UNIDAD DIDÁCTICA 7
TRABAJANDO CON VISUAL BASIC

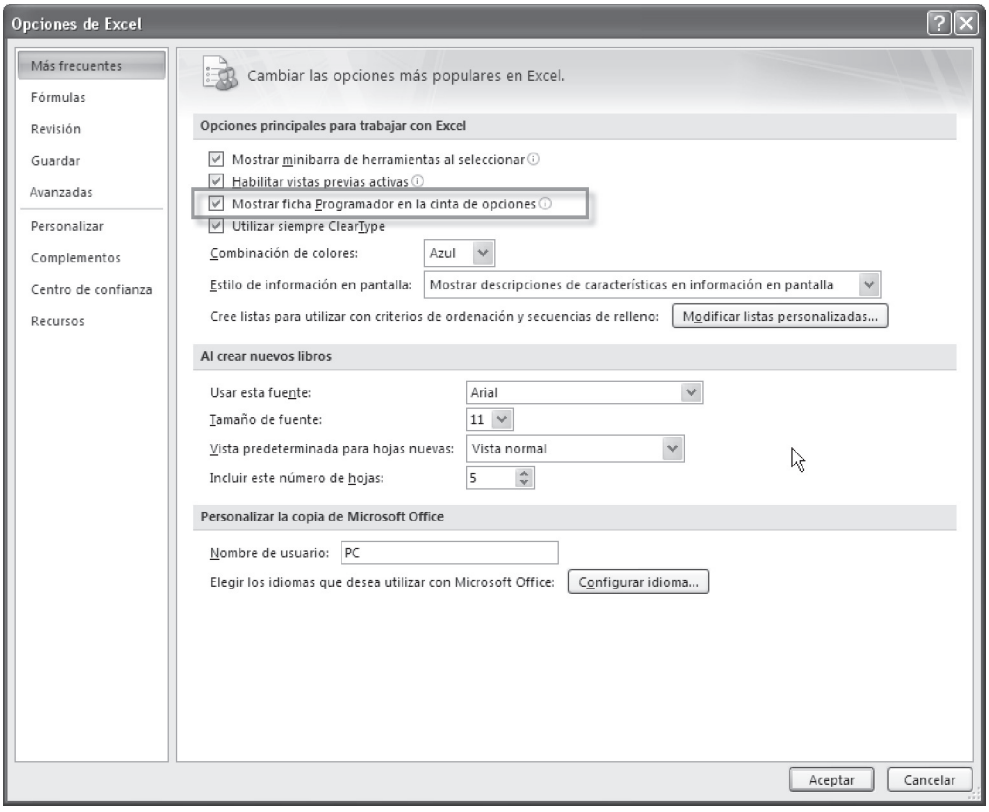
1. EL EDITOR DE VISUAL BASIC

Visual Basic para aplicaciones, es una combinación de un entorno de programación integrado denominado **Editor de Visual Basic** y del lenguaje de programación **Visual Basic**, permitiendo diseñar y desarrollar con facilidad programas en *Visual Basic*. El término *para aplicaciones* hace referencia al hecho de que el lenguaje de programación y las herramientas de desarrollo están integrados con las aplicaciones de **Microsoft Office**, en este caso, **Microsoft Excel**, de forma que se puedan desarrollar nuevas funcionalidades y soluciones a medida, con el uso de estas aplicaciones.

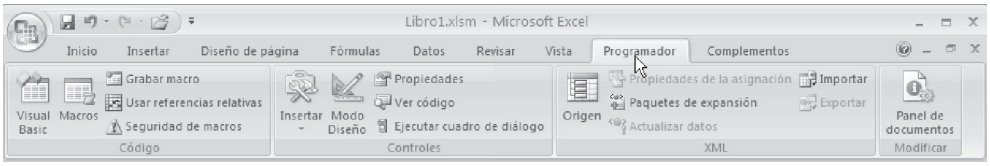
El **Editor de Visual Basic** contiene las herramientas de programación necesarias para escribir código en *Visual Basic* y crear soluciones personalizadas.

Este Editor, es una ventana independiente de *Excel*, y funciona igual para todas las aplicaciones de Microsoft Office. Cuando se cierre la aplicación, también se cerrará la ventana del Editor asociada.

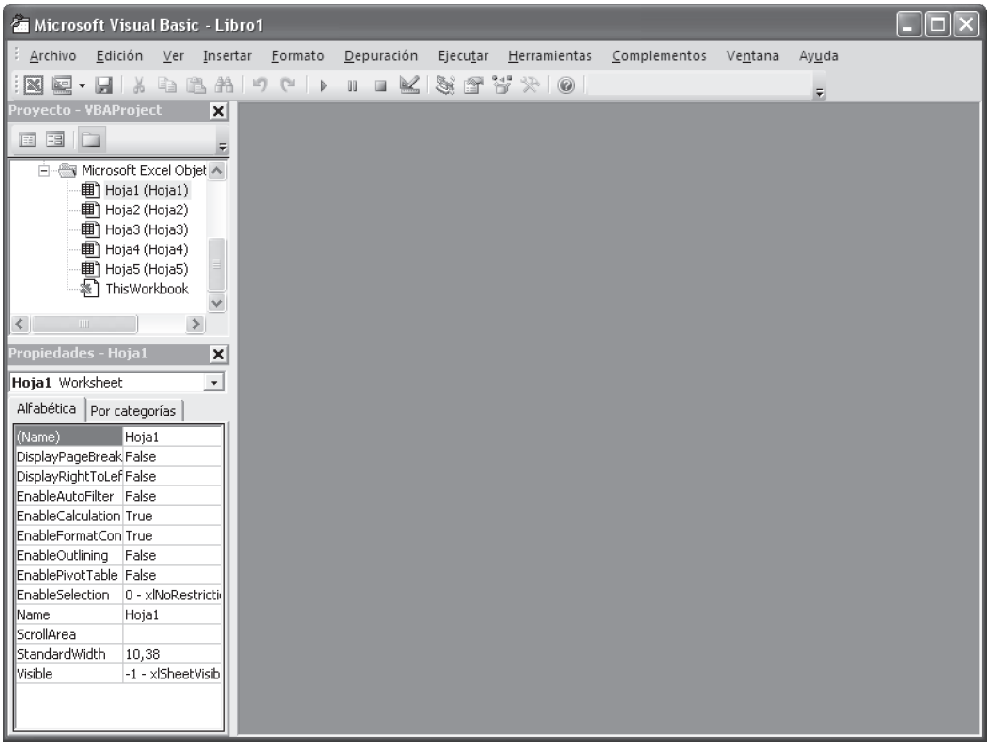
Antes de continuar, nos resultaría muy útil añadir una nueva ficha al programa. Se trata de la ficha **Programador**. Para ello, deberemos entrar a las opciones de *Excel* y activar la casilla **Mostrar ficha programador en la cinta de opciones**.



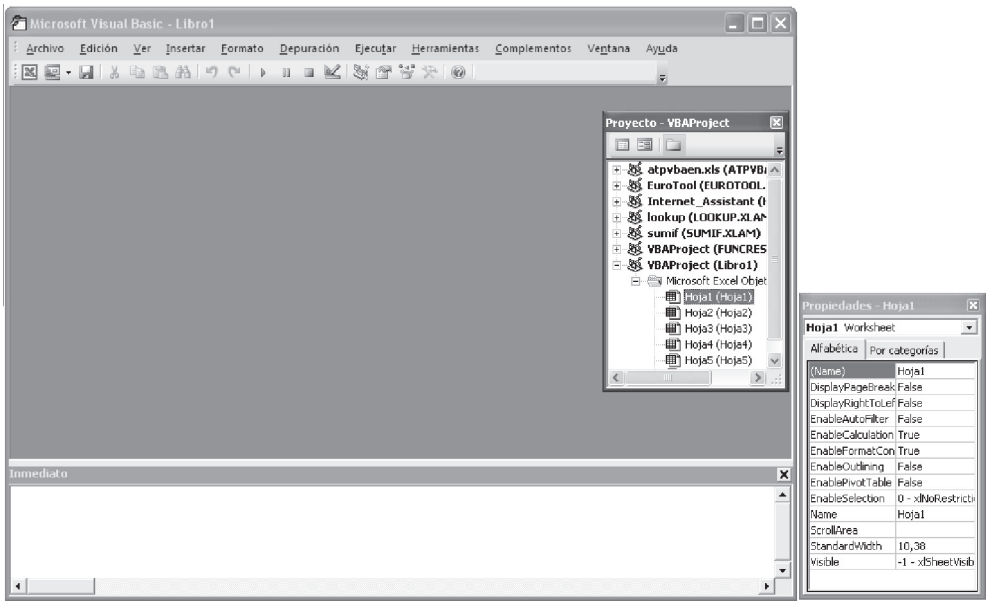
Una vez activada esta opción, tendremos una nueva ficha en el programa.



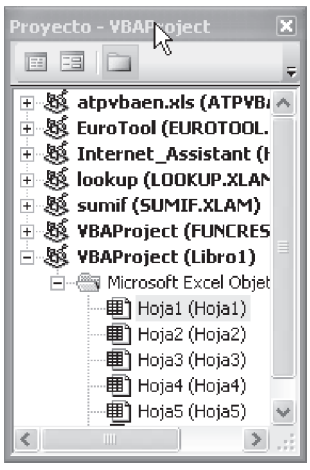
Para acceder al editor de *Visual Basic*, como ya leyó anteriormente, debemos pulsar el botón **Visual Basic** de la ficha **Programador**, o bien, pulsar la combinación de teclas **Alt + F11**. Se abrirá una ventana como la de la siguiente imagen.



En ella, podemos distinguir el menú de opciones y una barra de herramientas en la parte superior y dos paneles de la zona izquierda de la ventana. Estos paneles son el de proyecto y el de propiedades. Estos paneles los podemos mover a nuestro antojo, de forma que los podamos acoplar entre sí o desacoplar según nos resulte más cómodo.

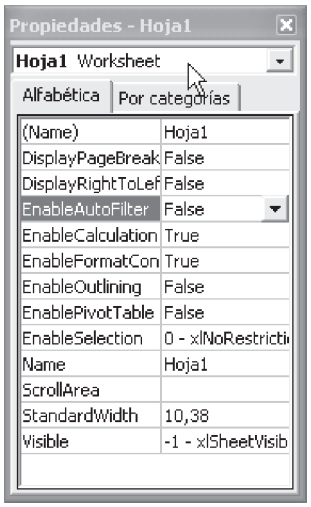


Aunque en nuestro caso vamos a utilizar el editor de *Visual Basic* para crear macros y funciones, también se podrá utilizar para crear proyectos más complejos, como módulos de complementos. Para gestionar estos módulos, tenemos el **Examinador de proyectos**.



Cada proyecto aparecerá en esta ventana. En esta lista, aparecen distintas carpetas conteniendo elementos como formularios, que son ventanas y cuadros de diálogo; módulos de código y módulos de clase, así como otros objetos relacionados con la aplicación.

Como hemos dicho anteriormente, cada elemento de *Excel* cuenta con una serie de propiedades. Éstas, sus nombres y valores actuales, aparecen la ventana de **Propiedades.**

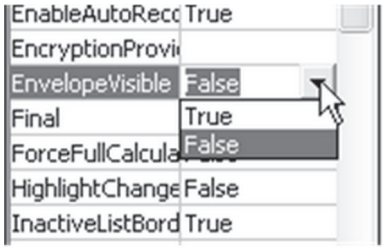


A esta ventana estaremos ligados cada vez que queramos *diseñar un nuevo formulario*, o *editar los existentes*. La ventana de propiedades enumera las propiedades de tiempo de diseño correspondientes a los objetos seleccionados y sus valores actuales. Nos permite cambiar estas propiedades en tiempo de diseño. Cuando selecciona múltiples controles, la ventana de Propiedades contiene una lista de las propiedades comunes a todos los controles seleccionados.

Las propiedades pueden estar ordenadas por orden alfabético o por categoría.

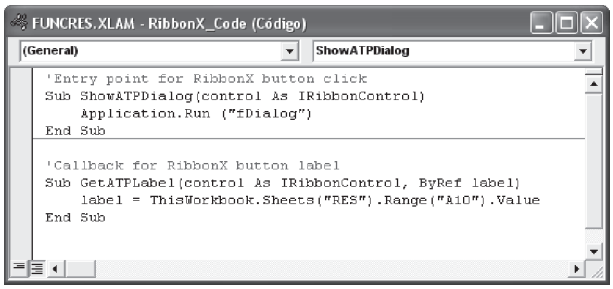
Los nombres de las propiedades aparecen en inglés, ya que en este momento no se encuentra en el entorno de *Excel* como usuario básico sino en un entorno de desarrollo y en calidad de usuario avanzado o, incluso programador.

Puede probar a modificar algunas propiedades del libro actual o cualquiera de sus hojas. Por ejemplo, puede asignar el valor **True** a la propiedad **EnvelopeVisible**, verá cómo se muestran en la hoja actual los elementos necesarios para enviarla por correo electrónico. Pruebe también a modificar la propiedad **Name** de cualquiera de las hojas y observe las pestañas que hay en la parte inferior del libro, verá cómo cambian.

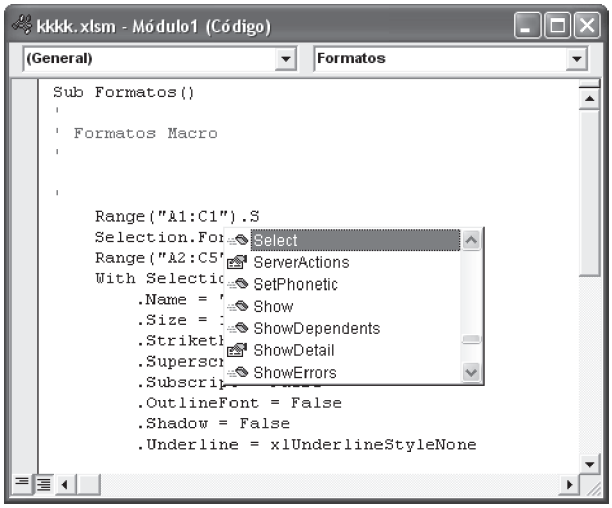


2. EL EDITOR DE CÓDIGO

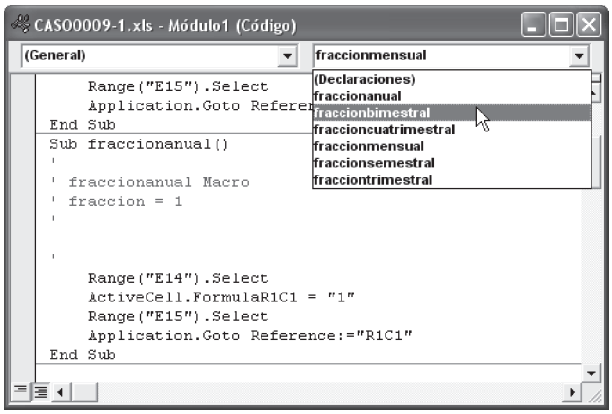
Al hacer doble clic sobre cualquiera de los elementos del **Examinador de proyectos**, se abrirá el editor de código, también lo abriremos pulsando la tecla **F7** o desplegando el menú **Ver** y seleccionando la opción **Código**. Este editor es como cualquier editor de texto, con la diferencia que en él introduciremos las ordenes que forman parte de las funciones y macros.



Un aspecto importante del editor, es la lista que aparece automáticamente al introducir el nombre de un objeto. En esta lista aparecen todas las propiedades y métodos del objeto, facilitando su selección y haciendo innecesaria la memorización de tantos elementos.



Al escribir las órdenes en él, nos damos cuenta que dependiendo de lo que escribamos, el texto tomará un color u otro. En principio, los comentarios aparecen en verde, las palabras reservadas de *Visual Basic* en azul, los objetos y variables en negro, y los errores en rojo.



Observe las dos cajas de la parte superior. A estas cajas les denominamos **Lista de Objetos** y **Lista de Procedimientos,** respectivamente. Haciendo clic sobre la flecha de cada lista, se despliega un menú, en la lista de objetos se desplegará una lista con los nombres de cada objeto existente en ese momento dentro del formulario. Haciendo clic sobre uno de los nombres, nos presentará la ventana de código de ese objeto. Todos los objetos gráficos (controles) existentes dentro de un formulario y el propio formulario aparecerán en la misma lista de objetos.

Haciendo clic sobre la flecha de la lista de procedimientos, se despliega la lista con todos los procedimientos o funciones posibles para ese objeto. Siempre saldrá uno. Si tenemos escrito código en uno de los procedimientos, saldrá por defecto ese procedimiento para el cual hemos escrito el código. Si no hay código en ninguno de los procedimientos, saldrá el que tenga por defecto cada objeto.

Solamente nos queda por decir qué es un procedimiento.

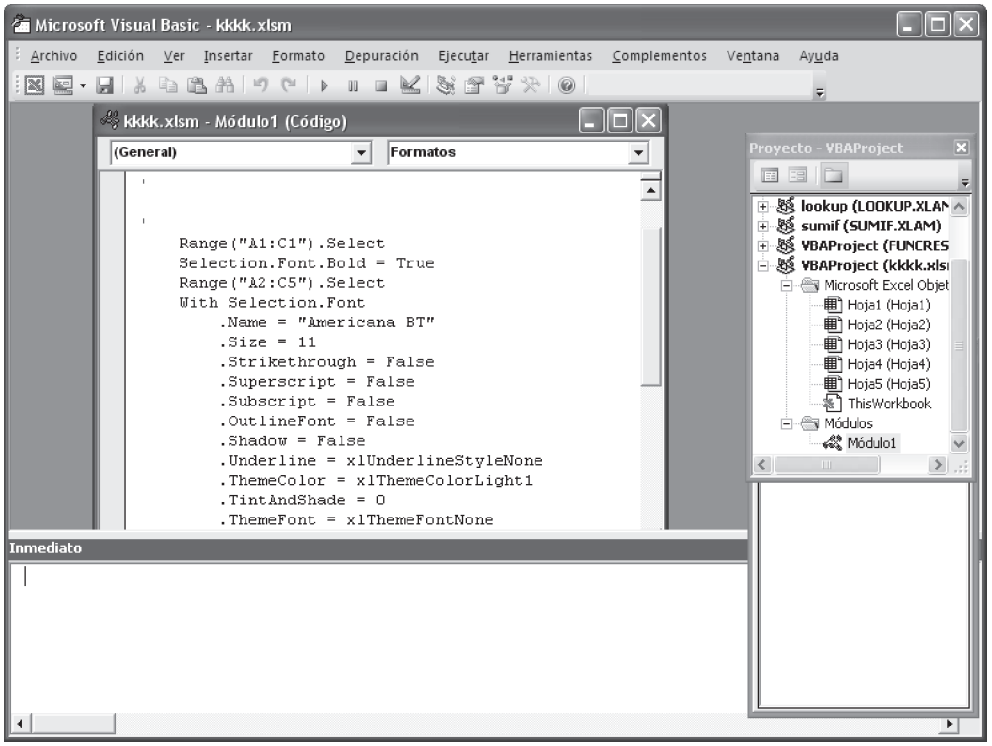
Para ello vamos a explicar lo que es un evento. Un **Evento** es algo que le puede ocurrir a un objeto. En una interface gráfica, lo que le puede ocurrir a un objeto es que se le haga clic, doble clic, que se pase el cursor del ratón por encima,

etc. Este es el **Evento**. El **Procedimiento** es la respuesta por parte de ese objeto, al evento que le está sucediendo.

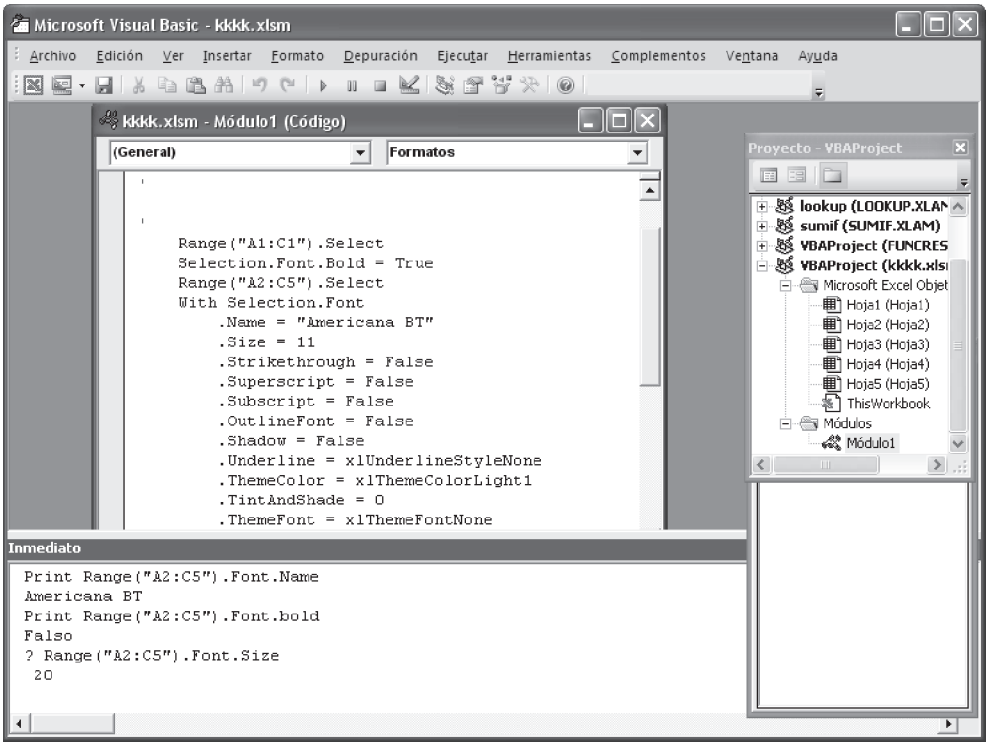
Esa respuesta, esa forma de ***Proceder*** del objeto al evento que le está sucediendo, debemos programarla según nuestras necesidades, es decir, debemos escribir el código que necesite nuestra aplicación como respuesta al evento que acaba de ocurrir. Posiblemente, no queramos ninguna respuesta a muchos de los eventos que pueden acaecer a un objeto. Cada objeto tiene muchos eventos y solamente queremos aprovechar los que nos interesan. Para que un evento no produzca ningún efecto, basta con dejar sin código el procedimiento correspondiente a ese evento. En los eventos que queramos que realice alguna operación, le escribiremos en su procedimiento el código necesario para que esa operación se realice.

3. LA VENTANA INMEDIATO

La mayoría de las veces, para encontrar la causa de un problema mientras ejecutamos paso a paso una parte del código de nuestra macro, necesitamos asignar valores, evaluar expresiones y verificar procedimientos. Este tipo de operaciones son las que nos permite realizar la ventana Inmediato.



Para visualizar valores en esta ventana hay dos formas: utilizar la sentencia **Print** o **?** directamente sobre ésta ventana para escribir los valores actuales de las variables y propiedades deseadas, o colocar sentencias **Debug.Print** en el código de la aplicación.



Una vez que ha entrado en la ventana inmediata, haciendo clic sobre ella o ejecutando la orden **ventana Inmediato** del menú **Ver,** para moverse dentro ella utilice las teclas convencionales de cualquier procesador de textos (teclas de movimiento del cursor, página arriba, página abajo, suprimir, etc.)

También es posible asignar valores, evaluar expresiones y verificar resultados de procedimientos.

4. VARIABLES, CONSTANTES Y TIPOS DE DATOS

Cuando se programa, a menudo es necesario almacenar valores para realizar determinados cálculos, para ello utilizaremos las variables. Estas variables tendrán definido un tipo de dato, que determina la clase de datos que puede almacenar la variable.

VARIABLES

En *Visual Basic* puede utilizar variables para almacenar valores durante la ejecución del programa. Las variables se dividen en:

- **Nombre:** palabra que identifica el valor que contiene la variable.
- **Tipo de dato:** determina la clase de datos que la variable puede almacenar.

Para declarar una variable lo haremos mediante la instrucción **Dim** seguido del nombre y el tipo de dato. Su sintaxis es:

Dim nombre de variable **As** tipo

Cuando declaremos variables, hemos de tener en cuenta las siguientes reglas:

- El nombre de la variable ha de comenzar con una letra.
- No se pueden utilizar puntos o caracteres de declaración de tipo.
- El nombre no ha de sobrepasar los 255 caracteres.
- Debe tener un único nombre dentro del mismo alcance de la variable.

Las variables pueden ser declaradas de forma implícita o explícita.

Declaración implícita: permite utilizar una variable sin haberla declarado, y aunque es más cómodo, también conlleva errores, por ejemplo, veamos este código:

Var 1= 34

Var 2= 26

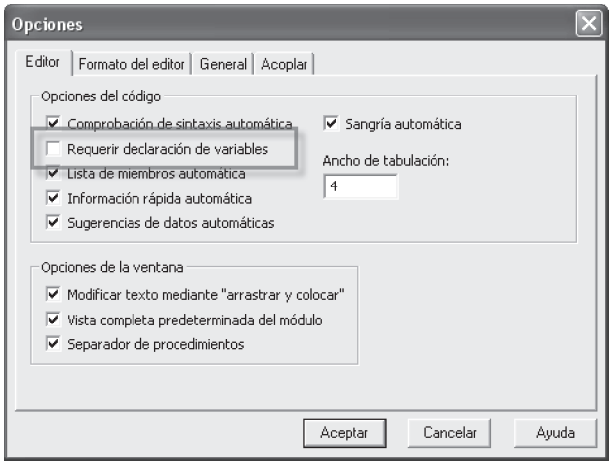
Var3= Var + Var 2

A simple vista el código parece correcto, sin embargo provocará un error que *Visual* no detectará, es decir la suma será errónea porque hemos escrito Var en lugar de Var1.

Declaración Explícita: con este tipo de declaración, *Visual* provocará un error si se hace uso de una variable que no ha sido definida.

Disponemos de dos formas de definir la declaración explícita:

1. Incluir esta instrucción en la sección Declaraciones del módulo de clase, formulario o estándar: *Option Explicit*
2. Y la otra manera, es a través de los menús de *Visual Basic*, en el apartado **Herramientas - Opciones**, y activando la opción '**Requerir declaración de variables**'



CONSTANTES

Casi siempre los códigos contienen valores constantes que reaparecen una y otra vez, es decir números que por sí mismos, no tienen un significado obvio.

Una constante es un nombre significativo que sustituye a un número o cadena que no varía. La diferencia que existe con una variable, es que una constante no puede modificarla o asignarle un nuevo valor. Pueden ser:

- Definidas por el sistema proporcionadas por aplicaciones y controles. Se denominan **constantes 'intrínsecas'**.
- Definidas por el usuario, se declaran mediante la instrucción ***Const***. Se denominan **constantes 'simbólicas'**.

La forma de definir nuestras constantes es:

[Public /Private] Const nombre_de_constante **As** tipo = expresión

TIPOS DE DATOS

El tipo de dato de una variable indica cómo se almacenan los bits que representan esos valores en la memoria del ordenador. *Visual Basic* dispone de un tipo de datos predeterminado, el tipo de datos **Variant**. Este tipo de datos es como un comodín, puede representar diferentes tipos de datos en distintas situaciones.

Pero para que *Visual Basic* trate de forma más eficiente los datos, es preferible utilizar un tipo de datos determinado.

En las declaraciones de variables con tipos de datos se utilizan las instrucciones **Private, Dim, Public o Static**. Algunos ejemplos son:

Dim variable1 **As** String

Public variable3 **As** Double

Private variable2 **As** Integer

También es posible combinar diferentes tipos en la declaración de variables:

Dim Nombre **As** String, valor **As** Integer

Los tipos de datos numéricos que proporciona Visual Basic son:

- **Integer:** Variable entera de 2 bytes, su rango es de -32768 a 32767. El carácter de declaración de tipo para el tipo Integer es el signo de porcentaje (%)
- **Long**: Variable entero largo de 4 bytes , su rango es de -2.147.483.648 a 2.147.483.647. El carácter de declaración de tipo para Long es el signo &.
- **Single:** Variable real simple de 4 bytes, su rango es de -3.40E+308 a 1.79+308. El carácter de declaración de tipo para Single es el signo i.
- **Double**: Variable real doble precisión de 8 bytes, su rango es de 1,79769313486232E308 a -4,94065645841247E-324 para valores negativos y de 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos. El carácter de declaración de tipo para Double es el signo de número (#).
- **Currency:** Variable con punto decimal fino de 8 bytes, su rango es de 922.337.203.685.477.5807 a. -922.337.203.685.477,5808 el carácter de declaración de tipo para currency es el signo @.

Otros tipos de datos no numéricos son:

- **String:** Emplee este tipo de dato cuando utilice una variable que siempre contenga una cadena y nunca un valor numérico. Es una cadena de longitud variable, es decir, que crece o disminuye según le asigne nuevos datos. Para definir la variable String de longitud fija debemos escribir:

Dim Nombre **As** String * 20

- **Byte:** Se utiliza cuando la variable contenga datos binarios. Todos los operadores que funcionan con enteros son válidos con el tipo Byte excepto la **resta unaria**, ya que Byte es un tipo sin signo con el rango de 0 a 255 y no puede representar valores negativos.
- **Boolean:** Utilice este tipo de dato cuando trabaje con variables que contengan solamente información de verdadero o falso, sí o no, o activado o desactivado. El valor predeterminado de un variable declarada como Boolean es False.
- **Date:** Cuando necesite almacenar valores de fecha y hora, utilice el dato Date. Cuando se convierten otros tipos de datos numéricos en Date, los

valores que hay a la izquierda del signo decimal representan la información de fecha, mientras que los que haya a la derecha representan la hora.

CONVERSIONES DE TIPO

Frecuentemente, nos encontramos con el problema de tener que pasar un dato de cierto tipo a otro. Hay variables que permiten pasar datos entre ellos, aunque si es de una variable de mayor capacidad a una de menor capacidad, los datos se recortarán para que quepan en la de menor capacidad. Para convertir una expresión en un tipo de datos específico, existen una serie de funciones que pasamos a ver a continuación:

- **CInt(expresión):** Convierte el resultado de una expresión de tipo Integer.

  Ejemplo: Resultado = CInt(345,645)

  'Resultado está declarada como Integer = 346

- **CLng(expresión):** Convierte el resultado de una expresión de tipo Long.

  Ejemplo: Resultado = CLng(321798,23)

  'Resultado está declarada como Long = 321798

- **CBool(expresión):** Convierte el resultado de una expresión de tipo Bolean.

  Ejemplo: Resultado = CBool(4 = 5)

  'Resultado declarada como Boolean = False

- **CByte(expresión):** Convierte el resultado de una expresión de tipo Byte.

  Ejemplo: Resultado = CByte(35,6912)

  'Resultado está declarada como Byte = 36

- **CCur(expresión):** Convierte el resultado de una expresión de tipo Currency.

  Ejemplo: Resultado = CCur(852,567476)

  'Resultado está declarada como Currrency = 852,5674

- **CDate(expresión):** Convierte el resultado de una expresión de tipo Date.

  Ejemplo: Resultado = CDate(«12/10/2000»)

  'Resultado está declarada como Date = 12/10/2000

- **CSng(expresión):** Convierte el resultado de una expresión de tipo Single.

  Ejemplo: Resultado = CSng(12,678432)

- **CDbl(expresión):** Convierte el resultado de una expresión de tipo Double.

  Ejemplo: Resultado = CDbl(524,245*12,4)

  'Resultado está declarada como Double = 6500,638

- **CVar(expresión):** Convierte el resultado de una expresión de tipo Variant.

  Ejemplo: Resultado = CVar(12 & «avo»)

  'Resultado está declarada como Variant = «12avo»

- **CStr(expresión):** Convierte el resultado de una expresión de tipo String.

  Ejemplo: Resultado = CStr(275)

  'Resultado está declarada como String = «275»

El alcance de una variable define qué partes del código son conscientes de su existencia.

Cuando declara una variable en un procedimiento, sólo el código de dicho procedimiento puede tener acceso o modificar el valor de la variable; tiene un alcance que es local al procedimiento. A veces, sin embargo, se necesita utilizar una variable con un alcance más general, como aquella cuyo valor está disponible para todos los procedimientos de toda la aplicación. *Visual Basic* le permite especificar el alcance de una variable cuando la declara.

Si se declara la variable a nivel de procedimiento puede ser:

- **Privado**: Las variables son privadas del procedimiento en el que aparecen.
- **Público**: No es aplicable. No se pueden declarar variables públicas dentro de un procedimiento.

Y a nivel de módulo:

- **Privado:** Las variables son privadas del módulo en el que aparecen.
- **Público:** La variables están disponibles para todos los módulos.

Las variables a nivel de procedimiento se las conoce como variables locales. Se declaran mediante las palabras clave **Dim** o **Static.** Un ejemplo de estas son:

Dim temporal **As** Integer

Static permanente **As** Integer

Los valores de las variables locales declaradas con **Static** existen mientras se ejecuta la aplicación, mientras que las variables declaradas con **Dim** sólo existen mientras se ejecuta el procedimiento.

Las variables locales declaradas con **Dim** tienen la ventaja de liberar su espacio de memoria cuando termina el procedimiento, y además no interfiere con otra variable que tenga el mismo nombre pero en otro procedimiento.

Las variables declaradas a nivel de módulo están disponibles para todos los procedimientos del módulo, pero no para el código de otros módulos. La forma de declararla sería:

Private variable **As** String

A nivel de módulo, no hay diferencia entre escribir **Private** y **Dim.**

Para hacer que una variable a nivel de módulo esté a disposición de otros módulos, se ha de utilizar la palabra **Public** para declarar la variable. Su sintaxis sería:

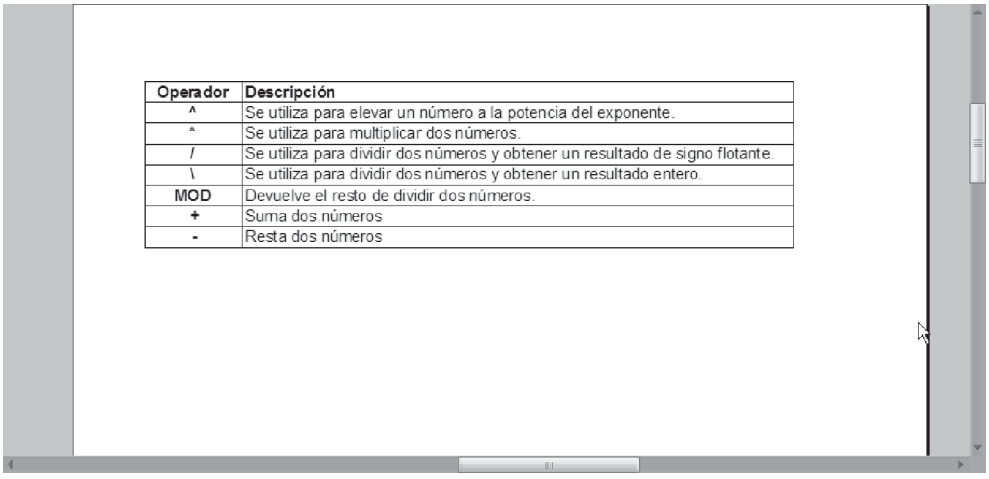
Public variable **As** String

Por defecto, las variables declaradas en un módulo son **Private**.

5. LOS OPERADORES EN VBA

Cuando trabajamos con datos suministrados por el usuario, por norma general debemos tratar esos datos, ya sea realizando cálculos aritméticos, comparaciones, etc. *Visual Basic* nos ofrece una serie de operadores que nos sirven para tratar esos datos. Sus tipos y operadores son:

OPERADORES ARITMÉTICOS



Operador	Descripción
^	Se utiliza para elevar un número a la potencia del exponente.
*	Se utiliza para multiplicar dos números.
/	Se utiliza para dividir dos números y obtener un resultado de signo flotante.
\	Se utiliza para dividir dos números y obtener un resultado entero.
MOD	Devuelve el resto de dividir dos números.
+	Suma dos números
-	Resta dos números

Ejemplo:

```
Dim Valor
...
Valor = 2 ^ 3 ' Devuelve 8
Valor = 2 * 2 ' Devuelve 4
```

Valor = 3 / 2 ' Devuelve 1,5

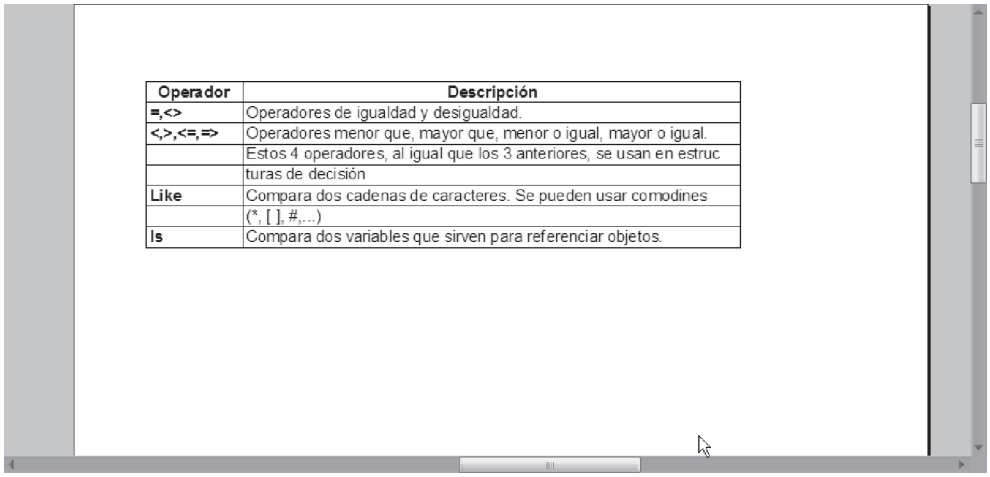
Valor = 2 \ 2 ' Devuelve 1

Valor = 7 Mod 3 ' Devuelve 1

Valor = 9 + 6 ' Devuelve 15

Valor = 8 - 3 ' Devuelve 5

OPERADORES DE COMPARACIÓN



Operador	Descripción
=,<>	Operadores de igualdad y desigualdad.
<,>,<=,=>	Operadores menor que, mayor que, menor o igual, mayor o igual.
	Estos 4 operadores, al igual que los 3 anteriores, se usan en estruc
	turas de decisión
Like	Compara dos cadenas de caracteres. Se pueden usar comodines
	(*, [ ], #,...)
Is	Compara dos variables que sirven para referenciar objetos.

Ejemplo:

Dim Valor

...

if valor <> 5 then 'si la variable 'valor' es distinta de 5, se 'ejecutan las expresiones ligadas a

'la sentencia if

Valor = "abba" like "abc" 'Devuelve False

Dim Objeto1, Objeto2, Objeto3, Resultado

...

```
Set Objeto2 = Objeto1      'Sirve para asignar referencias de 'objeto.
...
Resultado = Objeto1 Is Objeto2 ' Devuelve True.
Resultado = Objeto3 Is Objeto2 ' Devuelve False.
'se asume que Objeto1 <> Objeto3
```

OPERADORES LÓGICOS

Sólo sirven con valores booleanos.

Operador	Descripción
Not	Convierte el valor de la variable en su opuesto.
And	Realiza una conjunción lógica.

VALOR1	VALOR2	RESULTADO
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
TRUE	NULL	NULL
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE
FALSE	NULL	FALSE
NULL	TRUE	NULL
NULL	FALSE	FALSE

Or Realiza una disyunción lógica.

VALOR1	VALOR2	RESULTADO
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	NULL	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE
FALSE	NULL	NULL
NULL	TRUE	TRUE
NULL	FALSE	NULL

Xor Realiza una exclusión lógica.

VALOR1	VALOR2	RESULTADO
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Eqv Efectúa una equivalencia lógica

VALOR1	VALOR2	RESULTADO
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	TRUE

Imp Efectúa una implicación lógica.

VALOR1	VALOR2	RESULTADO
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
TRUE	NULL	NULL
FALSE	TRUE	TRUE
FALSE	FALSE	TRUE
FALSE	NULL	TRUE
NULL	TRUE	TRUE
NULL	FALSE	NULL

Ejemplo:

```
Dim A, B, C, D, Resultado
A = Null: B = 2: C = 3: D = 4 ' Inicialización de variables.
Resultado = Not D > C ' Devuelve False.
```

Resultado = D > C And B > C ' Devuelve False

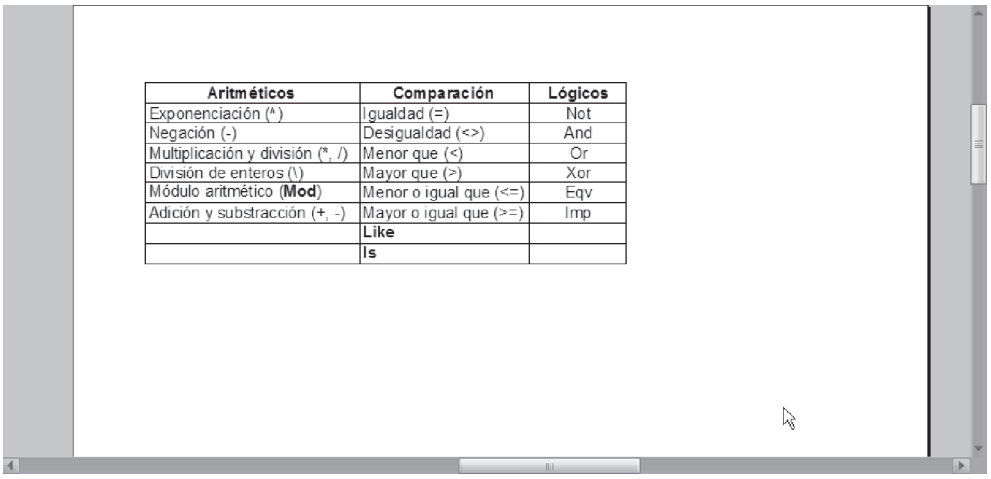
Resultado = D > C Or B > C ' Devuelve True.

Resultado = D > C Xor C > B ' Devuelve False.

Resultado = D > C Eqv C > B ' Devuelve True.

Resultado = C > D Imp C > B ' Devuelve True.

Cuando hay expresiones que contienen operadores de más de una categoría, se resuelven antes las que tienen operadores aritméticos, a continuación las que tienen operadores de comparación y por último las de operadores lógicos. Los operadores de comparación tienen todos la misma prioridad, es decir, se evalúan de izquierda a derecha, en el orden en que aparecen. Los operadores lógicos y aritméticos se evalúan en el siguiente orden de prioridad (de mayor a menor):



Aritméticos	Comparación	Lógicos
Exponenciación (^)	Igualdad (=)	Not
Negación (-)	Desigualdad (<>)	And
Multiplicación y división (*, /)	Menor que (<)	Or
División de enteros (\)	Mayor que (>)	Xor
Módulo aritmético (**Mod**)	Menor o igual que (<=)	Eqv
Adición y substracción (+, -)	Mayor o igual que (>=)	Imp
	Like	
	Is	

6. MATRICES

Las matrices le permiten hacer referencia por el mismo nombre a una serie de variables y utilizar un número (índice) para distinguirlas. Esto ayuda a crear código más pequeño y simple en muchas ocasiones ya que puede establecer bucles que traten de forma eficiente cualquier número de casos mediante el número del índice. Las matrices tienen un límite superior e inferior y los elementos de la matriz son contiguos dentro de esos límites.

Puesto que *Visual Basic* asigna espacio para cada número de índice, evite declarar las matrices más grandes de lo necesario.

Todos los elementos de una matriz tienen el mismo tipo de datos.

Si es de tipo **Variant**, los elementos individuales pueden contener distintas clases de datos (objetos, cadenas, números, etc.). Puede declarar una matriz de cualquier de los tipos de datos fundamentales, incluyendo los tipos definidos por él y variables de objetos. En *Visual Basic* hay dos tipos de matrices: las matrices de tamaño fijo que tienen siempre el mismo tamaño y las matrices dinámicas cuyo tamaño cambia en tiempo de ejecución.

En la línea siguiente, por ejemplo declaramos una variable que contará con diez elementos, numerados de 1 al 10, de tipo **Variant**.

```
Dim Celdillas(10) As Variant
```

La variable **Celdillas** sería como un rango de celdillas que están en una sola fila o una columna de la hoja de cálculo. Si el rango tiene varias filas y varias columnas, el equivalente sería una matriz bidimensional, es decir, con dos dimensiones:

```
Dim Celdillas (10,5) As Variant
```

Una vez declarada la matriz, para poder leer o modificar cualquiera de los valores, es necesario facilitar el índice o índices apropiados. Es un mecanismo

similar al usado con la función **Indice**, para obtener datos de una tabla. En este caso, la tabla no es un rango de celdillas sino una variable, pero el funcionamiento es equivalente.

Teniendo declarada la matriz con dos dimensiones, podríamos obtener un valor de uno de los elementos, de la siguiente forma.

Debug.Print Celdillas(5,1)

En este caso y asumiendo que el primer índice fuese la fila y el segundo la columna, estaríamos recuperando el valor almacenado en la quita fila de la primera columna. En una hoja de cálculo la referencia equivalente es nA5**.**

7. SENTENCIAS CONDICIONALES

Estas estructuras son las encargadas de comprobar condiciones, dependiendo de los resultados, realizar diferentes operaciones.

Veamos la primera de ellas:

ESTRUCTURA IF....THEN

Ejecuta una o más instrucciones dependiendo de una condición. La sintaxis de esta estructura es la siguiente:

If condición **Then** instrucción

Por ejemplo, vamos a comprobar si el valor de la celda activa, si la condición es cierta, pondremos el contenido de la celda en negrita:

```
If ActiveCell.Value > 1000 Then ActiveCell.Font.Bold=True
```

Si debemos ejecutar más de una instrucción, si la condición es cierta, la sintaxis varía a la forma:

```
If condición Then
        Instrucciones...
End If
```

Aunque es preferible usar esta última en cualquier caso, para una mejor legibilidad del código.

ESTRUCTURA IF ..... THEN .....ELSE

Esta estructura se utiliza para definir varios bloques de instrucciones, donde sólo uno de los cuales se ejecutará, veamos la sintaxis de esta estructura:

```
If Condición1 Then
        Instrucciones...
ElseIf Condición2 Then
        Instrucciones...
Else
        Instrucciones...
End If
```

Primero se evalúa la *condición1*, si es **False**, se procede a evaluar la *condición2* y así sucesivamente, hasta que se encuentre una condición **True**. Una vez se encuentra una condición **True**, se ejecuta el bloque de instrucciones y el código que se encuentre después de la sentencia **End If.**

No obstante, podemos incluir un bloque de instrucciones **Else,** que se ejecutará en caso de que todas las condiciones tengan el valor False.

En este ejemplo, comprobaremos que si el valor introducido en la celda activa es mayor a 1000, el contenido se mostrará en negrita y si es inferior aparecerá en cursiva.

```
If ActiveCell.Value > 1000 Then
        ActiveCell.Font.Bold=True
Else
        ActiveCell.Fond.Italic=True
End If
```

ESTRUCTURA SELECT CASE

Esta estructura se utiliza como alternativa a la estructura **If...Then...Else**. Se usa cuando evaluamos muchos valores distintos de una misma expresión, puesto que con **Select**

Case la estructura queda de forma más clara. Su sintaxis es:

```
Select Case expresión
        Case lista_de _expresiones1
                Instrucciones...
        Case lista_de_expresiones2
                Instrucciones....
        ...
        Case Else
                Instrucciones...
        End Select
```

Cada lista_de_expresiones es una lista de uno o más valores. Si hay más de un valor, se separan con comas. Cada Instrucciones contienen cero o más instrucciones. En caso de que más de un **Case** coincida con la expresión, sólo se ejecutará la primera de ellas. Por último, la cláusula **Case Else** se ejecutará en caso de que ningún valor de la lista de expresiones coincida.

En este otro ejemplo, que podemos ver a continuación, realizamos una operación booleana con las variables A y B, según sea el valor de la variable OPCION.

```
Dim A, B, OPCION, RESULTADO

A = 10

B = 5

        Select Case OPCION

                Case 1

                'Si la variable OPCION tiene el valor 1, se realiza

                'la suma de las variables A y B

                        RESULTADO = A+B

                Case 2

                'Si la variable OPCION tiene el valor 2, se realiza

                'la resta de las variables A y B

                        RESULTADO = A-B

                Case 3

                'Si la variable OPCION tiene el valor 3, se realiza

                'la multiplicación de las variables A y B

                        RESULTADO = A*B

                Case Else

                 'En caso que la variable OPCION no tenga un valor
                 'que concuerde con alguno de los anteriores,
                 'inicializa las variables

                        A = 0

                        B = 0

                End Select
```

8. SENTENCIAS DE REPETICIÓN

Las estructuras de bucle se utilizan para ejecutar una o más líneas de código repetidamente. Disponemos de varias estructuras de bucles, veamos cuales son:

DO...LOOP

El bucle **Do...Loop** permite ejecutar un bloque de instrucciones un número indefinido de veces. Su sintaxis es:

```
Do While condición
        Instrucciones
Loop
```

Este bucle primero evalúa la condición. Si es **False**, se salta todas las instrucciones, en caso contrario (**True**) se ejecutan las instrucciones y vuelve a la instrucción **Do While** y a probar de nuevo la condición. Nunca se ejecutarán las instrucciones si condición es **False** inicialmente.

Por lo tanto, este bucle se puede ejecutar cualquier número de veces, siempre y cuando condición no sea falsa (distinta de 0).

Existen dos variantes del bucle **Do..Loop**, éstas son las siguientes:

Do..Until..Loop Se ejecuta cero o más veces.

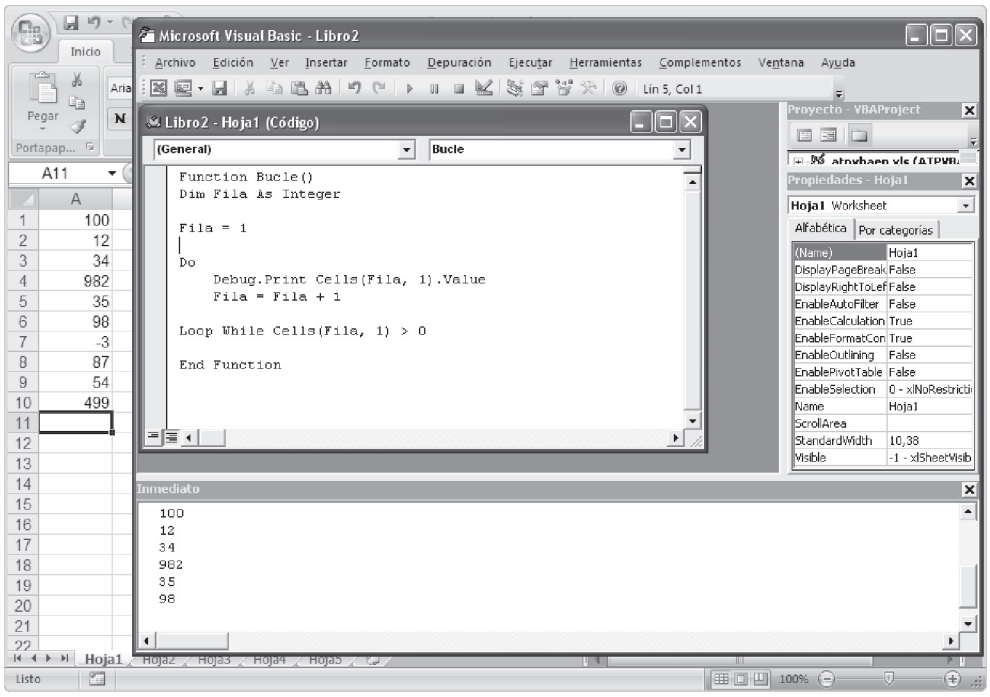
```
Do Until condición
        instrucciones
Loop
```

Do..Loop..Until Se ejecuta al menos una vez.

```
Do
        instrucciones
Loop Until
```

Los bucle**s Do** es conveniente utilizarlos cuando no se sabe cuántas veces se necesitará ejecutar las instrucciones.

Con el bucle siguiente conseguiríamos mostrar en la ventana **Inmediato** el contenido de todas las celdas que existan en la columna A, deteniéndose el bucle cuando se encuentre el valor cero o inferior.



Observe como el bucle solo ha llegado hasta el número 98 ya que el siguiente es negativo.

BUCLE WHILE/WEND

Visual Basic permite una variante del bucle **Do While** (es decir, con la comprobación al principio). En lugar de decir:

```
Do While X = 0
Loop
```

Se puede decir

```
While X = 0
Wend
```

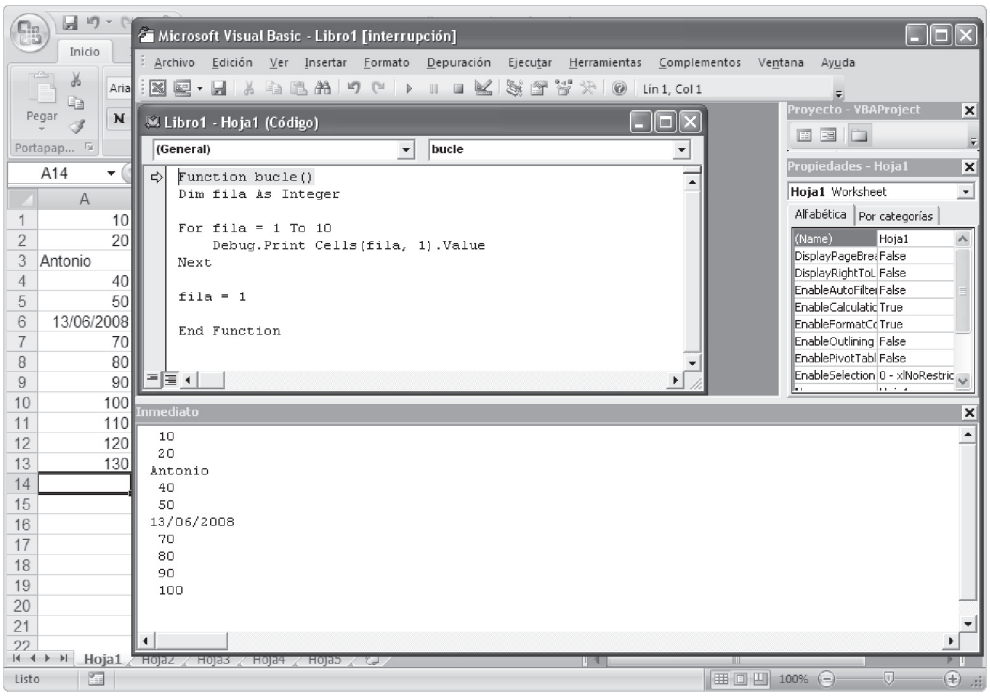
FOR...NEXT

Esta clase de bucles se utilizan cuándo sabemos el número de veces que se van a ejecutar las instrucciones. A diferencia del bucle **Do**, este bucle utiliza una variable llamada contador que incrementa o reduce su valor en cada repetición. Su sintaxis es:

```
For contador=inicio To Fin Step incremento
        Instrucciones
Next
```

El parámetro incremento es opcional, puede ser positivo o negativo. En caso de ser positivo, el parámetro inicio debe ser menor o igual que el parámetro fin o no se ejecutará el bucle. En caso contrario (negativo), inicio debe ser mayor o igual que fin para que se ejecute el bucle. Si el parámetro **Step** no se especifica, el incremento será de 1.

En el siguiente ejemplo, tomamos los valores contenidos en las celdillas A1:A10 y lo mostramos en la ventana inmediato.



FOR EACH...NEXT

Esta sentencia de control es parecida a **For...Next**, con la diferencia que ésta repite un grupo de instrucciones por cada elemento de una colección de objetos o de una matriz.

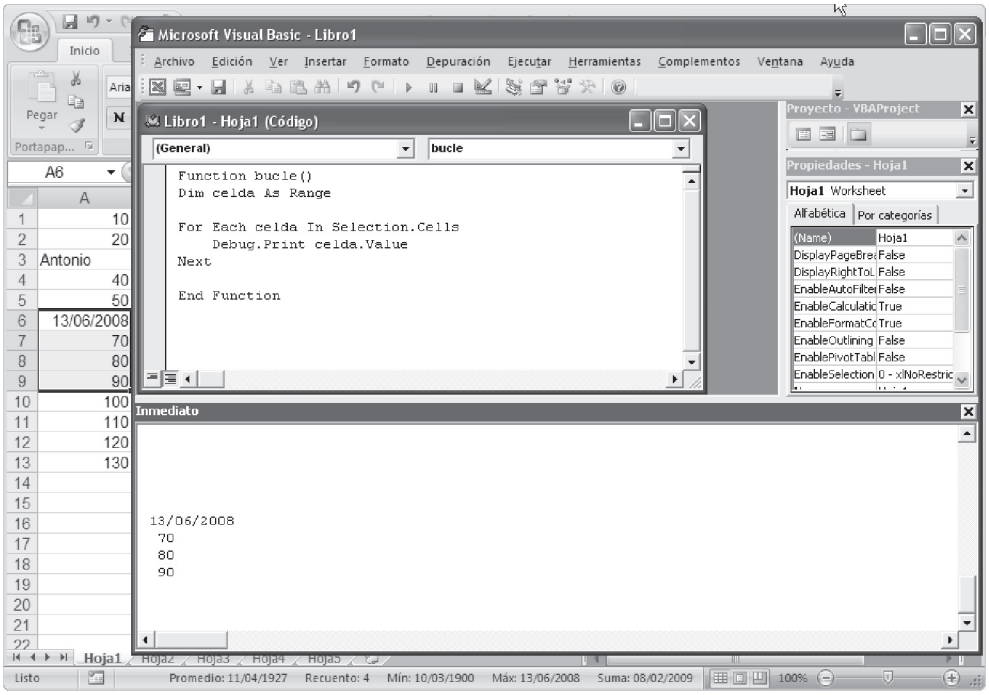
Este bucle es útil cuando no se sabe cuántos elementos hay en una colección. Su sintaxis es:

```
For Each elemento In grupo
        Instrucciones
Next elemento
```

Hay que tener en cuenta las siguientes restricciones cuando se trabaja con el bucle **For Each....Next:**

- El parámetro elemento sólo puede ser de tipo **Variant** cuando se trata de matrices.
- No se puede utilizar este bucle para una matriz de tipos definidos por el usuario, ya que el tipo **Variant** no puede contener un tipo definido por el usuario.
- El parámetro elemento sólo puede ser una variable de tipo **Variant** o una variable **Object.**

Observe el siguiente ejemplo.



Cuando trabajamos con bucles, a veces es necesario una instrucción que nos permita salir del bucle sin que se haya cumplido la condición. Estas son:

TIPO DE BUCLE	SALIDA DE BUCLE...
DO...WHILE...LOOP	EXIT DO
DO...UNTIL...LOOP	EXIT DO
FOR...NEXT	EXIT FOR