PHIXMONTI

v. 1.2, Autor: Galileo, 2019, 2020

Phixmonti es un lenguaje desarrollado en Phix (http://phix.x10.mx/) e inspirado en Monti (https://lduck11007.github.io/MontiLang/). 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. Las variables que se creen, así como la pila, tienen ámbito global, y sólo existe un signo separador de palabras: el espacio en blanco.

Instalación

Phixmonti 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 .**PMT**, aunque se puede utilizar cualquier otra, siempre y cuando lo indique al ejecutar el intérprete.

Después, escriba en su consola el comando: `phixmonti nombrefichero.pmt` 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.

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 inmediatamente anterior al punto donde se produjo el fallo (resaltado este en amarillo).

Comentarios

Los comentarios se escriben entre las palabras de apertura de comentario (/#) y cierre de comentario (#/). Recordar siempre que las palabras sólo son reconocidas si estan separadas de las demás por al menos un espacio en blanco.

```
/# 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 no pueden escribirse tal como se muestran, sino que han de 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: el primer caracter no puede ser un número. Por lo demás, Phixmonti no distingue mayúsculas de minúsculas

Las siguientes palabras reservadas no pueden ser utilizadas como identificadores, ya que forman parte del núcleo del lenguaje:

```
+ - * / mod neg abs max min sum sqrt rand sin arcsin cos arccos tan
arctan log sign int power pi tonum tochar tostr tolist len get set put
del flush head tail repeat reverse flatten split trim subst slice find
sort upper lower chain . drop clear dup swap rot nip over pstack stklen
var true false inf nan _version _platform time date msec sleep bitand
bitor bitxor bitnot int->bits bits->int fopen fgets fputs fclose < >
== != <= >= and or not xor cmd quit print input nl cls if else endif
while endwhile for endfor def enddef include getid exec argument
```

Operadores Aritméticos

Se implementan los operadores más habituales, como la suma, resta, multiplicación y división, así como el más y menos unario.

```
3.5 3 +  -- 6.5
3 5 -    -- -2
6 2 *    -- 12
7 2 /    -- 3.5
-8.1     -- -8.1
+8       -- +8
```

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 <     -- 8.8 menor que 8.7 (false)
-4.4 -4.3 >   -- -4.4 mayor que -4.3 (false)
8 7 <=        -- 8 menor que o igual a 7 (false)
4 4 >=        -- 4 mayor que o igual a 4 (true)
1 10 ==       -- 1 igual a 10 (false)
8.7 8.8 !=    -- 8.7 no igual a 8.8 (true)
[1,2,3] [4,5,6] == -- [1,2,3] igual a [4,5,6] (false)
[7,8,9] [7,8,9] == -- [7,8,9] igual a [7,8,9] (true)
```

Operadores lógicos

Los operadores lógicos **and** , **or** , **xor** , y **not** 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:

```
"Hola mundo" var saludo
```

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 conseguir esto se usa la palabra reservada `var`.

Las variables no tienen tipo, así que pueden contener cualquier tipo de dato:

```
36.5 var temperatura

"Julia Torres" var DirectorEjecutivo

"Tornillo" 9 0.15 3 tolist var articulo /# su contenido es ["Tornillo", 9, 0.15] #/

"Historia de Roma" var 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 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]`.

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]`

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`.

¿Cómo se crea una lista vacía? Simplemente escribiendo `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)

```
"Hola"      /# introducimos en la pila la cadena de caracteres #/

0 tolist    /# creamos una list vacía #/

var ascii   /# la gurdamos en una variable de nombre 'ascii' #/

len for     /# ejecutamos un bucle para recorrer la cadena de
         caracteres. Inmediatamente después de la palabra for
se apila un número que indica la posición que vamos a
examinar (el índice) #/

      get   /# extrae el caracter indicado del texto y lo apila #/

      ascii /# se apila la lista guardada en 'ascii' #/

      swap  /# se intercambian sus posiciones #/

      0     /# indicamos la posición donde queremos insertar el
```

```
         caracter (al final de la lista) /#

      put   /# lo insertamos #/

      var ascii   /# guardamos el resultado #/

endfor      /# repetimos el bucle hasta recorrer toda la cadena de
      caracteres #/

ascii print /# imprimimos el resultado #/
```

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.

```
"" var texto len for get texto swap 0 put var texto endfor texto print
```

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 #/
```

Es decir, que tendríamos esto: `[[]]`

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 endfor     /# apilamos los números del 1 al 10 #/

10 tolist         /# los guardamos en una lista #/
```

```
var del1al10      /# guardamos la lista en la variable del1al10 #/
0 tolist          /# creamos una lista vacia #/
3 for             /# ejecutamos 3 veces este bucle #/
      .           /# descartamos el índice #/
      del1al10 0 put    /# insertamos la lista 'del1al10' al final #/
endfor
```

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

Ahora, si hacemos `2 get 4 get` obtendremos el elemento 4 de la segunda lista. Y si ejecutamos `2 get 50 4 set 2 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
```

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

`4 get 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
```

El resultado será: `["Hola", 16, [11, 22, 33], 3.1416]`

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

Un ejemplo práctico de su uso:

```
"Jhon" "Smith" 2 tolist 45000 27 185.5 4 tolist
"Jose" "Lopez" "Perez" 3 tolist 23000 56 182.0 4 tolist
2 tolist
```

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 get 1 get -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.

Sublistas y otras operaciones

Existe una serie de palabras destinada a extraer fragmentos de listas.

Head devuelve una copia del primer elemento de una lista. Si en la pila tuviésemos la lista `[10, 20, 30, 40]`, con `head` obtendríamos el primer elemento (10).

Tail devuelve una copia de la lista menos el primer elemento. Usando la lista del ejemplo, el resultado de `tail` sería `[20, 30, 40]`.

Slice devuelve una copia de los elementos de una lista indicados por un punto de inicio y el número de elementos a extraer. Así, siguiendo con la lista del ejemplo, `2 2 slice` daría como resultado `[20, 30]`.

Se relacionan a continuación otras palabras interesantes relacionadas con listas.

Repeat genera una lista consistente en un elemento repetido un número determinado de veces. Así, `0 100 repeat` daría como resultado una lista compuesta por 100 ceros.

Reverse invierte la posición de los elementos de una lista. Aplicado a la lista del primer ejemplo de esta sección el resultado sería `[40, 30, 20, 10]`.

Flatten convierte una serie de listas anidadas en una sola lista. Así, siendo la lista a tratar `[10, 20, 30, [1000, 2000], 40]`, el resultado de aplicarle esta palabra sería `[10, 20, 30, 1000, 2000, 40]`.

Split convierte una cadena de caracteres en una lista consistente en las palabras constituyentes de dicha cadena. Se toma como separador de palabras el espacio en blanco. Dada la cadena de caracteres `"Esto es una prueba"`, el resultado sería la lista `["Esto", "es", "una", "prueba"]`.

Trim elimina los espacios en blanco extras al principio y al final de una cadena de caracteres. Si la cadena es `"  Esto es una prueba  "` el resultado será `"Esto es una prueba"`.

Subst substituye cada aparición del elemento indicado en primer lugar por el segundo. Así, `"prueba" "a" "o" subst` dará como resultado `"pruebo"`.

Find devuelve la posición del elemento buscado en una lista. Si no se encuentra devuelve cero. Por lo tanto, al hacer `"prueba" 'e' find` el resultado será 4. Nótese que, para

indicar caracteres individuales, se usan comillas simples, no dobles.

Sort ordena los elementos de una lista. Aplicado a la lista `[23, 1, 9, 2, 15]` el resultado sería `[1, 2, 9, 15, 23]`.

Upper convierte los caracteres de una cadena a mayúsculas. `"hola" upper` dará como resultado `"HOLA"`.

Lower es la inversa de la anterior palabra. `"HOLA" lower` producirá `"hola"`.

Flush vacía la lista que se encuentre en la cima de la pila. Si no es una lista no hace nada.

Chain concatena dos listas. Ejemplo: `1 2 3 3 tolist 4 5 6 3 tolist chain` dará como resultado la lista `[1, 2, 3, 4, 5, 6]`.

`"Hola " "mundo" chain` producirá la cadena de caracteres `"Hola mundo"`.

Estructuras de control

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

Declaración IF

La palabra **if** comprueba si en la cima de la pila se encuentra un valor no falso (es decir, distinto de cero). En caso afirmativo se ejecuta el código que sigue a continuación, hasta encontrar la palabra **else** o **endif**. De lo contrario, salta hasta la palabra que sigue a **else** (si existe) y continua desde ahí la ejecución. Si no existe la palabra **else** la ejecución continúa después de la palabra **endif**.

Por ejemplo:

```
2 3 > if "Esto no se ejecuta" print endif

2 3 < if "Esto si se ejecuta" print endif

2 3 > if "Esto no se ejecuta" print else "Pero esto si" print endif
```

Declaración WHILE

La palabra **while** comprueba si en la cima de la pila se encuentra un valor no falso (distinto de cero). En caso afirmativo se ejecuta el código que sigue a continuación hasta llegar a la palabra **endwhile**. Desde ahí retorna a la palabra **while** y vuelve a comprobar si el valor de la cima de la pila es no falso. Este bucle se repetirá hasta que el valor de la cima de la pila sea falso (cero), momento en que la ejecución del programa proseguirá a continuación de la palabra **endwhile**.

Ejemplo:

`10 dup while 1 - print dup endwhile . pstack` (se imprimirá `9876543210`)

Declaración FOR

La palabra **for** da comienzo a un bucle que se repetirá un numero determinado de veces. La porción de código a ejecutar está delimitada por la palabra **endfor**.

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 for print . " " print endfor` 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. Hay que señalar que, si el valor inicial es mayor que el final, el bucle se ejecutará una vez. Ejemplo: `5 10 2 tolist for print . " " print endfor` 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 de cualquier tipo (positivos, negativos, enteros o reales): Por ejemplo: `20 10 -2 3 tolist for print . " " print endfor` imprimirá los números pares del 20 al 10 (20, 18, 16, 14, 12, 10).

exitfor, por su parte, permite abandonar el bucle *la próxima vez* que se llegue a la palabra **endfor**. Por ejemplo: `100 for dup 10 > if drop exitfor else print endif endfor` imprimirá sólo los números del 1 al 10. Recalcar que **exitfor** *no abandona inmediatamente* el bucle.

Subrutinas

Una subrutina es un fragmento de código con un nombre. Se construye con la palabra `def` <nombre> y queda delimitado con `enddef`. 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:

```
def suma2

      2 +

enddef

4 suma2 print
```

Al ejecutar el programa anterior se imprimirá un 6.

Para que una subrutina 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.

La forma de utilizarlo es `include <nombrefichero.pmt>`. 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.pmt". Entonces podríamos hacer lo siguiente:

```
include fichsuma2.pmt

40 suma2 print
```

El resultado impreso sería 42.

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 el espacio en blanco como delimitador para trocearla.

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

subst - 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.

0 put - añade un nuevo elemento al final de una lista.

1 put - añade un nuevo elemento al principio de una lista.

set - substituye un elemento de una lista.

del - elimina un elemento de una lista.

flush - elimina todos los elementos de una lista.

head - devuleve el primer elemento de una lista.

tail - devuelve una lista con todos los elementos de otra, menos el primero.

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.
Phixmonti 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 '=' 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_1 l_2 -- l

Descripción Junta dos listas en una única lista de elementos.

Comentario Los argumentos se consumen.

Ejemplo
```
[1, 2, 3] [4, 5, 6] chain  -- [1, 2, 3, 4, 5, 6]
```

split

Definición s -- l

Descripción Trocea una cadena de caracteres cuyos componentes están separados por espacios en una lista de esos componentes.

Comentario El argumento se consume.

Ejemplo
```
"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"]*

subst

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" subst -- *["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.

Ejemplo "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"*

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' 4 set   -- ["Esta es una prueba"]
["Esto", "es", "una", "prueba"] 'a' 4 set   -- ["Esto", "es", "una", 97] (código ASCII de 'a')
```

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' 4 put   -- ["Estao es una prueba"]
["Esto", "es", "una", "prueba"] 'a' 4 put   -- ["Esto", "es", "una", 97, "prueba"] (código ASCII de 'a')
"Esto es una prueba" 'a' 1 put   -- ["aEsto es una prueba"]
"Esto es una prueba" 'a' 0 put   -- ["Esto es una pruebaa"]
```

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"] 0 del   -- ["Esto", "es", "una"]
```

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  -- []
```

head

Definición l/s -- o

Descripción Devuelve una copia del primer elemento de una lista o secuencia de caracteres.

Comentario El argumento se preserva.

Ejemplo [18,[19, [45]], [18.4, [], 29.3]] head -- *18*

tail

Definición l/s -- o

Descripción Devuelve todos los elementos de una lista/cadena de caracteres menos el primero.

Comentario El argumento se preserva.

Ejemplo [18,[19, [45]], [18.4, [], 29.3]] tail -- *[[19, [45]], [18.4, [], 29.3]]*

slice

Definición l/s n n -- o

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

Comentario Los parámetros 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"*

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 n_1 n_2 / s/l -- o

Descripción Devuelve el mayor de dos elementos o de una lista.

Comentario Los argumentos se consumen.

Ejemplo "Hola" lower *-- 111 (el código ASCII de la letra 'o')*

min

Definición n_1 n_2 / s/l -- o

Descripción Devuelve el menor de dos elementos o de una lista.

Comentario Los argumentos se consumen.

Ejemplo "Hola" lower *-- 72 (el código ASCII de la letra 'H')*

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' 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 *-- 4*

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

arcsin -calcula el ángulo con un seno dado

cos -calcula el coseno de un ángulo

arccos -calcula el ángulo con un coseno dado

tan -calcula la tangente de un ángulo

arctan -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.

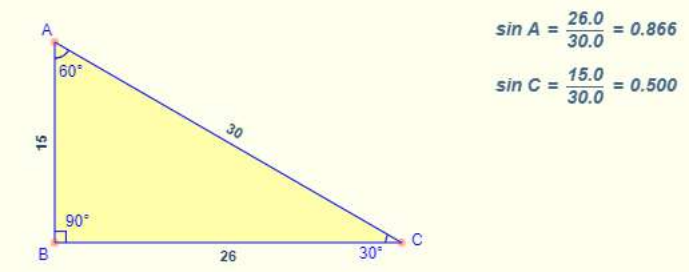
Ejemplo rand -- *0.406172*

sin

Definición n -- n

Descripción Calcula el seno de un ángulo.

Comentario El argumento se consume.



Ejemplo 9 sin -- *0.783327*

arcsin

Definición n -- n

Descripción Devuelve un ángulo para un seno dado.

Comentario El argumento se consume.

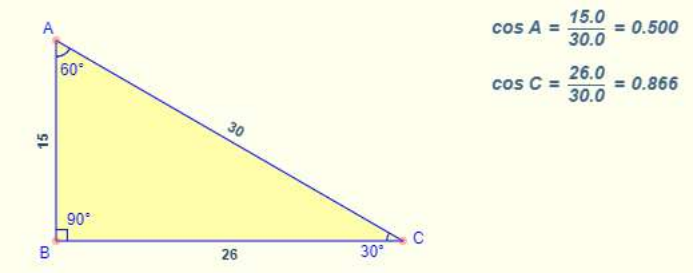
Ejemplo 1 arcsin -- *1.5708*

cos

Definición n -- n

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

Comentario El argumento se consume.



Ejemplo .5 cos -- *0.877583*

arccos

Definición n -- n

Descripción Devuelve un ángulo para un coseno dado.

Comentario El argumento se consume.

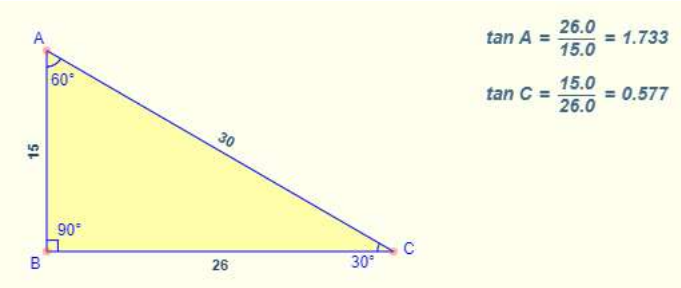
Ejemplo -1 arccos -- *3.14159*

tan

Definición n -- n

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

Comentario El argumento se consume.



Ejemplo 1 arcsin *-- 1.55741*

arctan

Definición n -- n

Descripción Devuelve un ángulo para una tangente dada.

Comentario El argumento se consume.

Ejemplo 2 arctan *-- 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.

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.

bitand	-realiza la operación AND a nivel de bits
bitor	-realiza la operación OR a nivel de bits
bitxor	-realiza la operación XOR a nivel de bits
bitnot	-realiza la operación NOT a nivel de bits
int>bit	-convierte un número en su representación en bits
bit>int	-convierte una secuencia de bits en su correspondiente número

bitand

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 bitand -- *8 (00001010 00011001 -- 00001000)*

bitor

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 bitor -- *27 (00001010 00011001 -- 00011011)*

bitxor

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 bitxor *-- 19 (00001010 00011001 -- 00010011)*

bitnot

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 bitnot *-- -201 (11001000 -- 00110111)*

int>bit

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 int>bit *-- [0, 0, 0, 1, 0, 0, 1, 1, 0, 0]*

bit>int

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] bit>int -- *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 s n --

Descripción Escribe una cadena de caracteres en el fichero indicado por el segundo parámetro.

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
quit	-termina la ejecución del programa
sleep	-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
getid	-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
argument	-apila la lista de los argumentos pasados al programa
number?	-indica si el elemento de la cima de la pila es un número
string?	-indica si el elemento de la cima de la pila es una cadena de caracteres

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]*

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 -- *[2019, 10, 25, 5, 298]*

cmd

Definición s -- n

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

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)*

quit

Definición n --

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

Comentario El argumento se consume.

Ejemplo 0 quit -- *(devuelve 0, que significa finalización correcta)*

sleep

Definición n --

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

Comentario El argumento se consume.

Ejemplo 10 sleep -- *(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.

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. Se puede usar la palabra "." (punto) como alias.

Comentario El argumento se consume.

Ejemplo
16 "Adios" 0.55 drop -- *[16, "Adios"]*
16 "Adios" 0.55 . -- *[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"]*

getid

Definición -- n

Descripción Apila el identificador de la palabra definida por el usuario que viene a continuación.

Comentario El argumento en este caso viene a continuación de esta palabra (otra palabra) y se omite su ejecución.

Ejemplo getid 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 getid suma2 exec -- *6*

argument

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.

Ejemplo Phixmonti.exe prueba.pmt "Hola mundo" -- *["prueba.pmt", "Hola mundo"]*

number?

Definición o -- o f

Descripción Indica si el elemento de la cima de la pila es un número.

Comentario El argumento se mantiene.

Ejemplo "Hola mundo" number? -- *"Hola mundo" false*

string?

Definición o -- o f

Descripción Indica si el elemento de la cima de la pila es una cadena de caracteres.

Comentario El argumento se mantiene.

Ejemplo "Hola mundo" string? -- *"Hola mundo" true*