Comencemos a programar con

VBA - Access

Entrega 05

Tipos de datos y Declaraciones

Eduardo Olaz

Declaración de variables

En entregas anteriores hemos visto cómo se declaran las variables. En esta entrega vamos a ampliar conceptos.

Una variable es un elemento del código que apunta a una dirección de memoria en donde se almacena un dato.

Haciendo referencia a la variable se puede devolver el dato al que apunta e incluso modificarlo.

Las **constantes** son similares a las variables, sólo que su contenido se le asigna en el momento en el que se declaran, y después no es posible cambiarlo.

Hay tres temas que debemos considerar en una variable

- El **nombre** de la variable
- El **tipo** de dato al que apunta
- El **ámbito** en el que es visible.

Construcción del nombre de una variable (o constante).

Nombre

El **nombre** de una variable está compuesto por caracteres ASCII.

Para su construcción hay que ceñirse a las siguientes reglas:

- No se pueden usar caracteres que tienen un uso especial en Access, como son el Punto "**.**",los paréntesis "**(**" "**)**", la barra vertical "**|**", o los caracteres que se pueden utilizar como operadores; entre ellos
  + - / * < >.
- Una variable debe empezar por una letra ó por el signo de subrayado
  Estos nombres serían correctos, lo que no quiere decir que sean aconsejables.
  A123456 _Pepe R2P2
- El nombre de una variable no puede tener espacios en blanco.
  Por ejemplo no sería válido el nombre Apellidos Nombre.
  En cambio sí sería válido Apellidos_Nombre ó ApellidosNombre.
- Una variable puede terminar con un carácter de declaración de tipo
  % & ! # @ $
  Estos caracteres sólo se pueden usar al final del nombre de la variable.
  Nota: Estos caracteres también se pueden usar para declarar el tipo de dato que devuelve una función.
  Por ejemplo esta cabecera de función sería válida:
  ```
  Public Function Nombre$()
  ```
  Que sería equivalente a:
  ```
  Public Function Nombre() As String
  ```
- No se puede usar como nombre de variable una palabra reservada de VBA.
  Por ejemplo no se pueden usar String, integer, For, If como nombres de variable.
- El nombre de una variable puede tener hasta 255 caracteres - aunque yo de ti no lo intentaría forastero -.

- No se pueden declarar dos variables con el mismo nombre dentro del mismo procedimiento o en la cabecera de un mismo módulo.

Tipos de datos

Además de las **Variables** hay otra serie de elementos que manejan datos, como son las **Constantes**, Procedimientos **Sub** y procedimientos **Function** que son capaces de manejar datos de distintos tipos, e incluso las funciones que devuelven datos.

Pero ¿qué tipos de datos podemos manejar? Y ¿qué características tienen?

Hay varios tipos de datos.

Entre ellos podemos definir

Numéricos

Booleanos

Fecha / Hora

De texto (cadenas)

Variant

De objeto

Registros de datos definidos por el usuario, . . .

Datos numéricos

Existen dos familias de datos numéricos.

Datos numéricos de **número entero**

Datos numéricos de **coma flotante**.

Como **datos enteros** tenemos los siguientes tipos:

Nombre del Tipo	Tamaño	Valor inferior	Valor Superior	Sufijo	Prefijo
Byte	1 Byte	0	255		**byt**
Integer	2 Bytes	-32.768	32.767	**%**	**int**
Long	4 Bytes	-2.147.483.648	2.147.483.647	**&**	**lng**

Por **Sufijo** entendemos un **carácter de definición de tipo** que se puede añadir a la Variable ó Constante, para definir el tipo al que pertenece.

Por ejemplo `Dim ValorLong&` declara implícitamente la variable `ValorLong` como **Long**.

Esta declaración equivale a la siguiente

```
Dim ValorLong as Long
```

Si hubiéramos hecho `Dim ValorLong` a secas, la variable `ValorLong` se hubiera declarado como **Variant**.

El uso de estos caracteres de definición de tipo, es una herencia del Basic primitivo.

Esos caracteres Sufijo se pueden también aplicar a las declaraciones de constantes

```
Public Const MesesAño As Integer = 12
```

Equivale a

```
Public Const MesesAño% = 12
```

Si se utilizan estos caracteres, no hay que utilizar la declaración explícita de tipo.

Por ejemplo

```
Dim Asistentes% As Integer
```

Daría error ya que sería una declaración redundante porque `Dim Asistentes%` ya ha definido `Asistentes` como **`Integer`**.

El uso del sufijo es aplicable también a los valores. Esto puede ser muy útil en determinadas circunstancias.

Por ejemplo intenta hacer lo siguiente:

Abre el editor de código, ponte en la ventana Inmediato y escribe lo siguiente:

```
a = 256 * 256 : Print a
```

Un comentario: los dos puntos actúan como separador de las dos sentencias.

Al presionar la tecla [**Intro**] nos aparece un bonito mensaje de error de desbordamiento.

Modifiquemos la línea anterior y pongamos lo siguiente (siempre en la ventana inmediato).

```
a = 256& * 256 : Print a
```

Ahora sí nos imprime `65536`

El error se produce porque VBA deduce que en la primera expresión debe producir como máximo un valor del tipo **`Integer`** y `65536`, que es su resultado, está por encima del máximo admitido para un **`Integer`**.

No me preguntéis por qué lo hace, alguien de Microsoft tomó esta decisión en su momento.

En la segunda expresión, al poner uno de los términos como **`Long`** usando `256&`, hace que la expresión la considere como **`Long`**, que admite perfectamente el valor resultante.

Para ver que el error no lo produce a, sino la expresión, escribe directamente en la ventana inmediato lo siguiente:

```
Print 256 * 256
```

Se produce exactamente el mismo error.

En cambio si escribimos

```
Print 256 * 256&
```

El error desaparece.

Prefijo

Por prefijo entenderemos el conjunto de letras que “es aconsejable” poner delante del nombre de una variable, para indicarle a la persona que escribe, ó lee, el código, el tipo de dato que contiene una variable.

> Aquí voy a tratar de seguir, no de forma absoluta, algunas de las llamadas ***“Convenciones de nombres de Leszynski para Microsoft Access”***.
>
> Incluyo un resumen de las normas de Leszynski en el **Apéndice 02**.
> También puedes encontrar una descripción sucinta del convenio de Leszynski entre las páginas 19 y 27 de este documento.
> http://guajiros.udea.edu.co/fnsp/Documentos/Direccion/SII/Normalizacion.pdf
> Además incluye interesantes reglas sobre normas para el desarrollo de programas.
>
> También podéis encontrar información en el sitio de McPegasus:
> http://www.iespana.es/mcpegasus/CONTENT/leszynski.htm

> En las páginas del MSDN de Microsoft podéis encontrar información muy interesante sobre la normativa de codificación:
> http://msdn.microsoft.com/library/spa/default.asp?url=/library/SPA/vbcn7/html/vbconProgrammingGuidelinesOverview.asp

La ventaja de utilizar estos métodos se aprecia inmediatamente. Así, sólo con leer que una tiene como nombre **`lngDiasTrabajados`** podríamos deducir que es una variable de tipo Long que probablemente sirva para manejar los días trabajados.

Si en una línea de código aparece el nombre **cmdSalir**, podríamos deducir que es un botón de comando que probablemente sirva para salir de un formulario u otro sitio.

Igualmente **lblNombre** es una etiqueta y **txtApellido1** es un cuadro de texto. Ya iremos viendo estas cositas poco a poco.

Repito que los prefijos, **int**, **lng**, **txt**, **lbl**, **cmd**, etc. no modelan el tipo de contenido de la variable, sino que sirven como información adicional para la persona que escribe ó lee el código.

Números de Coma Flotante

Los números de coma flotante son unos tipos de datos que admiten valores no enteros.

Por ejemplo estos son valores de coma flotante:

3.14159265358979 2.71828182845905 2856.1# 4.00!

Como **datos de coma flotante** tenemos los siguientes tipos:

Nombre	Tamaño	Negativos	Positivos	Sufijo	Prefijo
`Single`	4 Bytes	De -3,402823E38 a -1,401298E-45	De 1,401298E-45 a 3,402823E38	!	sng
`Double`	8 Bytes	-1.79769313486231E308 -4,94065645841247E-324	4,94065645841247E-324 1,79769313486232E308	#	dbl
`Currency`	8 Bytes	-922337203685477,5808 a 0	0 a 922337203685477,5807	@	cur
`Decimal`	12 Bytes	-79.228.162.514.264.337 .593.543.950.335	79.228.162.514.264.337 .593.543.950.335	&	dec

Por **Sufijo** entendemos un **carácter de definición de tipo** que se puede añadir a la Variable

Para el manejo de valores monetarios se suele utilizar el tipo `Currency`.

Este tipo no da errores de redondeo y permite manejar hasta magnitudes de 15 dígitos exactos en su parte entera. En posteriores capítulos haremos mención a algunos errores de redondeo que pueden llegar a dar los tipos de coma flotante.

Tipo Decimal

El tipo `Decimal` es un tipo muy especial.

Permite trabajar con hasta 29 dígitos enteros exactos ó hasta con 28 dígitos exactos en su parte decimal.

Es un tipo con muy poco uso en Access, ya que normalmente no se necesita ese tipo de precisión, y además resulta engorroso su uso.

Algunas de sus peculiaridades se explican en el Apéndice 01.

Tipo Date

El tipo `Date` es un tipo de dato adecuado para manejar datos de tipo Fecha / Hora.

El valor 0 representa las 0 horas del 30 de diciembre de 1899.

La parte entera representa el número de días que han pasado desde esa fecha.

La parte decimal representa el tanto por 1 de día adicional que ha pasado.

Por ejemplo, ahora son las 18 horas 15 minutos 30 segundos del 20 de enero del 2005

Internamente este dato lo guarda como 38372,7607638889.

¿Qué quiere decir?

Han pasado 38372 días desde las 0 horas del 30 de diciembre de 1899

Pero además ha transcurrido 0,7607638889 días, lo que equivale a 18 horas 15 minutos 30 segundos frente a 24 horas

Si pasamos la hora actual a segundos nos da 3600 * 18 + 15 * 30 + 30 → 65730 segundos

Un día completo tiene 3600 seg/hora * 24 horas = 86400

Si dividimos 65730 entre 86400 nos da aproximadamente 0,7607638889

Para ver cómo utilizar un tipo date, vamos a crear un procedimiento Sub en un módulo estándar que nos permita ver lo explicado en las líneas anteriores.

En realidad un tipo **Date** es un tipo **Double** y como él utiliza 8 Bytes.

Por cierto, el prefijo para indicar que es tipo **Date** es **dat**.

Su código será el siguiente

```
Public Sub PruebaTipoDate()
    Dim datFechaActual As Date
    Dim datSemanaProxima As Date
    Dim datAyer As Date
    Dim datMañana As Date

    datFechaActual = #1/20/2005 6:15:30 PM#
    datSemanaProxima = datFechaActual + 7
    datAyer = datFechaActual - 1
    datMañana = datFechaActual + 1

    Debug.Print "Tipo Doble", CDbl(datFechaActual)
    Debug.Print "Ahora", datFechaActual
    Debug.Print "Próx. Semana", datSemanaProxima
    Debug.Print "Ayer", datAyer
    Debug.Print "Mañana", datMañana
End Sub
```

Si desde la ventana inmediato escribimos

```
PruebaTipoDate y presionamos [Enter] nos mostrará:


Tipo Doble      38372,7607638889
Ahora          20/01/2005 18:15:30
```

```
Próx. Semana  27/01/2005 18:15:30
Ayer          19/01/2005 18:15:30
Mañana        21/01/2005 18:15:30
```

Puntualizaciones:

Ya os habréis dado cuenta que VBA utiliza el Inglés como lenguaje, y más concretamente el Inglés americano. Esto hace que los datos usen ese mismo modelo.

Por ejemplo, la coma flotante nuestra, ellos la cambian por el punto.

Algo así pasa con las fechas. El formato americano de fecha usa el siguiente orden:

Mes / Día / Año

Por eso en la línea

```
datFechaActual = #1/20/2005 6:15:30 PM#
```

se pone `1/20/2005`, en vez de `20/1/2005` como hacemos nosotros.

Esto es importante recordarlo ya que nos evitará futuros errores.

Además VBA tiene un comportamiento que puede despistar.

Si hubiéramos hecho

```
datFechaActual = #20/1/2005 6:15:30 PM#
```

Hubiera funcionado exactamente igual, ya que VBA deduciría que el 20 no puede referirse al mes, pues existen sólo 12 meses, por lo que deduce “por su cuenta” que lo que en realidad queríamos era introducir el 20 como día y el 1 como mes.

Estos comportamientos que a veces tiene VBA de tomar decisiones por su cuenta es algo “que me saca de quicio” (nada ni nadie es perfecto).

Resumiendo

La forma de construir una fecha / hora es

Mes / Día / Año Horas:Minutos:Segundos

Dependiendo de la configuración de Windows que tengas, si por ejemplo introduces

```
datFechaActual = #1/20/2005 18:15:30#
```

VBA te lo puede convertir automáticamente a

```
datFechaActual = #1/20/2005 6:15:30 PM#
```

añadiéndole por su cuenta el PM

Y no sólo toma decisiones de ese tipo.

Si escribes

```
datFechaActual = #1-20-2005#
```

VBA te cambia automáticamente a

```
datFechaActual = #1/20/2005#
```

Para construir una fecha no es necesario introducirle la hora.

Tampoco es necesario introducirle el año entero.

Estas fechas son perfectamente válidas

```
#1/20/2005 18:15:30#
#1/20/2005 6:15:30 PM#
#1/20/2005#
#1/20/05#
#01/20/05#
#18:15:30#
```

Puedes obtener más información en la ayuda de Access.

Hay toda una batería de funciones de VBA para el manejo y manipulación de fechas.

Estas funciones las veremos más adelante.

Tipo Boolean (boleano)

El tipo **`Boolean`** (boleano) es un tipo de dato adecuado para manejar datos de tipo **Sí** / **No**, **True** / **False**.

Sorprendentemente el tipo **`Boolean`** ocupa 2 Bytes, supongo que por compatibilidad con otros tipos.

Es como si estuviera definido como de tipo **`Integer`**, y como veremos, algo de esto hay.

El prefijo para un boleano es **bln**.

A una variable Boleana se le puede asignar un valor de varias formas

Por ejemplo esto sería correcto:

```
Public Sub PruebaTipoBoleano()
    Dim blnDatoBoleano As Boolean

    blnDatoBoleano = True
    Debug.Print "Valor True " & blnDatoBoleano
    blnDatoBoleano = False
    Debug.Print "Valor False " & blnDatoBoleano
    blnDatoBoleano = -1
    Debug.Print "Valor -1 " & blnDatoBoleano
    blnDatoBoleano = 0
    Debug.Print "Valor 0 " & blnDatoBoleano
    blnDatoBoleano = 4
    Debug.Print "Valor 4 " & blnDatoBoleano
    Debug.Print "Valor entero de Verdadero " & CInt(blnDatoBoleano)
End Sub
```

Este código nos mostrará en la ventana Inmediato

```
Valor True Verdadero
Valor False Falso
Valor -1 Verdadero
Valor 0 Falso
Valor 4 Verdadero
```

```
Valor entero de Verdadero -1
```

Nota: La instrucción `CInt(blnDatoBoleano)` convierte a un valor Entero el contenido de la variable `blnDatoBoleano`.

Para asignar a una variable booleana el valor **falso** le podemos asignar **`False`** ó **`0`**.

Para asignarle cierto lo podemos hacer pasándole directamente **`True`** ó **cualquier valor numérico diferente de Cero**, incluso aunque sea de coma flotante.

Si convertimos el contenido de una variable Boleana que contenga **`True`** a un valor numérico, nos devolverá **`-1`**.

Otra forma de asignarle un valor sería asignarle una expresión que devuelva `True` ó `False`.

Por ejemplo

```
blnDatoBoleano = 8<7
```

Asignaría `False` a `blnDatoBoleano` (es falso que 8 sea menor que 7).

```
blnDatoBoleano = 8>7
```

Asignaría `True` a `blnDatoBoleano`.

Tipo String (cadena)

El tipo **`String`** está especializado en almacenar datos de tipo Cadena (Texto).

El carácter para definir el tipo es el signo de Dólar **`$`**.

Hay dos tipos String

- String de longitud variable
- String de longitud fija

Para asignar una cadena de **longitud variable**, la declaración se hace de forma similar al resto de los tipos

```
Dim strNombre As String
Dim strNombre$
Public strNombre As String
Private strNombre As String
```

Para declarar una variable como de longitud fija se utiliza el asterisco (**`*`**) y a continuación el número de caracteres que va a contener

```
Dim strCuentaCorriente As String * 20
```

Para declarar cadenas de longitud fija no se puede usar el carácter **`$`**.

Por ejemplo, esto daría error

```
Dim strCuentaCorriente$ * 20
```

Una cadena de longitud variable podría en teoría almacenar hasta 2^31 caracteres, es decir unos 2000 millones de caracteres. Lógicamente siempre que el PC tuviera suficiente memoria.

Una cadena de longitud fija puede contener hasta 65536 caracteres (64 KBytes).

Si por ejemplo hacemos:

```
Dim strCliente As String * 40
strCliente = "Pepe Gotera"
```

`strCliente` contendrá la cadena **`Pepe Gotera`** seguida de 29 **espacios en blanco** (para completar los 40 que debe contener la variable.

Tipo Variant

El tipo **`Variant`** es un tipo de dato que puede contener prácticamente cualquier tipo de datos. El prefijo que se suele utilizar para identificar una variable **`Variant`** es **`var`**.

Hay excepciones, por ejemplo **no puede contener una cadena de longitud fija**.

Cuando declaramos una variable o escribimos la cabecera de una función, sin especificar el tipo que va a contener ó devolver, implícitamente las estamos declarando como de tipo Variant.

> *Si declaramos una constante, sin especificar su tipo, el tipo que tome lo hará en función del dato que le asignemos en la propia declaración.*
>
> *Por ejemplo si declaramos en la cabecera de un módulo estándar*
>
> ```
> Global Const conEmpresa = "Manufacturas ACME s.l."
> ```
>
> *Hace que la constante* `conEmpresa` *se configure como del tipo* **`String`**.
>
> *He usado aquí el modificador de alcance* `Global`. *Este modificador del Alcance de la variable sería, en este caso, equivalente a haber utilizado* `Public`.
>
> *Las declaraciones de constante siguientes serían equivalentes a la anterior*
>
> ```
> Public Const conEmpresa = "ACME s.l."
> Public Const conEmpresa As String = "ACME s.l."
> Public Const conEmpresa$ = "ACME s.l."
> ```

La forma de declarar una variable **explícitamente** como Variant es la siguiente.

```
ModificadorDeAlcance Nombre As Variant
Dim varMiVariable As Variant
```

Ya hemos dicho que si se declara una variable sin especificar su tipo, implícitamente la declara como Variant.

La última declaración sería equivalente a

```
Dim varMiVariable
```

Empty, Null, Cadena vacía

Existen unos valores especiales que no son ni números ni texto.

Por ejemplo **`Empty`** (vacío) es valor que toma por defecto el tipo Variant después de ser declarado y antes de que se le asigne otro valor.

Para ver si una variable de tipo Variant ha sido inicializada, podemos utilizar la función **`IsEmpty`**`(variable)` que devolverá `True` ó `False`.

Para comprobarlo, vamos a crear un procedimiento que utilice la función `IsEmpty` sobre una variable, antes y después de asignarle un valor; en este caso `Null`.

```
Public Sub PruebaEmpty()
    Dim varVariant As Variant
    Debug.Print "Variable vacía = "; IsEmpty(varVariant)
    varVariant = Null
    Debug.Print "Variable vacía = "; IsEmpty(varVariant)
    Debug.Print "Valor actual = "; varVariant
End Sub
```

Si llamamos a este procedimiento desde la ventana Inmediato, nos imprimirá en ella lo siguiente:

```
Variable vacía = Verdadero
Variable vacía = Falso
Valor actual = Nulo
```

La primera vez que se ejecuta `IsEmpty(varVariant)` devuelve `True` porque todavía no se le ha asignado ningún valor. Tras hacer `varVariant = Null` devuelve `False`.

`Null` es un dato nulo. Equivale al código `ASCII` **0**.

Por ejemplo, un cuadro de texto que no contenga ningún valor, devolverá el valor **`Null`**.

El valor **`Null`** sólo se puede asignar a una variable del tipo **`Variant`**, o a una variable del tipo **`Objeto`**.; tipo de variable que veremos en otra entrega.

Una Cadena Vacía es un dato del tipo String que no contiene ningún carácter.

La cadena vacía se puede asignar a una variable del tipo **`Variant`** y a una variable de tipo **`String`**.

Para asignar la cadena vacía a una variable se escribe la variable seguida del operador de asignación **=** y a continuación 2 comillas dobles.

```
strCadenaVacia = ""
```

Declaraciones múltiples en una línea.

VBA permite declarar múltiples variables en una misma línea de código.

La forma correcta de hacerlo sigue esta estructura:

```
Alcance Variable1 As Tipo, Variable2 As Tipo, Variable3 As Tipo
```

Por ejemplo, esto sería correcto

```
Dim lngUnidades As Long, strNombre As String
```

Otros lenguajes de programación permiten hacer una declaración del siguiente estilo:

```
Dim strNombre, strApellido1, strApellido2 As String
```

Tras esto las tres variables serían del tipo String.

En cambio en VBA, si hubiéramos declarado así las variables, `strNombre` y `strApellido1` serían del tipo **`Variant`**, y `strApellido2` sería del tipo **`String`**.

Valores por defecto

Cuando declaramos una variable, ésta se inicializa con un valor por defecto.

Dependiendo del tipo de dato de la variable el valor que tome por defecto será diferente.

Las variables de tipo **numérico** toman el valor **0**

Las de tipo **fecha** también el valor **0**, que se corresponde a las 0 horas del 30 de diciembre de 1899.

Las de tipo **boleano** el valor **`False`**, ó **0**.

Las de tipo **cadena** la cadena vacía **""**.

Las de tipo **variant**, el valor **`Empty`**.

Ámbito ó alcance de una declaración

Hemos visto varias instrucciones que afectan al alcance ó visibilidad de una Variable, Constante o Procedimiento.

Estas instrucciones son:

`Dim`

`Public`

`Global`

`Private`

Hay otras formas de declarar Variables como son las instrucciones:

`Static`

`Friend`

Dim

Si la instrucción **`Dim`** se utiliza para declarar una variable en la cabecera de un módulo, esa variable sólo será "**visible**" por los procedimientos que estén dentro de ese módulo.

De forma semejante, si la instrucción **`Dim`** se utiliza dentro de un procedimiento, esa variable sólo podrá ser vista por el código del interior del procedimiento.

Private

La instrucción **`Private`** se suele utilizar para definir de forma explícita que una constante, variable o procedimiento sólo van a ser visibles desde dentro de un módulo.

Se suele utilizar para la declaración de variables y constantes en las cabeceras de módulos, así como para definir el alcance de los procedimientos de ese módulo.

Dentro de un procedimiento se usa **`Dim`** en vez de **`Private`**.

Public

La instrucción **`Public`** se suele utilizar para definir que una constante, variable o procedimiento van a ser visibles desde cualquier parte de la aplicación.

Se suele utilizar para la declaración de variables y constantes en las cabeceras de módulos, así como para definir el alcance de los procedimientos de ese módulo.

Por ello no es factible declarar una variable como **`Public`** dentro de un procedimiento.

Sí se puede en cambio, declararla en la cabecera de cualquier módulo **estándar** o **de clase**.

Nota: Una aclaración respecto a los elementos declarados en un módulo de clase.

Si declaramos una variable, constante ó procedimiento como **Public** en un módulo de clase, por ejemplo en un formulario, para poder usarlo hay que hacerlo a través de una **Instancia** de ese módulo de clase.

> A las variables y constantes públicas de un módulo de clase, así como a los procedimientos Property, se les llama **Propiedades** de la clase.
>
> Al resto de los procedimientos de la clase que sean públicos se les llama **Métodos** de la clase.

Cuando veamos las clases, veremos que una **instancia** es un **ejemplar** de una clase.

La clase como tal es el propio código, pero una instancia es un objeto creado mediante la clase. Se que es un poco críptico, pero más adelante lo entenderéis sin problema.

Por ejemplo una instancia de un formulario es el propio formulario, o si tenemos la clase Persona y creamos una persona llamada Pepe, la instancia de la clase es Pepe.

Si hemos declarado una variable, por ejemplo **Nombre** como **Public** para poder usarla deberemos hacer referencia a ella mediante la instancia de la clase, que es Pepe

```
Pepe.Nombre
```

Lo mismo ocurriría si hemos declarado como **public** la función **Edad**, es decir el método **Edad**.

```
Pepe.Edad
```

Así mismo, si en el formulario **Pedidos** declaramos **datFechaUltimoPedido** como **Public**, para acceder a ese dato lo tenemos que hacer a través del propio formulario.

```
Pedidos.datFechaUltimoPedido
```

Desde el código del formulario, no es necesario hacer referencia al mismo formulario, aunque sí puede hacerse utilizando la palabra clave **Me**, que es una variable declarada implícitamente y que representa a la instancia de la clase u objeto creado mediante la clase.

Desde el propio formulario sería lo mismo

```
Me.datFechaUltimoPedido
```

que

```
datFechaUltimoPedido
```

No tenéis por qué entender todo esto todavía. Lo iré comentando poco a poco y lo desarrollaré en la entrega en la que hablemos de las **Clases**.

Global

La instrucción **Global** se utiliza de forma semejante a **Public**, dentro de módulos estándar.

Yo personalmente no la suelo usar; prefiero utilizar la instrucción **Public**.