UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MANUAL BÁSICO PARA EMPEZAR A TRABAJAR CON MACROS DE VISUAL BASIC PARA EXCEL

Introducción al Visual Basic

Índice

1. INTRODUCCION (a propósito de Visual Basic)

Visual Basic para aplicaciones es una combinación de un entorno de programación integrado denominado ***Editor de Visual Basic*** y del lenguaje de programación Visual Basic, permitiendo diseñar y desarrollar con facilidad programas en Visual Basic. El término "para aplicaciones" hace referencia al hecho de que el lenguaje de programación y las herramientas de desarrollo están integrados con las aplicaciones del ***Microsoft Office*** (en este caso, el ***Microsoft Excel***), de forma que se puedan desarrollar nuevas funcionalidades y soluciones a medida, con el uso de estas aplicaciones.

El ***Editor de Visual Basic*** contiene todas las herramientas de programación necesarias para escribir código en Visual Basic y crear soluciones personalizadas.

Este Editor, es una ventana independiente de ***Microsoft Excel***, pero tiene el mismo aspecto que cualquier otra ventana de una aplicación ***Microsoft Office***, y funciona igual para todas estas aplicaciones. Cuando se cierre la aplicación, consecuentemente también se cerrará la ventana del ***Editor de Visual Basic*** asociada.

Este manual ha sido elaborado por José Pedro García Sabater con la colaboración de Gonçal Bravo i Reig y Alberto López Gozalbes a lo largo de diversas versiones de la hoja de cálculo ***Microsoft Excel***. Es posible que a lo largo del mismo se hallen algunas inexactitudes ligadas entre otras razones a la evolución de Excel. Si encuentran errores sería estupendo que nos lo hicieran saber para así corregirlos.

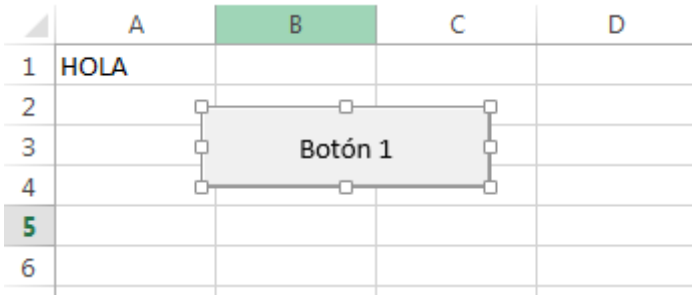
2. OBJETIVOS

El documento está inicialmente dirigido a alumnos de ingeniería que con mínimos conocimientos de programación pueden entender cómo funciona el VBA de Excel.
No se pretende enseñar a programar, sólo a utilizar el entorno y a sacar partido al mínimo conocimiento en programación que tienen mis alumnos de ingeniería.
Lo que se pretende con este manual es presentar de una manera práctica, diferentes utilidades, funciones, sentencias…, en el ***Editor de Visual Basic***, y que con posterioridad serán útiles para el desarrollo del ejercicio concreto de que consta la práctica.
Los ejemplos son sencillos e incluso un poco tontos, y desde luego inútiles en sí mismos.
En cada ejemplo se presentan una o varias funcionalidades.
Tomando ejemplos sencillos, se irán mostrando sucesivamente las diferentes utilidades a realizar o utilizar. Son utilidades básicas cómo definir un botón de ejecución de programa, cómo dar valores a celdas de la página de ***Microsoft Excel*** (mediante un programa definido en el ***Editor de Visual Basic***), cómo definir e introducir bucles y condiciones,…

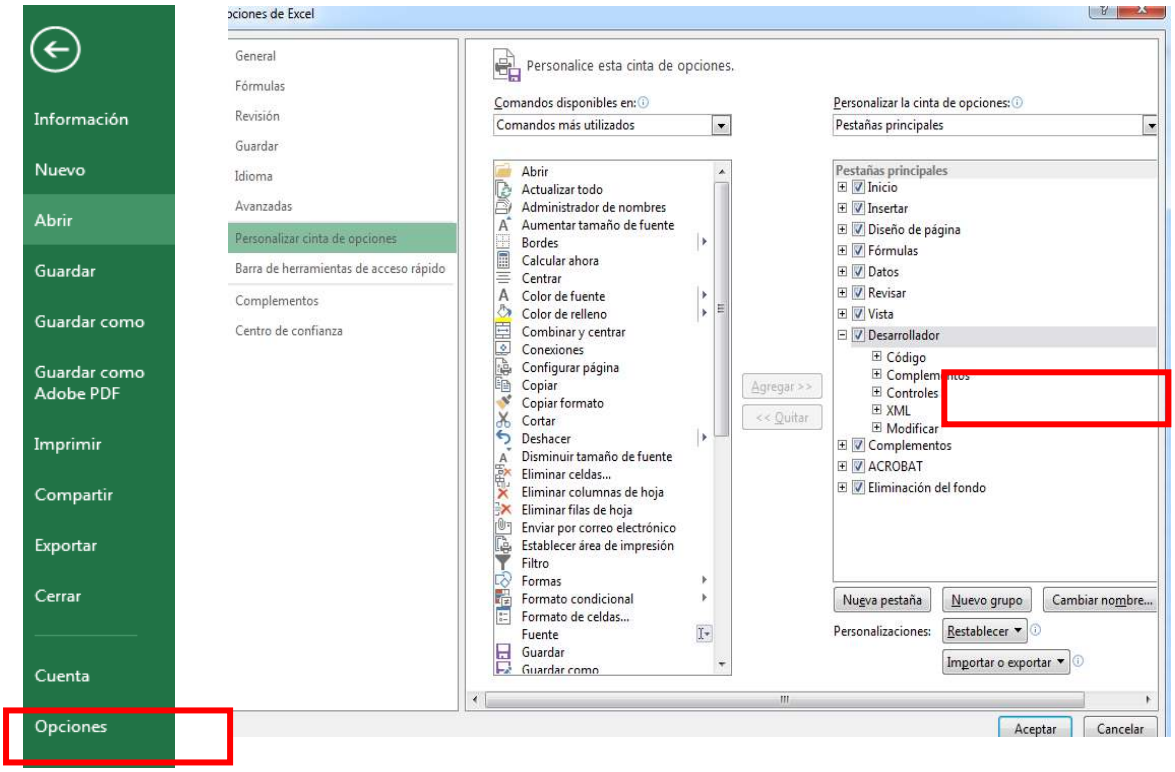
3. DESARROLLO DE LOS EJEMPLOS

3.1. Creación de un "botón" que al apretarlo escriba HOLA.

Vamos a crear un botón, que al hacer clic sobre él, muestre en la celda A1 la expresión "HOLA".

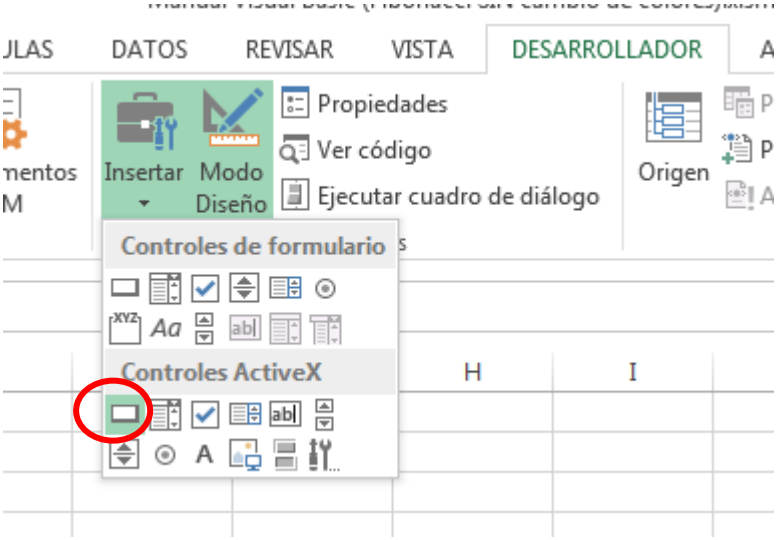


Para ello, en primer lugar, se instalará en el documento de ***Microsoft Excel***, el menú ***Programador*** (Menú ***Archivo*** -> ***Opciones -> Personalizar cinta de opciones*** *y se selecciona la casilla* ***Programador)***.



Una vez hecho esto, aparecerá la pestaña **Desarrollador** desde la que se pueden añadir los botones dentro de la pestaña.

En él se tomará el icono que representa a un botón, desplegándose en la Hoja1, por ejemplo, del documento Excel. De los dos botones que hay (tanto en ***formularios*** como en ***ActiveX***), se seleccionará el de ***Controles de ActiveX***, ya que de este modo se podrá cambiar el color y otras opciones del propio botón.



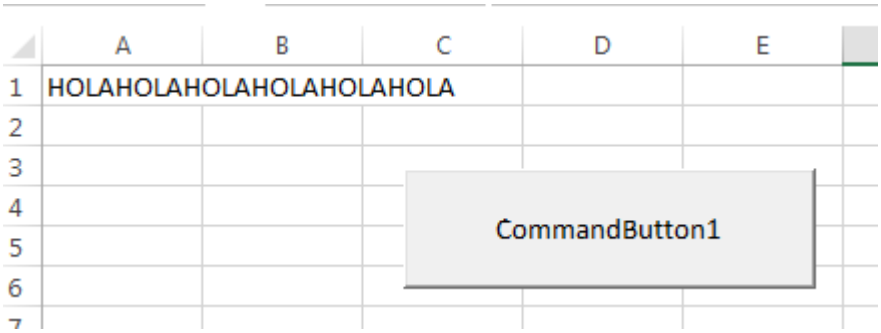
Una vez hecho esto, se pulsará dos veces sobre dicho botón para acceder así al ***Editor de Visual Basic***, con el que se realizará el pequeño programa requerido, tal y como sigue:

```
CommandButton1
Private Sub CommandButton1_Click()
Hoja1.Cells(1, 1) = "HOLA"

End Sub
```

3.2. Acumulación de "HOLA"'s en la misma celda.

Ahora vamos a cambiar el programa anterior, cambiando una de las líneas de programa, para hacer que cada vez que se haga un clic en el botón, se acumule un nuevo "HOLA" (igual que podría ser cualquier otro valor numérico o cadena de caracteres) al anterior. De esta forma, se identificará el contenido de la primera celda como un contador, acumulándose, en cada clic sobre el botón, una nueva cadena de texto en dicha celda contador.

```
CommandButton1
Private Sub CommandButton1_Click()

    Hoja1.Cells(1, 1) = Hoja1.Cells(1, 1) + "HOLA"

End Sub
```



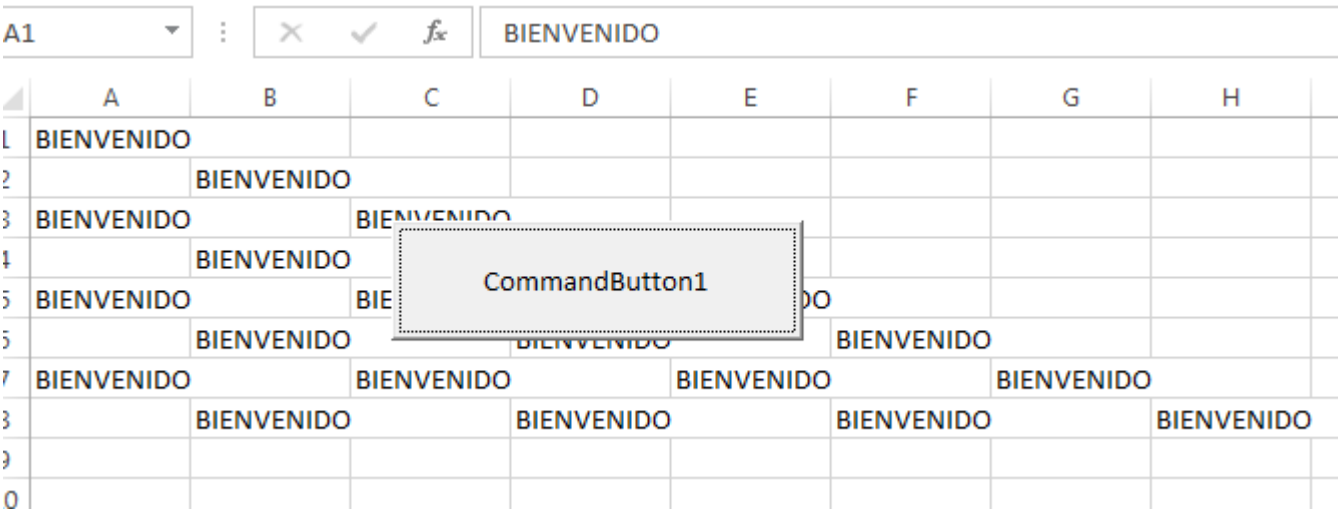
3.3. Acumulación de texto en varias diagonales sucesivas.

Continuando el ejemplo anterior, vamos a definir una lista en varias diagonales, en las que se mostrará el texto previamente definido ("BIENVENIDO"). En la nueva versión del programa anterior, se podrá observar cómo utilizar la función "condición" (representada por la función ***if***) y el bucle (mediante la aplicación de la función ***for***, entre otras opciones).

Así, para hacer que la palabra "BIENVENIDO" aparezca colocada siguiendo varias diagonales un número determinado de veces. Se definen, inicialmente, dos variables contador como enteros (función ***Dim… As Integer***), y que representan además los índices de las celdas de la Hoja de Cálculo (filas y columnas). Se define el texto en la primera celda. Seguidamente, se define la condición de que la suma de los índices de celda (variables contadores) sean números pares, con la utilización de la función ***mod*** (función resto, dividiendo el número requerido por dos, si el resto es 0, el número es par), así se tendrían definidas las diferentes diagonales. Esta "condición" estaría colocada dentro de un doble bucle ***for*** (bucle anidado), en el que el valor de cada nueva celda de la diagonal, tendrá el mismo valor que la anterior.

CommandButton1

```
Private Sub CommandButton1_Click()

Dim i As Integer
Dim j As Integer

Hoja1.Cells(1, 1) = "BIENVENIDO"
    For i = 2 To 8
        For j = 1 To i
            If ((i + j) Mod 2) = 0 Then
                Hoja1.Cells(i, j) = Hoja1.Cells(1, 1)
            End If
        Next j
    Next i

End Sub
```



3.4. Jugando con las series de Fibonacci.

En este caso, vamos a desarrollar código que cumplirá las siguientes características:
Utilización de una serie de Fibonacci de números aleatorios.

- Se tomarán exclusivamente la cifra de unidades de los números de la serie anterior.
- Se ordenarán estos valores de mayor a menor (para poder trabajar con ellos).
- Se mostrará cómo realizar el diagrama de barras correspondiente a la serie anterior (cada barra con el tamaño y el color correspondiente al número de la serie).

Y en él, se utilizarán además las funciones y opciones del ***Editor de Visual Basic / Microsoft Office*** siguientes:

- Cambio de nombre de un botón.
- Utilización y grabación de macros.
- Utilización de la función ***Call*** para llamar a una función definida en otro lugar.
- Cambio de color.

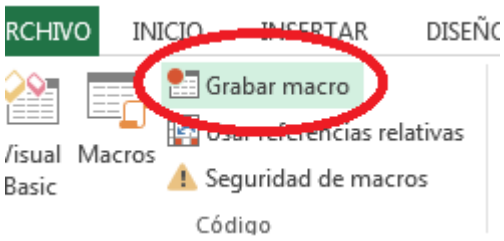
La serie de Fibonacci cumple que cada elemento de la serie es el resultado de la suma de los dos precedentes, es decir: **an+2 = an+1 + an**
Así, se introducirá la fórmula anterior mediante la utilización de un bucle ***Do While…Loop*** (una de las opciones posibles), previa definición de los dos valores iniciales. De esta forma, se van a definir estos valores iniciales como aleatorios; para ello, se va a utilizar la función de generación de números aleatorios ***rnd*** (tal y como se ve en el programa). Se evitan números excesivamente grandes o en coma flotante, tomando la variable como ***int***, para evitar la aparición de decimales. Además, se ve cómo se utiliza la función ***With***, para definir la selección de color. Este código se ha tomado del de la macro grabada a partir del cambio de color de una celda cualquiera (mediante la utilización de la opción del menú ***Cambio de color***).

CommandButton1 | Click

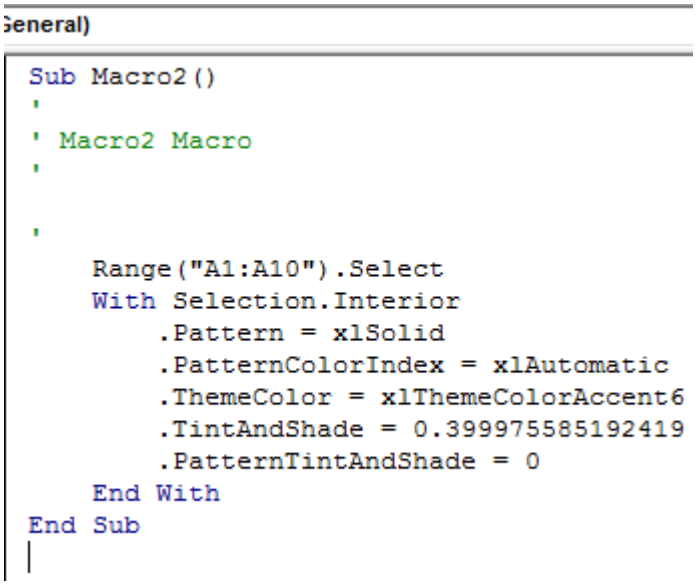
```
Private Sub CommandButton1_Click()

Hoja1.Cells(1, 2) = Int(Rnd() * 10)
Hoja1.Cells(2, 2) = Int(Rnd() * 10)
i = 1

Do While i < 21
    Hoja1.Cells(i + 2, 2) = Hoja1.Cells(i + 1, 2) + Hoja1.Cells(i, 2)
    Hoja1.Cells(i, 2).Select
    With Selection.Interior
        .ColorIndex = Int(Rnd() * 10)
        .Pattern = xlSolid
    End With
    i = i + 1
    If (i = 21) Then
        Hoja1.Cells(21, 2).Select
        With Selection.Interior
            .ColorIndex = Int(Rnd() * 10)
            .Pattern = xlSolid
        End With

        Hoja1.Cells(22, 2).Select
        With Selection.Interior
            .ColorIndex = Int(Rnd() * 10)
            .Pattern = xlSolid
        End With
    End If
Loop

End Sub
```

Pero, ¿qué es una macro?, y ¿cómo se graba una macro?

En primer lugar, se debería considerar que una macro es un pequeño programa ejecutable desde la Hoja de Cálculo, y que realiza funciones repetitivas o comunes en la normal ejecución de la actividad con la herramienta de cálculo. Así, y en el caso particular de grabar una macro para poder cambiar de color una serie de celdas de la Hoja de Cálculo, se procede de la siguiente forma. En el menú, se toma la opción ***Desarrollador***, y en ésta, ***Grabar macro***. Acto seguido, se realiza la acción a grabar en la macro, en este caso, cambiar de color el color de una columna de la hoja de cálculo.

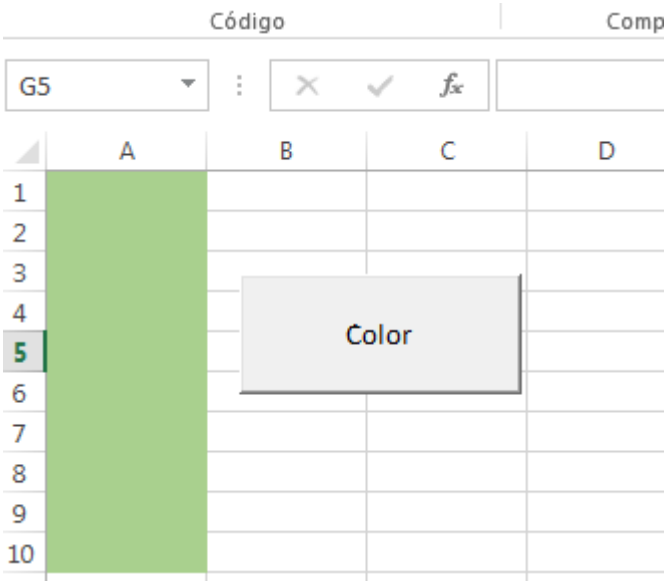


Abriendo la opción de Visual Basic, la macro grabada quedaría reflejada de la siguiente manera:

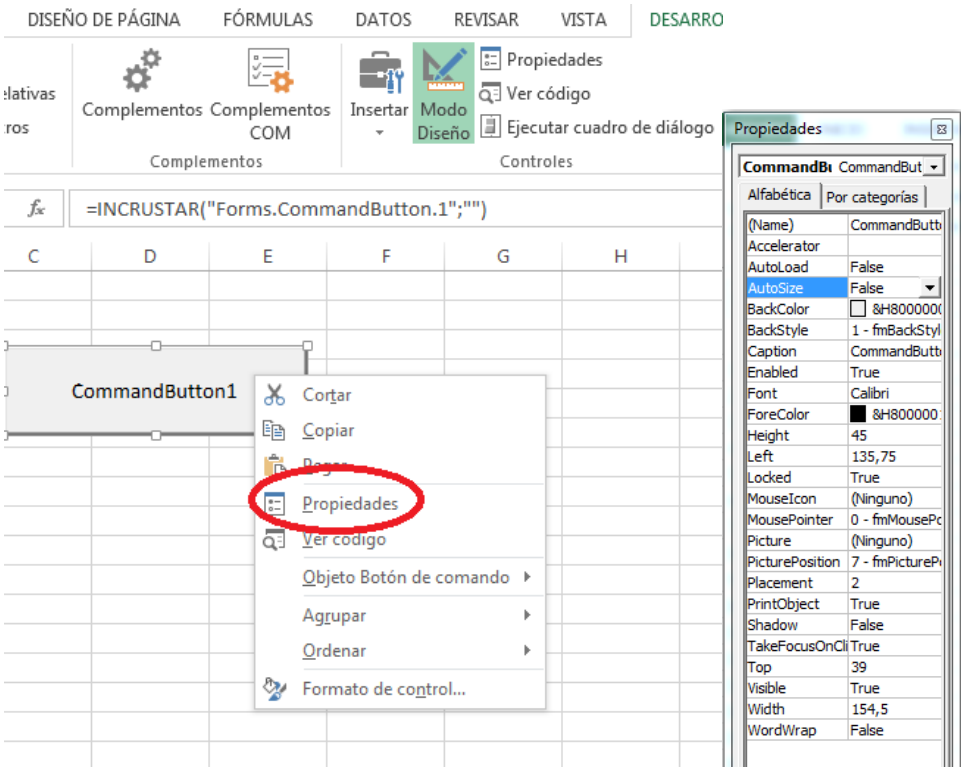


En el paso anterior se ve, en el código definido por la macro, la opción ***Range***; esto define el rango de aplicabilidad de la opción escogida con el código, en ese caso el cambio de color de las celdas A1 hasta la A10.

Además se le puede cambiar el nombre al botón para que deje de "llamarse" ***CommandButton1*** y así poder ponerle el nombre deseado y cambiar otras propiedades como el color del botón. Pero, ¿cómo se consigue cambiar el nombre al botón?



Para ello, se selecciona el Modo Diseño del cuadro de controles de la pestaña Desarrollador, una vez ahí, se haría clic con el botón derecho del ratón, sobre el botón al que se le quiere cambiar el nombre. Acto seguido, se selecciona la opción Propiedades y dentro de estas se cambia la opción ***Caption***.



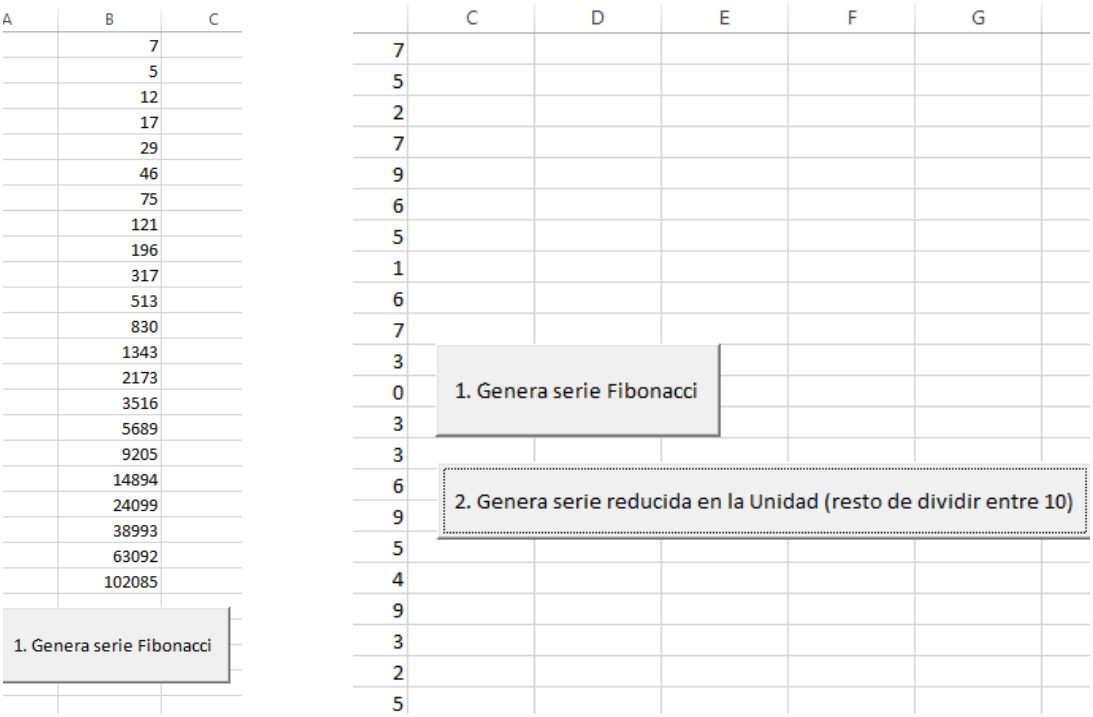
Una vez mostradas las acciones anteriores, se va a pasar a definir el ejemplo concreto. Así, y como ya habíamos dicho, vamos a definir el código de programa necesario para por un lado generar la serie de Fibonacci de términos aleatorios, y por el otro, tomar de los valores de la serie anterior exclusivamente las cifras correspondientes a las unidades.

CommandButton2 | Click

```
Private Sub CommandButton1_Click()

Hoja1.Cells(1, 2) = Int(Rnd() * 10)
Hoja1.Cells(2, 2) = Int(Rnd() * 10)
i = 1
'Generación de la serie de Fibonacci

Do While i < 21
    Hoja1.Cells(i + 2, 2) = Hoja1.Cells(i + 1, 2) + Hoja1.Cells(i, 2)
    Hoja1.Cells(i, 2).Select
    i = i + 1
Loop

End Sub

Private Sub CommandButton2_Click()
i = 1
'Generación de la serie anterior reducida a la unidad

Do While i < 23
    Hoja1.Cells(i, 2) = Hoja1.Cells(i, 2) Mod 10
    i = i + 1
Loop

End Sub
```

Aquí pueden observarse dos bloques diferenciados de programa, cada uno para un botón diferente (que se pueden ver en la transparencia siguiente). En el primero se crea una serie de

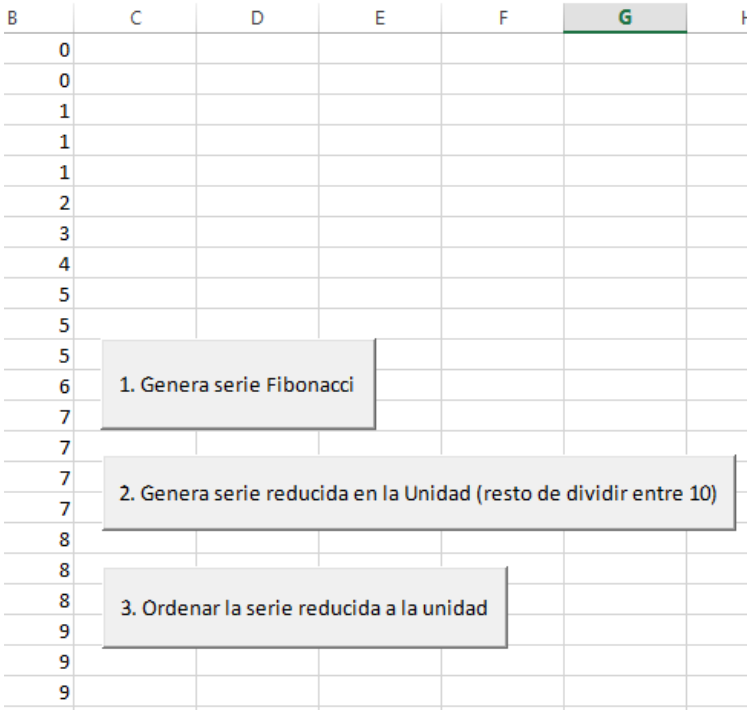
Fibonacci, tal y como ya se ha explicado, y acto seguido, se reduce cada uno de los números de dicha serie a su cifra de unidades. Esta sería el resto obtenido de dividir dicho número de la serie original, por 10.
Esto se consigue con la utilización de la función ***mod***. Todo ello dentro de su correspondiente bucle para ir tomando todos los valores de la serie.



Se ve el resultado obtenido. Primero, haciendo clic en el primer botón, se obtendría la serie, y seguidamente, haciendo clic sobre el segundo botón, se obtiene la cifra correspondiente a la cifra de unidades de la serie de Fibonacci anterior.

Ahora, se deberá definir una función que tome una serie de números y los ordene de mayor a menor. Esto se haría mediante la grabación de una macro llamada ordenar, en la que se graba la acción de **Ordenar** (función perteneciente al menú datos de la barra de menú) de mayor a menor los valores de la primera columna, se obtiene el código de programa necesario para implementar un tercer botón, por ejemplo (código que se ve abajo).

```
Private Sub CommandButton3_Click()

'Ordenar la serie

    Range("B1:B22").Select
    ActiveWorkbook.Worksheets("Hoja 1").Sort.SortFields.Clear
    ActiveWorkbook.Worksheets("Hoja 1").Sort.SortFields.Add Key:=Range("B1:B22") _
        , SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
    With ActiveWorkbook.Worksheets("Hoja 1").Sort
        .SetRange Range("B1:B22")
        .Header = xlGuess
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With
End Sub
```



Clicando en el tercer botón se obtiene la serie numérica resultante de ordenar la serie de cifras unidad de la serie de Fibonacci (de la transparencia anterior). Si la macro se hubiera grabado en sentido descendente (del número 9 al 1), sólo habría que grabar la macro cambiando el orden por descendente, o bien modificar el código de manera que apareciera la palabra ***Descending*** en lugar de ***Ascending***.

```
Range("B1:B22").Select
ActiveWorkbook.Worksheets("Hoja 1").Sort.SortFields.Clear
ActiveWorkbook.Worksheets("Hoja 1").Sort.SortFields.Add Key:=Range("B1:B22") _
    , SortOn:=xlSortOnValues, Order:=xlDescending, DataOption:=xlSortNormal
With ActiveWorkbook.Worksheets("Hoja 1").Sort
    .SetRange Range("B1:B22")
    .Header = xlGuess
    .MatchCase = False
    .Orientation = xlTopToBottom
    .SortMethod = xlPinYin
    .Apply
```

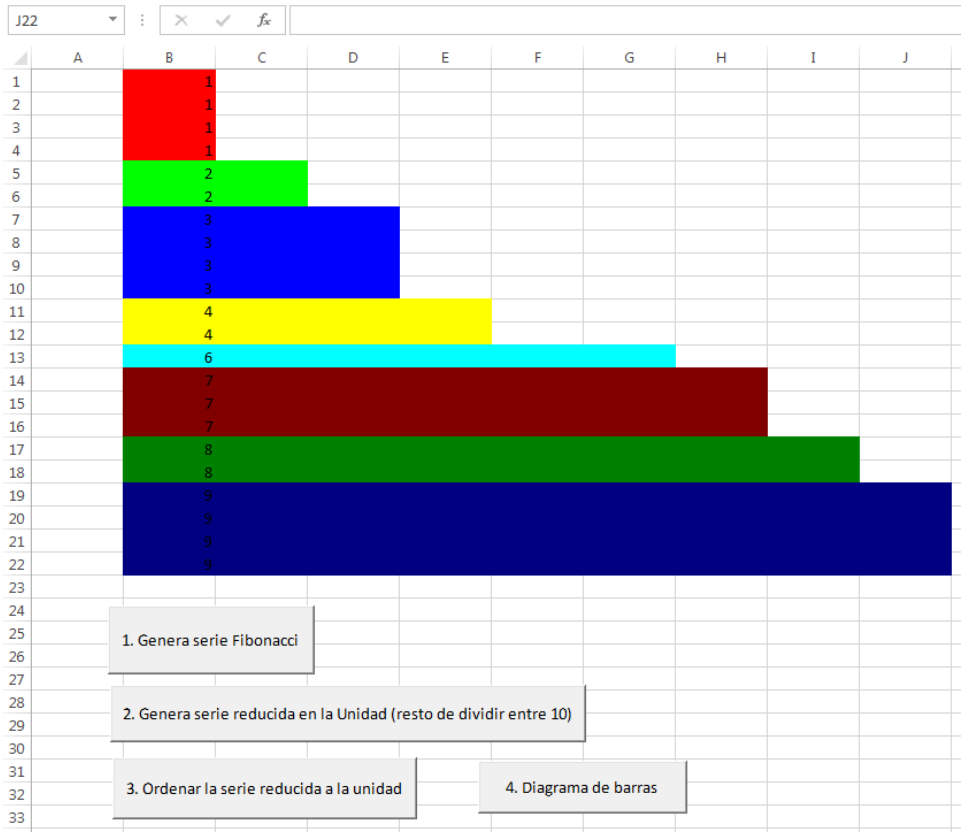
Este sería el último del conjunto de programas individuales (definidos mediante botones), con el se conseguiría el objetivo buscado.

En la página siguiente se muestra el código del diagrama de barras correspondiente a los valores de la serie anterior.
Este código muestra, después de un corto programa para borrar el diagrama que pueda existir con anterioridad (obtenido a partir del código de la macro grabada durante el borrado de un diagrama con las dimensiones requeridas, dándole al color el valor “sin relleno”), cómo hacer el diagrama de barras.
Primeramente, se define que el número de celdas a colorear (barras del diagrama), sea igual al número de la serie en cada fila. Después, se define una condición para evitar colorear una celda de la primera columna, cuando se tuviera un cero. Además, se define que el color corresponda

al número presente en cada celda, pero evitando el negro (correspondiente al 0), y el blanco (correspondiente al 1).

```
Private Sub CommandButton4_Click()

'Borrar color del Gantt anterior
Range("B1:J22").Select
    Selection.Interior.ColorIndex = xlNone

'Diagrama de Gant
Dim i As Integer
Dim j As Integer
Dim x As Integer
For i = 1 To 22
    x = Hoja1.Cells(i, 2)
    If 1 <> 0 Then
        For j = 2 To x + 1
            Hoja1.Cells(i, j).Select
            With Selection.Interior
                .ColorIndex = x + 2
                .Pattern = xlSolid
            End With

        Next j
    End If
Next i
```

Tras lo definido anteriormente, y haciendo clic sobre el cuarto botón, se obtendría el diagrama de Gantt correspondiente a la serie previamente calculada, cambiando cada vez que se ejecutara todo el proceso completo. ***Botón 1 → Botón 2 → Botón 3 → Botón 4.***



Una vez realizado lo anterior, vamos a mostrar como emplear la función de Visual Basic, ***Call***. Con esta función lo que pretendemos, es poder hacer llamadas desde dentro de un programa a otro que puede ser utilizado varias veces, y de esta forma, evitaríamos tener que definir el programa correspondiente cada vez.

```
(General)
Private Sub CommandButton1_Click()

Range("B1:J22").Select
    Selection.Interior.ColorIndex = xlNone

Hoja1.Cells(1, 2) = Int(Rnd() * 10)
Hoja1.Cells(2, 2) = Int(Rnd() * 10)
i = 1

'Generación de la serie de Fibonacci

Do While i < 21
    Hoja1.Cells(i + 2, 2) = Hoja1.Cells(i + 1, 2) + Hoja1.Cells(i, 2)
    Hoja1.Cells(i, 2).Select
    i = i + 1
Loop
Call Reducir
Call Ordenar
Call BorrarColorDiagrama

End Sub

Private Sub Reducir()
```

En este caso, vemos como una vez definida la serie de Fibonacci (de la misma forma que ya se ha visto previamente en varias ocasiones, siguiendo el mismo ejemplo), se introducen tres

llamadas a otras tantas funciones independientes previamente definidas (como se ha visto en las transparencias precedentes), mediante la función ***call***.
Así, una vez calculada mediante el bucle ***Do While*** la serie de Fibonacci, se llamaría inicialmente a la función ***Reducir***. Ésta, como ya se ha visto, tomaría el resultado anterior, "reduciéndolo" a la cifra de unidades correspondiente a cada uno de los elementos de la serie anterior.
Se vería, de la misma forma que se veía en un punto anterior, como con la utilización de la función resto ***mod***, entre 10, conseguimos tomar o "reducir" la cifra correspondiente a las unidades de los elementos de la serie de Fibonacci previamente calculada.

```
Private Sub Reducir()

i = 1

'Generación de la serie anterior reducida a la unidad

Do While i < 23
    Hoja1.Cells(i, 2) = Hoja1.Cells(i, 2) Mod 10
    i = i + 1
Loop

End Sub

Private Sub Ordenar()
```

A continuación, se llama a la función ***Ordenar***, que realizará la ordenación de los elementos de la serie numérica previamente calculada, de mayor a menor (siendo este código obtenido, como ya se había explicado, a partir de la grabación de una macro utilizando la función ***ordenar*** del menú). Tomando como rango de elementos a ordenar, la primera columna (A), desde la celda 1 a la 15, en este caso.

```
Private Sub Ordenar()

Range("B1:J22").Select
    Selection.Interior.ColorIndex = xlNone

'Ordenar la serie reducida

    Range("B1:B22").Select
    ActiveWorkbook.Worksheets("Hoja 1").Sort.SortFields.Clear
    ActiveWorkbook.Worksheets("Hoja 1").Sort.SortFields.Add Key:=Range("B1:B22") _
        , SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
    With ActiveWorkbook.Worksheets("Hoja 1").Sort
        .SetRange Range("B1:B22")
        .Header = xlGuess
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With
End Sub

Private Sub BorrarColorDiagrama()
```

La última llamada realizada desde la función principal, sería la realizada a la función encargada de definir el diagrama de barras, en tamaño y en color, además de definir otra subfunción que se encargaría de borrar el diagrama anterior, cada vez que se hiciera clic en el botón para obtener una nueva serie y un nuevo diagrama de barras.

```
Private Sub BorrarColorDiagrama()

'Borrar color del Gantt anterior
Range("B1:J22").Select
    Selection.Interior.ColorIndex = xlNone

'Diagrama de Gant
Dim i As Integer
Dim j As Integer
Dim x As Integer
For i = 1 To 22
    x = Hoja1.Cells(i, 2)
    If 1 <> 0 Then
        For j = 2 To x + 1
            Hoja1.Cells(i, j).Select
            With Selection.Interior
                .ColorIndex = x + 2
                .Pattern = xlSolid
            End With

        Next j
    End If
Next i


End Sub
```

Como ya se ha explicado antes, se definiría una función encargada de tomar el valor de cada uno de los elementos de la serie en la columna A, luego, y mientras ésta fuera diferente de 0, se entraría en el bucle, en el se definiría el tamaño y el color de la barra en función del número de la serie en cada posición.

4. NUEVOS EJEMPLOS: OBTENCIÓN DE NÚMEROS PRIMOS

Una vez visto todo lo realizado previamente, se va a pasar a describir estos nuevos ejemplos. En ellos, vamos a mostrar cómo hacer dos programas, el primero para saber si un número es primo, y el segundo, para obtener listas de números primos.

Para esto, en el primer programa, mostraremos qué funciones se deben utilizar para declarar menús de trabajo, y cómo trabajar con ellas, además de cómo llamar a otras funciones sin utilizar la función que se había visto previamente para este propósito (función ***call***). En el segundo programa, veremos de qué forma se podrán declarar listas de números primos, en un número indicado previamente por nosotros mismos.

Vamos a ver ahora qué es lo que deberemos hacer para poder declarar y utilizar menús de trabajo, aplicándolo de manera práctica para poder declarar si un número dado al programa es primo o no.

```
CommandButton1
Private Sub CommandButton1_Click()

    Dim numero As String
    Dim valor As Integer
    numero = InputBox("DIME UN NUMERO")
    valor = Val(numero)
    Dim primo As Boolean
    If esprimo(valor) Then MsgBox ("ES PRIMO") Else MsgBox ("NO ES PRIMO")

End Sub

Function esprimo(x) As Boolean
```

Como se puede ver en la pantalla anterior del ***Editor de Visual Basic***, el programa previamente descrito se ha dividido en dos partes. En la parte que vemos aquí (declarada a partir del botón) mediante la función ***InputBox***, se declarará un menú que se verá en la página de la Hoja de Cálculo del ***Microsoft Excel***, presentando el texto "DIME UN NUMERO", identificado con la variable ***numero*** definida como ***string***. Esta cadena (que recibe el número que se introduciría desde teclado) mediante la función ***Val***, registrará el valor numérico deseado que se pasaría a la otra función (la que calcularía si dicho número es primo o no).
Esto también se podría haber conseguido de una manera un poco más simple, declarando únicamente ***valor*** como entero y guardando el número introducido en la ***InputBox*** directamente como entero como se puede ver en la siguiente captura:

```
(General)
Private Sub CommandButton1_Click()

    Dim valor As Integer
    valor = InputBox("DIME UN NUMERO")
    Dim primo As Boolean
    If esprimo(valor) Then MsgBox ("ES PRIMO") Else MsgBox ("NO ES PRIMO")

End Sub

Function esprimo(x) As Boolean
```

Una vez hecho esto, dentro de una condición ***if***, y utilizando la función ***MsgBox*** (esta función, al igual que la previamente definida ***InputBox***, tiene como misión el mostrar en pantalla un mensaje en forma de menú de Windows, pero ahora presentando un resultado determinado y definido desde programa) se mostraría un mensaje sobre la Hoja de Cálculo, diciendo si el número previamente introducido es primo o no.

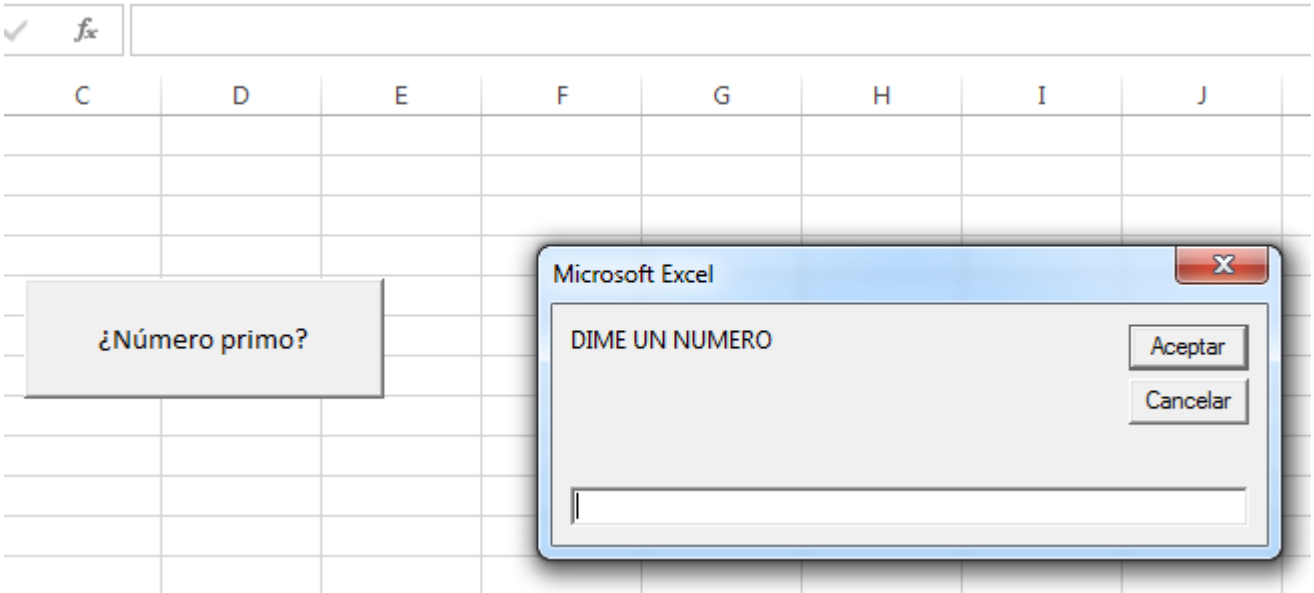
Tal como se ha visto previamente, tomando el valor de la variable ***valor*** se llama a la función ***esprimo (x)***, donde la variable ***x*** equivale al valor enviado ***valor***. Así, definiendo esta función como ***Boolean***, la cual daría como resultado una respuesta verdadera o falsa (***true*** o ***false***), se entraría en un bucle ***Do While*** (que utiliza como condiciones que el número introducido es

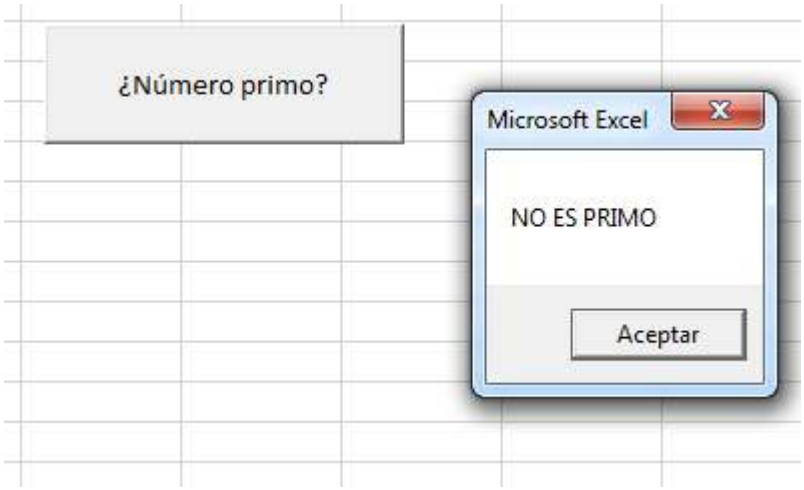
inicialmente primo, para entrar en el bucle, y que el último número por el que se dividirá el introducido, para comprobar si es primo o no, deberá ser menor o igual a la raíz cuadrada del introducido). En este bucle, dentro se pondría una condición ***if***, en la que indica que para que un número no sea primo, el resto de dividirlo por otro menor que él debe ser cero.

```
End Sub

Function esprimo(x) As Boolean
    Dim n As Integer
    esprimo = True
    n = 2
    Do While esprimo And n <= Sqr(x)
        If (x Mod n = 0) Then
            esprimo = False
        End If
        n = n + 1
    Loop

End Function
```

Como se puede comprobar, al trabajar con variables Booleanas, se devuelve o recibe un ***True*** o un ***False***, que en función de la definición de la condición ***if*** del siguiente programa (el definido por el botón), se dará como resultado lo correspondiente al "*si*" (***if***) o al "*sino*" (***else***).

Ahora se ve cómo quedaría en la pantalla de la hoja de Excel lo expuesto previamente. Se ve, en la página siguiente, como al hacer clic sobre el botón, aparecería el menú pidiendo un número, y acto seguido se diría si éste es primo o no.





Ahora se van a definir los dos programas necesarios para obtener un número determinado de números primos, siguiendo el mismo esquema previamente definido. Primero se ve cómo se define con la función ***InputBox***, un nuevo menú en el que se pide el número de números primos deseado. Además, se incluye una línea de código para poder borrar el listado previo de números primos cada vez que se haga clic sobre el botón (para que salga un nuevo menú).

```
CommandButton1
Private Sub CommandButton1_Click()

    Dim valor As Integer
    Call Borrar
    valor = InputBox("¿Cuántos números primos quieres?")
    Call esprimo(valor)

End Sub

Function esprimo(x)
```

Esta llamada mediante la función ***Call***, se hace a una macro grabada mientras se seleccionaba toda la columna A y se borraba su contenido, como se puede ver.

```
(General)
Sub Borrar()
'
' Borrar Macro
'

'
    Columns("A:A").Select
    Selection.ClearContents
End Sub
```

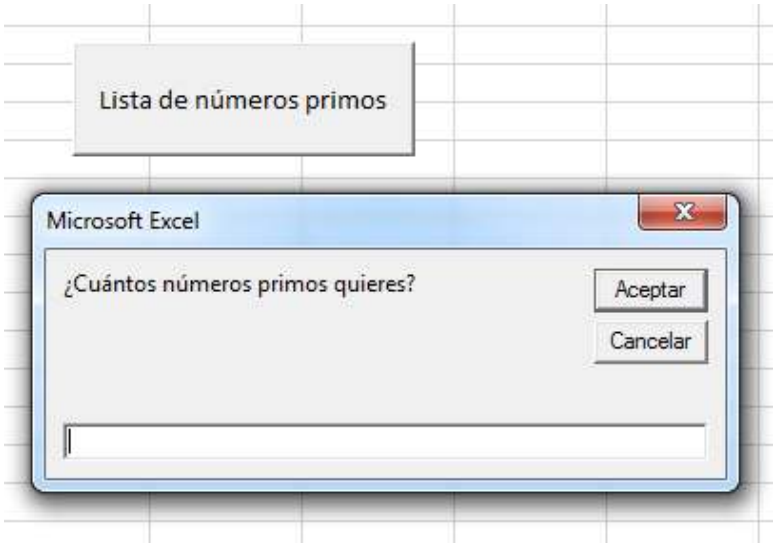
Aquí se puede observar el segundo programa, llamado por el primero, y pasándole el número de números primos a generar (tamaño de la lista) empezando por el 2.

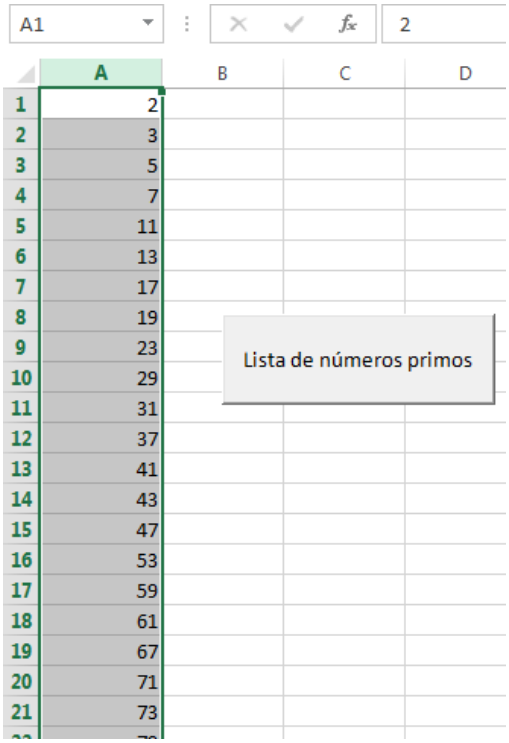
```
Function esprimo(x)
    Dim n As Integer
    Dim num As Integer
    cont = 1
    n = 2
    Do While cont <= x
        primo = True
        j = 2
            Do While primo And j <= Sqr(n)
                If (n Mod j = 0) Then primo = False
                j = j + 1
            Loop
            If primo Then
                Hoja1.Cells(cont, 1) = n
                cont = cont + 1
            End If
            n = n + 1
        Loop

End Function
```

Aquí se ve como una vez se recibe la información de la otra función, se definen dos contadores para controlar los dos bucles. El primero controlaría la acumulación de números primos hasta la cantidad indicada (***x***), y a continuación, al igual que antes, definiendo la variable ***primo*** como booleana, se entra al segundo bucle (encargado del cálculo de los primos) suponiendo que la primera entrada es un número primo (2 es primo) y que además el número por el que se divida cada número para comprobar que sea primo, deberá ser inferior a la raíz cuadrada de dicho número.

Finalmente, con una condición, se irían acumulando en la columna los diferentes números primos encontrados hasta llegar a la cantidad deseada.

Aquí se verá ahora el resultado deseado.





5. FINAL

Una vez presentados y explicados los ejemplos anteriores, esperamos que sirvan de ayuda real a la realización de los problemas concretos.

También, y porque no, esperamos que este pequeño manual pueda llegar a servir como herramienta de inicio de otros posibles futuros trabajos encaminados en esta materia.

Esperamos, de la misma forma, que la exposición haya sido suficientemente sencilla y clarificadora de lo que inicialmente se pretendía y se presentaba como objetivos.

6. ANEXO (Sentencias y funciones habituales)

Antes de empezar con el anexo queremos incorporar una nota. Por un motivo que desconocemos Excel ha empeorado su comportamiento desde algunas versiones hacia aquí. Macros que funcionaban estupendamente se han convertido en muy lentas. Tras una indagación en la web parece que si se pega esto al principio de las aplicaciones mejora el funcionamiento.

Application.screenupdating=False
Application.calculation=xlCalculationManual
Application.EnableEvents=False

Hay que pegar esto al final antes del end sub:

Application.screenupdating=True
Application.calculation=xlCalculationAutomatic
Application.EnableEvents=True
Application.CutCopyMode = False

En estos anexos se podrán encontrar instrucciones para Visual Basic y para las hojas de cálculo de Excel. A éstas últimas se les puede reconocer fácilmente dentro de los anexos porque no van introducidas dentro de ningún "Sub" y además las instrucciones referidas a las hojas de cálculo Excel van escritas en mayúsculas. Un ejemplo de una instrucción referida a las hojas de cálculo Excel puede ser:

DESREF (C11; 0; SI (C6>C3;-C3;-C6); 1; 1)

6.1. Problemas con variables (¿por qué no se dejan definir las variables?)

6.1.1. Option Explicit:

Con esta aplicación, se avisaría en caso de no tener definida una variable, o en caso de utilizar datos de páginas diferentes a la activa.

6.1.2. Dim…. As [Integer, Double, String, Boolean…]:

Con esto queda la variable perfectamente definida, si no se pusiera no ocurriría posiblemente nada, salvo que se utilizaría una mayor cantidad de memoria de la necesaria, al definirse instantáneamente en el momento de utilizarla como de tipo ***Value***.

6.2. Condición If…, etc. (diversas posibilidades):

If … Then … / If … Then … Else … / If … Then … ElseIf … Then …
¿Cuándo poner el ***EndIf***?, ¿cuándo no?, ¿cuándo se deberían usar los ":" (dos puntos)?

Las instrucciones ***If...Then...Else*** se pueden presentar en varios formatos, con unas características determinadas. Normalmente, se presentan anidadas en tantos niveles como sea necesario. Esto, sin embargo, puede hacer menos legible el código, por lo que es aconsejable utilizar una instrucción ***Select Case*** en vez de recurrir a múltiples niveles de instrucciones ***If...Then...Else*** anidadas (únicamente en caso de que el excesivo número de anidamientos pudiera dar problemas en la legibilidad del programa, o errores en la depuración de éste).

Así, si realizamos la condición en varias líneas de código, será necesario cerrar el anidamiento con un ***End If***; instrucción que no se usaría en caso de realizar la condición en un sola línea (***If Then***, condición cierta).

6.2.1. Select Case

En este caso, esta instrucción será más útil que la Condición ***If...***, cuando se ejecute uno de varios grupos de instrucciones, dependiendo del valor de una expresión condición a cumplir.

6.2.2. Ejemplos de utilización

Ahora se presentan una serie de ejemplos prácticos, con los que aclarar y facilitar el uso de las condiciones ***If*** en la programación en Visual Basic.

Básicamente, en el ejemplo siguiente se observa, como se deberían de utilizar los anidamientos consecutivos de ***If… Then***, ***ElseIf… Then*** y ***Else***. Se observa aquí, cómo se utilizaría el ***End If***, siempre en el caso del anidamiento de condiciones, y no en el caso de escribir la condición ***If*** (general) en una sola línea.

```
If b > 0 And c > 0 Then
        Hoja1.Cells(i, 8) = a
Else
        If b > 0 And c < 0 Then
                Hoja1.Cells(i, 8) = a + 360
        Else
                Hoja1.Cells(i, 8) = a + 180
        End If
End If
```

6.3. Bucles: For… To … Next/Do While… Loop/Do Loop… Until (Utilización y posibles problemas):

Las estructuras de bucle también son conocidas por el nombre de estructuras de control. Permitiendo la repetición de determinadas acciones.

Uno de los errores más comunes que se producen en la utilización de bucles de este tipo, es la no inicialización de las variables utilizadas como contadores de iteraciones. Así que habrá que prestar una atención especial en este punto. Una opción para evitar este posible error, sería la definición al principio del programa, como primera línea de código de éste, el ya comentado ***Option Explicit***.

A continuación se presentan las diferentes opciones que permite el Visual Basic para definir bucles, es decir, repetición y/o acumulación de acciones determinadas, entre unos límites definidos. La no definición de estos límites concretos, sería otro error común y más problemático, al producirse la entrada en bucles infinitos, que bloquean el módulo de cálculo de nuestro ordenador.

6.3.1. Do… Loop Until

Esta estructura de control se puede usar para ejecutar un bloque de instrucciones un número indefinido de veces. Las instrucciones se repiten hasta que una condición llegue a ser ***True***.

Un ejemplo podría ser el siguiente:

```
Sub ComPrimeroUntil ()
        contador = 0
        miNum = 20
        Do Until miNum = 10
                miNum = miNum – 1
                contador = contador + 1
        Loop
        MsgBox "El bucle se ha repetido " &contador& " veces."
End Sub
```

6.3.2. Do While… Loop

Siguiendo lo explicado en el punto inicial, otro error común sería el no introducir la línea de acumulación del contador (por ejemplo: i = i + 1), con lo que el bucle entraría cada vez en el cálculo, quedándose colgado en este punto.

En este caso, las instrucciones se repiten mientras una condición sea ***True*** (al contrario que con el ***Do… Loop Until***).

Este tipo de bucle se utilizaría normalmente en caso de tener que cumplirse una condición marcada por el ***While***. Así, en este tipo de bucles, se puede dar el caso de que no se entre desde el primer momento, debido al no cumplimiento de esta condición.

Ejemplo de utilización de esta función (hay que fijarse en la inicialización previa de la variable contador ***i***):

```
i = 5
Do While Hoja1.Cells(i, 2) <> ""
        b = Hoja1.Cells(i, 5)
        c = Hoja1.Cells(i, 6)
        d = (b ^ 2) + (c ^ 2)
        a = Sqr(d)
        Hoja1.Cells(i, 7) = a
        i = i + 1
Loop
```

6.3.3. For… To… Next

Mediante la palabra clave Step, se puede aumentar o disminuir la variable contador en el valor que se desee (***For j = 2 To 10 Step 2***).
Se pueden anidar bucles ***For...Next***, colocando un bucle ***For...Next*** dentro de otro. Para ello, hay que proporcionar a cada bucle un nombre de variable único como su contador. La siguiente construcción es correcta:

```
For i = 1 To 10
        For j = 1 To 10
```

```
            ...
        Next j
    Next i
```

Si se omite un contador en una instrucción ***Next***, la ejecución continúa como si se hubiera incluido. Se produce un error si se encuentra una instrucción ***Next*** antes de su instrucción ***For*** correspondiente.

Al contrario de lo que se comentaba para los bucles ***Do… While***, los bucles ***For…***, se ejecutarán hasta agotar el intervalo de acumulación del contador, es decir, siempre se entraría en el bucle, y no se pararía de ejecutar hasta no terminar el contador.

Un ejemplo concreto podría ser el siguiente:

```
For i = 1 To 15
        x = Hoja1.Cells(i, 1)
        If x <> 0 Then
                For j = 2 To x + 1
                        Hoja1.Cells(i, j).Select
                         With Selection.Interior
                            .ColorIndex = x + 2
                            .Pattern = xlSolid
                          End With
                Next j
        End If
Next i
```

Donde se ve como se deben anidar varios bucles consecutivos, y como se introducen funciones condicionales como ***If***, y otras estructuras de control, como el ***With***, que pasamos a comentar a continuación.

6.3.4. With

Estructura de control, que permite ejecutar una serie de instrucciones sin necesidad de recalificar un objeto, es decir, sobre el mismo objeto cada vez; entendiendo por objeto toda combinación de código y datos que se pueden tratar como una unidad, por ejemplo, un control, un formulario o un componente de una aplicación. Cada objeto se define por una clase.

Así, un ejemplo de utilización de la función ***With***, sería el siguiente, donde ha sido anidada dentro de la estructura de un bucle ***For***, y mediante la opción ***Select*** (utilizada para trabajar en entornos gráficos), se adjudicaría a cada celda de la columna identificada por el contador ***j***, un color y un tipo de letra determinado.

```
For j = 2 To x + 1
        Hoja1.Cells(i, j).Select
        With Selection.Interior
                .ColorIndex = x + 2
```

```
                .Pattern = xlSolid
            End With
        Next j
```

6.4. Coordenadas polares: ¿Cómo pasar de coordenadas cartesianas (x,y) a polares (r,α)?:

6.4.1. Radio (calculado a partir de las coordenadas x e y de los puntos en cuestión) r = RaizCuadrada(x^2+y^2):

Para este caso, se definiría la estructura de control siguiente (o bucle), definida mediante la utilización de la función ***DoWhile… Loop***. En ella se definirían una serie de variables que acumularían los valores previamente definidos en celdas de la Hoja de Cálculo, para realizar la operación de calcular la raíz cuadrada de la suma de los cuadrados de los catetos opuesto y contiguo del triángulo definitorio del ángulo a calcular. Posteriormente, se le daría dicho valor a otra variable, que estaría encargada de ir dándole dichos valores a las celdas correspondientes de la Hoja de cálculo anterior.

```
i = 5
Do While Hoja1.Cells(i, 2) <> ""
        b = Hoja1.Cells(i, 5)
        c = Hoja1.Cells(i, 6)
        d = (b ^ 2) + (c ^ 2)
        a = Sqr(d)
        Hoja1.Cells(i, 7) = a
        i = i + 1
Loop
```

6.4.2. Ángulo (calculado a partir de las coordenadas x e y de los puntos en cuestión) α=Arctan (x/y):

Se considera el mismo proceso anterior, pero en este caso, y para poder presentar los valores del ángulo correspondiente en grados entre 0^{o} y 360^{o}, puesto que Excel sólo los presenta en valores entre 90^{o} y -90^{o}, se utiliza la estructura condicional que se puede observar en el programa.

```
i = 5
Do While Hoja1.Cells(i, 2) <> ""
        b = Hoja1.Cells(i, 5)
        c = Hoja1.Cells(i, 6)
        d = c/b
        a = (180 / PI) * Atn(d)
        If b > 0 And c > 0 Then
                Hoja1.Cells(i, 8) = a
        Else
```

```
                If b > 0 And c < 0 Then
                        Hoja1.Cells(i, 8) = a + 360
                Else
                        Hoja1.Cells(i, 8) = a + 180
                End If
        End If
        i = i + 1
Loop
```

Hay que darse cuenta de que se utiliza la condición anidada ***If ... Then ... Else ... End If***, porque ***Excel***, da valores de ángulo en el plano de las X positivas (1er y 4º cuadrantes), por lo que para poder tener una visión clara de la posición de cada punto en función de su ángulo (tenerlo marcado de 0^{o} a 360^{o}), se debería sumar 180 a los valores de ángulo obtenidos de los puntos situados en el 2º y 3er cuadrantes, y 360 a aquellos situados en el 4º cuadrante.

6.5. Cambiar criterios de ordenación:

Aquí se puede ver cómo se podrían definir criterios de ordenación (ascendente o descendente), en función de la necesidad del programador, y respecto a una columna o rango predefinida.

```
Application.AddCustomList ListArray:=Range("J2:J21")
  numlista = Application.CustomListCount

  Selection.Sort Key1:=Range("B2"), Order1:=xlAscending, Header:=xlGuess, _
        OrderCustom:=numlista + 1, MatchCase:=False, Orientation:=xlTopToBottom,_
        DataOption1:=xlSortNormal
  Application.DeleteCustomList ListNum:=numlista
```

6.6. Menús…

Se definen menús específicos tomándolos como variables definidas como barras de comandos de control, o de otros tipos, y dándoles a su vez los nombres correspondientes a estos menús de trabajo.

```
Dim MenuAyuda As CommandBarControl
Dim MenuNuevo As CommandBarPopup
Dim Plan As CommandBarControl
Call BorrarMenu
Set MenuAyuda = CommandBars(1).FindControl(ID:=30010)
If MenuAyuda Is Nothing Then
        Set MenuNuevo = CommandBars(1).Controls.Add(Type:=msoControlPopup,
        Temporary:=True)
Else
        Set MenuNuevo = CommandBars(1).Controls.Add(Type:=msoControlPopup,
        Before:=MenuAyuda.Index, Temporary:=True)
```

```
End If
MenuNuevo.Caption = "Plan de Recuento"
Set Plan = MenuNuevo.Controls.Add(Type:=msoControlButton)
Plan.Caption = "Plan de Recuento"
Plan.OnAction = "CalculaPlan"
```

6.7. Para Ordenar

Básicamente, se busca lo mismo que cuando se hablaba del cambio de criterios de ordenación.

```
Range ("D18:F23").Select
Selection.Sort Key1:=Range("D19"), Order1:=xlAscending, Header:=xlYes, _
        OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom, _
        DataOption1:=xlSortNormal
```

6.8. Quitar el signo de los números convertidos en string:

Estas líneas de código sirven para poder tomar números positivos siempre, aunque se introdujeran negativos (por error o cálculo), a través de un menú, cálculo,… El resultado sería semejante a la utilización de la función del ***Editor de Visual Basic***, ***Abs (numero)*** (que devuelve el valor absoluto de todo número introducido entre paréntesis.

```
nombre = Str(i)
nombre = Right(nombre, Len(nombre) - 1)
```

6.9. Cuando queremos poner referencias relativas a variables en la fórmula:

Referencias que corresponderían con los valores de la celda correspondiente de la Hoja de Cálculo con la que se está trabajando.

```
DESREF (C11; 0; SI (C6>$C$3;-$C$3;-C6); 1; 1)
```

6.10. Temporizador:

Función encargada de dar un intervalo de tiempo, previamente a la obtención de un resultado, o por otra razón necesitada por el programador.

```
Dim ppio As Single
ppio = Timer
Do While ppio + 10 > Timer
Loop
```

6.11. Funciones:

Definición de funciones (con la forma que se requiera, sea ***Integer*** para entero, o de cualquier otro tipo), dentro del programa, en el ***Editor de Visual Basic***, con la intención de tenerlas

definidas a parte de la programación del botón en la Hoja de Cálculo, o para una rellamada a posteriori, por ejemplo con la función ***Call***.

```
Function fact(x) As Integer
End Function
```

6.12. Zoom de la ventana:

Zoom, agrandará o empequeñecerá la presentación del formulario preseleccionado, en la ventana activa de trabajo.

```
ActiveWindow.Zoom = 25
```

6.13. Para cancelar el botón:

Bastaría con introducir la orden siguiente:

```
End
```

6.14. Procedimiento que empieza con un formulario:

```
Sub Prevision_Userform()
CommandButton1.Caption = "Previsión"
End Sub
```

6.15. Otro modo de cambiar el color:

Esta es una de las opciones válidas para el cambio de color, en una celda, o en cualquier otro objeto seleccionado. Hay que considerar, que en este caso se realizaría mediante una graduación de los tres colores básicos disponibles (rojo, verde y azul), aunque también podría hacerse mediante valores numéricos globales, representando las mezclas correspondientes de estos colores básicos.

```
Label10.BackColor = RGB(242, 148, 150)
```

6.16. Para abrir un formulario:

Línea de código que mostraría/abriría un formulario, que en este caso ha sido llamado Prevision.

```
frmPrevision.Show
```

6.17. Para ocultar un formulario:

```
frmPrevision.Hide
```

Ambas sentencias (la 6.16 y la 6.17), sencillas como se puede comprobar, se refieren a la apertura de formularios referentes a objetos determinados. También estarían aquí relacionados los ***UserForm***.

6.18. Procedimiento que empieza automáticamente:

Este procedimiento, abriría…

```
Sub auto_open()
```

6.19. Borrar Menu:

En este caso se borraría un menú previamente creado (ver el punto previo 6.6, por ejemplo), en este caso, el menú “Nuevo Análisis”.

```
On Error Resume Next
CommandBars(1).Controls("Nuevo Análisis").Delete
```

6.20. Crear Rango:

Aquí se crearía un rango, sin tener que seleccionarlo previamente en la página de trabajo de la Hoja de Cálculo, desde la celda B5.

```
rango = Str(nuevoprod - 1)
rango = "B5:D" + Right(rango, Len(rango) - 1)
```

6.21. Entero y Logaritmo:

Con la sintaxis siguiente, se transforman números reales logarítmicos (obtenidos mediante la función logaritmo ***Log***), en un número entero; para utilizarlo, por ejemplo, en el caso de disponer de poca memoria para una variable, o por necesidad de trabajar con números pequeños.

```
aux = Int(Log(x) / Log(2))
```

Esta transformación (no sería una transformación en sí mismo, sino que se tomaría simplemente la parte entera del número real) de un número real en otro entero, se ha visto también utilizada en este manual en el caso de trabajar con series de números aleatorios, para obtener así números sin la coma flotante, y por consiguiente, más manejables.

6.22. Poner bordes:

Aquí, se puede observar otra utilización de la función ***With*** como estructura de control, para a través de la función ***Select***, darle un formato gráfico determinado al rango de la Hoja de Cálculo con la que se está trabajando. En este caso, el formato gráfico buscado, sería el de un borde a un texto o zona de texto, con un tipo de línea, grosor y color determinados.

```
With Selection.Borders (xlEdgeLeft)
        .LineStyle = xlContinuous
        .Weight = xlMedium
        .ColorIndex = xlAutomatic
End With
```

6.23. Pregunta un número:

Otra nueva utilización de los cuadros de mensaje para captación de datos (o menús) del tipo ***Inputbox***, para, en este caso capturar una cantidad determinada con la que el programa en cuestión realizará los cálculos deseados.

```
InputBox ("Dime un número")
```

6.24. Ventana de mensajes:

Lo mismo que en el punto anterior, pero en este caso, mostrando un mensaje determinado en la Hoja de Cálculo de trabajo.

```
MsgBox ("Hola")
```

6.25. Se mueve a la siguiente celda a la derecha:

Sentencia explícita, para seleccionar una celda contigua a la tomada previamente en la Hoja de Cálculo de trabajo, como celda activa donde tomar o mostrar datos.

```
ActiveCell.Next.Select
```

6.26. Pegado transpuesto:

Línea de código resultante de la grabación de la macro correspondiente a la selección de la opción de pegado especial transpuesto, para poder copiar una fila en una columna o viceversa. Opción presente en la barra de menú de la Hoja de Cálculo.

```
Selection.PasteSpecial Transpose:=True
```

6.27. Copiar un rango de una página a otra:

Con esta opción se copiarán los datos presentes en el rango seleccionado en la página de trabajo (en este caso la Hoja1), en el rango seleccionado correspondiente en la página siguiente de la Hoja de Cálculo, u Hoja2.

```
Hoja1.Range(rango).Copy Destination:=Hoja2.Range(rango)
```

6.28. Definición de Rango Automático:

Este método, tomaría la celda activa de la Hoja de Cálculo de trabajo, en realidad, se le daría la la celda activa final del rango deseado de trabajo, y finalmente se le indicaría, si dicha selección del rango se debe realizar hacia arriba o hacia abajo (como es este el caso).

```
ActiveCell , ActiveCell.End(xlDown)
```

6.29. Cálculo de Máximo:

Aplicación del ***Editor de Visual Basic***, mediante la que se calcularía directamente el valor máximo del rango de la Hoja de Cálculo de trabajo previamente seleccionado.

```
Application.Max (Rango)
```

6.30. Formato interior de Celda:

Con este código de programa, se le daría a la celda activa, o seleccionada (o al rango activo o seleccionado en su interior), unos valores determinados de grado de color, y de formato de texto.

```
Selection.Interior.ColorIndex=34
Selection.Interior.Pattern=xlsolid
```

6.31. Enteros aleatorios entre límites:

Para producir enteros aleatorios en un intervalo dado, usa esta fórmula:

```
Int ((Límite_superior - límite_inferior + 1) * Rnd + límite_inferior)
```

Aquí, ***límite_superior*** es el número mayor del intervalo y ***límite_inferior*** es el número menor del intervalo.

Nota: Para repetir secuencias de números aleatorios, se debe llamar a la función ***Rnd*** con un argumento negativo antes de utilizar la función ***Randomize*** con un argumento numérico. Al utilizar la instrucción ***Randomize*** con el mismo valor de número, no se repite la secuencia anterior.

6.32. Suprimir los cuadraditos en un texto importado:

*Se ha importado en la columna **A** un texto desde otro programa pero todo aparece lleno de pequeños cuadraditos que se deberían suprimir.*

Para ello, se puede utilizar esta macro para conocer los códigos de los caracteres que los generan.

```
Sub acode()
   For i = 1 To 255
   Range("a" & i) = Chr(i)
   Next
End Sub
```

Para reemplazar estos caracteres por un espacio, se puede utilizar esta otra macro:

```
Sub Macro1Cuadrados()
  Dim c
  For Each c In Range("A1:" & _
    Range("A1").SpecialCells(xlCellTypeLastCell).Address)
      For i = 1 To 31
        Application.StatusBar = c.Address & " " & i
        On Error Resume Next
        Range(c.Address) = Application.Substitute(c, Chr(i), " ")
        'Err.Clear
```

```
        'Resume
    Next
    Range(c.Address) = Application.Substitute(c, Chr(127), " ")
    Range(c.Address) = Application.Substitute(c, Chr(129), " ")
    Range(c.Address) = Application.Substitute(c, Chr(141), " ")
    Range(c.Address) = Application.Substitute(c, Chr(143), " ")
    Range(c.Address) = Application.Substitute(c, Chr(144), " ")
    Range(c.Address) = Application.Substitute(c, Chr(157), " ")
  Next
  Application.StatusBar = False
End Sub
```

6.33. Seleccionar los caracteres en una celda Excel:

¿Cómo elegir por orden alfabético creciente o decreciente una celda Excel que contenga una cadena de caracteres?

Para esto, se puede utilizar esta función de T.Shuttleworth con algunas modificaciones:

```
Option Compare Text
Function SortString(ByVal iRange, Optional Creciente As Boolean = True)
Dim i%, j%, sTemp$
    For j = 1 To Len(iRange) – 1
        For i = 1 To Len(iRange) – 1
            If Mid(iRange, i, 1) > Mid(iRange, i + 1, 1) Then
                sTemp = Mid(iRange, i, 1)
                Mid(iRange, i, 1) = Mid(iRange, i + 1, 1)
                Mid(iRange, i + 1, 1) = sTemp
            End If
        Next i
    Next j
      If Creciente = False Then
      For i = Len(iRange) To 1 Step -1
          SortString = SortString & Mid(iRange, i, 1)
       Next
       Exit Function
End If
SortString = iRange
End Function
```

También se puede utilizar esta solución mediante fórmula MATRICIAL utilizando la XLL (morefun.xll) que se puede cargar en:
http://www.freewarefiles.com/downloads_counter.php?programid=14922

```
=MCONCAT(TRIV(STXT(A1;SIGUIENTE(NBCAR(A1);1);1);;1))
```

6.34. Insertar automáticamente retornos de carro en un texto:

Al introducir texto mediante una macro en una celda, se pretende que este texto sea cortado cada 100 caracteres, pero sin cortar las palabras.

El texto en cuestión podría ser truncado con la macro siguiente:

```
Function Corte(TxTronque As String, LgMax As Integer) As String
    Dim i As Integer
    Dim p As Integer
    Dim FinLigne As Long
p = 1
i = 0
Do While i < Len(TxCorte)
FinLigne = InStr(p, TxCorte, Chr(10))
    If FinLigne > LgMax Then
    i = i + LgMax
    Else: i = FinLigne + LgMax
    End If
        Do While Mid(TxCorte, i, 1) <> " "
        i = i - 1
            If i = 0 Then
                If FinLigne = 0 Then i = p + LgMax: Exit Do
                i = FinLigne + LgMax: Exit Do
            End If
        Loop
        Mid(TxCorte, i, 1) = vbCr
            i = i + LgMax
        p = i + 1
Loop
Corte = TxCorte
End Function
```

6.35. Comodines de búsqueda:

Existen comodines en Excel para reemplazar los caracteres en una búsqueda.

El operador "***" puede reemplazar un grupo de caracteres, y el comodín "**?**" uno solo.

La utilización en la función búsqueda de "***"** y de "**?"** puede causar desórdenes importantes en los ficheros. Así, si se busca por ejemplo la palabra "completándola" y se quieren recuperar todas las posibilidades de escritura con los acentos o no, se utilizarían los comodines "*" y "**?**" de la siguiente forma "complet?ndola" o "complet*a"

El método siguiente reemplaza el contenido de todas las celdas seleccionadas por la palabra de reemplazamiento:

- Seleccionar una columna de textos
- Reemplazar todas las palabras "**ejemplo**" por "*"→ OK
- Después, reemplazar todos los "*" por la palabra "**ejemplo**" → borrando todas las celdas conteniendo el "*" y reemplazándolo por la palabra "**ejemplo**".

Se debe señalar que la forma más rápida para vaciar una hoja es reemplazar "*" por "" (se puede probar aunque sin grabarlo).

6.36. Reemplazar un carácter en una variable:

¿Cómo reemplazar en una variable un punto y coma por una coma?

Por ejemplo, si en A1 hay: "B1;B2;B3" y en el código principal, MiVariable = Range("A1").Value

¿Cómo reemplazarla en MiVariable sin tocar la celda A1?
Se tiene que pasar por una macro:

```
Private Sub CommandButton1_Click()
        Dim MiVariable As String
        MiVariable = Range("A1").Value
        Call reemplazo(MiVariable)
End Sub

Sub reemplazo(MiVariable As String)
        MiVariable = Replace(MiVariable, ";", ",")
        MsgBox MiVariable
End Sub
```

6.37. Reemplazo complejo conservando los 0:

*En una columna en formato texto cuando se aplica la función reemplazar el valor siguiente **85 por nada sobre un número del tipo **850005256325 los 0 desaparecen (5256325). ¿Cómo evitarlo?*

Por defecto... y cuando la búsqueda de "reemplazar" del Excel sea demasiado "inteligente"...

```
Sub remplt()
        txtSup = InputBox("¿Qué cadena de caracteres" _ & "deseas suprimir?")
        txtRemp = InputBox("¿Por cuál desea reemplazarla?")
        Application.ScreenUpdating = False
        For Each c In Selection
                c.Value = Replace(c.Value, txtSup, txtRemp)
        Next c
End Sub
```

6.38. Suprimir espacios:

Macro para suprimir los espacios que se encuentran delante de las cifras cortadas/pegadas a partir de una web.

```
Sub EliminarEspacios()
   Dim celda As Range
   For Each cellule In ActiveSheet.UsedRange
      cell.Value = LTrim(cell.Value)
   Next
End Sub
```

6.39. Lista de las letras del alfabeto:

¿Cómo conseguir una lista de letras del alfabeto que se incremente automáticamente?

Añadir esta lista como macro:

```
Sub AnadirListaPers()
    Application.AddCustomList ListArray:=Array("A", "B", "C", "D", "E", "F", "G", "H", "I",
    "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z")
    MsgBox "La nueva lista es la número: " & Application.CustomListCount
End Sub
```

Así se podrá incrementar una serie manualmente en Excel, además de lo demandado.

Para poner al día la lista circulante:

```
Private Sub UserForm_Initialize()
   Dim NuListe As Byte
   NuListe = 5
   For n = 1 To UBound(Application.GetCustomListContents(NuListe))
         ComboBox1.AddItem Application.GetCustomListContents(NuListe)(n)
   Next n
End Sub
```

6.40. Extraer una cadena de texto en medio de otra:

Ante textos del tipo: blablabla > texto a extraer cualquiera <123
Sabiendo que los únicos puntos de referencia son los < y > (únicos en el texto) y el número fijo de caracteres después del <

```
=EXTRAE(A1;BUSCA(">";A1)+1;NBCAR(A1)-BUSCA(">";A1)-4)
```

O si no, también podría funcionar:

```
=SUSTITUYE(EXTRAE(A1;ENCUENTRA(">";A1)+1;999);"<123";)
```

6.41. Quitar los números de una cadena de caracteres:

¿Es posible, en una celda, eliminar la cifra que sigue a un nombre?
Por ejemplo: en una tabla, se tienen los nombres siguientes con un número (sin espacio) ZAZA1, ZAZA2 etc.

El objetivo es el de encontrar ZAZA, quitando los números y sabiendo que a veces se pueden encontrar también ZAZA11, ZAZA252 y hasta ZA345ZA.

```
Function SoloTexto(s As String)
  For a = 1 To Len(s)
    If Mid(s, a, 1) <= 9 Then
    Else
      SoloTexto = SoloTexto + Mid(s, a, 1)
    End If
  Next
End Function
```

Atención: si una cifra se encuentra en el medio de la palabra igualmente se suprime.

O incluso: (aquí se conserva en lugar de quitar)

```
Range("B1")= Left(Range("A1"),4)
```

6.42. Buscar una cadena de caracteres en otra:

Se abre un fichero de texto y se lee línea a línea: debiéndose verificar que cada vez que se pasara de línea (retstring) se tuviera la cadena de caracteres ".htm". ¿Cómo se haría?

Por ejemplo, para buscar una “a” en hablar.

```
Position = InStr([inicio], "hablar", "a")
```

Si la cadena buscada se encuentra, el resultado es la posición del primer carácter de la cadena buscada en la cadena comprobada. Como esta función diferencia entre mayúsculas y minúsculas, se debería, o comprobar las dos, o comprobarlo todo en mayúsculas o todo en minúsculas.

Pudiéndose obtener algo así:

```
Do While f.AtEndOfStream <> True
    retstring = f.Readline
    Position = InStr(UCase(f.Readline), ".HTM")
    If Position > 0 Then
      'Instrucciones en caso de que se cumpla la condición
    Else
      'Instrucciones en caso de que NO se cumpla la condición
```

```
    End If
  Loop
```

6.43. Trocear una frase sin cortar las palabras:

Se querría cortar una frase, sin cortar las palabras, de tal manera que cada trozo de frase, puesto en celdas adyacentes, no comportara más de 20 caracteres.

```
Public Sub Parse20PerCell()
  Dim bigString As String
  Dim tempStr As String
  Dim cell As Range
  Dim pos As Integer
  Set cell = Range("A1")
  bigString = cell.Text
  Do While bigString <> ""
    Set cell = cell.Offset(0, 1)
    If Len(bigString) < 21 Then
      cell.Value = Trim(bigString)
      bigString = ""
    Else
      tempStr = Right(StrReverse(bigString), 21)
      pos = InStr(tempStr, " ")
      If pos = 0 Then
        MsgBox "More than 20 contiguous characters between spaces."
      Else
        cell.Value = Trim(StrReverse(Mid(tempStr, pos + 1, 255)))
        bigString = Mid(bigString, 22 - pos, 255)
      End If
    End If
  Loop
End Sub
```

Otra solución:

El número máximo de caracteres de la frase es, en este ejemplo, inferior a 1000 y el número de celdas en las que el texto se tiene que repartir se supone inferior a 100.

```
Sub test1()
  Set Rng = Sheets(1).Range("A1")
  iTotal = Mid(Rng, k + 1, 1000) & " "
  For j = 2 To 100
    For i = 21 To 1 Step -1
      If Mid(iTotal, i, 1) = " " Then
```

```
            k = i
        Exit For
          End If
        Next
        Rng(1, j).Value = Mid(iTotal, 1, k - 1)
        iTotal = Mid(iTotal, k + 1, 1000)
      Next
    End Sub
```

6.44. Última palabra de una frase:

¿Cómo conseguir con una fórmula de la hoja extraer la última palabra de una frase?

```
=DERECHA(A1;ENCONTRAR(" ";COINCIDIR(A1;LARGO(A1)-
FILA(INDIRECTO(""&LARGO(A1)));1);0))
```

6.45. Borrar el carácter de la derecha:

¿Cómo borrar el carácter situado más a la derecha en una celda?

Por ejemplo, si el texto se encuentra en B10

```
=IZQUIERDA(B10;LARGO(B10)-1)
```

6.46. Comprobar la presencia de una cadena de caracteres:

¿Cómo verificar que una cadena de caracteres se encuentre en una celda o en una variable?

```
=CONTAR.SI(A1;"*texto*")=1
```

Reenvía TRUE si la cadena de caracteres (texto) se encuentra en la celda A1.

7. Pequeños Ejercicios

1- Activar la pestaña Desarrollador/Programador que habilita el uso de Macros/Visual Basic.
2- Añadir un botón permitiendo que se le cambie el texto, el tipo de letra y el color del botón desde Propiedades.
3- Grabar un vídeo en el que se muestre cómo mostrar la palabra “HOLA” en la celda A1 del Excel al pulsar sobre el botón.
4- Grabar un vídeo en el que se muestre cómo mostrar una acumulación de “HOLA”s seguidos en la celda A1 del Excel al pulsar sobre el botón.
5- Grabar un vídeo en el que se muestre cómo mostrar una acumulación de “HOLA”s en varias diagonales seguidas.

6- Crear una macro en la que apretando un botón, aparezca un formulario e introduciendo un número, muestre si éste es par o impar.
7- Crear una macro en la que apretando un botón, aparezca un formulario que permita introducir 3 valores en distintas celdas del formulario, muestre en el mismo formulario, cuál es el valor máximo y cuál el valor mínimo.
8- Crear una macro en la que apretando un botón, aparezca un formulario en el que se puedan introducir dos números y escoger la operación aritmética a realizar con ellos dos (suma, resta, multiplicación o división) y muestre por pantalla en el mismo formulario, el resultado de la operación escogida.
9- Crear una macro en la que apretando un botón, se pregunte por pantalla cuántos números de la Serie de Fibonacci se quieren mostrar, y tras escribirlo, se muestren escritos en la primera columna de la Excel tantos números (empezando en el 0) como se hayan estimado.
10- Crear una macro en la que apretando un botón, se muestre un formulario en el que se solicite la altura y diámetro de un cilindro y muestre en el mismo formulario como resultado la superficie y su volumen.
11- Crear una macro en la que apretando un botón, se muestre por pantalla un formulario que pregunte número de Turismos, número de Todoterreno, capacidad de combustible de los Turismos y capacidad de combustible de los Todoterrenos y muestre por pantalla los requisitos totales de combustible necesarias en total.
12- Crear una macro en la que escribiendo una serie de números introducidos por teclado en la primera columna de la Excel, y tras apretar un botón, aparezca en dos celdas distintas de la Excel la Suma de los números introducidos y la Media de ellos.
13- Igual que el anterior, pero además de dar el resultado, muestre al poner el ratón encima del resultado, la fórmula empleada para calcularlas.
14- Crear una macro en la que apretando un botón se muestre un formulario para que introduciendo un número por teclado, y después de pulsar un botón, transforme el número introducido de grados Celsius a Farenheit y viceversa.
15- Crear una macro en la que apretando un botón se muestre un formulario solicitando un número de días, y tras introducirlo, muestre en la pantalla de la Excel su equivalente en Horas, Minutos y Segundos. Escribiendo encima de estos si son Horas, Minutos o Segundos.
16- Crear una macro que pregunte en un formulario un número de Segundos a introducir y tras introducirlo muestre en la pantalla de la Excel la equivalencia de esos Segundos en Días, Horas, Minutos y Segundos como números ENTEROS sin que aparezcan decimales. Escribiendo encima de estos si son Días, Horas, Minutos o Segundos.
17- Crear una macro que tras introducir 10 números en la primera columna de la Hoja Excel, los pegue en la segunda columna en orden descendente.