



# ERDÖS – MANUAL DE ROSARIO

## MANUAL DE USUARIO

Version Del Manual 1.0

Erdös es una aplicación desarrollada en Python que permite diseñar y ejecutar diagramas de Nassi-Schneiderman (Chapin)

Autores: Forconi; Rivera; Virga

## ERDÖS - Manual de Usuario

## Contents

1.	Introducción.....	2
2.	Requisitos del Sistema.....	2
2.1	Instalación.....	2
2.1.1	Windows .....	2
2.1.2	Linux .....	3
3.	Entorno de Trabajo.....	4
3.1	Vista General.....	4
3.2	Barra de Menús .....	5
3.2.1	Menú Archivo .....	5
3.2.2	Menú Edición.....	5
3.2.3	Menú Herramientas .....	6
3.2.4	Menú Ayuda .....	6
3.3	Barra de Figuras .....	7
3.3.1	Figura Sentencia .....	7
3.3.2	Figura "Entrada".....	8
3.3.3	Figura "Salida".....	10
3.3.4	Figura "Si" (IF Condicional).....	11
3.3.5	Figura "Para" (For).....	16
3.3.6	Figura "Mientras" (While).....	18
3.3.7	Figura "Hasta" (Until Repeat) .....	19
3.3.8	Figura "Caso" (Case).....	21
3.4	Sintaxis del Lenguaje.....	23
3.4.1	Tipos de Datos.....	23
3.4.2	Operadores .....	24
3.5	Ejecución del Diagrama.....	24
3.6	Manejo de Archivos .....	26
3.6.1	Abrir Diagrama.....	26
3.6.2	Guardar Diagrama.....	27
3.7	Manejo de Errores .....	27
3.7.1	Error de Sintaxis .....	27
3.7.2	Error de Tipos .....	28
3.7.3	Error de Nombres.....	28
3.7.4	Error de Índices .....	30
3.7.5	Error de División por Cero .....	30

## 1. Introducción

Erdös es una aplicación desarrollada en Python que permite diseñar y ejecutar diagramas de Nassi-Schneiderman, a través de un entorno de trabajo sencillo e intuitivo. En programación de computadoras los diagramas de Nassi-Schneiderman (popularmente conocidos como "Diagramas de Chapin") son una representación gráfica que muestra el diseño de un programa estructurado.

El objetivo de Erdös es facilitar a la persona iniciada en programación la construcción de dichos diagramas y poder ejecutar el algoritmo descrito por ellos.

## 2. Requisitos del Sistema

Erdös corre en cualquier PC (desktop, notebook, netbook) que posea las siguientes características:

- Procesador: Intel Pentium IV, AMD Phenom a 2 Ghz.
- Memoria: 512 GB de RAM.
- Espacio en Disco: Se requieren 80 MB libres para la instalación.
- Sistema Operativo: Windows Xp, Linux 2.6.28, Mac OS X 10.1 o superior.

### 2.1 Instalación

#### 2.1.1 Windows

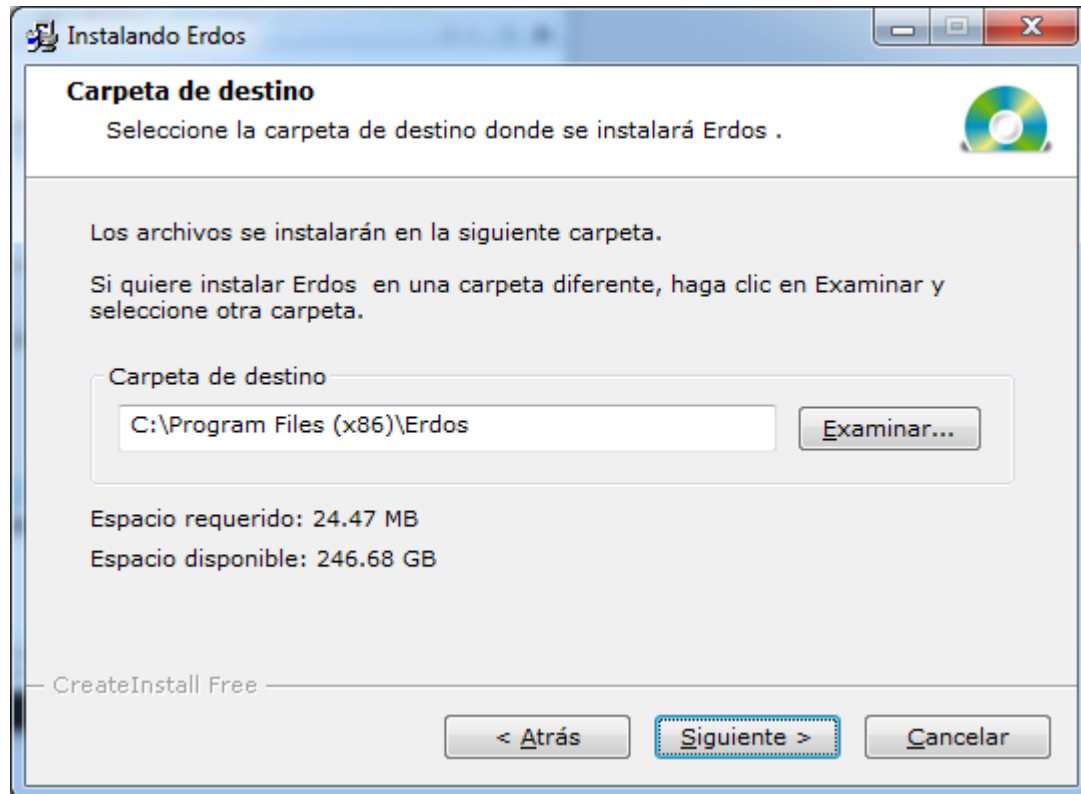
En un entorno Windows, se cuenta con un instalador (setup.exe) que se encarga de este proceso. Se realiza en tres sencillos pasos.

Pasos para la instalación:

1. Ejecutar el setup del programa.



2. Elegir la ruta deseada para la instalación.



3. Una vez finalizada, se nos generara un acceso directo en el escritorio. El programa ya está instalado y se puede ejecutar.



### 2.1.2 Linux

Todas las distribuciones GNU/Linux actuales cuentan con Python 2.7 o mayor. Solo se requiere instalar las librerías necesarias y correr el script desde los archivos fuentes.

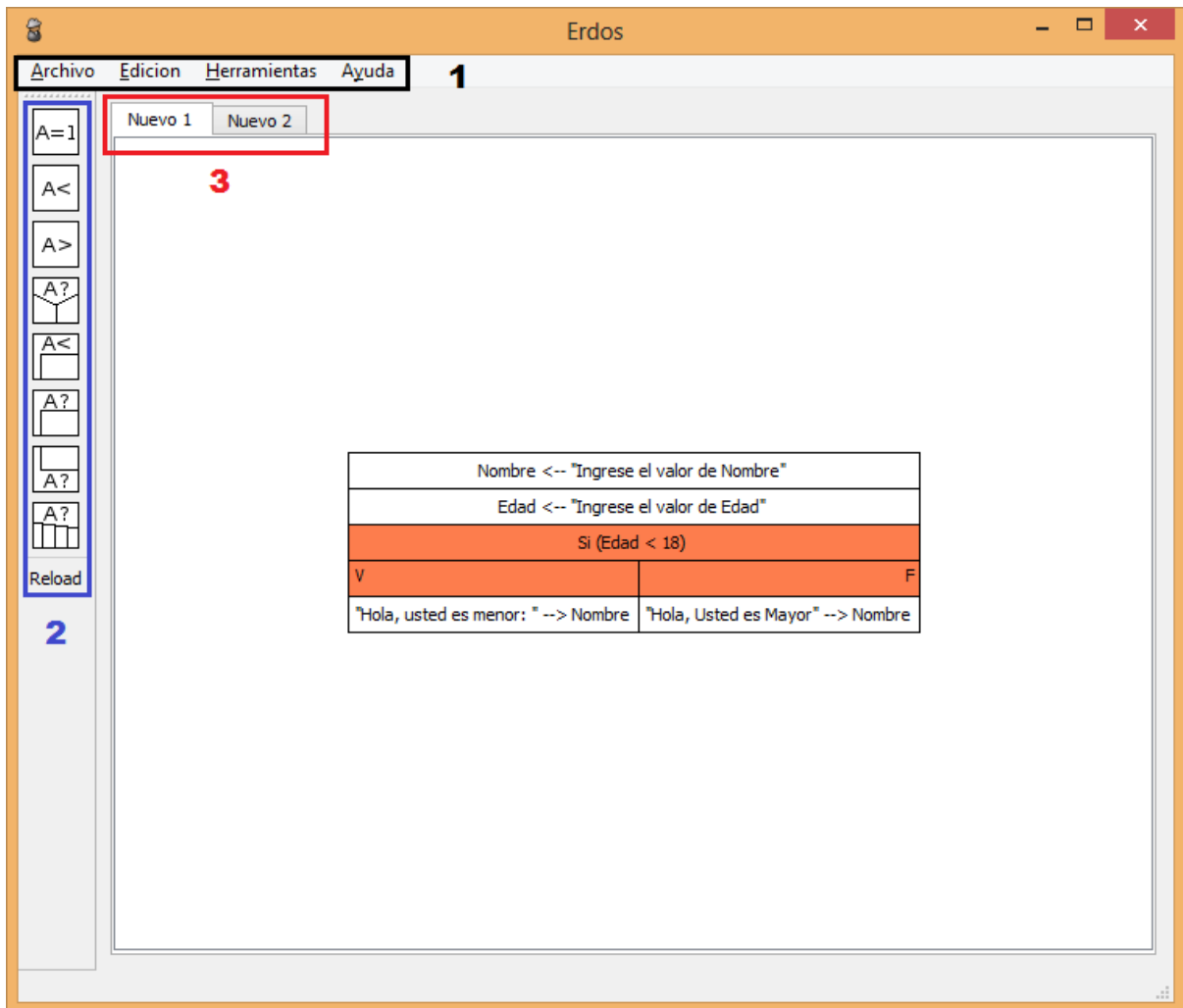
#### Pasos para la instalación:

1. Instalar las librerías *ipython* y *pyqt* con su gestor de paquetes favorito.
2. Descomprimir los archivos fuentes en un directorio.

3. Lanzar *python erdos.py* en una consola cada vez que se lo desee ejecutar.

## 3. Entorno de Trabajo

### 3.1 Vista General



(Fig. 1) vista del entorno de trabajo Erdös.

El entorno de trabajo de Erdös está compuesto por los siguientes elementos:

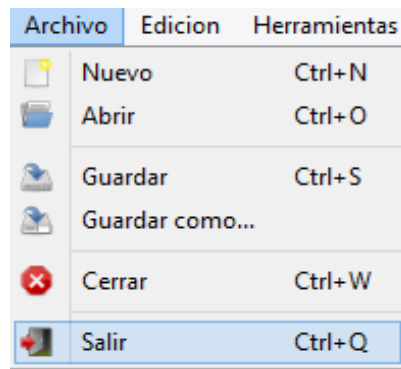
1. **Barra de Menús:** permite acceder a los menús contextuales que contienen las diversas opciones que ofrece la aplicación.
2. **Barra de Figuras:** contiene todas las figuras que pueden utilizarse para diseñar el diagrama de Chapín.

3. **Pestañas de Trabajo:** las pestañas alojan el espacio de trabajo donde pueden diseñarse varios diagramas de chapín al mismo tiempo.

## 3.2 Barra de Menús

La barra de menús ofrece acceso a las opciones del programa. Los menús que contiene son:

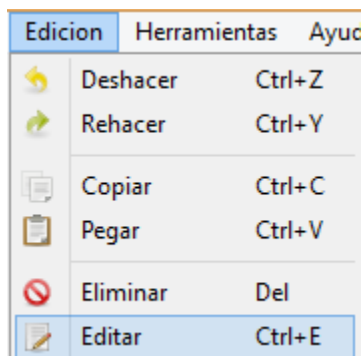
### 3.2.1 Menú Archivo



(Fig. 2) vista del Menú Archivo.

Este menú contiene las opciones relacionadas al manejo de archivos, permite crear una nueva pestaña de trabajo donde dibujar, abrir diagramas que hayamos creado anteriormente en Erdös (y que estén guardados en Disco). Guardar el o los diagramas que estemos diseñando y por ultimo Salir de la aplicación.

### 3.2.2 Menú Edición

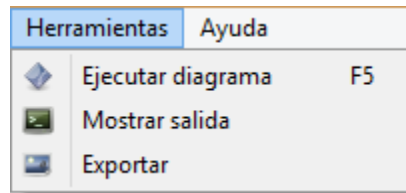


(Fig. 3) vista del Menú Edición.

El menú edición permite acceder a comandos útiles para facilitar el diseño de nuestros diagramas. Desde aquí podemos "Deshacer" o "Rehacer" las acciones realizadas en el

orden que fueron ejecutadas. Los comandos "Copiar" y "Pegar" permiten la copia de figuras en el diagrama, la opción "Eliminar" lógicamente elimina la figura seleccionada y por ultimo "Editar" permite cambiar los parámetros de la figura seleccionada.

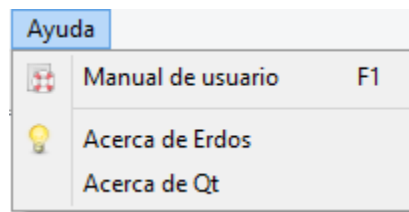
### 3.2.3 Menú Herramientas



(Fig. 4) vista del Menú Herramientas.

El menú Herramientas contiene los comandos de Ejecutar Diagrama, Mostrar Salida y Exportar los cuales se ampliará la información más adelante.

### 3.2.4 Menú Ayuda

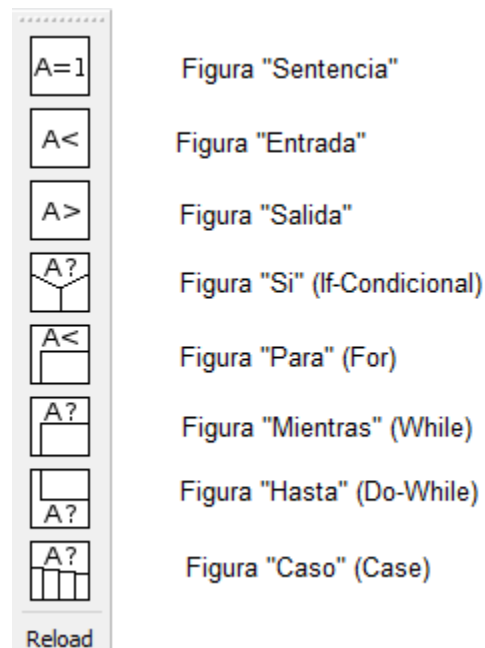


(Fig. 5) vista del Menú Ayuda.

Este menú contiene acceso al Manual de Usuario así como información de los creadores de la aplicación y de las licencias utilizadas en la construcción de Erdös.

### 3.3 Barra de Figuras

Las figuras son el elemento esencial de los diagramas de Chapín, cada una de ellas representan elementos de control que definen la característica del algoritmo que estamos diseñando. Erdős permite insertar, editar y eliminar elementos de control en un diagrama de Chapín a través de la barra de Figuras.



(Fig. 6) vista de la barra de Figuras

#### 3.3.1 Figura Sentencia

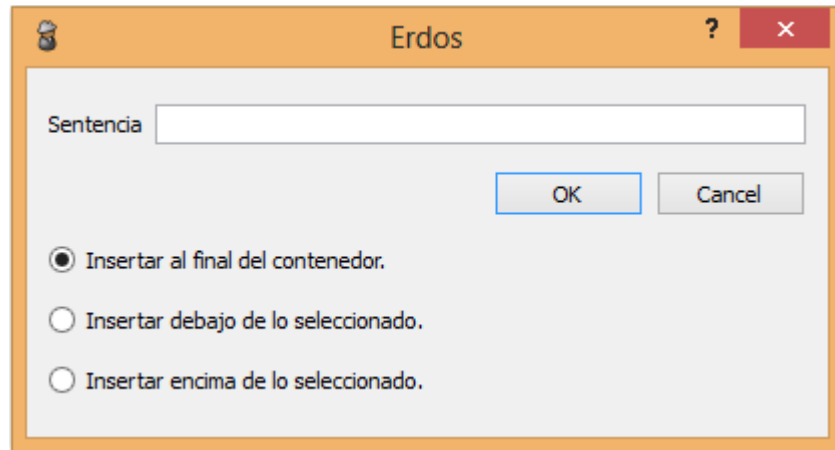
Esta figura permite insertar sentencias de asignación y operación. Las sentencias de asignación permiten dar valor a una variable a través del operador de asignación "igual" (=). Estas sentencias pueden construirse también utilizando la figura "Entrada" que veremos más adelante.

##### Operadores

Las sentencias de operación permiten realizar un cálculo matemático o lógico utilizando una serie de operadores. Para consultar la lista de operadores válidos ir a la sección "3.4 - Sintaxis del Lenguaje".



A continuación se observan los parámetros de configuración de la figura "Sentencia":



(Fig. 7) vista de la ventana de parámetros de la figura Sentencia.

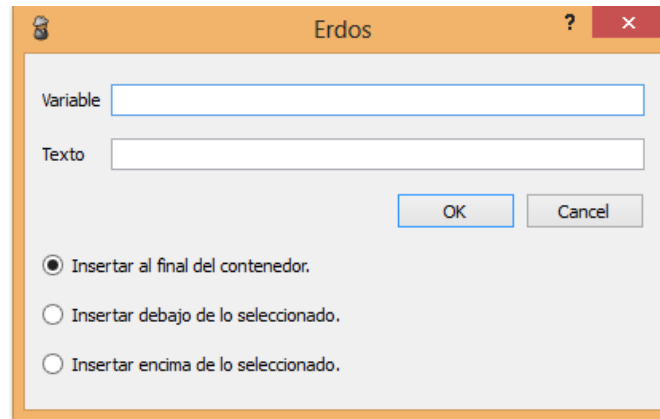
Como podemos observar el campo de texto "Sentencia" permite el ingreso de la sentencia en cuestión. Debajo de dicho campo, encontramos las opciones de posición de la figura, las cuales ampliaremos la información más adelante.

### 3.3.2 Figura "Entrada"

La figura entrada actúa como una sentencia de asignación a una variable en particular, permite declarar la misma y mostrar un mensaje de manera que el usuario ingrese el valor de la variable en tiempo de ejecución en una ventana que mostrará el mensaje especificado. Erdős automáticamente define el tipo de variable de acuerdo con el valor que le asignemos a partir de la figura entrada en tiempo de ejecución.

Ejemplo: Edad = 25. Asigna a la variable edad el valor 25 (entero).

Nombre = "Paul". Asigna a la variable Nombre el valor "Paul" (string).



(Fig. 8) vista de la ventana de parámetros de la figura Entrada.

## Parámetros

La figura cuenta con dos parámetros: variable (donde se especifica el nombre de la variable) y texto que permite definir el texto que se mostrará por pantalla en tiempo de ejecución, previo a la asignación de datos de la variable.

Ejemplo: Insertamos la figura entrada con los siguientes parámetros:

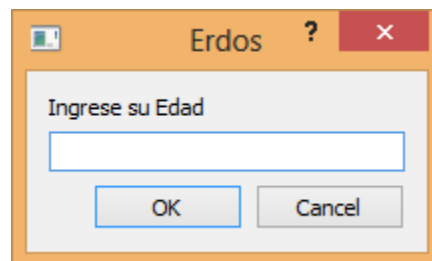
Variable: Edad; Texto: "Ingrese su Edad"

En el entorno de trabajo la figura se observa de la siguiente forma:

Edad <-- "Ingrese su Edad"

(Fig. 9) Figura Entrada desde el Entorno de Trabajo.

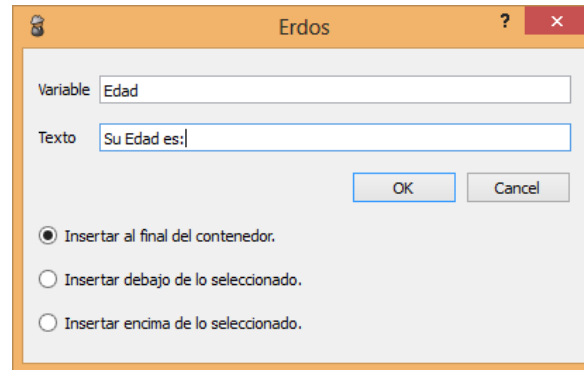
Mientras que en tiempo de ejecución se observa la ventana de ingreso de datos:



(Fig. 10) Figura Entrada en tiempo de Ejecución.

### 3.3.3 Figura “Salida”

La figura salida permite mostrar el valor de una variable previamente declarada en el entorno de trabajo y mostrarla en la salida de Ejecución.



(Fig. 11) vista de la ventana de parámetros de la figura Salida.

#### Parámetros

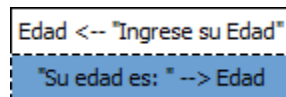
Su utilización es similar a la de figura entrada, podemos definir que variable es la que utilizaremos para mostrar en la salida y definir un mensaje que acompañe al valor de variable que se mostrará en la salida.

#### Ejemplo

Insertamos la figura salida con los siguientes parámetros:

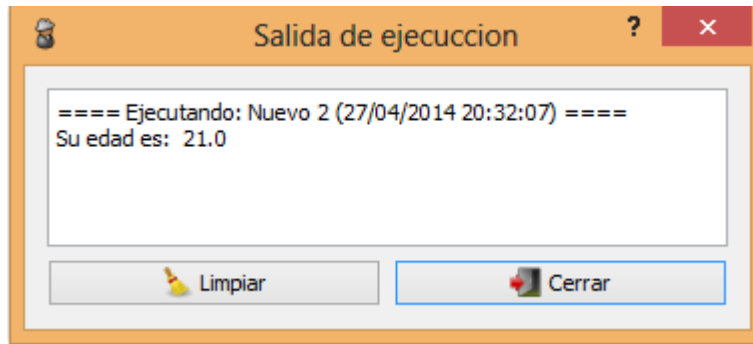
Variable: Edad; Texto: “Su Edad es:”

En el entorno de trabajo la figura se observa de la siguiente forma:



(Fig. 12) Figura Salida desde el Entorno de Trabajo (coloreada en azul).

En tiempo de ejecución se observará el valor de la variable en Salida de Ejecución:



(Fig. 13) Salida generada en tiempo de Ejecución.

### 3.3.4 Figura “Si” (IF Condicional)

Las estructuras de control de un programa son la base de la programación estructurada, permiten modificar el flujo de ejecución de un programa de acuerdo a ciertas condiciones.

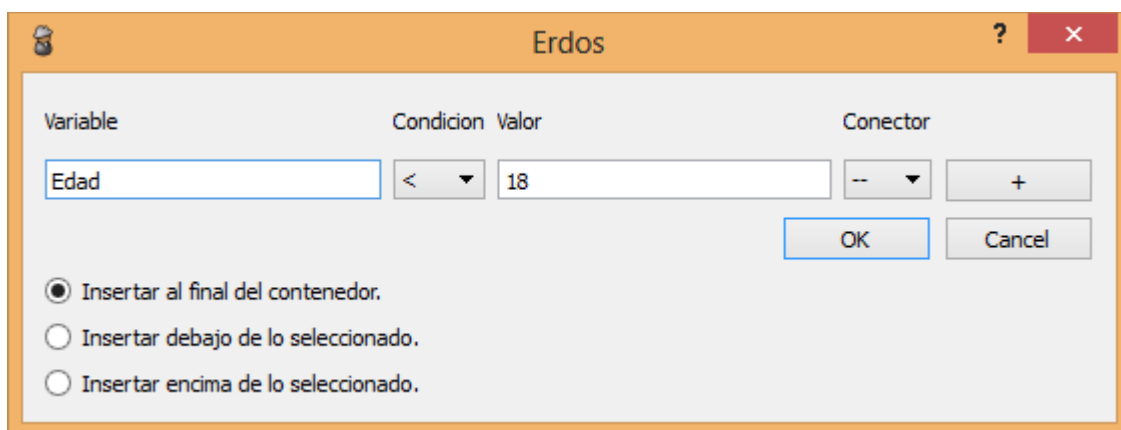
La estructura de control más sencilla es la condicional (también conocida como “IF” o en castellano “SI”). Se trata de una estructura de control que permite redirigir un curso de acción según la evaluación de una condición simple, sea verdadera o falsa.

Si (Condicion == )	
V	F
SENTENCIA 1	SENTENCIA 2

(Fig. 14) Vista de la figura “SI” en el Entorno de Trabajo.

En el ejemplo anterior si la sentencia que se evalúa (condición) es verdadera, se ejecuta el bloque de sentencias 1, sino, se ejecuta el bloque de sentencias 2.

#### Parámetros



(Fig. 15) vista de la ventana de parámetros de la figura Sí.

La figura Si cuenta con cuatro parámetros: nombre de variable, condición, valor y conector. La condición permite especificar el operador con el que se evaluará la variable, los operadores de comparación válidos son:

<	Menor	5 < 3	FALSE
>	Mayor	5 > 3	TRUE
<=	Menor o igual	8 <= 6	FALSE
>=	Mayor o igual	7 >= 7	TRUE
==	Igual	9 == 9	TRUE
!=	Distinto	1 != 0	TRUE

### Ejemplo 1

Insertamos la figura salida con los siguientes parámetros:

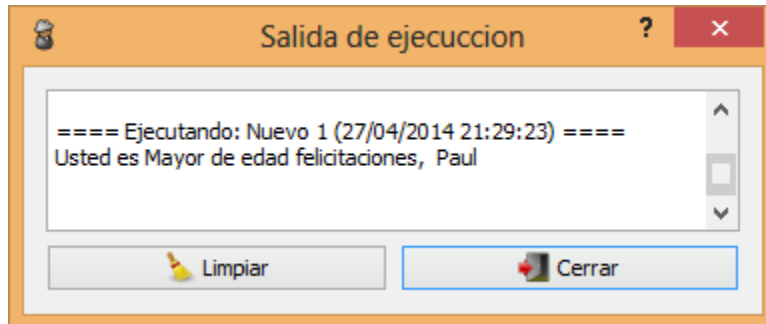
Variable: Edad; Condición: <; Texto: Valor: 18; Conector: Ninguno.

En el entorno de trabajo la figura se observa de la siguiente forma:

Nombre <-- "Ingrese su nombre: "	
Edad <-- "Ingese su Edad"	
Si (Edad < 18)	
V	F
"Usted es menor de edad, " --> Nombre	"Usted es Mayor de edad felicitaciones, " --> Nombre

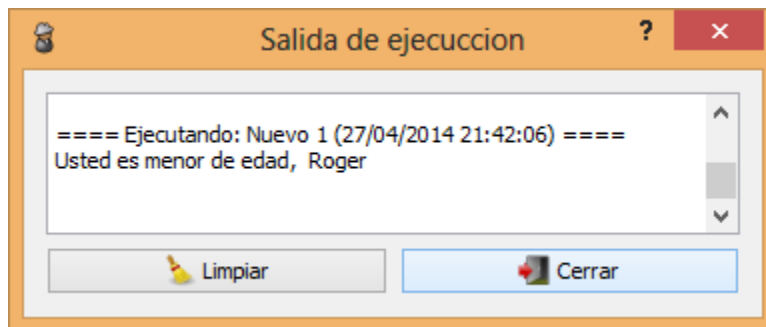
(Fig. 16) Un ejemplo del uso de la Figura Sí.

Si ejecutamos el programa de acuerdo al valor que le demos a la variable “Edad” observaremos un determinado mensaje de salida. En el ejemplo de abajo ingresamos un valor de Edad = 33 y el mensaje que nos muestra es el siguiente:



(Fig. 17) Ventana de Salida de Ejecución.

Si ejecutamos nuevamente el programa esta vez dándole un valor menor a 18 a la variable Edad, el mensaje que nos mostrará la Salida de Ejecución es el siguiente:



(Fig. 18) Ventana de Salida de Ejecución.

Por lo tanto de acuerdo al resultado de la evaluación de la ejecución se ejecutará una u otra sentencia.

El conector sirve para armar sentencias compuestas que poseen más de una condición asociada, por ejemplo:

- Si (edad  $\geq$  18) Y (edad  $\leq$  65)
- Si (Nombre == "Jagger") O (Nombre == "Richards")

El conector define de qué manera van a evaluarse las sentencias compuestas, los valores válidos que puede tomar el conector son: AND, OR.

Ejemplo 2

En el sig. ejemplo observamos un simple programa que determina si estamos en edad laboral de acuerdo al valor ingresado (Se considera a toda persona en "edad laboral" si su edad alcanza o supera los 18 años y es inferior o igual a 65 años).

Variable	Condicion	Valor	Conector
Edad	>	18	and
Edad	<=	65	--

☒ Insertar al final del contenedor.  
☐ Insertar debajo de lo seleccionado.  
☐ Insertar encima de lo seleccionado.

El conector utilizado es "and" lo cual significa que Erdos analizará por separado las dos cláusulas (Edad  $\geq$  18); (Edad  $\leq$  65) y mostrará en pantalla el mensaje "Usted está en Edad Laboral" únicamente si ambas cláusulas son verdaderas, si alguna de las dos es falsase mostrará el mensaje: "Usted NO debe Trabajar".

Nombre <-- "Ingrese su nombre: "	
Edad <-- "Ingrese su Edad"	
Si (Edad $\geq$ 18 and Edad $\leq$ 65)	
V	F
"Su edad es: " --> Edad	"Su edad es: " --> Edad
"Usted está en Edad Laboral: " --> Nombre	"Usted NO debe Trabajar: " --> Nombre

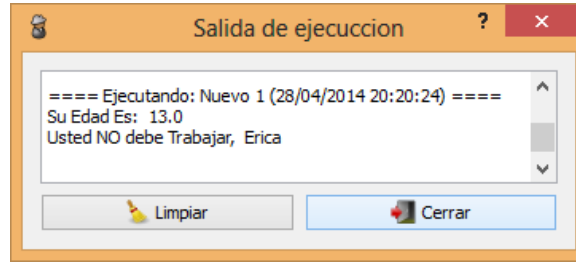
Ejecutamos el programa y cargamos como Edad = 35, el programa muestra:

==== Ejecutando: Nuevo 1 (28/04/2014 20:17:59) ====

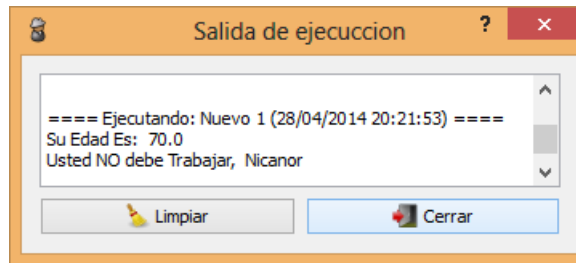
Su Edad Es: 35.0

Usted esta en Edad Laboral: Pablo

A continuación volvemos a ejecutar el programa y cargamos como Edad = 13::



Volvemos a ejecutar el programa y esta vez cargamos como Edad = 70:



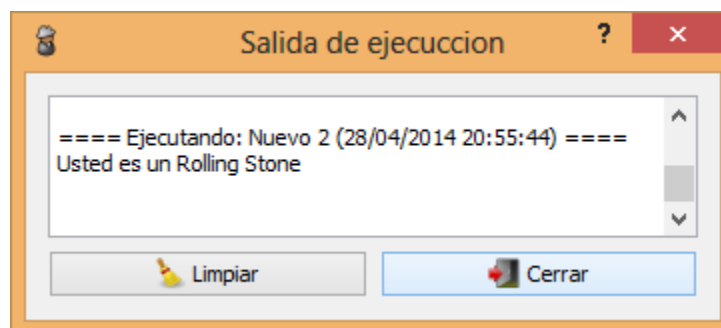
Por lo tanto se observa claramente que ante la presencia de un conector "and" se ejecutará la sentencia verdadera si y solo si ambas cláusulas condicionales se cumplen.

### Ejemplo 3

Utilizando un conector de tipo "OR" se analizaran por separado ambas clausulas condicionales y se ejecutará la sentencia verdadera **si por lo menos una de ambas clausulas es verdadera.**

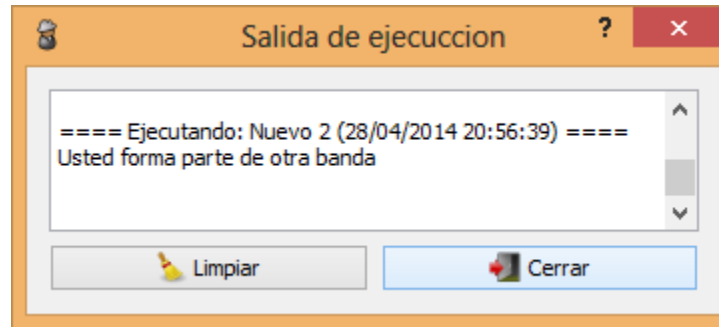
Nombre <-- "Ingrese el nombre del integrante de la banda: "	
Si (Nombre == 'Jagger' or Nombre == 'Richards')	
V	F
"Usted es un Rolling Stone" -->	"Usted forma parte de otra banda" -->

Ejecutamos el programa y cargamos el valor de 'Jagger' en nombre:





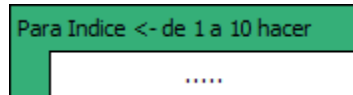
Ejecutamos nuevamente y esta vez cargamos el valor de 'Ringo' en nombre:



### 3.3.5 Figura “Para” (For)

La figura “Para” (también denominada “FOR”) es una estructura de control que permite iteraciones de una o varias sentencias en el flujo del programa. Popularmente se denomina a la iteración “loop” o “bucle”.

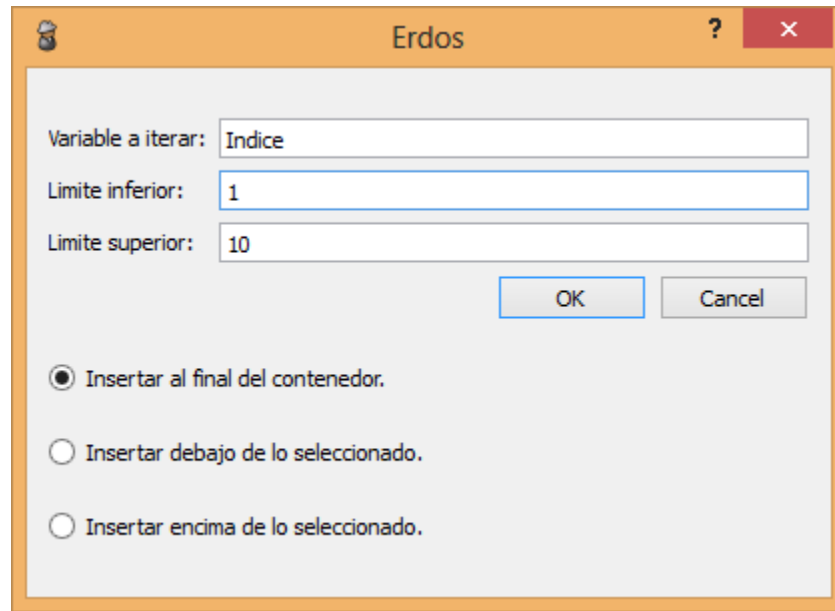
La figura Para permite iterar de acuerdo a límites establecidos sobre un índice (inferior y superior), el programa comienza iterando en el límite inferior y se detiene cuando el índice alcanza el límite superior.



(Fig. 19) Figura Para vista desde el Entorno de Trabajo.

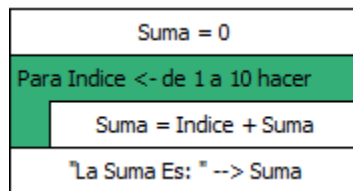
#### Parámetros

La figura cuenta con tres parámetros: variable a iterar: (donde se especifica el nombre de la variable o índice sobre el cual se iterará), y los valores del límite inferior y límite superior respectivamente.



### Ejemplo

Un sencillo ejemplo de uso consiste en realizar un programa que realice la suma de los números del 1 al 10, es decir:  $1 + 2 + 3 + \dots + 10$ , cuyo resultado es: 55.



(Fig. 20) Ejemplo sencillo de iteración usando "Para".

La primer figura es una figura sentencia que inicializa el valor de la variable  $\text{Suma} = 0$ . Luego se ejecuta la primer iteración mediante la figura Para, que suma el valor actual del índice a la variable Suma de la siguiente forma:

$$\text{Suma} = \text{Indice} + \text{Suma}$$

$$\text{Suma} = 1 + 0 = 1$$

En la segunda iteración, el valor del índice cambia a 2, por ende se ejecuta la sentencia  $\text{Suma} = \text{Índice} + \text{Suma}$  con nuevos valores:

$$\text{Suma} = 2 + 1 = 3$$

Siguiendo con esta lógica en la tercera iteración se actualiza el valor del índice a 3 y la sentencia se ejecuta con estos valores:

$$\text{Suma} = 3 + 3 = 6$$

El proceso continúa de la siguiente forma:

Iteración 4: Suma = 4 + 6 = 10

Iteración 5: Suma = 5 + 10 = 15

Iteración 6: Suma = 6 + 15 = 21

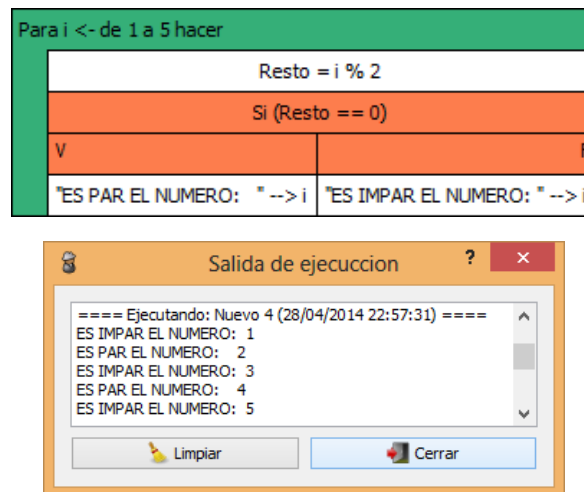
Iteración 7: Suma = 7 + 21 = 28

Iteración 8: Suma = 8 + 28 = 36

Iteración 9: Suma = 9 + 36 = 45

Iteración 10 (final): Suma = 10 + 45 = 55

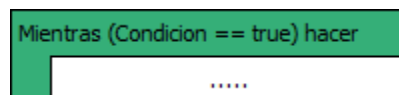
Más allá de este ejemplo la estructura “Para” admite la iteración de múltiples sentencias como se observa en el ejemplo siguiente:



### 3.3.6 Figura “Mientras” (While)

La figura “Mientras” (también denominada “WHILE”) es una estructura de control que al igual que la figura “Para” permite iteraciones de una o varias sentencias en el flujo del programa. La diferencia principal que esta figura posee con “para” es que la primera permite iterar con índice restringido por un límite inferior y superior, lo que condiciona la iteración a un determinado número de casos. En su lugar “Mientras” permite iterar un número de sentencias mientras se cumpla con una determinada condición previamente fijada.

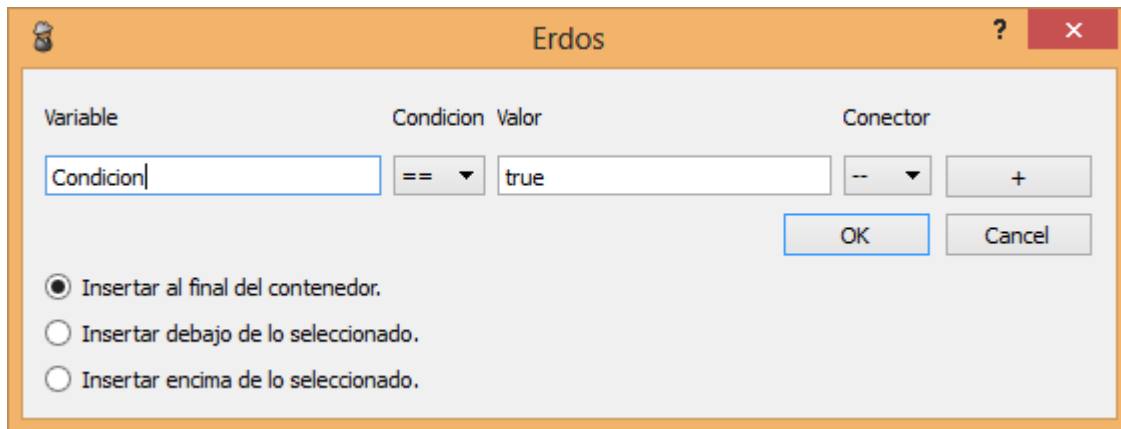
Mientras la condición sea verdadera se iteraran las sentencias, cuando finalmente la condición no se cumpla (sea falsa) se romperá con el bucle y se continuará ejecutando el resto de las sentencias del programa fuera del bucle. Cabe aclarar que si la condición no se cumple de entrada entonces no se ejecutan las sentencias dentro del bucle.



(Fig. 21) Figura Mientras vista desde el Entorno de Trabajo.

### Parámetros

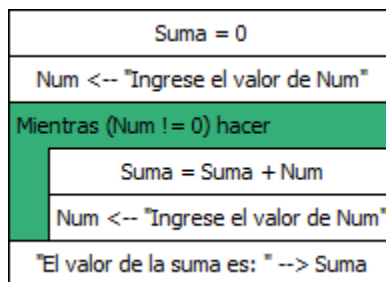
La figura cuenta con cuatro parámetros: variable: (donde se especifica el nombre de la variable sobre la cual se ejecutará la condición), condición: operador de comparación con el que se comparará el valor de la variable; valor: con el cual se comparará la variable y conector que permite armar una sentencia condicional compuesta.



(Fig. 22) ventana de Parámetros de la figura "Mientras".

### Ejemplo

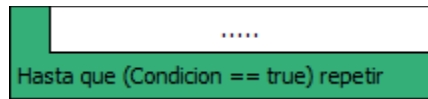
El siguiente ejemplo utiliza la estructura de control "mientras" en un programa que suma los valores ingresados hasta que ingresemos un cero. Eso hace que se detenga la iteración contenida dentro del Where.



#### 3.3.7 Figura "Hasta" (Until Repeat)

La figura "Hasta" (también denominada "UNTIL REPEAT") es una estructura de control que al igual que la figuras "Para" o "Mientras" permite iteraciones de una o varias sentencias en el flujo del programa. La figura "Hasta" **itera una serie de sentencias hasta**

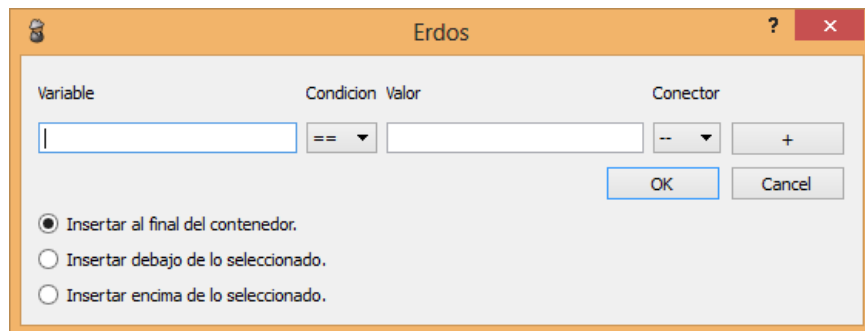
que el programa alcance una determinada condición, una vez que dicha condición es verdadera automáticamente se detiene la ejecución de sentencias dentro del bucle.



(Fig. 23) Figura Hasta vista desde el Entorno de Trabajo.

## Parámetros

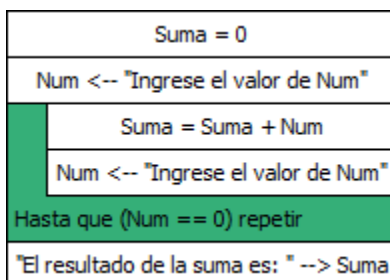
La figura cuenta con cuatro parámetros: variable: (donde se especifica el nombre de la variable sobre la cual se ejecutará la condición), condición: operador de comparación con el que se comparará el valor de la variable; valor: con el cual se comparará la variable y conector que permite armar una sentencia condicional compuesta.



(Fig. 24) ventana de Parámetros de la figura "Hasta".

## Ejemplo

El siguiente ejemplo es similar al anterior pero esta vez se utiliza una figura "Hasta" para construir el bucle en reemplazo de la figura "Mientras". El programa sumará los valores ingresados hasta que ingresemos un cero. Eso hace que se detenga la iteración contenida dentro del UNTIL REPEAT.



### 3.3.8 Figura “Caso” (Case)

La figura “Caso” (también denominada “Case”) es una estructura de control que al igual que el IF Condicional permite dirigir el flujo de un programa de acuerdo al cumplimiento o no de una determinada condición. La figura “Caso” sería el equivalente de poseer varias figuras “Si” encadenadas, de manera que permite evaluar varias condiciones y definir para cada una de ellas un curso de acción diferente.

Si (Var) vale										
1	2	3	4	5	6	7	8	9	10	otro
*****	*****	*****	*****	*****	*****	*****	*****	*****	*****	*****

(Fig. 25) Figura Caso vista desde el Entorno de Trabajo.

Concretamente la figura permite evaluar el valor de una variable numérica y a partir de dicho valor definir distintos cursos de acción. Se utilizan límites (inferior y superior) los cuales definen los valores en los cuales se comparará la variable.

Por ejemplo para una figura que posea un límite inferior de 1 y un límite superior de 5 la figura tiene la siguiente forma:

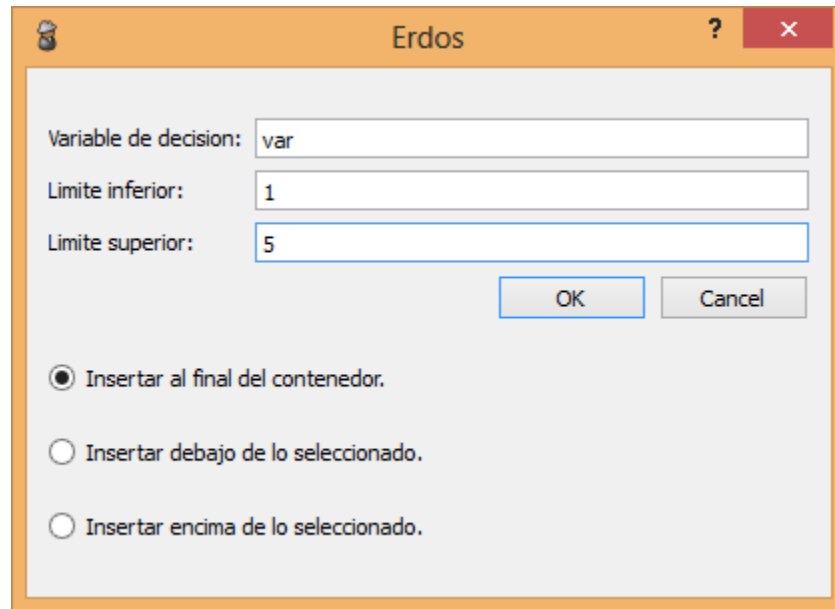
Si (var) vale					
1	2	3	4	5	otro
*****	*****	*****	*****	*****	*****

(Fig. 26) Figura Caso con límite inferior de 1 y límite superior de 5.

En el ejemplo anterior se evalúa el valor de la variable *var*, para cada valor definido en el rango del límite inferior o límite superior se sigue un camino distinto. Ahora bien si *var* toma un valor fuera de ese rango, se ejecutarán las sentencias incluidas en el camino denominado “Otro”.

#### Parámetros

La figura cuenta con tres parámetros: variable de decisión: que especifica la variable sobre la cual se evaluarán los valores, límite inferior y límite superior: que especifican el rango de valores que se van a evaluar.

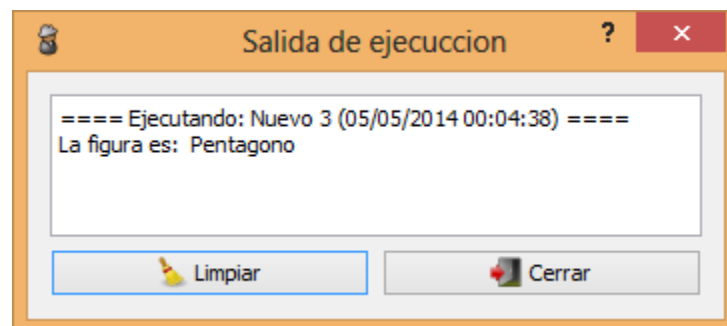


### Ejemplo

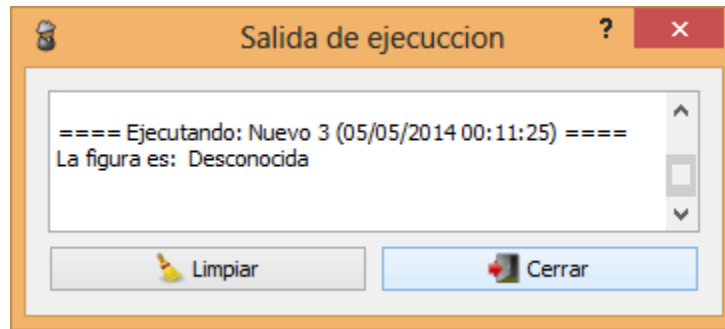
El siguiente programa hace uso de la figura “Caso”, se pide ingresar un número de lados (mínimo 3 y máximo 8) y el algoritmo automáticamente determinará a partir del número de lados el nombre del polígono asociado (triángulo, cuadrado, pentágono, etc.)

Num <-- "Ingrese el Numero de lados (minimo: 3, maximo: 8)"						
Si (Num) vale						
3	4	5	6	7	8	otro
Poligono = 'Triangulo'	Poligono = 'Cuadrado'	Poligono = 'Pentagono'	Poligono = 'Hexagono'	Poligono = 'Heptagono'	Poligono = 'Octagono'	Poligono = 'Desconocida'
"La figura es: " --> Poligono						

Al ejecutar el programa si ingresamos un número de cinco lados:



Si por el contrario especificamos un valor fuera del rango (por ejemplo: 16), el programa mostrará lo siguiente:



## 3.4 Sintaxis del Lenguaje

### 3.4.1 Tipos de Datos

Erdös permite utilizar variables que al utilizarse por primera vez se instancian con un determinado tipo de datos. Una vez instanciada el contenido de una variable puede modificarse pero solamente se puede utilizar el tipo de datos con el que previamente fue instanciada.

Los tipos de datos admitidos por Erdös son:

Tipo de Dato	Descripción	Ejemplos
<b>N Numérico</b>	Cualquier valor numérico sobre el cual pueden realizarse operaciones aritméticas	3.14159 -101 2e+10
<b>String</b>	Cualquier valor alfanumérico (cadenas de texto, caracteres, etc.) encerrado entre comillas simples ('').	'a' 'palabra' '\$#@/&?'
<b>Booleano</b>	Tipo de Dato que solo admite dos valores: Verdadero (True) o Falso (False)	True False
<b>Array</b>	Es un arreglo de datos estático que contiene elementos, los mismos pueden ser accedidos a través de los índices del array. Los elementos de un array pueden ser de los tipos de datos anteriormente citados (numérico, string o booleano).	[1,2,3] ['a','b','c','d'] [true, false, true]

Aclaración: A los elementos de cada array se acceden mediante el nombre del array y un índice que indica la posición del elemento.

#### Ejemplo

Se declara el siguiente array:

**Var = [10 , 5 , 3 , 25 , 7]**

El elemento Var[0] tiene el valor de: 10.

El elemento Var[1] tiene el valor de: 5.



El elemento Var[3] tiene el valor de: 25.

### 3.4.2 Operadores

Los operadores son utilizados en las sentencias para realizar cálculos aritméticos o condicionales. La lista de operadores válidos es la siguiente:

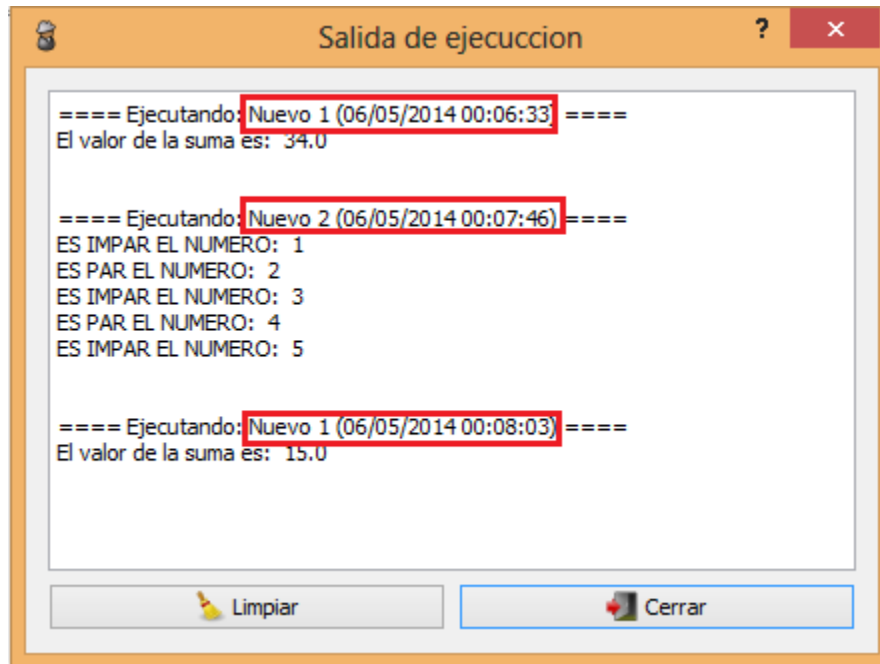
Símbolo	Descripción	Ejemplo	Resultado
=	Operador de Asignación	a = 5	5
+	Operador de Suma	5 + 3	8
-	Operador de Resta	7 - 4	3
*	Operador de Multiplicación	2 * 3	6
/	Operador de División	14 / 3	4,666667
//	Operador de División Entera	14 / 3	4
%	Operador de Módulo (Resto)	16 % 4	0
**	Operador de Exponenciación	2 ** 8	256
and	Operador Booleano AND	true and false	FALSE
or	Operador Booleano OR	true or false	TRUE
<	Operador de Comparación menor	5 < 3	FALSE
>	Operador de Comparación menor	5 > 3	TRUE
==	Operador de Comparación igual	9 == 9	TRUE
<=	Op. De Comparación menor o igual	8 <= 6	FALSE
>=	Op. De Comparación mayor o igual	7 >= 7	TRUE
!=	Operador de Comparación distinto	1 != 0	TRUE

### 3.5 Ejecución del Diagrama

Luego de diseñar el diagrama podemos ejecutar el algoritmo diseñado en el entorno de trabajo. Para iniciar la ejecución de un diagrama podemos ir al menú **Herramientas** → **Ejecutar Diagrama** o directamente presionar la tecla F5.

La ejecución del algoritmo podrá eventualmente abrir ventanas de ingreso de datos, si el algoritmo diseñado cuenta con figuras “Entrada”.

Una vez finalizada la ejecución se observarán los resultados de la ejecución en la ventana de Ejecución. También es posible acceder a esta ventana a través del menú **Herramientas** → **Mostrar Resultados**.



(Fig. 27) vista de la venta de Ejecución de Erdös.

Como puede observarse en la figura anterior la ventana de Salida de ejecución muestra un histórico de las ejecuciones que hemos efectuado con Erdös. Las mismas se ordenan por fecha y hora, asimismo puede observarse a que pestaña (es decir a que diagrama) corresponde cada ejecución. Los resultados pueden borrarse con el botón Limpiar.

### 3.6 Manejo de Archivos

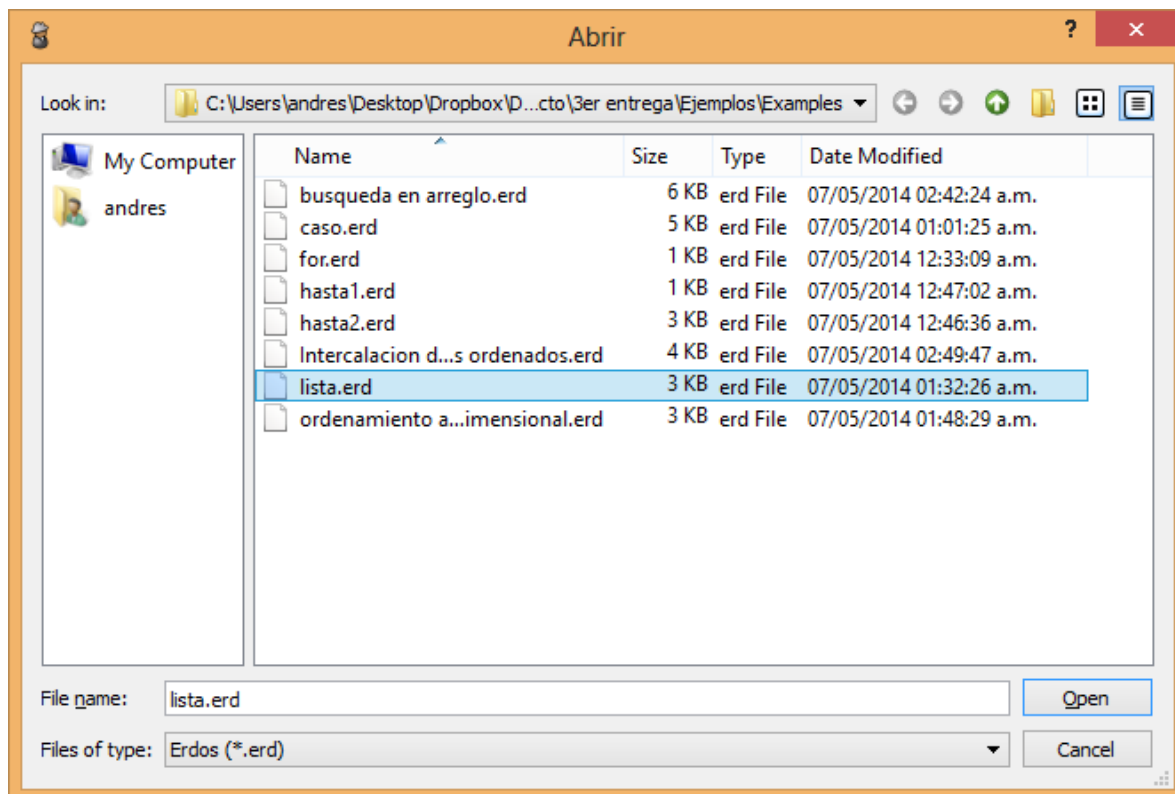
Erdös permite guardar y recuperar los diagramas que hayamos diseñado, de manera de poder diseñar, guardar y retomar el trabajo cuantas veces queramos.

Los diagramas generados en Erdös tienen una extensión **.erd** lo que permite identificar fácilmente este tipo de archivos.

Asimismo el programa permite exportar los diagramas a un formato de imagen ampliamente difundido como lo es el **.png**

#### 3.6.1 Abrir Diagrama

Para recuperar un trabajo guardado debemos ir al menú **Archivo → Abrir**, allí se desplegará el navegador de archivos, aquí debemos seleccionar la carpeta que contiene los diagramas, seleccionar un archivo y clicar la opción de Open.



(Fig. 28) vista de la ventana "Abrir" de Erdös.

### 3.6.2 Guardar Diagrama

Existen dos formas para guardar nuestro progreso en los diagramas diseñados, dichas opciones son: **Guardar** y **Guardar Como**. Ambas se encuentran en el Menú **Archivo**.

La opción **Guardar** nos permite guardar los cambios realizados a un archivo previamente abierto y aplicarlos en el archivo del mismo nombre. Se recomienda ejecutar esta opción cada cierto tiempo de manera de no ir perdiendo nuestro progreso en diagramas que requieran una gran cantidad de trabajo.

La opción **Guardar Como** permite guardar el diagrama que estemos diseñando en un archivo nuevo en la ubicación y con el nombre que especifiquemos, para realizar esto vamos al menú **Archivo** → **Guardar Como**, seleccionamos la carpeta, escribimos el nombre y cliqueamos en **Save**.

## 3.7 Manejo de Errores

Como hemos visto hasta el momento Erdös permite diseñar Diagramas de Chapín y ejecutar el algoritmo descrito en estos diagramas. Un algoritmo puede presentar errores en tiempo de ejecución, Erdös permite detectar cinco tipos de errores los cuales se describen a continuación:

### 3.7.1 Error de Sintaxis

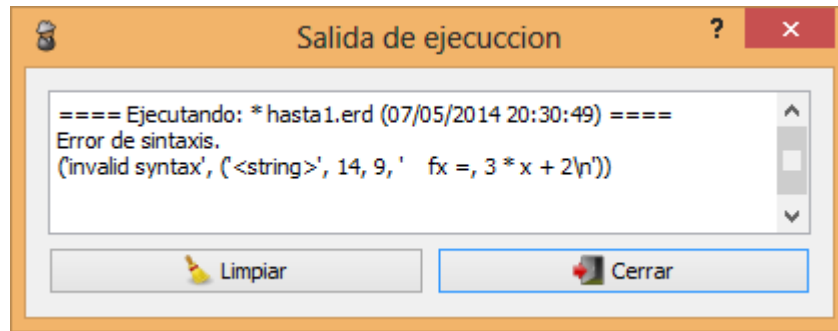
Es el tipo de error más común, consiste en la violación de alguna de las normas de sintaxis impuestas para una figura Sentencia. Por ejemplo el mal uso de los operadores conlleva a un error de sintaxis.

Ejemplo:

Dentro de un programa agregamos por error una coma (,) a la sentencia:

$$F_x = , 3 * X + 2$$

Erdös nos mostrará el siguiente mensaje al realizar la ejecución:



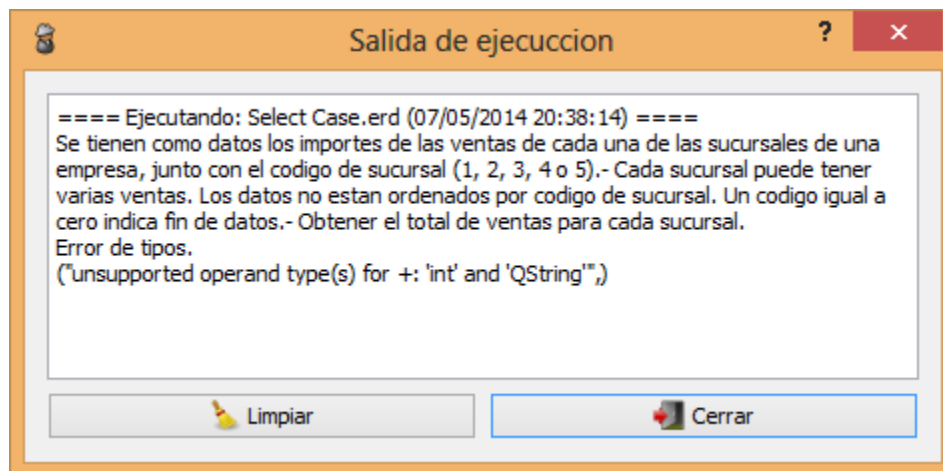
Se observa la descripción del error y la sentencia que está generando dicho error.

### 3.7.2 Error de Tipos

El error de Tipos proviene por una mala utilización de una variable previamente instanciada. Se puede acceder y modificar el contenido de una variable en cualquier parte del ciclo de vida del programa, pero solamente se puede utilizar el tipo de datos con el que previamente dicha variable fue instanciada.

#### Ejemplo

A un programa que realiza la suma de importes de una lista, en tiempo de ejecución ingresamos un String en lugar de un número para el monto, la ventana de ejecución muestra lo siguiente:



### 3.7.3 Error de Nombres

Si bien Erdős permite definir e instanciar variables de manera dinámica (es decir que la creación y la instanciación se realizan en simultaneo), un programa no puede contener

sentencias que hagan referencia a variables que no han sido previamente definidas e instanciadas.

### Ejemplo

El siguiente ejemplo no produce errores debido a que las variables 'a' y 'b' se crean e instancian en la entrada, y la variable 'c' se crea y se instancia en la figura sentencia.

a <-- "Ingrese el valor de a"
b <-- "Ingrese el valor de b"
c = a + b
"Valor de c:" --> c

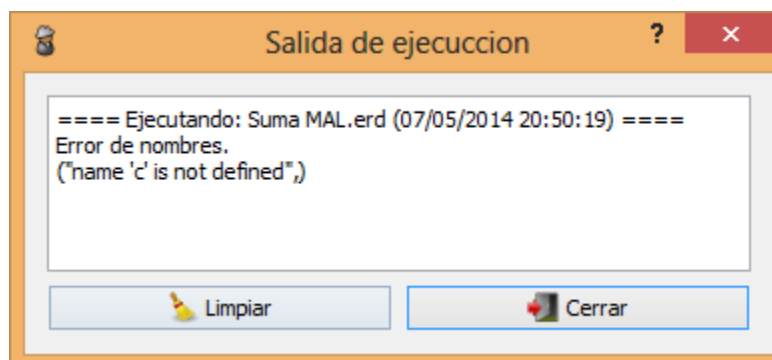
(Fig. 29) Ejemplo de Suma hecha correctamente.

No obstante el siguiente ejemplo si producirá un error, debido a que en este caso la variable 'c' no está previamente creada ni instanciada en ninguna parte del programa.

a <-- "Ingrese el valor de a"
b <-- "Ingrese el valor de b"
d = a + b + c
"Valor de d:" --> d

(Fig. 30) Ejemplo de Suma que produce Error de Tipos.

### Salida de ejecución



### 3.7.4 Error de Índices

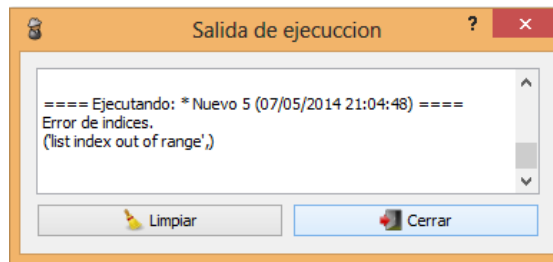
Este tipo de error se produce ante un mal manejo de índices en los array. Si intentamos acceder a una posición del arreglo que no ha sido utilizada se producirá el error.

Ejemplo

Se declara el siguiente array:

**Var = [10 , 5 , 3 , 25 , 7]**

Si intentamos acceder a un elemento que esta fuera del rango, por ejemplo Var[9]:



### 3.7.5 Error de División por Cero

Este tipo de error se produce en un cálculo que efectúe una división por cero.

Ejemplo

Numero <-- "Ingrese el valor de Numero"
Divisor <-- "Ingrese el valor de Divisor"
Resultado = Numero / Divisor
"Valor de Resultado:" --> Resultado

Ejecutamos el ejemplo anterior e ingresamos un cero en el divisor, se produce lo siguiente:

