelectorColorReactivo({super.key});

  @override
  State<SelectorColorReactivo> createState() => _SelectorColorReactivoState();
}

// La clase State donde reside la lógica y las variables que cambian.
class _SelectorColorReactivoState extends State<SelectorColorReactivo> {
  // Variables para almacenar los valores de los componentes RGB (0 a 255).
  double rojo = 100;
  double verde = 150;
  double azul = 200;

  @override
  Widget build(BuildContext context) {
    // Se calcula el color final combinando los valores actuales de los sliders.
    // .toInt() es necesario porque Color.fromRGBO pide enteros, pero el Slider usa doubles.
    Color colorActual = Color.fromRGBO(
      rojo.toInt(),
      verde.toInt(),
      azul.toInt(),
      1, // Opacidad total (1.0).
    );

    return Scaffold(
      backgroundColor: colorActual, // El fondo de la pantalla cambia según los sliders.
```

PARADIGMA REACTIVO

Es un paradigma declarativo enfocado en el procesamiento asincrónico de eventos y flujos de datos.

Flutter es el marco de trabajo que nos permite desarrollar sistemas multiplataforma, es el CÓMO se ve y REACCIONA.

Gracias al manejo de widgets en Flutter, es posible crear interfaces de usuario personalizadas y altamente reactivas de manera eficiente. Cada elemento visual es un widget que se puede componer, reutilizar y actualizar dinámicamente, permitiendo construir aplicaciones con diseños complejos y un rendimiento óptimo en múltiples plataformas.

Principio fundamental: Cuando los DATOS cambian, la UI se ACTUALIZA automáticamente.

Aunque su principal enfoque está en aplicaciones móviles, tiene la capacidad de funcionar perfectamente como aplicación de escritorio o web.

Dart es un lenguaje de propósito general, orientado a objetos, desarrollado por Google y optimizado para aplicaciones multiplataforma. Es donde se encuentra la LÓGICA de CÓMO funciona el sistema

La Disciplina de Tipos que manera lo hace muy seguro y menos propenso a errores al ser FUERTE y ESTÁTICO, su inferencia de tipos provee la flexibilidad de que el programa pueda deducir qué tipo es.

Sus tipos de datos nos permiten una gran variedad de operaciones, como en cualquier lenguaje de propósito general, pero sin tener la flexibilidad de lenguajes como Python.

Sus estructuras de control nos permiten dictar la lógica del programa, el CÓMO se comporta, con sus ciclos y condicionales.

GRACIAS

POR SU ATENCIÓN