TEMA 2: SCRIPTS.

Son pequeños programas escritos en u lenguaje de script dentro del código de una página Web para conseguir efectos especiales o interactuar con el usuario.

LENGUAJES DE SCRIPTS.

Funcionan solo dentro de la aplicación para la que han sido creados. Disponen de una serie de variables, métodos y objetos predefinidos que les permiten interactuar con dichos usuarios.

Son lenguajes interpretados no compilados.

Su objetivo es conseguir que la programación sea sencilla, aunque a veces esto implica una disminución del rendimiento y de la legibilidad del código. En Internet se utilizan Java Script y Visual Basic Script, Microsoft ha desarrollado una versión de java llamada J-Script que es distinta al Java Script desarrollado por Netscape.

Dependiendo de la aplicación Visual Basic Script dispone de una jerarquía de objetos deferente y adaptada a cada aplicación para poder interactuar con ella, lo mismo ocurre con Java Script, el consorcio W3C ha publicado un documento llamado DOM en el cual se establece un estándar sobre la jerarquía de objetos predefinidos.

MANERAS DE EJECUTAR UN SCRIPT.

Se pueden ejecutar de dos maneras:

1.- Al cargar la página Web.

2.- En respuesta a sucesos (eventos) producidos por el usuario.

Casi cualquier caso que pueda realizar el usuario en la página Web tiene un suceso o evento relacionado.

Script que se ejecuta al cargar la página: se utilizará la siguiente sintaxis:

a) Caso de Visual Basic Script, escribiremos la etiqueta <Script language=”vbscript”>.....</Script>, y entre ellos hay que poner obligatoriamente las etiquetas que indican comentario<!--.....--> y entre ambos el código de Visual Basic Script.

```
<Script language=”vbscript”>
        <!—
                .........
                ....
                Código
                ....
                ........
        -->
</Script>
```

b) caso de Java Script:

```
<Script language=”javascript o jscrpt”>
        <!—
                ......
                Código
                ......
        -->
</Script>
```

El código Visual Basic Script puede escribirse en principio de dos maneras:

1.- Directamente línea a línea, en este caso se ejecuta una sola vez al cargarse la página Web.

2.- Dentro de un procedimiento, es este caso el código que está dentro del procedimiento solo se ejecuta si se llama a dicho procedimiento en otra parte del programa, y se ejecuta tantas veces como se llame.

-document.write: es una instrucción que escribe en la página Web el texto que recibe como parámetro o el valor de las variables que se escriban a continuación.
-document.lastmodified: es una variable predefinida que almacena la fecha y la hora de la última actualización de la página Web.
-msgbox (navigator.oppversion), presenta en una ventana la versión de HTML, del navegador y del sistema operativo.
-msgbox, presenta el valor de una variable (x) en una ventana, msgbox (x).

2.- Script controlado por suceso o eventos del usuario. El siguiente Script está controlado por el suceso (evento) OnMouseOver (poner el ratón encima).

For=”Id de la etiqueta a la que se aplica el script”
Event=” Suceso que activa el Script”
Language=”lenguaje de Script utilizados”

```
<Script For=”Img3” event=”onmouseover” language=”Vbscript”
<!—
        Código (otra imagen que suplante a la imagen de id=”img3”)
        Img3.src=”nueva imagen”
-->
</Script>
```

Nota: los scripts se pueden escribir en cualquier parte del código html, suele utilizarse la cabecera para escribirlos con el fin de tenerlos agrupados.

Otras posibles sintaxis son:

1.-<Input type=2button” id=”b1” value=”texto” onclick=”img2=”nuevaimagen” ” language=”vbscript”>

2.-<Img id=”img4” src=”imagen1” onmouseover=”imagen4.src=”imagen2”” language”vbscript”>

VISUAL BASIC SCRIPT VBS 5.6

CONCEPTOS BÁSICOS.

-Los comentarios se pueden obtener bien con la instrucción REM o bien con el apostrofe (‘).

-VBS script es un lenguaje interpretado, es decir, el interprete lee una instrucción, la traduce a código máquina y la ejecuta, a continuación lee la siguiente instrucción, la traduce y la ejecuta y así hasta que termina con la última instrucción.

- No se distingue entre mayúsculas y minúsculas.

- Hay instrucciones que pueden escribirse en una sola línea o instrucciones que requieren varias líneas.

-Se pueden escribir varias instrucciones en una sola línea separadas por el carácter :, ejemplo: instrucción1: instrucción2: instrucción3.

-las instrucciones de una sola línea que sean muy largas se pueden dividir en varias líneas escribiendo el final de cada línea un espacio y un carácter de subrayado, ejemplo: ESTO ES UNA INS _
TRUCCIÓN.

TIPOS DE DATOS DE VISUAL BASIC SCRIPT.

En vbs sólo hay un tipo de datos que se llama VARIAL, éste tipo de datos puede contener diferentes tipos de información dependiendo de cómo se utiliza (número, cadenas de texto, fechas, objetos, etc.) el interprete analiza el contexto en que se encuentran los datos y según sea esté así maneja el dato, por ejemplo12 se maneja como un número en un contexto numérico y como una cadena en un contexto de cadenas de caracteres.

A cada tipo de información que puede almacenarse en un variant se le llama subtipo.

Subtipos: empty, null, bolean, byte, intgeger, currency, long, sigle, double, date, string, objet, error. (ver fotocopia).

Existen funciones de conversión para convertir un subtipo en otro subtipo.

VARIABLES.

Una variable es un nombre que hace referencia a una zona de la memoria, en la cual, se puede almacenar información que puede cambiar durante el tiempo en que se ejecuta el programa.

Además del nombre, toda variable tiene un tipo y un valor.

En VBS las variables siempre son de tipo variant, el valor de la variable, es la información almacenada a la que hace referencia.

DECLARACIÓN DE LAS VARIABLES.

Toda variable hay que declararla, asignarle un valor y operar con ella, en VBS no es obligatorio declarar las variables, es decir, si en el código escribimos cualquier nombre dentro de una expresión el interprete declara automáticamente una nueva variable, no obstante las variables se pueden declarar explícitamente mediante la instrucción DIM, por ejemplo DIM a, aa, b, a, finl, z. también se pueden declarar con PUBLIC o PRIVATE.

La declaración de variables se puede hacer en cualquier lugar del código, pero se deben especificar todas juntas en una zona determinada.

Resulta conveniente poner como primera instrucción OPTION EXPLICIT para obligatoriamente tener que declarar todas las variables.

NOMBRES DE LAS VARIABLES.

Han de cumplir las siguientes reglas:

1.- Debe de comenzar por una letra.

2.- No puede contener el carácter punto.

3.- Han de ser menor a 255 caracteres.

4.- Debe ser único dentro del alcance de la variable.

5.- No puede coincidir con una palabra clave.

ALCANCE Y VIDA DE LA VARIABLES.

En VBS se pueden considerar dos tipos de variables:

-Variables declaradas dentro de un procedimiento o variables locales, el alcance de estas variables se limita al procedimiento en el que están declarados, es decir, sólo pueden ser accedidos por el código de ese procedimiento y modificar su valor.

- Aquellas que están definidas directamente en el código fuera del procedimiento, estas variables pueden ser accedidas por todos los procedimientos y por todo el código, por lo tanto su alcance es todo el programa.

La vida de las variables es el tiempo que la información que almacena permanece en la memoria. Las variables definidas en un procedimiento permanecen en la memoria desde que se declaran hasta que se termina de ejecutar el procedimiento, cuando este termina se borran de la memoria las variables declaradas en él.

Las variables globales tienen una vida desde que se declaran hasta que termina el programa.

Nota: dos variables generales no pueden tener el mismo nombre y análogamente dentro de un procedimiento no puede haber dos variables con el mismo nombre. Sin embargo una variable de un procedimiento puede tener el mismo nombre que otra variable en otro procedimiento.

ASIGNACIÓN DE VALORES A LAS VARIABLES.

Para asignar valores a una variable se crea una expresión, a la izquierda estará el nombre de la variable, el signo igual y a la derecha el valor asignado, a=25; b=”casa”; c=#04/03/2004#.

Una variable que tiene un único valor se dice que es una variable escalar.

Para poder averiguar el subtipo de una variable se utiliza la función VARTYPE (nombre variable), esta función devuelve un número según cual sea el subtipo de la variable, los diferentes números pueden ser:

0 → Empty.
1 → Null.
2 → Integer.
3 → Entero Largo.
4 → Single.
5 → Double.
6 → Currency.
7 → Date.
8 → String.
9 → Objeto automático.

10 → Error.
11 → Boolean.
12 → Variant.
13 → Objeto de acceso a datos.
17 → Byte.
8204→ Array.

ARRAYS O MATRICES.

Son variables que pueden almacenar varios valores (variables vectoriales) en un solo nombre, para hacer referencias a esos valores utilizaremos un conjunto de índices.

Un conjunto fundamental en las matrices es la dimensión, una matriz es de dimensión 1 si solo se utiliza un índice para referenciar los diversos valores, es de dimensión 2 si se utilizan dos índices y así sucesivamente, en VBS se pueden definir matrices de hasta 60 dimensiones.

Para declarar una matriz se utiliza la instrucción DIM, a continuación el nombre de la variable que representa la matriz y luego entre paréntesis un número, si es de una dimensión y varios números separados por comas para cada dimensión, tantos números como dimensiones, por ejemplo: DIM asses (20).

El número de elementos que hay en cada dimensión es igual al número que se escribe en la declaración más uno por que en VBS las matrices son de base 0.

A parte de DIM podemos utilizar public o private, ejemplo: PUBLIC cc1 (32: PRIVATE cc2 (12,45).

Las matrices anteriores son de tamaño fijo, por que el interprete al declararlas les asigna una zona de memoria con un tamaño que no se cambia durante la vida de la matriz.

También existen matrices dinámicas o de tamaño variable, son aquellas en las cuales el tamaño que ocupan en la memoria puede cambiar durante su vida, es decir, durante su vida el intérprete puede borrar la zona de la memoria y asignarle otra, o lo que es lo mismo el programador puede redimensionarlas a lo largo del programa.

Para declarar un matriz dinámica podemos utilizar DIM, PROVATE o PUBLIC con paréntesis vacíos, ejemplo DIM cosas ( ).

Para dimensionar o definir la matriz dinámica se utiliza la instrucción REDIM, por ejemplo REDIM cosas (23,45).

Una matriz dinámica se puede declarar y dimensionar al mismo tiempo utilizando directamente REDIM, REDIM cosas3 (12,10,4)

Si se redimensiona una matriz se borra el contenido de la matriz anterior, liberándose su espacio de memoria y se reserva un nuevo espacio de memoria ara la matriz nueva. Si queremos redimensionar una matriz sin borrar la información utilizaremos: REDIM preserve cosas 3 (12,10,30).

Nota: sólo se podrá modificar la última dimensión, y además no se puede cambiar el número de dimensiones.

CONSTANTES.

Una constante es un nombre significativo que contiene un número o una cadena que nunca cambia durante la ejecución del programa, VBS posee una serie de constantes predefinidas, por ejemplo VBCRLF que combina los caracteres retorno de carro y salto de línea.

A parte de las constantes predefinidas puede crear otras constantes, para ello se utiliza la instrucción CONST y a continuación se escribe el nombre de la variable y se iguala a un determinado valor. Ejemplo: CONST pepe=23, paco=”bght”.

Para escribir valores constantes los números se escriben directamente, las cadenas de caracteres entre comillas y las fechas entre almohadillas.

OPERADORES EN VBS.

Permiten combinar variables para formar expresiones (ver fotocopia)

ESTRUCTURAS PARA CONTROLAR EL FLUJO DE UN PROGRAMA.

Hay dos tipos de estructura de control del flujo del programa fundamentales :
1.- Estructuras condicionales o selectivas.
2.- Estructuras repetitivas o bucles.

1.- Estructuras selectivas o condicionales:

a) Condicional de una sola línea, se hace: IF condición THEN instrucción [ELSE instrucción2], si se cumple la condición se ejecuta la instrucción primera y si a condición es falsa si existe cláusula ELSE se ejecutan las instrucciones 2 y si no se siguen con la línea siguiente del programa. Ejemplo_

```
a= inputbox ("introduzca un número")
IF a<>50 THEN msgbox"su número no tiene premio"ELSE msgbox "premio"
IF a=50 Then msgbox " su número tiene premio"
```

b) Condicional o selectiva de bloque, la estructura general es la siguiente:

ESTRUCTURA	EJEMPLO

IF condición THEN
Instrucciones1
[ELSE
Instrucciones 2]
END IF
	IF a=50 THEN
	Msgbox "premio"
	ELSE
	Msgbox "numero no premiado"
	END IF

c) Selectiva de condiciones múltiples, la sintaxis es la siguiente:

```
IF condición 1 THEN
        Instrucciones 1
[ELSEIF condición 2 THEN
        Instrucciones2
[Elseif condición 3 THEN
        Instrucciones 3
.....
ELSE
        Instrucciones]
ENDIF
```

Si la condición uno es verdadera se ejecutan las instrucciones uno, y una vez terminadas se salta a la línea siguiente a ENDIF, si la condición uno es falsa se examina la condición dos, si es verdadera se ejecutan las instrucciones 2 y se salta a la línea anterior a ENDIF, si ninguna de las condiciones es verdadera y existe ELSE se ejecutan las instrucciones de ELSE. Ejemplo:

```
IF a=50 THEN
        'no hace nada
ELSEIF a >50 THEN
        Msgbox "ha escrito un número mayor"
ELSEIF a >50 THEN
        Msgbox" ha escrito un número menor"
END IF
```

d) Select case, la sintaxis es la siguiente:

```
SELECT CASE expresión de prueba
        CASE lista de expresiones
                Instrucciones1
        CASE lista de expresiones
                Instrucciones n
        CASE ELSE
                Otras instrucciones
END SELECT
```

En Internet Explorer sólo funciona con listas de valores, no con expresiones. A continuación veamos un ejemplo con SELECT CASE:

```
b= inputbox ("introduzca un número menor de 100")
Select case b
        Case 50
           Msgbox "premio"
        Case 47,34,25
           Msgbox " su número es menor"
        Case 70,90
           Msgbox " su número es mayor"
        Case Else
           Msgbox " el número introducido no es válido"
End select
```

Nota: pueden ser números, cadenas de caracteres y fechas separadas por comas.

Anidamiento de estructuras selectivas:

Se puede anidar una estructura selectiva dentro de otras, ejemplo:

```
IF condición 1 THEN
        Instrucciones 1
        IF condición 2 THEN
        Instrucciones2
        Else
        Instrucciones 2b
        End if
Else
        Instrucciones 1b
```

End if

2.- Estructuras repetitivas o bucles:
a) bucles DO-WHILE, la sintaxis es:

```
DO while condición1
        Instrucciones1
[IF condición2 THEN EXIT DO
        Instrucciones 2]
LOOP
```

Haz mientras la condición uno sea verdadera las instrucciones 1, si la condición 2 es falsa haz las instrucciones 2 y después vuelve a empezar. El bucle termina cuando la instrucción uno sea falsa y/o la condición dos sea verdadera.

```
DO
        Instrucciones 1
[IF condición 2 THEN EXIT DO
        Instrucciones 2]
LOOP while condición1
```

Haz las instrucciones 1, si la condición 2 es falsa haz las instrucciones 2, repite mientras la condición uno sea verdadera.

Ejemplo 1:	Solución 1:
Dim c,b C=0 B=100 DO while c<100 C=c+1 B=b-1 Loop Msgbox “c=” &c &vbcrlf &”b=” &b	C=100 B=0
Ejemplo 2:	Solución 2:
Dim c,b C=0 B=100 DO while c<100 C=c+1 IF b=50 then exit DO LOOP Msgbox “c=” &c &vbcrlf &”b=” &b	C=100 B=100
Ejemplo 3:	Solución 3:
Dim c,b C=0: b=100 DO while c<100	C=100 B=0

C=c+1 B=b-1 LOOP Msgbox “c=” &c &vbcrlf &”b=” &	

b) bucles DO-LOOP, la sintaxis es:

```
DO until condición 1
        Instrucciones 1
[IF condición 2 THEN EXIT DO
        Instrucciones 2]
LOOP
```

Haz instrucciones uno hasta que sea verdadera la condición 1, si la condición 2 es falsa haz las instrucciones 2 y después vuelve a empezar.

```
DO
        Instrucciones 1
[IF condición 2  THEN EXIT DO
        Instrucciones2]
LOOP until condición 1
```

Haz las instrucciones uno, si la condición dos es falsa haz las instrucciones 2, repite hasta que la condición uno sea verdadera.

```
DO
        Instrucciones 1
IF condición 2  THEN EXIT DO
        Instrucciones2
LOOP
```

Haz las instrucciones uno, si la condición dos es falsa haz las instrucciones dos.

Observemos algunos ejemplos:

Ejemplo 1:	Solución 1:
Dim c,b C=0 :b=100 DO until c< 100 C=c+1 B=b-1 LOOP Msgbox “c=” &c &vbcrlf &”b=” &	C=0 B=100
Ejemplo 2:	Solución 2:
Dim a A=100	

DO A=a+1 LOOP While A<100 Msgbox “a=” &a	A=101
<u>Ejemplo 3:</u> Dim c,b C=0: b=100 DO C=c+2 IF b=50 then exit do B=b+1 Loop until c>100	<u>Solución 3:</u> B= 50 C= 102

<u>Ejemplo 4:</u>
 	
Dim b,c
B= 100: c=0
DO
C=c+2
If c >= 50 Then exit do
B=b+1
C=c-2
If b <= 540 Then exit do
C=c+1
LOOP
Msgbox “c=” &c &vbcrlf &”b=” &
	<u>Solución 4:</u>
	
	C=50
	B=52

c) Bucle infinito:

```
DO
  Instrucciones1
LOOP
```

d) Bucle WHILE-WEND, su sintaxis es:

```
WHILE condición
        Instrucción
WEND
```

Mientas sea verdad la condición haz las instrucciones y repite mientras la condición sea verdadera, ejemplo:

Ejemplo 1:	Solución 1:
Dim d D=0 While d<100 D=d+1 Wend Msgbox “d=” &d	D=100

e) Bucle FOR, toda la sintaxis de DO-LOOP y de WHILE-WEND se utiliza cuando no sabemos el número de veces que tiene que repetirse en bucle, si previamente sabemos el número de veces que se va a repetir el bucle utilizaremos la estructura FOR-NEXT, la sintaxis es la siguiente:

```
FOR contador=Inicio TO fin [Step paso]
        Instrucciones 1
        IF condición THEN exit FOR
        Instrucciones 2
NEXT
```

Contador es una variable que comienza en inicio y se va incrementando según indique el número del STEP, puede ser entero o decimal, positivo o negativo, si no se especifica ningún STEP el incremento es positivo y de uno en uno.

-Para contador igual a inicio hasta fin con un intervalo de incremento de contador igual a paso haz las instrucciones uno, si condición es verdadera sal del bucle, si es falsa haz las instrucciones dos y repite.

-Para un paso positivo el bucle solamente se ejecuta si inicio es menor que fin, en caso contrario no se ejecuta. Para paso negativo el borde solamente se ejecuta si el valor inicio es mayor que el valor fin

Ejemplo:

Ejemplo 1:	Solución 1:
Dim ct, m(20) FOR ct=0 TO 20 STEP 1 M(ct)=ct Msgbox m(ct) NEXT FOR ct=0 TO 20 STEP 2 M(ct)=ct^2 Msgbox m(ct) NEXT FOR ct=0 TO 20 Msgbox m(ct) NEXT	M(ct)=ct → 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 M(ct) ct^2→ 0 4 16 36 64 100 144 196 256 324 400. M(ct)→ 0 1 4 3 16 5 36 7 64 9 100 11 144 13 196 15 256 17 324 19 400

f) Bucle FOR –EACH, la sintaxis es:

```
FOR EACH elemento IN grupo
        Instrucciones 1
```

```
IF condición THEN exit FOR
        Instrucciones 2
NEXT
```

Este bucle se aplica a matrices o a colecciones de elementos.

Elemento: Es una variable que representa el contenido de cada elemento de una matriz, representa un objeto en las colecciones de elementos.

Para cada elemento en la matriz hay las instrucciones 1, si la condición es verdadera haz las instrucciones 2, si es falsa sale del bucle, repite hasta que se terminen los elementos de la matriz o colección de objetos, ejemplo:

Ejemplo 1:	Solución:
DIM ct, ele, cad1, cad2, m() REDIM m(20) Cad1="":cad2="" FOR ct=0 TO 20 STEP 1 M(ct)=ct^2 Cad1=cad1&","&m(ct) NEXT FOR EACH ele IN m IF ele>250 THEN exit FOR Cad2=cad2&","&ele NEXT Msgbox cad1 &vbcrlf&cad2	Cad1=cad1&","&m(ct) → 0,1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400 Cad2=cad2&","&ele → 0,1,4,9,16,25,36,49,64,81,100,121,144,169,196,225

Nota: Todos los bucles se pueden anidad entre sí y con las estructuras selectivas.

PROCEDIMIENTOS.

Sirven para realizar una tarea concreta que se necesita utilizar varias veces. El código necesario se escribe en un bloque llamado PROCEDIMIENTO al que se le da un nombre.

Aunque el bloque de código que forma un procedimiento esté insertado entre el código de n programa, no se ejecuta automáticamente al ejecutar el programa. Si se desea ejecutar las tareas realizadas por un procedimiento, hay que llamarlo escribiendo su nombre adecuadamente dentro del código del programa. Al terminar de ejecutarse el procedimiento, el programa pasa a ejecutar la instrucción siguiente a la llamada.

En general, para que un procedimiento pueda realizar las acciones que seamos, tendrá que intercambiar información con el programa que lo llama. Sobre éstas, actuará el código del procedimiento para realizar sus acciones y en ciertos casos devolver algún valor o resultado.

Al definir un procedimiento se establecen una serie de PARAMETROS FORMALES que determinan el conjunto de variables que el procedimiento puede intercambiar con el programa que lo llama.

Pueden distinguirse tres tipos de parámetros formales:

-Parámetros de entrada: el procedimiento utiliza el parámetro pero no cambia su valor.

-Parámetros de entrada salida: el procedimiento utiliza el parámetro y cambia su valor.

-parámetros de salida: el procedimiento le da un valor al parámetro pero no utiliza su valor original.

La instrucción que llame al procedimiento contendrá una lista de ARGUMENTOS o parámetros actuales que se corresponderán uno a uno con los parámetros formales establecidos en la definición del procedimiento. Es decir, han de tener el mismo número, el mismo orden y estructura (variables escalares variant o matrices variant).

Sin embargo, el nombre de un argumento puede ser distinto del nombre del parámetro formal correspondiente. Dentro del código del procedimiento, se utilizará el nombre del parámetro formal y dentro del programa que hace la llamada se utilizara el nombre del argumento.

Los argumentos y parámetros formales pueden intercambiar su valor de dos maneras:

-Transmisión por valor: se crea una copia temporal de la variable original, esta copia es la que maneja el procedimiento, por lo que, si el procedimiento cambia su valor, no se modifica el valor de la variable original.

-Transmisión por referencia: el procedimiento se le da la dirección de memoria en que se encuentra la variable que se le pasa como argumento, por lo que, si cambia su valor se cambia el valor de la variable original.

En VBScript 5.6 existen dos tipos de procedimientos generales: los SUB y los FUNCTION además pueden definir clases con los bloques CLASS y en ellos los procedimientos PROPERTY: PROPERTY SET, PROPERTY GET y PROPERTY LET para la definición y manejo de las propiedades y métodos de las clases.

PROCEDIMIENTOS SUB.

[PUBIC|PRIVATE] SUB nombre [(lista parámetros formales separados por comas)]

```
        Instrucciones 1
        IF condición THEN exit SUB
        Instrucciones 2
END SUB
```

-Public: indica que el procedimiento SUB es accesible por todos los procedimientos de todas las secuencias de comandos.

-Private: indica que el procedimiento SUB sólo es accesible por los procedimientos de la secuencia de comandos en que ha declarado el procedimiento. Por defecto los procedimientos SUB son privados.

-Lista de parámetros formales: variables escalares o matrices que representan los argumentos que se pasan al procedimiento cuando se le llama, tiene la siguiente sintaxis:

([BYVAL|BYREF] param 1[()], [BYVAL|BYREF] param 2[()],....,)

En donde:

BYVAL: el argumento se pasa por valor, se crea una copia temporal de la variable original. Esta copia se le pasa al procedimiento, por lo que si el procedimiento cambia su valor no se modifica el valor de la variable original.

BYREF: el argumento se pasa por referencia, es decir, al procedimiento se le da la dirección de memoria en que se encuentra la variable que se le pasa como argumento, por lo que si cambia su valor, se cambia el valor de la variable original. Por defecto los valores se pasan por referencia.

-No se pueden definir procedimientos SUB dentro de otro procedimiento.

-El valor de las variables locales (declaradas dentro del procedimiento), no se conserva entre llamadas de procedimiento.

-Un procedimiento SYB no devuelve ningún valor, y no se le puede utilizar formando parte de una expresión.

-para llamar a un procedimiento se escribe su nombre seguido de la lista de argumentos separados por comas:

Nombre Procedimiento arg1, arg2, arg3....

Se puede utilizar la instrucción CALL pero entonces los argumentos hay que escribirlos entre paréntesis:

CALL Nombre Procedimiento 8arg1, arg2, arg3,...)

-Exit SUB implica la salida inmediata del procedimiento, se continúa en la siguiente instrucción a llamada del procedimiento.

-Los procesos SUB son recursivos, es decir, pueden llamarse a si mismos. Esto puede ser peligroso pues, puede desbordarse el tamaño de la pila.

A continuación podemos observar algunos ejemplos de procedimientos SUB:

Procedimiento	***Llamada al Procedimiento***	***Solución***
SUB suma (soy) DIM s S=x+y Msgbox “la suma_ de”&x&”y”&y&”es:”&S X=5 Y=2 END SUB	A=20 :b=30 Sumaa a,b Msgbox” a valía 20_ después de la llamada_ vale:”&a&vbcrlf&”b valía 30 después de la llamada_ vale:”&B	-La suma de 20 y 30 es 50 -A valía 20 después de la llamada vale5, b valía 30 después de la llamada vale:2
SUB suma b (BYVAL x,_ BYVAL y) DIM s S=x+y Msgbox “la suma_ de”&x&”y”&y&”es:”&S X=5 Y=2 END SUB	A=20 :b=30 Sumaa a,b Msgbox” a valía 20_ después de la llamada_ vale:”&a&vbcrlf&”b valía 30 después de la llamada_ vale:”&B	-La suma de 20 y 30 es 50 -A valía 20 después de la llamada vale20, b valía 30 después de la llamada vale:30
SUB sumac (x,y,s) S=x+y END SUB	A=20: b=30: c=0 Sumac a,b,c Msgbox “a vale”&a&vbcrlf&”b vale”&b&vbcrlf&”c vale”&C	A = 20 B=30 C=50

PROCEDIMIENTOS FUNCTION.

[Public|Private] FUNCTION nombre [(lista de parámetros formales separados por comas)]

```
        Instrucciones 1
        [nombre=expresión]
        If condición THEN exit FUNCTION
        Instrucciones 2
    END FUNCTION
```

Tiene las mismas normas y comportamiento que SUB salvo que:

1.- Los procedimientos FUNCTION devuelven un valor en una variable cuyo nombre es igual al nombre del procedimiento.

2.- El subtipo devuelto por una FUNCTION puede ser cualquiera de las estudiadas.

3.- Se puede utilizar el nombre de un procedimiento FUNCTION en la parte de la derecha de una expresión:

Variable=nombre (parámetros) OPERADOR expresión

-Observar que para manejar el valor devuelto por una FUNCTION se escriben los argumentos entre paréntesis.

4.- SI dentro del cuerpo de la función no se asigna ningún valor al nombre:

-Una función numérica devuelve 0.

-Una función de cadena, devuelve "" (cadena vacía).

-Una función que devuelve una referencia de objeto devuelve NOTHING.

5.- En una expresión aritmética no se utiliza una FUNCTION si la función cambia el valor de alguna de las variables existentes en la función.

En VBScript existe la función LOG que nos da el logaritmo neperiano de un número (la base es e=2,71872) pero no existe una función que nos del logaritmo decimal (base 10) podemos definirla de la siguiente manera:

```
FUNCTION log10 (x)
    Log10 = log (x) / log (10)
END FUNCTION
```

A continuación podemos ver unos ejemplos de procedimientos con FUNCTION:

Procedimiento	***Llamada al procedimiento***	***Solución***
FUNCTION suma (soy) Suma=x+y END FUNCTION	a=20 :b=30 c=suma (a,b) d=suma (a,c)^2 msgbox "a="&a&vbcrlf&"b="&b&vbcrlf&"c="&c& vbcrlf&"(a+c)^2="&D	A=20 B=30 C=50 (a+c)^2= 4900
FUNCTION log10 (x) Log10 = log (x) / log (10) END FUNCTION	A=1: b=100: c=1789 Msgbox log10(a)&vbcrlf&log10(b)&vbcrlf&log10(c)	Log10(a)=0 Log10(b)=2 Log10(c)=3,2526

FUNCTION factorial (n) Dim ini,ct Ini=1 FOR ct=1 to n Ini=ini*ct Next Factorial=ini END FUNCTION	Msgbox "el factorial de 6 es igual a 6*5*4*3*2*1="&factorial(6)	6!=720

FUNCTION factor (n)
IF n=1 THEN
Factor=1
ELSE
Factor=n*factor(n-1)
END IF
FUNCTION
	Msgbox "el factorial de 6 es igual a 6*5*4*3*2*1="&factorial(6)
	
	(Definición de una función factorial utilizando la recursividad)
		6!=720

Nota: Si bien puede utilizarse una FUNCTION sin devolver ningún valor por consistencia con el lenguaje no se hará y se sustituirá por un procedimiento SUB.

Nota: Si bien la instrucción CALL no es necesaria, muchos programadores la usan para resaltar que se está llamando a un procedimiento SUB. Si se utiliza para llamar a un procedimiento FUNCTION se descarta el valor devuelto por la función.

PROCEDIMIENTOS CONTROLADOS POR SUCESOS (EVENTOS).

Estos procedimientos se ejecutan cuando se produce un evento en un control, la sintaxis es:

[Public|Private] SUB control_evento ([lista parámetros formales separados por comas)]
Instrucciones 1
IF condición THEN exit SUB
Instrucciones 2
END SUB

-Control: es el valor asignado al atributo ID o al atributo NAME del control.
-Evento: es el nombre del evento que provoca la ejecución del procedimiento.

Los eventos del ratón son:
- Onclick: se hace clic con el ratón sobre el control.
- Ondblclick: se hace doble clic con el ratón sobre el control dentro del intervalo de tiempo establecido en el sistema.
- Onmousedown: se produce cuando se pulsa el botón del ratón estando encima del control. Se produce antes del evento onclick.

- Onmopuse up: estando el ratón pulsado sobre el control el usuario deja de pulsarlo. Se produce después del evento onclick.

- Onmouseover: se produce cuando el puntero del ratón entra en la zona del control.

- Onmousemove: se produce siempre que se mueve el ratón sobre la zona del control.

- Onmouseout: se produce cuando el ratón sale de la zona del control.

- Onmouseenter: se produce cuando entra el ratón en la zona de control.

- Onmousewheel: se produce cuando se pulsa el botón central del ratón sobre el control.

Ejemplo:

```
SUB bt1_onclick()
        Msgbox "pepe"
ENDSUB

SUB bt2_ondblclick()
        Msgbox "hola que tal"
END SUB.
```

FUNCIONES MATEMÁTICAS.

· **Función ATN**: devuelve el arco tangente de un número (función inversa de la tangente). ATN= 1/tangente, el ángulo lo devuelve en radianes.

-Sintaxis: ATN (número)

-El argumento número puede ser cualquier expresión numérica válida.

-Para convertir grados en radianes, se multiplican los radianes por pi/180.

-Para convertir radianes en grados, se multiplican los radianes por 180/pi.

-El valor de pi no está predefinido en VBS. Para manejarlo, previamente se define una constante: CONST pi: 3.14159265358979.

-Ejemplo: cálculo del valor de pi; se utilizan las relaciones 45º=pi/4, tangente (45º)=1

```
FUNCTION fpi
        Fpi=4*ATN(1)
END FUNCTION
SUN rut 1
        Msgbox "pi="&fpi
END SUB

SUB rut2
A=ATN (1)
B=ATN (0.6789)
C=ATN (999999999)
        Msgbox "ATN(1)=" &a &"radianes" &"=" &a*180/pi &"º" &vbcrlf &"ATN(0.678)=" &b &"radianes" &"=" &b*180/pi &"º" &vbcrlf &"ATN (999999999)=" &c&"radianes" &"=" &c*180/pi &"º"
END SUB
```

· **Función COS**: devuelve el coseno de un ángulo.

-Sintaxis: COS (ángulo)

-El argumento ángulo puede ser cualquier expresión numérica válida que exprese un ángulo en radianes. El valor del coseno está entre -1 y1

-Ejemplo, a partir de la función coseno definimos la función secante.

```
FUNCTION sec(x)
        Sec =1/cos(x)
END FUNCTION
SUB rut3
        A= inputbox ("escriba un ángulo en radianes")
        Msgbox "coseno(a)=" &cos(a) &vbcrlf &"secante(a)=" &sec(A)
END SUB
```

· **Función SIN**: devuelve el seno de un ángulo expresado en radianes.

-Sintaxis: SIN (ángulo)

-El argumento ángulo puede ser cualquier expresión numérica válida que expresa un ángulo en radianes, el valor del seno está en el intervalo -1 a 1.

-Ejemplo, a partir de la función seno definimos la función cosecante.

```
FUNCTION cosec (x)
        Cosec=1/sin(x)
END FUNCTION
SUB rut4
        A= inputbox ("escriba un ángulo en radianes")
        Msgbox "seno(a)=" &sin(a) &vbcrlf &"cosecante(a)=" &cosec(A)
END SUB
```

· **Función TAN**: devuelve la tangente de un ángulo.

-Sintaxis: TAN(ángulo)

-El argumento ángulo puede ser cualquier expresión numérica válida que expresa un ángulo en radianes.

-Ejemplo, cálculo de la cotangente a partir de la tangente.

```
FUNCTION cotg(x)
        Cotg=1/tan(x)
END FUNCTION
SUB rut5
        A= inputbox ("escriba un ángulo en radianes")
        Msgbox "Tangente(a)=" &TAN(a) &vbcrlf &"cotangente(a)="
&cotg(A)
END SUB
```

· **Función EXP**: devuelve el número e=2.718282 (la base de los logaritmos neperianos) elevada a una potencia.

-Sintaxis: EXP (exponente)

-EL argumento exponente puede ser cualquiera expresión numérica válida. Si el valor del exponente sobrepasa 709,782712893 se produce un error.

-Ejemplo, cálculo del seno hiperbólico (sinh)

```
FUNCTION sinh(x)
```

```
        Sinh=(Exp(x)-Exp(-1*x))/2
    END FUNCTION
    SUB rut 6
        A=Inputbox ("escriba un número")
        Msgbox "e^" &a  &"=" &EXP(a) &vbcrlf &"seno hiperbolico(a)="
&sinh(a)
```

· **Función LOG**: devuelve el logaritmo neperiano (natural) de un número.

-Sintaxis: LOG (número)

-El argumento número puede ser cualquier expresión numérica válida mayor que cero.

-Ejemplo, cálculo de logaritmos en base decimal.

```
    FUNCTION log10 (x)
        Log10 = log (x) / log (10)
    END FUNCTION
    Sub rut7
        A=("escriba un número mayor de cero")
        Msgbox "logaritmo neperiano de(" &a &")=" &LOG(a) &vbcrlf
&"logaritmo decimal de(" &a &")=" &log10(a)
```

· **Función SQR**: devuelve la raíz cuadrada de un número.

-Sintaxis SQR (número)

-El argumento número puede ser cualquier expresión numérica válida mayor o igual que cero.

-Ejemplo

```
    Sub rut8
        A= inputbox ("escriba un número")
        Msgbox "raíz cuadrada(" &a &")=" &SQR (a)
    END SUB
```

· **Función RANDOMIZE:** inicializa el generador de números aleatorios.

-Sintaxis: RANDOMIZE [num]

-El argumento num puede ser cualquier expresión numérica válida.

-RANDOMIZE utiliza num para inicializar el generados de números aleatorios de la función RND, para ello se utiliza como semilla el valor devuelto por el temporizador del sistema.

Si no se utiliza RANDOMIZE, la función RBD (sin argumentos) la primera vez que se le llama utiliza el mismo número como semilla y después utiliza como semilla el último número generoso.

Para repetir secuencias de números aleatorios, se llamará RND con un argumento negativo inmediatamente antes de utilizar DANDOMIZE con el mismo valor para num no repite la secuencia anterior.

-Ejemplo, se inicia el generador de números aleatorios y genera un valor aleatorio entre 1 y 6.

```
    RANDOMIZE
    SUB rut9
```

```
        DO until a=vbno
                B=INT((g*RND)+1)
                Msgbox b
                A="msgbox ("¿desea repetir?", VbYesNo)
        LOOP
END SUB
```

· **Función RND**: devuelve un número aleatorio
-Sintaxis RND [(num)]
-El argumento num puede ser cualquier expresión numérica válida.
-La función RND devuelve un valor >= 0 y < 1.
-El valor de num determina como RND genera el número aleatorio:

Si número es:	**RND genera:**
Menor que cero:	El mismo número siempre, utilizando num como inicialización.
Mayor que cero:	El siguiente número aleatorio de la secuencia.
Igual que cero:	El número generado más recientemente.
No se proporciona	El siguiente número aleatorio de la secuencia.

-Para cualquier inicialización dada, se genera la misma secuencia de números ya que cada llamada sucesiva a la función RND utiliza el número anterior como inicialización para el siguiente número de la secuencia.

-Antes de llamar a RND, se utilizará la instrucción RANDOMIZE sin argumento para inicializar el generador de números aleatorios con una semilla basada en el temporizador del sistema.

-Para producir números enteros aleatorios dentro de un intervalo dado, se utilizará la siguiente fórmula:

INT ((Superior – Inferior + 1) + RND + Inferior)

Donde superior es el número mayor del intervalo e inferior es el número menor del intervalo.

-Ejemplo:

Con RANDOMIZE:	
Sub txt1_onclick C=fr1.t2.value A=fr1.txt1.value RANDOMIZE B=RND(c) A=a&vbcrlf &b Fr1.txt1.value=a END SUB	-Con un número negativo genera el mismo número. -Si el número es igual a cero generas números aleatorios. -Si el número es positivo genera números aleatorios.
Con RANDOMIZE (num):	
Sub txt2_onclick C=fr1.t2.value	-Con un número negativo genera el mismo número.

A=fr1.txt2.value RANDOMIZE (4) B=RND(c) A=a&vbcrlf &b Fr1.txt2.value=a END SUB	-Si el número es igual a cero generas el mismo número. -Si el número es positivo genera números aleatorios.
Sin RANDOMIZE: Sub txt3_onclick C=fr1.t2.value A=fr1.txt3.value B=RND(c) A=a&vbcrlf &b Fr1.txt3.value=a END SUB	-Con un número negativo genera el mismo número. -Si el número es igual a cero generas el mismo número. -Si el número es positivo genera números aleatorios.
<form id="fr1"> <Input type="text" name="t2"> <Div id="cp1" style="position absolute; visibility:true; top:30px; left:50px; width:600px; height:600px"> <textarea id="txt1" rows="20" cols="18"> Con Randomize </textarea> <textarea id="txt2" rows="20" cols="18"> Con Randomize (num) </textarea> <textarea id="txt3" rows="20" cols="18"> Sin Randomize </textarea> </form>	

-Ejemplo: Genera un valor aleatorio entre -6 y 25

Do until a=vbno
*Randomize b=INT ((25+6+1)*GND (2)-6)*
A=msgbox ("¿Desea repetir?", VbYesNo)

FUNCIONES DE REDONDEO:

· **Función ABS:** valor absoluto
-Sintaxis ABS (num)
-El argumento num puede ser cualquier expresión numérica válida. Si num contiene NULL, se devuelve NULL, si es una variable sin inicializar (empty) se devuelve cero.

-Ejemplo:

a=-45.63	**Solución**:

dim pepe pepe045 pepe=empty msgbox ABS(a) &Vbcrlf &ABS(pepe)	ABS (a) = 45.63 ABS(pepe)= 0

· **Función INT**: devuelve la parte entera de un número.

-Sintaxis INT (num)

-El argumento num puede ser cualquier expresión numérica válida.

-Si num contiene NULL, devuelve NULL.

-Si num es positivo elimina la parte decimal y devuelve la parte entera.

-Si num es negativo INT devuelve el primer número entero negativo menor o igual que el número. Por ejemplo INT convierte -6.3 en 7

· **Función FIX:**

-Sintaxis FIX (num)

-Hace lo mismo que INT salvo con los números negativos, para los que, FIX devuelve el primer número entero negativo mayor o igual que num. Por ejemplo FIX convierte -6.3 en -6.

-FIX (num)=SGN (num)*INT (ABS (num)).

-Ejemplo:

a=INT (99.8) → devuelve 99
b=FIX (99.2) → devuelve 99
c=INT (-99.8) → devuelve -100
d=FIX (-99.8) → devuelve -99
e=INT (-99.2) → devuelve -100
f=FIX (-99.2) → devuelve -99
Msgbox a &vbcrlf &b &vbcrlf &c &vbcrlf &d &vbcrlf &e &vbcrlf &f &vbcrlf

· **Función ROUND:** devuelve un número redondeado a un número especificado de cifras decimales.

-Sintaxis ROUND (expresión [, cifras])

Donde:

-Expresión.- Expresión numérica que se redondea.

-Cifras.- Opcional. Número que indica cuantas cifras decimales se incluyen en el redondeo. Si se omite, la función ROUND devuelve números enteros.

-Ejemplo:

a=345.567394
b= Round (a,2) → b=345,57
c= Round (a) → c=346
d= Round (a,3) → d=345,567
Msgbox a &vbcrlf &b &vbcrlf &c &vbcrlf &d

· **Función SGN**: devuelve un número entero que indica el signo de un número.

-Sintaxis: SGN (num)

-El argumento num puede ser cualquier expresión numérica válida. La función SGN devuelve los siguientes valores:

Si num es:	SGN devuelve:
Mayor que cero	1
Igual que cero	0
Menor que cero	-1

-Ejemplo:

a=12: b=-2.4: c=0
d=SGN (a) → devuelve 1
e=SGN (b) → devuelve -1
f=SGN (c) → devuelve 0
dim result
result=d+e+f
Msgbox result → presenta 0

TRATAMIENTO DE MATRICES O ARRAYS.

· **Función ARRAY**: devuelve un tipo variant que contiene una matriz.
-Sintaxis: ARRAY(lista de argumentos)
-Ejemplo:

```
sub rut1
        a=”pepe”
        Msgbox a
        b=”paco”
        a=ARRAY (10,b,30)
        msgbox a (10) &Vbcrlf &a (1) &vbcrlf &a (2)
End SUB
```

· **Instrucción ERASE**: reinicializa los elementos de las matrices de tamaño fijo y borra de la memoria las matrices dinámicas.
-Sintaxis: ERASE nombre_matriz
-Ejemplo

```
Dim m1 (8), m2 ()
Redim m2 (8)
M1(2)=5
M2 (2)=5
Erase m1
Erase m2
Msgbox m1 (2)
‘Msgbox m2 (2)
```

· **Función ISARRAY**: devuelve un Variant de subtipo BOOL que indica si una variable es una matriz.
-Sintaxis: ISARRAY (variable)
-Ejemplo:

```
Sub rut3
        Dim a5,m3(5),m4(),z1
        a5=”pepe”
        redim m4(6)
        m3(1)=4
        M4(2)=8
        If ISARRAY(m4)=true then msgbox”pepe”
        If not ISARRAY (a5) then msgbox”maria” &vbcrlf &ISARRAY(m3)
         &vbcrlf &ISARRAY (z1)
End Sub
```

-La función ISARRAY devuelve el valor TRUE si la variable es una matriz y el valor FALSE si no es una matriz.

· **Función LBOUND**: devuelve el índice menor disponible para la dimensión indicada de la matriz.

-Sintaxis: LBOUND(nombre matriz[,dimensión])

-Ejemplo:

```
Sub rut 4
        Dim m6(6),m10(5,4,6)
        M6(10)=5
        m6(1)=7
        m6(2)=5
        m6(3)=4
        m6(4)=5
        m6(5)=7
        m6(6)=2
        Msgbox LBOUND(m6)
        Msgbox LBOUND(m(10),2)
End Sub
```

· **Función UBOUND**: devuelve el subíndice superior disponible para la dimensión indicada de una matriz.

-Sintaxis: UBOUND 8nombre matriz[,dimensión])

-Ejemplo:

```
Sub rut5
        Dim m8(6), m9()
        Redim m9(34)
        Msgbox UBOUND (m8)
        Msgbox UBOUND (m9)
End sub
```

TRATAMIENTO DE CADENAS DE CARACTERES.

· **Función ASC,** devuelve el código del carácter ANSI que corresponde a la primera letra de una cadena.

-Sintaxis: ASC (cadena)

-el argumento cadena es cualquier expresión de cadena válida.
-Si la cadena no contiene caracteres, se producirá un error en tiempo de ejecución.
-Ejemplo:

```
Sub rut1
        a=ASC("A")          ' Devuelve 65
        b=ASC("a")          ' Devuelve 97
        c=ASC("Action")     ' Devuelve 65
        d=ASC("¬a")         ' Devuelve 172
        Msgbox "A-" &a &vbcrlf &"a-" &b &vbcrlf &"Action-" &c &vbcrlf
         &"¬a-" &d
End sub
```

- La función ASCB se utiliza con los datos de tipo byte contenidos en una cadena.
- Se proporciona ASCW para las plataformas de 32bits que utilizan caracteres unicode.

· **Función CHR,** devuelve el carácter asociado con el código de carácter ANSI especificado.
-Sintaxis: CHR (códigocar)
-El argumento códigocar es un número que identifica un carácter.
-Los números del 0 al 31 son los mismos que el código ASCII estándar no imprimibles, por ejemplo CHR(10) devuelve un carácter de avance de línea.
-Ejemplo:

```
Sub rut2
        a=CHR(65)           'Devuelve A
        b=CHR(97)           'Devuelve a
        c=CHR(62)           'Devuelve >
        d=CHR(37)           'Devuelve %
        Msgbox "65-" &a &vbcrlf &"97-" &b &vbcrlf &"62--" &c &vbcrlf
         &"37-" &d
                For c=0 to 255
                Document.write c &"-" &chr(c)  &("<br>")
                next
End sub
```

- La función CHRB se utiliza con los datos de tipo byte contenidos en una cadena.
- Se proporciona CHRW para las plataformas de 32bits que utilizan caracteres unicode.

· **Función SPACE,** devuelve una cadena formada por el número de espacios especificados.
-Sintaxis: SPACE(número)
-Donde: número es el número de espacios que tendrá la cadena.
-Ejemplo:

```
Sub rut3
        a=SPACE(10)          'Devuelve una cadena con 10 espacios
        b= "Hola"& SPACE(10) &"a todos"      ' Inserta 10 espacios entre
                                             dos cadenas.
        Msgbox "a=" &a &Vbcrlf &"b=" &b
                For c=0 to 255
                Documento.Write c& Space(5) &CHR(c) &("<br>")
                Next
End Sub
```

· **Función STRING,** devuelve una cadena de caracteres repetidos, con un número de caracteres igual al especificado.

-Sintaxis: STRING (número, carácter)

-Donde:

-Número.- número de veces que se va a repetir, si el número es NULL, devuelve NULL.

-Carácter.- código de carácter que especifica el carácter o expresión de cadena cuyo primer carácter se utiliza para construir la cadena de retorno, si carácter es NULL, devuelve NULL. SI carácter es mayor que 255, STRING, convierte el número en un código de carácter válido, para ello utiliza la fórmula: *carácter MOD 256.*

-Ejemplo:

```
Sub rut4
        a=STRING (5,"*")          ' Devuelve *****
        b=STRING (5,42)           ' Devuelve *****
        c=STRING (10,"ABC")       ' Devuelve AAAAAAAAA
        d=STRING(10,298)          '298 MOD 256=42, Devuelve **********
        Msgbox a &vbcrlf &b &vbcrlf &c &vbcrlf &d
End sub
```

· **Función LEN,** devuelve el número de caracteres que tiene una cadena o el número de bytes necesarios para almacenar una variable.

-Sintaxis: LEN(cadena|variable)

-Donde:

-Cadena, cualquier expresión de cadena válida, si la cadena contiene NULL, se devuelve NULL.

-Variable, cualquier nombre de variable válido, si variable contiene NULL, se devuelve NULL.

-Ejemplo:

```
Sub rut5
        dim m1(25), nueva
        nueva="ppppp"
        a=LEN ("esto es una prueba")        'a=18
        b=LEN (a)                           'b=2
        c=LEN(nueva)                        'c=5
        d=2345678
        e=LEN(d)                            'e=7
```

```
f="2345678"
g=LEN(f)                                    'g=7
Msgbox a &vbcrlf &b &vbcrlf &c &vbcrlf &e &vbcrlf &g
End Sub
```

La función LENB se utiliza con datos de tipo byte 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.

· **Función LEFT,** devuelve una subcadena formada por un número de caracteres desde la izquierda igual a un número especificado.

-Sintaxis: LEFT (cadena, longitud)

-Donde:

-Cadena, expresión de cadena de la que se devuelven los caracteres que estén las a la izquierda, si cadena contiene NULL, devuelve NULL.

-Longitud, expresión numérica que indica cuantos caracteres se van a devolver, si es 0 se devuelve una cadena de longitud 0 (""), si es mayor o igual que el número de caracteres en cadena, se devuelve toda la cadena.

-Ejemplo:

```
Sub rut6
    a="esto es una prueba"
    b=2
    c=5
    d="más de lo mismo"
    Msgbox left (a,7)          'Devuelve esto es
    Msgbox left (a,b+c)        'Devuelve esto es
    Msgbox a &d &"  " &left (a&d,b+c)        'Devuelve esto es una
                                                    prueba
End Sub
```

La función LEFTB se utiliza con datos de tipo byte contenidos en una cadena. En ligar de especificar el número de caracteres que se van a devolver, longitud especifica el número de bytes.

· **Función RIGHT,** Devuelve una subcadena formada por un número de caracteres desde la derecha igual a un número especificado.

-Sintaxis: RIGHT(cadena, longitud)

-Donde:

-Cadena, expresión de cadena de la que se devuelven los caracteres que estén las a la izquierda, si cadena contiene NULL, devuelve NULL.

-Longitud, expresión numérica que indica cuantos caracteres se van a devolver, si es 0 se devuelve una cadena de longitud 0 (""), si es mayor o igual que el número de caracteres en cadena, se devuelve toda la cadena.

-Ejemplo:

```
Sub rut7
    a="esto es una prueba"          'Devuelve rueba
```

```
        Msgbox right (a,5)
End Sub
```

La función RIGHTB se utiliza con datos de tipo byte contenidos en una cadena. En ligar de especificar el número de caracteres que se van a devolver, longitud especifica el número de bytes.

· **Función MID**, devuelve un número de caracteres especificado de una cadena, desde uno inicial a otro final.

-Sintaxis: MID(cadena, inicio[,longitud])

-Donde:

-Cadena: expresión de cadena de la que se devuelven caracteres, si cadena es NULL, devuelve NULL.

-Inicio: Posición del carácter en la que comienza la subcadena que se va a tomar, si inicio es mayor que el número de caracteres en cadena, MID devuelve una cadena de longitud cero (“”).

-Longitud: Número de caracteres que se va a devolver, si se omite o si existen menos caracteres del número especificado por la longitud (incluido el carácter de inicio), se devuelven todos los caracteres desde la posición de inicio hasta el final de la cadena.

-Ejemplo:

```
Sub rut8
        a=”esto es una prueba más larga”
        Msgbox mid(a,3,7)                    ‘Devuelve to es un
End Sub
```

-Admite MIDB y MIDW.

· **Función LCASE,** devuelve la cadena suministrada con todas las letras en minúscula.

-Sintaxis: LCASE(cadena)

-El argumento cadena es cualquier expresión de cadena válida, si cadena contiene NULL se devuelve NULL.

-Sólo las letras mayúsculas se convierten en minúsculas. Todas las letras minúsculas y los caracteres que no son de letra (abecedario) permanecen sin cambio.

-Ejemplo:

```
Sub rut9
        a=”esTO Cambia el TAMAÑO de las leTRAS el resto 123|?¿$% se
        quEda Igual”
        Msgbox Lcase(a)          ‘Devuelve: esto cambia el tamaño de las
                                 letras el resto 123?¿$% se queda igual.
End Sub
```

· **Función UCASE,** devuelve la cadena suministrada con todas las letras mayúsculas.

-Sintaxis: UCASE (cadena)

-El argumento cadena es cualquier expresión de cadena válida, si cadena contiene NULL se devuelve NULL.

-Sólo las letras mayúsculas se convierten en minúsculas. Todas las letras minúsculas y los caracteres que no son de letra (abecedario) permanecen sin cambio.

-Ejemplo:

```
Sub rut9
        a="esTO Cambia el TAMAÑO de las leTRAS el resto 123|?¿$% se
        quEda Igual"
        Msgbox UCASE(a)     'Devuelve: ESTO CAMBIA EL TAMAÑO DE LAS
                            LETRAS EL RESTO 123?¿$% SE QUEDA IGUAL.
```

· **Función StrReverse**, devuelve una cadena en la que se invierten el orden de caracteres de la cadena especificada.

-Sintaxis: StrReverse(cadena)

-Donde cadena es la cadena cuyas características se van a invertir, si cadena es una cadena e longitud 0("") se devuelve una cadena de longitud cero, si cadena es null se devuelve null

-Ejemplo:

```
Sub rut1
        A="este es el orden de las letras"
        Msbox StrReverse (a)
End Sub
```

· **Función LTRIM**, devuelve la cadena suministrada sin los posibles espacios que tuviese a la izquierda.

-Sintaxis: LTRIM(cadena)

-El argumento cadena es cualquier expresión de cadena válida, si contiene null se devuelve null.

-Ejemplo:

```
Sub rut2
        a="          se eliminan los espacios de la izquierda      )"
        Msgbox a&vbcrlf LTRIM(a)
End Sub
```

· **Función RTRIM,** devuelve la cadena suministrada sin los posibles espacios que tuviese a la derecha.

-Sintaxis: RTRIM (cadena)

-El argumento cadena es cualquier expresión de cadena válida, si contiene null se devuelve null.

-Ejemplo:

```
Sub rut2
        a="(se eliminan los espacios de la derecha      )"
        Msgbox a &vbcrlf &"pepe" &vbcrlf &RTRIM(a) &"pepe"
End Sub
```

· **Función TRIM**, devuelve la cadena suministrada sin los posibles espacios que tuviese a la izquierda y a la derecha.

-Sintaxis: TRIM (cadena)

-EL argumento de cadena es cualquier expresión de cadena válida, si cadena contiene null, se devuelve nulll.

· **Función StrComp**, devuelve un valor que indica el resultado de una comparación de cadenas.

-Sintaxis: StrComp (cadena1,cadena2[,comparar])

Donde:

-Cadena1, cualquier expresión de cadena válida.

-Cadena2, cualquier expresión de cadena válida.

-Comparar.- Opcional. Valor numérico que indica el tipo de comparación que se va a actualizar cuando se evalúen cadenas. Si se omite, se realiza una comparación binaria.

-El argumento comparar puede tener los siguientes valores:

Constante	**Valor**	**Descripción**
VbBinaryCompare	0	Realiza una comparación binaria
CbTextCompare	1	Realiza una comparación textual, es decir, no distingue de mayúsculas y minúsculas

-La función StrCompare tiene los siguientes valores de retorno:

Si	**StrCompare devuelve:**
Cadena 1 es menor que cadena2	-1
Cadena 1 es igual que cadena 2	0
Cadena 1 es mayor que cadena 2	1
Cadena 1 o cadena 2 es null	NULL

-Ejemplo:

```
Sub rut5
    Dim cd1,cd2,comp1,comp2,comp3
    cd1="ABCD" : cd2="abcd"
    Comp1=StrComp (cd1,cd2,1)          'Devuelve0
    Comp2=StrComp (cd1,cd2,0)          'Devuelve-1
    Comp3=StrComp (cd2,cd1)            'Devuelve1
    Msgbox"Se comparan Cd1=" &cd1&valor devuelto" &comp1 &vbcrlf
&"se hace una comparación binaria de cd1 con cd2"& valor devuelto" &comp2
&vbcrlf &"rehace una comparación binaria de cd2 con cd1"&"valor devuelto"
&comp3
End Sub
```

· **Función REPLACE**, donde una cadena A devuelve otra cadena B en la que una subcadena C de A se ha sustituido por otra subcadena D un número de veces especificado.

-Sintaxis: REPLACE(expresión,buscar,reemplazarcon[,inicio[,num[comparar]]])

Donde:

-Si se expresa num hay que especificar inicio.

-Si se especifica comparar hay que especificar num e inicio.

-Expresión, expresión de cadena que contiene la subcadena que se va a reemplazar.

-Buscar, subcadena que se va a reemplazar.

-ReemplazarCon, subcadena que se va a reemplazar, si se omite, se supone 1. Se debe utilizar junto con num, se aconseja poner siempre1

-Num, opcional, numero de sustituciones de subcadena que se van a realizar, si se omite, el valor predeterminado es -1, que significa hacer todas las sustituciones posibles. Se debe utilizar junto con inicio.

-Comparar, igual que función StrComp

-Replace devuelve los siguientes valores:

Si	**Replace devuelve**
Expresión de de longitud 0	Cadena de longitud 0 (“”)
Expresión es Null	Un error
Buscar es de longitud 0	Copia de expresión
Reemplazar es de longitud 0	Copia de expresión con todas las apariciones
Inicio>Len(expresión)	Cadena de longitud 0
Num es 0	Copia de expresión

El valor de retorno de la función REPLACE es una cadena, con sustituciones realizadas que el comienza en la posición especificada por inicio y termina al final de la cadena expresión. No es una copia de la cadena original de principio a fin.

-Ejemplo:

```
Sub rut6
    A=”verde que te quiero verde, verde como la albahaca VERDE”
    B= Replace (a,”verde”,”azul”)
    C=Replace (a,”verde”,”azul”,1,-1,1)
    D=Replace (a.”verde”,”azul”,1,2,1)
    Msgbox “Replace (a,””verde””,””azul””):” &b &Vbcrlf & Replace
    (a,””verde””,””azul””,1,-1,1):” &c &vbcrlf & Replace
    (a,””verde””,””azul””,1,2,1):” &d
End Sub
```

· **Función INSTR**, devuelve la posición de la primera aparición de una cadena dentro de otra.

-Sintaxis: INSTR([inicio,]cadena1,cadena2[,comparar])

Donde:

-Inicio, opcional, expresión numérica que establece la posición de inicio para cada búsqueda. Si se omite, la búsqueda comienza en la primera posición de la cadena 1. Si inicio contiene null, se produce un error. El argumento inicio es necesario si se especifica comparar.

-Cadena 1, expresión de cadena en la que se busca.

-Cadena2, Expresión de cadena buscada

-Comparar, igual que función StrComp

-La función InStr devuelve los siguientes valores:

Si	InStr devuelve:
Cadena 1 es de longitud 0.	0
Cadena 1 es Null.	Null
Cadena 2 es de longitud 0.	Inicio
Cadena 2 es Null.	Null
Cadena 2 no se encuentra.	0
Cadena 2 se encuentra dentro de cadena1	Posición en la que se encuentra la coincidencia
Inicio >Len(cadena2)	0

-Existe INSTRB que trabaja con datos de tipo byte.
-Ejemplo:

```
Sub rut7
        a=”siempre te diré verde que te quiero verde, verde como la albahaca
         VERDE”
        b=”Verde”
        c=InStr (a,b)
        d=Instr (40,a,b,1)
        Msgbox a &Vbcrlf &”InStr(a,b):” &c & Vbcrlf &”InStr (40,a,b,1):” &d
End sub
```

· **Función InStrRev,** devuelve la posición de una ocurriencia de una cadena dentro de otra, desplazándose desde el final de la cadena o desde la posición específica hacia la izquierda. Las posiciones sierre se cuentan desde la izquierda. La búsqueda se hace de derecha a izquierda.

-Sintaxis: InStrRev (cadena1,cadena2[,inicio[,comparar]])
Donde:
-Cadena1: Expresión de cadena en la que se busca.
-Cadena2: Expresión de cadena que se busca.
-Inicio: opcional, expresión numérica que establece la posición de inicio para cada búsqueda. Si se omite, se utiliza -1, que significa que la búsqueda comienza en la posición del último carácter. Si inicio contiene null, se produce un error.
-Comparar, igual que StrComp.
-InStrRev devuelve los siguientes valores:

Si	InStr devuelve:
Cadena 1 es de longitud 0.	0
Cadena 1 es Null.	Null
Cadena 2 es de longitud 0.	Inicio
Cadena 2 es Null.	Null
Cadena 2 no se encuentra.	0
Cadena 2 se encuentra dentro de cadena1	Posición en la que se encuentra la coincidencia
Inicio >Len(cadena2)	0

-Ejemplo:

```
Sub rut8
        a="siempre te diré verde que te quiero verde, verde como la albahaca
         VERDE"
        b="Verde"
        c=InStRev (a,b)
        d=InstrRev (a,b,30,1)
        Msgbox a &Vbcrlf &"InStrRev(a,b):" &c & Vbcrlf &"InStrRev
        (a,b,30,1):" &d
End sub

Sub rut9
        a="Pedro tenia una parra. Parra tenía una perra"
        b="p"
        c=InStrRev (a,b,10,0)          'Devuelve 1
        d=InStrRev (a,b,-1,1)          'Devuelve 40
        e=InStrRev (a,b,8)             'Devuelve 1
        Msgbox c &Vbcrlf &d &vbcrlf &e
End sub
```

· **Función JOIN,** devuelve una cadena creada al combinar un número de subcadenas contenidas en una matriz.

-Sintaxis: JOIN(nombrematriz[,delimitador])

Donde:

-Nombrematriz: nombre de una matriz unidimensional que contiene subcadenas que se van a combinar.

-Delimitador: opcional, carácter de cadena utilizado para separar las subcadenas en la cadena de retorno. Si se omite se utiliza el carácter espacio(" "), si delimitador es una cadena de longitud cero todos los elementos de la lista están concatenador sin ningún delimitador.

-Ejemplo:

```
Sub rut10
        Dim a
        Dim m(3)
        m(0)="Sr."
        m(1)="Juan"
        m(2)="Pérez"
        m(3)="Rodríguez"
        a=Join (m)                 'A contiene "Sr. Juan Pérez Rodríguez"
        b=Join (m,"-")             'B contiene"Sr.-Juan-Pérez-Rodriguez"
        Msgbox a &vbcrlf &b
End sub
```

· **Función SPLIT,** devuelve una matriz unidimensional con base cero que contiene un número de subcadenas especificado.

-Sintaxis: SPLIT (expresión[,delimitador[,número[,comparar]]])

-Expresión: expresión de cadena que contiene subcadenas y delimitadores, si expresión es una cadena de longitud cero, SPLIT devuelve una matriz vacía, es decir, una matriz sin elementos ni datos.

-delimitador: opcional, carácter de cadena utilizado para identificar límites de subcadenas. Si se omite, se supone que el carácter de espacio (“ “) es el delimitador. Si delimitador es una cadena de longitud cero, se devuelve una matriz de elemento único que contiene la cadena expresión completa.

-Número: opcional, número de subcadenas que se van a devolver, -1 indica que se devuelven todas las subcadenas.

-Comparar: igual que función StrComp.

-Ejemplo:

```
Sub rut11
    Dim a,mt
    a=”En un lugarXdela ManchaX de cuyo nombre no quiero acordarme”
    mt=Split (a,”x”,-1,1)
    ‘mt(0)contiene “En un lugar”
    ‘mt(1) contiene “de la Mancha”
    ‘mt(2) contiene”de cuyo nombre no quiero acordarme”
    f=ubound(mt)
        for c=0 to f
        document.write mt(c) &(“<br>”)
        next
    Msgbox mt(0) &vbcrlf &mt(1) &vbcrlf &mt(2)

End sub
```

· **Función FILTER**, devuelve una matriz de base cero que contiene un subconjunto de una matriz de cadena obtenido mediante un filtro especificado.

-Sintaxis: FILTER (matriz entrada, valor[,incluir[,comparar]])

Donde:

-Matriz Entrada.- es una matriz unidimensional de cadenas en la que se va a buscar.

-Valor.- Cadena que se va a buscar.

-Incluir.- Valor Booleano. Indica si se devuelven subcadenas que incluyan o excluyan un valor. Si incluir es TRUE, FILTER devuelve el subconjunto de la matriz que contiene valor como subcadena. Si incluir es FALSE, FILTER devuelve el subconjunto de la matriz que no contenga el valor como una subcadena.

-Comparar: igual que función StrComp.

-Si no se encuentran coincidencias el valor dentro de matriz entrada, FILTER devuelve una matriz vacia. Se produce un error si matriz entrada es NULL o no es una matriz unidimensional. La matriz devuelta por la función FILTER contiene sólo elementos suficientes para contener el número de elementos coincidentes.

-Ejemplo:

```
Sub rut12
    Dim m(6),a
    M(0)=”domingo”
    m(1)=”lunes”
    m(2)=”martes”
    m(3)=”miércoles”
    m(4)=”jueves”
```

```
        m(5)="viernes"
        m(6)="sabado"
        A=FILTER 8m,"lunes",false)
        FOR c=0 to UBOUND (m)
                Document.write a(c)&("<br>")
        Next
End Sub
```

PROGRAMACIÓN ORIENTADA A OBJETOS (OOP) EN VBSCRIPT.

En la oop se considera que el mundo está formado por objetos.

Para resolver un problema se identificará cada una de las partes del problema con un objeto.

Todos los objetos tienen dos características, un estado y un comportamiento, el estado se define mediante una serie de atributos o propiedades que permiten diferenciar el objeto de otros objetos del mismo tipo.

El comportamiento se define mediante las acciones o métodos que puede realizar y los secesos o eventos a los que puede responder.

Un suceso se manifiesta como una relación entre dos objetos, en general al establecerse a relación que producirá una respuesta o reacción de los objetos, los cuales se manifiestan como un conjunto de métodos propios que se activa.

CLASES.

Una clase es el prototipo que define un tipo de objeto determinado, la clase es un concepto abstracto que generalmente no utilizamos directamente en nuestros programas.

Lo que utilizaremos son objetos concretos que son instancias, materializaciones o clases de una clase determinada, pro ejemplo el molde para hacer galletas es una clase y las galletas que hacemos a partir del molde son objetos concretos o instancias creadas a partir de las características definidas por el molde.

Si comparamos las clases y objetos de la OOP con la programación estructurada tradicional puede decirse que las clases son los tipos de datos y los objetos concretos o instancias son las variables de esos tipos de datos, por ejemplo: un tipo INTEGER en OOP diríamos que es la clase INTEGER, u y una variable de tipo INTEGER diríamos que es un objeto o una instancia de la clase INTEGER.

El soporte de la programación orientada a objetos es VBS es limitado, una clase permite declarar las propiedades y métodos que poseerá una instancia de la clase. No se pueden declarar eventos para las clases, solamente existen dos elementos predefinidos, el evento INITIALIZE y el evento TERMINATE. Tampoco soporta la herencia.

Nota: el VBS 6.0 se pueden declarar eventos para las clases, se soporta el polimorfismo y un tipo especial de herencia basada en el polimorfismo. En VBS se soporta la OOP completa.

CREACIÓN DE CLASES EN VBScript.

Una clase define los atributos, los métodos y las propiedades que va a poseer los objetos o instancias de esa clase, para crear una clase se utiliza un bloque CLASS con la siguiente sintaxis:

```
CLASS nombre clase
        Código normal de VBS
        Declaración de atributos
        Declaración de métodos
        Declaraciones de propiedades
END CLASS
```

Dentro de un bloque CLASS se pueden definir variables y procedimientos que pueden ser públicos o privados, por defecto son públicos. Si son privados sólo puede accederse a ellos dentro del bloque CLASS y si son públicos puede accederse a ellos desde fuera, desde cualquier secuencia de comandos.

Al programar con objetos en VBS es conveniente distinguir entre lo que se puede y lo que se debe hacer. Los atributos de una clase (definen el estado de la clase) se representan mediante variables y se declaran de la siguiente manera:

```
{Private|Public}nombre_variable
```

Aunque se pueden declarar como públicos o privados deben declararse como privados para que se pueda acceder a ellos depende fuera de la clase (por defecto son públicos).

Dentro de un bloque CLASS se pueden declarar procedimientos SUB o FUNCTION que se pueden definir como públicos o como privados, por defecto son públicos, los procedimientos que se declaran como públicos serán métodos de la clase. Todos los procedimientos que no vallan a ser métodos, que no vallan a definir el comportamiento de la clase deben declararse como privados.

Una clase puede tener un y sólo un método público por defecto (método que se ejecuta siempre que se especifique otro) para establecer el método por defecto se utiliza la palabra clave DEFAULT.

La sintaxis para declarar un método es la siguiente:

```
PUBLIC [DEFAULT] Sub nombre[(lista parámetros formales)]
        [Instrucciones 1]
        [If condición THEN EXIT SUB]
        [Instrucciones 2]
End Sub
PUBLIC [DEFAULT] Function nombre[(lista parámetros)]
        [instrucciones 1]
        [nombre=expresión *]
        [IF condición THEN EXIT Function]
        [Instrucciones 2]
        [nombre=expresión * ]
End Function
```

* Al menos debe de haber uno de los dos

· **Métodos PROPERTY (propiedades)**, hay un tipo especial de métodos llamados property que se utiliza para devolver o acceder a los atributos de una clase de manera controlada. Mediante los métodos property se puede acceder (asignar un valor u obtener un valor) a los atributos declarados como privados dentro de la clase.

De esta manera podemos encapsular los atributos de una clase para que su implementación permanezca oculta fuera de la clase, existen tres métodos property:

-Property LET, permite asignar un valor, es decir, define la escritura o modificación de una propiedad.

-Property SET, hace lo mismo que el anterior cuando la propiedad es un objeto.

-Property GET, permite obtener un valor, es decir, define la lectura de una propiedad.

Las propiedades pueden ser:

1.- De sólo lectura, sólo tienen un método property GET

2.- De sólo escritura, cada propiedad solo tiene un método property LET o si es un objeto un método property SET.

3.- De lectura y escritura, cada propiedad tendrá un método property GET y un método property LET con el mismo nombre (si la propiedad es un objeto puede ser property SET).

Los métodos property pueden ser públicos o privados pero deben de ser públicos y sólo uno de ellos puede ser el método por defecto.

Si bien pueden tener una lista de parámetros formales no deben de tenla.

En consecuencia, la sintaxis de un método o propiedad es la siguiente:

```
PUBLIC [DEFAULT] property GET nombre
        [instrucciones 1]
        [[SET ]nombre= expresión *]
        [IF condición THEN exit property]
        [Instrucciones 2]
        [[SET ]nombre= expresión *]
END PROPERTY
```

* Al menos debe de haber unos de los dos.

Se utiliza SET cuando el valor devuelto en nombre sea un objeto.

Sintaxis del método property LET:

```
PUBLIC [DEFAULT] property LET nombre (valor*)
        [Instrucciones 1]
        [IF condición THEN exit Property]
        [Instrucciones 2]
END PROPERTY
```

* Valor asignado a la propiedad.

Si la propiedad es un objeto se utiliza con Property SET con la siguiente sintaxis:

```
PUBLIC [DEFAULT] property SET nombre (referencia)
        [Instrucciones 1]
        [IF condición THEN exit Property]
        [Instrucciones 2]
END PROPERTY
```

Nota: si por las razones que sea se utilizase una lista de parámetros formales en la declaración de una propiedad de lectura-escritura (tendrá property GET y property LET con el mismo nombre) el número de parámetros formales correspondientes tendrán el mismo nombre en el property GET y en el property LET.

PROPERTY GET nombre1(x,y,z)
PROPERTY LET nombre1 (x,y,z)

EVENTOS INITIALIZE Y TERMINATE.

El evento INITIALIZE se produce siempre que se crea una instancia de la clase y el evento TERMINATE se produce cuando se borra de la memoria una instancia de la clase, para estos eventos se pueden escribir procedimientos controlados por sucesos que tendrán la siguiente sintaxis:

```
PRIVATE sub CLASS_ INITIALIZE
        [Instrucciones]
END SUB
PRIVATE sub CLASS_TERMINATE
        [Instrucciones]
END SUB
```

-Ejemplo:

```
CLASS rectángulo
    'Definición de los atributos de la clase
        PRIVATE acolor, abase,aaltura
    'Acciones que se producen al inicializar una instancia de la clase
        PRIVATE sub CLASS_INITIALIZE
            acolor="verde"
        END SUB
    'Acciones que se producen al borrar una instancia de la clase
        PRIVATE sub CLASS_TERMINATE
            msgbox "el objeto ha sido borrado de la memoria"
        END SUB
    'Propiedad de la clase
    'Propiedad de Sólo lectura: color
        PUBLIC property GET color
            Color=acolor
        END PROPERTY
    'Propiedad de lectura y escritura: base
        PUBLIC DEFAULT property GET base      'por defecto
            base0abase
        END PROPERTY
        PUBLIC property LET base (V)
    'Se hace una verificación de los datos que se le pasan
            IF v<0 THEN
                Abase=0
            ELSE
```

```
                    Abase=v
                END IF
            END PROPERTY
        'Propiedad de lectura y escritura: altura
            PUBLIC property GET altura
                altura=aaltura
            END PROPERY
            PUBLIC property LET altura (w)
                IF w>0 THEN
                    Aaltura=0
                ELSE
                    Aaltura=0
                END IF
            END PROPERTY
        'Se define el método para calcular el área del rectángulo.
            PUBLIC function area
                area=abase*aaltura
            END FUNCTION
        'Se define un método para calcular una potencia del área del rectángulo.
            PUBLIC Function potencia (e)
                Potencia=area^e
            END FUNCTION
        'Se define un método para calcular el perímetro del rectángulo.
            PUBLIC Function perímetro
                Perímetro =2*abase+2*aaltura
            END FUNCTION
        'Se define un método que nos dice si el área es mayor, menor o igual.
            PUBLIC Sub acertijo (n1)
                IF area<n1 THEN
                    Msgbox "el área es menor"
                ELSEIF area>n1 THEN
                    Msgbox "el área es mayor"
                ELSEIF area=n1 THEN
                    Msgbox "ha acertado"
                END IF
            END SUB
END CLASS
```

CREACIÓN DE INSTANCIAS DE UNA CLASE.

Para crear una instancia o un objeto concreto de una clase se utiliza la instrucción SET con la siguiente sintaxis:

```
SET NombreObjeto=New Nombre_Clase
SET rtg1=New rectángulo
```

Utilización de las propiedad de un objeto o una instancia: para establecer (escribir o asignar) el valor de una propiedad se utiliza la siguiente sintaxis:

```
NombreObjeto.NombrePropiedad=expresión
```

Por ejemplo: le damos a la propiedad base del rectángulo rtg1 un valor negativo

```
Rtg1.base=-3
```

Para obtener o leer el valor de una propiedad se hace lo siguiente:

```
Variable=NombreObjeto.NombrePropiedad
```

O manejar directamente NombreObjeto.NombrePropiedad.

No obstante si se va a utilizar el valor de una propiedad varias veces el código se ejecutará más rápido si se almacena el valor de la propiedad en una variable.

```
'Se crean nuevos objetos de la clase rectángulo
    SET rtg2=New rectángulo
    SET rtg3=New rectángulo
'Comprobamos los valores del color
    Msgbox "color de rtg1=" &rtg1.color &vbcrlf_
        &"Color de rtg2=" &rtg2.color &vbcrlf_
        &"Color de rtg3=" &rtg3.color
```

```
Dim txt,ct,pr
pr=0
FOR ct=1 to LEN(x)
    txt=MID(x,ct,1)
    IF ASC(txt)>47 AND ASC(txt)<58 THEN
        Msgbox"le título no es correcto"
        Pr=1
        EXIT FOR
    END IF
NEXT

DIM vv
Vv=Inputbox( "escriba un título")
rtg1.tit=""
DO WHILE rtg1.tit=""
    vv=Inputbox ("escriba un título)
    rtg1.tit=vv
LOOP
Msgbox rtg1.tit
```

Comprobamos los valores de las bases con diversas sintaxis:

```
Dim base1
base1=rtg1.base
Msgbox "Base de rtg1=" &base1 &Vbcrlf_
    &"Base de rtg2=" &rtg2.base &vbcrlf_
    &"Base de rtg3=" &rtg3      'Es la propiedad por defecto
```

Utilización de los métodos en el código:

- Métodos sin argumento, la sintaxis será: NombreObjeto.NombreMétodo

‘Obtenemos el área de los rectángulos

```
Msgbox”Área rtg1=” &rtg1.area &vbcrlf_
      “Área rtg2=” &rtg2.area &vbcrlf_
      “Área rtg3=” &rtg3.area &vbcrlf
```

- Si el método necesita argumentos y ha sido declarado con FUNCTION los argumentos se escribirán entre paréntesis.
- Si el método necesita argumentos y se definió con SUB los argumentos pueden escribirse entre paréntesis o sin paréntesis.

‘Queremos obtener el área del rectángulo elevado al cubo

```
Msgbox “el cubo del área de rtg1=” &rtg1.potencia(3)
```

· **Instrucción WITH,** ejecuta un conjunto de instrucciones en un único objeto, la sintaxis será:

```
WITH objeto
      Instrucciones
END WITH
```

En donde:

-Objeto, puede ser el nombre de un objeto o de una función que devuelve un objeto.

-Instrucciones, puede ser cualquier conjunto de instrucciones válidas en Visual Bascis Script (VBScript).

Nota 1: no se saltará dentro o fuera de un bloque with, si se ejecutan instrucciones de un bloque with pero no se ejecuta la instrucción inicial with o la instrucción final end with pueden producirse errores.

Nota 2: Objeto no puede cambiarse dinámicamente, el bloque with sólo se puede aplicar a un único objeto.

Nota 3: Se pueden anidar bloque with dentro de otros bloques with.

Nota 4: Estos bloques son muy útiles para establecer las propiedades de un objeto, por ejemplo:

```
WITH rtg1
      ‘base=25
      ‘altura=10
END WITH
Msgbox rtg1.base &vbcrlf &rtg1.altura &vbcrlf
```

‘Borramos de la memoria los objetos rtg1, rtg2, rtg3.

```
SET rtg1=Nothing
SET rtg2=Nothing
SET rtg3=Nothing
```

· **Función GETREF,** devuelve una referencia a un procedimiento que se puede enlazar con un evento.

-Sintaxis: SET NombreObjeto.NombreEvento=GETREF
("nombreprocedimiento")

-Permite conectar un procedimiento de VBS (Sub o Function) a cualquier evento en una página Web. Los eventos y los objetos disponibles son los establecidos en el modelo de objetos de Microsoft DHTML (que actualmente sigue el DOM del W3C)

-Ejemplo:

```
Sub prueba
        Msgbox"pepe"
End Sub
Set window.onload=getref("prueba")
```

OBJETOS PREDEFINIDOS.

VBS incorpora un conjunto de clases u objetos predefinidos, como por ejemplo ERR, REGEXP, MATCH, y otros que soporta Windows Scripting Host.

Instancias de un objeto u clase predefinido, para poder manejar las clases predefinidas hay que crear una instancia de ellos y asignarla a una variable para poder referirse al objeto con un nombre, la sintaxis general es:
SET variable=CreateObjet("Aplicación_Suministradora _del _Objeto.NombreObjeto")

En caso de que sea VBScript la aplicación que suministra el objeto se utiliza la palabra SCRIPTING con lo que quedará:

SET variable=CreateObjet("Scripting.Nombre_Objeto")

OBJETO ERR.

Es un objeto intrínseco o predefinido con alcance global y no es necesario una crear una instancia del elemento en el código.

La propiedad predeterminada es NUMBER por lo que es equivalente a escribir ERR.NUMBER o ERR, esta propiedad guarda el código de error correspondiente al error producido. Si no se produce ningún error esta propiedad vale 0. Por lo tanto para detectar si se ha producido un error se usara:

IF ERR>0 THEN..... o bien IF ERR.NUMBER>0THEN.....

· **Propiedad DESCRIPTION:** ofrece una descripción del error producido, existe una descripción predeterminada (ver fotocopia) pero se puede cambiar.

-Sintaxis: ERR.DESCRIPTION 'descripción predefinida.
ERR.DESCRIPTION="texto 'establce una nueva descripción del error

-Ejemplo:

```
IF ERR=451 THEN msgbox Err.description="el objeto no pertenece a la
                                                colección"
```

· **Propiedad SOURCE:** indica el nombre del objeto o aplicación que causó el error.

-Sintaxis: ERR.SOURCE 'nos da el nombre del objeto o aplicación que generó el error

ERR.SOURCE="texto" 'Establece el nombre del objeto o aplicación que originó el error

· **Propiedad HELPFILE:** establece o devuelve una ruta de acceso completa a un archivo de ayuda.

-Sintaxis: ERR:HelpFile[="ruta de acceso"]

· **Propiedad HELPCONTEXT:** establece o devuelve una cadena descriptiva asociada a un error.

-Sintaxis: ERR.HelpContext["identificador"]

-El identificador ha de ser válido para un tema del archivo de ayuda.

· **Método CLEAR:** borra toda la configuración de propiedades del objeto ERR, VBS llama automáticamente a este método después de:

-ON ERROR RESUME NEXT

-Exit SUB

-Exit FUNCTION

-Sintaxis: ERR.CLEAR

· **Método RAISE:** genera un error en tiempo de ejecución.

-Sintaxis:

ERR.RAISE(número,[origen],[descripción],[archivodeayuda],[contextoayuda]

Donde:

-Número: código de error que va de 0 a 65535

-Origen: Es una cadena que da nombre al objeto o aplicación que originalmente generó el error.

-Descripción: una cadena que describe el error.

-Archivo ayuda: ruta de acceso completa del archivo de ayuda.

-Contexto ayuda: identifica un tema dentro de archivo de ayuda.

Nota: Si no se especifican los valores optativos, se obtendrán los valores anteriores correspondientes que no han sido borrados.

TRATAMIENTO DE ERRORES EN VBScript.

El tratamiento de errores en VBS es bastante limitado, para capturar errores en código de VBS se utiliza la sentencia ON ERROR RESUME NEXT, a partir del momento en que se encuentra una sentencia de este tipo, el interprete de VBS después de cada línea de código que ejecuta comprueba si se ha producido algún error y rellena las propiedades del objeto ERR adecuadamente, inmediatamente después de ejecutar esta sentencia, todos las propiedades del objeto ERR se reestablecerán a cero o a cadena vacía.

Si posteriormente se produce un error en la ejecución de un programa, las propiedades de ERR almacenan la información que identifica el error producido de forma única y que es útil para procesar después el error.

Puesto que no existe una sentencia de tipo ON ERROR GOTO etiqueta, no se puede escribir una rutina o procedimiento para tratar el error que permita realizar las acciones adecuadas, por este motivo si se quiere informar sobre el error producido habrá que escribir después de cada línea de código algo parecido a lo siguiente:

```
IF ERR>0 THEN e1=”se ha producido el error=”&ERR&ERR.DESCRIPTION
Msgbox e1
```

A partir del momento en que se ejecuta ON ERROR RESUME NEXT si se produce un error en tiempo de ejecución no se detiene la ejecución del programa si no que se usa a ejecutar la siguiente instrucción.

Para detener la captura de errores se ejecutará la siguiente sentencia: ON ERROR GOTO 0, a partir del momento en que se ejecute cualquier error que se produzca en tiempo de ejecución se detiene la ejecución del programa.

Si la captura de errores se activa con ON ERROR RESUME NEXT dentro de un procedimiento la captura de errores se desactiva al salir del procedimiento o al llamar a otro procedimiento. En consecuencia se utilizará la sentencia ON ERROR RESUME NEXT al comienzo de todos los procedimientos si se quieren capturar todos los errores.

CONTROLES ACTIVEX.

Los controles ActiveX son controlados de la misma manera por VBS y por Internet Explorer. Los controles ActiveX están en archivos independientes, para incluir uno de estos controles en una página Web se hará lo siguiente:

Se incluye el control mediante un bloque <objet></objet> en la que normalmente se establecerán los atributos siguientes:

-ID, para identificar el control de una manera única y que pueda hacer referencia a él desde otros elementos HTML.

-CLASSID, identificador de clase, establece la forma en que se selecciona el control que se va a utilizar. Es una cadena de caracteres bastante compleja, por ejemplo:

```
CLASSID=”CLASID:8BD21D40-EC42-11CE-9EOD-00AA00-6002F3”
```

SCRIPTS EN ARCHIVOS INDEPENDIENTES.

Si el código de un script es demasiado grande se puede guardar en un archivo de texto independiente. La extensión puede ser cualquiera, pero quizá debe ponerse .VBS, la sintaxis para llamar de una página a ese archivo es la siguiente:

```
<script language=”vbscript” src=”trayectoria\archivo.vbs”>
<!—
...
-->
</script>
```

GESTIÓN DE SUCESOS O EVENTOS EN DHTML.

Un suceso o evento es una notificación del navegador referente a que algo ha cambiado. Generalmente debido a que el usuario ha realizado alguna acción.

Existen muchos tipos de sucesos, pero los sucesos básicos pueden agruparse en cuatro tipos:

1.- Sucesos del ratón.

2.- Sucesos del teclado.

3.- Sucesos de fofo y selección.

4.- Sucesos de cambio de estado, generalmente no se producen por acciones del usuario, sino cuando cambia el estado del documento, por ejemplo, al cargar la página.

Los sucesos del ratón y del teclado son sucesos generales y están disponibles para todos los elementos o etiquetas de un documento.

Todos los tipos de sucesos se pueden generar y se pueden gestionar mediante scripts.

Sucesos del teclado:

1.- Suceso OnKeyDown, se genera al pulsar una tecla y antes de soltarla.

-El valor de la tecla pulsada se almacena en la propiedad KEYCODE del objeto WINDOW.EVENT.

a=Window.Event.KeyCode.Value

-En a se almacena un variant del subtipo Integer que es igual al código Unicote de la tecla pulsada.

2.- Suceso ONKEYPRESS, se genera siempre que se pulsa y se suelta una tecla.

-El valor de la tecla pulsada se almacena en la propiedad KEYCODE del objeto WINDOW.EVENT.

a=Window.Event.KeyCode.Value

-En a se almacena un variant del subtipo Integer que es igual al código Unicote de la tecla pulsada.

-Si al mismo tiempo que se pulsa una tecla se pulsa:

-la tecla ALT, se puede detectar porque Window.altkey.value=true, sino sería False.

-la tecla CTRL se puede detectar porque Window.ctrlkey.value=true, sino sería False.

-la tecla SHIFT, se puede detectar porque Window.shiftkey.value=true, sino sería False.

3.- Suceso ONKEYUP, se genera al soltar una tecla pulsada anteriormente.

-El valor de la tecla pulsada se almacena en la propiedad KEYCODE del objeto WINDOW.EVENT.

a=Window.Event.KeyCode.Value

-En a se almacena un variant del subtipo Integer que es igual al código Unicote de la tecla pulsada.

4.- Suceso ONHELP, se genera cuando se solicita ayuda pulsando una tecla relacionada con la ayuda en el teclado, generalmente Help o F1.

Sucesos del Foco, se generan cuando el usuario indica a la aplicación que está interactuando con un cierto elemento de la página. Esto se hace pulsando con el ratón en el elemento o bien navegando hasta él con la tecla TAB.

1.- Suceso ONFOCUS, se genera cuando el elemento recibe el foco (atención del usuario).

2.- Suceso ONBLUR, se genera siempre que un elemento pierde el foco. Es decir, se pulsa con el ratón en otro elemento o la tecla TAB para pasar a otro elemento.

Sucesos de selección, se genera cuando se pulsa y se arrastra el ratón por encima de ciertos elementos de la página.

1.- Suceso ONSELECTSTART, se genera cuando el usuario pulsa para comenzar la selección.

2.- Suceso ONSELECT, se genera cuando el usuario hace realmente una selección (mantiene el botón del ratón pulsado y lo arrastra seleccionando elementos).

3.- Suceso ONDRAGSTART, se produce cuando se tiene una zona seleccionada y se desea arrastrarla.

Sucesos de cambio de estado.

1.- Suceso ONREADYSTARTCHANGE, se genera siempre que el documento alcanza un hito en un proceso de carga, esto permite seguir la carga del documento y tomar decisiones basándose en la información obtenida.

Se puede saber el estado actual del documento con la propiedad READY.STATE del objeto DOCUMENT, la sintaxis sería:

A= Document.Ready.State

Los valores que puede tener A son:

-Complete. El documento está totalmente cargado.

-Interactive. Se puede interactuar con el documento aunque no está totalmente cargado.

-Loading. El documento está cargado.

-UnInitialized. Se está descargando el documento desde el servidor.

2.- Suceso ONLOAD, se genera cuando el documento acaba de cargarse. Es decir, se ha cargado el archivo HTML y todos los elementos en archivos independientes 8imágenes, hojas de estilo, código VBS,..).

3.- Suceso ONUNLOAD, se genera cuando el navegador descarga el documento del servidor, cuando el documento sufre una modificación y cuando se pulsa el botón de refresco o la tecla F5 que vuelve a cargar el documento.

4.- Servidor ONABORT, sólo se utiliza con la etiqueta <IMG>. Se genera cuando se detiene la descarga de imágenes desde el servidor.

La carga de imágenes se puede detener por dos motivos:

-El usuario pulsa el botón de parada del navegador.

-El usuario pulsa en un enlace a otra página antes de que se carguen todas las imágenes.

DESENCADENAMIENTO DE SUCESOS.

Se llama así al proceso mediante el cual se notifica a un programa que se ha producido un suceso.

Cuando se produce un suceso el navegador comprueba si el documento puede responder al suceso, si es así desencadena la respuesta del documento adecuada.

Cuando se produce un suceso un documento o un elemento responde a él si se posee un procedimiento llamado gestor del suceso asociado al suceso y que se ejecuta al producirse este. Los sucesos se desencadenan después de haber sido generados por el navegador. Los gestores de sucesos son procedimientos escritos en un lenguaje de script.

Se pueden generar sucesos para todos los tipos de elementos de un documento y para el propio documento en sí, cada gestor de suceso estará asociado al elemento que recibirá el suceso.

Se puede hacer que el suceso sea gestionado por:

1.- El elemento en que se produce.

2.- El bloque que contiene al elemento en que se produce (<DIV>,<SPAN>,...)

3.- El cuerpo del documento (<body>)

4.- EL documento o etiqueta <HTML>

5.- Otros lugares del objeto documento como controles de un formulario, imágenes, sonidos,...

ENLAZADO DE UN PROCEDIMIENTO A UN SUCESO.

1.- En el propio elemento: <nombre etiqueta tipo suceso=”gestor sucesos()”>

```
<H1 onclick=”proced()” language=”vbscript”>Prueba</H1>
<Script language=”vbscript”>
        Function proced()
                Instrucciones
        End Function
</Script>
```

Puede haber más de un gestor de sucesos enlazados a un elemento, en este caso se escribe uno después de otro separados por espacios:

```
<H1 onclick=”proced()” onmousemove=”proced2()” language=”vbscript”>
  Prueba</H1>
<Script language=”vbscript”>
        Function proced()
                Instrucciones
        End Function
        Sub proced2()
                Instrucciones
        End Sub
</Script>
```

2.- Mediante Script...For:

```
<H1 id=”n1”> Esto es una prueba </H1>
<Script For=”n1” Event=”onclick” language=”vbscript”>
</Script>
```

3.- Mediante procedimientos controlados por sucesos:

```
<H1 id=”n1”> Prueba </H1>
```

```
<script language=”vbscript”>
        Sub n1_Onclick()
                Instrucciones
        End Sub
```

4.- Mediante la función GetRef:

```
Set NombreObjeto.NombreSuceso=GetRef(“nombre proceso”)
<Script language=”vbscript”>
        {Sub|Function} Nombre procedimiento
                Instrucciones
        {End Sub|End Function}
</script>
```

GESTIÓN DE SUCESOS POR DEFECTO. CAMBIO DE COMPORTAMIENTO POR DEFECTO.

Hay etiquetas que poseen mecanismos de gestión de sucesos predefinidos, por ejemplo la etiqueta <A> de los enlaces, se puede redefinir el comportamiento por defecto de estos sucesos, para ellos se harán dos cosas:

1.- Se creará un gestor de sucesos para el suceso que se desencadena.

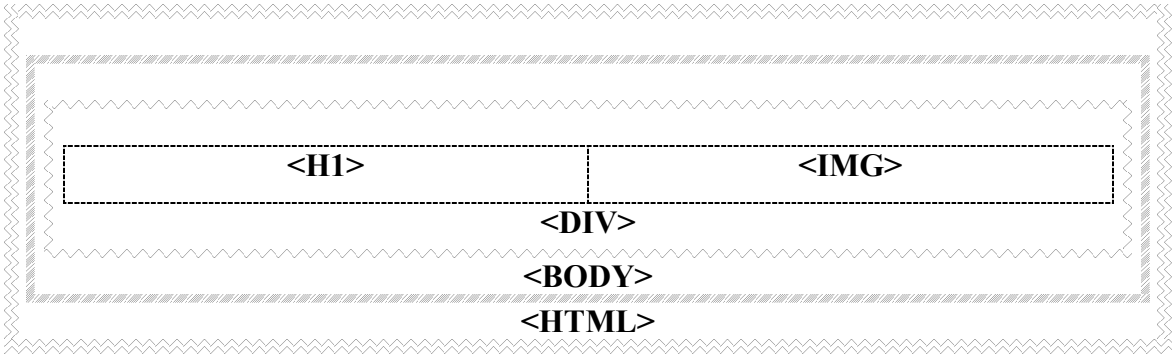
2.- En el gestor del sucesor se escribirá: Window.Event.Return.Value=FALSE, esto anula el gestor de sucesos predefinido.

Ejemplo:

```
<a id=”n1” onclick=”proced1()” href=”...”> Texto </a>
<script language=”vbscript”>
        Function proced1()
                Window.Event.Return.Value=FALSE
                [Instrucciones]
        End Function
</script>
```

PROMODIÓN O SUBIDA DE SUCESOS.

```
<HTML>
        <BODY>
                <DIV>
                        <H1>.....</H1>
                        <IMG SRC=”...”>
                </DIV>
        </BODY>
</HTML>
```



Supongamos que se produce un suceso onclick sobre el elemento H1, para gestionar dicho suceso el navegador lleva a cabo las siguientes acciones:

1.- Comprueba si el elemento H1 posee un gestor de sucesos para el suceso onclick, si es así lo ejecuta.

2.- Comprueba si el elemento padre que contiene a H1 ( ) posee un gestor de sucesos para el suceso onclick, si es así lo ejecuta.

3.- Comprueba si el elemento padre de DIV (<body>) posee un gestor de sucesos para el suceso onclick, si es así lo ejecuta.

4.- Comprueba si el documento (<html>) posee un gestor de sucesos para onclick, si es así lo ejecuta.

-Posibles aplicaciones de la promoción de sucesos:

1.-Supongamos que se quiere ejecutar un procedimiento cuando se produce un suceso en <H1> o en <IMG>, para ahorrar código escribimos si nos interesa el gestor en la etiqueta <DIV>

2.- dado que la promoción de sucesos recorre toda la jerarquía de anidamiento o contención es posible que sean llamados varios gestores de sucesos para un solo suceso, lo cual puede ser un inconveniente, en ese caso es útil saber donde comienza a promocionarse un suceso, eso puede saberse mediante la propiedad:

Window.Event.SrcElement

Esta propiedad almacena el nombre del elemento que detecta en primer lugar el suceso, antes de que este empiece a promocionarse.

Por defecto cuando un suceso se gestiona en cierto nivel seguirá promocionándose hacia arriba por lo que si vuelve a encontrar otro gestor lo ejecuta.

Si se desea detener la promoción de sucesos en un punto de la jerarquía se puede hacer igualando la propiedad Window.Event.CancelBubble a TRUE, para ello se enlaza un gestor de sucesos en el punto de la jerarquía en el que se quiere cortar la promoción y en el se escribe: Window.Event.CancelBubble=True.

Entonces, si se cancela la promoción en un elemento E1, los únicos gestores de sucesos que se ejecutan son los del elemento E1 y los de los elementos contenidos en él, pero no se ejecutan los gestores de los elementos que lo contienen.

ESTILOS DINÁMICOS.

Se llama así a la modificación en tiempo de ejecución de los elementos de estilo característicos de una página Web, por ejemplo modificar el tipo de letra, el tamaño, el color,...., ocultar o visualizar un texto o una imagen,..., cambiar de posición un elemento,...

Ejemplo 1:

```
<H1 id=nn1” Style=”font-family:serif;font-size:16”> Se conmuta entre los tipos de letra “Times New Roman” y “Arial” </H1>
Sub nn1_onmouseover
        nn1.style.color=”#ff20ff”
        nn1.style.fontsize=”11”
        [document.all.]nn1.style.fontfamily=”arial”
End Sub
Sub nn1_onmouseout
        nn1.style.color=”#000000”
        nn1.style.fontsize=”16”
        nn1.style.fontfamily=”Times New Roman”
End Sub
```

Ejemplo 2:

```
<div id=”n2” style=”color:#00ff00;text-decoration:underline;font-style:normal”> Esto cambia al subir y bajar el ratón sobre este texto </Div>
Sub n2_onmouseover
        n2.style.color=”#883366”
        n2.style.textdecoration=”overline”
        n2.style.fontstyle=”Italic”
End Sub
Sub n2_onmouseup
        n2.style.color=”#000000”
        n2.style.textdecoration=”underline”
        n2.style.fontstyle=”normal”
End Sub
```

Ejemplo 3:

```
<Div id=”n3 Style=”color:blue;font-family:arial;font-size:12pt”> Haz click sobre mí y desaparezco, al salir vuelvo a aparecer. </Div>
Sub n3_onclick
        n3.style.color=”#00ff00”
        n3.Style.visibility=”hidden”
End Sub
Sub n3_onmouseout
        n3.style.color=”#00ff00”
        n3.Style.visibility=”visible”
End Sub
```

Ejemplo 4:

```
<H1 id=”m1”> Animales </H1>
<H2 id=”m2” Style=”visibility:hidden;margin-left:20px”> Vertebrados </H2>
<Div id=”m3” Style=”visibility:hidden; margin-left:40px”>
        Aves<br>
```

```
        Reptiles<br>
        Mamíferos<br>
        <Img Src=”Imagen.jpg”>
</Div>
Sub m1_onclick
        If m2.style.visibility=”hidden” THEN
                m2.style.visibility=”visible”
        End IF
End Sub
Sub m1_ondblclick
        If m2.style.visibility=”visible” THEN
                m2.style.visibility=”hidden”
        End IF
End Sub
Sub m2_onclick
        If m3.style.visibility=”hidden” THEN
                m3.style.visibility=”visible”
        End IF
End Sub
Sub m2_ondblclick
        If m3.style.visibility=”visible” THEN
                m3.style.visibility=”hidden”
        End IF
End Sub
```

Ejemplo 4:

```
<Div id=”imajen.jpg” Style=”position:absolute;top:-150px;left:25px”>
        <Img Src=”Imagen.jpg”>
</Div>
Sub image_onclick
Z=imagen.style.top
Z=MID (z,1,LEN(z)-2)
Y=CINT (z)
V=imagen.style.left
x=Cint(u)
If x< 400 and <150 THEN
        x=x+2
        y=y+3
        Imagen.style.top=y
        Imagen.style.left=x
        Window.settimeout”imagen_onclick”,1 ‘ Dibuja las imágenes paso a
                                                paso, no funciona con bucles.
                                               El tiempo de retarde es de 1MS
Elseif x>30and x>200 then
        x=x-1
        y=y-1
        imagen.style.top=y
        Window-settimeout “imagen_onclick”,1
End sub
```

CONTENIDOS DINÁMICOS.

Se llama así a la capacidad de manipular textos, imágenes, objetos y etiquetas de una página Web, después que se ha cargado la página.

Se puede cambiar el contenido mediante scripts que se temporizan para ejecutarse cuando ha transcurrido un cierto tiempo o mediante activadores de sucesos colocados en la etiqueta del usuario.

El DOM de DHTML permite manipular mediante scripts todos los elementos de una página. En concreto el objeto TextRange permite cambiar cualquier etiqueta HTML o su contenido mediante un script.

El contenido dinámico permite tener una información actualizada continuamente y modificar el contenido sin conectar con el servidor.

· Objeto TextRange, una página HTML está formada por etiquetas, scripts y otros elementos definidos y el texto o flujo de texto.

-l flujo de texto es una cadena de caracteres formada por todo el texto que se presenta en la página, sin tener en cuenta las etiquetas que dan formato.

-Ejemplo:

<H1> título destacado </H1>
 Texto en un párrafo
<h2> otro título </h2>

-El objeto textRange permite seleccionar un intervalo de código HTML y manipularlo para cambiarlo.

Los objetos de tipo TextRange se definen mediante el método CreateTextRange del objeto Body contenido en el objeto Document. La sintaxis sería:

Set Variable=Document.Body.CreateTextRange()

En donde variable almacenará el código HTML de la página Web.

· Propiedad HTMLText, contiene una cadena formada por el código correspondiente al intervalo seleccionado.

· Propiedad Text, contiene sólo el texto del intervalo seleccionado.

· Método CreateTextRange, permite seleccionar un intervalo de código.

· Método MoveStart, permite modificar el punto inicial de un intervalo TextRange.

· Método MoveEnd, se utiliza para trasladar el punto final del intervalo TextRange actual.

MODIFICACIÓN DEL DOCUMENTO COMPLETO.

Se selecciona un intervalo que sea igual a todo el documento :Set variable=Document.Body.CreateTextRange, y se sustituye por el texto correspondiente al nuevo documento con: Variable.PasteHTML(“nuevo código html”)

-Ejemplo:

```
<body id=”n1”>
        <h1 id=”n1”> Texto H! inicial </h1>
        <h3 id=”n2”> Borra el contenido de la página </h3>
        <h2 id=”texto H2 inicial </h2>
        <br>
        <h3 id=”p1”> Sólo queremos modificar este texto al hacer clic en él
        </h3>
</body>
Sub nb_onclick
‘Se cambia el contenido de la página al hacer clic en la página.
        Dim it
        Set it=document.body.CreateTextRange()
        msgbox “valor de it.htmltext:” &it.htmltext &vbcrlf &vbcrlf &”valor de_
                it.text:”it.text
        it.pastehtml(“<body id=””nb””>”_
                &”<h1 id=””n1””>Nuevo texto H1 al pulsarlo en la página </h1>”_
                &”<h2> nuevo texto h2 al pulsar en la página </h2>”_
                &”</body>”)
End Sub
```

BORRAR EL CONTENIDO DE UNA PÁGINA.

```
Sub n2_onclick
‘borra el contenido de la página al hacer clic en H3
        Dim it
        Set it=document.body.createdtextrange()
        It.pastehtml(“”)
End Sub
```

OTROS EJEMPLOS.

```
Sub n3_onclick
‘se cambia el contenido de la página al hacer click en H2
        Dim it
        Set it=document.body.createdtextrange()
        It.pastehtml(“<h1 id=””n1””> Nuevo Contenido </h1>”_
                        &”<img id=””n4”” src=””imagen.jpg””>”)
End Sub
```

MODIFICACIÓN PARCIAL DEL TEXTO DE UNA PÁGINA.

```
Sub p1_onclick
‘Sólo se modifica el contenido del texto de DIV
        Dim it
        Set it=document.body.createtextrange()
        It.movestart “Word”,17  ‘ Se cambia el origen en que comienza el intervalo
                                (Palabra 17)
        it.text=”texto modificado”
        window.event.cancelbubble=True
End Sub
```

ANIMACIONES.

```
<img id=”ti” style=”position:absolute;top:20;left:100;width:214px;height:214;
src=”imagen1”>
<img id=”ti” style=”position:absolute;top:20;left:400;width:340px;height:340;
src=”imagen2”>
<img id=”ti” style=”position:absolute;top:20;left:400;width0px;height:0;
src=”imagen3”>
<img id=”ti” style=”position:absolute;top:20;left:400;width0px;height:0;
src=”imagen4”>

Sub t1_onclick()
        dim mx
        mx=t1.style.pixewidth ‘Almacena el ancho de la imagen en píxeles
        if mx <112 THEN
                escalaB
        elseif mx>=112 then
                escalaA
End Sub
Sub escalaA
        IF t1.style.pixelwidth>=0 then
                T1.style.pixelwidth=t1.pixelwidth -1
                t1.style.pixelheight=t1.style.pixelheight -1
                IF t1.style.pixewidth >0 then
                        Window.settimeout “escalaA”,1
                ElseIf t1.style.pixelwidth=0 then
                        Window.settime “escalaB”,1
                EndIF
        End IF
End Sub
Sub escalaB
        IF t1.style.pixelwidth<=214 then
                T1.style.pixelwidth=t1.pixelwidth +1
                t1.style.pixelheight=t1.style.pixelheight +1
                IF t1.style.pixewidth <214 then
                        Window.settimeout “escalaB”,1
                ElseIf t1.style.pixelwidth=0 then
                        Window.settime “escalaA”,1
                EndIF
        End IF
End Sub

                        _____

Sub t2_onclick()
        Dim mx
        Mx=t2.style.pixel.width
        If mx<170 then
                escalaC
        Elseif mx>=170 then
                escalaD
        End If
End Sub
Sub escalaC
        IF t2.style.pixelwidth <=340 then
                t2.style.pixelheight=t2.style.pixelheight+2
                t2.style.pixelweight=t2.style.pixelwidth+2
```

```
                IF t2.style.pixelwidth <340 then
                        Window.settimeout “escalaC”,1
                Elseif t2.style.pixelwidth >= 340 then
                        Window.settimeout “escalad”,1
                End IF
        End IF
End Sub
Sub escalaD
        IF t2.style.pixelwidth >=0 then
                t2.style.pixelheight=t2.style.pixelheight-2
                t2.style.pixelweight=t2.style.pixelwidth-2
                IF t2.style.pixelwidth >0 then
                        Window.settimeout “escalaD”,1
                Elseif t2.style.pixelwidth <=0 then
                        T3.style.pixelwidth=0
                        t3.style.pixelheight=0
                        Window.settimeout “escalaE”,1
                End IF
        End IF
End Sub
Sub escalaE
        IF t3.style.pixelwidth <=340 then
                t3.style.pixelheight=t3.style.pixelheight+2
                t3.style.pixelweight=t3.style.pixelwidth+2
                IF t3.style.pixelwidth <340 then
                        Window.settimeout “escalaE”,1
                Elseif t3.style.pixelwidth >= 340 then
                        Window.settimeout “escalaF”,1
                End IF
        End IF
End Sub
Sub escalaF
        IF t3.style.pixelwidth >=0 then
                t3.style.pixelwidth=t3.style.pixelwidth-2
                t3.style.pixelheight=t3.style.pixelheight-2
                IF t3.style.pixelwidth >0 then
                        Window.settimeout “escalaF”,1
                Elseif t3.style.pixelwidth = 0 then
                        T4.style.pixelwidth=0
                        t4.style.pixelheight=0
                        Window.settimeout “escalaG”,1
                End IF
        End IF
End Sub
Sub escalaG
        IF t4.style.pixelwidth <=340 then
                t4.style.pixelheight=t4.style.pixelheight+2
                t4.style.pixelweight=t4.style.pixelwidth+2
                IF t4.style.pixelwidth <340 then
                        Window.settimeout “escalag”,1
                Elseif t3.style.pixelwidth >= 340 then
                        Window.settimeout “escalaH”,1
                End IF
        End IF
End Sub
```

```
Sub escalaH
        IF t4.style.pixelwidth >=0 then
                t4.style.pixelwidth=t4.style.pixelwidth-2
                t4.style.pixelheight=t4.style.pixelheight-2
                IF t4.style.pixelwidth >0 then
                        Window.settimeout “escalaH”,1
                Elseif t3.style.pixelwidth = 0 then
                        Window.settimeout “escalaC”,1
                End IF
        End IF
End Sub
```