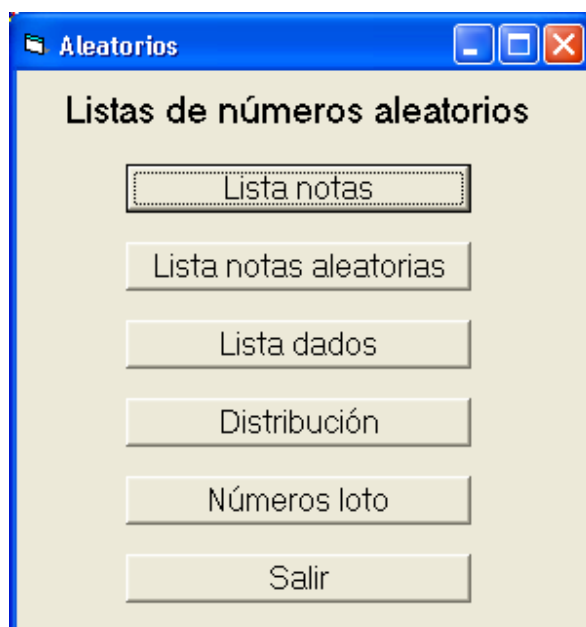


**Objetivos:**

- ❖ Adquirir habilidades en el uso de **vectores** (arrays de una dimensión)
- ❖ Conocer la generación de números pseudo-aleatorios
- ❖ Límites inferior y superior de un vector en VB: **LBound** y **UBound**
- ❖ Inicialización de vectores mediante la orden **Array**

## Programa de números aleatorios

### Interfaz



**Figura 11.1.** Interfaz del programa de números aleatorios.

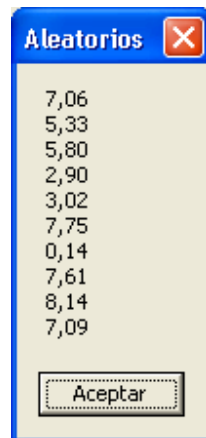
Vamos a resolver parte de este ejercicio para demostrar el uso de **variables pseudo-aleatorias** en Visual Basic.

La función **Rnd** nos devuelve un número real pseudo-aleatorio de **distribución uniforme** en el intervalo  $[0, 1)$  (del 0 al 1, éste excluido). Lo que queremos expresar con distribución uniforme es que todo número tiene las mismas posibilidades de salir que cualquier otro, como sucede cuando lanzamos una moneda no trucada: el número de caras tenderá a ser igual al número de cruces.

Se llaman variables pseudo-aleatorias porque no son realmente aleatorias: dentro de la serie, a un número dado siempre le sigue el mismo número; partiendo del mismo número siempre obtendremos la misma serie. El primer número de la serie se llama **semilla**.

### Botón “Lista notas”

El primer botón que vamos a describir nos muestra el escenario de un profesor arbitrario que utiliza un programa para generar 10 notas aleatorias del 0 al 10 (aunque nunca pone un 10). Se comprobará que si se pulsa este botón nada más ejecutar el programa siempre obtendremos las mismas notas, que son las que se muestran en la Figura 11.2.



**Figura 11.2.** Lista de notas nada más ejecutar el programa.

El hecho de que siempre salga la misma serie de números (ya veremos la codificación más adelante) se debe al hecho de que, si no se le indica lo contrario, Visual Basic siempre da el mismo valor inicial a la semilla. Si utilizamos un subprograma `IniVectorNotas` para inicializar el vector a una lista de valores aleatorios y el subprograma `SacaVectorDbl` para mostrar todo el vector mediante un `MsgBox` la acción del botón es relativamente sencilla:

```
Sub cmd1_Click()  
    Dim v(1 To 10) As Double  
    Call IniVectorNotas(v)  
    Call SacaVectorDbl(v)  
End Sub
```

El procedimiento `IniVectorNotas` hará uso de los límites inferior (**LBound**) y superior (**UBound**) para rellenar todo el vector de valores en el intervalo  $[0, 10)$ , multiplicando por 10 los valores obtenidos en el intervalo  $[0, 1)$ . Obsérvese cómo se pasa el vector como parámetro. Podríamos haber omitido la palabra clave **ByRef** ya que es el valor por defecto.

```
Sub IniVectorNotas(ByRef v() As Double)  
    Dim i As Integer  
    For i = LBound(v) To UBound(v) Step 1  
        v(i) = Rnd * 10  
    Next i  
End Sub
```

El procedimiento `SacaVectorDbl` obtendrá una cadena con la lista de valores de un vector y luego mostrará el resultado mediante un **MsgBox**.

Para que todos los valores salgan alineados utilizaremos la función **Format** que ya vimos en el laboratorio anterior.

```
Sub SacaVectorDbl(ByRef v() As Double)
    Dim i As Integer
    Dim s As String
    s = ""
    For i = LBound(v) To UBound(v) Step 1
        s = s & Format(v(i), "0.00") & vbCrLf
    Next i
    MsgBox s
End Sub
```

### Botón “Lista notas aleatorias”

Este botón se distingue del anterior tan solo en el hecho de que no va a mostrarnos siempre la misma lista de números aleatorios ya que vamos a utilizar la instrucción **Randomize** que modificará la semilla (primer número de la serie) cada vez que se le llama utilizando el reloj del sistema.

Podemos reutilizar todo lo visto para el botón anterior (no será preciso redefinir los subprogramas `IniVectorNotas` y `SacaVectorDbl`).

El código será (en gris la parte que no cambia):

```
Sub cmd2_Click()
    Dim v(1 To 10) As Double
    Randomize
    Call IniVectorNotas(v)
    Call SacaVectorDbl(v)
End Sub
```

### Botón “Lista dados”

Este botón va a sacar una lista de números del 1 al 6. Para hacer esto a partir de una variable aleatoria que devuelve números necesitaremos una función que los genere manteniendo una distribución uniforme<sup>1</sup>.

Esta función se da como receta:

```
Function Aleatorio(ByVal min As Long, ByVal max As Long) As Long
    Aleatorio = Int((max - min + 1) * Rnd) + min
End Function
```

Con esto el código del botón que muestra la lista de dados será la siguiente:

<sup>1</sup> Para ilustrar esto imaginemos que queremos obtener números del 1 al 4 a partir de números del 1 al 6 obtenidos mediante el lanzamiento de un dado. Si calculamos un número mediante el resto de dividir el número obtenido con el dado por 4 más 1 obtendríamos sólo números del 1 al 4 pero los números del 1 y 2 serían el doble de frecuentes que los números 3 y 4 ya que recibirían las ocurrencias propias y las del 5 y 6 respectivamente.

```

Sub cmd3_Click()
    Dim v(1 To 10) As Integer
    Randomize
    Call IniVectorDados(v)
    Call SacarVectorInt(v)
End Sub

```

Hemos utilizado dos procedimientos nuevos: `IniVectorDados` y `SacarVectorInt`. El primero es similar al de rellenar el vector de reales con notas, sólo que ahora lo rellenaremos con números del 1 al 6 mediante una llamada a la función `Aleatorio`.

```

Sub IniVectorDados(ByRef v() As Integer)
    Dim i As Integer
    For i = LBound(v) To UBound(v) Step 1
        v(i) = Aleatorio(1, 6)
    Next i
End Sub

```

El procedimiento `SacarVectorInt` es similar al ya visto para reales.

### Botón “Distribución”

Este botón mide la “calidad” del generador de números aleatorios. Declara un vector de 600 enteros y lo rellena con números aleatorios del 1 al 6 como en el ejercicio anterior (“lanza” 600 veces el dado). A continuación contará cuántas veces ha aparecido el 1, cuántas el 2 y así sucesivamente. Con una distribución uniforme pura cada uno de los números debería aparecer la misma cantidad de veces.

### Botón “Números loto”

Este botón genera 6 números distintos del 1 al 49 para rellenar el boleto de la lotería.

Para ello utiliza una función `Presente` que nos dice si un número `num` se encuentra en un vector `v` suponiendo que hay `n` elementos válidos.

La función tiene la siguiente cabecera:

```

Function Presente(ByVal num As Integer, ByRef v() As Integer,
                 ByVal n As Integer) As Boolean

```

Es de resaltar que en esta función el vector se pasa por referencia pero es un parámetro de entrada, es decir, no se modifica.

Alternativamente se puede utilizar una función `PosEnVector` que devuelve la posición en que se encuentra el número `num` en el vector `v` suponiendo que hay `n` elementos válidos y que el índice del primer elemento del vector es el 1. Si `num` no se encuentra en el vector nos devolverá un 0.

## Comprobación de cuenta con un vector de coeficientes (resuelto)

### Interfaz

Figura 11.3 Comprobador de cuenta bancaria

### Funcionamiento

En el laboratorio anterior se propuso un ejercicio para la comprobación de una cuenta bancaria (ejercicio 10.2). En este laboratorio se propone, a modo de demostración, una solución para el cálculo de los dígitos de comprobación mediante la utilización de un vector de coeficientes (ver tabla 10.3 del laboratorio anterior).

### Código propuesto

1. El código asociado al botón será similar aunque ahora definiremos un vector de coeficientes `k()`.
2. La definición de la variable será como Variant.

```
Dim k() As Variant
```

3. Si queremos dar un conjunto de valores a los elementos del vector `k` lo hacemos mediante la instrucción **Array** de VB.

```
k = Array(4, 8, 5, 10, 9, 7, 3, 6)
```

4. Definido el vector de coeficientes podemos realizar el cálculo del dígito de control en una función `CalDigCtrl`. Utilizando esta función el código del botón **Comprobar** resultante puede ser (se han simplificado los controles de corrección mediante puntos suspensivos):

```
Sub cmdComprobar_Click()  
  Dim d1 As Integer, d2 As Integer  
  Dim k() As Variant  
  ...  
  k = Array(4, 8, 5, 10, 9, 7, 3, 6)  
  d1 = CalDigCtrl(txtEnt.Text & txtOfi.Text, k)  
  k = Array(1, 2, 4, 8, 5, 10, 9, 7, 3, 6)  
  d2 = CalDigCtrl(txtCta.Text, k)  
  If txtCtr.Text = d1 & d2 Then
```

```

    MsgBox "Cuenta bancaria correcta"
    ...
End Sub

```

5. La función de cálculo CalDigCtrl hará uso de los límites inferior (**LBound**) y superior (**UBound**) del vector. Nótese que esto se hace para simplificar el paso de parámetros a la función.

```

Function CalDigCtrl(ByVal str As String, ByRef k() As Variant) As Integer
    Dim d As Integer, i As Integer
    d = 0
    For i = LBound(k) To UBound(k) Step 1
        d = d + Val(Mid(str, i + 1, 1)) * k(i)
    Next i
    d = 11 - d Mod 11
    If d > 9 Then
        d = 11 - d
    End If
    CalDigCtrl = d
End Function

```

|                                    |   |
|------------------------------------|---|
| <b>Rnd</b>                         | Número pseudo-aleatorio de distribución uniforme en [0, 1)            |
| <b>Randomize</b>                   | Inicializar la semilla de números aleatorios con el reloj del sistema |
| <b>LBound</b> (v()) <u>As Long</u> | Obtener el subíndice inferior de un vector                            |
| <b>UBound</b> (v()) <u>As Long</u> | Obtener el subíndice superior de un vector                            |

**Tabla 11.1.** Lista de funciones relevantes de Visual Basic