Alfaomega

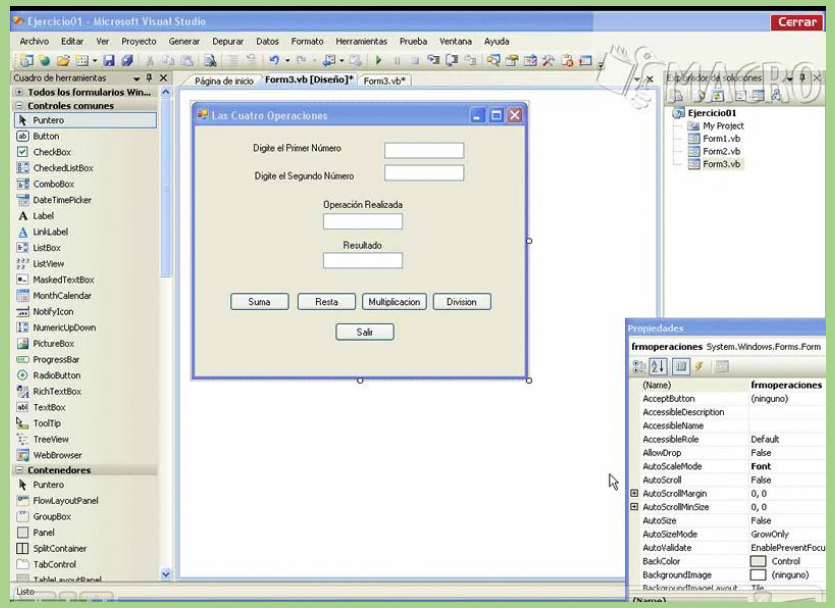
Capítulo 9

Programación orientada a objetos

Continuar

Programación orientada a objetos

Visual Basic para Aplicaciones® es un lenguaje de programación orientado a objetos. A grandes rasgos, esto significa que a cada objeto se le puede agregar código para que cuando se active el objeto el código se ejecute.



Procedimientos SUB

Los procedimientos SUB son una serie de instrucciones. Con SUB iniciamos una macro y la terminamos con END SUB. Ambos se desarrollan en los módulos dentro del Editor de Visual Basic®. Este tipo de procedimientos pueden realizar cualquier tipo de acción dentro de Excel®, ya sea agregar una hoja o construir un sistema administrativo.

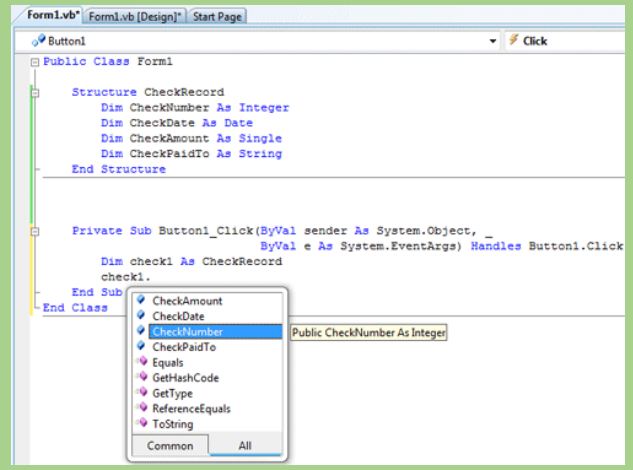
```
Sub practica7()

Dim Fila As Integer
ActiveSheet.Range("A1").Activate
For Fila = 0 To 4
ActiveCell.Offset(Fila, 0).Value = Fila
Next Fila

End Sub
```

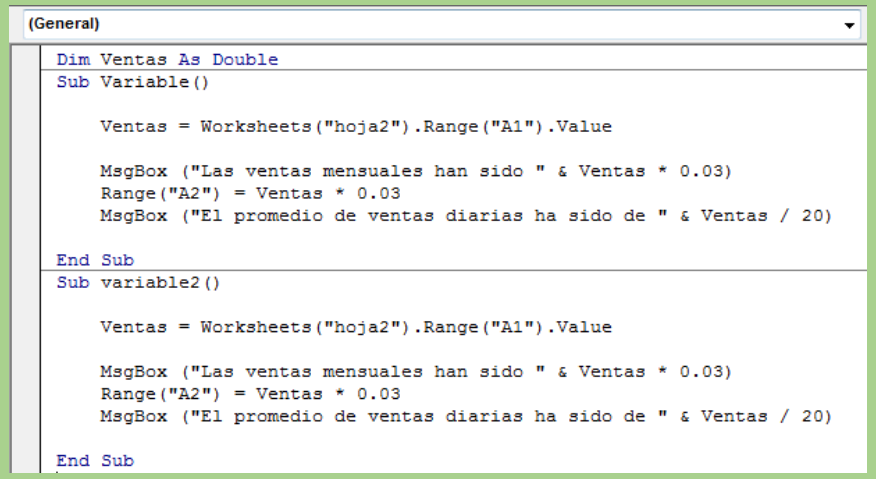
Variables

Así como Excel® maneja miles de variables (celdas) para manejar la información, Visual Basic para Aplicaciones® también puede tener sus propias variables para el manejo de los datos. Una variable es un pedazo de memoria reservada para que nosotros pongamos cierta información ahí. Las variables son como cajones en un enorme archivero muy similar a la hoja de cálculo. La diferencia radica en que cada vez que se necesite una variable en Visual Basic® hay que darle un nombre y un tipo.



Declarar variables en una Macro

La declaración de variables tiene que ser parte del hábito de un buen programador, ya que nos permite tener mayor control sobre los programas largos y complejos que desarrollemos. Si una variable no se declara, Visual Basic para Aplicaciones® la define como Variant, haciendo que se pueda aceptar cualquier tipo de información durante la ejecución del programa.



Variables locales

Estas variables se declaran dentro de la macro, después de la instrucción Sub, y sólo retienen sus datos mientras el procedimiento se está ejecutando; cuando éste termina, la variable no existe. Además, los datos sólo existen dentro de procedimiento.

Before

```
public static void AssertKnownType(Type type)
{
    IList<Type> typesKnown =
        new Type[] {TypeOf.Boolean, TypeOf.Int32, TypeOf.String};

    if(!typesKnown.Contains(type))
```

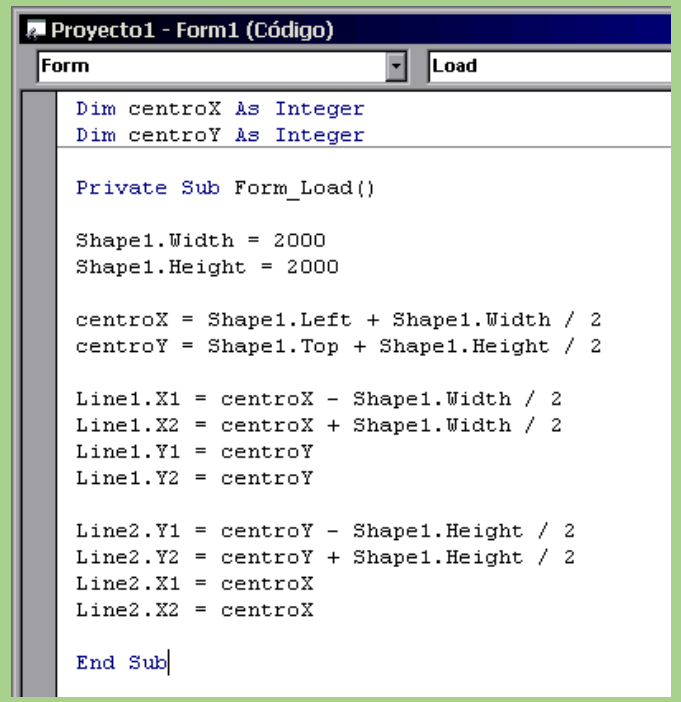
After

```
public static void AssertKnownType
    (Type type, Func<IList<Type>, bool> contains)
{
    IList<Type> typesKnown =
        new Type[] {TypeOf.Boolean, TypeOf.Int32, TypeOf.String};

    if(!contains(typesKnown))
```

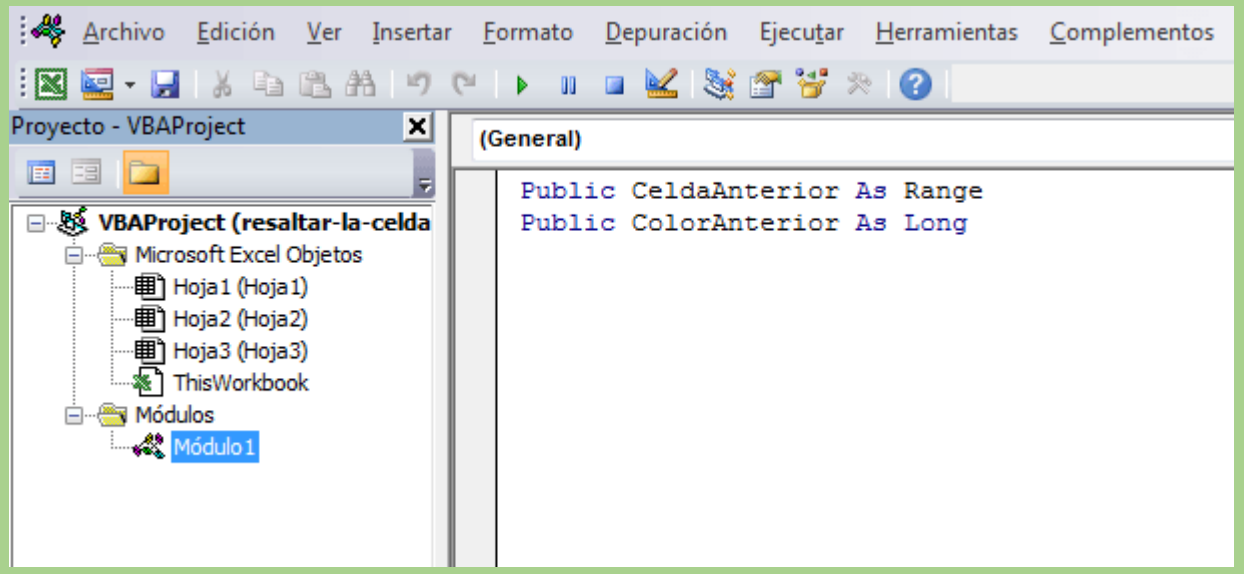
Variables de módulo

Estas variables pueden conservar sus datos cuando se llama a otra macro desde el procedimiento que se está ejecutando, y si requiere usar la información de la variable en la otra macro, estos datos estarán disponibles para usarse.



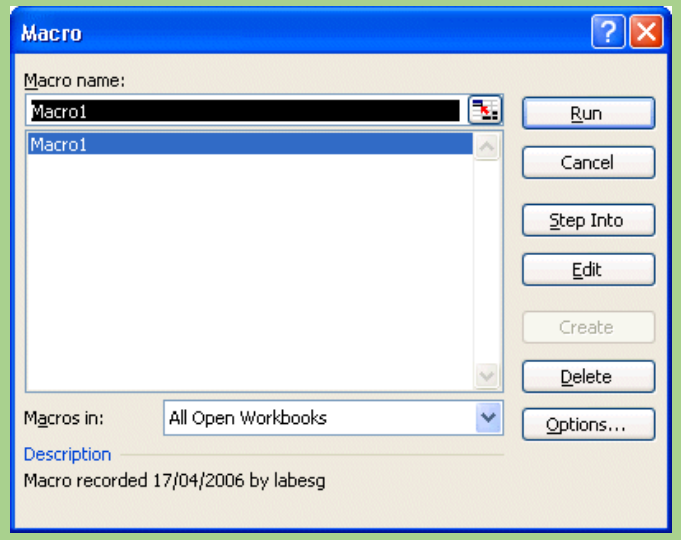
Variables públicas

Estas variables tienen la característica de que su información está disponible para todos los módulos que integran el proyecto. Este tipo de variables se debe definir con la instrucción Public en vez de la instrucción DIM. Se declaran a nivel módulo, es decir, antes de cualquier macro.



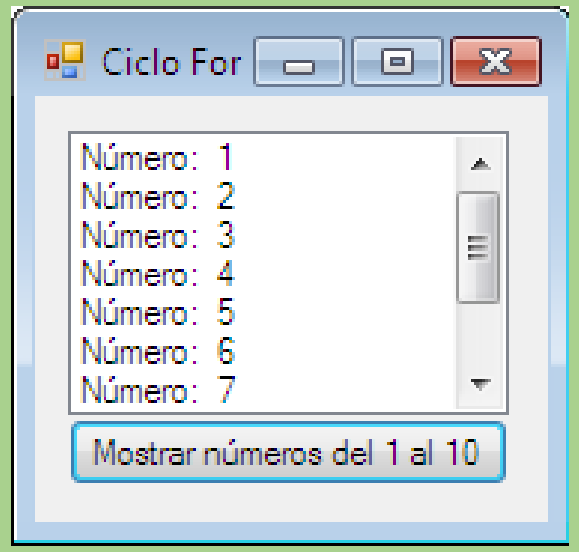
Decisiones dentro de una Macro

El flujo del programa siempre se hace de arriba hacia abajo, pero eso es un flujo de programa lineal, y no es muy efectivo. Los programas que no permiten brincar líneas de código o tomar decisiones a partir de ciertas condiciones no son programas útiles, porque aunque un programa siempre hace lo mismo, debemos poder tomar cierto tipo de decisiones sobre la ejecución de la macro.



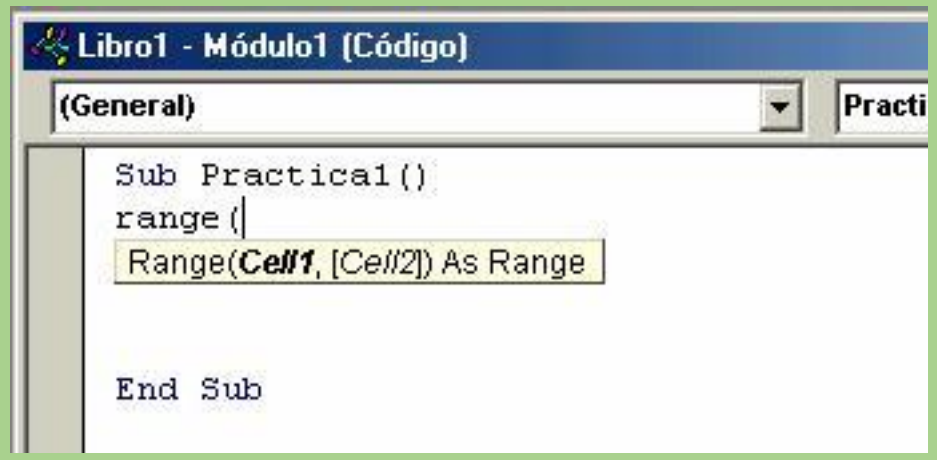
Ciclos

Los ciclos son una parte fundamental de nuestro trabajo dentro de las macros, ya que nos permiten realizar una tarea un número definido de veces; por ejemplo, si se requiere revisar una lista de información para que se pueda llevar a cabo cierta evaluación, esto lo tenemos que realizar con un ciclo para que recorra todas las celdas de la lista.



Propiedad Range

La propiedad Range nos permite manejar la información que exista dentro de las celdas de una manera rápida y eficiente.



Propiedad Cells

La propiedad Cells, al igual que la propiedad Range, nos permite manejar la información de las celdas.

```
Private Sub Celda()
    Cells(12, 3).Value = "Solo " & 2
    ActiveSheet.Cells(10, 6).Value = "Paris"
    'La Celda 10,6 es la F10
    Range("C13:D14").Value = "Cuadrado"
    Range(Cells(15, 3), Cells(16, 4)).Value = "Cubo"
    Range("C17:F20").Cells(2, 1).Value = "Elipse" 'Esto solo pone una elipse
End Sub
```