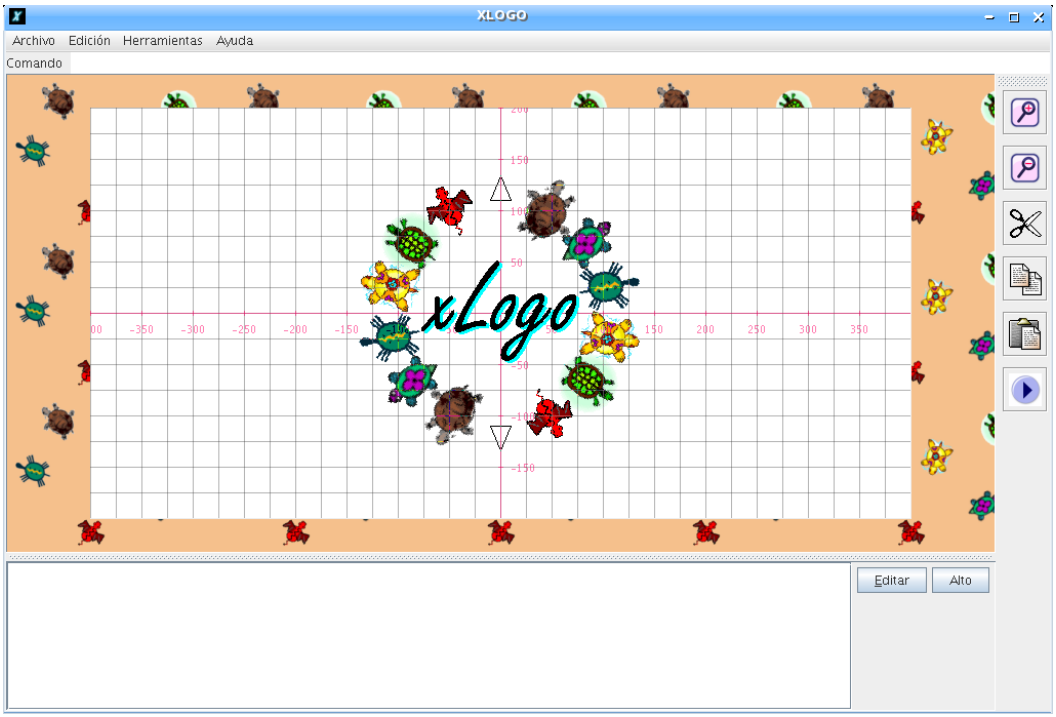
Manual del Usuario

Original en Francés: Loïc Le Coq

Traducción: Álvaro Valdés y Marcelo Duschkin

http://xlogo.tuxfamily.org



Índice general

Capítulo 1

Presentación

1.1. Introducción

Logo es un lenguaje desarrollado a finales de los años 60 por Seymour Papert. Es un lenguaje excelente para comenzar a estudiar programación, y enseña lo básico acerca de temas como bucles, condicionales, procedimientos, etc. El usuario puede mover un objeto llamado "tortuga" dentro de la pantalla, usando instrucciones (comandos) simples como "`avanza`", "`retrocede`", "`giraderecha`" y similares. Con cada movimiento, la tortuga deja un "rastro" (dibuja una línea) tras de sí, y de esta manera se crean gráficos. También es posible operar con palabras y listas.

Este estilo gráfico hace de Logo un lenguaje ideal para principiantes, y especialmente fácil para los niños.

XLogo es un intérprete Logo escrito en Java. Actualmente soporta diez idiomas (Francés, Inglés, Español, Portugués, Árabe, Esperanto, Gallego, Asturiano y Griego) y se distribuye bajo licencia GPL. Por lo tanto, este programa es libre en cuanto a libertad y gratuidad.

1.2. Java

Como hemos mencionado, XLogo está escrito en Java. Java es un lenguaje que tiene la ventaja de ser multi-plataforma; esto es, XLogo podrá ejecutarse en cualquier sistema operativo que soporte Java; tanto usando Linux como Windows, MacOS o Solaris, XLogo funcionará sin problemas.

Capítulo 2

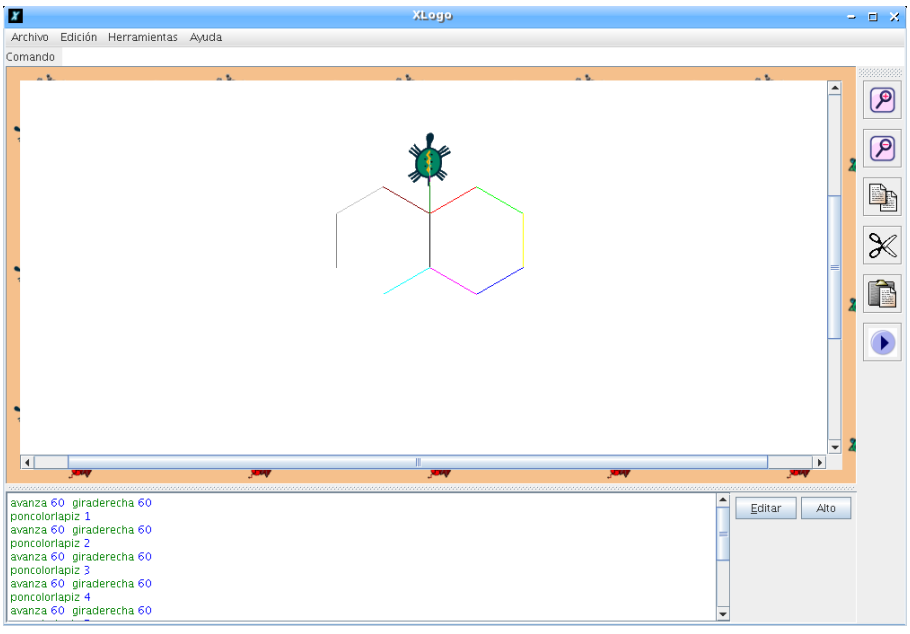
Características de la Interfaz

2.1. Primera Ejecución

La primera vez que ejecute XLogo (o si ha borrado el fichero `.xlogo` – ver sección 14.1) deberá elegir el idioma con que quiere trabajar, seleccionando la bandera correspondiente y haciendo *click* en el botón `OK`.

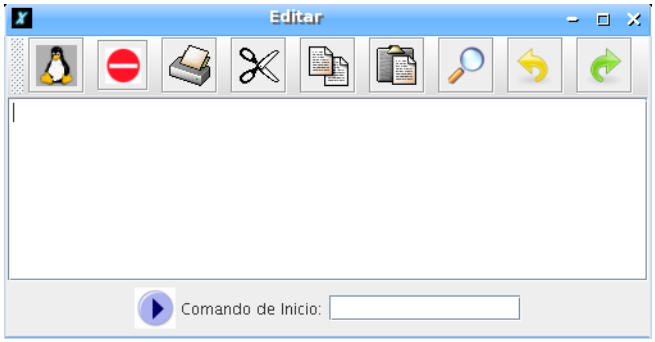
Esta elección no es definitiva; puede elegir otro idioma en cualquier momento desde las opciones de menú (sección 3.3)

2.2. La ventana principal



- En la fila superior, están las entradas típicas de menú: **Archivo**, **Edición**, **Herramientas**, **Ayuda**
- Justo debajo está la **Línea de Comando**, donde se escriben las instrucciones Logo.
- En el medio de la pantalla, está el **Área de Dibujo** (donde se mueve la tortuga).
- A la derecha del área de dibujo se encuentra una barra de herramientas vertical con las funciones *zoom* (acercar y alejar), *copiar*, *cortar*, *pegar* y *Comando de Inicio*.
- Al pie, está la ventana del **Histórico de Comandos**, que muestra todo lo ingresado y sus respuestas asociadas. Para reutilizar un comando previamente ingresado, hay dos opciones: Hacer un *click* en un comando del histórico, o usar las teclas de flecha arriba y flecha abajo del teclado (lo que es más práctico).
- A la derecha de la ventana del histórico hay dos botones: **Editar** y **Alto**.
  - **Editar** permite abrir la ventana del editor de procedimientos.
  - **Alto** interrumpe la ejecución del programa ingresado.

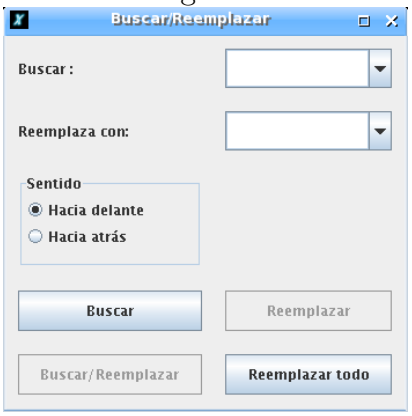
2.3. El editor de procedimientos



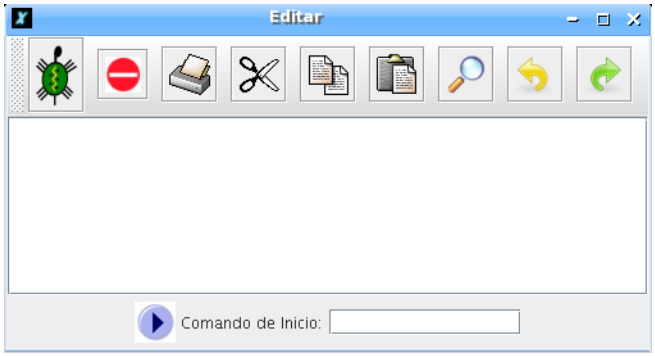
Hay cuatro maneras de abrir el editor:

- Escribir `editatodo` o `edtodo` en la **Línea de Comando**. La ventana del editor se abrirá mostrando todos los procedimientos definidos hasta ese momento.
- Si deseas editar un procedimiento en especial (o algunos), debes usar `ed` o `edita` en la línea de comandos seguido del nombre de procedimiento, o la lista con los nombres de procedimientos que deseas editar:
  `edita "nombre_procedimiento`
  o:
  `edita [proc_1 proc_2]`
- Hacer *click* en el botón **Editar**.
- Usar el atajo de teclado `Alt+E`

Estos son los diferentes botones que encontrarás en la ventana del Editor:

	Guarda en memoria los cambios hechos en el editor y cierra la ventana. Es este botón el que se debe usar cada vez que quieras aplicar los procedimientos recientemente incorporados. Atajo de teclado: `ALT+Q`.
	Cierra la ventana del editor sin guardar los últimos cambios. Ten presente que NO aparece ningún mensaje de confirmación. Asegúrate bien de que realmente no hay nada que guardar. Atajo de teclado: `ALT+C`.
	Imprime el contenido del editor.
	Copia el texto seleccionado al portapapeles. Atajo de teclado: `Control+C`.
	Corta el texto seleccionado y lo copia al portapapeles. Atajo de teclado: `Control+X`.
	Pega el contenido del portapapeles. Atajo de teclado: `Control+V`.
	Permite realizar búsquedas y reemplazos en los procedimientos. Para ello se abre la ventana siguiente: 
	Permite deshacer los últimos cambios realizados en los procedimientos.
	"Rehace" lo deshecho con el botón anterior.

Nota: Aunque aquí se representa la imagen de Tux (mascota de Linux) en el botón "*Guardar*", en realidad se muestra la tortuga activa para dar la idea de "*enviar la información a la tortuga*"; por ejemplo, si la tortuga activa es la número 3 (sección 3.3):



En la parte inferior se encuentra línea donde definir el `Comando de Inicio`, que se activa con el botón situado a la derecha del Área de Dibujo.

Al pulsar el botón, se ejecuta inmediatamente el `Comando de Inicio` sin necesidad de escribirlo en la Línea de Comandos, lo que es útil para:

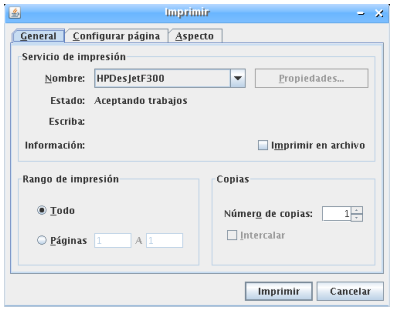
- Ahorrar tiempo mientras se desarrolla un programa
- Al enviar un programa a alguien que se inicia en LOGO, simplemente tiene que hacer "*click*" en ese botón para ejecutarlo
- ...

IMPORTANTE:

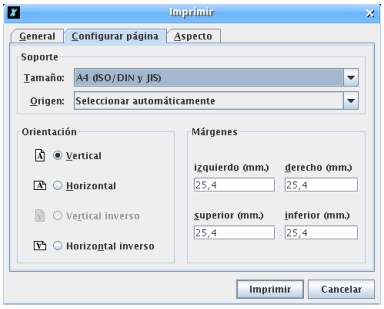
- Nota que hacer *click* en el icono de cierre () de la barra de título de la ventana del Editor, no hace nada. Solamente funcionan los dos botones principales.
- Para borrar los procedimientos que no se necesitan, usa los comandos `borra` y `borratodo` o en la barra de menús: Herramientas → Borra procedimientos.

Al hacer *click* para imprimir, aparecerá una ventana de diálogo donde podremos configurar distintas opciones de impresión:

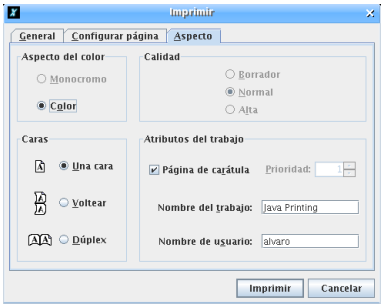
- **General**: Impresora a utilizar, Imprimir a un archivo, Rango de Impresión y Número de copias.



- **Configurar Página**: Tipo de papel, Origen del papel, Orientación de la Hoja y Márgenes



- **Aspecto**: Color (cuando disponible), Calidad, Caras y otros Atributos



2.4. Salir

Para salir simplemente seleccionamos: **Archivo** → **Salir**, o hacemos *click* en en el icono de cierre ( ). XLogo presentará una ventana de confirmación:

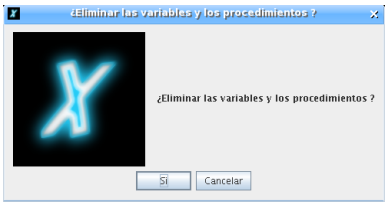
Pulsamos `Sí` y termina la ejecución.

Capítulo 3

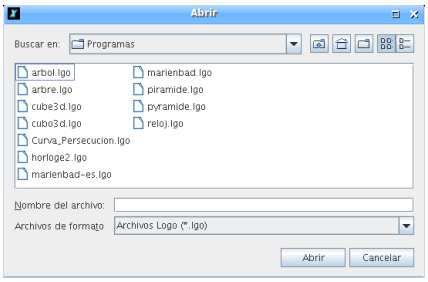
Opciones del Menú

3.1. Menú "*Archivo*"

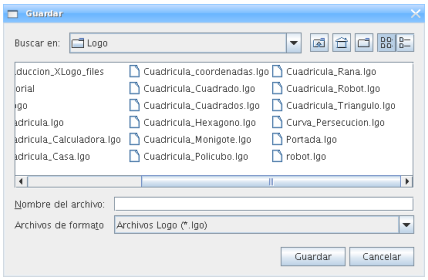
- **Archivo** → **Nuevo**: Elimina todos los procedimientos y variables definidos hasta el momento para comenzar un nuevo espacio de trabajo. Se abrirá una ventana para confirmar la eliminación de todos los procedimeintos y variables:



- **Archivo** → **Abrir**: Carga un archivo LOGO previamente guardado en disco.



- **Archivo** → **Guardar como...**: Graba un archivo (`.lgo`) de procedimientos definidos hasta ese momento en el disco, permitiendo asignarle un nombre.

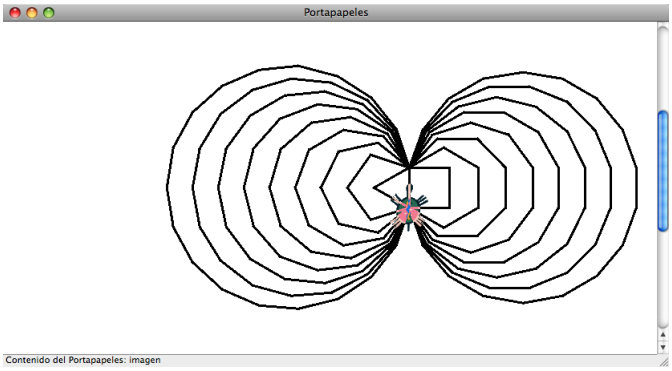


- **Archivo** → **Guardar**: Graba un archivo (`.lgo`) con los procedimientos definidos hasta ese momento en el disco. Esta opción estará deshabilitada mientras no se le haya asignado un nombre como se acaba de explicar en el punto anterior.

- **Archivo → Capturar imagen → Guardar imagen como...**: Permite guardar la imagen del área de dibujo en formato jpg o png. Para ello, puedes seleccionar previamente una parte de la imagen pulsando el botón izquierdo del ratón y arrastrando.

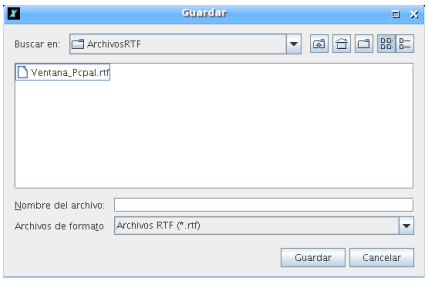
- **Archivo → Capturar imagen → Imprimir imagen**: Permite imprimir la imagen del área de dibujo. Se puede seleccionar una zona a imprimir tal como se explicó para **Guardar**.

- **Archivo → Capturar imagen → Copiar al portapapeles:** Permite enviar la imagen al portapapeles del sistema. Del mismo modo que para **Imprimir** y **Guardar**, se puede seleccionar una zona. Esta opción funciona perfectamente en entornos Windows y Mac:



  pero no así en entornos Linux (el portapapeles no tiene el mismo funcionamiento).

- **Archivo → Zona de texto → Guardar en formato RTF**: Guarda el contenido del **Histórico de Comandos** en un fichero con formato RTF (*Rich Text Format*), manteniendo los colores de los mensajes. Si no se escribe la extensión `.rtf`, se añade automáticamente.



- **Archivo → Salir**: Finaliza la ejecución de XLogo. También puede terminarse la ejecución desde la Línea de comandos con la primitiva `adiós`.

3.2. Menú "*Edición*"

- **Edición → Copiar**: copia el texto seleccionado en el portapapeles. Atajo de teclado: `Control+C`

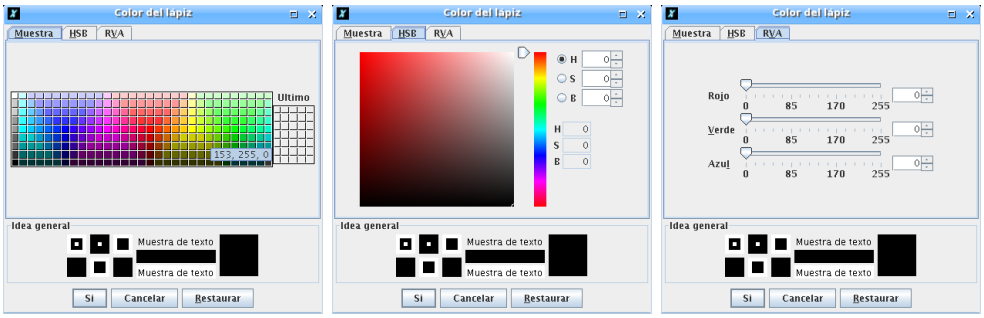
- **Edición → Cortar**: corta el texto seleccionado y lo copia en el portapapeles. Atajo de teclado: `Control+X`

- **Edición → Pegar**: pega el texto desde el portapapeles a la línea de comandos. Atajo de teclado: `Control+V`

- **Edición → Seleccionar todo**: Selecciona todo lo que se encuentra escrito en la **Línea de Comandos**.

3.3. Menú "*Herramientas*"

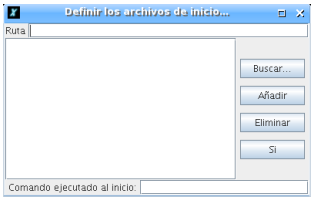
- **Herramientas → Elegir el color del lápiz**: Permite elegir el color con que la tortuga dibujará, desde la paleta de colores, mediante una definición `HSB` (*Hue*, *Saturation*, *Brightness* - Tonalidad, Saturación, Brillo) o desde una codificación `RVA` (Rojo, Verde y Azul).



También accesible con el comando `poncolorlapiz`. (Sección 5.1.2)

- **Herramientas → Elegir el color de fondo (papel)**: Pone un color como fondo de pantalla, con las mismas características que **Elegir Color del Lápiz**. También accesible con el comando `poncolorpapel`. (Sección 5.1.2)

- **Herramientas → Definir archivos de inicio**: Permite establecer la ruta a los archivos de inicio.



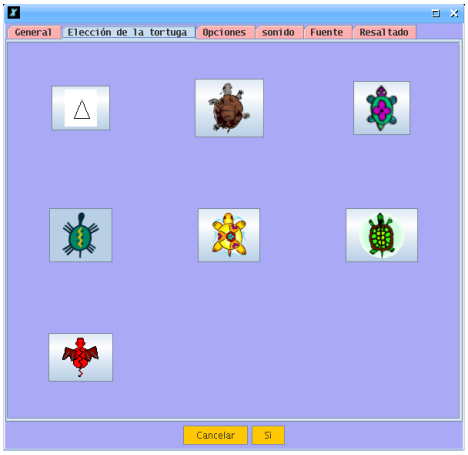
Cualquier procedimiento contenido en estos archivos `.lgo` se convertirán en "*seudo-primitivas*" del lenguaje XLogo. Pero no pueden ser modificadas por el usuario. Así se pueden definir primitivas personalizadas .

- **Herramientas → Traducir procedimientos**: Abre una caja de diálogo que permite traducir los comandos XLogo entre dos idiomas. Es muy útil, en particular, cuando se obtienen códigos Logo en inglés (de internet, por ejemplo) para expresarlos en el idioma deseado (Español, Francés, . . .)

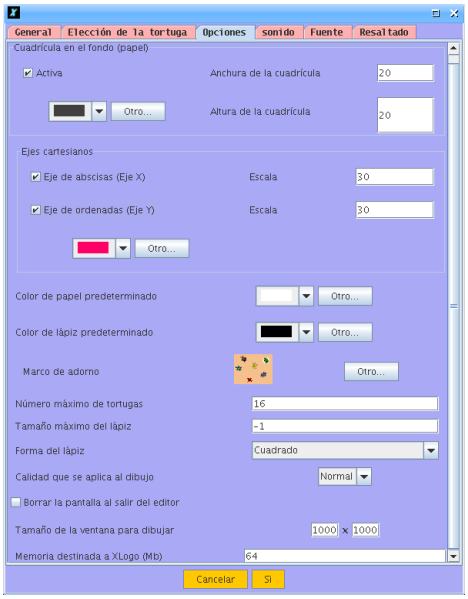
- **Herramientas → Borra procedimientos**: Abre una caja de diálogo que permite selccionar el procedimiento que se desea borrar, de una forma más sencilla que con la primitiva `borra`.

- **Herramientas → Preferencias**: Abre una caja de diálogo donde se pueden configurar varios aspectos:

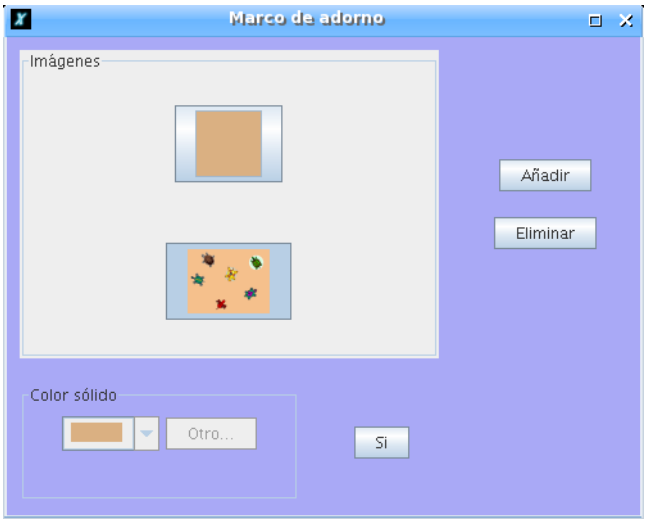
  - **General:**
    - **Idioma**: Permite elegir entre Francés, Inglés, Español, Galés, Portugués, Esperanto, Árabe y Gallego. Nota que las primitivas se adecúan a cada lenguaje.
    - **Aspecto**: Permite definir el aspecto de la ventana XLogo. Están disponibles los estilos "Windows", "Metal" y "Motif".
    - **Velocidad de la tortuga**: Si prefieres ver todos los movimientos de la tortuga, puedes reducir la velocidad con la barra deslizante.



  - **Elección de la tortuga**: Elige entre siete tortugas distintas. También accesible con el comando `ponforma` (Sección 5.1.2)
  - **Opciones:**



- **Cuadrícula en el fondo**: Establece (o elimina) una cuadrícula en el fondo del Área de dibujo, así como las medidas y el color de la misma. También accesible con las primitivas `cuadricula`, `detienecuadricula` y `poncolor-cuadricula`. (Sección 5.1.2)
- **Ejes cartesianos**: Muestra (o retira) los ejes cartesianos (X e Y) del Área de dibujo, establece su escala (separación entre marcas) y su color. También accesible con las primitivas `ejes`, `detieneejes`, `ejex`, `ejey` y `poncolorejes` (Sección 5.1.2).
- **Color de papel predeterminado**: Permite elegir el color por defecto del papel, es decir, el mostrado al iniciar XLogo.
- **Color de lápiz predeterminado**: Permite elegir el color por defecto del lápiz, es decir, el utilizado al iniciar XLogo.
- **Marco de adorno**: Permite elegir qué marco de adorno se muestra alrededor del **Área de Dibujo**, una imagen o un color sólido. Para no superar la memoria asignada, la imagen no puede ocupar más de 100 kb.



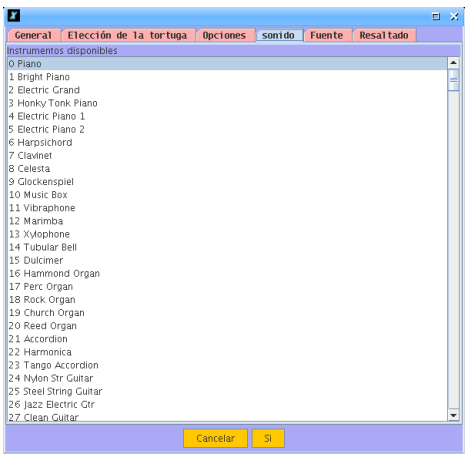
- **Número máximo de tortugas**: Para el modo multitortuga (sección 8). Por defecto 16.
- **Tamaño máximo del lápiz**: Puedes fijar un tamaño límite para el lápiz.

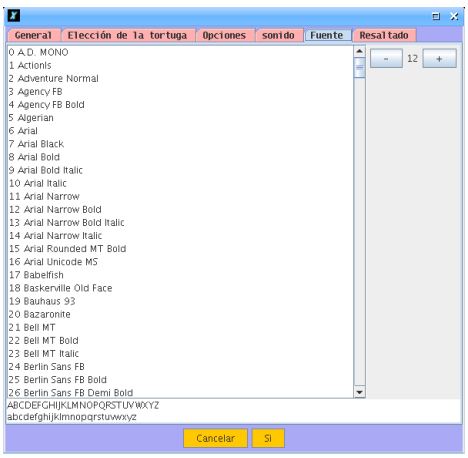
Si no se va a utilizar esta limitación, introduce el número `-1` en el recuadro asociado.

- **Forma del lápiz**: Cuadrado o Redondo
- **Tamaño de la ventana para dibujar**: Puedes establecer un tamaño personalizado para el **Área de Dibujo**. Por defecto XLogo establece un área de 1000 por 1000 pixels.
  Atención: según aumenta el tamaño de la imagen, puede ser necesario aumentar la memoria destinada a XLogo. Un mensaje de error advertirá si ocurre esto.
- **Calidad que se aplica al dibujo**: Normal, Alto o Bajo. En calidad "Alta", no se aprecia ningún efecto en particular, pero puede producirse una ralentización de la ejecución.
- **Borrar pantalla al salir del editor**: Sí o No
- **Tamaño de la ventana para dibujar**: Determina las dimensiones del Área de dibujo.
- **Memoria destinada a XLogo**: (en Mb) Por defecto, el valor asignado es 64 Mb. Puede ser necesario aumentarlo cuando el tamaño del **Área de Dibujo** sea demasiado grande. La modificación de este parámetro sólo se hará efectiva tras cerrar y reiniciar XLogo.
  Atención: no aumentar en exceso y/o sin razón este valor, ya que puede ralentizar considerablemente el sistema. Un ejemplo en el que se puede necesitar bastante memoria disponible es al trabajar en el modo 3D com diseños muy complejos.

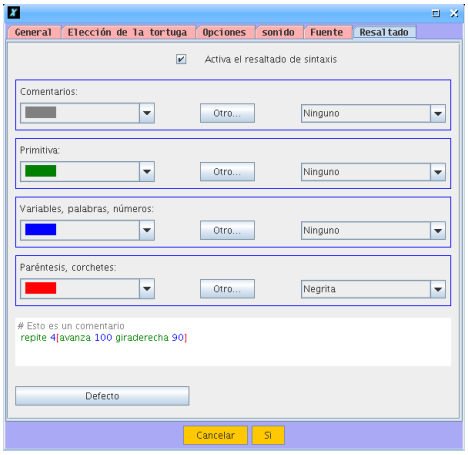
- **Sonido**: La lista de instrumentos que puede imitar la tarjeta de sonido a través de la interfaz MIDI. Puedes seleccionar un instrumento concreto en cualquier momento, mediante la primitiva `poninstrumento n`.

  Si usas Windows, esta lista puede estar vacía. Esto se debe a que la versión de Java para Windows no incluye los *bancos de sonido*, y deben instalarse manualmente. Echa un vistazo a la sección 15.1.





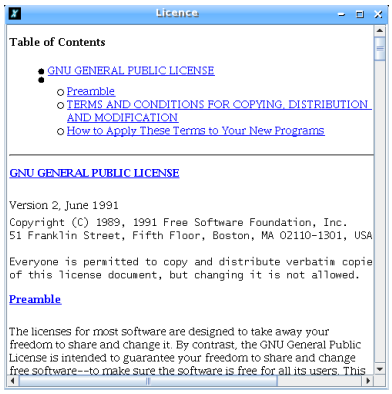
- **Fuente**: Elige el tipo de letra y su tamaño

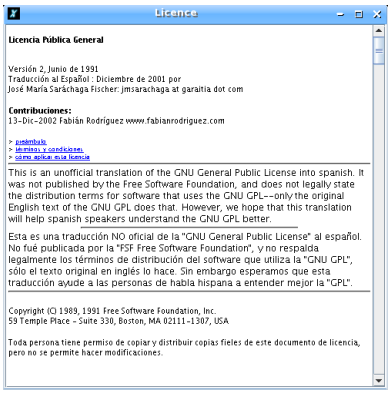


- **Resaltado**: Elige los colores que se utilizarán en el resaltado de primitivas, palabras, variables números, paréntesis y corchetes.

3.4. Menú "*Ayuda*"

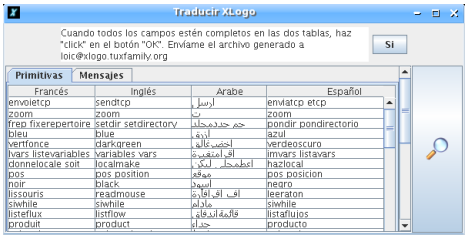
- **Ayuda → Licencia**: Muestra la licencia original GPL (en Inglés) bajo la cual es distribuido este programa.



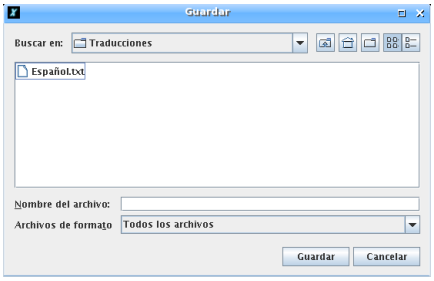


- **Ayuda → Traducción de la Licencia**: Refiere a una traducción al español de la licencia GPL, sin efectos legales, sólo como referencia para entender la versión en Inglés.

- **Ayuda → Traducir XLogo**: Abre una ventana para añadir y/o corregir traducciones.



Desde ella pueden crearse y/o modificarse tanto las primitivas como los mensajes. Una vez creados/modificados, haz *click* **fuera** de la celda que acabas de escribir y pulsa el botón `Si`. Se abrirá una ventana donde debes elegir el fichero de texto donde guardar los cambios y que debes enviar a `loic@xlogo.tuxfamily.org`



- **Ayuda → Acerca de...**: Lo de siempre ... y ¡¡`xlogo.tuxfamily.org` para guardar en favoritos!! o:)

Capítulo 4

Convenciones adoptadas para XLogo

Esta sección define aspectos claves acerca del lenguaje Logo en general, y sobre XLogo en particular.

4.1. Comandos y su interpretación

El lenguaje Logo permite que ciertos eventos sean iniciados por comandos internos. Estos comandos son llamados *primitivas*. Cada primitiva puede tener un cierto número de parámetros que son llamados *argumentos*. Por ejemplo, la primitiva `bp`, que borra la pantalla, no lleva argumentos, mientras que la primitiva `suma` tiene dos argumentos.

`escribe suma 2 3` devuelve `5`.

Los argumentos en Logo son de tres tipos:

- **Números**: Algunas primitivas esperan números como su argumento: `av 100` es un ejemplo.
- **Palabras**: Las palabras se escriben precedidas por `"`. Un ejemplo de una primitiva que tiene una palabra como argumento es `escribe`.

  `escribe "hola` devuelve `hola`

  Nota que si olvidas el `"`, el intérprete devuelve un mensaje de error. En efecto, `escribe` esperaba ese argumento, pero para el intérprete, `hola` no representa nada, ya que no fue definido como número, ni palabra, ni lista, ni procedimiento.
- **Listas**: Se definen encerrándolas entre corchetes.

Los números son tratados a veces como un valor (por ej: `av 100`), o bien como una palabra (por ejemplo: `escribe vacio? 12` devuelve `falso`).

Algunas primitivas tienen una forma general, esto es, pueden ser utilizadas con números o *argumentos opcionales*. Estas primitivas son:

```
escribe   suma   producto     o
   y      lista    frase    palabra
```

Para que el intérprete las considere en su forma general, tenemos que escribir las órdenes entre paréntesis. Observa los ejemplos:

```
escribe (suma 1 2 3 4 5)
```

devuelve:

```
15
```

También:

```
escribe (lista [a b] 1 [c d])
```

devuelve:

```
[a b] 1 [c d]
```

y

```
si (y 1=1 2=2 8=5+3) [avanza 100 giraderecha 90]
```

4.2. Procedimientos

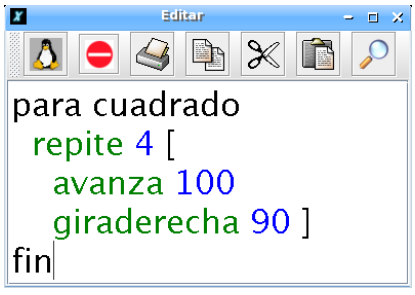
Además de las primitivas, puedes definir tus propios comandos. Estos son llamados *procedimientos*. Los procedimientos son iniciados por la palabra `para` y concluyen con la palabra `fin`. También pueden crearse usando el editor interno de procedimientos XLogo. Veamos un pequeño ejemplo:

```
para cuadrado
  repite 4 [
    avanza 100
    giraderecha 90 ]
fin
```

El proceso para crear el procedimiento es el siguiente:

1. Escribir en la **Línea de Comando**: `para cuadrado` y pulsar `[Enter]`, escribir `ed` y `[Enter]` o hacer *click* con el ratón en el botón `Editar`

2. Se mostrará la **Ventana del Editor**, donde completamos todo el procedimiento



3. Pulsar  con el ratón, o hacer `Alt+Q`

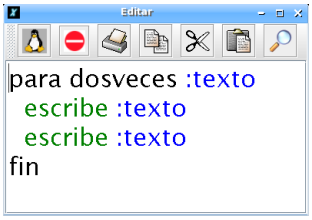
4. En la ventana del **Histórico de Comandos**, debe aparecer el mensaje:

   `Acaba de definir cuadrado` El intérprete XLogo no detecta los posibles errores en este momento, sino cuando se utilice el procedimiento por primera vez.

5. Desde ese momento, puede invocarse la orden `cuadrado` en la **Línea de Comandos**

Los procedimientos también pueden aprovechar las ventajas de los argumentos. Para hacer esto, se usan *variables*. Una variable es una palabra (un nombre) al que se le puede asignar un valor.

Ejemplo:



Probando el ejemplo:

```
dosveces [1 2 3]
```

devuelve

```
1 2 3
1 2 3
```

Al final del manual se incluyen varios ejemplos de procedimientos.

4.3. El caracter especial \

El caracter \ (barra invertida o *backslash*) permite que las "palabras" (sección 4.1) contengan espacios o saltos de línea.

- `\n` produce un salto de línea
- \␣ produce un espacio entre palabras (␣ representa un espacio en blanco)

Ejemplos:

```
es "xlogo\ xlogo     produce    xlogo xlogo
es "xlogo\nxlogo     produce    xlogo
                                xlogo
```

Esto tiene implicaciones a la hora de obtener el caracter \ en la **Línea de Comandos**: se debe teclear `\\`. Todo caracter \ es ignorado. Este aviso es importante en particular para la gestión de archivos (sección 5.7). Para establecer como directorio de trabajo `c:\Mis Documentos` se debe escribir:

```
pondirectorio "c:\\Mis\ Documentos
```

Nota el uso de \␣ para indicar el espacio entre `Mis` y `Documentos`. Si se omite el doble *backslash*, la ruta definida sería interpretada como:

```
c:Mis Documentos
```

y el intérprete mostrará un mensaje de error.

4.4. Mayúsculas y minúsculas

XLogo no distingue entre mayúsculas y minúsculas en el caso de nombres de procedimientos y primitivas. Así, en el procedimiento `cuadrado` como fue definido antes, si escribes `CUADRADO` o `cuAdRAdO`, el intérprete de comandos interpretará y ejecutará correctamente `cuadrado`. Por otro lado, XLogo sí respeta mayúsculas y minúsculas en listas y palabras:

```
escribe "Hola
```

proporciona

```
Hola
```

(la H inicial se mantuvo)

4.5. Operadores y sintaxis

Hay dos maneras para escribir ciertos comandos. Por ejemplo, para sumar 4 y 7, puedes usar la primitiva `suma` que espera dos argumentos:

```
suma 4 7
```

o puedes usar el operador `+`:

```
4 + 7
```

Ambos tienen el mismo efecto. Esta tabla muestra la relación entre operadores y primitivas:

4.5.1. Operadores aritméticos

suma	diferencia	producto	división
+	-	*	/

4.5.2. Operadores lógicos

o	y	iguales?
|	&	=

Para comparaciones numéricas, disponemos de cuatro operadores sin primitiva asociada:

- El operador “menor”: `<`
- El operador “mayor”: `>`

Por analogía con otros lenguajes, XLogo incorpora otros dos:

- El operador “menor o igual”: `<=`
- El operador “mayor o igual”: `>=`

si bien es evidente que no serían estrictamente necesarios:

- `a <= b` es equivalente a `no (a >b)`
- `a >= b` puede sustituirse por `no (a <b)`

Nota aclarativa: Los operadores `|` y `&` son específicos de XLogo. No se encuentran en otras versiones tradicionales de Logo. veamos algunos ejemplos de su uso:

```
escribe 3+4 = 7-1        devuelve    falso
escribe 3=4 | 7<=49/7    devuelve    cierto
escribe 3=4 & 7=49/7     devuelve    falso
```

4.6. Las tildes

Desde la versión 0.9.92 las primitivas en español XLogo admiten tildes. Tratándose de un software para uso educativo, es importante que la ortografía sea la adecuada.

Para la acentuación de las primitivas se siguen las normas ortográficas habituales, especialmente en aquellas primitivas compuestas por varias palabras. Por ejemplo:

- `poncolorlápiz`. La palabra `lápiz` lleva tilde y la mantiene al formar parte de la primitiva, ya que la acentuación de esta recae sobre la "`a`"
- `leelineaflujo`. Aunque `línea` lleva tilde al ser esdrújula, al pronunciar la primitiva completa, observamos que es una palabra llana (el acento se encuentra en la "`u`" de `flujo`), así que no se le asigna tilde.

  Sí que lleva tilde en `definelínea` y `finlínea`, por el mismo motivo explicado antes para `lápiz`
- Se procede del mismo modo en las formas cortas de las primitivas. Las formas cortas de `definepolígono` y `finpolígono` son, respectivamente, `defpoli` y `finpoli`. Escuchando a los alumnos pronunciarlas, se optó por considerarlas llanas y sin tilde.

Dicho lo anterior, debemos tener una idea sobre la distinta codificación de caracteres que usan los Sistemas Operativos.

La codificación de caracteres es el método que convierte un carácter de un lenguaje natural en un código numérico. Es muy habitual (más de lo que sería deseable) que los sistemas operativos (Windows, Linux, MacOS, Solaris, ...) usen distintos sistemas de codificación. Existen varias normas: ASCII, Unicode, UTF, ISO, ..., y eso afecta negativamente a los caracteres especiales del español:

- Vocales acentuadas: á, é, í, ó, ú
- Letras eñe y "cedilla": ñ y ç
- Apertura de exclamación, interrogación, ...: ¿, ¡ , ...

Si tienes intención de compartir tus programas por Internet, intenta evitar estos caracteres y utiliza primitivas sin tilde. Si estás en una aula, recomendamos el uso acentuado de las mismas.

Capítulo 5

Listado de primitivas

Como decíamos, la tortuga se controla por medio de comandos internos llamados *primitivas*. Las siguientes secciones definen estas primitivas:

5.1. Movimientos de la tortuga; poner lápiz y colores

5.1.1. Movimientos

Esta primera tabla define las primitivas que gobiernan el movimiento de la tortuga, y sólo necesitan un argumento:

Primitivas	Argumentos	Uso
`avanza`, `av`	`n: número de pasos`	Mueve la tortuga hacia adelante `n` pasos en la dirección que actualmente está mirando.
`retrocede`, `re`	`n: número de pasos`	Mueve la tortuga hacia atrás `n` pasos en la dirección que actualmente está mirando.
`giraderecha`, `gd`	`n: ángulo`	Gira la tortuga `n` grados hacia la derecha de la dirección que actualmente está mirando.
`giraizquierda`, `gi`	`n: ángulo`	Gira la tortuga `n` grados hacia la izquierda de la dirección que actualmente está mirando.

En esta segunda tabla el movimiento se controla mediante **coordenadas** en la pantalla. Para ver mejor dichas coordenadas, se dispone de las primitivas `cuadrícula`, que muestra una cuadrícula en pantalla de las dimensiones deseadas, y `ejes`, que muestra los ejes cartesianos con las correspondientes etiquetas:

Primitivas	Argumentos	Uso
`cuadrícula`	`a b: números`	Dibuja una cuadrícula en el **Área de dibujo** de dimensiones `a` x `b` y borra la pantalla
`borracuadrícula`	`no`	Quita la cuadrícula del **Área de dibujo** y borra la pantalla
`poncolorcuadrícula`, `pcc`	`primitiva`, `lista` o `numero`	Establece el color de la cuadrícula del **Área de dibujo**
`colorcuadrícula`	`no`	Devuelve el color actual de la cuadrícula.

Primitivas	Argumentos	Uso
`ejes`	`a: número`	Dibuja los ejes cartesianos (X e Y) de escala (separación entre marcas) `a`, con las etiquetas correspondientes.
`ejex`	`a: número`	Dibuja el eje de abscisas (eje X) de escala (separación entre marcas) `a`, con las etiquetas correspondientes.
`ejey`	`a: número`	Dibuja el eje de ordenadas (eje Y) de escala (separación entre marcas) `a`, con las etiquetas correspondientes.
`borraejes`	`no`	Quita los ejes del **Área de dibujo** y borra la pantalla
`poncolorejes pce`	`primitiva`, `lista` o `numero`	Establece el color de los ejes en el **Área de dibujo**
`colorejes`	`no`	Devuelve el color actual de los ejes.
`centro`	`no`	Lleva la tortuga a la posición original, es decir coordenadas `[0 0]` con rumbo `0`.
`posición`, `pos`	`no`	Devuelve las coordenadas X e Y de la posición actual de la tortuga.
`ponposición`, `ponpos`	`[x y]: lista de dos números`	Mueve la tortuga a las coordenadas especificadas por los dos números en la lista (`x` es la abscisa, `y` la ordenada).
`ponx`	`x: eje x`	Mueve la tortuga horizontalmente hasta el punto de abscisa `x`
`pony`	`y: eje y`	Mueve la tortuga verticalmente hasta el punto de ordenada `y`
`ponxy`	`x y`: coordenadas `x` e `y`	Idéntico a `ponpos [x y]` `x` e `y` son números, no una lista.
`punto`	`a: lista`	El punto definido por las coordenadas de la lista se resaltará con el color del lápiz.

Esta tercera tabla muestra las primitivas que controlan el rumbo, dirección en grados respecto de la vertical y mirando hacia arriba:

Primitivas	Argumentos	Uso
`rumbo`	`no`	Devuelve el rumbo o el ángulo de la tortuga.
`ponrumbo`, `ponr`	`n: rumbo`	Orienta la tortuga en la dirección especificada. `0` corresponde a mirar hacia arriba verticalmente.
`hacia`	`a: lista`	La lista debe contener dos números que representen coordenadas. Devuelve el rumbo que la tortuga deberá seguir hacia el punto definido por las coordenadas.
`distancia`	`a: lista`	La lista debe contener dos números que representen coordenadas. Devuelve el número de pasos desde la actual posición y el punto definido por las coordenadas.

5.1.2. Propiedades

Esta tabla descrine las primitivas que permiten ajustar las propiedades de la tortuga. Por ejemplo, ¿estará visible en la pantalla? ¿Con qué color dibujará cuando se mueva?

Primitivas	Argumentos	Uso
`muestratortuga`, `mt`	`no`	Hace que la tortuga se vea en pantalla.
`ocultatortuga`, `ot`	`no`	Hace invisible a la tortuga.
`bajalápiz`, `bl`	`no`	La tortuga dibujará una línea cuando se mueva.
`subelápiz`, `sl`	`no`	La tortuga no dibujará cuando se mueva.
`goma`, `go`	`no`	La tortuga borrará toda traza que encuentre.
`inviertelápiz`, `ila`	`no`	Pone la tortuga en "modo inverso", y lápiz abajo.
`ponlápiz`, `pla`	`no`	Pone la tortuga en el modo normal de dibujo y lápiz abajo.
`poncolorlápiz`, `poncl`	`a: número, primitiva o lista [r v a]`	Cambia el color del lápiz. La especificación del color se detalla en la sección 5.1.4
`pongrosor`	`n: número`	Define el grosor del trazo del lápiz (en pixels). Por defecto es 1. La forma es cuadrada.
`colorlápiz`, `cl`	`a: lista`	Devuelve el color actual del lápiz.
`encuentracolor`, `ec`	`a: lista`	Devuelve el color del punto definido por las coordenadas.
`grosorlápiz`, `gl`	`no`	Devuelve el grosor del lápiz.
`ponformalápiz`, `pfl`	`n: 0 ó 1`	Fija la forma del lápiz: `pfl 0`: cuadrada; `pfl 1`: ovalada.
`formalápiz`, `fl`	`no`	Devuelve la forma del lápiz.
`ponforma`, `pforma`	`n: número`	Puedes elegir tu tortuga preferida en la segunda etiqueta del menú **Herramientas → Preferencias**, pero también es posible con `ponforma`. El número `n` puede ir de 0 a 6. (0 es la forma triangular del Logo tradicional).
`forma`	`no`	Devuelve un número que representa la forma actual de la tortuga.

El control del Área de dibujo se realiza con las primitivas siguientes:

Primitivas	Argumentos	Uso
`poncolorpapel`, `poncp`	`a: número, primitiva o lista [r v a]`	Cambia el color del papel (fondo). La especificación del color se detalla en la sección 5.1.4

Primitivas	Argumentos	Uso
`colorpapel`	`a: lista`	Devuelve el color actual del "papel" (fondo, área de dibujo).
`poncalidaddibujo`, `pcd`	`n: 0, 1 ó 2`	Fija la calidad del dibujo: `pcd 0`: normal; `pcd 1`: alta; `pcd 2`: baja;
`calidaddibujo`, `cdib`	`no`	Devuelve la calidad del dibujo
`tamañopantalla`, `tpant`	`no`	Devuelve una lista que contiene el tamaño de la pantalla
`pontamañopantalla` `ptp`	`a: lista`	Fija el tamaño de la pantalla. Ejemplo: `ptp [1000 1000]`
`modoventana`	`no`	La tortuga puede salir del área de dibujo (pero no dibujará nada).
`modovuelta`	`no`	Si la tortuga sale del área de dibujo, vuelve a aparecer en el lado opuesto
`modojaula`	`no`	La tortuga queda confinada al área de dibujo. Si intenta salir, aparecerá un mensaje de error avisando cuántos pasos faltan para el punto de salida.
`tamañoventana`, `tv`, `esquinasventana`	`no`	Devuelve una lista con cuatro elementos, las coordenadas de la esquina superior izquierda y de la esquina inferior derecha. Por ejemplo, si devuelve `[-200 200 400 -300]`, significa que las coordenadas de la esquina superior izquierda son `(-200,200)` y las de la esquina inferior derecha `(400,-300)`
`zoom`	`a: número`	Acerca o aleja el Área de dibujo. En concreto, el valor de `a` es el factor de escala respecto a la imagen original: `(a>1)` acerca el Área de dibujo; `(0<a<1)` aleja el Área de dibujo.
`borrapantalla`, `bp`	`no`	Vacía el área de dibujo, situando a la tortuga en el centro de la pantalla.
`limpia`	`no`	Vacía el área de dibujo, dejando a la tortuga en el lugar donde estaba tras la ejecución anterior.

Finalmente, las primitivas que controlan la escritura en pantalla, los mensajes al usuario y simplifican determinados dibujos:

Primitivas	Argumentos	Uso
`rotula`	`a: palabra o lista`	Dibuja la palabra o lista especificada, en la posición actual, y en la dirección que está mirando.
`largoetiqueta`	`a: lista`	Devuelve, en píxels, la longitud que tendrá en pantalla la lista.
`ponfuente`, `pf`	`n: número`	Cuando se escribe con la primitiva `rotula`, modifica el tamaño de la tipografía. Por defecto, el tamaño es 12.

Primitivas	Argumentos	Uso
`fuente`	`no`	Devuelve el tamaño de la tipografía cuando se escribe en pantalla con la primitiva rotula.
`mensaje`, `msj`	`a: lista`	Muestra una caja de diálogo con el mensaje que está en la lista. El programa se detiene hasta que el usuario hace un *click* en el botón "Aceptar"
`círculo`	`n: radio`	Dibuja una circunferencia de radio `n` alrededor de la tortuga
`arco`	`n: radio` `a b: ángulos`	Dibuja un arco de circunferencia de radio `n` alrededor de la tortuga, comprendido entre los ángulos `a` y `b`, midiendo desde el rumbo de la tortuga.

La primitiva `largoetiqueta` permite saber si al escribir en pantalla con `rotula` tienes suficiente espacio.

Ejemplo:

`largoetiqueta [Hola, ¿cómo estás?]` devuelve, en píxels la longitud en pantalla de la frase *Hola, ¿cómo estás?*

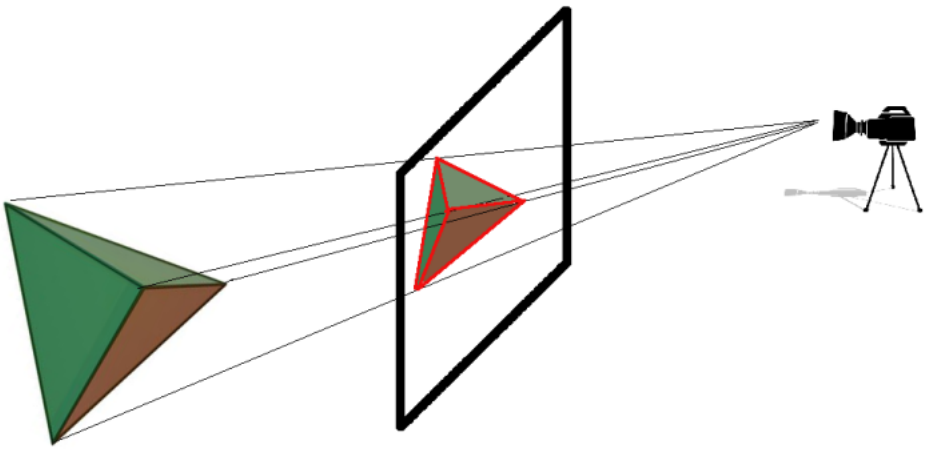
5.1.3. La tortuga en Tres Dimensiones

Desde la versión 0.9.92, nuestra tortuga puede dejar el plano para trasladarse a un espacio en tres dimensiones (3D). Para cambiar a esta modalidad, Usaremos la primitiva `perspectiva`. ¡Bienvenido a un mundo en 3D!

Para recuperar el modo bidimensional (2D), debemos indicarle que vuelva a uno de los modos "planos": `modojaula`, `modoventana` o `modovuelta`.

La proyección en perspectiva

Para representar un espacio 3D en un plano 2D, Xlogo utiliza una proyección en perspectiva. Es equivalente a tener una cámara grabando la escena en 3D, y mostrando en la pantalla la imagen de la proyección. Veamos un esquema gráfico para explicarlo mejor:



Disponemos de primitivas para fijar la posición de la cámara, mientras que la pantalla de proyección se encuentra en el punto medio entre la cámara y el objeto.

Entender la orientación en el mundo tridimensional

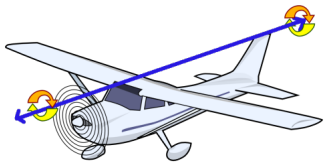
En el plano, la tortuga la orientación se define únicamente por su rumbo. Sin embargo, en el mundo tridimensional la orientación de la tortuga necesita de tres ángulos. Si usamos la orientación por defecto de la tortuga en 3D (en el plano `XY` mirando hacia el semieje `Y` positivo):

Balanceo: la rotación en torno al eje `OY`

Cabeceo: la rotación según el eje `OX`

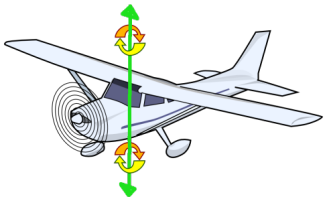
Rumbo: la rotación según el eje `OZ`

De hecho, para moverse en el mundo tridimiensional, la tortuga se comportará de modo muy similar a un avión. De nuevo, ilustremos con una imagen los 3 ángulos:



Balanceo

Cabeceo



Rumbo

Parece bastante complicado la primera vez que se estudia, pero veremos que muchas cosas son similares a los movimientos en el plano bidimensional. Estas son las primitivas básicas para moverse en el mundo 3D:

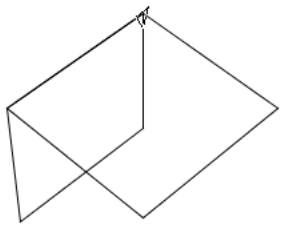
- `avanza`, `retrocede`, `av`, `re`: Idénticas al mundo 2D
- `giraderecha`, `giraizquierda`, `gd`, `gi`: Idénticas al mundo 2D, producen una rotación alrededor del eje transversal y vertical de la tortuga
- `balanceaderecha`, `bd`: la tortuga gira `n` grados a la derecha respecto a su eje longitudinal
- `balanceaizquierda`, `bi`: la tortuga gira `n` grados a la izquierda respecto a su eje longitudinal
- `cabeceaarriba`, `subenariz`, `sn`: La tortuga "sube el morro" `n` grados respecto a su eje transversal y horizontal
- `cabeceaabajo`, `bajanariz`, `bn`: La tortuga "baja el morro" `n` grados respecto a su eje transversal y horizontal

En el plano bidimensional, para dibujar un cuadrado de `200` pasos de tortuga, escribimos:

```
repite 4 [ avanza 200 giraderecha 90 ]
```

Estas órdenes siguen existiendo el mundo 3D, y el cuadrado puede dibujarse perfectamente en modo `perspectiva`.

Si la tortuga baja "el morro" 90 grados, podemos dibujar otro cuadrado, y obtenemos:



```
borrapantalla
repite 4 [ avanza 200 giraderecha 90 ]
bajanariz 90
repite 4 [ avanza 200 giraderecha 90 ]
```

Puedes (debes) probar otros ejemplos para entender perfectamente la orientación de la tortuga y el uso de los ángulos y ¡convertirte en un experto!

También debes entender que las tres primitivas que controlan la rotación en 3D están relacionadas entre sí; por ejemplo, al ejecutar:

```
borrapantalla
balanceaizquierda 90 subenariz 90 balanceaderecha 90
```

El movimiento de la tortuga es equivalente a:

```
giraizquierda 90
```

(Puedes probar con tu mano si no lo entiendes bien)

Primitivas disponibles tanto en 2D como 3D

Las siguientes primitivas están disponibles tanto en el plano como en el mundo 3D. La única diferencia son los argumentos admitidos por las primitivas.

Estas precisan de los mismos argumentos que en el plano:

`círculo` `arco` `centro`
`ponx` `pony` `rumbo`
`ponrumbo` `rotula` `largoetiqueta`

Las siguientes primitivas siguen esperando una lista como argumento, pero ahora debe contener **tres** argumentos, correspondientes a las tres coordenadas de un punto en el espacio: `[x y z]`.

`hacia` `distancia` `pos`, `posición`
`ponpos`, `ponposición` `punto`

Primitivas sólo disponibles en 3D

- `ponxyz` Esta primitiva mueve a la tortuga al punto elegido. Esta primitiva espera tres argumentos que representan las coordenadas del punto.

  `ponxyz` es muy similar a `ponposición`, pero las coordenadas no están escritos en una lista.

  Ejemplo, `ponxyz -100 200 50` traslada a la tortuga hasta el punto `x = -100`; `y = 200`; `z = 50`

- `ponz` Esta primitiva mueve a la tortuga al punto de "altura" o "profundidad" (desconozco si el término *applikate* usado en Alemania tiene traducción al castellano más allá de **tercera coordenada**) dada. `ponz` recibe un número como argumento, de modo idéntico a `ponx` y `pony`

- `ponorientación` Fija la orientación de la tortuga. Esta primitiva espera una lista que contiene tres números: `[balanceo cabeceo rumbo]`

  Ejemplo: `ponorientación [100 0 58]`: la tortuga tendrá balanceo: 100 grados, cabeceo: 0 grados y rumbo: 58 grados.

  Por supuesto, el orden de los números es importante. Si, por ejemplo, el valor de la orientación es `[100 20 90]`, esto significa que si quieres esa misma orientación partiendo del origen (después de un `borrapantalla`, por ejemplo) deberás escribir la siguiente secuencia:

  ```
  cabeceaderecha 100
  subenariz 20
  giraderecha 90
  ```

  Si en esta instrucción cambiamos el orden, no obtendremos la orientación deseada.

- `orientación` Devuelve la orientación de la tortuga en una lista que contiene: `[balanceo cabeceo rumbo]`

- `ponbalanceo` La tortuga gira en torno a su eje longitudinal y adquiere el ángulo de balanceo elegido.

- `balanceo` Devuelve el valor actual del balanceo

- `ponbalanceo` La tortuga gira en torno a su eje transversal, y se orienta con el ángulo de cabeceo indicado.

- `balanceo` Devuelve el valor actual del cabeceo

Visor 3D

XLogo incluye un visor 3D que permite visualizar los dibujos realizados en tres dimensiones. Este módulo usa las librerías de Java3D, por lo tanto es necesario tener instalado todo el Java3D.

Las reglas a tener en cuenta para utilizar el Visor 3D son:

Al crear una figura geométrica sobre el Área de Dibujo, hay que indicar al Visor 3D qué formas desea grabar para una futura visualización. Es posible grabar polígonos (superficies), líneas, puntos o texto. Para utilizar esta función, las primitivas son:

- `empiezapolígono`, `definepolígono`, `defpoli`: Los movimientos de la tortuga posteriores a esta llamada se guardan para crear un polígono.

- `finpolígono`, `finpoli`: Desde la ejecución de `definepolígono`, la tortuga habrá pasado por varios vértices. Este polígono se habrá "registrado" y su color se definirá en función del color de todos sus vértices.

  Esta primitiva finaliza el polígono.

- `empiezalínea`, `definelínea`, `deflínea`: Los movimientos de la tortuga posteriores a esta llamada se guardan para crear un quebrado, es decir, una línea con varios vértices que no tiene por qué ser cerrada.

- `finlínea`: Desde la ejecución de `definelínea`, la tortuga habrá pasado por varios vértices. Se guardará esta línea y su color se definirá en función del color de todos sus vértices.

  Esta primitiva finaliza la definición del quebrado

- `empiezapunto`, `definepunto`, `defpto`: Los movimientos posteriores de la tortuga definen los vérices de un quebrado cuyos vértices se guardan para crear un conjunto de puntos.

- `finpunto`, `finpto`: Esta primitiva finaliza la definición del conjunto de puntos.

- `empiezatexto`, `definetexto`, `deftxt`: Cada vez que el usuario muestre un texto sobre el Área de Dibujo con la primitiva `rotula`, se almacenará y luego será representada por el visor 3D.

- `fintexto`, `fintxt`: Esta primitiva la grabación de texto.

- `vista3d`, `vistapolígono` Inicia el visor 3D, todos los objetos guardados se dibujan en una nueva ventana.

  Disponemos de controles para mover la "cámara" que muestra la escena:

  - Para hacer rotar la imagen haciendo *click* con el botón izquierdo del ratón y arrastrando.
  - Para desplazar la imagen haciendo *click* con el botón derecho del ratón y arrastrando.
  - Para hacer *zoom* sobre la escena, usaremos la rueda del ratón

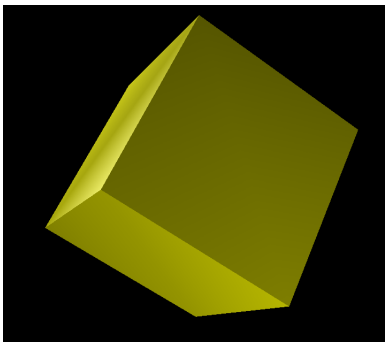
Dibujando un cubo

Todas las caras miden 400 pasos de tortuga. El programa es:

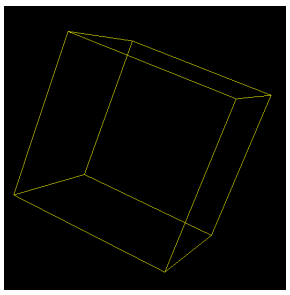
```
para cuadrado
# Grabamos los vértices del cuadrado
  empiezapolígono
  repite 4 [ avanza 400 giraderecha 90 ]
  finpolígono
fin

para cubosimple
# Cubo Amarillo
  borrapantalla perspectiva
  poncolorlápiz amarillo
# Caras laterales
  repite 4
    [ cuadrado subelápiz
      giraderecha 90 avanza 400 giraizquierda 90
```

```
      balanceaderecha 90 bajalápiz ]
# Parte inferior
  bajanariz 90 cuadrado subenariz 90
# Cara Superior
  avanza 400 bajanariz 90 cuadrado
# Visualización
  vista3d
fin
```

Estamos listos a ejecutar el comando: `cubosimple`:



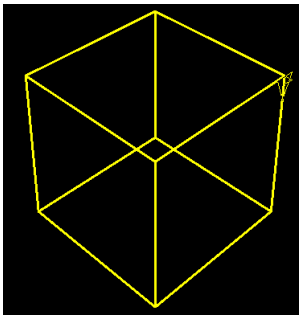
Al sustituir en el procedimiento `cuadrado`, `empiezapolígono` por `empiezalínea`, y `finpoligono` por `finlínea`:



Si hubiéramos usado `empiezapunto` y `finpunto` en lugar de `empiezalínea` y `finlínea`, deberíamos ver en la pantalla sólo los ocho vértices del cubo.

Estas primitivas son muy útiles para mostrar el conjunto de puntos en el espacio 3D.

En todos los casos, en el Área de Dibujo se muestran las aristas del cubo que luego se verá "macizo" con el Visor:



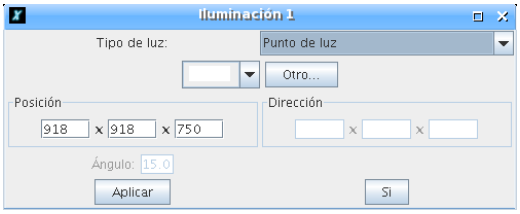
Efectos de luz y niebla

Desde la versión 0.9.93 se pueden añadir efectos artísticos a las imágenes generadas en el Visor. Estos pueden ser efectos de luz y de niebla, y se accede a ellos con los botones presentes en el visor 3D.

Efectos de luz

Se pueden utilizar cuatro tipos de luz en las imágenes en tres dimensiones, a las que se accede haciendo *click* en uno de los cuatro botones mostrados bajo la leyenda `Iluminación`.

Al trazar por primera vez una imagen en 3D sólo se utilizan dos tipos de luz, ambos **Luz Puntual**, pero pulsando en cualquiera de los cuatro botones de `Iluminación`, aparece el siguiente cuadro de diálogo:



donde podemos elegir entre los siguientes tipos de luz:

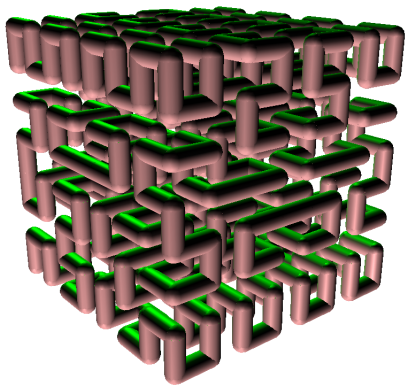
Luz Ambiente: Luz uniforme de la que sólo puede modificarse el color

Luz Direccional: Se genera respecto a una dirección fija. Se parece a la luz ambiente cuando la fuente está muy lejos del objeto (por ejemplo, el sol)

Punto de Luz: La fuente está en una posición determinada, como en el caso de un faro.

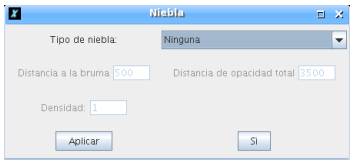
Foco: Es como el punto de luz, pero el haz de luz se abre formando un cono cuya abertura debe fijarse.

La mejor forma de entenderlo, es practicar con ello.



Efectos de niebla

Se pueden añadir efectos de niebla en la imagen tridimensional. Pulsa el botón con "nubes" y obtendrás este cuadro de diálogo:



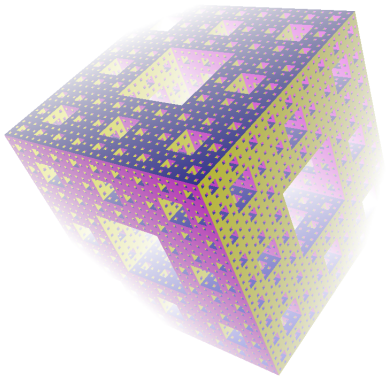
Disponemos de dos tipos de niebla:

Niebla Lineal o progresiva: La imagen se va difuminando de modo lineal, pudiendo variar dos parámetros:

- La distancia a la que empieza la niebla
- La distancia a la que la niebla no deja ver nada (opacidad total)

Niebla Densa: La niebla es uniforme en toda la escena, y sólo necesitamos especificar la densidad de la misma.

Este es un ejemplo con niebla lineal:



5.1.4. Acerca de los colores

El color en XLogo está especificado por una lista de tres números `[r v a]` comprendidos entre 0 y 255. El número `r` es el componente rojo, `v` el verde y `a` el azul (`[r g b]` en inglés). XLogo tiene 17 colores predefinidos, a los que se puede referir con un número, con su lista `[r v a]` o con una primitiva. Las primitivas correspondientes son:

Número	Primitiva	[R V A]	Color
0	`negro`	`[0 0 0]`	
1	`rojo`	`[255 0 0]`	
2	`verde`	`[0 255 0]`	
3	`amarillo`	`[255 255 0]`	

Número	Primitiva	[R V A]	Color
4	azul	[0 0 255]	
5	magenta	[255 0 255]	
6	cyan	[0 255 255]	
7	blanco	[255 255 255]	
8	gris	[128 128 128]	
9	grisclaro	[192 192 192]	
10	rojooscuro	[128 0 0]	
11	verdeoscuro	[0 128 0]	
12	azuloscuro	[0 0 128]	
13	naranja	[255 200 0]	
14	rosa	[255 175 175]	
15	violeta	[128 0 255]	
16	marrón	[153 102 0]	

Ejemplo: Estas tres órdenes son la misma:

```
poncolorlápiz naranja
poncolorlápiz 13
poncolorlápiz [255 200 0]
```

Veremos un ejemplo de su uso en la sección 13.7

5.1.5. Animación

Existen dos primitivas llamadas `animación` y `refresca` (por compatibilidad con otras versiones de Logo se mantiene la forma en infinitivo `refrescar`) que permiten escribir órdenes sin que la tortuga las realice.

Primitivas	Uso
`animación`	Se accede al modo de animación.
`detieneanimación`	Detiene el modo animación, retornando al modo *normal*.
`refresca`	En modo de animación, ejecuta las órdenes y actualiza la imagen

Mientras se escriben las órdenes en el modo de animación (una cámara de cine aparece a la izquierda del **Histórico de Comandos**), éstas no son ejecutadas en el **Área de Dibujo** sino que son almacenadas en memoria hasta que se introduce la orden `refresca`.

Haciendo *click* en este icono, se detiene el modo de animación, sin necesidad de usar la primitiva `detieneanimación`.

Esto es muy útil para crear animaciónes o conseguir que los dibujos se realicen rápidamente.

5.1.6. Propiedades del Histórico de Comandos

Esta tercera tabla define las primitivas que permiten ajustar las propiedades de texto del área del histórico de comandos. Aquellas primitivas que controlan el color y tamaño de este área, sólo están disponibles para ser usadas por las primitivas `escribe` y `tipea`.

Primitivas	Argumentos	Uso
`borratexto`, `bt`	`no`	Borra el **Área de comandos**, y el área del **Histórico de comandos**.
`escribe`, `es`	`a: número, palabra o lista`	Muestra en el **Histórico de Comandos** el argumento indicado, `a`.
`tipea`, `mecanografía`	`a: número, palabra o lista`	Idéntico a `escribe`, pero el cursor queda en la línea donde se mostró el contenido del argmento.
`ponfuentetexto`, `pft`	`n: número`	Define el tamaño de la tipografía del área del **Histórico de comandos**. Sólo disponible para ser usada por la primitiva `escribe`.
`fuentetexto`, `ftexto`	`no`	Devuelve el tamaño de la tipografía usada por la primitiva `escribe`.
`poncolortexto`, `pctexto`	`a: número o lista`	Define el color de la tipografía del área del **Histórico de comandos**. Sólo disponible para ser usada por la primitiva `escribe`.
`colortexto`	`no`	Devuelve el color de la tipografía usada por la primitiva `escribe` en el área del **Histórico de comandos**.
`ponnombrefuentetexto`, `pnft`	`n: número`	Selecciona la tipografía número `n` para escribir en el área del **Histórico de comandos** con la primitiva `escribe`. Puedes encontrar la relación entre fuente y número en el menú **Herramientas** → **Preferencias** → **Fuente.**
`nombrefuentetexto`, `nft`	`no`	Devuelve una lista con dos elementos. El primero es un número correspondiente a la fuente utilizada para escribir en el área del **Histórico de comandos** con la primitiva `escribe`. El segundo elemento es una lista que contiene el nombre de la fuente.

Primitivas	Argumentos	Uso
`ponestilo`, `pest`	`lista o palabra`	Define los efectos de fuente para los comandos en el **Histórico de comandos**. Puedes elegir entre siete estilos: `ninguno`, `negrita`, `cursiva`, `tachado`, `subrayado`, `superíndice` y `subíndice`. Si quieres aplicar varios estilos a la vez, escríbelos en una lista. Mira los ejemplos al final de la tabla.
`estilo`,	`no`	Devuelve una lista que contiene todos los efectos de fuente utilizados por las primitivas `escribe` y `tipea`.
`ponseparación`, `ponsep`	`n: número comprendido entre 0 y 1`	Determina la proporción de pantalla ocupada por el **Área de Dibujo** y el **Histórico de Comandos**. Si `n` vale 1, el **Área de Dibujo** ocupará toda la pantalla. Si `n` vale 0, será el **Histórico** quien la ocupe.
`separación`	`no`	Devuelve el valor de la proporción de pantalla ocupada por el **Área de Dibujo** y el **Histórico de Comandos**.

Ejemplos de estilos de fuente:

```
ponestilo [negrita subrayado] escribe "Hola
```

Devuelve:

Hola

```
ponestilo "tachado mecanografía [Tachado] ponestilo "cursiva tipea "\ x
            ponestilo "superíndice escribe 2
```

Devuelve:

~~Tachado~~ x^2

5.2. Operaciones aritméticas y lógicas

Esta es la lista de los operadores lógicos :

Primitivas	Argumentos	Uso
`o`	`a b: booleanos`	Devuelve `cierto` si `a` ó `b` son ciertos, si no, devuelve `falso`
`y`	`a b: booleanos`	Devuelve `cierto` si `a` y `b` son ciertos, si no, devuelve `falso`
`no`	`a: booleano`	Devuelve la negación de `a`. Si `a` es `cierto`, devuelve `falso`. Si `a` es `falso`, devuelve `cierto`.

Esta es la lista de los comandos relacionados con números:

Primitivas	Argumentos	Uso
suma, +	a b: números a sumar	Devuelve el resultado de sumar a y b.
diferencia, -	a b: números a restar	Devuelve el resultado de restar b de a.
cambiasigno, cs	a: número	Devuelve el opuesto de a.
producto, *	a b: números	Devuelve el resultado de multiplicar a por b
división, div,/	a b: números	Devuelve el resultado de dividir a por b
cociente	a b: números enteros	Devuelve el resultado de la dividisión entera de a entre b
resto	a b: números enteros	Devuelve el resto de la división de a por b
redondea	a: número	Devuelve el entero más próximo al número a
truncar, trunca	a: número	Devuelve el entero inmediatamente anterior al número a
potencia	a b: números	Devuelve a elevado a la potencia b
exp	a: número	Devuelve e ($e = 2{,}71828183\ldots$) elevado a a
raizcuadrada, rc	a: número	Devuelve la raiz cuadrada de a.
logneperiano, ln	a: número	Devuelve el logaritmo neperiano de a.
log10, log	a: número	Devuelve el logaritmo decimal de a.
seno, sen	a: número en grados	Devuelve el seno del número a.
coseno, cos	a: número en grados	Devuelve el coseno del número a.
tangente, tan	a: número en grados	Devuelve la tangente del número a.
arcocoseno, acos	a: número	Devuelve el ángulo, en grados, cuyo coseno vale a.
arcoseno, asen	a: número	Devuelve el ángulo, en grados, cuyo seno vale a.
arcotangente, atan	a: número	Devuelve el ángulo, en grados, cuya tangente vale a.
pi	no	Devuelve el número π (3.141592653589793)
azar	a: número entero	Devuelve un número al azar mayor o igual que 0 y menor que a.
absoluto, abs	a: número	Devuelve el valor absoluto (distinto de cero) del número a

Ejemplos:

`suma 40 60`	devuelve	`100`
`diferencia 100 60`	devuelve	`40`
`cambiasigno 5`	devuelve	`-5`
`cambiasigno -285`	devuelve	`285`
`división 3 6`	devuelve	`0.5`
`cociente 3 6`	devuelve	`0`
`redondea 6.4`	devuelve	`6`
`potencia 3 2`	devuelve	`9`

Importante: Ten cuidado con las primitivas que requieren dos parámetros, como `ponxy a b`. Si `b` es negativo, por ejemplo,

```
ponxy 200 -10
```

El intérprete XLogo realizará la operación `200 - 10` (o sea, le restará 10 a 200). Y determinará que hay un solo parámetro (190) cuando esperaba dos, y entonces generará un mensaje de error. Para evitar este tipo de problemas, se usa la primitiva "`cambiasigno`" para especificar un número negativo:

```
ponxy 200 cambiasigno 10
```

aunque también es válido:

```
ponxy 200 (-10)
```

Como sabemos, la presencia de paréntesis modifica el orden en que se deben realizar las operaciones. XLogo realiza las operaciones (como no podía ser de otra manera) obedeciendo a la prioridad de las mismas. Así si escribimos:

```
escribe 3 + 2 * 4
```

XLogo efectúa primero el producto y luego la suma, siendo el resultado `11`.

Sin embargo, si escribimos:

```
escribe (3 + 2) * 4
```

XLogo efectuará la suma antes que el producto, y el resultado será `20`.

Hay que tener cuidado, y esto es muy importante, si se usan las primitivas `suma`, `diferencia`, `producto`, `divisi\'on`, `potencia`, ... Por ejemplo, si queremos efectuar la operación 3^5 `+ 2 * 4 - 7`, podríamos escribir:

```
escribe potencia 3 5 + 2 * 4 - 7
```

pero observamos que XLogo devuelve:

```
729
```

¿Cómo es posible? `potencia` espera dos parámetros, la base y el exponente, así que interpreta que `3` es la base y el resto es el exponente, así que efectúa la operación `5 + 2 * 4 - 7`, y toma el resultado como exponente; es decir:

$$\texttt{potencia 3 5 + 2 * 4 - 7} = 3^{\texttt{5 + 2 * 4 - 7}} = 3^{6} = 729$$

Para que realmente se efectúe 3^{5} `+ 2 * 4 - 7`, debemos escribir:

```
(potencia 3 5) + 2 * 4 - 7
```

o bien:

```
diferencia suma potencia 3 5 producto 2 4 7
```

que se entiende mejor usando paréntesis:

```
diferencia (suma (potencia 3 5) (producto 2 4) ) 7
```

En este caso, hemos usado los paréntesis para hacer más legible el programa. Nunca olvides que un programa debe ser entendible por otra persona.

5.3. Operaciones con listas

Primitivas	Argumentos	Uso
`palabra`	`a b: palabras`	Concatena las dos palabras `a` y `b`.
`lista`	`a b`	Devuelve una lista compuesta de `a` y `b`.
`frase`, `fr`	`a b`	Devuelve una lista compuesta de `a` y `b`. Si `a` o `b` son una lista, entonces cada uno de los componentes de `a` y `b` se convierten en elementos de la lista creada. (los corchetes son suprimidos).
`ponprimero`, `pp`	`a b: a cualquiera, b lista`	Inserta `a` en la primera posición de la lista `b`.
`ponúltimo`, `pu`	`a b: a cualquiera, b lista`	Inserta `a` en la última posición de la lista `b`
`invierte`	`a: lista`	Invierte el orden de los elementos de la lista `a`
`elige`	`a: palabra o lista`	Si `a` es una palabra, devuelve una de las letras de `a` al azar. Si `a` es una lista, devuelve uno de los elementos de `a` al azar.
`quita`	`a b: a cualquiera, b lista`	Elimina el elemento `a` de la lista `b`, si aparece dentro.
`elemento`	`a b: a número entero, b lista o palabra`	Si `b` es una palabra, devuelve la letra `a` de la palabra (`1` señala la primera letra). Si `b` es una lista, devuelve el elemento número `a` de la lista.
`menosúltimo`, `mu`	`a: palabra o lista`	Si `a` es una lista, devuelve toda la lista menos el último elemento. Si `a` es una palabra, devuelve la palabra sin la última letra.
`menosprimero`, `mp`	`a: palabra o lista`	Si `a` es una lista, devuelve toda la lista menos el primer elemento. Si `a` es una palabra, devuelve la palabra sin la primera letra.

Primitivas	Argumentos	Uso
último	a: palabra o lista	Si a es una lista, devuelve el elemento de la lista. Si a es una palabra, devuelve la última letra de la palabra.
primero, pr	a: palabra o lista	Si a es una lista, devuelve el primer elemento de la lista. Si a es una palabra, devuelve la primera letra de la palabra.
miembro	a b	Investiga a en b
agrega	l1: lista n: número l2: palabra o lista	Dada la lista l1, inserta en la posición número n la palabra o lista l2. **Ejemplo**: agrega [a b c] 2 8 proporciona [a 8 b c]
reemplaza	l1: lista n: número l2: palabra o lista	Dada la lista l1, reemplaza el elemento n por la palabra o lista l2. **Ejemplo**: reemplaza [a b c] 2 8 proporciona [a 8 c]
cuenta	a: palabra o lista	Si a es una palabra, devuelve el número de letras de a. Si a es una lista, devuelve el número de elementos de a.

Para la primitiva miembro:

- Si b es una lista, investiga dentro de esta lista; hay dos casos posibles:
  - Si a está incluido en b, devuelve la sub-lista generada a partir de la primera aparición de a en b.
  - Si a no está incluido en b, devuelve la palabra falso.
- Si b es una palabra, investiga los caracteres a dentro de b. Dos casos son posibles:
  - Si a está incluido en b, devuelve el resto de la palabra a partir de a.
  - Si no, devuelve la palabra falso.

Ejemplos:

```
palabra "a 1              devuelve    a1
lista 3 6                 devuelve    [3 6]
lista "otra "lista        devuelve    [otra lista]
fr [4 3] "hola            devuelve    [4 3 hola]
fr [dime como] "vas       devuelve    [dime como vas]
ponprimero "cucu [2]      devuelve    [cucu2]
ponúltimo 5 [7 9 5]       devuelve    [7 9 5 5]
invierte [1 2 3]          devuelve    [3 2 1]
quita 2 [1 2 3 4 2 6]     devuelve    [1 3 4 6]
miembro "u "cucu          devuelve    ucu
miembro 3 [1 2 3 4]       devuelve    [3 4]
```

5.4. Booleanos

Puede ser `cierto` o `falso`. Un booleano es la respuesta a las primitivas terminadas con `?`

Primitivas	Argumentos	Uso
`cierto`	`cualquiera`	Devuelve `"cierto`
`falso`	`cualquiera`	Devuelve `"falso`
`palabra?`	`a`	Devuelve `cierto` si `a` es una palabra, `falso` si no.
`numero?`	`a`	Devuelve `cierto` si `a` es un número, `falso` si no.
`entero?`	`a: número`	Devuelve `cierto` si `a` es un número entero, `falso` si no.
`lista?`	`a`	Devuelve `cierto` si `a` es una lista, `falso` si no.
`vacío?`	`a`	Devuelve `cierto` si `a` es una lista vacía o una palabra vacía, `falso` si no.
`iguales?`	`a b`	Devuelve `cierto` si `a` y `b` son iguales, `falso` si no.
`antes?`, `anterior?`	`a b: palabras`	Devuelve `cierto` si `a` está antes que `b` siguiendo el orden alfabético, `falso` si no.
`miembro?`	`a b`	Si `b` es una lista, determina si `a` es un elemento de `b`. Si `b` es una palabra, determina si `a` es un caracter de `b`.
`bajalápiz?`, `bl?`	`cualquiera`	Devuelve la palabra `cierto` si el lápiz está abajo, `falso` si no.
`visible?`	`cualquiera`	Devuelve la palabra `cierto` si la tortuga está visible, `falso` si no.
`primitiva?`, `prim?`	`a: palabra`	Devuelve `cierto` si la palabra es una primitiva de XLogo, `falso` si no.
`procedimiento?`, `proc?`	`a: palabra`	Devuelve `cierto` si la palabra es un procedimiento definido por el usuario, `falso` si no.
`variable?`, `var?`	`a: palabra`	Devuelve `cierto` si la palabra es una variable definida por el usuario, `falso` si no.
`cuadrícula?`	`no`	Devuelve `cierto` si la cuadrícula está activa, `falso` si no.
`ejex?`	`no`	Devuelve `cierto` si está activo el eje de abscisas (eje X), `falso` si no.
`ejey?`	`no`	Devuelve `cierto` si está activo el eje de ordenadas (eje Y), `falso` si no.

5.5. Trabajando con procedimientos y variables

5.5.1. Acerca de los procedimientos

5.5.2. Procedimientos

Un procedimento es una especie de programa que, al ser llamado, ejecuta las instrucciones que contiene. Los procedimientos empiezan por la orden `para` y terminan con `fin`.

```
para nombre_de_procedimiento :var1 :var2 ... [:varA :varB ...]
  Cuerpo del procedimiento
fin
```

donde:

- `nombre_de_procedimiento` es el nombre dado al procedimiento
- `:var1 :var2 ...` son las variables *locales* usadas por el procedimiento
- `:varA :varB ...` son las variables *opcionales* que podemos añadir al procedimento (ver sección 5.5.4)
- `Cuerpo del procedimiento` representa el conjunto de órdenes que conforman el procedimiento

Veamos un pequeño ejemplo:

```
para cuadrado
  repite 4 [
    avanza 100 giraderecha 90 ]
fin
```

El procedimiento se llama `cuadrado`, y no admite nigún argumento.

Se pueden agregar **comentarios**, precediéndolos del signo `#`. En el ejemplo anterior:

```
para pentalfa
# Este procedimiento permite dibujar
# una estrella de cinco puntas
  repite 5 [
    avanza 200 giraderecha 144 ]
fin
```

5.5.3. Variables fijas

Una variable es una palabra (un nombre) a la que se le puede asignar un valor. En el ejemplo anterior podemos incluir una variable:

```
para cuadrado :lado
  repite 4 [
    avanza :lado giraderecha 90 ]
fin
```

El procedimiento se llama `cuadrado`, y admite una variable `lado`, de modo que ejecutando

```
cuadrado 200
```

la tortuga dibujará un cuadrado de lado 200 pasos. Al final del manual se incluyen varios ejemplos de procedimientos.

5.5.4. Variables opcionales

En un procedimiento pueden usarse variables opcionales, es decir, variables cuyo valor puede ser dado por el usuario y, si no lo hace, disponer de un valor *por defecto*.

```
para polígono :vértices [:lado 100]
   repite :vértices
    [ avanza :lado
      giraderecha 360/:vértices ]
fin
```

El procedimiento se llama `polígono`, lee una variable *forzosa* `vértices` que debe ser introducida por el usuario, y otra variable opcional `lado`, cuyo valor es `100` si el usuario no introduce ningún valor. De este modo que ejecutando

```
polígono 8
```

durante la ejecución, la variable `:lado` se sutituye por su valor por defecto, esto es, `100`, y XLogo dibuja un octógono de lado `100`.

Sin embargo, ejecuando

```
(polígono 8 300)
```

XLogo dibuja un octógono de lado `300`. Es importante fijarse en que ahora la ejecución se realiza encerrando las órdenes entre paréntesis. Esto indica al intérprete que se van a usar variables opcionales.

5.5.5. Conceptos acerca de variables

Hay dos tipos de variables:

- **Variables globales**: están siempre accesibles desde cualquier parte del programa.
- **Variables locales**: sólo son accesibles dentro del procedimiento donde fueron definidas.

En esta implementación del lenguaje Logo, las variables locales no son accesibles desde un sub–procedimiento. Al finalizar el procedimiento, las variables locales son eliminadas.

Primitivas	Argumentos	Uso
`haz`	`a b: a palabra, b cualquiera`	Si la variable local `a` existe, se le asigna el valor `b`. Si no, será la variable global `a` la asignada con el valor `b`.
`local`	`a: palabra`	Crea una variable llamada `a`. Atención: la variable no es inicializada. Para asignarle un valor, hay que usar `haz`.
`hazlocal`	`a b: a palabra, b cualquiera`	Crea una nueva variable llamada `a` y le asigna el valor `b`.
`define`, `def`	`palabra1 lista2 lista3`	Define un nuevo procedimiento llamado `palabra1`, provisto de las variables contenidas en `lista2` y las instrucciones a ejecutar contenidas en `lista3`.
`borra`, `bo`	`a: palabra`	Elimina el procedimiento cuyo nombre es `a`.

Primitivas	Argumentos	Uso
`cosa`, `objeto`	`a: palabra`	Reenvía el valor de `a`. `cosa "a` y `:a` son notaciones equivalentes
`borravariable`, `bov`	`a: palabra`	Elimina la variable `a`.
`borratodo`,	`no`	Elimina todas las variables y procedimientos actuales.
`imts`, `listaprocs`	`no`	Enumera en una lista todos los procedimientos actualmente definidos.
`imvars`, `listavars`	`no`	Enumera en una lista todas las variables actualmente definidas.
`listaspropiedades`, `listasprop`	`no`	Enumera en una lista todas las listas de propiedades definidas.
`contenido`	`no`	Devuelve una lista que contiene tres sub-listas: la primera con los procedimientos definidos, la segunda con los nombres de las variables existentes y la tercera con las liastas de propiedades.
`ejecuta`	`a: lista`	Ejecuta la lista de instrucciones contenida en la lista.

Ejemplos:

`haz "a 100` asigna `100` a la variable `a`

```
define "polígono [nlados largo]
  [ repite :nlados
     [ avanza :largo giraderecha 360/:nlados ] ]
```

Esto define un procedimiento llamado `polígono` con dos variables (`:nlados` y `:largo`), y permite trazar un polígono regular donde `nlados` es el número de lados, y `largo` su tamaño.

5.5.6. La primitiva `trazado`

Para seguir el desarrollo de un programa, es posible conocer los procedimientos que se están ejecutando en cada momento. Igualmente, también se puede determinar si los procedimientos están recibiendo correctamente los argumentos usando la primitiva `devuelve`.

La primitiva `trazado` activa el modo **trazado**:

```
trazado
```

mientras que para desactivarla:

```
detienetrazado
```

Un ejemplo puede verse en el cálculo del factorial (ejemplo 13.3).

```
trazado
escribe fac 4
fac 4
fac 4
 fac 3
  fac 2
   fac 1
   fac devuelve 1
  fac devuelve 2
 fac devuelve 6
fac devuelve 24
24
```

5.6. Listas de Propiedades

Desde la versión 0.9.92, pueden definirse Listas de Propiedades con XLogo. Cada lista tiene un nombre específico y contiene varias parejas de "clave + valor", que constan de un identificador y del elemento en sí. Es decir, son listas cuyos elementos no se etiquetan mediante números, sino con otro nombre.

Por ejemplo, podemos considerar una lista de propiedades llamado "coche", que contiene la clave "color" asociado al valor "rojo", y la clave "tipo" con el valor "4x4".

Para manejar estas listas, podemos utilizar los siguientes primitivas:

- `ponpropiedad`, `ponprop`

  Sintaxis: `ponpropiedad nombre.lista clave valor`

  Añade una propiedad a la lista de propiedades llamada `nombre.lista`. El `valor` será accesible con la `clave`.

  Si no existe una lista de propiedades llamado `nombre.lista` entonces será creada.

- `leepropiedad`, `leeprop`, `devuelvepropiedad`

  Sintaxis: `leepropiedad nombre.lista clave`

  Devuelve el valor asociado a la `clave` de la lista de propiedades llamada `nombre.lista`. Si esta propiedad no existe o si no existe ninguna clave válida, devuelve una lista vacía.

- `borrapropiedad`, `boprop`

  Sintaxis: `borrapropiedad nombre.lista clave`

  Elimina la correspondiente pareja clave-valor en la lista de propiedades `nombre.lista`

- `listapropiedades`, `listaprop`

  Sintaxis: `listapropiedades nombre.lista`

  Muestra todos los pares clave-valor que figuran en la lista de propiedades llamada `listapropiedades`

- `borralistapropiedades`, `bolisprop`

  Sintaxis: `borralistapropiedades "nombre.lista` o `borralistapropiedades [ lista.de.nomb ]`

  Borra la/s lista/s indicada/s con su nombre o en la lista

Vamos a volver a la lista de propiedades llamada "coche".

```
# Llenado de la Lista de Propiedades
ponpropiedad "Coche "Color "Rojo
ponpropiedad "Coche "Tipo "4x4
ponpropiedad "Coche "Vendedor "Citroen

# Mostrar un valor
escribe leepropiedad "Coche "Color    ---> Rojo

# Mostrar todos los elementos
escribe listapropiedades "Coche       ---> Color Roja Tipo 4x4 Vendedor Citroen
```

5.7. Manejo de archivos

Primitivas	Argumentos	Uso
`catálogo`, `cat`	`no`	Lista el contenido del directorio actual. (Equivalente al comando `ls` de Linux, `dir` de DOS)
`pondirectorio`, `pondir`	`l: lista`	Especifica el directorio actual. La ruta debe ser absoluta. El directorio debe especificarse dentro de una lista, y la ruta no debe contener espacios.
`cambiadirectorio`, `cd`	`a: palabra o lista`	Cambia el directorio de trabajo desde el directorio actual (ruta relativa). Puede utilizarse `..` para referirse a la ruta del directorio superior.
`directorio`, `dir`	`no`	Da el directorio actual. Por defecto, es `/home/tu_nombre` en Linux, `C:\WINDOWS` en Windows.
`guarda`	`a: palabra, l: lista`	Guarda en el archivo `a` los procedimientos especificados en `l`, en el directorio actual. (Ver ejemplo)
`guardatodo`	`a: palabra`	Guarda en el archivo `a` todos los procedimientos definidos, en el directorio actual. (Ver ejemplo)
`carga`	`a: palabra`	Abre y lee el archivo `a`.
`abreflujo`	`n: número, a: nombre_fichero`	Para poder leer o escribir en un fichero, es necesario crear un flujo hacia él. El argumento `nombre_fichero` debe ser su nombre, que se refiere al directorio de trabajo. El argumento `n` es el número que identifica a ese flujo.
`listaflujos`	`l: lista`	Carga una lista con los flujos abiertos indicando su identificador
`leelineaflujo`	`n: número`	Abre el flujo cuyo identificador es `n`, y lee una línea del fichero

Primitivas	Argumentos	Uso
`leecarflujo`	`n: número`	Abre el flujo cuyo identificador es `n`, después lee un caracter del fichero. Esta primitiva devuelve el número correspondiente al caracter unicode (como `leecar` - sec. 10.1)
`escribelineaflujo`	`n: número, l: lista`	Escribe la línea de texto indicada en `l` al principio del fichero indicado por el flujo `n`. Atención: la escritura no se hace efectiva hasta que se ciera el fichero con `cierraflujo`.
`agregalineaflujo`	`n: número, l: lista`	Escribe la línea de texto indicada en `l` al final del fichero indicado por el flujo `n`. Atención: la escritura no se hace efectiva hasta que se ciera el fichero con `cierraflujo`.
`cierraflujo`	`n: número`	Cierra el flujo `n`.
`finflujo?`	`n: número`	Devuelve `cierto` si se ha llegado al final del fichero, y `falso` en caso contrario.
`cargaimagen`, `ci`	`a: palabra`	Carga el archivo de imagen indicado por la palabra.

En la primitiva `cargaimagen`, se debe tener en cuenta que la esquina superior izquierda se ubica en la posición actual de la tortuga. Los únicos formatos soportados son jpg y png. La ruta debe especificarse previamente con `pondirectorio` y debe ser absoluta, empezando en el nivel superior del árbol de directorios. **Ejemplo**:

```
pondirectorio [/home/alumnos/mis\ imagenes]
cargaimagen "turtle.jpg
```

Ejemplos:

- `guarda "trabajo.lgo [proc1 proc2 proc3]` guarda en el directorio actual un archivo llamado `trabajo.lgo` que contiene los procedimientos `proc1`, `proc2` y `proc3`.
- `guardatodo "trabajo.lgo` guarda en el directorio actual un archivo llamado `trabajo.lgo` que contiene la totalidad de los procedimientos actualmente definidos.

En ambos casos, si no se indica la extensión `.lgo`, será añadida. La palabra especifica una ruta relativa a partir del directorio corriente. No funciona colocar una ruta absoluta.

Para borrar todos los procedimientos definidos y cargar el archivo `trabajo.lgo`, debes usar:

```
borratodo carga "trabajo.lgo
```

La palabra especifica una ruta relativa a partir del directorio corriente. No funciona colocar una ruta absoluta.

Ejemplo 2:

El objetivo es crear el fichero `ejemplo` en el directorio personal: `/home/tu_nombre`, en Linux, `C:\`, en windows que contiene:

```
ABCDEFGHIJKLMÑNOPQRSTUVWXYZ
abcdefghijklmñnopqrstuvwxyz
0123456789
```

Para ello:

```
# abre un flujo hacia el fichero indicado
# identificara el flujo con el numero 2
pondirectorio "/home/tu_nombre
abreflujo 2 "ejemplo
# escribe las lineas que quiero
escribelineaflujo 2 [ABCDEFGHIJKLMNÑOPQRSTUVWXYZ]
escribelineaflujo 2 [abcdefghijklmnñopqrstuvwxyz]
escribelineaflujo 2 [0123456789]
# cerramos el flujo para acabar la escritura
cierraflujo 2
```

Ahora, comprobamos que está bien escrito:

```
# abre un flujo hacia el fichero indicado
# identificara el flujo con el numero 0
pondirectorio "/home/tu_nombre
abreflujo 0 "ejemplo
# lee las lineas del fichero consecutivamente
escribe leelineaflujo 0
escribe leelineaflujo 0
escribe leelineaflujo 0
# cerramos el flujo
cierraflujo 0
```

Si queremos que nuestro fichero termine con la línea `Formidable!`:

```
pondirectorio "c:\\
abreflujo 1 "ejemplo
agregalineaflujo 1 [Formidable!]
cierraflujo 1
```

5.8. Función avanzada de relleno

Las primitivas `rellena` y `rellenazona` permiten pintar una figura. Se pueden comparar a la función "`rellena`" disponible en la mayoría de los programas de dibujo. Esta funcionalidad se extiende hasta los márgenes del área de dibujo. Hay tres reglas a tener en cuenta para usar correctamente estas primitivas:

1. El lápiz debe estar bajo (`bl`).

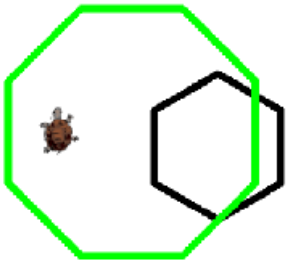
2. La tortuga no debe estar sobre un punto del mismo color que se usará para rellenar. (Si quieres pintar rojo, la tortuga no puede estar sobre un punto rojo).

3. Observar si `cuadrícula` está o no activada.

Veamos un ejemplo para explicar la diferencia entre estas dos primitivas:

Los píxeles por donde pasa la tortuga son, en este momento, blancos. La primitiva `rellena` va a colorear todos los píxeles blancos vecinos con el color elegido para el lápiz hasta llegar a una frontera de cualquier color (incluida la cuadrícula).

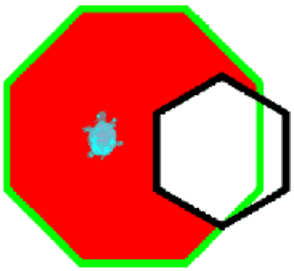
Supongamos que tenemos este dibujo en el Área de Dibujo:



y tecleamos:

```
poncolorlápiz 1
rellena
```

produce:

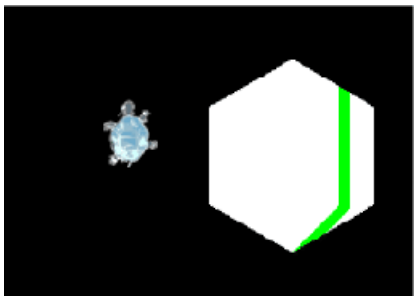


es decir, ha coloreado de rojo la región cerrada en la que se encuentra la tortuga.

Sin embargo, si hacemos:

```
poncolorlápiz 0
rellenazona
```

se obtiene:

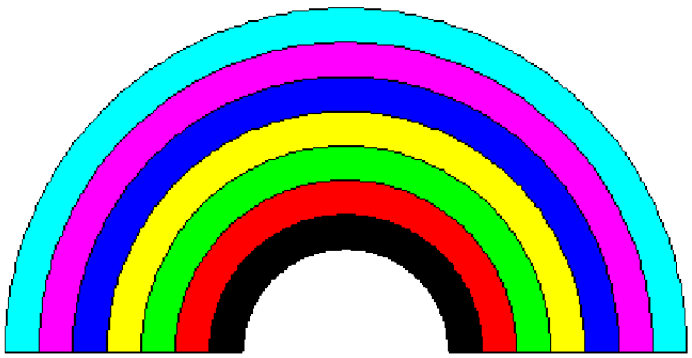


es decir, rellena todos los píxeles vecinos hasta encontrar una "frontera" del color activo.

Este es un buen ejemplo para usar la primitiva `rellena`:

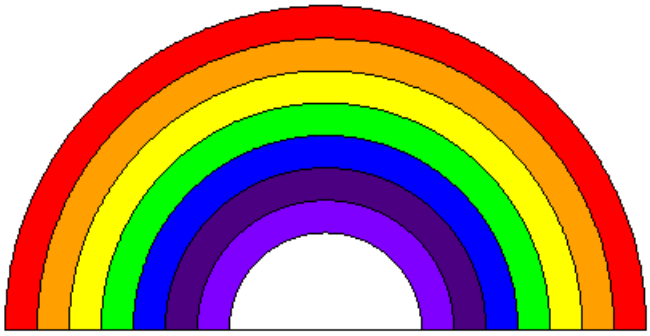
```
para mediocirc :c
# dibuja un semicírculo de diametro :c
```

```
  repite 180 [
    avanza :c * tan 0.5
    giraderecha 1 ]
  avanza :c * tan 0.5
  giraderecha 90 avanza :c
fin

para arcohueco :c
# Utiliza el procedimiento mediocirc para dibujar un arcoiris sin colores
  si :c < 100 [alto]
  mediocirc :c
  giraderecha 180 avanza 20 giraizquierda 90
  arcohueco :c - 40
fin

para arcoiris
   borrapantalla ocultatortuga arcohueco 400
   subelápiz giraderecha 90 retrocede 150
   giraizquierda 90 avanza 20 bajalápiz
   repitepara [color 0 6]
    [ poncolorlápiz (6-:color) rellena
      subelápiz giraderecha 90 avanza 20
      giraizquierda 90 bajalápiz ]
fin
```



o bien, más realista:

```
para arcoiris2
   borrapantalla ocultatortuga arcohueco 400
   subelápiz giraderecha 90 retrocede 150
   giraizquierda 90 avanza 20 bajalápiz
   haz "color [ [255 0 0] [255 160 0] [255 255 0] [0 255 0]
        [0 0 255] [75 0 130] [128 0 255] ]
   repitepara [colores 1 7]
      [ poncolorlápiz elemento :colores :color rellena
       subelápiz giraderecha 90 avanza 20 giraizquierda 90 bajalápiz ]
fin
```



5.9. Comandos de ruptura de secuencia

XLogo tiene tres comandos de ruptura de secuencia: `alto`, `detienetodo` y `devuelve`.

- `alto` puede tener dos resultados:
  - Si está incluído en un bucle `repite` o `mientras`, el programa sale del bucle inmediatamente.
  - Si está en un procedimiento, este es terminado.
- `detienetodo` interrumpe total y definitivamente todos los procedimientos en ejecución
- `devuelve` (`dev`) permite salir de un procedimiento "llevándose" un resultado.

Al final del manual hay numerosos ejemplos con el uso de estas primitivas.

Capítulo 6

Condicionales

La primitiva básica que define el condicional en XLogo es `si`. Su uso es simple:

```
si expresión_lógica [comandos]
```

que ejecuta `comandos` únicamente cuando `expresión_lógica` sea `cierto`, o bien:

```
si expresión_lógica [comandos1] [comandos2]
```

donde `comandos1` y `comandos2` son, respectivamente, las órdenes a ejecutar en los casos en los que `expresión_lógica` sea `cierto` o `falso`.

Ejemplos:

- Procedimiento que compara un número dado con 4 y contesta `MAYOR` si el número es mayor que 4:

```
para mayor :X
  si :x > 4 [escribe "MAYOR]
fin
```

- Procedimiento que compara un número con 4, para ver si es mayor que 4 o no lo es:

```
para compara :X
  si :x > 4 [escribe "SI] [escribe "NO]
fin
```

Si queremos que los argumentos para `cierto` y `falso` estén guardados en sendas variables, no podemos usar `si`. En este caso, la primitiva correcta es:

`sisino`

En este ejemplo, XLogo mostrará un error:

```
haz "Opcion_1 [escribe "cierto]
haz "Opcion_2 [escribe "falso]
si 1 = 2 :a :b
```

ya que la segunda lista nunca será evaluada:
¿Qué hacer con [escribe "falso]?

La sintaxis correcta es:

```
haz "Opcion_1 [escribe "cierto]
haz "Opcion_2 [escribe "falso]
sisino 1 = 2 :a :b
```

que devolverá:

```
"falso
```

Capítulo 7

Bucles y recursividad

XLogo tiene cinco primitivas que permiten la construcción de bucles: `repite`, `repitepara`, `mientras`, `paracada` y `repitesiempre`.

7.1. Bucle con `repite`

Esta es la sintaxis para `repite`:

```
repite n [ lista_de_comandos ]
```

`n` es un número entero y `lista_de_commandos` es una lista que contiene los comandos a ejecutarse. El intérprete XLogo ejecutará la secuencia de comandos de la lista `n` veces. Esto evita copiar los mismos comandos repetidas veces.

Ejemplos:

```
repite 4 [avanza 100 giraderecha 90]     # un cuadrado de lado 100

repite 6 [avanza 100 giraderecha 60]     # un hexágono de lado 100

repite 360 [avanza 2 giraderecha 1]      # abreviando, casi un círculo
```

Observa que con el bucle `repite`, se define una variable interna `contador` o `cuentarepite`, que determina el número de la iteración en curso (la primera iteración está numerada con el 1)

```
repite 3 [escribe cuentarepite]
repite 3 [escribe contador]
```

proporcionan ambas

```
1
2
3
```

7.2. Bucle con repitepara

`repitepara` hace el papel de los bucles `for` en otros lenguajes de programación. Consiste en asignar a una variable un número determinado de valores comprendidos en un intervalo y con un incremento (paso) dados. Su sintaxis es:

```
repitepara [ lista1 ] [ lista2 ]
```

La `lista1` contiene tres parámetros: el nombre de la variable y los límites inferior y superior del intervalo asignado a la variable. Puede añadirse un cuarto argumento, que determinaría el incremento (el paso que tendría la variable); si se omite, se usará 1 por defecto.

Ejemplo 1:

```
repitepara [i 1 4] [escribe :i*2]
```

proporciona

```
2
4
6
8
```

Ejemplo 2:

```
# Este procedimiento hace variar i entre 7 y 2, bajando de 1.5 en 1.5
# nota el incremento negativo
repitepara [i 7 2 -1.5]
  [es lista :i potencia :i 2]
```

proporciona

```
7 49
5.5 30.25
4 16
2.5 6.25
```

7.3. Bucle con mientras

Esta es la sintaxis para `mientras`:

```
mientras [lista_a_evaluar] [ lista_de_comandos ]
```

`lista_a_evaluar` es la lista que contiene un conjunto de instrucciones que se evalúan como booleano (`cierto` o `falso`). `lista_de_comandos` es una lista que contiene los comandos a ser ejecutados. El intérprete XLogo continuará repitiendo la ejecución de `lista_de_comandos` todo el tiempo que `lista_a_evaluar` devuelva cierto.

Ejemplos:

```
   mientras ["cierto]
      [giraderecha 1] # La tortuga gira sobre si misma eternamente.

# Este ejemplo deletrea el alfabeto en orden inverso:
haz "lista1 "abcdefghijklmnñopqrstuvwxyz
mientras [no vacío? :lista1]
   [escribe último :lista1 haz "lista1 menosúltimo :lista1]
```

7.4. Bucle con paracada

La sintacis de `paracada` es:

```
paracada nombre_variable lista_o_palabra [ lista_de_comandos ]
```

La variable va tomando como valores los elementos de la lista o los caracteres de la palabra, y las órdenes se repiten para cada calor adquirido.
Ejemplos:

```
paracada "i "XLogo
   [escribe :i]
```

muestra:

```
X
L
o
g
o
```

```
paracada "i [a b c]
   [escribe :i]
```

muestra:

```
a
b
c
```

```
haz "suma 0
paracada "i 12345
   [haz "suma :suma+:i]
```

muestra:

```
15
```

(la suma de los dígitos de 12345)

7.5. Bucle con repitesiempre

Aunque un bucle como este es muy peligroso en programación, ya vimos al explicar el bucle `mientras` un ejemplo donde puede simularse un bucle infinito. La sintaxis es:

```
repitesiempre [ lista_de_comandos ]
```

El ejemplo anterior sería:

```
repitesiempre [giraderecha 1] # La tortuga gira sobre si misma eternamente.
```

De nuevo: **Mucho cuidado al usar bucles infinitos**

7.6. Recursividad

Un procedimiento se llama *recursivo* cuando se llama a sí mismo (es un *subprocedimiento* de sí mismo).

Un ejemplo típico es el cálculo del **factorial**. En lugar de definir
`n! = n * (n - 1) * ...* 3 * 2* 1`,
podemos hacer:

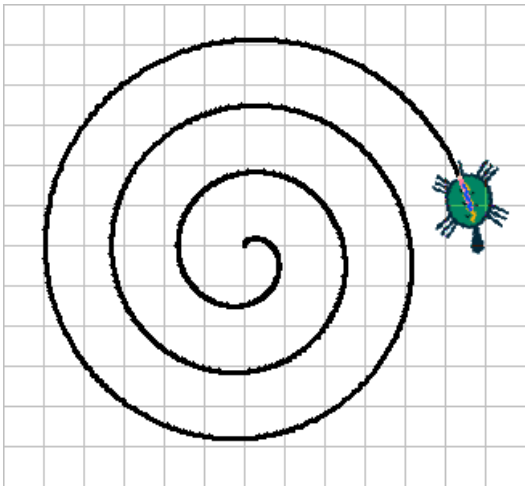
$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n \neq 0 \end{cases} \qquad \forall\, n \in \mathbb{N}$$

En XLogo:

```
para factorial :número
   si :número = 0
    [ devuelve 1 ]
    [ devuelve :número * factorial (:número - 1) ]
fin
```

Un segundo ejemplo recursivo es la **espiral**:

```
para espiral :lado
  si :lado > 1250
    [ alto ]
    [ avanza :lado / 500
      giraderecha 1
      espiral :lado + 1 ]
fin
```



Capítulo 8

Modo multitortuga

Se pueden tener varias tortugas activas en pantalla. Nada más iniciarse XLogo, sólo hay una tortuga disponible. Su número es 0. Si quieres "crear" una nueva tortuga, puedes usar la primitiva `pontortuga` seguida del número de la nueva tortuga. Para evitar confusión, la nueva tortuga se crea en el centro y es invisible (tienes que usar `muestratortuga` para verla). Así, la nueva tortuga es la activa, y será la que obedezca las clásicas primitivas mientras no cambies a otra tortuga con `pontortuga`. El máximo número de tortugas disponibles puede fijarse en el menú **Herramientas** → **Preferencias**.

Estas son las primitivas que se aplican al modo multitortuga:

Primitiva	Argumentos	Uso
`pontortuga`, `ptortuga`	`a: número`	La tortuga número `a` es ahora la tortuga activa. Por defecto, cuando XLogo comienza, está activa la tortuga número 0.
`tortuga`	`no`	Da el número de la tortuga activa.
`tortugas`	`no`	Da una lista que contiene todos los números de tortuga actualmente en pantalla.
`eliminatortuga`	`a: número`	Elimina la tortuga número `a`
`ponmaximastortugas`, `pmt`	`n: número`	Fija el máximo número de tortugas
`maximastortugas`, `maxt`	`no`	Devuelve el máximo número de tortugas

Capítulo 9

Tocar música (MIDI)

Ya comentamos anteriormente (sección 3.3) que la versión para Windows de `jre` no incorpora las API (*Application Programming Interface* – Interfaz de Programación de Aplicaciones) que contienen los instrumentos y que deben ser instaladas manualmente (Preguntas frecuentes, 15.1). Es importante recordarlo porque, si no lo haces, con la instalación por defecto de JAVA no tendrás instrumentos disponibles.

Las primitivas que nos ocupan son:

Primitivas	Argumentos	Uso
`secuencia`, `sec`	`a: lista`	Carga en memoria la secuencia incluída en la lista. Siguiendo a esta tabla, se indica cómo escribir una secuencia de notas músicales.
`tocamúsica`	`no`	Toca la secuencia de notas en memoria.
`instrumento`, `instr`	`no`	Da el número que corresponde al instrumento actualmente seleccionado.
`poninstrumento`, `pinstr`	`a: número`	Queda seleccionado el instrumento número `a`. Puedes ver la lista de instrumentos disponibles en el menú **Herramientas** → **Preferencias** → **Sonido**.
`indicesecuencia`, `indsec`	`no`	Da la posición del puntero en la secuencia corriente.
`ponindicesecuencia`, `pindsec`	`a: número`	Pone el puntero en la posición `a` de la secuencia corriente.
`borrasecuencia`, `bos`	`no`	Elimina de memoria la secuencia corriente.

Para tocar música, primero hay que poner en memoria una lista de notas llamada *secuencia*. Para crear una secuencia, puedes usar la primitiva `sec` o `secuencia`. Para crear una secuencia válida, hay que seguir las siguientes reglas:

- `do re mi fa sol la si`: Las notas usuales de la primera octava.
- Para hacer un re sostenido, anotamos `re +`
- Para hacer un re bemol, anotamos `re -`

- Para subir o bajar una octava, usamos ":" seguido de "+" o "−".

  Ejemplo:

  Después de `:++` en la secuencia, todas las notas sonarán dos octavas más altas.

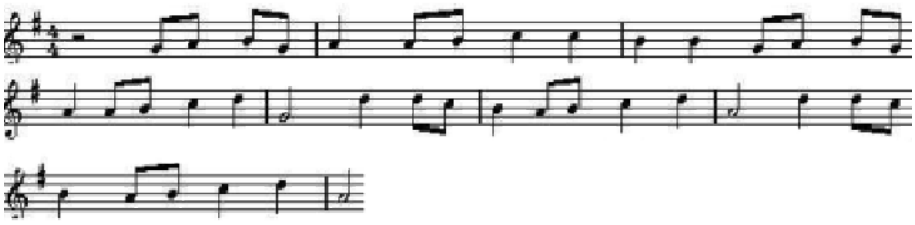
Por defecto, todas las notas tienen una duración uno. Si quieres aumentar o disminuir la duración, debes escribir un número correspondiente.

Ejemplos:

```
secuencia [sol 0.5 la si]
```

tocará sol con la duración 1 y la y si con la duración 0.5 (el doble de rápido).

Otro ejemplo:



```
para partitura
# crea la secuencia de notas
  secuencia [0.5 sol la si sol 1 la 0.5 la si 1 :+ do do :- si si
     0.5 sol la si sol 1 la 0.5 la si 1 :+ do re 2 :- sol ]
  secuencia [:+ 1 re 0.5 re do 1 :- si 0.5 la si 1 :+ do re 2 :- la ]
  secuencia [:+ 1 re 0.5 re do 1 :- si 0.5 la si 1 :+ do re 2 :- la ]
  secuencia [0.5 sol la si sol 1 la 0.5 la si 1 :+ do do :- si si
     0.5 sol la si sol 1 la 0.5 la si 1 :+ do re 2 :- sol ]
fin
```

Para escuchar la música, ejecuta las primitivas:
`partitura tocamúsica`.

Ahora veamos una aplicación interesante de la primitiva `ponindicesecuencia`:

```
borrasecuencia # elimina toda secuencia de memoria
partitura      # pone en memoria las notas
pindsec 2      # pone el cursor en el segundo "la"
partitura      # pone en memoria las mismas notas, pero movidas 2 lugares.
tocamúsica     # Grandioso!
```

También puedes elegir un instrumento con la primitiva `poninstrumento` o en el menú **Herramientas → Preferencias → Sonido**. Encontrarás la lista de instrumentos disponibles asociados a un número. (Si usas Windows, echa un vistazo a las Preguntas Frecuentes si no lo has hecho aún)

Capítulo 10

Recibir entrada del usuario

10.1. Interactuar con el teclado

Durante la ejecución del programa, se puede recibir texto ingresado por el usuario a través de 3 primitivas: `tecla?`, `leecar` y `leelista`.

- `tecla?`: Da `cierto` o `falso` según se haya pulsado o no alguna tecla desde que se inició la ejecución del programa.
- `leecar` o `leetecla`:
  - Si `tecla?` es `falso`, el programa hace una pausa hasta que el usuario pulse alguna tecla.
  - Si `tecla?` es `cierto`, da la última tecla que haya sido pulsada.

  Estos son los valores que dan ciertas teclas:

A → 65	B → 66	C → 67	...	Z → 90
◁ → -37 ó -226	△ → -38 ó -224	▷ → -39 ó -227	▽ → -40 ó -225	
`Esc` → 27	`F1` → -112	`F2` → -113	...	`F12` → -123
`SHIFT` → -16	`ESPACIO` → 32	`CTRL` → -17	`ENTER` → 10	

  Si tienes dudas acerca del valor que da alguna tecla, puedes probar con: `es leecar`. El intérprete esperará hasta que pulses una tecla, y escribirá su valor.

- `leelista [título] "palabra` o `leeteclado [título] "palabra`: Presenta una caja de diálogo titulada `título`. El usuario puede escribir un texto en el área de entrada, y esta respuesta se guardará seleccionando automáticamente si en forma de número o de lista en la variable `:palabra` cuando se haga *click* en el botón `OK`.

Las primitivas `caracter`, (su forma corta es `car` y cuyo argumento es `n`: un número) y `unicode "a`, devuelven, respectivamente, el carácter unicode que corresponde al número `n` y el número unicode que corresponde al carácter `a`.

Ejemplo:

```
unicode "A          devuelve      65
caracter 125        devuelve      }
```

Ejemplos:

```
para edades
  leelista [¿Qué edad tienes?] "edad
  si :edad < 18 [escribe [Eres menor]]
  si :edad > 17 [escribe [Eres adulto]]
  si :edad > 69 [escribe [Con todo respeto!!]]
fin

para dibujar
# La tortuga es controlada con las flechas del teclado.
# Se termina con Esc.
si tecla?
  [ haz "valor leecar
    si :valor=-37 [giraizquierda 90]
    si :valor=-39 [giraderecha 90]
    si :valor=-38 [avanza 10]
    si :valor=-40 [retrocede 10]
    si :valor=27 [alto] ]
 dibujar
fin
```

10.2. Interactuar con el ratón

Durante la ejecución del programa, se pueden recibir eventos del ratón a través de tres primitivas: `leeratón`, `ratón?` y `posratón`.

- `leeratón`: el programa espera hasta que el usuario hace un *click* o un movimiento. Entonces, devuelve un número que representa ese evento. Los posibles valores son:
  - 0 → El ratón se movió.
  - 1 → Se hizo un *click* izquierdo.
  - 2 → Se hizo un *click* central (se usa en Linux).
  - 3 → Se hizo un *click* derecho.
- `posratón`: Da una lista que contiene la posición actual del ratón.
- `ratón?`: Devuelve `cierto` o `falso` según toquemos o no el ratón desde que comienza la ejecución del programa

Ejemplos:

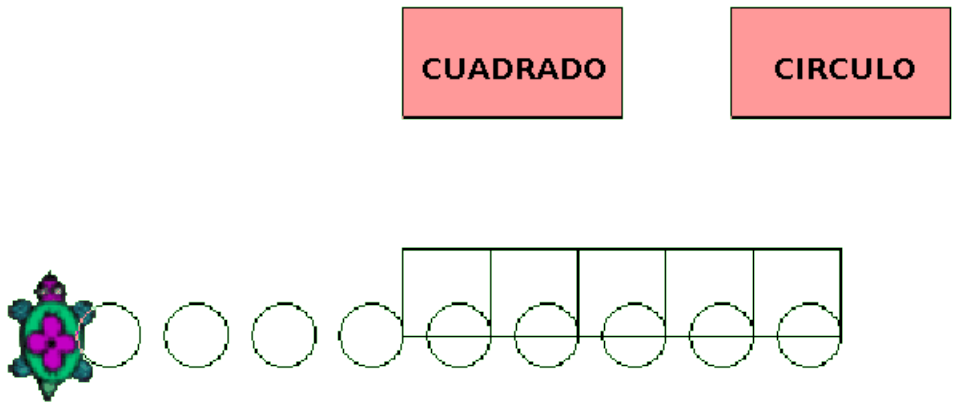
En este primer procedimiento, la tortuga sigue los movimientos del ratón por la pantalla.

```
para seguir
# cuando el ratón se mueve, la tortuga cambia de posición
    si leeratón=0 [ponposición posratón]
    seguir
fin
```

Este segundo procedimiento es similar, pero hay que hacer *click* izquierdo para que la tortuga se mueva.:

```
para seguir2
   si leeratón = 1 [ponposición posratón]
   seguir2
fin
```

En este tercer ejemplo, hemos creado dos botones rosa. Si hacemos *click* izquierdo, la tortuga dibuja un cuadrado de lado 40. Si hacemos *click* derecho, la tortuga dibuja un pequeño círculo. Por último si hacemos *click* derecho en el botón derecho, se detiene el programa.




```
para botón
# crea un botón rectangular color rosa, de 50 x 100
   repite 2 [
   avanza 50 giraderecha 90 avanza 100 giraderecha 90 ]
   giraderecha 45 subelapiz avanza 10
   bajalápiz poncolorlápiz [255 153 153]
   rellena retrocede 10 giraizquierda 45 bajalápiz poncolorlápiz 0
fin

para empieza
   borrapantalla botón subelápiz ponposición [150 0]
   bajalápiz botón subelápiz ponposición [30 20] bajalápiz
   rotula "Cuadrado subelápiz ponposición [180 20]
   bajalápiz rotula Çírculo
   subelápiz ponposición [0 -100] bajalápiz
   ratón
fin

para ratón
# ponemos el valor de leeratón en la variable ev
# ponemos la primera coordenada en la variable x
# ponemos la segunda coordenada en la variable y
```

```
   haz "ev leeratón
   haz "x elemento 1 posratón
   haz "y elemento 2 posratón
# si hay click izquierdo
   si :ev=1 & :x>0 & :x<100 & :y>0 & :y<50 [cuadrado]
# si hay click derecho
   si :x>150 & :x<250 & :y>0 & :y<50 [
   si :ev=1 [circunferencia]
   si :ev=3 [alto] ]
   ratón
fin

para circunferencia
   repite 90 [avanza 1 giraizquierda 4]
   giraizquierda 90 subelápiz avanza 40 giraderecha 90 bajalápiz
fin

para cuadrado
   repite 4 [avanza 40 giraderecha 90]
   giraderecha 90 avanza 40 giraizquierda 90
fin
```

10.3. Componentes Gráficos

Desde la versión 0.9.90, XLogo permite añadir componentes gráficos en el Área de dibujo (botones, menús, ...)

Las primitivas que permiten crear y modificar estos componentes terminan con el sufijo `igu` (Interfaz Gráfica de Usuario – *Graphical User Interface, gui* son sus siglas inglesas).

10.3.1. Crear un componente gráfico

La secuencia de pasos que debes seguir es: **Crear** → **Modificar** sus propiedades o características → **Mostrar**lo en el Área de dibujo.

Crear un Botón

Usaremos al primitiva `botonigu`, cuya sintaxis es:

```
# Esta primitiva crea un botón llamado b
# y cuya leyenda es: Click
botonigu "b "Click
```

Crear un Menú

Disponemos de la primitiva `menuigu`, cuya sintaxis es:

```
# Esta primitiva crea un menú llamado m
# y que contiene 3 opciones: opción1, opción2 y opción3
menuigu "m [opción1 opción2 opción3]
```

Modificar las propiedades del componente gráfico

`posicionigu` determina las coordenadas donde se colocará el elemento gráfico. Por ejemplo, para colocar el botón definido antes en el punto de coordenadas `(20 , 100)`, escribiremos:

```
posicionigu "b [20 100]
```

Si no se especifica la posición, el objeto será colocado por defecto en la esquina superior izquierda del Área de dibujo.

Eliminar un componente gráfico

La primitiva `eliminaigu` elimina un componente gráfico. Para eliminar el botón anterior

```
eliminaigu "b
```

Definir acciones asociadas a un componente gráfico

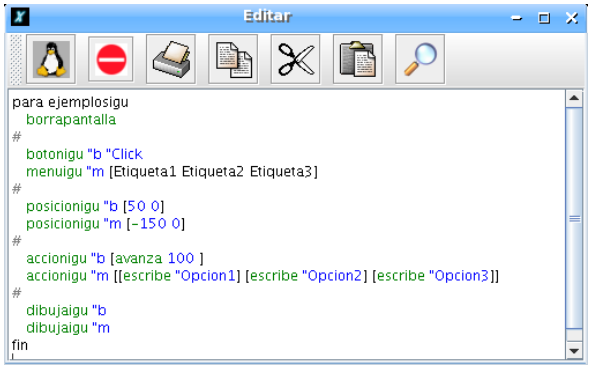
La primitiva `accionigu`, define una acción asociada al componente, y que se realizará cuando el usuario interactúa con él.

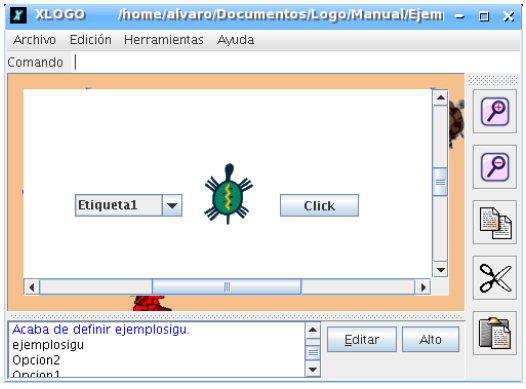
```
# Que la tortuga avance 100 al pulsar el boton "b
accionigu "b [avanza 100 ]
# En el menú, cada opción indica su acción
accionigu "m [[escribe "Opción1] [escribe "Opción2] [escribe "Opción3]]
```

Dibujar el componente gráfico

La primitiva `dibujaigu`, muestra el componente gráfico en el Área de dibujo. Para mostrar el botón que estamos usando como ejemplo:

```
dibujaigu "b
```

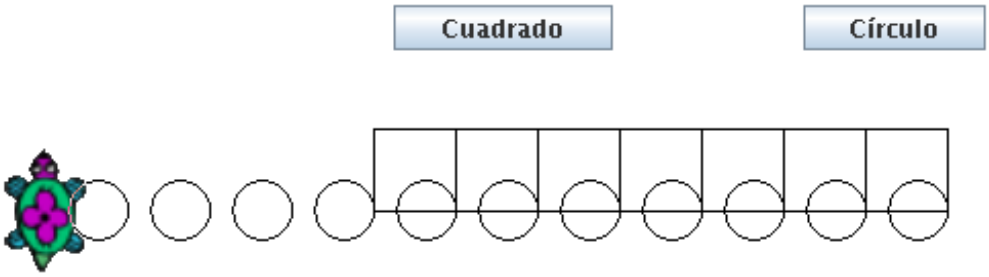




Corrijamos el ejemplo anterior utilizando las nuevas primitivas:

```
para empieza
   botonigu "Boton.Circ "Círculo
   botonigu "Boton.Cuad "Cuadrado
   posicionigu "Boton.Circ [50 100]
   posicionigu "Boton.Cuad [-150 100]
   accionigu "Boton.Circ [ circunferencia ]
```

```
   accionigu "Boton.Cuad [ cuadrados ]
   dibujaigu "Boton.Circ
   dibujaigu "Boton.Cuad
fin

para circunferencia
   repite 90 [av 1 gi 4]
   giraizquierda 90 subelápiz avanza 40 giraderecha 90 bajalápiz
fin

para cuadrado
   repite 4 [avanza 40 giraderecha 90]
   giraderecha 90 avanza 40 giraizquierda 90
fin
```



Capítulo 11

Gestión de tiempos

XLogo dispone de varias primitivas que permiten conocer la hora y la fecha o utilizar un cronómetro descendente (útil para repetir una tarea a intervalos fijos).

Primitivas	Argumentos	Uso
`espera`	`n: número entero`	Hace una pausa en el programa, la tortuga espera `(n/60)` segundos.
`cronómetro`, `crono`	`n: número entero`	Inicia un conteo descendiente de `n` segundos. Para saber que la cuenta ha finalizado, disponemos de la primitiva `fincrono?`
`fincronómetro?`, `fincrono?`	`no`	Devuelve `"cierto` si no hay ningún conteo activo. Devuelve `"falso` si el conteo no ha terminado.
`fecha`	`no`	Devuelve una lista compuesta de 3 números enteros que representan la fecha del sistema. El primero indica el día, el segundo el mes y el último el año. `[día mes año]`
`hora`	`no`	Devuelve una lista compuesta de 3 números enteros que representan la hora del sistema. El primero representa las horas, el segundo los minutos y el último los segundos. `[horas minutos segundos]`
`tiempo`	`no`	Devuelve el tiempo, en segundos, transcurrido desde el inicio de XLogo.

Veamos un procedimiento de ejemplo:

```
para reloj
 # muestra la hora en forma numerica (actualizada cada 5 segundos)
  si fincrono? [
   bp ponfuente 75 ot
   haz "ho hora
   haz "h primero :ho
   haz "m elemento 2 :ho
 # muestra dos cifras para los minutos (completando el 0)
   si :m - 10 < 0 [
    haz "m palabra 0 :m ]
```

```
    haz "s ultimo :ho
  # muestra dos cifras para los segundos
    si :s - 10 < 0 [
     haz "s palabra 0 :s ]
     rotula (palabra :h ": :m ": :s) crono 5 ]
   reloj
fin
```

Capítulo 12

Utilización de la red con XLogo

12.1. La red: ¿cómo funciona eso?

En primer lugar es necesario explicar los conceptos básicos de la comunicación en una red para usar correctamente las primitivas de XLogo.

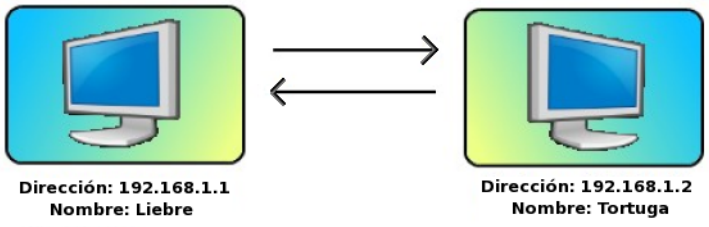


Figura: noción de red

Dos ordenadores pueden comunicarse a través de una red si están equipados con una tarjeta de red (llamada también tarjeta *ethernet*). Cada ordenador se identifica por una dirección personal: su dirección I.P. Esta dirección IP consta de cuatro números enteros comprendidos entre 0 y 255 separados por puntos. Por ejemplo, la dirección IP del primer ordenador del esquema de la figura es `192.168.1.1`

Dado que no es fácil acordarse de este tipo de dirección, también se puede hacer corresponder a cada dirección IP un nombre más fácil de recordar. Sobre el esquema anterior, podemos comunicar con el ordenador de la derecha bien llamándolo por su dirección IP: `192.168.1.2`, o llamándolo por su nombre: `tortuga`.

No nos extendamos más sobre el significado de estos números. Añadamos únicamente una cosa que es interesante saber, el ordenador local en el cual se trabaja también se identifica por una dirección: `127.0.0.1`. El nombre que se asocia con él es habitualmente `localhost`.

12.2. Primitivas orientadas a la red

XLogo dispone de 4 primitivas que permiten comunicarse a través de una red: `escuchatcp`, `ejecutatcp`, `chattcp` y `enviatcp`.

En los ejemplos siguientes, mantendremos el esquema de red de la subsección anterior.

- `escuchatcp`: esta primitiva es la base de cualquier comunicación a través de la red. No espera ningún argumento. Permite poner al ordenador que la ejecuta a la espera de instrucciones dadas por otros ordenadores de la red.

- `ejecutatcp`: esta primitiva permite ejecutar instrucciones sobre otro ordenador de la red.

  Sintaxis: `ejecutatcp palabra lista` → La palabra indica la dirección IP o el nombre del ordenador de destino (el que va a ejecutar las órdenes), la lista contiene las instrucciones que hay que ejecutar.

  Ejemplo: desde el ordenador `liebre`, deseo trazar un cuadrado de lado 100 en el otro ordenador. Por tanto, hace falta que desde el ordenador `tortuga` ejecute la orden `escuchatcp`. Luego, desde el ordenador `liebre`, ejecuto:

  ```
  ejecutatcp "192.168.2.2 [repite 4 [avanza 100 giraderecha 90]]
  ```

  o

  ```
  ejecutatcp "tortuga [repite 4 [avanza 100 giraderecha 90]]
  ```

- `chattcp`: permite chatear entre dos ordenadores de la red, abriendo una ventana en cada uno que permite la conversación.

  Sintaxis: `chattcp palabra lista` → La palabra indica la dirección IP o el nombre del ordenador de destino, la lista contiene la frase que hay que mostrar.

  Ejemplo: `liebre` quiere hablar con `tortuga`.
  `tortuga` ejecuta `escuchatcp` para ponerse en espera de los ordenadores de la red.
  `liebre` ejecuta entonces: `chattcp "192.168.1.2 [buenos días]`.
  Una ventana se abre en cada uno de los ordenadores para permitir la conversación

- `enviatcp`: envía datos hacia un ordenador de la red.

  Sintaxis: `enviatcp palabra lista` → La palabra indica la dirección IP o el nombre del ordenador de destino, la lista contiene los datos que hay que enviar. Cuando XLogo recibe los datos en el otro ordenador, responderá `Si` , que podrá asignarse a una variable o mostrarse en el **Histórico de comandos**.

  Ejemplo: `tortuga` quiere enviar a `liebre` la frase `"3.14159 casi el número pi"`.
  `liebre` ejecuta `escuchatcp` para ponerse en espera de los ordenadores de la red.
  Si `tortuga` ejecuta entonces: `enviatcp "liebre [3.14159 casi el número pi]`,
  `liebre` mostrará la frase, pero en `tortuga` aparecerá el mensaje:
  `¿Qué hacer con [ Si ] ?`
  Deberíamos escribir:
  `es enviatcp "liebre [3.14159 casi el número pi]`
  o
  `haz "respuesta enviatcp "liebre [3.14159 casi el número pi]`
  En el primer caso, el **Histórico de comandos** mostrará `Si`, y en el segundo `"respuesta` contendrá la lista `[ Si ]`, como podemos comprobar haciendo
  `es lista? :respuesta`

```
cierto
es :respuesta
Si
```

Con esta primitiva se puede establecer comunicación con un robot didáctico a través de su interfaz de red. En este caso, la respuesta del robot puede ser diferente, y se podrán decidir otras acciones en base al contenido de `:respuesta`.

Un pequeño truco: lanzar dos veces XLogo en un mismo ordenador.

- En la primera ventana, ejecuta `escuchatcp`.
- En la segunda, ejecuta

```
ejecutatcp "127.0.0.1 [repite 4 [avanza 100 giraderecha 90]]
```

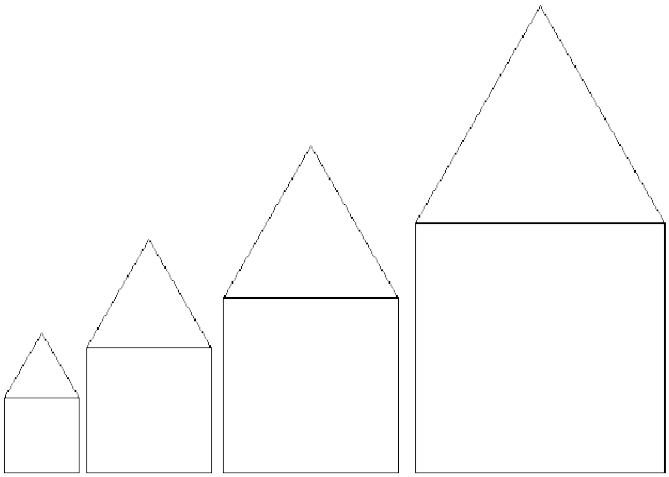
¡Así puedes mover a la tortuga en la otra ventana! (¡Ah sí!, esto es así porque `127.0.0.1` indica tu dirección local, es decir, de tu propio ordenador)

Capítulo 13

Ejemplos de programas

13.1. Dibujar casas

```
para casa :c
 repite 4 [
   avanza (20*:c) giraderecha 90 ]
 avanza (20*:c)
 giraderecha 30
 repite 3 [
   avanza (20*:c) giraderecha 120 ]
fin

para colocar :c
 subelápiz
 giraizquierda 30
 retrocede (:c*20)
 giraderecha 90
 avanza (:c*22)
 giraizquierda 90
 bajalápiz
fin

para casas
 borrapantalla
 ocultatortuga
 subelápiz
 giraizquierda 90
 avanza 200
 giraderecha 90
 bajalápiz
 repitepara [n 3 7 2]
   [ casa :n colocar :n ]
 casa 10
fin
```



13.2. Dibujar un rectángulo sólido

```
para rect :alto :largo
 si :alto = 0 | :largo = 0 [alto]
 repite 2 [
    avanza :alto
    giraderecha 90
    avanza :largo
    giraderecha 90 ]
 rect :alto -1 :largo -1
fin
```

13.3. Factorial

Recordatorio: `5! = 5 * 4 * 3 * 2 * 1`

```
para factorial :n
  si :n = 1
   [devuelve 1]
   [devuelve :n * factorial (:n - 1)]
fin
```

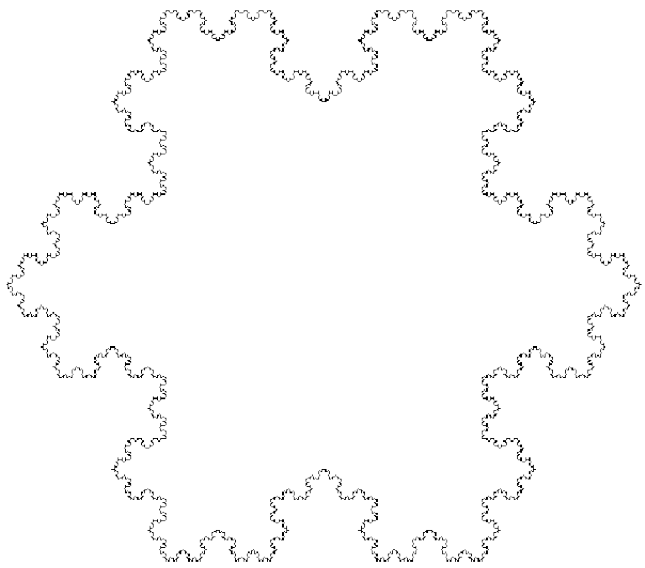
Ejemplo:

```
escribe factorial 5 --> 120.0
escribe factorial 6 --> 720.0
```

13.4. Copo de nieve (Gracias a Georges Noël)

```
para copo :orden :lar
  si (:orden < 1) | (:lar < 1)
   [av :lar alto]
  copo :orden-1 :lar/3
  giraizquierda 60
  copo :orden-1 :lar/3
  giraderecha 120
  copo :orden-1 :lar/3
  giraizquierda 60
  copo :orden-1 :lar/3
fin

para coponieve :orden :lar
  repite 3 [
    giraderecha 120
    copo :orden :lar ]
fin
```

Ej: `coponieve 5 450`



13.5. Escritura

```
para escribir
  ocultatortuga
```

```
  repite 40 [
    avanza 30
    giraderecha 9
    poncolorlapiz azar 7
    rotula [XLogo es genial!] ]
fin
```

13.6. Conjugación (sólo verbos regulares)

13.6.1. Primera versión

```
para futuro :verbo
  es frase "yo palabra :verbo "é
  es frase "tú palabra :verbo "ás
  es frase "él palabra :verbo "á
  es frase "nosotros palabra :verbo "emos
  es frase "vosotros palabra :verbo "éis
  es frase "ellos palabra :verbo "án
fin
```

Ejemplo: `futuro "amar`

```
yo amaré
tú amarás
él amará
nosotros amaremos
vosotros amaréis
ellos amarán
```

13.6.2. Segunda versión

```
para futuro :verbo
  haz "pronombres [yo tú él nosotros vosotros ellos]
  haz "terminaciones [é ás á emos éis án]
  repitepara [i 1 6]
   [ es fr elemento :i :pronombres palabra :verbo elemento :i :terminaciones ]
fin
```

Ejemplo: `futuro "amar`

```
yo amaré
tú amarás
él amará
nosotros amaremos
vosotros amaréis
ellos amarán
```

13.6.3. Tercera versión (con recurrencia)

```
para futuro :verbo
  haz "pronombres [yo tú él nosotros vosotros ellos]
  haz "terminaciones [é ás á emos éis án]
  conjugar :verbo :pronombres :terminaciones
fin


para conjugar :verbo :pronombres :terminaciones
  si vacio? :pronombres [alto]
  es fr primero :pronombres palabra :verbo primero :terminaciones
  conjugar :verbo mp :pronombres mp :terminaciones
fin
```

Ejemplo: `futuro "amar`

```
yo amaré
tú amarás
él amará
nosotros amaremos
vosotros amaréis
ellos amarán
```

13.7. Colores

13.7.1. Introducción

Primero, algunas aclaraciones: Habrás visto en la sección 5.1.4 que el comando `poncl` puede tomar como argumento tanto un número como una lista. Aquí nos centraremos en codificar valores RVA. Cada color en XLogo está codificado usando tres valores: rojo, verde y azul, de ahí RVA (RGB en inglés).

Estos tres números conforman una lista que es argumento de la primitiva `poncl`, por lo que representan respectivamente los componentes rojo, verde y azul de un color. Esta

manera de codificar no es muy intuitiva, así que para tener una idea del color que obtendrás puedes usar la caja de diálogo **Herramientas → Elegir color del lápiz**.

Sin embargo, usando esta forma de codificar colores, se hace muy fácil transformar una imagen. Por ejemplo, si quieres convertir una foto color en escala de grises, puedes cambiar cada punto (píxel) de la imagen a un valor promedio de los 3 componentes RVA. Imagina que el color de un punto de la imagen está dado por `[0 100 80]`. Calculamos el promedio: `(0 + 100 + 80)/3 = 60`, y asignamos el color `[60 60 60]` a este punto. Esta operación debe ser realizada para cada punto de la imagen.

13.7.2. Práctica: Escala de grises

Vamos a transformar una imagen color de 100 por 100 a escala de grises. Esto significa que tenemos `100 * 100 = 10000` puntos a modificar.

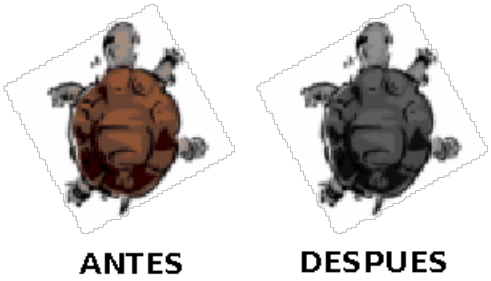
La imagen de ejemplo utilizada aquí está disponible en la siguiente dirección:

`http://xlogo.tuxfamily.org/images/transfo.png`

Así es como vamos a proceder: primero, nos referiremos al punto superior izquierdo como `[0 0]`. Luego, la tortuga examinará los primeros 100 puntos (pixeles) de la primera línea, seguidos por los primeros 100 de la segunda, y así sucesivamente. Cada vez tomaremos el color del punto usando `encuentracolor`, y el color será cambiado por el promedio de los tres `[r v a]` valores. Aquí está el código principal: (No olvides cambiar la ruta del archivo en el procedimiento!)

```
para transform
# Debes cambiar la ruta de la imagen transfo.png
# Ej: cargaimagen [/home/usuario/imagenes/transfo.png]
  borrapantalla ocultatortuga
  pondirectorio "/home/usuario/imagenes
  cargaimagen "transfo.png
  escalagris
fin

para escalagris
  repitepara [y 0 -100 -1]
    [ repitepara [x 0 100]
# asignamos el promedio de color del punto al color del lapiz
      [ poncolorlapiz pixel encuentracolor lista :x :y
# convertimos el punto escala de grises
        punto lista :x :y ] ]
fin

para pixel :lista1
# devuelve el promedio de los 3 numeros [r v a]
  haz "r primero :lista1
  haz "lista1 menosprimero :lista1
  haz "v primero :lista1
  haz "lista1 menosprimero :lista1
  haz "a primero :lista1
  haz "color redondea (:r+:v+:a)/3
```

```
  devuelve frase :color frase :color :color
fin
```

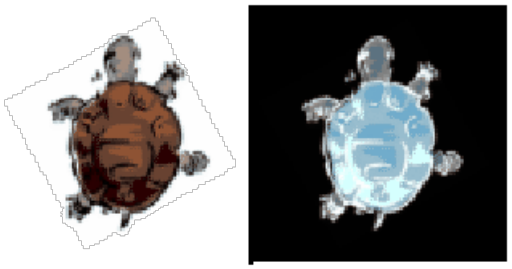


13.7.3. Negativo

Para cambiar una imagen a su negativo, se puede usar el mismo proceso de la escala de grises, excepto que en lugar de hacer el promedio de los números `[r v a]`, los reemplazamos por su complemento, o sea la diferencia a 255.
Ejemplo: Si un punto (píxel) tiene un color `[2 100 200]`, lo reemplazamos con `[253 155 55]`. Podríamos usar el mismo código que en el ejemplo anterior, cambiando únicamente el procedimiento `pixel`, pero veamos un procedimiento recursivo:

```
para transform2
# Debes cambiar la ruta de la imagen transfo.png
# Ej: c:\Mis Documentos\Mis imagenes\transfo.png
  borrapantalla
  ocultatortuga
  pondirectorio "c:\\Mis\ Documentos\\Mis\ imagenes
  cargaimagen "transfo.png
  negativo 0 0
fin

para negativo :x :y
  si :y = -100
   [ alto ]
   [ si :x = 100
     [ haz "x 0 haz "y :y-1]
     [ poncolorlapiz pixel2 encuentracolor lista :x :y
       punto lista :x :y ] ]
  negativo :x+1 :y
fin

para pixel2 :lista1
# devuelve el promedio de los 3 numeros [r v a]
  haz "r primero :lista1
  haz "lista1 menosprimero :lista1
  haz "v primero :lista1
  haz "lista1 menosprimero :lista1
  haz "a primero :lista1
  devuelve frase (255 - :r) frase (255 - :v) (255 - :a)
```

```
fin
```



13.8. Listas (Gracias a Olivier SC)

Supongo que apreciarás este hermoso programa:

```
para revertir :w
  si vacio? :w
    [devuelve "]
    [devuelve palabra ultimo :w revertir menosultimo :w ]
fin

para palindromo :w
 si :w = revertir :w
   [devuelve "cierto]
   [devuelve "falso]
fin

para palin :n
  si palindromo :n
   [escribe :n alto]
   [haz "texto suma :n revertir :n
    haz "texto frase "igual\ a :texto
    haz "texto frase revertir :n :texto
    haz "texto frase "mas :texto
    haz "texto frase :n :texto
    escribe :texto
    palin :n + revertir :n ]
fin
```

Ejemplo: `palin 78`

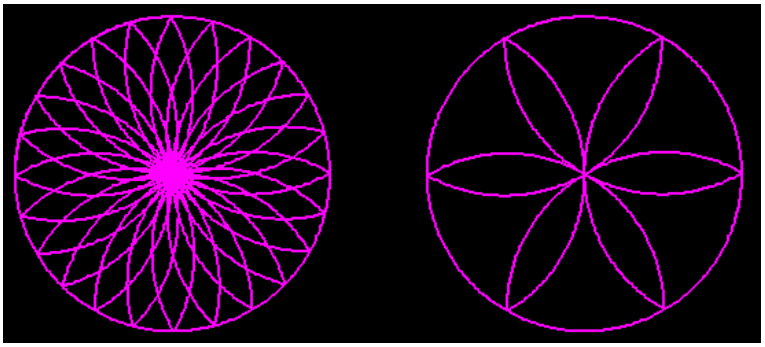
```
78 mas 87 igual a 165
165 mas 561 igual a 726
726 mas 627 igual a 1353
1353 mas 3531 igual a 4884
4884
```

13.9. Un lindo medallón

```
para roset
  pongrosor 2
  repite 6 [
    repite 60
     [avanza 2 giraderecha 1]
    giraderecha 60
    repite 120
     [avanza 2 giraderecha]
    giraderecha 60 ]
  pongrosor 1
fin

para roseton
  roset
  repite 30
    [avanza 2 giraderecha 1]
  roset
  repite 15
    [avanza 2 giraderecha 1]
  roset
  repite 30
    [avanza 2 giraderecha 1]
  roset
fin
```

Ejemplo:

```
borrapantalla ocultatortuga
poncolorpapel 0 poncolorlapiz 5
roset
subelapiz ponposicion [-300 0] bajalapiz
ponrumbo 0 roseton
```



Capítulo 14

Acerca de XLogo

14.1. Desinstalar

Para desinstalar XLogo, todo lo que hace falta es borrar el archivo `XLogo.jar` y el archivo de configuración `.xlogo` que se encuentra en `/home/tu_nombre` en Linux, o `c:\windows\.xlogo` en Windows.

14.2. El sitio

Para conseguir la última versión y corrección de errores, visita el sitio de XLogo de vez en cuando:

`http://xlogo.tuxfamily.org`

Siéntete libre de contactar al autor si tienes algún problema con la instalación o el uso. Toda sugerencia será bienvenida.

14.3. Acerca de este documento

XLogo: Manual del Usuario

- Original en francés revisado por Loïc (18 de Febrero de 2008).
- Traducido al español por Marcelo Duschkin y Álvaro Valdés (14 de junio de 2008)
- Kevin Donnelly: traducción al inglés del manual, y traducción al galés del programa.

Agradecimientos

- Olivier SC: por sus sugerencias, y por las invalorables pruebas que me permitieron depurar el intérprete XLogo.
- Daniel Ajoy, por sus sugerencias en cuanto a la compatibilidad de las primitivas en español y su valiosa colaboración en pruebas de esa versión.

Capítulo 15

Carnaval de Preguntas – Artimañas – Trucos que conocer

15.1. Preguntas frecuentes

Por más que borro un procedimiento en el Editor, reaparece todo el tiempo!

Cuando se sale del **Editor**, éste se limita únicamente a guardar o poner al día los procedimientos definidos en él. La única forma de borrar un procedimiento en XLogo es utilizar la primitiva `borra` o `bo`.

Ejemplo: `borra "toto` → borra el procedimiento `toto`

¿Cómo hago para escribir rápidamente una orden utilizada previamente?

- Primer método: con el ratón, haz *click* en la zona del **Histórico** sobre la línea deseada. Así reaparecerá inmediatamente en la **Línea de Comando**
- Segundo método: con el teclado, las flechas Arriba y Abajo permiten navegar por la lista de los comandos anteriores (más práctico)

En la opción Sonido del cuadro de diálogo Preferencias, no hay disponible ningún instrumento

Como decíamos en la sección 3.3, esto se debe a que la versión de Java para Windows no incluye los *bancos de sonido*, y deben instalarse manualmente.

En primer lugar, hay que descargarlos desde:
`http://java.sun.com/products/java-media/sound/soundbank-min.gm.zip`
la versión mínima (unos 350 kb),
`http://java.sun.com/products/java-media/sound/soundbank-mid.gm.zip`
la versión intermedia (algo más de 1 Mb) y
`http://java.sun.com/products/java-media/sound/soundbank-deluxe.gm.zip`
la versión *de luxe* (casi 5 Mb).

Una vez descargados, debemos descomprimirlos en el directorio `audio` de la instalación Java que, dependiendo de la versión, puede ser:

```
C:\Archivos de programa\Java\jre1.6.0\lib\audio
```

creando el directorio `audio` si éste no existe.

Hecho esto, la lista de instrumentos estará disponible.

Tengo problemas de refresco de pantalla cuando la tortuga dibuja!

Problema también conocido y típico de JRE >1.4.2. intentaré ponerle remedio en lo sucesivo, quizá pueda hacer algo. Dos soluciones por el momento:

- Minimizar la ventana y volver a aumentarla (restaurarla)
- Utilizar siempre la versión más moderna de JAVA.

Utilizo la versión en Esperanto, pero no puedo escribir los caracteres especiales

Cuando escribes en la **Línea de comandos** o en el **Editor**, si haces *click* con el botón derecho del ratón, aparece un menú contextual. En ese menú se encuentran las funciones habituales de Edición (copiar, cortar, pegar) y los caracteres especiales del Esperanto cuando se selecciona ese idioma.

Utilizo la versión en Español, y no puedo utilizar las primitivas `animacion`, `division`, `separacion` y `ponseparacion`

Corregido desde la versión `0.9.20e`. Para versiones anteriores de XLOGO:

- `animacion`, se escribe `animacicn`, con "`c`" en lugar de "`o`".
- `division`, `separacion` y `ponseparacion` se escriben con tilde: `división`, `separación` y `ponseparación`.

Obviamente, el mejor consejo es que actualices a la versión más moderna de XLOGO.

Uso Windows XP y tengo correctamente instalado y configurado el JRE; pero hago doble *click* en el icono de XLogo y no pasa nada!!

A veces en la primera ejecución de XLOGO en Windows XP pasa eso. Dos opciones:

- Utiliza la versión `xlogo-new.jar` también disponible en nuestra web.
- Si presionas `Alt+Contrl+Supr` y en el **Gestor de Procesos** "matas" el correspondiente a `javaw`, se inicia XLOGO. Desde ese momento, funciona correctamente haciendo "doble *click*" sobre el icono del archivo `xlogo.jar`.

15.2. ¿Cómo puedo ayudar?

- Informándome de todos los errores ("*bugs*") que encuentres. Si puedes reproducir sistemáticamente un problema detectado, mejor aún
- Toda sugerencia dirigida a mejorar el programa es siempre bienvenida
- Ayudando con las traducciones, en particular el inglés ...
- Las palabras de ánimo siempre vienen bien

Índice alfabético