



Estructuras de Control

Centro de Servicios y Gestión Empresarial
SENA Regional Antioquia

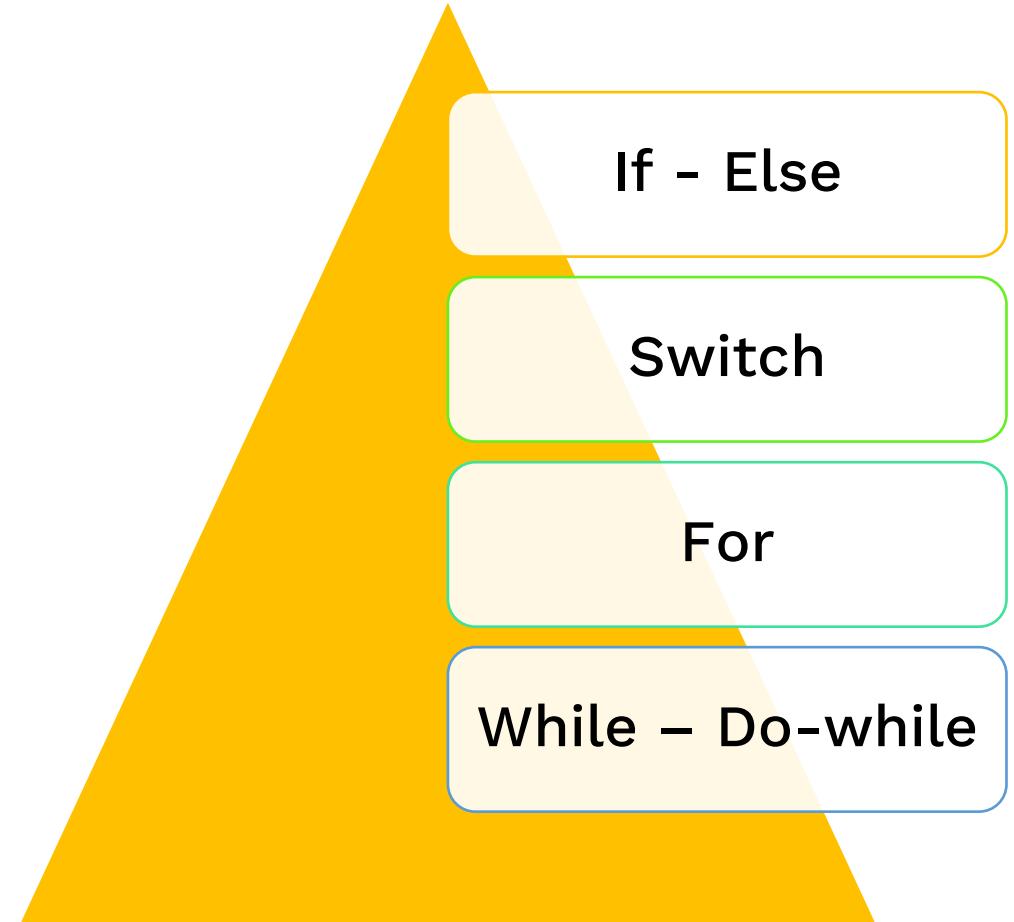


www.sena.edu.co

Contextualización

Estructuras de Control

Las estructuras de control permiten ejecutar diferentes bloques de código en función de ciertas condiciones o repetir acciones varias veces. En JavaScript, las principales estructuras de control son:



Estructura Condicional if - else

Estructura Condicional if - else

Se usa cuando queremos ejecutar código dependiendo de una condición.

Sintaxis:

```
if (condicion) {  
    // Código si la condición es verdadera  
} else {  
    // Código si la condición es falsa  
}
```

Estructura Condicional if - else

Ejemplo:

💡 Un script que verifica si una persona es mayor de edad:

```
let edad = Number(prompt("Ingrese su edad"));

if (edad >= 18) {
    console.log("Eres mayor de edad.");
    alert("Eres mayor de edad.");
} else {
    console.log("Eres menor de edad.");
    alert("Eres menor de edad.");
}
```

Estructura Condicional if - else

Ejercicio:

💡 Un evento solo permite la entrada a personas mayores de 18 años. Si el usuario tiene entre 15 y 17 años, se le ofrece un permiso con autorización de un adulto. Si tiene menos de 15, no puede entrar.

Estructura Condicional if - else



- ❖ Un evento solo permite la entrada a personas mayores de 18 años. Si el usuario tiene entre 15 y 17 años, se le ofrece un permiso con autorización de un adulto. Si tiene menos de 15, no puede entrar.

```
let edad = Number(prompt("Ingrese su edad"));

if (edad >= 18) {
    alert("Puedes entrar al evento.");
} else if (edad >= 15) {
    alert("Necesitas autorización de un adulto para entrar.");
} else {
    alert("No puedes entrar al evento.");
}
```

Estructura switch

(Selección Múltiple)

Estructura switch (*Selección Múltiple*)

Se usa cuando tenemos varias opciones posibles y queremos ejecutar diferentes bloques de código según el caso.

Sintaxis:

```
switch (valor) {  
    case opcion1:  
        // Código si valor === opcion1  
        break;  
    case opcion2:  
        // Código si valor === opcion2  
        break;  
    default:  
        // Código si ninguna opción coincide  
}
```

Estructura switch (*Selección Múltiple*)

Ejemplo:

💡 Un script que informe los requisitos para la jubilación:

```
let genero = prompt("Ingrese su género (M/F):");
switch (genero) {
    case "M":
        alert("Los requisitos de jubilación para hombres son: 62 años de edad y 1.300
semanas de cotización.");
        break;
    case "F":
        alert("Los requisitos de jubilación para mujeres son: 57 años de edad y 1.100
semanas de cotización.");
        break;
    default:
        console.log("Género no válido.");
        alert("Género no válido.");
}
```

Estructura switch (*Selección Múltiple*)

Ejercicio:

💡 El usuario seleccionará una opción de conversión de moneda:

- Dólares a Euros
- Dólares a Pesos Colombianos
- Dólares a Yen Japonés

El programa pedirá la cantidad en dólares y mostrará el monto convertido.

Estructura switch (*Selección Múltiple*)



```
let opcion = Number(prompt("Seleccione una opción de conversión: \n1. USD a EUR \n2. USD a COP \n3. USD a JPY"));
let cantidad = Number(prompt("Ingrese la cantidad en dólares:"));
let resultado;
switch (opcion) {
  case 1:
    resultado = cantidad * 0.92; // Conversión USD a EUR
    alert(`${cantidad} USD equivalen a ${resultado.toFixed(2)} EUR`);
    break;
  case 2:
    resultado = cantidad * 4000; // Conversión USD a COP (ejemplo)
    alert(`${cantidad} USD equivalen a ${resultado.toFixed(2)} COP`);
    break;
  case 3:
    resultado = cantidad * 150; // Conversión USD a JPY (ejemplo)
    alert(`${cantidad} USD equivalen a ${resultado.toFixed(2)} JPY`);
    break;
  default:
    alert("Opción no válida.");
}
```

Bucle for

(Ciclo Controlado)

Bucle for (Ciclo Controlado)

Se usa cuando queremos repetir un bloque de código un número determinado de veces.

Sintaxis:

```
for (inicio; condición; incremento) {  
    // Código a ejecutar en cada iteración  
}
```

Bucle for (Ciclo Controlado)

Ejemplo:

💡 Un restaurante desea calcular la propina total que deben dejar cinco clientes después de pagar su cuenta. Se asume que la propina a dejar es el 10% del valor de la cuenta.

```
let propina = 0;
for (let i = 1; i <= 5; i++) {
    let cuenta = Number(prompt("Ingrese el total de la
cuenta:"));
    propina += cuenta * 0.1;
}
console.log(`La propina total es de: ${propina}`);
```

Bucle for (Ciclo Controlado)

Ejercicio:

💡 Un profesor quiere calcular el promedio de calificaciones de 5 estudiantes. Se pedirá cada calificación y se calculará el promedio.

Bucle for (Ciclo Controlado)



```
let sumNotas = 0;

for (let i = 1; i <= 5; i++) {
    let nota = Number(prompt(`Ingrese la calificación No.${i}
del estudiante: `));
    sumNotas += nota;
}

let promedio = sumNotas / 5;
sumNotas /= 5; //Opcional

alert(`El promedio de calificaciones es:
${promedio.toFixed(2)}`);
```

Bucle while

(Ciclo Indeterminado)

Bucle while (*Ciclo Indeterminado*)

Se usa cuando no sabemos cuántas veces se ejecutará el código, pero queremos repetirlo mientras una condición sea verdadera.

Sintaxis:

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

Bucle while (*Ciclo Indeterminado*)

Ejemplo:

💡 Un programa que solicita al usuario ingresar su contraseña. Si es correcta mostrar un mensaje de acceso correcto y si no es correcta seguirá solicitando la contraseña.

```
let clave = prompt("Ingrese la contraseña:");

while (clave !== "1234") {
    alert("Contraseña incorrecta. Inténtelo de nuevo.");
    clave = prompt("Ingrese la contraseña:");
}

alert("Contraseña correcta. Acceso concedido.");
```

Bucle while (*Ciclo Indeterminado*)

Ejercicio:

💡 Un supermercado tiene 50 productos en stock. Cada vez que se vende un producto, el stock se reduce. El sistema debe solicitar cuántos productos se venden en cada transacción y actualizar el stock. El proceso se repite hasta que no queden productos disponibles.

Bucle while (*Ciclo Indeterminado*)



```
let stock = 50;

while (stock > 0) {
    let venta = Number(prompt(`Stock actual: ${stock}\nIngrese la cantidad de productos vendidos:`));

    if (venta > stock) {
        alert("No hay suficiente stock disponible.");
    } else {
        stock -= venta;
        alert(`Venta realizada exitosamente. Stock restante: ${stock}`);
    }
}

alert("¡Stock agotado! No se pueden realizar más ventas.");
```

Bucle do-while

(Ciclo Indeterminado con Ejecución Obligatoria)

Bucle do-while (*Ciclo Indeterminado con Ejecución Obligatoria*)



Es similar a **while**, pero siempre ejecuta el código al menos una vez, incluso si la condición es falsa.

Sintaxis:

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

Bucle do-while (*Ciclo Indeterminado con Ejecución Obligatoria*)



Ejemplo:

❖ Un banco necesita un sistema que permita a los usuarios retirar dinero de su cuenta. Sin embargo, el cajero solo permite múltiplos de \$10. Si el usuario ingresa un valor incorrecto, el sistema debe volver a solicitar un monto hasta que sea válido.

```
let monto;
do {
    monto = Number(prompt("Ingrese el monto a retirar (debe ser
múltiplo de 10):"));
} while (monto % 10 !== 0);
console.log(`Retiro exitoso: ${monto}`);
alert(`Has retirado: ${monto}`);
```

Bucle do-while (*Ciclo Indeterminado con Ejecución Obligatoria*)



Ejercicio:

💡 Un sistema de registro debe pedir al usuario un nombre válido para continuar. Se considera válido si tiene al menos 5 caracteres. Si el nombre ingresado es demasiado corto, el sistema debe solicitarlo nuevamente hasta que sea válido.

Bucle do-while (*Ciclo Indeterminado con Ejecución Obligatoria*)

Ejercicio:

💡 Un sistema de registro debe pedir al usuario un nombre válido para continuar. Se considera válido si tiene al menos 5 caracteres. Si el nombre ingresado es demasiado corto, el sistema debe solicitarlo nuevamente hasta que sea válido.

```
let nombre;

do {
    nombre = prompt("Ingrese su nombre (mínimo 5
caracteres):");
} while (nombre.length < 5);

alert(`Registro exitoso. Bienvenido, ${nombre}!`);
```

Retroalimentación

Estructura	Uso	Condición	Ejecuta al menos una vez?
if-else	Ejecutar código según una condición	if (condicion)	✗ No
switch	Selección múltiple según valor	switch (valor)	✗ No
for	Repetir un bloque un número fijo de veces	for (inicio; condición; incremento)	✗ No
while	Repetir mientras la condición sea verdadera	while (condicion)	✗ No
do-while	Repetir al menos una vez y luego validar la condición	do { ... } while (condicion)	✓ Sí

Parseo de Valores

Parseo de Valores

El **parseo** (*parsing*) en JavaScript es el proceso de convertir datos de un tipo a otro, generalmente de string a número o viceversa. Se utiliza cuando obtenemos valores de entrada (`prompt()`, formularios HTML, APIs, etc.) y necesitamos manipularlos correctamente.

Cuando obtenemos un valor con **`prompt()`**, se almacena como string, incluso si ingresamos un número. Para hacer cálculos, debemos convertirlo a número.

Parseo de Valores

Convertir String a Número (parseInt, parseFloat, Number)

Sintaxis:

Método	Conversión	Ejemplo	Resultado
<code>parseInt()</code>	<code>String → Entero</code>	<code>parseInt("42")</code>	42
<code>parseFloat()</code>	<code>String → Decimal</code>	<code>parseFloat("42.5")</code>	42.5
<code>Number()</code>	<code>String → Número (entero o decimal)</code>	<code>Number("42.5")</code>	42.5

Parseo de Valores

Ejemplo:

```
let numeroTexto = prompt("Ingrese un número:");
let numeroEntero = parseInt(numeroTexto); // Convierte a entero
let numeroDecimal = parseFloat(numeroTexto); // Convierte a decimal
let numeroGeneral = Number(numeroTexto); // Detecta el tipo (entero
o decimal)

console.log("Número como entero:", numeroEntero);
console.log("Número como decimal:", numeroDecimal);
console.log("Número detectado automáticamente:", numeroGeneral);
```

Parseo de Valores



Resumen de Métodos de Parseo en JavaScript

Conversión	Método	Ejemplo	Resultado
String → Entero	parseInt()	parseInt("42")	42
String → Decimal	parseFloat()	parseFloat("42.5")	42.5
String → Número (auto)	Number()	Number("42.5")	42.5
Número → String	toString()	(42).toString()	"42"
String → Boolean	Boolean()	Boolean("Hola")	true
Boolean → Número	Number()	Number(true)	1
String → Array	split()	"Juan, Ana".split(", ")	["Juan", "Ana"]

Convertir Texto a Mayúsculas y Minúsculas

Convertir Texto a Mayúsculas y Minúsculas



🚀 En JavaScript, podemos modificar el formato de un texto con los métodos:

toUpperCase() → Convierte un string a mayúsculas
toLowerCase() → Convierte un string a minúsculas

Ejemplo:

```
let mensajeUno = "Hola, ¿cómo estás?";
let mayusculas = mensajeUno.toUpperCase();
console.log(mayusculas); // "HOLA, ¿CÓMO ESTÁS?"

let mensajeDos = "ESTE ES UN MENSAJE IMPORTANTE";
let minusculas = mensajeDos.toLowerCase();
console.log(minusculas); // "este es un mensaje importante"
```

Manejo de Errores

1 Importancia

- Evita que el programa se bloquee cuando ocurre un error.
- Permite manejar errores de manera controlada y mostrar mensajes personalizados.
- Mejora la experiencia del usuario en aplicaciones web, evitando que se muestren errores inesperados.
- Facilita la depuración de código, registrando errores sin interrumpir la ejecución.

Try - catch

El bloque try-catch en JavaScript se usa para manejar errores de forma segura sin detener la ejecución del programa. Permite capturar excepciones y ejecutar un código alternativo en caso de error.

Sintaxis:

```
try{  
    // Código que podría generar un error  
}catch(error){  
    // Código que se ejecuta si se produce un error  
}finally{  
    // Código que se ejecuta siempre al final - opcional  
}
```

Try - catch

2 Casos comunes:

- Operaciones con datos inválidos (ejemplo: dividir por cero).
- Errores en conversiones de datos (ejemplo: convertir un string a número).
- Errores en el acceso a objetos o propiedades no definidas.
- Errores en conexiones con servidores o bases de datos (ejemplo: `fetch()` fallido).
- Lectura de datos de APIs o archivos externos
- Manipulación del DOM (si un elemento no existe)
- Conversión de datos (`JSON.parse()`)
- Errores en operaciones asincrónicas (ejemplo: uso de `async/await`).

Try - catch

Ejemplo:

```
const dividendo = 10n; // Usando BigInt  
const divisor = 0n;    // Divisor cero (BigInt)  
const resultado = dividendo / divisor; // Intento de división por cero  
console.log(resultado);
```

1. El usuario no ingrese un valor numérico
2. Los números deben ser positivos
3. El divisor debe ser mayor a cero

Try - catch

Ejemplo:

```
try {
    const dividendo = 10n; // Usando BigInt
    const divisor = 0n;    // Divisor cero (BigInt)
    const resultado = dividendo / divisor; // Intento de
división por cero
    console.log(resultado);
} catch (error) {
    console.log("¡Error detectado: No se puede dividir por
Cero!");
}
```

Try - catch

💡 Propiedades y Métodos del Objeto Error en JavaScript

Cuando se captura un error con `catch(error)`, el objeto `error` tiene varias propiedades y métodos útiles que nos permiten obtener más información sobre el problema:

Propiedad	Descripción	Ejemplo
message	Muestra el mensaje de error	<code>error.message</code> → "División por cero no permitida"
name	Nombre del error (<code>Error</code> , <code>TypeError</code> , <code>ReferenceError</code> , etc.)	<code>error.name</code> → "TypeError"
stack	Traza del error (muestra en qué línea ocurrió)	<code>error.stack</code> → "Error at script.js:10:5"
cause (opcional, en ES2022)	Describe la causa del error	<code>error.cause</code> → "API no disponible"

Try - catch

Ejemplo: Operaciones con datos inválidos

```
try {
    const dividendo = 10n; // Usando BigInt
    const divisor = 0n;    // Divisor cero (BigInt)
    const resultado = dividendo / divisor; // Intento de división por cero
    console.log(resultado);
} catch (error) {
    console.log("¡Error detectado: No se puede dividir por Cero!");
    console.log("Mensaje de error:", error.message);
    console.log("Nombre del error:", error.name);
    console.log("Traza del error:", error.stack);
    console.log("Causa del error:", error.cause);
}
```

Try - catch

Ejemplo: Errores en conversiones de datos

```
try {
  let valor = "abc123"; // Cadena no numérica
  let numero = BigInt(valor); // Intento de conversión a BigInt
  console.log(numero);
} catch (error) {
  console.log("¡Error detectado:", error.message + "!");
}
```

Try - catch

Ejemplo: Errores en el acceso a objetos o propiedades no definidas.

```
try {  
    // Simulamos un objeto que no fue inicializado correctamente  
    let usuario = null;  
    console.log(usuario.nombre); // Esto generará un error TypeError  
} catch (error) {  
    console.log("Error detectado: No se puede acceder a una propiedad  
de un objeto nulo.");  
}
```

Try - catch

Ejemplo:

```
try{
    let cantidad = 10;
    let total = cantidad * precio; // precio no está
definido
    console.log(total);
} catch (error) {
    console.log("Error detectado: No se puede multiplicar
una variable no definida.");
}
```

Try - catch

throw: Generar Errores Personalizados

El operador **throw** nos permite crear y lanzar errores propios, en lugar de esperar a que JavaScript genere uno automáticamente. Se usa en validaciones o situaciones en las que queremos controlar el comportamiento del programa.

Sintaxis

```
throw new Error("Mensaje de error personalizado");
```

Try - catch



```
// Verificar si es undefined
try {
  let cantidad = 10;
  if (typeof precio === "undefined") {
    throw new Error("La variable precio no está definida.");
  }
} catch (error) {
  console.log("Error detectado:", error.message);
}

// Verificar si es null
try {
  let cantidad = null;
  if (cantidad === null) {
    throw new Error("La variable cantidad es nula.");
  }
} catch (error) {
  console.log("Error detectado:", error.message);
}
```

Try - catch



```
// Verificar si es NaN - No es un número
try {
  let cantidad = "Hola JS";
  if (isNaN(cantidad)) {
    throw new Error("La variable cantidad NO es un número.");
  }
} catch (error) {
  console.log("Error detectado:", error.message);
}

// // Verificar si es un número
try {
  let cantidad = "Hola JS";
  if (typeof cantidad !== "number") {
    throw new Error("La variable cantidad NO es un número.");
  }
} catch (error) {
  console.log("Error detectado:", error.message);
}
```

Try - catch



```
// Verificar si es un string
try {
  let cantidad = 10;
  if (typeof cantidad !== "string") {
    throw new Error("La variable cantidad NO es un string.");
  }
} catch (error) {
  console.log("Error detectado:", error.message);
}

// Verificar si es vacio
try {
  let cantidad = "";
  if (cantidad === "") {
    throw new Error("La variable cantidad está vacía.");
  }
} catch (error) {
  console.log("Error detectado:", error.message);
}
```

Try - catch



```
// Verificar si es Infinity
try {
    let division = 13/0;
    if (division === Infinity) {
        throw new Error("No se puede dividir por cero.");
    }
} catch (error) {
    console.log("Error detectado:", error.message);
}

try {
    let division = 13/0;
    if (isFinite(division)) {
        throw new Error("No se puede dividir por cero.");
    }
} catch (error) {
    console.log("Error detectado:", error.message);
}
```

Ejercicio de Aplicación

Try - catch



```
let edad = 23;

if (edad >= 18) {
    console.log("Eres mayor de edad.");
} else {
    console.log("Eres menor de edad.");
}
```

Try - catch



```
try {
    let edad = "SENA"; // Cadena no numérica
    let edad = null; // Valor nulo
    let edad = ""; // Valor vacío
    let edad = 18; // Valor numérico

    // Verificar si el valor es un número válido
    if (isNaN(edad) || edad === null || edad === "") {
        throw new Error("Entrada no numérica"); // Lanzar error manualmente
    }

    if (edad >= 18) {
        console.log("Eres mayor de edad.");
    } else {
        console.log("Eres menor de edad.");
    }
} catch (error) {
    console.log("Error: ", error.message);
}
```



G R A C I A S

Presentó: Alvaro Pérez Niño

Instructor Técnico

Correo: aperezn@sena.edu.co

<http://centrodesserviciosygestionempresarial.blogspot.com/>

Línea de atención al ciudadano: 01 8000 910270

Línea de atención al empresario: 01 8000 910682



www.sena.edu.co