Primera edición

Excel VBA
Guía Rápida
para principiantes

Aprende a crear proyectos VBA compatibles que se ejecutan en Windows y en Mac OS X

Excel y Más por Otto J Gonzalez

Acerca del Autor

Mi nombre es Otto Javier González, técnico programador de Sistemas informáticos.
Mi trabajo se desarrolla dando soporte técnico a la pequeña y mediana empresa.

En mi experiencia laboral, siempre ha estado presente Excel. Es como la matemática, que está presente en nuestro diario vivir. Yo soy de la opinión, que si la tecnología nos da las herramientas necesarias para optimizar nuestro trabajo… pues hay que aprovecharlas al máximo.

OK... Muchos se preguntarán ¿Qué es Excel?

Excel es una Aplicación que nos permite administrar, calcular y analizar datos. Forma parte de Microsoft Office, un conjunto de productos que combina varios tipos de aplicaciones, que nos ayudan a crear documentos, presentaciones, hojas de cálculo, bases de datos, administrar correo electrónico, entre otras cosas.

La aplicación Microsoft Excel, es una herramienta que muchas veces ha sido subestimada por los usuarios, desconociendo su potencial real y no saben que podrían hacer, desde la tarea más sencilla, hasta los trabajos más complejos e inimaginables.

Conocer el funcionamiento de esta aplicación, es fundamental para cada usuario, ya que hoy en día, la pequeña, mediana y gran empresa, cuentan con esta herramienta de Office. Cabe mencionar también, que los que conocen su potencial, no invierten en otros programas para administrar sus datos, porque saben lo que tienen en sus manos.

Sin embargo no basta solo tener instalado este software, ya que si no sabemos cómo utilizarlo, no podremos sacarle todo el funcionamiento que nos puede llegar a ofrecer.

¡Entrénate conmigo, mejora tus habilidades y sácale provecho a esta útil herramienta!

Otto J Gonzalez
www.excelymas.com
El Autor

© Excel y Más por Otto J González

El objetivo de esta publicación es, dar una **Guía Rápida** para quienes desean aprender a programar en VBA para Excel, **obteniendo las bases necesarias** de una forma práctica y sobre todo inmediata, **en cuatro sencillos capítulos**.

Soy MVP de Microsoft desde el año 2015, en la categoría Office Apps & Services. Puedes visitarme en https://www.youtube.com/ottojaviergonzalez en donde encontrarás contenido audiovisual totalmente gratuito.

"El ***MVP de Microsoft*** es un galardón que se otorga a todas aquellas personas que con sus contribuciones ayudan y colaboran activamente en las comunidades técnicas del mundo sobre herramientas de ***Microsoft***

Pasos iniciales

1

Si eres un entusiasta y un amante de Microsoft Excel, pero que únicamente te has dedicado a utilizar la herramienta de una manera poco productiva, déjame decirte que no necesitas tener amplios conocimientos de programación para automatizar Excel según tus necesidades.

Debes tomar en cuenta que Excel ya incorpora de forma nativa Visual Basic orientado a todas las aplicaciones que conforman la Suite Ofimática de Microsoft.

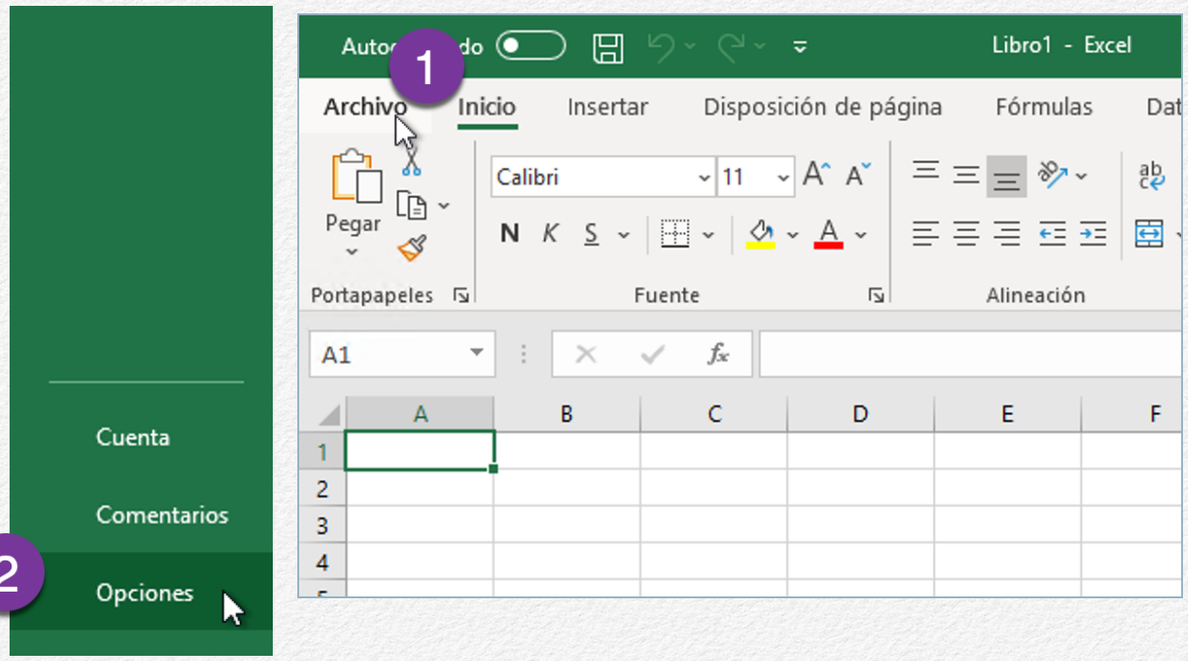
Partiendo de ahí, ya tienes una herramienta muy importante e imprescindible para alguien que desea ser más productivo en su entorno laboral.

Activar la pestaña Programador

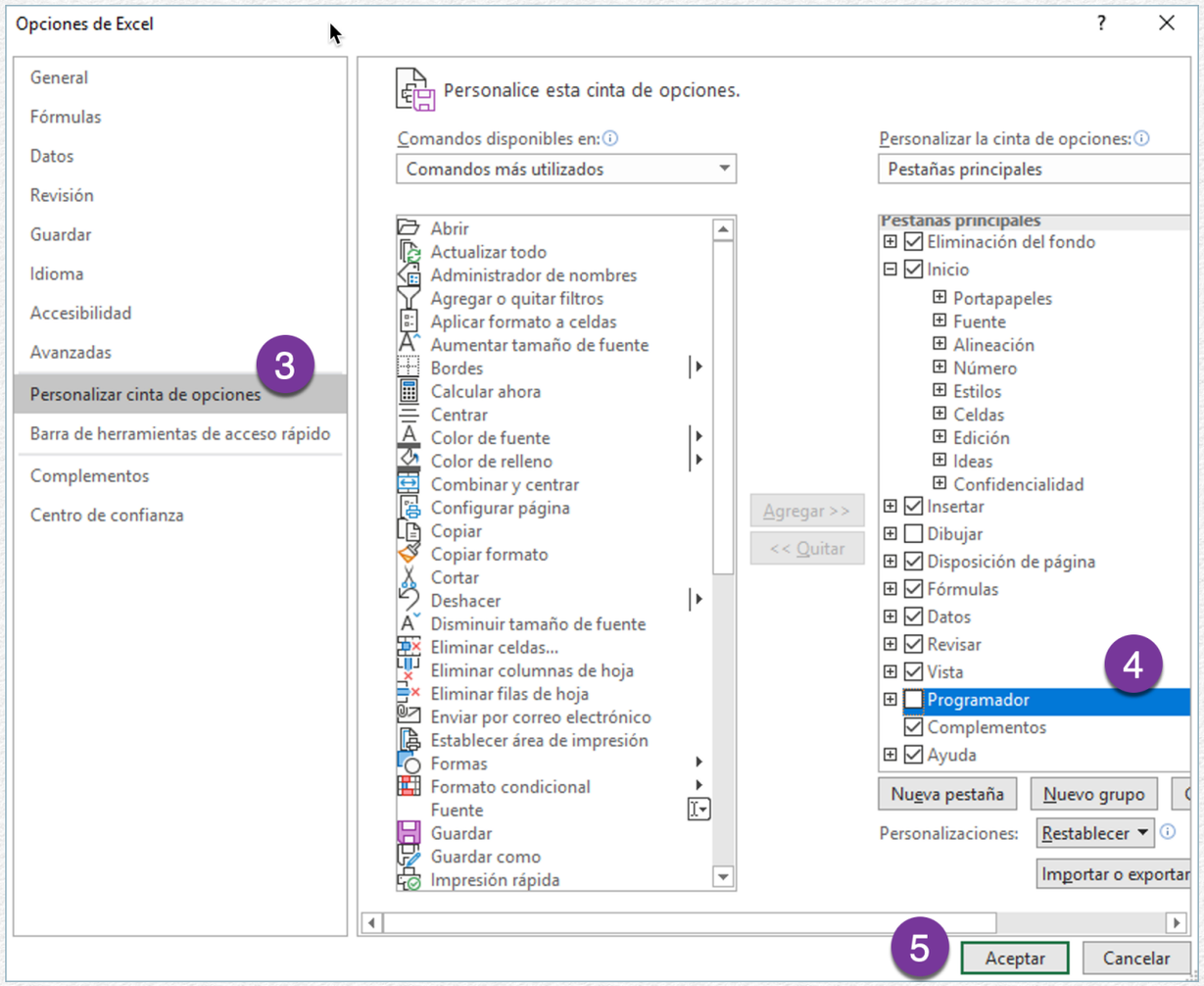
Lo primero que debes hacer es activar la pestaña Programador en la cinta de Opciones

Si eres usuario de Windows

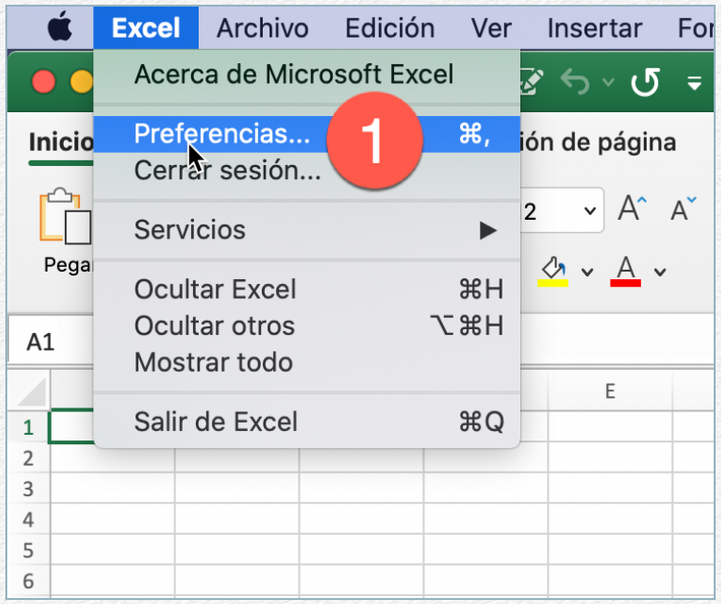
Te diriges a *Archivo* > *Opciones*



Luego te aparecerá la siguiente ventana en donde seleccionas *Personalizar cinta de opciones* y marcas la casilla de verificación correspondiente a la pestaña *Programador* y finalizas dando clic en Aceptar



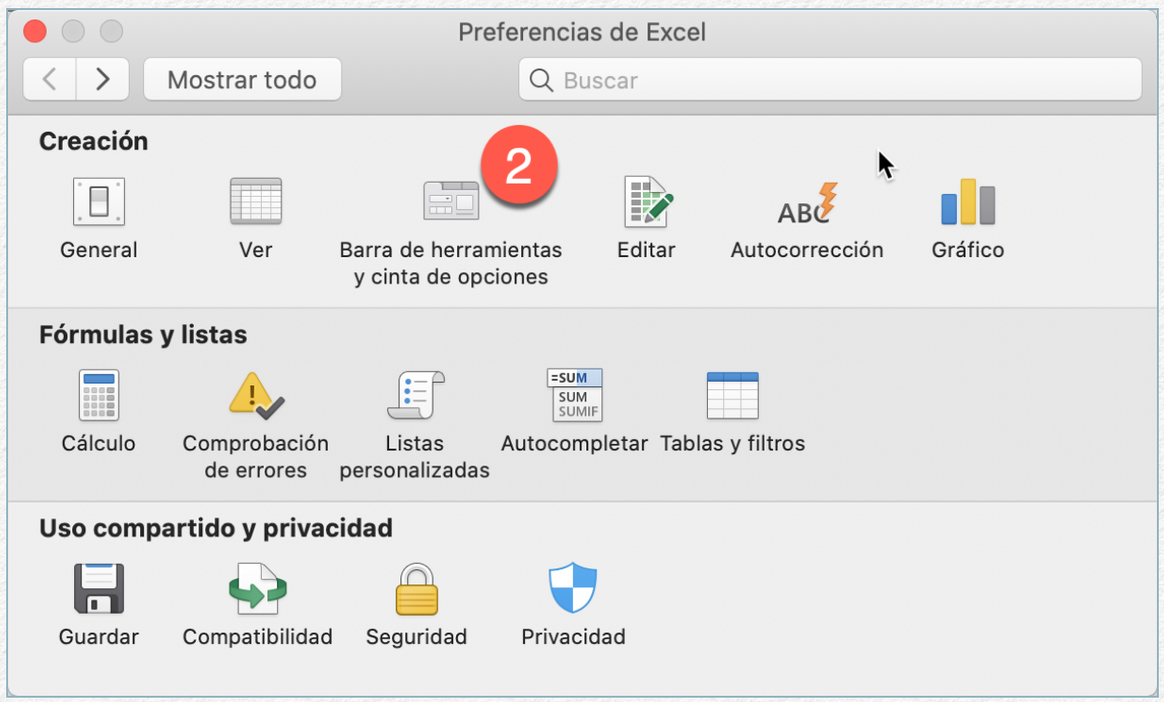
Si eres usuario de Mac OS



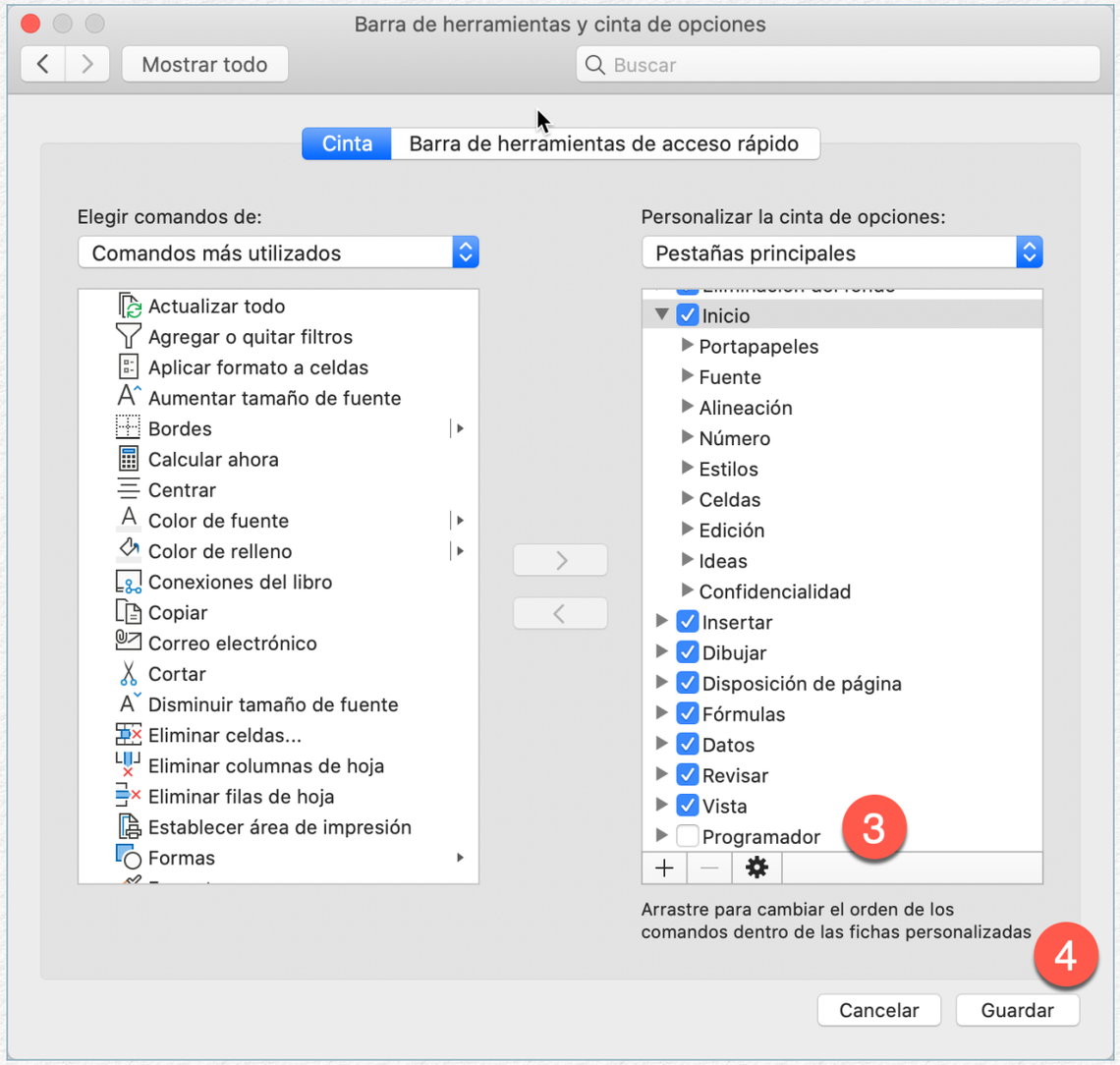
Te diriges a *Preferencias*

Te aparecerá la siguiente ventana

Seleccionas *Barra de Herramientas y cinta de opciones*



En la siguiente ventana encontraremos la opción para activar la pestaña *Programador*. Y finalizamos dando click en *Guardar*

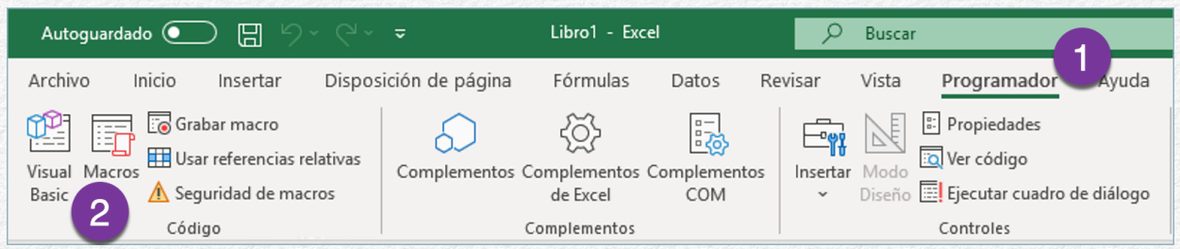


Conociendo el entorno de VBA

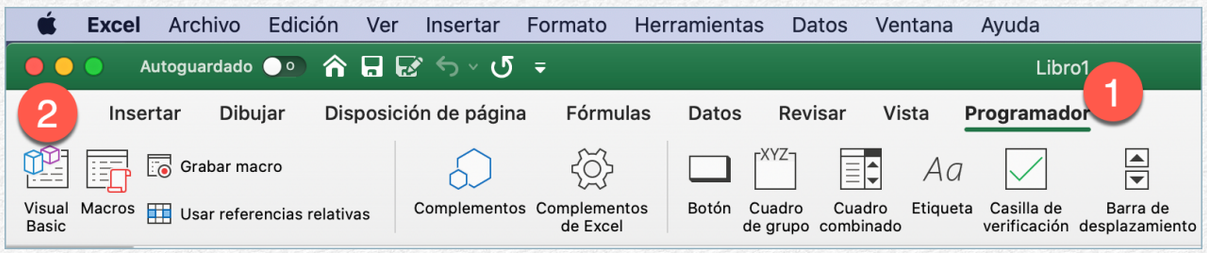
Si eres usuario de Windows o Mac OS

Una vez activa la pestaña *Programador (1)*, en el extremo izquierdo encontrarás el botón que te permite ir al *Editor de Visual Basic para Aplicaciones (2)*

En Windows



En Mac OS



El entorno de Visual Basic o VBA es muy simple y fácil de entender cada una de las herramientas.

En la parte superior izquierda del Editor, encontrarás el Explorador de proyectos(1), y en la parte inferior, la Ventana de Propiedades (2)

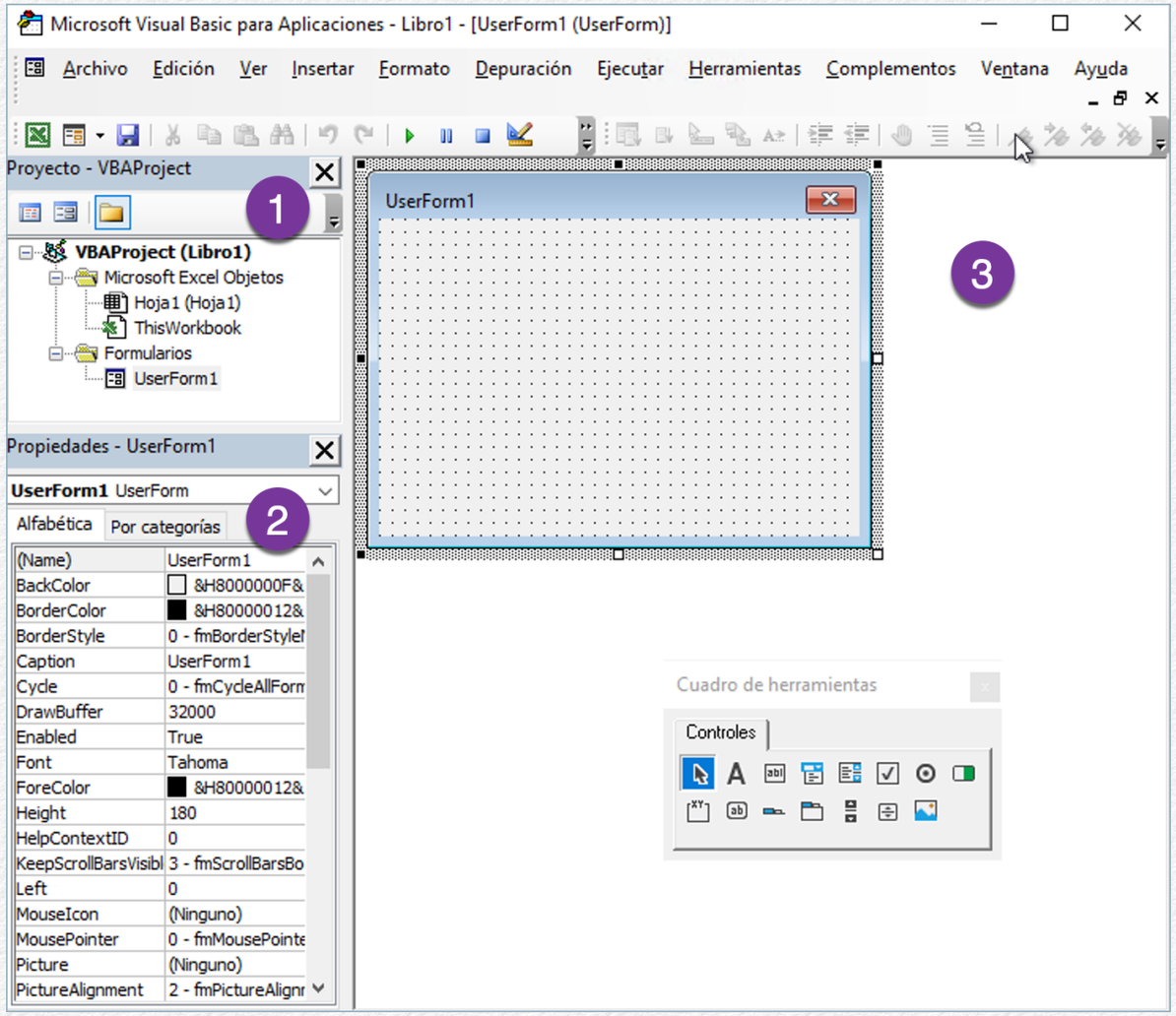
> Dato interesante
>
> Visual Basic para Aplicaciones fue implementado por primera vez en 1993 en la versión 5.0 de Excel, en Sistema Operativo Windows

El área de trabajo(3) la tienes a tu derecha, que cuando agregues objetos, ya sean Formularios de Usuarios, Módulos Estándar o Módulos de Clase, ahí será en donde visualizarás el código de programación o el diseño de los Formularios.

Si bien es cierto el entorno de VBA es muy parecido en Windows y en MacOS, sin embargo, en Sistemas Mac, en la última versión de Excel, ya no es posible agregar Formularios de Usuarios como interfaz de entrada.

Pero más adelante les mostraré una alternativa la cual nos permitirá agregar Formularios de usuario en la versión MacOS

Si no tienes ningún conocimiento de programación previo, no te preocupes. Únicamente toma en consideración algunos aspec-



tos y fundamentos para aprender a programar en Visual Basic para aplicaciones y verás que no es difícil.

Digamos que esta guía viene siendo la receta perfecta para alguien que desea iniciar en la programación y automatización de Excel en Visual Basic para Aplicaciones.

¿Y cuáles son los aspectos y fundamentos a los que me refiero?

Pues, sencillamente son los siguientes:

Formularios de Usuario o UserForms

Nos permiten crear nuestros propios diseños para automatizar las entradas de datos en una hoja de cálculo

- *Los Objetos de Excel*
- *Declaración de Variables*
- *Tipos de Datos*
- *Asignación de Datos a las Variables*
- *Ambitos de las Variables (Su alcance dentro del proyectoVBA*
- *Procedimientos*

Aspectos y Fundamentos

2

No creas que cada uno de los puntos que mencioné en el capítulo anterior son difíciles de asimilar, ya que una vez vayas conociendo cada objeto, cada tipo de dato, cada ámbito, verás que la automatización en Excel es muy divertida.

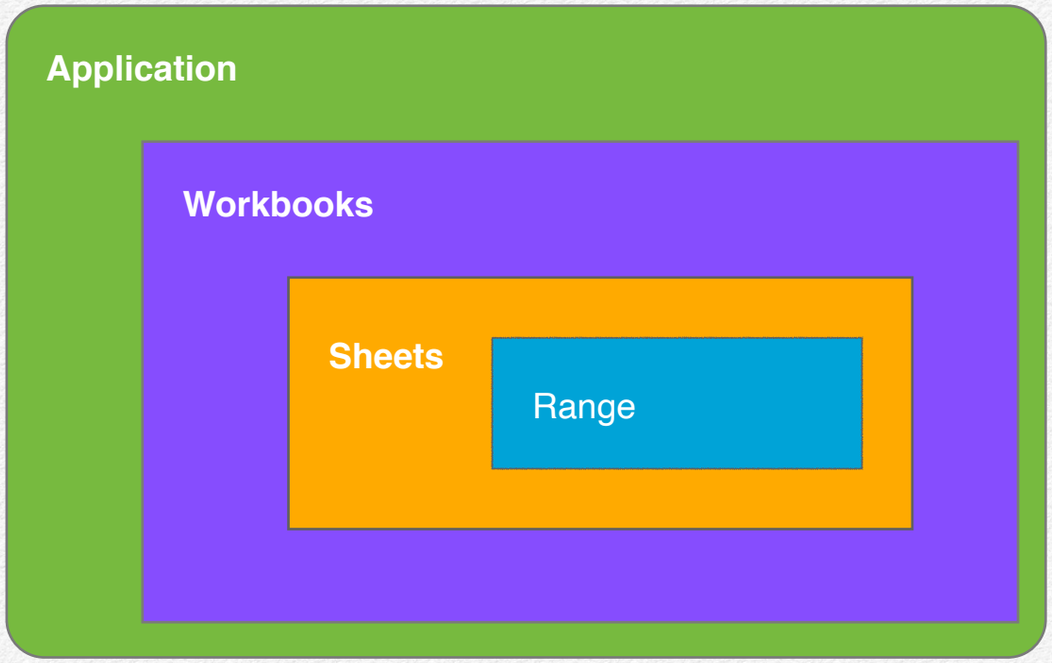
Los Objetos de Excel

Antes de iniciar una automatización en VBA, tendrás que conocer cuáles son los Objetos de Excel y de esa forma poder establecer referencias a dichos Objetos mediante código de programación VBA.

¿Y cuáles son los Objetos de Excel?

- ***Application***
  - *La aplicación o instancia principal de Excel*
- ***Workbooks***
  - *Son los Libros de Trabajo dentro de la aplicación*
- ***Sheet***
  - *Son las Hojas de Cálculo dentro de un Libro de Trabajo*
- ***Range***
  - *Hace referencia a las celdas que contiene una Hoja de Cálculo*

Para una mejor comprensión, observa la siguiente imagen, que te dará una mejor idea del contexto de los Objetos de Excel



Importante

En programación cada una de las líneas de código y objetos relacionados, se establecen en inglés

Hay más objetos de Excel si hablamos del entorno VBA, ya que los objetos anteriormente mencionados, hacen referencia a la interfaz de usuario. Que para automatizar una determinada tarea, es muy importante entender cómo funciona cada uno de ellos.

Declaración de Variables

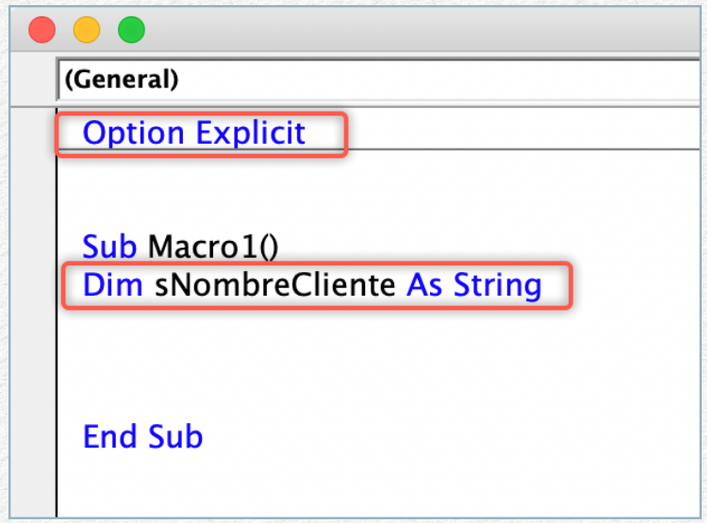
Antes que nada ¿Qué es una Variable?

En términos prácticos y sin tanto tecnicismo, una variable es un espacio en memoria que reservamos mediante la declaración de un nombre, asignándole el tipo de dato que vamos a almacenar. En la mayoría de los casos las variables no son declaradas por los desarrolladores, lo cual permite al sistema asignarle por defecto, un tipo de dato del tipo *Variant*

> **Toma Nota**
>
> Para dar seguimiento a las variables declaradas, es recomendable colocarle un prefijo descriptivo que haga referencia al tipo de dato establecido

Es muy importante utilizar la sentencia *Option Explicit* para que el sistema nos obligue a declarar las variables y que les asignemos el tipo de dato que corresponde La palabra clave reservada para declarar una variable es *Dim* seguida del nombre de la variable, luego asignando el tipo de dato que utilicemos, sin olvidar anteponer al tipo de dato la palabra reservada *As*, según el ejemplo

Dim NombreVariable *As* TipoDato



Es de tomar en cuenta que las variables se declaran al principio de un procedimiento. De esa forma, si necesitas cambiar algún tipo de dato, simplemente te diriges al inicio del procedimiento y ahí estarán todas las variables que gestionan el procedimiento *Sub*

Asignación de Tipos de Datos

Dependiendo de las tareas que vamos a automatizar, así será la cantidad de variables a declarar y el tipo de datos a utilizar. Algunas veces tendremos que crear variables con el tipo de datos *Variant*, ya que dicha variable tendría que almacenar cadenas de texto, valores numéricos enteros, largos o tipos de fechas. En esos casos pues declaramos variables de tipo *Variant*

Entre los tipos de datos que podemos establecer, tenemos los siguientes:

Currency, el cual podemos utilizar para los valores de moneda.

Date, para los datos de tipo fecha.

Integer, para los números enteros.

Long, para los valores numéricos de gran tamaño.

String, utilizado para cadenas de texto.

Los tipos de datos antes mencionados son los que con más frecuencia se utilizan, sin embargo, existen una gran cantidad de tipos de datos que podemos establecer.

Es importante conocer y saber diferenciar los tipos de datos numéricos. Si bien es cierto tenemos los tipos numéricos *Integer* y *Long*, sin embargo existen otros tipos numéricos, que dependiendo del tamaño de almacenamiento que necesitemos, así tendremos que implementarlos en nuestros desarrollos.

Checa los siguientes gráficos para que tengas una mejor idea.

Byte	**Integer**	**Long**
0 a 255	-32768 a 32767	-2.147.483.648 a 2.147.483.647
8 Bits	2 Bytes	4 Bytes

El tipo *Byte* lo podemos utilizar para almacenar edades, ya que no llegaríamos a utilizar un espacio tan grande como el tipo de dato *integer* para almacenar una edad y por supuesto, sería contraproducente utilizar el tipo de dato Long para pretender almacenar datos de edades de personas.

Ámbitos de las Variables

Cuando trabajamos con variables debemos tomar en cuenta su alcance dentro de un proyecto, ya que dependiendo el ámbito en el que las declaremos, así tendrá efecto en parte o en el proyecto completo.

Ámbito a nivel de procedimiento: implica que la variable tendrá su alcance en un procedimiento en particular.

```
(General)
Option Explicit


Sub Macro1()
Dim sTexto As String

sTexto = "Hola mundo"

  MsgBox sTexto


End Sub
```

Importante destacar

Las variables se destruyen cuando el procedimiento finaliza. Por lo que después de ejecutada, dicho espacio en memoria se limpiará.

Ámbito a nivel de módulo: En este ámbito podremos declarar variables que alcancen a varios procedimientos que se encuentren en un módulo. De esta forma los valores asignados a dichas variables, tendrán efecto en los procedimientos de donde se les llame, obteniendo los valores almacenados en ellas.

```
(General)
Option Explicit
Dim sTexto As String


Sub Macro1()

sTexto = "Hola mundo"

   MsgBox sTexto

Call Macro2

End Sub

Sub Macro2()

sTexto = "El futuro es hoy"

   MsgBox sTexto

End Sub
```

Toma muy en cuanta

Cuando declaras variable a nivel de módulo, podrás cambiar la asignación de sus valores en tiempo de ejecución desde cualquier otro procedimiento, obteniendo resultados diferentes para cada uno de ellos

Ámbito a nivel de proyecto: Dichas variables pueden declararse en un módulo, pero si las declaramos anteponiendo la palabra clave reservada *Public*, haremos que tengan un alcance global, es decir, a nivel de todo el proyecto.

De tal manera que no importa si llamamos dichas variables desde un procedimiento ubicado en otro módulo o en un UserForm, de forma inmediata obtendremos los valores almacenados, que si aún no tienen asignados valores, pues podremos asignarlos desde cualquier punto del proyecto también.

```
(General)
Option Explicit
Public sTexto As String

Sub Macro1()

sTexto = "Hola mundo"

    MsgBox sTexto

Call Macro2

End Sub

Sub Macro2()

sTexto = "El futuro es hoy"

    MsgBox sTexto

End Sub
```

Toma nota

observa que prácticamente es un escenario parecido al anterior. Sin embargo debes notar que la declaración la hacemos de forma pública para que su alcance tenga efecto en todo el proyecto, pudiendo hacer la llamada a dicha variable desde cualquier otra ubicación, ya sea UserForm o un módulo distinto.

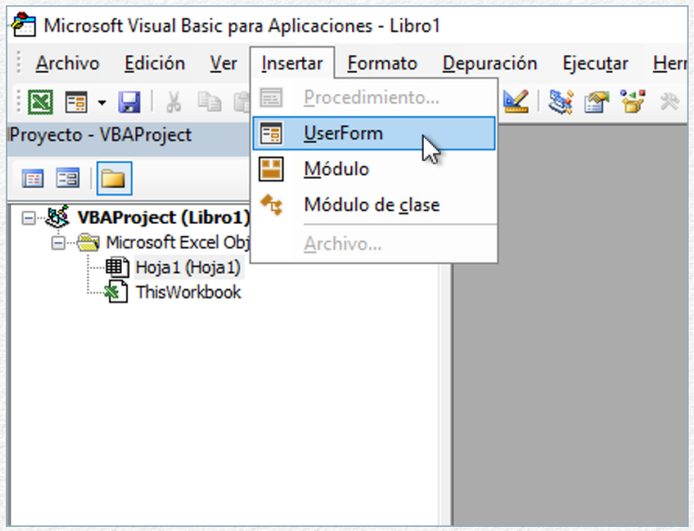
Ejercicio Práctico

Hagamos un ejercicio práctico, insertando y automatizando un UserForm (Formulario de Usuario), lo cual nos permitirá enviar datos a una hoja de cálculo.

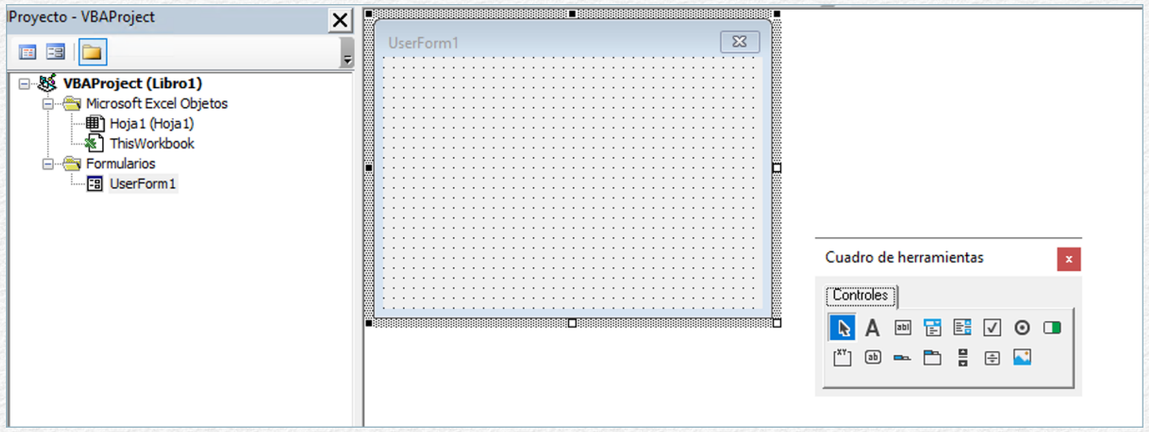
3

UserForm en Windows

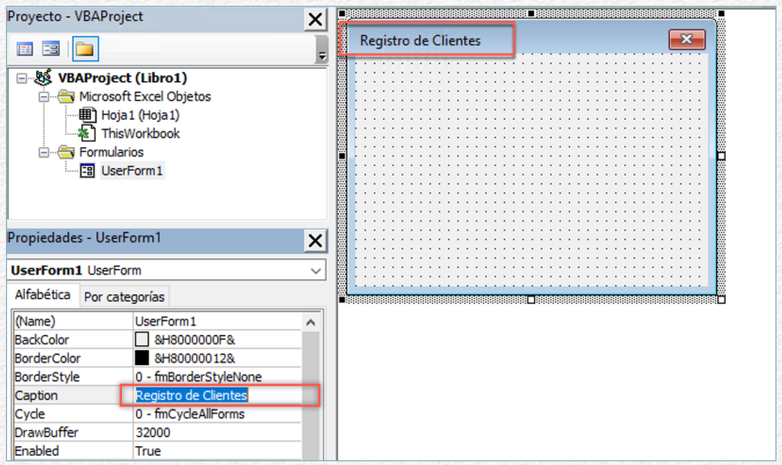
Insertando un UserForm



En el editor de Visual Basic, damos click en insertar y seleccionamos la opción *UserForm*, y el formulario se mostrará en modo de diseño para que le podamos insertar los objetos que necesitemos para nuestra automatización, entre los cuales tenemos cuadros de texto, cuadros combinados, cuadros de lista y botones de comando. Hay muchísimos objetos para cada necesidad, por lo que nuestros desarrollos tendrán muchas posibilidades.

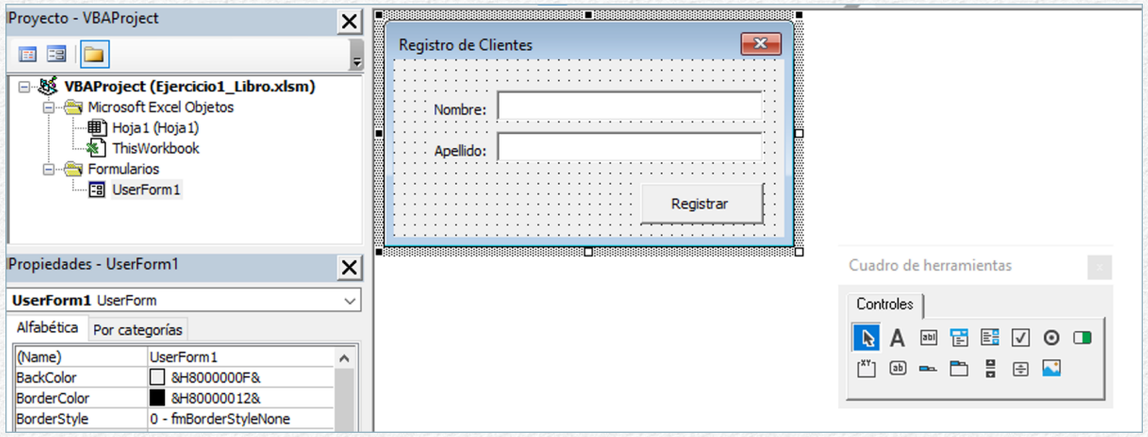


Debes tomar muy en cuenta que cada objeto o control que agregues, tendrás la posibilidad de cambiar sus propiedades según lo que necesites.

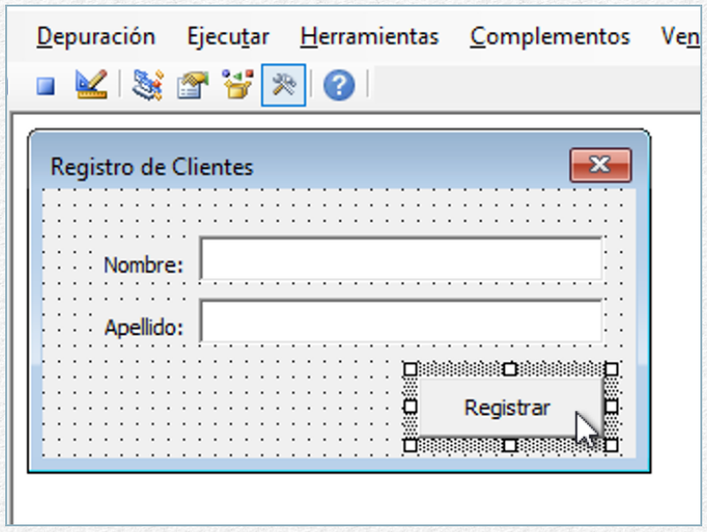


Como por ejemplo cambiar la propiedad *Caption* del *UserForm* para que en su barra de título se muestre “Registro de Clientes”

De esta forma podrás cambiar detalles a la interfaz de usuario que estés desarrollando.



Muy bien, agreguemos un par de cajas de texto, dos etiquetas y un botón de comando con el propósito de enviar datos a una hoja de cálculo.

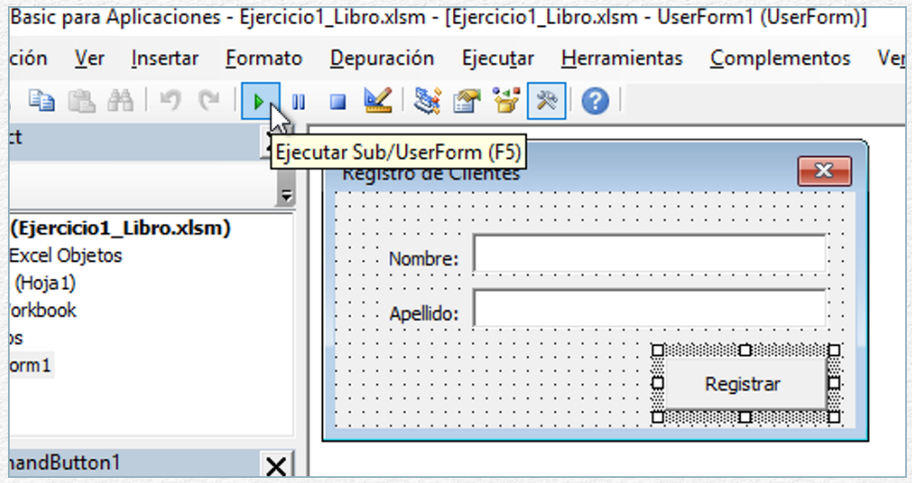


Una vez tengamos nuestro formulario diseñado a nuestro gusto, debemos aplicarle la programación correspondiente al botón de comando, por lo cual daremos doble click sobre él, para ingresar a su evento click.

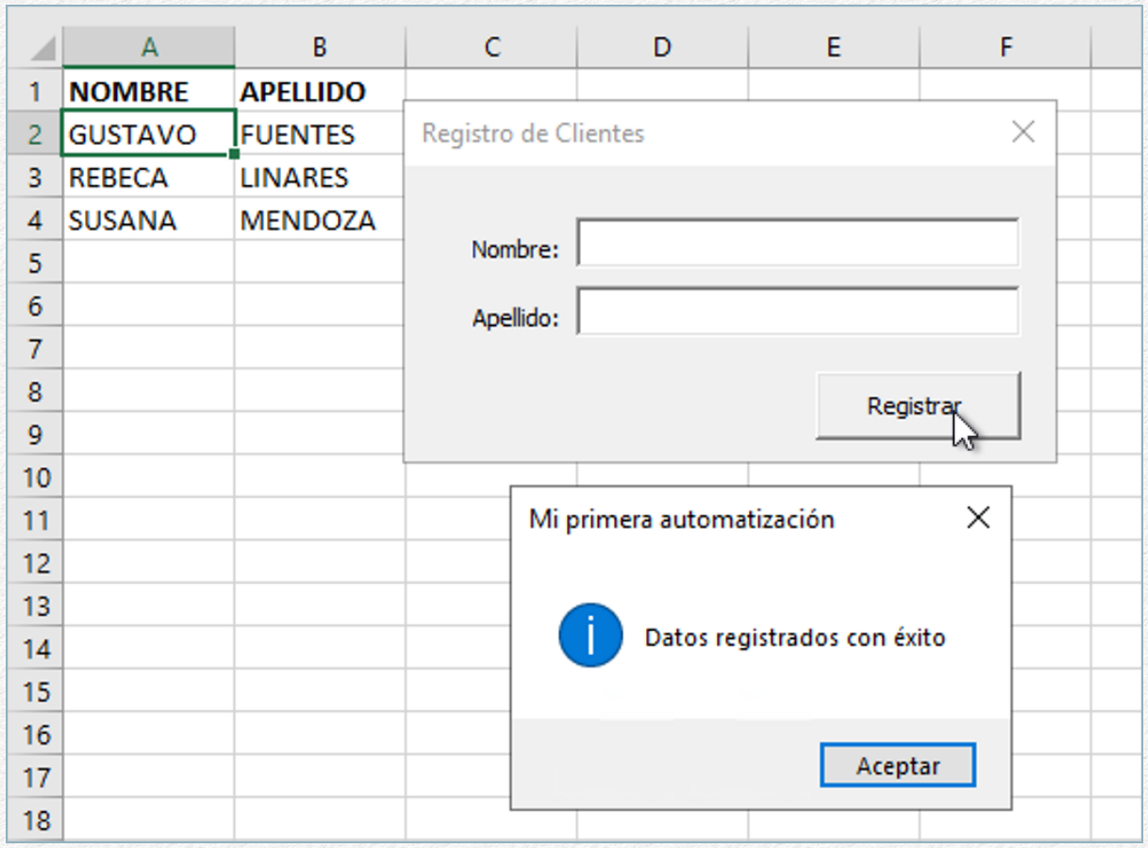
Nuestro código quedaría de la siguiente forma.

```
CommandButton1                                   Click
Private Sub CommandButton1_Click()
Dim sMensaje As String
Dim sTitulo As String
Dim UltFila As Long

UltFila = Hoja1.Range("A1").CurrentRegion.Rows.Count + 1
sMensaje = "Datos registrados con éxito"
sTitulo = "Mi primera automatización"

With Hoja1
    .Cells(UltFila, 1) = Me.TextBox1.Text
    .Cells(UltFila, 2) = Me.TextBox2.Text
End With

    Me.TextBox1.Text = ""
    Me.TextBox2.Text = ""

    Me.TextBox1.SetFocus

MsgBox sMensaje, vbInformation, sTitulo
End Sub
```

Damos click en el comando ejecutar y precedemos a registrar datos



Y ya haz programado tu primer UserForm que inserta datos en la Hoja1 de tu libro de trabajo.



Que después de cada registro, te saldrá una caja de mensaje confirmando que los datos fueros registrados con éxito.

Este código de programación te servirá de base para que puedas realizar tus propias automatizaciones, ya que contiene los elementos claves, como el poder identificar la última fila en los registros para agregar uno nuevo

Voy a hacer una breve explicación del código de programación.

Se han declarado tres variables, de las cuales, dos son del tipo *String* y una del tipo *Long*.

```
Dim sMensaje As String
Dim sTitulo As String
Dim UltFila As Long
```

Con la variable *UltFila* identificamos el número de registros contenidos en el rango de datos, asignándole el conteo de las filas que conforman la región de dicho rango, para lo cual sumamos uno, para ubicarnos en la fila vacía y de esa forma poder escribir un nuevo registro.

```
UltFila = Hoja1.Range("A1").CurrentRegion.Rows.Count + 1
```

A la variable *sMensaje* que es de tipo *String*, se le asigna una cadena de texto para que notifique al usuario final que los datos han sido registrados con éxito.

```
sMensaje = "Datos registrados con éxito"
```

La variable la utilizamos al final del procedimiento, haciendo uso de la función *MsgBox,* la cual pide diferentes argumentos,

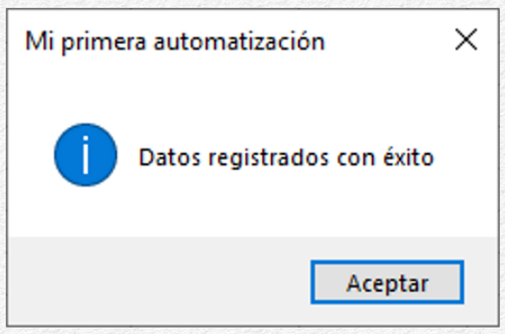
```
MsgBox sMensaje, vbInformation, sTitulo
MsgBox(Prompt, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult
```

entre los cuales, los relevantes son: el *Prompt* o cadena de caracteres que se muestran de manera informativa; tenemos también el estilo de botones e ícono que denota el tipo de información a notificar; y el título, lo cual aparece en la parte superior del mensaje

```
sTitulo = "Mi primera automatización"
```

A la variable sTitulo que también es del tipo *string*, se le ha asignado el título que llevará la ventana del *Msgbox*

Pudimos haber puesto directamente las cadenas de texto en el *MsgBox*, sin embargo, lo he realizado así con propósitos de aprendizaje y que veas cómo se declaran las variables, y se les asignan datos para luego mostrar sus resultados.



Generalmente yo en lo personal, cuando creo proyectos grandes y manejo muchas cajas de mensaje, lo que hago es declarar las variables correspondientes al mensaje y al título de forma pública, para hacer la llamada desde cualquier ubicación del proyecto, y simplemente obtener su resultado con la función *MsgBox*.

Con este ejemplo tendrás una noción más clara y precisa de cómo gestionar información entre un UserForm y la hoja de cálculo.

Y ahora surge la gran pregunta...

¿Cómo podemos insertar *UserForms* en un Libro de Excel sobre un Sistema Operativo Mac?

Compatibilidad Windows y Mac OS

4

En este capítulo aprenderemos a crear proyectos que trabajen de forma compatible para Windows y Mac

Insertar un UserForm

Compatibilidad para Win OS y Mac OS

Los UserForm o formularios de usuario nos permiten crear una interfaz de entrada para que el usuario final pueda interactuar entre el formulario y la hoja de cálculo. Como mencioné anteriormente en la versión actual de Office para Mac, no tiene soporte para insertar formularios de usuario en VBA, esto quedó vigente únicamente hasta la versión 11 de la suite ofimática para Mac.

Sin embargo para los usuarios de Mac, deberán tener acceso a un PC con Windows o una máquina virtual para poder crear desarrollos que incluyan formularios en la versión de Office para Mac, de lo contrario tendrán que limitarse a realizar automatizaciones únicamente a nivel de módulos.

Para crear un proyecto que incluya formularios en Mac, tendrán que crear el proyecto desde Windows, realizando ciertas implementaciones y preparando los formularios para que sean compatibles con un sistema operativo Mac. Yo se que eso implicará la necesidad de contar con más recursos, pero por el momento es la única alternativa.

Al momento de insertar *UserForms* en Windows y cuando ejecutamos dichos *UserForms* en un sistema Mac, el problema más notable es, la diferencia de resolución de pantalla entre las dos plataformas. En Mac son 96 ppp, donde cada punto representa un pixel, mientras que en Windows, es de 72 ppp(puntos por pulgada) y cada punto representa un “punto”.

Si no resolvemos y corregimos las diferencias de unidad de medida, cuando ejecutemos el *UserForm* en Mac, resultará un formulario más pequeño, por lo que su lectura será difícil para el usuario final.

Jon Peltier nos resuelve este problema de una forma muy práctica y precisa, mediante la inserción de cierto código de programación que cambia el tamaño de los formularios en Mac en concordancia con las medidas en un sistema operativo Windows.

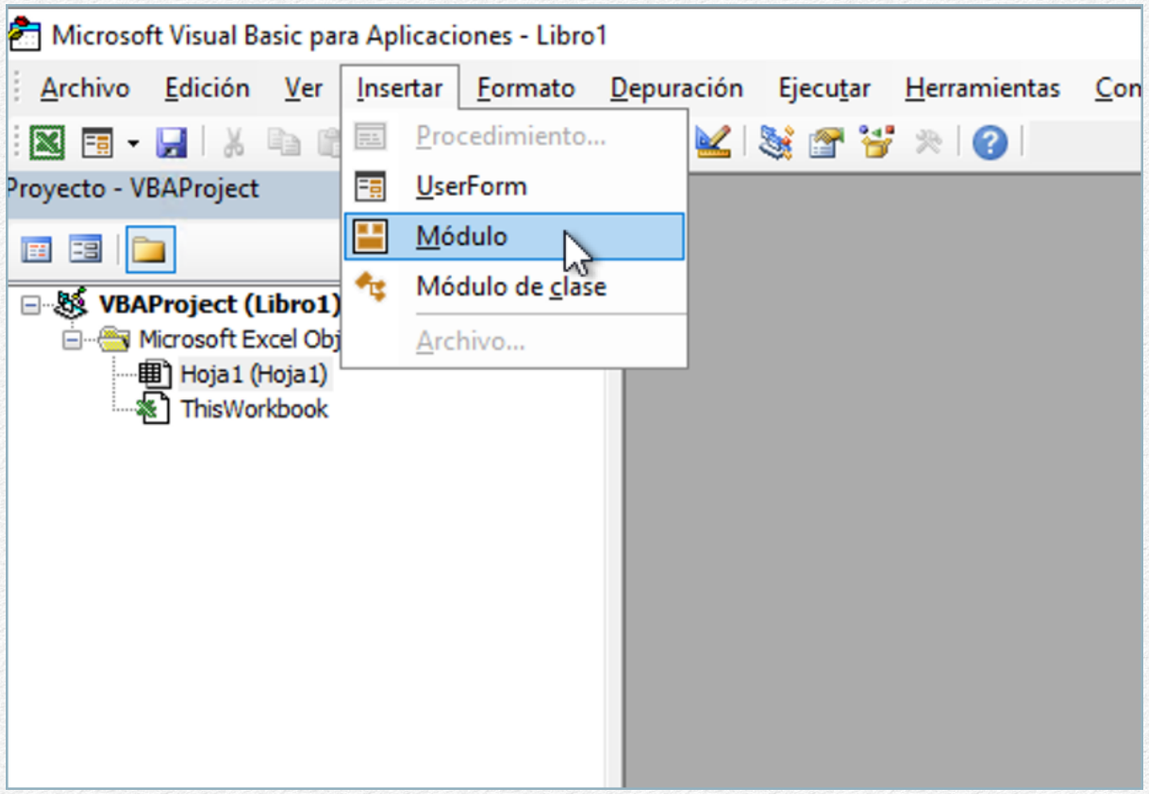
Aquí te dejo el enlace de su publicación original

https://peltiertech.com/userforms-for-mac-and-windows

> “While Microsoft has substantially improved the VB editor on the Mac, you still can’t work with UserForms on the Mac. You have to build them into your file in Windows and them move the file to the Mac.”
>
> Jon Peltier

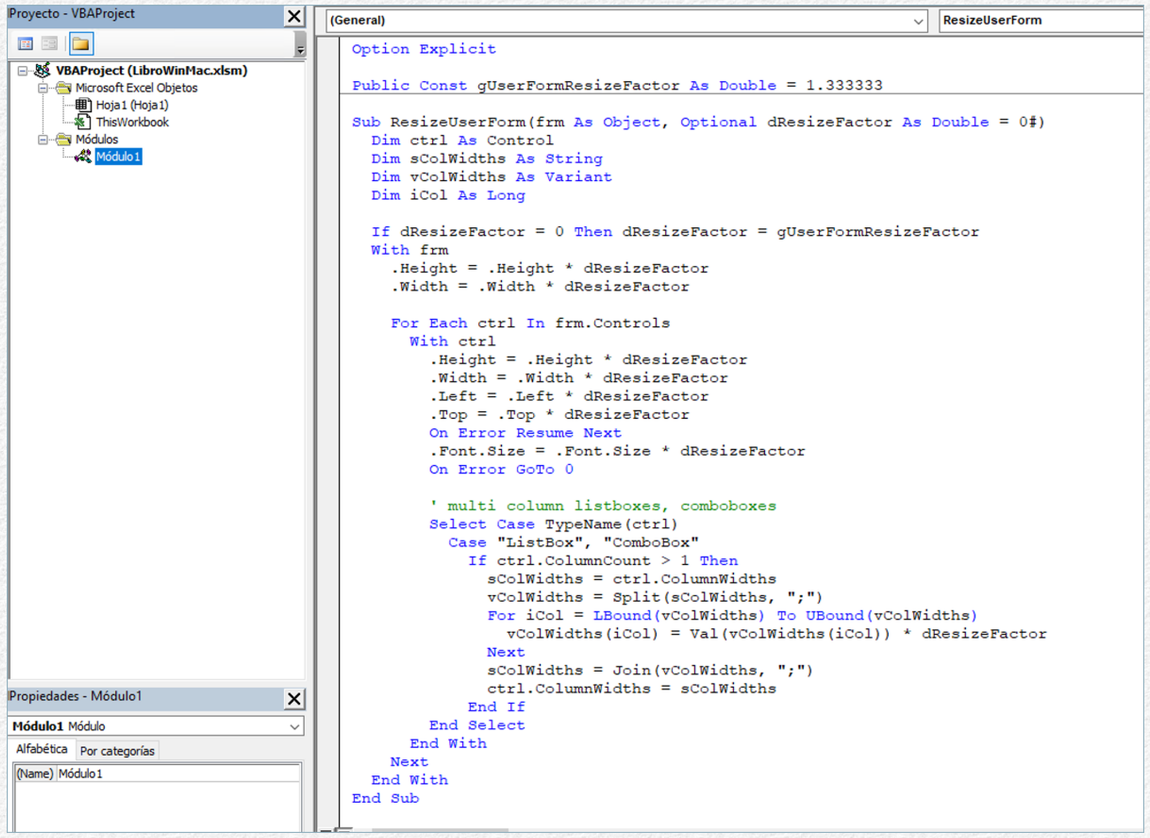
A continuación hagamos un ejemplo según la solución que nos ofrece Jon Peltier.

En Windows insertaré un *UserForm*, pero antes agreguemos un módulo estándar en el cual colocaremos el código fuente que nos proporciona Jon Peltier.



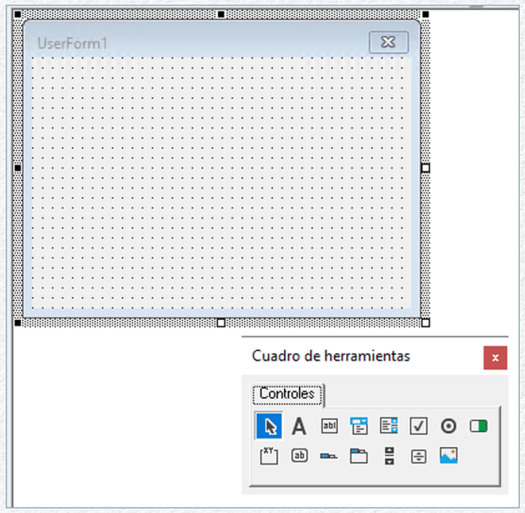
Una vez tengamos nuestro módulo agregado, podríamos cambiarle de nombre, pero yo lo dejaré con el nombre de Módulo1.

Procedo a agregarle el código de programación de Peltier y me quedaría de la siguiente manera.



Este código lo pueden copiar y pegar desde el enlace que les compartí anteriormente.Agreguemos el *UserForm*, el cual llevará las siguientes líneas en su evento *Initialize*

```
Private Sub UserForm_Initialize()
  #If Mac Then
    ResizeUserForm Me
  #End If
End Sub
```

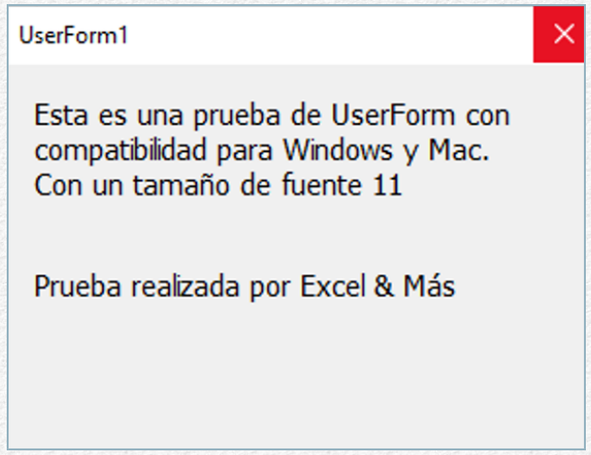


Ingresamos al evento Initialize y quedaría de la siguiente manera.

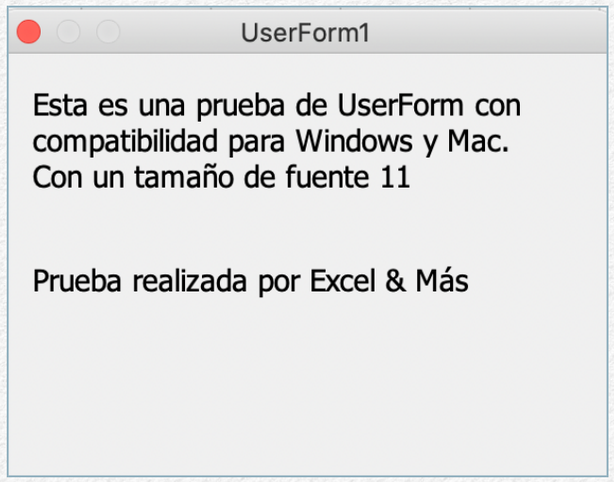
```
UserForm
Option Explicit

Private Sub UserForm_Initialize()
  #If Mac Then
    ResizeUserForm Me
  #End If
End Sub
```

En esta condición *If*, establecemos que, si el Libro de trabajo es abierto en un Mac, que haga la llamada al procedimiento que cambia el tamaño de un *UserForm*, en donde puedes notar que le pasamos como argumento el nombre del UserForm que estamos lanzando, en su defecto *Me*. También es de notar que dicho procedimiento contempla un argumento opcional que permite establecer el porcentaje del tamaño del formulario, sin embargo prefiero dejarlo por defecto al porcentaje seleccionado por Peltier, que eso trabaja muy bien.

Que si abres el libro de trabajo en Windows, como en el ejemplo de Peltier, obviamente la condición *If* será ignorada para mostrar el formulario en su tamaño original.



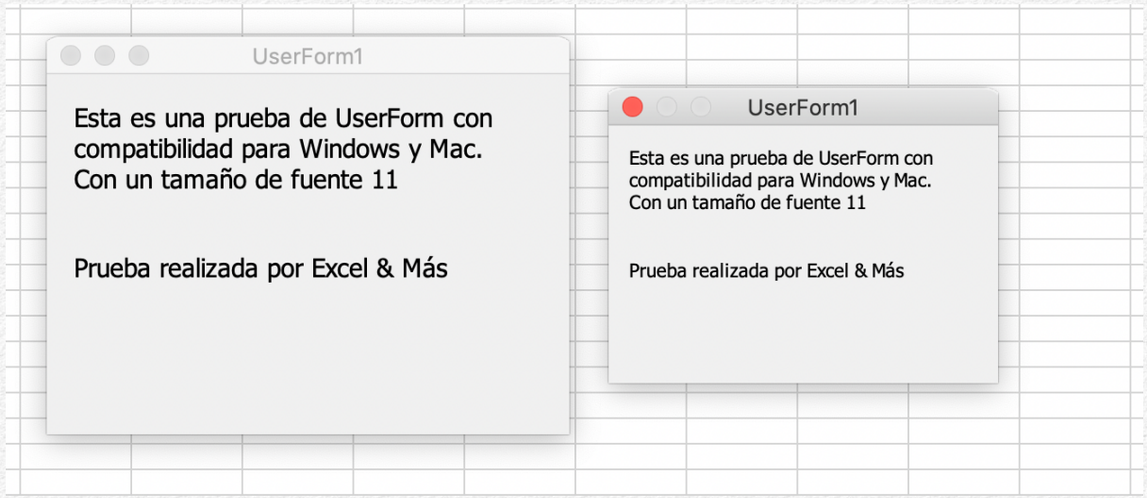
Observa cómo se lanza en un Mac



¿Qué es lo que hace el procedimiento de Peltier?, pues ajustar los valores de medidas que les explicaba anteriormente, para que cuando se abra en un Mac, el texto y formulario se ajuste

al tamaño según las medidas que podríamos tener en Windows, dando un tamaño proporcional en el Mac, para que el *UserForm* sea legible a la vista del usuario final. Pueden leer el artículo original donde Jon Peltier lo explica de mejor manera :)

Si no tuviéramos la solución de Peltier, pues el UserForm en los Mac se vería así



Hago la comparación con el *UserForm* que sí tiene aplicada la solución de Peltier (izquierda) con el *UserForm* que no cuenta con el ajuste de tamaño (derecha), que como puedes observar el *UserForm* es muy pequeño. El *UserForm* aunque pequeño, veo que sí es legible, pero al momento de lanzarlo, no se ve en concordancia con la escala de las ventanas que manejamos en Mac, lo cual puede provocar una sensación de desajuste.

En lo personal estoy muy agradecido con la solución de Jon Peltier, ya que me ha sido de muchísima utilidad en mis desarrollos que distribuyo en Mac.

Comentario Final

Hay que tomar muy en cuenta que la limitante principal para automatizar *UserForms* en un Mac, es precisamente eso, la carencia de la posibilidad de insertar *UserForms* directamente en el Editor VBA de un Mac, y es por ello que debemos hacer uso de estas alternativas, desarrollando antes nuestros proyectos desde Windows, pensando en la posibilidad que puedan abrirse en un Mac. Además, es de considerar que algunos elementos que insertemos en Windows puede que no sean compatibles en Mac, como la inserción de botones ActiveX que se insertan en la interfaz de usuario en Windows, que en Mac no serán compatibles, por lo que dichos botones tendrán que insertarse desde el propio Mac. Y de esta forma iremos creando y equilibrando un proyecto compatible para ambos sistemas operativos.

Muchas gracias por haberme acompañado en esta lectura, nos veremos en una próxima publicación.

Otto J González
El Autor