

## QL

Palabras  
clave en  
el QL

La Guía de Referencia de Palabras Clave lista todas las palabras clave del lenguaje SuperBasic por orden alfabético. También se da una breve explicación de la función de cada palabra clave seguida por una definición de la sintaxis y ejemplos de utilización. La explicación de la definición sintáctica se da en la Guía de Referencia de *Conceptos*, bajo el encabezamiento *sintaxis*.

Cada "entrada" en este índice de palabras clave indica con qué grupo de operaciones (si lo hay) está relacionada dicha palabra clave; por ejemplo DRAW es una *operación gráfica*, y se puede obtener más información al respecto en la sección de gráficos de la *Guía de Referencia de Conceptos*.

En algunos casos necesitamos tratar con más de una palabra clave a un tiempo, ejemplo IF, ELSE, THEN, END, IF. Todas ellas se listan bajo el encabezamiento IF.

Se incluye un índice que intenta cubrir todas las formas posibles en las que se puede describir una palabra clave de SuperBASIC. Por ejemplo, el comando que deja limpia (vacía) la pantalla, CLS se lista también bajo el encabezamiento *clear screen* y *screen clear*.



# ABS

funciones  
matemáticas

ABS devuelve el valor absoluto del parámetro. Si el parámetro es positivo, ABS devolverá el parámetro, y si es negativo, devolverá cero menos el parámetro.

sintaxis:     **ABS** (*expresión\_\_numérica*)

*ejemplo..*    *i.*    PRINT ABS(0.5)  
              *ii.*   PRINT ABS(a-b)

# ADATE

reloj

DATE permite ajustar el reloj

sintaxis:     *segundos., = expresión\_numérica*  
              ADATE *segundos*

*ejemplo:*    *i.*    ADATE 3600 {avanzará el reloj una hora}  
              *ii.*   ADATE -60 {atrasará el reloj un minuto}



# AT

ventanas

Permite modificar la posición de impresión sobre una rejilla imaginaria de filas/columnas basándose en el tamaño que tienen en ese momento los caracteres. AT utiliza una forma modificada de sistema de coordenadas de pixels, donde la fila 0, columna 0 se encuentra en el ángulo superior izquierdo de la ventana. AT afecta a la posición de impresión de la ventana relacionada con el canal especificado o con el canal de omisión.

sintaxis:    *línea:*            = *expresión numérica*  
              *columna.,*        = *expresión-numérica*

AT [*canal,* ] *línea, columna*

ejemplo:    **AT 10,20: PRINT "Esto está en la línea 10 columna 20"**

# ACOS, ASIN ATAN, ACOT

funciones  
matemáticas

ACOS y ASIN calculan el arco coseno y el arco seno respectivamente. ATAN calcula el arco tangente y ACOT el arco cotangente. No existe límite efectivo para el tamaño del parámetro.

sintaxis:    *ángulo;* = *expresiónnumérica* [en radianes]

ACOS(*ángulo*)    ASIN(*ángulo*)

ATAN(*ángulo*)    ACOT(*ángulo*)

ejemplo:    i.    **PRINT ATAN(ángulo)**  
              ii.   **PRINT ASIN(1)**  
              iii. **PRINT ACOT(3.6574)**  
              iv. **PRINT ATAN(a-b)**



# BEEP

## sonido

BEEP activa las funciones de sonido conformadas en el interior del QL. BEEP puede aceptar un número variable de parámetros que dan muchos niveles de control sobre el sonido producido. Las especificaciones mínimas necesitan únicamente indicar una duración y una frecuencia. Si se utiliza BEEP sin ningún parámetro no aparecerá sonido alguno.

sintaxis: *duración:* = *expresión\_\_numérica* {gama de -32768 ..32767}  
*frecuencia;* = *expresión\_\_numérica* {gama de 0.. ..255}  
*envolvente:* = *expresión\_\_numérica* {gama 0..15}  
*grad\_x:* = *expresión\_\_numérica* {gama de 0...32768.....32767}  
*grad\_y.* = *expresiónnumérica* {gama de 8....7}  
*emborronamiento:* = *expresión-numérica* {gama de 0..15}  
*c. aleatoria:* = *expresiónnumérica* {gama de 0... 15}

BEEP [*duración, frecuencia*  
[, *frecuencia\_\_2, grad\_x, grad\_y*  
[, *envolvente*  
[, *emborronamiento*  
[, *aleatorio*]]]]]

donde

*duración* especifica la duración del sonido en unidades de 72 microsegundos. Una duración de 0 continuará el sonido hasta finalizarlo con otro comando BEEP

*frecuencia* especifica la frecuencia del sonido (tono). Una frecuencia de 1 es alta y una frecuencia de 255 es baja

*frecuencia\_\_2* especifica un nivel de frecuencia superior sobre el cual oscila el sonido.

*grad\_x* especifica la velocidad con la que el sonido *grad* y oscila entre los dos frecuencias especificadas.

*envolventes* fuerza al sonido a oscilar sobre sí mismo el número de veces especificado. Si wrap es igual a 15, el sonido oscilará de un modo indefinido

*emborronamiento* define la cantidad que debe añadirse al sonido

*componente aleatoria* define la componente aleatoria que debe añadirse al sonido

# BEEPING

sonido BEEPING es una función que devuelve un cero (falso) si en ese momento el QL no está emitiendo un beep (sonido de sirena) y devuelve un número diferente de cero (verdad) si está emitiendo un beep.

sintaxis: BEEPING

ejemplo: 

```
100 DEFinePROCedure tranquilo
110     BEEP
120 END DEFine
130 IF BEEPING THEN tranquilo
```

# BLOCK

ventanas BLOCK rellena un bloque del tamaño y forma especificados en una posición especificada relativa al origen de la ventana unida al canal especificado o de omisión.

BLOCK utiliza el sistema de coordenadas de pixels

sintaxis: *ancho*, = *expresión numérica*  
*alto*: = *expresión-numérica*  
*x*: = *expresión-numérica*  
*y*: = *expresión-numérica*

ejemplo: i. **BLOCK** 1 0,10,5,5,7 {bloque blanco del Ox 10 pixels, situado en 5,5}

ii. 

```
100 REMark diagrama de barras
110 CSIZE 3,1
120 PRINT "Diagrama de barras"
130 LET fondo = 100 : ancho = : izda = 10
140 FOR barra = 1 TO 10
150 LET color = RND(0 TO 255)
160 LET alto = RND(2 TO 20)
170 BLOCK ancho, alto, izda + barra * ancho, fondo - alto, 0
180 BLOC ancho - 2, alto - 2, izda + barra *
    ancho + 1, fondo - alto + 1, color
190 END FOR barra
```

{utilice LET color = RND(0 TO 7) en las televisiones}



# BORDER

ventanas

BORDER añade un borde a la *ventana* relacionada con el canal especificado, o el canal de omisión.

En todas las operaciones que se realicen a continuación en dicha ventana, excepto BORDER, el tamaño de la ventana se reduce para dejar espacio al reborde. Si se utiliza otro comando BORDER, se restaurará el tamaño original de la ventana anterior (antes de ser añadido el borde). Así pues, varios comandos BORDER tienen el efecto de cambiar el tamaño y color de un borde único. Los bordes múltiples sólo se crean mediante acciones específicas.

Si se utiliza el comando BORDER sin especificar ningún color, se creará un borde de la anchura especificada y color transparente.

sintaxis:     *anchura*.. = *expresión\_\_numérica*  
              BORDER [*cana*/,] *tamaño*[,*color*]

ejemplo:     i.    BORDER 10,0,7                {borde con patrón blancos y negros}  
              ii. 10 REMark Bordes extravagantes  
                  20 FOR espesor = 50 to 2 STEP -2  
                  30    BORDER espesor, RND(0 TO 255)  
                  40 END FOR espesor  
                  50 BORDER 50  
                  {utilice RND (0 TO 7) para televisiones}

# CALL

Qdos

Desde el SuperBASIC se puede acceder directamente al código máquina utilizando el comando CALL-llamada. CALL puede aceptar hasta 13 parámetros que pueden almacenar en el 68008 datos y direcciones de registros en secuencia (D1 a D7, Ao a A5).

CALL no devuelve datos

sintaxis:     *dirección*: = *expresión numérica*  
              *datos*.    = *expresión-numérica*  
              CALL *dirección*, \*[*datos*]\* { 13 parámetros máximo}

ejemplo:     i.    CALL 262144,0,0,0  
              ii.  CALL 262500,12,3,4,1212,6

La dirección del registro A6 no debe utilizarse en rutinas llamadas con este comando. Para volver al SuperBASIC, utilice las instrucciones siguientes:

```
MOVEQ #0,D0
RTS
```

advertencia

# CHR\$

BASIC CHR\$ es una función que devuelve el carácter cuyo valor se ha especificado en forma de parámetro.

CHR\$ es la función inversa de CODE.

sintaxis: CHR\$ (expresión\_\_numérica)

ejemplo: i. PRINT CHR\$(27) {impresión del carácter ASCII escape}  
ii. PRINT CHR\$(65) {imprime una A}

# CIRCLE CIRCLE\_R

gráficos

CIRCLE dibuja un círculo (o una elipse en un ángulo especificado) en la pantalla, y con una posición y tamaño específicos. El comando CIRCLE utiliza el sistema gráfico de coordenadas. El círculo se trazará en la ventana unida al canal especificado o al de omisión.

El comando CIRCLE utiliza el sistema gráfico de coordenadas y puede utilizar coordenadas absolutas (relativas al origen *del gráfico*) y relativas (referidas al *cursor gráfico*). Si se van a utilizar coordenadas relativas debe usarse CIRCLE\_R.

Se pueden trazar círculos o elipses múltiples con una única llamada al comando CIRCLE. Cada conjunto de parámetros debe ir separado de los otros por puntos y comas, (;).

Si es necesario, puede sustituirse la palabra clave ELLIPSE por la palabra clave CIRCLE.

sintaxis:  $x :=$  *expresión\_numérica*  
 $y :=$  *expresión\_numérica*  
 $radio :=$  *expresión numérica*  
 $excentricidad.. =$  *expresión-numérica*  
 $ángulo.. =$  *expresión\_numérica* {gama de 0 =...2π}  
 $parámetros :=$  | x, y  
| *radio, excentricidad, ángulo 2*

donde 1 dibujará un círculo

2 dibujará una elipse de un ángulo y excentricidad especificadas

CIRCLE [*cana*/, ] *parámetros* \*[: *parámetros*]\*

x desplazamiento horizontal desde el origen gráfico o desde el cursor gráfico

y desplazamiento vertical desde el origen gráfico o desde el cursor gráfico

*radio* radio del círculo

*excentricidad* relación entre el eje mayor y el eje menor de una elipse

*ángulo* orientación del eje mayor de la elipse respecto de la vertical. Este ángulo debe especificarse en radianes.

ejemplo: i. CIRCLE 50,50,20 {círculo de radio 20 y situado en 50,50}  
ii. CIRCLE 50,50,20,0.5,0 {elipse de eje mayor 20, excentricidad 0.5, situada en 50,50 y alineada con el eje vertical}

# CLEAR BASIC

El comando CLEAR limpia el área de variables de SuperBASIC que está dispuesta para el programa en curso, y libera el espacio para el Qdos.

sintaxis: CLEAR

ejemplo: CLEAR

CLEAR puede utilizarse para restaurar el Sistema SuperBASIC en un estado conocido. Por ejemplo, si un programa se interrumpe (o se detiene debido a un error) mientras se encontraba en un procedimiento, el SuperBASIC continuará en el procedimiento incluso después de la detención del programa. CLEAR volverá a inicializar el SuperBASIC.

comentario

# CLOSE

dispositivos

El comando CLOSE cierra el canal especificado. Cualquier ventana asociada a él será desactivada.

sintaxis: *canal*: = #*expresión numérica*  
CLOSE *c a n a l*

ejemplo: i. CLOSE #4  
ii. CLOSE #entrada-canal

# CLS

## ventana

Limpiar (vaciará) la ventana unida al canal especificado o al canal de omisión con color de fondo —PAPER— vigente en ese momento, excluyendo el borde, si se especificó. CLS acepta un parámetro opcional que especifica sólo si debe limpiarse una parte de la ventana.

sintaxis: *parte* = *expresión\_\_numérica*

CLS [*canal*,] [*parte*]

donde *parte* = 0 pantalla completa (es la de omisión si no se especifica parámetro alguno)

*parte* = 1 parte superior excluyendo la línea del cursor

*parte* = 2 parte inferior excluyendo la línea del cursor

*parte* = 3 la línea del cursor completa

*parte* = 4 el final de la línea del cursor incluyendo la posición de éste

- ejemplo:
- i. CLS {la ventana completa}
  - ii. CLS 3 {vacía la línea del cursor}
  - iii. CLS #2,2 {limpia el final de la ventana del canal 2}

# CODE

CODE es una función que devuelve el código interno que se utiliza para representar un determinado carácter. Si se especifica una cadena, CODE devuelve la representación interna del primer carácter de la cadena.

CODE es la función inversa de CHR\$

sintaxis: CODE (*expresión\_\_de\_\_cadena*)

- ejemplo:
- i. PRINT CODE("A") {imprime 65}
  - ii. PRINT CODE("superBASIC") {imprime 83}

# CONTINUE RETRY

tratamiento de los errores

El comando CONTINUE permite continuar un programa que se ha interrumpido. El comando RETRY permite que se vuelva a ejecutar una instrucción que dio error.

sintaxis:     CONTINUE  
              RETRY

ejemplo:     **CONTINUE**  
              **RETRY**

Un programa sólo puede continuar si

1. No se han añadido líneas nuevas al programa
2. No se han añadido nuevas variables al programa
3. No se ha cambiado ninguna línea

Puede establecerse o cambiarse el valor de las variables.

precaución

# COPY COPY\_N

dispositivos

El comando COPY copiará un archivo desde un dispositivo de entrada a un dispositivo de salida hasta que detecte un indicador de fin de archivo. El comando COPY\_N borrará cualquier cabecera asociada con un archivo de Microdrive y permitirá que los archivos de Microdrive se copien correctamente en otro tipo de dispositivo.

Las cabeceras están asociadas con dispositivos tipo directorio, y deben eliminarse mediante COPY\_N cuando se copie en dispositivos que no sean tipo directorio. (Por ejemplo, mdvl es un dispositivo tipo directorio; ser1 es un dispositivo que no es tipo directorio.

sintaxis:     COPY *dispositivo* TO *dispositivo*  
              COPY\_N *dispositivo* TO *dispositivo*

Debe poderse producir la entrada desde el dispositivo fuente y debe estar disponible para salida el dispositivo de destino.

ejemplo:     i.    **COPY mdvl\_arch\_datos TO con\_**     {copia en la ventana de omisión}

              ii.   **COPY neti\_3 TO mdv1\_\_data**     {copia los datos de la estación de la red local al mdv\_data}

              iii . **COPY\_N     mdvl\_ltest\_data     TO     ser1{ c o p i a mdvl\_test\_data en la puerta en serie 1 eliminando la información de la cabecera}**

# COS

funciones  
matemáticas

COS calcula el coseno del argumento especificado.

sintaxis: *ángulo*. = *expresión\_\_numérica* {gama de -60000... 60000 en  
radianes)

COS (*ángulo*)

ejemplo: i. PRINT COS(theta)  
ii. PRINT COS(3.141592654/2)

# COT

funciones  
matemáticas

COT calculará la cotangente del argumento especificado

sintaxis: *ángulo*: = *expresión\_\_numérica* (gama desde -30000... 30000  
en radianes)

COT(*ángulo*)

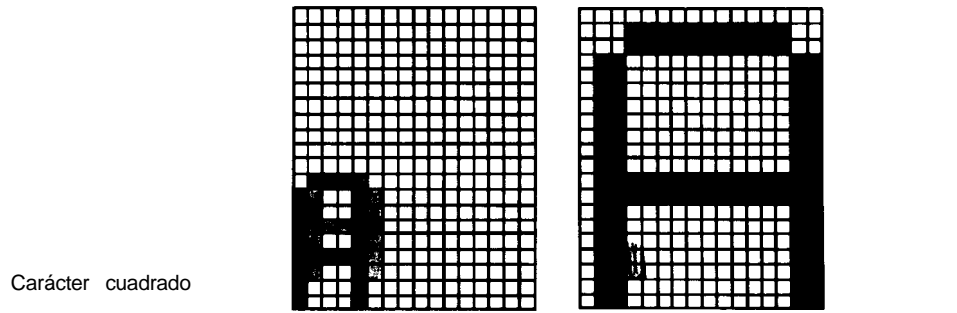
ejemplo: i. PRINT COT(3)  
ii. PRINT COT(3.141592654/2)

# Csize

ventanas

Establece un nuevo tamaño para los caracteres de una ventana asociada a un *cana/* especificado o de omisión. El tamaño estándar es 0,0 en el modo 512, y 2,0 en el *modo 256*.

La anchura define el tamaño del espacio de carácter en horizontal, y la altura define el tamaño vertical del espacio de carácter. El tamaño de carácter se ajusta para llenar el espacio dinible.



ancho	tamaño	altura	tamaño
0	6 pixels	0	10 pixels
1	8 pixels	1	20 pixels
2	12 pixels		
3	16 pixels		

sintaxis: *ancho.. = expresión numérica* {gama desde el 0 al 3}  
*alto: = expresión numérica* {gama de 0 a 1}  
 Csize [*cana/*, ] *ancho*, *alto*

ejemplo: i. Csize 3,0  
 ii. Csize 3,1

# Cursor

ventana

EL comando CURSOR permite colocar el cursor de pantalla en cualquier lugar de la ventana asociada al *canal* especificado o de omisión.

El comando CURSOR utiliza el *sistema de coordenadas de pixels* relativas al origen de la ventana y define la posición del ángulo superior izquierdo del cursor. El tamaño del cursor depende del tamaño de los caracteres que esté vigente en ese momento.

Si se utiliza el comando CURSOR con cuatro parámetros, los dos primeros se interpretan como coordenadas gráficas (que utilizan el sistema de gráfico de coordenadas) y el segundo par, se toma como posición del cursor con respecto al primer punto (en el sistema de coordenadas del pixels).

Con ello, los diagramas resultan relativamente fáciles de escribir.

sintaxis: *x: = expresión numérica*  
*y: = expresión numérica*  
 CURSOR [*canal*, ] *x,y* [, *x,y*]

ejemplo: i. CURSOR 0,0  
 ii. CURSOR 20,30  
 iii. CURSOR 50,50,10,10

# DATA READ RESTORE

## BASIC

READ, DATA y RESTORE son tres comandos que permiten que los datos contenidos en un programa de SuperBASIC se asignen a unas variables en el momento de la ejecución del programa.

DATA se utiliza para marcar y definir los datos. El comando READ accede a los datos y los asigna a las variables, y RESTORE permite la selección de ciertos datos específicos

**DATA:** permite definir los datos en el interior de un programa. Estos datos pueden leerse mediante una instrucción READ colocada a continuación y asignados a las variables. El SuperBASIC ignora las instrucciones DATA al encontrárselas durante el proceso normal.

sintaxis: DATA \*[expresión, ]\*

**READ** lee los datos contenidos en una instrucción DATA y los asigna a una lista de variables. Inicialmente el indicador se fija en la **primera instrucción** DATA del programa, y se va incrementando después de cada nueva lectura READ. Si se vuelve a ejecutar el programa, el indicador de datos no se vuelve a inicializar de forma que, en general, los programas deben incluir RESTORE explícitamente.

Si se intenta ejecutar un comando READ sin la correspondiente instrucción DATA , se generará un error apareciendo el mensaje consiguiente.

sintaxis: READ \*[identificador,]\*

**RESTORE** restaura el indicador de los datos, es decir la posición desde la cual los comandos subsiguientes READ leerán los datos. Si RESTORE va seguido de un parámetro, el indicador se establecerá en ese valor. Si no se especifica ningún parámetro el indicador de los datos se vuelve a establecer al comienzo del programa.

sintaxis: RESTORE [línea\_\_número]

ejemplo:    i.    **100 REMark Ejemplo de instrucción DATA**  
              **110 DIMension días\$(7,4)**  
              **120 RESTORe**  
              **130 FOR cuenta = 1 TO 7:**  
                  **READ días\$(cuenta)**  
              **140 PRINT días\$**  
              **150 DATA "LUN", "MAR", "MIER", "JUE", "VIER"**  
              **160 DATA "SAB", "DOM"**

              ii.    **100 DIM mes\$(12,10)**  
                  **110 RESTORE**  
                  **120 REMark Ejemplo de instrucción DATA**  
                  **130 FOR cuenta = 1 TO 12 :**  
                      **READ mes\$ (cuenta)**  
                  **140 PRINT mes\$**  
                  **150 DATA "Enero", "Febrero", "Marzo"**  
                  **160 DATA "Abril", "Mayo", "Junio"**  
                  **170 DATA "Julio", "Agosto", "Septiembre"**  
                  **180 DATA "Octubre", "Noviembre", "Diciembre"**





# DAY\$

reloj

DAY!\$ es una función que devuelve el día vigente de la semana. Si se especifica algún parámetro lo interpreta como una fecha y por tanto devolverá el día de la semana correspondiente a esa fecha.

sintaxis:     **DAY\$** {toma el día del reloj}  
              **DAY\$** {toma el día del parámetro}

ejemplo:     i. PRINT DAY\$                 {da como salida el día  
              ii. PRINT DAY\$(234567)     {da como salida el día  
  que representa 234567 en segundos}

# DEG

funciones  
matemáticas

DEG es una función que convierte un ángulo expresado en radianes en otro expresado en grados.

sintaxis:     DEG (expresión\_\_numérica)

ejemplo:     PRINT DEG (PI/2) {imprimirá 90}

# DEFine FuNction END DEFine

funciones y  
procedimientos

**DEFine FuNction** define una función del lenguaje SuperBASIC. La función está constituida por la secuencia de instrucciones que se encuentran entre la función **DEFine** y **END DEFine**. La definición de la función puede también incluir una lista de *parámetros formales*, que suministran los datos a la función. Tanto los *parámetros formales* como los *actuales* deben ir entre paréntesis. Si la función no necesita *parámetros*, no es necesario especificar los paréntesis vacíos.

Los *parámetros formales* toman su tipo y características de los correspondientes *parámetros actuales*. El tipo de los datos devueltos es igual al tipo de la expresión dada.

Si se añade a la expresión una instrucción **RETurn** ésta devolverá una respuesta. El tipo de los datos viene indicado por el carácter unido al nombre de la función. Un signo de dólar, \$, indica unos datos de cadena, un signo de porcentaje indica datos enteros, y si no aparece ningún carácter unido al nombre, indicará que los datos llevan coma flotante.

Las funciones se activan incluyendo su nombre en el punto adecuado de la expresión en SuperBASIC.

Las llamadas a funciones en SuperBASIC pueden ser recursivas, es decir, una función puede llamarse así misma directa o indirectamente a través de una secuencia de otras llamadas.

sintaxis: *parámetros formales*: = (expresión \*[, expresión]\*)  
*parámetros\_actuales*: = (expresión \*[, expresión]\*)

tipo: = | \$ % |

```
DEF FuNction identificador tipo [parámetros __formales]  
  [LOCAL identificador*[,identificador]*]  
  instrucciones  
  RETurn expresion  
END DEFine
```

**RETurn** puede encontrarse en cualquier posición dentro del procedimiento. Las instrucciones **LOCAL** (locales) deben preceder a la primera instrucción ejecutable de la función.

ejemplo: 10DEFine FuNction media (a,b,c)  
20 LOCAL respuesta  
30 LET respuesta = (a + b + c) / 3  
40 RETURN respuesta  
50 END DEFine  
60 PRINT media (1,2,3)

Para mejorar la lectura de los programas se puede añadir el nombre de la función a la instrucción **END DEFine**. Este nombre, sin embargo, no será comprobado por el SuperBASIC.

comentario

# TIPOS DE DATOS VARIABLES

## enteros

Enteros son números de la gama de -32767 a +32768. Las variables se suponen enteras cuando el identificador de las variables lleva un signo de porcentaje % como sufijo. En el lenguaje SuperBASIC no existen constantes enteras, todas las constantes se almacenan como números de coma flotante.

sintaxis: *identificador%*

ejemplo: i. **contador%**  
ii. **tamaño\_límite %**  
iii. **esta\_variable\_entera%**

## coma flotante

LOS números con coma flotante de la gama de  $\pm(10^{-615}$  to  $10^{+615}$ ) con 8 dígitos significativos. En el lenguaje SuperBASIC el tipo de datos de omisión es la coma flotante. Todas las constantes se guardan en esta forma y se pueden introducir utilizando notación exponencial.

sintaxis: *identificador | constante*

ejemplo: i. **ventas-diarias**  
ii. **76.2356**  
iii. **354E1 5**

## cadena

Una cadena es una secuencia de caracteres válidos en SuperBASIC cuya longitud puede llegar hasta 32.768 caracteres. Las *variables* se suponen tipo cadena si su nombre lleva el signo del dólar \$ como sufijo. Las cadenas de datos se representan encerrando la adecuada selección de caracteres de SuperBASIC entre comillas simples o dobles.

sintaxis: *identificador\$*

ejemplo: i. **variables\_de\_cadena\$**  
ii. **«esto es una cadena de datos»**

## nombre

El tipo nombre tiene la misma forma que un *identificador* estándar del SuperBASIC y se utilizan por el sistema para nombrar los *archivos* de *Microdrive*, etcétera.

sintaxis: *identificador*

ejemplo: i. **mdvl\_arch\_datos**  
ii. **serle**

Para mejorar la lectura de los programas, se puede añadir el nombre del procedimiento al final de la instrucción END DEFine. Sin embargo, el lenguaje SuperBASIC no comprueba este nombre.

comentario

## DELETE

Microdrives

El comando DELETE elimina un archivo del directorio del cartucho en el Microdrive especificado.

sintaxis:     **DELETE** *dispositivo*

En las especificaciones del dispositivo, debe consignarse que éste es un Microdrive.

ejemplo:     i.   **DELETE** mdv1\_\_datos\_\_antiguos  
              ii. **DELETE** mdv1\_\_archivo\_\_cartas

# DIM

## matrices

Define una matriz en SuperBASIC. Pueden definirse matrices de *cadena*, *enteras* y de *coma flotante*. Las matrices de cadena manejan cadenas de una longitud establecida, y la última instrucción DIM se toma como longitud de la cadena.

Los índices de las matrices pueden ir desde 0 al máximo índice especificado en la instrucción *dimension*. Así pues, DIM genera una matriz con un elemento más en cada dimensión que el que está especificado. Cuando se especifica una matriz se inicializa en cero (para las matrices numéricas) y en longitud cero para las matrices de cadena.

sintaxis: *index*: = *expresión numérica*  
*matriz*.. = *identifica*& (*index* \*[, *index*]\*)

DIM *matriz* \*[, *matriz*]\*

ejemplo: i. DIM *matriz\_cadena*\$(10,10,50)  
ii. DIM *matriz* (100,100)

# DIMN

## matrices

La función DIM devuelve el tamaño máximo de una dimensión especificada de la matriz que se indica. Si no se especifica ninguna dimensión, se supone la primera dimensión. Si la dimensión especificada no existe, o el identificador no es una matriz, devuelve un cero.

sintaxis: *matriz*., = *identificador*  
*index*:= *expresión numérica* { 1 para dimensión 1, etc.}

DIMN(*matriz* [,*index*])

ejemplo: considere la matriz definida por: DIM a(2,3,4)

i. PRINT DIMN(A,1) {imprimirá un 2}  
ii. PRINT DIMN(A,2) {imprimirá un 3}  
iii. PRINT DIMN(A,3) {imprimirá un 4}  
iv. PRINT DIMN(A) {imprimirá un 2}  
v. PRINT DIMN(A,4) {imprimirá un 0}

## DIV

operador

DIV es un operador que realiza una división entera

sintaxis: *expresión\_\_numérica* DIV *expresión\_\_numérica*

ejemplo: i. **PRINT 5 DIV 2** {la salida será 2)  
ii. **PRINT -5 DIV 2** {la salida será -3}

## DIR

Microdrives

El comando DIR obtiene y visualiza el directorio (índice) del Microdrive especificado, en la *ventana* asociada al canal especificado o al canal de omisión.

sintaxis: DIR *dispositivo*

La especificación debe incluir como dispositivo un *Microdrive válido*.

El formato del directorio obtenido como salida de un comando DIR es el siguiente:

sectores libres: = número de sectores libres  
sectores\_disponibles:= número máximo de sectores sobre su cartucho  
nombre de \_archivo:= nombre de archivo en SuperBASIC  
formato de pantalla: *Nombre del volumen*  
*sectores\_\_libres/sectores\_\_disponibles*  
Sectores  
*nombre de archivo*  
  
*nombre de archivo*

ejemplo: i. DIR mdv1\_\_  
ii. DIR "mdv2\_\_"  
iii. DIR "mdv\_\_" & num\_\_microdrives\$ & "\_\_"  
formato de pantalla: **BASIC\_**  
183/221 **sectores**  
demo\_\_1  
**demo. 1 .antiguo**  
demo\_\_2

# DLINE

## BASIC

DLINE es un comando que borra una línea única o una gama de líneas de un programa en SuperBASIC

sintaxis: gama: = 

núm línea TO núm línea	2
núm-línea TO	3
TO núm línea	3
núm línea	4

DLINE RANGO\*[rango]\*

donde: 1 borra un rango de líneas  
2 borra desde la línea especificada hasta el final  
3 borra desde el comienzo de la líneas especificada  
4 borra la línea especificada

- ejemplo:
- i. DLINE 10 TO 70,80,200 TO 400  
{borra las líneas de la 10 a la 70 inclusive, la línea 80 y las líneas de la 200 a la 400 inclusive}
  - ii. DLINE  
{no borra nada}

# EDIT

El comando EDIT introduce el editor de línea del SuperBASIC.

El comando EDIT está estrechamente relacionado con el comando AUTO. La única diferencia está en los valores de omisión de ambos. El valor de omisión del comando EDIT para el incremento del número de línea es cero, y por tanto edita una única línea si no se especifica un segundo parámetro para definir el incremento de- la línea.

Si el número de línea especificado ya existe, se visualiza la línea y puede comenzar la edición. Si la línea no existe, se visualiza el número de línea y puede introducirse ésta

El cursor se puede manipular por la línea de edición utilizando las teclas estándar del QL.



cursor a la derecha



cursor a la izquierda



igual que ENTER pero facilita automáticamente la línea anterior para su edición.



cursor hacia abajo igual que ENTER pero facilita automáticamente la línea siguiente para su edición



borra el carácter derecho



borra el carácter izquierdo

cuando la línea es correcta, puede introducirse en el programa pulsando la tecla ENTER.

Si se especifica un *incremento*, se editará la siguiente línea en secuencia. En caso contrario terminará la edición.

sintaxis: *incremento.. = expresión\_\_numérica*  
EDIT *línea\_\_línea número* [,espacio entre líneas]

- ejemplo:
- i. EDIT 10 {sólo edita la línea 10}
  - ii. EDIT 20,10 {edita las líneas 20,30,etc.}



## EOF dispositivos

EOF es una función que determina si se ha encontrado una condición de fin de archivo en un canal especificado. Si se utiliza EOF sin especificar canal alguno, EOF determinará si se ha encontrado el final de las instrucciones de datos incluidos en el programa.

sintaxis: EOF [(canal)]

ejemplo:    i. IF EOF (#6) THEN STOP  
           ii. IF EOF THEN PRINT “No hay datos”

## EXEC EXEC\_W Q d o s

EXEC y EXEC\_\_W cargan una secuencia de programas y los ejecutan en paralelo.

EXEC vuelve al procesador de comandos después de que todos los procesos hayan comenzado su ejecución.

EXEC W espera hasta que hayan terminado todos los procesos antes de volver

sintaxis: programa:= dispositivo {se usa para especificar el archivo de Microdrive que contiene el programa}

EXEC programa

- i. EXEC mdvl\_comunicaciones
- ii. EXEC\_W mdvl\_proceso\_de\_impresión

# EXIT

repetición EXIT continuará el proceso al final (END) de una estructura FOR o REPEAT.

sintaxis: EXIT *identificador*

- ejemplo:
- i. 

```
100 REM comienzo de un bucle
110 LET cuenta = 0
120 REPEAT bucle
130 LET cuenta = cuenta + 1
140 PRINT cuenta
150 IF cuenta = 20 THEN EXIT bucle
160 END REPEAT bucle
```

{se saldrá del bucle cuando la cuenta alcance el valor 20}
  - ii. 

```
100 FOR n = 1 TO 1000
110 REM instrucciones de programa
120 REM instrucciones de programa
130 IF RND >.5 THEN EXIT n
140 END FOR n
```

{se saldrá de ambos bucles cuando se genere un número aleatorio mayor que 0,5}

# EXP

funciones matemáticas

La función EXP devuelve el valor de e elevado a la potencia del parámetro especificado.

sintaxis: EXP (*expresión\_\_numérica*) {gama desde -500.. 500}

- ejemplo.,
- i. PRINT EXP(3)
  - ii. PRINT EXP(3.141592654)

## FILL

gráficos

El comando FILL conmuta el coloreado en los gráficos. Fill llena cualquier figura convexa dibujada con los procedimientos gráficos o con los gráficos de "tortuga" mientras se va creando la figura. Las figuras cóncavas deben partirse en figuras menores sin formas entrantes para garantizar un coloreado con éxito.

Cuando haya terminado de utilizar esta función de coloreado, debe introducir FILL 0.

sintaxis:     *conector* = *expresión\_\_numérica* {*gama* 0.. 1 }  
          FILL [*canal*, ] *conector*

- ejemplo:     i.    FILL 1 : LINE 10,10 TO 50,50 TO 30,90 TO 10,10 : FILL 0  
                  {dibujará un triángulo relleno de color}
- ii.   FILL 1 : CIRCLE 50,50,20 : FILL 0  
                  {dibujará un círculo lleno de color}

## FILL\$

matrices de cadena

FILL\$ es una función que devuelve una cadena de una longitud determinada, con la repetición de uno o dos caracteres.

sintaxis:     FILL\$(*expresión\_\_de\_\_cadena* *expresión\_\_numérica*)

La expresión de cadena para FILL\$ debe ser uno o dos caracteres de longitud.

- ejemplo:     i.    PRINT FILL\$("a",5)                             {imprimirá aaaaa}
- ii.   PRINT FILL\$("oO",7)                        {imprimirá 0000000}
- iii.  LET a\$ = a\$ + FILL\$("",10)

# FLASH

ventana

FLASH es un comando que conecta y desconecta la intermitencia. Sólo es efectivo en el modo de baja resolución.

FLASH será efectivo en la ventana asociada al canal de omisión especificado.

sintaxis: conector:= expresión\_\_numérica {gama de 0... 1 }

FLASH [canal, ] *conector*

donde: *conector* = 0 eliminará la intermitencia  
conector = 1 conectará la intermitencia

ejemplo: 10 PRINT "A"  
20 FLASH 1  
30 PRINT "intermitencia";  
40 FLASH 0  
50 PRINT "palabra"

advertencia

No debe escribirse sobre un carácter con intermitencia, ya que el resultado es poco claro.

# FOR END FOR

repetición

La instrucción FOR permite la repetición de un grupo de instrucciones de SuperBASIC un número de veces controlado. La instrucción FOR puede usarse tanto en su forma corta, como en su forma larga.

NEXT y END FOR pueden utilizarse juntos en el interior del mismo bucle FOR para proporcionar un bucle epílogo, es decir, un grupo de instrucciones de SuperBASIC que no se ejecutarán si se sale de un bucle a través de una instrucción EXIT, pero que sí se ejecutarán si el bucle FOR termina normalmente.

definición: *for\_\_elem*:= | *expresión numérica*  
| *expresión\_\_numérica* TO *expresión\_\_numérica*  
| *expresión\_\_numérica* TO *expresión\_\_numérica*  
STEP  
*expresión\_\_numérica*

*for\_\_list*:= *for\_\_elem* \*[, *for\_\_elem*]\*

corta

La instrucción FOR va seguida en la misma línea lógica por una secuencia de instrucciones de SuperBASIC. La secuencia de instrucciones se ejecuta repetidamente bajo el control de la instrucción FOR. Cuando la instrucción FOR se ha acabado, continúa el proceso con la línea siguiente. La instrucción FOR no necesita otra instrucción final, como por ejemplo NEXT o END FOR. Los bucles FOR contenidos en una sola línea no deben anidarse.

sintaxis: FOR *variable* = *for\_\_list* : *instrucción* \*[:*instrucción*]\*

ejemplo: i. FOR i=1,2,3,4 TO 7 STEP 2 : PRINT i  
ii. FOR elem = primero TO ultim: LET buffer (elem) = 0

La instrucción FOR es la última instrucción en la línea. Las líneas siguientes contienen series de instrucciones en SuperBASIC, terminadas por una instrucción END FOR. Las instrucciones incluidas entre la instrucción FOR y la instrucción END FOR se procesarán bajo el control de la instrucción FOR.

larga

sintaxis:     FOR *variable* = *for\_list*  
                  *instrucciones*  
           END FOR *variable*

ejemplo:     10 INPUT "datos por favor" !x  
              20 LET factorial = 1  
              30 FOR valor = x TO 1 STEP -1  
              40    LET factorial = factorial \* valor  
              50    PRINT x !!!! factorial  
              60    IF factorial >1E20 THEN  
              70      PRINT "Número muy grande"  
              80      EXIT valor  
              90    END IF  
             100 END FOR valor

En un bucle FOR simple (bucle sin salida —EXIT—) se pueden intercambiar las instrucciones END FOR y NEXT.

comentarios

Para controlar un bucle For debe utilizarse una coma flotante.

advertencia

## FORMAT

El comando FORMAT "formatea" y deja preparado para su uso el cartucho que contiene el Microdrive especificado.

Microdrives

sintaxis:     FORMAT [canal,] *dispositivo*

Dispositivo especifica el Microdrive que debe utilizarse para formatear y el identificador que forma parte de la especificación se utiliza como nombre del volumen para ese cartucho. El comando FORMAT escribirá el número de sectores buenos y el número total de sectores disponibles que se encuentran en el cartucho en el canal de omisión o en el especificado.

Es muy útil formatear un cartucho nuevo varias veces antes de su utilización. Con ello, se acondiciona la superficie de la cinta y le da mayor capacidad.

ejemplo:     i. **FORMAT mdv1\_datos\_cartucho**  
              ii. **FORMAT mdv2\_wp\_letras**

El comando FORMAT puede utilizarse para reinicializar un cartucho usado. Por tanto, es evidente que todos los datos incluidos en el cartucho se perderán.

advertencia

# GOSUB

Por razones de compatibilidad con otros lenguajes BASICs, el SuperBASIC atiende la instrucción GOSUB. Esta instrucción GOSUB transfiere el proceso a la línea cuyo número se ha especificado; la instrucción **RETURN** transferirá de nuevo el control del proceso a la instrucción que sigue a la instrucción GOSUB.

Para la especificación del número de línea, se puede introducir una expresión del tipo:

sintaxis:     GOSUB *número\_línea*

ejemplo:     i. GOSUB 100  
              ii. GOSUB 4\*selecc\_\_variable

**comentario** Las estructuras de control que posee el lenguaje SuperBASIC hacen redundante la instrucción GOSUB.

# GOTO

Por razones de compatibilidad con otros lenguajes BASICs, el SuperBASIC posee la instrucción GOTO. Esta instrucción transfiere incondicionalmente el proceso a la instrucción cuyo número se ha especificado. La instrucción donde se especifica dicho número, puede ser una expresión del tipo:

sintaxis:     GOTO *número\_línea*

ejemplo:     i. GOTO comienzo\_programa  
              ii. GOTO 9999

**comentario** Las estructuras de control disponibles en el lenguaje SuperBASIC hacen redundante a esta instrucción.

# IF THEN ELSE END IF

La instrucción **IF** permite probar las condiciones, y el resultado de esta prueba controlará el cauce siguiente del programa.

La instrucción **IF** puede utilizarse de dos formas, una forma corta y otra larga;

La palabra clave **THEN** seguida en la misma línea lógica por una secuencia de palabras clave de SuperBASIC. Esta secuencia de instrucciones puede contener la palabra clave **ELSE**. Si la expresión de la instrucción **IF** es verdadera (tiene un valor diferente de cero), se procesarán las instrucciones contenidas entre las palabras clave **THEN** y **ELSE**. Si la condición es falsa (tiene valor 0) se procesarán las instrucciones que se encuentren entre **ELSE** y el final de la línea.

corta

Si la secuencia de instrucciones de SuperBASIC no contiene la palabra clave **ELSE**, y si la expresión de la instrucción **IF** es verdadera, se procesarán las instrucciones entre la palabra clave **THEN** y el final de la línea. Si la expresión es falsa, el proceso continúa en la línea siguiente.

sintaxis: *instrucciones* = *instrucciones* \*[:*instrucciones*]

*IF* *expresión* **THEN** *instrucciones* [:**ELSE** *instrucciones*]

- ejemplo:
- i. **IF a = 32 THEN PRINT "Límite" :ELSE PRINT "OK"**
  - ii. **IF test > máximo THEN LET máximo =test**
  - iii. **IF "1" +1 = 2 THEN PRINT "coerción correcta"**

La palabra clave **THEN** es la última entrada de la línea lógica. La secuencia de instrucciones de SuperBASIC se escribe siguiendo las instrucciones **IF**. La secuencia termina con la instrucción **END IF**. Esta secuencia de instrucciones de SuperBASIC sólo se ejecuta si la expresión que contiene la instrucción **IF** tiene un valor diferente de cero. La palabra clave **ELSE** y la segunda secuencia de instrucciones de SuperBASIC son opcionales.

larga 1

La palabra clave **THEN** es la última entrada en la línea lógica. Le sigue una secuencia de instrucciones de SuperBASIC en las líneas siguientes, que termina con la palabra clave **ELSE**. Si la expresión evaluada por la instrucción **IF** resulta ser cero, se procesará esta segunda secuencia de instrucciones.

larga 2

sintaxis: **IF** *expresión* **THEN**  
*instrucciones*  
**[ELSE**  
*instrucciones*]  
**END IF**

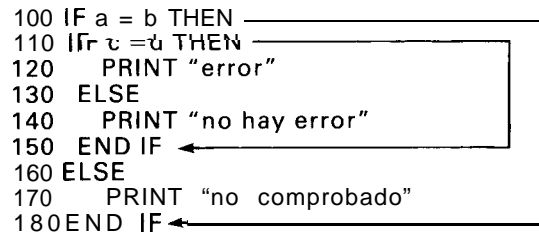
ejemplo:

```
100 LET límite = 10
110 INPUT "teclea un número" ! número
120 IF número > límite THEN
130 PRINT "Fuera de gama"
140 ELSE
150 PRINT "Dentro del límite"
160 END IF
```

**comentario** En las tres formas de la instrucción IF, THEN es opcional. En la forma corta puede sustituirse por dos puntos para distinguir el final de la instrucción IF y el comienzo de la siguiente instrucción. En las dos formas largas puede eliminarse totalmente.

**anidado** Las instrucciones IF pueden anidarse con toda la profundidad que sea necesaria (dependiendo de la memoria disponible). Sin embargo, puede crearse confusión, sobre qué ELSE, END IF, está asociado al IF. El lenguaje SuperBASIC asocia las instrucciones ELSE, etc., anidadas, con la instrucción IF más próxima, por ejemplo

```
100 IF a = b THEN
110 IF c = d THEN
120   PRINT "error"
130 ELSE
140   PRINT "no hay error"
150 END IF
160 ELSE
170   PRINT "no comprobado"
180 END IF
```



ELSE de la línea 130 se ha asociado al segundo IF. El ELSE de la línea 160 se ha asociado con el primer IF (línea 100)

## INK

**ventana** Esta instrucción establece el color con el que se escribirá la salida. El comando INK será efectivo para la ventana asociada al canal especificado o de omisión.

sintaxis: INK [canal,] color

ejemplo: i. INK 5  
ii. INK 6.2  
iii. INK # 2,255



# INKEY\$

**INKEY\$** es una función que devuelve un único carácter de entrada desde el canal especificado o desde el canal de omisión.

Puede especificarse un tiempo opcional de espera antes de volver, puede volver inmediatamente o puede esperar para siempre. Si no se especifica ningún parámetro, **INKEY\$** volverá inmediatamente.

sintaxis:      **INKEY\$** [(canal)  
                  (canal, tiempo)  
                  (tiempo)]  
donde: tiempo = 1..32768 {espera el número de pantallas  
                                  especificados}  
                  tiempo = -1        {espera siempre}  
                  tiempo = 0        {vuelve inmediatamente}

ejemplo:      i.    **PRINT INKEY\$**                    {entrada desde el canal de omisión}  
                ii.    **PRINT INKEY\$(# 4)**        {entrada desde el canal 4}  
                iii. **PRINT INKEY(50)**        {espera 50 pantallas y vuelve de cualquier modo}  
                iv.    **PRINT INKEY\$(0)**        {vuelve inmediatamente (consulta al teclado)}  
                v.    **PRINT INKEY\$(#3,100)**    {espera 100 pantallas la entrada desde el canal 3, para volver de cualquier modo}

# INPUT

El comando **INPUT** permite que el usuario introduzca los datos dentro del programa de SuperBASIC directamente desde el teclado de su QL. El lenguaje SuperBASIC para el programa hasta que los datos se han introducido, y a continuación el programa continúa. Cada elemento de datos debe terminar con la tecla ENTER.

La instrucción **INPUT** introducirá los datos desde el canal especificado o desde el de omisión.

Si se necesita una entrada en un canal especial, el cursor de la ventana asociada a ese canal aparecerá y comenzará a hacerse intermitente...

sintaxis:      *separador*.=!  
                  |  
                  |\  
                  |  
                  TO

*prompt* =[canal,] *expresión\_separador*  
**INPUT** [*prompt*] [*cana/*] [*variable*][, *variable*] \*

ejemplo:      i.    **INPUT ("Ultimo pronóstico" & pron & "Nuevo pronóstico?") !**  
                  pron  
                ii.    **INPUT "Pronóstico"; pronóstico**  
                iii. 10 **INPUT "tamaño de la matriz" ! long**  
                  20 **DIM matriz (long-1)**  
                  30 **FOR elemento = 0 to limit- 1**  
                  40 **INPUT ("datos para el elemento" & elemento) ! matriz**  
                  (b) **(elemento)**  
                  50 **END FOR elemento**  
                  60 **PRINT matriz**

# INSTR

operador

INSTR es un operador que determina si una determinada subcadena está contenida en una cadena especificada. Si se encuentra la cadena, se devolverá a la posición de la subcadena. Si no se encuentra la cadena, INSTR devuelve un cero.

Este cero puede interpretarse como falsedad; ejemplo, la subcadena no estaba contenida en la cadena dada. Para un valor diferente de cero la posición de la subcadena puede interpretarse como verdad; ejemplo, la subcadena estaba contenida en la cadena especificada.

sintaxis: *expresión de cadena* INSTR *expresión de cadena*

ejemplo: i. PRINT "a" INSTR "gato" {imprime 2}  
ii. PRINT "CAT" INSTR "concatena" {imprime 4}  
iii. PRINT "x" INSTR "huevos" {imprime 0}

# INT

funciones  
matemáticas

La función INT devuelve la parte entera de la expresión de coma flotante especificada.

sintaxis: INT (*expresión numérica*)

ejemplo: i. PRINT INT(X)  
ii. PRINT INT (3.14159265412)

# KEYROW

KEYROW es una función que investiga el estado que tienen en un momento dado una fila de teclas. En la tabla que se muestra más abajo puede verse la colocación de dichas teclas en una matriz de 8 filas por 8 columnas. Esta función necesita un parámetro, que debe ser un entero dentro de la gama del 1 al 7, y que es el que indica la fila de teclas a investigar. El valor que devuelve KEYROW es un entero dentro de la gama de 0 a 255, y es una representación en binario de las teclas que se pulsaron en la fila elegida.

Como KEYROW se utiliza como alternativa al mecanismo normal de entrada por teclado, que es mediante INKEY\$ o INPUT, borra todos los caracteres de la memoria del teclado. De este modo, todas las pulsaciones que tuvieron lugar antes de llamar a KEYROW no podrán leerse por los comandos INKEY\$ o INPUT realizados a continuación.

Observe que si las pulsaciones son múltiples, los resultados pueden ser sorprendentes. En particular, si se pulsan las tres teclas de los ángulos de un rectángulo de la matriz simultáneamente, el resultado es como si se hubiera pulsado también la tecla del cuarto ángulo. Las teclas CTRL, SHIFT y ALT son una excepción a la regla, y no interaccionan con ninguna otra del modo indicado.

sintaxis: *fila* = expresión numérica {de 0.. 7}  
KEYROW (*fila*)

```
ejemplo: 10 REM ejecute este programa y pulse algunas teclas
          20 REPEAT bucle
          30 CURSOR 0,0
          40 FOR fila = 0 to 7
          50 PRINT fila !!!KEYROW(fila); " "
          60 END FOR fila
          70 END REPEAT bucle
```

## KEY BOARD MATRIX

COLUMNA								
FILA	1	2	4	8	16	32	64	128
7	↑ SHIFT	CTRL	ALT	X	V	< /	N	
6	8	2	6	Q	E	0	T	U
5	9	W	I	TAB	R	-	Y	8
4	L	3	H	1	A	P	D	J
3	[ ']	CAPS LOCK	K	S	F	=	G	;
2	] `	Z		C	B	[ £	M	~
1	↵ ENTER	⇧	⇧ up	ESC	⇩	] \	SPACE	⇩ down
0	F4	F1	5	F2	F3	F5	4	7

# LBYTES

dispositivos  
Microdrives

El comando LBYTES carga un archivo de datos en una dirección especificada dentro de la memoria.

sintaxis: *dirección\_comienzo*: = *expresión\_\_numérica*  
LBYTES *dispositivo*, *dirección de .comienzo*

- ejemplo:
- i. LBYTES mdv1\_screen, 131072  
{carga una imagen de pantalla}
  - ii. LBYTES mdv1\_program, dirección\_de\_comienzo  
{carga un programa en una dirección especificada}

# LEN

matrices de cadena

LEN es una función que devuelve la longitud de una expresión de cadena especificada.

sintaxis: LEN (*expresión . de .cadena*)

- ejemplo:
- i. PRINT LEN ("LEN averiguará la longitud de esta cadena")
  - ii. PRINT LEN(salida\_cadena\$)

# LET

LET comienza una instrucción de asignación en SuperBASIC. El uso de la palabra clave LET es opcional. Esta asignación puede usarse tanto para asignaciones numéricas como de cadena. El SuperBASIC convertirá inmediatamente los tipos inadecuados en formas adecuadas, siempre que le sea posible.

sintaxis: [LET] variable = *expresión*

ejemplo: i. LET a = 1 + 2  
 ii. LET a\$ = "12345"  
 iii. LET a\$ = 6789  
 iv. b\$ = prueba-de-datos

# LINE LINE\_R gráficos

El comando LINE permite trazar una línea recta entre dos puntos en la *ventana* asociada al canal especificado o de omisión. Los finales de línea se especifican utilizando el sistema gráfico de *coordenadas*.

Con un simple comando LINE se pueden dibujar también líneas múltiples.

Las especificaciones normales necesitan los dos puntos finales de la línea, y esos puntos se pueden especificar bien en coordenadas absolutas (relativas al *origen gráfico*) o en coordenadas relativas (relativas a la posición del *cursor gráfico*). Si se omite el primer punto, la línea se trazará desde la posición del cursor gráfico hasta el punto especificado. Si no se especifica un segundo punto, se moverá el cursor gráfico pero no se dibujará ninguna línea.

LINE siempre dibuja líneas en coordenadas absolutas (es decir relativas al origen).

LINE\_R siempre dibuja líneas en coordenadas relativas al cursor gráfico.

sintaxis: x:= *expresión numérica*  
 y:= *expresión numérica*  
 punto:= x, y

parámetros__2:=	TO punto	1
	,punto TO punto	2
parámetro-l :=	TO punto, ángulo	1
	TO punto	2
	punto	3

LINE [canal, ] *parámetro 1\** [, *parámetros\_\_2\**]

LINE\_R [canal, ] *parámetro\_1\** [, *parámetros\_\_2\**]

donde 1 dibujará desde un punto especificado hasta otro punto especificado  
 2 dibujará desde el último punto que se trazó hasta el punto especificado  
 3 se moverá a un punto especificado pero no se trazará ninguna línea.

ejemplo: i. LINE 0,0 TO 0,50 TO 50,50 TO 50, TO 0,0 {un cuadrado}  
 ii. LINE TO 0.75,0.5 {una línea}  
 iii. LINE 25,25 {mueve el cursor gráfico}

# LIST

El comando LIST permite el listado de una línea o un grupo de líneas de SuperBASIC en un canal especificado o en el de omisión:

LIST debe finalizar con CTRL y espacio

sintaxis:  $línea: = \left| \begin{array}{l} \text{número de línea TO número\_de\_línea} \\ \text{número\_de\_línea TO} \\ \text{TO número\_de\_línea} \\ \text{número de línea} \end{array} \right. \begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}$

LIST [*cana*] *línea*\*[,*línea*]\*

donde: 1 lista desde la línea especificada hasta la línea especificada  
2 lista desde la línea especificada hasta el final  
3 lista desde el comienzo de la línea especificada  
4 lista la línea especificada  
5 lista el archivo completo

ejemplo: i. LIST {lista todas las líneas}  
ii. LIST 10 TO 300 {lista las líneas desde la 10 a la 300}  
iii. LIST 12,20,50 {lista las líneas 12,20 y la 50}

comentario Si la salida de LIST se ha dirigido a un canal abierto, como, por ejemplo, un canal de impresora, el comando LIST habrá creado una copia dura.

# LOAD

dispositivos  
Microdrives

El comando LOAD carga un programa de SuperBASIC desde cualquier dispositivo de SuperBASIC. Este comando realiza un comando NEW automáticamente antes de cargar otro programa, por tanto, LOAD borra cualquier programa cargado con anterioridad.

Si durante la carga, una línea de entrada tiene una sintaxis incorrecta, se inserta la palabra MISTAKE equivocación entre el número de línea y el cuerpo de la línea. Durante la ejecución este tipo de líneas generará errores de algún tipo.

sintaxis: LOAD *dispositivo*

ejemplo: i. LOAD "mdv1\_programa\_test"  
ii. LOAD mdv1\_juegos  
iii. LOAD neti\_3  
iv. LOAD ser1\_e

# LN LOGIO

funciones  
matemáticas

LN devuelve el logaritmo natural del argumento especificado. LOGIO devuelve el logaritmo normal. no existe techo en el parámetro que se suministre si no es el número que pueda almacenar el ordenador.

sintaxis: LOGIO (*expresión\_númerica*) {gama mayor que cero}  
LN(*expresión\_númerica*) {gama mayor que cero}

ejemplo: i. PRINT LOG10(20)  
ii. PRINT LN(3.141592654)

# LOCAL

funciones y  
procedimientos

LOCAL permite definir los *identificadores* como LOCALes a una *función* o a un procedimiento. Los identificadores locales sólo existen dentro de la función o el procedimiento en el que están definidos. Se pierden cuando la función o el procedimiento terminan. Los identificadores locales son independientes de otros identificadores que lleven el mismo nombre y que se encuentren fuera de la función o procedimiento. También pueden definirse *matrices* locales, dimensionándolas con la instrucción LOCAL.

Las instrucciones LOCAL deben preceder a la primera instrucción ejecutable en la función o en el procedimiento en el que se utilizan.

sintaxis: LOCAL *identificador*\*[*identificador*]\*

ejemplo: i. LOCAL a,b,c(10,10)  
ii. LOCAL datos\_temp

Al definir como LOCALes a las variables estamos evitando ciertos nombres que se utilizan en el interior de funciones y procedimientos pueden estropear otras variables con el mismo nombre fuera del procedimiento o de la función.

comentario

# LRUN

devices Microdrives

LRUN carga y ejecuta un *programa* de SuperBASIC desde un dispositivo especificado. Este comando realiza un comando NEW antes de cargar cualquier programa, por tanto, todo programa de SuperBASIC almacenado con anterioridad se borrará.

Si durante la carga una línea de entrada tiene incorrecta su sintaxis de SuperBASIC, se insertará la palabra MISTAKE -equivocación-, entre el número de línea y el cuerpo de la línea. En el momento de su ejecución las líneas de este tipo generarán errores.

sintaxis: LRUN *dispositivo*

ejemplo: i. **LRUN mdvl\_TEST**  
ii. **LRUN mdv1\_juegos**

# MRUN

dispositivos  
Microdrives

El comando MRUN interpreta un *archivo* como si fuera un *programa* de SuperBASIC y lo intercala con el programa cargado en ese momento.

Si se utiliza como *comando directo*, MRUN ejecutará el programa desde su comienzo. Si se une como *instrucción* de un programa, MRUN continuará el proceso en la línea siguiente a MRUN.

Si durante la intercalación una línea de entrada tiene una sintaxis de SuperBASIC incorrecta, se insertará la palabra mistake-equivocación entre el número de la línea y el cuerpo de ésta. En el momento de la ejecución, este tipo de líneas generará errores.

sintaxis: MRUN *dispositivo*

ejemplo: i. **MRUN mdvl\_programa\_encaden**  
ii. **MRUN mdvl\_nuevos\_datos**



# MERGE

dispositivos  
Microdrives

El comando MERGE cargará un archivo desde el dispositivo especificado y lo interpretará como si fuera un programa de SuperBASIC. Si el archivo contiene un número de línea que no aparece en el programa, se añadirá la línea. Si el nuevo archivo contiene una línea para sustituir por otra ya existente, se sustituirá dicha línea. El resto de las líneas del antiguo programa permanecerán inalteradas.

Si durante la acción del comando MERGE alguna línea de entrada tiene una sintaxis de SuperBASIC incorrecta, se insertará la palabra MISTAKE -equivocación- entre el número de línea y el cuerpo de la línea. Este tipo de líneas generan errores en el momento de la ejecución.

sintaxis:     M ERG E *dispositivo*

ejemplo:     i.   MERGE **mdvl\_programa\_superp**  
              ii. **MERGE mdvl\_nuevos\_datos**

# MOD

operadores

MOD es un operador que da el módulo o resto de una división entre dos enteros.

sintaxis:     *expresión\_\_numérica* MOD *expresión\_\_numérica*

ejemplo:     i.   **PRINT 5 MOD 2**     {se imprimirá un 1}  
              ii. **PRINT 5 MOD 3**    {se imprimirá un 2}

# MODE

pantalla

El comando MODE establece la resolución de la pantalla y el número de colores simples que se pueden visualizar. El comando MODE limpia (vacía) todas las ventanas que se encuentran en ese momento en la pantalla, pero mantiene su forma y posición. Si se cambia a modo de baja resolución (8 colores) se establecerá en 2,0 el tamaño mínimo por los caracteres:

sintaxis:      MODE *expresión\_numérica*

                  donde: 8 ó 256 seleccionan el modo de baja resolución  
                          4 ó 512 seleccionan el modo de alta resolución

ejemplo:      i.  MODE 256  
                  ii. MODE 4

# MOVE

gráficos de "tortuga"

El comando MOVE mueve la "tortuga" en la ventana asociada al canal especificado o de omisión, una distancia especificada en la dirección que esté vigente en ese momento. Esa dirección puede especificarse mediante los comandos TURN o TURNTO. El factor de escala se utiliza para determinar la distancia a la que se mueve la tortuga. Si especificamos una distancia negativa, la "tortuga" se moverá hacia atrás.

La "tortuga" se mueve en la ventana asociada a una canal especificado o de omisión.

sintaxis:      *distancia.*, = *expresión\_numérica*

                  MOVE: [*cana*/, ] *distancia*

ejemplo.,      i.  MOVE #2,20 {mueve la tortuga en el canal 2, 20 unidades  
  hacia delante}  
                  ii. MOVE -50 {mueve la "tortuga" en el canal de omisión 50  
  unidades hacia atrás}

NET permite establecer el número de la estación de la Red Local. Si no se especifica el número de la estación, el QL supone como número de estación el 1.

sintaxis: estación: = *expresión\_númerica* {gama de 1. 127)

NET estación

ejemplo: i. NET 63

ii. NET 1

N E T  
red local

Si más de una estación de la Red Local tiene el mismo número puede crearse confusión.

comentario

El comando NEW limpia totalmente el antiguo programa y las variables y *canales* que no sean 0, 1 y 2.

sintaxis: NEW

ejemplo: **NEW**

N E W

# NEXT

repetición

NEXT se utiliza para terminar (o crear un bucle *epílogo*) en los bucles REPEAT y FOR.

sintaxis:      NEXT *identificador*

El identificador debe estar en concordancia con el bucle que controla NEXT.

- ejemplo:
- i.   10 REMark este bucle no termina nunca  
      20 REPEAT bucle\_infinito  
      30   PRINT "sigue el bucle"  
      40 NEXT bucle\_infinito
  - ii.  10 REMark este bucle se repite 20 veces  
      20 LET límite = 20  
      30 FOR índice = 1 TO límite  
      40   PRINT índice  
      50 NEXT índice
  - iii. 10 REMark este bucle prosigue hasta encontrar el 30  
      20 REPEAT bucle  
      30   LET número = RND (1 TO 100)  
      40   IF número <> 30 THEN NEXT bucle  
      50   PRINT número; "es 30"  
      60 EXIT bucle  
      79 END REPEAT bucle

en REPEAT

Si NEXT se utiliza dentro de un bucle REPEAT - END REPEAT, forzará a que el proceso continúe en la instrucción que sigue a la instrucción que cumple REPEAT.

en FOR

La instrucción NEXT puede utilizarse para repetir el bucle FOR con la variable de control establecida en su próximo valor. Si el bucle FOR se ha terminado, el proceso continuará en la instrucción que sigue a NEXT, en caso contrario, el proceso continúa en la instrucción que sigue a FOR.

# ON...GOTO ON...GOSUB

Por razones de compatibilidad con otros BASICs, el SuperBASIC soporta las instrucciones ON GOTO y ON GOSUB. Estas instrucciones permiten a una variable seleccionar una línea para ser procesada en una instrucción GOTO o GOSUB de entre una lista de posibles *números de línea*. Si se especifican demasiado pocas líneas en la lista, se generará un error.

sintaxis:      ON *variable* GOTO *expresión* \*[, *expresión*]\*  
              ON *variable* GOSUB *expresión* \*[, *expresión*]\*

- ejemplo:
- i.   ON x GOTO 10, 20, 30, 40
  - ii.  ONselecc\_variable GOSUB 1000, 2000, 3000, 4000

comentario

SELECT se puede utilizar en sustitución de los dos comandos del BASIC antes discutidos.

# OPEN OPEN IN OPEN-NEW

dispositivo7  
Microdrives

El comando OPEN permite al usuario relacionar (link) un *cana/ lógico* a un *dispositivo* físico del QL, para tareas de E/S.

Si el canal se relaciona con un Microdrive, el archivo de Microdrive puede ser un archivo existente o un nuevo archivo. En este caso, OPEN\_IN abrirá un archivo de Microdrive ya existente para entradas y OPEN\_NEW creará un nuevo archivo de Microdrive para salida.

sintaxis: *canal*: = #*expresión\_\_numérica*  
OPEN *canal*, *dispositivo*

- ejemplo:
- i. OPEN #5, n\_archivo\$
  - ii. OPEN\_\_9, "mdvl\_nom\_arch"  
{abre el archivo) mdvl\_nom\_arch}
  - iii. OPEN-NEW #7, mdvl\_arch\_datos  
{abre el archivo MDV1}\_\_arch\_datos
  - iv. OPEN#6, con\_\_10×20a20×20-32  
{abre el canal 6 al dispositivo consola, creando un tamaño de ventana de 10x20 pixels en la posición 20,20 con un almacenamiento auxiliar-buffer- tipo teclado de 32 byte}.
  - v. OPEN#8, mdvl\_arch\_lect\_escr.

# OVER

ventanas

El comando OVER selecciona el tipo de sobreimpresión que se ha especificado para la ventana asociada al canal especificado o de omisión. Una vez seleccionado el tipo, su efecto permanece hasta la siguiente utilización de OVER.

sintaxis: *conmutador*: = *expresión-numérica* {*gama* -1. 1 }

OVER [*canal*, ] *conmutador*

donde: *conmutador* = 0 imprime en color *ink* sobre tiras  
*conmutador* = 1 imprime en color *ink* sobre tiras transparentes.  
*conmutador* = - 1 imprime sobre el contenido anterior de la pantalla (XOR)

- ejemplo:
- i. 10 REMark escritura sombreada
  - ii. 20 PAPER 7 : INK 0 : OVER 1  
30 CSIZE 3.1  
40 FOR i = 0 TO 10  
50 CURSOR i, j  
60 IF i = 10 THEN INK 2  
70 PRINT "Sombra"  
80 END FOR i

# PAN

## ventana

Desplaza la ventana entera vigente en ese momento el número de pixels especificado. El color de fondo se mueve para llenar el área vacía.

Se puede especificar a otro parámetro opcional para desplazar sólo parte de la pantalla.

sintaxis: *distancia.*, = *expresión\_numérica*  
*parte:* = *expresión\_numérica*

PAN [*canal,* ] *distancia* [,*parte*]

donde *parte* = 0 (o sin parámetro) la pantalla completa.

*parte* = 3 la totalidad de la línea del cursor.

*parte* = 4 la mitad derecha de la línea del cursor incluyendo la posición de este.

Si la expresión toma un valor positivo, el contenido de la pantalla se mueve a la derecha.

ejemplo: i. PAN #2,50 {se mueve a la izquierda 50 pixels}  
ii. PAN -1 00 {se mueve a la derecha 1 00 pixels}  
iii. PAN 50,3 {mueve la totalidad de la línea del cursor 50 pixels a la izquierda}

## advertencia

Si se utilizan patrones de mezclas (*stipples*), o se está en modo de baja resolución, para mantener el patrón de *stipples* de la pantalla es necesario mover ésta por múltiplos de 2 pixels

# PAPER

## ventana

El comando PAPER establece un nuevo color de fondo (ejemplo el color utilizado por CLS, PAN, SCROLL, etc.). El color paper (del fondo) elegido, permanece vigente hasta que vuelva a utilizarse el comando PAPER. Este comando también establece el color STRIP.

El comando PAPER cambia el color del fondo en la ventana asociada al canal especificado o de omisión.

sintaxis: PAPER [*canal,* ]*color*

ejemplo: i. PAPER #3,7 {fondo blanco en el canal 3}  
ii. PAPER 7,2 {patrón mezcla blanco y rojo}  
iii. PAPER 255 {patrón mezcla negro y blanco a franjas}  
iv. 10 REMark Muestra de colores y mezclas  
11 FOR color = 0 TO 7  
12 FOR contraste = 0 TO 7  
13 FOR MEZCLA = 0 TO 3  
14 PAPER color, contraste, mezcla  
15 SCROLL 6  
16 END FOR mezcla  
17 END FOR contraste  
18 END FOR color  
{no es válido para las televisiones domésticas}

# PAUSE

El comando PAUSE hace que un programa espere durante un período de tiempo especificado. Estas esperas se especifican en unidades de 16.67 ms. Si no se especifica el retraso, el programa se parará indefinidamente. Cualquier tecla que se pulse terminará la pausa -PAUSE- y reiniciará la ejecución del programa. Si no se especifica ningún período de espera, el programa esperará indefinidamente.

sintaxis: *retraso.. = expresión\_\_numérica*

PAUSE [*retraso*]

- ejemplo:
- i. PAUSE 50 {espera 1 segundo}
  - ii. PAUSE 500 {espera 10 segundos}

# PEEK PEEK W PEEK-L BASIC

La función PEEK devuelve el contenido de la posición de memoria especificada.

PEEK tiene tres formas, cada una de las cuales accede respectivamente a un byte (8 bits), una palabra (16 bit) o una palabra larga (32 bit).

sintaxis: *dirección: = expresión\_\_numérica*

PEEK(*dirección*) {acceso por bytes}  
PEEK\_W(*dirección*) {acceso por palabras}  
PEEK\_L(*dirección*) {acceso por palabras largas}

- ejemplo:
- i. PRINT PEEK(12245) {contenido del byte de la posición 12245}
  - ii. PRINT PEEK\_W( 12) {contenido de la palabra formada por las posiciones 12 y 13}
  - iii. PRINT PEEK\_L( 1000) {contenido de la palabra larga que comienza en la posición 1000}

En los casos de palabras y palabras largas, la dirección especificada debe ser una dirección par.

advertencia

# PENUP PENDOWN

gráficos de “tortuga”

Estos comandos operan la “pluma” de los gráficos de “tortuga”. Si la pluma se encuentra hacia arriba, no se dibuja nada. Si la pluma se encuentra para abajo, se dibujan líneas (el efecto es el de una tortuga que se moviese por la pantalla).

La línea se escribirá en la *ventana asociada al canal especificado o al canal de omisión*. La línea se escribirá en el color de texto (INK) del canal al que se ha dirigido la salida.

sintaxis:     PENUP *[canal]*  
              PENDOWN *[canal]*

ejemplo:     i. PENUP            {levanta la pluma en el canal de omisión}  
              ii. PENDOWN #2    {baja la pluma en la ventana asociada al canal 2}

# PI

funciones  
matemáticas

La función PI devuelve el valor de la constante  $\pi$

sintaxis:     PI

ejemplo:     PRINT PI



# POINT POINT\_R graphics

Este comando traza un punto en la posición especificada, en la ventana asociada al canal especificado o al canal de omisión. El punto se traza utilizando para ello el sistema de *coordenadas gráficas*, con referencia al *origen de los gráficos*. El comando POINT R se utiliza cuando los puntos se especifican en valores relativos a la pos& del cursor gráfico, y se dibujan por tanto en relación con los demás puntos. POINT puede trazar puntos.

Con una sola llamada al comando POINT pueden trazarse múltiples puntos

sintaxis:    *x*: = *expresión numérica*  
              *y*: = *expresión-numérica*

*parámetros* = *x*, *y*

POINT [*cana*/,] *parámetros* \*[,*parámetros*]\*

ejemplo:    i.   POINT 256,128                                {traza un punto en (256,128)  
              ii.   POINT *x*, *x\*x*                                {traza un punto en (*x*,*x\*x*)  
              iii. 10 REPEAT ejemplo  
                  20 INK RND (255)  
                  30 POINT RND(100),RND(200)  
                  40 END REPEAT ejemplo

# POKE POKE W POKE\_L BASIC

El comando POKE permite el cambio de una posición de la memoria. Para los accesos por palabras o palabras largas las direcciones especificadas deben ser pares.

El comando POKE tiene tres formas, que acceden respectivamente a bytes (8 bits), a palabras (16 bits) o a palabras largas (32 bits).

sintaxis:    *dirección..* = *expresión numérica*  
              *datos*: =    *expresión-numérica*

POKE *dirección*, *datos*                    {accede a un byte}  
POKE W *dirección*, *datos*                {accede a una palabra}  
POKE\_L *dirección*, *datos*                {accede a una palabra larga]

ejemplo:    i.   POKE 12235,0  
              ii. POKE\_L 131072, 12345

Si se utiliza este comando en las zonas de la memoria utilizadas por el QDOS, puede hacerse que el sistema falle y se pierdan los datos. Por tanto no se recomienda utilizar estas áreas.

advertencia

# PRINT

dispositivos  
microdrives

Permite enviar la salida al canal especificado o de omisión. Normalmente PRINT se utiliza para enviar datos a la pantalla del QL.

sintaxis:      *separador*.. = !  
                                  , \  
  '  
  TO expresión \_\_numérica  
  
*elemento*.. = expresión  
                                  *canal*  
                                  *separador*

PRINT \* [*elemento*] \*

Se permiten muchos separadores de impresión. Por lo menos debe utilizar uno para separar las especificaciones de canal y las expresiones.

- ejemplo:
- i.    PRINT "Hola Mundo"  
      {y dará la salida Hola Mundo en el dispositivo de salida de omisión, que es la pantalla}
  - ii.   PRINT #5, "datos", 1, 2, 3, 4  
      {da salida de los datos en el canal 5 (que antes debía haberse abierto)}
  - iii.   PRINT TO 20; "Esto está en la columna 20"

separadores

! Debe mirarse mejor como un espacio inteligente. Es normal el insertar un espacio entre los elementos de salida en la pantalla. Si el elemento no cabe en la línea sobre la que se está actuando, se generará una nueva línea. Si la posición de impresión en ese momento está al comienzo de una línea, no se da salida a un espacio. El separador ! afecta al siguiente elemento que va a imprimirse y, por tanto, debe colocarse frente al elemento de impresión que se está imprimiendo. También se debe colocar un punto y coma ; o una admiración ! al final de una lista de impresión si debe continuarse el espaciado sobre series de instrucciones PRINT.

Separador normal, el lenguaje SuperBASIC tabula la salida cada 8 columnas

Fuerza una línea nueva

Se abandona la posición de impresión inmediatamente después de la impresión del último elemento. Si no se toman las medidas oportunas, la salida se imprime en forma de corriente continua.

TO Realiza una operación de tabulación. TO seguido por una expresión \_\_numérica avanza la posición de impresión a la columna especificada por esta expresión \_\_numérica. Si la columna de que se trata no tiene sentido para esa operación, o si se ha sobrepasado dicha posición, no se llevará a cabo ninguna acción.

# RAD

funciones  
matemáticas

RAD es una función que convierte un ángulo especificado en grados, en el mismo ángulo especificado en radianes.

sintaxis:      RAD (*expresión \_\_numérica*)

ejemplo:      PRINT RAD (180) {imprimirá 3.1415593}

# RANDOMISE

funciones  
matemáticas

El comando RANDOMISE pone en funcionamiento de nuevo el generador de números aleatorios. Si se especifica un parámetro, éste se tomará como nuevo "germen" para la determinación de un nuevo valor aleatorio. Si no se especifica ningún parámetro el generador se regenera con información interna.

sintaxis: RANDOMISE [expresión\_\_numérica]

ejemplo: i. RANDOMISE {Torna como germen datos internos}  
ii. RANDOMISE 3.2235 {Toma como germen 3.2235}

# RND

funciones  
matemáticas

RND genera números aleatorios. Pueden especificarse hasta dos parámetros para el comando RND. Si no se especifican parámetros, RND devuelve un número pseudo aleatorio de coma flotante en una gama de 0 a 1 ambos excluidos. Si se especifica un parámetro único, RND devuelve un entero de una gama de 0 al parámetro especificado ambos incluidos. Si se especifican dos parámetros, RND devuelve un entero en la gama limitada por los dos parámetros especificados, ambos incluidos.

sintaxis: RND ([expresión\_\_numérica] [TO expresión\_\_numérica])

ejemplo: i. PRINT RND {número de coma flotante entre 0  
y 1}  
ii. PRINT RND (10 TO 20) {entero entre 10 y 20}  
iii. PRINT RND (1 TO 6) {entero entre 1 y 6}  
iv. PRINT RND (10) {entero entre 0 y 10}

# RECOL

ventanas

El comando RECOL vuelve a colocar pixels individuales sobre la ventana asociada al canal especificado o al canal de omisión, de acuerdo con algún patrón preestablecido. Cada parámetro especifica, respectivamente, el color en el que se vuelve a colorear cada pixel: es decir, el primer parámetro especifica el color que deben tomar todos los pixels negros, el segundo los pixels azules, etc.

En las especificaciones deben aparecer colores simples y, por tanto, de la gama del 0 al 7.

sintaxis: *c0*: = nuevo color para el negro  
*c1*: = nuevo color para el azul  
*c2*: = nuevo color para el rojo  
*c3*: = nuevo color para el morado  
*c4*: = nuevo color para el verde  
*c5*: = nuevo color para el cian0  
*c6*: = nuevo color para el amarillo  
*c7*: = nuevo color para el blanco

RECOL [canal, ] *c0, c1, c2, c3, c4, c5, c6, c7,*

ejemplo: RECOL 2,3,4,5,6,7,1,0 {pone el azul en morado, el rojo en verde, el morado en ciano, etc.}

# REMark

BASIC

Esta instrucción permite introducir en el programa un texto con aclaraciones. El lenguaje SuperBASIC ignora el resto de la línea.

sintaxis: REMark texto

ejemplo: REMark Esto es un comentario sobre el programa

comentario

Debe utilizarse REMark para incluir comentarios en el programa en los lugares en los que pueda añadir claridad.

## RENUM BASIC

El comando RENUM permite cambiar la numeración de un grupo o una serie de grupos de líneas de SuperBASIC. Si no se especifican parámetros, el comando RENUM remunerará el programa desde el comienzo hasta el final, comenzando por la línea 100 y numerando por saltos de 10.

Si se especifica una línea de comienzo, la nueva numeración comenzará en esa línea.

Si se especifica un rango, la renumeración afectará a las líneas comprendidas en él.

Si una instrucción GOTO o GOSUB contiene una expresión que comienza por un número, ese número se trata como si fuera un número de línea y se renumera también.

sintaxis:    *línea de comienzo:=*    *expresión\_\_numérica*    {{comienza la nueva numeración}}

*línea\_\_final:* =                *expresiónnumérica*    {final de la nueva numeración}

*número de comienzo..* =    *expresión\_\_numérica*    {número de la línea base}

*saltos:* =                    *expresión\_\_numérica*    {saltos}

RENUM [*línea\_\_de\_\_comienzo* [TO *línea\_\_de\_\_final*;][*número\_\_de\_\_comienzo* [, *saltos*]

ejemplo:    i. **RENUM**                    {renumera el programa completo comenzando en 100 y saltando de 10 en 10}

              ii. **RENUM 100 TO 200** {renumera desde el 100 al 200 y de 10 en 10}

No debe intentar utilizar RENUM para renumerar las líneas del programa fuera de su secuencia, por ejemplo, para mover líneas por el programa. RENUM no debe usarse en programas.

advertencia

## RETurn funciones y procedimientos

RETurn se utiliza para forzar el final de una función o un procedimiento y continuar el proceso en la instrucción que se encuentra detrás de la llamada a la función o al procedimiento. Cuando se utiliza en una función, la instrucción RETurn sirve para devolver los valores de las funciones.

sintaxis:    RETURN [*expresión*]

ejemplo:    i.    **100 PRINT ack (3,3)**  
              **110 DEFineFUNction ack (m,n)**  
              **120 IF m=0 THEN RETurn n+1**  
              **130 IF n=0 THEN RETurn ack (m-1,1)**  
              **140 RETurn ack (m-1, ack(m,n-1))**  
              **150 END DEFine**

```

ii. 10 LET señal = 1
    20 LET núm. error = RND (0 TO 10)
    30 error num. error
    40 DEFinePROCedure error(n)
    50 IF señal THEN
    60   PRINT "CUIDADO";
    70   SElectON n
    80     ON n = 1
    96     PRINT "Microdrive lleno"
    100    ON n = 2
    110    PRINT "No hay espacio"
    120    ON n = REMAINDER
    130    PRINT "Error de programa"
    140    END SElect
    150  ELSE
    160    RETurn
    17  END IF
    180 END DEFine

```

**comentario** No es obligatoria la inclusión de RETURN es un procedimiento. Si el proceso alcanza el final del procedimiento (END DEFine), éste retorna automáticamente.

RETurn utilizado sin parámetros se utiliza para volver de una instrucción GO-SUB.

## REPeat END REPeat

**repetición** El comando REPeat permite construir bucles de repetición generales. Para obtener un efecto mejor debe usarse REPeat con EXIT. Este comando REPeat se puede usar tanto en su forma corta como en su forma larga.

**corta** La palabra clave y el identificador del bucle van seguidas en la misma línea lógica por un punto y una secuencia de instrucciones de SuperBASIC. EXIT finaliza el procesado normal en la línea lógica siguiente.

sintaxis: REPeat *identificador* : *instrucciones*

ejemplo: REPeat **espera** :IF inkey\$ <> "" THEN EXIT **espera**

**larga** La palabra clave REPEAT y el identificador del bucle son las únicas instrucciones en la línea lógica. Las líneas siguientes contienen series de instrucciones de SuperBASIC y con una instrucción END REPeat al final.

Las instrucciones entre REPeat y END REPeat se procesan repetidamente por el SuperBASIC.

sintaxis: REPeat *identificador*  
*instrucciones*  
 END REPeat *identificador*

```

ejemplo: 10 LET número = RND (1 TO 50)
          11 REPEAT adivinanza
          12 INPUT "Adivine", adivinanza
          13 IF adivinanza = número THEN
          14   PRINT "Adivinó correctamente"
          15   EXIT adivinanza
          16 ELSE
          17   PRINT "No lo adivinó"
          18 END IF
          19 END REPEAT adivinanza

```

Normalmente al menos una de las instrucciones del bucle REPEAT es una instrucción EXIT.

comentario

## RESPR

Qdos

RESPR es una función cuya misión es reservar parte del espacio de los procedimientos residentes. (Puede ser, por ejemplo, para ampliar la lista de procedimientos de SuperBASIC.)

sintaxis: *espacio* = *expresión\_\_numérica*

RESPR (*espacio*)

ejemplo: PRINT RESPR( 1024)  
{imprimirá la dirección de base de un bloque de 1024 byte}

# RUN

programas  
basic

El comando RUN permite el comienzo de un programa de SuperBASIC. Si se especifica una línea en el comando RUN, el programa comenzará en ese punto. En caso contrario, el programa comienza en el número de línea más bajo.

sintaxis: RUN (expresión\_\_numérica)

ejemplo: i. **RUN** {ejecución desde el comienzo}  
ii. **RUN 10** {ejecución desde la línea 10}  
iii. **RUN 2\*20** {ejecución desde la línea 40}

comentarios

Aunque el comando RUN se puede utilizar en el interior de un programa, lo normal es utilizarlo para arrancar la ejecución, tecleándolo como comando *directo*.

# SAVE

dispositivos  
Microdrives

El comando SAVE guarda un programa de SuperBASIC en el interior de cualquier dispositivo QL.

sintaxis: línea: = 

	expresión numérica TO expresión__numérica	1
	expresión-__numérica TO	2
	TO expresión__numérica	3
	expresión__numérica	4
		5

**SAVE dispositivo línea \*[, línea]\***

donde 1 guarda desde la línea especificada hasta la otra línea especificada.  
2 guarda desde la línea especificada hasta el final.  
3 guarda desde el comienzo hasta la línea especificada  
4 guarda la línea especificada  
5 guarda el archivo completo

ejemplo: i. **SAVE mdv1\_programa 20 TO 70**  
{guarda las líneas de la 20 a la 70 en el mdv1\_programa}  
ii. **SAVE mdv2\_prueba\_programa 10,20,40**  
{guarda las líneas 10,20,40 en el mdv2\_prueba\_programa}  
iii. **SAVE neto-3**  
{guarda el programa completo en la red local}  
iv. **SAVE ser 1**  
{guarda el programa completo en el canal serie 1}



# SBYTES

dispositivos  
Microdrives

SBYTES permite el almacenamiento de ciertas áreas de la memoria del QL en un dispositivo del QL.

sintaxis: *comienzo\_dirección.. = expresión\_numérica*  
*longitud: = expresión\_numérica*

**SBYTES** *dispositivo, comienzo\_dirección, longitud*

- ejemplo:
- i. **SBYTES mdv1\_datos\_pantalla 131072,32768**  
{guarda 32768 bytes a partir del 131072 sobre el archivo mdv1\_datos\_pantalla}
  - ii. **SBYTES mdv1\_progdat, 50000,10000**  
{guarda 10000 bytes a partir de la posición de memoria 50000, en el mdv1\_progdat}
  - iii. **SBYTES neto-3 32768,32678**  
{guarda desde la posición de memoria 32767 una longitud de 32768 bytes en la red local}
  - iv. **SBYTES ser 1,0,32768**  
{guarda la memoria desde 0 con longitud 32768 bytes, en el canal serie 1}

# SDATE

reloj

El comando SDATE permite reinicializar el reloj

sintaxis: *año: = expresión numérica*  
*mes: = expresión-numérica*  
*día: = expresión-numérica*  
*horas: = expresión-numérica*  
*minutos.. = expresión-numérica*  
*segundos.. = expresión-numérica*

**SDATE** *ario, mes, día, horas, minutos, segundos*

- ejemplo:
- i. **SDATE 1984,4,2,0,0,0**
  - ii. **SDATE 1984,1,12,9,30,0**
  - iii. **SDATE 1984,21,3,0,0,0**

# SIN

funciones matemáticas

La función SIN calcula el seno de la función específica

sintaxis: *ángulo*: = *expresión\_\_numérica* {gama -60000.. 60000 en radianes}

SIN (*ángulo*)

- ejemplo:
- i. PRINT SIN(3)
  - ii. PRINT SIN(3.141592654/2)

# SCALE

gráficos

SCALE permite alterar un factor de escala utilizado en los procedimientos *gráficos*. Una escala de "x" implica que una línea vertical de longitud "x" llenará el eje vertical de la ventana en la que se dibuja la figura. La escala de omisión es una escala de 100. El comando SCALE también permite especificar el origen de coordenadas: Con todo ello permitiremos que la ventana que se utiliza para los gráficos pueda moverse por un espacio gráfico mucho mayor.

sintaxis: *x*: = *expresión numérica*  
*y*: = *expresión\_\_numérica*  
*origen.*, = *x,y*  
*escala*: = *expresión\_\_numérica*

SCALE [*canal*, ] *escala*, *origen*

- ejemplo:
- i. SCALE 0.5,0.1,0.1 {establece una escala de 0.5 con el origen en 0.1,0.1}
  - ii. SCALE 10,0,0 {establece una escala de 10 con el origen en 0,0}
  - iii. SCALE 100,50,50 {establece una escala de 100 con el origen en 50,50}

# SCROLL

gráficos

El comando SCROLL desplaza verticalmente la ventana asociada al canal especificado o al canal de omisión el número de pixels indicado. Este desplazamiento puede ser hacia arriba o hacia abajo. El espacio que queda vacío se rellena con el color de fondo paper (en la parte superior o inferior).

Se puede especificar un tercer parámetro opcional para obtener un desplazamiento de sólo una parte de la pantalla.

sintaxis: *parte*: = *expresión numérica*  
*distancia*., = *expresión-numérica*

donde *parte* = 0 la pantalla entera (omisión si no se especifica parámetros)

*parte* = 1 la parte superior, excluyendo la línea del cursor

*parte* = 2 parte inferior de la pantalla, excluyendo la línea del cursor

SCROLL [*cana*/, ] *expresión numérica* [,*part*]

Si la expresión tiene un valor positivo, el contenido de la pantalla se desplaza hacia abajo.

ejemplo: i. SCROLL 10 {desplazamiento hacia arriba de 10 pixels}  
ii. SCROLL -70 {desplazamiento hacia abajo 70 pixels}  
iii. SCROLL -10,2 {desplaza 10 pixels la parte inferior de la ventana}

El comando SElect permite tomar varios tipos de acciones. La decisión se hace dependiendo del valor de la variable.

definición: *select\_variable*., = *variable -numérica*

*select\_elem*: = | *expresión TO expresión*  
| *expresión*

*select\_list*. = | *select\_elem* \*[, *select\_elem*]\*

Permite seleccionar acciones múltiples dependiendo del valor de una *select\_variable*. La *select\_variable* es el elemento último en la línea lógica. A continuación aparecen conjuntos de instrucciones de SuperBASIC terminados por la instrucción ON siguiente, o por la instrucción END SElect. Si el elemento seleccionado es una expresión, se realiza una comprobación con una aproximación de 1110 elevado a -7; en caso contrario se comprueba el rango de valores que aparece en la expresión TO; esta comprobación se hace sin aproximación e inclusiva. La instrucción ON REMAINDER permite incluir todos aquellos casos en que no se satisfacen otras condiciones de selección.

sintaxis: SElect ON *select\_variable*  
\*[[ON *select\_variable*] = *select\_list*  
*instrucciones*]\*  
[ON *select\_variable*] = REMAINDER  
*instrucciones*  
END SElect

ejemplo: 10 LET núm-error = RND(1 TO 10)  
20 SElect ON núm-error  
30 ON núm\_error = 1  
40 PRINT "Divide por cero"  
50 LET núm-error = 0  
60 ON núm\_error = 2  
70 PRINT "No se encuentra el archivo"  
80 LET núm-error = 0

# SElect END SElect

condiciones

larga

```

90  ON núm_error = 3 TO 5
100 PRINT "No se encuentra archivo de mdv"
110  LET núm-error = 0
120 ON núm_error = REMAINDER
130  PRINT "Error desconocido"
140 END SElect

```

Si la `select_variable` se utiliza en el cuerpo de la instrucción `SElect`, debe estar en concordancia con la `select_variable` dada en la cabecera.

**corta** La forma corta de la instrucción `SElect` permite hacer selecciones en una única línea. En la misma línea lógica se incluye a continuación un conjunto de instrucciones de SuperBASIC formando parte de la instrucción `SElect`. Si la condición definida en la instrucción `SElect` se satisface, se procesará la secuencia de instrucción de SuperBASIC.

sintaxis: `SElect ON select_variable = select_lista: instrucción *[:instrucción]*`

ejemplo: i. `SElect ON datos_test = 1 TO 10 :  
PRINT "Respuesta dentro del rango"`  
ii. `SElect ON respuesta = 0.00001 TO 0.00005 :  
PRINT "Buena precisión"`  
iii. `SElect ON a = 1 TO 10 : PRINT a ! "en la gama"`

**comentario** La forma corta de la instrucción `SElect` permite probar gamas con una facilidad mayor que si se hubiera utilizado una instrucción `IF`.

Compare el ejemplo ii de más arriba con la instrucción `IF` correspondiente

## SEXEC

**Qdos**

Este comando guarda un área de memoria de forma que pueda ser utilizada para su carga y ejecución con una instrucción `EXEC`.

Los datos guardados constituyen un programa en código de máquina

sintaxis: `direccion_comienzo = expresión_numérica {comienzo del área}`  
`longitud = expresión_numérica {longitud del área}`  
`espacio_datos = expresión_numérica {longitud del área de datos que será necesaria para el programa}`

`SEXEC dispositivo, direccion_comienzo, longitud, espacio_datos`

ejemplo: `SEXEC mdv1_programa,262144,3000,500`

**comentario** Antes de intentar utilizar este comando debe leerse la documentación del sistema Qdos.

## SQRT

funciones matemáticas

Este comando calcula la raíz cuadrada del argumento especificado. El argumento debe ser mayor que cero.

sintaxis:     SQRT (*expresión\_númerica*) {gama = 0}

ejemplo:     i. PRINT SQRT (3)                     {imprime la raíz cuadrada de 3}  
              ii. LET C = SQRT (a<sup>2</sup> + b<sup>2</sup>) {hace c igual a la raíz cuadrada de a<sup>2</sup> + b<sup>2</sup>}

## STOP

BASIC

STOP finaliza la ejecución del programa y devuelve el SuperBASIC al *intérprete de comandos*.

sintaxis:     STOP

ejemplo:     i.    STOP  
              ii. IF n = 100 THEN STOP

Después de un comando STOP puede continuar con el comando CONTINUE.

La última línea ejecutable del programa actuará como parada (stop) automática.

comentario

# STRIP

ventanas

STRIP establece el color STRIP vigente en la ventana asociada a un *cana/ específico* o de omisión. El color strip es el color de fondo utilizado cuando se selecciona **OVER 1**. en el momento de seleccionar PAPER, se establecerá automáticamente el color de strip con el nuevo color de fondo PAPER.

sintaxis: STRIP [*canal*, ] *color*

ejemplo: i. STRIP 7 {fija un strip blanco}  
ii. STRIP 0,4,2 {fija un strip a franjas verdes y negras}

comentario

El efecto de STRIP es semejante a utilizar una pluma que sobreillumine.

# TAN

funciones  
matemáticas

TAN calcula la tangente del argumento especificado. Este argumento debe estar dentro de la gama de valores de -30000 a 30000 y debe especificarse en radianes.

sintaxis: TAN [*expresión\_\_numérica*]{gama -30000.. 30000)

ejemplo: i. TAN(3) {imprime tan 3}  
ii. TAN(3.141592654/2) {imprime tan  $\pi/2$ }

# TURN TURNTO

gráficos “tortuga”

TURN permite girar la cabecera de la “tortuga” un ángulo especificado.

TURNTO permite girar a la “tortuga” hacia una dirección dada.

El giro se realiza en la *ventana* asociada a un canal especificado o de omisión.

El ángulo debe especificarse en grados. Si el número de grados del giro es positivo, la “tortuga” girará en sentido contrario a las agujas del reloj, y si es negativo el giro se efectuará en sentido de las agujas del reloj.

Inicialmente la “tortuga” se encuentra señalando el 0, es decir en el lado derecho de la ventana.

sintaxis: *ángulo* := *expresión\_\_numérica* {ángulo en grados}

TURN [*canal*, ] *ángulo*

TURNTO [*canal*, ] *ángulo*

ejemplo: i. **TURN 90** {gira 90 grados}  
ii. **TURNTO 0** {gira para ponerse en la dirección to 0 grados}

# UNDER

ventanas

Incluye o elimina el subrayado en las líneas de salida que aparecen a continuación. El subrayado se realiza en el color INK vigente en ese momento, y en la *ventana* ligada al canal especificado o al *cana*/ de omisión.

sintaxis: *conmutador* := *expresión\_\_numérica* {en la gama de 0... 1 }

UNDER [*canal*, ] *conmutador*

ejemplo: i. **UNDER 1** {subrayado}  
ii. **UNDER 0** {sin subrayado}

# WIDTH

dispositivo

WIDTH permite especificar la anchura de omisión para dispositivos no basados en la consola, como por ejemplo impresoras.

sintaxis: *anchura de línea:= expresión- numérica*

*WIDTH [canal, ] anchura de línea*

- ejemplo:
- i. WIDTH 80 {establece una anchura de dispositivo de 80}
  - ii. WIDTH #6, 72 {establece la anchura de 72 para el dispositivo asociado al canal 6}

# WINDOW

ventanas

Permite que el usuario cambie la posición y el tamaño de una determinada ventana. En el momento de la nueva definición de la ventana se suprimen los bordes existentes.

Las coordenadas se especifican utilizando el sistema de pixels relacionado con el origen de la pantalla.

sintaxis: *anchura:= expresión\_numérica*

*altura: = expresión\_numérica*

*x: = expresión numérica*

*y: = expresión-numérica*

*WINDOW [cana/,] ancho, altura, x, y*

- ejemplo: WINDOW 30, 40, 10, 10 {será una ventana de 30 x 40 pixels a partir de 10,10}