DEU PEP

# ZX Spectrum

## Programación BASIC actualizada y ampliada

*Independently published*
Depósito legal: M-16388-2025
ISBN: 9798291860526
Título: ZX Spectrum: Programación BASIC actualizada y ampliada
Autor: dev pep

# Índice

# Introducción

Bienvenido a este libro sobre programación BASIC en el *ZX Spectrum*.

El objetivo de la presente obra es tratar los mismos conceptos que el manual oficial que acompañaba al *Spectrum* en los años 80. Si bien se tratan los mismos temas, el libro que tiene en sus manos tiene un enfoque distinto al del manual original.

En el desarrollo de los distintos temas se ha intentado **profundizar todo lo posible**, de tal modo que, aunque puede resultar algo abrumador para los principiantes, se han redactado las explicaciones de la forma más clara posible. De esta manera, tanto principiantes como lectores avanzados hallarán aquí todo lo que necesiten. Para un principiante no será necesario comprender cada uno de los tópicos expuestos para seguir avanzando en su lectura.

Si bien no se asume experiencia previa en programación, no estaría de más cierta familiaridad con el funcionamiento del *Spectrum*. Es altamente recomendable saber cómo introducir comandos, escribir programas, editar y eliminar líneas del programa BASIC o ejecutar el código fuente. El libro *ZX Spectrum: Introducción actualizada y ampliada* o la misma guía de introducción original proporcionan este tipo de información, aunque también existen numerosos tutoriales al respecto. Alternativamente, puede resultar útil el apéndice 1 de este libro, que ofrece una breve introducción al funcionamiento básico del teclado, la entrada de comandos y la edición y ejecución de programas.

> El apéndice 5 muestra las disposiciones de teclado del *Spectrum 16K/48K* y del *Spectrum+*, de tal modo que se pueda practicar todo lo aquí expuesto, incluso en el caso de no disponer de un *Spectrum* físico. En este último caso, resultará útil la información sobre mapeo de teclas en emuladores, incluida en el apéndice 1.

Hablando del lenguaje BASIC, se puede afirmar que es idéntico en los modelos de 16 KB y 48 KB, así como en el *Spectrum+*. Las únicas diferencias estriban en la cantidad de memora RAM disponible, forzosamente distinta en el *16K* que en el *48K*.

En cuanto al *Spectrum+*, el funcionamiento es idéntico a sus dos antecesores. Por lo tanto, todo lo dicho aquí es igualmente válido para el *Spectrum 16K*, el *Spectrum 48K* y el *Spectrum+*.

> La ROM original, que marca el comportamiento del ordenador, es idéntica en los tres modelos mencionados.

Tanto si se dispone de un *Spectrum* como de un *Spectrum+*, o incluso si solo se dispone de un emulador *software*, la información contenida en este manual será de gran utilidad a quien desee zambullirse en el mundo de la programación retro.

El comportamiento del *Spectrum* viene específicamente establecido por **una sola aplicación**, escrita «a fuego» en su memoria ROM. Esta aplicación es análoga al sistema operativo de un ordenador: contiene rutinas de bajo nivel que interactúan con los periféricos y recursos, utilidades, **funcionalidad de instrucciones y funciones** BASIC, y un **entorno de programación** que incluye un **editor** de comandos y programas, así como un **intérprete** BASIC.

La ejecución de un programa BASIC es, en realidad, el intérprete en funcionamiento, que forma parte de esta aplicación única que se está ejecutando en todo momento. Aunque es posible que deje de ejecutarse este «sistema operativo» forzando un salto hacia una zona de memoria donde exista un programa debidamente ensamblado, esto queda fuera del alcance de este manual.

## *Instrucciones, funciones, sentencias y comandos*

Estos términos pueden dar pie a confusión si no se utilizan de forma consistente. Por lo tanto vamos a exponer cómo se usan en este manual.

En primer lugar, el lenguaje BASIC del *Spectrum* dispone de distintas **instrucciones** y **funciones**, expresadas mediante ciertas palabras o *tokens* del lenguaje, fácilmente utilizables por parte del usuario. Las instrucciones no retornan valor alguno, mientras las funciones sí lo hacen. Ejemplos de instrucciones son PRINT, INPUT o READ; algunas funciones son COS, RND o VAL.

Mediante estas instrucciones y funciones se pueden teclear **comandos**. Estos sirven para dar órdenes directas al sistema, sin necesidad de escribir un programa. Un ejemplo de comando:

```
PRINT "Buenas tardes, ";a$
```

Una misma línea puede incluir varios comandos, separados por dos puntos (:).

Por último, si en lugar de introducir un comando lo escribimos en una línea de programa, tendremos una **sentencia**. Básicamente, es lo mismo que un comando; solo cambia su ubicación y modo de introducirlo:

```
10 PRINT "Buenas tardes, ";a$
```

Una línea de programa puede incluir varias sentencias, separadas también por dos puntos.

## *Definición sintáctica*

A la hora de definir la sintaxis correspondiente a las distintas funciones e instrucciones, utilizaremos un mismo formato a lo largo de todo el libro para indicar inequívocamente los distintos elementos que componen dicha definición:

- Los elementos entre comillas angulares (‹›) indican que el texto en el interior de las comillas no es literal, sino la descripción del objeto en sí.

- Los corchetes ( [] ) indican que lo que hay en su interior es opcional. Los corchetes en sí no deben escribirse en el comando o sentencia.
- Los tres puntos (**...**) en una lista con elementos numerados indican una cantidad arbitraria de dichos elementos, aunque como mínimo deberá haber uno, a no ser que la lista esté entre corchetes, en cuyo caso podría no haber ninguno de ellos. El carácter separador (o posibles caracteres separadores) de los elementos se especifica justo antes de los tres puntos.
- En caso de haber múltiples opciones, se agrupa la lista de posibilidades dentro de llaves ( {} ) si es obligatorio incluir una, o dentro de corchetes ( [] ) si es opcional su inclusión. En ambos casos, las opciones quedan separadas entre sí por barras verticales ( | ).
- El resto sí es literal. Esto incluye paréntesis ( () ), comas ( , ) y cualquier otro signo o carácter escrito.

Los elementos sintácticos se especifican así:

Variables:

- <var> Identificador de variable de cualquier tipo.
- <v> Identificador de tipo numérico de una sola letra.
- <v$> Identificador de tipo *string*.

Valores (expresiones):

- <Exp> Expresión que produce un valor de cualquier tipo.
- <N> Expresión de tipo numérico (entero o punto flotante).
- <I> Expresión numérica entera.
- <S> Expresión de tipo *string*.

Otros elementos:

- <Eoc> Expresión o controlador (de posición o color).
- <Ecvl> Expresión, controlador, variable o elemento LINE.
- <Colc> Controlador de color.
- <Ios> Índice o *slicer* para *strings* y *arrays*.
- <Texto> Sucesión arbitraria de caracteres.
- <St> Sentencia completa.
- <Slc> *Slicer*.

Las **listas de elementos numerados** añadirán un símbolo guión bajo (_) y un número de índice a cada elemento. Si el número de elementos es arbitrario, k será el índice del último de ellos. Por ejemplo, una lista arbitraria de expresiones empezaría por el elemento <Exp_1> y acabaría con <Exp_k>. Nos referiremos a estos elementos como <Exp_i> genéricamente.

En una lista con un número arbitrario de elementos, deberá haber como mínimo uno de ellos, a no ser que la lista esté entre corchetes ( [] ), en cuyo caso podría no haber ninguno.

Los parámetros pueden tener, en lugar de un número de índice, una letra que especifique mejor su uso, como por ejemplo <N_x> o <N_y>.

> En el caso de que una instrucción o función espere recibir como argumento un valor numérico entero (<I>) y le pasemos un valor de punto flotante, se redondeará automáticamente al entero más próximo. Es también el caso de los índices y *slicers*.
>
> Para pasar un número de punto flotante al entero más próximo, lo que hace el intérprete internamente es sumar 0,5 al número y seguidamente truncar al entero más bajo (aplicando la función INT). Por ejemplo: 3,5 se redondearía al número 4, mientras que -3,5 se redondearía a -3.

Por ejemplo, en el caso de LET, la sintaxis se define como:

```
LET <var>[(<Ios>)] = <Exp>
```

Aquí, LET debe escribirse tal cual, mientras que <var>, <Ios> y <Exp> representan una variable, un índice o *slicer*, y una expresión respectivamente, con lo que no deben entenderse de forma literal. En cuanto a [(<Ios>)] indica que el fragmento (<Ios>) es opcional, pero de estar presente, se deben teclear los paréntesis y, en su interior, el índice o *slicer*.

En este ejemplo:

```
DIM <var>(<I_1>, <I_2>,... <I_k>)
```

Podemos ver que existe un número arbitrario de elementos `<I_i>` separados por comas (`,`), aunque como mínimo habrá un elemento (`<I_1>`).

Veamos una definición algo más compleja:

```
PRINT [[<Eoc_1>][;|,|']
      [<Eoc_2>][;|,|']
        ...
      [<Eoc_k>]]
```

A simple vista puede parecer un poco confuso, pero en realidad no lo es tanto: en primer lugar, podemos comprobar que la lista de parámetros consiste básicamente en una lista numerada encerrada entre corchetes, lo cual significa que puede haber un número arbitrario de elementos o ninguno en absoluto.

En segundo lugar, vemos que todos los elementos de la lista (`<Eoc_i>`) son del tipo «expresión o controlador», y que son todos opcionales.

También podemos ver que los separadores de los elementos de la lista pueden ser un punto y coma (`;`), una coma (`,`) o un apóstrofe (`'`).

Todo esto significa que pueden haber varios separadores consecutivos sin elemento alguno entre ellos.

Finalmente, dado que el último elemento de la definición es también opcional, la sentencia puede terminar tanto en separador como en un elemento `<Eoc>`.

## *Bases numéricas*

Los números corrientes (en la acostumbrada base 10), tanto enteros como fraccionarios, se presentan tal cual, utilizándose la coma (,) como separador de decimales: 234 (entero) o 23,4 (fraccionario). También pueden presentarse con el tipo de letra del *Spectrum* cuando forman parte del código fuente (`234` o `23,4`).

Cuando los números no están en base 10, se les incluye un subíndice que indica la base: $11101010_2$ (binario), $352_8$ (octal), o $EA_{16}$ (hexadecimal). En otros libros o artículos se suelen indicar prefijos (del tipo *0b-* o *0x-*) o sufijos (como *-b* o *-h*) para indicar este tipo de números.

## *Estructura del manual*

Los conceptos y temas se han estructurado de tal forma que la lectura de los capítulos se pueda realizar en orden. Dicho orden está altamente recomendado para quien no tenga experiencia en programación o en el lenguaje BASIC del *ZX Spectrum*. Los más expertos pueden consultar los capítulos y apéndices en el orden que precisen.

Para quien no esté familiarizado con el mundo de la programación, los apéndices proporcionan información que puede resultar muy útil. También contienen conceptos generales que no están directamente relacionados con el lenguaje BASIC. Esto incluye información de referencia sobre aspectos adicionales que no resultan estrictamente necesarios para comprender el lenguaje del *Spectrum*.

A continuación se proporciona un breve resumen de los distintos capítulos del libro:

- **Capítulo 1: Tipos de datos:** Se detallan los diferentes tipos de datos con los que trabaja el lenguaje BASIC del *Spectrum*, así como el concepto y funcionamiento de las variables, expresiones y operadores. Se explican las instrucciones DIM y LET, así como las funciones LEN, STR$, VAL y VAL$.
- **Capítulo 2: Primeras instrucciones y funciones:** Se ven algunas instrucciones (CLEAR, CONTINUE, NEW, REM, RUN y STOP) y funciones (CHR$ y CODE) útiles para empezar a escribir programas.
- **Capítulo 3: La pantalla:** Se detalla el funcionamiento de la escritura de caracteres en pantalla, incluyendo las instrucciones LIST y PRINT.
- **Capítulo 4: Color y dibujo:** Se explica cómo funcionan los atributos de color en el *Spectrum*, así como las instrucciones que tratan con el dibujo de píxeles en pantalla. Las instrucciones de este capítulo son BORDER, BRIGHT, CIRCLE, CLS, DRAW, FLASH, INK, INVERSE, OVER, PAPER y PLOT. Las funciones explicadas son ATTR, INT, PI, POINT y SCREEN$.
- **Capítulo 5: El teclado:** Se explica cómo obtener la entrada del usuario a través del teclado. Se ve la instrucción INPUT y la función INKEY$.
- **Capítulo 6: Impresora, *microdrive* y canales:** Explicaciones sobre la funcionalidad de los canales del *Spectrum*, el tratamiento de la impresora y algunos apuntes sobre los dispositivos de la unidad *ZX Interface 1*. Se habla sobre las instrucciones CAT, ERASE, FORMAT y MOVE, y se detalla la funcionalidad

de las instrucciones CLOSE #, COPY, LLIST, LPRINT y OPEN #.

- **Capítulo 7: Bifurcaciones, bucles y saltos:** Se explican las distintas maneras de modificar el flujo de ejecución de un programa, así como la forma de tomar decisiones que afectan a dicho flujo en tiempo de ejecución. Se ven las instrucciones FOR, GO SUB, GO TO, IF, NEXT, PAUSE, RANDOMIZE y RETURN, así como la función RND.
- **Capítulo 8: Series de datos:** Se explica cómo tratar de forma más eficiente los volúmenes de datos de mayor tamaño. Se detallan las instrucciones BEEP, DATA, READ y RESTORE.
- **Capítulo 9: Funciones:** Se detallan el resto de funciones no vistas hasta el momento, así como la técnica utilizada para definir funciones a medida. Esto incluye la instrucción DEF FN y las funciones ABS, ACS, ASN, ATN, COS, EXP, FN, LN, SGN, SIN, SQR, TAN y USR.
- **Capítulo 10: Memoria y periféricos:** Se explica cómo acceder a la memoria del *Spectrum*, así como el intercambio de información con la cinta de casete y con los puertos de entrada/salida de otros periféricos. Se ven las instrucciones LOAD, MERGE, OUT, POKE, SAVE y VERIFY, así como las funciones IN y PEEK.
- **Capítulo 11: Nuevas posibilidades:** Se consideran las distintas novedades tecnológicas que existen hoy día alrededor del mundo del *Spectrum*, tanto de tipo *software* como *hardware*, que no estaban disponibles en los años 80.

En cuanto a los apéndices, este es su contenido:

- **Apéndice 1: Teclado, comandos y pantalla:** Se trata de un breve resumen del libro *ZX Spectrum: Introducción actualizada y ampliada*. Son las bases para poder utilizar el *Spectrum* de forma básica.
- **Apéndice 2: Juego de caracteres:** Se detalla una lista completa del juego de caracteres del *Spectrum*, así como una explicación de los diversos tipos de caracteres que lo forman.
- **Apéndice 3: Sistema binario y hexadecimal:** Explicación detallada del funcionamiento de los métodos de numeración más utilizados en informática.
- **Apéndice 4: Gestión de memoria:** Se detalla el uso que hace el *Spectrum* de las diferentes zonas de memoria.
- **Apéndice 5: Teclados:** Guía visual de los teclados del *Spectrum* y *Spectrum+*.

- **Apéndice 6: Códigos de error:** Enumeración de los códigos de finalización de sentencias o comandos.
- **Apéndice 7: Variables del sistema:** Referencia de las variables del sistema, con una breve explicación de cada una de ellas.
- **Apéndice 8: La interrupción enmascarable:** Explicación sobre la interrupción enmascarable y su función dentro del sistema.
- **Apéndice 9: Índice del lenguaje BASIC:** Lista completa de palabras clave (*tokens*) pertenecientes al lenguaje BASIC del *Spectrum*.

# Capítulo 1: Tipos de datos

No son muchos los tipos de datos que puede manejar el *Spectrum*. Básicamente disponemos de los tipos numéricos (enteros y coma flotante) y las cadenas de caracteres (*strings*). En este capítulo hablaremos en detalle sobre ambos tipos y cómo se almacenan en memoria.

Por otro lado, existe la posibilidad de agrupar valores dentro de una misma variable, ya sea mediante una secuencia lineal o una matriz de dos o más dimensiones. Son los llamados *arrays*.

También veremos algunas funciones utilizadas para convertir datos de un tipo a otro.

## *Tipos numéricos*

Los números se almacenan de dos formas distintas: por un lado, en forma de enteros. Por otro lado, como números de coma flotante (también llamados de «punto flotante»).

## Números enteros

Cuando el sistema identifica un número como entero, es decir, sin posiciones decimales, almacena la magnitud de dicho número en dos *bytes*. Es decir, los enteros del *Spectrum* tienen 16 *bits*.

> Para una breve explicación sobre la expresión de números usando el sistema binario (y el hexadecimal), véase el apéndice 3.

> Para entender cómo gestiona el *Spectrum* la memoria RAM y cómo representa los distintos tipos de datos, véase el apéndice 4.

Para que el número se almacene de este modo, no solo tiene que estar el sistema seguro de que se trata de un entero (sin punto decimal). Además, este debe estar comprendido entre -65535 y 65535, ambos inclusive.

> El número 0 solo puede representarse como entero, y por ello siempre se codifica usando este formato.

Por ejemplo, como se verá en detalle más adelante, la instrucción `LET` asigna un valor a una variable:

```
LET a=3+4
LET b=3.5+3.5
LET c=3.0
LET d=3e4
LET e=.3e4
```

En estos ejemplos, todos los valores numéricos podrían considerarse enteros, pero no siempre serán identificados (y codificados) como tal. Aquí las variables `a`, `c` y `d` se almacenan como enteros, mientras que `b` y `e` lo son como números de punto flotante.

Por lo tanto, a no ser que necesitemos tener un control absoluto sobre la representación de nuestros números en memoria, lo mejor es olvidarse de ello y dejar que el *Spectrum* se encargue de codificar los números como crea conveniente.

Un **literal entero** se representa mediante una secuencia de dígitos del 0 al 9, con un carácter inicial de signo (`-` o `+`) opcional (si se omite, se entiende positivo).

Por otro lado, existe la posibilidad de indicar un valor entero mediante un **literal entero binario**. Esto se hace mediante el indicador de literal binario `BIN`. Este indicador debe ir seguido de cero o más dígitos `0` y `1`, siempre y cuando el valor especificado no supere 65535. No es posible indicar de este modo un número negativo ni fraccionario. `BIN` sin un número después equivale a 0. Ejemplos: `BIN 10010`, `BIN 0`, `BIN`. Ejemplos no válidos: `BIN 21`, `BIN a$`.

## Números de punto flotante

Cuando un número incluye posiciones decimales o está fuera del rango [-65535, 65535], su representación en memoria se realiza siempre en formato de punto flotante.

Este tipo de número acepta valores positivos desde aproximadamente $2{,}94\times10^{-39}$ ($2^{-128}$) hasta alrededor de $1{,}70\times10^{38}$ ($2^{127}$). El rango de negativos es análogo: desde aproximadamente $-2{,}94\times10^{-39}$ ($-2^{-128}$) hasta alrededor de $-1{,}70\times10^{38}$ ($-2^{127}$).

El problema de la representación en punto flotante es la precisión disponible. Dado que los valores de referencia que usamos los humanos suele estar basado en el sistema decimal (base 10), en el formato de punto flotante, basado en el sistema binario, no siempre pueden representarse de forma exacta todos los números decimales. Esto, sin embargo, no representa ningún problema para el tipo de aplicaciones que se utilizan en el *Spectrum*, ya que es una precisión más que suficiente para la gran mayoría de casos.

> Cualquier entero de hasta 32 bits, tanto positivo como negativo (-4294967295 a 4294967295), puede representarse con precisión absoluta mediante el formato de punto flotante. Fuera de ese rango, algunos números pueden codificarse con pequeños errores de precisión.
>
> En cuanto a los números con posiciones decimales, el problema de representación se soluciona mediante redondeo. Por ejemplo, el número 0,8 no es representable de forma exacta: el valor almacenado es algo así como 0,7999999998137355, aunque el *Spectrum* lo redondeará a 0,8 cuando toque representarlo.

Un **literal de punto flotante** es ligeramente más complejo que uno entero: empieza también con un carácter de signo opcional, y tiene una parte entera y una fraccionaria, separadas por un punto (`.`). Si la parte entera es cero, no es obligatorio incluirla. Son ejemplos `6.54`, `-8.333` o `.75`. También pueden representarse mediante notación científica, es decir, un número (entero o de punto flotante) seguido de un carácter `e` (o `E`) y un número entero (positivo o negativo). El valor que representa es el del primer número multiplicado por 10 elevado al segundo número. Por ejemplo, `56e-2` es igual a 56 multiplicado por 10

elevado a -2 ($56 \times 10^{-2}$), es decir, 0,56. Por otro lado, `-46.998e4` sería igual a -469980 ($46{,}998 \times 10^{4}$).

## *Cadenas de caracteres (strings)*

El *ZX Spectrum* utiliza una tabla de caracteres basada en el estándar ASCII y adaptada a sus necesidades específicas. Dicha tabla debe contener los símbolos, números, letras, caracteres gráficos y otros elementos necesarios para el buen funcionamiento del sistema.

Un *string* o cadena de caracteres no es más que una secuencia de caracteres agrupados de forma secuencial. Una de las formas de expresar un *string* es mediante un **literal**, consistente en una serie de caracteres entre comillas dobles.

Cada carácter tiene asociado un código entre 0 y 255, de tal modo que puede representarse con un solo *byte*.

## Literal *string*

Para crear un literal de tipo *string*, es necesario **escribir los elementos del mismo, encerrados entre comillas dobles** (‘‘). En el caso de desear que el carácter de las comillas dobles aparezca dentro del literal, **hay que indicarlas dos veces consecutivas** en su interior:

```
PRINT "Yo digo ""hola mundo"""
```

> La instrucción `PRINT` escribe en pantalla la información que se le proporciona. Se verá con detalla más adelante.

En este caso, dado que deseamos que el fragmento `hola mundo` vaya entrecomillado, debemos incluir dos comillas delante y dos al final de esas dos palabras. Dado que las dos del final van justo antes de las comillas de cierre del *string* completo, escribiremos las tres comillas seguidas.

Un *string* puede contener cualquier tipo de elemento del juego de caracteres: tanto caracteres comunes como caracteres de control de color o posición, *tokens* (nombres de instrucciones y funciones BASIC) o caracteres gráficos.

> Véase el apéndice 2 para una definición completa del juego de caracteres del *Spectrum*.

A la hora de definir un literal *string* a través del teclado, hay algunos caracteres que no pueden teclearse dentro de las comillas dobles. En tal caso, deberá usarse la función CHR$.

> Para saber cómo teclear caracteres de control de color, véase el capítulo 4.

> El tamaño máximo que podemos darle a un literal *string* dependerá de la memoria RAM disponible. Es importante señalar que estos literales se almacenan, como mínimo, dos veces: una en el código fuente del programa, y una vez realizada la asignación a una variable (durante la ejecución), también en la zona de variables. Evidentemente, si se asigna a más de una variable, se almacenará más veces en esa zona de variables.

## *Slicing* de *strings*

Dentro de un *string* es posible hacer referencia a un solo carácter o a un fragmento del mismo (llamado un *substring*). El *substring* más pequeño de un *string* cualquiera es el *string* vacío, es decir, sin caracteres. El más largo posible es el *string* original en sí.

Para «trocear» un *string* para obtener un carácter o fragmento, se debe indicar, tras el *string* (variable, literal o expresión *string*) y entre paréntesis, **un índice** (un solo valor numérico) o bien **un rango** entre dos valores numéricos. En este último caso, se utilizará un *slicer*, esto es, esos valores numéricos separados por el *token* TO.

Veamos un ejemplo:

```
10 LET a$="Hasta luego, mundo."
20 PRINT a$(4)
30 PRINT a$(7 TO 11)
40 LET a$=a$(14 TO 18)
50 PRINT a$(3)
```

Esto producirá el siguiente resultado:

```
t
luego
n
```

> Los índices y valores de rangos empiezan en 1, a diferencia de otros lenguajes de programación, en los que el primer carácter de un *string* tiene índice 0.
>
> Referencias a índices o *slicers* fuera del rango correcto (posiciones inferiores a 1, o superiores al tamaño del *string*) producen un error 3 (`Subscript wrong`).

La primera sentencia del ejemplo asocia el *string* `"Hasta luego, mundo."` a la variable `a$`. La segunda línea escribe el cuarto (4) carácter de dicho *string* (`"t"`) en pantalla. La línea 30 escribe el *substring* formado por los caracteres del 7 al 11, es decir, `"luego"`. A continuación, la variable `a$` pasa a contener un nuevo *string*: este es el *substring* del *string* inicial, compuesto por los caracteres del 14 al 18, es decir, `"mundo"`. La última sentencia escribe el tercer (3) carácter de ese nuevo *string* (`"n"`).

Como vemos, un índice (un solo valor numérico) hace referencia a una posición dentro del *string*, y no puede ser inferior a 1 ni superior a la longitud total del *string*.

En cuanto a un *slicer*, se define a modo de `<valor1> TO <valor2>`. En este caso, tanto `<valor1>` como `<valor2>` deben hacer referencia también a una posición válida (entre 1 y la longitud total). En caso de que `valor2` sea inferior a `valor1`, se obtendrá un *string* vacío (con cero caracteres). Si son iguales se obtendrá un solo carácter.

Tanto los índices como los valores de los *slicers* pueden ser variables numéricas, literales numéricos o expresiones numéricas (véase la definición de expresión, más adelante). En caso de ser valores de punto flotante, se realiza un redondeo automático al entero más cercano antes de obtener el *substring*.