CURSO
LOTUS SCRIPT
BASICO

1.- INTRODUCCIÓN

LotusScript es un lenguaje de script compatible con BASIC con unas extensiones de lenguaje potentes que permite el desarrollo de aplicaciones orientadas a objetos. LotusScript permite diseñar scripts más complejos y en más sitios que las macros tradicionales. LotusScript y sus herramientas de desarrollo, dan un ambiente de programación común en todas las aplicaciones Lotus en todas las plataformas que soportan Lotus.

LotusScript ofrece a los desarrolladores, una amplia variedad de características que se esperan en un lenguaje de programación moderno y completamente orientado a objetos. Su interfase con Domino se hace a través de clases de objetos predefinidos. Domino supervisa la compilación y la carga de los scripts que crean los usuarios y automáticamente incluye la definición de las clases de objetos Domino. Esto permite que se desarrolle el código de los programas de una forma eficiente.

Mientras las @funciones son ideales para codificar una lógica simple por ejemplo, para la traducción o la validación de campos, LotusScript brinda la habilidad de hacer bucles, construir sentencias 'case' bifurcaciones, subrutinas, etc.

También el Ambiente de Desarrollo Integrado (IDE) realiza una indentación automática, la cual sigue la lógica del programa como en sentencias IF-THEN-ELSE y bucles, y esto hace que los programas sean más fáciles de leer y mantener.

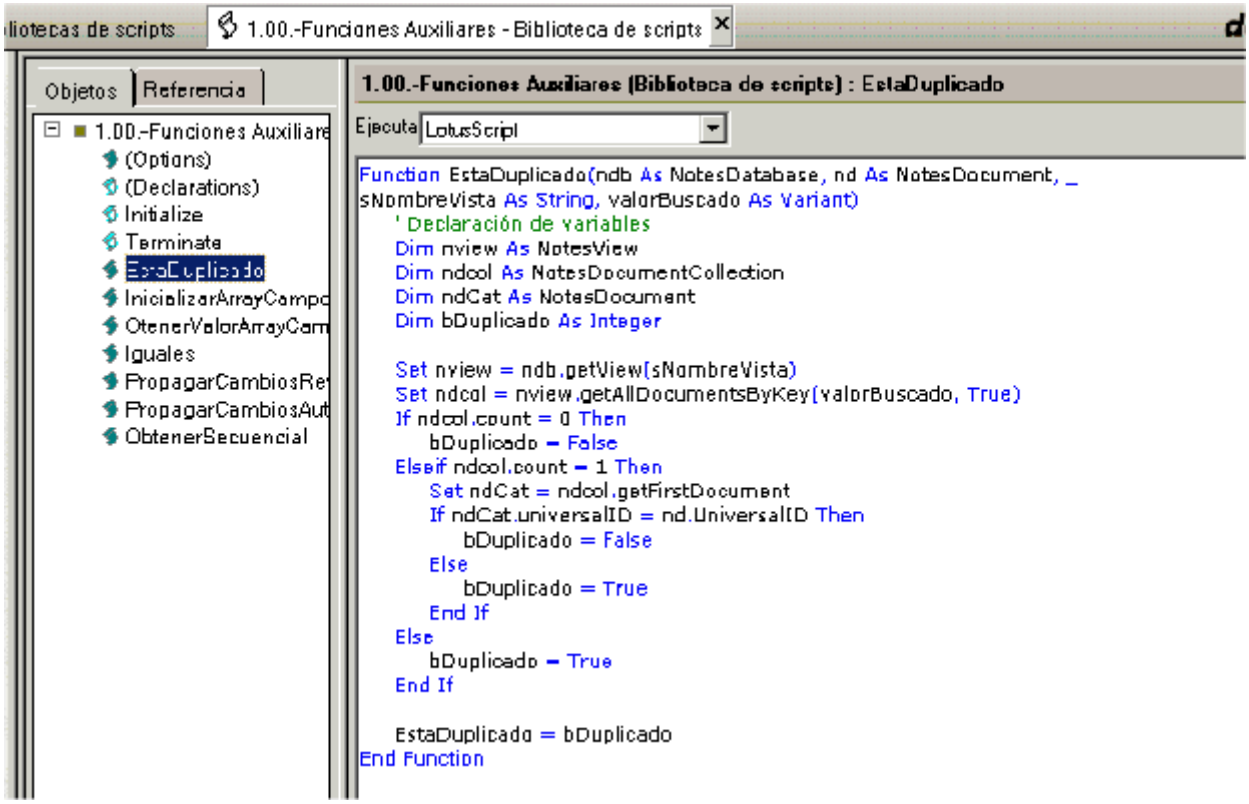

Fig. 1 - Ejemplo de indentación del IDE

La jerarquía de las clases de LotusScript representa el flujo de control que sigues en la interfase de usuario si pasas del icono de una base de datos a una vista, hasta un documento y hasta un campo específico dentro de este documento. Por ejemplo, si estás programando en LotusScript, tendrías que empezar con la clase UIWorkspace y bajar hasta la clase UIDocument que representa el documento abierto actualmente. Una vez que has inicializado estos objetos, tendrás acceso a los campos del documento.

El mismo principio se aplica si estás programando con las clases back-end de Domino, que representan los objetos con lo que puedes querer trabajar y que no están en la interfase del usuario. Empezarías en la clase NotesSession y bajarías a la clase NotesDatabase y a la clase NotesDocument, pero esto lo veremos con más detenimiento en el modelo de objetos de Domino.

2.- VENTAJAS LOTUSSCRIPT

LotusScript ofrece las siguientes ventajas:

- Derivado de BASIC

Ya que LotusScript se deriva del BASIC, es fácil de aprender, especialmente para usuarios de Visual Basic. Se pueden escribir scripts complicados usando condiciones, bifurcaciones, subrutinas, bucles y otras convenciones del lenguaje.

- Multi-plataforma

LotusScript es un lenguaje script parecido al BASIC multi-plataforma. Trabaja en plataformas como Windows, Macintosh, OS/2, UNIX, OS/390, y AS400. Los scripts desarrollados en Windows se ejecutan sin cambios en cualquier otra plataforma soportada. Esta portabilidad es importante cuando aplicaciones desktop se convierten en aplicaciones para trabajo en grupo y los documentos se envían por correo o son compartidos por otros usuarios.

- Orientado a objetos

Los productos Lotus nos traen una serie de clases y objetos que están disponibles para LotusScript. Se pueden escribir scripts para acceder y manipular estos objetos. Los scripts están manejados por eventos, ya sea una acción, haciendo clic en un objeto o botón, abriendo un documento o una vista.

- Soporte OLE

Usando LotusScript, Notes puede ser el contenedor perfecto para documentos de SmartSuite y otras aplicaciones OLE como Microsoft Office.

- Coexistencia con @functions

Lotus sigue soportando las @functions y LotusScript trabaja bien con ellas.

- Entorno de Desarrollo Integrado

El Entorno de Desarrollo Integrado de LotusScript (IDE) brinda una interfase para crear, editar y depurar los scripts, y además revisar las variables y propiedades de las clases. El IDE le permite escribir scripts más complejos en Notes.

- Librerías LotusScript

Se pueden crear funciones y librerías de clases y reutilizarlas en otras aplicaciones mediante la sentencia USE.

3.- Uso de Scripts contra el uso de fórmulas

En general, las fórmulas son mejores para trabajar con el objeto con el que el usuario está trabajando actualmente, por ejemplo, para dar un valor predeterminado a un campo o para determinar el criterio de selección de una vista. Los scripts se usan mejor para acceder objetos existentes, por ejemplo, para cambiar un valor en un documento basado en valores de otros documentos. Los scripts dan algunas capacidades que las fórmulas no dan, como la habilidad de manipular campos de texto enriquecido. Las fórmulas dan un mejor rendimiento en algunas situaciones y son convenientes para aplicaciones simples.

Cuando se nos plantee decidir cuándo usar LotusScript o fórmulas para una tarea determinada, depende normalmente de la complejidad de la misma. Hay que considerar ciertas cosas que detallamos a continuación.

LotusScript no está disponible en las siguientes áreas de Notes:

- Diseño de Vistas: fórmulas de formulario, fórmulas de selección, fórmulas de columnas.

- Diseño de formularios: títulos de vistas, títulos de secciones, acceso a secciones, inserción de subformularios, ocultación de párrafos.

- Diseño de campos: valores predeterminados, traducción de datos introducidos, validación de datos introducidos, valores calculados.

- Fórmulas de campos de palabras clave (botones radiales, casillas de verificación, cuadro de diálogo, etc.).

- Fórmulas de SmartIcons

Trata de utilizar la siguiente información para ayudarte a determinar cuándo utilizar fórmulas o LotusScript para realizar alguna tarea de programación en Notes:

- Utiliza una fórmula cuando exista una @función o @command que realice la tarea.

- Utiliza LotusScript para control de procesos complejos y para bucles(la versión 6 de Domino ya tiene @fórmulas para trabajar con bucles, no obstante suele ser mucho más aconsejable implementar estos con LotusScript).

- Utiliza LotusScript para acceder y manipular datos almacenados en documentos, particularmente para acceso entre varios documentos y varias bases de datos. Por ejemplo, LotusScript es una buena herramienta para crear un agente que busca en todas las bases de datos del área de trabajo y retorna información como el tamaño de las bases de datos, porcentaje usado, número de documentos, etc. LotusScript también es una buena herramienta para ejecutar una búsqueda de texto en varios documentos y realizar una acción con los resultados de la búsqueda.

- Utiliza LotusScript para realizar tareas que el lenguaje de fórmulas no soporte.

- ¿Utilizar clases del front-end o back-end? Las clases del front-end usan el mismo código Domino cómo los @commands equivalentes, así que LotusScript no se ejecutará mejor que las fórmulas cuando se usen estas clases. Las clases del back-end sin embargo, usan diferente código y se desempeñan mejor que sus @funciones equivalentes. Por ejemplo, evita utilizar la clase NotesUIDocument del front-end para realizar actualizaciones de muchos campos. La clase NotesDocument del back-end es mucho más rápida y te permite asignar tipos de datos y añadir nuevo campos (ocultos). La clase del front-end permite actualizar sólo campos que ya existen en el formulario, y te permite insertar sólo texto en el campo, como lo hace el *@Command([EditInsertText])*. Además, las clases del front-end no funcionarán en agentes programados a ejecutar por el servidor, sólo en agentes que se ejecutan en la estación de trabajo del usuario, por ejemplo desde el menú.

- Si utilizas muchas @funciones en una fórmula en secuencia, trata de cambiar las fórmulas a código LotusScript. Sin embargo, las fórmulas que necesitan sólo una @función, como *@Command([FilePrint])*, son más eficientes y tienen un mejor desempeño que scripts que hagan lo mismo.

4.- Scripts y Sentencias

Un script se compone de una o más líneas de texto llamadas *sentencias*. Las sentencias se procesan en secuencia por el compilador. LotusScript interpreta el carácter de nueva línea (retorno de carro) como el final de una sentencia.

Cada sentencia se escribe generalmente en una línea pero algunas se extienden a más de una línea. Para continuar una sola sentencia en varias líneas, usa el carácter del guión bajo (_) como carácter de continuación de línea.

Para utilizar el carácter de continuación de línea:
- Pon un espacio en blanco antes
- Utilízalo al final de una línea de texto.

Por ejemplo, el siguiente script muestra 2 sentencias. La primera está contenida en una sola línea y la segunda en dos:

```
Dim tNombre1 As String
Dim tNombre2 _
As String
```

Identificando los elementos del lenguaje

Los elementos del lenguaje utilizados para construir sentencias y scripts son:

- Identificadores
- Palabras clave
- Literales y constantes
- Operadores

Identificadores

Son nombres que se le dan a elementos individuales del script. Variables, constantes y funciones son algunos de los elementos que requieren un nombre o identificador. Cuando se crean identificadores, hay que tener en cuenta las siguientes reglas:

- El primer carácter debe ser una letra
- Los caracteres válidos son letras, dígitos y el carácter del guión bajo (_)
- El número válido de caracteres puede ser desde 1 hasta 40
- Caracteres con códigos ANSI superiores a 127 (aquellos fuera del rango ASCII) son legales.

Algunas clases de productos Lotus y clases OLE pueden definir propiedades o métodos cuyos identificadores utilizan caracteres ilegales para identificadores LotusScript. En estos casos, hay que poner como prefijo a tales caracteres con una tilde (~) para hacer que el carácter sea válido.

Ejemplo:

Call ProductClass.LoMethod$ ‘ Ilegal
Call ProductClass.LoMethod~$ ‘ Legal

- Los nombres no discriminan mayúsculas. Por ejemplo, *‘NEdad’* es lo mismo que *‘nedad’*.

Ejemplos de identificadores válidos son:

xyz
i
A_27
PrimerApellido
T1234_567

Palabras clave

Son palabras que LotusScript reserva para su propio uso. Tienen un significado o función particular en el lenguaje LotusScript. No puedes utilizar estas palabras en contextos diferentes al que fueron creadas. Estas nombran a sentencias, funciones predefinidas, constantes predefinidas, y tipos de datos. Las palabras clave ‘New’ y ‘Delete’ pueden utilizarse para nombrar subs (procedimientos) que definas en el script. Otras palabras clave no son nombres pero aparecen en las sentencias por ejemplo: ‘NoCase’ o ‘Binary’. Algunos de los operadores en LotusScript son palabras reservadas como ‘Eqv’ y ‘Imp’.

Literales

Son valores de datos que no cambian. Los valores de los literales pueden ser números o cadenas.
Las cadenas se escriben entre comillas (“ “), un par de barras verticales (| |) o un par de llaves ({ }), pero lo más común es utilizar las comillas.

Los siguientes valores son ejemplos de literales:
327 ‘ Literal numérico
“Sonia Agüero” ‘ Literal de texto o cadena

Constantes

Son nombres que se utilizan para representar valores que no cambiarán en los scripts. Puede pensarse que una constante es un nombre que damos para un valor literal fijo. Las constantes normalmente se utilizan para dar nombres en español o inglés a valores que son comprendidos por el ordenador.
Las constantes pueden ser:

- Predefinidas como parte de LotusScript y estas siempre estarán disponibles en una aplicación. La siguiente tabla muestra las constantes predefinidas por LotusScript:

Constante	**Valor**
NULL	Es un valor especial que representa que la información es

	“desconocida” o “que falta”. Sólo puede ser asignado a variables de tipo Variant.
EMPTY	Es el valor inicial de una variable Variant. Si se convierte a string, su valor es “ “ (una cadena vacía).
NOTHING	Es el valor inicial de una variable de referencia a un objeto.
TRUE y FALSE	Los valores numéricos respectivamente son -1 y 0. Estos valores son retornados por todas las operaciones de comparación y lógicas.
PI	La proporción de la circunferencia de un círculo a su diámetro. (3.14159265358979)

- Predefinidas por LotusScript, en el fichero LSCONST.LSS. Estas constantes están disponibles en un módulo sólo cuando el módulo incluye explícitamente el fichero en el que están definidas. Hay que utilizar la directiva %Include para incorporar el fichero a la aplicación en un módulo que debe ser cargado cuando necesites estas constantes. La sintaxis para incluir este fichero es:

%Include “LSCONST.LSS”.

- Definidas por el desarrollador de la aplicación, en un módulo de la aplicación o en un fichero que se incluya explícitamente en un módulo. Se utiliza la palabra reservada ‘Const’. Por ejemplo:

```
Const X = 123.45 ‘ Define una constante Double
Const Y = 123 ‘ Define una constante Integer
Cont NOMBRE = “Sonia” ‘ Define una constante String
```

Operadores

Como en otros lenguajes de programación, los operadores son símbolos o palabras clave que se utilizan para manipular datos en expresiones. El valor de una expresión se determina por el orden en que se evalúan sus partes. Los operadores tienen un orden de precedencia, los de mayor precedencia se evalúan antes que los de menor precedencia y los que están al mismo nivel, se evalúan de izquierda a derecha.

Para saltarse el orden normal de evaluación en una expresión, se utilizan los paréntesis. Las subexpresiones entre paréntesis se evalúan antes que otras partes de la expresión de izquierda a derecha. La siguiente tabla, sintetiza el orden de precedencia de los operadores existentes. Los operadores en la misma línea tienen la misma precedencia. En orden de mayor a menor, los operadores de LotusScript son:

Tipo de Operador	**Operador**	**Operación**
Aritméticos	^	Exponenciación
	-	Negación unitaria
	*, /	Multiplicación, división de punto flotante

	\	División de enteros
	Mod	División módulo (residuo)
	-, +	Resta, suma
Concatenación	&	Concatenación de Strings
Relacionales (Comparación)	=, <>, ><, <, <=, =<, >, >=, =>, Like	Comparación numérica y de string 'Igual que', 'No igual que', 'No igual que', 'menor que', 'menor o igual que', 'menor o igual que', 'mayor que', 'mayor o igual que', 'mayor o igual que', Contiene (emparejamiento de substrings)
Comparación de referencia de objetos (Misma precedencia que los relacionales)	Is, IsA	Comprueba el tipo de objetos, se refiere al mismo objeto
Lógicos	Not	Negación lógica
	And	Booleano o bitwise And
	Or	Booleano o bitwise Or
	Xor	Booleano o bitwise Or exclusivo
	Eqv	Booleano o bitwise de equivalencia lógica
	Imp	Booleano o bitwise de implicación lógica
Asignación	=	Asignación

5.- Identificando los tipos de datos

La unidad básica con la que trabajan los programas son los datos. Los datos son otra palabra para valores. Los valores pueden ser de diferentes tipo, como números o letras. El tipo de datos de un valor ayuda en LotusScript a identificar:

- Cuanta memoria se debe reservar para almacenar y manipular el valor.
- Qué operaciones pueden realizarse con el valor.

LotusScript reconoce los tipos de datos escalares:
- Numéricos: datos que pueden utilizarse para hacer cálculos
- String (cadenas): datos que representan cualquier número de letras.

Además, LotusScript reconoce tipos de datos agregados, como arrays de los que hablaremos más adelante. También hay un tipo de datos 'Variant', que puede utilizarse para representar cualquiera de los otros tipos de datos no agregados.

Tipos de datos escalares

Los tipos de datos strings y numéricos, representan datos que tienen un solo valor en cualquier momento. Estos son conocidos como los tipos de datos escalares:

Tipo de dato	Palabra clave	Sufijo	Descripción	Valores
Integer	INTEGER	%	Números sin componente fraccional	-32.768 a +32.767
Long	LONG	&	Enteros largos	-2.147.483.648 a +2.147.483.647
Double	DOUBLE	#	Valores de punto flotante de doble precisión	+ o – 17 dígitos con un punto flotante
Single	SINGLE	!	Valores de punto flotante	+ o – 7 dígitos con un punto flotante
Currency	CURRENCY	@	Datos numéricos, representando valores monetarios	Formato de punto flotante con hasta 15 dígitos a la izquierda del punto decimal y 4 a la derecha.
String	STRING	$	Cadenas de caracteres de tamaño variable	0 a 32k caracteres
Fixed String	STRING * n	Ninguno	Cadenas de caracteres de un tamaño específico	0 a 32k caracteres

6.- Declaración y Asignación de Variables

Los scripts manipulan varios tipos de datos usando constantes y variables. Una variable, es una ubicación de memoria que tiene un nombre y que almacena el valor de un dato asignado a esa ubicación. El valor de una variable puede cambiar durante la ejecución del programa al contrario que las constantes que tienen su valor fijo. Al proceso de definir el nombre de una variable y su tipo de datos se le llama "declaración de variables".

Sigue estos pasos para declarar una variable y asignarle un valor:
1. Declara la variable usando la sentencia 'DIM'.
2. Asigna datos a la variable una o muchas veces.

Usando la sentencia DIM

Sintaxis
Utiliza uno de los dos métodos para declarar una variable:
DIM nombreVariable AS tipo_de_dato
DIM nombreVariable(caracter sufijo)

Ejemplo:
Dim dirección as String
Dim interes! ' la variable 'interes', se ha declarado de tipo SINGLE

Asignando datos

Una vez que la variable ha sido definida, utiliza el operador de asignación, el símbolo igual (=), para asignar datos a la variable.
Ejemplo:
Las siguientes sentencias muestran la definición de una variable, asignación de un valor y mostrarlo:

Dim nombre as String
nombre = "Pepe Perez"
MessageBox(nombre)

Comunicación con los usuarios

Con LotusScript tenemos 2 formas de comunicarnos con los usuarios:
- La función InputBox
- La función MessageBox

Solicitando valores (Función InputBox)
La función InputBox crea un cuadro de diálogo que pide al usuario que ingrese algo. El indicador utilizado para solicitar información es definido por el usuario. Los valores ingresados se devuelven al script.

La sintaxis para la función InputBox es:
InputBox(‘indicador’)

Mostrando mensajes (Función MessageBox)
La función MessageBox habre un cuadro de diálogo que muestra información al usuario, espera por una respuesta y devuelve información sobre esa respuesta.
La sintaxis simple para la función MessageBox es:
MessageBox(“Este es el mensaje”)

Retornando valores de respuesta
La función MessageBox también devuelve información sobre una respuesta.

Ejemplo:
El siguiente script muestra un messagebox con un botón de Sí y otro de No y devuelve un valor indicando qué botón fue seleccionado:

```
DIM respuesta AS INTEGER
respuesta = MessageBox(“¿Desea continuar?”, 4)
MessageBox(respuesta)
```

Para esto se nos hace necesario conocer la sintaxis completa de la función MessageBox, donde cada botón que queramos que se muestre se lo pasamos como parámetro a la función y cada botón devuelve un valor entero determinado, siguiendo esta tabla:

Nombre de Constante	Valor	Botones que muestra
MB_OK	0	OK (Aceptar)
MB_OKCANCEL	1	OK y Cancel (Aceptar y Cancelar)
MB_ABORTRETRYIGNORE	2	Abort, Retry, e Ignore (Anular, Reintentar y Omitir)
MB_YESNOCANCEL	3	Yes, No, y Cancel (Sí, No y Cancelar)
MB_YESNO	4	Yes y No (Sí y No)
MB_RETRYCANCEL	5	Retry y Cancel (Reintentar y Cancelar)

Tabla de Botones de la función MessageBox

Igualmente, puedes personalizar el icono que se mostrará en el cuadro con el mensaje, ya sea para indicar un aviso, error, etc.

Nombre Constante	Valor	Icono mostrado
MB_ICONSTOP	16	Señal de Stop
MB_ICONQUESTION	32	Marca de Interrogación
MB_ICONEXCLAMATION	48	Punto de exclamación
MB_ICONINFORMATION	64	Información

Tabla de Iconos de la función MessageBox

La sintaxis completa de la función MessageBox es la siguiente:

MessageBox (*mensaje* [**,** [*botones* + *icono* + *default* + *modo*] [**,** *Título*]] **)**
Donde:
Mensaje: es un string que incluye el mensaje que mostrará el cuadro de mensaje.
Botones: Define el número y tipo de botones que se mostrarán en el cuadro de mensaje. Es una constante numérica que se ajusta a la "tabla de botones".
Icono: Define el icono que se mostrará en el cuadro de mensaje. Es una constante numérica que se ajusta a la "tabla de iconos".
Default: Define el botón predeterminado en el cuadro de mensaje. Presionar ENTER tiene el mismo efecto que pulsar el botón predeterminado.

	Nombre Constante	**Valor**	**Botón predeterminado**
	MB_DEFBUTTON1	0	Primer Botón
	MB_DEFBUTTON2	256	Segundo Botón
	MB_DEFBUTTON3	512	Tercer Botón

Modo: Define la modalidad del cuadro de mensaje

	Nombre Constante	**Valor**	**Descripción**
	MB_APPLMODAL	0	Modo de aplicación. Detiene la aplicación actual hasta que el usuario responde al cuadro de mensaje.
	MB_SYSTEMMODAL	4096	Modo de sistema. Detiene todas las aplicaciones hasta que el usuario responda al cuadro de mensaje.

Título: Es el string que aparecerá en la barra de título del cuadro de mensaje. Para saber qué botón ha presionado el usuario, tenemos la siguiente tabla de valores numéricos por cada tipo de botón, que es un valor numérico entre 1 y 7 que devuelve la función MessageBox.

	Valor Devuelto	**Botón**	**Constante**
	1	OK	IDOK
	2	Cancel	IDCANCEL
	3	Abort	IDABORT
	4	Retry	IDRETRY
	5	Ignore	IDIGNORE
	6	Yes	IDYES
	7	No	IDNO

Actividad 3

Esta actividad produce un cuadro de diálogo que pide al usuario información de un depósito y asigna el valor a una variable llamada nDepósito.
1. Crea un botón en el formulario **xxxx** en la base de datos **xxxx.nsf**
2. En el panel de programación, ingresa el siguiente código:

```
Dim nDeposito as Single
nDeposito = InputBox("Ingrese el monto del depósito")
MessageBox(nDeposito)
```

3. Para ejecutar el script, selecciona Diseñar – Previsualizar en Notes. Luego pulsa en el botón que has creado.

Actividad 4

En esta actividad, utilizarás la concatenación para combinar variables string y mostrarlas en un MessageBox. El ampersand (&) se utiliza para combinar variables string
1. Crea un botón en el formulario **xxxx** en la base de datos **xxxx.nsf**

2. En el panel de programación, ingresa el siguiente código en el evento click del botón:

```
Dim snombre as string
Dim sprofesion as string

snombre = InputBox("Ingrese su nombre: ")
sprofesion = InputBox("Ingrese su profesión: ")

MessageBox(snombre & " trabaja como " & sprofesion)
```

3. Prueba el botón.

7.- Datos Variant, Constantes y utilizar el fichero LSCONST.LSS

Utilizando el tipo de datos Variant

Es un tipo de datos de uso general. Una variable de tipo Variant puede contener un número, un string, los valores de las constantes NULL o EMPTY, un array, una referencia a un objeto o una lista. Cuando asignas un valor a una variable Variant, LotusScript determina su tipo de datos basándose en el valor en sí.

¿Cuándo utilizar Variants?

Las variables de tipo Variant, dan al programa mucha flexibilidad, ya que las variables pueden utilizarse de forma diferente en distintas operaciones. Utiliza los Variants cuando no estés seguro del tipo de datos de los valores que el programa va a manipular.

Observación: LotusScript no tiene un tipo de datos de fecha u hora. Los Variants se utilizan para operaciones de fecha y hora.

Desventajas de utilizar Variants

A menos que los variants vayan a tener un verdadero propósito en el script, trata de no usarlas por las siguientes razones:
- Los Variants utilizan más memoria que otros tipos de datos.
- Los Variants se procesan más lentamente.
- Puedes perder el rastro del tipo de dato del valor con el que estás trabajando una o más variables son Variant.

Creando Constantes

Una constante, al igual que una variable, nombra una ubicación en memoria. Mientras el valor de una variable puede cambiar durante la ejecución del programa, el valor de una constante se mantiene fijo. Utiliza constantes para valores que no son susceptibles a cambiar.
Sigue estos pasos para crear una constante:

1. Declara la constante, utilizando la sentencia CONST
2. Asigna el valor una vez, y ya está declarada.

Utiliza la sentencia CONST con la siguiente sintaxis:
CONST nombre_constante = valor

El nombre de la constante puede ser cualquier identificador LotusScript válido. El valor de una constante es un literal que es uno de los tipos de datos escalares; también puede ser una expresión que se evalue a un valor constante.

Ejemplo:

La siguiente expresión crea una constante string para una dirección
Const DIRECCION = "Calle Barco 41"

LotusScript también trae un conjunto de constantes con nombres predefinidos que pueden utilizarse en los scripts. Estas constantes están definidas en un fichero llamado LSCONST.LSS.

Ventajas de utilizar LSCONST.LSS
Cuando se incluye el fichero en una aplicación, puedes referirte a las constantes utilizando sus nombres en inglés en lugar de usar su equivalente numérico. Por ejemplo: la combinación de botones "Sí" y "No" de la función MessageBox puede referirse a su nombre parecido al inglés 'MB_YESNO' en lugar de su valor numérico.

Sintaxis
Para incluir el fichero LSCONTS.LSS en el script, escribe la siguiente sentencia en el módulo 'Globals' de las declaraciones del evento:
%Include "LSCONST.LSS"

Observación: Si el fichero está ubicado en otro directorio que no sea Notes, incluye la ruta.

La directiva %Include
Esta directiva, inserta código de otro script en el fichero actual en tiempo de compilación. Utiliza la siguiente sintaxis:
%INCLUDE "nombre_fichero.LSS"
- La directiva debe ser el único comando en una línea excepto que se ponga un comentario opcional.
- La directiva debe estar seguida de un espacio en blanco, caracter tabulador o un retorno de carro.
- Se puede incluir una ruta para indicar un directorio específico.

Actividad 5

Esta actividad muestra el uso de la función MessageBox con los argumentos de botones e iconos para obtener información del usuario utilizando el fichero LSCONST.LSS.
1. Crea un botón en el formulario **xxxx**
2. En el panel de diseño, selecciona el módulo 'Globals' del formulario y el evento 'Declarations' para incluir el código

%Include "LSCONST.LSS"

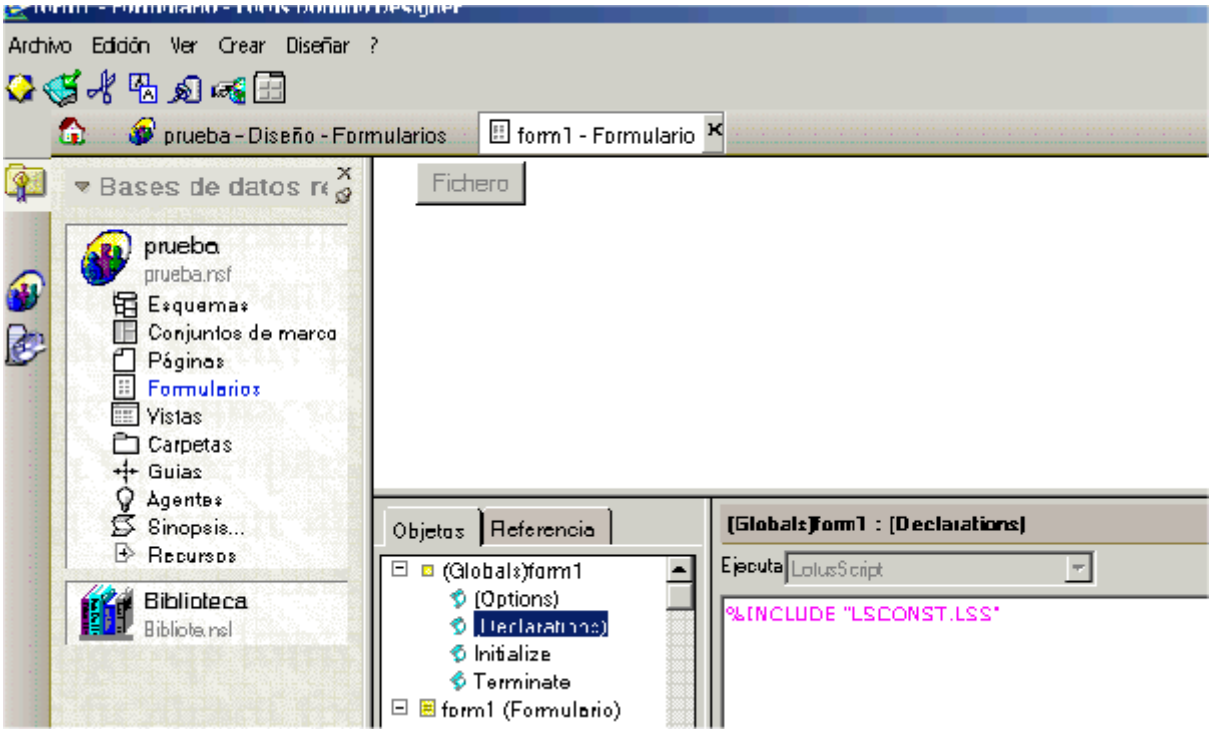

Fig. 6 – Incluir el fichero LSCONST.LSS

3. En el evento click del botón que has creado, incluye el siguiente código:

```
Dim irespuesta as integer
irespuesta = MessageBox("¿Desea continuar?", MB_YESNO +
MB_ICONQUESTION)
MessageBox(irespuesta)
```

4. Prueba el botón

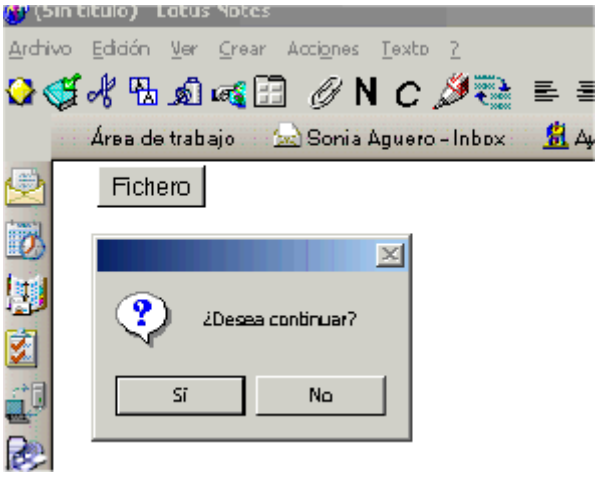

Fig. 7 – MessageBox utilizando LSCONST.LSS

8- Creación de Comentarios

Creación de Comentarios

Los comentarios sirven como documentación del programa, para permitir que otros comprendan fácilmente el propósito y el flujo de tu código.
Para añadir comentarios a tu código LotusScript, utiliza cualquiera de los siguientes métodos:

- Un caracter de apóstrofe (‘): marca el principio de un comentario en una sentencia. El compilador de LotusScript no procesa cualquier cosa que esté en una línea después del apóstrofe. Este puede ser colocado en cualquier lugar de la línea. El final de la línea marca automáticamente el final del comentario.

Ejemplo:

```
Dim x as integer ‘ Declara una variable
x = 1 ‘ Asigna un valor a x
messagebox(x) ‘ Muestra el valor
```

- La palabra reservada REM: utiliza este para comentarios que ocupan una sola línea. El compilador no procesa cualquier cosa en una línea que comience con REM.

Ejemplo:

```
REM La siguiente línea declara la variable x de tipo integer
Dim x as integer
```

- Las directivas %REM y %ENDREM: utiliza estas directivas de compilador para comentarios que utilizan varias líneas. %REM marca el principio del comentario y debe aparecer al principio de una línea y %ENDREM marca el final del comentario y debe ser la primera palabra en la línea.

Ejemplo:

```
%REM
La primera sentencia en este script declara una variable llamada snombre. La
segunda línea asigna el valor “Pablo” a snombre. La tercera sentencia muestra
el valor de snombre.
%ENDREM

Dim snombre as string
Snombre = “Pablo”
Messagebox(snombre)
```

Tarea 1 (Utilizando InputBox)
Escribe un script en la base de datos xxxx.nsf. Habrá que crear un botón que calcule el reembolso por distancia recorrida. El script debe hacer lo siguiente:
1. Utilizar la función InputBox para pedir al usuario el número de kilómetros.
2. Calcular la cantidad del reembolso por la distancia recorrida:
o La tarifa de reembolso es de 0,315 céntimos por kilómetro y es un valor constante.
o El reembolso por la distancia recorrida es el número de kilómetros multiplicado por la

tarifa de reembolso.
3. Mostrar el resultado del script en un cuadro de mensaje y preceder el mensaje con la frase “El reembolso es de “.

9- Arrays

Conceptos Básicos

Un array es un ejemplo de un tipo de dato agregado. De forma similar a las variables escalares, un array se declara creando un nombre, especificando el tipo de datos y asignándole valores. Pero se distingue de las variables escalares en que una variable array se refiere a múltiples valores en lugar de sólo uno.

Elementos del Array

Un array utiliza un nombre de variable para referirse a valores múltiples del mismo tipo de datos almacenados en memoria. Cada valor almacenado como parte de un array se le conoce como 'elemento'. Los elementos del array son referenciados utilizando un índice en combinación con el nombre de variable. El índice identifica un elemento específico
del array.
Por ejemplo, un elemento en el array miArray puede ser referenciado como miarray(1).
El siguiente diagrama representa un array con siete elementos y sus índices:

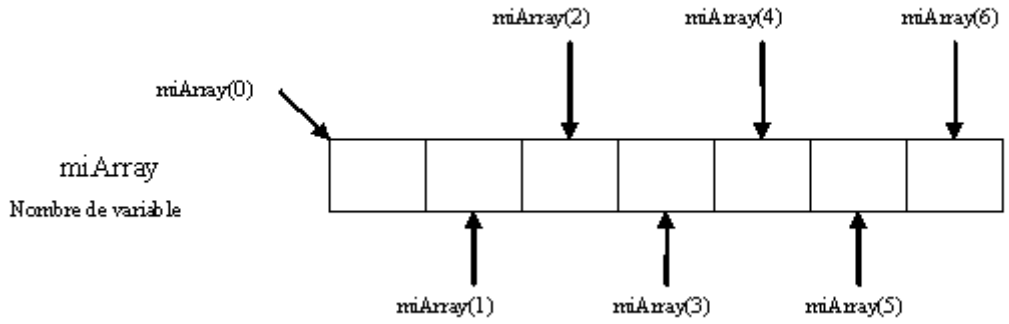

Fig. 8 – Elementos del array

Límites del array

El rango de los índices que se utilizan para especificar los elementos del array son los límites del array. Los límites se identifican por el superior y el inferior. Entre los límites del array están el número de elementos del array.

Los límites del array se especifican en el formato:
límite_inferior to límite_superior

Los arrays tienen un límite inferior predeterminado de cero (0). A menos que se especifique lo contrario, el primer elemento de un array llamado miArray, será miArray(0).

Declarando una variable Array

La sentencia DIM se utiliza para declarar una variable array.

Sintaxis
DIM nombre_array(limite_inferior TO limite_superior) AS tipo_dato

Ejemplo:
Para declarar una variable array para almacenar los días de la semana como 7 elementos string, escribe lo siguiente:
Dim dias(1 to 7) as String

Asignación y recuperación de valores en arrays

Los datos de los elementos de un array se asignan y recuperan en una base de elemento a elemento. El formato para asignar datos a un elemento del array es:
nombre_array(indice) = dato

Actividad 6

Para ilustrar la asignación de strings a cada elemento de un array llamado misDias, haz lo siguiente:
1. Abrir el formulario **xxxx**
2. Crear un botón y escribe lo siguiente en el evento Click:
Dim misDias(1 to 7) as string
misDias(1) = "Domingo"
misDias(2) = "Lunes"
misDias(3) = "Martes"
misDias(4) = "Miércoles"
misDias(5) = "Jueves"
misDias(6) = "Viernes"
misDias(7) = "Sábado"
MessageBox("El tercer elemento de misDias contiene " & misDias(3))
3. Prueba el botón

Creando arrays dinámicos

Puedes cambiar los datos almacenados como elementos del array asignando nuevos datos a los elementos. El número de elementos en las dimensiones del array puede cambiar creando un array dinámico.
Los arrays fijos se declaran una vez, su tamaño se establece permanentemente cuando se declaran, y se reserva el espacio en memoria basado en ese tamaño. Los arrays dinámicos se declaran una vez pero se les puede indicar un tamaño varias veces mientras se esté ejecutando una aplicación, y por lo tanto no utiliza espacio innecesario.
Sigue estos pasos para crear y guardar espacio en memoria para un array dinámico:
1. Utiliza la sentencia DIM, sin especificar límites para declarar el array.
2. Utiliza la sentencia REDIM para establecer los límites y dimensiones del array una vez que haya sido declarado.
3. Utiliza sentencias REDIM adicionales para indicar un nuevo tamaño al array si es necesario.

Utilizando la sentencia REDIM

Utilízala con la siguiente sintaxis para guardar espacio en memoria para un array:

Sintaxis
REDIM nombre_array(limites)

Ejemplo
El siguiente ejemplo muestra cómo declarar un array llamado misDias y luego establecer sus límites utilizando REDIM
Dim misDias() as String
REDIM misDias(1 to 7)

Cambiando el tamaño de un array
Los arrays dinámicos pueden cambiarse su tamaño muchas veces. Cada vez que se utiliza la sentencia REDIM, se puede establecer un nuevo límite superior.

Utilizando la opción PRESERVE
Utiliza la opción PRESERVE para prevenir que los datos existentes previamente en el array se sobrescriban.
La sintaxis para utilizar la sentencia PRESERVE es la siguiente:
REDIM PRESERVE nombre_array(limites)

10- Sentencias Condicionales

Estas sentencias toman decisiones que resultan en distintas acciones. El principal componente de estas sentencias es una condición, la cual especifica la decisión que se debe tomar. LotusScript da dos formas de realizar estas condiciones:
- Sentencias IF
- Sentencias SELECT CASE

Sentencias IF

Se utiliza la sentencia IF cuando un script necesita una acción cuando una expresión es verdadera y otra diferente cuando la expresión es falsa. Las acciones se determinan basándose en el valor de una o más expresiones llamadas condiciones.

Expresiones

Una expresión es una serie de elementos del lenguaje que evalúan a un solo valor en tiempo de ejecución.

Operadores

Los operadores combinan, comparan o manipulan valores específicos dentro de una expresión. Hay cuatro grupos de operadores:

- Numéricos: llevan a cabo operaciones aritméticas (+, -, *, /)

- Relacionales: evalúan la relación entre dos operandos (>, <, <> y =) Cuando dos valores se comparan usando estos operadores, el resultado de la expresión siempre va a retornar TRUE (1) o FALSE (0).

- Lógicos: también conocidos como 'booleanos', se utilizan para especificar la relación entre dos cantidades o conceptos. Los más comunes son NOT, AND y OR. Estos devuelven TRUE (-1) o FALSE (0).

- String: se utilizan para manipular datos de cadenas. Hay disponibles para tres tipos de operaciones con strings:

o Concatenación (&)
o Comparación (operaciones relacionales)
o Operadores de emparejamiento de patrones (Like)

Sintaxis
El bloque IF ... END IF tiene la siguiente sintaxis:
IF condicion THEN
Sentencias
[ELSE
Sentencias]
END IF
La parte que está entre corchetes es opcional.

Actividad 7

El siguiente script prueba si un balance de cuentas es mayor que 300€

```
Dim nbalance as single
Dim ninteres as single
nbalance = InputBox("Ingrese el balance de la cuenta")
if nbalance > 300 then
ninteres = 0.015
else
ninteres = 0
end if
nbalance = (ninteres * nbalance) + nbalance
MessageBox("Su balance es: " & Format(nbalance, "Currency"))
```

11- Sentencias Select-Case

Estas sentencias realizan una operación de correspondencia para determinar el curso de acción. Con SELECT CASE, los posibles valores a los que una expresión puede corresponder están listados con una acción correspondiente si la correspondencia se encuentra. Utiliza SELECT CASE cuando la aplicación debe elegir entre varias opciones.

Sintaxis
Lo siguiente representa la sintaxis de la sentencia SELECT CASE

SELECT CASE select_expresion
[CASE condicion1]
[sentencias]]
[CASE condicion2]
[sentencias]]
...
[CASE ELSE
[sentencias]]
END SELECT

Actividad 8

La siguiente actividad utiliza sentencias SELECT CASE para realizar las pruebas:

```
Sub Click(Source As Button)
Dim snomEmpleado As String
Dim icodDept As Integer
Dim fechaCont As Variant
Dim info As Variant

snomEmpleado = InputBox("Ingrese el nombre del empleado")
fechaCont = InputBox("Ingrese la fecha de contratación del empleado")
info = snomEmpleado & ", " & fechaCont
icodDept = InputBox("Ingrese el código del departamento del empleado (10, 20 o 30):")

Select Case icodDept
Case 30: Messagebox(info & ", " & "Depto. de Marketing")
Case 20: Messagebox(info & ", " & "Depto. de Ventas")
Case 10: Messagebox(info & ", " & "Depto. de Producción")
Case Else: Messagebox("Ingrese un código de departamento válido (10, 20 o 30): ")
End Select
End Sub
```

Especificando una lista de condiciones o rango

Además de las condiciones individuales que ya hemos visto, puede especificarse un rango de condiciones para una sentencia SELECT CASE. Para especificar un rango:
- Listar varias condiciones específicas, separándolos por comas. Por ejemplo: CASE 10, 20, 30
- Utilizar la palabra reservada TO para especificar un límite inferior y un superior para

un rango. Por ejemplo: CASE 1 TO 10.
- Utilizar la palabra reservada IS en combinación con operadores de combinación, para evaluar si una expresión relacional es verdadera. Por ejemplo: CASE IS > 10.

Actividad 9

En esta actividad demostramos el uso de las palabras reservadas TO y IS para probar el rango de los valores:

```
Dim ndeposito as currency
ndeposito = Inputbox("Ingrese el monto del depósito, en cientos: ")
Select Case ndeposito
Case Is < 100 : Messagebox("Has depositado menos de 100€")
Case 100 To 500 : Messagebox("Has depositado entre 100 y 500€)
Case Is >= 1000: Messagebox("Has depositado 1000€ o más)
Case else: Messagebox("Has depositado un cantidad entre 501 y 999€)
End Select
```

Utilizando las funciones Today() y Month() para construir una condición
La función de LotusScript Month() devuelve el mes del año como un entero entre 1 y 12 para cualquier argumento de fecha/hora. La función Today() devuelve la fecha del sistema como un valor de fecha. Combinando estas dos funciones, es posible determinar el mes de la fecha de hoy.

Actividad 10

Lo siguiente, muestra el uso de las funciones Month() y Today() en un Select Case

```
Dim imes as integer
Dim srespuesta as String
imes = Month(Today)
SELECT CASE imes
Case 1: srespuesta = "Enero"
Case 2: srespuesta = "Febrero"
Case 3: srespuesta = "Marzo"
... (sentencias para los casos 4 a 12)
END SELECT
MessageBox("Estamos en el mes de " & srespuesta)
```

Tarea 2 (Utilizando SELECT CASE)
En el formulario xxxx, hay que crear un botón que evalúe la fecha actual para determinar en que trimestre del año nos encontramos.
- Utilizar la sentencia SELECT CASE para realizar la prueba condicional.
- Mostrar el trimestre en el que nos encontramos con un Messagebox.

12- Sentencias de Iteracción-Bucles

Estas sentencias nos permiten realizar bucles dentro de nuestro código, porciones de código que queremos que se repita si cumple ciertas condiciones. Son muy útiles para arrays y listas.

Sentencia For ... Next

La sentencia For...Next se usa muy a menudo para almacenar valores en un array.
Sintaxis
FOR contador = primero TO último
[sentencias]
NEXT [contador]

Ejemplo
El siguiente ejemplo muestra un bucle que almacena valores en un array

```
Dim i as integer
Dim familia(1 to 7) as String

FOR i = 1 TO 5
familia(i) = Inputbox("Ingrese los nombres de 5 miembros de su familia:")
NEXT i
```

Actividad 11

Para mostrar el uso del ciclo For ... Next, haz lo siguiente:
1. Abre el formulario xxxx
2. Crea un botón y escribe lo siguiente en el evento Click

```
Dim misDias(1 to 7) as String

For i = 1 to 7
misDias(i) = Inputbox("Ingrese los días de la semana")
MessageBox("Acaba de ingresar el día " & misDias(i))
Next i
```

3. Prueba el botón.

Tarea 3 (Sentencias iterativas)
En un formulario, crea un botón con la etiqueta "Ingreso de Depósito" y que haga lo siguiente:
- Pida al usuario que ingrese tres depósitos
- Una vez ingresados los 3 depósitos, mostrar cada depósito al usuario con cuadros de mensajes.

Sentencia FORALL

Esta sentencia ejecuta repetidamente un bloque de sentencias por cada elemento en un array, lista o colección. FORALL recupera o pone todos los valores almacenados en un array, lista o colección.
La sintaxis es como sigue:
FORALL variable_referencia IN contenedor
Sentencias

END FORALL

Donde
variable_referencia: es una variable de referencia para los elementos del array, lista o colección. En el cuerpo del bucle Forall, se utiliza 'variable_referencia' para hacer referencia a cada elemento del array, lista o colección llamada por 'contenedor'. La 'variable_referencia' no puede tener un caracter sufijo para indicar el tipo de dato.

contenedor: es el array, lista o colección cuyos elementos quieres procesar.

Ejemplo:
Para recorrer todos los depósitos en un array llamado misDepositos, podrías escribir:
Forall depositos In misDepositos
MessageBox(depositos)
End Forall

Actividad 12

Para mostrar el uso de la sentencias Forall, crearemos un array con valores y en luego pondremos todo el array a 0. Haz lo siguiente
1. Abre el formulario xxxx
2. Crea un botón y en el evento Click escribe lo siguiente:
Dim array(5) As Integer
array(0) = 7
array(1) = 14
array(2) = 21
ForAll x In array
x = 0
End ForAll
3. Prueba el botón.

Sentencia DO

Este es un ciclo indeterminado porque no se conoce de antemano el número de veces en que se ejecuta un bloque de operaciones. Este tipo de sentencias se ejecutan mientras una condición dada sea verdadera o hasta que se vuelva verdadera.

Las sentencias DO...LOOP tienen una sintaxis de dos formas diferentes:
DO [WHILE | UNTIL condicion]
[sentencias]
LOOP

y la otra
DO
[sentencias]
LOOP [WHILE | UNTIL condicion]

Probar antes vrs después

La condición para la ejecución del bucle se prueba ya sea antes os después de cada iteración del bucle, dependiendo de si está pegada al DO o al LOOP. Como resultado, un bucle DO que prueba la condición antes del bucle puede que no se ejecute ni una

vez. Por ejemplo:

```
Dim inum as Integer
inum = 0
DO WHILE inum < 100
inum = inum + InputBox("Ingresa un número")
MessageBox("El número actual es " & inum)
LOOP
Messagebox("El último número fue & inum)
```

Ahora un ejemplo que pruebe la condición después de que se ejecuta el bucle por lo menos una vez.

```
Dim inum as Integer
inum = 0
DO
inum = inum + 2
Print "el número actual es " & inum
LOOP WHILE inum < 50
Messagebox("El número final fue " & inum)
```

Sentencia WHILE ... WEND

Esta sentencia también ejecuta un bloque de sentencias repetidamente mientras la condición sea verdadera. LotusScript prueba la condición antes de entrar en el bucle y antes de cada repetición. El bucle se repite mientras la condición sea TRUE. Cuando la condición es FALSE, la ejecución continua con la primera sentencia que siga al WEND. Es equivalente al uso de DO WHILE.

Sintaxis

```
WHILE condicion
[sentencias]
WEND
```

Ejemplo:

En este ejemplo pondremos un contador, dado un número final por el usuario:

```
Dim cont as Integer
Dim inum as Integer

inum = Inputbox("Ingrese un número")
i = 0
While i <> inum
i = i + 1
Messagebox("El número es: " & i)
Wend
```

13.- Objetos UI del Front-End de Domino

Estos se utilizan para manipular la interfase actual del usuario. Se utilizan para programar eventos y dar acceso al objeto Domino con que está trabajando actualmente el usuario. Estos son los más importantes:

Clase	***Definición***
NotesUIWorkspace	Representa la ventana del área de trabajo actual de Notes.
NotesUIDatabase	Representa la base de datos utilizada actualmente.
NotesUIView	Representa la vista utilizada actualmente.
NotesUIDocument	Representa el documento que está abierto actualmente.
Los siguientes objetos sólo tienen eventos asociados a ellos	
Button	Representa un botón.
Field	Representa un campo.
Navigator	Representa un guía.

14.- Objetos del Back-End de Domino

Estos objetos se utilizan para manipular los datos de Domino. No soportan ningún evento o interacción de la interfase del usuario. Sin embargo, se puede combinar objetos del back-end con los del front-end en scripts UI. Por ejemplo, el objeto 'NotesUIDocument' tiene una propiedad llamada 'Document' que nos da acceso al documento subyacente. Estos son los objetos más importantes(no están incluidos los relacionados con XML):

Clase	***Definición***
NotesSession	Representa el entorno Domino del script actual, dando acceso a las variables de entorno, directorios de Domino, información sobre el usuario conectado, etc
NotesDbDirectory	Representa las bases de datos Domino en un servidor específico o la estación de trabajo.
NotesDatabase	Representa una base de datos Domino.
NotesACL	Representa la Lista de Control de Acceso (ACL) de una base de datos.
NotesACLEntry	Representa una sola entrada en la Lista de Control de Acceso. Una entrada puede ser para una persona, grupo o servidor.
NotesAgent	Representa un agente.
NotesView	Representa una vista o carpeta de una base de datos y da acceso a los documentos dentro de ella.
NotesViewColumn	Representa una columna en una vista o carpeta.
NotesDocumentCollection	Representa una colección de documentos de una base de datos, seleccionada de acuerdo a un criterio específico.
NotesDocument	Representa un documento en una base de datos.
NotesItem	Representa un trozo de información en un documento(habitualmente un campo). Todos los ítems en un documento son accesibles a través de LotusScript, sin importar qué formulario se esté usando para mostrar el documento en la interfase del usuario.
NotesRichTextItem	Representa un ítem de tipo texto enriquecido.
NotesRichTextStyle	Representa los atributos del campo de texto enriquecido.
NotesEmbeddedObject	Representa objetos incrustados, objetos enlazados y ficheros adjuntos.
NotesDateTime	Representa una fecha y hora. Permite realizar operaciones de comparación y actualización entre fechas.
NotesDateRange	Contiene un rango de NotesDateTime. Un objeto de tipo NotesDateTime representa una fecha y hora dadas.
NotesLog	Permite que se puedan registrar acciones y errores que tienen lugar durante la ejecución de scripts. Se pueden registrar las acciones y errores en una base de datos Notes,

	un correo memo o un fichero (para los scripts que se ejecutan localmente).
NotesNewsLetter	Representa un documento que contiene información de, o enlaces hacia otros documentos.
NotesForm	Representa un formulario en una base de datos Notes.
NotesInternational	Este objeto contiene propiedades que dan información sobre los parámetros internacionales, por ejemplo, formato de fecha del entorno en el que se está ejecutando Domino.
NotesName	Las propiedades de este objeto contienen información sobre el nombre de un usuario de Domino.
NotesTimer	Los objetos representan un temporizador en Domino.
NotesRegistration	Representa la creación o administración de un fichero ID.
NotesOutline	Representa los atributos de un esquema de Notes.
NotesOutlineEntry	Representa una entrada en un esquema de Notes.
NotesReplication	Representa los parámetros de replicación de una base de datos.
NotesRichTextParagraphStyle	Representa los atributos del párrafo RichText.
NotesRichTextTab	Representa los atributos de tabulación RichText.
NotesViewEntry	Representa una entrada de una vista. Esto es una fila en una vista.
NotesViewEntryCollection	Representa una colección de entradas de vistas, seleccionadas de acuerdo a un criterio específico.
NotesViewNavigator	Representa un navegador de una vista. Esto nos da acceso a todas o a un subconjunto de entradas en una vista.

15.- Jerarquía de Objetos

Hay una relación jerárquica para los objetos Domino. Los objetos más altos en la jerarquía contienen a los de abajo. La siguiente figura es un ejemplo de la relación jerárquica entre unos cuantos objetos de Domino:

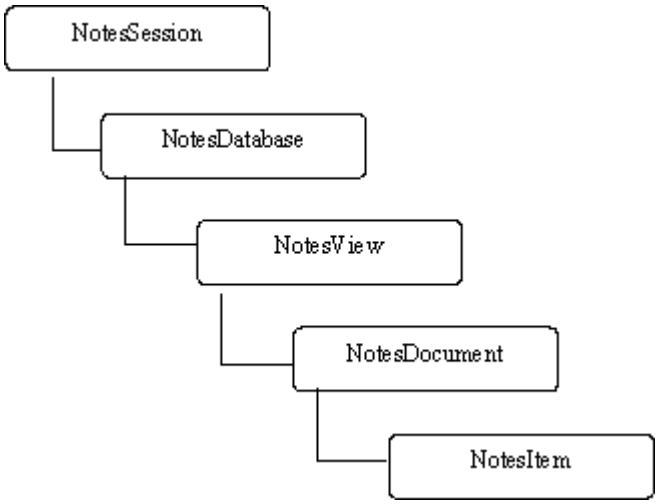

Fig. 2 - Jerarquía de Objetos Domino

Cada objeto tiene definidos miembros, propiedades y métodos. Usando estos miembros, se pueden acceder a otros objetos. La relación de contenido y acceso significa que el objeto superior tiene la propiedad o el método para acceder al inferior.

Por ejemplo, se puede acceder a todas las vistas cuando se abre una base de datos. Esto significa que la base de datos abierta (objeto) en el área de trabajo incluye las vistas (objeto). Aún más, se pueden ver los documentos cuando seleccionas uno de la vista. Esto significa que tu vista seleccionada (objeto) contiene los documentos (objeto). Esta jerarquía es importante cuando se usan objetos Domino. NotesSession es el objeto en la cima del modelo de objetos de Domino. Se puede acceder a cualquier objeto Domino si se empieza desde NotesSession.

Usando Objetos Domino con LotusScript

Ahora veremos un ejemplo de código que usan objetos en LotusScript para comprender el movimiento en la ruta del mapa del Modelo de Objetos de Domino:

Ejemplo:
Obteniendo el texto del campo Asunto

```
Dim ses as new NotesSession
Dim db as NotesDatabase
Dim view as NotesView
Dim doc as NotesDocument
Dim item as NotesItem
```

```
Set db = ses.CurrentDatabase
Set view = db.GetView("Todos")
Set doc = view.GetFirstDocument
Set item = doc.GetFirstItem("Asunto")
```

Primero se declara la variable *ses* como tipo *NotesSession*, y *New* se utiliza para crear una instancia de ese objeto.

Declaramos las variables *db, view, doc, item* como tipos *NotesDatabase, NotesView, NotesDocument* y *NotesItem*, respectivamente.

Para obtener el texto del campo *Asunto*, necesitamos seguir la ruta jerárquica desde la cima hasta el más bajo. En este ejemplo, vamos desde *NotesSession* hasta *NotesItem.*

Inicializamos la variable *db* con la propiedad *CurrentDatabase* del objeto de nivel superior.

Inicializamos la variable *view* usando el método *GetView*, dando el nombre de la vista.

Las siguientes sentencias son iguales a lo anterior: usamos los métodos *GetFirstDocument* para obtener el primer documento de la vista y *GetFirstItem* para obtener el campo *Asunto* del documento.

Comprendiendo las clases Front-end y Back-end

Primero, necesitas considerar cómo se guarda la información en Domino. Puedes pensar que un documento dentro de una base de datos Domino es un registro, pero un documento Domino es más sofisticado que un registro de una base de datos típica. Puede contener texto enriquecido, imágenes, objetos y otros tipos de información. Por ejemplo, si accedes a un documento Domino usando las clases back-end, puedes manipular el contenido de los campos, añadir nuevos campos al documento, o eliminar campos del documento. Sin embargo, si haces tales cambios, las fórmulas de traducción y de validación que existen en el formulario no se ejecutarán.

Por otro lado, si trabajas con objetos del front-end, tus cambios a los campos son visibles al usuario. Por ejemplo, si invocas el método *Refresh* de la clase *NotesUIDocument*, las fórmulas de traducción y de validación se ejecutarán.

La siguiente figura representa un documento en el back-end y muestra cómo los datos se almacenan en una base de datos Domino:

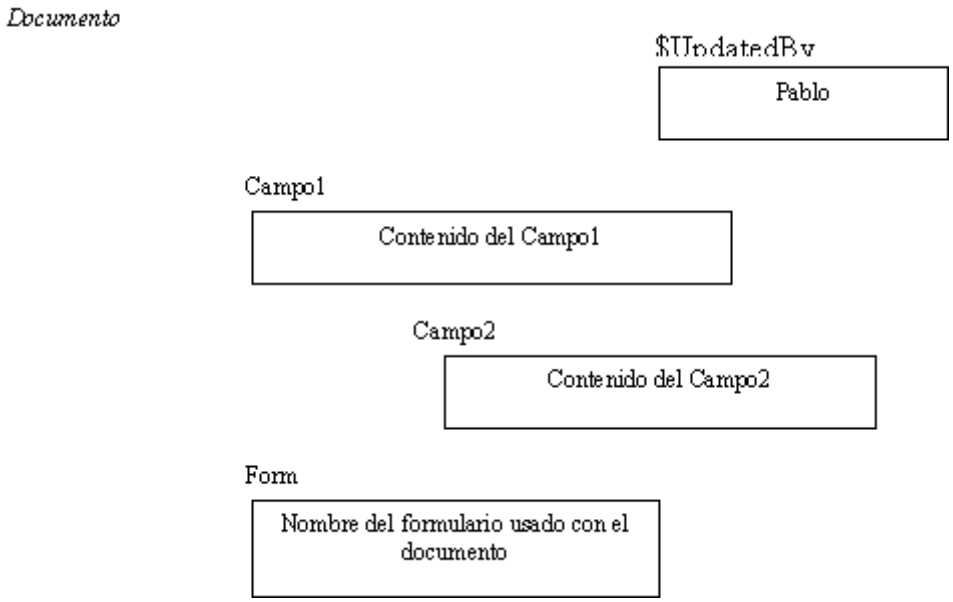

Fig. 3- Datos almacenados en una base de datos Domino

Los campos *Campo1* y *Campo2* han sido diseñados en el formulario que fue utilizado para crear este documento. El nombre de este formulario se almacena en el campo *Form*. Si cambias el valor del campo *Form* usando un agente o con LotusScript, el documento se presentará al usuario usando el otro formulario cuando se abra la siguiente vez.

Observación: Si no hay un campo *form* en un documento, Domino mostrará este documento usando el formulario predeterminado de la base de datos. Si no hay un formulario predeterminado, el documento no se puede mostrar.

El campo *$UpdatedBy* es un campo interno creado por Domino y contiene una lista de los usuarios que han trabajado con este documento.

Observación: En su mayoría, los campos que comienzan con $ los usa y mantiene Domino.

16.- Uso de los principales objetos Domino

Objeto Notessession: Es el objeto que contiene al resto. Representa el entorno de ejecución en el que se encuentra el usuario.
Se declara e instancia a la vez con la siguiente expresión:

Dim s as new notessession

El objeto **NotesSession** tiene un gran número de propiedades y métodos. Para una completa descripción de los mismos acceda a la ayuda de Domino Designer.

Objeto NotesDatabase: Hace referencia a una base de datos física existente en el sistema.
Se declara con la siguiente expresión:

Dim bd as notesdatabase

Para instanciarlo tenemos dos opciones:

a) Si se trata de la base de datos actual, en la que estamos trabajando, la expresión será:

   Set bd=s.currentDatabase (donde s es un objeto notessession que representa a la sesión actual)

b) Si queremos instanciar otra base de datos cualquiera la expresión será:

   Set bd=New NotesDatabase(servidor,rutaBD) (donde servidor, es el servidor donde reside la base de datos y rutaBD es la ruta física de la base de datos tomando como raíz el directorio data del servidor)

El objeto **NotesDatabase** tiene un gran número de propiedades y métodos. Para una completa descripción de los mismos acceda a la ayuda de Domino Designer.

Objeto NotesView: Hace referencia a un elemento de diseño de tipo vista. Recordemos que una vista es una especie de índice que contiene enlaces a documentos de la base de datos en función de la fórmula de selección que se haya determinado.
Se declara con la siguiente expresión:

Dim vista as notesview

Para instanciarlo tenemos dos opciones:

a) Hacerlo a partir del objeto de front-end notesuiview. La expresión quedaría entonces así:

   Set vista= uivista.view (donde uivista es el objeto de front-end que representa a la vista actual en pantalla)

b) Instanciar el objeto de back-end directamente a través del método getView perteneciente a la clase notesdatabase:

Set vista=bd.getView("Clientes") (donde bd es la base de datos que contiene a la vista que buscamos y "Clientes" es el nombre ó alias de la vista que deseamos instanciar)

El objeto NotesView es uno de los más importantes de Domino pues a través de él podemos acceder a los documentos del sistema que a la postre son los que contienen toda la información de una aplicación. Por ello vamos a ver como desplazarnos por los documentos de una vista.

Para hacer esto existen dos métodos en la clase NotesView, que son:

a) *Set doc=vista.getFirstDocument()* ---donde doc es un objeto NotesDocument (tal como se verá en el siguiente apartado) y vista es un objeto notesView. Con este método obtenemos el primer documento de una vista con vistas a ir recorriendo esta de manera secuencial
b) *Set docSig=vista.getNextDocument(doc)* ---donde docSig es el objeto NotesDocument que queremos obtener, doc es el objeto NotesDocument actual y vista es un objeto notesView. Con este método obtenemos el siguiente documento a doc(docSig) según el orden de la vista

El objeto **NotesView** tiene un gran número de propiedades y métodos. Para una completa descripción de los mismos acceda a la ayuda de Domino Designer.

Objeto NotesDocument: Representa a un documento en la base de datos Notes. Es la unidad principal de información en el entorno Notes/Domino.
Los documentos son creados normalmente por una acción del usuario aunque también pueden ser creados desde código. El campo Form del documento contendrá el nombre del formulario con el que fue creado.

Existen varias maneras de instanciar un documento:

a) Crearlo desde el propio código:

Set doc=New NotesDocument(bd) donde bd es el objeto que representa la base de datos física donde se almacenará el nuevo documento creado

b) Recuperarlo de una vista

Set doc=vista.getFirstDocument()
Set docSig=vista.getNextDocument(doc)

c) Recuperarlo de una colección de documentos

Set doc=col.getFirstDocument()

Set docSig=vista.getNextDocument(doc)

En este caso funcionan igual que al recuperarlos de una vista. Se obtiene el primer documento y luego se van obteniendo el resto, de manera secuencial, a partir del anterior.

Set docCualquiera=col.getNthDocumentt(pos)

Con la instrucción anterior, recuperamos un documento que se encuentra en la posición indicada por **pos** dentro del orden definido por la colección.

Otras acciones a realizar con documentos:

a) Salvar

Call doc.Save(a,b)

La acción necesita dos parámetros booleanos. (Valores posibles: true or false)

El primero nos pregunta si el documento se guardará aunque esté editado por otro usuario en ese momento. Si se indica true, el documento se guardará, si se indica false no se guardará.

El segundo parámetro depende del primero. Si el a es true, el valor de b es ignorado(no quiere decir que se pueda omitir). En caso de que a sea false se producirá la siguiente situación:

-Si b=true el documento se guardará como conflicto de replicación.
-Si b=false el documento no se guardará

b) Borrar

Call doc.Remove(a)

La acción necesita un parámetro booleano. (Valores posibles: true or false)

El parámetro decidirá si el documento se borrará aunque esté editado por otro usuario en ese momento. Si se indica true, el documento se borrará, si se indica false no se guardará.

c) Enviar

Call doc.Send(a)

La acción necesita un parámetro booleano. (Valores posibles: true or false)

El parámetro decidirá si el documento que se envía llevará incluido un formulario para su visualización. Si se indica true, el documento lo llevará, si se indica false no lo llevará.

Para acceder a los contenidos del documento(los campos) deberemos tener en cuenta lo siguiente:

a) El sistema trata a los campos de un documento como un array de múltiples valores.
b) La primera posición de ese array siempre es la 0
c) Para hacer referencia al contenido de un campo con un único valor tendremos que utilizar la siguiente notación:

   Cliente=Doc.Nombre(0) Esta sentencia carga en la variable cliente el contenido del campo Nombre que está en el documento doc

El objeto **NotesDocument** tiene un gran número de propiedades y métodos. Para una completa descripción de los mismos acceda a la ayuda de Domino Designer.

Objeto NotesDateTime: Representa una fecha

Acciones a realizar con objetos fecha:

a) Creación

   Para crear un nuevo objeto NotesDateTime hay que usar la siguiente sintaxis:

   *Set nuevaFecha= New NotesDatetime(**patronFecha**)*

   Donde **patronFecha** es un string, una fecha o cualquier patrón que el sistema reconozca como una fecha válida

b) Modificación

   Para modificar un objeto NotesDateTime utilizaremos los siguientes métodos del mismo.

   -*Call nuevaFecha.AdjustDay(**n**)* –Donde n ha de ser un número entero. Si n es positivo nuevaFecha se adelanta en el número de días que indique n. Si n es negativo nuevaFecha se retrasa en el número de días que indique n.

   -*Call nuevaFecha.AdjustMonth(**n**)* –Donde n ha de ser un número entero. Si n es positivo nuevaFecha se adelanta en el número de meses que indique n. Si n es negativo nuevaFecha se retrasa en el número de meses que indique n.

   -*Call nuevaFecha.AdjustYear(**n**)* –Donde n ha de ser un número entero. Si n es positivo nuevaFecha se adelanta en el número de años que indique n. Si n es negativo nuevaFecha se retrasa en el número de años que indique n.

   Existen métodos similares para modificar la parte horaria del objeto nuevaFecha. Consultar la ayuda de Domino Designer para obtener más información acerca de ellos.

c) Comparación entre fechas

Para comparar dos fechas se utiliza el método TimeDifference asociado a una de ellas. Por ejemplo para comparar dos objetos fecha, fecha1 y fecha2 la sintaxis sería la siguiente:

Diferencia=fecha1.TimeDifference(fecha2)

La ejecución de esta sentencia devuelve la **diferencia en segundos** que existe entre fecha1 y fecha2.
En este caso si fecha1 es menor que fecha2 diferencia tendrá un valor negativo mientras que si es al revés diferencia tendrá un valor positivo.
Si las dos fechas fueran idénticas el resultado sería 0

d) Obtener parte fecha sólo

Para obtener la parte fecha usaremos la siguiente sintaxis:

SoloFecha=nuevaFecha.DateOnly --El resultado devuelto y almacenado en soloFecha es un string

e) Obtener parte horaria solo

Para obtener la parte horaria usaremos la siguiente sintaxis:

SoloHora=nuevaFecha.TimeOnly --El resultado devuelto y almacenado en soloFecha es un string

El objeto **NotesDateTime** tiene un gran número de propiedades y métodos. Para una completa descripción de los mismos acceda a la ayuda de Domino Designer.

Objeto NotesUIWorkspace: Este es un objeto de UI-End y representa el entorno de trabajo en el que se está desarrollando la sesión Domino.

Para instanciar un objeto de este tipo deberemos usar la siguiente sintaxis:

Dim wk as new notesuivorkspace

El objeto NotesUIWorkspace nos será de gran utilidad cuando queramos trabajar con la información que tengamos en cada momento en pantalla

Las acciones más importantes a desarrollar con este objeto son:

a) Obtener el documento actual

Set uidoc=wk.CurrentDocument --En uidoc tenemos el documento en pantalla

Set doc=uidoc.document –En doc tenemos el documento de back-end al que representa el uidoc

b) Crear un nuevo documento en base a un formulario y mostrarlo

*Set uidoc= **wk.ComposeDocument(** [server$ [, file$ [, form$ [, windowWidth# [, windowHeight#]]]]] **)***

c) Editar o mostrar un documento de back-end

*Set uidoc= wk,**EditDocument(** [editMode [, notesDocument [, notesDocumentReadOnly [, documentanchor$]]]] **)***

d) Actualizar la vista actual en pantalla

Call vk.ViewRefresh

El objeto **NotesUIWorkspace** tiene un gran número de propiedades y métodos. Para una completa descripción de los mismos acceda a la ayuda de Domino Designer.

Objeto NotesUIDocument: Este es un objeto de UI-End y representa al documento activo en las sesión domino

Para ver como crear, editar o mostrar un documento de UI-End, repasar el objeto anterior NotesUIWorkspace.

Para acceder a los valores de un campo desde un objeto notesUIDocument usaremos la siguiente sintaxis:

Valor=uidoc.FieldGetText("Nombre") –En valor se almacena lo que contenga el campo Nombre en formato string

Para modificar los valores de un campo desde un objeto notesUIDocument usaremos la siguiente sintaxis:

Call uidoc. FieldSetText(nombreCampo, nuevoValor)–Donde nombreCampo es el campo que se quiere modificar(deber ir entrecomillado) y nuevoValor es el valor que se le asignará a dicho campo(también debe ir entrecomillado)

El objeto **NotesUIDocument** tiene un gran número de propiedades y métodos. Para una completa descripción de los mismos acceda a la ayuda de Domino Designer.