INTRODUCCIÓN

QUÉ ES UNA MACRO DE EXCEL

Una macro puede definirse como una serie de instrucciones o comandos que se utilizan para automatizar tareas repetitivas o recurrentes. Esta automatización se puede realizar a través la introducción manual usando un editor de texto o a través de la grabación de un conjunto de acciones que, más tarde, podrán ser o no reproducidas a petición del usuario.

Las macros, también nos ofrecen la posibilidad de realizar tareas complejas de una forma más clara, organizada y eficiente. Por ejemplo, podríamos crear una macro para formatear automáticamente una tabla, realizar unos cálculos específicos, generar unos informes personalizados o, incluso, realizar una serie de acciones específicas en función de ciertas situaciones o circunstancias.

Cuando un usuario crea una macro en Excel, en realidad, lo que se está haciendo es crear una secuencia de comandos, fórmulas, formatos u otras operaciones que normalmente se realizarían de manera manual dentro de un módulo de código denominado VBA, acrónimo de Visual Basic for Applications (Visual Basic para Aplicaciones) y que se almacena dentro del propio archivo de Excel que se está utilizando en ese momento.

Es por ello que, si tras la creación de una macro miramos su contenido, lo que veremos es un conjunto de instrucciones escritas mediante un lenguaje de programación, que denominamos de alto nivel, diseñado específicamente para la automatización de tareas en aplicaciones de la suite de Microsoft Office, lo que incluye Microsoft Excel.

Pero, ¿y cómo se activa esta macro que hemos creado? Pues, una vez que hemos creado nuestra macro, la podemos asignar a cualquier acción que pueda ser detectada o usada como disparador. Esto es, la pulsación de un botón, de una combinación de teclas, el cambio de valor de una celda o, incluso, la activación o enfocado de una hoja de cálculo concreta.

A continuación, se muestra el ejemplo de una macro sencilla que se ejecuta de manera automática cuando el valor de la celda A1 cambia:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    ' Verifica si la celda modificada es A1
    If Target.Address = "$A$1" Then
        ' Instrucciones a ejecutar
        MsgBox "El valor de A1 ha cambiado a: " & Target.Value
    End If
End Sub
```

En este ejemplo, la macro se dispara o ejecuta con el evento de cambio **`Worksheet_Change`** que, aunque se verá más adelante, indica al Excel que se ejecute cuando se produzca un cambio en cualquier celda de la hoja de cálculo. Sin embargo, gracias al argumento **`Target`** que recibimos en la llamada del evento, podremos preguntar si es la celda deseada (en este caso A1) mediante el bloque **`If`** declarado a continuación. Este bloque viene a decir que, cuando la propiedad **`Target.Address`** sea la celda **`A1`** ejecute todas las instrucciones que están delimitadas por él.

LA SEGURIDAD Y LAS MACROS

La seguridad en las macros de Excel es un aspecto más que importante debido, fundamentalmente, a que contienen fragmentos o trozos de código ejecutable que pueden utilizarse con fines maliciosos. Por esta razón, las macros de Excel siempre vienen desactivadas, porque, su activación y uso puede representar un riesgo para la seguridad de los datos y/o del sistema.

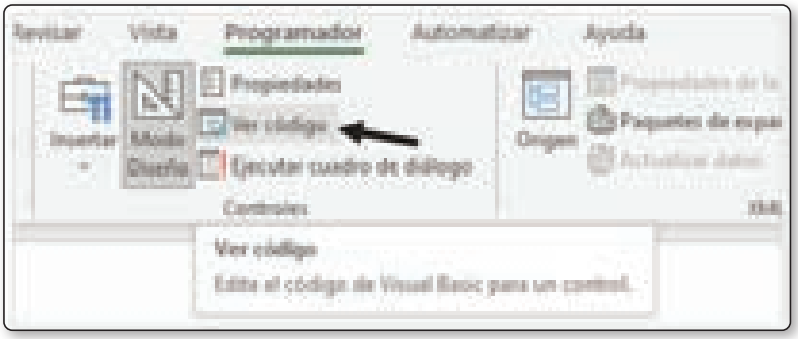
A continuación, se comentan algunas consideraciones para garantizar la seguridad cuando se trabaja con macros en Excel:

- **Advertencia antes de la habilitación de macro**: Excel presenta una configuración de seguridad predeterminada que controla cómo se manejan las macros en los archivos. En general, se puede personalizar en función de nuestras necesidades, no obstante, lo recomendable y más frecuente es dejar que Excel muestre una advertencia antes de habilitar o deshabilitar las macros. De esta manera, tendremos la opción de revisar el código y asegurarnos de que proviene de una fuente confiable antes de ejecutarlo.

- **Fuentes confiables**: siempre se debe estar alerta cuando se vaya a abrir un archivo de Excel que contenga macros. Sin embargo, esta precaución debe incrementarse si procede de una fuente desconocida o no confiable. Esto es, nunca se deben abrir archivos que no permitan una verificación previa de su origen y contenido.

- **Habilitar las macros estrictamente cuando sea necesario**: dado que las macros pueden ser utilizadas como vectores de ataque, lo recomendable es que sólo se habiliten en momentos puntuales, y deshabilitándolas cuando ya no sean necesarias. Además, con ello, reduciremos el riesgo de que se ejecuten de forma accidental o malintencionada.

- **Prevenir antes que curar**: una buena práctica es que, antes de habilitar una macro, revisemos su código. Con ello, no sólo evitaremos la ejecución de instrucciones maliciosas, sino que también podremos evitar acciones inesperadas.



- **Firmas digitales**: aunque no es obligatorio, la recomendación es utilizar firmas digitales. Con ello, no sólo podremos verificar la autenticidad y su origen, sino que también podremos chequear la integridad de las macros ya que, al firmarlas digitalmente, podremos garantizar que no ha sido modificada por terceros y que proviene de una fuente confiable. Para ver las firmas digitales debemos ir a la opción `Archivo > Información`

`> Ver Firmas`. Una vez que estemos ahí, deberemos hacer clic en la `flecha` que apunta hacia `abajo` y seleccionar `Detalles de la Firma`.

- **Actualizaciones y parches al día**: cuando se utiliza un software como Excel, es más que recomendable que esté actualizado con la última versión y con todos los parches de seguridad de Microsoft aplicados. Las actualizaciones periódicas ayudarán a abordar las posibles vulnerabilidades y mejorar la seguridad general del sistema.

VISUAL BASIC Y SU EDITOR DE EXCEL

Visual Basic (VB) es un lenguaje de programación de alto nivel desarrollado por Microsoft y que, habitualmente, forma parte del entorno de desarrollo integrado (IDE) llamado Visual Studio. Visual Basic es un lenguaje de programación que se utiliza comúnmente para desarrollar aplicaciones de software en entornos de Windows y para la automatización de tareas en aplicaciones de Microsoft Office, como Excel, Word o Access.

Entre sus principales características podemos encontrar que:

1. **Resulta fácil de aprender**: esto es debido a que Visual Basic utiliza una sintaxis sencilla y basada en lenguaje natural, lo que lo vuelve más "accesible" para todas aquellas personas que se quieren iniciar en la programación.

2. **Está orientado a eventos**: esto viene a decir que las acciones del usuario, como hacer clic en un botón, lanzan o disparan eventos que pueden ser controlados y respondidos mediante código, lo que hace que el desarrollo de interfaces de usuario interactivas sea más intuitivo.

3. **Integración con aplicaciones de Windows y Office**: esto es evidente, puesto que ya lo hemos comentado. Dado que Visual Basic se integra estrechamente con todas las aplicaciones de Windows y Office, podemos aprovechar todas sus capacidades y características para crear programas, macros de Excel, para desarrollar aplicaciones de Windows Forms, etcétera.

4. **Soporte de la comunidad**: como buen lenguaje vivo, Visual Basic cuenta con una amplia comunidad de desarrolladores y documentación disponible en línea, lo que ayuda y facilita el aprendizaje, la resolución de problemas y el acceso a recursos adicionales.

5. **Programación basada en componentes**: Visual Basic permite la programación basada en componentes utilizando el modelo de desarrollo conocido como Component Object Model (COM), lo que nos permite crear componentes reutilizables y utilizarlos en diferentes aplicaciones o sistemas.

En cuanto al Editor de Visual Basic de Excel, podemos decir que es una herramienta integrada que permite crear, editar y administrar código creado en Visual Basic for Applications (VBA).

Para acceder al Editor de Visual Basic desde Excel, bastará con presionar **ALT + F11** o pulsar en el **icono Visual Basic**, situado como primer elemento a la izquierda, dentro de la opción **Programador**.

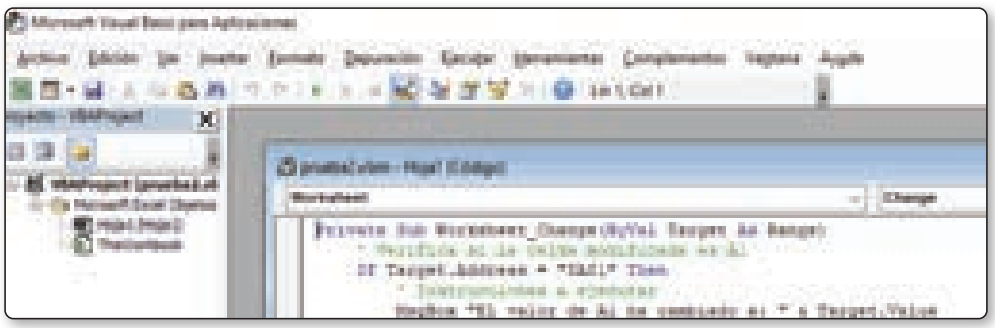
ⓘ NOTA

Para visualizar la pestaña reservada a Visual Basic y Macros, debemos hacer clic en la pestaña **Archivo** y después pulsar en el botón que indica **Opciones**. A continuación, deberemos seleccionar la categoría **Personalizar cinta de opciones** y, en la zona **Pestañas principales**, pulsar la opción de **Desarrollador**. Tras ello, pulsar en el botón de **Aceptar**.

Aunque en la imagen, la opción de "Programador" aparece en segundo lugar, lo normal es que se muestre a continuación de "Automatizar".

Si queremos ver las macros, podemos hacerlo pulsando en el icono que pone "Macros" de esta pestaña o también desde su icono identifico en la pestaña "Vista".

Si pulsamos en el icono de "Visual Basic" o la combinación ALT + F11 deberíamos poder ver algo como:



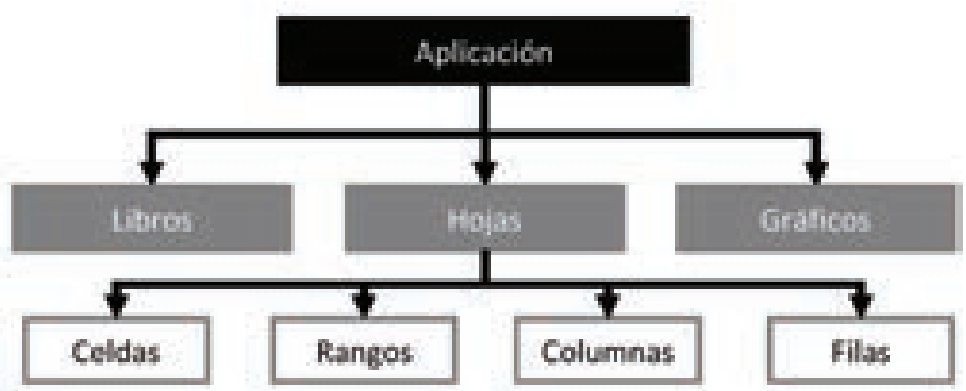
OBJETOS, PROPIEDADES, MÉTODOS Y EVENTOS EN EXCEL

Objetos

Un **objeto** es aquel elemento que representa a una entidad dentro de una aplicación. Esto es, por ejemplo, una hoja de cálculo, una celda, una imagen, un rango, un formulario, etcétera.

Todos los objetos podrán contener otros objetos dentro, no obstante, es frecuente diferenciar este tipo de casos. Mientras que un objeto es una entidad que posee unas determinadas propiedades, métodos y eventos, una colección es un conjunto de objetos que, a su vez, tiene propiedades, métodos y eventos.

Este es el caso del objeto Application, el cual contiene otros objetos como son Workbook, Worksheet, Chart, Range, Column, Row o Cells.



Como uno se podrá imaginar, en Excel existe una jerarquía y, esta jerarquía empieza por el objeto Application. De este objeto dependen, por decirlo así, una multitud de objetos, y cada uno de ellos tiene un propósito. Esto es, Excel posee objetos para manejar los estilos, los nombres definidos para celdillas y rangos, los gráficos, los libros, las hojas de cálculo, los rangos, las ventanas, los complementos y, cómo no, los proyectos de Visual Basic.

Propiedades

Una **propiedad** es una palabra clave que hace referencia a un valor dado y que, por lo general, define una característica determinada. Por ejemplo, si

estuviésemos hablando de una persona, sus propiedades podrían ser el color de sus ojos, el color del pelo, su altura, peso, sexo, número de pie, etcétera.

Cuando hablamos de Excel, sus propiedades pueden ser la dirección o el valor de una celda.

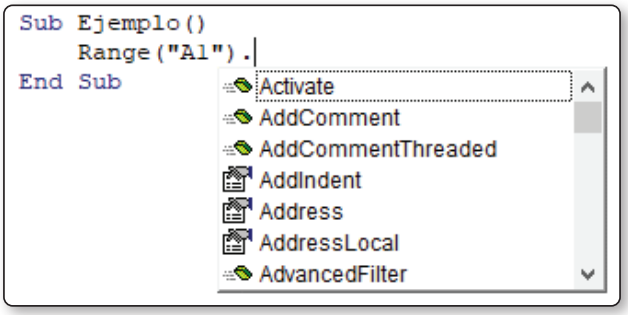
```
Range("A1").Value = "Hola"
```

Métodos

Un **método** es una acción que puede hacer o ejecutar un determinado objeto y que, normalmente, no está vinculado directamente a una petición de usuario. Si esta idea la contextualizamos a Excel, este tipo de acciones podrían ser seleccionar una celda o borrar su contenido:

```
Range("A1").Select
Range("A1").Clear
```

Si se desean consultar o ver las propiedades, métodos y eventos de un objeto determinado bastará con escribir el nombre del objeto y presionar la tecla de punto:



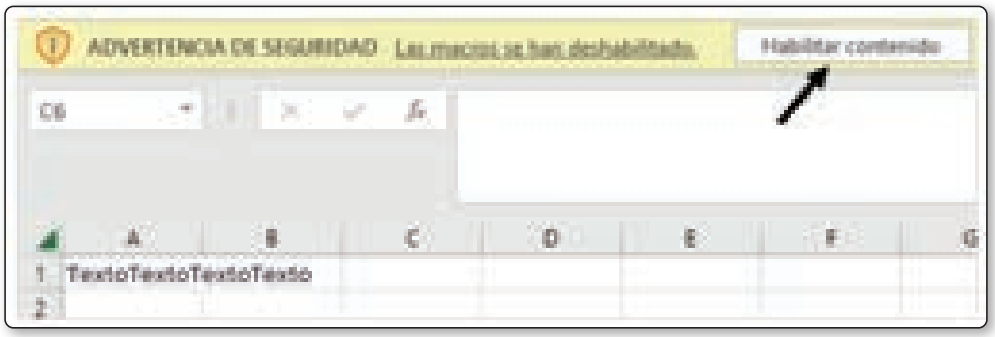
Eventos

Un **evento**, por su parte, es una acción asociada a un determinado suceso que produce, por lo general, el usuario. Los eventos de un objeto, al igual que las propiedades y métodos, podrán variar mucho en función de lo que representan, como ya se verá más adelante, pero, un ejemplo podría ser el evento **Activate**:

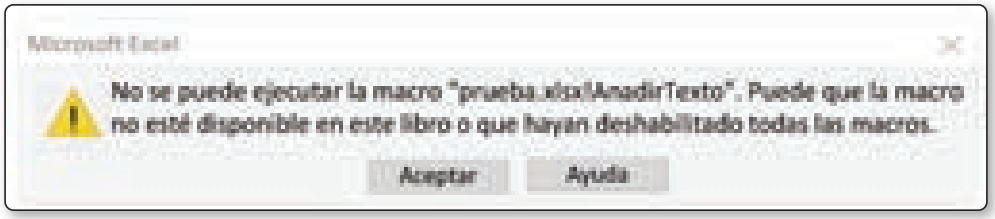
```
Worksheet_Activate()
```

CÓMO HABILITAR/DESHABILITAR LAS MACROS EN EXCEL

Cuando se abre un archivo de Excel que contiene macros, lo frecuente es que se nos pregunte si deseamos habilitar o no su contenido:



Mientras no pulsemos el botón de **Habilitar contenido**, todas las acciones, eventos y botones que estén asociados a una macro no se ejecutarán. En su lugar, se presentará, o debería presentarse, un mensaje similar al que se muestra a continuación:



Como se ha comentado anteriormente, antes de habilitar las macros será importante comprobar si el archivo es de una fuente confiable y si el contenido ejecutable es seguro.

Para ver si el contenido es seguro sólo podremos hacerlo activando las macros previamente, lo cual puede ser un riesgo puesto que podrían ejecutarse macros sin previo aviso.

No obstante, aunque no podamos ver su contenido y, aunque estén totalmente deshabilitadas, siempre podremos ver las macros definidas pulsando **Alt + F8** o pulsando en el icono **Macros**, situado al final de la pestaña **Vista**, y haciendo clic en **Ver Macros** posteriormente.

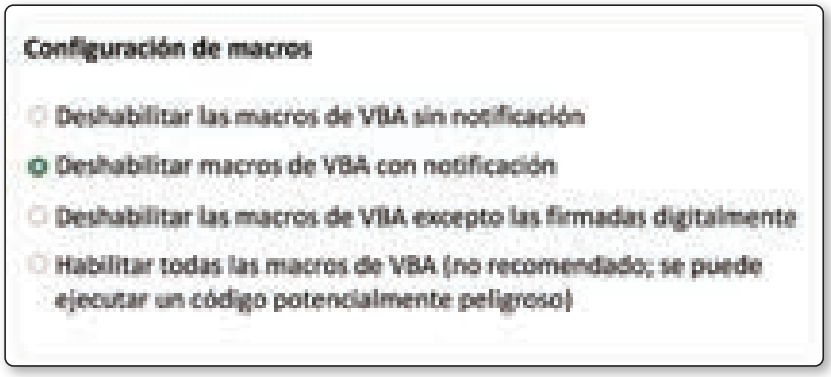
Si se está seguro, los pasos a seguir para habilitar las macros serán:

1. Pulsamos en **Archivo / Opciones**.

2. En el diálogo emergente o pantalla que se nos muestra, pulsamos en la opción de **Centro de confianza** y, una vez allí, en Configuración **del Centro de Confianza**.

3. Tras ello, se nos mostrará otro diálogo emergente o pantalla donde podremos configurar todo lo referido al "Centro de Confianza" y haremos clic en el apartado **Configuración de macros** y seleccionaremos, o deberemos seleccionar, la opción que mejor se ajuste a nuestras necesidades.

   En general, nos encontraremos con las siguientes opciones:



4. Finalmente, debemos hacer clic en **Aceptar**.

1

INTRODUCCIÓN A VISUAL BASIC FOR APPLICATIONS

1.1 TIPOS DE DATOS EN VBA

Existen múltiples tipos de datos en Visual Basic for Applications, aunque, los más comunes suelen ser los tipos String, Boolean, Integer y Double. A continuación, se muestra una tabla con todos los tipos de datos disponibles y una pequeña explicación:

Tipo de Datos	Tamaño en Bytes	Descripción / Posibles valores
Boolean	2	`True` o `False`. En conversiones 0 se corresponde con False y todo lo demás en True.
Byte	1	Entre 0 y 255
Integer	2	Entre -32.768 y 32.767
Single	4	Entre -3,4028235E+38 a -1,401298E-45 para valores negativos y, entre 1,401298E-45 a 3,4028235E+38 para valores positivos.
Long	4	Entre -2.147.483.648 y 2.147.483.648

Double	8	Entre -1,79769313486231570E+308 y -4,94065645841246544E-324 para valores negativos y, entre 4,94065645841246544E-324 y 1,79769313486231570E+308 para valores positivos.
Decimal	16	+/-79.22.162.514.264.337.593.543.950.335 sin decimales y,+/-7.92281625142643375935439503 35 con decimales.
Date	8	Entre 1 de enero del año 0001 y el 31 de diciembre del año 9999, con horas comprendidas entre las 00:00:00 y las 23:59:59.9999999
Currency	8	Con posibles valores entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807
String	N	Entre 0 y 2.000 millones de caracteres.
Object	4	Dirección a cualquier tipo de datos ya sea numérico, **`Boolean`**, **`String`**, **`Date`**, estructura o enumeración.
Variant		Puede contener cualquier tipo de datos exceptuando los de longitud fija, incluyendo los valores especiales **`Empty`**, **`Error`**, **`Nothing`** o **`Null`**.

1.2 DECLARACIÓN DE VARIABLES

LA declaración de variables en VBA se realiza a través de las palabras reservadas **`Dim`** y **`As`**.

```
Dim str As String
Dim numero As Decimal
Dim esVacio As Boolean
Dim fecha As Date
Dim media As Double
Dim edad as Byte
```

1.3 OPERADORES

Existen múltiples tipos de operadores entre los que podemos encontrar de tipo aritmético, lógico, comparación o de concatenación. A continuación, se muestran todos los tipos de operadores principales:

1.3.1 Aritméticos

Operador	Descripción
+	Suma o incremento. `Ejemplo: 7 + 2 = 9`
-	Resta. `Ejemplo: 7 - 2 = 5`
*	Multiplicación. `Ejemplo: 7 * 2 = 14`
/	División decimal. `Ejemplo: 7 / 2 = 3,5`
\	División entera. `Ejemplo: 7 \ 2 = 3`
^	Exponenciación o potencia. `Ejemplo: 7 ^ 2 = 49`
mod	Resto de la división entera. `Ejemplo: 7 mod 2 = 1`

1.3.2 Concatenación

Operador	Descripción
&	Concatenación o unión de valores de texto con conversión automática, siempre que sea posible. `Ejemplo: "Feliz" & " " & "Año" & " " & 2024`
+	Concatenación o unión de valores de texto sin conversión automática. `Ejemplo: "Feliz" + " " + "Año" + " " + "2024"`

En el caso de la concatenación con el símbolo sumar, si no hubiésemos puesto comillas al valor numérico, se habría generado un error de tipos y no se habría llevado a cabo la concatenación.

1.3.3 Comparación

Operador	Descripción
+	Igualdad. `Ejemplo: 7 = 2  ' Falso`
-	Diferencia. `Ejemplo: 7 <> 2  ' Verdadero`
*	Mayor o igual. `Ejemplo: 7 >= 2  ' Verdadero`
/	Menor o igual. `Ejemplo: 7 <= 2  ' Falso`

1.3.4 Lógicos

Operador	Descripción

And	Ambas condiciones o expresiones deben cumplirse o ser verdaderas.
	`' Suponiendo que a = 7 y b = 3:`
	`Ejemplo: a > 7 And b > 2  ' Falso`

Or	Una de las dos condiciones o expresiones debe ser cumplirse o ser verdadera.
	`' Suponiendo que a = 7 y b = 3:`
	`Ejemplo: a > 7 Or b > 2  ' Verdadero`

Not	Niega o contrapone el valor.
	`' Suponiendo que a = True:`
	`Ejemplo: Not a  ' False`

1.3.5 Operador Like

Permite comparar dos cadenas y averiguar si una cadena empieza, termina o incluye a otra. Para ello se vale o alimenta de patrones, los cuales se describen a continuación:

1.3.5.1 PATRÓN ASTERISCO (*)

Permite ver si una cadena está contenida dentro de otra, está al principio o está al final.

```
Dim str As String
Dim res As Byte

str = "Excel VBA"
If str Like "Ex*" Then
    res = 0            ' La cadena empieza por "Ex"
ElseIf str Like "*Ex" Then
    res = 1            ' La cadena termina por "Ex"
ElseIf str Like "*Ex*" Then
    res = 2            ' La cadena contiene "Ex"
End If

MsgBox res
```

En el ejemplo anterior, utilizamos la estructura `If` para preguntar si se cumple una u otra condición, el cual veremos más adelante. Dicho esto, si ejecutásemos el código podríamos comprobar que la condición que se cumplirá es la primera, por lo que el resultado que se nos mostrará en el cuadro de diálogo será `0`.

1.3.5.2 PATRÓN INTERROGACIÓN (?)

Permite ver o comprobar si hay coincidencia exacta con un único carácter.

```
Dim str As String
str = "Excel VBA"
res = 0

If str Like "Excel V?A" Then
    res = 1
End If

MsgBox res
```

Si ejecutásemos el código podríamos comprobar que la condición se cumple, por lo que el resultado que se nos mostrará en el cuadro de diálogo será `1`.

1.3.5.3 PATRÓN ALMOHADILLA (#)

Permite ver o comprobar si hay coincidencia exacta con un único dígito.

```
Dim str As String
str = "Excel VBA"
res = 0

If str Like "Excel V?A" Then
    res = 1
End If

MsgBox res
```

Si ejecutásemos el código podríamos comprobar que la condición NO se cumple, por lo que el resultado que se nos mostrará en el cuadro de diálogo será `0`.

1.3.5.4 PATRÓN CORCHETES ([])

Permite ver o comprobar si hay coincidencia o no con una lista de caracteres. Si lo que se desea es comprobar si NO están los caracteres de la lista en la cadena, deberemos añadir el símbolo admiración antes de los caracteres.

```
Dim str As String

str = "Excel VBA"
res = 0

If str Like "Excel V[A-Z]A" Then
    res = 1
End If

MsgBox res
```

Si ejecutásemos el código podríamos comprobar que la condición se cumple, por lo que el resultado que se nos mostrará en el cuadro de diálogo será `1`.

Ahora, si ejecutásemos el código siguiente podríamos comprobar que la condición NO se cumple, por lo que el resultado que se nos mostrará en el cuadro de diálogo será `0`.

```
Dim str As String

str = "Excel VBA"
res = 0
```

```
If str Like "Excel V[!A-Z]A" Then
    res = 1
End If

MsgBox res
```

1.3.5.5 PATRÓN ALMOHADILLA (#)

Permite ver o comprobar si hay coincidencia exacta con un único dígito.

```
Dim str As String

str = "Excel VBA"
res = 0

If str Like "Excel V?A" Then
    res = 1
End If

MsgBox res
```

Si ejecutásemos el código podríamos comprobar que la condición NO se cumple, por lo que el resultado que se nos mostrará en el cuadro de diálogo será `0`.

1.3.5.6 PATRÓN CORCHETES CON RANGOS ([-])

Permite ver o comprobar si hay coincidencia con los caracteres del alfabeto o rango indicado. Si indicamos un rango A-Z estaremos diciendo que mire si hay o no coincidencia con las mayúsculas. Si indicamos un rango a-z estaremos diciendo que mire si hay o no coincidencia con las minúsculas. Si indicamos un rango 0-9 estaremos diciendo que mire si hay o no coincidencia con los números.

```
Dim str As String

str = "Excel VBA"
res = 0

If str Like "Excel V[A-Za-z0-9]A" Then
    res = 1
End If

MsgBox res
```

Si ejecutásemos el código podríamos comprobar que la condición se cumple, por lo que el resultado que se nos mostrará en el cuadro de diálogo será `1`.

1.4 BUCLES

En lo que a bucles se refiere, VBA dispone de cuatro estructuras o bloques. A continuación, se explican cada uno de ellos:

1.4.1 For Each

Permite recorrer los objetos como si de una colección se tratase, por lo que no se basa en los valores de índice, sino en los nombres de sus propiedades o atributos.

```
Dim sheet As Worksheet

For Each sheet In Worksheets
    MsgBox "La hoja " & sheet & " está a " & sheet.Visble
Next
```

1.4.2 For Next

Permite recorrer los objetos a través de su índice o valores enteros. Para poder usarlo se requieren dos valores. El de inicialización, que se asigna como primer valor a la variable y, el último que se añade a continuación de la palabra reservada **To**.

```
Dim x As Byte

For x = 0 To 10
    MsgBox x
Next
```

1.4.3 Do While

Permite recorrer los objetos mientras se cumpla la condición. Este tipo de bucle podrá ejecutarse o no en función de si se cumple o no la condición declarada en el bucle.

```
Dim x As Byte

X = 1

Do While x < 10
    MsgBox x
Loop
```

1.4.4 Do Until

Permite recorrer los objetos hasta que se cumpla la condición. Este tipo de bucle podrá ejecutarse o no en función de si se cumple o no la condición declarada en el bucle.

```
Dim x As Byte

X = 1

Do Until >= 10
    MsgBox x
Loop
```

1.5 CONDICIONANTES

En lo que a condicionantes se refiere, VBA dispone de dos estructuras o bloques. A continuación, se explican cada uno de ellos:

1.5.1 If

La palabra reservada **`If`** permite ejecutar bloques de instrucciones en función de la condición previa que se declare. En general, suele llevar asociadas consigo las palabras **`Else`** y **`ElseIf`**, las cuales permiten ejecutar otros bloques en función de otras condiciones o carencia de estas.

También se sirve de la palabra reservada **`End`** para finalizar el bloque.

```
Dim num As Integer
Dim res As String

num = Int(1 + Rnd * (1000 - 1 + 1))

If num < 10 Then
    res = "00" & num
ElseIf num < 100 Then
    res = "0" & num
Else
    ' En cualquier otro caso
    res = num
End If

MsgBox res
```

El código anterior presenta una forma de conseguir un valor aleatorio entre dos números. Para ello, recurrimos a los comandos **Rnd** e **Int** y su sintaxis es la siguiente:

```
Int(valor_mínimo + Rnd * (valor_máximo – valor_mínimo + 1))
```

Al final de este código lo que se presenta es el valor aleatorio en un cuadro de diálogo emergente.

1.5.2 Case

La palabra reservada Case permite ejecutar uno de diferentes bloques de instrucciones en función una única expresión declarada. En general, Case suele llevar asociadas consigo las palabras **Select**, **To** y **Else**, las cuales permiten afinar los posibles valores asignables a cada bloque.

También se sirve de las palabras reservadas **End** y **Select** para finalizar el bloque.

```
Dim num As Integer

num = Int(1 + Rnd * (1000 - 1 + 1))

Select Case num
    Case 1, 2, 3, 4, 5, 6, 7, 8, 9 ' Si está entre 1 y 9
        res = "00" & num
    Case 10 To 99                  ' Si está entre 10 y 99
        res = "0" & num
    Case Else                      ' En cualquier otro caso
        res = num
End Select

MsgBox res
```

En el código anterior podemos observar que se hace la misma operación que el caso del comando **If** y cómo declarar un rango de valores mediante el separador coma o la palabra reservada **To**.

1.6 ARRAYS

Un array o matriz no es más que una colección de elementos del mismo tipo. Aunque en ocasiones pueden tener múltiples tipos, en realidad se pueden considerar como un único tipo global. Este tipo, en VBA, bien podría ser el tipo **Variant**.

Dicho esto, para definir un array sólo hay que recurrir a declararlo entre paréntesis, es decir, algo como:

```
Dim miArray(64) As Variant
```

Los valores entre paréntesis nos indica la longitud de la dimensión, lo que en el ejemplo anterior se traduce como una única dimensión. Ahora bien, si queremos crear arrays o matrices de múltiples dimensiones, lo que deberemos hacer es separar la longitud de cada dimensión entre comas. Así, si deseamos definir un array de dos o tres dimensiones únicamente deberemos hacer algo como:

```
Dim miArray(64, 64) As Variant       ' Para un array de 64x64
Dim miArray(64, 64, 64) As variant   ' Para un array de 64x64x64
```

Los argumentos, es decir, los valores de cada dimensión son opcionales, por lo que, si no se especifica ningún argumento, se creará una matriz de longitud cero.

En el siguiente ejemplo se crea una variable de tipo array o matriz de una única dimensión con tres posibles valores y se muestra en pantalla su conversión a cadena mediante el comando **`Join`**, el cual se alimenta del array como primer argumento y de un carácter o cadena como unificador:

```
Dim A(3) As Variant

A(0) = "a"
A(1) = "b"
A(2) = "c"

MsgBox Join(A, ",")
```

Esto debería mostrar como resultado algo como:

Cabe destacar que, por defecto, los índices de un array se definen a partir de `0` (cero), y no a partir de `1` como pueda suceder en otros lenguajes de programación.

De hecho, si hubiésemos definido los valores desde `1`, en vez de `0`, el resultado habría sido algo como:

No obstante, también podríamos replicar esta misma funcionalidad a través de la función **`Array`**, la cual nos permitirá definir una colección de elementos en el momento de crear la estructura. Por ejemplo:

```
Dim A As Variant

A = Array("a", "b", "c")

MsgBox Join(A, ",")
```

1.6.1 IsArray, LBound, UBound y otros comandos

Existen diferentes comandos para trabajar con matrices o arrays:

Comando	Descripción
`Array`	Permite crear una matriz o array.
`Dim`	Permite declarar e inicializar un array.
`IsArray`	Permite verificar si es o no un array.
`Join`	Permite convertir o unificar los elementos a un formato cadena y separándolos a partir de una subcadena o caracter.
`LBound`	Permite recuperar el límite inferior del array.
`Option Base`	Permite cambiar el límite inferior predeterminado. Su posibles valores son `0` y `1`.
`UBound`	Permite recuperar el límite superior del array.
`ReDim`	Permite reinicializar un array.

Ahora, para tener una mejor visión y, posiblemente, entender mejor estas funciones, lo más adecuado es que hagamos una práctica sencilla. Se trata de crear un array con los valores `1`, `2` y `3` y crear otro que nos devuelva la suma en su celda `0` y multiplicación en la celda `1`.

```
Option Base 1

Sub calcular()
    Dim numeros() As Variant
    Dim resultados(1 To 2) As Variant
    Dim x As Byte

    ReDim numeros(3) ' Asignamos 3 elementos
    numeros = Array(1, 2, 3)

    If IsArray(numeros) And IsArray(resultados) Then
        resultados(2) = 1
        For x = LBound(numeros) To UBound(numeros)
            resultados(1) = resultados(1) + numeros(x)
            resultados(2) = resultados(2) * numeros(x)
        Next
    End If

    MsgBox Join(resultados, ",")
End Sub
```

Si nos fijamos en el código anterior podemos observar que se establece la opción de **`Option Base 1`** antes de la definición del procedimiento calcular. Esto es así, y es muy importante, porque **`Option Base`** debe aparecer en el módulo antes de cualquier procedimiento como primera instrucción y una única vez ya que, de lo contrario, no funcionará y/o generará un error.

También podemos observar que hemos definido el array de resultados con unos índices de **`1 To 2`**. Esto se podría haber hecho de forma reducida, como ya hemos visto anteriormente, con la forma **`Dim resultados (2) As Variant`**, y debería funcionar de igual forma.

Y, por último, pero no menos importante, también podemos observar que, primero, se ha declarado la variable **`numeros`** como un array dinámico y que, unas líneas más abajo, se ha definido como un array de tamaño 3 con el comando **`ReDim`**.

1.7 CONVERSIONES A OTROS FORMATOS

```
Ejemplo: 7 + 2 = 9
```

1.7.1 Formatos numéricos

Formato	Descripción
Número	Es un valor numérico sin separadores de millar y que puede llevar el separador decimal.
Moneda	Es un valor numérico con separadores de millar y que puede llevar separador decimal.
Fijo	Es un valor con al menos un dígito a la izquierda y dos dígitos a la derecha del separador decimal.
Estándar	Es un valor con separador de millar, con al menos un dígito a la izquierda y dos dígitos a la derecha del separador decimal.
Porcentaje	Es un valor con al menos un dígito a la izquierda y que siempre va con dos dígitos a la derecha del separador decimal.
Científico	Es un valor basado en la notación científica estándar.
Sí/No	Es un valor que se corresponde con No si es `0` y Sí en cualquier otro caso.
Verdadero/Falso	Es un valor que se corresponde con False si es `0` y True en cualquier otro caso.
Activo/Inactivo	Es un valor que se corresponde con Inactivo si es `0` y Activo en cualquier otro caso.

1.7.2 Formatos de tipo fecha

Formato	Descripción
Fecha general	Realiza una correspondencia de un número real a una fecha con o sin hora. Si el valor no tiene parte decimal sólo se mostrará la fecha sin hora, es decir, DD/MM/YYYY. Si no tiene parte entera únicamente mostrará la hora, es decir, HH:MM: SS. El orden de los miembros de la fecha DD, MM, YYYY, HH MM, SS, PM, AM, ... vendrá prestablecida por la configuración del sistema que se esté utilizando.

Fecha larga	Muestra la fecha según el formato largo del sistema.
Fecha media	Muestra la fecha con el formato apropiado para el idioma seleccionado en el sistema.
Fecha corta	Muestra la fecha con el formato abreviada configurado en el sistema.
Hora larga	Muestra la fecha con el formato HH:MM:SS, es decir, de horas, minutos y segundos.
Hora media	Muestra la hora bajo el formato de 12 horas HH:MM que incluye las horas y minutos con el sufijo AM/PM.
Hora corta	Muestra la hora bajo el formato de 24 horas HH:MM que incluye las horas y minutos sin el sufijo AM/PM.

1.7.3 Formatos de tipo cadena

@	Marcador de posición de carácter. Muestra un carácter o un espacio. Si la cadena tiene un carácter en la posición en la que aparece la almohadilla (@) en la cadena de formato, lo muestra; de lo contrario, muestra un espacio en esa posición. Los marcadores de posición se rellenan de derecha a izquierda, a menos que haya un carácter de signo de exclamación (!) en la cadena de formato.
&	Marcador de posición de carácter. Muestra un carácter o nada. Si la cadena tiene un carácter en la posición en la que aparece la y comercial (&), la muestra; en caso contrario, no muestra nada. Los marcadores de posición se rellenan de derecha a izquierda, a menos que haya un carácter de signo de exclamación (!) en la cadena de formato.
<	Fuerza a minúsculas. Muestra todos los caracteres en minúsculas.
>	Fuerza a mayúsculas. Muestra todos los caracteres en mayúsculas.
!	Fuerza el relleno de marcadores de posición de izquierda a derecha. El comportamiento predeterminado es rellenar los marcadores de posición de derecha a izquierda.

1.7.4 Formatos definidos por el usuario

Los formatos definidos por el usuario son muchos y muy diferentes según la situación y/o necesidad. Todos ellos se pueden consultar en la dirección o URL de Microsoft:

https://learn.microsoft.com/es-es/office/vba/language/reference/user-interface-help/format-function-visual-basic-for-applications

1.7.5 Conversión y manipulación de cadenas

Si un array o matriz no es más que una colección de elementos del mismo tipo, una cadena de caracteres bien podría tomarse como un array o matriz de caracteres. Sin embargo, en función de las necesidades se usarán de un modo o de otro.

De hecho, si creamos una variable que contenga un array con las letras B, I, E, N y otra variable que contenga la palabra "BIEN", podríamos ir comparando carácter a carácter ambas variables y comprobando que son iguales y están en la misma posición.

Para esto y otras muchas cosas VBA dispone de múltiples comandos, los cuales se comentan a continuación:

1.7.5.1 CHR

Permite recuperar el carácter asociado a un determinado código. Este código suele ser un valor de 0 a 255, sin embargo, en los sistemas DBCS, el intervalo de valores es de -32768 a 65535.

```
str1 = Chr(65)    ' Returns "A"
str1 = Chr(97)    ' Returns "a"
```

1.7.5.2 FORMAT

Permite dar a una cadena otro formato. Este formato puede ser convertir de mayúsculas a minúsculas, formatear un valor como si fuese una fecha u hora o dar formato a un número dado.

```
Format(1234.5, "##,##0.00")            ' 1,234.50
Format(2, "0.00%")                     ' 200.00%
Format("ABC", "<")                     ' abc
Format("abc", ">")                     ' ABC
Format(#16:02:34#, "hh:mm:ss am/pm")   ' 04:02:34 pm
Format(#Jul 5, 2023#, " dddd, dd/mm/yyyy") ’ Miércoles 05/07/2023
Format("Excel VBA", "<")               ' "excel vba"
```

Cabe destacar que VBA no sólo nos proporciona una forma de convertir valores, también nos prevee de una serie de constantes que podemos usar como si de valores numéricos se tratase. Entre estos están los días de la semana nombrados en inglés con el prefijo "vb" delante. Todas ellas se pueden consultar en la URL *https://learn.microsoft.com/es-es/office/vba/language/reference/user-interface-help/format-function-visual-basic-for-applications*.

1.7.5.3 LEN

Permite conocer la longitud de una cadena o el número de bytes necesarios para almacenar una variable.

```
Dim num As Integer
size = Len(num)        ' Devuelve 2
size = Len("AbC")      ' Devuelve 3
```

1.7.5.4 LCASE Y UCASE

La función `LCase` permite convertir una cadena a minúsculas mientras que, la función `UCase` permite convertir una cadena a mayúsculas.

```
str1 = LCase("AbC")    ' Devuelve "abc"
str1 = UCase("AbC")    ' Devuelve "ABC"
```

1.7.5.5 LEFT Y RIGHT

La función **Left** permite recuperar una subcadena de otra cadena con una longitud fija empezando por la izquierda. La función **Right** permite recuperar una subcadena de otra cadena con una longitud fija empezando por la derecha.

```
str1 = Left("Excel VBA", 5)    ' Devolverá "Excel"
str1 = Right("Excel VBA", 5)   ' Devolverá "l VBA"
```

1.7.5.6 LSET Y RSET

La función **LSet** permite justificar una cadena a la izquierda mientras que, la función **RSet** permite justificar una cadena a la derecha.

```
LSet str2 = str1   ' Esto hará que "AAA" sea menor que "aaa"
RSet str2 = str1   ' Esto hará que "AAA" sea igual que "aaa"
```

Cabe destacar que esta alineación sólo se produce cuando la cadena a alinear es más corta que su espacio contenedor dentro de la variable, por tanto, si se intenta alinear una cadena que no contiene espacios a los lados no funcionará. Eso sin olvidar que, además, son instrucciones de versiones antiguas de Visual Basic que ya no tienen casi uso.

1.7.5.7 MID

Permite recuperar una subcadena de otra cadena con una longitud fija empezando desde una posición inicial concreta. El último argumento, que define la longitud, es opcional.

```
str1 = Mid("Excel VBA", 7, 3)   ' Devolverá "VBA"
str1 = Mid("Excel VBA", 3)      ' Devolverá "cel VBA"
```

1.7.5.8 OPTION COMPARE

Permite definir el método de comparación predeterminado que se utilizará al comparar valores de tipo string o cadenas.

```
Option Compare Binary      ' Esto hará que "AAA" sea menor que "aaa"
Option Compare Text        ' Esto hará que "AAA" sea igual que "aaa"
```

1.7.5.9 SPACE

Permite construir una cadena de una longitud fija rellena de espacios.

```
str1 = "|" & Space(5) & "|"    ' Devuelve "|     |"
```

1.7.5.10 STR

Permite convertir números en strings o cadenas.

```
str1 = Str(123.45)       ' Devuelve " 123.45"
str1 = Str(-123.45)      ' Devuelve "-123.45"
```

1.7.5.11 STRCOMP

Permite comparar dos cadenas. Devuelve un valor Variant que indica el resultado de la comparación. Su sintaxis incluye 3 argumentos, pero sólo el último es opcional. Este último argumento especifica el tipo de comparación que puede ser binaria, textual o detección automática.

```
StrComp("ABC", "abc", vbTextCompare)
' Devuelve 0, es decir, que es igual
StrComp("ABC", "abc", vbBinaryCompare)
' Devuelve -1, es decir, que es menor
StrComp("ABC", "abc", vbUseCompareOption)
' Devuelve 1, es decir, que es mayor
```

1.7.5.12 SPACE

Permite construir una cadena de una longitud fija rellena de una secuencia fija de caracteres.

```
str1 = String(4, "·")      ' Devuelve "····"
```

1.7.6 Conversiones de tipo

En lo que a conversiones a otros formatos se refiere, VBA dispone de múltiples comandos.

No obstante, cabe destacar que, para que la conversión no produzca un error, el valor proporcionado debe entrar dentro de los límites definidos por el tipo de dato del comando o función conversora. Esto es, si intentamos realizar la conversión del valor 3207 a Byte se producirá un error puesto que el tipo Byte sólo admite valores entre 0 y 255.

A continuación, se explican cada uno de ellos.

```
CBool(4 = 3)         ' Devuelve false
```

Función	Tipo devuelto	Descripción
CBool	Boolean	Convierte a True o False el valor proporcionado. `CBool(4 = 3)  ' Devuelve falso`

CByte	Byte	Convierte a formato byte el valor proporcionado.
		`CByte(25)  ' Devuelve 25`
		`CByte("255")  ' Devuelve 25`
		`CByte(25 & 32)  ' Produce error`

CCur	Currency	Convierte a moneda el valor proporcionado.
		`CCur(25)  ' Devuelve 25`
		`CCur(255 & ".12")  ' Devuelve 25512`
		`CCur(12 & ",345")  ' Devuelve 12,345`

CDate	Date	Convierte a fecha y/o tiempo el valor proporcionado.
		`CDate(#8/13/2023#)  'Devuelve 13/08/2023`
		`CDate(#12/31/2023#)  'Devuelve 31/12/2023`
		`CDate(#Aug 16, 2023#)  'Devuelve 16/08/2023`
		`CDate(#2/29/2023#)  'Produce Error`
		`CDate(#4:59:00#)  'Devuelve 04:59:00`
		`CDate(#5:0:0 PM#)  'Devuelve 17:00:00`
		`CDate(#24:00:00#)  'Produce error`

CDbl	Double	Convierte a Double el valor proporcionado.
		`CDbl(#4:59:00#)  ' Devuelve 0,207638888888889`
		`CDbl(8.2 * 0.001)  ' Devuelve 0,0082`
		`CDbl("1345")  ' Devuelve 1345`

CDec	Decimal	Convierte a decimal el valor proporcionado.
		`CDec(#4:59:00#)  ' Devuelve 0,207638888888889`
		`CDec(8.2 * 0.001)  ' Devuelve 0,0082`
		`CDec("1345")  ' Devuelve 1345`

CInt	Integer	Convierte a Integer el valor proporcionado.
		`CInt(#4:59:00#)  ' Devuelve 0`
		`CInt(8.2 * 0.001)  ' Devuelve 0`
		`CInt("1345")  ' Devuelve 1345`
		`CInt("a")  ' Produce error`

CLng	Long	Convierte a Long el valor proporcionado.
		`CLng(#4:59:00#)  ' Devuelve 0`
		`CLng(8.2 * 0.001)  ' Devuelve 0`
		`CLng("1345")  ' Devuelve 1345`
		`CLng("a")  ' Produce error`

CLngPtr	LongPtr	Convierte a entero de 32 bits un valor de 64 bits.
		`CLngPtr(#4:59:00#)  ' Devuelve 0`
		`CLngPtr(8.2 * 0.001)  ' Devuelve 0`
		`CLngPtr("1345")  ' Devuelve 1345`
		`CLngPtr("a")  ' Produce error`

CSng	Single	Convierte a Single el valor proporcionado.
		`CSng(#4:59:00#)  ' Devuelve 0,2076389`
		`CSng(8.2 * 0.001)  ' Devuelve 0,0082`
		`CSng("1345")  ' Devuelve 1345`
		`CSng(vbTextCompare)  ' Devuelve 1`
		`CSng("a")  ' Produce error`

CStr	String	Convierte a cadena el valor proporcionado.
		`CStr(#4:59:00#)  ' Devuelve 4:59:00`
		`CStr(8.2 * 0.001)  ' Devuelve 0,0082`
		`CStr(123.45)  ' Devuelve 123,45`
		`CStr(vbTextCompare)  ' Devuelve 1`

CVar	Variant	Convierte a Variant el valor proporcionado. Posee el mismo intervalo que Double para valores numéricos Doble y que String para valores no numéricos.
		`CStr(#4:59:00#)  ' Devuelve 4:59:00`
		`CStr(8.2 * 0.001)  ' Devuelve 0,0082`
		`CStr(vbTextCompare)  ' Devuelve 1`
		`CVar(123 & "000")  ' Devuelve 123000`

Fix	Long	Trunca, es decir, NO redondea, un valor dado. La principal diferencia con **Int** es que devuelve el primer entero negativo mayor o igual que el número dado.
		`Fix(99.2)  ' Devuelve 99`
		`Fix(99.8)  ' Devuelve 99`
		`Fix(-99.2)  ' Devuelve -99`
		`Fix(-99.8)  ' Devuelve -99`

Int	Integer	Trunca, es decir, redondea, un valor dado. La principal diferencia con **Fix** es que devuelve el primer entero negativo menor o igual que el número dado.
		`Int(99.2)  ' Devuelve 99`
		`Int(99.8)  ' Devuelve 99`
		`Int(-99.2  ' Devuelve -100`
		`Int(-99.8)  ' Devuelve -100`

Round	Integer	Redondea hacia arriba o hacia abajo en función del valor proporcionado.
		`Round(99.2)  ' Devuelve 99`
		`Round(99.8)  ' Devuelve 100`
		`Round(-99.2)  ' Devuelve -99`
		`Round(-99.8)  ' Devuelve -100`

1.8 RANGOS Y CELDAS

A lo que nosotros denominamos rango, Excel lo llama el objeto **Range**. Este objeto puede representar una celda, una fila, una columna o una agrupación de estos. Por otro lado, Excel nos provee del objeto **Cells**, el cual nos permite acceder a un objeto Range que representa un conjunto de celdas.

Por tanto, podemos decir que tanto **Range** como **Cells** nos permitirán el acceso a una celda o agrupación de ellas. No obstante, existe una pequeña diferencia de uso. Mientras que **Cells** generalmente se utiliza para hacer referencia a una única celda, **Range** se suele utilizar para hacer referencia a una agrupación de éstas.

Por otro lado, también es importante destacar que cuando se trabaja con estos objetos sin declarar ninguna, se tomará por defecto la propiedad **Value**. Esto es:

```
Range("A1") = Range("B1")
Cell(1, 1) = Cells(1, 2)
```

Es equivalente a:

```
Range("A1").Value = Range("B1").Value
Cell(1, 1).Value = Cells(1, 2).Value
```

Por otro lado, los rangos y celdas pueden ser accedidos mediante el nombre de la columna o a través de un **Ofsset**.

```
Range("J1").Offset(4, 2)
Cells("1, "J").Offset(4, 2)
```

Y dentro del contexto de los desplazamientos, también podemos usar las palabras reservadas **xlDown**, **xlUp**, **xlToRight** y **xlToLeft** de forma conjunta con la función **End**, la cual permite desplazarse a una celda específica dentro de la región actual en la que se está trabajando.

```
Range("J1").End(xlDown).Select
```

Pero esto no es todo ya que, tanto **Range**, como **Cells**, nos permite acceder a los objetos de estilo, es decir, a los objetos **Interior** y **Font**, los cuales son los encargados de proporcionar color, tipo de letra, estilo de borde, etcétera.

A continuación, se muestran algunos ejemplos:

```
Range("A1").Interior.Color = xlNone
Cells(1, "A").Font.Color = vbBlack
Range("A1").Font.Bold = False
```

Y también nos permite borrar con el método **Clear**, o asignar fórmulas con la propiedad **Formula**.

```
If ActiveSheet.Range("C1").Formula = "" Then
    ' La llamada a Clear no es necesaria, pero sive como ejemplo
    Range("C1").Clear

    ActiveSheet.Range("C1").Formula = "=A1+B1"
End If
```

1.9 CONTROL DE ERRORES

A continuación, se muestran algunas formas para controlar posibles errores que se puedan dar durante el tiempo de ejecución de nuestras macros:

1.9.1 CVErr

La función **CVErr** devuelve un valor de tipo Variant que representa el error que, previamente, se ha producido.

```
Select Case err
    Case CVErr(xlErrDiv0)
    MsgBox "Error detectado: #DIV/0!"
    Case CVErr(xlErrNA)
    MsgBox "Error detectado: #N/A"
    Case CVErr(xlErrName)
    MsgBox "Error detectado: #NAME?"
    Case CVErr(xlErrNull)
    MsgBox "Error detectado: #NULL!"
    Case CVErr(xlErrNum)
    MsgBox "Error detectado: #NUM!"
    Case CVErr(xlErrRef)
    MsgBox "Error detectado: #REF!"
    Case CVErr(xlErrValue)
    MsgBox "Error detectado: #VALUE!"
    Case Else
    MsgBox "This should never happen!!"
End Select
```

Cabe destacar que, aunque a menudo se usa de manera básica, como se ha mostrado en el código anterior, también se puede usar para crear errores tipificados por el usuario dentro de procedimientos definidos por el usuario.

1.9.2 On Error

La instrucción **On Error** permite definir una rutina de control de errores dentro de un procedimiento, aunque también se puede usar para deshabilitar una rutina de control de errores.

```
Sub Divide(num1, num2)
    On Error GoTo ErrorHandler
```

```
    Dim res As Double

    res = 4 / 0

    MsgBox "Continuamos..."
    Exit Sub

ErrorHandler:
    MsgBox "Se ha producido un error"
    Resume Next
End Sub
```

Para deshabilitar el control de errores en un determinado procedimiento, se debe poner lo siguiente:

```
On Error GoTo 0
```

Cabe destacar que, como se puede ver el ejemplo anterior, la declaración de la instrucción **`On Error`** debe estar definida como primera declaración dentro del procedimiento.

1.10 MANIPULACIÓN DE COLORES

La manipulación de colores en VBA Excel puede realizarse a través de la función **`RGB`**, a través de su enumeración **`XlRgbColor`** o mediante la propiedad **`ColorIndex`**.

La función **`RGB`** nos permite asignar un color determinado a través de la intensidad de los colores primarios de la luz. Sin embargo, lo que, en realidad devuelve, es un número entero Long.

```
MsgBox RGB(128, 128, 128)        ' Devuelve 8421504
```

La enumeración **`XlRgbColor`** nos permite asignar un color determinado a través de un nombre predefinido. Sin embargo, al igual que antes, lo que devuelve es un número entero Long.

```
MsgBox rgbBrown                  ' Devuelve 2763429
```

Todos los nombres de este tipo enumerado están en la URL o dirección web *https://learn.microsoft.com/es-es/office/vba/api/excel.xlrgbcolor*.

Y, por último, la propiedad **`ColorIndex`** nos permite asignar un color determinado a través de un valor de índice de la paleta de colores activa o mediante

una de las siguientes constantes **xlColorIndexAutomatic**, que representa una forma de indicar que el color sea automático y **xlColorIndexNone**, que representa una forma de indicar que no se desea color.

```
MsgBox rgbBrown                ' Devuelve 2763429
```

La tabla de colores se muestra a continuación:

	1		15		29		43
	2		16		30		44
	3		17		31		45
	4		18		32		46
	5		19		33		47
	6		20		34		48
	7		21		35		49
	8		22		36		50
	9		23		37		51
	10		24		38		52
	11		25		39		53
	12		26		40		54
	13		27		41		55
	14		28		42		56

Esta lista de colores también está disponible en la URL o dirección web *https://learn.microsoft.com/es-es/office/vba/api/excel.colorindex*.