Lección 6

En esta lección vamos a familiarizarnos con las estructuras de repetición, las cuales nos sirven para realizar una misma instrucción un número determinado de veces o indeterminado dependiendo de una condición.

Introducción a las estructuras de repetición

El número de veces que se repetirá la instrucción o instrucciones puede depender de un contador o de una condición.

En esta lección vamos a ver los dos tipos de bucles: con contador o con condición.

For... Next

Esta es una estructura de repetición o bucle, la cual depende de un contador que nos controla el número de veces que se deberá repetir una o varias instrucciones.

En esta estructura siempre deberemos especificar la variable (**contador**), un **valor inicial** y un **valor final**. Normalmente el contador incrementará de uno en uno a no ser que nosotros indiquemos lo contrario.

La estructura del bucle utilizando un contador es la siguiente:

```
For Contador = Inicio To Fin [Step Incremento]
    [Instrucciones]
Next Contador
```

Vamos a explicar las diferentes partes de esta estructura:

Contador: Aquí es donde nosotros escribiremos el nombre de la variable que queremos utilizar como contador.

Inicio: Valor inicial de la variable.

Fin: Valor final de la variable. Cuando la variable llegue a este valor, el bucle no se volverá a realizar.

Step: Esta instrucción es opcional. Si no la ponemos el contador irá incrementando de uno en uno. Si especificamos un número detrás de **Step** hacemos que nuestro contador aumente un número determinado de pasos.

Incremento: Número que marcará los pasos que debe aumentar el contador. Este número puede ser tanto positivo como negativo. Eso sí, siempre deberemos tener cuidado con los valores iniciales y finales para que no se produzca ningún tipo de error. No podemos hacer, por ejemplo, que el valor inicial sea 10 y el final 1 siempre y cuando no pongamos como **step** un valor negativo.

Instrucciones: Aquí escribiremos la o las instrucciones que queremos que se repitan.

Next Contador: Línea que indica que se termina el bucle y hace que aumente el contador según el valor que nos indique **step** en caso de tenerlo.

. Práctica 1

Vamos a realizar una simple aplicación en el que utilizaremos una estructura de repetición utilizando un contador.

La aplicación consistirá en una simulación de una tirada de un dado.

Te iremos especificando que tipo de objetos deberás añadir en nuestro formulario y algunas de las propiedades que deberás cambiar. El aspecto de los objetos y su situación corren por tu cuenta. Puedes poner tantos objetos **Label** como quieras para aclarar para que sirven cada uno de los elementos insertados en el formulario.

*1. Sitúa en un formulario nuevo un **ListBox** al que deberás poner como **(Nombre): Dado**.*

Aquí será donde el ordenador nos muestre las diferentes tiradas que realizamos.

*2. Coloca un **CommandBotton**, que tendrá como **(Nombre)** y **Caption: Tirada**.*

Al pulsar este botón se realizarán las diferentes tiradas.

*3. Coloca un **TextBox** al que pondremos como **(Nombre): NumTiradas**. Borra el contenido que aparece por defecto dentro de este objeto.*

Aquí será donde indiquemos cuantas tiradas queremos realizar.

Una vez colocados los objetos vamos a pensar en el código.

Nosotros en esta práctica queremos que se realicen tantas tiradas de dado como nos indique el usuario dentro del **TextBox**. Para esto nos interesaría crear una estructura de repetición que debería empezar en **1** y terminar en el **número que indica el usuario**. Los incrementos que sufrirá el contador deberá ser de uno en uno, por lo que la parte del **step** no la especificaremos.

4. Haz doble clic dentro del botón y escribe el siguiente código.

```
For Contador = 1 To NumTiradas.Text
   Dado.AddItem (Int(6*Rnd)+1)
Next Contador
```

En la primera línea de este pequeño código, que más adelante depuraremos, hemos iniciado el **contador** (nueva variable) a 1. No hace falta que definamos la variable. Al no definirla esta es de tipo Variant[1].

En esta primera línea también definimos en que valor queremos que termine el bucle. Este valor será el valor que introduzca el usuario dentro del **TextBox**.

Generar valores aleatorios

En la segunda línea, **Dado.AddItem (Int(6*Rnd)+1))**, hacemos que **Visual Basic** nos busque un valor aleatorio. Esto lo conseguimos con la instrucción **Rnd**. Nosotros como queremos conseguir un número aleatorio dado un intervalo, del **1** al **6** (valores que tiene un dado común), necesitamos utilizar una estructura determinada:

Int ([*Valor superior*] – [*Valor inferior*] + 1) * Rnd + [*Valor inferior*]

Valor inferior: nos indica el valor mínimo que tiene el intervalo.

Valor superior: nos indica el valor máximo del intervalo.

En nuestro ejemplo esto quedaría de la siguiente manera. Recuerda que queremos valores enteros, por eso utilizamos la instrucción **Int(Valor)**, entre el **1** y el **6**.

Int(6-1+1*Rnd)+1 Resolviendo las operaciones la instrucción quedaría de la

siguiente forma **Int(6*Rnd)+1**. Con esto conseguiríamos números aleatorios entre el **1** y el **6**, ambos inclusive.

Pongamos otro ejemplo: imaginemos que ahora queremos obtener valores aleatorios entre el **10** y el **20**. La instrucción quedaría de la siguiente forma. **Int(20-10+1*Rnd)+10** resolviendo las operaciones, la instrucción quedaría así: **Int(11*rnd)+10**.

Añadir valores a una lista

Para añadir elementos a una lista deberemos utilizar la instrucción **AddItem**. Cada vez que se pasa por esta línea se inserta un nuevo elemento ocupando el puesto de **último índice + 1**. Recuerda que el primer elemento ocuparía la posición con **índice 0**. La estructura de esta instrucción es la siguiente:

[*Nombre de la lista*].AddItem [*Cadena a añadir*]

Nombre de la lista: es el nombre que le hemos puesto a la lista donde queremos que se añadan los diferentes elementos.

Cadena a añadir: es el valor, cadena, variable... que queremos añadir a nuestra lista.

Observa que esta instrucción, aunque se trate de una asignación, no utiliza el signo igual.

En nuestra aplicación queremos añadir el valor aleatorio obtenido anteriormente. Así que la línea de código quedará de la siguiente manera: **Dado.AddItem (Int(6*Rnd)+1)**

Observaciones con números aleatorios

5. Inicia una ejecución de prueba.

*6. Indica que quieres realizar **5** tiradas.*

*7. Pulsa sobre el botón: **Tirada**.*

8. Observa con detenimiento la secuencia de números que han aparecido en la lista.

9. Detén la ejecución del programa.

10. Vuelve ha ejecutar el programa.

*11. Indica que quieres realizar nuevamente **5** tiradas.*

*12. Pulsa sobre el botón: **Tirada**.*

13. Observa la secuencia de tiradas de la lista.

Si recuerdas la primera secuencia que ha aparecido en nuestra primera ejecución y la comparas con la actual, podrás ver que es exactamente igual. Esto es debido a que, mientras no indiquemos lo contrario, la secuencia de números aleatorios obtenidos con **Rnd** siempre será la misma. Como podrás ver esto no nos interesa en la gran mayoría de casos, con lo que utilizaremos una nueva instrucción que nos permitirá obtener valores completamente aleatorios.

14. Detén la ejecución del programa.

Inicio de valores aleatorios

Vamos a ver una manera para que cada vez que se inicia el programa los valores que se consiguen con la instrucción **Rnd** sean diferentes.

*15. Pulsa **doble clic** sobre el fondo del **formulario**.*

Te aparecerá la ventana de código con un nuevo evento **Form_Load()**. Este evento se ejecuta justo en el momento en el que se carga el formulario. En este caso, como solo disponemos de un formulario, este evento se ejecutará al poner en funcionamiento la aplicación.

*16. Escribe dentro de dicho evento **Randomize**.*

Esta instrucción nos sirve para iniciar con valores, cada vez diferentes, la secuencia de números aleatorios. De tal forma que cada vez que ejecutemos nuestra aplicación obtendremos secuencias aleatorias diferentes.

*17. Vuelva a realizar los pasos del **5** al **13**.*

Pero esta vez observa como la primera y la segunda secuencia son diferentes.

*18. Sin detener la ejecución del programa, vuelve a pedir que se realicen **5** tiradas más.*

Observa como en la lista se han añadido **5** valores más a los que ya teníamos, de esta forma ahora tenemos **10** valores (**5** tiradas anteriores y **5** actuales). Si nosotros seguimos realizando tiradas, los valores de las nuevas tiradas se van añadiendo a la lista de forma indefinida. Observa que cuando la cantidad de valores superan el tamaño de la lista aparece una barra de desplazamiento vertical que nos permite poder visualizar los valores que hemos conseguido en tiradas anteriores.

Si deseas ver los valores de la lista, en lugar de en filas en columnas deberías acceder a la propiedad **Columns** de la lista y cambiar el número de columnas que deseas ver. Si modificas este valor, en el momento que tengamos más elementos de los que caben en la lista aparecerá una barra de desplazamiento horizontal en lugar de vertical. Prueba esta propiedad.

A nosotros, en esta práctica, lo que nos interesaría es conseguir que cada vez que se realice una nueva tirada se borre el contenido de la tabla y aparezcan las nuevas tiradas.

Borrar una lista

Vamos a ver como podemos borrar la lista cada vez que realizamos una tirada nueva.

19. Detén la ejecución del programa.

*20. Pulsa doble clic en el botón: **Tirada**.*

21. Completa el código que ya tienes, para que quede de la siguiente forma:

```
Dado.Clear
For Contador = 1 To NumTiradas.Text
   Dado.AddItem (Int(6*Rnd)+1)
Next Contador
```

La instrucción **Clear** sirve para borrar el contenido de la tabla. La sintaxis de esta instrucción es la siguiente: **[*Nombre lista*].Clear**.

De esta forma cada vez que queramos realizar una nueva tirada, primero se

borrará el contenido de la lista y después se añadirán los elementos nuevos. Cada vez que se borran los elementos de la lista, el **Índice** de la lista vuelve a tener como valor **0**.

Filtrar la entrada de valores

En este apartado vamos ha hacer que el usuario solo pueda poner números en el número de tiradas que quiere realizar, y no pueda introducir ningún tipo de carácter más. Esto es una medida de depuración del programa, ya que de esta forma evitamos que la aplicación aborte al producirse un error.

Veamos que ocurre si introducimos una letra en el número de tiradas deseadas.

. Practica 2

1. Ejecuta la aplicación.

2. Escribe una letra en la casilla para indicar el número de tiradas que deseas realizar.

*3. Pulsa el botón: **Tirada**.*

Observa como te aparece una ventana indicando que se ha producido un **error**. El mensaje de error es: **No coinciden los tipos**. Esto quiere decir que **Visual Basic** no ha podido utilizar lo que nosotros hemos escrito en el interior del número de tiradas como contador para nuestro bucle. **Visual Basic** necesita un número y no una letra.

*4. Pulsa el botón **Terminar**, que aparece en la pantalla de **error**.*

De esta forma podemos volver a la edición del código.

Esta ventana de error es la que tendremos que evitar en muchos casos, para que el usuario no se encuentre con la aplicación colgada.

*5. Haz doble clic sobre el **TextBox**.*

Observa como el evento que se ha abierto ha sido **Change**. El código que escribimos dentro de este evento se ejecutará en el momento en el que se produce un cambio dentro del **TextBox**.

*6. Abre la lista desplegable de los eventos de este objeto y selecciona **KeyPress**.*

Fíjate como ha aparecido un nuevo procedimiento: **Private Sub NumTiradas_KeyPress(KeyAscii As Integer)**. La parte que se encuentra dentro de los paréntesis, devuelve al procedimiento un valor **KeyAscii** siendo este un valor numérico que representa la tecla que se ha pulsado. Esta tabla tiene 256 elementos numerados del 0 al 255, y cada uno de ellos representa un carácter diferente.

*7. Inserta estas líneas de código dentro de **KeyPress**.*

```
If (KeyAscii < 48 Or KeyAscii > 57) Then
   If (KeyAscii <> 8) Then KeyAscii = 0
End If
```

Con estas líneas de código, conseguiremos que el usuario en el momento de pulsar alguna tecla que no sea un número no se escriba dentro del **TextBox**.

8. Realiza una ejecución de prueba e intentar escribir alguna letra.

Observa como no se escribe nada en el interior de este objeto.

*9. Escribe cualquier valor numérico y pulsa **Tirada**.*

10. Detén la ejecución del programa.

Do... Loop

Ahora vamos a ver un tipo de estructura de repetición que depende de una condición. Las instrucciones que hay dentro del bucle se repiten **mientras se cumpla la condición**, mientras la condición sea **Verdadera**.

Tenemos dos tipos de estructuras **Do...Loop**, una en la que se mira la condición antes de realizar ninguna instrucción y otra que se mira después de realizar, al menos, una vez las instrucciones que tenemos dentro del bucle.

Vamos a ver las dos estructuras y después pasaremos a comentar sus diferencias:

Do While [*Condición*] [*Instrucciones*] Loop Do [*Instrucciones*] Loop While [*Condición*]

Condición: lugar reservado para colocar la pregunta que queremos realizar para ver si es **verdadero** o **falso**.

Instrucciones: líneas de código que se ejecutan mientras que la condición sea **verdadera**.

En la primera estructura de repetición lo primero que se mira es la **condición**, si esta se cumple pasamos a realizar las instrucciones que tenemos en el interior del bucle, si no se cumple nos saltamos todas las instrucciones hasta llegar al **Loop** que nos indica el final de dicho bucle.

La segunda estructura de repetición es diferente, primero entramos en el bucle y realizamos todas las instrucciones una vez, después miramos la condición, si esta se cumple volvemos a realizar las instrucciones que tenemos dentro del bucle, por lo contrario si esta no se cumple salimos del bucle.

Es difícil explicar en que momentos se necesitará una u otra instrucción ya que esto dependerá de cada caso y nada mejor que aprenderlo sobre la marcha.

Existen dos estructuras a las que hemos visto antes pero con la diferencia que el bucle se repetirá **mientras no se cumpla la condición**, mientras la condición sea **Falsa**.

La estructura sería la siguiente:
Do Until [*Condición*] [*Instrucciones*] Loop Do [*Instrucciones*] Loop Until [*Condición*]

Observa la diferencia de estas dos estructuras con las vistas anteriormente.

La única diferencia es que en las primeras utilizamos la palabra, **While** y en estas últimas **Until**, por lo demás todo el "funcionamiento" es exactamente igual.

Bucles anidados

En este apartado vamos a ver como podemos anidar, poner dentro de otro, diferentes estructuras de repetición.

Para esto vamos a realizar una práctica en la que intentaremos ordenar una tabla de elementos que inicialmente están desordenados. Para ordenar una tabla existen multitud de métodos diferentes. Algunos de ellos muy simples y poco eficaces, otros son complejos y con un alto grado de eficacia. La dificultad del sistema de ordenación la escogeremos según la cantidad de elementos que deseamos ordenar.

En nuestro caso realizaremos una aplicación que nos ordenará una pequeña tabla que contiene datos aleatorios. Utilizaremos el método de ordenación más sencillo y menos eficaz. Este método, llamado *Método de la burbuja*, es ideal para tablas con pocos datos.

Método de la burbuja

El método de la burbuja se basa en el intercambio de elementos de dos en dos. Si nosotros queremos ordenar la tabla **ascendentemente**, el intercambio de los elementos se produce cuando el primero de ellos es mayor que el segundo. Repitiendo este proceso por lo largo de la tabla conseguimos que el elemento más grande pase a estar en el último lugar de la tabla. El elemento sube por la tabla hasta que ocupa la posición más alta. De ahí viene el nombre de ordenación de la burbuja, el elemento sube como si se tratase de una burbuja dentro de un recipiente.

Los pasos que se siguen exactamente en esta ordenación son los siguientes:

1. Se compara el primer elemento con el segundo de la tabla. Si están desordenados (el primero es más grande que el segundo, en el caso de la ordenación **ascendente**) se intercambian. Luego comparamos el segundo con el tercero, si es necesario los intercambiamos. Continuamos con los intercambios hasta que comparamos el penúltimo con el último.

2. Como segundo paso, volvemos a repetir el primero pero esta vez hasta llegar a comparar el antepenúltimo con el penúltimo, ya que el último elemento ya está ordenado gracias al primer paso.

3. Volvemos a repetir exactamente lo mismo que en el paso uno, pero esta vez con un elemento menos, ya que los dos últimos ya están ordenados.

Este método termina en el momento en el que hemos hecho tantas pasadas como **elementos menos 1** hay en la lista. Realizamos una pasada menos de la cantidad de elementos que hay en la tabla, ya que si todos los elementos de la tabla se han ido ordenando según hemos pasado, como es lógico este último elemento a ordenar ya estará ordenado.

. Práctica 3

Vamos a realizar esta aplicación. Como en todas las prácticas sigue los pasos que te indicamos.

Insertar los elementos

1. Abre un proyecto nuevo.

*2. Inserta un **CommandButton** al que le pondrás como **(Nombre)**: **Nueva**. Cambia su **Caption** y escribe **Nueva**.*

Este botón servirá para borrar la tabla que tengamos en pantalla y crear otra.

Para poder visualizar las tablas que vallamos creando utilizaremos un **ListBox**.

*3. Inserta un **ListBox**. Cambia su tamaño hasta llegar aproximadamente a **855 x 3180**.*

*4. Cambia el **(Nombre)** de este **ListBox** por **Lista**.*

No introduzcas nada en su interior.

*5. Inserta otro **CommandButton**. A este llámale **Ordenar** y ponle como **Caption**: **Ordenar**.*

Al pulsar este botón realizaremos la ordenación de la tabla y la visualizaremos en nuestro **ListBox**.

Recuerda que puedes cambiar todas las propiedades que desees de los objetos insertados en este formulario.

Creación de la tabla

Vamos ha definir la tabla en la que guardaremos todos los valores.

Vamos a pensar como crear esta aplicación para que sea fácil de modificar en el momento en el que deseemos cambiar el número de elementos que componen la tabla. Para ello vamos a crear una **constante** que utilizaremos a lo largo del programa. En el momento que deseemos utilizar una tabla con más o menos elementos y que el programa funcione exactamente igual, solo deberemos cambiar el valor de esta **constante**.

*6. Accede al apartado **General - Declaraciones** de nuestra página de código y escribe lo siguiente:*

```
Const Elementos = 12
```

Con esto crearemos una constante llamada **Elementos** que podremos consultar a lo largo de todo nuestro programa.

Vamos a crear en el mismo apartado una tabla que tenga el número de elementos que marca la constante anteriormente creada. Además esta tabla, para facilitar la comprensión de nuestro código, pondremos como primer elemento el número **1** y como último **Elemento**.

7. Escribe la siguiente línea de código a continuación de la que ya teníamos:

```
Dim Tabla(1 To Elementos) As Integer
```

Observa la declaración del tamaño de la tabla, desde el elemento número 1 al elemento **Elementos**.

La tabla la hemos definido como **Integer** (números enteros).

Iniciar proyecto

Vamos a escribir el código necesario para que al iniciar el proyecto nos aparezca una tabla de número aleatorios dentro de nuestra tabla.

8. Pulsa un clic en el fondo del escritorio.

Seguidamente te aparecerá la ventana de código. Observa el evento que tenemos abierto. **Private Sub Form_Load()** Esto nos indica que todo lo que escribamos dentro de este evento se realizará en el momento en el que carguemos el formulario.

9. Copia las siguientes instrucciones dentro de dicho evento.

```
Private Sub Form_Load()
   Randomize
   For Contador = 1 To Elementos
      Tabla(Contador) = Int((9 * Rnd) + 1)
      Lista.AddItem Tabla(Contador)
   Next Contador
End Sub
```

Observa que en la primera línea, dentro del evento, hemos escrito la instrucción **Randomize** para iniciar la función de números aleatorios.

En la segunda línea hemos iniciado un bucle que se repetirá hasta que **Contador** llegue hasta el número de elementos que hemos definido en **Elementos**. Observa como la variable **Contador** no la hemos definido anteriormente.

Dentro de este bucle se realizará el relleno de los elementos de nuestra **Tabla** con números aleatorios generados mediante la instrucción **Int((9 * Rnd) + 1)**. Observa que para movernos por la tabla utilizamos como índice, nuestro **Contador**.

A la vez que llenamos la tabla vamos añadiendo a nuestra **Lista** los elementos que se acaban de crear. De esta manera a la vez que los creamos los pasamos a la lista, así no tenemos que volver a realizar otra pasada por la tabla.

10. Cierra la ventana de código.

*11. Haz doble clic en el botón **Nueva**.*

Como ya hemos dicho, en el momento en el que pulsemos este botón borraremos lo que haya en la **Lista** e introduciremos una nueva tabla.

12. Sube por el código de la aplicación hasta llegar al evento que hemos escrito anteriormente.

Copiar y pegar

13. Selecciona el contenido de dicho evento.

Para seleccionar líneas de código, simplemente debes ponerte en el margen izquierdo de la ventana de código a la altura de la primera línea de código. Seguidamente pulsa el botón izquierdo del ratón y mientras lo tienes pulsado muévete hasta la última la ultima línea de código dentro de este procedimiento. Fíjate como ha cambiado el color del fondo del texto.

Con el texto seleccionado:

*14. Selecciona la opción **Copiar** del menú **Edición**.*

*15. Sitúa el cursor en el interior del evento del botón **Nueva**.*

*16. Selecciona la opción **Pegar** del menú **Edición**.*

El código se ha copiado.

Vamos ha realizar unas pequeñas modificaciones.

17. Selecciona la primera línea de código y bórrala.

*18. Escribe lo siguiente: **Lista.Clear***

Recuerda que esta instrucción sirve para borrar el contenido de la **Lista** para así

poder poner insertar otra lista nueva.

19. Realiza una ejecución de prueba para ver el funcionamiento de los dos eventos programados hasta el momento.

20. Detén la ejecución.

Ordenación

Ahora vamos a dedicarnos a lo que es en sí la ordenación de la **Tabla.**

*21. Haz doble clic sobre el botón **Ordenar**.*

22. Escribe dentro de este evento las siguientes instrucciones.

```
1  Private Sub Ordenar_Click()
2     I = 1
3     Do
4        For J = 1 To Elementos - I
5           If Tabla(J) >= Tabla(J + 1) Then
6              Cambio = Tabla(J)
7              Tabla(J) = Tabla(J + 1)
8              Tabla(J + 1) = Cambio
9           End If
10             Next J
11             I = I + 1
12          Loop Until I > (Elementos - 1)
13          Lista.Clear
14          For Contador = 1 To Elementos
15             Lista.AddItem Tabla(Contador)
16          Next Contador
17       End Sub
```

Los números que aparecen en cada línea no debes copiarlos, los utilizaremos para facilitar la explicación del funcionamiento del código.

En la línea **2** iniciamos una variable llamada **I** a **1**. Esta variable nos controlará, dentro de un bucle que crearemos en líneas consecutivas, las veces que tenemos que recorrer la tabla, para que esté completamente ordenada. El número de veces será: **el número de elementos de la tabla menos 1**.

En la línea **3** escribimos la primera línea de nuestra estructura **Do...Loop** que termina en la **12**. Utilizamos una estructura **Loop Until** ya que deseamos que se repita este bucle **mientras no se cumpla la condición**. Recuerda que en este tipo de estructuras la condición está en la última línea del bucle.

En la línea **4** iniciamos otro bucle, en este caso un **For... Next** ya que nos interesa que se repitan unas serie de instrucciones un número de veces determinado. En esta línea definimos una nueva variable llamada **J** con valor **1**. Esta variable es la encargada de controlar el bucle. Este bucle se repetirá hasta que **J** llegue al valor que se le indica después del **To**. En nuestro caso cada vez que se ejecute este bucle se repetirá hasta un número diferente, ya que deberemos llegar hasta **Elementos – I**. Vamos a explicarlo con un ejemplo: imaginemos una tabla con **5 elementos**, el ordenador la primera vez que entre en el bucle principal la variable **I** tendrá como valor **1**. Entonces el segundo bucle deberá repetirse hasta que **J** llegue a **5** menos el valor de **I** que es **1**, por lo tanto **4**. Así nos aseguramos que al comparar el elemento que nos marca **J** con el siguiente, en la línea **5** no nos pasemos del índice de la tabla produciéndose un desbordamiento.

En la línea **5** realizamos la comparación del elemento de la tabla cuyo índice es

J con el siguiente. Si resulta que **Tabla(J)** es más grande **(>)** que **Tabla(J+1)** realizamos el cambio de los valores, haciendo pasar el valor de **Tabla(J+1)** a **Tabla(J)**. Este cambio lo efectuamos de la siguiente manera en las líneas **6**, **7** y **8**.

En la línea **6** acumulamos el valor de **Tabla(J)** en una nueva variable que utilizaremos de puente entre las dos posiciones de la tabla. A esta variable puente la llamaremos **Cambio**.

En la línea **7** pasamos el contenido de **Tabla(J+1)** a **Tabla(J)** con lo que escribimos encima del valor que esta tenía, pero no importa ya que este valor está copiado en la variable **Cambio**.

En la línea **8** completamos el cambio pasando el contenido de la variable **Cambio** a **Tabla(J+1)**, machacando el valor antiguo que ya está copiado en **Tabla(J)**.

En la línea **12** termina nuestro **Do... Loop Until**.

Cuando se ejecuta la línea **13** la tabla ya debe estar completamente ordenada, con lo que podemos pasar a visualizarla en nuestro **ListBox**.

Para que no se mezclen la tabla ordenada con la desordenada, primero es preferible borrar la lista, para ello utilizamos la instrucción **Lista.Clear**.

Para pasar el contenido de la tabla a la lista utilizamos un nuevo bucle (línea **14** y **16**), utilizando la variable **Contador** que vuelve a tomar como primer valor **1** y como último el número de elementos de la tabla definido por la constante definida en un principio.

Al pasar por la línea **15** el contenido de la tabla en la posición que nos indica el **contador** pasa a añadirse a la lista. De esta forma volvemos a ver el contenido de la lista que en este caso ya está completamente ordenada.

23. Realiza una ejecución de prueba.

24. Observa los valores de la lista.

*25. Pulsa en el botón **Ordenar**.*

Observa como en un breve espacio de tiempo vuelve a aparecer los mismos valores, pero esta vez completamente ordenados.

*26. Pulsa en el botón **Nueva**.*

27. Vuelve a ordenar la lista.

28. Detén la ejecución del programa.

29. Vuelve al modo diseño.

*30. Accede a la línea en la que se le asigna un valor a la constante **Elementos**.*

*31. Modifica este valor, poniendo **5**.*

32. Realiza otra ejecución de prueba.

Observa como en esta ocasión solo aparecen **5** elementos en la lista. Esto a sido gracias a que en todo nuestro código utilizábamos una constante que controlaba el número de elementos que deseábamos que aparecieran en nuestra tabla. Mira cuantas líneas de código hubiésemos tenido que cambiar si no hubiésemos utilizado una constante.

33. Detén la ejecución.

34. Graba el formulario y el proyecto.

Antes de seguir adelante vamos a aprovechar el ejemplo que acabamos de realizar para explicar un elemento que podemos utilizar en muchas ocasiones el cual nos facilitará un poco la tarea de programar.

Procedimientos

Si observamos el ejemplo que acabamos de realizar podremos observar como hay unas cuantas líneas que se repiten en dos eventos diferentes.

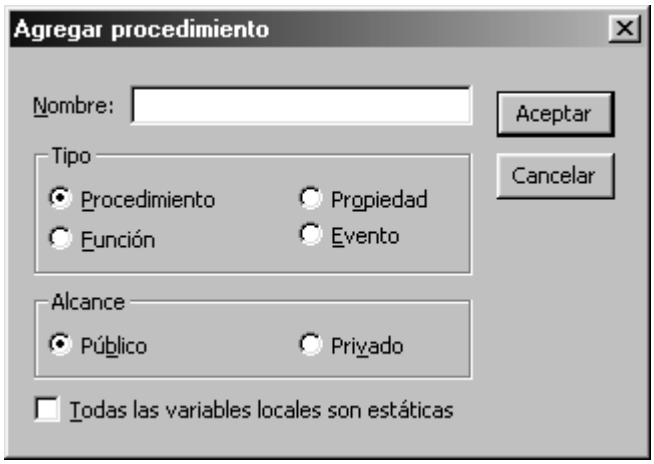
```
For Contador = 1 To Elementos
    Tabla(Contador) = Int((9 * Rnd) + 1)
    Lista.AddItem Tabla(Contador)
Next Contador
```

Estas líneas están dentro de los eventos: **Form_Load()** y **Nueva_Click()**.

En esta ocasión no ocurre no es importante que estas líneas se repitan ya que son sólo 4. Pero podemos tener otras aplicaciones en las que el número de líneas que se repitan puedan ser muchas más, con lo que el número total de líneas de código se vería incrementado haciendo más difícil la localización de un posible error.

Vamos a ver una forma de poder compartir estas líneas de código y utilizarlas en el momento en el que deseemos.

*35. Selecciona la opción: **Agregar procedimiento** dentro de la opción **Herramientas**.*



Seguidamente te aparecerá una ventana como la siguiente:

De esta nueva ventana vamos a explicar las opciones que nos interesa.

Dentro del apartado **Alcance** tenemos dos posibles opciones: **Público** o **Privado**. La primera de ellas se utilizaría en el momento que deseamos crear un procedimiento que se pueda mirar desde cualquier formulario que tuviera una aplicación. Mientras que la segunda opción la utilizaríamos en el momento en el que queremos que el procedimiento sólo pueda ser consultado por el formulario en el que nos encontramos.

Dentro del apartado **Tipo** tenemos 4 opciones de las cuales sólo nos interesan

2: **Procedimiento** y **Función**. Vamos a ver que son cada una de estas opciones.

Procedimiento o Sub:

Un procedimiento ejecuta una tarea específica dentro de un programa sin devolver ningún valor.

Función o Function:

Una función ejecuta una tarea específica dentro de un programa, pero nos devuelve un valor.

En nuestra aplicación lo que nos interesa es un **Procedimiento (Sub)** ya que lo que deseamos es que se realicen una serie de instrucciones, pero no necesitamos que se nos devuelva ningún valor concreto.

*36. Dentro de la ventana **Agregar procedimiento** escribe **Crear** en el apartado **Nombre**.*

*37. Deja seleccionada la opción **Procedimiento** y escoge la opción **Privado** dentro del apartado **Alcance**.*

Observa como dentro del código han aparecido estas líneas de código:

```
Private Sub Crear()

End Sub
```

Podrás ver que son muy parecidas a las líneas de código de los eventos de los diferentes elementos.

Dentro de estas dos nuevas líneas de código vamos a escribir el código que se repite dentro de nuestra aplicación.

```
Private Sub Crear()
   For Contador = 1 To Elementos
      Tabla(Contador) = Int((9 * Rnd) + 1)
      Lista.AddItem Tabla(Contador)
   Next Contador
End Sub
```

Ahora ya estamos preparados para hacer que estas líneas se ejecuten en el momento en el que nosotros deseemos.

Para llamar a este procedimiento simplemente deberemos poner el nombre de este en el punto de la aplicación que deseemos. Por ejemplo vamos a ver como lo haríamos dentro del evento: **Form_Load()**

Anteriormente este evento estaba creado de la siguiente forma:

```
Private Sub Form_Load()
   Randomize
   For Contador = 1 To Elementos
      Tabla(Contador) = Int((9 * Rnd) + 1)
      Lista.AddItem Tabla(Contador)
   Next Contador
End Sub
```

Si miramos las líneas que aparecen dentro del procedimiento que hemos creado

anteriormente podremos ver donde deberemos hacer la llamada a **Crear**.

```
Private Sub Form_Load()
   Randomize
   Crear
End Sub
```

Concatenación de texto

En este apartado vamos a ver una operador que nos permite concatenar elementos. La concatenación es la unión de dos o más elementos que están separados, para formar uno nuevo.

Vamos a ver este operador mediante un ejemplo muy simple.

. Practica 4

1. Crea un nuevo formulario.

*2. Inserta dos **TextBox**. Elimina el contenido y deja los nombres que tienen por defecto.*

*3. Inserta un **Label**. Elimina el contenido y ponle como nombre: **Union**.*

*4. Inserta un **CommandButton**. Pon el texto que quieras. No hace falta que cambies su nombre.*

Con este pequeño ejemplo queremos que al pulsar el botón, aparezca en el **Label** de nuestro formulario la concatenación del contenido de los dos **TextBox**.

*5. Pulsa doble clic en el **botón**.*

6. Escribe las siguientes líneas de código:

```
Private Sub Command1_Click()
   Union.Caption = Text1.Text & Text2.Text
End Sub
```

Recuerda que siempre que se realiza una asignación, pasa el contenido de la derecha del igual a la izquierda de este.

Para unir el contenido de los dos objetos insertados utilizamos el operador **&**.

7. Inicia una ejecución de prueba.

*8. Escribe lo que quieras dentro de los dos **TextBox**.*

*9. Pulsa el **Botón**.*

Observa como el contenido de los dos **TextBox** pasa dentro del **Label**.

*10. Cambia el contenido de los dos **TextBox**.*

*11. Pulsa nuevamente el **Botón** que hemos insertado en el formulario.*

Nuevamente vuelven ha aparecer los dos **TextBox** unidos dentro de nuestro **Label**.

¿Qué tendríamos que hacer para que entre ellos apareciera un espacio en blanco?

La respuesta es muy sencilla, tendríamos que concatenar entre ellos un espacio en blanco.

12. Modifica la línea de código de la siguiente manera:

```
Private Sub Command1_Click()
   Union.Caption = Text1.Text & " " & Text2.Text
End Sub
```

Observa detenidamente la línea donde se produce la concatenación de los diferentes objetos. Primero concatenamos el contenido de **Text1**, después un espacio en blanco **" "** y por último el contenido de **Text2**.

13. Realiza una ejecución de prueba.

14. Detén la ejecución una vez hayas introducido diferentes textos en las correspondientes casillas.

Si nosotros quisiéramos concatenar las casilla de texto separadas entre sí mediante una conjunción **y** deberíamos hacerlo de la siguiente manera:

15. Modifica las líneas de código para que queden así:

```
Private Sub Command1_Click()
   Union.Caption = Text1.Text & " y " & Text2.Text
End Sub
```

Observa que el texto que delante y detrás del texto que deseamos aparezca entre los elementos a concatenar hemos dejado un espacio, esto lo hacemos para que no salga junto a los elementos concatenados y la conjunción **y**. Observa igualmente que este texto está entre comillas.

16. Realiza una ejecución de prueba.

17. Detén la ejecución.

18. Modifica el código de nuestro ejemplo para cambiar el orden de la concatenación, escribiendo un punto posterior, el texto concatenado entre comillas, etc. Realiza todos los cambios que se te ocurran

*19. Sal de **Visual Basic** sin guardar los cambios.*

Recomendamos repasar con profundidad las **estructuras de decisión** (lección anterior) y las **estructuras de repetición** (lección actual).

Fin de la lección 6

[1] Variable en la que podemos almacenar cualquier tipo de dato.