Unidad Didáctica 6

Conociendo Visual Basic

Contenido

1. Introducción

Visual Basic o, como se denomina en *Excel, Visual Basic para aplicaciones,* es un entorno de programación cuyo lenguaje de programación, del mismo nombre, permite diseñar y desarrollar aplicaciones de forma que se pueden crear nuevas funcionalidades y soluciones para el programa *Excel.*

El **editor de *Visual Basic*** contiene todas las herramientas necesarias para escribir el código necesario, así como la creación de elementos interactivos, como pueden ser botones, listas, combos, etiquetas etc., que necesitarán las aplicaciones.

Este editor aparece como una ventana independiente de *Excel,* como si fuese un programa aparte. Cuando se cierre *Excel,* también se cerrará este editor.

Antes de continuar, hay que saber que el lenguaje de programación *Visual Basic* es muy extenso. De hecho, hay cursos especializados que solo ofrecen esta materia. Desde este manual, se intenta enseñar a manejar el editor de código y los fundamentos básicos de programación para desarrollar sus funciones. Una vez que se aprendan los principios básicos, si se quisiera adentrarse más en este mundo, se deberán adquirir los conocimientos en manuales específicos de *Visual Basic.*

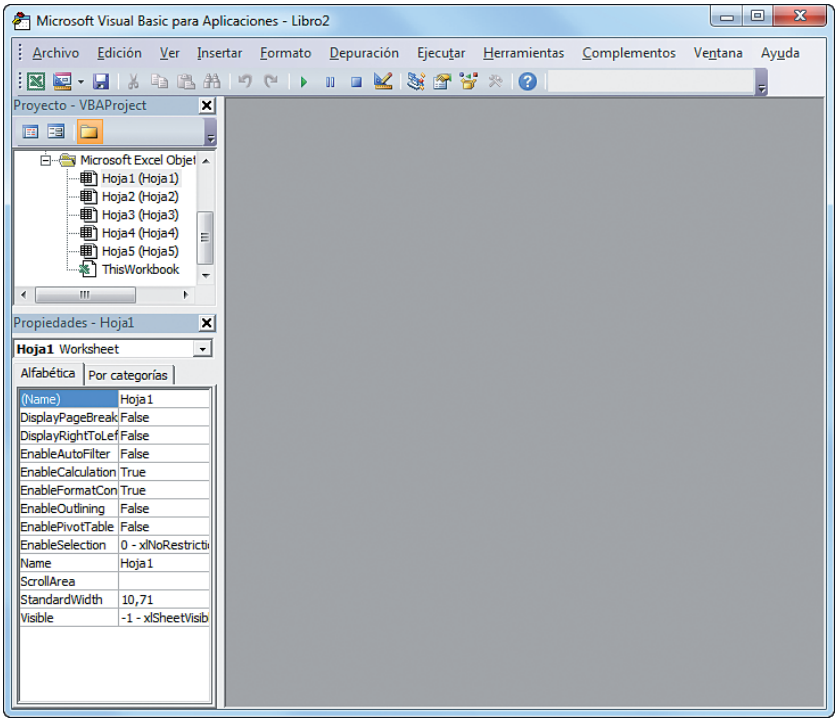
Aunque todo esto parezca algo complejo, practicando y viendo los resultados directamente en pantalla, se motivará para crear nuevas soluciones personalizadas.

2. La pantalla de Visual Basic

Para acceder al editor de *Visual Basic,* como ya se sabe, se debe presionar la combinación de teclas [Alt] + [F11] o pulsar el botón **Visual Basic** de la ficha **Programador.** En ambos casos, se abrirá una ventana como la que se muestra a continuación.

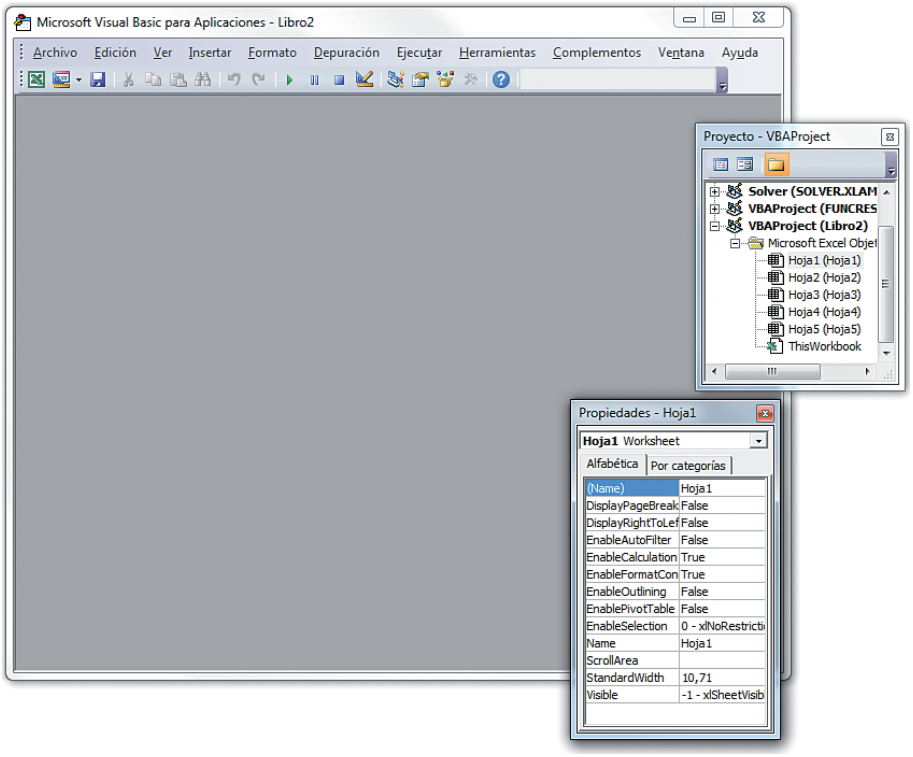
Nota

Este entorno de programación viene incluido en el paquete Office y, para cada una de las aplicaciones Office, el editor será el mismo.



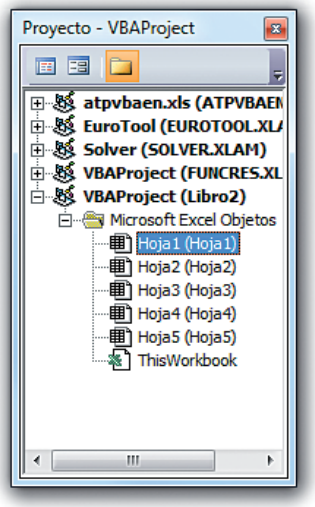
Editor de Visual Basic

En esta ventana, se puede distinguir, en la parte superior, el menú de opciones y una barra de herramientas con los botones más usados. En la zona de la izquierda, aparecen dos paneles: el **Panel de proyecto** y el **Panel de propiedades.** Estos paneles son flotantes y podrán moverse haciendo clic sobre su barra de título y arrastrándolos hasta el lugar deseado.



Paneles flotantes

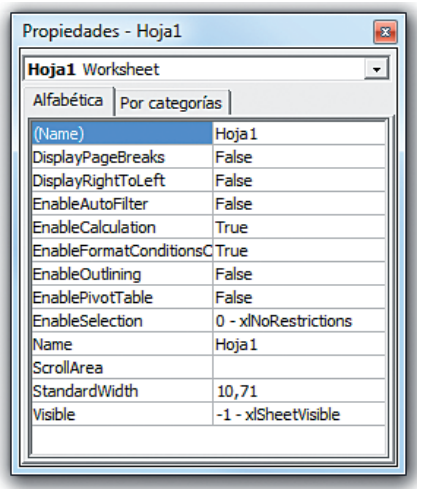
Aquí se va a utilizar el editor de *Visual Basic* para crear macros y funciones, pero se pueden crear verdaderos proyectos muy complejos. Para gestionar estos proyectos, se tiene el **Examinador de proyectos.**



Examinador de proyectos

En esta lista, aparecerán los elementos en sus respectivas carpetas, como pueden ser formularios, módulos de código y otros objetos relacionados con el proyecto.

Como ya se sabe, cada elemento de *Excel* tiene sus propias propiedades. Estas aparecen, junto con el valor que tienen, en la ventana **Propiedades.** Esta ventana resultará de muchísima ayuda a la hora de construir formularios, ya que en ella se podrán configurar infinidad de propiedades de los objetos interactivos.



Propiedades

Esta ventana permitirá cambiar las propiedades a un objeto en tiempo de diseño. No es necesario saber todos los valores de una propiedad, ya que muchas de ellas presentan una lista desplegable con las propiedades que admiten.

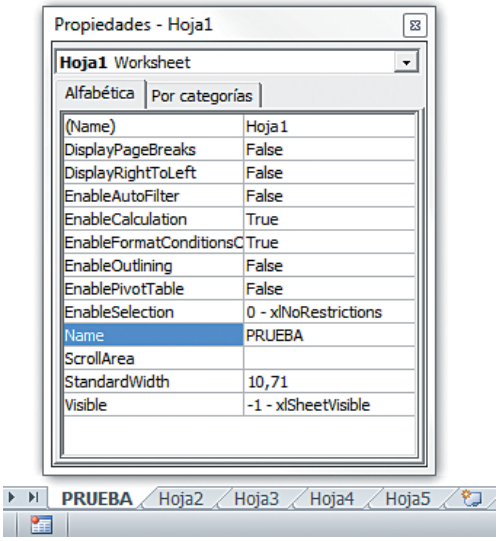
Estas propiedades pueden estar ordenadas alfabéticamente o por categorías.

Al igual que ocurre con todos los lenguajes de programación, todos están estandarizados al idioma inglés. Es por esto que todas las propiedades e incluso el código de programación aparecerá en este idioma.

Se puede probar a cambiar alguna propiedad del libro actual o de cualquiera de sus hojas.

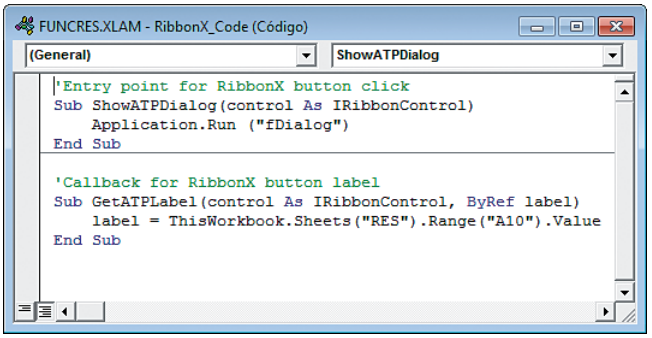
Ejemplo

En este caso, aparecen las propiedades de la Hoja1 y, si se cambia la propiedad **Name,** se verá cómo cambia el nombre de la hoja en el libro.



2.1. El editor de código

Volviendo a la ventana **Examinador de proyectos,** si se hace doble clic sobre cualquiera de sus elementos, se abrirá el editor de código. También se abrirá pulsando la tecla [F7] o desplegando el menú **Ver** y seleccionando la opción **Código.** Este editor es similar a un editor de texto y será en él donde se introduzcan las instrucciones que forman las macros y funciones.



Editor de código

Nota

En el código de una macro, se puede ver que el texto aparece en colores. Lo normal es que los comentarios aparezcan en verde, las palabras reservadas de Visual Basic en azul, las variables y propiedades en negro y, si existe algún error, aparecerá en rojo.

En esta ventana, hay dos listas desplegables: la que está a la izquierda es la **Lista de objetos** y la de la derecha la **Lista de procedimientos.** La primera mostrará los nombres de los objetos que existan en un formulario. La segunda mostrará los procedimientos o funciones disponibles para un objeto. Siempre aparecerá en esta lista un procedimiento. Si se ha escrito código en uno, aparecerá por defecto ese procedimiento. Si no hay código en ninguno, saldrá el que tenga por defecto cada objeto.

En este punto, cabe preguntarse qué es un procedimiento. Para explicarlo, primero hay que saber lo que es un evento. Un **evento** es una acción que le puede ocurrir a un objeto, por ejemplo, a un botón, se le puede hacer clic, doble clic, que pase el cursor del ratón por encima de él, etc. Estas acciones son los eventos. El **procedimiento** sería la respuesta por parte del botón al evento que en ese momento le está afectando.

Esa respuesta es la que se debe programar para que actúe como uno quiera. No todos los objetos tienen que tener procedimientos, esto dependerá de lo que se quiera hacer con ellos.

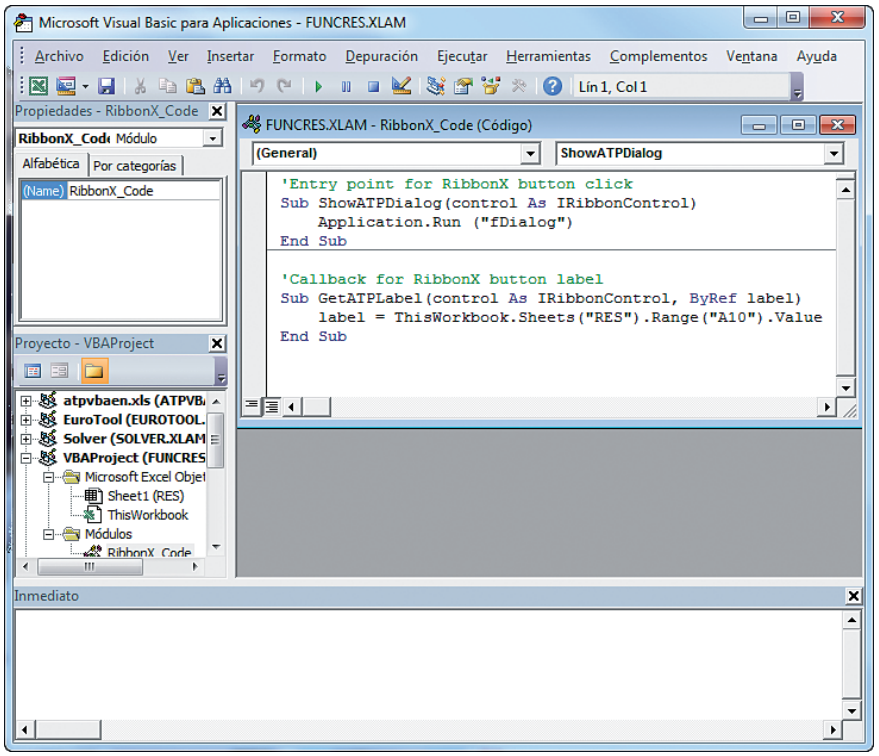
Nota

Para que un evento no ejecute un procedimiento, tan solo hay que dejarlo vacío.

3. La ventana Inmediato

A veces, ocurrirá que el código que se escriba no funcione correctamente o que se quiera saber cómo actúa una determinada expresión. Para ello, se acudirá a la ventana **Inmediato.** En esta ventana, se podrán evaluar expresiones y ver el comportamiento de los procedimientos.

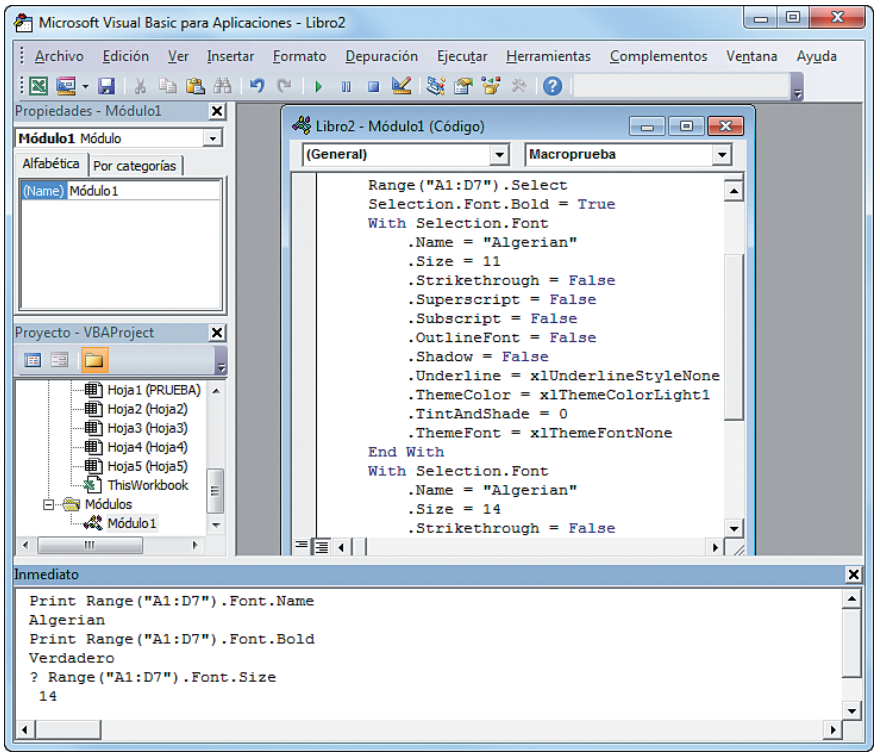
Esta ventana se puede activar desde el menú **Ver** o presionando la combinación de teclas [Ctrl] + [G].



Ventana Inmediato (abajo)

Para ver valores en esta ventana, se puede hacer de dos formas:

- Utilizando la sentencia **Print** o **?** directamente sobre la ventana.
- Colocando sentencias **Debug.Print** en el código de la aplicación**.**



Ver valores en Inmediato

Una vez en la ventana Inmediato, se escribe y actúa como en cualquier procesador de texto.

Como se puede ver en el gráfico, la primera sentencia que se ha escrito ha devuelto el nombre de la fuente:

```
Print Range ("A1:D7"). Font.Name
Algerian
```

La segunda ha devuelto si el rango de celdas estaba en negrita o no. En este caso, era cierto que estaba en negrita:

```
Print Range ("A1:D7"). Font.Bold
Verdadero
```

La tercera expresión da como resultado el tamaño de la fuente:

```
? Range ("A1:D7"). Font.Size
14
```

También será posible asignar valores a variables o expresiones y ver el resultado que se ofrece.

4. ¿Qué son y para qué sirven las variables y constantes?

Cuando se programa, será necesario guardar los valores que se van obteniendo en variables. Las variables deben tener un tipo de dato establecido, es decir, qué tipo de dato se va a guardar en ellas.

Nota

En una variable declarada de tipo texto, no se podrá almacenar un número y viceversa. Esto debe tenerse claro, ya que el programa lanzará un error si ocurriera.

4.1. Variables

Como ya se ha dicho, las variables se utilizarán para almacenar un dato durante la ejecución del programa. La declaración de la variable se divide en tres partes y su sintaxis es la siguiente:

```
Dim nombre de variable As tipo
```

- **Dim…As:** palabra reservada de *Visual Basic* para declarar una variable.
- **Nombre de variable:** nombre que se le pondrá a la variable. Este nombre suele ser significativo, según el valor que contenga.
- **Tipo:** tipo de dato que la variable puede almacenar.

A la hora de declarar variables, se deben tener en cuenta las siguientes condiciones:

- 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.
- Se debe tener un único nombre dentro del mismo alcance de la variable.

Las variables se pueden declarar de dos formas:

- Declaración implícita.
- Declaración explícita.

Declaración implícita

Permite asignar valores a las variables sin haberlas declarado. Hay que tener cuidado, ya que se pueden cometer errores y *Visual Basic* no informará de ello.

Ejemplo

Véase en el siguiente código:

```
Var1 = 34
Var2 = 26
Var3 = Var + Var2
```

A simple vista, el código parece correcto, pero, al fijarse bien, se ve que este código provocará un error que Visual Basic no detectará, ya que se ha escrito Var en lugar de Var1 y lo pasará por alto. En realidad, la suma estará mal y los cálculos que se estén realizando serán erróneos.

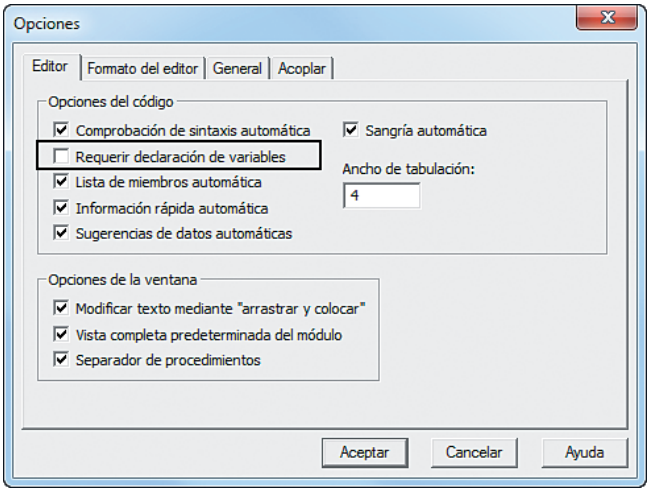
Declaración explícita

Aquí, *Visual Basic* sí detectará un error en caso de que las variables no estén declaradas correctamente. Para ello, en el comienzo de un módulo (que se verá más adelante) de clase, de formulario o estándar, se debe escribir la instrucción **Option Explicit.**

También puede hacerse a través del menú **Herramientas** de *Visual Basic,* seleccionando la opción **Opciones.** En el cuadro de diálogo que aparece, se activará la opción **Requerir declaración de variables.**

Consejo

Lo aconsejable es declarar todas las variables. De esta forma, se sabrá cuándo hay un error y se podrá corregir más fácilmente.



Requerir declaración de variables

4.2. Constantes

Una constante también almacena datos, pero es un dato que por sí solo no tiene un significado obvio.

Una constante en realidad sustituye a un número o cadena de texto. La diferencia con la variable es que la constante siempre tendrá el mismo valor.

Las constantes pueden ser de dos tipos:

- **Constatntes intrínsecas:** definidas por el sistema, no las puede definir uno mismo.
- **Constantes simbólicas:** será uno mismo el que las defina con la instrucción **Const.**

La sintaxis para declarar una constante es la siguiente:

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

4.3. Tipos de datos

El tipo de dato a la hora de declarar una variable indica cómo se guardarán los datos en la memoria del ordenador. Existe un tipo de datos predeterminado, que es el tipo **Variant** y es como un comodín que podrá representar distintos tipos de datos, tanto numéricos como alfanuméricos.

Nota

Lo correcto a la hora de declarar una variable sería asignarle el tipo que le corresponda sabiendo el dato que se va a almacenar.

En la declaración de una variable, se pueden utilizar las siguientes instrucciones: **Private, Dim, Public** o **Static.**

Unos ejemplos serían:

```
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 se pueden asignar a las variables 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 entera larga 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 **¡.**
- **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:** se utilizará cuando la variable almacene una cadena de texto y nunca un valor numérico. La cadena puede ser de una longitud variable, es decir, que crece o disminuye según se le asignen nuevos datos. Para definir la variable **String** de longitud fija, se debe escribir:

```
Dim Nombre As String * 20
```

En este caso, la variable no almacenará más de 20 caracteres.

- **Byte:** se utiliza cuando la variable contenga datos binarios.
- **Boolean:** se utiliza cuando la variable solo vaya a contener información de verdadero o falso, sí o no o activado o desactivado. El valor predeterminado de un variable declarada como **Boolean** es **False.**

- **Date:** se utiliza cuando la variable vaya a almacenar un valor de fecha y hora.

4.4. Conversiones de tipo

Frecuentemente, se encuentra el problema de tener que pasar un dato de cierto tipo a otro. Hay variables que permiten pasar datos entre ellas, 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 ven a continuación:

- **CInt(expresión):** convierte el resultado de una expresión de tipo **Integer.**
  Ejemplo: Resultado = CInt(345,645)
  Resultado: declarada como Integer = 346
- **CLng(expresión):** convierte el resultado de una expresión de tipo **Long.**
  Ejemplo: Resultado = CLng(321798,23)
  Resultado: declarada como Long = 321798
- **CBool(expresión):** convierte el resultado de una expresión de tipo **Boolean.**
  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: declarada como Byte = 36
- **CCur(expresión):** convierte el resultado de una expresión de tipo **Currency.**
  Ejemplo: Resultado = CCur(852,567476)
  Resultado: declarada como Currrency = 852,5674
- **CDate(expresión):** convierte el resultado de una expresión de tipo **Date.**
  Ejemplo: Resultado = CDate(«12/10/2000»)
  Resultado: 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: declarada como Double = 6500,638

- **CVar(expresión):** convierte el resultado de una expresión de tipo **Variant.**
  Ejemplo: Resultado = CVar(12 & «avo»)
  Resultado: declarada como Variant = «12avo»
- **CStr(expresión):** convierte el resultado de una expresión de tipo **String.**
  Ejemplo: Resultado = CStr(275)
  Resultado: declarada como String = «275»

Cuando se declara una variable dentro de un procedimiento, solo el código que contiene el procedimiento podrá tener acceso a esa variable. Es lo que se denomina una variable de alce local. Cuando se necesite que la variable tenga un alcance más grande, deberá declararse de una manera determinada.

Visual Basic permite definir el alcance que tendrá la variable, pudiendo ser:

- **Públicas:** las variables estarán accesibles desde cualquier punto.
- **Privadas:** las variables solo estarán accesibles en el lugar en que se declaren.

Nota

Dentro de un procedimiento, no se podrán declarar variables públicas.

Ya se ha visto cómo se declara una variable a nivel de procedimiento. tambien se podría declarar con la instrucción **Static.** Por ejemplo:

```
Dim temporal As Integer
Static permanente As Integer
```

La diferencia está en que las variables declaradas con **Dim** solo existirán en memoria mientras se esté dentro del procedimiento (después se liberan de la memoria del ordenador), mientras que las **Static** existen mientras la aplicación esté ejecutándose.

Otra dos formas de declarar variables es con las instrucciones **Private** y **Public:**

```
Private variable As String
Public variable As String
```

La diferencia está en que la privada solo se utilizará en el módulo en el que se encuentre y la pública será accesible desde cualquier parte del programa.

Consejo

No puede haber dos variables con el mismo nombre dentro de un procedimiento, ni en un módulo. Lo aconsejable es que asigne un solo nombre por cada variable que se declare.

5. Tipos de operadores en Visual Basic

Al igual que en las fórmulas de *Excel,* en *Visual Basic* se pueden manejar datos y realizar cálculos aritméticos, comparaciones de texto, etc. a continuación, se verán los tipos de operadores que hay y algún ejemplo.

5.1. Operadores aritméticos

Los operadores aritméticos que se pueden utilizar son los siguientes:

OPERADORES ARITMÉTICOS	
Operador	**Descripción**
^	Se utiliza para elevar un número a la potencia del exponente
*	Se utiliza para multiplicar 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
```

5.2. Operadores de comparación

Los operadores de comparación que se pueden utilizar son los siguientes:

OPERADORES DE COMPARACIÓN	
Operador	**Descripción**
=;<>	Operadores de igualdad y desigualdad
<, >, <=, =>	Operadores menor que, mayor que, menor o igual, mayor o igual
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
```

5.3. Operadores lógicos

Estos operadores solo sirven con valores booleanos, es decir verdaderos o falsos. En las siguientes tablas, se verá el resultado que se obtiene al utilizar los operadores con unos posibles valores:

- **Not:** convierte el valor de la variable en su opuesto.
- **And:** realiza una conjunción lógica.

Valor 1	Valor 2	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

- **Xor:** realiza una exclusión lógica.

Valor 1	Valor 2	Resultado
True	True	False
True	False	True
False	True	True
False	False	False

- **Eqv:** efectúa una equivalencia lógica.

Valor 1	Valor 2	Resultado
True	True	True
True	False	False
False	True	False
False	False	True

- **Imp:** efectúa una implicación lógica.

Valor 1	Valor 2	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 haya expresiones con varios operadores, primero se resolverán los operadores aritméticos, a continuación las expresiones que contienen operadores de comparación y, por último, los operadores lógicos. Los operadores de comparación siempre se evalúan de izquierda a derecha. Los aritméticos y lógicos se evalúan según la prioridad de cada uno. Reflejada en la siguiente tabla.

PRIORIDAD DE LOS EVALUADOORES		
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	

Una vez validado el fichero de la contratación por el SPEE, se podrá proceder a descargar el mismo. Para ello, situándose en la ventana de inicio del programa *Contrat@,* se seleccionará la opción de **Seguimiento de las comunicaciones realizadas.**

6. Sentencias condicionales

Las sentencias condicionales son las encargadas de comprobar condiciones y, dependiendo del resultado, realizar una operación u otra.

6.1. Estructura If….Then

Ejecuta una instrucción o varias dependiendo de una condición. La sintaxis es la siguiente:

```
If condición Then instrucción
```

En el siguiente ejemplo, se va a comprobar si la celda que está seleccionada tiene un valor mayor a 1.000. Si se cumple la condición, se pondrá la fuente en negrita.

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

En este caso, solo se ha ejecutado una instrucción. Para ejecutar más de una, se debe utilizar la siguiente sintaxis:

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

Nota

Se puede traducir la sentencia como:

```
Si condición entonces
    Instrucciones...
Fin del Si
```

6.2. Estructura If……Then…..Else

Con esta estructura, se pueden ir anidando sentencias y definiendo bloques de instrucciones, pero solo uno se ejecutará, el que cumpla la condición. Su sintaxis es:

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

Primero, se evaluará la primera condición. Si resulta falsa, se pasará a evaluar la segunda condición, si también resulta falsa se ejecutará automáticamente el tercer bloque de instrucciones.

Siguiendo con el ejemplo anterior, ahora, si la celda activa es menor a 1000, el contenido de la celda aparecerá en cursiva:

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

Nota

Se podría traducir de la siguiente manera:

```
Si celda_activa > 1000 entonces
    Celda activa en negrita
En caso contrario
    Celda activa en cursiva
Fin del Si
```

6.3. Estructura Select Case

Esta estructura se puede utilizar en sustitución de la anterior. En lugar de ir anidando **If…Then…Else,** se utiliza **Select Case** y se podrán evaluar muchos valores distintos de una misma expresión. 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** puede ser una lista de uno o más valores. El **Case** irá evaluando la lista de expresiones y, cuando una sea verdadera, ejecutará las instrucciones. Por último, el **Case Else** se ejecutará cuando haya revisado todas las listas de expresiones y hayan sido falsas.

He aquí un ejemplo con variables booleanas:

```
Dim A, B, OPCION, RESULTADO
A = 10
B = 5
Select Case OPCION
    Case 1
        RESULTADO = A + B
    Case 2
        RESULTADO = A - B
    Case 3
        RESULTADO = A * B
    Case Else
        A = 0
        B = 0
End Select
```

Lo que ocurre dentro del **Select Case** es:

```
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
```

7. Sentencias de repetición

Las sentencias de repetición son también conocidas como estructuras de bucle o bucles simplemente. Se utilizan para ejecutar instrucciones repetidamente hasta que no se cumpla la condición. A continuación, se ven estas estructuras.

7.1. Do...Loop

Este buque permite ejecutar un bloque de instrucciones indefinidamente hasta que no se cumpla la condición. Su sintaxis es:

```
Do While condición
   instrucciones
Loop
```

Primero, se evalúa la condición y, si es falsa, se salta todas las instrucciones y termina. En caso que de que sea verdadera, ejecutará las instrucciones y volverá a la línea **Do While** para comprobar de nuevo la condición.

Importante

Este bucle se podrá ejecutar un número indefinido de veces, pero hay que asegurarse de que alguna vez la condición no se cumpla, ya que, de lo contrario, entraría en un bucle infinito que bloquearía el programa.

Existen dos variantes de este bucle:

- **Do..Until..Loop:** se ejecuta cero o más veces.

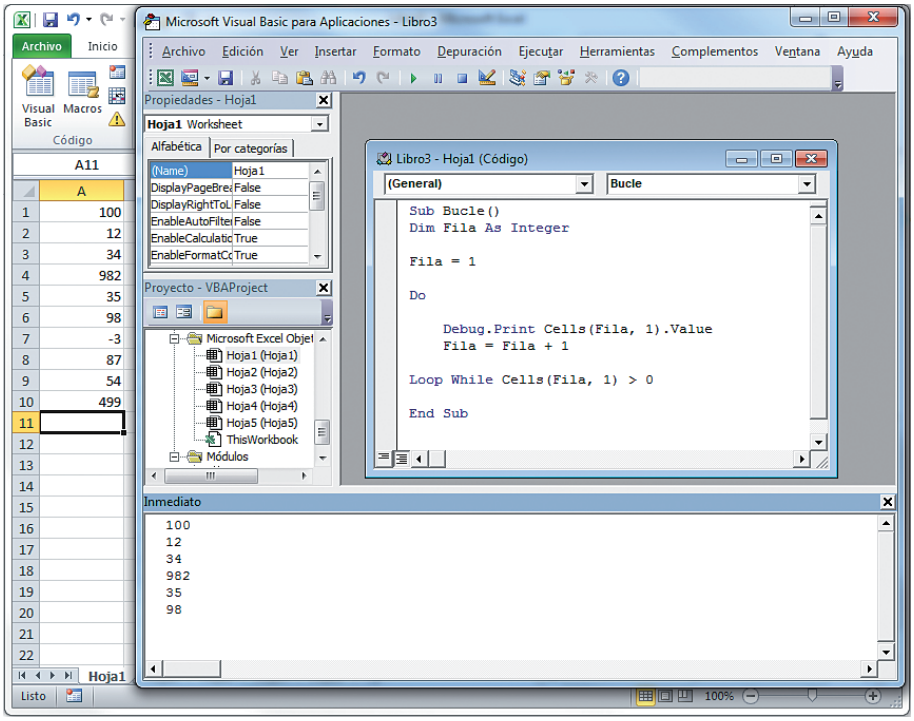
```
Do Until condición
    instrucciones
Loop
```

- **Do..Loop..Until:** se ejecuta al menos una vez.

```
Do
    instrucciones
Loop Until
```

Los bucles **Do** se pueden utilizar cuando no se sabe cuántas veces se necesitará ejecutar las instrucciones.

En el siguiente ejemplo, se va a mostrar el contenido de todas las celdas de la columna **A** y solo se detendrá el bucle cuando encuentre un valor cero o inferior.



Ejemplo de bucle

Obsérvese que el bucle se ha detenido en el número 98, ya que el siguiente era negativo.

7.2. Bucle While/Wend

Este bucle es una variante del bucle **Do While** y su sintaxis es:

```
While condición
    Intrucciones
Wend
```

Nota

Se puede traducir el bucle While por la palabra Mientras.

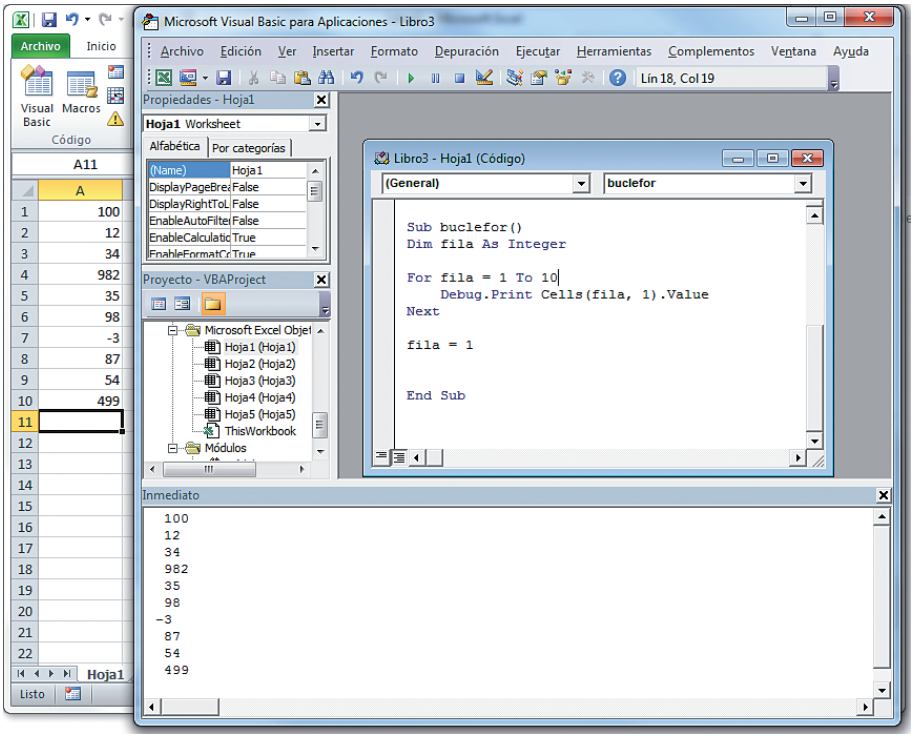
7.3. For…Next

Este bucle también es repetitivo, pero, en este caso, sí se sabe las veces que se va a repetir un bloque de instrucciones. Este bucle utiliza una variable que hace de contador e incrementa o reduce su valor en cada repetición. Su sintaxis es:

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

El parámetro **Step** incremento es opcional. Si no se pone, se entiende que el incremento será de uno en uno. Este parámetro será el que vaya incrementando o reduciendo el contador hasta que sea igual que el parámetro **Fin,** en cuyo caso el bucle habrá terminado.

En el siguiente ejemplo, se mostrarán los datos que hay en el rango **A1:A10.**



Ejemplo de datos del rango A1:A10