Introducción a la Programación en Visual Basic Script

PARTES DEL TUTORIAL:

SOBRE LOS TUTORIALES:

Visual Basic Script es un lenguaje de programación mediante Scripts (no genera .exe al no ser compilado), de alto nivel. En general es uno de los lenguajes más básicos y es, sin duda alguna, la base del Visual Basic 6, ya que todo lo que se aprende aquí, luego puede ser usado en él sin ningún cambio. En otras palabras, es el lenguaje que se suele aprender antes del Visual Basic 6. Es por eso que todo programador de Visual Basic que se precie debe conocerlo.

Con este motivo, para aquellos que se quieran iniciar en el mundo de la programación con un lenguaje sencillo de aprender, antes de estudiar en la universidad ya otros más avanzados, he decidido poner aquí un tutorial de aprendizaje de Visual Basic Script desde lo más básico hasta lo más avanzado, y posteriormente, como adaptar lo aprendido a Visual Basic 6, y un tutorial sobre él. En definitiva, lo que yo llamo un proceso de aprendizaje desde lo más simple hasta el nivel que te permitirá realizar programas como los que yo ofrezco...

- **Introducción en Programación en Visual Basic Script.**
- Programación con Objetos (ActiveX, ".ocx")
- Uso avanzado y Ejemplos de VBScript

De estos tres simples, pero largos documentos está compuesto el tutorial de Visual Basic Script. Deberás aprender por orden cada uno de ellos para continuar con el siguiente, pero no te apures, no es difícil.

Y no lo olvides, para dudas o consultas escribe a: scswinter@hotmail.com

Y no dudes en visitar: http://scswinter.110mb.com/

Programación en Visual Basic Script: Introducción

Visual Basic Script es un lenguaje de programación de alto nivel no compilado para Windows. Un entendido podría describirte este lenguaje con esta simple frase, pero ¿Qué significa realmente? Vayamos por pasos:

Visual Basic Script es un lenguaje de programación, esto es, una forma de decirle al Sistema que debe hacer. Si, con los lenguajes de programación los "informáticos" crean sus programas. De hecho, incluso el propio Windows está hecho con uno. Por decirlo de alguna manera, es la *forma de comunicarnos con el ordenador* para crear programas.

Por otro lado, existen muchísimos lenguajes de programación hoy en día. Si buscáis en Internet os aparecerán más de cien, por eso, necesitan de una clasificación para poder elegir el más apropiado para cada ocasión. Porque *no todos los lenguajes son iguales y sirven para lo mismo*. De esta forma, inicialmente se hizo una clasificación, que dividía a los lenguajes en dos categorías:

- **De bajo nivel:** Son aquellos que utilizan expresiones y recursos que controlan directamente todo lo que pasa en el ordenador a nivel lógico. Es decir, que por ejemplo, en Ensamblador, para escribir en un archivo, debes enviar una interrupción al procesador y enviarle los datos de acceso a registros del procesador concretos, esperando a la vez una respuesta. Son lenguajes difíciles de aprender, costosos de programar (los programas mas sencillos ocupan mas de mil líneas...) pero mas rápidos y eficaces, pues tienes el control absoluto sobre el programa.
- **De alto nivel:** Son aquellos que utilizan expresiones y recursos familiares a la lengua diaria (inglesa, por supuesto). Dan por supuestas muchas cosas para facilitar el trabajo a los programadores y son mucho más fáciles de aprender y programar. Hoy en día, casi todos los lenguajes son de alto nivel, ya que con ellos, por ejemplo, escribir en un archivo es tan sencillo como indicar el nombre del mismo, y lo que quieres escribir. Existen muchísimos (C, Delphi, Ruby, VB, Pascal...), y entre ellos se encuentra el VBScript.

Con esto, queda explicado que significa ser un lenguaje de alto nivel, pero es necesario también conocer que significa que el VBScript no es compilado. Independientemente de que un lenguaje sea de alto o de bajo nivel, pueden estar compilados o no. Esto significa que, *el ordenador no entiende lo que nosotros escribimos*, ya que este se rige por impulsos eléctricos, definidos como bits (0 y 1). Para que el ordenador lo entienda, es necesario que se traduzca el lenguaje a la cadena de 1 y 0 que el es capaz de tratar. Para ello se utilizan básicamente dos cosas:

- **Compilar:** Un programa llamado compilador, lee lo que has escrito y lo traduce según las normas del lenguaje usado a algo que el ordenador entiende, llamado comúnmente código maquina. Esta "traducción" da como resultado en Windows un archivo acabado en .exe que se puede ejecutar siempre que se quiera.
- **Uso de un interprete (no compilar):** Esto consiste en que el código fuente (el código que has escrito), sea leído cada vez que se ejecuta por un programa llamado interprete, que lo va traduciendo al mismo tiempo que se está

ejecutando. Estos programas tienen una extensión diferente a la general (.rb, .js, .vbs, .shs...) pues lo que se ejecuta es el código fuente cada vez.

Supongo, que con esto queda claro un poco que significan cosas como "*compilado*", "*código fuente*", "*alto nivel*", "*código maquina*", "*bajo nivel*", "*interprete*"... Decir también que esto es una vista general, ya que como VBScript no es un lenguaje compilado, ni es de bajo nivel, no se necesitan más explicaciones, aunque debe quedar claro que no todo es tan sencillo como aqui se describe.

Todas estas características están muy bien, pero no sirven de nada si no se ven en la práctica. Por tanto, comenzaremos viendo que aspecto tiene un programa escrito en VBScript:

Si, como veis, un programa en VBScript es un **archivo de texto**, en cuyo interior escribiremos el código ateniéndonos a las normas del lenguaje, que luego se renombra a "loquequieras.vbs", adoptando este archivo. Cuando lo tenemos con este aspecto, haciendo **doble clic encima se ejecuta**, y haciendo **clic con el botón derecho y eligiendo editar, se modifica**.

Nombremos los pasos que hay que seguir para crear un script en este lenguaje:

- Abrir el Bloc de Notas (los scripts son archivos de **texto sin formato**).
- Escribir dentro el código (en este caso, dejémoslo en blanco).
- Hacer clic en Archivo, Guardar como...
- Seleccionar la ruta y escribir como nombre de archivo "loquequieras.vbs" (**comillas incluidas**).

Una vez guardado, podéis comprobar como, ciertamente, si hacemos doble clic no pasa nada (se ejecuta, pero como no hay nada escrito, no pasa nada) y si hacemos clic con el botón derecho y elegimos editar, nos aparecerá en blanco (si no funciona, probad *abrirlo con el comando Abrir del Bloc de Notas*).

Para asegurarte de que se ha creado correctamente, debes fijarte en el titulo que muestra el Bloc de Notas, cuando lo abres para editar, que se debe corresponder al siguiente:

Es un error tener un nombre como "nombre.vbs.txt". Es absolutamente necesario que acabe en ".vbs" para que se ejecute correctamente.

Por último, antes de pasar a ver como se edita un Script, mencionar lo que ocurre cuando haces doble clic en un archivo acabado en ".vbs" porque a veces suele ser muy útil saberlo. Esto se resume en el siguiente esquema:

```
Se ejecuta Wscript.exe --> Wscript.exe lee el script --> Wscript.exe ejecuta el script --> Se cierra Wscript.exe
```

Hay un archivo en la carpeta Windows de tu PC, que se llama "*Wscript.exe*", este es el **intérprete de VBScript**. En sus versiones más antiguas podía ser eliminado para impedir la ejecución de los ".vbs", pero ahora está bastante protegido. Pero bueno, saber que si alguna vez se te "cuelga" algún script, solo hay que darle a Crtl+Alt+Supr y en la pestaña de procesos, cerrar el que tiene ese nombre.

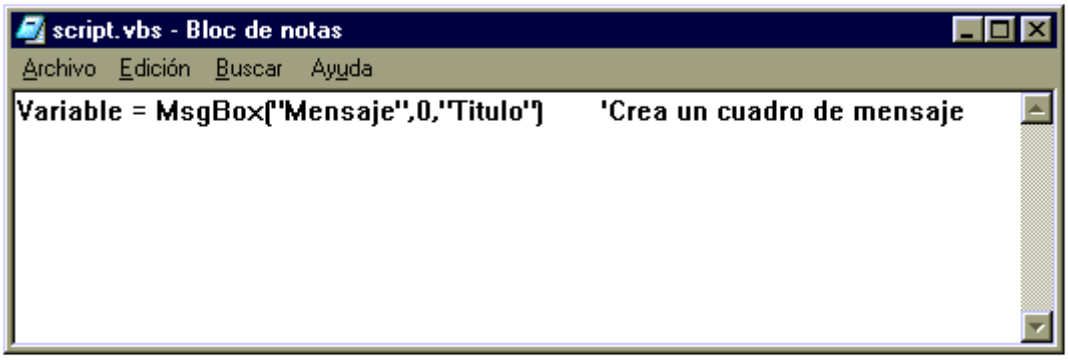
Para editar un Script, como se ha explicado antes, se puede hacer con el bloc de notas, aunque existen programas especializados en edición de este tipo de lenguaje que facilitan mucho las cosas. Pese a esto, lo edites como lo edites, hay una serie de **normas** y cosas que hay que tener en cuenta a la hora de escribir cualquier cosa. Estas son las siguientes:

- El script se ejecuta de forma secuencial, es decir, que la acción escrita en la primera línea, se ejecuta antes que la que hay en la segunda línea. La que haya en la segunda línea, antes que la que hay en la tercera, etc.
- Existen comentarios (aclaraciones) escritas fuera de la regla del lenguaje. Estos van precedidos de una comilla simple ('), que se escribe pulsando la tecla de al lado del 0, o de la palabra *Rem* (esta solo se puede poner al principio de una línea, no por la mitad...). Es decir, si en una línea nos encontramos *acción 'Esto sirve para...* la aclaración (todo lo que va detrás de la comilla) no se ejecuta. Es decir, son simples aclaraciones, ignoradas en la ejecución del programa.
- Solo se puede escribir una orden por línea. Cada línea nueva indica el fin de una orden y el comienzo de otra. No se puede partir una orden en dos líneas ni colocar más de una en una misma línea. (Existen excepciones que se comentarán mas adelante...)
- Si hay alguna orden errónea, o algún error de algún tipo, se mostrará un mensaje con el contenido de dicho error, y en la línea en la que ocurre (la información que muestra dicho mensaje no siempre es fiable, pero por regla general, le haremos caso...)

A la hora de seguir este tutorial, encontrareis **cuadros** como el siguiente, que muestran ejemplos de lo que se explica:

```
Variable = MsgBox(" Mensaje ",0," Titulo " 'Crea un cuadro de mensaje
```

Estos cuadros deben ser **copiados a vuestro script** borrando todo lo demás o en una línea nueva vacía (recuerda las normas que se acaban de explicar), y pueden ser modificados siguiendo las instrucciones que se indican en el tutorial. Concretamente, en este cuadro de ejemplo, si lo copias, veréis el resultado que provoca al ejecutarlo, aunque lo que significa y como modificarlo, ya será explicado mas adelante.



En este caso, la imagen muestra como se copia, y ved que no tiene mas complicación. Fijaros que el código, contiene una instrucción en una línea y además posee un comentario, que no influye en nada ya que es simplemente informativo.

Si, por algún motivo incumplimos alguno de los puntos anteriores, se nos mostrara el mensaje de error correspondiente.

Cuando algo en nuestro script es incorrecto, se nos muestra un mensaje de error advirtiéndonos del mismo. Este mensaje tiene la apariencia mostrada a continuación:

Podemos ver que en el mensaje de error hay varios datos, algunos como el código o el origen no nos importan, pero otros nos pueden ser útiles. Estos son los siguientes:

- Secuencia de comandos: Hace referencia al archivo que ha producido el error. En este caso de ejemplo, el error habría sido dado por un archivo que está en la carpeta VBScript en el disco duro D, y que se llama script.vbs.
- Línea: Indica la línea en la que se ha producido el error indicado. Esto es muy útil ya que reduce la búsqueda a una zona muy pequeña del script, aunque no siempre es fiable, orienta mas o menos donde puede haber sido.
- Error: Los errores suelen ser básicamente de dos tipos: Errores de sintaxis y Errores en el tiempo de ejecución. Los errores de sintaxis se producen cuando hay algo mal escrito (por ejemplo se parte una orden en dos líneas...) a lo largo del script. Los errores en el tiempo de ejecución se producen cuando se quieren realizar operaciones con resultado erróneo (dividir entre cero, raíces de números negativos, tomar valores de texto como números...); son errores que se producen, no porque el script esté mal diseñado, sino porque se introducen mal los datos.

Básicamente, cuando hay algún error, si es de sintaxis, se ve a la línea indicada y se comprueba si todo está escrito correctamente. Si es en el tiempo de ejecución hay que comprobar si se ha intentado hacer alguna operación inválida. Por otro lado, en el apartado Error nos pueden aparecer mensajes más explícitos que estos dos, como por ejemplo, "constante de cadena sin terminar" o "división por cero". Dichos mensajes nos dan más información aún sobre lo que puede haber ocurrido en nuestro script.

Por otro lado, puede que alguna vez sepas que se produce algún error, pero quieras que pese a esto el script continúe su ejecución, o bien tengas un script terminado y correctamente escrito, pero quieres protegerlo contra posibles errores en el tiempo de ejecución; para ello se puede utilizar una función cuyo cometido es que *si en una línea se produce un error, salta a la siguiente* sin mostrar ningún error ni nada por el estilo. Esta función es:

```
On Error Resume Next
```

Si en una línea escribimos esto, a partir de esa (todas las que tenga debajo) aunque se produzcan errores, ni se mostrarán ni se parará la ejecución del script; por eso, lo normal es ponerlo en la primera línea. De hecho, si copiáis esto y debajo escribís cualquier cosa, no mostrará ningún error, mientras que si borráis esta línea, si lo mostrará. Si queremos que *vuelva a mostrar errores* unas líneas mas abajo, lo único que hay que hacer es colocar esta otra función:

```
On Error Go To 0
```

Y a partir de esa línea se volverán a mostrar mensajes de error con normalidad.

Si buscáis por Internet o en ayudas los posibles errores veréis que la mayoría vienen en tres columnas. Una que indica el **numero de error** (no es mas que un numero identificativo), el **mensaje de error** que se muestra, y la **descripción del mismo**.

Programación en Visual Basic Script: Las Variables

Esta parte es, sin duda, la más importante en la programación no solo de VBScript, sino de cualquier lenguaje de programación de cualquier tipo. Podemos comparar una variable a una caja a la que la etiquetamos con un nombre y guardamos y sacamos cosas de ella. También podemos compararla con la "x" de las funciones matemáticas a la que le podemos dar cualquier valor. En cualquier caso, una variable tiene un nombre identificativo y un contenido. El nombre identificativo será siempre el mismo y no variará, en cambio, el contenido puede variar continuamente.

Existen dos métodos para usar las variables en VBScript, uno mas *"libre"* y otro mas *similar a los demás lenguajes*. Para que no cojais malas costumbres, dejaremos de lado el libre y os enseñaré el más común. Este se activa colocando en la **primera línea** la función siguiente:

```
Option Explicit
```

Y, luego, en el momento que se quiera crear una variable se utiliza la palabra Dim, seguido del nombre identificativo que le queramos dar, colocándolo todo, claro está, en una sola línea. Por ejemplo:

```
Option Explicit
Dim variable1
```

Así habremos creado una variable con el nombre identificativo variable1. Si queremos crear mas de una variable a la vez, solo hay que colocar una coma y poner luego todas las demás:

```
Option Explicit
Dim variable1, var2, varmama, nombre
```

Así habremos creado cuatro variables con los nombres variable1, var2, varmama y nombre. Recordad que el Option Explicit solo se coloca una vez al principio del Script, pero el Dim, se puede colocar tantas veces como se quiera para crear nuevas variables a lo largo del script:

```
Option Explicit
Dim variable1, var2
'Mas funciones...
Dim varmama, nombre
```

Sabemos ya pues, como crear variables, pero ahora debemos aprender como darles valores. Para introduciros debo deciros que básicamente trabajaremos con tres tipos de valores: los valores numéricos, los valores alfanuméricos y los valores booleanos.

- Valores numéricos: Son valores numéricos que van desde números negativos con decimales hasta números positivos decimales. Es decir, cualquier **número real** que exista.
- Valores alfanuméricos: Podemos introducir textos, nombres, frases, párrafos, etc. en una variable. Admite **cualquier carácter** (letras y números) excepto el salto de línea y la doble comilla, que se introducen de forma especial (ya se verá...).
- Valores booleanos: Son valores que indican si es **verdadero o falso**. Podemos darle a una variable el valor verdadero o falso (en números equivaldría a un 0 o un 1).

Una vez tenemos una variable creada, podemos introducirle un valor. Veamos primero como introducimos valores numéricos, que son, los mas sencillos de usar y entender. Para darle el valor numérico a una variable, colocaremos el **nombre de la variable seguido de un igual y el valor que le queramos dar** (si el numero es decimal se colocará entre comillas, aunque es mejor ponerlo en forma de fracción). Por ejemplo:

```
Option Explicit
Dim varPi
varPi = 3
```

El valor se le puede cambiar siempre que se quiera a lo largo del código, como se muestra en el siguiente ejemplo (en este ejemplo ya se *obvia la creación de la variable*, añade el código correspondiente a dicha creación antes del ejemplo):

```
varPi = 3
'Mas código y funciones
varPi = "3,14"
```

De esta forma, todas las funciones que utilicen la variable varPi entre después de darle el valor 3 usarán ese valor, mientras que todas las funciones que vayan después de darle el valor 3,14 usaran el 3,14 y no el 3. Es decir, como he explicado antes, actúan como almacenes de datos.

Queda explicado pues, como se asigna de forma básica valores numéricos a las variables. Pero esto no es lo único que se puede hacer con variables numéricas, porque para algo existe la **suma (+), la resta (-), la multiplicación (*), la división (/), la potencia (^), los paréntesis [( )], etc.** A las variables se les puede dar como valor el resultado de una operación:

```
varPi = (7*2)+1
```

De esta forma, por ejemplo, varPi obtendrá el valor 15 (7 por 2 son 14, mas 1, da 15). También, se pueden introducir otras variables en la operación. Pongamos un ejemplo un poco mas complejo (obviamos la creación):

```
var1 = (7*2)+1
var2 = 7*(2/3)
var3 = var1 + var2
var4 = (1/2)* var3
```

De esta forma, var1 tendrá el valor 15, var2 el valor 4,6..., var3 el valor 19,6... y var4 el valor 9,83... Haz los cálculos y verás como es así, pero ante todo fíjate como hemos sumado las variables var1 y var2 para que den var3, colocando simplemente el nombre de la variable. Para copiar el valor de una variable a otra solo hay que hacer lo siguiente:

```
var1 = (7*2)+1
var2 = 7*(2/3)
var2 = var1
```

Y de esta forma, var2 no tendrá el valor 4,6... sino 15. Es sencillo, solo tienes que imaginar que donde pone var1 esta el contenido de var1. También puedes hacer **incrementos en una variable**, por ejemplo así:

```
var1 = 15
var1 = var1 + 1
'Mas código y funciones
var1 = var1 + 1
```

Y así, y con un poco de imaginación, se pueden hacer muchas operaciones...

Antes de pasar a otro tipo de valores para las variables, comentar que si habéis ejecutado el script escribiendo los ejemplos dentro, no habrá ocurrido nada de nada. Esto es porque las variables son almacenes, pero para mostrarlas al usuario, se utilizan unas funciones que se explicarán más adelante. Vayamos poco a poco...

Otro tipo de datos que se pueden introducir son cadenas de texto. Hay que tener en cuenta que estos datos no se pueden mezclar con los numéricos, porque no son de la misma naturaleza. Al igual que con los datos numéricos, se asigna mediante el uso del símbolo igual, pero en este caso, el texto debe ir entre comillas, para que no se confundan con nombres de variables. Veamos un ejemplo (*la creación de la variable se obvia*):

```
var1 = "Hola, me llamo Perico y me he comprado un periquito..."
```

Como veis, es sencillo darle a una variable un valor de cadena de texto. Pero eso no es todo, existe un operador especial para cadenas que es la concatenación de cadenas (&). Se usa de forma similar a los operadores matemáticos, pero su función es de unir cadenas de texto. Por ejemplo:

```
nombre = "Perico"
texto = "Hola, me llamo " & nombre & " y me he comprado un periquito..."
```

En este caso, que puede resultar un poco mas complejo, vemos como tenemos una cadena ("Hola, me llamo "), luego concatenamos al contenido de la variable nombre ("Perico"), y por ultimo, concatenamos a otra cadena (" y me he comprado un periquito..."); dando como resultado lo mismo que en el primer ejemplo. Esto se suele utilizar cuando, por ejemplo, el nombre de la persona puede cambiar al ser introducido por el usuario (mas adelante se enseñará como).

Queda claro como introducir cadenas de texto en variables, pero llegados a este punto surge un problema: **No podemos introducir ni comillas ni un salto de línea en las variables**, ya que cometeríamos un error. Para esto se han creado tres funciones que solucionan este problema: el vbCrLf, el Chr y el Asc.

- El vbCrLf:

Esta función no tiene argumentos, y simplemente devuelve el valor de un **salto de línea**, permitiéndonos almacenarlo en una variable. Por ejemplo:

```
var1 = "Hola, me llamo Perico." & vbCrLf & "Me he comprado un periquito."
```

Esto dará como resultado lo siguiente:
Hola me llamo Perico.
Me he comprado un periquito.

Y de esa forma, siempre que queramos, podremos introducir un salto de línea en las variables.

- El Chr:

Esta función tiene un argumento, es decir, cuando la usemos tendremos que introducirle un dato para que nos devuelva otro. En este caso, supongo que sabréis que es el código **Ascii**. Y si no, buscad por Internet información relacionada con esto. Pero a lo que vamos, esta función se le introduce el código numérico correspondiente a un carácter ascii para que devuelva el carácter correspondiente. Con esto, y sabiendo que las comillas se corresponden con el 34, pongamos un ejemplo:

```
var1 = "Hola, me llamo " & Chr(34) & "Perico" & Chr(34) & "."<7a>
```

Esto dará como resultado lo siguiente:
Hola me llamo "Perico".

Y de esa forma, siempre que queramos, podremos introducir comillas en las variables. Si se os ocurre cambiar el numero que hay dentro del Chr, veréis que los caracteres que devuelve son diferentes.

- El Asc:

Esta función es simplemente la inversa del Chr. En este caso debemos introducir el carácter entre comillas y nos devolverá el código Ascii de ese carácter. Por ejemplo:

```
var1 = Asc("a")
```

Esto dará como resultado el numero 97, que si se coloca en la función Chr, dará "a". Esta función se suele utilizar para la encriptación de datos.

Para concluir con esta explicación, intentad sabed que valor tendrá ResX en el siguiente ejemplo: (**clic aquí para ver solución**)

```
var1 = "Pedro"
var2 = " y "
ResX = "Hola, me llamo " & Chr(34) & var1 & Chr(34) & "." & vbCrLf & "Soy guay" &
var2 & "guapo."
```

Y hasta aquí la explicación de como introducir valores de cadena en las variables. Recordad que antes de usarlas debéis crearlas con el Option Explicit y el Dim.

Otro tipo de valor que se utilizará mucho en las variables es el booleano. Es decir, darle el valor Verdadero o Falso a una variable, simulando así un enunciado de la lógica matemática (filosofía), ya que funciona igual. Me explico (se obvia la creación de las variables):

```
var1 = false
var2 = true
```

De esta forma, var1 tiene el valor falso y var2 el valor true. Normalmente, este tipo de valores no se mezcla con los numéricos y los de cadena, pero si se diese el caso, las equivalencias serían:

true	"Verdadero"	1
false	"Falso"	0

Pero ya he dicho, que mezclar los valores, **no sería del todo correcto**, y, claro está, no sería normal...

Como en los demás casos, se pueden realizar operaciones para darle valores a las variables. En este caso, serían operadores lógicos, que son los siguientes:

- And

El operador And (adición) se utiliza con dos operadores (como si fuera la suma), devuelve un valor Verdadero/Falso según la tabla siguiente:

1º Operador	2º Operador	Resultado
True	True	True
True	False	False
False	True	False
False	False	False

De forma que si por ejemplo tenemos lo siguiente, el resultado de var2 sería False:

```
var1 = false
var2 = (true And false) And var1
```

Como veis, también **se pueden introducir variables** con valores booleanos en las operaciones...

- Or

El operador Or (disyunción) se utiliza con dos operadores (igual que el And), devuelve un valor Verdadero/Falso según la tabla siguiente:

1º Operador	2º Operador	Resultado
True	True	True
True	False	True
False	True	True
False	False	False

De forma que si por ejemplo, tenemos el ejemplo anterior con el Or, el resultado de var2 sería True:

```
var1 = false
var2 = (true Or false) Or var1
```

Seguro que los que habéis estudiado **lógica matemática** esto os suena mucho...

- Xor

El operador Xor (disyunción exclusiva) se utiliza con dos operadores y devuelve un valor Verdadero/Falso según la tabla siguiente:

1º Operador	2º Operador	Resultado
True	True	False
True	False	True
False	True	True
False	False	False

Y, seguro que con los ejemplos anteriores, podéis aclararos: Solo hay que sustituir el Or por el Xor y comprobar el resultado.

- Not

El Not (negador) es un operador lógico que utiliza un **solo operando**, devolviendo el valor contrario que tiene. Es decir, que en el ejemplo siguiente, devolverá False:

```
var1 = Not true
```

Hay que fijarse que el Not va delante del valor y no detrás...

- El Eqv

El Eqv realiza la equivalencia lógica de dos valores booleanos. Se utiliza igual que el And o el Or, pero los valores que devuelve vienen dador por la siguiente tabla:

1º Operador	2º Operador	Resultado
True	True	True
True	False	False
False	True	False
False	False	True

- El Imp

El Imp realiza la implicación lógica de dos valores booleanos. Se utiliza igual que el Eqv, pero los valores que devuelve vienen dador por la siguiente tabla:

1º Operador	2º Operador	Resultado
True	True	True

True	False	False
False	True	True
False	False	True

Y con esto, ya está todo sobré el uso básico de las variables explicado: La creación, los valores numéricos, los valores de cadena y los valores boléanos.

Programación en Visual Basic Script: Control del Flujo

El control del flujo en un script es esencial, y para ello veremos diversas funciones que lo permiten. La primera es la sentencia If. La sentencia If, recibe el nombre de sentencia porque no va en una sola línea, y además, altera el flujo de ejecución del script: es un condicional. Me explico, la sentencia If se encarga de evaluar una expresión y ejecutar un trozo de script si se cumple. Pongamos un ejemplo:

```
If var1 = 0 Then
'Funciones que se ejecutaran si la condición se cumple (variable es igual a 0)
Else
'Funciones que se ejecutaran si la condición no se cumple (variable diferente a 0)
End If
```

Como veis, esta sentencia **no se coloca en una sola línea, sino en varias**. Se encarga de que, si la variable var1 es 0, se ejecuten unas acciones, mientras que si no lo es, se ejecuten otras. La palabra If indica el inicio de la condicional, luego le sigue una expresión que indica la condición, luego la palabra Then, que indica la ejecución de unas funciones si la expresión es verdadera. Mas abajo se encuentra el Else, que se podría traducir como 'si no', y finalmente, el End If que indica el final de la condicional.

¿Liado? Es más sencillo de lo que parece. Veamos otro ejemplo:

```
var1 = 3
If var1 = 0 Then
var2 = "El valor introducido es 0"
Else
var2 = "El valor introducido no es 0"
End If
```

Esta claro que en este caso var2 obtendrá el valor "El valor introducido no es 0", porque si damos a var1 el valor 3, y luego decimos que si es 0 reciba un valor, y si no otro, claro está, que el valor que recibirá es el de debajo del Else.

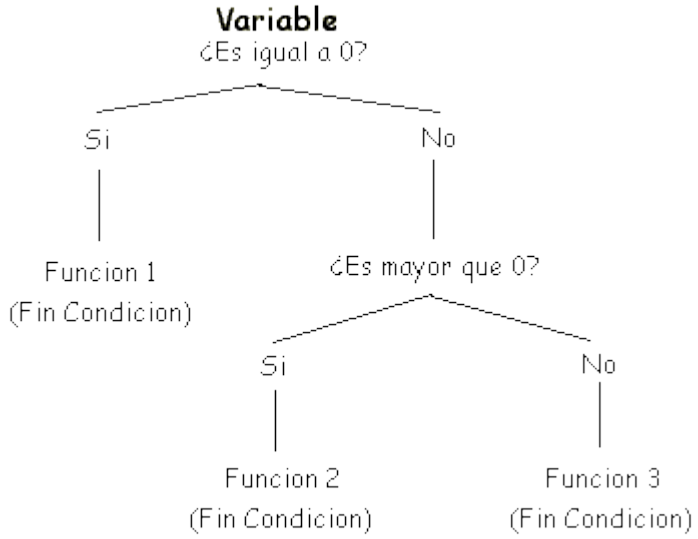
Bien, ahora fijémonos en la expresión que impone una condición: var1 = 0. El signo igual no es el único que se puede utilizar en una condicional, también existen el mayor que (>), menor que (<), menor o igual que (<=), mayor o igual que (>=) y diferente de (<>). De esa forma, en el siguiente ejemplo, se impone una condición diferente a la igualdad:

```
var1 = -3
If var1 < 0 Then
var2 = "El valor introducido es menor que 0"
Else
var2 = "El valor introducido es mayor que 0"
End If
```

Supongo que no tiene mas dificultad, ya que, obviamente, var2 obtendrá el valor "El valor introducido es menor que 0". Es necesario saber que también se pueden introducir **condicionales dentro de condicionales**, parece lioso, pero no lo es tanto. Pongamos un ejemplo, que puede resultar ya un poco más complejo:

```
If variable = 0 Then
'Funciones que se ejecutaran si variable es igual a 0. (Función 1)
Else
If variable > 0 Then
'Funciones que se ejecutaran si variable es diferente a 0 y es mayor que 0. (Función 2)
Else
'Funciones que se ejecutaran si variable es diferente a 0 y es menor que 0. (Función 3)
End If
End If
```

Esto daría como resultado algo así:



Hemos visto la sintaxis básica de la sentencia If, pero hay otras formas mas cortas de usarla, si cumplimos una serie de condiciones. Si recordáis, la sintaxis de la sentencia If, era así:

```
If var1 = 0 Then
'Funciones que se ejecutaran si la condición se cumple (variable es igual a 0)
Else
'Funciones que se ejecutaran si la condición no se cumple (variable diferente a 0)
```

```
End If
```

Como vemos, en este caso podemos ejecutar *una serie de acciones si la condición se cumple y otras si no lo hace*. Pero si a nosotros solo nos interesa cuando se cumple la condición, podemos usar una **variante de la sentencia**, mas acortada:

```
If var1 = 0 Then
'Funciones que se ejecutarán si se cumple la condición.
End if
```

Como veis, de esta forma se ejecutan solo acciones cuando la condición se cumple, pero no cuando no se cumple. Es una forma mas abreviada de utilizar la sentencia if, si solo nos interesa saber si se da o no una condición.

Pero aun existe otra forma mas corta aun de utilizar la sentencia if, que es la siguiente:

```
If var1 = 0 Then var2 = true
```

Es decir, **todo en una línea**. Este se utiliza cuando solo se quiere ejecutar una orden si la condición se cumple. En este caso, la orden es var2 = true, pero podría haber sido cualquier otra orden, pero solo una.

Y, esto es lo más básico en lo que respecta al uso de la Sentencia If en un script.

Otra de las sentencias que alteran el flujo es la **sentencia Do…Loop**. La **sentencia Do...Loop** sirve para **crear un bucle** del que solo se puede salir si una variable contiene un valor determinado. Se coloca en el script de la siguiente manera:

```
Do
'Funciones que se ejecutaran infinitamente hasta que Variable sea igual a 1.
Loop Until Variable="1"
```

En la primera línea colocamos la partícula Do que indica el comienzo del bucle. Cabe destacar que en Visual Basic 6, es aquí, y no al lado del Loop donde va el Until Variable = 1, por si algún día avanzáis mas en la programación que no se os olvide.

En la segunda línea estarían las funciones que se irían realizando continuamente hasta que la condición de la tercera línea se cumpliera. Si no se cumple, el bucle es infinito: no pararía nunca de repetirse…

En la tercera línea encontramos la partícula Loop que indica que vuelva al punto donde esta la partícula Do. A la derecha de Loop se coloca Until y el nombre de una variable seguido de un operador de comparación (Igualdad (=), Desigualdad (<>), Menor que (<), Mayor que (>), Menor o igual que (<=) o Mayor o igual que (>=)) y una expresión (similar a la expresión que se coloca en la Sentencia If) que hará que si se cumple, se salga del bucle y se continúe la ejecución del script. En realidad su uso es muy parecido al de la Sentencia If, solo que modifica el flujo de otra forma.

Un apunte: La partícula Until se puede omitir, pero esto crearía un bucle infinito que solo podríamos cerrar apagando el ordenador o cerrando el proceso Wscript.exe mediante el administrador de tareas. En este caso el script se quedaría así:

```
Do
'Funciones que se ejecutaran infinitamente.
Loop
```

Un ejemplo de uso de **la sentencia Do** con las variables seria el siguiente (se obvia la creación de la variable con el Option Explicit y el Dim):

Variable = 1	Le damos a Variable el valor 1.
Do	Creamos bucle.
variable=variable+1	Añadimos a variable 1 a su valor actual.
Loop Until variable = "6"	Salimos del bucle cuando la variable es 6.

Esto hace que **el bucle se repita un total de 5 veces**, ya que el valor inicial de Variable es 1, y para romper el bucle (una vez entrado en el) Variable debe tomar el valor 6, y por tanto 6-1 = 5 repeticiones. Para que lo entendáis mejor, este ejemplo anterior es como poner el siguiente script, solo que como ya veremos, habrá veces que no sabremos cual es el valor de variable en ese momento, o las veces que se debe repetir, etc:

```
Variable = 1
variable=variable+1
variable=variable+1
variable=variable+1
variable=variable+1
variable=variable+1
```

También podemos meter un bucle dentro de otro, o un bucle dentro de una sentencia If, o la sentencia If en el bucle, etc… Las posibilidades son infinitas, y cada vez van aumentando. Recomiendo que **se te quede claro todo lo visto** hasta ahora antes de continuar con la siguiente sentencia.

Continuemos, pues. La sentencia Do…Loop…Until crea, como hemos visto, bucles de condición. Es decir, si se cumple la condición que se expresa se sale del bucle, o bien se mantiene en el bucle mientras la "*Not condicion*" se cumple. Pero que pasa si lo que queremos es un bucle de repetición; es decir, un bucle que se repita X veces y punto. Pues para eso tenemos la **Sentencia For…Next**.

La **sentencia For** sirve para **crear un bucle** a partir de una variable (a la que se le da un valor numérico inicial) que rompe ese bucle al llegar a un determinado valor. Se expresa de la siguiente forma:

```
For variable = 0 To 10 Step 2
'Funciones
Next
```

Esto hará que el bucle se repita hasta que variable (que al principio toma el valor 0) tome el valor 10, sumando automáticamente 2 cada vez que el bucle se va a repetir.

En la primera línea colocamos la partícula For seguida del nombre de una variable. A esta mediante el signo igual (=) se le asigna un valor inicial. A continuación colocamos la partícula To seguida de un valor final, que hará que el bucle se detenga al alcanzarlo. Luego va la partícula Step, que indica el incremento de la variable en cada repetición. Cada vez que se pasa por la partícula Next a la variable se le suma el valor de después de Step, y de esa forma el bucle se romperá cuando la variable alcance o supere el valor final.

En la segunda línea se colocan las funciones que se ejecutaran durante el bucle. Estas se repetirán hasta que variable alcance el valor final. Por supuesto aquí pueden haber tantas líneas de código como se desee, y se pueden colocar dentro todos los alteradores del flujo que hay (excepto posteriormente el Function).

En la tercera línea colocamos la partícula Next que hace retornar el bucle a la partícula For y añade al valor de la variable el valor de paso (después de Step). Si al llegar aquí el valor de la variable es mayor que el final el bucle se romperá y el script continuará su ejecución.

Un ejemplo de uso de esta sentencia seria el siguiente (de nuevo, obviamos la creación de las variables):

```
For variable=1 To 10
variable2=variable 1 * variable2
Next
```

Como vemos, hemos omitido el **Step X**, lo cual le indica al programa que el paso es de 1, o lo que es lo mismo: poner Step 1 es lo mismo que omitirlo. En este caso, el bucle se repetiría 10 veces, el valor final de variable1 sería 10 y el de variable2… Quien sabe…

Pero sin duda la forma más difícil de usarlo, con resultados mas extraños, pero siempre útil saberlo sería la siguiente:

```
numero = 1
For variable=numero - 1 To numero +1 Step numero
numero = numero +1
variable = variable - 1
Next
```

Aunque parezca raro, es completamente legal. La situación inicial sería *For variable = 0 To 2 Step 1*, pero tras pasar el bucle una vez la situación sería *For variable = 0 To 3 Step 2*, y a la tercera *For variable = 0 To 4 step 3*… Lo cual crearía… Si: Un bucle infinito. Bueno, en este caso lo que pasaría sería un error de desbordamiento de memoria cuando los números fuesen muy grandes, pero bueno… Sin duda este ejemplo deja abiertas muchas puertas en cuanto a sucesiones numéricas y sistemas de ecuaciones se trata (si aunque parezca que no…). Pero bueno, volvamos a lo sencillo con otro ejemplo:

```
For variable=1 to 10
  ‘Funciones que se ejecutaran 10 veces.
  For variable2=1 to 2
    ‘Funciones que se ejecutaran 20 veces.
  Next
Next
```

En este ejemplo, vemos un bucle dentro de otro. En este tipo de bucles hay que tener en cuenta que si el bucle principal, en este caso se repite diez veces y el secundario se repite dos veces, como el secundario esta dentro del principal, **cada vez que el principal se repita una vez el secundario lo hará dos** y de esa forma el secundario se repetirá veinte veces. En realidad no es difícil. Hace falta un poco de práctica y ya está. Pongamos un último ejemplo sencillo:

```
For variable=1 to 10
  ‘Funciones que se repetirán 10 veces…
  If variable = 5 Then
    ‘Funcion que se realizará cuando variable se 5…
  End if
Next
```

Como vemos, podemos incluir condicionales dentro de bucles, bucles dentro de bucles, condicionales dentro de condicionales y bucles dentro de condicionales. Las posibilidades son muchas y variadas, y he de recordar que esto es por ahora lo más básico de la programación en Visual Basic Script, cuando lo sepáis todo, la cosa dará a juego a muchísimo, mas de lo que os creéis; pero como en todo, lo que se necesita ante todo es práctica y práctica, así que animo.

Por último en cuanto a control de flujo básico se refiere, cabe destacar la Sentencia Function. La **sentencia Function** sirve para **crear una función personalizada** a partir de unas **otras funciones simples,** que puede ser llamada desde cualquier parte del código. Se expresa de la siguiente forma y se coloca generalmente o al principio o al final del script, por motivos de estandarización de código:

```
Function nombrefuncion(variables)
  ‘Funciones simples
End Function
```

Donde **nombrefuncion** es el nombre que le queremos dar a la función que hace la tarea de una variable, y **variables** son las variables que operaran en ella a las que le daremos un valor desde fuera, digamos que equivale por ejemplo, en la función de **Chr** vista antes a sus diversas partes. **Chr** sería en **nombrefuncion** y el valor **numérico** que ponemos dentro de chr sería **variables**. Además, lo que hay entre Function y End Function no se ejecutará nunca en el código a no ser que la llames de la siguiente forma:

```
Variable = nombrefunción(variable)
```

Pongamos un ejemplo, para aclarar las cosas de uso de esta función con unas variables al azar (se obvia la creación de las variables):

```
Variable = 2
Function Incrementar(numero)
Incrementar = numero + 1
End Function
Variable 2 = Variable
Variable = Incrementar(Variable2)
```

Como veis, hemos puesto la sentencia en medio del código, pero esto es indiferente pues no se ejecutará. Aun así lo más correcto habría sido ponerla al principio o al final, como se muestra a continuación (el siguiente código es igual que el anterior):

```
Variable = 2
Variable2 = Variable -1
Variable = Incrementar(Variable2+Variable)

Function Incrementar(numero)
Incrementar = numero + 1
End Function
```

Ahora vamos a lo importante: expliquemos. Primero **Variable** recibe el valor 2, y **Variable2** el valor 1. Hasta ahí claro. Luego, vemos que se le aplica a **Variable** la función incrementar de una suma (que da 3). Entonces lo que ocurre es que se llama a la función que tenemos debajo de la siguiente forma (en negrita esta **numero**):

Function Incrementar(**1+2**)
Incrementar = (**1+2**) + 1
End Function

En tal caso, la función incrementar recibe el valor 1+2+1 = 4, y se lo pasa a **Variable**, por tanto, **Variable** ahora tiene el valor 4. No es difícil, solo abre tu ingenio. Veamos otro ejemplo sencillo de esto:

```
Variable = 0
Variable2 = 100
For x = 1 To 50
  Variable = Incrementar(Variable)
  Variable2 = Decrecer (Variable2)
Next
Function Incrementar(numero)
Incrementar = numero + 1
End Function

Function Decrecer(numero)
Decrecer = numero - 1
End Function
```

En este caso tenemos dos funciones, y el código en general lo que haría sería dar a Variable el valor 0 y a Variable2 el valor 100, y luego repetir 50 veces un bucle donde

uno se incrementa y el otro decrece. Eso significa que este código y el que hay a continuación realizan lo mismo:

```
Variable = 0
Variable2 = 100
For x = 1 To 50
  Variable = Variable + 1
  Variable2 = Variable2 - 1
Next
```

Como he dicho, hacen lo mismo, y a primera vista el segundo es más sencillo que el primero. Pero no lo he explicado todo sobre esta función, ahora veréis la verdadera utilidad, que reside en que **variables** (ej. anterior **numero**) pueden ser varias. A continuación un ejemplo que no deja de mostrar lo interesante que puede llegar a ser esto:

Variable = 0	Al principio Variable tiene el valor 0.
For x = 1 To 100	Empezamos un bucle de 100 repetición.
If Variable <= 50 then	Si variable es <= 50… llamamos a la
Variable = matematicas(Variable, True, 2)	función sumando a variable 2.
Else	Si no, llamamos a la función restando
Variable = matematicas(Variable, False, 1)	a variable 1.
End if	Fin de la Condicional.
Next	Fin del Bucle.
Function matematicas(numero,modo,valor)	Función con tres argumentos.
If modo = True Then	Si modo es Verdadero sumamos.
matematicas = numero + valor	Sumamos valor al numero dado.
Else	Si modo es Falso restamos.
Matematicas = numero - valor	Restamos valor al numero dado.
End If	Fin de la Condicional
End Function	Fin de la Función.

Este ejemplo es largo pero no por ello complejo. Si bien, este es nuestro primer encuentro con funciones complejas. Solo decir que en una función existe lo que se llaman **argumentos**. En la función Chr(3) hay un argumento que es numérico. En el caso de esta hay tres argumentos (**los argumentos se separan por comas**), dos numéricos y el de en medio booleano. Sabido esto, no hay más misterio con el código. Mas adelante, cuando explique más funciones, no solo os acostumbrareis, sino que lo comprenderéis mejor. Aún es pronto para querer hacer nada serio.

Programación en Visual Basic Script: Funciones Gráficas

Antes de empezar con las funciones ya mas serias (pero aún básicas), es el momento de aprender las funciones gráficas, mas que nada para poder empezar a practicar y saber con seguridad que está pasando en nuestro script. Estas funciones son dos: **El Cuadro de Dialogo y El Cuadro de Mensaje**, y sirven para mostrar la información de las

variables en pantalla, y/o interactuar con el usuario dejando que sea el quien de valores a las variables. Empecemos pues por el Cuadro de Mensaje, que es la función MsgBox.

La función MsgBox crea un cuadro de mensaje con el siguiente aspecto (puede variar):

Del cuadro de mensaje puedes **editar** el **mensaje**, el **titulo**, la **imagen de la izquierda** y los **botones que aparecen**. Por otro lado, el cuadro de mensaje va unido a una variable, para poder saber cual de todos los botones se ha pulsado.

El cuadro de mensaje básico (el tercero de los que hay arriba) se hace con el siguiente código, que es la versión más simple de todas (obviamos de nuevo la creación de variables):

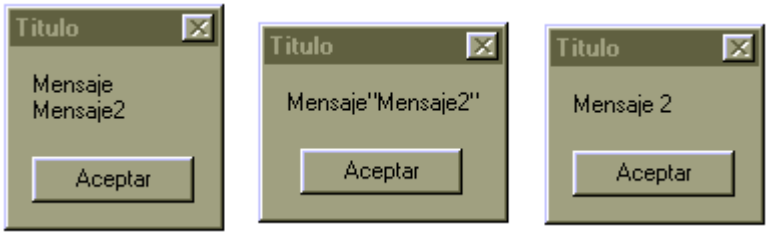
```
Variable = MsgBox("Mensaje",0,"Titulo")
```

Donde **Variable** es el nombre de la variable que acompaña al MsgBox y recibirá información del botón pulsado, **Mensaje** es el mensaje que se muestra en el cuadro de mensaje (argumento de texto, que puede suplirse por una variable de cadena), **Titulo** es el titulo de este (del mismo tipo que Mensaje) y 0 es el numero magico que indica la imagen, el numero de botones, el botón predeterminado y el tipo de mensaje a la vez. En breve explicaré detalladamente como tratar este numero "mágico", por ahora va siendo hora de que abras el bloc de notas y empieces a probar.

Si en el Mensaje desea colocar comillas, saltos de línea o una referencia a otra variable (es decir, que muestre lo que tiene) debe realizarlo **uniendo las expresiones** con el **símbolo de concatenación** (&) de la siguiente manera (igual que dándole valor a una variable de cadena):

```
Variable = MsgBox("Mensaje"&chr(13)&"Mensaje2",0,"Titulo")
Variable = MsgBox("Mensaje"&chr(34)&"Mensaje2"&chr(34),0,"Titulo")
Variable2 = 2
Variable = MsgBox("Mensaje "&Variable2,0,"Titulo")
```

Esto quedaría respectivamente así, probadlo y comprobadlo:



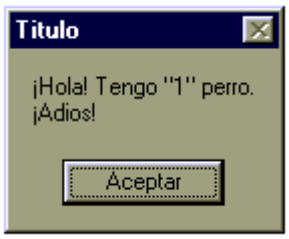
En el primero, hay dos mensajes en dos líneas. El mensaje uno esta unido con un símbolo de concatenación a una constante de VBScript que equivale a un salto de línea (se explicó anteriormente), y este, con otro símbolo de concatenación al segundo mensaje. Todo esto esta puesto en la zona del mensaje **separado de las demás con una coma fuera de las comillas**.

En el segundo hay un mensaje y otro entre comillas. El mensaje uno esta unido con un símbolo de concatenación a la función Chr(x) que equivale a las comillas ("), y este, con otro símbolo de concatenación al mensaje dos. Después esta otra constante igual a la anterior unida con otro símbolo de concatenación para así cerrar las comillas que habíamos abierto anteriormente de la misma manera.

En el tercero primero damos el valor dos a la variable "*Variable2*", aunque le podeis dar cualquier valor, numérico o de cadena. Lugo ponemos un mensaje unido con un símbolo de concatenación a la variable "*Variable2*" lo cual nos mostrara solo su valor. Es decir, que podemos mostrar en pantalla el valor de las variables.

El siguiente ejemplo muestra una mezcla de los tres casos anteriores que son los más comunes, y lo que más se suele utilizar:

```
Variable2 = "¡Hola! "
Variable3 = 1
Variable = MsgBox(Variable2&"Tengo "&chr(34)&Variable3&chr(34)&"
perro."&chr(13)&"¡Adios!",0,"Titulo")
```



Si metemos esto dentro de un bucle For, de forma que Variable3 aumente (*For variable3=1 To 10*), tendremos un total de 10 mensajes en los que irá aumentando la cantidad de perros que tienes. **No estaría mal, una vez llegados aquí, repasar todo lo visto mostrando los resultados con esta función para comprobar sus efectos y practicar**.

Pero continuemos, en el **segundo campo**, (el que contiene el numero “magico”) hay que introducir un numero, el cual definirá la imagen, el numero de botones, el botón predeterminado y el tipo de mensaje. Para saber que numero hay que poner para que te salga un mensaje así o asa, debes escoger una opción de cada una de las tablas siguientes y sumar sus valores, y el resultado colocarlo en lugar del 0:

Alternativo*	Valor	Función que realiza
vbOKOnly	*0*	*Muestra sólo el botón **Aceptar**.*
vbOKCancel	*1*	*Muestra los botones **Aceptar** y **Cancelar**.*
vbAbortRetryIgnore	*2*	*Muestra los botones **Anular**, **Reintentar** e **Ignorar**.*

vbYesNoCancel	*3*	*Muestra los botones **Sí**, **No** y **Cancelar**.*
vbYesNo	*4*	*Muestra los botones **Sí** y **No**.*
vbRetryCancel	*5*	*Muestra los botones **Reintentar** y **Cancelar**.*
Alternativo*	**Valor**	**Función que realiza**
	0	*No muestra iconos.*
vbCritical	*16*	*Muestra el icono **Mensaje crítico**.*
vbQuestion	*32*	*Muestra el icono **Consulta de advertencia.***
vbExclamation	*48*	*Muestra el icono **Mensaje de advertencia.***
vbInformation	*64*	*Muestra el icono **Mensaje de información.***
Alternativo*	**Valor**	**Función que realiza**
vbDefaultButton1	*0*	*El **primer botón** es el predeterminado.*
vbDefaultButton2	*256*	*El **segundo botón** es el predeterminado.*
vbDefaultButton3	*512*	*El **tercer botón** es el predeterminado.*
vbDefaultButton4	*768*	*El **cuarto botón** es el predeterminado.*
Alternativo*	**Valor**	**Función que realiza**
vbApplicationModal	*0*	*Cuadro de **diálogo modal de la aplicación**. El usuario debe responder al cuadro de diálogo antes de continuar trabajando en la aplicación actual.*
vbSystemModal	*4096*	*Cuadro de **diálogo modal del sistema**. Se suspenden todas las aplicaciones hasta que el usuario responda al cuadro de mensaje.*

*Alternativo significa que en vez de poner un numero también puedes colocar ese nombre uniéndolos con el signo de suma (+). Ej: vbOKOnly+vbCritical sería lo mismo que 0+16 o bien 16 (la suma echa).

De esta forma, si yo por ejemplo quiero un mensaje con los botones **Sí** y **No,** con el **icono de información** y que el botón **No** sea el predeterminado tendré que sumar 4, 64 y 256: 4+64+256= 324 Y de esa forma sustituiré el 0 por 324:

```
Variable = MsgBox("Mensaje",324,"Titulo")
```

Lo cual nos mostrará el interesante siguiente cuadro de Mensaje:

Ahora que tenemos en juego varios botones, nos interesa saber cual pulsa el usuario. Así que, por último, la Variable asignada a la función tomara unos valores dependiendo del botón que se pulse cuando salga el mensaje, según la siguiente tabla:

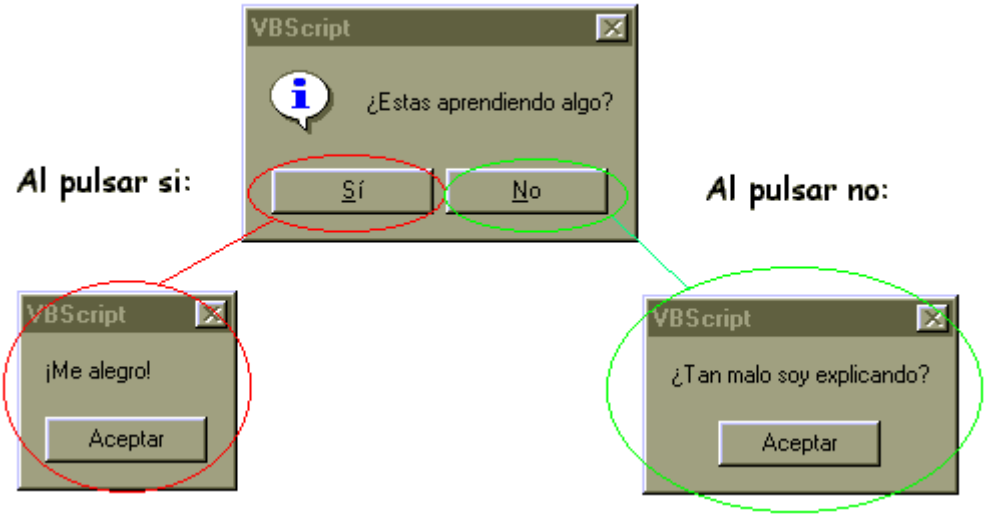
Valor que recibe la Variable	**Botón pulsado**
1	Aceptar

2	Cancelar
3	Anular
4	Reintentar
5	Ignorar
6	Sí
7	No

Esto combinado con la **sentencia If (***explicada anteriormente***)** y otros **MsgBox** puede dar como resultado una respuesta inteligente, bastante interesante, como por ejemplo:

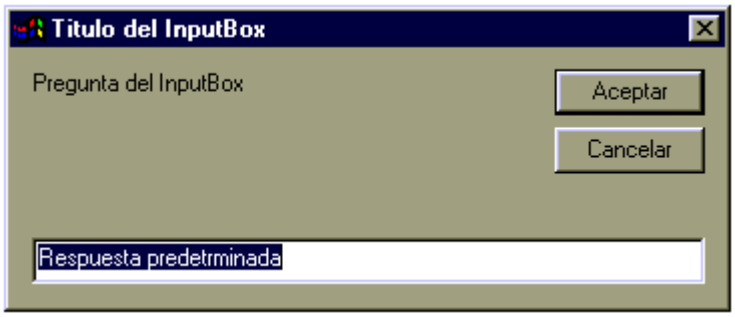
Variable = MsgBox("¿Estas aprendiendo algo?",324,"VBScript")
If Variable = 6 Then
Variable2 = MsgBox("¡Me alegro!",0,"VBScript")
Else
Variable3 = MsgBox("¿Tan malo soy explicando?",0,"VBScript")
End If
	Aparece un mensaje con los botones Si y No…
	Si pulsamos el botón si...
	...sale un MsgBox con el botón Aceptar.
	Si no pulsamos el botón si...
	...sale un MsgBox con el botón Aceptar.
	Fin de la condición.

Obviamente, los **MsgBox** están en la **misma línea** (no partidos), pero aquí por causa del tamaño del papel salen partidos. Esto dará como resultado la siguiente imagen. Cosa que con un poco de imaginación, nos abre las fronteras de lo que podemos hacer a estas alturas (¡y con lo poco que sabemos!):



Sin duda la cosa ahora a mejorado bastante, e insisto de nuevo en *repasar todo lo visto*, o bien ahora, o bien al final de ver el **InputBox**, también llamado **Cuadro de Dialogo**.

La función InputBox crea una ventana de la siguiente forma, lo cual, si lo pensáis bien, abre más aun nuestros horizontes, aunque realmente sea una ventana un poco fea:



Del cuadro de dialogo o InputBox puedes **editar** el **mensaje**, el **titulo,** la **respuesta predeterminada** y la **posición en la pantalla** donde aparecerá. Por otro lado, el cuadro de dialogo va también unido a una variable, que recibirá la respuesta.

El **cuadro de dialogo básico** (sin determinar la posición en la que aparecerá, es decir, aparecerá centrado) se hace con el siguiente código (todo en la misma linea):

```
Variable = InputBox("Pregunta del InputBox","Titulo del InputBox","Respuesta predetrminada")
```

Donde **Variable** es el nombre de la variable que acompaña al **InputBox**, **Pregunta** del InputBox es el mensaje que se muestra en el cuadro de dialogo, **Titulo** del InputBox es el titulo de este y **Respuesta** predeterminada es la respuesta que aparecerá al principio, luego editable. Como vemos es una función de tres argumentos de cadena.

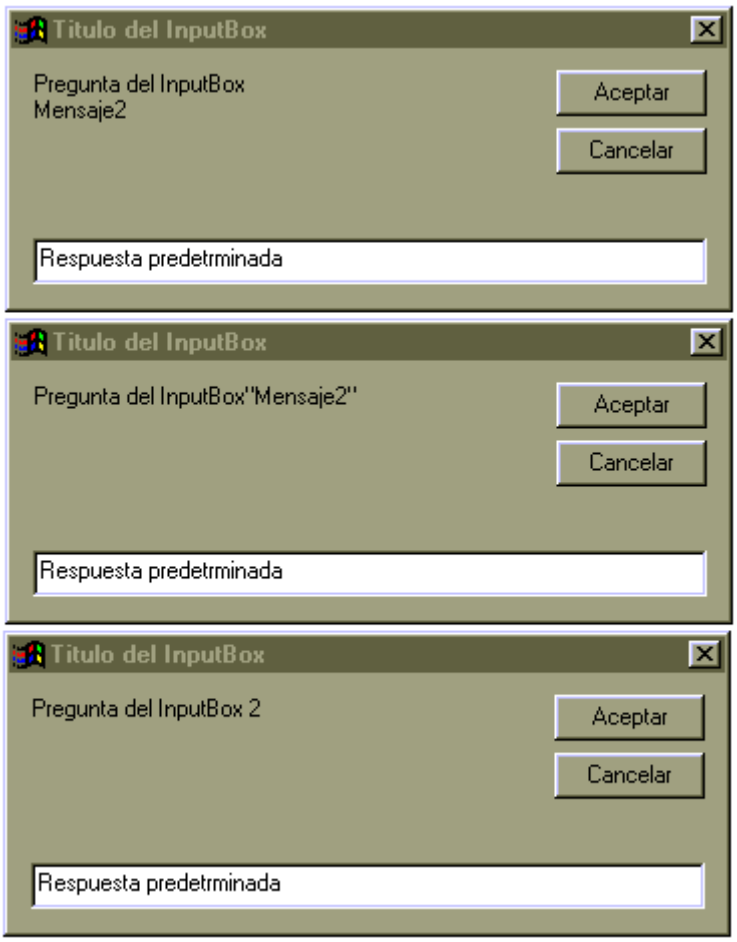
Si en la Pregunta del InputBox desea colocar comillas, saltos de línea o una referencia a otra variable (es decir, que muestre lo que tiene) debe realizarlo **uniendo las expresiones** con el **símbolo de concatenación** (&) de la siguiente manera (la respuesta y el título son única y exclusivamente de una línea):

```
Variable = InputBox("Pregunta del InputBox"&chr(13)&"Mensaje2","Titulo del InputBox","Respuesta predeterminada")
```

```
Variable = InputBox("Pregunta del InputBox"&chr(34)&"Mensaje2"&chr(34),"Titulo del InputBox","Respuesta predeterminada")
```

```
Variable2 = 2
Variable = InputBox("Pregunta del InputBox "&Variable2,"Titulo del InputBox","Respuesta predeterminada")
```

Esto quedará respectivamente así:



En el primero, hay dos mensajes en dos líneas. El mensaje uno esta unido con un símbolo de concatenación a una constante de VBScript que equivale a un salto de línea, y este, con otro símbolo de concatenación al segundo mensaje. Todo esto esta puesto en la zona del mensaje **separado de las demás con una coma**.

En el segundo hay un mensaje y otro entre comillas. El mensaje uno esta unido con un símbolo de concatenación a la función Chr(x) que equivale a las comillas ("), y este, con otro símbolo de concatenación al mensaje dos. Después esta otra constante igual a la anterior unida con otro símbolo de concatenación para así cerrar las comillas que habíamos abierto anteriormente.

En el tercero primero damos el valor dos a la variable "Variable2", aunque le podéis dar cualquier valor. Lugo ponemos un mensaje unido con un símbolo de concatenación a la variable "Variable2" lo cual nos mostrara solo su valor.

Es bastante sencillo, y muy parecido al **MsgBox**. Además, si queremos posicionar el cuadro en la pantalla debemos de usar el siguiente código:

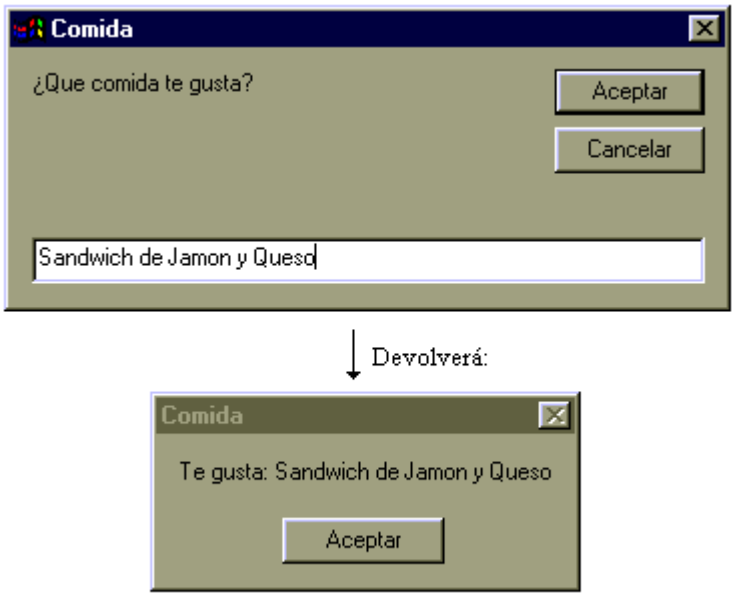
```
Variable = InputBox("Pregunta del InputBox","Titulo del InputBox","Respuesta predetrminada",0,0)
```

Donde el primer 0 es una expresión numérica que especifica, en twips, la **distancia horizontal** del borde izquierdo del cuadro de diálogo respecto al borde izquierdo de la pantalla. Si se omite, el cuadro de diálogo se centra horizontalmente. Y el segundo 0 es una expresión numérica que especifica, en twips, la **distancia vertical** del borde superior del cuadro de diálogo hasta la parte superior de la pantalla. Si se omite, el cuadro de diálogo se ubica verticalmente aproximadamente a un tercio de la parte inferior de la pantalla. Probadlo vosotros mismos y vereis…

Destacar también que la Variable **guarda la respuesta enviada,** si no se escribe nada o se pulsa cancelar la Variable guarda una cadena de longitud cero (""). De esa forma, combinándolo con un **MsgBox** podemos devolver lo que ha colocado en la respuesta:

```
Variable = InputBox("¿Que comida te gusta?","Comida","Introduce comida favorita")
Variable2 = MsgBox("Te gusta: "&Variable,0,"Comida")
```

Esto dará como resultado:



Es pues grande el repertorio de posibilidades que esto ofrece, ya que por ejemplo puedes pedir que te digan “cuantos perros quieres contar” y luego contarlos con el **MsgBox** y un bucle **For**, por ejemplo. Las posibilidades dependen solo de tu imaginación y de nada más.

Por último, para delimitar varias respuestas, se puede seguir el siguiente procedimiento mediante el uso de la **sentencia If**, la **sentencia Do** y las **operaciones con Variables**:

Do Variable = InputBox("¿Qué te gusta mas, el Jamon, el Queso o el Atun?","Comida","Responde aquí") If Variable="Queso" Then	Creamos bucle. Preguntamos con un InputBox. Si la respuesta es Queso: Variable2 obtiene valor ok

Variable2="ok"
End If
If Variable="Jamon" Then
Variable2="ok"
End If
If Variable="Atun" Then
Variable2="ok"
End If
Loop Until Variable2 = "ok"
	Fin de la Condición
	Si la respuesta es Jamon:
	Variable2 obtiene valor ok
	Fin de la Condición
	Si la respuesta es Atun:
	Variable2 obtiene valor ok
	Fin de la Condición
	Salimos del bucle si Variable2= ok.

Esto impedirá que desaparezca el cuadro de dialogo si no se contesta una de esas tres respuestas (Queso, Jamón o Atún). En otras palabras, delimitas las contestaciones posibles, pudiendo crear así un sistema de claves o algo similar.

Programación en Visual Basic Script: Funciones Básicas

Visual Basic Script posee diversas clases de funciones básicas: las que operan con cadenas, con números, con valores boléanos y con tiempo. Las que operan con valores boléanos ya se han visto y tratado anteriormente. Pasemos pues a las que operan con valores numéricos, que son las más sencillas. Los valores numéricos que guardan las variables tienen varias funciones que los pueden modificar. Antes de empezar a ver las funciones, es recomendable que sepas ya bien las **variables en Visual Basic Script.**

La primera función es ***Fix***. Esta función elimina la parte decimal de un número. Si no hay decimales lo deja como está. Un ejemplo de uso de esta función es el siguiente:

Variable = "4,567" Variable2 = **Fix(**Variable**)**	En este caso Variable tiene el valor 4,567. Tras aplicarle la función ***Fix*** a Variable, Variable2 obtiene el valor 4.

La segunda función es ***Abs***. Esta función devuelve el valor absoluto de un número. Si el número es positivo lo deja como está. Un ejemplo de uso de esta función es el siguiente:

Variable = "-4,567" Variable2 = **Abs(**Variable**)**	En este caso Variable tiene el valor -4,567. Tras aplicarle la funcion ***Abs*** a Variable, Variable2 obtiene el valor 4,567.

La tercera función es ***Rnd***. Esta función devuelve un numero aleatorio decimal (0,33), y por eso es conveniente multiplicarlo por 100 y luego aplicarle la función ***Fix***. Como vemos, se pueden pues mezclar y poner unas funciones dentro de otras sin problemas. Un ejemplo de uso de esta función es el siguiente:

Randomize Variable = **Fix(Rnd*100)**	En este caso Variable recibirá un numero aleatorio decimal que al multiplicarlo por 100 dará un numero aleatorio decimal con dos números enteros. Al aplicarle la función Fix se convertirá en un número aleatorio entero de dos cifras.

La cuarta función es ***Sgn***. Esta función devuelve 1 si el signo del valor de Variable es positivo, -1 si es negativo y 0 si su valor es 0. Un ejemplo de uso de esta función es el siguiente:

Variable = "-4,567" Variable2 = **Sgn(**Variable**)**	En este caso Variable tiene el valor -4,567. Tras aplicarle la función ***Sgn*** a Variable, Variable2 obtiene el valor -1.

La quinta función es ***Sqr***. Esta función devuelve la raíz cuadrada de un número. Si el numero es negativo genera un error, y por eso es recomendable usar antes la funciona ***Abs***. Un ejemplo de uso de esta función es el siguiente:

Variable = "4" Variable2 = **Sqr(**Variable**)**	En este caso Variable tiene el valor 4. Tras aplicarle la función ***Sqr*** a Variable, Variable2 obtiene el valor 2.

La sexta función es ***Log***. Esta función devuelve el logaritmo natural de un numero, es decir el logaritmo en base e(2,718282) de un numero. Por eso es recomendable usar una división de logaritmos para hallar logaritmos de la base que queramos siguiendo el siguiente ejemplo, donde Variable es el numero al que se le aplica el logaritmo y 8 es la base de este (sustituye 8 por la base sobre la cual quieres operar):

Variable = "64" Variable2 = **Log(Variable) / Log(8)**	En este caso Variable tiene el valor 64. Tras aplicarle la función ***Log*** a Variable con base **8**, Variable2 tiene el valor 2.

Las últimas funciones son ***Cos***, ***Tan*** y ***Sin***. Estas funciones devuelven el coseno, la tangente y el seno respectivamente, de un número que tiene un valor que expresa el ángulo en ***Radianes***. Para **convertir *grados* en *radianes***, multiplique los grados por **pi / 180**. Para convertir los radianes en grados, multiplique los radianes por **180/pi**. Un ejemplo de uso de esta función es el siguiente:

Variable = "3,1415926535897932" Variable2 = **Cos(Variable)**	Variable tiene el valor pi. Tras aplicarle la función ***Cos*** a Variable, Variable2 obtiene el valor -1.

Estas son, en definitiva las funciones numéricas mas usadas, que dan mucho juego junto con las operaciones matemáticas vistas antes. A estas alturas, deberíais probar en intentar hacer un script donde tu pides la longitud de dos lados de un triangulo, y el programa, mediante Pitágoras, saca el tercero; pues es una muy buena práctica.

Por otro lado, las cadenas que guardan las variables tienen varias funciones que las pueden modificar. Estas son:

La primera función es ***LCase***. Esta función convierte toda una cadena a minúsculas. Si esta toda en minúsculas la deja como está. Un ejemplo de uso de esta función es el siguiente:

Variable = "Hola" Variable2 = **LCase(**Variable**)**	En este caso Variable tiene la cadena Hola. Tras aplicarle la función ***LCase*** a Variable, Variable2 tiene “hola”.

La segunda función es ***UCase***. Esta función convierte toda una cadena a mayúsculas. Si esta toda en mayúsculas la deja como está. Un ejemplo de uso de esta función es el siguiente:

Variable = "Hola" Variable2 = **UCase(**Variable**)**	En este caso Variable tiene la cadena Hola. Tras aplicarle la función ***UCase*** a Variable, Variable2 tiene “HOLA”.

La tercera función es ***Left***. Esta función coge un número de caracteres específicos desde la izquierda de una cadena y los guarda en una variable. Esta función, junto con Len y Right es de las más importantes y usadas en la programación (VB). Un ejemplo de uso de esta función es el siguiente, donde Variable es una variable que guarda una cadena y 6 el número de caracteres que quieres coger:

Variable = "Hola a todos" Variable2 = **Left(**Variable,6**)**	En este caso Variable tiene la cadena Hola a todos. Tras aplicarle la función ***Left*** a Variable especificando 6 caracteres, Variable2 obtiene la cadena Hola a.

La cuarta función es ***Right***. Esta función coge un número de caracteres específicos desde la derecha de una cadena y los guarda en una variable. Un ejemplo de uso de esta función es el siguiente, donde Variable es una variable que guarda una cadena y 7 el número de caracteres que quieres coger:

Variable = "Hola a todos" Variable2 = **Right(**Variable,7**)**	En este caso Variable tiene la cadena Hola a todos. Tras aplicarle la función ***Right*** a Variable especificando 7 caracteres, Variable2 obtiene la cadena a todos.

La quinta función es ***Mid***. Esta función coge un número de caracteres específicos desde un punto específico de una cadena y los guarda en una variable. Un ejemplo de uso de esta función es el siguiente, donde Variable es una variable que guarda una cadena, 6 el carácter-1 desde el que empieza a contar y 3, el número de caracteres que quieres coger:

Variable = "Hola a todos" Variable2 = **Mid(**Variable,6,3**)**	En este caso Variable tiene la cadena “Hola a todos”. Tras aplicarle la función ***Mid*** a Variable especificando 3 caracteres desde el carácter 5 (6-1=5), Variable2 obtiene la cadena “a t”.

La sexta función es ***Len***. Esta función lee el número de caracteres de una cadena y devuelve un número con su cantidad. Un ejemplo de uso de esta función es el siguiente:

Variable = "Hola a todos" Variable2 = **Len(**Variable**)**	En este caso Variable tiene la cadena Hola a todos. Tras aplicarle la función ***Len*** a Variable, Variable2 obtiene el valor numérico 12.

La séptima función es ***StrReverse***. Esta función invierte la posición de los caracteres de una cadena, colocando la primera letra la ultima, la última la primera y así sucesivamente. Un ejemplo de uso de esta función es el siguiente:

Variable = "Hola a todos" Variable2 = **StrReverse(**Variable**)**	En este caso Variable tiene la cadena Hola a todos. Tras aplicarle la función ***StrReverse*** a Variable, Variable2 obtiene la cadena “sodot a aloH”.

La octava función es ***Trim***. Esta función elimina los espacios que pueda haber al inicio y final de una cadena de caracteres. Un ejemplo de uso de esta función es el siguiente:

Variable = " Hola " Variable2 = **Trim(**Variable**)**	En este caso Variable tiene la cadena " Hola ". Tras aplicarle la función ***Trim*** a Variable, Variable2 obtiene la cadena "Hola".

La novena función es ***InStr***. Esta función busca un una palabra, dentro de una cadena con alguna frase, y devuelve el numero de veces que la ha encontrado. Los modos de búsqueda son vbTextCompare (Texto: No respeta mayúsculas) y vbBinaryCompare (Binario: Respeta mayúsculas). Por ejemplo, un ejemplo de uso es el siguiente:

Variable = "Hola a todos" Variable2 = **InStr(**1,Variable, "Hola", vbTextCompare)	En este caso Variable tiene la cadena Hola a todos. Tras aplicarle la función ***InStr*** a Variable, Variable2 obtiene el valor 1, por el hola que hay.

La décima y última función es ***Replace***. Esta función busca un una palabra, dentro de una cadena con alguna frase, y la sustituye por otra que le indiquemos nosotros. Los modos de búsqueda son vbTextCompare (Texto: No respeta mayúsculas) y vbBinaryCompare (Binario: Respeta mayúsculas). Esta función también es muy usada, y también sirve para eliminar partes o palabras de un texto, reemplazando por una cadena de longitud 0 (“”). Por ejemplo, un ejemplo de uso es el siguiente:

Variable = "Hola a todos" Variable2 = **Replace(**Variable, "todos", "ella", vbTextCompare)	En este caso Variable tiene la cadena Hola a todos. Tras aplicarle la función ***Replace*** a Variable, Variable2 obtiene la cadena Hola a ella.

Bueno, esto es todo de las funciones de cadena más comunes e importantes. Solo quedan las de tiempo, pero antes de verlas, quiero mostrar un ejemplo del uso conjunto de estas funciones mediante la sentencia Función estudiada anteriormente:

```
Function encriptar(texto,clave)
```

```
 On error resume next: Dim i
 For i = 1 To Len(texto)
   encriptar = encriptar & Chr(Asc(Mid(texto,i, 1)) Xor clave)
 Next
End Function
```

Esta función encripta un texto al aplicarle una clave y desencripta un texto encriptado al introducirle la misma clave. Para usarla hay que poner: Variable=encriptar("Hola",3) donde "Hola" es una cadena entre comillas y 3 un valor numérico entero sin comillas que representa la clave de encriptación. Al usar esto Variable obtendrá Klob y si haces Variable=encriptar("Klob",3), Variable obtendrá Hola. **Fíjate que en los dos casos debe estar la misma clave (3).** Este es un ejemplo de los muchos que se pueden hacer.

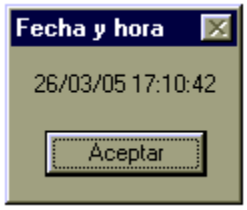
Por último, quedan funciones que trabajan con el tiempo. La principal de ellas, y su bas es la **función Now**. La **función Now** sirve para **obtener la fecha y hora actuales** y guardarla en una variable (que lo recibe como un valor alfanumérico o de cadena). Se expresa de la siguiente forma:

```
Variable = Now
```

Como vemos, esta función carece de argumentos, y por lo tanto se trata de una palabra que no puede ser usada para definir una variable. Esto se puede combinar con un **MsgBox** de la manera siguiente, para poder ver que valor nos da la función:

```
Variable = MsgBox(Now,0,"Fecha y hora")
```

Este ejemplo daría como resultado el valor de cadena mostrado en la siguiente imagen:



En este caso anterior, como Now guarda la fecha y hora en **formato alfanumérico** se podría operar con este valor con los **operadores de cadena**. Aparte también, la función Now tiene sus propios **operadores de tiempo** que lo pueden modificar para que guarde solo la hora, el minuto, el año, u otros datos concretos y no solo la parrafada anterior... En estos casos la función Now guardaría el valor como un **valor numérico y de cadena a la vez,** por lo que se podría modificar **primero** con **operadores de numeros** y **luego** con los de cadena. Un ejemplo de esto es el siguiente**:**

```
Variable = Day(Now)
Variable2 = Sqr(Variable)+1
Variable3 = StrReverse(Variable2&Variable)
Variable4 = MsgBox(Variable3,0,"Numero")
```

La tarea de este código es la siguiente (se que no he explicado las funciones aún, pero no tardaré, es para que veáis que se le puede tratar de diversas formas): Primero, averigua el día de hoy (un numero) y lo guarda en Variable. Luego, coge el valor de Variable y le aplica la raíz cuadrada sumándole una unidad al resultado y lo guarda en Variable2. Mas tarde coge el valor de Variable2 y Variable y los une, para luego invertir el orden de sus cifras y guardarlo en Variable3. (Los une de forma que si por ejemplo Variable2 es 3,44 y Variable es 10, el número final obtenido es 3,4410) Por ultimo abre un MsgBox y muestra el valor de Variable3, para ver el resultado.

También se puede combinar la función Now con la **sentencia If** para hacer que un trozo del script solo aparezca un día determinado, o un mes concreto... A continuación un ejemplo de uso de la función Now modificada por el **operador de Día** con la sentencia If para que el MsgBox solo aparezca el día 5 de cada mes (repito que el operador de día lo explicaré ya enseguida):

Variable = Day(Now)
If Variable = 5 Then
Variable2 = MsgBox("Hoy es dia 5",0,"Fecha")
End If
	Averigua el día y lo guarda en Variable.
	Comprueba si Variable es 5.
	Si Variable es 5 muestra un mensaje
	Fin de la condición.

De esta forma podemos jugar con la fecha y crear eventos temporales (bucles que se repitan x segundos, mensajes que aparezcan días determinados, etc…). Pero no voy a entretenerme más, lo prometido es deuda. La **función Now** tiene a su vez varias funciones que la pueden modificar:

La primera función es ***Day,*** que la hemos utilizado en los ejemplos anteriores. Esta función hace que la función Now de solamente un valor numérico que representa el día del mes. Es decir, si estamos a 23 de Marzo, esta función dará el valor 23. Un ejemplo de uso de esta función es el siguiente:

Variable = **Day(**Now**)**	En este caso Variable obtiene el valor correspondiente al día.

La segunda función es ***WeekDay***. Esta función hace que la función Now de solamente un valor numérico que representa el día de la semana. Es decir, si estamos a lunes 23 de Marzo, esta función dará el valor 1 mientras que si estamos en domingo, el valor que devolverá será 7:

Variable = **WeekDay(**Now**,2)**	El **numero 2** es una constante que indica que el primer día de la semana es el lunes, no se debe omitir y no es necesario modificarlo (si pones 1 el primer día es domingo, etc).

La tercera función es ***Month***. Esta función hace que la función Now devuelva un número entero entre 1 y 12, inclusive, que representa el mes del año. Es decir, si estamos a 23 de Marzo, esta función dará el valor 3 mientras que si estamos en Junio, el valor que devolverá será 6:

Variable = **Month(**Now**)**	En este caso Variable obtiene el valor correspondiente al mes.

La cuarta función es ***Year***. Esta función hace que la función Now devuelva un número entero de cuatro cifras, que representa el año. Un ejemplo de uso de esta función es el siguiente:

Variable = **Year(**Now**)**	En este caso Variable obtiene el valor correspondiente al año.

La quinta función es ***Hour***. Esta función hace que la función Now devuelva un número entero entre 0 y 23, inclusive, que representa la hora del día. Un ejemplo de uso de esta función es el siguiente:

Variable = **Hour(**Now**)**	En este caso Variable obtiene el valor correspondiente a la hora.

La sexta función es ***Minute***. Esta función hace que la función Now devuelva un número entero entre 0 y 59, inclusive, que representa el minuto de una hora. Un ejemplo de uso de esta función es el siguiente:

Variable = **Minute(**Now**)**	En este caso Variable obtiene el valor correspondiente al minuto.

La séptima función es ***Second***. Esta función hace que la función Now devuelva un número entero entre 0 y 59, inclusive, que representa el segundo de un minuto. Un ejemplo de uso de esta función es el siguiente:

Variable = **Second(**Now**)**	En este caso Variable obtiene el valor correspondiente al segundo.

La octava función es ***WeekDayName***. Esta función hace que la función Now devuelva una **cadena en minúsculas** con el nombre de la semana. Esta función va acompañada siempre de la función ***WeekDay.*** La palabra False indica que guarda el nombre completo, si pones True guardará solo las tres primeras letras. El número 2 tiene la misma función que el 2 de la función ***WeekDay.*** Un ejemplo de uso de esta función es el siguiente:

Variable = **WeekDayName(WeekDay(**Now**,2), False, 2)**	En este caso Variable obtiene el valor correspondiente al nombre del dia de la semana.

La novena función es ***MonthName***. Esta función hace que la función Now devuelva una **cadena en minúsculas** con el nombre del mes. Esta función va acompañada siempre de

la función ***Month.*** La palabra False indica que guarda el nombre completo, si pones True guardará solo las tres primeras letras. Un ejemplo de uso de esta función es el siguiente:

Variable = **MonthName(Month(**Now**), False)**	En este caso Variable obtiene el valor correspondiente al nombre del mes.

Las últimas funciones son ***Time*** y ***Date***. Estas funciones actúan de forma independiente y devuelven, la primera la hora, el minuto y el segundo; y la segunda, el mes, año y día. Estas pueden sustituir al Now pero es recomendable usar antes el Now que estas. Estas solo se utilizan para mostrar mensajes o con todos los datos de la hora o con la fecha entera. Un ejemplo de uso de estas funciones es el siguiente:

Variable = **Date** Variable2 = **Time** Variable3 = Variable&" "&Variable2	En este caso Variable obtiene el valor de la fecha. Variable2 obtiene el valor de la hora. Variable3 obtiene los dos valores juntos que equivalen a la función Now.

Y bueno, esto es todo en cuanto a funciones de tiempo se refiere, aunque creo que son suficientes para darnos un control sobre el tiempo bastante grande. Recomiendo de nuevo practicar y repasar todo lo visto hasta ahora, ya que **MsgBox** puede mostrar los resultados en pantalla. Por otro lado, he obviado la creación de variables en todos los ejemplos, pero no vendría mal que volvieras a ver el uso del **Option Explicit** y el **Dim**.

Y por último, de este Tutorial de introducción decir que ya se pueden hacer muchas cosas, que practiquéis y que lo aprendáis todo bien antes de continuar con la segunda parte: **El Uso de los Objetos en VBscript**, pues daré todo esto por sabido y aprendido.

Y no lo olvides, para dudas o consultas escribe a: scswinter@hotmail.com

O bien visita: http://scswinter.110mb.com/