1.-Tipos y subtipos de datos en VBS

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.

Subtipos de Variables y breve descripción:

- Empty: la variant no está inicializado. El valor es 0 para variables numéricas o una cadena de longitud cero para variables de cadena.
- Null: la variant, de manera intencionada no contiene ningún dato válido.
- Boolean: contiene trae o false.
- Byte: Contiene un número entero entre 0 y 255.
- Integer: contiene un número entero entre -32.768 y 32.767
- Currency: de -922.337.203.685.477,5808 a 922.337.203.685.477,5807
- Long: contiene un número entero entre -2.147.483.648 y 2.147.483.647.
- Single: Contiene un número de coma flotante de precisión simple comprendido entre -3,402823E38 y -1,401298E-45 para valores negativos y entre 1,401298E-45 y 3,402823E38 para valores positivos.
- Double: contiene un número de coma flotante de precisión doble comprendido entre -1,79769313486232E308 y -4,94065645841247E-324 para valores negativos y entre 4,94065645841247E-324 y 1,79769313486232E308 para valores positivos
- Date (time): contiene un número que representa una fecha entre el uno de enero del año 100 y el 31 de diciembre del año 9999.
- String: contiene una cadena de longitud variable que puede ser de hasta aproximadamente 2 mil millones de caracteres.
- Objet: contiene un objeto.
- Error: contiene un error.

2.-Definición, declaración y tipos de 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.

3.- Bloque selectivo con condiciones múltiples, sintaxis, explicación de su funcionamiento.

```
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.

4.- bucle FOR-NEXT. Sintaxis y explicación:

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.

5.- Procedimientos FUNCTION. Sintaxis, paso de argumentos, valores devueltos.

[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.

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.

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

Nombre Procedimiento (arg1, arg2, arg3....)

6.-Escribir dos procedimientos que permitan calcular el factorial de un número, uno recursivo y el otro no.

Sin recursividad:

```
FUNCTION factorial (n)
 Dim ini,ct
 Ini=1
 FOR ct=1 to n
  Ini=ini*ct
 Next
 Factorial=ini
END FUNCTION
```

Con recursividad:

```
FUNCTION factor (n)
 IF n=1 THEN
  Factor=1
 ELSE
  Factor=n*factor (n-1)
 END IF
FUNCTION
```

7.-Construir un procedimiento que al pasarle una cadena de más de ocho caracteres permita obtener en mayúsculas del tercer al octavo carácter y el código ANSI del tercer carácter.

```
<script language="vbscript">

Sub Cadena
Dim x, a, b, c, d
do until Len(X)>8
        x= inputbox ("Introduzca una cadena de más de ocho caracteres")
loop

        a=MID(x,3,6)
        b=Ucase(a)
        C=MID(x,3,1)
        d=ASC(C)
        msgbox b &vbcrlf &d
End sub
</Script>
```

8.- Construir un procedimiento tal que permita introducir datos arbitrariamente al usuario pero sólo admita aquellas entradas cuyos caracteres sean números.

```
<script language="vbscript">
```

```
x=inputbox ("introduzca una cadena")
Dim txt, ct
        FOR ct=1 to LEN(x)
                txt=MID(x,ct,1)
                IF ASC(txt)<47 then
                        Msgbox"El título no es correcto"
                        EXIT FOR
                ELSEIF ASC(txt)>58 THEN
                        Msgbox"El título no es correcto"
                        EXIT FOR
                END IF
        NEXT
</script>
```

9.- Tenemos una clase llamada TRIRECT para representar a los triángulos rectángulos, tiene como atributos la longitud de los dos catetos C1y C2. Establecer el código para definir una propiedad de lectura y escritura llamada base para el cateto C1 y un método llamado ARETRI para obtener el área del triángulo.

```
Class TRIRECT
                'Definición de los atributos de la clase
        PRIVATE c1, c2
                'Propiedad de lectura y escritura: base
        PUBLIC DEFAULT property GET base
                Base=C1
        END property
        PUBLIC property LET base (v)
                ….
                Código HTML
                ….
        END property
                'Se define él método para calcular el área del triángulo.
        PUBLIC function aretri
                Aretri= (c1*c2)/2
        END function
END class
```

10.- Si para el otro cateto tenemos una propiedad llamada altura, ¿Qué código se escribirá para obtener el área de un triángulo cuyos catetos valen 2 y 3?, ¿Cómo estableceríamos base=8?

```
                'Creamos una instancia de la clase TRIRECT
        SET tri1=NEW TRIRECT
                'Le damos valores a la altura y ala base
        tri1.base=2
        tri1.altura=3
                'Calculamos el área
        Msgbox "el área del triángulo es:"aretri
                'Establecemos base=8
```

tri1.base=8

11.- ¿Cómo se activa la gestión de errores? ¿Cómo se desactiva? ¿Qué nos da ERR? y ¿ERR.SOURCE? y ¿ERR.CLEAR?

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.

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 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.
```

· **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

12.- Usando la etiqueta SCRIPT escribir la sintaxis de los métodos para gestionar sucesos.

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

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

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

13.-Sintaxis de otros dos métodos para gestionar sucesos.

```
SUB bt1_onclick ()
        Msgbox “pepe”
ENDSUB

SUB bt2_ondblclick ()
        Msgbox “hola que tal”
END SUB.
```

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

14.- ¿Para que sirve WINDOW.EVENT.RETURNVALUE=FALSE? Y ¿WINDOW.EVENT.CANCELBUBBLE=TRUE?

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>
```

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.

-oOo-