Comencemos a programar con

VBA - Access

Entrega 04

Primeros conceptos 2

Eduardo Olaz

Funciones

En la entrega anterior hemos visto una introducción a los procedimientos `Sub`.

Otro de los tipos de procedimientos son los procedimientos `Function`, que al igual que los procedimientos Sub están delimitados por `Function` y `End Function`.

La principal diferencia entre un procedimiento Sub y un procedimiento Function es que este último devuelve un valor.

Entre los dos delimitadores se escribe el código que va a realizar las tareas que deseemos, y al final devolverá algún valor.

Son las llamadas Funciones.

Veamos la forma como lo explica la ayuda de Access.

Si vamos a la ventana Inmediato, escribimos Function, ponemos el cursor en lo escrito y presionamos [F1], entre otras cosas nos aparece la sintaxis de su declaración:

[**Public** | **Private** | **Friend**] [**Static**] **Function** *nombre* [(*lista_argumentos*)] [**As** *tipo*]
[*instrucciones*]
[*nombre* = *expresión*]
[**Exit Function**]
[*instrucciones*]
[*nombre* = *expresión*]

End Function

Vemos que para declarar una función empezaríamos, por ejemplo poniendo la palabra Public (si quisiéramos que la función sea accesible desde cualquier parte de Access) a continuación la palabra Function, después abriríamos un paréntesis y pondríamos los parámetros que necesitáramos, cerraríamos el paréntesis y finalmente pondríamos el tipo de datos que devolvería la función.

Supongamos que quisiéramos escribir una función en la que pasándole el ángulo y el radio, nos devolviera la longitud de un arco de circunferencia.

La cabecera de esta función sería:

```
Public Function Arco(Angulo As Single, Radio As Single) As Single
```

Hay un truco para que una línea con mucho texto se pueda dividir en varias líneas, sin que deje de ser una única línea.

Se coloca al final de la línea que se quiere dividir un espacio en blanco y la raya de subrayado.

Con ello la línea anterior se podría poner así:

```
Public Function Arco( _
                Angulo As Single, _
                Radio As Single _
                ) As Single
```

Aquí declaramos la función como pública, para que se pueda utilizar desde cualquier parte de la aplicación, siempre que se escriba en un módulo normal.

A continuación escribimos la palabra Function, seguida de su nombre, en este caso Arco.

Abrimos paréntesis y escribimos los dos parámetros que necesitamos, Angulo y Radio, declarándolos del tipo Single (tipo numérico de coma flotante de 4 Bytes).

Cerramos el paréntesis y declaramos la función Arco también del tipo Single.

Si escribís la cabecera de la función en un módulo normal, veréis que VBA, al igual que pasaba con los procedimientos Sub, os añade automáticamente el final correspondiente, es decir

```
End Function
```

Con esto la función quedaría así:

```
Public Function Arco( _
                Angulo As Single, _
                Radio As Single _
            ) As Single


End Function
```

Por ahora no es gran cosa, ya que no hace nada.

Permitidme que escriba el código completo de esta función y os la explique paso a paso.

Como en la entrega anterior, voy a declarar la constante Pi como pública en la cabecera del módulo, para que pueda ser utilizada por la aplicación.

Si ya la tuviera declarada en otro módulo este paso no sería necesario.

El código sería el siguiente:

```
Option Compare Database
Option Explicit

Public Const Pi as Double = 3.14159265358979

Public Function Arco( _
                Angulo As Single, _
                Radio As Single _
            ) As Single
     Arco = Pi * Radio * Angulo / 180
End Function
```

Aparte de la fórmula matemática de Trigonometría elemental, vemos lo siguiente.

Asignamos al nombre de la función `Arco` el resultado de la fórmula de la izquierda.

Esto hará que la función `Arco` devuelva el resultado de la operación matemática.

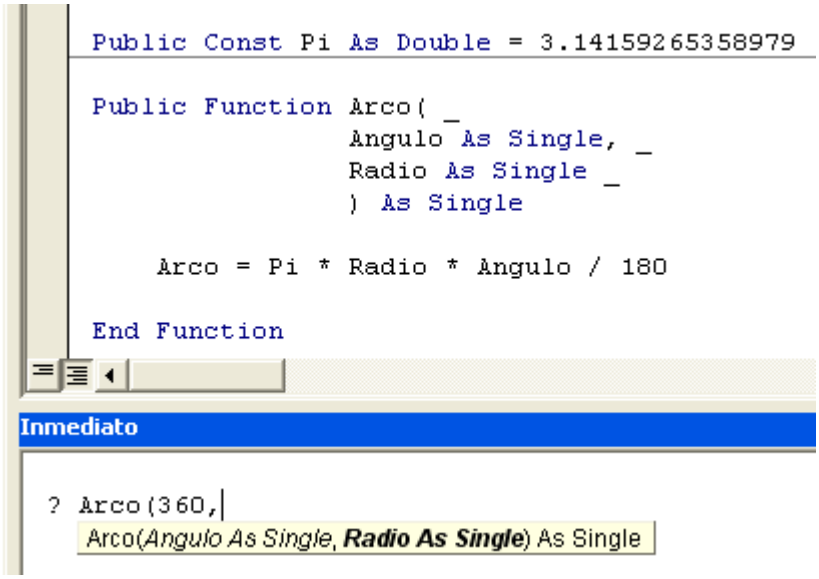
Vamos a centrarnos en lo que aparece entre **Function** y el final de la línea.

Vamos a probar la función.

Para ello escribiremos en la ventana Inmediato.

```
? Arco(360,1)
```

Supongo que ya no será sorpresa para vosotros la ayuda en línea que os va prestando el propio VBA de Access:



Tras completar la expresión, y darle a [Enter] nos escribirá: **6,283185** que es la longitud de un arco de radio 1 y ángulo 360º, lo que equivale a la circunferencia completa.

Vamos a hacer un experimento: ¿Qué pasa si cambiamos la declaración de tipo de esta función?

Vamos a cambiar el Single de la declaración de la cabecera, por Double.

Double es un número de coma flotante de más precisión que single (8 Bytes de Double frente a 4 de single). La cabecera quedará así:

```
Public Function Arco( _
                Angulo As Single, _
                Radio As Single _
            ) As Double
```

Para ello escribiremos **? Arco(360,1)** en la ventana Inmediato, y presionamos [Enter].

Efectivamente la precisión ha aumentado, ya que el valor escrito ha sido:

6,28318530717958

Por ello si cambiamos ahora la cabecera:

```
Public Function Arco( _
                Angulo As Double, _
                Radio As Double _
            ) As Double
```

Se supone que los cálculos serán más precisos.

Efectivamente así es, pero en una aplicación crítica deberíamos jugar que el tipo de precisión adecuada, para hacer la aplicación lo más ligera posible.

Si probamos a hacer el cambio vemos que en nuestra función, con los datos anteriores apenas cambia el resultado. Pero ¿qué pasaría si estuviéramos programando la órbita de una nave con destino a Marte? En este caso la precisión sería un elemento clave.

Funciones en formularios

Vamos a hacer una pequeña y elemental calculadora que nos permita sumar dos cifras.

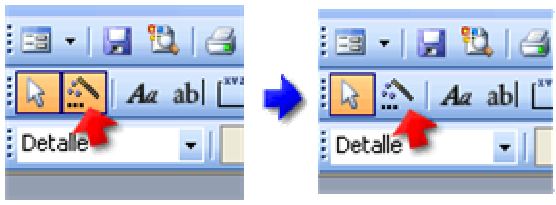
Cerramos los módulos y creamos, ó abrimos un formulario en modo diseño.

Este formulario no lo enlazamos con ninguna tabla ni consulta.

Ahora vamos a deshabilitar el asistente para controles.

En la barra de controles, pulsamos en la “Varita mágica” para deshabilitarla.

Deberá quedar sin resaltar, como se indica en el gráfico con las flechas rojas.



Ponemos dos cuadros de texto en el formulario.

Por defecto, Access les pondrá un nombre semejante a Texto2 y Texto2.

Vamos a cambiar esos nombres.

Abrimos la hoja de propiedades, os recuerdo Menú Ver > Propiedades, ó pulsando las teclas [Alt] + [Enter].

En la pestaña [Otras] ponemos el cursor en la primera propiedad, que “casualmente” es la propiedad Nombre. Ahí es donde, pulsando en los correspondientes cuadros de texto, vamos a cambiar su nombre.

Les ponemos, al primer Cuadro de texto el nombre de txtSumando_1, y al segundo txtSumando_2.

Ahora le vamos a poner un botón, cambiando su nombre , algo así como Comando5, por cmdSumar, y su título ó texto del botón, por **Sumar**.

> Al programar es muy importante que los objetos, variables, constantes, procedimientos, etc. tengan nombres con cierto sentido.
>
> Si no fuese así, si más adelante tuvieras que revisar tu código, o si lo tuviese que hacer otra persona, tendríais que invertir una gran cantidad de tiempo en averiguar qué es y qué hace cada cosa en el código.

Más adelante hablaremos sobre normalización de código, pero fíjate que el nombre de los cuadros de texto está precedido por las letras **txt** y el del botón por **cmd**.

Así si en un pedazo de código veo una palabra que empieza por **cmd** sé que se refiere a un botón, y si empieza por **txt** sé que se refiere a un cuadro de texto.

Otro ejemplo: Los nombres de las etiquetas los suelo empezar por **lbl**.

Estas son **convenciones de normalización** “más ó menos” estándares.

A partir de este momento las iré aplicando al código y veréis como su uso se irá convirtiendo en algo cada vez más natural.

A lo que íbamos.

Tenemos en un formulario los siguientes elementos:

txtSumando_1	Cuadro de texto
txtSumando_2	Cuadro de texto
cmdSumar	Botón de comando

Probablemente al poner los cuadros de texto se nos hayan colocado dos etiquetas.

De momento no vamos a cambiar su nombre, pero les vamos a poner como Título

Operando 1

Operando 2

Ahora vamos a poner una etiqueta nueva, con título Resultado, y de nombre lblResultado.

Más de uno se habrá dado cuenta que lbl viene de **Label** (etiqueta en inglés), txt de **Text** y cmd de **Command**.

A esta etiqueta le vamos a poner un texto un poco más grande y como color de texto el rojo.

¿Pero qué quieres que hagamos con todo esto?

Tan simple como tener un formulario en el que escribir dos números en las correspondientes casillas, y que al pulsar un botón nos muestre su suma.

¿Y cómo sabe Access cuando se ha pulsado el botón y qué es lo que tiene que hacer?

Para eso utilizaremos el Evento **Clic** del botón cmdSumar.

Ya he comentado que un Evento es un "*mensaje*" que manda Windows, *cuando pasa algo*, que puede ser captado por Access y a su vez reenviado a Windows con un código que se tiene que ejecutar.

Seleccionamos el botón, y en la [Hoja de Propiedades] pulsamos en la pestaña Eventos.

Seleccionamos el cuadro correspondiente a "Al hacer clic" y pulsamos en el pequeño botón con tres puntos que nos aparece.

A continuación veremos una pequeña pantalla que nos permite seleccionar entre

- Generador de expresiones
- Generador de macros
- Generador de código

Lógicamente seleccionamos **Generador de código**, y pulsamos [**Aceptar**] ó doble clic en [Generador de código].

Inmediatamente se nos abre el editor de código con el siguiente código

```
Option Compare Database
Option Explicit

Private Sub cmdSumar_Click()

End Sub
```

El cursor estará seguramente posicionado entre las dos líneas.

¿Qué es esto?

Es el procedimiento que captura el evento Clic del botón **cmdSumar**.

Ahora lo que quiero es que tras poner un valor en cada uno de los cuadros de texto, si pulsamos el botón, nos muestre el resultado de la suma en la etiqueta lblResultado.

El contenido de un cuadro de texto lo maneja su propiedad **`Value`**, que se puede leer y escribir.

El contenido de una etiqueta (su título) lo maneja su propiedad **`Caption`**, y también es de lectura y Escritura.

Vamos al curro.

Escribimos lo siguiente

```
Private Sub cmdSumar_Click()
Me.lblResultado.Caption = Me.txtSumando_1.Value + Me.txtSumando_2.Value
End Sub
```

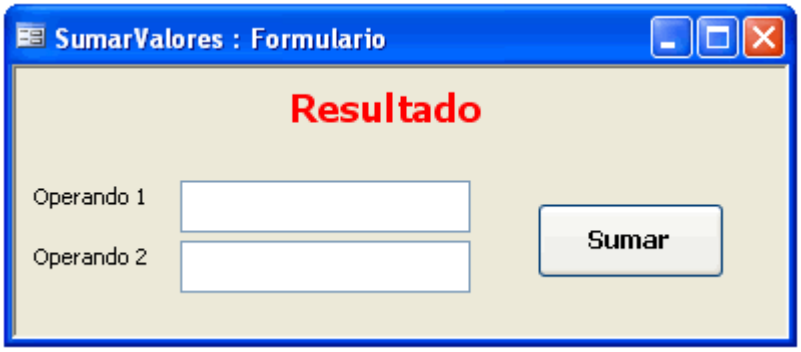
Que traducido “al cristiano” quiere decir:

Cuando se presione sobre el botón `cmdSumar` (evento clic) pon como título de la etiqueta `lblResultado` (propiedad **caption**) el resultado de sumar el contenido (propiedad **Value**) del cuadro de texto `txtSumando_1` y del cuadro de texto `txtSumando_2`.

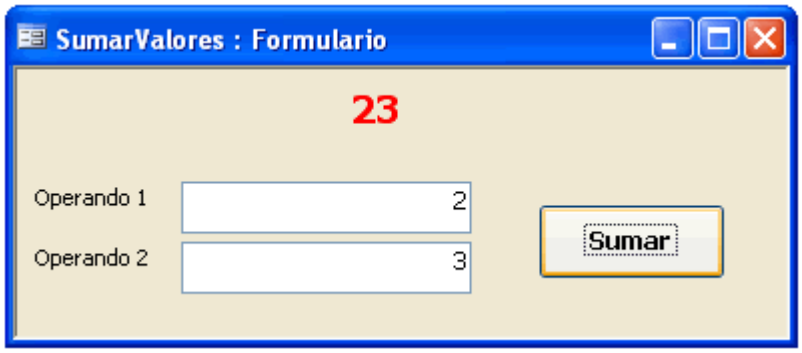
El control Cuadro de texto es un TextBox y la Etiqueta un control Label.

Bueno, grabamos el código y el formulario y lo abrimos.

Nos quedará algo así como



Si ahora introducimos, por ejemplo el valor 2 en el primer cuadro de texto, el valor 3 en el segundo y presionamos el botón Sumar, nos llevamos una pequeña sorpresa:



¿Qué ha pasado?.

¿Por qué en vez de un 5 nos da 23? al fin y al cabo 2 más 3 son 5

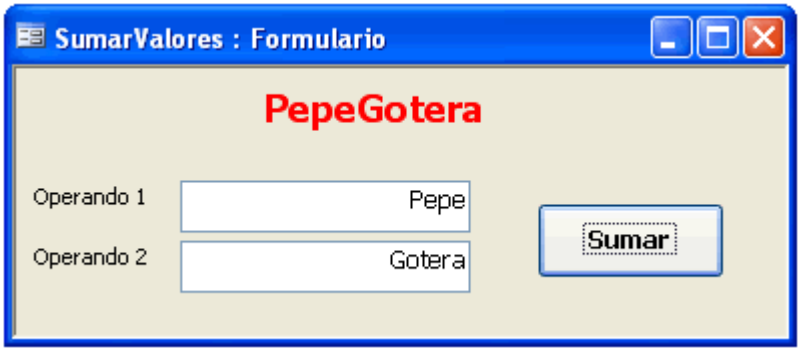
Estas son algunas de las cosillas que desaniman a los principiantes…

No es que Access nos esté tomando el pelo. Recordemos que estamos hablando del contenido de un [Cuadro de Texto], y que como tal todo lo que contiene lo interpreta como texto.

Aunque hemos introducido un 2 para el control es como si introdujéramos el texto “2”.

Vamos a probar a introducir en el primer Cuadro de texto **Pepe**, y en el segundo **Gotera**.

Al presionar el botón nos da



Un par de párrafos más adelante veremos cómo solucionar este problema.

Primero un par de matizaciones sobre el código.

Tomemos el texto

```
Me.lblResultado.Caption
```

Vamos a desmenuzarlo.

`Me` representa el **formulario** al que pertenece el código.

```
Me.lblResultado
```

`lblResultado` representa a la **etiqueta** `lblResultado` del formulario actual.

```
Me.lblResultado.Caption
```

`Caption` representa a la propiedad **Título** de la etiqueta.

La utilización de Me no es necesaria; pero facilita a la ayuda en línea el que te presente las diferentes opciones a escribir.

Tomemos ahora el texto

```
Me.txtSumando_1.Value
```

Ya hemos visto que me representa al formulario.

```
Me.txtSumando_1
```

`txtSumando_1` representa al **cuadro de texto** `txtSumando_1`

```
Me.txtSumando_1.Value
```

`Value` representa el contenido del cuadro de texto ó la propiedad **`Value`**.

La propiedad Value de los cuadros de texto tiene dos peculiaridades:

La primera es que es la propiedad por defecto del control.

Eso quiere decir, que si se escribe en el código el nombre del cuadro de texto, sin especificar nada más, hará implícitamente referencia a la propiedad Value.

Por ello son equivalente es igual de válidos los siguiente códigos:

```
Me.lblResultado.Caption = Me.txtSumando_1.Value + Me.txtSumando_2.Value
```

```
    lblResultado.Caption = txtSumando_1 + Me.txtSumando_2
```

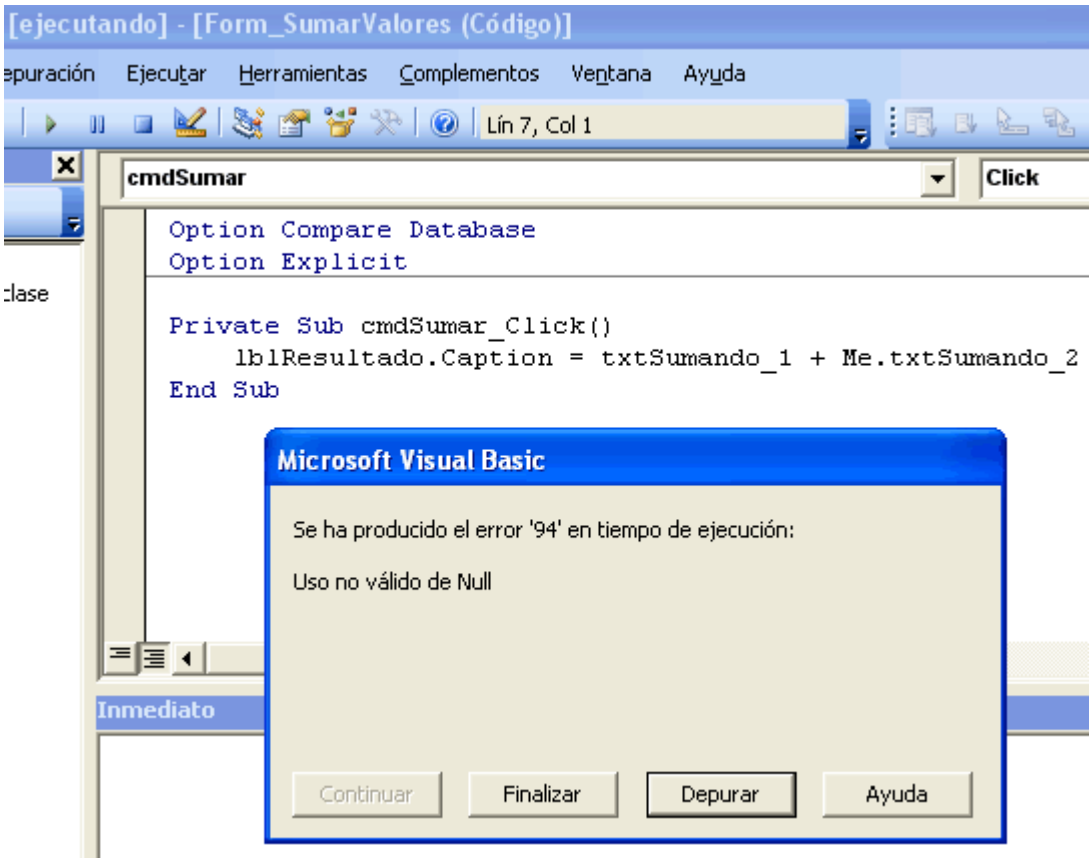
He quitado las referencias al formulario **`Me`** y la propiedad **`Value`**.

Cambiamos el código anterior por éste:

```
Private Sub cmdSumar_Click()
    lblResultado.Caption = txtSumando_1 + Me.txtSumando_2
End Sub
```

Ahora vamos a ejecutar el formulario, pero antes de presionar el botón de sumar dejamos uno de los cuadros de texto sin nada escrito en él.

¡Un nuevo problema!...



Esto hace referencia a la segunda peculiaridad de los cuadros de texto:

Si el cuadro de texto contiene algún valor, nos devolverá un texto con ese valor, pero si no contiene nada nos devolverá el valor **Nulo** (**Null**).

Para poder funcionar así, la propiedad Value del cuadro de texto tiene que ser del tipo **Variant**. Sí ese tipo de dato capaz de tragarlo todo.

Un valor nulo no es cero, ni una cadena de texto vacía. Un valor nulo en realidad se corresponde al resultado del código `ASCII 0`. Por el momento no os importe no saber qué significa esto.

Como uno de los cuadros de texto tenía el valor **Nulo** (**Null**), VBA no puede sumarlo al contenido del otro cuadro de texto. Podríamos decir que un nulo no se puede sumar con nada.

¿Pero qué tenemos que hacer para poder sumar los datos?

Primero deberíamos convertir los valores contenidos en los cuadros de texto a valores numéricos.

Para efectuar esa conversión existe una función de VBA llamada **`Val(Valor)`**

Si escribimos en la ventana inmediato la palabra Val, ponemos en ella el cursor y presionamos [**F1**] nos aparece la ayuda de la función Val.

Con la siguiente información:

> Devuelve los números contenidos en una cadena como un valor numérico del tipo adecuado.
>
> **Sintaxis**
>
> **Val(***cadena***)**
>
> El argumento obligatorio *cadena* es cualquier expresión de cadena válida.

Vamos a hacer pruebas en la ventana Inmediato.

Escribimos las sucesivas expresiones y presionamos [Enter]

Expresión	**Resultado impreso**
? "2" + "3"	`23`
? 2 + 3	`5`
? Val("2") + Val("3")	`5`
? Val("12.234")	`12,234`
? Val(" 23 Caballos")	`23`
? Val(" 3,1416")	`3`
? Val(“Pepe”)	`0`
? Val(Null)	Genera un error (**Nulo** no tiene un valor numérico)
? Val(Nz(Null,""))	`0`
? Nz(Null,”Valor Nulo”)	`Valor Nulo`
? Nz(“Pepe”,”Valor Nulo”)	`Pepe`

Después de estos pequeños experimentos dejo que saquéis vuestras propias conclusiones.

Vemos aquí una cosa nueva, la función `Nz(Valor)`.

Vamos a escribir **`Nz`** en la ventana Inmediato y pulsamos [**F1**].

Pone algo semejante a esto

> **Nz**(ValorVariant[,ValorSiEsNulo])

El resumen de la información es que la función Nz devuelve uno de esto dos tipos de datos.

Si el primer argumento no es nulo, devuelve el primer argumento.

Si fuera nulo, devolvería el valor del segundo argumento.

Y con esto ¿qué conseguimos?.

Vamos a efectuar un nuevo cambio en el evento Clic del botón y escribimos lo siguiente:

```
Private Sub cmdSumar_Click()
    Dim dblSumando1 As Double
    Dim dblSumando2 As Double

    dblSumando1 = Val(Nz(txtSumando_1, ""))
    dblSumando2 = Val(Nz(txtSumando_2, ""))

    lblResultado.Caption = CStr(dblSumando1 + dblSumando2)
End Sub
```

Si ahora no escribimos nada en alguno, o los dos cuadros de texto, ya no nos genera el error.

Igualmente si escribimos valores numéricos, nos muestra el dato correcto.

Si escribimos un dato no numérico lo cambia al valor Cero, ó al resultado de Val(Expresión).

Tenemos una nueva función.

En la penúltima línea del código vemos:

```
lblResultado.Caption = CStr(dblSumando1 + dblSumando2)
```

La función **`CStr(Valor)`** convierte un valor numérico a un texto, para asignarlo a la propiedad **`caption`** de la etiqueta **`lblResultado`**., que sólo admite textos.

Este último paso no es estrictamente necesario, ya que VBA efectúa implícitamente la conversión.

Fijaros ahora que en el código repetimos dos veces la expresión `Val(Nz(...`

Si pensamos un poco vemos que nos resultaría interesante crearnos una función más avanzada que Val, que nos devolviera un valor numérico como ésta pero que no diera error si se le pasa un valor nulo.

Podríamos llamarla ValorNumerico(Valor) y devolvería, por ejemplo un valor de tipo coma flotante Doble.

Como deseamos que se pueda utilizar desde cualquier parte de Access la haremos pública y la escribiremos en un módulo Estándar y el código sería algo así:

```
Public Function ValorNumerico ( _
                    Valor As Variant _
                    ) As Double
     ValorNumerico = Val(Nz(Valor,""))
End Function
```

Por cierto, las dos comillas seguidas ”” equivalen a una cadena de texto de longitud 0.

No confundir con Null. Ya se que es lioso, pero por ahora creedme.

Esta función devuelve el valor Cero cuando le pasamos un valor que sea texto no numérico, o un valor nulo.

El parámetro `Valor` se declara como `Variant` para que pueda admitir tanto Cadenas de Texto, como Números e incluso el valor `Null`.

Os recuerdo que la función la debéis escribir en un módulo estándar si la queréis usar desde cualquier otra parte de Access.

Tras esto podríamos simplificar el código del formulario y cambiarlo por el siguiente:

```
Private Sub cmdSumar_Click()
     Dim dblSumando1 As Double
     Dim dblSumando2 As Double

      dblSumando1 = ValorNumerico(txtSumando_1)
     dblSumando2 = ValorNumerico(txtSumando_2)
     lblResultado.Caption = CStr(dblSumando1 + dblSumando2)
End Sub
```

Ya se que todo esto os puede resultar un poco indigesto, pero haciendo pruebas es como mejor lo podréis asimilar.

En la próxima entrega haremos un repaso y ampliación de lo visto hasta ahora.