Manual del usuario

Bienvenido a Raven, el lenguaje de programación diseñado para aquellos que ven el código no solo como instrucciones, sino como un lienzo en blanco. Raven no es solo una herramienta, es un puente entre la lógica y la imaginación, donde cada línea que escribas fluye con la elegancia de un verso y la

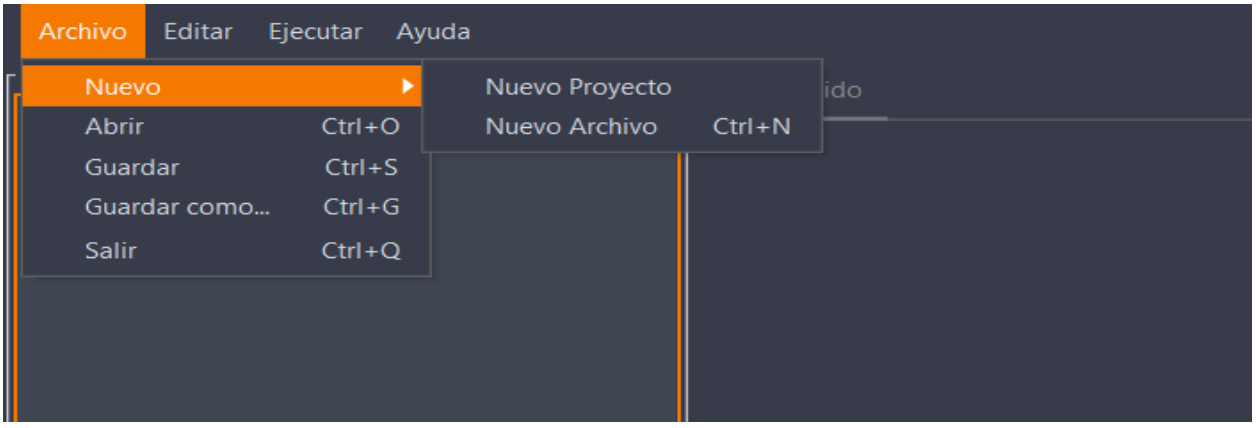
Imagina un lenguaje donde la sintaxis es intuitiva, donde la interfaz te guía como un mentor paciente y donde cada función que creas parece una pieza de arte en movimiento. Raven está construido para desarrolladores y soñadores que buscan expresar sus ideas sin límites.

Este manual es tu mapa para navegar por el universo Raven. Ya seas un explorador curioso o un maestro del código, aquí encontrarás las claves para dominar este lenguaje y dar vida a tus ideas.

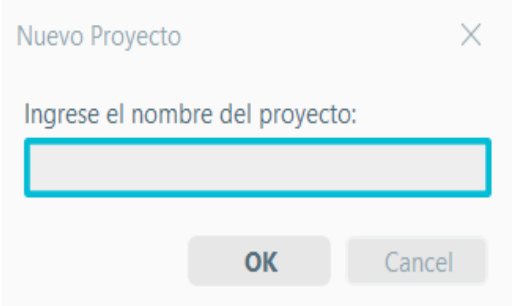
¡Comencemos el viaje!

Menú

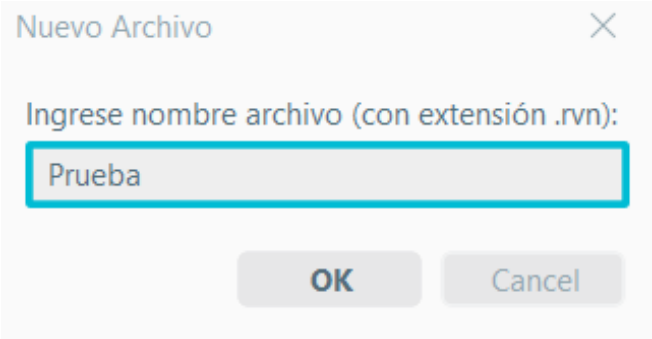
El menú cuenta con cuatro pestañas: Archivo, Editar, Ejecutar y Ayuda:



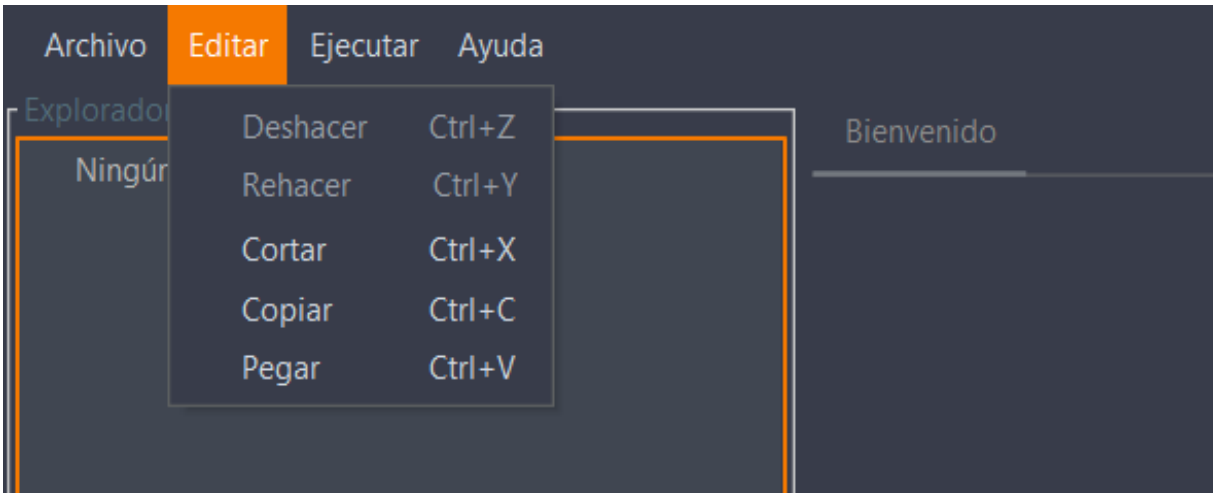
La pestaña de Archivo contiene las opciones más comunes, como Nuevo en donde se despliegan las opciones de “Nuevo Proyecto” (Carpeta donde se guardará el proyecto) y “Nuevo Archivo” (archivo sobre el que se trabajará), Abrir (Ctrl+O), Guardar (Ctrl+S), Guardar como... (Ctrl+G) y Salir (Ctrl+Q).



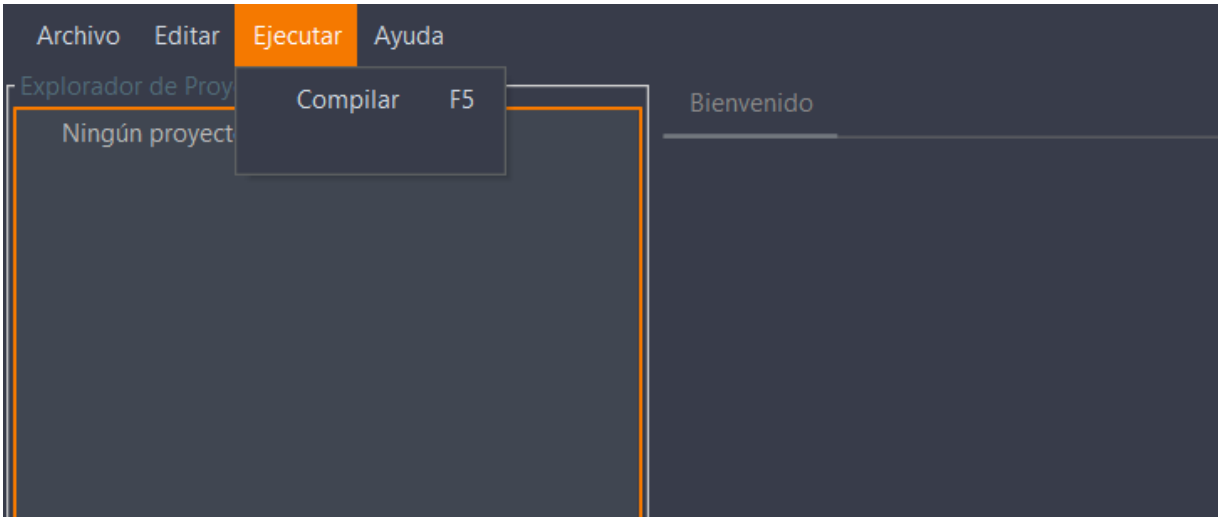
Principalmente debemos crear un proyecto, ya que no se puede crear un archivo sin un proyecto.



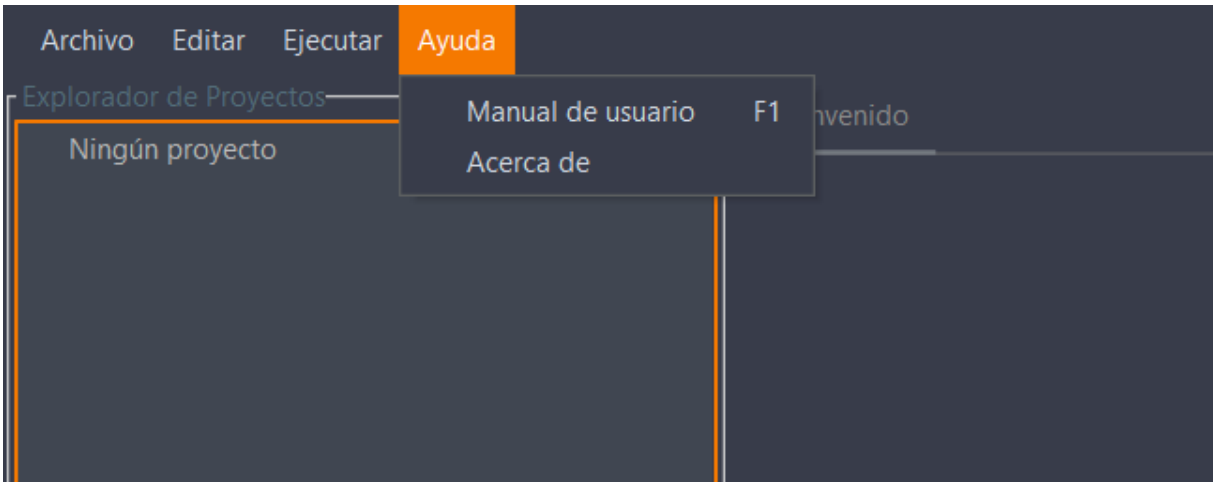
Esta ventana es para la opción de "Nuevo Archivo", diseñado para crear y guardar archivos de código fuente con la extensión propia del lenguaje (.rvn).



La sección de Editar contiene las opciones de Deshacer, Rehacer, Cortar, Copiar y Pegar, que permiten realizar acciones básicas de edición. Así mismo, se muestran los atajos de teclado correspondientes a cada una de las acciones de edición.

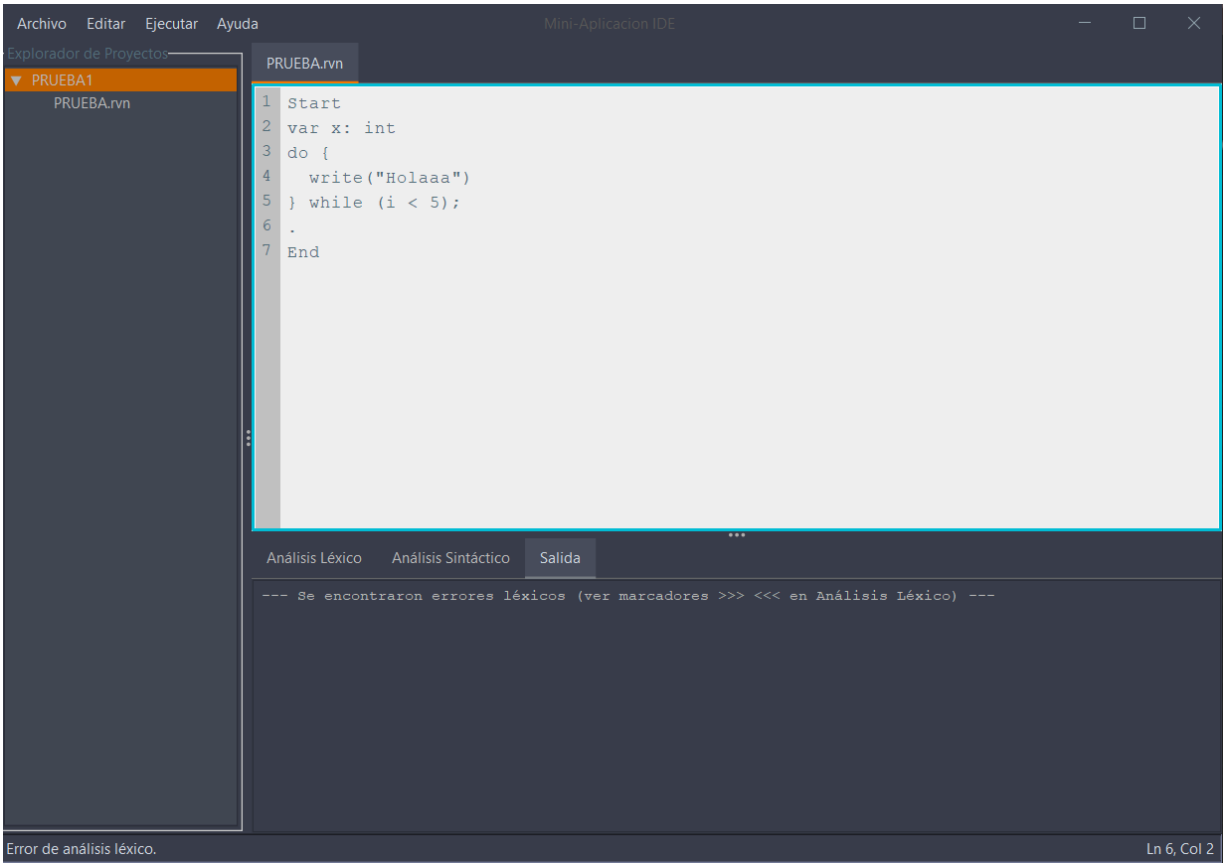


En la sección de Ejecutar solo contiene la opción de Compilar (F5), esta opción permite que se ejecute el programa para mostrar la salida, el análisis léxico y sintáctico (donde se mostrarán los errores del código).

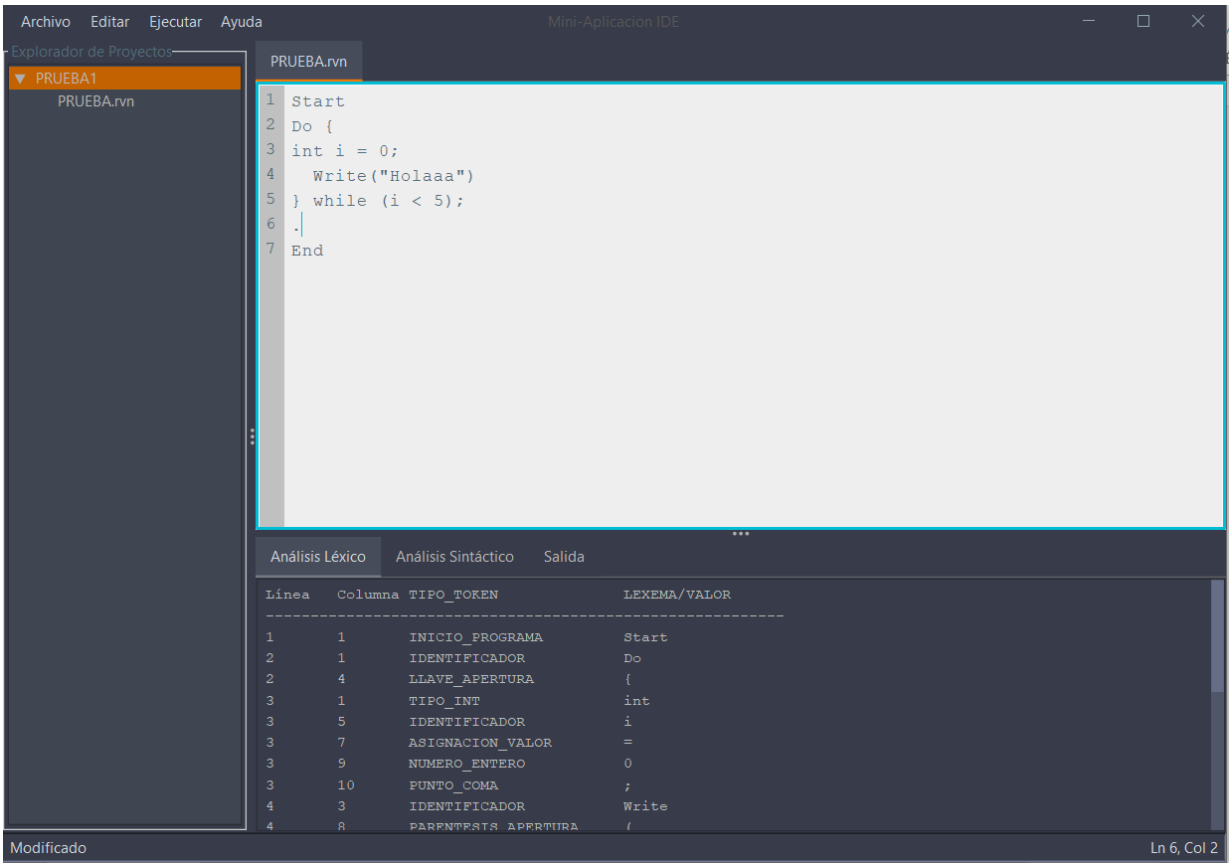


En la sección de Ayuda están las opciones de “Manual de usuario” y “Acerca de” (donde se encontrarán datos del programa).

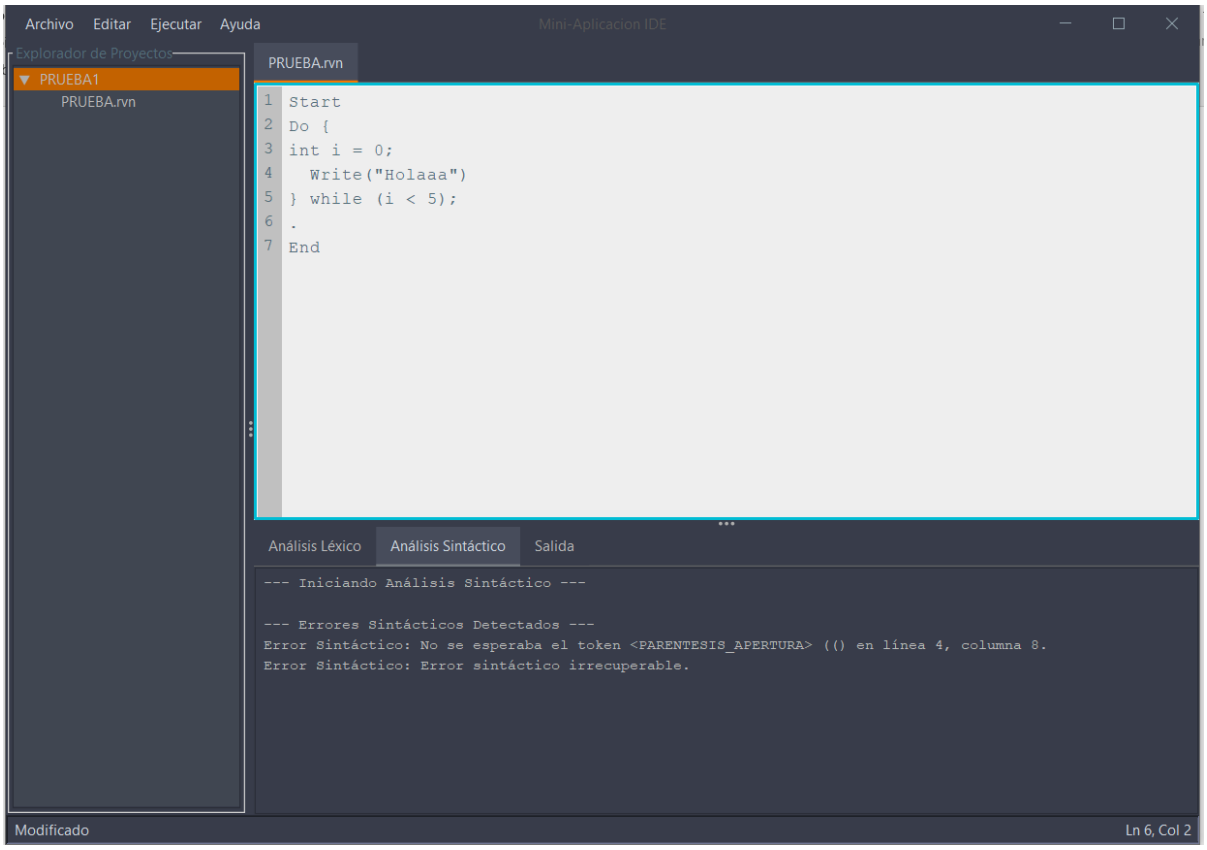
Salida



Al momento de compilar el programa en la salida se mostrará un mensaje en caso de que marque errores, y si no los hay no mostrará nada.



La sección de "Análisis Léxico" muestra una tabla detallada con información sobre los tokens y valores léxicos que se encontraron en el código, incluyendo el tipo de token (como "INICIO_PROGRAMA", "IDENTIFICADOR", "DOS_PUNTOS", etc.) y el valor del token (como "Start", "int", "write", etc.).



La sección de "Análisis Sintáctico" muestra los resultados del análisis de la estructura del código fuente. Específicamente, esta sección muestra los "Errores Sintácticos" que el analizador ha detectado en el código. Estos errores indican problemas en la forma en que se han escrito las instrucciones.

Por ejemplo, en la imagen se indica un "Error Sintáctico: Error sintáctico irrecuperable, cerca de línea 2, columna 5". Esto significa que el analizador encontró un error en la sintaxis del código, ubicado aproximadamente en la línea 2, columna 5.

Sintaxis del lenguaje

- **Palabras reservadas**

Categoría	Palabras Reservadas	Descripción
Inicio/Fin	start, end	Indican el comienzo y fin de un bloque de código o programa.
Modificadores de acceso	public, private, protected, static	Controlan la visibilidad y el comportamiento de clases, métodos o variables.
Funciones	fun, return	**fun** declara una función y **return** se usa para devolver un valor.
Estructuras de control selectivas	if, else, elif, when	Permiten tomar decisiones según condiciones. elif es "else if".
Estructuras de control repetitivas	for, while, do	Ejecutan bloques de código repetidamente mientras se cumpla una condición.
Tipos de datos	String, char, int, float, double, boolean	Definen el tipo de valor que puede almacenar una variable.
Entrada/Salida de datos	write, writeln, read	Se usan para mostrar información en pantalla (write/writeln) o leerla (read).
Booleanos literales	true, false	Representan los valores lógicos verdadero o falso.

- **Operadores: relacionales, lógicos y aritméticos**

Categoría	**Operador**	**Descripción**
Relacionales	“>”, “<”, “==”, “!=”, “>=”, “<=”	Comparan dos valores y devuelven un valor booleano (true o false).
Lógicos	“&&”, “||”, “!”, “&”, “|”	**&&:** Operador AND lógico; devuelve true si ambas condiciones son verdaderas. **||:** Operador OR lógico; devuelve true si al menos una condición es verdadera. **!:** Operador NOT; invierte el valor lógico (de true a false y viceversa). **& y |:** Operadores bit a bit; comparan bits individuales de números enteros.
Aritméticos	“+”, “-”, “*”, “/”, “%”	Realizan operaciones matemáticas básicas.
Incremento/Decremento	“++”, “--”	Aumentan o disminuyen en 1 el valor de una variable.
Atribución	"+=", "-=", "*=", "/=", "%="	Asignan un valor a una variable realizando una operación previa.
Símbolos especiales	“()”, “[]”, “{}”, “#”, “$“, “:”, “;”, “,”, “=”	Se usan en estructuras del lenguaje como listas, bloques, parámetros, etc.

- **Estructuras de control selectivas y repetitivas**

Selectivas: if-else, if-else if (elif), when.
Estas estructuras permiten tomar decisiones basadas en condiciones.

Estructuras	Descripción	Ejemplos

if-else	Evalúa una condición booleana.
	Si es verdadera, se ejecuta un bloque de código; de lo contrario, se ejecuta otro.
		if (edad >= 18) {
		writeln(“Es mayor de edad");
		} else {
		writeln("Es menor de edad");
		}

if-else if (anidados)	Permite evaluar múltiples condiciones secuenciales.	if (nota >= 90) {
		writetln("Excelente");
		} else if (nota >= 70) {
		writeln("Aprobado");
		} else {
		writeln("Reprobado");
		}

when	Permite comparar el valor de una variable con varios casos (case). Cuando se encuentra una coincidencia, se ejecuta el bloque de código asociado. Si no hay coincidencia, se puede usar un default como alternativa por defecto.	sueldo = read(Int);
		when {
		sueldo > 5000 -> {
		writeln("Sueldo alto");
		}
		sueldo > 2000 -> {
		writeln("Sueldo medio");
		}
		else -> {
		writeln("Sueldo bajo");
		}

Repetitivas: for, while, do-while.
Estas estructuras permiten ejecutar un bloque de código múltiples veces.

Estructuras	Descripción	Ejemplos

for	Se usa cuando se conoce de antemano cuántas veces debe repetirse el bloque.	for (int=0; x < 5; x++) {
		writeln("Hola Mundo" $ x);
		}

while	Se ejecuta mientras una condición booleana sea verdadera. Ideal cuando no se sabe cuántas veces se repetirá.	int=0;
		
		while (i < 5) {
		
		write("Hola Mundo");
		
		}

do-while	Similar a while, pero garantiza al menos una ejecución del bloque, ya que la condición se evalúa al final.	Int=0;
		
		do {
		write("Hola Mundo");
		
		} while (i < 5);

- **Entrada-Salida de datos**

1. **write – Imprimir**
2. Imprime un mensaje en pantalla sin salto de línea al final.
3. **writeln - Imprimir con salto de línea.**
4. Imprime un mensaje y agrega un salto de línea automáticamente al final.
5. **write("texto")– Imprimir mensaje específico**
6. Imprime el contenido del texto entre comillas. Puede ser con print o println según si se desea salto de línea o no.
7. **read - Leer**
8. Indica que el programa debe esperar una entrada del usuario. La sintaxis es la siguiente: read(TipoDeDato); Entre paréntesis se debe especificar el tipo de dato que se quiere leer para asignar a la variable, la cual debe ser del mismo tipo.
9. **var x: int; – Declarar una variable**
10. Se declara una variable llamada x de tipo entero.
11. **read x; – Leer y guardar valor en una variable**
12. Captura el valor introducido por el usuario y lo guarda en la variable x.

- **Arreglos**

Un arreglo (array) es una estructura de datos que almacena múltiples valores del mismo tipo en una sola variable, accesibles por su índice (posición).

Lo definimos de la siguiente manera:

```
var numeros : int [5];
var numeros : int [2] = {0, 1};
```

- **Funciones y procedimientos**

```
fun imprimir(){
write(“Hola mundo”);
}

fun imprimirNombre(nombre: String){
write(“Hola $nombre”);
}

fun imprimir(): String{
return;
}
```

Tabla de tokens + (símbolos)

Tipo de token (P.R)	Lexema
INICIO_PROGRAMA	Start
FIN_PROGRAMA	End
IF	if
ELSE	else
DO	do
WHILE	while
FOR	for
PUBLICO	public
PRIVADO	private
PROTEGIDO	protected
ESTATICO	static
FUNCION	fun
ELIF	elif
RETORNO	return
WHEN	when
TIPO_STRING	String
TIPO_CHAR	char
TIPO_INT	int
TIPO_FLOAT	float
TIPO_DOUBLE	double
TIPO_BOOLEAN	boolean
TRUE	true
FALSE	false

IMPRIMIR	write
IMPRIMIR_CON_SALTO	writeln
ENTRADA	read
NUMERO_ENTERO	1,2,3,4,5,6,7,8,9…
IDENTIFICADOR	i,x…
ERROR	., ´, ?,¡,¿...

Tipo de token (operadores)	Lexema
OPERADOR_INCREMENTO	++
OPERADOR_DECREMENTO	--
SUMA_Y_ASIGNACION	+=
RESTA_Y_ASIGNACION	-=
MULTIPLICACION_Y_ASIGNACION	*=
DIVISION_Y_ASIGNACION	/=
MODULO_Y_ASIGNACION	%=
IGUAL_A	==
DIFERENTE_DE	!=
MAYOR_IGUAL	>=
MENOR_IGUAL	<=
MAYOR_QUE	>
MENOR_QUE	<
AND	&&
OR	||
NEGACION	!
SUMA	+
RESTA	-
MULTIPLICACION	*
DIVISION	/
MODULO	%
ASIGNACION_VALOR	=

Tipo de token (S.E)	Lexema
PARENTESIS_APERTURA	(
PARENTESIS_CIERRE	)
CORCHETE_APERTURA	[
CORCHETE_CIERRE	]
LLAVE_APERTURA	{
LLAVE_CIERRE	}
SIGNO_DOLAR	$
COMENTARIO	#
DOS_PUNTOS	:
PUNTO_COMA	;