-> LENGUAJE TYPESCRIPT
-> Tipos de Datos en LibreScript

Tipos de Datos en LibreScript

En librescript, el **tipado es obligatorio**.

numero: Abarca tanto enteros como números con decimales.

- Ejemplo de declaración y asignación: $a: numero = 0;

texto: Para cadenas de texto.

- Ejemplo de declaración y asignación: $mensaje: texto = "Hola, librescript!";

booleano: Para valores lógicos verdadero y falso.

- Ejemplo de declaración y asignación: $estaHecho: booleano = falso;, $ejemplo: booleano = verdadero;

Objeto: Para representar estructuras de datos con pares clave-valor.

- Ejemplo de declaración y asignación: $persona: Objeto = {nombre: "Ana"};

Arreglos (Arrays): Se definen con [] después del tipo base.

- Ejemplo de declaración y asignación de un arreglo de texto: $nombres: texto[] = ["Ana", "Luis", "Carlos"];

Declaración y Asignación de Variables en LibreScript:

Prefijo para Variables: El signo de **dólar ($)** es un prefijo **obligatorio** para todas las variables.

- Ejemplo: $miVariable

Separador de Tipo y Nombre: El **colon (:)** es el separador entre el nombre de la variable y su tipo.

- Ejemplo: $nombre: texto

Declaración Explícita: La declaración de variables es **explícita**, lo que significa que siempre debe indicarse el tipo de dato de la variable. No se utilizan palabras clave como `var`, `let` o `const`. La mera presencia de `$nombre: tipo` ya indica una declaración.

- Ejemplo: `$cantidad: numero = 100;`

Tipado Obligatorio e Inmutable: En LibreScript, el tipo de cada variable debe ser declarado explícitamente y **no se puede cambiar** una vez asignado.

- Ejemplo: `$edad: numero = 25;` (No puedes cambiar `$edad` a tipo `texto` más tarde).

Declaración de Constantes: Se utiliza el prefijo **doble dólar (`$$`)** para declarar una constante. Una constante **no puede cambiar su valor** una vez definido.

- Ejemplo: `$$PI: numero = 3.1416;`

Conversión de Tipos: La conversión de tipos debe ser **explícita** y se realiza mediante funciones específicas: `aNum()`, `aTxt()`, `aBool()`.

- Ejemplo:

$a: numero = 5;

$b: texto = "5";

$suma: numero = $a + aNum($b);

Reasignación: Una vez declarada una variable (que no sea una constante `$$`), se le puede asignar un nuevo valor utilizando el **operador de asignación (`=`)**. Es importante que el nuevo valor **coincida con el tipo declarado** de la variable.

$a: numero = 10; // Declaración inicial

$a = 20; // Reasignación del valor (el tipo de $a sigue siendo 'numero')

$mensaje: texto = "Hola";

$mensaje = "Adiós"; // Reasignación de una cadena

Operadores y Expresiones en LibreScript

En LibreScript, las expresiones se construyen utilizando variables, constantes, literales y una variedad de operadores. Los operadores matemáticos y lógicos siguen una prioridad estándar para evitar ambigüedades en la evaluación

Operadores Aritméticos:

+ (Suma): Suma dos operandos numéricos. También se utiliza para la concatenación de cadenas de texto.

- **- (Resta):** Resta el segundo operando del primero.
- *** (Multiplicación):** Multiplica dos operandos.
- **/ (División):** Divide el primer operando entre el segundo.
- **% (Módulo):** Obtiene el residuo de una división.
- **** (Potencia):** Eleva el primer operando a la potencia del segundo operando.
- **++ (Incremento):** Incrementa el valor de un operando en 1. (Asumiremos que es un operador unario, como `$a++;` o `++$a;`).
- **-- (Decremento):** Decrementa el valor de un operando en 1. (Asumiremos que es un operador unario, como `$a--;` o `--$a;`).

Operadores de Asignación Compuesta

+= (Asignación con Suma): `$a += 5;` es equivalente a `$a = $a + 5;`

-= (Asignación con Resta): `$a -= 5;` es equivalente a `$a = $a - 5;`

***= (Asignación con Multiplicación):** `$a *= 5;` es equivalente a `$a = $a * 5;`

/= (Asignación con División): `$a /= 5;` es equivalente a `$a = $a / 5;`

Operadores de Comparación

Estos operadores se utilizan para comparar dos valores y siempre devuelven un resultado de tipo `booleano`.

- **== (Comparación de Igualdad):** Verdadero si ambos operandos son iguales.
- **!= (Desigualdad):** Verdadero si ambos operandos son diferentes.
- **> (Mayor que):** Verdadero si el primer operando es mayor que el segundo.
- **< (Menor que):** Verdadero si el primer operando es menor que el segundo.
- **>= (Mayor o igual que):** Verdadero si el primer operando es mayor o igual que el segundo.
- **<= (Menor o igual que):** Verdadero si el primer operando es menor o igual que el segundo.

Operadores Lógicos

&& (Y Lógico): Verdadero si ambas expresiones son verdaderas.

|| (O Lógico): Verdadero si al menos una de las expresiones es verdadera.

! (NO Lógico): Invierte el valor booleano de una expresión (verdadero se convierte en falso y viceversa).

Concatenación de Cadenas

- Las cadenas de texto en LibreScript pueden manejarse utilizando **comillas dobles (") o simples (')**.
- El operador **+** se utiliza para concatenar (unir) cadenas de texto.
    - Ejemplo: `$saludo: texto = "Hola" + " " + "Mundo";`

.

Nivel de Prioridad	Operadores	Asociatividad
Más alta	`( )` (paréntesis para agrupamiento)	Izquierda
	`++`, `--`, `!` (unarios)	Derecha
	`**` (Potencia)	Derecha
	`*`, `/`, `%`	Izquierda
	`+`, `-`	Izquierda
	`>`, `<`, `>=`, `<=`	Izquierda

	`==`, `!=`	Izquierda
	`&&`	Izquierda
	`	`	`
Más baja	`=`, `+=`, `-=`, `*=`, `/=` (Asignación)	Derecha

Estructuras de Control en LibreScript

LibreScript provee las siguientes estructuras de control para manejar el flujo de ejecución del programa:

Condicionales (si-siNo si-siNo)

La estructura condicional `si` permite ejecutar bloques de código basados en la evaluación de una expresión booleana.

- **Palabras clave:** `si`, `siNo si`, `siNo`.
- **Sintaxis:**
  - La condición se encierra entre paréntesis `()`.
  - Los bloques de código se delimitan con llaves `{}`.
  - La cláusula `siNo si` permite encadenar múltiples condiciones.
  - La cláusula `siNo` es opcional y ejecuta un bloque de código si ninguna de las condiciones anteriores es verdadera.
-

Estructuras de Control: `si(if)`

```
$temperatura: numero = 25;

si ($temperatura > 30) {
  imprimir("Hace mucho calor.");
} siNo si ($temperatura > 20) {
  imprimir("La temperatura es agradable.");
} siNo {
  imprimir("Hace frío.");
}
```

```
// If sin siNo
$lluvia: booleano = falso;
si ($lluvia) {
  imprimir("Lleva un paraguas.");
}
```

Bucles (mientras)

El bucle `mientras` permite ejecutar un bloque de código repetidamente mientras una condición sea verdadera.

- **Palabras clave:** `mientras`.
- **Sintaxis:**
  - La condición se encierra entre paréntesis `()`.
  - Los bloques de código se delimitan con llaves `{}`.

Estructuras de Control: `mientras(while)`

```
$contador: numero = 0;
mientras ($contador < 5) {
  imprimir("Contador: " + $contador);
  $contador++;
}
```

Bucles (para)

El bucle `para` proporciona una forma concisa de iterar. Es útil para bucles con un número predefinido de iteraciones.

- **Palabras clave:** `para`.
- **Sintaxis:**
  - La inicialización, condición y expresión de incremento/decremento se separan por punto y coma `;` dentro de paréntesis `()`.
  - Los bloques de código se delimitan con llaves `{}`.
-

```
Estructuras de Control: para (for)
para ($i: numero = 0; $i < 3; $i++) {
  imprimir("Bucle i: " + $i);
}
```

Estructura de Selección Múltiple (segun)

La estructura `segun` permite ejecutar diferentes bloques de código basados en el valor de una expresión.

- **Palabras clave:** `segun`, `caso`, `romper`, `pordefecto`.
- **Sintaxis:**
  - La expresión a evaluar se encierra entre paréntesis `()` después de `segun`.
  - Los bloques de `caso` y `pordefecto` se definen dentro de llaves `{}`.
  - Cada `caso` debe terminar con `romper` para evitar la "caída" (fall-through) al siguiente `caso`.
  - `pordefecto` es opcional y se ejecuta si ninguna de las expresiones `caso` coincide.

```
Estructuras de Control: segun (Switch)
 $opcion: numero = 2;

segun ($opcion) {
  caso 1:
    imprimir("Seleccionaste la opción 1.");
    romper;
  caso 2:
    imprimir("Seleccionaste la opción 2.");
    romper;
  caso 3:
    imprimir("Seleccionaste la opción 3.");
    romper;
  pordefecto:
    imprimir("Opción no reconocida.");
}

$fruta: texto = "manzana";
segun ($fruta) {
    caso "pera":
        imprimir("Es una pera.");
        romper;
    caso "manzana":
        imprimir("Es una manzana.");
        romper;
}
```

Entrada y Salida (I/O) en LibreScript

La función imprimir() se utiliza para mostrar información en la consola.
Permite múltiples argumentos: Puedes pasar uno o varios argumentos separados por comas. Cada argumento se convertirá automáticamente a su representación de texto antes de ser mostrado.
Tipos de datos: Puede imprimir cualquier tipo de dato (`numero`, `texto`, `booleano`, `Objeto`, arreglos). Para `Objeto` y arreglos, se mostrará una representación de texto genérica (similar a `[object Object]` o `1,2,3` para arreglos.
Salto de línea: Cada llamada a `imprimir()` termina con un salto de línea por defecto, es decir, el siguiente `imprimir()` comenzará en una nueva línea.

```
$nombre: texto = "Mundo";
$edad: numero = 30;
$esActivo: booleano = verdadero;
$miObjeto: Objeto = {clave: "valor"};
$misNumeros: numero[] = [1, 2, 3];

imprimir("Hola,", $nombre, "!");
imprimir("Tienes", $edad, "años.");
imprimir("Activo:", $esActivo);
imprimir("Mi objeto:", $miObjeto); // Imprimirá una representación de texto del objeto
imprimir("Mis números:", $misNumeros); // Imprimirá una representación de texto del arreglo
```

Entrada de Datos

La función **`leer()`** se utiliza para obtener una línea de texto del usuario a través de la consola.

- **Sintaxis:** `$variable: tipo = leer();`

Devuelve `texto`: La función `leer()` siempre devuelve el valor introducido por el usuario como una cadena de texto (`texto`). **Conversión explícita:** Si necesitas el valor en otro tipo (como `numero` o `booleano`), deberás usar las funciones de conversión explícita (`aNum()`, `aBool()`)

```
$nombreUsuario: texto;

imprimir("Por favor, ingresa tu nombre:");

$nombreUsuario = leer();

imprimir("¡Hola,", $nombreUsuario, "!");
```

```
$edadTexto: texto;

$edadNumero: numero;

imprimir("Ingresa tu edad:");

$edadTexto = leer();

$edadNumero = aNum($edadTexto); // Convertir a número

imprimir("Tu edad es:", $edadNumero);
```

Funciones en LibreScript

En LibreScript, las **funciones** son bloques de código reutilizables diseñados para realizar tareas específicas. Permiten organizar el código de manera modular y mejorar la legibilidad.

Palabra clave: `funcion`.

Nombre de la función: Los nombres de las funciones siguen las reglas del token `IDENTIFICADOR_GRAL` (alfanumérico, empieza con letra, sin el prefijo `$`). Esto proporciona coherencia con el estilo de TypeScript para nombres de funciones.

- Ejemplo: `saludar`, `sumar`, `esMayor`.

Parámetros: Se declaran dentro de los paréntesis `()` después del nombre de la función. Cada parámetro debe tener un **nombre de variable con prefijo `$` y su tipo explícito**, separados por dos puntos `:`. Los parámetros se separan por comas.

- Ejemplo: `($a: numero, $b: numero)`

Tipo de Retorno: Es **obligatorio** especificar el tipo de valor que la función devuelve después de los paréntesis de los parámetros, usando dos puntos `:`.

- Para funciones que no devuelven explícitamente un valor, se utiliza el tipo `vacio`.

Sintaxis General:

funcion nombreDeFuncion(parametro1: tipo1, parametro2: tipo2, ...): tipoDeRetorno {

```
    // Cuerpo de la función
    // ...
    devolver valor; // Obligatorio si el tipoDeRetorno no es 'vacio'
}
EJEMPLOS DE DECLARACION DE FUNCIONES
funcion saludar(): texto {
    imprimir("¡Hola desde la función!");
    devolver "Saludo completado";
}

funcion sumar($a: numero, $b: numero): numero {
    devolver $a + $b;
}

funcion esMayor($num1: numero, $num2: numero): booleano {
    devolver $num1 > $num2;
}

// Función que no devuelve ningún valor
funcion mostrarMensaje($msg: texto): vacio {
    imprimir("Mensaje:", $msg);
}
```

Llamada a Funciones:

Para **invocar una función**, se utiliza su nombre seguido de paréntesis ( ) que contienen los argumentos (valores reales) correspondientes a sus parámetros, separados por comas.

- **Sintaxis de Invocación:** `nombreDeFuncion(argumento1, argumento2, ...);`

- **Asignación de Retorno:** Si una función devuelve un valor (es decir, su tipo de retorno no es `vacio`), este valor puede ser **asignado directamente a una variable**. Es crucial que el tipo de la variable receptora coincida con el tipo de retorno de la función.

    - Sintaxis: `$variable: tipo = nombreDeFuncion(argumento1, ...);`

```
// Llamada a función que devuelve texto
$mensaje: texto = saludar();
imprimir($mensaje); // Salida: "Saludo completado"

// Llamada a función con parámetros y retorno numérico
$suma: numero = sumar(10, 5);
imprimir("La suma es: " + $suma); // Salida: "La suma es: 15"

// Llamada a función con retorno booleano
$mayor: booleano = esMayor(7, 3);
imprimir("¿7 es mayor que 3? " + $mayor); // Salida: "¿7 es mayor que 3? verdadero"

// Llamada a función que no devuelve valor
mostrarMensaje("LibreScript es genial."); // Salida: "Mensaje: LibreScript es genial."
```

Ámbito:
LibreScript manejará el ámbito de las variables siguiendo un modelo de **ámbito léxico (o estático)**, similar al de JavaScript/TypeScript.
Variables Locales: Las variables declaradas **dentro de una función** (incluidos sus parámetros) son **locales a esa función**. Solo son accesibles desde dentro del cuerpo de esa función.

Acceso a Ámbitos Superiores: Una función puede **acceder a variables declaradas en un ámbito superior** (variables globales o variables de funciones "padre" si está anidada). Sin embargo, una función **no puede modificar directamente** variables de ámbitos superiores si estas son constantes (`$$`) o si se rige por un principio de inmutabilidad estricta (que es el caso con tu tipado obligatorio). Si la variable de ámbito superior es una variable normal (`$`), la función puede leer y modificarla.

Si una variable local tiene el mismo nombre que una variable en un ámbito superior, la variable local "sombrea" a la de ámbito superior, lo que significa que dentro de la función se accederá a la versión local.
EJEMPLO:

```
$globalVar: numero = 100; // Variable global
```

```
funcion miFuncionEjemplo($param: numero): vacio {
    $localVar: numero = 20; // Variable local
    imprimir("Dentro de la función:");
    imprimir("  Parametro: " + $param);   // Acceso a parámetro
    imprimir("  Variable local: " + $localVar); // Acceso a variable local
    imprimir("  Variable global: " + $globalVar); // Acceso a variable global
    $globalVar = 101; // Modificación de variable global (si no es constante)
}

miFuncionEjemplo(5);
imprimir("Fuera de la función, globalVar es: " + $globalVar);
```

Clases y Objetos en LibreScript

LibreScript soporta la Programación Orientada a Objetos (POO) a través de clases, permitiendo la creación de objetos que encapsulan datos (propiedades) y comportamiento (métodos).

- **Declaración de Clase:** Se usa la palabra clave `clase` seguida del nombre de la clase (que usará `IDENTIFICADOR_GRAL`). El cuerpo de la clase se encierra entre llaves `{}`.
    - **Propiedades:** Las propiedades se declaran con un nombre (que usará `IDENTIFICADOR_GRAL`) seguido de dos puntos `:` y su tipo.
        - **Propiedades Privadas:** Se usa el prefijo `#` para indicar que una propiedad es privada (ej: `#nombrePrivado`). Estas propiedades solo son accesibles desde dentro de la clase. El lexer ya tokeniza esto, y el analizador semántico deberá aplicar la restricción de acceso.
        - **Propiedades Públicas:** No llevan prefijo (ej: `edad`). Son accesibles desde cualquier lugar.
- **Constructor:** Es un método especial nombrado **`constructor`** que se ejecuta al crear una nueva instancia de la clase. Recibe parámetros para inicializar las propiedades del objeto. Dentro del constructor y otros métodos de instancia, se utiliza la palabra clave **`este`** para referirse al objeto actual y acceder a sus propiedades y métodos.
- **Métodos:** Son funciones declaradas dentro de una clase. Siguen la misma sintaxis que las funciones globales (excepto por la palabra clave `funcion` dentro de la clase).
    - Ejemplo: `nombreDeMetodo(parametros): tipoRetorno { ... }`
    - Los métodos pueden ser públicos o privados. Por defecto serán públicos.

Creación de Objetos: Se utiliza la palabra clave **nuevo** seguida del nombre de la clase y los argumentos para el constructor entre paréntesis `()`.

- Ejemplo: `$juan: Persona = nuevo Persona("Juan", 30);`

Acceso a Miembros:

- Se utiliza el operador de punto (`.`) para acceder a las propiedades y métodos de un objeto.
- Ejemplo de acceso a propiedad: `$juan.edad`
- Ejemplo de llamada a método: `$juan.saludar()`

```
// Declaración de clase simple
clase Persona {
  #nombrePrivado: texto; // Propiedad privada (para verificación semántica)
  edad: numero; // Propiedad pública

  constructor($nombre: texto, $edadInicial: numero) {
    este.nombrePrivado = $nombre;
    este.edad = $edadInicial;
  }

  saludar(): texto { //metodo publico
    devolver "Hola, mi nombre es " + este.nombrePrivado + " y tengo " + este.edad + " años.";
  }

  // Ejemplo de método con parámetro
  cambiarEdad($nuevaEdad: numero): vacio {
      este.edad = $nuevaEdad;
  }
}

// Creación de objeto
$juan: Persona = nuevo Persona("Juan", 30);
imprimir($juan.saludar());

// Acceso y asignación de propiedad pública
$juan.edad = 31;
imprimir("Nueva edad de Juan: " + $juan.edad);

// Llamada a método
$juan.cambiarEdad(32);
imprimir($juan.saludar());
```

Arreglos (Arrays):

LibreScript soporta arreglos para almacenar colecciones ordenadas de elementos del mismo tipo.

- **Declaración:** Se declara el tipo de los elementos seguido de corchetes `[]`.
    - Ejemplo: `$numeros: numero[];`
- **Inicialización:** Pueden inicializarse vacíos (`[]`) o con una lista de valores separados por comas entre corchetes.
    - Ejemplo: `$edades: numero[] = [25, 30, 22, 40];`
- **Acceso a Elementos:** Se usa el nombre del arreglo seguido de un índice numérico entre corchetes `[]`. Los índices son base 0 (el primer elemento está en el índice 0).
    - Ejemplo: `$edades[0]`
- **Asignación a Elementos:** Se puede asignar un nuevo valor a un elemento específico utilizando su índice.
    - Ejemplo: `$edades[1] = 31;`
- **Arreglos Multidimensionales (Matrices):** Se declaran añadiendo pares de corchetes por cada dimensión.
    - Ejemplo: `$matriz: numero[][] = [[1, 2], [3, 4]];`
    - Acceso a elementos multidimensionales: `$matriz[1][0]`

```
// Declaración de arreglo vacío
$numeros: numero[] = [];
// Declaración e inicialización de arreglo de números
$edades: numero[] = [25, 30, 22, 40];
// Acceso a elementos del arreglo
imprimir("Primera edad: " + $edades[0]);
imprimir("Última edad: " + $edades[3]);
// Asignación a un elemento
$edades[1] = 31;
imprimir("Nueva segunda edad: " + $edades[1]);

// Arreglos de texto
$nombres: texto[] = ["Ana", "Luis", "Marta"];
imprimir("Primer nombre: " + $nombres[0]);

// Arreglo de arreglos (matriz)
$matriz: numero[][] = [[1, 2], [3, 4]];
imprimir("Elemento central: " + $matriz[1][0]);
$matriz[0][1] = 99;
imprimir("Elemento modificado: " + $matriz[0][1]);
```

Operaciones y Expresiones Complejas:

LibreScript maneja expresiones complejas con la **prioridad y asociatividad estándar** de operadores, permitiendo el uso de paréntesis `()` para forzar un orden de evaluación.

- **Precedencia de Operadores:**
    - `**` (Potencia) tiene mayor precedencia.
    - `*`, `/`, `%` (Multiplicación, División, Módulo) tienen mayor precedencia que `+`, `-`.
    - `+`, `-` (Suma, Resta)
    - Operadores de comparación (`>`, `<`, `==`, `!=`, etc.)
    - Operadores lógicos (`&&`, `||`)
    - Operadores de asignación (`=`, `+=`, etc.)
- **Operador Unario Menos (`-`):** Se usa para negar un valor numérico.
    - Ejemplo: `-$a`

```
$a: numero = 10;
$b: numero = 20;
$c: numero = 5;

$res1: numero = $a + $b * $c;       // Precedencia: 10 + (20 * 5) = 110
$res2: numero = ($a + $b) * $c;     // Paréntesis: (10 + 20) * 5 = 150
$res3: numero = $a ** 2;            // Potencia: 10 * 10 = 100
$res4: numero = $b / $c + $a % 3;   // División (20/5=4), Módulo (10%3=1), Suma (4+1=5)

$cond1: booleano = ($a > $b) && ($b < $c);  // (falso) && (falso) = falso
$cond2: booleano = ($a < $b) || ($b < $c);  // (verdadero) || (falso) = verdadero
$cond3: booleano = !($a == $b);             // !(falso) = verdadero

imprimir("Resultados de expresiones:");
imprimir($res1); // Salida: 110
imprimir($res2); // Salida: 150
imprimir($res3); // Salida: 100
imprimir($res4); // Salida: 5
imprimir($cond1); // Salida: falso
imprimir($cond2); // Salida: verdadero
imprimir($cond3); // Salida: verdadero

$negativo: numero = -$a;
imprimir($negativo); // Salida: -10
```

LibreScript soporta dos tipos de comentarios para mejorar la legibilidad del código:

- **Comentarios de una línea:** Comienzan con `//`. Todo lo que sigue hasta el final de la línea se considera un comentario.
- **Comentarios de bloque:** Comienzan con `/*` y terminan con `*/`. Pueden abarcar múltiples líneas.

```
// Este es un comentario de línea

/*
  Este es un comentario de bloque.
  Puede ocupar múltiples líneas.
*/
```