

Comencemos a programar con
VBA - Access

Entrega **09**

Estructuras de Control

Estructuras de Control.

Las estructuras de control son segmentos de código que nos permiten tomar decisiones en base a unos datos dados, o repetir procesos (bucles) mientras sucedan determinadas condiciones en los parámetros controlados por el código.

Ya hemos comentado algunas de ellas en las entregas anteriores.

Estas estructuras vienen determinadas por una serie de instrucciones, entre las que destacaremos:

Estructuras de Decisión

- `If Then`
- `If Then Else`
- `IIF`
- `Select Case`

Estructuras de Bucle

- `For Next`
- `For Each . In Next`
- `While Wend`
- `Do Loop`

Instrucción de Salto

- `Goto`

Nota:

Antes de seguir adelante, adoptaré el sistema habitual para las expresiones de la sintaxis de una sentencia.

Las partes de código situadas entre corchetes [] son opcionales.

De las partes contenidas entre Llaves { } hay que seleccionar una de ellas.

Estructuras de Decisión.

La Instrucción If

Permite ejecutar un grupo de instrucciones de código, en función de que el valor de una expresión sea Cierta o Falsa **True** / **False**.

La forma más básica de esta instrucción es:

If *condición* **Then** [*instrucciones*]

Condición debe ser una expresión, numérica, relacional ó lógica que devuelva **True** ó **False**.

Por ejemplo:

```
If Divisor<>0 then Cociente = Dividendo/Divisor
```

Si el valor de la variable `Divisor` es diferente a Cero, entonces haz que la variable `Cociente` tome el valor de la división entre la variable `Dividendo` y la variable `Divisor`.

Esta forma de la instrucción **If** sólo se puede poner en una única línea de código, aunque admite múltiples instrucciones separadas por los dos puntos ":".

```
If Divisor<>0 then C = Dividendo/Divisor : Debug.print C
```

Segunda forma

```
If condición Then  
    [instrucciones]  
End If
```

El ejemplo anterior podría haberse escrito:

```
If Divisor<>0 then  
    Cociente = Dividendo/Divisor  
End If
```

Esta forma permite la ejecución de varias líneas de sentencias entre **Then** y **End If**.

Esta sintaxis es preferible a la primera, ya que genera un código más claro de interpretar.

La instrucción **If** permite ejecutar otro grupo de sentencias, si el resultado de la evaluación de la expresión fuera falso.

```
If condición Then  
    [instrucciones para el caso de que condición sea  
    True]  
Else  
    [instrucciones para el caso de que condición sea  
    False]  
End If
```

Ejemplo:

```
Public Sub PruebaIf01()  
    Dim Dividendo As Single  
    Dim Divisor As Single  
    Dim Cociente As Single  
  
    Dividendo = 4  
    Divisor = 2  
  
    If Divisor <> 0 Then  
        Cociente = Dividendo / Divisor  
        Debug.Print Cociente  
    Else  
        MsgBox "No puedo dividir entre cero", _  
            vbOKOnly + vbCritical, _  
            "División por cero"  
    End If  
  
End Sub
```

En este caso, como `Divisor <> 0` devuelve `False`, se ejecutará la línea que aparece entre `Else` y `End If`, con lo que mostrará el mensaje de error.

Estas sentencias admiten aún un modo adicional, y es usar `Else If`. Es una nueva evaluación tras una anterior que da como resultado falso.

Supongamos que queremos hacer una función que devolviera el Nombre de provincia en función de un código. Acepto por adelantado que habría otras formas más adecuadas, pero es sólo un ejemplo.

Ejemplo:

```
Public Function Provincia(ByVal Codigo As Long) As String
    If Codigo < 1 Or Codigo > 52 Then
        Provincia = "Código de provincia incorrecto"
    ElseIf Codigo = 1 Then
        Provincia = "Álava"
    ElseIf Codigo = 8 Then
        Provincia = "Barcelona"
    ElseIf Codigo = 20 Then
        Provincia = "Guipuzcoa"
    ElseIf Codigo = 28 Then
        Provincia = "Madrid"
    ElseIf Codigo = 31 Then
        Provincia = "Navarra"
    ElseIf Codigo = 31 Then
        Provincia = "Navarra"
    ElseIf Codigo = 26 Then
        Provincia = "La Rioja"
    ElseIf Codigo = 48 Then
        Provincia = "Vizcaya"
    ElseIf Codigo = 50 Then
        Provincia = "Zaragoza"
    Else
        Provincia = "Otra Provincia"
    End If
End Function
```

Con este código `Provincia(31)` devolvería `"Navarra"`

Las instrucciones `If` se pueden **anidar**, (poner unas dentro de otras).

```
If comparación1 Then
    [Instrucciones de 1]
    If comparación2 Then
        [Instrucciones de 2]
    End If
```

```
End If
```

La Función **IIf**

Es una función similar a la estructura **If . . Then . . Else**

Devuelve uno de entre dos valores, en función del resultado de una expresión:

Su sintaxis es

```
IIf(expresión, ValorTrue , ValorFalse)
```

Se evalúa la expresión y si es **True**, devuelve **ValorTrue**; caso contrario devuelve **ValorFalse**.

Por ejemplo

```
Public Function EsPar(ByVal Numero As Long) As Boolean
    EsPar = IIf(Numero Mod 2 = 0, True, False)
End Function
```

La función **IIF**, en este caso, sería igual a hacer

```
Public Function EsPar2(ByVal Numero As Long) As Boolean
    If Numero Mod 2 = 0 Then
        EsPar2 = True
    Else
        EsPar2 = False
    End If
End Function
```

Nota:

El operador **Mod** devuelve el resto de dividir `Numero` entre 2.

La Instrucción **Select Case**

Con **If . . Then** es posible crear estructuras de decisión complejas como hemos visto en la función **Provincia** de un ejemplo anterior.

El que sea posible ni impide que resulte un tanto “Farragoso”.

Para este tipo de casos existe en VBA la instrucción **Select Case** que simplifica esas operaciones, creando un código más potente, ordenado y claro.

Si vemos la ayuda de Acces podemos leer que la sintaxis de **Select Case** es:

```
Select Case expresión_prueba
[Case lista_expresion-1
[instrucciones-1]] ...
[Case lista_expresion-2
[instrucciones-2]] ...
- - - - -
[Case lista_expresion-n
[instrucciones-n]] ...
```

```

[Case Else
[instrucciones_else]]
End Select

```

expresión_prueba debe ser una variable, o expresión que devuelva una cadena ó un número.

lista_expresion son una serie de valores, del tipo que da *expresión_prueba*.

Si *expresión_prueba* coincide con alguno de los valores de *lista_expresion*, se ejecutarán las instrucciones que existen a continuación, hasta llegar al siguiente **Case**, ó **End Select**.

A partir de este punto se saldría de la estructura y se seguiría con el siguiente código.

Si no se cumpliera la condición de ninguno de los **Case** *lista_expresion*, y hubiera un **Case Else**, se ejecutarían las líneas de código contenido a partir de **Case Else**.

Ejemplo de las expresiones que pueden estar contenidas en las *lista_expresion*:

```

Case 3
Case 3, 5, 6, 7
Case 1 To 8, 0, -5
Case Is < 8
Case Is > 3
Case Is >= lngDias
Case "A", "B", "C", "Z"

```

Voy a poner un ejemplo para clarificarlo:

Supongamos que queremos crear una función que nos cualifique el tipo de los pagarés de los clientes en función del tiempo que queda hasta su cobro.

La función recibirá como parámetro la fecha del vencimiento. Si la fecha es anterior al día de hoy, deberá devolver la cadena **"Pago vencido"**. Si es del día de hoy **"Vence hoy"**, si quedan entre 1 y 3 días **"Cobro inmediato"**, si menos de 31 días **"Corto Plazo"** si son menos de 181 días **"Medio Plazo"** y si es mayor **"Largo Plazo"**

La función podría ser ésta:

```

Public Function TipoVencimiento( _
    Vencimiento As Date _
) As String

Dim lngDias As Long
' Date devuelve la fecha de hoy
lngDias = Vencimiento - Date

Select Case lngDias
    Case Is < 0 ' Si lngDias es menor que cero
        TipoVencimiento = "Pago vencido"
    Case 0 ' Si es cero
        TipoVencimiento = "Vence hoy"
    Case 1, 2, 3 ' De 1 a 3
        TipoVencimiento = "Cobro inmediato"

```

```
Case 4 To 30 ' De 4 a 30
    TipoVencimiento = "Corto Plazo"
Case 31 To 180 ' De 31 a 180
    TipoVencimiento = "Medio Plazo"
Case Else ' Si ninguno de los anteriores
    TipoVencimiento = "Largo Plazo"
End Select
End Function
```

Aquí mostramos algunas de las posibilidades de elaboración de la *lista_expresion*.

```
Case Is < 0
```

Is se puede utilizar junto con operadores de comparación.

Estos operadores son

```
= Igual a
< Menor que
<= Menor ó igual que
> Mayor que
>= Mayor ó igual que
<> Diferente que
```

Se pueden poner diferentes expresiones separadas por comas

Esta línea sería válida:

```
Case Is < 0, 4, 8, is > 10
```

Se ejecutarían las líneas correspondientes al *Case* para cualquier valor que sea menor que 0, mayor que 10 ó si su valor es 4 u 8.

Este sistema también puede aplicarse a cadenas de texto.

Supongamos que queremos clasificar a los alumnos de un colegio en 4 grupos, en función de su apellido.

Aquéllos cuyo apellido empiece por una letra comprendida entre la A y la D pertenecerán al grupo 1, entre la E y la L al grupo 2, entre la M y la P al 3 y entre la Q y la Z al 4.

La función sería

```
Public Function Grupo( _
    ByVal Apellido As String _
) As Long

    Apellido = Trim(UCase(Apellido))
    Select Case Apellido
        Case Is < "E"
            Grupo = 1
        Case "E" To "LZZZZ"
            Grupo = 2
        Case "M" To "PZZZZ"
```

```

        Grupo = 3
    Case "Q" To "TZZZZ"
        Grupo = 3
    Case Is >= "U"
        Grupo = 4
    End Select
End Function

```

Nota:

Hemos utilizado, como auxiliares dos funciones de VBA. En concreto en la línea

```
Apellido = Trim(UCase(Apellido))
```

Primero se aplica la función **Ucase** sobre el parámetro **Apellido** y después la función **Trim**.

Ucase convierte las minúsculas que pueda haber en **Apellido** a Mayúsculas

Trim elimina los posibles espacios en blanco que pudiera haber a la izquierda y a la derecha de **Apellido**.

En concreto, si **Apellido** contuviera el valor " Olaz ", lo convertiría a "OLAZ".

La función **Grupo** (" Olaz "), devolvería el valor 3.

Al ser "OLAZ" mayor que "M" y menor que "PZZZZ" ejecutaría la línea

```
Grupo = 3
```

Nota:

Para que dos cadenas sean iguales, deben tener los mismos caracteres.

Una cadena A es menor que otra B si aplicando los criterios de ordenación, A estaría antes que B. En este caso podemos decir que B es mayor que A porque si estuvieran en una lista ordenada alfabéticamente, B estaría después que A.

El definir si "OLAZ" es menor que "Olaz" ó es igual, se especifica en la primera línea que aparece en el módulo.

```
Option Compare Database|Text|Binary
```

A continuación de Compare podemos poner **Database**, **Text** ó **Binary**

Si aparece **Text**, Olaz sería igual a OLAZ

Si aparece **Binary** Olaz sería mayor que OLAZ

Si aparece **Database** utilizaría el criterio de ordenación por defecto de la base de datos.

Option Compare Database es específico de Access.

Por ejemplo **Visual Basic** no lo contempla.

La expresión **CadenaInferior To CadenaSuperior** se utiliza de forma similar a

```
ValorNumericoInferior To ValorNumericoSuperior
```

En la función **TipoVencimiento** teníamos la siguiente línea

```
Case 1, 2, 3 ' De 1 a 3
```

Esto hacía que si la diferencia de días fuese de 1, 2 ó 3 se ejecutara el código de ese Case.

Esta forma de generar una lista de comparaciones también se puede realizar con caracteres de texto. Sería válido, por ejemplo `Case "A", "B", "C"`

Estructuras de Bucle.

Las Instrucciones For - - - Next

Supongamos que tenemos que construir una función que nos devuelva el Factorial de un número.

Os recuerdo que Factorial de n es igual a $1*2*3* \dots *(n-1)*n$, para n entero y mayor que cero.

Adicionalmente se define que Factorial de Cero tiene el valor 1.

Cómo se haría esta función:

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

Efectivamente funciona, ya que **Factorial** devuelve resultados correctos para valores de n entre 0 y 12.

Pero esta función no sería operativa para un uso profesional ya que tiene una serie de fallos. Por ejemplo, si hacemos **Factorial(-4)** devuelve el valor 1, lo que no es correcto, ya que no existe el factorial de un número negativo. Igualmente podemos pasarle valores superiores a 12, que nos darían un error de desbordamiento, ya que 13! supera el alcance de los números **Long**. Probad haciendo en la ventana inmediato

```
? Factorial(13).
```

Observad este código:

```
Public Function Factorial(ByVal n As Integer) As Long
    Dim i As Integer
    Select Case n
    Case Is < 0
        MsgBox "No existe el factorial de un número negativo", _
            vbCritical + vbOKOnly, _
            " Error en la función Factorial"
        Exit Function
    Case 0
        Factorial = 1
        Exit Function
    Case 1 To 12
        Factorial = 1
        For i = 1 To n
            Factorial = Factorial * i
```

```
        Next i
    Case Else
        MsgBox "Número demasiado grande", _
            vbCritical + vbOKOnly, _
            " Error en la función Factorial"
        Exit Function
    End Select
End Function
```

He puesto una serie de sentencias `Case` para filtrar los valores que darían resultados incorrectos, o producirían error, avisándole al usuario de que ha tratado de utilizar la función `Factorial` con unos valores fuera de su rango válido.

Así el mayor valor lo obtenemos de

12! = 479.001.600

El tener como rango válido del 0 a 12 ¿no resulta un poco corto ?.

Dependiendo para qué, sí.

Supongamos que estamos programando un sistema estadístico que hace uso de cálculos combinatorios grandes. Probablemente esta función no serviría, aunque se podrían usar trucos para saltarse sus limitaciones.

El problema surge porque el resultado supera el rango de los números `Long`, pero en el Capítulo 5º, y también en el Apéndice 01, vemos que existen dos tipos de números que superan esa limitación. Uno es el de los `Currency` y el otro el de los `Decimal`.

Vamos a cambiar el código para trabajar con `Currency`:

```
Public Function FactorialCurrency(ByVal n As Integer) As Currency
    Dim i As Integer
    Select Case n
    Case Is < 0
        MsgBox "No existe el Factorial de un número negativo", _
            vbCritical + vbOKOnly, _
            " Error en la función FactorialCurrency"
        Exit Function
    Case 0
        FactorialCurrency = 1
        Exit Function
    Case 1 To 17
        FactorialCurrency = 1
        For i = 1 To n
            FactorialCurrency = FactorialCurrency * i
        Next i
    Case Else
        MsgBox "Número demasiado grande", _
            vbCritical + vbOKOnly, _
```

```
        " Error en la función FactorialCurrency"  
        Exit Function  
    End Select  
End Function
```

Ahora nos permite trabajar desde 0 a 17.

17! = 355.687.428.096.000

Esta ya es una cifra importante. Pero supongamos que nos contrata el Fondo Monetario Internacional, para hacer unos estudios estadísticos a nivel mundial.

Es posible la capacidad de calcular hasta el factorial de 17, se nos quede corta.

Para ello echamos mano de un tipo numérico de rango aún más alto. El tipo **Decimal**.

El tipo **Decimal** tiene la particularidad de que VBA no puede trabajar directamente con él.

La variable que lo vaya a contener debe ser primero declarada como **Variant**, y luego convertida a **Decimal**.

```
Public Function FactorialDecimal( _  
    ByVal n As Integer _  
    ) As Variant  
    Dim i As Integer  
    Dim Resultado As Variant  
    ' Aquí hacemos la conversión de Variant a Decimal  
    Resultado = CDec(Resultado)  
  
    Select Case n  
    Case Is < 0  
        MsgBox "No existe el factorial de un número negativo", _  
            vbCritical + vbOKOnly, _  
            " Error en la función Resultado"  
        Exit Function  
    Case 0  
        FactorialDecimal = 1  
        Exit Function  
    Case 1 To 27  
        Resultado = 1  
        For i = 1 To n  
            Resultado = Resultado * CDec(i)  
        Next i  
    Case Else  
        MsgBox "Número demasiado grande", _  
            vbCritical + vbOKOnly, _  
            " Error en la función FactorialDecimal"  
        Exit Function  
    End Select  
End Function
```

```

    FactorialDecimal = Resultado
End Function

```

La función `FactorialDecimal`, trabaja en el rango de **0!** a **27!**.

27! = 10888869450418352160768000000

Es decir con 29 cifras exactas.

Una puntualización respecto al tipo `Decimal`. El tipo decimal no es el que permite un rango mayor de valores; es el que permite un rango de valores con más cifras exactas.

Los bucles del tipo `For` - - `Next` se pueden anidar (Poner unos dentro de otros).

Por ejemplo, supongamos que tenemos que generar un procedimiento que nos imprima las tablas de multiplicar que van del 1 al 10.

```

Public Sub TablasDeMultiplicar()
    Dim n As Integer, m As Integer

    For n = 1 To 10
        Debug.Print "-----"
        For m = 1 To 10
            Debug.Print n & " x " & m & " = " & n * m
        Next m
    Next n
End Sub

```

Para cada valor que tomara `n`, ejecutaría el bucle completo de `For m --- Next m`, imprimiendo en la ventana Inmediato los resultados de las tablas

```

- - -
- - -
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90
-----
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100

```

En los ejemplos anteriores hemos utilizado la estructura más típica de VBA para la creación de Bucles.

La instrucción `For m --- Next m`, repite el código contenido entre la línea que contiene la palabra `For` y la línea que contiene a su correspondiente `Next`.

Su sintaxis es

```
For contador = principio To fin [Step incremento]
[instrucciones]
[Exit For]
[instrucciones]
Next [contador]
```

Contador es una variable numérica que irá tomando sucesivos valores, con incrementos ó decrementos iguales al valor de **incremento**.

Si no se pusiera el valor **incremento**, **contador** iría creciendo en una unidad cada vuelta.

El código se irá repitiendo hasta que contador tome el valor de **fin**, ó se encuentre con la instrucción **Exit For**.

En el siguiente ejemplo, el bucle **For Next** se ejecutará hasta que **lngSuma** sea mayor que **100**, momento en que saldrá del bucle o se imprima el número de impares especificado en el parámetro **Numero**. Si el parámetro **Numero** fuese cero ó menor, se sale directamente del procedimiento sin ejecutarse el bucle.

```
Public Sub ImprimeImpares (Numero As Long)
    Dim i As Long
    Dim lngImpar As Long
    Dim lngSuma As Long
    If Numero < 1 Then
        Exit Sub
    End If
    For i = 1 To Numero
        lngImpar = 2 * i - 1
        lngSuma = lngSuma + lngImpar
        If lngSuma > 100 Then
            Exit For
        End If
        Debug.Print i & " - " & lngImpar & " - " & lngSuma
    Next i
End Sub
```

La llamada al procedimiento se haría, por ejemplo para 4 impares

```
ImprimeImpares 4
```

Después de la palabra `Next`, no es imprescindible escribir el nombre de la variable que sirve como contador. Por ejemplo este bucle es válido a pesar de no escribir `Next i`:

```
For i = 1 To 10
    Debug.Print i
Next
```

La Instrucción For Each - - - Next

Esta estructura de bucle ya la hemos visto en los capítulos anteriores, por ejemplo cuando vimos las colecciones.

Es similar a la sentencia `For`, sólo que esta sentencia repite un grupo de instrucciones para cada elemento de una colección ó una matriz, siempre que ésta última no contenga una estructura tipo Registro, definida por el usuario.

La sintaxis es:

```
For Each elemento In grupo
[instrucciones]
Exit For
[instrucciones]
Next [elemento]
```

Como en el caso de `For - - - Next`, es posible salir del bucle utilizando la instrucción `Exit For`.

En las entregas anteriores, hemos puesto ejemplos de uso con **Colecciones**. El siguiente ejemplo extrae elementos de una **Matriz**.

```
Public Sub PruebaForEachConMatrices()
    Dim Datos() As String
    Dim Dato As Variant
    ' Llamamos al procedimiento _
    ' Que rellena la matriz con datos
    RellenaMatriz Datos
    ' Leemos los elementos de la matriz
    For Each Dato In Datos
        Debug.Print Dato
    Next Dato
End Sub

Public Sub RellenaMatriz(ByRef Matriz As Variant)
    Dim i As Long
    ReDim Matriz(1 To 20)
    For i = 1 To 20
        Matriz(i) = "Dato " & Format(i, "00")
    Next i
End Sub
```

De este código lo único que no hemos visto es :

```
Matriz(i) = "Dato " & Format(i, "00")
```

La función Format la veremos detenidamente más adelante. Aquí lo que hace es añadir "01", "02", "03", ..., "10", de forma sucesiva a la cadena "Dato ".

El resultado de este procedimiento es

```
Dato 01  
Dato 02  
Dato 03  
Dato 04  
Dato 05  
Dato 06  
Dato 07  
Dato 08  
Dato 09  
Dato 10  
Dato 11  
Dato 12  
Dato 13  
Dato 14  
Dato 15  
Dato 16  
Dato 17  
Dato 18  
Dato 19  
Dato 20
```