MICROSOFT EXCEL

HERRAMIENTAS INFORMÁTICAS II

Prof. Ing. Norma Cañizares

Microsoft Excel

Visual Basic for Applications

Rutina de Tratamiento de Errores

- Generalidades sobre los Tipos de Errores
- ¿Que es la Rutina de Tratamiento de Errores?
- Formato General para Implementar una Rutina de Tratamiento de Errores
- Obtener Información del Error: Objeto ERR
- Errores Interceptables
- Definir Errores Propios
- Continuar el programa después de la ejecución de la Rutina de Tratamiento de Errores: Instrucción Resumen
- Consideraciones a tener en cuenta antes de escribir el código para una Aplicación
- Ejemplo: División de 2 Números

Microsoft Excel

GENERALIDADES SOBRE LOS TIPOS DE ERRORES

- **Errores en Tiempo de Compilación.** Son los típicos errores que impiden hacer funcionar el programa debido, por ejemplo, a errores de sintaxis en las instrucciones, llamadas a funciones que no existen o llamadas con el tipo o el número de parámetros incorrectos, etc. Este tipo de errores no dan demasiados problemas, primero porque el compilador avisa donde se han producido y luego porque simplemente revisando la sintaxis se solucionan rápidamente.

- **Errores en tiempo de ejecución.** *Estos errores se producen por una mala programación del código al* ***no haber previsto determinados casos concretos o especiales****, como por ejemplo intentar abrir un archivo que no existe, intentar acceder a miembros que no existen, etc****. Cuando se produce este tipo de errores se detiene la ejecución del programa*** *y normalmente se informa del tipo de error que se ha producido. Muchos de* ***estos errores se pueden solucionar mediante rutinas o funciones de tratamiento de errores.***

- **Errores de lógica.** Son los más complicados de detectar ya que ni se detectan en la fase de ejecución, ni provocan la detención del programa, son debidos a la incorrecta programación de algún proceso y como resultado se obtienen datos erróneos. Errores de este tipo son cálculos mal hechos, bucles infinitos, devolución de valores incorrectos, etc. **Como ni los detecta el compilador, ni provocan la interrupción del programa deben revisarse a mano.** Visual Basic, y todos los entornos de programación incorporan **herramientas para facilitar la detección de este tipo de errores, son las herramientas de depuración.**

HERRAMIENTAS PARA CONTROLAR/DEPURAR ERRORES

Errores en Tiempo de Compilación	Errores en tiempo de ejecución	Errores de lógica
Al ejecutar el código el compilador informa en que lugar del código se ha producido el error. • Para corregir el error se debe **revisar la sintaxis del código**.	*Se puede usar:* • ***Rutinas o Funciones de Tratamiento de Errores.***	Herramientas de Depuración: • Ejecutar el código paso a paso con la Tecla **F8**. • Usar el Método **Assert** del Objeto **Debug** para controlar y depurar el código.

¿QUE ES LA RUTINA DE TRATAMIENTO DE ERRORES?

- Generalmente una Rutina de Tratamiento de Errores reacciona ante casos esperados por el programador
- Mediante una Rutina de Tratamiento de Errores informaremos del error que se ha producido y direccionaremos la ejecución del programa hacia donde nos interese.
- En Visual Basic el tratamiento de errores es una parte normal de la ejecución del programa.

La instrucción para el tratamiento de errores es:

ON ERROR

Existen 3 maneras de utilizar la instrucción **On Error**.

Instrucción	Descripción
ON ERROR Goto Línea	Lo usamos para que al momento de que se detecte algún error, pase el control a una línea o a una etiqueta.
ON ERROR Resume Next	Si se detecta un error, se pasará el control a la siguiente línea, omitiendo el anterior.
ON ERROR Goto 0	La usamos para desactivar cualquier manejador de error. Regularmente se usa cuando utilizamos On Error Resume Next.

FORMATO GENERAL PARA IMPLEMENTAR UNA RUTINA DE TRATAMIENTO DE ERRORES

- ***La instrucción para el tratamiento de errores es*** :

ON ERROR GOTO Línea

Línea es una etiqueta o marca de línea que indica hacia donde debe dirigirse el programa en caso de error.

```
Sub prueba()
        On Error GOTO Tratar_errores
        'Instrucciones del procedimiento
        Exit Sub 'Salir del procedimiento
Tratar_Errores:
        'Instrucciones de tratamiento de error

End Sub.
```

Con **On Error GOTO *Tratar_Errores***, indicamos al programa que en caso de ocurrir un error, en tiempo de ejecución, vaya a ejecutar las líneas que siguen a la etiqueta o marca de línea *Tratar_Errores*.

Exit Sub, sirve para evaluar si el procedimiento se ha desarrollado correctamente. En este caso, el procedimiento termina con esta instrucción. Tenga en cuenta que si no se pusiera esta línea la ejecución continuaría secuencialmente y se ejecutarían las líneas de la rutina Tratar_Errores.

OBTENER INFORMACIÓN DEL ERROR: OBJETO ERR

Cuando **On Error Goto <Etiqueta>** se utiliza para capturar los errores, se devuelve un objeto tipo ERR.

El objeto ERR, tiene 2 ***propiedades*** que muestran información sobre el error producido.

Propiedades del objeto ERR.

- ✓ **Number,** es un número indicativo sobre el tipo de error que se ha producido,
- ✓ **Description**, es el mensaje asociado al número de error y es una cadena de texto que describe brevemente el error.

Por lo tanto, cuando Visual Basic dispara un objeto Err en tiempo de ejecución, podemos revisar su propiedad Number para saber que es lo que ha causado el error y actuar en consecuencia.

ERRORES INTERCEPTABLES

Código	Mensaje
3	Return sin Gosub
5	El argumento o la llamada al procedimiento no son válidos
6	Desbordamiento
7	Sin memoria
9	El subíndice está fuera del intervalo
10	La matriz está bloqueada de manera fija o temporal
11	División por cero
13	No coinciden los tipos
14	No hay suficiente espacio de cadena
16	La expresión es demasiado compleja
17	No se pudo realizar la operación solicitada
18	Se produjo una interrupción por parte del usuario
20	Resume sin Error
28	No hay suficiente espacio de pila
35	No se ha definido Sub, Function o Property
52	Nombre o número de archivo incorrecto
53	Archivo no encontrado
54	Modo de archivo incorrecto
55	El archivo ya está abierto
57	Error de E/S de dispositivo
58	El archivo ya existe
59	La longitud de registro es incorrecta

Fuente: Ayuda de Excel

DEFINIR ERRORES PROPIOS.

- Visual Basic incorpora la instrucción **ERROR** con la que es posible definir o generar errores propios.
- La generación de errores propios permite un mayor control en programas largos y complejos, para validar datos, y algunos procesos más.
- Para provocar un error simplemente se tiene que utilizar la instrucción:

Error(*Numero_de_Error*)

Numero_de_Error debe ser un valor comprendido entre **0 y 65535. E**s decir, debe estar dentro del rango de errores definidos por Visual Basic.

- Se recomienda utilizar siempre valores altos ya que sino, corre el riesgo de modificar un valor ya establecido por Visual Basic, por ejemplo a partir del valor 65535 y a continuación ir bajando, estos valores tan elevados nunca serán utilizados por Visual Basic.

CONTINUAR EL PROGRAMA DESPUÉS DE LA EJECUCIÓN DE LA RUTINA DE TRATAMIENTO DE ERRORES: INSTRUCCIÓN RESUMEN

En la mayoría de los casos, las Rutinas de Tratamiento de Errores son para que el programa no se detenga. El proceso habitual cuando se produce el error es tratar de corregirlo y continuar la ejecución del programa.

La continuación del programa se consigue mediante la **Instrucción Resume**. Esta instrucción da tres posibilidades:

Sintaxis:

- **Resume**. Ejecutar de nuevo la instrucción que ha causado el error.
- **Resume Next.** Continuar el programa por la instrucción siguiente a la que ha causado el error.
- **Resume Etiqueta**. Continuar por la instrucción que sigue a la *Etiqueta*.

CONSIDERACIONES A TENER EN CUENTA ANTES DE ESCRIBIR EL CÓDIGO PARA UNA APLICACIÓN

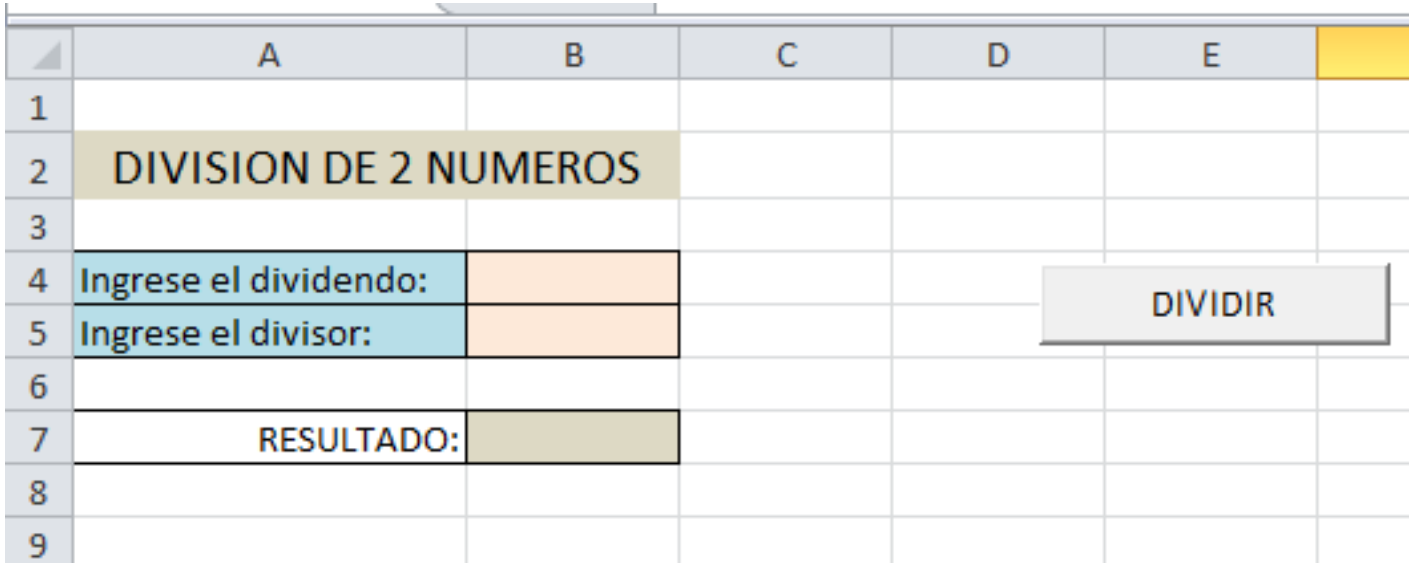
Antes de escribir el código, se debe:

1. Identificar el objetivo, proceso, que debe cumplir la aplicación. Por ejemplo, Aplicación para efectuar la división de 2 números.
2. Identificar los parámetros, y su tipo de dato, que se necesitan para que la aplicación cumpla con su objetivo. Para el ejemplo: Dividendo y Divisor.
3. Definir como se obtendrán los valores de los parámetros. Para el caso de Excel, tenemos 2 alternativas: Tomar estos valores de las celdas de la hoja de calculo o en su defecto solicitar los valores al usuario en tiempo de ejecución.
4. Analizar las posibles situaciones de error que se pueden producir en la lectura de los parámetros y/o en el procesamiento de los datos en base a esos parámetros.
5. Dibujar un Diagrama de Flujo que refleje el flujo de información durante su procesamiento.

Luego de este análisis, podemos empezar a escribir el código tomando como punto de partida el diagrama de flujo.

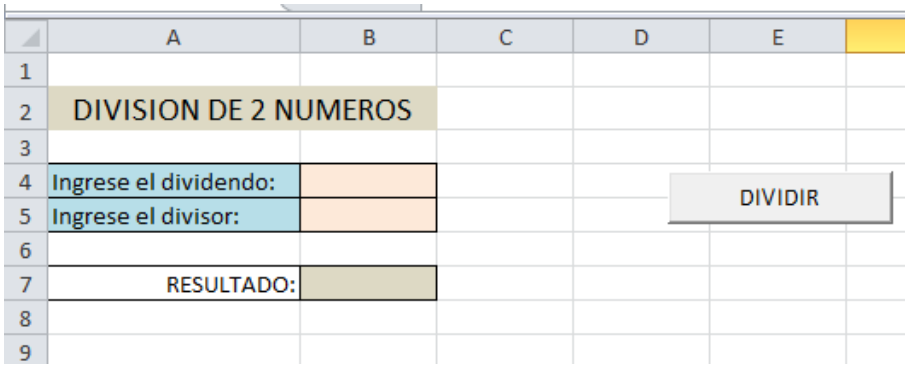
EJEMPLO: DIVISIÓN DE 2 NÚMEROS

Consigna: Escribir una macro que permita efectuar la división de 2 números.



ANÁLISIS DE LA SITUACIÓN PARA EFECTUAR LA DIVISIÓN DE 2 NÚMEROS (PARTE 1)

Objetivo de la aplicación: División de 2 números.



Parámetros para efectuar la división.

- Dividendo: (Real) Celda B4
- Divisor: (Real) Celda B5

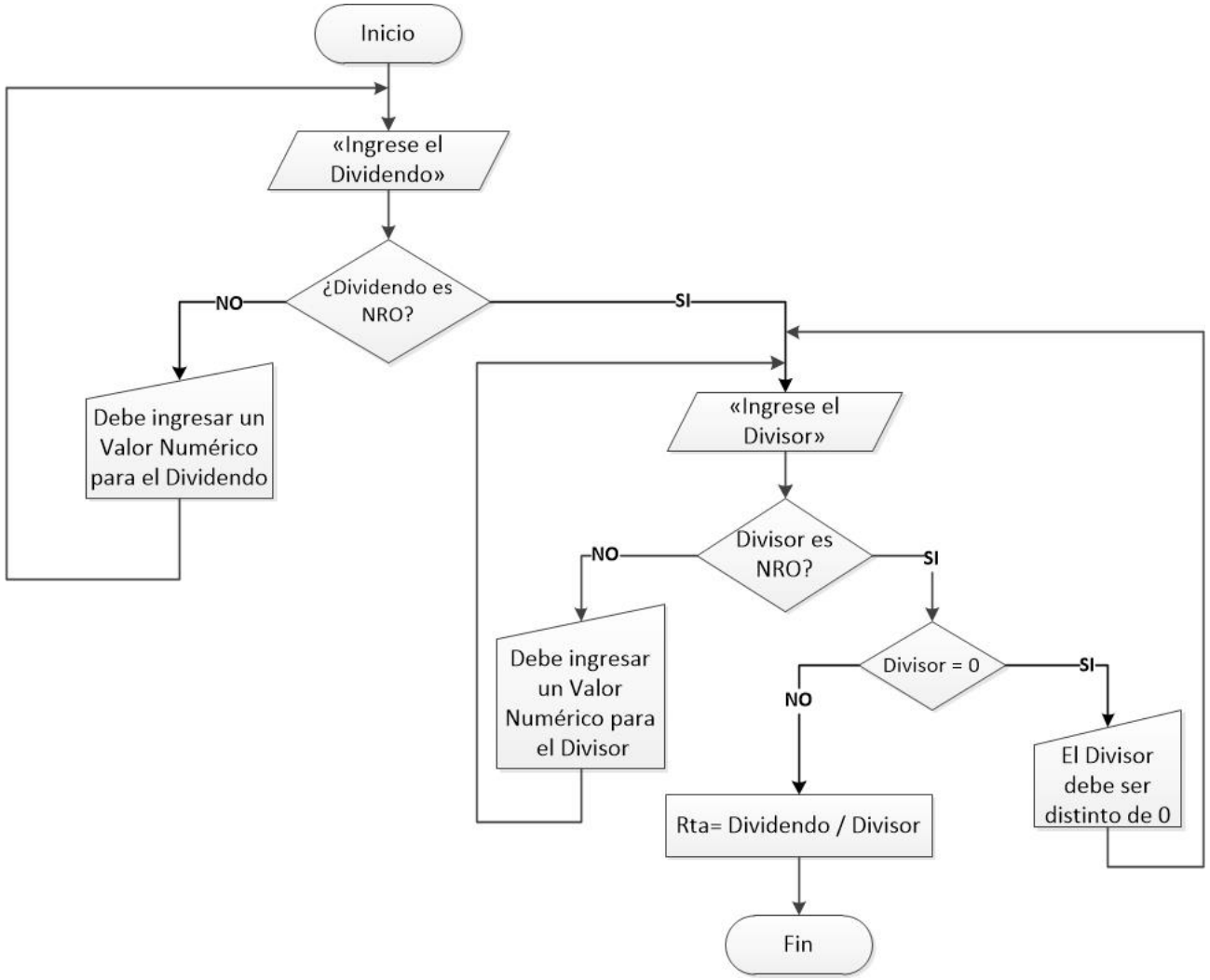
Posibles situaciones de error que se pueden producir al efectuar la división con los parámetros.

Respecto al Dividendo:
- Omisión en el ingreso del valor.
- Valor ingresado no es de tipo numérico.

Respecto al Divisor:
- Omisión en el ingreso del valor.
- Valor ingresado no es de tipo numérico.
- Valor ingresado igual a 0.

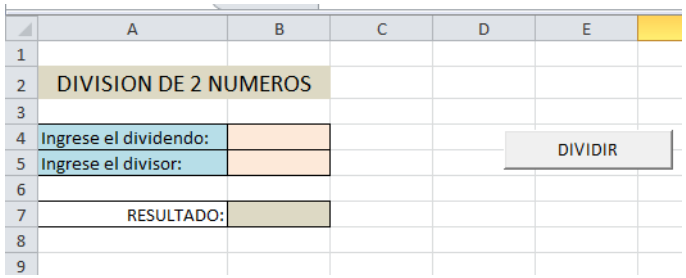
ANÁLISIS DE LA SITUACIÓN PARA EFECTUAR LA DIVISIÓN DE 2 NÚMEROS (PARTE 2)

Diagrama de Flujo.



DIVISIÓN DE 2 NÚMEROS

Resolución Ejemplo 1: Código con Rutina de Tratamiento de Errores con Sentencia Resume



Código controlado mediante la Sentencia ON ERROR GOTO.
Cuando se produce algún error en alguna línea de este código, la ejecución se redirige, según el nro de error producido, a las instrucciones para el Tratamiento de Errores.

Instrucciones para el Tratamiento de Errores

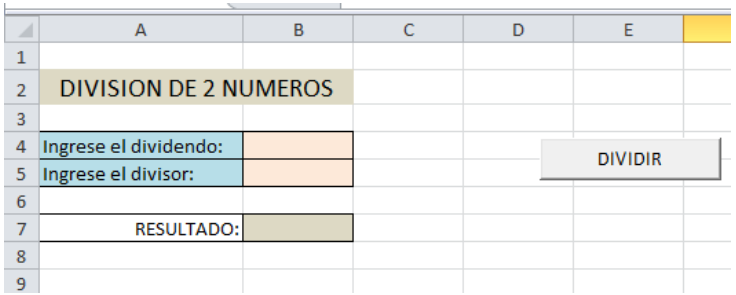
```
Option Explicit

Private Sub cmdDividir_Click()
'Parámetros, Dividendo y Divisor, solicitados en tiempo de ejecucion
Dim Dividendo As Double
Dim Divisor As Double
Dim Texto As String
'Texto, es una var. auxiliar para mostrar
'el mensaje adecuado en caso de ocurrir un error
'durante la asignacion del valor ingresado x el usuario
'a la var dividendo o divisor
On Error GoTo TratarErrores
    Range("B4").Value = ""
    Range("B5").Value = ""
    Range("B7").Value = ""
    Texto = "para el Dividendo"
    Dividendo = InputBox("Ingresar el Dividendo")
    Range("B4").Value = Dividendo
Linea_Divisor:
    Texto = "para el Divisor"
    Divisor = InputBox("Ingresar el Divisor")
    Range("B5").Value = Divisor
    Range("B7").Value = Dividendo / Divisor
Exit Sub

TratarErrores:
    Select Case Err.Number
        Case 13
            MsgBox ("Debe ingresar un Valor Numérico " & Texto)
            Resume
        Case 11
            MsgBox ("El Divisor debe ser distinto de 0")
            Resume Linea_Divisor
        Case Else
            MsgBox ("Se ha producido el Error Nro.: " & Err.Number & vbCrLf & _
            "Descripcion: " & Err.Description)
    End Select
End Sub
```

En este código, el control de las posibles situaciones de error:

- *Durante la asignación de los valores a las variables dividendo y divisor (declaradas como de tipo double (real)), o*
- *Durante el calculo de la división, cuando el divisor es 0,*

Se lleva a cabo mediante la implementación de una Rutina de Tratamiento de Errores como parte del código.
En este caso, la Rutina de Tratamiento de Errores, también considera la posibilidad, ante una situación de error, que el usuario vuelva a reingresar el valor. Esta situación, controlada con el uso de la sentencia Resume.

DIVISIÓN DE 2 NÚMEROS



Código controlado mediante la Sentencia ON ERROR GOTO.
Cuando se produce algún error en alguna línea de este código, la ejecución se redirige, según el nro de error producido, a las instrucciones para el Tratamiento de Errores.
En este caso, los nros de errores son errores propios.

Instrucciones para el Tratamiento de Errores:
Según el nro de error producido, se muestra un mensaje informativo de la situación y se retorna la ejecución del código al punto que indica la etiqueta en la sentencia Resume.

Resolución Ejemplo 2: Código con Rutina de Tratamiento de Errores, con Sentencia Resume y con Códigos de Errores Propios

```
Option Explicit

Private Sub cmdDividir_Click()
'Parámetros, Dividendo y Divisor, solicitados en tiempo de ejecucion
Dim Dividendo As Variant
Dim Divisor As Variant

On Error GoTo TratarErrores
    Range("B4").Value = ""
    Range("B5").Value = ""
    Range("B7").Value = ""
Linea_Dividendo:
    Dividendo = InputBox("Ingresar el Dividendo")
    If IsNumeric(Dividendo) Then
        Range("B4").Value = Dividendo
Linea_Divisor:
        Divisor = InputBox("Ingresar el Divisor")
        If IsNumeric(Divisor) Then
            If Divisor = 0 Then
                Error (65533)
            Else
                Range("B5").Value = Divisor
                Range("B7").Value = Dividendo / Divisor
            End If
        Else
            Error (65534)
        End If
    Else
        Error (65535)
    End If
Exit Sub

TratarErrores:
    Select Case Err.Number
        Case 65535
            MsgBox ("Debe ingresar un Valor Numérico para el Dividendo")
            Resume Linea_Dividendo
        Case 65534
            MsgBox ("Debe ingresar un Valor Numérico para el Divisor")
            Resume Linea_Divisor
        Case 65533
            MsgBox ("El Divisor debe ser distinto de 0")
            Resume Linea_Divisor
        Case Else
            MsgBox ("Se ha producido el Error Nro.: " & Err.Number & vbCrLf & _
            "Descripcion: " & Err.Description)
    End Select
End Sub
```

***Observación**: En este código, para evitar la posibilidad de error durante la asignación de los valores al dividendo y al divisor, se declara las variables como de tipo Variant.*

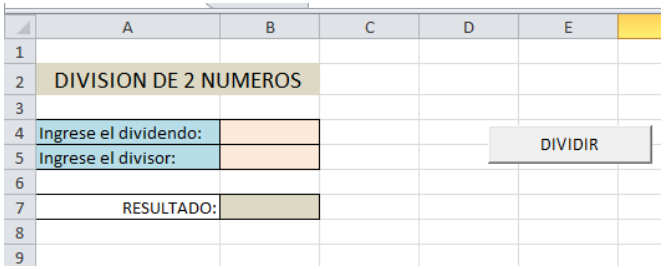
DIVISIÓN DE 2 NÚMEROS

Alternativa de solución: Uso de Estructuras de Control de Repetición y/o Evento Change de Worksheet.

- Para el ejemplo que se está trabajando, División de 2 números, las situaciones de error que se pueden producir al realizar la operación, también pueden ser controladas mediante el uso de Estructuras de Control de Repetición y/o el uso del Evento Change de Worksheet.
- En las siguientes diapositivas se presentan 2 alternativas de solución para el código.

DIVISIÓN DE 2 NÚMEROS

Resolución Ejemplo 3: Código con Estructura de Control de Repetición DO LOOP UNTIL



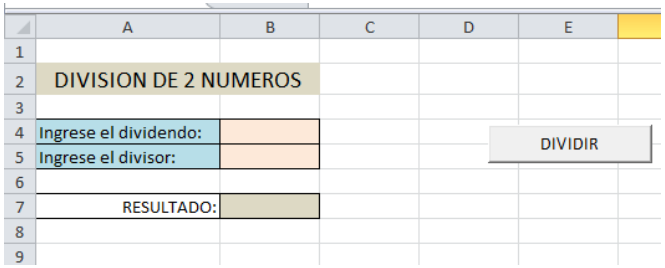
En este ejemplo, para evitar la posibilidad de error durante la asignación de los valores del dividendo y divisor (solicitados en tiempo de ejecución), se declara las variables como de tipo Variant.
La validación del tipo de dato para los parámetros se lleva a cabo mediante la Estructura de Control de Repetición DO LOOP UNTIL.

```
Option Explicit

Private Sub cmdDividir_Click()
'Parámetros, Dividendo y Divisor, solicitados en tiempo de ejecucion
Dim Dividendo As Variant
Dim Divisor As Variant
Dim Band As Boolean

Range("B4").Value = ""
Range("B5").Value = ""
Range("B7").Value = ""
Band = False
Do
    Dividendo = InputBox("Ingresar el Dividendo")
    If IsNumeric(Dividendo) = False Then
        MsgBox ("Debe ingresar un Valor Numérico para el Dividendo")
        Band = False
    Else
        Range("B4").Value = Dividendo
        Band = True
    End If
Loop Until Band
Band = False
Do
    Divisor = InputBox("Ingresar el Divisor")
    If IsNumeric(Divisor) = False Then
        MsgBox ("Debe ingresar un Valor Numérico para el Divisor")
        Band = False
    Else
        If Divisor = 0 Then
            MsgBox ("El Divisor debe ser distinto de 0")
            Band = False
        Else
            Range("B5").Value = Divisor
            Range("B7").Value = Dividendo / Divisor
            Band = True
        End If
    End If
Loop Until Band

End Sub
```

DIVISIÓN DE 2 NÚMEROS

Resolución Ejemplo 4: Código con Estructura de Control de Repetición DO LOOP UNTIL y Evento Change de Worksheet

Procedimiento principal para efectuar la operación de la división, que para el caso solo se encarga de: asignar los valores de los parámetros ingresados por el usuario a las celdas correspondientes y verificar que los mismos tengan algún valor distinto de vacío.



En este ejemplo, la validación del tipo de dato requerido para el dividendo y el divisor se realiza en el momento de asignar el valor ingresado; por el usuario; a las celdas correspondientes, momento en el que se activa el Evento Change de Worksheets para validar el valor ingresado.

```
Option Explicit

Private Sub cmdDividir_Click()
'Parámetros, Dividendo y Divisor, solicitados en tiempo de ejecucion
Dim Dividendo As Double
Dim Divisor As Double

    Range("B4").Value = ""
    Range("B5").Value = ""

    Do
        Range("B4").Value = InputBox("Ingresar el Dividendo")
    Loop Until Not IsEmpty(Range("B4").Value)
    Dividendo = Range("B4").Value

    Do
        Range("B5").Value = InputBox("Ingresar el Divisor")
    Loop Until Not IsEmpty(Range("B5").Value)
    Divisor = Range("B5").Value

    Range("B7").Value = Dividendo / Divisor
End Sub
```

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Select Case Target.Address
        Case "$B$4"
            Range("B7").Value = Empty
            If Target.Value <> "" Then
                If IsNumeric(Target.Value) = False Then
                    MsgBox ("Debe ingresar un Valor Numérico para el Dividendo")
                    Target.Value = Empty
                End If
            End If
        Case "$B$5"
            Range("B7").Value = Empty
            If Target.Value <> "" Then
                If IsNumeric(Target.Value) = True Then
                    If Target.Value = 0 Then
                        MsgBox ("El Divisor debe ser distinto de 0")
                        Target.Value = Empty
                    End If
                Else
                    MsgBox ("Debe ingresar un Valor Numérico para el Divisor")
                    Target.Value = Empty
                End If
            End If
    End Select
End Sub
```

El Objeto Debug y su Método Assert

- **El Método Assert del Objeto Debug** nos ayuda a controlar y depurar nuestros fallos de programación en VBA para Excel.
- ***Debug.Assert*** provoca que el código entre en modo de interrupción en tiempo de ejecución si una condición específica es falsa.
- El formato para una llamada a Debug.Assert en el código es:

Debug.Assert (Condición Lógica)

Donde condición lógica es cualquier expresión que se evalúe como un valor Verdadero o Falso.

Ejemplo:

```
Option Explicit
Sub ControlErrores_Assert_1()

Dim Valor As Integer

Valor = InputBox("Introducir un valor positivo")

'Se detendrá/depurará si la condición es Falsa
'(Por Ej: si es negativo el dato introducido)

Debug.Assert (Valor >= 0)

MsgBox ("Dato Introducido: " & Valor)

End Sub
```