

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.

Resumen de las Estructuras de Control en JavaScript

Introducción

- ▶ Las estructuras de control son fundamentales en la programación porque permiten dirigir el flujo de ejecución de un programa, tomando decisiones, repitiendo bloques de código y evaluando condiciones.
- ▶ En JavaScript, las principales estructuras de control son condicionales, bucles, y estructuras de control de interrupción.
- ▶ En las siguientes diapositivas vamos a repasar los tipos principales y cuándo se deben usar.
 - ✓ Condicionales
 - ✓ Bucles
 - ✓ Interrupción
 - ✓ Excepciones

1.CONDICIONALES

Decisiones en el Código

- 1.1 if, else if y else
- 1.2 switch

▶ 1.1 if, else if y else

- ▶ **Uso:** Para ejecutar un bloque de código solo si se cumple una condición.

- ▶ **Sintaxis:**

```
if (condicion) {  
    // Código a ejecutar si la condición es verdadera  
} else if (otraCondicion) {  
    // Código a ejecutar si la segunda condición es verdadera  
} else {  
    // Código a ejecutar si ninguna de las condiciones es verdadera  
}
```

- ▶ **Cuándo usarlo:** Cuando necesitas tomar decisiones binarias o múltiples basadas en condiciones.

1.CONDICIONALES

Decisiones en el Código

- 1.1 if, else if y else
- 1.2 switch

▶ 1.2 switch

- ▶ **Uso:** Para evaluar una variable con múltiples casos específicos.
- ▶ **Sintaxis:**

```
switch (expresion) {  
    case valor1:  
        // Código si expresion ===  
valor1  
        break;  
    case valor2:  
        // Código si expresion ===  
valor2  
        break;  
    default:  
        // Código si ninguno de los casos coincide  
}
```

- ▶ **Cuándo usarlo:** Cuando hay que evaluar un valor con muchos casos posibles (por ejemplo, un menú de opciones).

2.BUCLES

Repetir Acciones

- 2.1 for
- 2.2 While
- 2.3 Do...while
- 2.4 for...of

▶ 2.1 for

- ▶ **Uso:** Se utiliza para repetir un bloque de código un número específico de veces.

- ▶ **Sintaxis:**

```
for (inicializacion; condicion; incremento) {  
    // Código a ejecutar en cada iteración  
}
```

- ▶ **Cuándo usarlo:** Ideal para **recorrer arrays**, o para repetir un bloque de código un número determinado de veces.

2.BUCLES

Repetir Acciones

- 2.1 for
- 2.2 While
- 2.3 Do...while
- 2.4 for...of

▶ 2.2 While

- ▶ **Uso:** Se ejecuta **mientras** se cumpla una condición.
- ▶ **Sintaxis:**

```
while (condicion) {  
    // Código a ejecutar mientras la condición sea verdadera  
}
```

- ▶ **Cuándo usarlo:** Cuando no se sabe exactamente cuántas veces se repetirá el bucle, pero la ejecución depende de una condición.

2.BUCLES

Repetir Acciones

- 2.1 for
- 2.2 While
- 2.3 Do...while
- 2.4 for...of

▶ 2.3 Do...while

- ▶ **Uso:** Similar a while, pero la condición se evalúa **después** de ejecutar el bloque al menos una vez.

- ▶ **Sintaxis:**

```
do {  
    // Código a ejecutar al menos una vez  
} while (condicion);
```

- ▶ **Cuándo usarlo:** Cuando quieres **asegurarte de que el bloque se ejecute al menos una vez**, incluso si la condición inicial es falsa.

2.BUCLES

Repetir Acciones

- 2.1 for
- 2.2 While
- 2.3 Do...while
- 2.4 for...of

▶ 2.4 for...of

▶ **Uso:** Para recorrer **cada elemento** de un array.

▶ **Sintaxis:**

```
for (let elemento of array) {  
    // Código a ejecutar por cada elemento  
}
```

▶ **Cuándo usarlo:** Cuando necesitas **recorrer los elementos** de un array o una colección iterable.

3. Estructuras de Control de Interrupción

Modificar el flujo de ejecución dentro de bucles o condicionales.

- 3.1 break
- 3.2 continue

▶ 3.1 break

- ▶ **Uso:** Para recorrer **cada elemento** de un array.
- ▶ **Sintaxis:**

```
break;
```

- ▶ **Cuándo usarlo:** Cuando quieres **salir anticipadamente** de un bucle o detener la ejecución de un switch.

3. Estructuras de Control de Interrupción

Modificar el flujo de ejecución dentro de bucles o condicionales.

- 3.1 break
- 3.2 continue

▶ 3.2 continue

- ▶ **Uso:** Salta a la siguiente iteración de un bucle.

- ▶ **Sintaxis:**

```
continue;
```

- ▶ **Cuándo usarlo:** Cuando quieres omitir el resto de la iteración actual y pasar a la siguiente.

4. Estructuras de Control de Excepciones

Manejar errores en el flujo del programa para evitar bloqueos.

➤ 4.1 try...catch

▶ 4.1 try...catch

▶ **Uso:** Para **atrapar errores** en bloques de código y manejar excepciones.

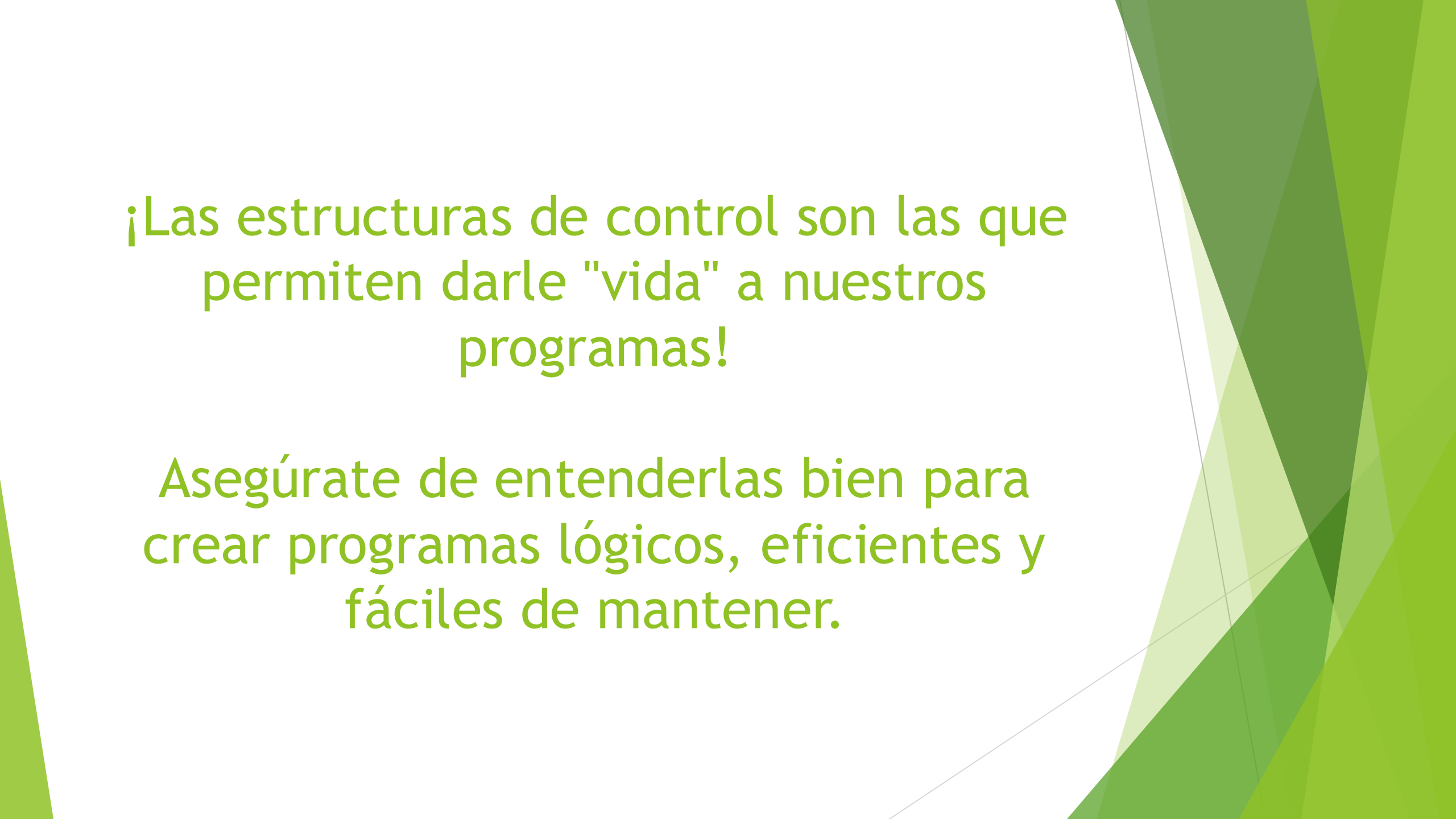
▶ **Sintaxis:**

```
try {  
    // Código que puede generar un error  
} catch (error) {  
    // Código para manejar el error  
}
```

▶ **Cuándo usarlo:** Para envolver código que puede fallar (por ejemplo, acceso a archivos o peticiones de red) y prevenir que el programa se detenga.

Recomendaciones

- ▶ Usa **if-else** para decisiones simples y **switch** para decisiones múltiples.
- ▶ Evita bucles **while** si puedes usar **for** con un número conocido de repeticiones, ya que los **while** pueden causar bucles infinitos si no se actualiza correctamente la condición.
- ▶ Usa **continue** y **break** con moderación, ya que pueden hacer que el código sea difícil de seguir si se abusa de ellos.
- ▶ Prefiere **for...of** en lugar de **for** tradicional para recorrer arrays cuando no necesitas el índice.
- ▶ Maneja errores con **try...catch** siempre que realices operaciones que pueden fallar (como acceso a APIs o archivos).

The background of the slide features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic look.

¡Las estructuras de control son las que
permiten darle "vida" a nuestros
programas!

Asegúrate de entenderlas bien para
crear programas lógicos, eficientes y
fáciles de mantener.