

Comencemos a programar con
VBA - Access

Entrega **12**

**Trabajando con
procedimientos**

Procedimientos

Hemos hablado sobre diversos aspectos de los procedimientos **Function** y **Sub**.

Relacionados con ellos, he dejado en el tintero completar la explicación de una serie de temas que vamos a tratar en esta entrega.

Entre ellos cabe destacar

- Declaración de variables como **Static**, a nivel de procedimiento
- Paso de parámetros **Por Valor** y **Por Referencia**, **ByVal** y **ByRef**
- Parámetros **Opcionales** **Optional**
- Resolución de problemas mediante Procedimientos **Recursivos** e **Iterativos**.
- Parámetros **Tipo Matriz** mediante **ParamArray**.
- Paso de parámetros **Con Nombre** mediante el operador de asignación **:=**
- Constantes **Enumeradas** **Enum** . . . **End Enum**
- Cuadros de diálogo predefinidos para intercambiar información entre VBA y el usuario **MsgBox** e **InputBox**

Variables Static

A nivel de procedimiento hemos comentado que una variable puede declararse con **Dim**.

No puede declararse ni como **Public** ni como **Private**.

Declarada con **Dim**, cuando acaba el procedimiento, el contenido de la variable desaparece, y la siguiente vez que se llama al procedimiento, tras su declaración, la variable se reinicia con el valor por defecto correspondiente a su tipo.

Como aclaración de esto último, por ejemplo, si tenemos el procedimiento:

```
Public Sub ValorConDim()  
    Dim n As Long  
    n = n + 1  
    Debug.Print "Valor de n: " & n  
End Sub
```

Cada vez que llamemos al procedimiento `ValorConDim` imprimirá en la ventana Inmediato

```
Valor de n: 1
```

Vamos ahora a escribir un nuevo procedimiento muy parecido al anterior

```
Public Sub ValorConStatic()  
    Static n As Long  
    n = n + 1  
    Debug.Print "Valor de n: " & n  
End Sub
```

Sólo cambia el nombre y la declaración de `n` que está declarada como **Static** en vez de con **Dim**.

Ahora la primera vez que se ejecuta muestra también `Valor de n: 1`

Pero las siguientes veces imprimirá:

```
Valor de n: 2
Valor de n: 3
Valor de n: 4
. . .
```

Comprobamos que cuando una variable es declarada como **Static**, se mantiene el valor que va tomando después de ejecutarse el procedimiento.

Vamos a usar ahora el primer procedimiento, cambiando sólo su cabecera

```
Public Static Sub ValorConStatic2()
    Dim n As Long
    n = n + 1
    Debug.Print "Valor de n: " & n
End Sub
```

Si ejecutamos ahora varias veces seguidas el procedimiento `ValorConStatic2` veremos que se comporta igual que en `ValorConStatic`, a pesar de haber declarado la variable `n` con **Dim**.

Esto ocurre porque al poner delante de **Sub** ó **Function**, la palabra **Static**, hace que las variables declaradas dentro del procedimiento, sean declaradas como **Static**, y por tanto mantengan su valor entre diferentes llamadas al mismo.

Paso de parámetros Por Valor y Por Referencia

A lo largo de los ejemplos de las entregas anteriores, hemos utilizado los argumentos **ByVal** y **ByRef**, en la declaración de los parámetros en la cabecera de los procedimientos.

Para averiguar lo que implica declararlas de una u otra manera vamos a escribir el siguiente código.

```
Public Sub DemoByValByRef()
    Dim lngN As Long, lngM As Long, lngO As Long
    lngM = 1
    lngN = 1
    lngO = 1
    PorValor lngM
    PorReferencia lngN
    PorDefecto lngO
    Debug.Print
    Debug.Print "lngM = " & lngM
    Debug.Print "lngN = " & lngN
    Debug.Print "lngO = " & lngO
End Sub

Public Sub PorValor(ByVal VariableByVal As Long)
    VariableByVal = 2 * VariableByVal
    Debug.Print "VariableByVal = " & VariableByVal
End Sub
```

```
Public Sub PorReferencia(ByRef VariableByRef As Long)
    VariableByRef = 2 * VariableByRef
    Debug.Print "VariableByRef = " & VariableByRef
End Sub
```

```
Public Sub PorDefecto(Variable As Long)
    Variable = 2 * Variable
    Debug.Print "Variable = " & Variable
End Sub
```

Si ejecutamos el procedimiento `DemoByValByRef`, nos imprimirá:

```
VariableByVal = 2
VariableByRef = 2
Variable = 2
```

```
lngM = 1
lngN = 2
lngO = 2
```

Vemos que `lngM` no ha cambiado su valor. Mantiene el valor 1 que se ha pasado al procedimiento `PorValor`.

En cambio tanto `lngM` como `lngO` han pasado de tener el valor 1 a tomar el valor 2.

Esto es así porque cuando se pasa una variable **Por Valor** a un procedimiento, se está haciendo que el procedimiento trabaje con una copia de la variable pasada al procedimiento, no con la variable misma. Por ello, los cambios que hagamos al valor del parámetro en el procedimiento, no afectarán a la variable original.

En cambio, si pasamos una variable a un procedimiento, **Por Referencia**, los cambios que hagamos en el parámetro se verán reflejados en la variable original.

Tanto el **VBA** de **Access** como el **VBA** de **Visual Basic**, o de otras aplicaciones, pasan por defecto los parámetros **Por Referencia**.

Esto no ocurre con otros programas, como **Pascal**, **C** o la plataforma de desarrollo **Net**, con programas como **VB.Net**, y **C#**, en las que la forma por defecto es **Por Valor**.

El hecho de que por defecto se pasen los parámetros por referencia puede dar lugar a errores de código difícilmente detectables. Por ello, y como norma general, recomiendo que cuando se pase un parámetro se indique siempre si es **Por Valor** o **Por Referencia** usando **ByVal** o **ByRef**.

No todo son inconvenientes, el paso de parámetros **Por Referencia**, permite que un procedimiento **Sub** o **Function** modifiquen varios valores simultáneamente.

Supongamos que necesitamos un procedimiento para obtener la potencia n de tres valores de tipo `Long` que se pasan como parámetros.

```
Public Sub ProbarByRef()
    Dim lng1 As Long, lng2 As Long, lng3 As Long
    Dim lngExponente As Long
    lng1 = 2
    lng2 = 3
```

```

lng3 = 4
lngExponente = 4
' Pasamos por referencia las tres variables
TresPotencias lng1, lng2, lng3, lngExponente
' Las variables lng1, lng2 y lng3 _
  han sido elevadas a la potencia 4
Debug.Print "Variable 1 " & lng1
Debug.Print "Variable 2 " & lng2
Debug.Print "Variable 3 " & lng3
End Sub

Public Sub TresPotencias( _
    ByRef Valor1 As Long, _
    ByRef Valor2 As Long, _
    ByRef Valor3 As Long, _
    ByVal Exponente As Long)
    Valor1 = Valor1 ^ Exponente
    Valor2 = Valor2 ^ Exponente
    Valor3 = Valor3 ^ Exponente
End Sub

```

Al ejecutar el procedimiento `ProbarByRef` nos mostrará

```

Variable 1 = 16
Variable 2 = 81
Variable 3 = 256

```

Que son los valores que han tomado `lng1`, `lng2` y `lng3` al elevarlos a la 4ª potencia.

Si en el procedimiento `TresPotencias` hubiéramos definido que los tres primeros parámetros iban a ser pasados Por Valor, `ProbarByRef` nos hubiera mostrado

```

Variable 1 = 2
Variable 2 = 3
Variable 3 = 4

```

Parámetros Opcionales

Tanto en los procedimientos **Function**, como en los **Sub** podemos definir algunos, e incluso todos los parámetros como Opcionales.

¿Qué quiere decir esto?.

- Tan simple como que podemos hacer que la introducción de su valor en un parámetro opcional no sea estrictamente necesaria.

Puntualizaciones sobre parámetros opcionales.

- Para definir que un parámetro es opcional ponemos delante de cualquier otro posible modificador, como **ByVal** ó **ByRef**, la palabra reservada **Optional**.
- Si en un procedimiento hay varios parámetros, y uno de ellos está definido como opcional, los parámetros que le sigan también deberán estar definidos como **Optional**.

- Si en un procedimiento, tenemos varios parámetros opcionales, que no vamos a introducir y varios de ellos son los últimos, y consecutivos, no será necesario introducir ninguna coma que actúe como separador.
- Si alguno de los parámetros que no vamos a introducir está entre dos que sí introduciremos, el espacio ocupado por el/los parámetros no introducidos los pondremos como un espacio en blanco. Por ejemplo:

```

MiProcedimiento Parámetro1, , , Parámetro4

```
- Si el que no vamos a introducir es el primero, pondremos una coma.

```

MiProcedimiento , Parámetro2 , Parámetro3

```
- Si no definimos un valor por defecto, para un parámetro opcional, y en la llamada al procedimiento tampoco se le asigna valor alguno, tomará el valor por defecto del tipo de dato. En los tipos numéricos ese valor será el Cero, en las cadenas de texto la cadena vacía y en los booleanos el valor False.
- Si necesitáramos saber si se ha pasado un valor a un parámetro opcional de tipo **Variant**, utilizaremos la función **IsMissing**.

```

Public Sub VariantOpcional( _
    Optional ByVal Valor As Variant)
    If IsMissing(Valor) Then
        Debug.Print "No se ha asignado Valor"
    Else
        Debug.Print "Se ha asignado un Valor"
    End If
End Sub

```

Si llamamos a este procedimiento sin asignar ningún contenido al parámetro Valor, por ejemplo llamando directamente

```

VariantOpcional

```

nos imprimirá

```

No se ha asignado Valor

```

En cambio si la llamamos

```

VariantOpcional "Parámetro String"

```

Nos mostrará

```

Se ha asignado un Valor

```

Supongamos ahora que tenemos que desarrollar una función que nos devuelva la edad en años de una persona; función que luego queremos utilizar, por ejemplo en consultas.

Para ello creamos un nuevo módulo y en él escribimos lo siguiente.

```

Public Const Pi As Double = 3.14159265358979

Public Function Edad( _
    ByVal Nacimiento As Date, _
    Optional ByVal FechaEdad As Date = Pi)
    ' Esta no es una función absolutamente exacta, _
    pero sí razonablemente operativa.

```

```

' 365.25 es por los años bisiestos
Const DiasAño As Double = 365.25

' Si no se introduce FechaEdad (FechaEdad = Pi) _
  le asignamos Date (la fecha de Hoy).
If FechaEdad = Pi Then
    FechaEdad = Date
End If
' Fix quita la parte decimal y devuelve _
  la parte entera de un número de coma flotante
Edad = Fix((FechaEdad - Nacimiento) / DiasAño)
End Function

```

Supongamos que hoy es el 14 de febrero de 2005

Llamar a la función `Edad(#9/3/53#, #2/14/05#)` sería lo mismo que llamarla con `Edad(#9/3/53#)`; en ambos casos nos devolvería **51**,

Primero la variable `FechaEdad` tomaría el valor **Pi**, por haber sido llamada la función `edad` por no haber pasado ningún valor al parámetro.

A continuación, tras comprobar que su valor es **Pi**, le asigna la **fecha de hoy** mediante la función de VBA `Date()`.

Puede resultar sorprendente la asignación del valor definido para **Pi** como valor por defecto del parámetro `FechaEdad`.

Ésta es una decisión puramente personal, basada en la práctica imposibilidad de que esa sea la fecha/hora en la que se quiera averiguar la edad de una persona.

Además **Pi** es un número por el que tengo “cierta debilidad”.

Por cierto, para el que sienta curiosidad, el valor usado como **Pi** está entre el segundo 53 y el segundo 54 de las 3 horas, 23 minutos del día 2 de enero del año 1900.

Resumiendo lo anterior, podemos definir en los procedimientos parámetros que serán opcionales, e incluso asignarles valores que tomarán si al llamar al procedimiento no se les asignara ningún valor.

Procedimientos Recursivos frente a Iterativos

Procedimientos Iterativos son aquéllos en los que se producen una serie de repeticiones.

Procedimientos recursivos son aquéllos que se van llamando a sí mismos, hasta que encuentran una condición base que permite deshacer todas las llamadas hacia atrás.

Ya sé que ésta es una definición un tanto Farragosa, pero lo entenderemos mejor con un ejemplo.

Vamos a definir la función `Factorial` que nos devolverá el factorial de un número.

Os recuerdo que factorial de un número entero positivo n es:

$$n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$

Se define el 1 como valor del factorial de cero $0! = 1$.

Una forma tradicional (iterativa) de resolver este problema sería el que mostramos en la entrega 09:

```
Public Function Factorial(ByVal n As Integer) As Long
    Dim i As Integer
    If n < 0 Then Exit Function
    Factorial = 1
    For i = 1 To n
        Factorial = Factorial * i
    Next i
End Function
```

En esta función no hay puesto un control de errores; de hecho para un valor de n superior a 12 el resultado supera el rango de los Long y dará error de desbordamiento.

Obvio estos detalles de cara a la simplificación del código, aunque en un desarrollo profesional esto sería imperdonable...

Este código es un claro caso de resolución mediante un procedimiento iterativo.

Por si sirve para aclarar conceptos, en el diccionario se define la iteración como el acto de repetir.

Vemos que en este procedimiento se repite una operación hasta obtener el valor que deseamos, utilizando para ello un bucle que va **iterando** ó **repitiendo** la operación producto, hasta que la variable i obtiene el valor de n .

La definición del factorial de un número n se puede hacer de otra manera:

$$n! = n * (n-1)! \text{ y el valor de } 0! \text{ es } 1$$

Nos dice que el factorial de un número n es igual a ese número multiplicado por el factorial de $n-1$.

Podemos realizar una función que actúe tal cual está definido el factorial de n :

```
Public Function Factorial( _
    ByVal n As Integer _
) As Long
    Dim i As Integer
    If n < 0 Then Exit Function
    If n = 1 Then
        Factorial = 1
        Exit Function
    End If
    Factorial = n * Factorial(n - 1)
End Function
```

Vemos que la función factorial se va llamando a sí misma:

$$\text{Factorial} = n * \text{Factorial}(n - 1)$$

Cada resultado parcial de `Factorial` lo va guardando en una posición de memoria.

Esto ocurre hasta que en alguna de las llamadas se encuentra con que el parámetro $n-1$ pasado a la función `Factorial` toma el valor 1.

En general un método recursivo se va llamando a sí mismo hasta que encuentra una solución elemental ó básica que le permite construir el proceso inverso y salir del mismo.

`Factorial(n - 1)` devolverá el valor 1 tras encontrarse con la línea en la que compara el valor de `n`. A partir de ese momento va realizando de forma inversa las operaciones con los resultados parciales anteriores de `factorial`.

Es decir calcula $1 * 2 * 3 * \dots * (n-1) * n$.

En este caso el método recursivo, para calcular el factorial de un número, necesita efectuar `n` llamadas a la función, almacenando los correspondientes resultados parciales en memoria.

Podemos deducir en este caso, que el método recursivo requiere más recursos de memoria y procesador que el método iterativo.

Hay una definición que afirma que cualquier método recursivo se puede transformar en iterativo, y viceversa.

Generalmente los métodos recursivos requieren más memoria y procesador que sus equivalentes iterativos. Entonces ¿para qué usarlos?

Hay problemas en los que una resolución iterativa es muchísimo más compleja que su correspondiente recursiva.

Apunto aquí un problema clásico como es el de “Las torres de Hanoi”, un juego de origen oriental que consiste en tres barras verticales y un conjunto de discos de diferentes diámetros con un orificio central, apilados en una de ellas y que hay que ir pasando a otra de las barras.

No voy a explicar la mecánica del juego ni la resolución del mismo. Si alguno tiene interés en conocerlo, Google, MSN, Altavista o cualquiera de los buscadores os indicarán una inmensa cantidad de páginas con información sobre el juego, el problema matemático que representa y sus algoritmos para resolverlo.

La solución Recursiva es casi “elemental” en cambio la iterativa es mucho más compleja.

Esto mismo es aplicable a ciertos algoritmos, por ejemplo de ordenación o de trazado gráfico de figuras englobadas en la “geometría fractal”.

Pasar un parámetro, tipo matriz, mediante ParamArray

En una entrega anterior, vimos que se podía asignar a un parámetro de tipo **Variant**, una matriz, y que incluso una función podía devolver una matriz mediante un **Variant**.

Supongamos que queremos definir una función que nos calcule la media de una serie de valores. Podemos utilizar una alternativa frente a las variables **Variant**.

Por definición, y de entrada, no sabemos cuántos valores vamos a manejar.

VBA nos permite resolverlo mediante la siguiente función utilizando **ParamArray**:

```
Public Function Media( _  
    ParamArray Sumandos() As Variant _  
    ) As Double  
    Dim dblSuma As Double  
    Dim lngElementos As Long  
    Dim n As Variant  
    For Each n In Sumandos  
        dblSuma = dblSuma + n  
    Next n  
    lngElementos = UBound(Sumandos) + 1
```

```

    Media = dblSuma / lngElementos
End Function

```

Si desde la ventana inmediato escribimos

```
? media(1,2.3,5,6,9,8)
```

Nos mostrará 5,21666666666667, que efectivamente es la media de esos 6 valores.

Puntualizaciones:

- **Ubound** nos devuelve el valor más alto del índice de una matriz; como en este caso el índice más bajo de Sumandos es 0, el número de elementos de la matriz es el devuelto por **Ubound** más 1.
- Si en un procedimiento vamos a utilizar **ParamArray**, la matriz correspondiente será el último parámetro del procedimiento.
- **ParamArray** no puede usarse con **ByVal**, **ByRef** u **Optional**.

Uso de parámetros Con Nombre

Supongamos que hemos definido un procedimiento **Function** con la siguiente cabecera:

```

Public Sub ConNombre( _
    Optional ByVal Nombre As String, _
    Optional ByVal Apellido1 As String, _
    Optional ByVal Apellido2 As String, _
    Optional ByVal Sexo As String)

```

En un momento dado debemos pasar el dato "Manolo" y "Masculino".

La forma "habitual" de hacerlo sería

```
ConNombre "Manolo", , , "Masculino"
```

Esto genera un código "confuso".

Para facilitar la introducción de los parámetros ó argumentos, tenemos la posibilidad de efectuar la introducción **Con ó Por Nombre**.

En la introducción por nombre se indica el Nombre del parámetro, seguido del operador de asignación (dos puntos seguido del signo igual) En el caso anterior se haría así:

```
ConNombre Nombre := "Manolo", Sexo:= "Masculino"
```

Como curiosidad indicaré, que este operador es el mismo que el de asignación del lenguaje Pascal.

- El uso de parámetros con nombre se puede usar tanto en procedimientos **Function** como **Sub**
- Habrá que poner todos los argumentos no opcionales
- El orden de colocación de los parámetros es libre
- Algunos de los objetos de VBA no admiten parámetros con nombre
- El uso de parámetros con nombre está especialmente indicado en procedimientos que tengan múltiples parámetros opcionales.

Constantes Enumeradas

En algunos tipos de procedimientos o funciones, vemos que alguno de los parámetros sólo debe tomar un número limitado de valores. Sería muy interesante que a la hora de ir escribiendo un procedimiento, VBA nos fuera sugiriendo los valores válidos.

Para ayudar a este fin tenemos las **Constantes Enumeradas**.

Las Constantes Enumeradas son un conjunto de constantes agrupadas en un tipo enumerador. Los valores que pueden admitir son sólo del tipo **Long**

La forma de construir las es

```
[Public | Private] Enum nombre
    nombre_miembro [= expresión_constante]
    nombre_miembro [= expresión_constante]
    . . .
End Enum
```

Se declaran a nivel del **encabezado de un módulo**, pudiendo ser declaradas como **Public** o **Private**. En los módulos de clase sólo pueden declararse como **Public**

La declaración comienza opcionalmente indicando primero su alcance a continuación la instrucción Enum seguida del nombre del tipo.

En las sucesivas líneas se van añadiendo los nombres de los sucesivos valores, opcionalmente con su valor correspondiente.

```
Public Enum Booleano
    blFalso
    blCierto
End Enum
```

En este caso blFalso tendría el valor 0 y blCierto tomaría el valor 1.

Podríamos haber hecho

```
Public Enum Booleano
    blFalso
    blCierto = -1
End Enum
```

Con lo que blFalso tendría el valor 0 y blCierto tomaría el valor -1.

Las constantes enumeradas, si no se les asigna un valor, toma el de la constante anterior, incrementada en 1.

Ya hemos dicho que si a la primera no se le asigna un valor toma el valor 0.

```
Public Enum DiasSemana
    dsLunes = 1
    dsMartes
    dsMiercoles
    dsJueves
    dsViernes
    dsSabado
    dsDomingo
```

```
End Enum
```

En este caso, por ejemplo `dsJueves` toma el valor 4.

He dicho que a las constantes enumeradas sólo se les puede asignar valores Long.

Esto no es del todo cierto. Esto por ejemplo funcionaría

```
Public Enum TipoSexo
    tsFemenino = "1"
    tsMasculino
End Enum
```

En esta declaración `tsFemenino` tendría el valor numérico 1 y `tsMasculino` el valor 2.

- ¿Y cómo es posible esto?

- Forma parte de las “pequeñas incoherencias” que tiene Visual Basic heredadas de las “viejas versiones”; incoherencias que en teoría son para ayudar al usuario, pero que pueden acabar despistándolo.

Uso de las Constantes Enumeradas

Supongamos que tenemos que crear una función, para una empresa de “Relaciones Interpersonales” que nos escriba el tratamiento que debemos escribir en la cabecera de las cartas dirigidas a sus “socios”.

La empresa define que a todos los hombres se les debe tratar como

Estimado Sr. Apellido

A las mujeres,

Si son solteras ó divorciadas

Estimada Sta. Apellido

En el resto de los casos

Estimada Sra. Apellido

Resultaría interesante que al escribir, por ejemplo este procedimiento, nos mostrara las posibilidades que podemos utilizar para cada parámetro.

Vamos a definir las constantes enumeradas correspondientes a cada caso.

Podrían ser algo así como:

```
Public Enum TipoSexo
    tsFemenino = 1
    tsMasculino = 2
End Enum
```

```
Public Enum EstadoCivil
    ecSoltero
    ecCasado
    ecSeparado
    ecDivorciado
    ecViudo
```

End Enum

Como hemos comentado asignar el valor 2 a `tsMasculino` no sería necesario.

Una vez definidas estas constantes, están inmediatamente disponibles en la Ayuda Contextual, como podemos ver en la siguiente imagen, pudiéndose utilizar como un tipo de datos más.

```
Public Enum TipoSexo
    tsFemenino = 1
    tsMasculino = 2
End Enum
```

```
Public Enum EstadoCivil
    ecSoltero
    ecCasado
    ecSeparado
    ecDivorciado
    ecViudo
End Enum
```

```
Public Function Tratamiento( _
    ByVal Apellido As String, _
    Sexo As TipoSexo, _
    Optional ByVal Estado As Es|
End Function
```



Además, conforme vayamos desarrollando el código nos mostrará los posibles valores concretos que podrán tomar los parámetros

```
Public Function Tratamiento( _
    ByVal Apellido As String, _
    Sexo As TipoSexo, _
    Optional ByVal Estado As EstadoCivil _
) As String

    If Sexo = tsMasculino Then
    ElseIf Sexo = |
        tsFemenino
        tsMasculino
    End If
End Function
```

Completemos la función:

```
Public Function Tratamiento( _
    ByVal Apellido As String, _
    Sexo As TipoSexo, _
    Optional ByVal Estado As EstadoCivil _
) As String
```

```

If Sexo = tsMasculino Then
    Tratamiento = "Estimado Sr. " & Apellido
ElseIf Sexo = tsFemenino Then
    Select Case Estado
        Case ecSoltero, ecDivorciado
            Tratamiento = "Estimada Sta. " & Apellido
        Case Else
            Tratamiento = "Estimada Sra. " & Apellido
    End Select
Else
    Tratamiento = "Estimado/a Sr./Sra. " & Apellido
End If
End Function

```

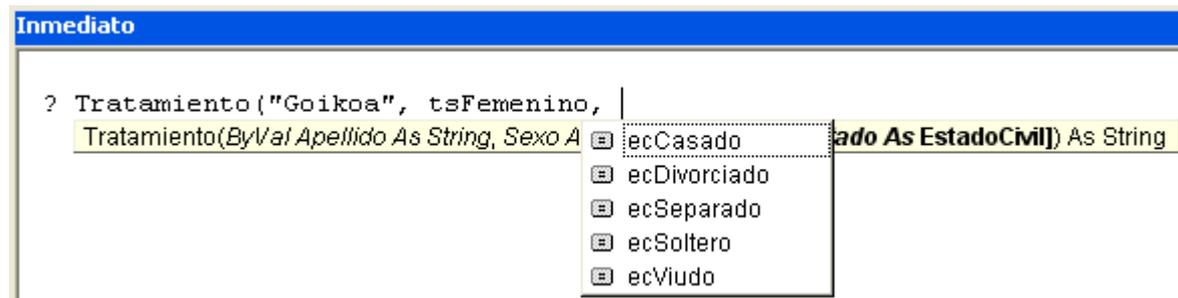
Con esta función si desde la ventana inmediato escribimos

```
? Tratamiento("Goikoa", tsFemenino,ecCasado)
```

Nos escribirá

```
Estimada Sra. Goikoa
```

Conforme escribimos la función nos irá apareciendo la ayuda



Si hacemos las pruebas con las diferentes posibilidades veremos que responde a lo solicitado por la empresa contratante.

Visto el éxito, nos piden que también creamos una función para la línea de la dirección de forma que nos ponga:

```
A la Att. de Don Nombre Apellido1
```

```
A la Att. de Doña Nombre Apellido1
```

Si no existiera el Nombre que pusiera

```
A la Att. del Sr. Apellido1
```

```
A la Att. de la Sta. Apellido1
```

```
A la Att. de la Sra. Apellido1
```

El desarrollo de esta función la dejo para el lector.

Para intercambiar información entre el usuario y VBA tenemos 2 procedimientos básicos que se corresponden a los cuadros de diálogo predefinidos en VBA.

- **MsgBox**
- **InputBox.**

Función MsgBox

La función **MsgBox**, muestra un mensaje en un Cuadro de diálogo con determinados botones, y puede devolver un número que identifica el botón pulsado por el usuario.

En este cuadro de diálogo se puede definir el texto de la barra de título, el mensaje mostrado, un icono que indica el tipo de mensaje y seleccionar botones de los predefinidos previamente por VBA.

La sintaxis es la siguiente

```
MsgBox(prompt[, buttons][, title][, helpfile, context])
```

Es decir

```
MsgBox(mensaje[, botones][, título][, ficheroayuda, contextoyuda])
```

La forma más básica posible sería

```
MsgBox mensaje
```

Por ejemplo

```
MsgBox "Hola a todos"
```

Esta instrucción mostraría el siguiente mensaje



He hecho una llamada a **MsgBox** como si fuera un procedimiento de tipo **Sub**, no como una función.

En este caso tendría sentido ya que al haber sólo un botón, MsgBox sólo va a devolver un valor.

MsgBox admite los parámetros, ó argumentos **Con Nombre**. Así el código anterior podría haberse escrito

```
MsgBox Prompt:= "Hola a todos"
```

El valor devuelto dependerá del botón que se haya pulsado.

El parámetro **Buttons** indica por una parte el tipo de icono que debe mostrar el Cuadro de diálogo ó cuadro de Mensaje y los botones que va a contener.

Supongamos que en un formulario hemos puesto un botón que al presionarlo hará que se formatee el disco C:

No hace falta ser muy listo para pensar que existe la posibilidad de una pulsación accidental.

Para prevenir este caso. antes de lanzar el proceso de formateo, podríamos hacer lo siguiente.

```
Public Sub Formateo()  
Dim strMensaje As String
```

```

Dim lngRespuesta As Long
strMensaje = _
"¿Está seguro de querer formatear el disco C:?" _
    & vbCrLf _
    & "¡Se perderán todos los datos!"
lngRespuesta = MsgBox(strMensaje, _
    vbExclamation + vbOKCancel + vbDefaultButton2, _
    "Ha solicitado formatear C:")
Select Case lngRespuesta
    Case vbOK
        Formatear "C"
    Case vbCancel
        MsgBox Prompt:= _
            "Ha interrumpido el Formateo del disco", _
            Buttons:=vbInformation, _
            Title:="Ha salvado sus datos"
    Case Else
        MsgBox lngRespuesta
End Select
End Sub

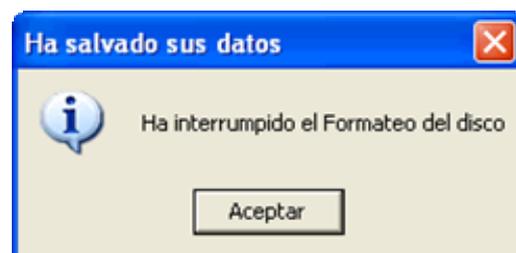
Public Sub Formatear(ByVal Disco As String)
    MsgBox Prompt:="Formateando el disco " _
        & Disco & ": ...", _
        Title:="Sus datos están siendo borrados"
End Sub

```

Este código mostrará el siguiente mensaje:



Dependiendo del botón pulsado mostrará



El tipo de icono que se va a mostrar lo dan las siguientes constantes de VBA.

Constante	Valor	Icono
VbInformation	64	
VbExclamation	48	
VbQuestion	32	
VbCritical	16	

La segunda parte del parámetro **Buttons** hace referencia a los botones que va a mostrar el cuadro de mensaje.

Sus posibilidades son

Constante	Valor	Botones
VbOKOnly	0	[Aceptar]
VbOKCancel	1	[Aceptar] [Cancelar]
VbAbortRetryIgnore	2	[Anular] [Reintentar] [Ignorar]
VbYesNoCancel	3	[Sí] [No] [Cancelar]
VbYesNo	4	[Sí] [No]
VbRetryCancel	5	[Reintentar] [Cancelar]
VbMsgBoxHelpButton	16384	Añade un botón [Ayuda] a los anteriores

El tercer grupo de constantes es el que define el botón seleccionado como predeterminado.

Será el que se active si directamente presionamos la tecla [**Intro**]

Constante	Valor	Botón predeterminado
VbDefaultButton1	0	Primer botón
VbDefaultButton2	256	Segundo botón
VbDefaultButton3	512	Tercer botón
VbDefaultButton4	768	Cuarto botón

El cuarto grupo de constantes define el tipo de presentación Modal del cuadro.

La presentación modal hace que tome prioridad el cuadro frente al resto de la aplicación, visualizándose en primer plano.

Constante	Valor	Tipo de visualización modal
VbApplicationModal	0	El cuadro es modal frente a la aplicación
VbSystemModal	4096	El cuadro es modal frente al resto de las aplicaciones

Hay una constante adicional **VbMsgBoxRight** que sirve para hacer que el texto se alinee a la derecha del mensaje. Su valor es 524288.

Las dos últimas constantes que permite usar MsgBox son **VbMsgBoxSetForeground** y **VbMsgBoxRtlReading**.

La primera define la ventana del cuadro como de primer plano, y la segunda afecta a la visualización de los textos en hebreo y árabe que se escriben de Derecha a Izquierda.

Usando las constantes:

Supongamos que queremos mostrar un mensaje con los botones

[Sí] [No] [Cancelar]

Como icono deberá tener el de **Exclamación**



Además queremos que su título sea **Proceso de Votación**

El mensaje que aparecerá será **Seleccione su voto**

El botón preseleccionado deberá ser el de [**Cancelar**].

Tras seleccionar una opción nos mostrará un mensaje informándonos del voto realizado.

Vamos a escribir el código que responde a estos requisitos:

```
Public Sub Votacion()
    Dim lngBotonesIcono As Long
    Dim strTitulo As String
    Dim strMensaje As String
    Dim strVoto As String
    Dim lngRespuesta As Long

    lngBotonesIcono = vbExclamation _
        + vbYesNoCancel _
        + vbDefaultButton3
    strTitulo = "Proceso de Votación"
    strMensaje = "Seleccione su voto"

    lngRespuesta = MsgBox( _
        strMensaje, _
        lngBotonesIcono, _
        strTitulo)

    ' Tras pasar el cuadro el botón seleccionado _
    ' a la variable lngRespuesta _
    ' mostramos el voto
    ' Para ello usamos la función TeclaPulsada
    strVoto = TeclaPulsada(lngRespuesta)
    If lngRespuesta = vbCancel Then
        strMensaje = "Has cancelado la votación"
    Else
        strMensaje = "Has votado: " _
            & TeclaPulsada(lngRespuesta)
    End If
End Sub
```

```
MsgBox strMensaje, _  
        vbInformation, _  
        "Resultado de la votación"  
End Sub
```

La función **NombreTecla**, llamada desde el procedimiento **Votacion**, sería la siguiente:

```
Public Function NombreTecla( _  
        ByVal Tecla As Long _  
        ) As String  
    Select Case Tecla  
        Case vbOK  
            NombreTecla = "Aceptar"  
        Case vbCancel  
            NombreTecla = "Cancelar"  
        Case vbAbort  
            NombreTecla = "Anular"  
        Case vbRetry  
            NombreTecla = "Reintentar"  
        Case vbIgnore  
            NombreTecla = "Ignorar"  
        Case vbYes  
            NombreTecla = "Sí"  
        Case vbNo  
            NombreTecla = "No"  
        Case Else  
            NombreTecla = "Desconocida"  
    End Select  
End Function
```

Las constantes aquí manejadas no son un caso particular del cuadro de mensaje.

Este tipo de constantes son muy habituales en VBA, y funcionan igual que las constantes enumeradas que hemos visto en esta misma entrega.

En concreto las últimas que hemos utilizado están definidas internamente como de los tipos enumerados **VbMsgBoxStyle** y **VbMsgBoxResult**.

Función InputBox

La función **InputBox**, al igual que **MsgBox**, puede mostrar un mensaje, pero se diferencia en que en vez de enviar un número Long en función de la tecla pulsada, nos pide que escribamos un texto, en un cuadro de texto. Su contenido será el dato que devuelva.

La sintaxis de **InputBox** es:

```
InputBox(prompt[, title][, default][, xpos][, ypos][,  
helpfile, context])
```

Parámetro	Descripción
<i>prompt</i>	Mensaje del cuadro de diálogo (obligatorio)
<i>title</i>	Texto a visualizar en la barra de título
<i>default</i>	Valor que devolverá, si no introducimos ningún texto
<i>xpos</i>	Coordenada X del cuadro (distancia desde el lado izquierdo)
<i>ypos</i>	Coordenada Y del cuadro (distancia desde arriba)
<i>helpfile</i>	Fichero de ayuda auxiliar
<i>context</i>	Contexto del fichero de ayuda

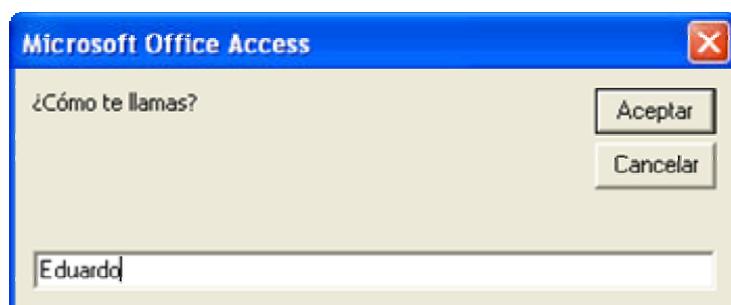
La forma más básica sería `inputbox ("Mensaje")`

Vamos a escribir el siguiente código

```
Public Sub Pregunta()
    Dim strRespuesta As String

    strRespuesta = InputBox("¿Cómo te llamas?")
    MsgBox "Tu nombre es " & strRespuesta, _
        vbInformation, " Respuesta suministrada"
End Sub
```

Si ejecutamos el procediendo nos muestra



Tras presionar el botón [Aceptar] el `MsgBox` nos mostrará el texto introducido en el `InputBox`.

Si hubiéramos introducido en el `InputBox` el texto por defecto, ya fuera mediante

```
strRespuesta = InputBox( _
    "¿Cómo te llamas?", , _
    "Introduce aquí tu nombre")
```

O mediante

```
strRespuesta = InputBox( _
    Prompt:="¿Cómo te llamas?", _
    Default:="Introduce aquí tu nombre")
```

El `InputBox`, nos mostraría



Si no borramos ese texto, por ejemplo al introducir algún texto, éste será el texto que devolverá **InputBox**.

Al ejecutar ese código podemos comprobar que el cuadro de entrada nos aparece en la práctica centrado en la pantalla.

InputBox tiene dos parámetros que le permiten ubicar el cuadro en una posición concreta. Estos parámetros son Xpos e Ypos.

He modificado el código del procedimiento para que muestre un título y ubicarlo cerca de la posición superior izquierda

```
Public Sub Pregunta()
    Dim strRespuesta As String

    strRespuesta = InputBox( _
        Prompt:="¿Cómo te llamas?", _
        Default:"Introduce aquí tu nombre", _
        Title:"¿Escriba su nombre?", _
        XPos:=500, YPos:=500)
    MsgBox "Tu nombre es " & strRespuesta, _
        vbInformation, " Respuesta suministrada"
End Sub
```

Las medidas de posición se dan en **Twips**.

Un **Twip**, **Twentieth of a Point** equivale a la Vigésima parte de un punto.

El Twip equivale a **1/1440 de pulgada**, es decir, **567 twips** equivalen a un centímetro.

Esto es válido para la teoría.

En la vida real, las pruebas con diferentes valores son las que nos indicarán qué valores serán los adecuados a cada caso.

Nota:

Los parámetros *helpfile* y *context* hacen referencia a ficheros de ayuda desarrollados al efecto para la aplicación.

En este nivel de estudio (básico), no vamos a abordar el análisis del desarrollo de ficheros de ayuda.

Esto no es óbice para indicar que, tanto con **MsgBox** como con **InputBox**, se puede hacer referencia a determinados contenidos de ficheros de ayuda desarrollados ex profeso para la aplicación, y no sólo al fichero en sí, sino a una sección concreta del mismo.

Hay abundante documentación disponible en Internet al respecto.

Sólo quiero reseñar el magnífico trabajo desarrollado por “mi amigo Guille” en su página:

http://www.elquille.info/vb/VB_HLP.HTM

En la próxima entrega examinan algunos de los **operadores** disponibles con VBA.

Trataremos también funciones y procedimientos implementados en VBA.

Entre ellas incluiremos

- Funciones para el tratamientos de cadenas
 - Left
 - Mid
 - Right
 - InStr
 -
- Funciones de conversión de datos
- Funciones para el tratamientos de fechas
-