KLINGPHIX

v. 1.5, Autor: Galileo, 2020

Descripcion

Klingphix es un intérprete puro desarrollado en Phix (http://phix.x10.mx/). Está basado en la utilización de una pila para el paso de datos entre palabras (el equivalente a los procedimientos tradicionales) y usa la notación polaca inversa, siendo procedimental e imperativo. La pila tiene ámbito global, y sólo existe un signo separador de palabras: el espacio en blanco.

Instalación

Klingphix no precisa instalacion. Basta con copiarlo a la carpeta que se desee.

Ejecución

Use el editor de su preferencia para escribir código, guarde el fichero con el programa usando (preferiblemente) la extensión .tlhy, aunque se puede utilizar cualquier otra, siempre y cuando lo indique al ejecutar el intérprete.

Después, escriba en su consola el comando: `Klingphix nombrefichero.tlhy` y su programa se ejecutará.

Para depurar el programa, lo mejor es insertar a intervalos dentro de su código la palabra `pstack`, la cual le mostrará el contenido de la pila en ese instante. También puede usar la palabra `words` para ver, si existen, las palabras definidas por el usuario.

Errores como intentar dividir entre cero, o tratar de obtener la raiz cuadrada de un número negativo (por ejemplo), ocasionarán la finalización del programa con un mensaje de error, el volcado del contenido de la pila y la presentación de la parte del código donde se produjo el fallo.

Klingphix puede utilizarse también en modo interactivo, principalmente para realizar pruebas de código, pero teniendo en cuenta que cuando se ejecute (al pulsar ENTER o RETURN) las variables declaradas dentro del mismo serán tratadas como locales: persisten mientras se ejecute el código, pero no más allá, así que no se podrá consultar su contenido a posteriori. Tampoco podrá usarse la palabra especial `include` (aunque sí `load`).

Comentarios

Los comentarios se escriben entre las palabras de apertura de comentario ({) y cierre de comentario (}). Si bien las palabras sólo son reconocidas si estan separadas de las demás por al menos un espacio en blanco, los comentarios no necesitan dicha separación.

```
{ Comentario en una línea }
{Comentario en
      varias líneas}
```

Tipos de datos

Existen tres tipos de datos: números, cadenas de caracteres y listas.

Ejemplos de números:

```
16 -5.4 +0.16e10
```

Ejemplos de cadenas de caracteres:

```
"Direccion de correo-e" "Hola mundo" "X"
```

Ejemplos de listas:

```
( 1 2 3 )
```

```
( 4 5 ( 6 7 ) 8)
```

```
( ( "Precio" 120.50 ) ( "Cantidad" 1000 ) )
```

```
( ( ( "Jose" "Lopez" "Perez" ) ( "Calle Mayor s/n" 30001 "Murcia" ) ) ( ( "Andres"
"Gomez" ) ( "Avd. Gregorio III, 16, 5 B" 30161 "Llano de Brujas" ) ) )
```

Las listas pueden escribirse tal como se muestran, o pueden crearse desde el contenido de la pila con la palabra `tolist`, indicándole previamente cuantos elementos de la pila van a formar parte de la lista. Véase el siguiente ejemplo:

```
1 2 3 3 tolist
```

Primero se apilan los elementos que van a formar parte de la lista (los números 1, 2 y 3). Después se apila el número que indica cuantos elementos van a formar parte de la lista (3) y luego, con `tolist`, se crea dicha lista, consumiendo los argumentos. Asi, el resultado final depositado en la pila será la lista `( 1 2 3 )`.

Las cadenas de caracteres son, en realidad, una modalidad de lista, por lo que se les puede aplicar las mismas funciones y operadores que las usadas en las listas, como se podrá comprobar más adelante, cuando se trate este tema con más profundidad.

Identificadores

Los identificadores son nombres de variables o de subrutinas, y pueden tener cualquier longitud, y utilizar cualquier símbolo, con una sola salvedad: no puden empezar por los siguientes signos especiales: `$ ! { } [ ] : ; % "` . Además, Klingphix distingue mayúsculas de minúsculas.

Las palabras reservadas incluidas en este manual no pueden ser utilizadas como identificadores, ya que forman parte del núcleo del lenguaje

Operadores Aritméticos

Se implementan los operadores más habituales, como la suma, resta, multiplicación, división y resto.

```
3.5 3 sum   -- 6.5
3 5 sub     -- -2
6 2 mul     -- 12
7 2 div     -- 3.5
7 2 mod     -- 1
```

También pueden usarse los signos habituales: `+ - *` y `/`.

Si el resultado de una operación es demasiado grande (fuera del intervalo entre -1e308 y +1e308 en 32 bits, o -1e4932 y +1e4932 en 64 bits) se devolverá un símbolo especial *+infinity* o *-infinity*. Aparecen como **inf** o **-inf** si se imprimen. También es posible que se genere **nan** o **-nan**. "nan" significa "not a number" (no es un número), es decir, un valor indefinido (como cuando se intenta dividir *inf* entre *inf*). Generalmente indica un error en la lógica del programa, aunque en algunos casos puede ser usado para obtener un valor correcto, como en la división `1 inf /`, cuyo resultado es 0, que se puede interpretar como válido.

Operadores relacionales

Cada uno de los siguientes operadores relacionales produce un resultado 1 (`true`) o 0 (`false`) que se deposita en la cima de la pila.

```
8.8 8.7 less     -- 8.8 menor que 8.7 (false)
-4.4 -4.3 great  -- -4.4 mayor que -4.3 (false)
8 7 lore         -- 8 menor que o igual a 7 (false)
4 4 gore         -- 4 mayor que o igual a 4 (true)
1 10 equal       -- 1 igual a 10 (false)
8.7 8.8 nequal   -- 8.7 no igual a 8.8 (true)
( 1 2 3 ) ( 4 5 6 ) equal -- ( 1 2 3 ) igual a ( 4 5 6 ) (false)
( 7 8 9 ) ( 7 8 9 ) equal -- ( 7 8 9 ) igual a ( 7 8 9 ) (true)
```

También pueden usarse los signos habituales: `<` `>` `>=` `<=` `==` y `#`.

Operadores lógicos

Los operadores lógicos **and** , **or** , **xor** , y **nor** se usan para determinar la condición de "verdad" de una expresión.

```
1 1 and      -- 1 (true)
1 0 and      -- 0 (false)
0 1 and      -- 0 (false)
0 0 and      -- 0 (false)
1 1 or       -- 1 (true)
1 0 or       -- 1 (true)
0 1 or       -- 1 (true)
0 0 or       -- 0 (false)
1 1 xor      -- 0 (false)
1 0 xor      -- 1 (true)
0 1 xor      -- 1 (true)
0 0 xor      -- 0 (false)
1 not        -- 0 (false)
0 not        -- 1 (true)
```

Pueden encontrarse estas operaciones aplicadas a números distintos de uno y cero. La regla es: 0 significa `false`, y cualquier cosa distinta de cero es `true`. Por ejemplo:

```
5 -4 and     -- 1 (true)
6 not        -- 0 (false)
```

Variables

Las variables son áreas de memoria designadas con un nombre donde se puede guardar cualquier tipo de dato. Por ejemplo:

```
%saludo

"Hola mundo" !saludo
```

Primero declara la variable usando el prefijo '**%**'. Si una variable no está declarada no existe. Las variables tienen un ámbito local. Si se declaran fuera de cualquier palabra definida por el usuario, son de ámbito global. La visibilidad de las variables es "hacia abajo", lo que significa que una palabra definida por el usuario puede crear una variable local que podrá ser "vista" por otras que esta invoque, salvo que las mismas declaren una variable homónima. En modo interactivo no existen variables globales.

Usando el prefijo '!' se almacena el contenido de la cima de la pila (en este caso, la cadena de caracteres *"Hola mundo"*) en la variable de nombre `saludo`. Para obtener el contenido de la variable se utiliza el prefijo '$'.

Las variables no tienen tipo, así que pueden contener cualquier clase de dato (incluido código de programa):

```
%temperatura 36.5 !temperatura
```

```
%DirectorEjecutivo "Julia Torres" !DirectorEjecutivo

%articulo ( "Tornillo" 9 0.15 ) !articulo { su contenido es ( "Tornillo" 9 0.15 ) }

"Historia de Roma" !articulo { ahora es "Historia de Roma" }
```

Operaciones con listas

Para obtener un elemento de una lista se utiliza la palabra `get`, como en el siguiente ejemplo:

Sea el contenido de la cima de la pila `( 10 20 30 40 )`, con `1 get` se obtiene una copia del primer elemento (`10`), que se deposita en la pila. Ahora la cima de la pila es el número `10`.

Para obtener el último elemento de la lista, primero deberemos averiguar cuántos elementos la componen, lo que se consigue con la palabra `len`.

Tomando la lista del ejemplo anterior, con `len get` extraemos una copia del último elemento (`40`), que se deposita en la cima de la pila.

No obstante, también pueden utilizarse índices negativos, con lo que `-1 get` daría el mismo resultado.

Si se trabaja con listas anidadas, hay que indicarle los índices del elemento cuya copia vamos a depositar en la cima de la pila, también en forma de lista de números enteros. Así: `( ( 2 4 6 ) ( 10 20 30 ) ) ( 2 3 ) get` depositará el número `30`.

Si lo que se desea es cambiar un elemento de la lista habrá de usarse la palabra `set`.

Siguiendo con la misma lista de los anteriores ejemplos, `50 2 set` producirá como resultado que se cambie el segundo elemento por el número 50, con lo que la lista quedaría así: `( 10 50 30 40)`.

Si se trata de listas anidadas, se procede de forma similar a lo visto anteriormente con `get`. Entonces, si se quisera modificar el número 30 del ejemplo por el número 50, se realizaría de la siguiente manera: `( ( 2 4 6 ) ( 10 20 30 ) ) 50 ( 2 3 ) set`.

Para borrar un elemento de la lista se utiliza la palabra `del`. Por ejemplo:

`2 del` eliminará el segundo elemento de la lista del ejemplo, quedando así:

`( 10 30 40 )`

De igual manera, `( ( 2 4 6 ) ( 10 20 30 ) ) ( 2 3 ) del` producirá como resultado la lista `( ( 2 4 6 ) ( 10 20 ) )`.

Para introducir un nuevo elemento se usa la palabra `put`. Ejemplo: `20 2 put`

Haciendo esto, la lista quedaría como estaba anteriormente: `( 10 20 30 40 )`.

Para introducir el elemento al principio de la lista se escribiría: `20 1 put`. Y, si se quisiera añadirlo al final, `20 0 put`.

Siguiendo la misma tónica, `( ( 2 4 6 ) ( 10 20 ) ) 30 ( 2 3 ) put` producirá como resultado la lista `( ( 2 4 6 ) ( 10 20 30 ) )`.

¿Cómo se crea una lista vacía? Simplemente escribiendo `( )` o `0 tolist`.

Como ya se indicó anteriormente, las cadenas de caracteres son, en realidad, listas de códigos ASCII que representan dichos caracteres.

Así, la cadena `"Hola"` es la representación "legible" de la lista: `( 72 111 108 97 )`

Comprobemos esto con un programa (no se preocupe por las palabras que aún no conoce)

```
%ascii
```

```
( ) !ascii
"Hola"
len [get $ascii swap 0 put !ascii] for
$ascii print nl pstack
```

A continuación, la descripción paso a paso.

```
%ascii       { se declara la variable 'ascii'}
( ) !ascii   { creamos una lista vacia y la guardamos en dicha variable }
"Hola"       { introducimos en la pila la cadena de texto "Hola" }
len          { obtenemos su longitud }
[            { este signo indica el comienzo de una "palabra" anónima, es decir, una
               porción de código sin nombre }
      get    { extrae el caracter indicado del texto y lo apila }
      $ascii { se apila la lista guardada en 'ascii' }
      swap   { se intercambian las posiciones de la lista y el caracter }
      0      { indicamos la posición donde queremos insertar el caracter (al final de la
               lista) }
      put    { lo insertamos }
      !ascii { guardamos el resultado }
]            { este signo indica el final de la "palabra" anónima }
for          { se repite la ejecución de la "palabra" tantas veces como caracteres tiene
               la cadena de texto }
$ascii       { se pone en la cima de la pila la lista guardada en 'ascii' }
print        { se imprime }
nl           { se ejecuta un "retorno de carro", esto es, el nuevo punto de impresión
               salta al inicio de la línea siguiente de la pantalla }
pstack       { con esta palabra se muestra el contenido actual de la pila, que en este
               caso es la cadena de texto "Hola" }
```

Ya tenemo la lista de caracteres ASCII que representa el texto "Hola". Recordar una vez más que se tratan sólo de distintas representaciones de la misma información, que internamente se almacena como la indicada lista de códigos ASCII.

A continuación se muestra (ya sin comentarios) el programa que realiza la operación inversa, es decir, que convierte una lista de códigos ASCII en una cadena de caracteres.

```
%texto "" !texto len [get $texto swap 0 put !texto] for $texto print nl
```

Hay que decir que cualquier intento de utilizar un índice que quede fuera del rango actual dará como resultado la finalización del programa con un mensaje que indicará el contenido de la pila y el punto del pograma donde se produjo el error.

Es decir, se generará un error al intentar hacer algo como `120 10 put` cuando la lista solo tiene, por ejemplo, 5 elementos.

Listas anidadas

La manera de crear listas anidadas (listas que contienen listas) sería como se muestra en el siguiente ejemplo:

```
0 tolist { se crea una lista vacía }

dup      { la duplicamos (ahora hay dos listas vacias en la pila) }

0 put    { insertamos la lista vacia de la cima de la pila en la anterior. Ahora queda
            una lista que contiene una lista vacia }
```

O, de forma más sencilla: `( ( ) )`

Si ahora, por ejemplo, realizamos un `1 get` obtendremos una copia de la lista vacia. Podemos, por ejemplo, insertar en ella un número: `230 0 put` y, luego, volver a guardarla en la lista contenedora. `1 set`. Ahora tendremos en la pila lo siguiente: `( ( 230 ) )`, es decir, una lista que contiene otra lista, la cual a su vez almacena un número.

Veamos otro ejemplo:

```
10 [] for          { apilamos los números del 1 al 10 }

10 tolist          { los guardamos en una lista }
```

O aún más sencillo: `( 10 [] for )`

```
%lista10 !lista10 { guardamos la lista en la variable lista10 }

( )               { creamos una lista vacia }

3                 { indicamos el número de veces que se va a repetir el bucle }

[ drop            { descartamos el índice }

 $lista10 0 put ] { insertamos la lista 'lista10' al final }

for               { ejecutamos 3 veces este bucle }
```

El resultado será una lista que contendrá tres listas de números del 1 al 10.

Ahora, si hacemos `( 2 4 )get` obtendremos el elemento 4 de la segunda lista. Y si ejecutamos `50 ( 2 4 ) set` habremos cambiado dicho elemento por el número 50.

Listas de cadenas de caracteres

Ejemplo de construcción de una lista de cadenas de caracteres:

```
"Esto" "es" "una" "prueba" 4 tolist
```

O más abreviadamente: `( "Esto" "es" "una" "prueba" )`

Cualitativamente, se puede considerar esta como una lista de listas. Así que se le pueden aplicar las mismas operaciones.

`( 4 2 ) get` nos proporcionará la letra "r" en forma de código ASCII (el 114), como podremos comprobar con `pstack`. Usando la palabra `tochar` convertiremos dicho código en su forma legible.

Igualmente es posible modificar, añadir o suprimir cadenas de caracteres completas o caracteres sueltos.

Listas heterogéneas

Nada impide crear listas de elementos de distintos tipos. Por ejemplo:

```
"Hola" 16 11 22 33 3 tolist 3.1416 4 tolist
```

Abreviadamente: `( "Hola" 16 ( 11 22 33 ) 3.1416 )`

Así, `( 3 2 ) get` dará como resultado 22. Y `( 1 2 ) get`, la letra "o".

Un ejemplo práctico de su uso:

```
(
	( ( "Jhon" "Smith" ) 45000 27 185.5 )
	( ( "Jose" "Lopez" "Perez" ) 23000 56 182.0 )
)
```

Esto podría ser un fichero de personas, compuesta de una lista con su nombre y apellido, su sueldo anual, su edad y su estatura.

Así, `( 2 1 -1 ) get` daría como resultado "Perez", que es el último elemento de la lista que compone el nombre (el segundo apellido) del segundo registro.

Nótese que el nombre del primer registro sólo esta compuesto por dos campos, mientras que el del segundo lo está por tres.

Más adelante se desglosarán toda una serie de palabras para la manipulación de listas y cadenas de caracteres.

Estructuras de control

Las estructuras de control permiten controlar el flujo de ejecución de un programa.

Declaración `if`

La palabra **if** ejecuta el código ubicado en la cima de la pila si comprueba que el elemento anterior de la pila es un valor no falso (es decir, distinto de cero). Se pueden definir dos palabras dentro de una lista: la primera se ejecutará cuando la condición sea cierta; y la segunda, cuando sea falsa.

Por ejemplo:

```
2 3 > ["Esto no se ejecuta" print] if
2 3 <  ["Esto si se ejecuta" print] if
2 3 > ( ["Esto no se ejecuta" print] ["Pero esto si" print] ) if
```

Declaración `while`

La palabra **while** utiliza el contenido dos palabras anónimas ubicadas en la cima de la pila: la de la izquierda expresa la acción a realizar si la ejecución de la palabra anónima de la derecha da como resultado el apilado de un valor no falso.

Ejemplo:

`10 [1 - dup print] [dup] while drop` (se imprimirá `9876543210`)

Declaración `until`

La palabra **until** también utiliza el contenido dos palabras anónimas, pero actua de manera ligeramente diferente: la de la izquierda expresa la acción a realizar hasta que la ejecución de la palabra anónima de la derecha da como resultado el apilado de un valor no falso.

Ejemplo:

`20 [dup 1 -] [dup 10 <] until drop` (apilará los números del 20 al 10)

Declaración `for`

La palabra **for** ejecuta una definición anónima tantas veces como lo indique el número situado en la cima de la pila.

Existen tres variaciones a la hora de indicar el número de repeticiones.

En la primera, sólo se especifica cuántas veces se va a repetir el bucle con un número. **for** apilará en cada iteración un valor (el índice) que irá desde 1 hasta el número indicado. Por ejemplo: `10 [print " " print] for` imprimirá los números del 1 al 10.

En la segunda, se le pasa una lista con dos elementos: el valor de inicio y el de final. Ámbos han de ser números, y el valor final ha de ser mayor que el de inicio, pues, en caso contrario, no se ejecutará el código. Ejemplo: `( 5 10 ) [print " " print] for` imprimirá los números del 5 al 10.

En la tercera, se pasará una lista con tres elementos: el valor de inicio, el de final y el incremento (los números pueden ser positivos o negativos, pero no reales): Por ejemplo: `( 20 10 -2 ) [print " " print] for` imprimirá los números pares del 20 al 10 (20, 18, 16, 14, 12, 10).

Subrutinas

Una subrutina es un fragmento de código con un nombre. Se construye anteponiendo el prefijo `:` al nombre de la palabra y queda delimitado con el símbolo `;`. Al incluir el nombre de la subrutina en un programa, cada vez que durante la ejecución del mismo se encuentre dicho nombre se procesarán las palabras que contiene. Por ejemplo:

```
:suma2 2 + ;

4 suma2 print
```

Al ejecutar el programa anterior se imprimirá un 6.

Para que una palabra sea reconocida se ha de definir antes de su uso.

Include

Esta palabra permite incluir en el programa actual código guardado en un fichero, consistente generalmente en colecciones de subrutinas orientadas a la resolución de algún tipo de tarea específica.

Hipotéticamente, podríamos tener una colección de subrutinas (también se le puede llamar un *diccionario de palabras*) para, por ejemplo, fines estadísticos, gestión de ficheros de texto, contabilidad, etc.

include suele situarse habitualmente al principio del programa (por claridad), pero no es obligatorio. Lo que sí es imperativo es que esté al principio de la línea donde se ubique.

La forma de utilizarlo es `include <nombrefichero.tlhy>`. Una vez más recordar que se puede poner cualquier extensión a los ficheros de código, pero siempre ha de especificarse para que el intérprete pueda encontrarlo.

Un ejemplo sería disponer en un fichero la subrutina suma2 que hemos descrito más arriba y, posteriormente, incluirla en nuestro programa. Digamos que se ha guardado en un archivo llamado "fichsuma2.tlhy". Entonces podríamos hacer lo siguiente:

```
include fichsuma2.tlhy

40 suma2 print
```

El resultado impreso sería 42.

Esta palabra es la única que no utiliza notación polaca inversa. A diferencia de **load** no ejecuta el código del fichero cargado. Ahora bien, si dicho fichero sólo contiene rutinas puede usarse **load** para la misma función.

Diccionario de palabras predefinidas.

Significado de las abreviaturas de las definiciones.

- l = lista

- s = cadena de caracteres

- n = número

- o = cualquiera de los anteriores

La parte izquierda de la definición son los datos que toma la palabra. La derecha, los que devuelve.

Por ejemplo, la definición de la suma (palabra '+') sería:

n1 n2 -- n3

lo que significa que toma dos números de la cima de la pila (los sumandos) y deposita uno (el resultado de la suma).

Manipulación de listas/cadenas de caracteres.

len -devuelve la longitud de una lista o cadena de caracteres.

repeat -genera una lista consistente en un número, cadena de caracteres o lista repetida un número determinado de veces.

reverse -invierte la disposición de los elementos de una lista o cadena de caracteres.

flatten -crea una lista única a partir de listas anidadas.

chain -concatena dos listas para formar una sola.

split -genera una lista de cadenas de caracteres a partir de una sola, usando un delimitador para trocearla.

trim -elimina los espacios iniciales y finales de una cadena de caracteres.

convert -reemplaza todas las apariciones de una subcadena.

tolist -convierte un número de elementos en una lista.

get -obtiene un elemento de una lista.

put -inserta un elemento en una lista.

set -substituye un elemento de una lista.

del -elimina un elemento de una lista.

flush -elimina todos los elementos de una lista.

pop -divide una lista en dos partes: el primer elemento y el resto.

slice -devuelve una copia de los elementos de una lista indicados por un punto de inicio y el número de elementos a extraer.

tostr -convierte un número en una cadena de caracteres.

tonum -convierte una cadena de caracteres numéricos en un número.

tochar -convierte un código ASCII en una cadena de caracteres con el símbolo que dicho código representa.

upper -convierte una cadena de caracteres a mayúsculas.

lower -convierte una cadena de caracteres a minúsculas.

min -devuelve el menor de dos elementos o de una lista de elementos.

max -devuelve el mayor de dos elementos o de una lista de elementos.

find -busca un elemento en una lista o cadena de caracteres.

sort -ordena una lista o cadena de caracteres.

len

Definición l/s -- n

Descripción Devuelve el número de elementos que constituyen una lista o cadena de caracteres.

Comentario El argumento no se consume.

Ejemplo 1 ((1 2) (3 4) (5 6)) len -- *3*

Ejemplo 2 "Hola" len -- *4*

Ejemplo 3 ( ) len -- *0*

repeat

Definición o n -- l /s

Descripción Genera una lista compuesta de n repeticiones de un elemento.

Comentario Si se indica 0 como número de repeticiones el resultado será una lista vacía. Klingphix interpreta los elementos numéricos enteros entre 7 y 255 como códigos ASCII, generando una cadena de caracteres.

Los argumentos se consumen.

Ejemplo 1 0 10 repeat -- *(0 0 0 0 0 0 0 0 0 0)*

Ejemplo 2 "Juan" 4 repeat -- *("Juan" "Juan" "Juan" "Juan")*

Ejemplo 3 "=" toasc 10 repeat -- *"=========="*

reverse

Definición l/s -- l/s

Descripción Invierte el orden de los elementos en una lista.

Comentario El argumento se consume.

Ejemplos
(1 3 5 7) reverse -- *(7 5 3 1)*
((1 2 3) (4 5 6)) reverse -- *((4 5 6) (1 2 3))*
(91) reverse -- *(91)*
"Hola" reverse -- *"aloH"*

flatten

Definición l -- l

Descripción Convierte una lista con listas anidadas en una única lista de elementos.

Comentario El argumento se consume.

Ejemplo (18 (19 (45)) (18.4 ( ) 29.3)) flatten -- *(18 19 45 18.4 29.3)*

chain

Definición l/s_1 l/s_2 -- l/s

Descripción Junta dos listas (o dos cadenas de caracteres) en una única lista de elementos (o cadena de caracteres).

Comentario Los argumentos se consumen.

Ejemplos
(1 2 3) (4 5 6) chain -- *(1 2 3 4 5 6)*
"Hola" " mundo" chain -- *"Hola mundo"*

split

Definición s -- l

Descripción Trocea una cadena de caracteres cuyos componentes están separados por un delimitador en una lista de esos componentes. Si no se indica un separador se usará el espacio en blanco.

Comentario El argumento se consume.

Ejemplos
"Esto es una prueba" split -- *("Esto" "es" "una" "prueba")*
("Esto-es-una-prueba" "-") split -- *("Esto" "es" "una" "prueba")*

trim

Definición s -- s

Descripción Elimina los espacios en blanco existentes en los extremos de una cadena de caracteres.

Comentario El argumento se consume.

Ejemplo "   Esto es una prueba   " trim -- *"Esto es una prueba"*

convert

Definición s_1 s_2 s_3 -- s

Descripción Substituye las instancias del segundo parámetro dentro del primero por el tercero.

Comentario Los argumentos se consumen.

Ejemplo "Esto es una prueba" "una" "otra" convert -- *"Esto es otra prueba"*

tolist

Definición o ... n -- l

Descripción Transforma en una lista los n elementos superiores de la pila.

Comentario Los argumentos se consumen.

Ejemplo "Adios" 9.15 1 2 3 3 tolist "fin" 4 tolist -- *("Adios" 9.15 (1 2 3) "fin")*

get

Definición s/l n -- o

Descripción Obtiene un copia del enésimo elemento de una lista o cadena de caracteres.

Comentario Se conserva la lista/cadena de caracteres de origen.

Ejemplos
"Esto es una prueba" 4 get -- *"Esto es una prueba" 111 {el código ASCII de la letra 'o'}*
("Esto" "es" "una" "prueba") 4 get -- *("Esto" "es" "una" "prueba") "prueba"*
("Esto" "es" "una" "prueba") (1 4) get -- *("Esto" "es" "una" "prueba") 111*

set

Definición s/l o n -- s/l

Descripción Substituye el enésimo elemento del primer parámetro por el segundo.

Comentario Los parámetros se consumen.

Ejemplos
"Esto es una prueba" "a" toasc 4 set -- *"Esta es una prueba"*
("Esto" "es" "una" "prueba") "otra" 3 set -- *("Esto" "es" "otra" "prueba")*
("Esto" "es" "una" "prueba") "a" toasc (1 4) set -- *("Esta" "es" "una" "prueba")*

put

Definición s/l o n -- s/l

Descripción Inserta el segundo parámetro en la posición enésima de la lista/cadena de caracteres del primer parámetro.

Comentario Los parámetros se consumen.

Ejemplos
"Esto es una prueba" "a" toasc 4 put -- *"Estao es una prueba"*
("Esto" "es" "una" "prueba") "nueva" 4 put -- *("Esto" "es" "una" "nueva" "prueba")*
"Esto es una prueba" "a" 1 put -- *"aEsto es una prueba"*
"Esto es una prueba" "a" 0 put -- *"Esto es una pruebaa"*
("Esto" "es" "una" "prueba") "a" (1 -1) put -- *("Estao" "es" "una" "prueba")*

del

Definición s/l n -- s/l

Descripción Elimina el enésimo elemento del primer parámetro.

Comentario Los parámetros se consumen.

Ejemplos
"Esto es una prueba" 4 del -- *"Est es una prueba"*
("Esto" "es" "una" "prueba") 4 del -- *("Esto" "es" "una")*
("Esto" "es" "una" "prueba") -1 del -- *("Esto" "es" "una")*
("Esto" "es" "una" "prueba") (3 -1) del -- *("Esto" "es" "un" "prueba")*

flush

Definición l -- l

Descripción Elimina todos los elementos de una lista, devolviendo una lista vacia.

Comentario El argumento se consume.

Ejemplo (18 (19 (45)) (18.4 ( ) 29.3)) flush -- *( )*

pop

Definición l/s -- o l

Descripción Divide una lista o cadena de caracteres en dos partes: su primer elemento y el resto.

Comentario El argumento se consume.

Ejemplo (18 (19 (45)) (18.4 ( ) 29.3)) pop -- *18 ((19 (45)) (18.4 ( ) 29.3))*

slice

Definición l/s n n -- o

Descripción Devuelve una copia de los elementos de una lista o cadena de caracteres indicados por un punto de inicio y el número de elementos a extraer.

Comentario Los dos últimos argumentos se consumen.

Ejemplo "Hola mundo" 3 3 slice -- *"la "*

tostr

Definición n -- s

Descripción Convierte un número en una cadena de caracteres.

Comentario El argumento se consume.

Ejemplo 123 tostr -- *"123"*

tonum

Definición s -- n

Descripción Convierte una cadena de caracteres numéricos en un número.

Comentario El argumento se consume. Si no es un número, devuelve *nan* (not a number, no es un número).

Ejemplo "123" tonum -- *123*

tochar

Definición n -- s

Descripción Convierte un código ASCII en su equivalente simbólico.

Comentario El argumento se consume.

Ejemplo 65 tochar -- *"A"*

toasc

Definición s -- n

Descripción Convierte un caracter en su código ASCII.

Comentario El argumento se consume.

Ejemplo "A" toasc -- *65*

upper

Definición s -- s

Descripción Convierte un caracter o cadena de caracteres a mayúsculas.

Comentario El argumento se consume.

Ejemplo "hola" upper -- *"HOLA"*

lower

Definición s -- s

Descripción Convierte un caracter o cadena de caracteres a minúsculas.

Comentario El argumento se consume.

Ejemplo "HOLA" lower -- *"hola"*

max

Definición o o -- o

Descripción Devuelve el mayor de dos elementos.

Comentario Los argumentos se consumen.

Ejemplos
5 7 max -- *7*
"Adios" "Hola" max -- *"Hola"*
(1 2 3) (1 2 4) max -- *(1 2 4)*

min

Definición o o -- o

Descripción Devuelve el menor de dos elementos.

Comentario Los argumentos se consumen.

Ejemplos
5 7 min -- *5*
"Adios" "Hola" min -- *"Adios"*
(1 2 3) (1 2 4) min -- *(1 2 3)*

find

Definición s/l n/s -- n

Descripción Busca el segundo parámetro dentro del primero, devolviendo su posición. Si no lo encuentra devuelve 0.

Comentario El segundo argumento se consume.

Ejemplos
"Hola mundo" "a" toasc find -- *4*
(10 20 30 40) 30 find -- *3*
"Busca cadena en cadenas" "cadena" find -- *7*
(10 20 30 40 50 60 70 80 90) (40 50 60) find -- *0*
((10 20 30) (40 50 60) (70 80 90)) (40 50 60) find -- *2*

sort

Definición l/s -- l/s

Descripción Ordena ascendentemente los elementos de una lista o cadena de caracteres.

Comentario El argumento se consume.

Ejemplos
(7 5 3 1) sort -- *(1 3 5 7)*
((4 5 6) (1 2 3)) sort -- *((1 2 3) (4 5 6))*
"Hola" sort -- *"Halo"*

Funciones matemáticas

abs	**-calcula el valor absoluto (sin signo) de un número**
sum	**-suma todos los números de una lista**
sqrt	**-calcula la raiz cuadrada de un número positivo**
rand	**-genera un número aleatoriamente (entre 0 y 1)**
sin	**-calcula el seno de un ángulo**
asin	**-calcula el ángulo con un seno dado**
cos	**-calcula el coseno de un ángulo**
acos	**-calcula el ángulo con un coseno dado**
tan	**-calcula la tangente de un ángulo**
atan	**-calcula el arcotangente de un número**
log	**-calcula el logaritmo natural de un número**
sign	**-devuelve -1, 0, o +1 para números negativos, cero o positivos, respectivamente**
mod	**-calcula el resto de la división de dos números**
int	**-devuelve la porción entera de un número**
power	**-calcula la potencia de un número**
pi	**-la constante matemática PI (3.1415926...)**

abs

Definición -n / +n -- n

Descripción Devuelve el número sin signo.

Comentario El argumento se consume.

Ejemplo -5 abs -- *5*

sum

Definición l -- n

Descripción Suma todos los números de una lista.

Comentario El argumento se consume.

Ejemplo (1 2 3 4) sum -- *10*

sqrt

Definición n -- n

Descripción Calcula la raiz cuadrada de un número.

Comentario El argumento se consume.

Ejemplo 16 sqrt -- *4*

rand

Definición -- n

Descripción Genera un número aleatorio entre 0 y 1.

Comentario Sin argumento.

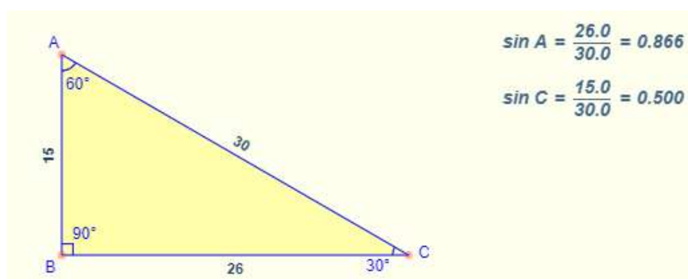
Ejemplo rand -- *0.406172*

sin

Definición n -- n

Descripción Calcula el seno de un ángulo.

Comentario El argumento se consume.



Ejemplo 9 sin -- *0.783327*

asin

Definición n -- n

Descripción Devuelve un ángulo para un seno dado.

Comentario El argumento se consume.

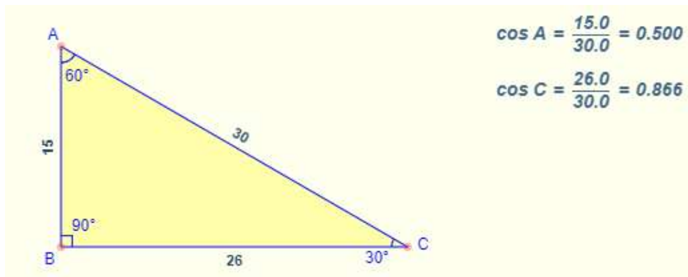
Ejemplo 1 asin -- *1.5708*

cos

Definición n -- n

Descripción Calcula el coseno de un ángulo (dado en radianes).

Comentario El argumento se consume.



Ejemplo .5 cos -- *0.877583*

acos

Definición n -- n

Descripción Devuelve un ángulo para un coseno dado.

Comentario El argumento se consume.

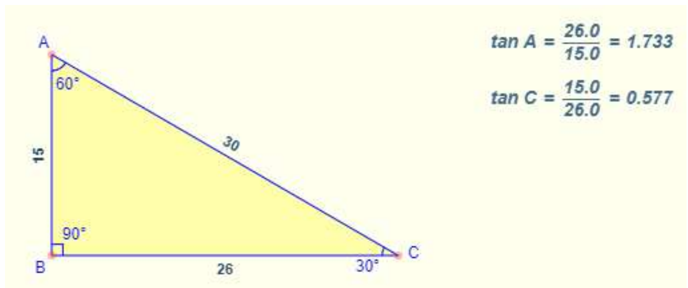
Ejemplo -1 acos -- *3.14159*

tan

Definición n -- n

Descripción Devuelve la tangente de n (dado en radianes).

Comentario El argumento se consume.



Ejemplo 1 tan -- *1.55741*

atan

Definición n -- n

Descripción Devuelve un ángulo para una tangente dada.

Comentario El argumento se consume.

Ejemplo 2 atan -- *1.10715*

log

Definición n -- n

Descripción Devuelve el logaritmo natural de un número.

Comentario El argumento se consume.

Ejemplo 100 log -- *4.60517*

sign

Definición n -- -1/0/+1

Descripción Devuelve el signo de un número.

Comentario El argumento se consume.

Ejemplo -23.45 sign -- *-1*

int

Definición n -- n

Descripción Devuelve la parte entera de un número con decimales.

Comentario El argumento se consume. **int** es un alias de **entier**.

Ejemplo 23.45 int -- *23*

mod

Definición n_1 n_2 -- n

Descripción Devuelve el resto de la división de los parámetros.

Comentario Los argumentos se consumen.

Ejemplo 10 3 mod -- *1*

power

Definición n_1 n_2 -- n

Descripción Devuelve el resultado de elevar un número a la potencia indicada por el segundo.

Comentario Los argumentos se consumen.

Ejemplo 10 3 power -- *1000*

pi

Definición -- n

Descripción Devuelve el valor del número pi.

Comentario Sin argumentos.

Ejemplo pi *-- 3.14159*

Operaciones lógicas binarias

No existen palabras para el desplazamiento a nivel de bit a izquierda o derecha, pero se puede obtener el mismo resultado multiplicando o dividiendo por potencias de 2.

band -realiza la operación AND a nivel de bits

bor -realiza la operación OR a nivel de bits

bxor -realiza la operación XOR a nivel de bits

bnot -realiza la operación NOT a nivel de bits

itob -convierte un número en su representación en bits

btoi -convierte una secuencia de bits en su correspondiente número

band

Definición n_1 n_2 -- n

Descripción Realiza la operación lógica AND entre los correspondientes bits de los parámetros. El resultado a nivel de bit será 1 sólo si los correspondientes bits de ámbos parámetros es 1. En caso contrario, el bit resultante será 0.

Comentario Los argumentos se consumen.

Ejemplo 10 25 band *-- 8 (00001010 00011001 -- 00001000)*

bor

Definición n_1 n_2 -- n

Descripción Realiza la operación lógica OR entre los correspondientes bits de los parámetros. El resultado a nivel de bit será 1 alguno de los correspondientes bits de ámbos parámetros es 1. En caso contrario, el bit resultante será 0.

Comentario Los argumentos se consumen.

Ejemplo 10 25 bor *-- 27 (00001010 00011001 -- 00011011)*

bxor

Definición n_1 n_2 -- n

Descripción Realiza la operación lógica XOR entre los correspondientes bits de los parámetros. El resultado a nivel de bit será 1 sólo si uno de los correspondientes bits de alguno de los parámetros es 1. En caso contrario, el bit resultante será 0.

Comentario Los argumentos se consumen.

Ejemplo 10 25 bxor *-- 19 (00001010 00011001 -- 00010011)*

bnot

Definición n -- n

Descripción Realiza la operación lógica NOT a un número. El resultado a nivel de bit será invertir unos y ceros.

Comentario Los argumentos se consumen.

Ejemplo 200 bnot *-- -201 (11001000 -- 00110111)*

itob

Definición n_1 n_2 -- l

Descripción Convierte un número en una lista compuesta por el número de bits indicados por el segundo parámetro.

Comentario Los argumentos se consumen.

Ejemplo 200 10 itob *-- (0 0 0 1 0 0 1 1 0 0)*

btoi

Definición l -- n

Descripción Convierte una lista compuesta por unos y ceros en el número representado por dichos bits.

Comentario Los argumentos se consumen.

Ejemplo (0 0 0 1 0 0 1 1 0 0) btoi *-- 200*

Operaciones de fichero

Para leer o grabar datos en un fichero primero hay que abrirlo, y, después de realizar las operaciones de lectura o escritura, cerrarlo.

Si se está escribiendo en un fichero, los datos se almacenan primero en una memoria intermedia (llamada *buffer*) hasta que hay los suficientes para realizar una escritura eficiente. Al cerrar un fichero, primero se escriben los datos del buffer que queden pendientes. La lectura de datos de un fichero también se realiza a través de buffers.

Cuando un programa termina, cualquier fichero que permaneciese abierto se cerrará automáticamente.

fopen	-abre un fichero
fclose	-cierra un fichero
fputs	-escribe una cadena de texto en un fichero
fgets	-lee la siguiente línea de texto de un fichero

fopen

Definición s_1 s_2 -- n

Descripción Abre el fichero cuyo nombre es el primer parámetro en el modo que indique el segundo parámetro. El resultado es un número que representa al fichero abierto. Si falla el intento de apertura del fichero, el número devuelto es -1.

Los modos de apertura de un fichero son los siguientes:

"r" - abre el fichero para lectura
"w" - crea un fichero para escritura. Si existe el fichero se sobreescribe.
"u" - abre el fichero para actualizar (permite lectura y escritura). Si no existe se crea
"a" - abre el fichero para añadir datos al final del mismo

A cada línea escrita en un fichero de salida se le añade automáticamente el caracter de *retorno de carro*. Al leer, ese caracter se elimina.

Comentario Los argumentos se consumen.

Ejemplo "fichero.txt" "r" fopen -- *1 (o cualquier otro número; o -1 si no puede abrirlo)*

fclose

Definición n --

Descripción Cierra el fichero indicado por n.

Cuando se termina la ejecución del programa, cualquier fichero abierto se cierra automáticamente.

Comentario El argumento se consume.

Ejemplo 1 fclose *(o cualquier otro número que haya devuelto la operación de apertura)*

fputs

Definición o n --

Descripción Escribe cualquier número, cadena de caracteres o lista en el fichero indicado por el segundo parámetro. No se permiten listas anidadas.

Comentario Los argumentos se consumen.

Ejemplo "Hola mundo" 1 fputs *(o cualquier otro número que haya devuelto la operación de apertura)*

fgets

Definición n -- o

Descripción Lée la siguiente línea del fichero indicado por el parámetro.

Si se alcanza el final del fichero el valor depositado en la pila será -1.

Comentario El argumento se consume.

Ejemplo 1 fgets -- *"Hola mundo"*

Sistema operativo, consola, manipulación de la pila

time -devuelve la hora, minuto, y segundos actuales

date -devuleve el año, mes, día, día de la semana y día del año

cmd -ejecuta un programa y espera a que concluya

end -termina la ejecución del programa

pause -suspende la ejecución del programa durante un periodo de tiempo

platform -devuelve el tipo de sistema operativo sobre el que se ejecuta

version -devuelve la versión actual del intérprete

msec -devuelve el número de segundos transcurridos desde un determinado momento

cls -borra la pantalla

print -imprime el elemento existente en la cima de la pila

nl -salta a la línea siguiente de la pantalla

input -lee información del teclado

pstack muestra el contenido de la pila

clear vacia la pila

drop -elimina el contenido de la cima de la pila

dup -apila una copia de la cima de la pila

swap -intercambia la posición de los dos elementos superiores de la pila

nip -elimina el elemento que está debajo de la cima de la pila

rot -mueve el tercer elemento a la cima de la pila

over -apila una copia del segundo elemento de la pila

@ -apila el identificador de una palabra definida por el usuario

exec -ejecuta la palabra definida por el usuario cuyo identificador está en la cima de la pila

arg -apila la lista de los argumentos pasados al programa

isnum -comprueba si el elemento superior de la pila es un número

isstr -comprueba si el elemento superior de la pila es una cadena de caracteres

words -muestra todas las palabras definidas por el usuario

load -carga y ejecuta un fichero de programa escrito en Klingphix.

eval -ejecuta la palabra anónima ubicada en la cima de la pila.

break -finaliza el número de bucles que se indique.

time

Definición -- l

Descripción Devuelve una lista con la hora, los minutos y los segundos del tiempo actual.

Comentario Sin argumentos.

Ejemplo time -- *(16 45 32) 16 horas 45 minutos 32 segundos*

date

Definición -- l

Descripción Devuelve una lista con el año, el més, el día, el día de la semana y el del año de la fecha actual.

El día de la semana sigue el formato lunes - domingo, donde lunes es 1 y domingo es 7.

Comentario Sin argumentos.

Ejemplo date -- *(2020 8 25 2 238) { Martes 25 de agosto de 2020, día 238 }*

cmd

Definición s -- n

Descripción Ejecuta comandos en el sistema y espera a que finalicen.

Se devuelve el resultado de la operación (0 si ha finalizado correctamente).

Comentario El argumento se consume.

Ejemplo "dir" cmd -- *{ muestra el contenido del directorio actual }*

end

Definición n --

Descripción Finaliza la ejecución del programa, devolviendo un valor de retorno.

Comentario El argumento se consume.

Ejemplo 0 end -- *{ devuelve 0, que significa finalización correcta }*

pause

Definición n --

Descripción Suspende la ejecución del programa durante el tiempo indicado en el parámetro.

Comentario El argumento se consume.

Ejemplo 10 pause -- *(suspende la ejecución durante 10 segundos)*

platform

Definición -- s

Descripción Devuelve el tipo de sistema operativo ("windows" o "linux").

Comentario Sin argumentos.

Ejemplo platform -- *"windows"*

version

Definición -- n

Descripción Devuelve la versión actual del intérprete.

Comentario Sin argumentos.

Ejemplo version -- *1*

msec

Definición -- n

Descripción Devuelve el tiempo transcurrido desde el inicio del programa en segundos. La precisión alcanza la milésima de segundo.

Comentario Sin argumentos.

Ejemplo msec -- *0.578*

cls

Definición --

Descripción Limpia la pantalla.

Comentario Sin argumentos.

Ejemplo cls -- *(se borra la pantalla)*

print

Definición o --

Descripción Imprime en pantalla el elemento existente en la cima de la pila. No introduce un cambio de línea al final.

Comentario El argumento se consume.

Ejemplo "Hola mundo" print -- *(imprime en pantalla el texto "Hola mundo"*

nl

Definición --

Descripción Salta a la línea siguiente.

Comentario Sin argumentos.

Ejemplo "Hola" print nl "mundo" print -- *(Imprime "Hola" en una línea, y "mundo" en la siguiente).*

input

Definición s -- s

Descripción Lee la cadena de texto escrita por el teclado y la deposita en la cima de la pila. Si en la cima de la pila existe una cadena de texto, la imprime previamente.

Comentario Si el argumento es una cadena de texto lo consume.

Ejemplo "Escribe tu nombre" input -- *(Imprime "Escribe tu nombre" y queda a la espera).*

pstack

Definición --

Descripción Muestra el contenido de la pila. Se utiliza en tareas de depuración del código.

Comentario Sin argumentos. Tiene sentido en cuando se usa Klingphix en modo interactivo o cuando se realizan tareas de depuración del código.

Ejemplo 16 "Adios" 0.55 pstack -- *(16, "Adios", 0.55)*

clear

Definición --

Descripción Borra el contenido de la pila.

Comentario Sin argumentos.

Ejemplo 16 "Adios" 0.55 clear -- *( )*

drop

Definición o --

Descripción Elimina el elemento de la cima de la pila.

Comentario El argumento se consume.

Ejemplo 16 "Adios" 0.55 drop -- *16 "Adios"*

dup

Definición o -- o_1 o_2

Descripción Apila una copia de la cima de la pila.

Comentario El argumento se mantiene.

Ejemplo 16 "Adios" 0.55 dup -- *16 "Adios" 0.55 0.55*

swap

Definición o_1 o_2 -- o_2 o_1

Descripción Intercambia las posiciones de los dos elementos superiores en la pila.

Comentario Los argumentos se mantienen.

Ejemplo 16 "Adios" 0.55 swap -- *16 0.55 "Adios"*

nip

Definición o_1 o_2 -- o_2

Descripción Elimina el elemento situado debajo de la cima de la pila.

Comentario El argumento situado bajo la cima se consume.

Ejemplo 16 "Adios" 0.55 nip -- *16 0.55*

rot

Definición o_1 o_2 o_3 -- o_2 o_3 o_1

Descripción Situa el tercer elemento de la pila en la cima.

Comentario Los argumentos se mantienen.

Ejemplo 16 "Adios" 0.55 rot -- *"Adios" 0.55 16*

over

Definición o_1 o_2 -- o_1 o_2 o_1

Descripción Apila una copia del segundo elemento.

Comentario Los argumentos se mantienen.

Ejemplo 16 "Adios" 0.55 over -- *16 "Adios" 0.55 "Adios"*

@

Definición -- n

Descripción No es una palabra, sino un prefijo. Apila el identificador de la palabra definida por el usuario que viene a continuación.

Comentario El argumento se consume.

Ejemplo @suma2 -- *<identificador de la palabra 'suma2'>*

exec

Definición n --

Descripción Ejecuta la palabra cuyo identificador se encuentra en la cima de la pila.

Comentario El argumento se consume.

Ejemplo 4 @suma2 exec -- *6*

arg

Definición -- l

Descripción Situa en la cima de la pila una lista con los argumentos pasados al programa. El primer elemento de la lista es el nombre del programa.

Comentario El argumento se consume.

Ejemplos (Si se invoca: Klingphix.exe prueba.tlhy "Hola mundo") arg -- *("prueba.tlhy", "Hola mundo")*
(Si se invoca: Klingphix.exe) arg -- *( )*

isnum

Definición o --o f

Descripción Devuelve true si el elemento superior de la pila es un número, o false si no lo es.

Comentario El argumento se mantiene.

Ejemplo 22 isnum -- *22 1 {true}*

isstr

Definición o -- o f

Descripción Devuelve true si el elemento superior de la pila es una cadena de texto, o false si no lo es.

Comentario El argumento se mantiene.

Ejemplo 22 isstr -- *22 0 {false}*

words

Definición --

Descripción Muestra las palabras definidas por el usuario (no las propias del lenguaje).

Comentario Sin argumentos. Tiene sentido cuando se usa Klingphix en modo interactivo o para tareas de depuración del código.

omit

Definición n --

Descripción Elimina las palabras definidas por el usuario (no las propias del lenguaje) desde la palabra cuyo identificador se encuentre en la cima de la pila hasta la última definida.

Comentario El argumento se suprime.

Ejemplo (Sea que se hayan defindio las palabras w1, w2 y w3) @w2 omit -- *{ eliminará las palabras w2 y w3 }*

load

Definición s --

Descripción Carga y ejecuta el fichero de código escrito en Klingphix cuyo nombre está situado en la cima de la pila.

Comentario El argumento se consume. Puede utilizarse dentro de un programa para invocar a otro, o para ejecutar un programa desde el modo interactivo.

Ejemplo "Prueba.tlhy" load -- *carga y ejecuta el programa Prueba.tlhy*

eval

Definición s --

Descripción Ejecuta el código escrito en Klingphix situado en la cima de la pila.

Comentario El argumento se consume.

Ejemplo ["Hola mundo" print] eval -- *ejecuta el código que da como resultado el mensaje "Hola mundo"*

break

Definición n --

Descripción Finaliza tantos bucles como se indique en la cima de la pila.

Comentario El argumento se consume. Puede usarse en cualquier bucle (**for**, **while**, **until**), pero teniendo presente que la ruptura no se produce hasta que finalice el ciclo en marcha.

Ejemplos 10 [10 [dup 5 great ([drop 1 break] [print " " print]) if] for print nl] for -- *imprime los números del 1 al 5 10 veces*

10 [10 [dup 5 great ([drop 2 break] [print " " print]) if] for print nl] for -- *imprime los números del 1 al 5 una vez*