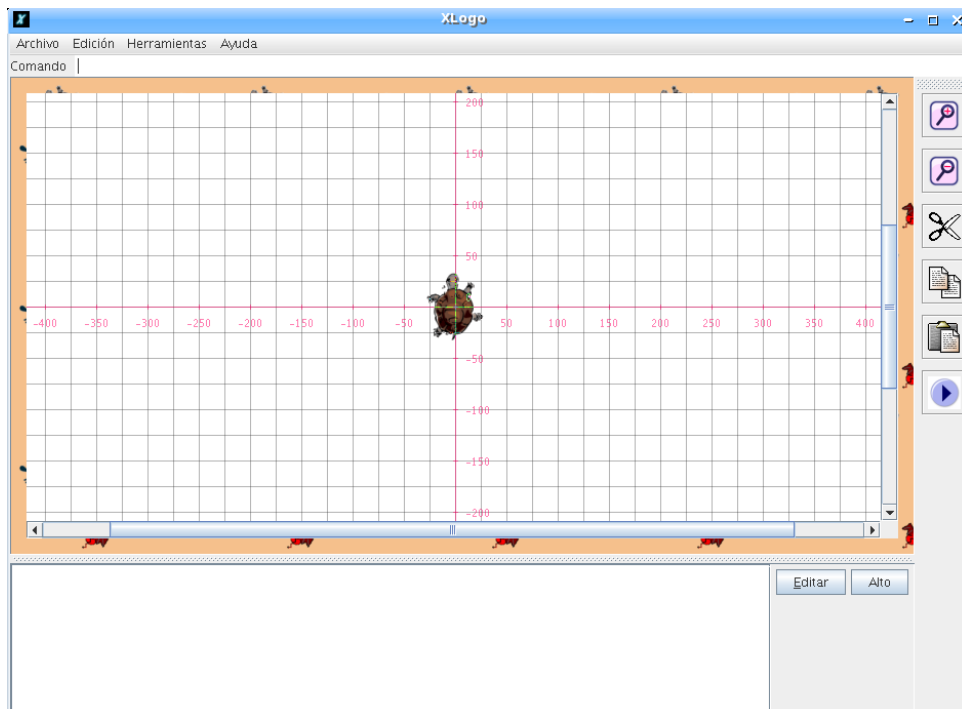




Curso de Iniciación

Álvaro Valdés Menenéndez
Loïc le Coq
Marcelo Duschkin

<http://xlogo.tuxfamily.org>





Introducción

LOGO es un lenguaje desarrollado a finales de los años 60 por Seymour Papert. Papert trabajó con Piaget en la Universidad de Ginebra desde 1959 hasta 1963 y, basándose en su Teoría del Constructivismo, desarrolló el aprendizaje Construcccionista:

El aprendizaje mejora si se aplica activamente a la vida cotidiana

resumido en la expresión *learning-by-doing*

LOGO es una potente herramienta para desarrollar los procesos de pensamiento lógico-matemáticos y un lenguaje excelente para comenzar a estudiar programación, que 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. Poder usar órdenes en el idioma natural favorece su aprendizaje y asimilación. También es posible operar con palabras y listas.

LOGO es un *lenguaje interpretado*. Esto quiere decir que las órdenes introducidas por el usuario son interpretadas por el ordenador y ejecutadas inmediatamente en el orden en que son escritas.

XLOGO es un intérprete LOGO escrito en JAVA. Actualmente (versión 0.9.93a) soporta diez idiomas (Francés, Inglés, Español, Portugués Galés, 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 y puede descargarse desde:

<http://xlogo.tuxfamily.org>

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 o MacOS, XLOGO funcionará sin problemas. Recientemente JAVA ha sido liberada bajo licencia GPL, lo que también garantiza su disponibilidad y gratuidad.

A lo largo del manual se irán planteando ejercicios y resolviendo ejemplos. Hemos elegido **no utilizar** tildes ni *eñes* en los mismos debido a que en la mayoría de los lenguajes de programación no las aceptan, pero XLOGO las admite y trabaja perfectamente con ellas, permitiendo el uso acentuado tanto de primitivas, como de variables y procedimientos.

Índice general

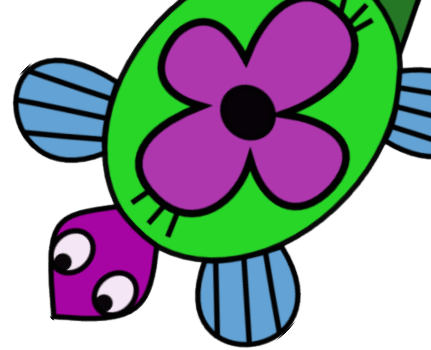
1. Características de la Interfaz	1
1.1. Instalación de JAVA	1
1.2. Ejecutar y configurar XLOGO	1
1.2.1. En Windows	1
1.2.2. En Mac y Solaris	3
1.2.3. En Linux	3
1.3. Primera Ejecución	4
1.4. La ventana principal	5
1.5. El editor de procedimientos	6
1.6. Salir	8
1.7. Reiniciar XLOGO	9
1.8. Desinstalar	9
1.9. Convenciones adoptadas para XLOGO	9
1.9.1. El caracter especial \	9
1.9.2. Mayúsculas y minúsculas	10
1.9.3. Las tildes	10
2. Opciones del Menú	11
2.1. Menú “ <i>Archivo</i> ”	11
2.2. Menú “ <i>Edición</i> ”	13
2.3. Menú “ <i>Herramientas</i> ”	13
2.4. Menú “ <i>Ayuda</i> ”	17
3. Presentación de la tortuga	19
3.1. Un programa de ejemplo	19
3.2. Ejercicios	23
3.3. Una ayuda al dibujo	24
4. Iniciación a la Geometría de la tortuga	27
4.1. Descripción de las primitivas o comandos	27
4.2. Movimientos de la tortuga	28
4.3. Ejercicios	29
4.4. Avanzando un poco	30

4.5. Ejercicios	31
5. Procedimientos y subprocedimientos	33
5.1. Procedimientos	33
5.2. Ejercicios	34
5.3. Sub-procedimientos	35
5.4. Ejercicios	36
6. Variables. Procedimientos con argumentos	37
6.1. Primitivas asociadas	37
6.2. Procedimientos con variables	39
6.3. Ejercicios	40
6.4. Conceptos acerca de variables	41
6.5. Desde la Línea de Comandos	43
6.5.1. La primitiva <code>define</code>	43
6.5.2. Las primitivas <code>borra</code> y <code>borratodo</code>	43
6.5.3. La primitiva <code>listaprocs</code>	44
6.5.4. La primitiva <code>ejecuta</code>	44
7. Operaciones	45
7.1. Operaciones binarias	45
7.1.1. Con números	45
7.1.2. Con listas	47
7.2. Ejercicios	49
7.3. Operaciones unitarias	51
7.4. Ejercicios	52
7.5. Cálculo superior	52
7.6. Ejercicios	53
7.7. Prioridad de las operaciones	54
8. Coordenadas y Rumbo	55
8.1. Cuadrícula y ejes	55
8.2. Coordenadas	56
8.3. Ejercicios	56
8.4. Rumbo	57
8.5. Ejercicios	58
9. Condicionales y Operaciones lógicas	59
9.1. Condicional	59
9.2. Operaciones Lógicas	60
9.3. Ejercicios	62
9.4. Booleanos	63
9.5. Ejercicios	64

10.Listas	65
10.1. Primitivas	65
10.2. Ejemplo: Conjugación	69
10.2.1. Primera versión	69
10.2.2. Segunda versión	69
10.2.3. Tercera versión	70
10.3. Ejercicios	70
10.4. Listas de Propiedades	72
11.Bucles y recursividad	73
11.1. Bucles	73
11.1.1. Bucle con <code>repite</code>	73
11.1.2. Bucle con <code>repitepara</code>	74
11.1.3. Bucle con <code>mientras</code>	75
11.1.4. Bucle con <code>paracada</code>	76
11.1.5. Bucle con <code>repitesiempre</code>	77
11.2. Ejemplo	77
11.3. Comandos de ruptura de secuencia	78
11.4. Ejercicios	79
11.5. Recursividad	82
11.6. Retomando el ejemplo	83
11.7. Ejercicios	84
11.8. Uso avanzado de procedimientos	85
11.8.1. La primitiva <code>devuelve</code>	85
11.8.2. Variables opcionales	85
11.8.3. La primitiva <code>trazado</code>	86
12.Recibir entrada del usuario	87
12.1. Comunicación con el usuario	87
12.2. Ejercicios	88
12.3. Propiedades del Histórico de Comandos	89
12.4. Ejercicios	90
12.5. Escritura en Pantalla	91
12.6. Ejercicios	92
12.7. Interactuar con el teclado	92
12.8. Ejercicios	93
12.9. Interactuar con el ratón	94
12.10Ejercicios	97
12.11Componentes Gráficos	98
12.11.1.Crear un componente gráfico	98
12.12Ejercicios	101

13. Técnicas avanzadas de dibujo	103
13.1. Más opciones para la tortuga	103
13.2. Control del color	105
13.2.1. Primitivas que controlan los colores	105
13.2.2. Descripción de los colores	106
13.2.3. Función avanzada de relleno	107
13.3. Ejercicios	109
13.4. Control del Área de dibujo	110
13.4.1. Control del dibujo	110
13.4.2. Control de las dimensiones	112
13.5. Ejercicios	113
13.6. Manejando imágenes	114
13.6.1. Introducción	114
13.6.2. Práctica: Escala de grises	115
13.6.3. Negativo	116
13.7. Ejercicios	117
14. Modo multitortuga y Animación	119
14.1. Multitortuga	119
14.1.1. Las primitivas	119
14.1.2. Ejemplo	120
14.2. Ejercicios	121
14.3. Animación	124
14.3.1. Ejemplo	124
14.4. Ejercicios	126
15. Manejo de Archivos	127
15.1. Las primitivas	127
15.1.1. Navegación por el sistema de archivos	127
15.1.2. Carga y guardado de procedimientos	128
15.1.3. Modificando archivos	130
15.2. Ejemplo avanzado: obtención aproximada de π	131
15.3. Complicuemos un poco más: π que genera π	133
16. Geometría de la tortuga en 3-D	135
16.1. La tortuga en Tres Dimensiones	135
16.1.1. La proyección en perspectiva	135
16.1.2. Entender la orientación en el mundo tridimensional	136
16.1.3. Primitivas	136
16.2. Primitivas disponibles tanto en 2D como 3D	138
16.3. Primitivas sólo disponibles en 3D	139
16.4. Ejercicios	140
16.5. El Visor 3D	140

16.5.1. Reglas	141
16.6. Ejemplos	143
17. Tocar música (MIDI)	147
17.1. Las primitivas	148
17.2. Ejercicios	149
18. Gestión de tiempos	151
18.1. Las primitivas	151
18.2. Actividad sobre las cifras de una calculadora	153
18.2.1. El programa	153
18.2.2. Creación de una pequeña animación	154
18.3. Ejercicios	155
19. Utilización de la red con XLogo	157
19.1. La red: ¿cómo funciona eso?	157
19.2. Primitivas orientadas a la red	158
20. Acerca de XLogo	160
20.1. El sitio	160
20.2. Acerca de este documento	160
21. Carnaval de Preguntas – Artimañas – Trucos que conocer	161
21.1. Preguntas frecuentes	161
21.2. ¿Cómo puedo ayudar?	162
22. Listado alfabético de primitivas	163
A. Iniciando XLogo desde la línea de comandos	182
B. Ejecutando XLogo desde la <i>web</i>	184
B.1. El problema	184
B.2. Cómo crear un fichero <code>.jnlp</code>	184
C. Programación Estructurada y Diseño Modular	186
C.0.1. Programación Estructurada	186
C.0.2. Diseño Modular	186



Capítulo 1

Características de la Interfaz

1.1. Instalación de Java

Antes que nada, debes tener instalado el JRE (*Java Runtime Environment*) en tu ordenador. Si no lo tienes instalado, lo puedes descargar de la página web de Sun (es libre y gratuito):

```
http://java.com/es/download/manual.jsp
```

Debes descargar la versión que corresponda a tu sistema operativo (Linux, Windows, ...). Si usas Linux es posible que JRE ya esté instalado. Para confirmarlo, escribe en una consola:

```
java -version
```

Si no responde con un error, Java está listo.

Lo siguiente entonces es descargar XLOGO desde

```
http://xlogo.tuxfamily.org/
```

eligiendo entre `xlogo.jar` y `xlogo-new.jar` según la versión de JAVA instalada. Desde este momento trabajaremos con esta última, ya que es la asociada a la última versión liberada por Sun.

1.2. Ejecutar y configurar XLogo

1.2.1. En Windows

En teoría, al hacer doble click en el archivo `xlogo-new.jar`, el programa debería iniciarse. Si es así, pasa a la siguiente sección.

Si no es así, y se ejecuta otra aplicación (WinZIP, WinRAR, . . .), eso se debe a que en realidad los archivos `.jar` (como se presenta XLOGO) son archivos comprimidos equivalentes a `.zip`. A veces, puedes darte cuenta de ello antes de hacer *click* según el icono que tenga el archivo:



Se abre con JAVA



Se abre con WinRAR



Se abre con WinZIP

La forma más fácil de solucionarlo consiste en hacer *click* con el botón derecho sobre el archivo, seleccionar

Abrir con . . .

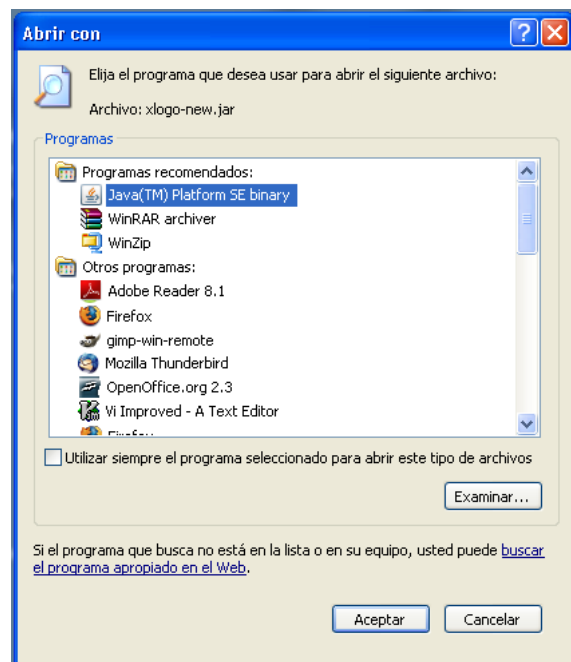
en el menú elegir JAVA y activar

Utilizar siempre . . .

siendo esta la forma recomendada.

Si no funciona, las alternativas son:

- eliminar las extensiones `.jar` en la configuración del compresor
- reinstalar JAVA



Finalmente, conviene incorporar las librerías asociadas a la música. La versión Windows de `jre` no incorpora los *bancos de sonido* que contienen los instrumentos, y 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 este no existe.

Hecho esto, la lista de instrumentos estará disponible.

1.2.2. En Mac y Solaris

Las cosas en un Mac son aún más fáciles. Desde hace tiempo Apple incorpora JAVA en sus sistemas operativos, así que simplemente haciendo *click* sobre el archivo `xlogo-new.jar` debería ejecutarse XLOGO.

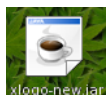


Solaris, por su parte, utiliza el escritorio *Java Desktop System*, en el que JAVA está perfectamente configurado.

1.2.3. En Linux

Las cosas en Linux pueden ser muy fáciles si la distribución que utilizas incorpora JAVA (algo bastante probable desde que Sun la liberó bajo licencia GPL). Si es así, simplemente haciendo doble *click* sobre el archivo `xlogo-new.jar` debería iniciarse XLOGO.

De nuevo, los iconos nos dan una pista cuando JAVA está instalado y configurado:



En KDE



En GNOME

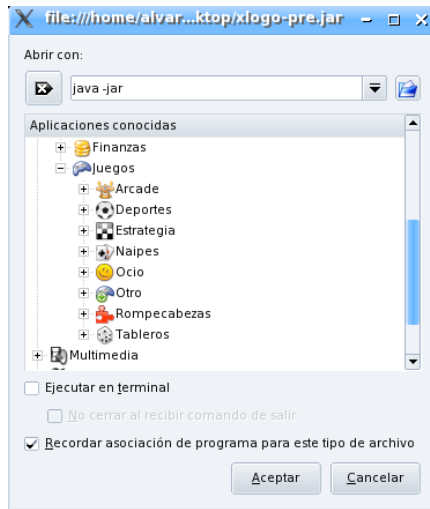
Si no, tras instalar JAVA, debes conocer en qué directorio se encuentra el ejecutable, habitualmente `/usr/bin/` o `/usr/java/` seguido de la versión instalada, por ejemplo:

```
/usr/java/jre1.6.0/bin/java
```

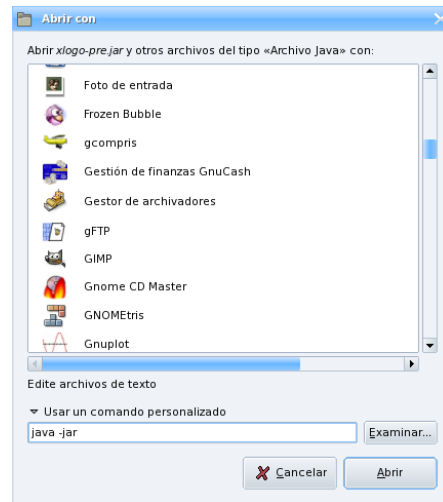
A continuación, descargamos XLOGO (por ejemplo en el escritorio), hacemos *click* con el botón derecho sobre el archivo `xlogo-new.jar` → **Abrir con** → **Otros ...** y en el cuadro de diálogo **Abrir con:** escribir

```
/usr/java/jre1.6.0/bin/java -jar
```

y en KDE activar **Recordar asociación de programa** para este tipo de archivo.



En KDE



En GNOME

A diferencia de lo que ocurre en Windows, Linux permite cambiar el icono únicamente al archivo `xlogo-new.jar` sin afectar a otros archivos `jar`, así que puedes elegir tu tortuga favorita, por ejemplo entre las cinco que puedes usar en XLOGO o incluso el dragón galés.



Este procedimiento puede variar ligeramente según uses **KDE**, **GNOME**, ... pero como ves, configurar Linux no es tan difícil como algunos creen.

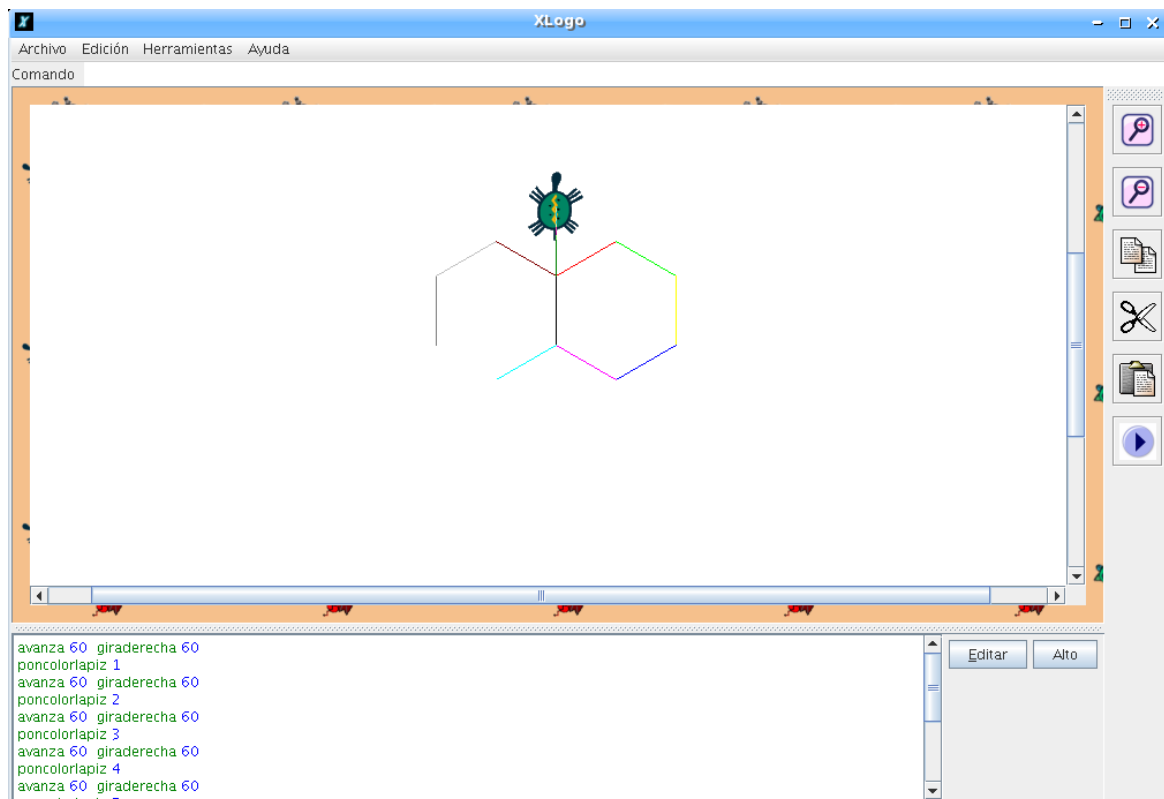
1.3. Primera Ejecución

La primera vez que ejecutes XLOGO (o si has borrado el fichero `.xlogo` – ver sección 1.8) deberás elegir el idioma con que quieres trabajar, seleccionando la bandera correspondiente y haciendo *click* en el botón OK.

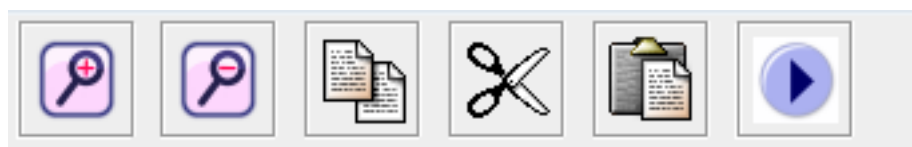


Esta elección no es definitiva; puedes elegir otro idioma en cualquier momento desde las opciones de menú (sección 2.3)

1.4. 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:

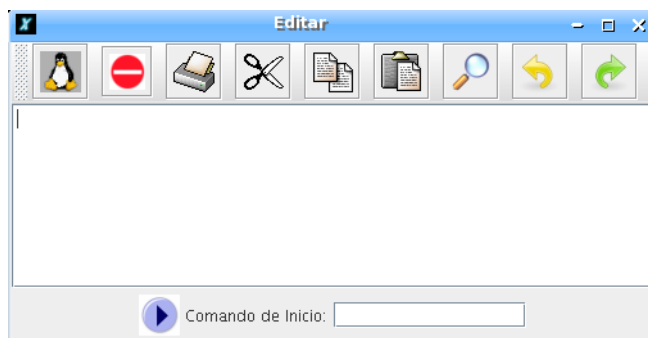


zoom (acercar y alejar), copiar, cortar, pegar y Comando de Inicio.

- 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.

1.5. El editor de procedimientos



Hay cuatro maneras de abrir el editor:



- Escribir `editatodo` o `edtudo` 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:

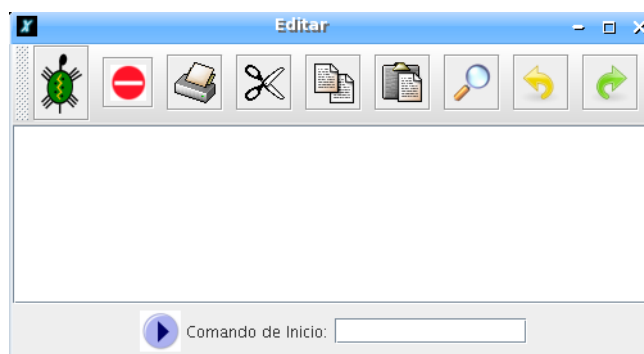

```
edita [proc_1 proc_2]
```
- Hacer *click* en el botón **Editar**.
- Usar el atajo de teclado **Alt+E**

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.
	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 2.3):



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.

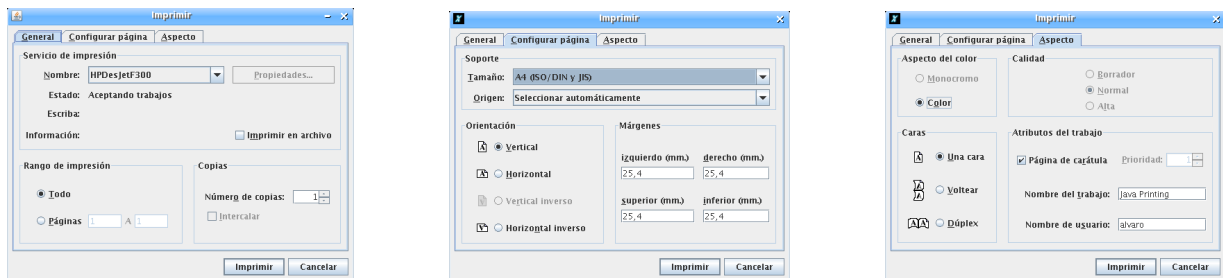
Al pulsar el botón, se ejecuta inmediatamente el **Comando de Inicio** sin necesidad de escribirlo en la Línea de Comandos, lo que es útil para:

- 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

1.6. Salir

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



Pulsamos **Sí** y termina la ejecución.

1.7. Reiniciar XLogo

Si en algún momento deseamos devolver al intérprete XLOGO a sus valores por defecto:

Color del lápiz:	negro	Color del papel:	blanco
Modo animación:	falso	Tamaño de fuente:	12 puntos
Forma del lápiz:	cuadrado	Calidad del dibujo:	normal
Número tortugas:	16	Tipo tortuga:	0

disponemos de las primitivas `inicializa` y `reponetodo` que, tecleadas en la Línea de comandos hacen que la tortuga “olvide” los ajustes realizados hasta ese momento.

1.8. 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.

1.9. Convenciones adoptadas para XLogo

Esta sección define aspectos “especiales” acerca del lenguaje XLOGO.

1.9.1. El caracter especial \

El caracter `\` (barra invertida o *backslash*) permite que las “palabras” (sección 6.1) contengan espacios o saltos de línea.

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

Ejemplos:

```

escribe "xlogo\ xlogo" produce xlogo xlogo
escribe "xlogo\_xlogo" 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 (capítulo 15). 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.

Del mismo modo, permite obtener símbolos con un significado especial en XLOGO:

```
\( → (      \) → )      \[ → [      \] → ]      \# → #      \" → "
```

1.9.2. Mayúsculas y minúsculas

XLOGO no distingue entre mayúsculas y minúsculas en el caso de nombres de procedimientos y primitivas. Así, la primitiva `pondirectorio` utilizada antes, si escribes `PONDIRECTORIO` o `PoNDiReCToRio`, el intérprete de comandos interpretará y ejecutará correctamente `pondirectorio`. Por otro lado, XLOGO sí respeta mayúsculas y minúsculas en listas y palabras:

```
escribe "Hola
```

proporciona

```
Hola
```

(la H inicial se mantuvo)

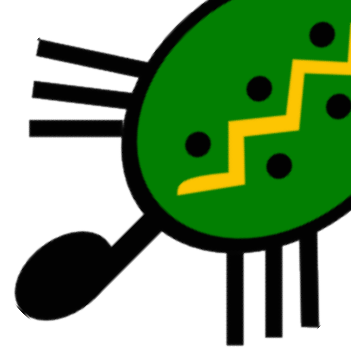
1.9.3. Las tildes

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 avisar de que por compatibilidad con otras versiones se mantiene la posibilidad de usar primitivas sin tilde, si bien recomendamos el uso acentuado de las mismas.

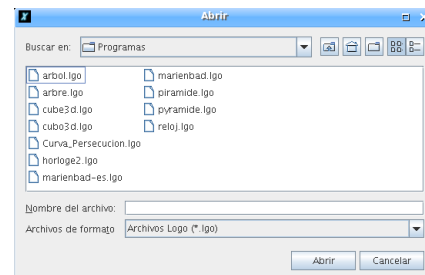
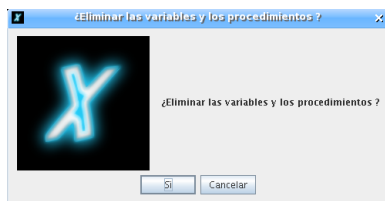


Capítulo 2

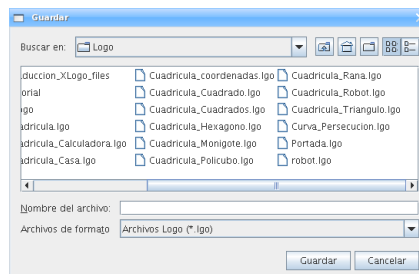
Opciones del Menú

2.1. 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 procedimientos y variables:

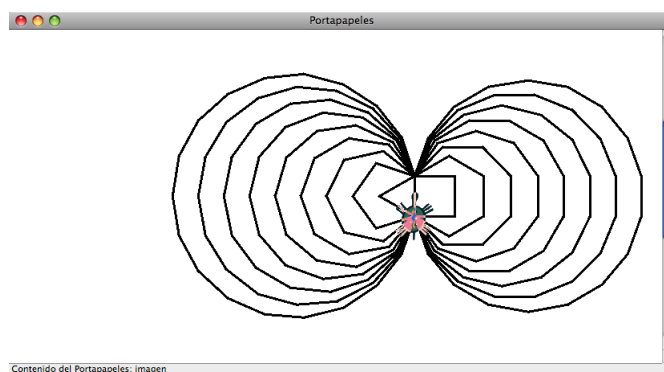


- **Archivo** → **Abrir**: Carga un archivo LOGO previamente guardado en disco.
- **Archivo** → **Guardar como...**: Graba un archivo (.lgo) de procedimientos definidos hasta ese momento en el disco, permitiendo asignarle un nombre.



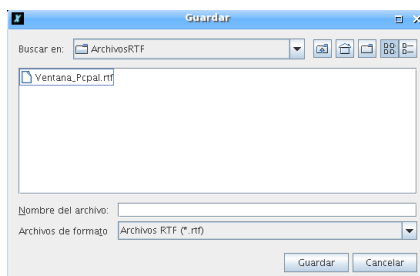
- **Archivo** → **Guardar**: Graba un archivo (.lgo) con los procedimientos definidos hasta ese momento en el disco. Esta opción estará deshabilitada mientras no se le haya asignado un nombre como se acaba de explicar en el punto anterior.

- **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.



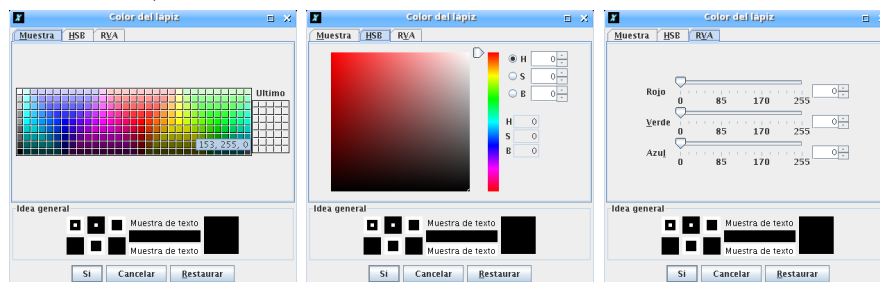
- **Archivo** → **Salir**: Finaliza la ejecución de XLOGO. También puede terminarse la ejecución desde la Línea de comandos con la primitiva `adios`.

2.2. 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.

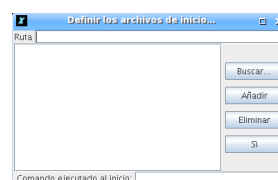
2.3. Menú “Herramientas”

- **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).



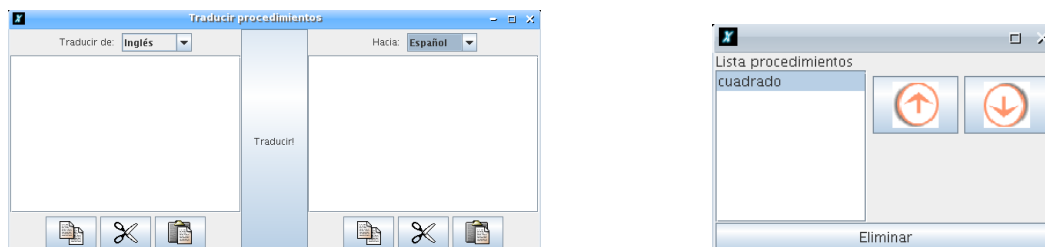
También accesible con el comando `poncolorlapiz`. (Sección 13.2.1)

- **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 13.2.1)
- **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.

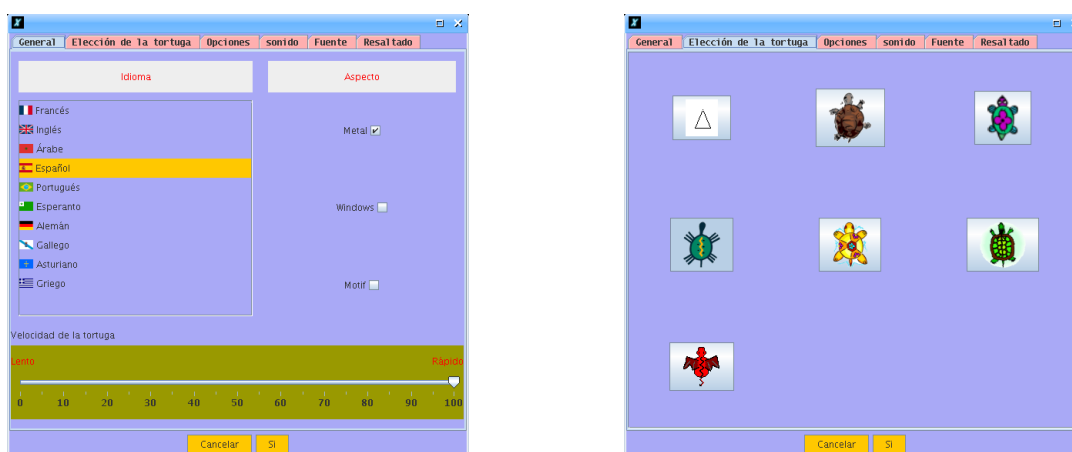
- **Herramientas** → **Traducir procedimientos**: Abre una caja de diálogo que permite traducir los comandos XLOGO entre dos idiomas. Es muy útil, en particular, cuando se obtienen códigos LOGO en inglés (de internet, por ejemplo) para expresarlos en el idioma deseado (Español, Francés, ...)



- **Herramientas** → **Borra procedimientos**: Abre una caja de diálogo que permite seleccionar 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:

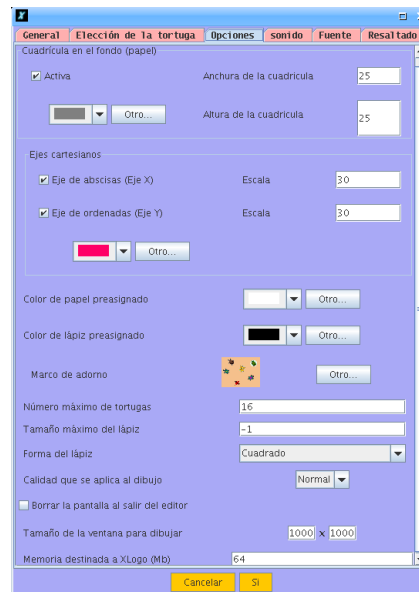
- **General:**

- **Idioma**: Permite elegir entre Francés, Inglés, Español, Galés, Portugués, Esperanto, Árabe, Gallego, Asturiano y Griego. 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 14.1)

- Opciones:

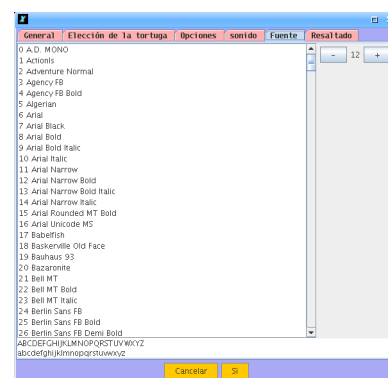
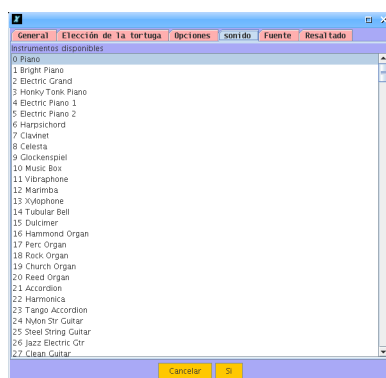


- **Cuadrícula en el fondo:** Establece (o elimina) una cuadrícula en el fondo del Área de dibujo, así como las medidas de la misma. También accesible con las primitivas `cuadrícula` y `borracuadrícula`. (Sección 8)
- **Ejes cartesianos:** Muestra (o retira) los ejes cartesianos (X e Y) del Área de dibujo, establece su escala (separación entre marcas) y su color. También accesible con las primitivas `ejes`, `borraejes`, `ejex`, `ejey` y `poncolorejes` (Sección 8).
- **Color de papel preasignado:** Permite elegir el color por defecto del papel, es decir, el mostrado al iniciar XLOGO.
- **Color de lápiz preasignado:** 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.



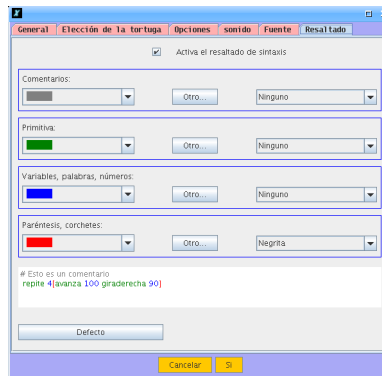
- **Número máximo de tortugas:** Para el modo multitortuga (sección 14.1). Por defecto 16.

- **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.
- **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, recuerda instalar las librerías `soundbank.gm` como se explicó en la sección 1.2.1.

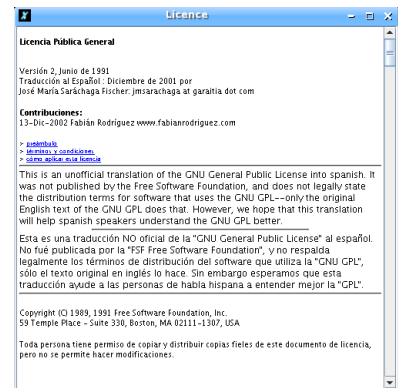
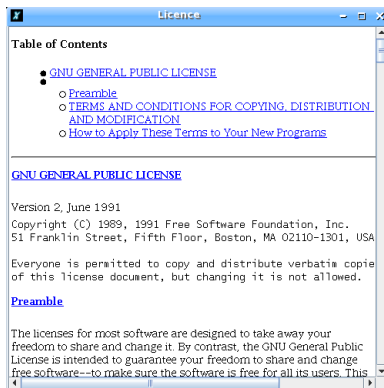
- **Fuente:** Elige el tipo de letra y su tamaño



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

2.4. Menú "Ayuda"

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



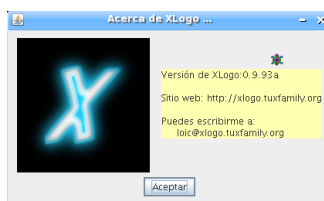
- **Ayuda → Traducción de la Licencia:** Refiere a una traducción al español de la licencia GPL, sin efectos legales, sólo como referencia para entender la versión en Inglés.
- **Ayuda → Traducir XLogo:** Abre una ventana para añadir y/o corregir traducciones.



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 3

Presentación de la tortuga

¿Cómo presentamos a nuestra tortuga?

3.1. Un programa de ejemplo

Empecemos por el siguiente procedimiento:

<http://xlogo.tuxfamily.org/sp/curso/presentacion.lgo>

Descarguemos el fichero a nuestro disco duro, abrámoslo (**Archivo** → **Abrir** → Elegir la ubicación donde lo hayamos descargado → *Click* en **Abrir**) y en la ventana que nos aparece (la del Editor de Procedimientos):

```
para presentacion
borrapantalla
si no miembro? "Angulo listavars
 [ haz "Angulo 90 ]
si no miembro? "Avance listavars
 [ haz "Avance 10 ]
#
botonigu "Av "Avanza
botonigu "Re "Retrocede
botonigu "Gd "Gira\ derecha
botonigu "Gi "Gira\ izquierda
botonigu "Sl "Subir/Bajar\ lápiz
botonigu "Ej "Ejecutar
botonigu "Ca "Cambiar\ ángulo
botonigu "Cd "Cambiar\ avance
menuigu "Cl [ Color\ Lápiz Azul Rojo Amarillo Verde Negro Blanco ]
menuigu "Gr [ Grosor\ Lápiz 1 2 3 4 5 ]
#
posicionigu "Av [-350 25]
```

Comando de Inicio: presentacion

escribamos en la Línea de Comando de inicio la palabra:

presentacion

(no te preocupes de momento por todo lo que hay escrito, en breve serás capaz de diseñarlo tú mismo/a). Finalmente, hacemos *click* en el pingüino y ... ¿no pasa nada?

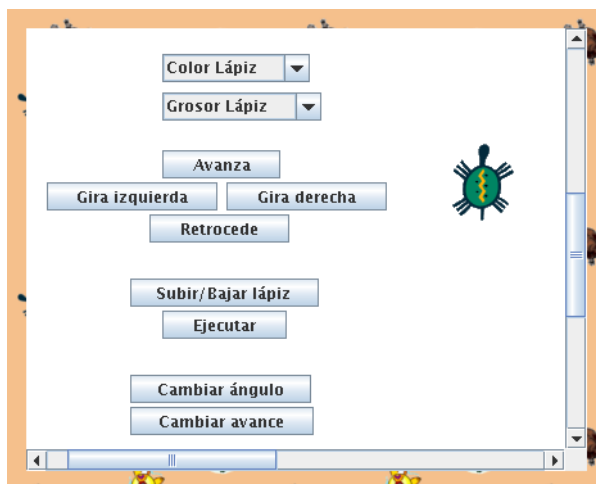
Observa el Histórico de Comandos. XLOGO te informa de que:

Acaba de definir presentacion.

pero además, has definido la orden que se ejecutará al pulsar el botón de Comando de inicio



No esperes más. Pulsa el botón y ... deberías obtener una pantalla como esta:



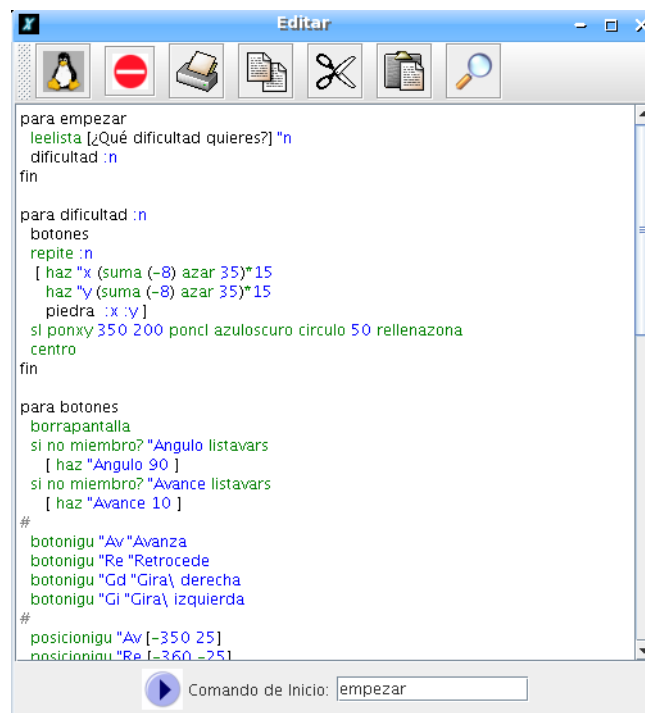
Haz todas las pruebas que necesites para “entender” cómo se mueve la tortuga, cómo mide los ángulos, las distancias, ... Fíjate, especialmente, en cómo realiza los giros y procura entender el punto de vista de la tortuga.

¿Qué podemos hacer con una pantalla como la de arriba? Podemos, por ejemplo, enseñar la lateralidad en los últimos cursos de Infantil y reforzarla en los primeros cursos de Primaria:

Descarguemos el fichero

<http://xlogo.tuxfamily.org/sp/curso/juego.lgo>

abrámoslo y cambiemos el Comando de Inicio por empezar:



```

para empezar
  lealista [¿Qué dificultad quieres?] "n
  dificultad :n
fin

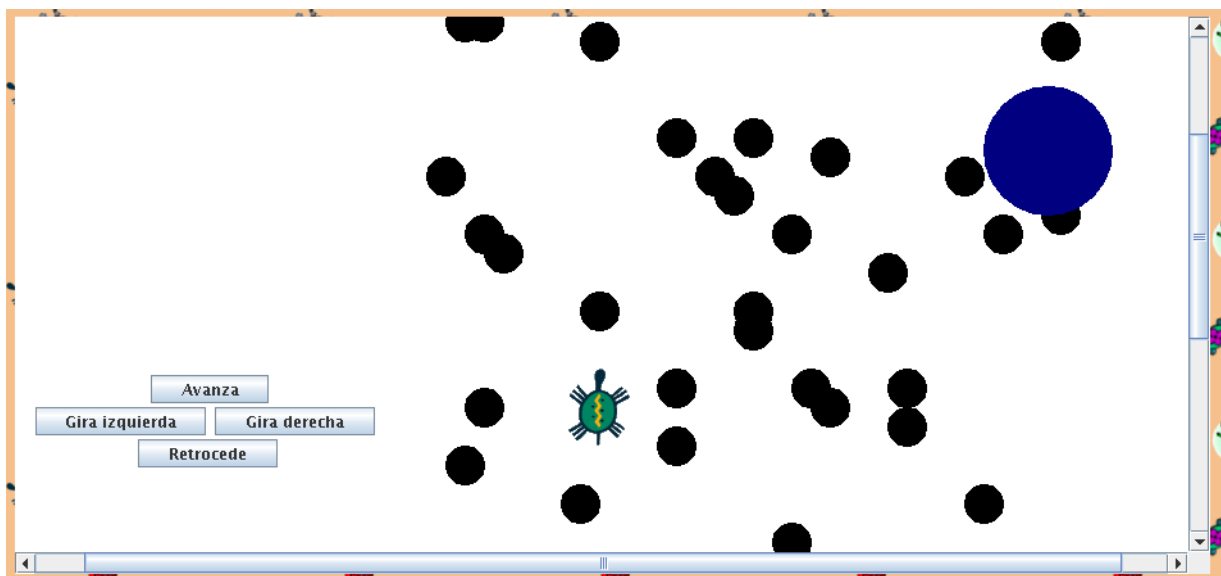
para dificultad :n
  botones
  repite :n
  [ haz "x (suma (-8) azar 35)" 15
    haz "y (suma (-8) azar 35)" 15
    piedra :x :y ]
  si ponxy 350 200 ponci azuloscuro circulo 50 rellenazona
  centro
fin

para botones
  borrar pantalla
  si no miembro? "Angulo listavars
  [ haz "Angulo 90 ]
  si no miembro? "Avance listavars
  [ haz "Avance 10 ]
#
  botonigu "Av "Avanza
  botonigu "Re "Retrocede
  botonigu "Gd "Gira\ derecha
  botonigu "Gi "Gira\ izquierda
#
  posicionigu "Av [-350 25]
  posicionigu "Re [-360 -25]

```

Comando de Inicio:

Guardamos (hacemos *click* en el pingüino) y después en el de Comando de inicio. Después de contestar a la pregunta (un valor de 20 ó 25 está bien) el resultado debe ser una pantalla similar a esta:

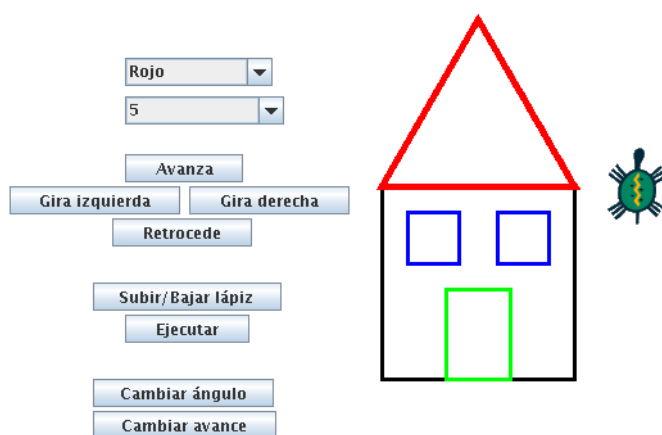


(Usa las barras de desplazamiento si no ves alguna de las figuras de esta captura).

Acabas de cargar un sencillo juego que consiste en llevar a la tortuga hasta el “lago” situado en la parte superior derecha usando solamente los botones de desplazamiento. Los círculos negros representan piedras y están colocadas aleatoriamente:

- Si “choca” con una piedra, nos aparece un aviso y vuelve al punto de partida
- Cuando llegue al lago, nos felicitará y sonará una cancioncilla (concretamente, *super-califragilisticexpialidoso* de la película **Mary Poppins**).

Sin embargo, la creación de este tipo de juegos no es la principal característica que XLOGO aporta al aula. El verdadero método de trabajo con LOGO se basa en proponer actividades (especialmente gráficas) para que el alumno se comporte como el **maestro de la tortuga** y le indique, razonadamente, qué pasos debe seguir para conseguir las. Por ejemplo, para dibujar esta casa:



El alumno debe:

1. Entender él mismo aquello que va a enseñar.
2. Planear una forma de impartirlo.
3. “Trocear” el problema en *mini*problemas más fácilmente abordables.
4. Saber cómo comunicarlo claramente.
5. Establecer este nuevo *conocimiento* como las bases de uno futuro.
6. Ser consciente del conocimiento que *su* “alumno” (la tortuga) ya tiene, y construir basándose en él.
7. Ser receptivo para explorar nuevas ideas según aparezcan.
8. Responder a los malentendidos de *su* alumno.

Además, lo más importante en el proceso de aprendizaje con LOGO **no** es el resultado final, sino **cómo haces lo que haces**; es decir, ver cómo se crea el diseño es más interesante y

educativo que el diseño en sí.

LOGO no está limitado a ningún área o tema en particular (a lo largo de este curso se irán planteando ejercicios de Lengua, Física, ...). Sin embargo es muy útil para matemáticas, ya que las gráficas generadas por la tortuga, medir sus movimientos en distancias y grados, ... permiten estudiar geometría mediante la construcción de polígonos y figuras.

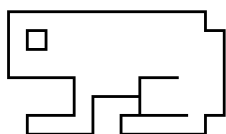


Echa un vistazo a las órdenes contenidas en los ficheros con los que hemos trabajado, e intenta “adivinar” qué hacen, qué parámetros les acompañan, por qué algunas están coloreadas y otras no, ...

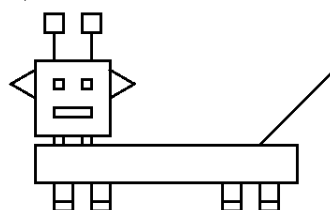
3.2. Ejercicios

Con la pantalla del programa *presentacion*, intenta dibujar:

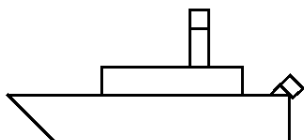
a) Una rana:



b) Un robot:



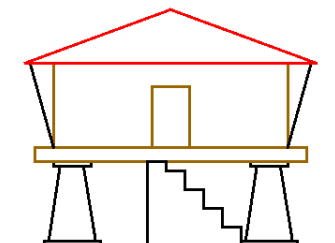
c) Un barco:



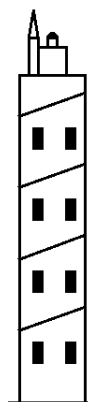
d) Tres pirámides:



e) Un hórreo:



f) Este faro:

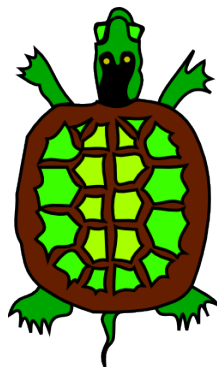
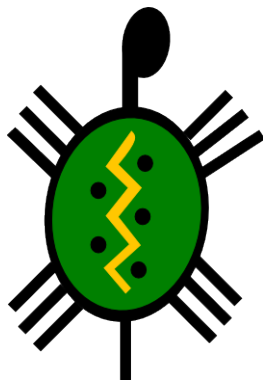
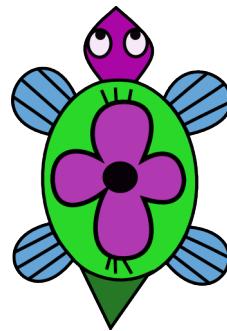


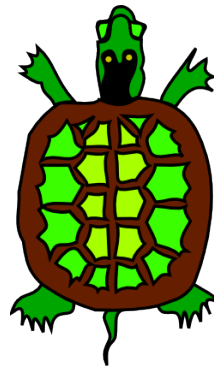
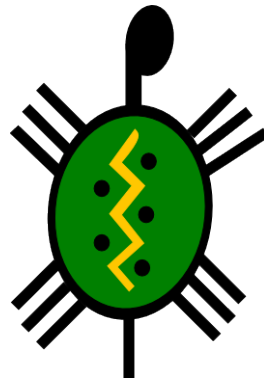
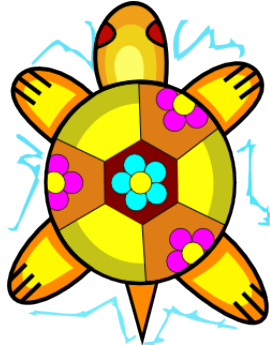
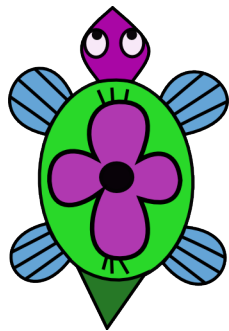
3.3. Una ayuda al dibujo

Antes de empezar a dar órdenes a nuestra tortuga, deberíamos acostumbrarnos a dibujar nosotros mismos las figuras en un papel (lo mismo vale cuando los procedimientos no dibujen, sino que hagan operaciones numéricas o con listas). Ver los dibujos en pantalla no es fácil y, en muchos casos, no obtendremos buenos resultados sin haber dibujado antes sobre el papel.

Una pequeña ayuda: en la página siguiente, recorta la imagen de la tortuga que prefieras, o dibuja tú otra que te guste más, y pónsela al lápiz. Así harás dibujar a la tortuga tanto en tu papel como en el ordenador.

Te aconsejamos que uses un lápiz con una goma al otro lado y que tengas cerca lápices de colores, ya que encontraremos órdenes que nos ayudarán a borrar lo que hicimos mal y ya has visto que podemos cambiar el color del lápiz.







Capítulo 4

Iniciación a la Geometría de la tortuga

Comenzamos la descripción de las *primitivas* de XLOGO.

4.1. Descripción de las primitivas o comandos

Las primitivas son órdenes básicas que el programa ya tiene incorporadas; al escribirlas en la *línea de comandos* ordenan a la tortuga realizar una acción determinada. La ventaja de LOGO frente a otros lenguajes radica en que las órdenes pueden escribirse en el idioma *natural* del usuario/programador, de modo que resultan muy fáciles de entender.

Cada primitiva puede tener un cierto número de parámetros que son llamados *argumentos*. Por ejemplo, la primitiva `pi`, que devuelve el valor del número π (3.141592653589793), no lleva argumentos: la primitiva `escribe` espera uno (`escribe` muestra en el Histórico de Comandos ese argumento), mientras que la primitiva `suma` (sección 7.1) tiene dos argumentos.

```
escribe suma 2 3 devuelve 5.
```

Los argumentos en LOGO son de tres tipos: **Número**, **Palabra** o **Lista**

- **Números:** Algunas primitivas esperan números como su argumento: `avanza 100` (sección 4.2) es un ejemplo.
- **Palabras:** Las palabras se escriben precedidas por ". Un ejemplo de una primitiva que admite una palabra como argumento es `escribe`.

```
escribe "hola devuelve hola
```

Nota que si olvidas las comillas ("), 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.

escribe [El gato es gris] devuelve El gato es gris

Los números son tratados a veces como un valor (por ej: `avanza 100`), o bien como una palabra (por ejemplo: `escribe vacio? 12` devuelve `falso` – sección 9.4).

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

- `escribe`
- `suma`, `producto` (sección 7.1)
- `o`, `y` (sección 9.2)
- `lista`, `frase`, `palabra` (sección 10.1)

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. Movimientos de la tortuga

Empecemos por las primitivas que controlan el movimiento de la tortuga:

Primitiva	Forma larga	Forma corta
AVanzar n pasos	avanza n	av n
REtroceder n pasos	retrocede n	re n
Gira Derecha n grados	giraderecha n	gd n
Gira Izquierda n grados	giraizquierda n	gi n
Llevar la tortuga al centro de la pantalla	centro	

Observa que no todas las primitivas son necesarias. Por ejemplo:

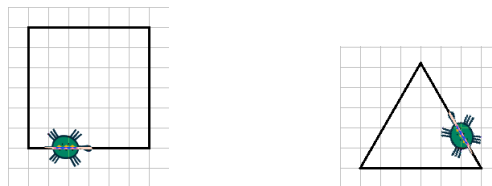
- giraizquierda 90 equivale a giraderecha -90
- retrocede 100 equivale a avanza -100

	<p>Haz todas las pruebas que necesites para entender perfectamente estas primitivas. Comprueba que la predicción del tema anterior de cómo realiza los giros es correcta, y si has entendido bien el punto de vista de la tortuga</p>
---	---

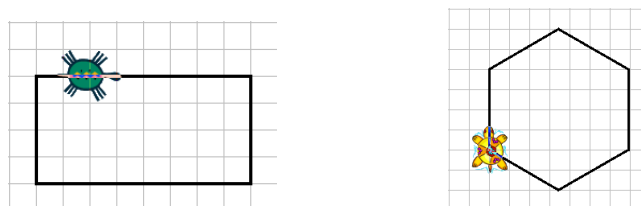
4.3. Ejercicios

En los dibujos, el lado de cada cuadrado de la cuadrícula mide 25 “pasos de tortuga”.

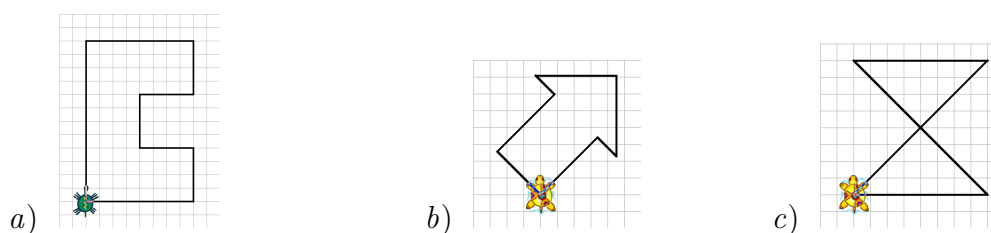
1. Dibuja el borde de un cuadrado en sentido *antihorario*
2. Dibuja el borde de un cuadrado en sentido *horario*

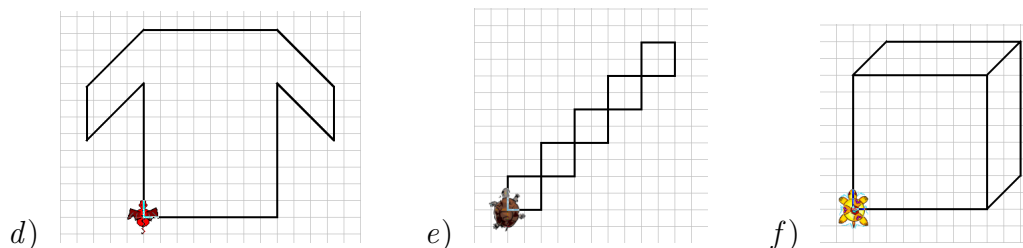



3. Dibuja el borde de un triángulo equilátero en sentido *antihorario*
4. Dibuja el borde de un rectángulo



5. Dibuja el borde de un hexágono regular
6. Dibuja:





	<p>La tortuga tiene muy “mala memoria”, así que todas las órdenes que has dado para dibujar los ejemplos se perderán. Puedes hacer dos cosas: copiarlas en un papel, o usar la Opción del Menú: Archivo → Zona de Texto → Guardar en formato RTF.</p>
---	--

4.4. Avanzando un poco

Continuemos con primitivas que controlan otros aspectos:

Primitiva	Forma larga	Forma corta
Borrar Pantalla y tortuga al centro	<code>borrapantalla</code>	<code>bp</code>
Subir Lápiz (no deja trazo al moverse)	<code>subelapiz</code>	<code>sl</code>
Bajar Lápiz (sí deja trazo al moverse)	<code>bajalapiz</code>	<code>bl</code>
Ocultar Tortuga	<code>ocultatortuga</code>	<code>ot</code>
Mostrar Tortuga	<code>muestratortuga</code>	<code>mt</code>
Cambiar el color del trazo con que dibuja	<code>poncolorlapiz n</code>	<code>poncl n</code>
Cambiar el grosor del trazo con que dibuja	<code>pongrosor n</code>	
Borrar por donde pasa, en vez de escribir	<code>goma</code>	<code>go</code>
Dejar de borrar y volver a escribir	<code>bajalapiz</code>	<code>bl</code>
Rellenar con el color activo una región cerrada	<code>rellena</code>	
Rellenar una región limitada por el color activo	<code>rellenazona</code>	
Limpiar la pantalla dejando la tortuga en el sitio	<code>limpia</code>	
Repetir <code>n</code> veces lo indicado entre corchetes	<code>repite n</code>	


La primitiva `poncolorlapiz` debe ir acompañada de un número (ver sección 13.2.2), y las opciones con las que trabajaremos de momento serán:

0: negro	1: rojo	2: verde	3: amarillo
4: azul	5 magenta	6: cyan	7 blanco

Respecto a `repite`, las órdenes a repetir deben ir entre corchetes, por ejemplo:

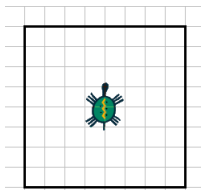
```
repite 4 [escribe "Hola ]
```

escribe 4 veces la palabra `Hola`

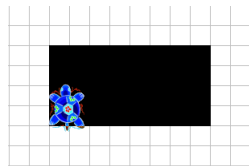
	<p>Analiza los programas con los que dibujaste antes los cuadrados, el triángulo y el hexágono. ¿Ves cómo se repiten varias veces las mismas órdenes? Piensa de qué modo puede ayudarte la primitiva <code>repite</code> a hacer más sencillo tu programa.</p>
---	---

4.5. Ejercicios

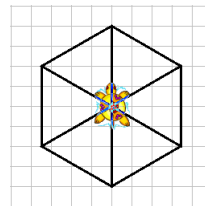
1. Dibuja el borde de un cuadrado, pero ahora usa la primitiva `repite`
2. Dibuja el borde de un triángulo equilátero usando la primitiva `repite`
3. Dibuja el borde de un hexágono regular usando la primitiva `repite`
4. Dibuja el borde de un cuadrado, cuyo centro esté en el centro de la pantalla
5. Dibuja un rectángulo, rellenando el interior



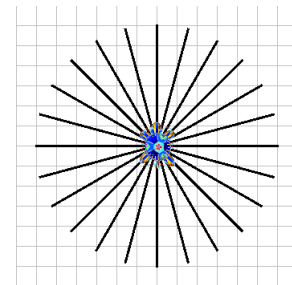
Problema 4



Problema 5




Problema 6



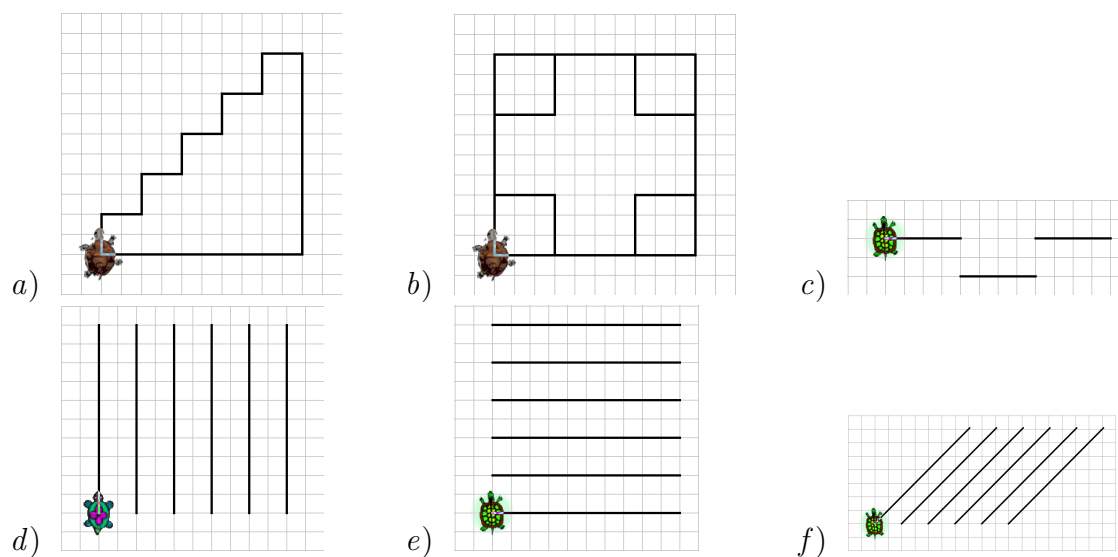
Problema 7

De nuevo, el lado de cada cuadrado de la cuadrícula mide 25 “pasos de tortuga”

6. Dibuja el borde de un hexágono regular y las diagonales cuyos extremos son dos vértices opuestos del mismo

	<p>Acabas de dibujar polígonos de 3, 4 y 6 lados. ¿Serías capaz de determinar una regla para hallar el ángulo de giro en cualquier polígono? Observa que la tortuga da una vuelta completa alrededor del polígono, que al final del proceso vuelve a estar mirando hacia arriba y fíjate cuántas veces tiene que girar.</p>
---	--

7. Dibuja los radios de una rueda. En total tienen que salirte 24
8. Dibuja:



Recuerda copiar las respuestas a los problemas en un papel, o guardarlas en el disco duro:
 Archivo → Zona de Texto → Guardar en formato RTF.



¿Qué te parece este método para dibujar? ¿Crees que puede mejorarse? ¿Cómo?
 ¿Qué pasa si quiero que los polígonos tengan lados más largos o más cortos? ¿Y si quiero cambiar el tamaño de las figuras que acabas de conseguir?



Capítulo 5

Procedimientos y subprocedimientos

5.1. Procedimientos

El capítulo anterior nos mostró varias cosas que deberían “preocuparnos”: En primer lugar, la tortuga “olvida” cómo dibujamos las figuras, es decir, las órdenes que le damos desaparecen de una vez para la siguiente.

En segundo, cada figura obligaba a teclear las órdenes cambiando las medidas una a una. Por ejemplo, para dibujar un cuadrado de lado 100, debíamos escribir:

```
repite 4 [ avanza 100 giraderecha 90 ]
```

cada vez que queríamos que apareciera en pantalla.

¿Y si es un rectángulo? La secuencia es más larga, y si quiero otro tamaño debería modificar **dos** medidas.



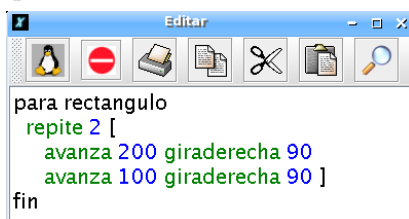
Podemos hacer que XLOGO “aprenda” nuevas primitivas, definiendo lo que se denomina procedimientos.

Haz *click* en el botón **Editar**, en la ventana emergente que acaba de aparecer escribe:

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

y haz *click* en el botón del pingüino/tortuga. Acabas de definir el procedimiento **cuadrado**, y la tortuga dibujará un cuadrado de lado 100 cada vez que le digas **cuadrado**.


Prueba ahora con el siguiente procedimiento:



```

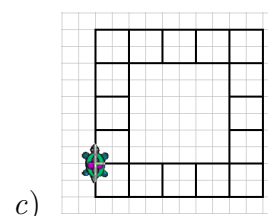
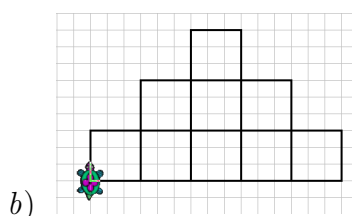
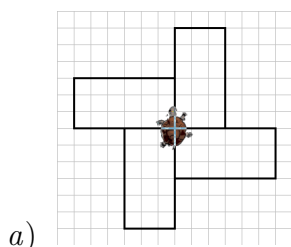
para rectangulo
  repite 2 [
    avanza 200 giraderecha 90
    avanza 100 giraderecha 90 ]
fin
  
```

¿Qué aparece al escribir `rectangulo` en la línea de comandos?

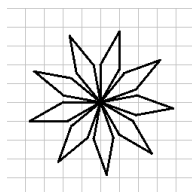
	<p>Intenta crear procedimientos llamados <code>avanza</code>, <code>cosa</code>, <code>cuadrícula</code> y <code>circulo</code>. ¿Qué observas al escribir el nombre? ¿Por qué crees que es? ¿Qué ocurre al guardar el procedimiento?</p>
---	---


5.2. Ejercicios

1. Plantea un procedimiento `triangulo`, que dibuje el borde de un triángulo equilátero
2. Plantea un procedimiento `hexagono`, que dibuje el borde de un hexágono regular
3. Plantea un procedimiento que dibuje un cuadrado en el centro de la pantalla
4. Plantea un procedimiento que dibuje los 24 radios de la rueda de una bicicleta
5. Usando los procedimientos `rectangulo` y `cuadrado` definidos antes, dibuja:



6. Plantea un procedimiento que dibuje un rombo, y con él dibuja la hélice:



	<p>Observa que los nombres de los procedimientos están relacionados con el dibujo (u objetivo) que deseamos. Esta es una norma de <i>buena educación</i> a la hora de programar, y hace más inteligibles los programas.</p>
---	---

5.3. Sub-procedimientos

Podemos conseguir efectos interesantes combinando procedimientos, es decir, haciendo que un procedimiento llame a otro.

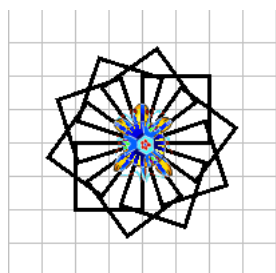
Los programas siguientes dibujan una colección de cuadrados con un vértice o un lado común, aprovechando el procedimiento `cuadrado` definido antes:

```

para colecuad
  repite 10
  [ cuadrado giraizquierda 36 ]
fin

para cuadrado
  repite 4 [
    avanza 50 giraderecha 90 ]
fin

```

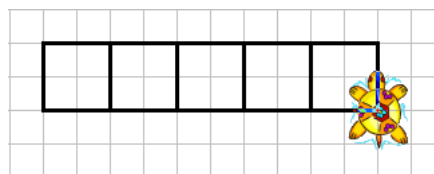


```

para filacuad
  repite 5 [
    cuadrado subelapiz
    giraderecha 90 avanza 50
    giraizquierda 90
    bajalapiz ]
fin

para cuadrado
  repite 4 [
    avanza 50 giraderecha 90 ]
fin

```



Antes citábamos una norma de *buen educación*, a partir de ahora la llamaremos *estilo*. También se aconseja *indentar* las líneas para reconocer más fácilmente dónde empiezan y dónde acaban determinadas secuencias de órdenes.

Finalmente, hablaremos de los *comentarios*. En XLOGO pueden añadirse líneas que NO serán interpretadas por la tortuga. Estas líneas se llaman **comentarios**; sirven para explicar qué hace un programa. Tecléa el siguiente procedimiento:

```

para rectangulo
  # Este procedimiento dibuja un rectángulo de base 100 y altura 200
  repite 2 [
    avanza 200 giraderecha 90
    avanza 100 giraderecha 90 ]
fin

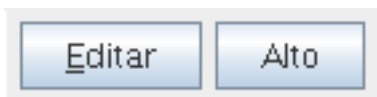
```

Puedes ver que la segunda línea empieza por #, y aparece en color gris. La “*almohadilla*” indica a la tortuga que es un comentario, así que ignora la línea y sigue leyendo las siguientes.



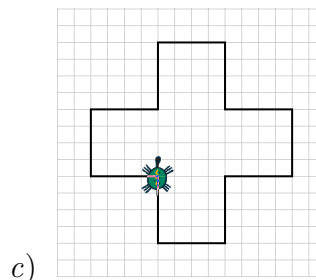
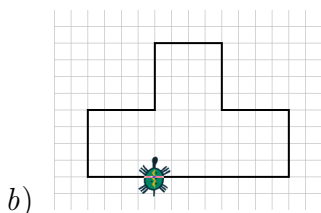
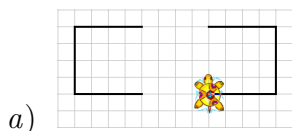
Usar la forma larga de las primitivas, buscar nombres para las variables relacionados con aquello con lo que estamos trabajando, indentar y comentar los programas ayuda a hacerlos más fácilmente legibles.

Para parar la ejecución de un procedimiento antes de llegar al final (es decir, hacerle saltar directamente hasta `fin`) se puede usar la primitiva `alto`, pero es mejor intentar no usarla. Igualmente, si vemos que un procedimiento no se termina por él mismo, podemos hacer *click* en el botón `Alto`, situado al lado del botón de `Editar`.

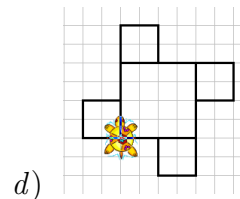
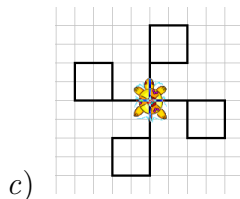
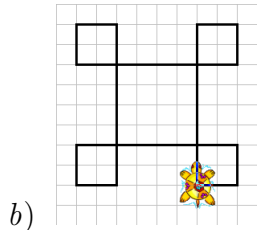
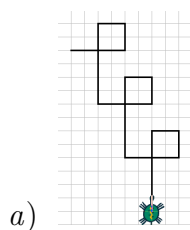


5.4. Ejercicios

1. Plantea un procedimiento que dibuje una colección de hexágonos regulares con un vértice en común, cada uno girado 60 grados respecto del anterior
2. Observa que en las siguientes figuras puedes ver un cuadrado al que le falta un lado. Modifica el procedimiento `cuadrado` para que sólo dibuje tres lados y dibuja:



3. Observa las siguientes figuras y busca un patrón que se repite. Úsalo para dibujar:



4. Plantea un procedimiento que dibuje una fila horizontal de cinco triángulos equiláteros, cuyas bases estén contenidas en la misma recta



IMPORTANTE: Los procedimientos pueden guardarse en el disco duro de tu ordenador de forma que luego puedes recuperarlos sin re-escribirlos como se explicó en 2.1



Capítulo 6

Variables. Procedimientos con argumentos

6.1. Primitivas asociadas

Definimos ahora seis nuevas primitivas:

Descripción	Primitiva	Ejemplo
Guardar un valor en una variable	haz	haz "lado 115
Utilizar el valor de a	:	escribe :a
	cosa	escribe cosa "a
	objeto	escribe objeto "a
Enumerar todas las variables definidas.	listavars	listavars
Eliminar la variable var.	borravariabile, bov	borravariabile "lado

Por compatibilidad con otros intérpretes LOGO, se admite `imvars` (imprime todas las variables) con la misma función que `listavars`.

Fíjate en la diferencia:

- Para definir la variable, se antepone "
- Para leer la variable, se precede de :, la forma más cómoda de las tres posibles: `cosa "a`, `objeto "a` y `:a` son notaciones equivalentes.

Aunque lo detallaremos más adelante, debemos comentar que XLOGO trata de distinta forma los números, las palabras y las frases. Para distinguir cuándo una variable almacena un tipo distinto, debemos usar un *vocabulario* específico:

Número: Para guardar en la variable `lado` el valor 100:

```
haz "lado 100
```

Palabra: Para guardar en la variable `animal` la palabra GATO:

```
haz "animal "GATO
```

Frase: Para guardar en la variable `descripcion` la frase El gato es gris:

```
haz "descripcion [El gato es gris]
```

En XLOGO (y en otros lenguajes de programación) se utiliza el término *Lista* para referirse a aquellas variables que constan de varios elementos, por ejemplo:

```
haz "primitiva [ 5 9 23 26 45 48 ]
```

que contiene una posible combinación del sorteo de la *Lotería primitiva* NO es una frase, ya que no consta de palabras. Es una **Lista**.

Una lista puede constar de varias *sublistas*, por ejemplo:

```
haz "primitiva [ [5 9 23 26 45 48] [5 8 18 26 40 46] [20 24 28 31 36 39] ]
```

consta de tres sublistas, y se pueden combinar variables de cualquier tipo:

```
haz "listado [ [[Pepe Perez] 15 CuartoA] [[Lola Lopez] 16 CuartoB] ]
```

contiene dos sublistas, cada una con una lista (nombre), un número (edad) y una palabra (el grupo de clase)

Si el valor que guarda la variable es un número, puede operarse con ella igual que con un número:

```
haz "lado 100
avanza :lado
```

e incluso pueden usarse para definir otras:

```
haz "alto 100
haz "ancho 2*:alto
repite 2 [ avanza :alto giraderecha 90
           avanza :ancho giraderecha 90 ]
```

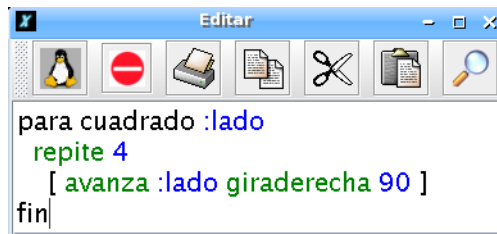
que dibuja un rectángulo de base doble que la altura.



¿Qué otras utilidades le ves al uso de variables? ¿Cómo las usarías para responder a las preguntas con las que cerrábamos el tema 4? ¿Puedes imaginar algún uso de las listas?

6.2. Procedimientos con variables

Resulta muy interesante la posibilidad de definir los procedimientos con *entradas*:



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

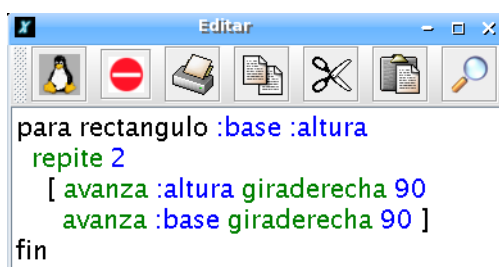
que dibuja, como ya habrás adivinado, un cuadrado. La diferencia está en que ahora el *lado* es desconocido, y debemos indicarle a la tortuga cuánto debe medir:

```
cuadrado 30
cuadrado 50
cuadrado 250
```

dibujarán cuadrados de lados 30, 50 y 250, respectivamente:



Podemos prever varios *argumentos*:



```

para rectangulo :base :altura
  repite 2
  [ avanza :altura giraderecha 90
    avanza :base giraderecha 90 ]
fin

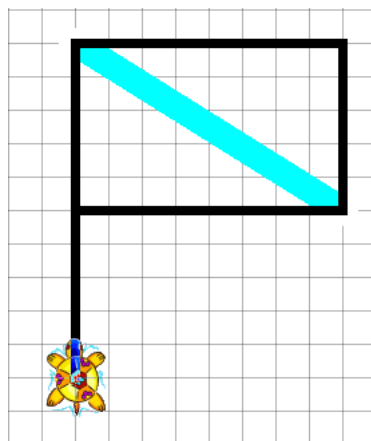
```

al que llamaremos escribiendo:

```
rectangulo 250 50
```

6.3. Ejercicios

1. Plantea un procedimiento **triangulo** que necesite una variable **lado** y que dibuje un triángulo equilátero cuyo lado sea ese valor
2. Plantea un procedimiento **rueda** que dibuje los 36 radios de longitud **largo** de una rueda
3. Plantea un procedimiento **bandera** que dibuje una bandera consistente en un mástil de longitud **mastil** y cuya tela sea un rectángulo de lados **ancho** y **alto**

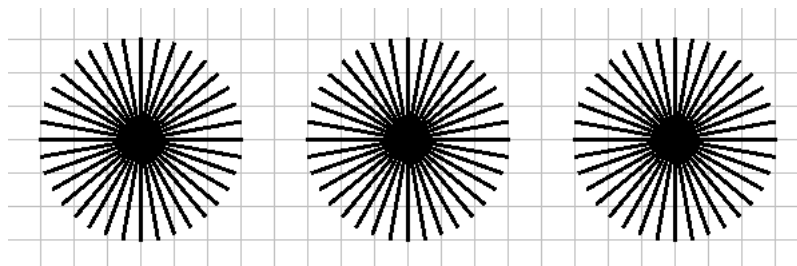


4. Plantea un procedimiento **poligono**, que reciba dos entradas: **n** y **largo**, y dibuje un polígono regular de **n** lados de longitud **largo**

Pista: Para hacer divisiones, XLOGO utiliza la primitiva `/`.

Por ejemplo: `escribe 256/5` devuelve `51.2`

5. Plantea un programa que dibuje una fila de **n** ruedas, cada una con 36 radios de longitud **largo**, de modo que la distancia entre los centros de dos ruedas contiguas sea **distancia**



Prueba con distintos valores de largo y distancia y observa qué ocurre. ¿Cuál es la relación entre **distancia** y **largo** cuando se superponen las ruedas? ¿Y cuando están separadas? ¿Qué podríamos hacer para que nunca se superpusieran?

6.4. 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 otro sub-procedimiento. Al finalizar el procedimiento, las variables locales son eliminadas.

Las primitivas asociadas son:

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 .

Supongamos que en el último ejercicio de la sección anterior quisiéramos controlar la separación entre ruedas para evitar que se superpongan unas con otras. Podríamos hacer que **distancia** fuera siempre algo más del doble que **largo**, para lo que planteamos dos *subprocedimientos* distintos:

```

para ruedas :n :largo
  repite :n
    [ rueda :largo
      separa :largo ]
fin

para rueda :radio
  repite 36
    [ avanza :radio retrocede :radio giraderecha 10 ]
fin

para separa :largo
  hazlocal "distancia 2.5 * :largo
  subelapiz
  giraizquierda 90 avanza :distancia giraderecha 90
  bajalapiz
fin

```

Observa que se usan tres variables relacionadas con la longitud: `largo`, `radio` y `distancia`. Al ejecutar el programa tecleando:

```
borrapantalla ruedas 3 100
```

la tortuga lee `largo`, y le asigna el valor 100. Sin embargo, `radio` sólo “existe” mientras se está ejecutando el procedimiento `rueda` y “desaparece” al finalizar este. Puedes comprobarlo modificando el procedimiento:

```

para ruedas :n :largo
  repite :n
    [ rueda :largo separa :largo ]
  escribe :largo
  escribe :radio
fin

```

que devolverá 100 (el valor de `largo`) y un mensaje de error:

```

En ruedas, línea 4:
radio no tiene valor.

```



Estudia qué ocurre con `distancia` en las dos definiciones que hemos hecho de `separa`. Cambia `escribe :radio` por `escribe :distancia` y observa qué responde XLOGO según hayas usado `haz` o `hazlocal`.

```

para ruedas :n :largo
  repite :n
    [ rueda :largo separa :largo ]
  escribe :largo
  escribe :distancia
fin

```



Las variables locales son muy útiles en programas largos, con varios procedimientos. Si cada uno usa sus propias variables, no es probable que haya errores debidos a que alguna de ellas sea modificada en el procedimiento equivocado.

6.5. Desde la Línea de Comandos

Los procedimientos pueden ser creados y borrados desde la Línea de Comandos. Igualmente, podemos determinar cuáles han sido ya definidos y cuáles no o ejecutar una serie de órdenes sin necesidad de crear un procedimiento asociado.

6.5.1. La primitiva define

La primitiva `define` crea un nuevo procedimiento sin usar el Editor. Para ello debemos proporcionar el nombre, las variables y las instrucciones a ejecutar:

```
define nombre [variables] [instrucciones]
```

Por ejemplo:

```
define "cuadrado [lado] [repite 4 [ avanza :lado giraderecha 90]]
```

crea el procedimiento `cuadrado` con el que ya hemos trabajado antes.

6.5.2. Las primitivas borra y borrar todo

La primitiva `borra` elimina el procedimiento indicado. La sintaxis es:

```
borra nombre
```

Por ejemplo:

```
borra "cuadrado
```

elimina el procedimiento `cuadrado` definido antes.

Por su parte, `borrar todo`, sin argumentos, elimina todas las variables y procedimientos actuales.



Mucho cuidado al usar esta primitiva. Teniendo en cuenta que es muy improbable que sea escrita por error, **NO** presenta ninguna ventana de confirmación, y realmente elimina **TODO** lo que hayas definido hasta ese momento.

6.5.3. La primitiva listaprocs

Esta primitiva no necesita argumentos, y enumera todos los procedimientos definidos hasta ese momento en el Histórico de Comandos. Por compatibilidad con otros intérpretes LOGO, se admite `imts` (imprime todos) con la misma función.

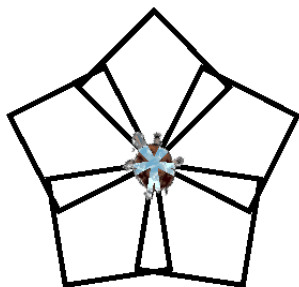
6.5.4. La primitiva ejecuta

Tecleando `ejecuta [lista]`, las órdenes contenidas en `lista` son ejecutadas consecutivamente.

Por ejemplo:

```
giraizquierda 27
ejecuta [ repite 5
          [ repite 4
            [ avanza 100 giraizquierda 90 ]
            giraderecha 72 ] ]
```

proporciona:

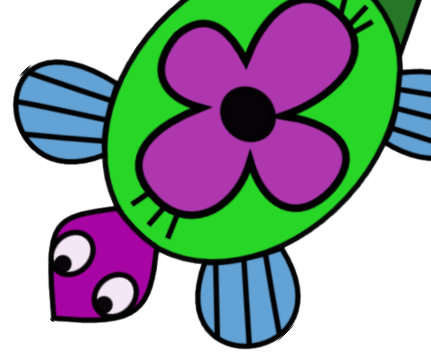


Un ejemplo más curioso de esta primitiva se muestra en la página de nuestro compañero Guy Walker:

<http://www.logoarts.ko.uk>

donde para dibujar un “arco iris” utiliza una lista que contiene las primitivas asociadas a seis colores (Sección 13.2.1) y con un bucle (Capítulo 11) cambia el color del lápiz (Sección 4.4) “ejecutando” su nombre:

```
...
haz "color [ rojo naranja amarillo verde azul violeta ]
repitepara [colores 1 6]
  [ poncolorlapiz ejecuta elemento :colores :color rellena
    subelapiz giraderecha 90 avanza 20 giraizquierda 90 bajalapiz ]
...
```



Capítulo 7

Operaciones

¿Qué ocurre si necesitamos realizar operaciones en XLOGO? Disponemos de las siguientes primitivas:

7.1. Operaciones binarias

Son aquellas que implican a dos elementos.

7.1.1. Con números

Descripción	Primitiva/s	Ejemplo
Sumar dos números	suma ó +	suma 15 20 ó 15 + 20 devuelven 35
Multiplicar dos números	producto ó *	producto (-5) 6 ó (-5) * 6 devuelven -30
Restar dos números	diferencia ó -	diferencia 35 (-10) ó 35 - (-10) devuelven 45
Dividir dos números reales	division ó /	division 35 3 ó 35 / 3 devuelven 11.666666666666666
Cociente entero	cociente	cociente 35 3 devuelve 11
Resto de una división entera	resto	resto 35 3 devuelve 2
Calcular una potencia	potencia	potencia 3 1.5 devuelve 5.196152422706632

Si se trata de sumas o productos de varios números, podemos usar la forma general de suma y producto:

$$13 + 7 + 2.5$$

(suma 13 7 2.5)

$$13 * 7 * 2.5$$

(producto 13 7 2.5)

Fíjate en los paréntesis de la segunda forma; son obligatorios. El resultado puede ser simplemente mostrado por pantalla:

```
escribe 13 + 7 + 2.5      escribe (suma 13 7 2.5)
escribe 13 * 7 * 2.5      escribe (producto 13 7 2.5)
```

o puede ser usado para dibujar:

```
avanza 13 + 7 + 2.5      giraderecha (suma 13 7 2.5)
retrocede 13 * 7 * 2.5   giraizquierda (producto 13 7 2.5)
```

Si, por ejemplo, queremos calcular cuántas formas distintas tenemos de rellenar una quiniela, le pediremos a la *tortuga*:

```
escribe potencia 3 15
```

devuelve

```
1.4348907E7
```

($3^{15} = 14\ 348\ 907$), el resultado se muestra en notación científica.



Analiza el siguiente procedimiento. Recuerda el uso de la barra invertida (sección 1.9.1) para generar espacios y saltos de línea. Te presentamos además la primitiva `tipea`, que escribe en el Histórico de comandos pero, a diferencia de `escribe`, no produce un salto de línea

```
para tabla
  haz "contador1 1
  haz "espacio [ \ ]
  haz "saltolinea [ \n ]
  repite 9
    [ haz "contador2 1
      repite 9
        [ tipea (:contador1 * :contador2)
          tipea :espacio
          haz "contador2 :contador2 + 1 ]
        tipea :saltolinea
        haz "contador1 :contador1 + 1 ]
  fin
```



¿Qué crees que hacen las variables `contador1` y `contador2`? ¿Por qué crees que las hemos usado? ¿Se te ocurre una forma mejor de conseguir lo mismo? ¿Cómo mejorarías el aspecto con el que salen los resultados?

7.1.2. Con listas

No sólo pueden efectuarse operaciones con números. Las palabras y las frases (listas de palabras) pueden *concatenarse* (ponerse una a continuación de la otra). Disponemos de las primitivas *frase* y *lista* (haremos un estudio más profundo de las listas en el capítulo 10). Por ejemplo:

```
escribe frase [El gato es ] [gris]
```

muestra en pantalla:

```
El gato es gris
```

pero:

```
escribe lista [El gato es ] [gris]
```

muestra en pantalla:

```
[El gato es] [gris]
```

es decir, crea una lista cuyos elementos son los argumentos, lo que en este caso lleva a obtener una *lista de listas*.

También es posible concatenar listas con números o variables. Por ejemplo, si la variable *area* contiene el valor 250 podemos pedirle a XLOGO:

```
escribe frase [La superficie es ] :area
```

que proporciona:

```
La superficie es 250
```

ya que *frase* ha concatenado la *lista* *La superficie es* y el valor de *:area*



Concatenar listas con variables es una forma de que los mensajes de XLOGO a la hora de presentar los resultados sea entiendan mejor.

Ejemplo: Vamos a crear un procedimiento que calcule el área de un triángulo dándole la base y la altura

Recuerda que el área de un triángulo es: $A = b * h / 2$

```
para area_triangulo :base :altura
haz "area (:base * :altura) / 2
escribe :area
fin
```

o bien:

```
para area_triangulo :base :altura
haz "area (division (producto :base :altura) 2)
escribe :area
fin
```

Para ejecutarlo, escribimos:

```
area_triangulo 3 5
```

Es posible ser un poco más elegante. ¿Tú sabrías lo que hace este programa sólo viendo los resultados o, incluso, leyéndolo? Cambiemos la penúltima línea:

```
para area_triangulo :base :altura
  haz "area (division (producto :base :altura) 2)
  escribe frase [El área del triángulo es ] :area
fin
```

o bien:

```
para area_triangulo :base :altura
# Este procedimiento calcula el área de un triángulo
# pidiendo su base y altura
  haz "area (division (producto :base :altura) 2)
  haz "texto frase [El área del triángulo de base] :base
  haz "texto frase :texto [y altura]
  haz "texto frase :texto :altura
  haz "texto frase :texto [es]
  haz "texto frase :texto :area
  escribe :texto
fin
```

que al ejecutarlo:

```
area_triangulo 3 5
```

proporciona:

```
El area del triangulo de base 3 y altura 5 es 7.5
```

¿No se entiende mejor?

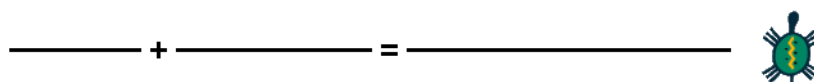
Observa otra capacidad del lenguaje XLOGO. Hemos *reutilizado* la variable `texto` varias veces, incluyendo en su definición **a ella misma**. Esto es útil cuando no quieres definir varias variables para un proceso que se refiere a una misma cosa (en este caso ir aumentando palabra a palabra el texto a mostrar en pantalla).

Podríamos haber aprovechado la *forma general* de la primitiva `frase`:

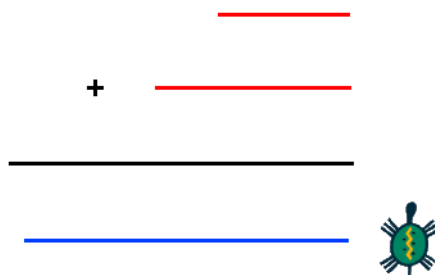
```
para area_triangulo :base :altura
# Este procedimiento calcula el área de un triángulo
# pidiendo su base y altura
  haz "area (division (producto :base :altura) 2)
  escribe (frase [El área del triángulo de base ] :base [y altura ] :altura [es ] :area)
fin
```


7.2. Ejercicios

1. Modifica el procedimiento `tabla` presentado como ejemplo para que muestre la *tabla de restar* y la *tabla de multiplicar*. ¿Qué observas?
2. Crea un procedimiento `acumula` que reciba dos números (`n1` y `n2`) como argumentos, dibuje dos segmentos cuyas longitudes sean precisamente `n1` y `n2` separados por un signo “más” y a continuación dibuje el segmento cuya longitud sea la suma de `n1` y `n2` precedido del signo “igual”.




3. Modifica el procedimiento anterior para que los segmentos aparezcan colocados como en las sumas tradicionales: los sumandos unos encima de otros, el signo “más” a la izquierda, una línea de “operación” y el resultado en la parte inferior. Usa colores para diferenciar los sumandos del signo y la línea y éstos del resultado



(Nota que en este caso los segmentos empiezan a dibujarse desde la derecha)

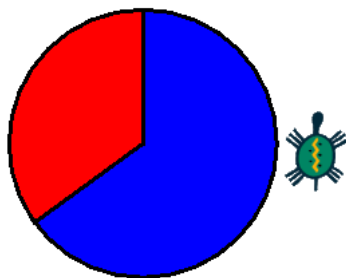
4. A partir de los procedimientos anteriores, construye los procedimientos `sustrae` que en lugar de sumas hagan “restas de segmentos”.

	Prueba distintas posibilidades, observando qué ocurre cuando <code>n1</code> es mayor que <code>n2</code> y viceversa.
---	--

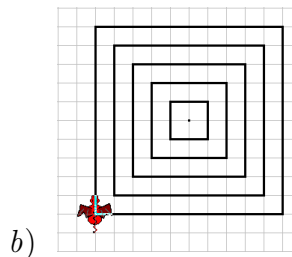
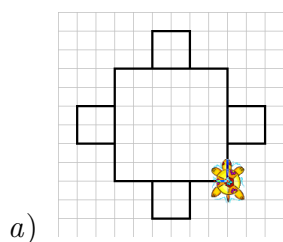
5. Crea un procedimiento `superficie` con dos argumentos, `n1` y `n2`, dibuje un rectángulo cuyos lados midan `n1` y `n2` y calcule su área.
6. Echa un vistazo a tu clase, y cuenta el número de chicos y de chicas que hay. Con esa información, crea el procedimiento `quesitos` que:
 - a) Calcule cuántos/as estudiantes hay en tu clase
 - b) Dibuje una circunferencia. Para ello usa la primitiva `circulo`, que necesita un argumento (`número`), y dibuja una circunferencia de radio `número` centrada en la posición actual de la tortuga

- c) Divida el círculo en dos partes proporcionales al número de chicas y chicos y los coloree con distintos colores

Por ejemplo, si en una clase hay un 40% de chicos y un 60% de chicas, el dibujo debería ser algo así:

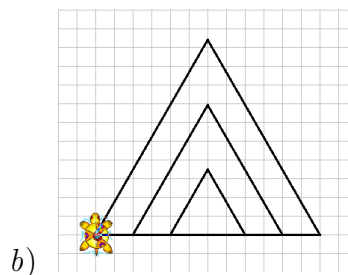
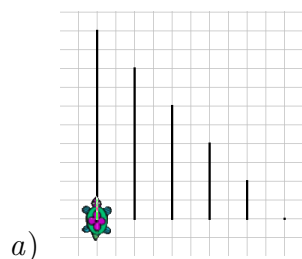


7. Con el procedimiento **cuadrado** definido en el capítulo anterior, dibuja:



8. Programa un procedimiento que dibuje una fila horizontal de **n** baldosas cuadradas cuyo lado sea la variable **lado** y que aparezcan centradas en pantalla

9. Dibuja:



10. Escribe un procedimiento **rayos**, que dibuje los **n** radios de longitud **largo** de una rueda (sólo los radios)

7.3. Operaciones unitarias

Son aquellas que sólo necesitan un elemento:

Descripción	Primitiva/s	Ejemplo
Hallar la raíz cuadrada de un número	raizcuadrada ó rc	raizcuadrada 64 ó rc 64 devuelven 8
Cambiar el signo	cambiasigno ó -	cs 5 devuelve -5
Valor absoluto de un número	absoluto ó abs	abs (-5) devuelve 5
Truncamiento de un número decimal	trunca	trunca 3.565 devuelve 3
Redondear un número al entero más cercano	redondea	redondea 3.565 devuelve 4
Generar un número aleatorio comprendido entre 0 y n-1	azar n	azar 6 devuelve un entero entre 0 y 5

La primitiva `truncar` puede usarse para conseguir un número entero cuando nos aparece en notación científica. En el apartado 7.1.1 calculábamos cuántas formas distintas tenemos de rellenar una quiniela, y escribimos:

```
escribe potencia 3 15
```

Si usamos

```
escribe truncar potencia 3 15
```

nos devuelve:

```
14348907
```

Ejemplo: Queremos un procedimiento que calcule el cociente y el resto de la división entera entre A y B, es decir, conseguir lo que proporcionan las primitivas `cociente` y `resto`, pero sin usarlas.

```
para division_entera :A :B
  haz "C truncar (:A / :B)
  escribe :C
  escribe :A - :B * :C
fin
```



La primitiva `azar` devuelve números aleatorios comprendidos entre 0 y n - 1. ¿Cómo harías para simular el lanzamiento de un dado? Lee con cuidado la sección 7.7 para evitar conclusiones erróneas.

7.4. Ejercicios

1. Modifica el procedimiento `division_entera` de modo que devuelva los resultados indicando con mensajes qué es cada número.
2. Plantea un procedimiento `Pitagoras` que refiriéndose a un triángulo rectángulo, acepte los valores de los catetos y calcule el valor de la hipotenusa.
3. Intenta escribir un procedimiento `suerte`, que genere un número al azar del conjunto $\{20, 25, 30, 35, 40, 45, 50\}$.¹
4. Plantea un procedimiento `dados`, que simule el lanzamiento de **dos** dados y cuya salida sea la suma de ambos
5. El procedimiento `juego` con el que presentamos a la tortuga en el capítulo 3 ubica las “piedras” de forma aleatoria. Piensa cómo harías para ubicar n “piedras” circulares y coloreadas en posiciones aleatorias
6. Plantea un procedimiento que simule un sorteo de Lotería Nacional, esto es, generar un número de 5 cifras por extracción consecutiva de 5 “bolas” con valores comprendidos entre 0 y 9 y colocarlas una a continuación de otras (recuerda la primitiva `tipea`)



En un sorteo de la “Lotería Primitiva” hay 49 bolas numeradas en un único bombo, y de él se extraen consecutivamente 6 bolas. ¿Podríamos usar la primitiva `azar` para simular un sorteo de la Primitiva?

7.5. Cálculo superior

XLOGO puede evaluar funciones trigonométricas y logarítmicas.

Descripción	Primitiva/s	Ejemplo
Calcular el seno de un ángulo	<code>seno</code> ó <code>sen</code>	<code>seno 45</code>
Calcular el coseno de un ángulo	<code>coseno</code> ó <code>cos</code>	<code>coseno 60</code>
Calcular la tangente de un ángulo	<code>tangente</code> ó <code>tan</code>	<code>tangente 135</code>
Obtener el ángulo cuyo seno es n	<code>arcoseno</code> ó <code>asen</code>	<code>asen 0.5</code>
Obtener el ángulo cuyo coseno es n	<code>arcoseno</code> ó <code>acos</code>	<code>acos 1</code>
Obtener el ángulo cuya tangente es n	<code>arcotangente</code> ó <code>atan</code>	<code>atan 1</code>
Evaluar el logaritmo decimal de un número	<code>log</code> ó <code>log10</code>	<code>log 100</code>
Evaluar el logaritmo neperiano de un número	<code>logneperiano</code> ó <code>ln</code>	<code>ln 100</code>
Exponencial de un número (e^x)	<code>exp</code>	<code>exp 2</code>

¹Recuerda que `azar 5 + 10` es lo mismo que `azar (5 + 10)`, es decir, `azar 15`. Igualmente `10 + azar 5` es equivalente a `(azar 5) + 10`

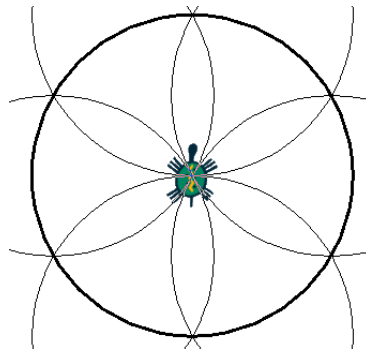
7.6. Ejercicios

Pista para los tres problemas siguientes: Puedes usar las primitivas `repite` y `contador`² y una variable en la que ir guardando los resultados parciales.

1. ¿Sabrías crear un procedimiento `poten :x :n` que calcule x^n (supuesto que n es un número natural) sin usar la primitiva `potencia`?
2. Crea un procedimiento `factorial :n`, que calcule el factorial del número n .
Recuerda: $n! = n * (n - 1) * \dots * 2 * 1$
3. Intenta ahora conseguir un procedimiento `suma_potencias :n` que calcule la suma $2 + 4 + 8 + \dots + 2^n$

Para los problemas siguientes necesitarás hacer algunos cálculos trigonométricos antes de ponerte a programar.

4. Plantea un procedimiento `poligono_regular`, que dibuje un polígono regular de n lados de longitud `lado` inscrito en una circunferencia
5. La flor *hexapétala* (conocida como flor de agua o flor galana en Asturias) es muy sencilla de dibujar únicamente con un compás:



Plantea un procedimiento `flor` que haga uso de la primitiva `arco` (`arco` necesita tres argumentos: el radio, el ángulo donde empieza y el ángulo donde termina, medido siempre en forma absoluta, es decir, 0° es **siempre** hacia arriba, independientemente de hacia dónde “mire” la tortuga) y que reciba un argumento `radio` para dibujar la flor galana.

6. Plantea un procedimiento con dos argumentos: `radio` y n , que dibuje un círculo con ese `radio`, y que inscriba en él un polígono de n lados.
7. (Este problema requiere importantes conocimientos de geometría y/o dibujo técnico) Plantea un procedimiento que dibuje una flor, pidiendo el `radio` y el número n de pétalos que tiene.

²Échale un vistazo a la sección 11.1.1 para entenderlas bien

7.7. Prioridad de las operaciones

Utilizando la forma corta, +, -, * y /, 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.

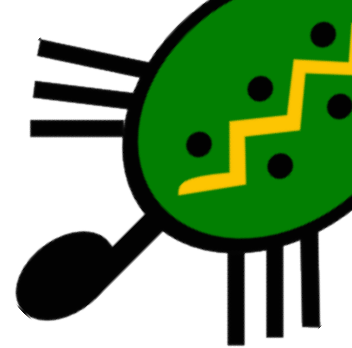
Como sabemos, la presencia de paréntesis modifica el orden en que se deben realizar las operaciones. Por ello, si escribimos:

```
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**, **division**, **potencia**, ..., ya que **internamente** tienen una prioridad inferior a las anteriores. Este es un comportamiento que estamos intentando solucionar, ya que incumple las reglas por todos conocidas. Hasta entonces, aconsejamos utilizar los paréntesis para evitar sorpresas desagradables:

```
escribe (seno 20) + 10      --> 10.3420201433256687
escribe (raizcuadrada 3) + 1 --> 2.7320508075688772
escribe (azar 6) + 1       --> 1, 2, 3, 4, 5 o 6
```



Capítulo 8

Coordenadas y Rumbo

¿Cómo podemos saber dónde se encuentra la tortuga? ¿Puedo enviarla a un punto concreto de la pantalla con una sola orden? ¿Hacia dónde está *mirando*? Estas preguntas se responden en este capítulo.

8.1. Cuadrícula y ejes

Comencemos por las primitivas que pueden ayudarnos enormemente en nuestro trabajo con las coordenadas:

Primitivas	Argumentos	Uso
cuadrícula	a b: números	Dibuja una cuadrícula en el Área de dibujo de dimensiones a x b y borra la pantalla
detienecuadrícula	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
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.
detieneejes	a: número	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



El sistema cartesiano con coordenadas ortogonales no es el único que existe. Investiga otros sistemas de representación e intenta crear procedimientos que los muestren en pantalla.

Nota: `detienecuadrícula` y `detieneejes` retiran la cuadrícula y los ejes de pantalla, y también la borra del mismo modo que `borrapantalla`

8.2. Coordenadas

Primitiva	Forma larga	Forma corta
Mostrar la posición (devuelve una lista)	<code>posicion</code>	<code>pos</code>
Mostrar sólo la abscisa (coordenada X)	<code>primero posicion</code>	<code>pr pos</code>
Mostrar sólo la ordenada (coordenada Y)	<code>ultimo posicion</code>	<code>ultimo pos</code>
Mover al punto [X , Y] (X,Y números)	<code>ponposicion [X Y]</code>	<code>ponpos [X Y]</code>
Pintar el punto de coordenadas [X , Y]	<code>punto [X Y]</code>	
Mover hacia [X , Y] (X,Y números o variables)		<code>ponxy :X :Y</code>
Llevar hacia el punto de abscisa X (número o variable)		<code>ponx :X</code>
Llevar hacia el punto de ordenada Y (número o variable)		<code>pony :Y</code>

De nuevo debemos tener cuidado con la prioridad de las primitivas. Si alguna coordenada es negativa, debemos usar paréntesis:

```
ponxy 100 (-60)
```

Para usar `ponposicion` con variables, debemos usar la primitiva frase:

```
ponposicion frase :abscisa :ordenada
```

8.3. Ejercicios

1. Dibuja un rectángulo usando solamente las primitivas `ponx` y `pony`
2. Dibuja un triángulo rectángulo isósceles usando únicamente `ponposicion`
3. Construye el procedimiento `segmento`, cuyas 4 entradas sean las coordenadas de dos puntos y que dibuje el segmento cuyos extremos son esos dos puntos.
4. Define el procedimiento `cuadrilatero`, cuyas OCHO entradas sean las coordenadas de cuatro puntos y que dibuje el cuadrilátero cuyos vértices son esos ocho puntos.

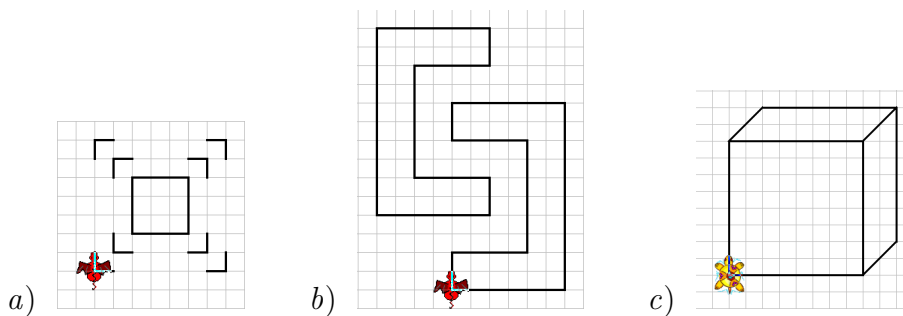
5. Plantea un procedimiento `dist_ptos`, que a partir del procedimiento `segmento` calcule la distancia entre los dos puntos (o lo que es lo mismo, la longitud del segmento)

Dato: La distancia entre dos puntos (x_0, y_0) y (x_1, y_1) se calcula mediante la fórmula:

$$d = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

6. Define el procedimiento `triangulo`, cuyas SEIS entradas sean los vértices de un triángulo, lo dibuje y lo rellene.

7. Dibuja:

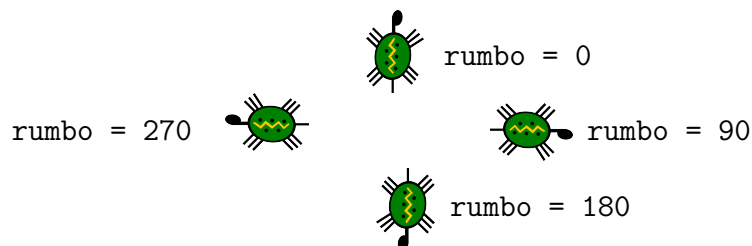


8.4. Rumbo

Las primitivas asociadas al rumbo son:

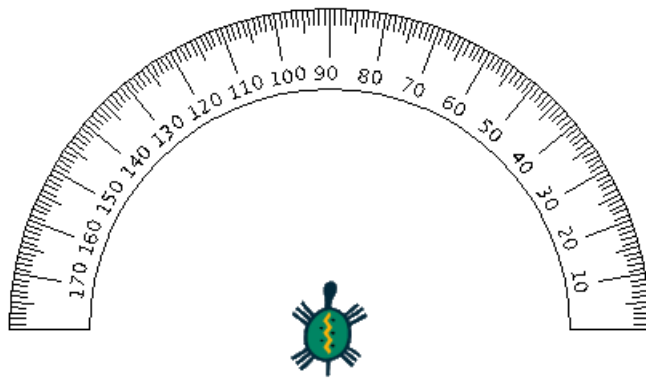
Primitiva	Forma larga	Forma corta
Orientar la tortuga hacia un rumbo <code>n</code>	<code>ponrumbo :n</code>	<code>ponr :n</code>
Pedir el rumbo (en grados respecto de la vertical y sentido horario)		<code>rumbo</code>
Para volver al origen con rumbo 0 (mirando hacia arriba)		<code>centro</code>
Pedir el rumbo que la tortuga debe seguir hacia el punto <code>[X Y]</code>		<code>hacia [X Y]</code>
Pedir la distancia (en pasos) hasta el punto <code>[X Y]</code>		<code>distancia [X Y]</code>

Una imagen vale más que mil palabras:



8.5. Ejercicios

1. Define procedimientos que pongan a la tortuga rumbo a los puntos cardinales: **norte**, **sur**, **este**, **oeste**, **noroeste**, **nordeste**, **sudeste** y **suroeste**.
2. En la sección anterior te pedíamos que buscaras otros sistemas de coordenadas. Te pedimos aquí que crees un procedimiento que represente en pantalla un sistema de coordenadas **polares**
3. Para manejar bien los rumbos es conveniente tener a la vista un transportador. Crea un procedimiento **transportador** que dibuje uno en pantalla:



Para indicar la escala, puedes usar la primitiva **rotula** (sección 12.1)



Capítulo 9

Condicionales y Operaciones lógicas

En ocasiones será necesario decidir qué acción realizar en función de una determinada condición, lo que en programación recibe el nombre de *flujo de control de un algoritmo o programa*.

Por ejemplo, si quiero calcular una raíz cuadrada, antes debo mirar si el número es positivo o no. Si es positivo, no tendré ningún problema, pero si es negativo XLOGO dará un mensaje de error, ya que la raíz cuadrada de un negativo no es un número real.



Las estructuras condicionales son uno de los pilares de la **Programación Estructurada**, es decir, una de las piezas clave en el desarrollo de programas complejos pero fáciles de leer.

9.1. Condicional

XLOGO define para ello la primitiva `si`, cuya sintaxis es:

```
si condicion
  [ Acciones a realizar si la condicion es cierta ]
```

Por ejemplo, el siguiente programa dice si un número es negativo:

```
para signo :numero
  si :numero < 0
    [ escribe [El numero es negativo] ]
fin
```

Ahora bien, no dice nada si el número es positivo o nulo. Disponemos de otra opción:

```
si condicion
  [ Acciones a realizar si la condicion es cierta ]
  [ Acciones a realizar si la condicion es falsa ]
```

de modo que podemos mejorar nuestro programa `signo`:

```
para signo :numero
  si :numero < 0
    [ escribe [El numero es negativo] ]
    [ escribe [El numero es positivo] ]
  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
```

9.2. Operaciones Lógicas

Con los condicionales es muy interesante conocer las *operaciones lógicas*:

Primitiva y Argumentos	Uso
<code>y condicion_1 condicion_2</code> ó <code>condicion_1 & condicion_2</code>	Devuelve cierto si ambas condiciones son ciertas. Si una (o las dos) son falsas, devuelve falso
<code>o condicion_1 condicion_2</code> ó <code>condicion_1 condicion_2</code>	Devuelve cierto si al menos una de las condiciones es cierta. Si las dos son falsas, devuelve falso
<code>no condicion</code>	Devuelve la negación de <code>condicion</code> , es decir, cierto si <code>condicion</code> es falsa y falso si <code>condicion</code> es cierta.

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

Operador “menor”	Operador “mayor”	Operador “menor o igual”	Operador “mayor o igual”
<	>	<=	>=

si bien es evidente que no serían estrictamente necesarios:

- $a \leq b$ es equivalente a $\text{no } (a > b)$
- $a \geq b$ puede sustituirse por $\text{no } (a < b)$

Por ejemplo (los paréntesis están para entender mejor el ejemplo):

```
para raiz_con_prueba :numero
  si o (:numero > 0) (:numero = 0)
    [ escribe raizcuadrada :numero ]
  fin
```

```
para raiz_con_prueba :numero
  si (:numero > 0) | (:numero = 0)
    [ escribe raizcuadrada :numero ]
  fin
```

que comprueba si el número es positivo **ó** cero antes de intentar calcular la raíz. Este procedimiento podría hacerse con **no**:

```
para raiz_con_prueba :numero
  si no (:numero < 0)
    [ escribe raizcuadrada :numero ]
  fin
```

y ahora comprueba que **no** sea negativo.

Imaginemos ahora un procedimiento que diga si la temperatura exterior es agradable o no:

```
para agradable :temperatura
  si y (:temperatura < 25) (:temperatura > 15)
    [ escribe [La temperatura es agradable] ]
  [ si no (:temperatura > 15)
    [ escribe [Hace frío] ]
    [ escribe [Hace demasiado calor] ] ]
  fin
```

```
para agradable :temperatura
  si (:temperatura < 25) & (:temperatura > 15)
    [ escribe [La temperatura es agradable] ]
  [ si no (:temperatura > 15)
    [ escribe [Hace frío] ]
    [ escribe [Hace demasiado calor] ] ]
  fin
```

que estudia primero si la temperatura está en el intervalo (15 , 25) – o sea $15 < T < 25$ – y si lo está dice *La temperatura es agradable*. Si no pertenece a ese intervalo, analiza si está por debajo de él (y dice *Hace frío*) o no (y devuelve *Hace demasiado calor*). Hemos *encadenado* condicionales.

Recuerda que tanto **y** como **o** admiten una forma general y, por tanto, pueden efectar más de una comparación:

```
si (y condicion1 condicion2 condicion3 ...) [ ... ]
si (o condicion1 condicion2 condicion3 ...) [ ... ]
```

son equivalentes a:

```
si (condicion1 & condicion2 & condicion3 & ...) [ ... ]
si (condicion1 | condicion2 | condicion3 | ...) [ ... ]
```

9.3. Ejercicios

1. Modifica el procedimiento `raiz_con_prueba` para **no** usar ninguna operación lógica
2. Plantea un procedimiento `no_menor` que decida si, dados dos números, el primero es mayor o igual que el segundo y responda **sí** en caso afirmativo
`no_menor 8 5` → **sí**
`no_menor 3 5` → (nada)
3. Plantea el procedimiento `edad_laboral`, que compruebe si la `edad` de una persona verifica la condición $17 < edad < 65$, respondiendo **Está en edad laboral** en caso afirmativo
4. Escribe el procedimiento `múltiplo?`, que verifique si un número `dato` es múltiplo de otro divisor, respondiendo **Es múltiplo** o **No es múltiplo**, en cada caso.

Pista: puedes usar `resto`, `cociente` y/o `division (/)`

En caso de que sea múltiplo, la tortuga debe dibujar un rectángulo cuya base sea el divisor y su área el múltiplo

`múltiplo? 18 5` → **No es múltiplo**

`múltiplo? 320 40` → **Es múltiplo**

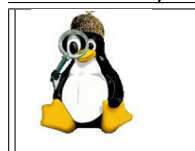
5. Diseña un procedimiento que diga si un año es bisiesto no. Recuerda que un año es bisiesto si es múltiplo de 4, pero no es múltiplo de 100 aunque sí de 400.
`bisiesto? 1941` → **1941 no es bisiesto**
`bisiesto? 1900` → **1900 no es bisiesto**
`bisiesto? 2000` → **2000 sí es bisiesto**
6. Diseña el procedimiento `calificaciones` que, dada una nota la califique de acuerdo con el baremo usual:

Nota	$n < 5$	$5 \leq n < 6$	$6 \leq n < 7$	$7 \leq n < 9$	$9 \leq n < 10$
Calificación	Suspenso	Aprobado	Bien	Notable	Sobresaliente

7. Diseña un programa que calcule la hipotenusa de un triángulo rectángulo dados sus catetos, pero que llame a un subprocedimiento que devuelva el cuadrado de un número dado, y que además dibuje el triángulo en pantalla

Pista: Coloca los catetos paralelos a los ejes cartesianos, y utiliza las primitivas asociadas a las coordenadas

8. Diseña un programa que dibuje un triángulo dados sus tres lados.



Cuidado: Dados tres lados, no siempre es posible construir un triángulo. Piensa qué condición debe cumplirse para que sea posible dibujarlo y después realiza los cálculos trigonométricos necesarios.

9. Plantea el procedimiento `mismo_signo`, que decida si dos números no nulos tienen el mismo signo.

Pista: Comprueba el signo de su producto

9.4. Booleanos

Una variable o primitiva es *booleana* si sus únicos valores posibles son **cierto** o **falso**. En XLOGO 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.
vacio?	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.
cuadricula?	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.
bajalapiz?, 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.


En la sección 5.1 te pedimos que hicieras pruebas con determinados nombres de procedimientos. Si queremos estar seguros de que un **nombre** está “libre” antes de usarlo, podemos preguntar:

```
variable? "nombre
```

y en caso que que la variable `nombre` no haya sido definida, asignarle un valor:

```
si no (variable? "nombre) & no (procedimiento? "nombre)
  [ haz "nombre pi ]
```

Las primitivas `cuadrícula?`, `ejes?`, ... permiten controlar aspectos que serán importantes, por ejemplo, al colorear regiones (sección 13.2.3).

	<p>Analiza ahora el siguiente procedimiento:</p> <pre style="border: 1px solid gray; padding: 5px;">para cuadrado_cortado haz "paso cambiasigno 50 repite 4 [repite 3 [si bajalapiz? [subelapiz] [bajalapiz] avanza 75] giraderecha 90] fin</pre> <p>¿Eres capaz de reproducir su resultado?</p>
---	--

9.5. Ejercicios

1. Utilizando las primitivas `listavars` y `listaprocs`, intenta crear un procedimiento que investigue si una determinada palabra es una variable o un procedimiento sin usar las primitivas `variable?` o `procedimiento?` (ni sus formas cortas)
2. Diseña un procedimiento que compruebe si la cuadrícula está activa o no.
 - Si está activa, que la borre y dibuje otra de dimensiones [100 * 100]
 - Si no lo está, que dibuje una de dimensiones [50 * 50]

A continuación, que sitúe a la tortuga en una posición generada por dos números aleatorios de la serie:

$$\{ -505 , -455 , -405 , \dots , 405 , 455 , 505 \}$$

y rellene o no el cuadrado en el que se encuentra siguiendo la secuencia de un tablero de ajedrez.

Finalmente, repetir la secuencia varias veces hasta ver si se consigue dibujar un tablero de ajedrez



Capítulo 10

Listas

Ya hablamos antes de listas. [53 gato [7 28] 4.9] es una lista en XLOGO; su primer elemento es 53, el segundo es gato, el tercero es [7 28] y el último es 4.9. Para almacenarlo en la variable ejemplo hacemos:

```
haz "ejemplo [ 53 gato [7 28] 4.9 ]
```

	Debemos entender que las listas permiten guardar la información ordenada, y que son un elemento muy importante para simplificar muchos programas.
--	---

10.1. Primitivas

Disponemos de las siguiente primitivas para trabajar con listas y con palabras:

Para manejarlas

Primitiva	Forma larga	Forma corta
Devolver el primer elemento	primero :ejemplo	pr :ejemplo
Devolver el último elemento	ultimo :ejemplo	
Devolver el elemento n-simo	elemento n :ejemplo	
Investigar algo en variable	miembro "algo "lista	
Contar el número de elementos	cuenta :ejemplo	
Devolver un elemento al azar	elige :lista	

Ejemplo con una lista:

```
haz "lista1 [Esto es una lista en xLogo]
#
  escribe primero :lista1          ---> Esto
```

```

escribe ultimo :lista1      ---> xLogo
escribe elemento 3 :lista1  ---> una
escribe miembro "es :lista1 ---> es una lista en xLogo
escribe cuenta :lista1     ---> 6
escribe elige :lista1      ---> en

```

Ejemplo con una palabra:

```

escribe primero "Calidociclos ---> C
escribe ultimo "Calidociclos  ---> s
escribe elemento 3 "Calidociclos ---> 1
escribe cuenta "Calidociclos  ---> 12
escribe miembro "es "Calidociclos ---> falso
escribe elige "Calidociclos   ---> i

```

obviamente, el resultado de `elige` variará de una ejecución a otra.

Para la primitiva `miembro`:

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

Para modificarlas

En este caso, sólo las dos primeras pueden usarse con palabras:

Primitiva	Forma larga	Forma corta
Quitar el primer elemento/letra	<code>menosprimero :ejemplo</code>	<code>mp</code>
Quitar el último elemento/letra	<code>menosultimo :ejemplo</code>	<code>mu</code>
Quitar el elemento <code>gato</code>	<code>quita "gato :ejemplo</code>	
Añadir algo el primero	<code>ponprimero "algo :ejemplo</code>	<code>pp</code>
Añadir algo el último	<code>ponultimo "algo :ejemplo</code>	<code>pu</code>
Intercalar algo en el lugar <code>n</code>	<code>agrega Lista n "algo</code>	
Reemplazar el elemento <code>n</code>	<code>reemplaza Lista n "algo</code>	
Invertir la lista	<code>invierte :lista</code>	

Ejemplo con una lista (`lista1` es la misma de antes):

```

escribe menosprimero :lista1      ---> es una lista en xLogo
escribe menosultimo :lista1      ---> Esto es una lista en
escribe quita "es :lista1        ---> Esto una lista en xLogo
escribe ponprimero "Super :lista1 ---> Super Esto es una lista en xLogo
escribe ponultimo "2007 :lista1  ---> Esto es una lista en xLogo 2007
escribe agrega :lista1 4 "super  ---> Esto es una super lista en xLogo
escribe reemplaza :lista1 3 "ye   ---> Esto ye una lista en xLogo
escribe invierte :lista1          ---> xLogo en lista una es Esto

```

Ejemplo con una palabra:

```

escribe menosprimero "Calidociclos ---> alidociclos
escribe menosultimo "Calidociclos  ---> Calidociclo

```

Para combinar

Primitiva	Forma larga	Forma corta
Combinar en una sola lista	<code>frase :ejemplo :algo</code>	<code>fr</code>
Concatenar en una sola palabra	<code>palabra :ejemplo :algo</code>	
Combinar en una lista de sublistas	<code>lista :ejemplo :algo</code>	

Ejemplo con una lista:

```

escribe frase :lista1 [y es genial] ---> Esto es una lista en xLogo y es genial
escribe lista :lista1 [y es genial] ---> [Esto es una lista en xLogo] [y es genial]

```

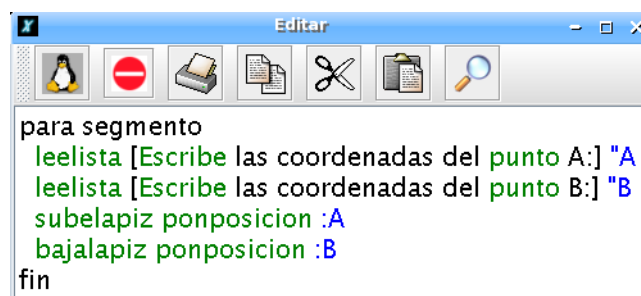
Ejemplo con una palabra:

```

escribe frase "Calidociclos "de\ Escher ---> Calidociclos de Escher
escribe lista "Calidociclos "de\ Escher ---> Calidociclos de Escher
escribe palabra "Super "Calidociclos    ---> SuperCalidociclos

```

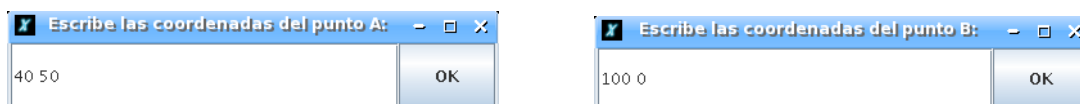
Existe una primitiva que permite que el *usuario* introduzca valores en XLOGO: `leelista`:



que se ejecuta tecleando:

```
segmento
```

XLOGO abrirá una ventana pidiendo las coordenadas de A. Contestamos, por ejemplo, 40 50 y pulsamos Intro; nos pide luego las coordenadas de B, por ejemplo 100 0 e Intro.



La tortuga dibujará el segmento cuyos extremos son los puntos cuyas coordenadas hemos introducido.

Observación: A y B son **listas**, no números. En ejemplo anterior A contiene [40 50], y podemos *convertir* sus elementos en números usando `primero`, `ultimo` ó `ultimo`, y usarlos con el resto de primitivas haciendo:

```
avanza primero :A
```

A la inversa, para convertir dos números en una lista tenemos la primitiva `lista`:

```
lista :lado :altura
```

Si se tratara de más de tres números disponemos de `frase` y `lista`:

```
haz "a 50 haz "b 60 haz "c 70
escribe lista :a frase :b :c
escribe lista :a lista :b :c
```

proporcionan: 50 [60 70] pero sus *formas generales*:

```
escribe (lista :a :b :c)           escribe (frase :a :b :c)
```

proporcionan: 50 60 70 esto es, una única frase. La diferencia con `palabra` es que esta concatena las variables. En el ejemplo anterior:

```
escribe palabra :a :b
escribe (palabra :a :b :c)
```

proporcionan:

```
5060
506070
```

esto es, una única palabra (en este caso, un número).

10.2. Ejemplo: Conjugación

Vamos a construir un programa que conjugue el futuro simple de un verbo (sólo para verbos regulares) de distintas formas, y veremos cómo el uso de listas simplifica el manejo de la información.

10.2.1. Primera versión

La primera posibilidad que se nos puede ocurrir es usar una línea para cada persona:

```
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
```

lo que consigue fácilmente nuestro objetivo:

Ejemplo: futuro "amar

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

10.2.2. Segunda versión

Podemos ser un poco más elegantes, con la primitiva `repite` o `repitepara` (sección 11.1.2) combinada con listas:

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

ya que, realmente, estamos repitiendo seis veces la misma estrategia: combinar el pronombre con el verbo y la terminación. El resultado, el mismo de antes.

10.2.3. Tercera versión

En esta versión usaremos la recurrencia, una característica muy útil en el uso de listas (ver sección 11.5)



Analiza qué hace este programa y cómo lo hace. Usa un papel para seguir la secuencia de pasos que da, y razona qué puede ser eso que llamamos *recurrencia* o *recursividad*

```
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]
  escribe frase primero :pronombres palabra :verbo primero :terminaciones
  conjugar :verbo menosprimero :pronombres menosprimero :terminaciones
fin
```

10.3. Ejercicios

1. Plantea un programa sobre conjugación de verbos que:
 - a) Pida un verbo con una ventana emergente
 - b) Determine a qué conjugación pertenece
 - c) Contenga dos listas:
 - "pronombres con los pronombres personales
 - "morfemas con los morfemas para conjugar el presente, siendo distinta en función de la conjugación del verbo
 - d) Combine la raíz del verbo con las terminaciones del presente en una única palabra
 - e) Combine en una frase los pronombres con la palabra generada, y las muestre en el Histórico de Comandos con `escribe`
2. ¿Cómo puede extraerse el 22 de la lista de listas `[[22 3] [4 5] [8 35]]`?
3. Plantea el procedimiento `prime`, con una entrada, `listado`, que devuelva su primer elemento, sin usar `primero`
4. Plantea el procedimiento `ulti`, con una entrada, `listado`, que devuelva su último elemento, sin usar `ultimo`

5. Diseña el procedimiento `triangulo`, que **pida** las coordenadas de los vértices de un triángulo uno a uno y lo dibuje
6. Escribe un procedimiento que pida la medida del lado de un cuadrado y devuelva la medida de su diagonal
7. Plantea el procedimiento `mengua` que pida escribir una serie de números y escriba las listas que se obtienen al ir eliminando un elemento de cada vez (por ejemplo el último) hasta quedar vacía.
8. Diseña un procedimiento `inversa` que reciba una lista y la devuelva con los elementos dispuestos en orden inverso al inicial
9. Plantea el procedimiento `maximo`, que pida una serie de números y devuelva el mayor de todos ellos
10. Diseña un procedimiento `suprime`, con dos entradas: `n` y `listado`, que devuelva la lista que se obtiene al suprimir el elemento `n`-simo, sin usar `quita`
11. Plantea el procedimiento `adjunta`, con tres entradas; `n`, `listado_1` y `listado_2`, que añada `listado_1` en la posición `n` de `listado_2` (sin usar `agrega`)
12. Escribe un procedimiento que determine si una palabra es un palíndromo, es decir, que se lee igual en la forma habitual que de derecha a izquierda
palíndromo? `reconocer` ---> cierto
palíndromo? `anilina` ---> cierto
palíndromo? `animal` ---> falso
13. ¿Cómo ampliarías el procedimiento anterior para que trabajar también con frases?
palíndromo? `Sé verlas al revés` ---> cierto
palíndromo? `Átale o me delata` ---> falso

Puedes encontrar palíndromos en:

<http://es.wikipedia.org/wiki/Palíndromo>

<http://es.wikiquote.org/wiki/Palíndromos>

<http://www.juegosdepalabras.com/palindromo.htm>

<http://www.carbajo.net/varios/pal.html>

10.4. Listas de Propiedades

Desde la versión 0.9.92, pueden definirse Listas de Propiedades con XLOGO. Cada lista tiene un nombre específico y contiene una pareja de “valores-clave”.

Para manejar estas listas, podemos utilizar las siguientes primitivas, para las que, por ejemplo, podemos considerar una lista llamada “coche”, que debe contener la clave “color” asociado al valor “rojo”, otra clave “tipo” con el valor “4x4” y una tercera clave “vendedor” asociada al valor “Citröen”,

Descripción	Primitiva	Ejemplo (con forma corta)
Añadir una propiedad a la lista	ponpropiedad	ponprop "Coche "Color "Rojo
Devolver el valor asociado a una clave de la lista	leepropiedad	leeprop "Coche "Color
Borrar el par clave-valor de una lista	borrpropiedad	boprop "Coche "Color
Mostrar todos los elementos de una lista	listapropiedades	listaprop "Coche
Mostrar todas las listas creadas	listaspropiedades	listasprop "Coche

Juguemos con los elementos de 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
```




Capítulo 11

Bucles y recursividad

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



Los bucles son otro de los pilares de la **Programación Estructurada**, es decir, una de las piezas clave en el desarrollo de programas complejos pero fáciles de leer.

11.1. Bucles

11.1.1. Bucle con repite

La sintaxis para `repite` es:

```
repite n [ lista_de_comandos ]
```

`n` es un número entero y `lista_de_comandos` 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.

Ya vimos varios ejemplos:

```
repite 4 [avanza 100 giraderecha 90]    # un cuadrado de lado 100
repite 6 [avanza 100 giraderecha 60]    # un hexagono de lado 100
repite 360 [avanza 2 giraderecha 1]     # abreviando, casi un circulo
```

aunque podemos añadir otro con un poco de humor:



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

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

proporciona

```
1
2
3
```

11.1.2. 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
```

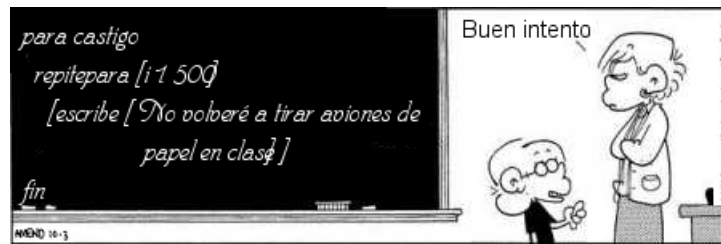
Ejemplo 2:

```
# 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]
  [escribe lista :i potencia :i 2]
```

proporciona

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

El mismo chiste de antes quedaría ahora:



11.1.3. 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 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 "abcdefghijklmnopqrstuvwxyz
mientras [no vacio? :lista1]
[es ultimo :lista1 haz "lista1 menosultimo :lista1]
```

Para usar `mientras`, ahora nuestro “alumno castigado” debe añadir una variable:



`mientras` es muy útil para trabajar con listas:

```
mientras no vacio? :nombre.lista
[ escribe primero :nombre.lista
  haz "nombre.lista menosprimero :nombre.lista]
```

irá mostrando todos los elementos de una lista, eliminando en cada iteración el primero de ellos.

11.1.4. Bucle con paracada

La sintaxis 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)

11.1.5. Bucle con repitesiempre

Aunque un bucle como este es muy peligroso en programación, es muy fácil crear un bucle infinito, por ejemplo con mientras:

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

La sintaxis de repitesiempre es:

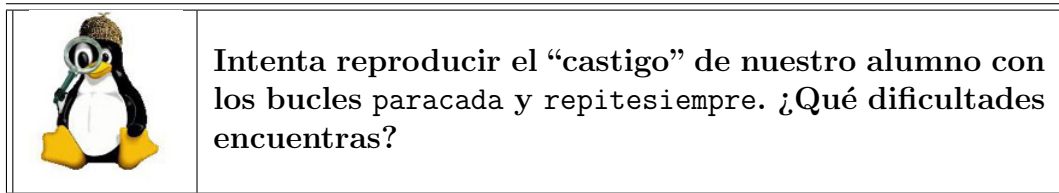
```
repitesiempre [ lista_de_comandos ]
```

El ejemplo anterior sería:

```
repitesiempre [giraderecha 1]
```

¿Cuándo se hace necesario un bucle infinito? De nuevo en la *web* de Guy Walker podemos encontrar respuestas: Simulaciones en Física, Química, Biología, ... como los movimientos planetario y browniano, la división celular, ... pueden hacer interesante que la animación (Sección 14.3) se mantenga activa durante el tiempo que dure una explicación.

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



11.2. Ejemplo

Observa el siguiente problema:

Halla el número n tal que, la suma de sus dígitos más él mismo, dé 2002:

$$n + S(n) = 2002$$

Podemos abordarlo de muchas maneras. Empecemos con una bastante intuitiva:

Es obvio que el número tiene que ser menor que 2002, ya que se suma a todos sus dígitos, así que:

para resuelve

```
repitepara [i 1 2002]
  [ haz "digitos cuenta :i # Contamos el numero de digitos
    haz "contador :i # Vamos a controlar la iteracion en curso
    haz "suma 0 # Guardaremos aqui la suma de los digitos
```

```

#                               # Debemos ponerla a cero en cada iteracion
  repite :digitos
    [ haz "suma :suma + primero :contador      # Voy sumando los digitos
      haz "contador menosprimero :contador ] # y quitando el primero
  si (:suma + :i) = 2002          # Comprobamos si cumple la condicion
    [ escribe :i ] ]            # En caso afirmativo, lo muestra
fin

```

Por supuesto, podríamos haber pensado un poco más el problema, y reducido el número de iteraciones, pero intentemos que sea general, y no dependa de que sea 2002 u otro número.

Hemos dicho varias veces que el problema se simplifica si lo dividimos en partes. Creemos un procedimiento para sumar los dígitos de un número dado:

```

para suma.digitos :numero
  hazlocal "suma 0                # Inicializo a cero la variable por si acaso
  mientras [numero? :numero]      # Cuando quite todos los digitos, no sera numero
    [ haz "suma :suma + primero :numero # Voy sumando los digitos y quitando
      haz "numero menosprimero :numero ] # siempre el primero. La variable es
  devuelve :suma                  # local, y no cambia el valor de "numero
fin                                # en el procedimiento principal

```

de modo que el procedimiento principal quedaría:

```

para resuelve
  repite 2002
    [ si 2002 = contador + suma.digitos contador
      [ escribe contador ] ]
fin

```

En este caso, no hemos necesitado usar variables auxiliares, ya que en ningún momento hemos modificado el valor de `contador` en el bucle, y la suma se hacía “fuera” del procedimiento principal.

11.3. Comandos de ruptura de secuencia

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

- `alto` puede tener dos resultados:
 - Si está incluido en un bucle `repite` o `mientras`, el programa sale del bucle inmediatamente.
 - Si está en un procedimiento, este es terminado.

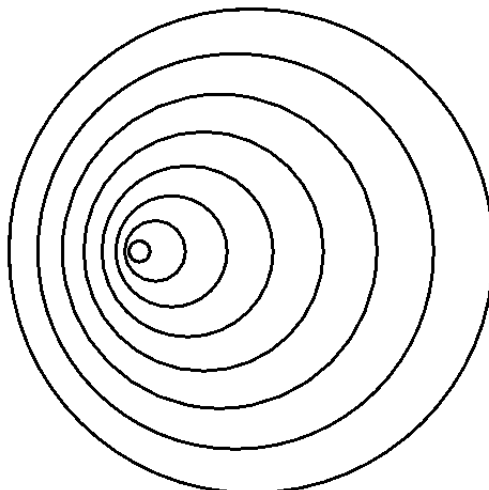
- `detienetodo` interrumpe total y definitivamente todos los procedimientos en ejecución
- `devuelve` (`dev`) permite salir de un procedimiento “llevándose” un resultado. (ver sección 11.8.1)

11.4. Ejercicios



Intenta hacer los ejercicios con los tres tipos de bucle. Si no puedes con alguno de ellos no desesperes, en programación se trata de encontrar la forma más fácil de programar, no de obtener la más rara.

1. ¿Cómo dibujarías esta serie de círculos NO concéntricos?



Puedes incluso dejar el número de círculos como variable

2. Escribe procedimientos que muestren las siguientes salidas en el Histórico de Comandos:

a)	1	2	3	4	5	6	7	8	9	10
b)	2	4	6	8	10	12	14	16	18	20
c)	20	22	24	26	28	30	32	34	36	38
d)	10	14	18	22	26	30	34	38	42	46
e)	45	40	35	30	25	20	15	10	5	0
f)	1	4	9	16	25	36	49	64	81	100
g)	2	5	10	17	26	37	50	65	82	101
h)	8	27	64	125	216	343	512	729	1000	1331
i)	1.0	0.5	0.33..	0.25	0.2	0.166..	0.1428	0.125		
j)	2	6	12	20	30	42	56	72	90	110

k) 1 10 100 1000 10000 100000 1000000
 l) 1.0 0.1 0.01 0.001 0.0001 0.00001 0.000001
 m) 1 -1 1 -1 1 -1 1 -1 1 -1

Los espacios son para ver mejor la serie para obtener. Tú usa `escribe`

3. Escribe un procedimiento que admita dos números y escriba la suma de enteros desde el primer número hasta el segundo.

```
suma.entre 30 32
La suma desde 30 hasta 32 es: 93
```

porque $30 + 31 + 32 = 93$

4. Escribe un procedimiento que pida un número y calcule su factorial:

$$n! = n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1$$

```
factorial 5
El factorial de 5 es 120
```

porque $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

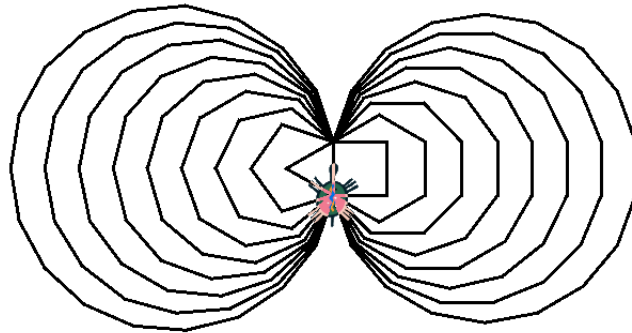
5. Escribe un procedimiento con un número como argumento y escriba sus divisores.

```
divisores 200
Los divisores de 200 son 1 2 4 5 8 10 20 25 40 50 100 200
```

6. Escribe un procedimiento con un número como argumento y determine si es primo o no.

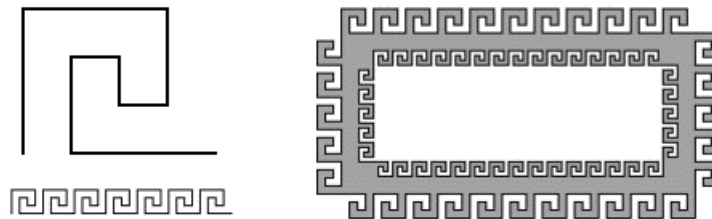
```
primo? 123
no es primo
primo? 127
127 es primo
```

7. Dibuja esta serie de polígonos, en la que los que tienen un número impar de vértices están hacia la izquierda, pero los que tienen un número par de vértices están hacia la derecha:



Puedes, como en el primer ejercicio, dejar el número de polígonos como variable

8. Intenta dibujar la figura de la derecha, basándote en el motivo mostrado a la izquierda:



9. Escribe un programa para jugar a adivinar un número. El procedimiento:

- admite dos argumentos, los valores entre los que está el número a adivinar
- se “inventa” un número al azar entre esos dos
- el usuario va probando valores y el programa va diciendo si son demasiado grandes o pequeños.

```
juego 0 100
```

```
-> A ver si adivinas un numero entero entre 0 y 100.
```

```
-> Escribe un numero: 20
```

```
-> Es mas grande: Intentalo de nuevo: 30
```

```
-> Es demasiado grande: Intentalo de nuevo: 30
```

```
-> Es demasiado grande: Intentalo de nuevo: 27
```

```
-> Acertaste. Te ha costado 3 intentos
```

10. Escribe un programa que permita crear una lista de palabras. Para ello, el procedimiento:

- Tiene un número como argumento
- Solicita ese número de palabras con un mensaje del tipo:

Dame el elemento numero ...

- c) Según se van introduciendo, se va aumentando la lista
- d) Por último, el programa tiene que escribir la lista.

```
crea.lista 3
-> Dame el elemento numero 1: Alberto
-> Dame el elemento numero 2: Benito
-> Dame el elemento numero 3: Carmen
La lista creada es: [Alberto Benito Carmen]
```

Modifica el procedimiento para que los elementos se vayan agregando a la lista en orden inverso al que se introducen

11. Escribe un procedimiento cuyo argumento sea una lista y que la ordene alfabéticamente (considera sólo la primera letra).

```
orden.alf [ Carmen Alberto Daniel Benito]
La lista ordenada es: [Alberto Benito Carmen Daniel]
```

11.5. Recursividad

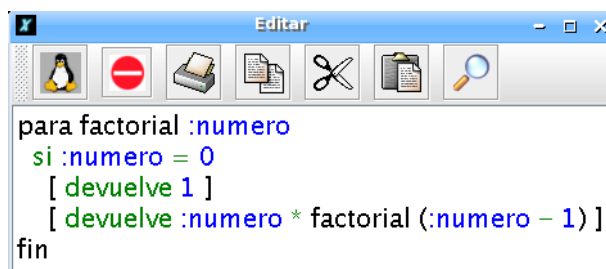
Un procedimiento se llama *recursivo* cuando se llama a sí mismo (es un *subprocedimiento* de sí mismo). Un ejemplo típico es el cálculo del **factorial**. En lugar de definir

$$n! = n * (n - 1) * \dots * 2 * 1,$$

podemos hacer:

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

En XLogo:



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

que va llamándose a sí mismo en cada ejecución, pero cada vez con el número reducido en una unidad. Cuando dicho número es cero, devuelve "1", y se termina la recursión. Es decir:

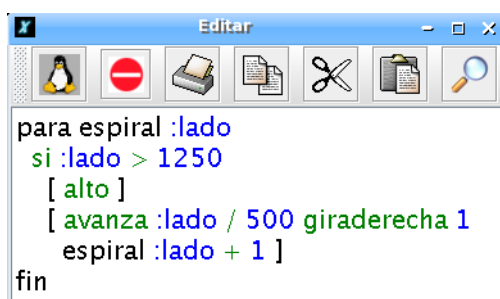
```

factorial 5
-> 5 * factorial 4
    -> 4 * factorial 3
        -> 3 * factorial 2
            -> 2 * factorial 1
                -> 1 * factorial 0
                    -> 1

```

siendo el resultado 120.

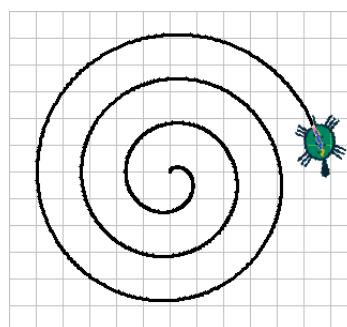
Un segundo ejemplo recursivo es la **espiral**:




```

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

```



que en cada llamada aumenta en una unidad el avance de la tortuga, siendo el límite 1250.

	<p>La recursividad o recurrencia es muy importante en robótica, cuando un sistema permanece “en espera” (ver ejemplo verbos.lgo). Sin embargo, debe ser utilizada con cuidado, ya que cada llamada recursiva va ocupando un espacio de memoria en el ordenador, y no la libera hasta que se cumple la condición final.</p>
---	--

Muchas de las cosas que pueden conseguirse con la recursividad pueden hacerse con bucles, pero eso conlleva en algunos casos complicar el diseño y dificultar la lectura del código y/o ralentizar su ejecución.

11.6. Retomando el ejemplo

Recuperemos el problema anterior:

Halla el número n tal que, la suma de sus dígitos más él mismo, dé 2002:

$$n + S(n) = 2002$$

¿Cómo haríamos para resolverlo recursivamente? Sencillamente modificando el procedimiento `suma.digitos`:

```
para resuelve
  borratexto
  repite 2002
    [ si 2002 = suma suma.digitos contador contador
      [ escribe contador ] ]
fin

para suma.digitos :numero
  si numero? :numero
    [ devuelve suma (primero :numero) (suma.digitos menosprimero :numero) ]
    [ devuelve 0 ]
fin
```

que devuelve las mismas (obviamente) soluciones que los programas anteriores:

$$n = 1982 \quad \text{y} \quad n = 2000$$

Observa que el procedimiento `suma.digitos` se llama a sí mismo, pero quitando el primer dígito de "numero".



Intenta ponerte en el lugar de la tortuga, coge un papel y realiza los mismo cálculos que ella. Por supuesto, elige solamente dos o tres números como muestra, siendo uno de ellos una de las soluciones. De esa manera podrás entender mejor la recursividad.

11.7. Ejercicios

1. Plantea un programa recursivo que calcule potencias de exponente natural
2. Plantea un programa recursivo que calcule el término n -simo de la sucesión de Fibonacci. Esta sucesión se obtiene partiendo de 1, 1, y cada término es la suma de los dos anteriores:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots$$

3. Diseña un procedimiento recursivo que devuelva la suma de los n primeros números pares (excluido el cero):

$$2 + 4 + 6 + 8 + \dots + 2n$$

4. Diseña el procedimiento `cuadrados` que tenga de entrada `lado` y dibuje, recursivamente, cuadrados de lados 15, 25, 35, 45, 55, 65 y 75. Es decir, debe ir incrementando el lado de 10 en 10, y debe tener un condicional para parar.
5. Diseña un programa `cuadrados_1000` que tenga una entrada `numero` y escriba los números naturales que sean menores que 1000 y cuadrados de otro natural.

Pista: No se trata de ir comprobando qué números son cuadrados perfectos, sino generar con un programa recursivo los cuadrados de los sucesivos naturales y que no pare mientras estos cuadrados sean menores que 1000

6. En la sección 7.5 obtuviste un procedimiento que dibujaba un polígono inscrito en una circunferencia. Modifica ese procedimiento para que:
 - a) Dibuje también un polígono circunscrito
 - b) Calcule el perímetro de ambos (inscrito y circunscrito)
 - c) Divida ambos resultados entre el doble del radio y lo muestre

Haz que el dibujo vaya aumentando progresivamente el número de lados, y piensa si el valor que obtienes te recuerda a algún número.

11.8. Uso avanzado de procedimientos

11.8.1. La primitiva devuelve

Es posible conseguir que un procedimiento se comporte como una función similar a las antes definidas en XLOGO. Por ejemplo:

```
para con_IVA :precio :IVA
# Este procedimiento aumenta el precio con el IVA
  devuelve :precio * (1 + :IVA / 100)
fin
```

permite escribir:

```
escribe (con_IVA 134 7 + con_IVA 230 16)
```

algo que no sería posible si usáramos `escribe` en vez de `devuelve`.

11.8.2. 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 poligono :vertices [ :lado 100 ]
  repite :vertices
    [ avanza :lado giraderecha 360/:vertices ]
fin

```

El procedimiento se llama `poligono`, lee una variable *forzosa* `vertices` 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

```
poligono 8
```

Durante la ejecución, la variable `:lado` se sustituye por su valor por defecto, esto es, 100, y XLOGO dibuja un octógono de lado 100. Sin embargo, ejecutando

```
(poligono 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.

11.8.3. 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 definido antes.

```

trazado
escribe factorial 4
factorial 4
factorial 4
  factorial 3
    factorial 2
      factorial 1
        factorial devuelve 1
      factorial devuelve 2
    factorial devuelve 6
  factorial devuelve 24
factorial devuelve 24
24

```



Capítulo 12

Recibir entrada del usuario

12.1. Comunicación con el usuario

Ya conocemos varias primitivas que permiten escribir mensajes, bien en el Histórico de comandos, bien en pantalla o incluso con una ventana emergente:

Primitivas	Argumentos	Uso
<code>escribe</code>	a: cualquiera	Escribe en el Histórico de comandos el argumento.
<code>tipea</code>	a: número, palabra o lista	Idéntico a <code>escribe</code> , pero el cursor queda en la línea donde se mostró el contenido del argumento.
<code>rotula</code>	a: palabra o lista	Dibuja la palabra o lista especificada, en la posición actual, y en la dirección que está mirando.
<code>largoetiqueta</code>	a: lista	Devuelve, en píxels, la longitud en pantalla de la lista.
<code>mensaje, msj</code>	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 <i>click</i> en el botón “Aceptar”

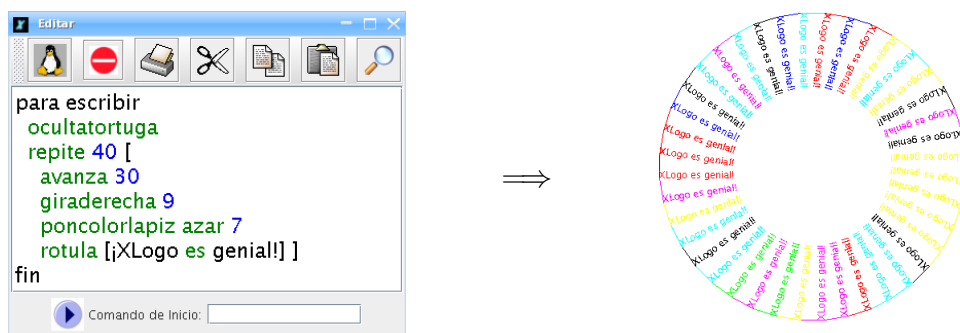
Llevamos ya tiempo trabajando con `escribe`, y utilizamos `tipea` al hablar de las operaciones (sección 7.1). Por su parte, `mensaje` apareció en el capítulo 3, en el programa `juego.lgo` para indicarnos si habíamos chocado con una “piedra” o llegado al “lago”.

La primitiva `largoetiqueta` permite saber, entre otras muchas posibilidades, 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?*

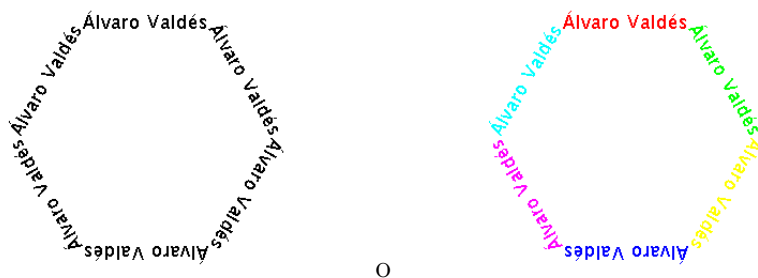
Ejemplo:



12.2. Ejercicios

1. Escribe tu nombre centrado en la pantalla
2. Plantea un procedimiento que recibe una lista como argumento, siendo esa lista el nombre y primer apellido de una persona. Con ello:
 - a) Determina el número de letras del nombre, n
 - b) Dibuja un polígono de n lados, siendo el lado el nombre y apellido antes dado.
 - c) Puedes intentar que cada lado sea de un color

En mi caso, sería: nombre.poligono [Álvaro Valdés], y el resultado:



3. Escribe un procedimiento que escriba en el Histórico de Comandos los números del 1 al 100, con diez en cada línea, es decir:


```

1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
...
91 92 93 94 95 96 97 98 99 100

```
4. Recupera el procedimiento `raiz_con_prueba` de la sección 9.2, y haz que cuando el radicando sea negativo, muestre un mensaje avisando de ello en una ventana, en vez de en el Histórico de Comandos.

12.3. Propiedades del Histórico de Comandos

Esta tabla define las primitivas que permiten ajustar y preguntar las propiedades de texto del área del Histórico de Comandos, es decir, las primitivas que controlan el color y tamaño del texto en este área.

Sólo afectan a las primitivas `escribe` y `tipea`. Para `rotula` se describen en la sección siguiente.

Primitivas	Argumentos	Uso
<code>borratexto, bt</code>	no	Borra el Área de comandos , y el área del Histórico de comandos .
<code>ponfuentetexto, pft</code>	n: número	Define el tamaño de la tipografía del área del Histórico de comandos . Sólo disponible para ser usada por la primitiva <code>escribe</code> .
<code>poncolortexto, pctexto</code>	a: número o lista	Define el color de la tipografía del área del Histórico de comandos . Sólo disponible para ser usada por la primitiva <code>escribe</code> .
<code>ponnombrefuentetexto, pnft</code>	n: número	Selecciona la tipografía número n para escribir en el área del Histórico de comandos con la primitiva <code>escribe</code> . Puedes encontrar la relación entre fuente y número en el menú Herramientas → Preferencias → Fuente .
<code>ponestilo, pest</code>	lista o palabra	Define los efectos de fuente para los comandos en el Histórico de comandos .
<code>fuentetexto, ftexto</code>	no	Devuelve el tamaño de la tipografía usada por la primitiva <code>escribe</code> .
<code>colortexto</code>	no	Devuelve el color de la tipografía usada por la primitiva <code>escribe</code> en el área del Histórico de comandos .
<code>nombrefuentetexto, nft</code>	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 <code>escribe</code> . El segundo elemento es una lista que contiene el nombre de la fuente.
<code>estilo,</code>	no	Devuelve una lista que contiene todos los efectos de fuente utilizados por las primitivas <code>escribe</code> y <code>tipea</code> .

Puedes elegir entre siete estilos:

- ninguno, utiliza la fuente sin ningún cambio
- **negrita**
- *cursiva*
- ~~tachado~~
- subrayado
- ^{superíndice}
- _{subíndice}

Si se quiere aplicar varios estilos a la vez, deben escribirse en una lista.

Ejemplos de estilos de fuente:

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

Devuelve:

Hola

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

Devuelve:

~~Tachado~~ x²

Las primitivas `character`, (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
character 125        devuelve }
```

12.4. Ejercicios

1. Escribe programas que, dado un valor r , calculen:

- a) el perímetro de una circunferencia
- b) el área de un círculo
- c) el área de una esfera
- d) el volumen de una esfera

en todos los casos de radio r , indicando claramente sus unidades y evitando que lea valores negativos:

```
perim.circ 5
```

El perímetro de una circunferencia de radio 5 es 3.141592 m²

2. Escribe un procedimiento con dos argumentos: **peso** y **altura**. y que calcule su índice de masa corporal (**i.m.c.**) mostrando cómo se calculó. El i.m.c. se calcula con la fórmula

$$\text{i.m.c.} = \frac{\text{peso}}{\text{altura}^2}$$

`i.m.c. 78 173`

Tu i.m.c. es `78 / 1732 = 26.061679307694877`

Un i.m.c. muy alto indica obesidad. Los valores “normales” de i.m.c. están entre 20 y 25, pero esos límites dependen de la edad, del sexo, de la constitución física, etc.

3. Escribe dos procedimientos que lean:

- a) una temperatura en grados Celsius y la devuelvan en Fahrenheit
- b) una temperatura en grados Fahrenheit y la devuelvan en Celsius

La relación entre grados Celsius (°C) y grados Fahrenheit (°F) es:

$$^{\circ}\text{F} - 32 = \frac{9}{5} \cdot (^{\circ}\text{C})$$

`C.a.F 35`

`35.0 °C son 95.0 °F`

OJO: se trata de obtener el “circulito” de “grados”: °, no vale usar el símbolo “primero”: º

4. Escribe un procedimiento que lea una fórmula química escrita directamente y la ponga en la forma habitual. Es decir, que convierta a los números en subíndices y deje las letras “normales”:

`fórmula.química H2S04 → H2SO4`

Pista: Utiliza el valor unicode asociado a los números

5. ¿Cómo ampliarías el procedimiento anterior para que, además, calculara la masa molecular del compuesto escrito?
6. Escribe un procedimiento que lea una lista de varias palabras y que la reescriba en el Histórico de Comandos en Mayúsculas, distinguiendo las que ya empezaban por mayúsculas:

`a.mayúsculas [Esto es un ejemplo]`

`ESTO ES UN EJEMPLO`

12.5. Escritura en Pantalla

En la sección anterior controlamos la escritura en el Histórico de Comandos. Modifiquemos parámetros al escribir en el Área de Dibujo.

Primitiva	Forma larga	Forma corta
ponnombrefuente, pnf	n: número	Modificar la tipografía con la que se escribe. Para ver la lista completa de fuentes disponibles, Menú Herramientas → Opciones → Pestaña Fuente
nombrefuente, nf	n: número	Devuelve el tipo de la letra con que se rotula.
ponfuente, pf	n: número	Modificar el tamaño de la tipografía con que se rotula. Por defecto, el tamaño es 12.
fuentes	no	Devuelve el tamaño de la letra con que se rotula.

Si queremos cambiar el color de las letras, debemos cambiar el color del lápiz.

12.6. Ejercicios

1. Observa la portada de este “libro”. ¿Cómo harías para conseguir la “sombra” azulada de las letras que forman la palabra XLOGO? (no hace falta que busques la misma fuente).
2. En la sección 10.2, creamos tres programas distintos para conjugar el futuro simple de un verbo regular. Modifica uno de ellos para que el resultado se muestre en el Área de Dibujo dispuesto. verticalmente
3. Haz lo mismo que en el ejercicio anterior, pero con el diseñado en 10.3 que conjuga presente y pasado, y añade una “etiqueta” encima de cada serie que indique el tiempo verbal.

12.7. 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, devuelve el valor unicode de 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	

En Mac, las teclas F1 a F12 no devuelven valor alguno, ya que el sistema las reserva para tareas “propias”.

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 [titulo] "palabra o leeteclado [titulo] "palabra`: Presenta una caja de diálogo titulada `titulo`. 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.

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
```

12.8. Ejercicios

1. Escribe un procedimiento que haga que la tortuga:

- dibuje en pantalla el número que se pulsa en el teclado
- rotule en pantalla las letras que se pulsen en el teclado

quedando en espera para dibujar o rotular más números o letras a la derecha de los ya introducidos. Necesitarás:

- a) Hacerlo recursivo, de modo que termine al pulsar **Esc**
- b) Ir desplazando a la tortuga hacia la derecha a medida que son dibujados los números o rotuladas las letras.
- c) OJO: Dibujados, no rotulados. Esto implica crear 10 procedimientos, uno por cada cifra.

No obstante, para ver si la parte que lee y desplaza a la tortuga está bien diseñada, puedes usar **rotula** para mostrar el resultado en el Área de Dibujo como con las letras.

- d) Para cambiar el tamaño de las letras, usa **ponfuente** (sección 12.5)

2. Escribe un procedimiento que lea una lista de varias palabras y que la reescriba en el Histórico de Comandos del siguiente modo:

- a) Si la palabra empieza por vocal, en cursiva (itálica)
- b) Si es un número:
 - 1) Como subíndice si es par
 - 2) Como superíndice si es impar
- c) Si la palabra empieza por consonante:
 - 1) Si está entre **b** y **l**, tachada
 - 2) Si está entre **m** y **z**, subrayada
- d) Si, además, empieza por mayúscula, en negrita

Pista: Observa que usando **unicode**, las letras mayúsculas **A - Z** devuelven valores consecutivos entre 65 y 90, mientras que las minúsculas **a - z** lo hacen entre 97 y 122. Debes tener cuidado, eso sí, con los caracteres especiales del castellano: vocales acentuadas y “eñes”.

3. Modifica el procedimiento **dibujar** para que suba o baje el lápiz al pulsar la barra espaciadora, borre al pulsar “borrar”, . . . y/o cualquier otra opción que se te ocurra (cambiar el color al pulsar la inicial del mismo,

12.9. Interactuar con el ratón

Durante la ejecución del programa, se pueden recibir eventos del ratón a través de tres primitivas: **leeraton**, **raton?** y **posraton**.

- `leeraton`: el programa hace una pausa hasta que el usuario hace un *click* o un movimiento. Entonces, da un número que representa ese evento. Los diferentes valores son:
 - 0 → El ratón se movió.
 - 1 → Se hizo un *click* izquierdo.
 - 2 → Se hizo un *click* central (Linux).
 - 3 → Se hizo un *click* derecho.

En Mac, sólo se reciben 0 y 1, ya que el ratón no tiene botón derecho, y la opción de mantener pulsado `Ctrl` no es interpretada por XLOGO como “botón derecho”.

- `posraton`: Da una lista que contiene la posición actual del ratón.
- `raton?`: Devuelve `cierto` o `falso` según toquemos o no el ratón desde que comienza la ejecución del programa

Ejemplos:

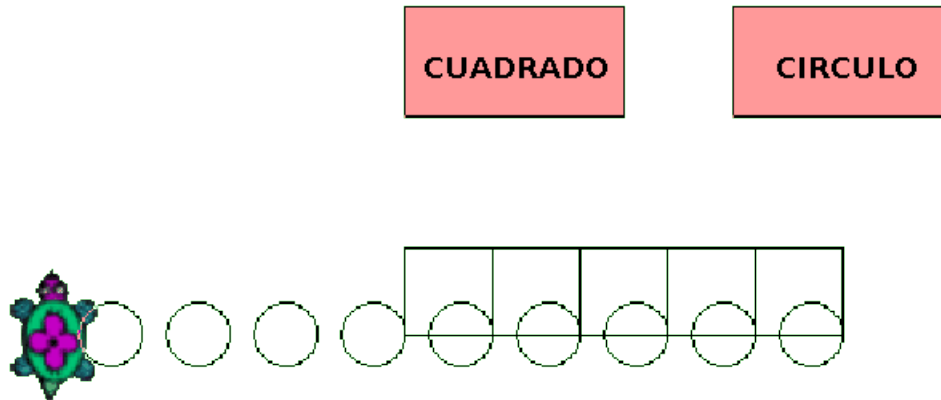
En este primer procedimiento, la tortuga sigue los movimientos del ratón por la pantalla.

```
para seguir
# cuando el raton se mueve, la tortuga cambia de posicion.
  si leeraton=0 [ponpos posraton]
  seguir
fin
```

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

```
para seguir2
  si leeraton = 1 [ponpos posraton]
  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 boton
# crea un boton rectangular color rosa, de 50 x 100.
  repite 2 [
    avanza 50 giraderecha 90 avanza 100 giraderecha 90 ]
  giraderecha 45 subelapiz avanza 10
  bajalapiz poncolorlapiz [255 153 153]
  rellena retrocede 10 giraizquierda 45 bajalapiz poncolorlapiz 0
fin

para empieza
  borrarantalla boton subelapiz ponpos [150 0] bajalapiz boton
  subelapiz ponpos [30 20] bajalapiz rotula "Cuadrado
  subelapiz ponpos [180 20] bajalapiz rotula "Circulo
  subelapiz ponpos [0 -100] bajalapiz
  raton
fin

para raton
# ponemos el valor de leeraton en la variable ev
# ponemos la primer coordenada en la variable x
# ponemos la segunda coordenada en la variable y
  haz "ev leeraton
  haz "x elemento 1 posraton
  haz "y elemento 2 posraton
# 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 [circulo]
    si :ev=3 [alto] ]
    raton
  fin

  para circulo
    repite 90 [av 1 gi 4]
    giraizquierda 90 subelapiz avanza 40 giraderecha 90 bajalapiz
  fin

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

```

12.10. Ejercicios

1. Diseña un procedimiento que dibuje un tablero de ajedrez, y determine el color de una casilla al hacer “*click*” sobre ella.
2. Observa el funcionamiento del ejemplo anterior, y piensa cómo podrías utilizarlo para diseñar un procedimiento que cargue en el área de dibujo un mapa “mudo” (Deberás utilizar la primitiva `cargaimagen`, sección 13.6.2) y el alumno tenga que ir haciendo “*click*” en un área según se le vaya mostrando el nombre de la misma (por ejemplo, Países, Comunidades Autónomas, Concejos, ...).

Simplifica lo más posible el planteamiento, ya veremos más adelante cómo “pulirlo” para tener en cuenta la forma de las fronteras.

3. Plantea un procedimiento que haga que la tortuga “persiga al ratón”. Para ello, la tortuga debe quedarse en un punto hasta que se hace *click* en otro, hacia donde se dirigirá despacio después de haberse orientado hacia allí. Puedes ralentizar el movimiento con un procedimiento del tipo:

```

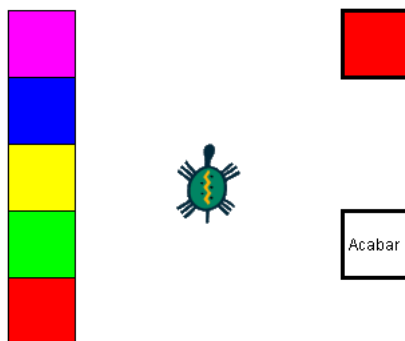
  para avance.lento :distancia
    repite :distancia
      [ avanza 1 espera 10 ]
  fin

```

Explicaremos `espera` en el capítulo 18.

4. Plantea un procedimiento que:
 - a) Dibuje varios cuadrados apilados y los rellene con distintos colores

- b) Dibuje otros dos, dejando uno en blanco y en el otro rotula “Acabar”
- c) Al hacer *click* en uno de los cuadrados coloreados, el que está en blanco adquirirá el color de aquél.
- Necesitarás la primitiva `encuentracolor`, que se explica en el capítulo siguiente.
- d) Al hacer *click* en el que dice “Acabar”, el programa terminará



5. **Este problema implica conocimientos de Física:** Plantea un procedimiento que calcule el potencial electrostático y/o gravitatorio en el origen de coordenadas de un sistema de cargas o masas ubicadas haciendo *click* con el ratón.

Con cada “*click*” del ratón, se debe dibujar un círculo centrado en ese punto.

12.11. 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).

12.11.1. Crear un componente gráfico

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

Crear un Botón

Usaremos al primitiva `botonigu`, cuya sintaxis es:

```
# Esta primitiva crea un boton 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 menu llamado m
# y que contiene 3 opciones: opcion1, opcion2 y opcion3
menuigu "m [opcion1 opcion2 opcion3]
```

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

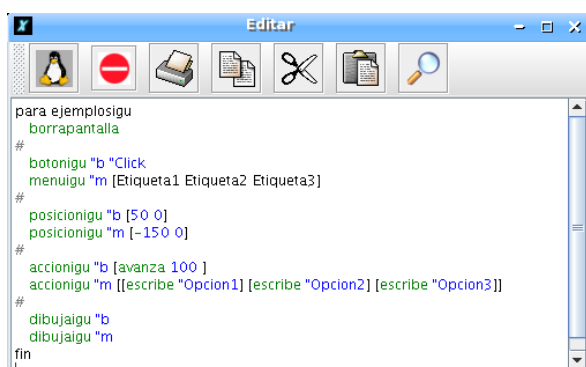
La primitiva `accionigu`, define una acción asociada al componente, y que se realizará cuando el usuario interactúa con él.

```
# Que la tortuga avance 100 al pulsar el boton "b
accionigu "b [avanza 100 ]
# En el menu, cada opcion indica su accion
accionigu "m [[escribe "Opcion1] [escribe "Opcion2] [escribe "Opcion3]]
```

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
```



Cambiamos el ejemplo anterior utilizando las nuevas primitivas:

para empieza

```

botonigu "Boton_Circ "Circulo
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 subelapiz avanza 40 giraderecha 90 bajalapiz
  
```

fin

para cuadrado

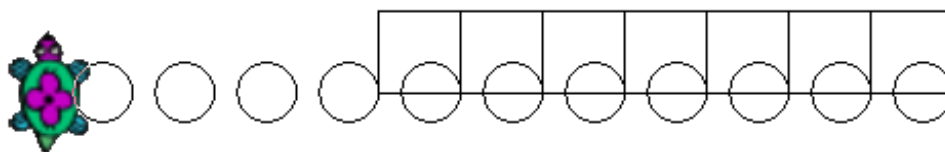
```

repite 4 [avanza 40 giraderecha 90]
giraderecha 90 avanza 40 giraizquierda 90
  
```

fin

Cuadrado

Círculo



12.12. Ejercicios

1. a) Crea cuatro botones: **Av**, **Re**, **Gd** y **Gi**, cuyas etiquetas sean, respectivamente: **Avanzar**, **Retroceder**, **Girar Derecha** y **Girar Izquierda**
- b) Ubica los botones anteriores en las posiciones `[-350 25]`, `[-360 -25]`, `[-300 0]` y `[-440 0]`
- c) Asígnales las acciones: **avanza 10**, **retrocede 10**, **giraderecha 90** y **giraizquierda 90**
- d) Dibuja los cuatro botones en pantalla y comprueba que “funcionan” correctamente.
- e) Crea dos menús **C1** y **G1**, cuyas opciones sean, respectivamente: **[Color Lápiz Azul Rojo Amarillo Verde Negro Blanco]** y **[Grosor Lápiz 1 2 3 4 5]**
- f) Asígnales las acciones:
 - 1) `[] [poncolorlapiz azul] [poncolorlapiz rojo] [poncolorlapiz amarillo] [poncolorlapiz verde] [poncolorlapiznegro] [poncolorlapiz blanco]`
 - 2) `[] [pongrosor 1] [pongrosor 2] [pongrosor 3] [pongrosor 4] [pongrosor 5]`

Observa que, en ambos casos, la primera opción NO hace nada

- g) Ubícalos en las posiciones `[-350 100]` y `[-350 70]`
- h) Dibuja los dos menús en pantalla y comprueba que “funcionan” correctamente.

Compara el resultado de los ejercicios con el XLOGO con que iniciamos el curso. ¿Serías capaz de diseñar, asignar y dibujar los elementos que faltan?

2. Crea un programa sobre conjugación de verbos que:

- a) Lea un verbo con **leelista**, analizando previamente si ya ha sido definido uno o no
- b) Conjugue correctamente en presente, pasado y futuro simple (sólo verbos regulares)
- c) El tiempo verbal se elija con un menú contextual
- d) Rotule en pantalla el tiempo verbal elegido
- e) Rotule **¿Acertaste?** tras mostrar el tiempo verbal
- f) Se pueda cambiar el verbo con un botón

XLogo conjuga verbos regulares. ¿Y tú?



XLogo conjuga verbos regulares. ¿Y tú?

PRESENTE Yo canto

Tú cantas

Él/Ella canta

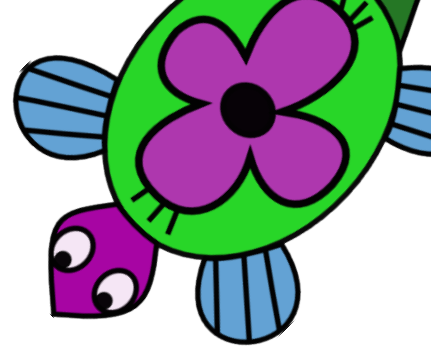
Nosotr@s cantamos

Vosotr@s cantáis

Ell@s cantan



¿Acertaste?



Capítulo 13

Técnicas avanzadas de dibujo

13.1. Más opciones para la tortuga

En la sección 4.4 presentamos algunas primitivas que controlan las opciones de la tortuga. Ampliemos las opciones:

Primitiva	Forma larga	Forma corta
muestratortuga, mt	no	Hace que la tortuga se vea en pantalla.
ocultatortuga, ot	no	Hace invisible a la tortuga.
pongrosor	n: número	Define el grosor del trazo del lápiz (en pixels). Por defecto es 1.
ponformalapiz, pfl	n: 0 ó 1	Fija la forma del lápiz: pfl 0: cuadrada; pfl 1: ovalada. Por defecto la forma es cuadrada.
grosorlapiz, gl	no	Devuelve el grosor del lápiz.
formalapiz, fl	no	Devuelve la forma del lápiz.

Entre otros, `ocultatortuga` es interesante en dos casos:

- No tapar parte del dibujo
- Dibujar más rápido

Analiza el procedimiento siguiente:

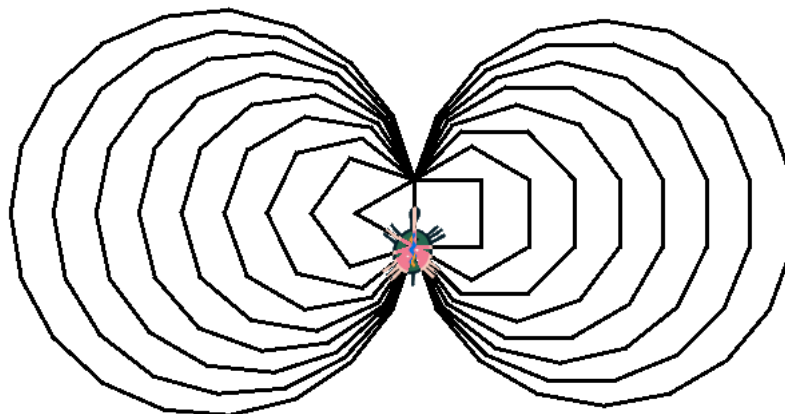
```
para poli.poligonos :cuantos
  repite :cuantos
```

```

[ hazlocal "angulo 360/(suma cuentarepite 2)
  si (0 = resto cuentarepite 2)
    [ haz "angulo (-:angulo)]
  repite (suma cuentarepite 2)
    [ avanza 50 giraderecha :angulo ] ]
fin

```

respuesta a un problema planteado en el capítulo 11.4:



Prueba a ejecutarlo de dos formas distintas:

```
borrapantalla poli.poligonos 100
```

```
borrapantalla ocultatortuga poli.poligonos 100
```

y observa cómo la segunda vez el dibujo se consigue mucho antes. (En el capítulo 18 veremos como medir exactamente el tiempo de ejecución de un programa).

Respecto a las demás primitivas, vamos a estudiar la siguiente hipótesis:

“Si pongo un grosor n y avanza n pasos, dibujaré un cuadrado”

Teclea las órdenes adecuadas y razona si la hipótesis es cierta o no.

Compara el resultado con el obtenido con la siguiente secuencia de comandos:

```
borrapantalla ponformalapiz 1 pongrosor 200 punto posicion
```

```
borrapantalla ponformalapiz 0 pongrosor 200 punto posicion
```

¿Qué observas? ¿Influye en el resultado del análisis anterior?



Prueba distintos dibujos modificando simultáneamente el grosor y la forma, observando claramente las diferencias entre un lápiz cuadrado y ovalado, cuándo se aprecia mejor esa diferencia, ...

13.2. Control del color

13.2.1. Primitivas que controlan los colores

Conozcamos ahora las primitivas que controlan el color del trazo, del papel (fondo) y del texto:


Primitiva	Forma larga	Forma corta
Cambiar el color del lápiz	poncolorlapiz n	poncl n
Cambiar el color del papel	poncolorpapel n	poncp n
Invertir el lápiz	inviertelapiz	ila
Averiguar el color del lápiz	colorlapiz	cl
Averiguar el color del papel	colorpapel	cp
Mirar el color de la posición [X Y]	encuentracolor [X Y]	ec [X Y]
Borrar por donde pasa	goma	go
Para volver a dibujar, debe usarse poncolorlapiz		

¿Qué es “invertir el lápiz”? Como veremos en la sección siguiente y detallaremos en 13.6, cada color en XLOGO está codificado usando tres valores: rojo, verde y azul, o R V A (RGB en inglés). Deberíamos conocer mínimamente la teoría del color asociada a la luz, que difiere de la que conocemos al tratar con pinturas:

<http://w3.cnice.mec.es/eos/MaterialesEducativos/mem2000/color/Intro/indice.htm>

Si trabajo sobre una paleta de pintor, sabemos que los colores básicos son tres, rojo, amarillo y azul; pero si son “colores de luz” los colores básicos cambian, y pasan a ser rojo, azul y verde y que si mezclo los colores obtengo:

Mezcla	Paleta	Luz
rojo + azul =	violeta	magenta
rojo + amarillo =	naranja	amarillo
azul + amarillo =	verde	cyan
azul + rojo + amarillo =	negro	blanco

	<p>Utiliza las opciones del menú para VER (literalmente) lo que ocurre al sumar colores en XLOGO. Menú Herramientas → Elegir color de lápiz (o de fondo) → pestaña RVA y “juega” con las barras de desplazamiento para comprobar lo que acabamos de contar.</p>
---	---

Pues bien, al “invertir el lápiz”:


















- si la zona del papel por la que pasa está en blanco, pinta del color activo
- si la zona del papel tiene un color traza, no la suma, sino la **resta** de colores:

$$\text{rojo} - \text{azul} = \text{verde} \quad \text{rojo} - \text{verde} = \text{azul} \quad \text{azul} - \text{verde} = \text{rojo}$$

Respecto a la primitiva `encuentracolor`, devuelve la lista [R V A] asociada al color, y que detallamos a continuación.

13.2.2. Descripción 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, que pueden ser indicados con su lista [r v a], con un número 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]	
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	marron	[153 102 0]	

Ejemplo: Estas tres órdenes son la misma:

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

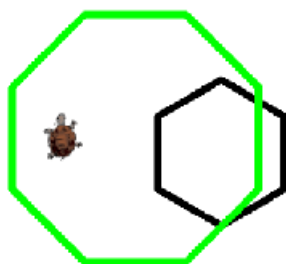
13.2.3. Función avanzada de relleno

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.

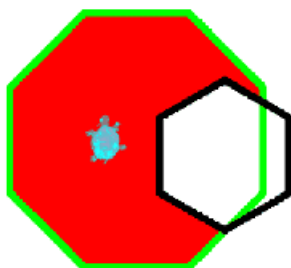
Veamos un ejemplo para explicar la diferencia entre estas dos primitivas:

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):



```
poncolorlapis 1
rellena
```

produce:

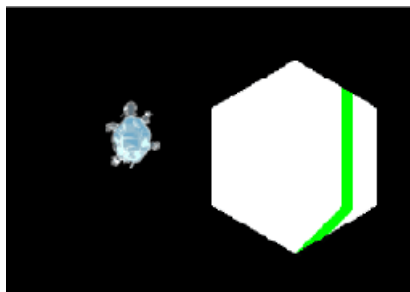


es decir, ha coloreado de rojo la región cerrada en la que se encuentra la tortuga.

Sin embargo, si hacemos:

```
poncolorlapiz 0
rellenazona
```

se obtiene:



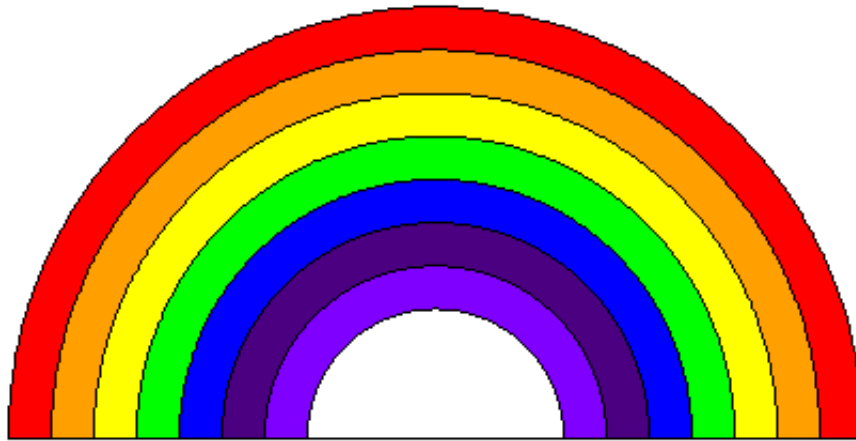
es decir, rellena todos los píxeles vecinos hasta encontrar una “frontera” del color activo.

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

```
para mediocirc :c
# dibuja un semicirculo de diametro :c
  repite 180 [
    avanza :c * tan 0.5
    giraderecha 1 ]
  avanza :c * tan 0.5
  giraderecha 90 avanza :c
fin

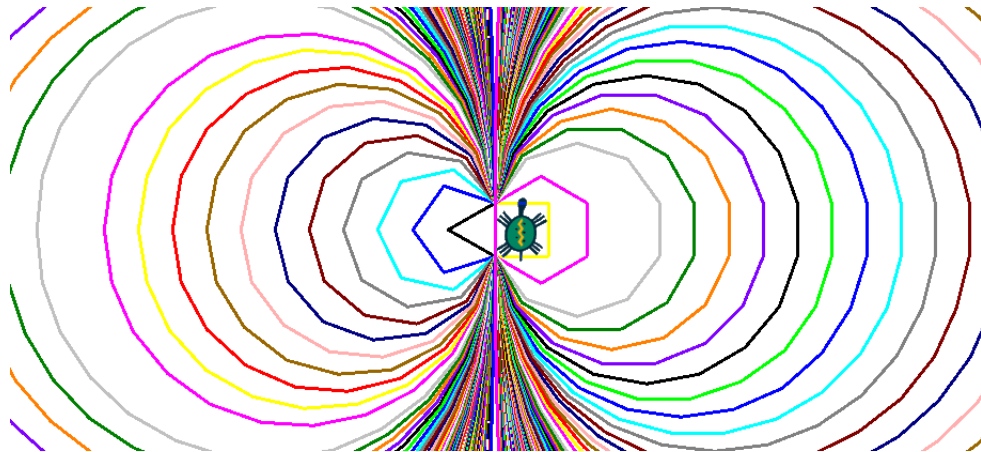
para arcohueco :c
# Utiliza el procedimiento mediocirc para dibujar un arcoiris sin colores
  si :c < 100 [alto]
  mediocirc :c
  giraderecha 180 avanza 20 giraizquierda 90
  arcohueco :c - 40
fin

para arcoiris
borrapantalla ocultatortuga arcohueco 400
subelapiz giraderecha 90 retrocede 150
giraizquierda 90 avanza 20 bajalapiz
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]
  [ poncolorlapiz elemento :colores :color rellena
    subelapiz giraderecha 90 avanza 20 giraizquierda 90 bajalapiz ]
fin
```

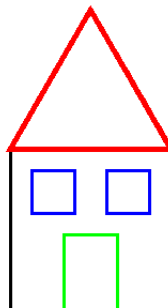


13.3. Ejercicios

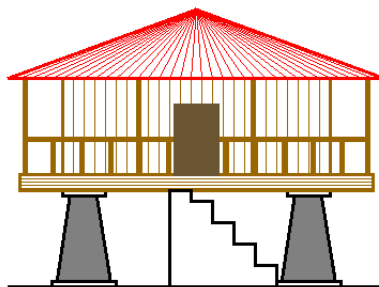
1. Modifica el procedimiento `poli.poligono` para obtener:



2. Combina adecuadamente los procedimientos `cuadrado`, `rectangulo` y `triangulo` que hemos ido diseñando a lo largo del libro para obtener la casa coloreada con la que presentamos a la tortuga “hace tiempo”:



3. ¿Te atreves con este hórreo asturiano con corredor?



4. En el capítulo anterior hablamos de usar un mapa mudo en el que el alumno tuviera que ir haciendo “click” en un área concreta según se le fueran mostrando nombres de la misma.

- a) ¿Cómo harías para tener en cuenta las fronteras reales?
- b) ¿Qué requisitos debería tener entonces el mapa para poder hacerlo?

13.4. Control del Área de dibujo

En esta sección podemos distinguir dos tipos de primitivas, las que controlan el tamaño del área de dibujo y las que determinan aspectos del dibujo:

13.4.1. Control del dibujo

Primitiva	Forma larga	Forma corta
zoom	a: número	Acerca o aleja el Área de dibujo.
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.
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

Ajusta el tamaño de la ventana (**Herramientas** → **Preferencias** → **Opciones** → **Tamaño de la ventana**) a 500 por 300, copia el siguiente procedimiento:

```
para modos
  borrar pantalla subelapiz ponx -170 bajalapiz
  repite 360 [avanza 3 giraderecha 1]
fin
```

y determina qué pasará en cada uno de los tres modos: **ventana**, **vuelta** y **jaula**.

¿Has podido? Este es el resultado:

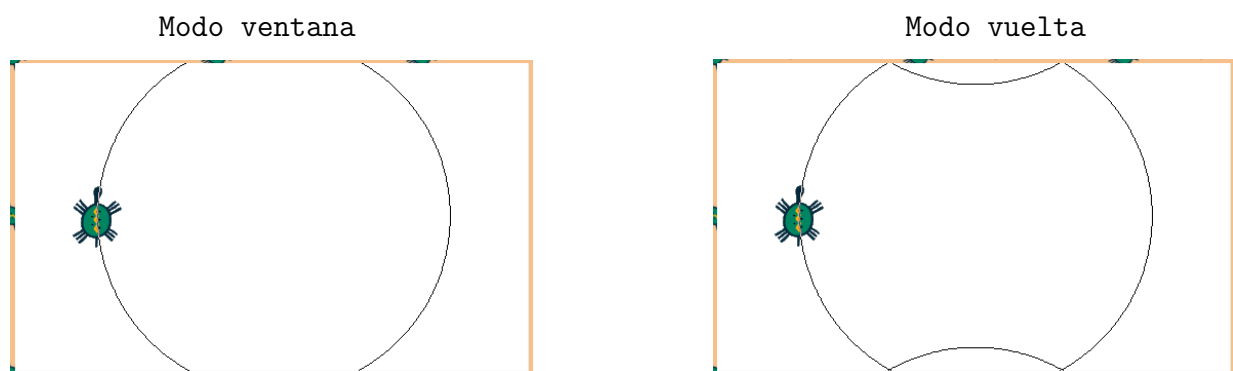
Obviamente, en **modovuelta** no se ha completado el dibujo y aparece el mensaje de error:

En modos, línea 2:


La tortuga sale de la pantalla.

Número de pasos antes de salir:0

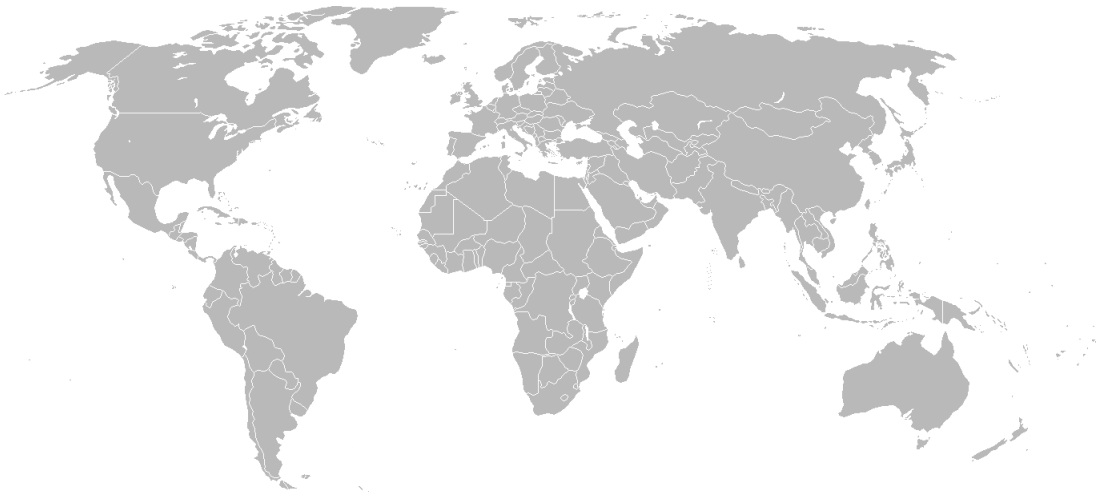
mientras que para **modovuelta** y **modoventana** nos encontramos con:



pero, cuidado, NO es que la tortuga haya rebotado en la “pared” superior, sino que vuelve a aparecer por la parte de abajo.



Prueba a cambiar el número de pasos que avanza la tortuga o ralentízala, para ver claramente cómo se comporta en el modo vuelta. ¿Se te ocurre para qué podríamos usar este modo? Una pista:



(Recuerda la primitiva cargaimagen)

13.4.2. Control de las dimensiones

Primitiva	Forma larga	Forma corta
pontamaño pantalla ptp	a: lista	Fija el tamaño de la pantalla.
tamaño pantalla, tpant	no	Devuelve una lista que contiene el tamaño de la pantalla
tamaño ventana, tv, esquinas ventana	no	Devuelve una lista con cuatro elementos, las coordenadas de la esquina superior izquierda y de la esquina inferior derecha.
ponseparacion, ponsep	n: número comprendido entre 0 y 1	Determina la proporción de pantalla ocupada por el Área de Dibujo y el Histórico de Comandos .
separacion	no	Devuelve el valor de la proporción de pantalla ocupada por el Área de Dibujo y el Histórico de Comandos .

El control de la separación también puede hacerse con el ratón, simplemente arrastrando la separación entre el Área de dibujo y el Histórico de comandos hacia arriba o hacia abajo. En `ponseparacion`, si `n` vale 1, el **Área de Dibujo** ocupará toda la pantalla. Si `n` vale 0, será el **Histórico** quien la ocupe.

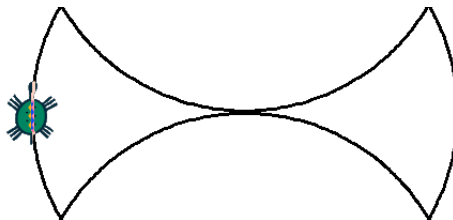
13.5. Ejercicios

1. Vamos a calcular el valor de π . Para ello, usa el procedimiento que dibuja una circunferencia (o mejor dicho un Trihectahexacontágono)¹ y:
 - a) Permite que el avance sea una variable (p.e. :lado)
 - b) Haz que la tortuga se desplace a través del diámetro hasta alcanzar el punto diametralmente opuesto al de partida, que localizará con `encuentracolor`
 - c) Determine la distancia hasta el punto de partida
 - d) Muestre nuestra estimación de π , el resultado de dividir:

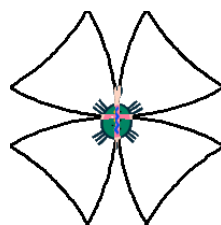
$$\pi \simeq \frac{360 \times \text{:lado}}{\text{distancia}}$$

Prueba para distintos valores de :lado y observa los resultados

2. En el ejemplo sobre `modojaula`, `modovuelta` y `modoventana`, ¿qué ocurre si usamos `circulo` para dibujar la circunferencia en vez del procedimiento anterior?
3. Observa el dibujo que obtuvimos con `modovuelta`. ¿A qué te recuerda? ¿Podrías usarlo para dibujar, por ejemplo, un murciélago?
4. ¿Cómo harías para que la circunferencia se divida y sea tangente **a sí misma**?

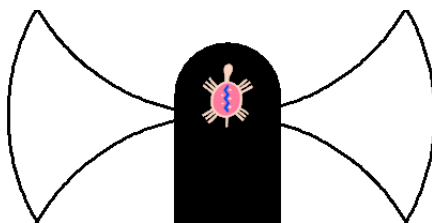


¿Y si hacemos que la pantalla sea cuadrada?:

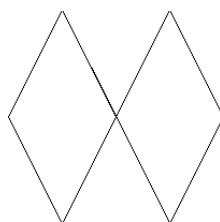


5. Aprovecha ese dibujo para que nuestra tortuga “hable” a través de dos altavoces:

¹Trihectahexacontágono: tri = 3, hecta = 100, hexaconta = 60, gono = ángulo



6. Intenta dibujar estos dos rombos:




(El tamaño de la pantalla es $200 * 200$)

7. ¿Podrías dibujar un rectángulo (o un cuadrado) **sin que haya giros de 90°** ?

(No vale sumar ángulos hasta que den 90°)

8. Dibuja los ejes cartesianos en pantalla

	<p>CUIDADO: Si pretendes usar estos dibujos como parte de otros redimensionando la pantalla a mitad de un dibujo, NO LO HAGAS. Al igual que cuadrícula, ejes, ... XLOGO borra la pantalla al modificar sus dimensiones. Debes guardarlo como imagen y cargarla después.</p>
---	---

13.6. Manejando imágenes

13.6.1. Introducción

Primero, algunas aclaraciones: Habrás visto en la sección 13.2.2 que el comando `poncolorlapiz` 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.

13.6.2. 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>

Utilizaremos la primitiva `cargaimagen` o `ci` que carga el archivo de imagen indicado con una palabra.

La esquina superior izquierda de dicha imagen se ubica en la posición actual de la tortuga. Los únicos formatos soportados son `jpg` y `png`. La ruta debe especificarse previamente con `pondirectorio` (capítulo 15) y debe ser absoluta, empezando en el nivel superior del árbol de directorios.

Ejemplo:

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

Así es como vamos a proceder: primero, nos referiremos al punto superior izquierdo como [0 0]. Luego, la tortuga examinará los primeros 100 puntos (píxeles) 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
```

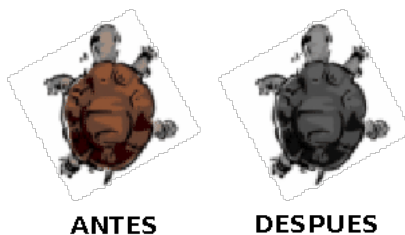
```
para escalagris
repitepara [y 0 -100 -1]
```

```

    [ 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

```



13.6.3. 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

```



ANTES

DESPUES

13.7. Ejercicios

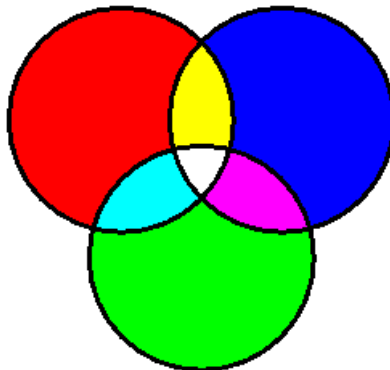
1. Carga la imagen de un laberinto (o mejor aún, que la tortuga dibuje uno) y puedas guiar a la tortuga por el camino de salida. Utiliza la primitiva `encuentracolor` para evitar que “atreviese” las paredes del mismo.

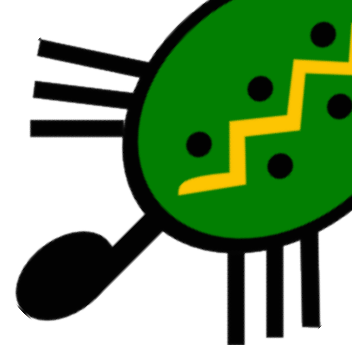
Intenta que la tortuga dibuje un rastro (recuerda, que debes dibujar el rastro después de encontrar el color del papel) de un color si el camino es el correcto y de otro cuando debes retroceder.

2. Busca o dibuja una casa para nuestra tortuga (no muy grande). Busca también imágenes de casas, parques, bancos, edificios gubernamentales y comerciales, . . . , y:

- a) Rediménsionalos hasta un tamaño de unos 50 píxeles.
 - b) Carga esas imágenes formando una “ciudad”
 - c) Diseña “recorridos”, con “tareas” que debe realizar la tortuga, y controla que esas tareas se realizan sin desplazarse por fuera de las “calles”.
 - d) Puedes dibujar puntos de distintos colores en las entradas de cada lugar que se debe visitar para controlar que los sitios a los que se desplaza son los correctos.
3. Vamos a dibujar un “Diagrama de Venn” de colores. Plantea un procedimiento que:
- a) Dibuje dos círculos de radios variables y los rellene con distintos colores
 - b) Analice si se cortan (dos circunferencias se cortan si la distancia entre sus centros es menor que la suma de los radios) y, en caso afirmativo:
 - 1) encuentre el color de cada círculo
 - 2) sitúe a la tortuga en la intersección de ambas circunferencias (por ejemplo, el punto medio entre sus centros)
 - 3) rellene esa intersección con el color resultante de sumar o promediar (tú eliges) ambos

¿Sabrías ampliarlo a tres circunferencias?





Capítulo 14

Modo multitortuga y Animación

14.1. Multitortuga

Se pueden tener varias tortugas activas en pantalla. Nada más iniciarse XLOGO, sólo hay una tortuga disponible y su número es 0.

14.1.1. Las primitivas

Estas son las primitivas que se aplican al modo multitortuga:

Primitiva	Argumentos	Uso
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.
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

Si quieres “crear” una nueva tortuga, puedes usar la primitiva `ponortuga` seguida del número de la nueva tortuga. Para evitar confusiones, 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 `ponortuga`.

El máximo número de tortugas disponibles también puede fijarse en el menú **Herramientas** → **Preferencias**.

14.1.2. Ejemplo

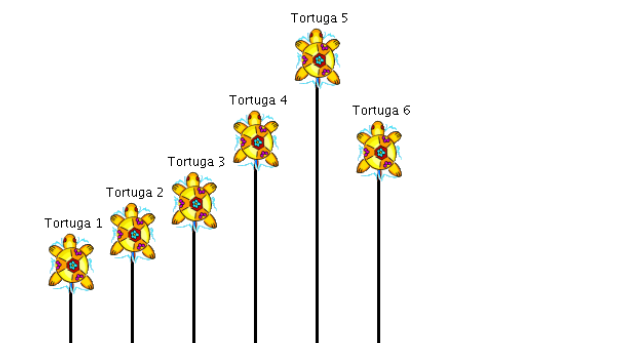
En este ejemplo vamos a ver la distribución de frecuencias cuando se lanzan dos dados y se calcula la suma de los puntos de ambos. En cada lanzamiento se obtiene un resultado comprendido entre 2 y 12, pero las frecuencias para cada número varían.

Para simular el lanzamiento de un dado, vamos a utilizar la primitiva `azar` (sección 7.3), dispondremos de 11 tortugas, numeradas de 2 a 12, que avanzarán un *paso de tortuga* cuando el resultado de la suma de la tirada de los dos dados coincida con su número.

La separación horizontal entre las tortugas es de 30 pasos de tortuga, y se colocarán a las tortugas con ayuda de los datos.

- Se colocará a la tortuga número 2 en $(-150 ; 0)$
- Se colocará a la tortuga número 3 en $(-120 ; 0)$
- Se colocará a la tortuga número 4 en $(-90 ; 0)$
- Se colocará a la tortuga número 5 en $(-60 ; 0)$
- ...

según la gráfica:



El programa final es:

```

para lanzar
  # simula el lanzamiento de un dado
  devuelve 1 + azar 6
fin

para inicia
  borrapantalla ocultatortuga
  repitepara [i 2 12]
  [ # coloca la tortuga
    pontortuga :i ponpos lista -150 + (:i - 2)*30 0
    # escribe el numero de la tortuga justo debajo
    subelapiz
    retrocede 15 rotula :i avanza 15
    bajalapiz ]
fin

para empezar
  inicia
  # Hacemos 1000 intentos
  repite 1000
  [ haz "suma lanzar+lanzar
    pontortuga :suma
    avanza 1 ]
  # indicamos las frecuencias de tirada
  repitepara [i 2 12]
  [ pontortuga :i
    # la ordenada de la tortuga representa el numero de tiradas
    hazlocal "frecuencia ultimo pos
    subelapiz avanza 10 giraizquierda 90
      avanza 10 giraderecha 90 bajalapiz
    rotula :frecuencia/1000*100 ]
fin

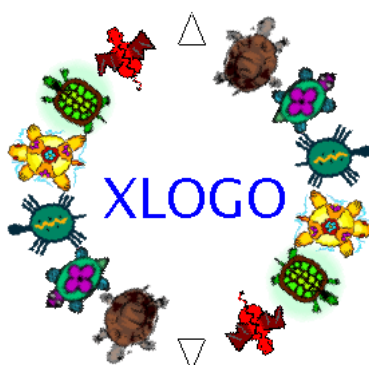
```

14.2. Ejercicios

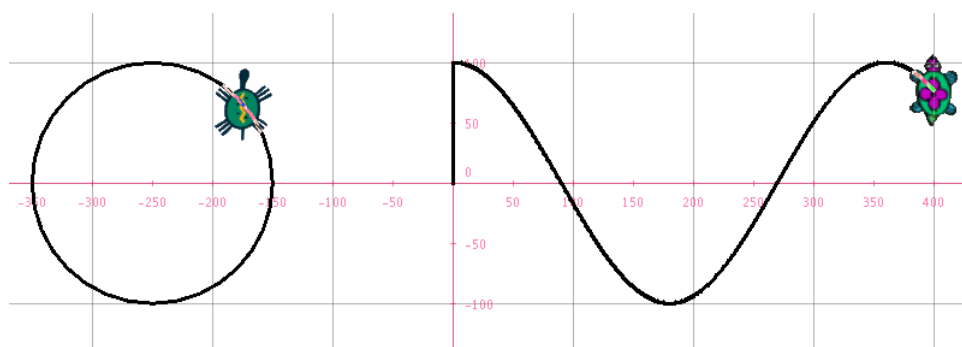
1. Utiliza el sistema mostrado en el ejemplo para leer una lista de datos discretos y que las tortugas vayan representando un diagrama de barras con ellos.

Pista: Deberás hacer más de una pasada a la lista, para determinar cuántos elementos distintos hay y ordenarlos de menor a mayor antes de releerla para hacer avanzar a las tortugas

2. Plantea un procedimiento que dibuje la “rosca” que aparece en la portada:



3. **Este problema implica conocimientos de Física:** Plantea un procedimiento que represente la muchas veces explicada analogía entre movimiento circular uniforme y movimiento armónico simple:



Para ello, necesitaremos tres tortugas:

- a) Una que describa el movimiento circular
- b) Otra que represente el paso del tiempo
- c) Una tercera que describa el movimiento armónico

La primera tortuga irá trazando una circunferencia a la par que la segunda avanza un pequeño número de pasos (debe ajustarse para que se vea bien el movimiento) y la tercera se desplaza de modo que:

- Su ordenada coincida con la de la tortuga 1
- Su abscisa coincida con la de la tortuga 2

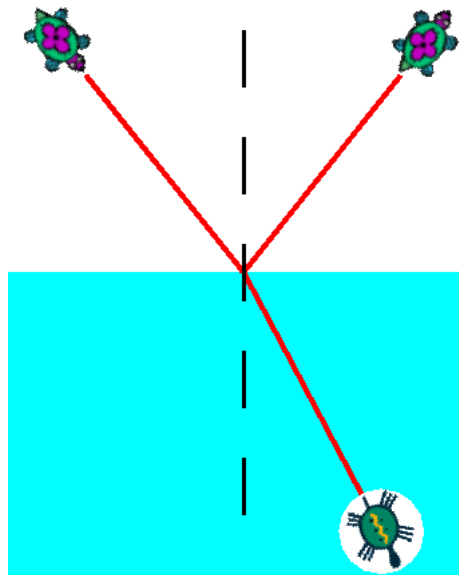
4. **Este problema implica conocimientos de Física:** Plantea un procedimiento que simule la reflexión y refracción de la luz. Para ello, necesitaremos dos tortugas, una para cada medio.

Debemos trazar una línea que represente la separación de medios, pudiendo colorear al menos uno para distinguirlos bien. Una tortuga se desplazará desde un punto (que puede dejarse como argumento) hasta el origen (orientada correctamente con ponrumbro) y allí

- Ella se “reflejará”, invirtiendo su desplazamiento vertical
- Apareceá la segunda tortuga que modificará su desplazamiento horizontal

en ambos casos, siguiendo las leyes de Snell:

$$\begin{aligned} \text{Reflexión:} \quad & \alpha_{\text{inc}} = \alpha_{\text{refl}} \\ \text{Refracción:} \quad & n_{\text{inc}} \cdot \text{sen } \alpha_{\text{inc}} = n_{\text{refr}} \cdot \text{sen } \alpha_{\text{refr}} \end{aligned}$$



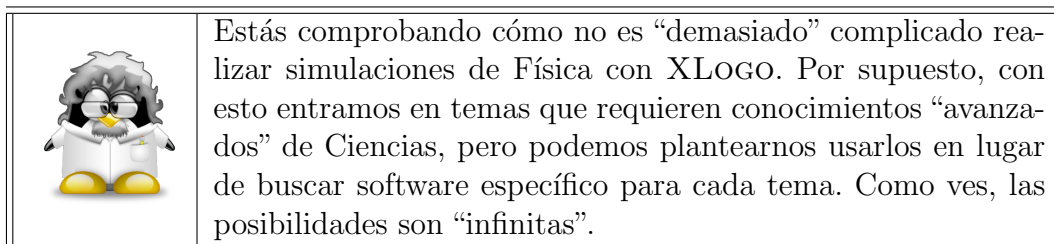
Puede “mejorarse” cambiando la forma de la tortuga para distinguir los dos “haces”

5. **Este problema implica conocimientos de Física:** Plantea un procedimiento que simule las fuerzas electrostáticas y gravitatorias de un sistema de masas o de cargas. Para ello:

- a) Ubicará n tortugas en posiciones que se indicarán como argumento
- b) Ubique otra tortuga con el ratón, con una “forma” distinta
- c) Determine vectorialmente la fuerza total en ese punto, y se desplace 3 ó 4 pasos en esa dirección

$$\vec{F} = K \cdot \frac{Q \cdot q}{r^2} \vec{u}_r$$

- d) Se detenga cuando la distancia a una de las tortugas sea, por ejemplo, 10



14.3. Animación

Existen dos primitivas llamadas `animacion` y `refrescar` que permiten escribir órdenes sin que la tortuga las realice. `animacion` hace que la tortuga dibuje pero no lo muestre, es decir, a nuestros ojos no hace nada; al recibir la orden `refrescar` muestra todo el trabajo almacenado en memoria.

Primitivas	Uso
<code>animacion</code>	Se accede al modo de animación.
<code>detieneanimacion</code>	Detiene el modo animación, retornando al modo <i>normal</i> .
<code>refrescar</code>	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 `refrescar`.



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

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

14.3.1. Ejemplo

Vamos a conseguir que un “camión” se desplace de izquierda a derecha de la pantalla. Empezamos por dibujar un camión (puedes cambiar el modelo, si este no te gusta) muy sencillo, un cuadrado como remolque, dos ruedas y una cabina simple:



```

para camion
  pongrosor 2 ocultatortuga
  repite 4
    [avanza 30 giraizquierda 90]
  giraderecha 90 avanza 15 giraizquierda 90 avanza 10
  giraizquierda 45 avanza (rc 2)*15 giraizquierda 135 avanza 25
  giraizquierda 90 goma avanza 5 ponlapiz circulo 5 rellena
  retrocede 20 goma retrocede 5 ponlapiz circulo 5 rellena
fin

```

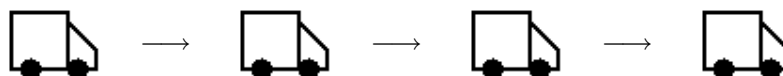
y concluimos con la parte asociada a la animación:

```

para moviendose
  animacion
  repitepara [ lugar -300 +300 2]
    [ borrarapantalla subelapiz ponx :lugar bajalapiz
      coche refrescar ]
  detieneanimacion
fin

```

El efecto final es el del camión desplazándose desde el punto $[-300\ 0]$ hasta el $[300\ 0]$



Si aún tienes dudas, en el **Tutorial** “Diez lecciones para aprender el lenguaje LOGO”

<http://xlogo.tuxfamily.org/sp/downloads-sp/tutorial.pdf>

hay una actividad sobre animación: El increíble monigote creciente y en la sección 18 mostraremos otra sobre las cifras de la calculadora (cuenta atrás).

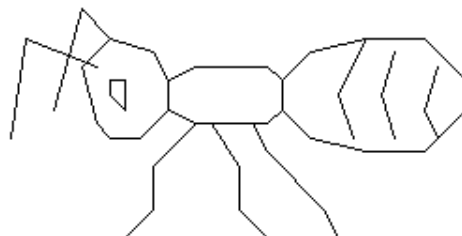
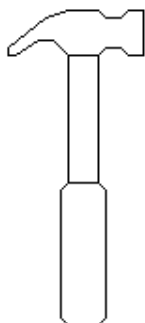
Este tipo de problemas pueden asociarse al estudio de los números complejos. Recordemos que la interpretación geométrica de un complejo permite sistematizar el análisis de:

- traslaciones
- rotaciones
- dilataciones

es decir, las **Homotecias**. En:

<http://neoparaiso.com/logo/numeros-complejos-aplicaciones.html>

muestran tres ejemplos de ello con las figuras de un coche, un martillo y una hormiga:



14.4. Ejercicios

Intenta reproducir con XLOGO las tres animaciones propuestas:

1. Desplazamiento de un coche
2. Giro de un martillo
3. Escalado de una hormiga

PERO usando correctamente las operaciones entre complejos.

Para representar los complejos con XLOGO deberás crear una lista con dos elementos, las coordenadas X e Y, y usar correctamente **primero** y **ultimo** para efectuar las operaciones.



Capítulo 15

Manejo de Archivos

15.1. Las primitivas

Como siempre, vamos a clasificar las primitivas en función de su objetivo:

15.1.1. Navegación por el sistema de archivos

Estas son las primitivas que nos permiten desplazarnos por el disco duro:

Primitivas	Argumentos	Uso
catalogo, cat	no	Lista el contenido del directorio actual. (Equivalente al comando <code>ls</code> de Linux, <code>dir</code> 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 <code>..</code> para referirse a la ruta del directorio superior.
directorio, dir	no	Da el directorio actual.

Supongamos la siguiente estructura de directorios en el disco duro de un alumno:

```
/home/alumno
--> /Documentos
    --> /Clase
    --> /XLogo
        --> /Programas
        --> /Capturas
--> /Escritorio
--> /Imagenes
```

Al iniciar XLOGO, la ruta de trabajo será `/home/alumno/`. Para llegar al directorio `Programas` podemos escribir alternativamente:

```
pondirectorio "/home/alumno/Documentos/XLogo/Programas
cambiadirectorio "Documentos/XLogo/Programas
```

Una vez en él, podemos ir hacia atrás de forma absoluta o relativa:

```
pondirectorio "/home/alumno/Documentos/XLogo
cambiadirectorio "..
```

y listar su contenido, que se mostrará en el Histórico de Comandos separando los directorios de los archivos:

```
catalogo
--> Directorio(s):
    Programas Capturas
    Archivo(s):
    manual.pdf prueba.lgo tortuga.png transfo.png xlogo.jar
```

Hay dos posibilidades de que esto no sea exactamente así:

- cuando se abre un archivo `.lgo` desde el Menú Archivo, el directorio donde se encontraba éste queda establecido como directorio de trabajo. Por ejemplo, si abrimos el fichero:

```
/home/alumno/Documentos/XLogo/rosa.lgo
```

el directorio de trabajo será

```
/home/alumno/Documentos/XLogo
```

- Si no es la primera vez que se trabaja con XLOGO. Al terminar una “sesión”, se guarda la última ruta de trabajo y se mantiene para ejecuciones posteriores.

Todo lo anterior afecta al guardado en disco de procedimientos o la carga desde disco duro de procedimientos e imágenes, NO a las capturas de pantalla que pueden hacerse desde el Menú Archivo → Capturar la imagen.

15.1.2. Carga y guardado de procedimientos

Finalizando el capítulo 5 te enseñamos a guardar en el disco duro usando las opciones de Menú:

- Menú Archivo → Guardar como
- Menú Archivo → Guardar

que difieren en que uno te permite asignar un nombre, y el otro sobrescribe el archivo abierto (si lo hay). Podemos también abrir procedimientos con:

Menú Archivo → Abrir

y disponemos de primitivas con las que conseguir lo mismo:

Primitivas	Argumentos	Uso
<code>carga</code>	<code>a: palabra</code>	Abre y lee el archivo <code>a</code> .
<code>guarda</code>	<code>a: palabra,</code> <code>l: lista</code>	Guarda en el archivo <code>a</code> los procedimientos especificados en <code>l</code> , en el directorio actual. (Ver ejemplo)
<code>guardatodo</code>	<code>a: palabra</code>	Guarda en el archivo <code>a</code> todos los procedimientos definidos, en el directorio actual. (Ver ejemplo)

Ejemplos:

- `carga "prueba.lgo` carga un archivo llamado `prueba.lgo` del directorio actual, SIN ABRIR el Editor de Comandos.
Difiere de la opción `Abrir` del Menú Archivo en que éste sí abre el Editor de Comandos.
- `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.

Es equivalente a usar la opción `Guardar` del Menú Archivo.

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, es decir:

```
borratodo carga "/home/alumno/XLogo/trabajo.lgo
```

no producirá ningún efecto.

15.1.3. Modificando archivos

Primitivas	Argumentos	Uso
abreflujo	n: número, a: nombre	Para poder leer o escribir en un fichero, es necesario crear un flujo hacia él. El argumento nombre debe ser su nombre, que se refiere al directorio de trabajo. El argumento n es el número que identifica a ese flujo.
cierraflujo	n: número	Cierra el flujo n.
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
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. 12.7)
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 cierra 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 cierra el fichero con cierraflujo .
finflujo?	n: número	Devuelve cierto si se ha llegado al final del fichero, y falso en caso contrario.

Ejemplo:

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

```

ABCDEFGHIJKLMÑOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
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 [ABCDEFGHJKLMNOPQRSTUVWXYZ]
escribelineaflujo 2 [abcdefghijklmnopqrstuvwxyz]
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

```

15.2. Ejemplo avanzado: obtención aproximada de π

Un resultado conocido de teoría de los números pone de manifiesto que la probabilidad que dos números tomados aleatoriamente sean primos entre ellos es de $\frac{6}{\pi^2} \simeq 0,6079$. Para intentar encontrar este resultado, vamos a:

- Tomar dos números al azar entre 0 y 1 000 000.
- Calcular su m.c.d.
- Si su m.c.d. vale 1, añadir 1 a una variable contador.
- Repetir 1000 veces
- La frecuencia de los pares de números primos entre ellos se obtendrá dividiendo la variable contador por 1000 (el número de pruebas)

```

para prueba
# Inicializamos la variable contador a 0
haz "contador 0
repite 1000
  [ si (mcd azar 1000000 azar 1000000) = 1
    [haz "contador :contador+1] ]
  escribe [Frecuencia:]
  escribe :contador/1000
fin

```

Nota: Como ya explicamos anteriormente, debemos a poner paréntesis en `mcd azar 1000000 azar 1000000` si no, el interprete va a intentar evaluar `1 000 000 = 1`. Para evitar este problema de paréntesis, escribe: `si 1 = mcd azar 1000000 azar 1000000`

```

prueba
0.609
prueba
0.626
prueba
0.597

```

Se obtienen valores próximos al valor teórico de 0,6097. Lo que es notable es que esta frecuencia es un valor aproximado de $\frac{6}{\pi^2} \simeq 0,6079$. Si tenemos en cuenta f , la frecuencia encontrada, se tiene pues: $f \simeq \frac{6}{\pi^2}$. Despejando:

$$\pi^2 \simeq \frac{6}{f} \quad \text{y así:} \quad \pi \simeq \sqrt{\frac{6}{f}}$$

Añadimos esta aproximación al programa, y transformamos el final del procedimiento prueba:

```

para prueba
# Inicializamos la variable contador a 0
haz "contador 0
repite 1000
  [ si (mcd azar 1000000 azar 1000000) = 1
    [ haz "contador :contador+1 ] ]
# Tras calcular la frecuencia
haz "f :contador/1000
# Mostramos el valor aproximado de pi
  escribe frase [Aproximacion de pi:] raizcuadrada (6/:f) fin
fin

```

que proporciona:

```
prueba
Aproximacion de pi: 3.164916190172819
prueba
Aproximacion de pi: 3.1675613357997525
prueba
Aproximacion de pi: 3.1008683647302115
```

Bien, modifiquemos el programa de modo que cuando lo lancemos, podamos indicar el número de pruebas deseadas.

```
para prueba :repeticiones
# Inicializamos la variable contador a 0
haz "contador 0
repite :repeticiones
  [ si (mcd azar 1000000 azar 1000000) = 1
    [ haz "contador :contador+1 ] ]
# Tras calcular la frecuencia
haz "f :contador/:repeticiones
# Mostramos el valor aproximado de pi
escribe frase [Aproximacion de pi:] raizcuadrada (6/:f)
fin
```

Probamos con 10000 repeticiones, y obtenemos en las tres primeras tentativas:

```
prueba 10000
Aproximacion de pi: 3.1300987144363774
prueba 10000
Aproximacion de pi: 3.1517891481565017
prueba 10000
Aproximacion de pi: 3.1416626832299914
```

No está mal, ¿verdad?

15.3. Compliquemos un poco más: π que genera π ...

¿Qué es un número aleatorio? ¿Es que un número tomado aleatoriamente entre 1 y 1.000.000 es un número realmente aleatorio?

Deberías darte cuenta rápidamente que nuestro modelo no hace más que aproximarse al modelo ideal. Bien, es precisamente sobre el modo de generar el número aleatorio sobre el que vamos a efectuar algunos cambios No vamos utilizar más la primitiva `azar`, sino que utilizaremos la secuencia de los decimales de π .

Me explico: los decimales de π siempre han intrigado a los matemáticos por su falta de regularidad, las cifras de 0 a 9 parecen aparecer en cantidades aproximadamente iguales

y de manera aleatoria. No se pueden predecir los decimales siguientes basándonos en los anteriores.

Vamos a ver a continuación como generar un número aleatorio con ayuda de los decimales de π . En primer lugar, debes obtener los primeros decimales de π (por ejemplo un millón). Existen dos programas que los calculan bastante bien. Aconsejamos PiFast para Windows y ScnhellPi para Linux.

Puedes acceder a Internet para conseguir un fichero de texto:

<http://3.141592653589793238462643383279502884197169399375105820974944592.com>

También en Internet, en la web de XLOGO:

<http://xlogo.tuxfamily.org/common/millionpi.txt>

Para crear los números aleatorios, agrupamos de 8 en 8 cifras la serie de decimales de π . Es decir, el fichero empieza así:

$$\underbrace{3,1415926}_{1^{\text{er}} \text{ numero}} \underbrace{53589793}_{2^{\text{o}} \text{ numero}} \underbrace{23846264}_{3^{\text{er}} \text{ numero}} \underbrace{33832795}_{\dots} 0288419716939\dots$$

Retiro la coma “,” del 3.14... que podría equivocarnos al extraer los decimales. Bien, todo está preparado, creamos un nuevo procedimiento llamado `azarpi` y modificamos ligeramente el procedimiento `prueba` para llamar al procedimiento `azarpi`:

```

para azarpi :n
  hazlocal "numero "
  repite :n [
# Si no hay ningun caracter en la linea
  si 0=cuenta :linea
    [haz "linea primero leelineaflujo 1]
# Asignamos a la variable :caracter el valor de primer caracter de la linea
  haz "caracter primero :linea
# despues eliminamos el primer caracter de la linea
  haz "linea menosprimero :linea
  haz "numero palabra :numero :caracter ]
  devuelve :numero
fin

```

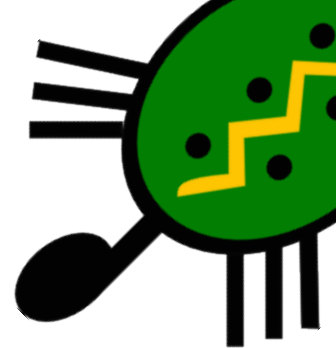
El resultado es:

```

prueba 10
Aproximacion de pi: 3.4641016151377544
prueba 100
Aproximacion de pi: 3.1108550841912757
prueba 1000
Aproximacion de pi: 3.081180112566604
prueba 10000
Aproximacion de pi: 3.1403714651066386

```

Encontramos pues una aproximación del número π ¡con ayuda de sus propios decimales!



Capítulo 16

Geometría de la tortuga en 3-D

16.1. 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!

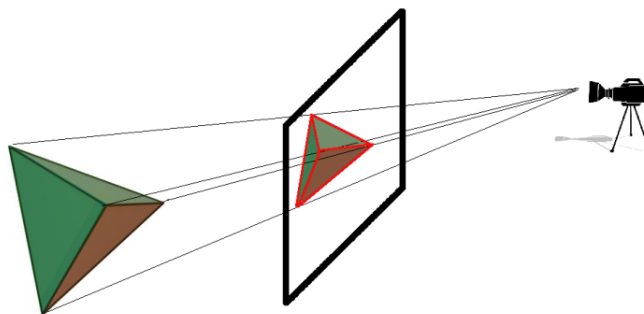
La tortuga cambia de forma y orientación, indicándonos en qué modo nos encontramos:



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

16.1.1. 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.

16.1.2. Entender la orientación en el mundo tridimensional

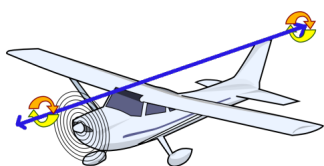
En el plano, la orientación de la tortuga se define únicamente por su rumbo. Sin embargo, en el mundo tridimensional la orientación de la tortuga necesita de tres ángulos. Si usamos la orientación por defecto de la tortuga en 3D (en el plano XY mirando hacia el semieje Y positivo):

Balaceo: la inclinación alrededor del eje OY

Cabeceo: la inclinación según el eje OX

Rumbo: la inclinación según el eje OZ

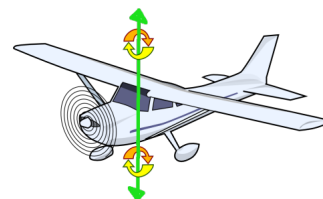
De hecho, para moverse en el mundo tridimensional, la tortuga se comportará de modo muy similar a un avión. De nuevo, ilustremos con una imagen los 3 ángulos:



Balaceo



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.

16.1.3. Primitivas

Estas son las primitivas básicas para moverse en el mundo 3D:

Primitiva	Forma larga
Pasar al modo 3D	perspectiva
Salir del modo 3D	modojaula, modovuelta o modoventana
Baja el "morro" n grados	cabeceabajo n ó bajanariz n , bn n
Sube el "morro" n grados	cabeceaarriba n ó subenariz n , sn n

Sube “ala” izquierda y baja “ala” derecha n grados	balanceaderecha n, bd n
Sube “ala” derecha y baja “ala” izquierda n grados:	balanceaizquierda n, bi n
Borrar Pantalla (y tortuga al centro, orientada “hacia nosotros” con el “timón de cola” hacia arriba)	borrarpantalla, bp
Devuelve la inclinación de las “alas”	balanceo
Devuelve la inclinación del “morro”	cabeceo

Además de las anteriores, podemos usar algunas de nuestras “viejas conocidas”:

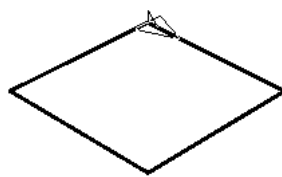
- avanza, av
- retrocede, re
- giraderecha, gd
- giraizquierda, gi

que realizan los mismos movimientos que en el “mundo 2D”.

Por ejemplo, en el plano bidimensional, para dibujar un cuadrado de 200 pasos de tortuga, escribimos:

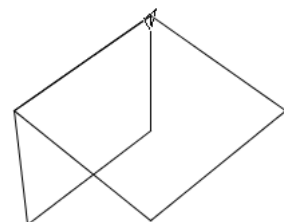
```
repite 4 [ avanza 200 giraderecha 90 ]
```

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:

```
borrarpantalla
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
```

es decir, que como ocurre en 2D, no todas las primitivas son necesarias, por ejemplo:

```
bajanariz 90
```

es equivalente a:

```
balancea 90 giraderecha 90 balanceaizquierda 90
```

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

16.2. Primitivas disponibles tanto en 2D como 3D

Las siguientes primitivas están disponibles en el plano o en el mundo 3D. La única diferencia son los argumentos admitidos por las primitivas. Estas precisan de los mismos argumentos que en el plano:

circulo	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, posicion
ponpos, ponposicion	punto	

16.3. 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 **ponposicion**, 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” (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**

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

Ejemplo: **ponorientacion [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.

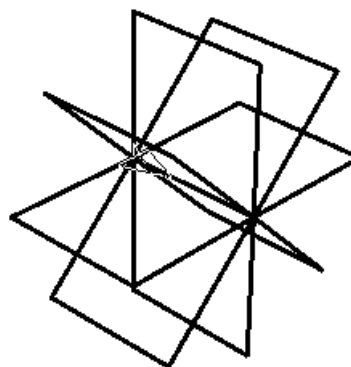
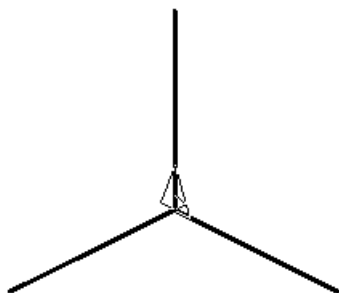
- **orientacion** Devuelve la orientación de la tortuga en una lista que contiene: [**balanceo cabeceo rumbo**]
- **ponbalanceo** La tortuga gira en torno a su eje longitudinal y adquiere el ángulo de balanceo elegido.
- **balanceo** Devuelve el valor actual del balanceo
- **ponbalanceo** La tortuga gira en torno a su eje transversal, y se orienta con el ángulo de cabeceo indicado.
- **balanceo** Devuelve el valor actual del cabeceo

Podemos dibujar los semiejes en la posición habitual tecleando, por ejemplo:

```
semieje y+: borrapantalla avanza 200 retrocede 200
```

semieje x+: giraderecha 90 avanza 200 retrocede 200

semieje z+: subenariz 90 avanza 200 retrocede 200



Podemos dibujar 8 planos verticales, cada uno girado respecto del anterior 45 grados, así:

```
borrapantalla
repite 8
  [ balanceaderecha 45
    repite 4
      [ avanza 100 giraderecha 90 ] ]
```

16.4. Ejercicios

1. Dibuja un taburete cuadrado con una pata en cada esquina.

Idea: repetir 4 veces “lado – pata”.

2. A partir del taburete anterior, dibuja una silla con el respaldo inclinado.
3. Reforma la silla anterior para dibujar una tumbona.
4. Dibuja una pirámide cuadrangular regular.

Idea: proceder como si fuera un taburete invertido, con las patas inclinadas 45 grados respecto del plano horizontal.

5. Dibuja un octaedro a partir de la pirámide anterior

16.5. El 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.

16.5.1. Reglas

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:

- **empiezapoligono, definepoligono:** Los movimientos de la tortuga posteriores a esta llamada se guardan para crear un polígono.
- **finpoligono:** Desde la ejecución de **definepoligono**, 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.

- **empieزالinea, definelinea:** Los movimientos de la tortuga posteriores a esta llamada se guardan para crear una banda (una línea).
- **finlinea:** Desde la ejecución de **definelinea**, 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 banda

- **empiezapunto, definepunto:** Los movimientos siguientes de la tortuga se guardan para crear un conjunto de puntos.
- **finpunto:** Esta primitiva finaliza el conjunto de puntos.
- **empiezatexto, definetexto:** Cada vez que el usuario muestre un texto sobre el Área de Dibujo con la primitiva **rotula**, se almacenará y luego será representado por el visor 3D.
- **fintexto:** Esta primitiva finaliza la grabación de texto.
- **vista3d, vistapoligono** 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:

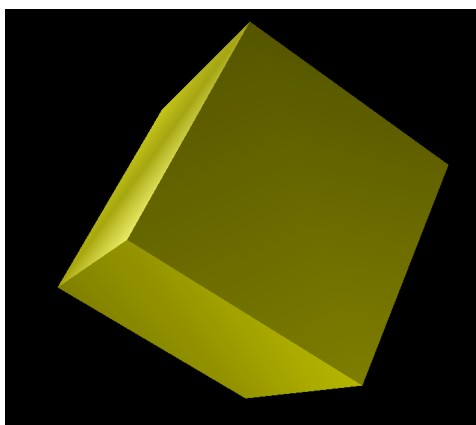
```

para cuadrado
# Grabamos los vertices del cuadrado
  empiezapoligono
  repite 4 [ avanza 400 giraderecha 90 ]
  finpoligono
fin

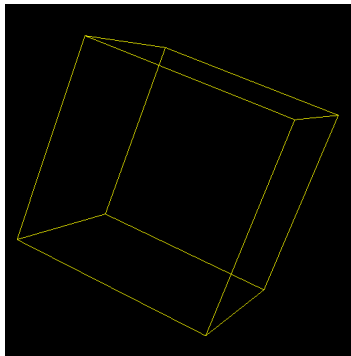
para cubosimple
# Cubo Amarillo
  borrapantalla perspectiva
  poncolorlapiz amarillo
# Caras laterales
  repite 4
    [ cuadrado subelapiz
      giraderecha 90 avanza 400 giraizquierda 90
      balanceaderecha 90 bajalapiz ]
# Parte inferior
  bajarariz 90 cuadrado subenariz 90
# Cara Superior
  avanza 400 bajarariz 90 cuadrado
# Visualizacion
  vista3d
fin

```

Estamos listos a ejecutar el comando: cubosimple:



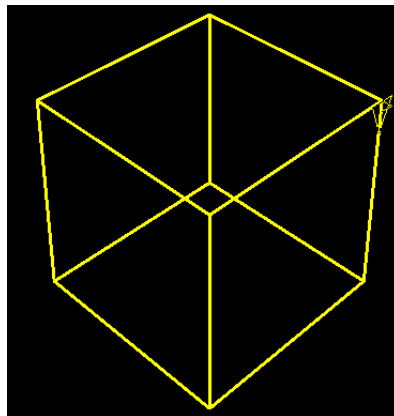
Al sustituir en el procedimiento cuadrado, empiezapoligono por empiezalinea, y finpoligono por finlinea:



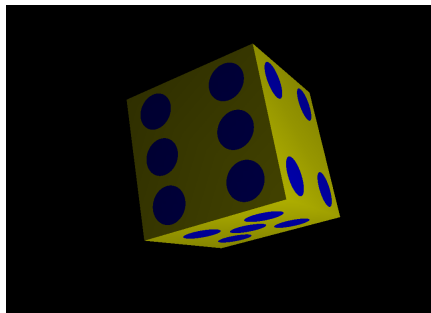
Si hubiéramos usado `empiezapunto` y `finpunto` en lugar de `empiezalinea` y `finlinea`, deberíamos ver en la pantalla sólo los ocho vértices del cubo.

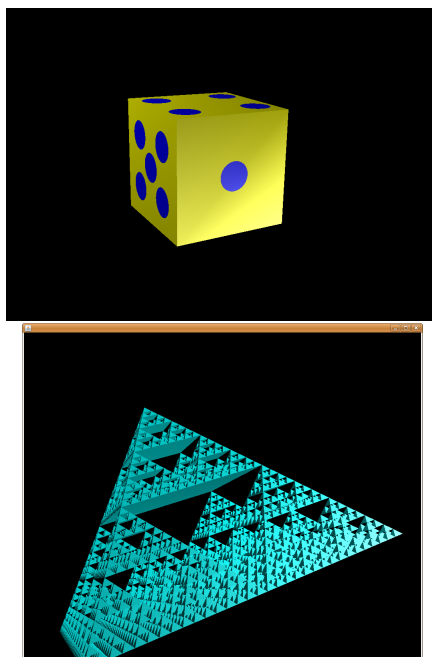
Estas primitivas son muy útiles para mostrar el conjunto de puntos en el espacio 3D.

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



16.6. Ejemplos





Efectos de luz y niebla

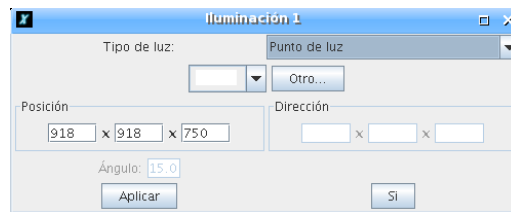
Desde la versión 0.9.93 se pueden añadir efectos artísticos a las imágenes generadas en el Visor. Estos pueden ser efectos de luz y de niebla, y se accede a ellos con los botones presentes en el visor 3D.



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:

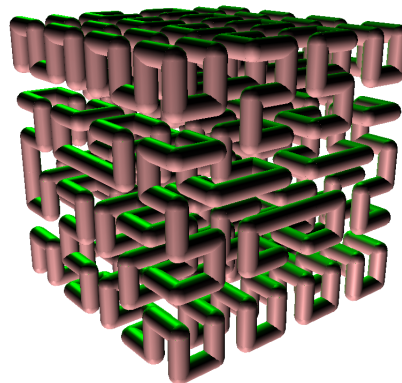
Luz Ambiente: Luz uniforme de la que sólo puede modificarse el color

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.

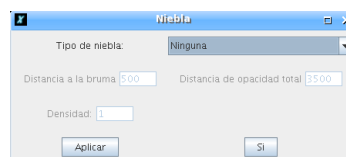
Foco: Es como el punto de luz, pero el haz de luz se abre formando un cono cuya abertura debe fijarse.

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:



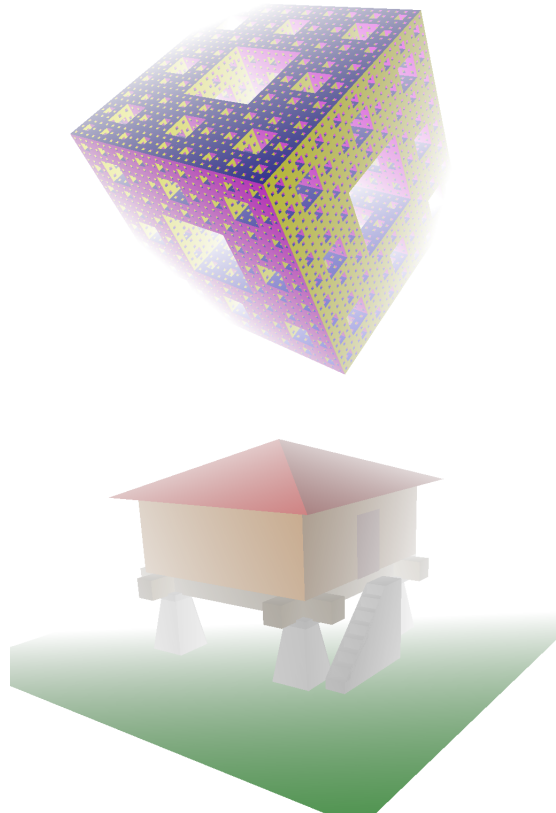
Disponemos de dos tipos de niebla:

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:



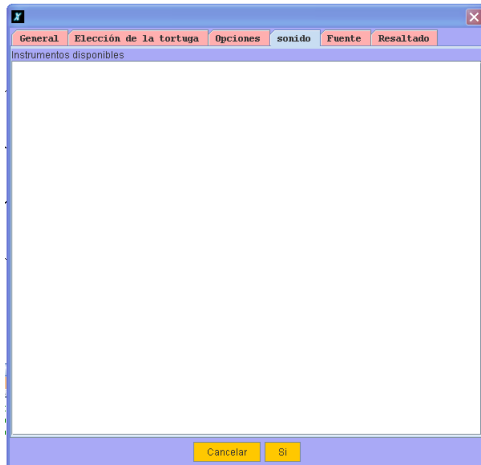


Capítulo 17

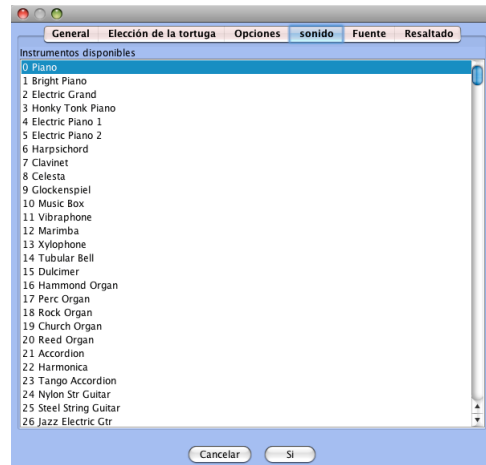
Tocar música (MIDI)

Ya comentamos anteriormente (sección 1.2.1) que la versión 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. Es importante recordarlo porque, si no lo haces, con la instalación por defecto de JAVA no tendrás instrumentos disponibles:

Instalación por defecto de JAVA



En Windows



En Linux y Mac

Puedes hacer pruebas con los distintos bancos de sonidos y ver las diferencias entre ellos para elegir el que más te guste, si bien el mínimo es más que suficiente para nuestros objetivos.

Ten en cuenta que el intérprete JAVA selecciona automáticamente el banco de mayor “calidad”, así que vete probando de “menor a mayor”, sin borrar el anterior hasta que te hayas decidido.

17.1. Las primitivas

Primitivas	Argumentos	Uso
secuencia, sec	a: lista	Carga en memoria la secuencia incluida en la lista. Siguiendo a esta tabla, se indica cómo escribir una secuencia de notas musicales.
tocamusica	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.
borrarsecuencia, 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]
```


- Quizá sea necesario que las redimensiones con algún programa externo, aunque existen formas de hacerlo con XLOGO
 - Por comodidad, la imagen del lago deberá tener un color uniforme
2. Distribuye aleatoriamente las imágenes de las piedras por la pantalla
 3. Ubica el lago en una esquina y a la tortuga en la contraria.
 4. Elige una melodía (te aconsejamos una sencilla, pero si te gusta mucho una, eres libre de usarla) y haz que suene cuando la tortuga llegue al lago.
 5. Crea los botones **derecha** e **izquierda** (ya lo hiciste en 12.12), pero modifica el de **avanza** y **retrocede** para que llamen a un procedimiento que vaya mirando el color según se mueve la tortuga
 - Si ese color es el del lago, que aparezca un mensaje de felicitación y que suene la melodía.
 - Si, por el contrario, no es el del fondo o el del lago, que aparezca un mensaje de “Inténtalo de nuevo” y devuelva la tortuga al inicio.



Capítulo 18

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).

18.1. Las primitivas

Primitivas	Argumentos	Uso
espera	n: número entero	Hace una pausa en el programa, la tortuga espera (n/60) segundos.
cronometro, crono	n: número entero	Inicia un conteo descendiente de n segundos. Para saber que la cuenta ha finalizado, disponemos de la primitiva <code>fincrono?</code>
<code>fincronometro?</code> , <code>fincrono?</code>	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.

En la sección 13.1 comentábamos que los dibujos eran más rápidos con la tortuga oculta que con ella en pantalla. Si añadimos las siguientes líneas al principio y al final del procedimiento:

```

para nombre.proc :a :b ...
  haz "tardanza tiempo
  ...
  ...
  escribe :tardanza - tiempo
fin

```

obtendremos los segundos que tardó en ejecutarse el procedimiento.

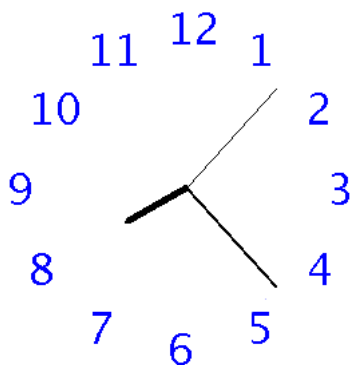
Veamos otro ejemplo:

```

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 palabra palabra palabra :h ": :m ": :s crono 5 ]
reloj
fin

```

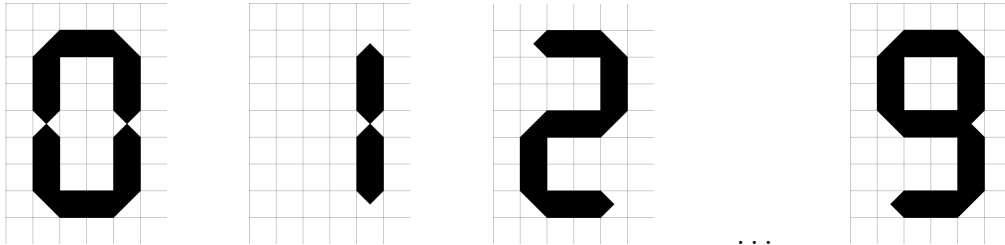
Podríamos plantearnos la forma de representar un reloj analógico:



¿Se te ocurre cómo hacerlo?

18.2. Actividad sobre las cifras de una calculadora

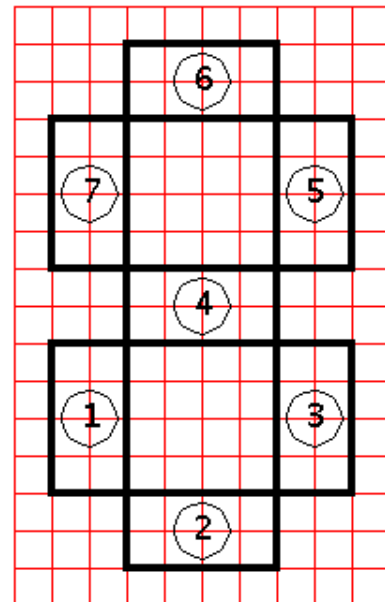
Al hablar de cifras de calculadora, nos referimos a las formas “tradicionales”:



aunque nada impediría hacerlo con las formas *pixeladas* más modernas. De todas formas, simplificaremos aún más las cifras para centrarnos en los aspectos relacionados con el tiempo.

Esta actividad se basa en el hecho de que todos los números de calculadora pueden obtenerse con ayuda de un patrón sí-no:

- para dibujar un “4”, “encenderemos” los rectángulos 3, 4, 5 y 7, pero no los 1, 2 y 6
- para dibujar un “8”, “encenderemos” los rectángulos 1, 2, 3, 4, 5, 6 y 7.
- para dibujar un “3”, encenderemos los rectángulos 2, 3, 4, 5 y 6, pero no los 1 y 7
- ...



18.2.1. El programa

Crearemos un rectángulo sólido de modo recursivo:

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

Supondremos aquí que la tortuga parte de la esquina inferior izquierda. Vamos a definir un procedimiento llamado `cifra` que lee 7 argumentos `:a`, `:b`, `:c`, `:d`, `:e`, `:f` y `:g`.

- Cuando `:a` vale 1, dibuja el rectángulo 1. Si `:a` vale 0, no se dibuja.
- Cuando `:b` vale 1, dibuja el rectángulo 2. Si `:b` vale 0, no se dibuja.
- Cuando `:c` vale 1, dibuja el rectángulo 1. Si `:c` vale 0, no se dibuja.
- ...

Se obtiene el siguiente procedimiento:

```
para cifra :a :b :c :d :e :f :g
# Dibujamos el rectangulo 1
  si :a=1 [rectangulo 160 40]
# Dibujamos el rectangulo 2
  si :b=1 [rectangulo 40 160]
  sl gd 90 av 120 gi 90 bl
# Dibujamos el rectangulo 3
  si :c=1 [rectangulo 160 40]
  sl av 120 bl
# Dibujamos el rectangulo 5
  si :e=1 [rectangulo 160 40]
# Dibujamos el rectangulo 4
  gi 90 sl re 40 bl
  si :d=1 [rectangulo 160 40]
# Dibujamos el rectangulo 6
  gd 90 sl av 120 gi 90 bl
  si :f=1 [rectangulo 160 40]
# Dibujamos el rectangulo 7
  sl av 120 gi 90 re 40 bl
  si :g=1 [rectangulo 160 40]
fin
```

18.2.2. Creación de una pequeña animación

Vamos a simular una cuenta atrás, que consiste en hacer aparecer sucesivamente las cifras de 9 a 0 en orden decreciente.

```
para cuentatras
  bp ot cifra 0 1 1 1 1 1 1 espera 60
  bp ot cifra 1 1 1 1 1 1 1 espera 60
  bp ot cifra 0 0 1 0 1 1 0 espera 60
  bp ot cifra 1 1 1 1 0 1 1 espera 60
```

```

bp ot cifra 0 1 1 1 0 1 1 espera 60
bp ot cifra 0 0 1 1 1 0 1 espera 60
bp ot cifra 0 1 1 1 1 1 0 espera 60
bp ot cifra 1 1 0 1 1 1 0 espera 60
bp ot cifra 0 0 1 0 1 0 0 espera 60
bp ot cifra 1 1 1 0 1 1 1 espera 60
fin

```

Pequeño problema: hay un efecto de parpadeo desagradable durante la creación de cada cifra. Para suavizarlo se van a utilizar las primitivas `animacion` y `refrescar`.

Se obtiene el siguiente programa modificado:

```

para cuentatras
# Entramos en modo animacion
animacion
bp ot cifra 0 1 1 1 1 1 1 refrescar espera 60
bp ot cifra 1 1 1 1 1 1 1 refrescar espera 60
bp ot cifra 0 0 1 0 1 1 0 refrescar espera 60
bp ot cifra 1 1 1 1 0 1 1 refrescar espera 60
bp ot cifra 0 1 1 1 0 1 1 refrescar espera 60
bp ot cifra 0 0 1 1 1 0 1 refrescar espera 60
bp ot cifra 0 1 1 1 1 1 0 refrescar espera 60
bp ot cifra 1 1 0 1 1 1 0 refrescar espera 60
bp ot cifra 0 0 1 0 1 0 0 refrescar espera 60
bp ot cifra 1 1 1 0 1 1 1 refrescar espera 60
# Volvemos al modo de dibujo habitual
detieneanimacion
fin

```

18.3. Ejercicios

1. Diseñemos un juego. A ver quién “cuenta” mejor un minuto mentalmente.
 - a) Debe haber un botón de “Inicio” (sólo nos preocupa el *click* del ratón)
 - b) Al hacer *click* sobre el botón, la tortuga puede:
 - Guardar el valor de `tiempo`
 - Guardar la `hora` actual
 - c) Permanece en espera hasta que se hace un segundo *click*. En ese momento, calcula la diferencia entre el `tiempo` o la `hora` almacenados y muestra si te has pasado, te has quedado corto o has acertado.

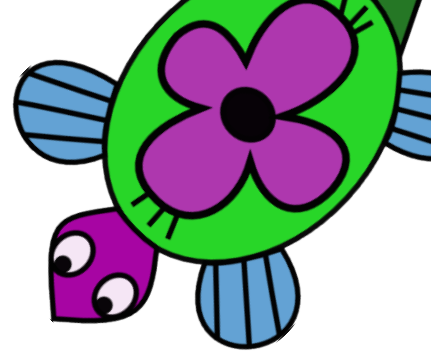
Puedes añadir cualquier efecto que deje claro cualquiera de los tres resultados (color de pantalla, de letra, música, ...)

2. Diseña un programa que se comporte como la alarma de un reloj:
 - a) Abra una ventana que pida la hora a la que debe sonar
 - b) Compruebe la hora recursivamente hasta que coincida con la introducida
 - c) Cuando coincidan la hora real y la introducida, que suene una melodía
3. Recupera el procedimiento `bisiesto?`, y haz que XLOGO te diga si el año actual es bisiesto o no.
4. Diseña un procedimiento que te pida la fecha de tu cumpleaños y determine cuántos días faltan para él.
Si hoy es tu cumpleaños, que suene el “Cumpleaños Feliz”.
5. Diseña un procedimiento que determine qué día de la semana es hoy.
(Dificultad alta) Amplíalo para que te pida una fecha cualquiera y diga qué día de la semana le corresponde. Para ello:
 - a) Elige una fecha de referencia de la que sepas qué día de la semana es.
 - b) Observa que los 365 días que tiene un año son:

$$365 = 364 + 1 = 7 \times 52 + 1$$

es decir, cada año normal desplaza un día el día de la semana de una fecha fija: por ejemplo, el 23 de Enero de 2006 fue Lunes, pero el mismo día de 2007 fue Martes.

- c) Ten en cuenta que los años bisiestos desplazan dos días en vez de uno
- d) Finalmente, haz un razonamiento análogo para cada mes, estudiando sus respectivos días



Capítulo 19

Utilización de la red con XLogo

19.1. 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.

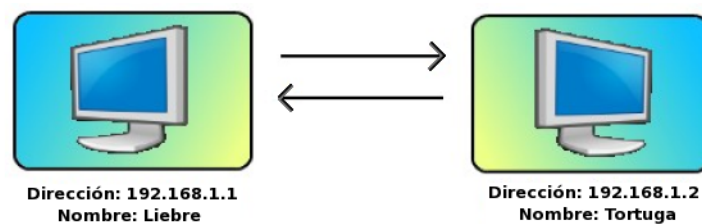


Figura: noción de red

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*.

19.2. 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
```

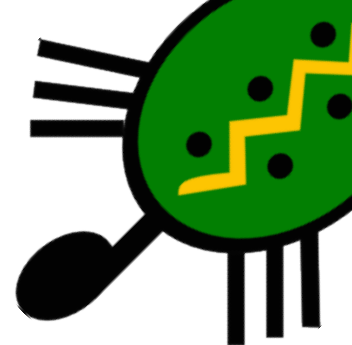
Con esta primitiva se puede establecer comunicación con un robot didáctico a través
de su interfaz de red. En este caso, la respuesta del robot puede ser diferente, y se
podrán decidir otras acciones en base al contenido de `:respuesta`.

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 20

Acerca de XLogo

20.1. 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.

20.2. Acerca de este documento

XLogo: Manual del Usuario

- Original en francés revisado por Loïc (30 de octubre de 2007).
- Traducido al español por Marcelo Duschkin y Álvaro Valdés (15 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 invaluables 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 21

Carnaval de Preguntas – Artimañas – Trucos que conocer

21.1. 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

Mira la solución en la sección del capítulo sobre música, 1.2.1

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 la JRE 1.4.1_07, propuesto en la web

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: `divisióñ`, `separacióñ` y `ponseparacióñ`.

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`.

21.2. ¿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



Capítulo 22

Listado alfabético de primitivas

A

Primitivas	Argumentos	Uso
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.
absoluto, abs	a: número	Devuelve el valor absoluto (distinto de cero) del número a
accionigu	a: palabra 1: lista	Asocia las acciones contenidas en lista al componente gráfico a
adios	no	Termina la ejecución de XLOGO
agrega	l1: lista n: número l2: palabra o lista	Dada la lista l1, inserta en la posición número n la palabra o lista l2.
agregalineafujo	n: número, l: lista	Escribe la línea de texto indicada en l al final del fichero indicado por el flujo n.
alto	no	Detiene la ejecución de un bucle o de un procedimiento.
amarillo	no	Establece este color [255 255 0] al elemento indicado.
animacion	no	Se accede al modo de animación.
antes?, anterior?	a b: palabras	Devuelve cierto si a está antes que b siguiendo el orden alfabético, falso si no.
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.

Primitivas	Argumentos	Uso
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.
avanza, av	n: número de pasos	Mueve la tortuga hacia adelante n pasos en la dirección que actualmente está mirando.
azar	a: número entero	Devuelve un número al azar mayor o igual que 0 y menor que a.
azul	no	Establece este color [0 0 255] al elemento indicado.
azuloscuro	no	Establece este color [0 0 128] al elemento indicado.

B

Primitivas	Argumentos	Uso
bajalapiz, bl	no	La tortuga dibujará una línea cuando se mueva.
bajalapiz?, bl?	cualquiera	Devuelve la palabra cierto si el lápiz está abajo, falso si no.
bajanariz	número	En 3D, baja el “morro” de la tortuga los grados indicados
balanceaderecha	número	Sólo en modo 3D, cambia el ángulo de balanceo de la tortuga. Sube el “ala” izquierda y baja la derecha n grados.
balanceaizquierda	número	Sólo en modo 3D, cambia el ángulo de balanceo de la tortuga. Sube el “ala” derecha y baja la izquierda n grados.
balanceo	no	Sólo en modo 3D, devuelve el ángulo de balanceo de la tortuga.
blanco	no	Establece este color [255 255 255] al elemento indicado.
borra, bo	a: palabra	Elimina el procedimiento cuyo nombre es a.
borracuadrícula	no	Quita la cuadrícula del Área de dibujo
borraejes	no	Quita los ejes del Área de dibujo
borrapantalla, bp	no	Vacía el área de dibujo, situando a la tortuga en el centro de la pantalla.
borrasecuencia, bos	no	Elimina de memoria la secuencia corriente.
borratexto, bt	no	Borra el Área de comandos , y el área del Historico de comandos .

Primitivas	Argumentos	Uso
borratodo,	no	Elimina todas las variables y procedimientos actuales.
borravARIABLE, bov	a: palabra	Elimina la variable a.
botonigu	a b: palabras	Crea un botón (componente gráfico) llamado a con la leyenda b.

C

Primitivas	Argumentos	Uso
cabeceaabajo	número	En 3D, baja el “morro” de la tortuga los grados indicados
cabeceaarriba	número	En 3D, sube el “morro” de la tortuga los grados indicados
cabeceo	no	Sólo en modo 3D, devuelve el ángulo del “morro” de la tortuga.
calidaddibujo, cdib	no	Devuelve la calidad del dibujo
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.
cambiasigno, cs	a: número	Devuelve el opuesto de a.
caracter, car	n: número	Devuelve el carácter unicode que corresponde al número n
carga	a: palabra	Abre y lee el archivo a.
cargaimagen, ci	a: palabra	Carga el archivo de imagen indicado por la palabra.
catalogo, cat	no	Lista el contenido del directorio actual. (Equivalente al comando <code>ls</code> de Linux, <code>dir</code> de DOS)
centro	no	Lleva la tortuga a la posición original, es decir coordenadas [0 0] con rumbo 0.
chattcp	a: palabra b: lista	Permite <i>chatear</i> con el PC palabra, enviando el mensaje contenido en lista
cierraflujo	n: número	Cierra el flujo n.
cierto	cualquiera	Devuelve "cierto"
circulo	n: radio	Dibuja una circunferencia de radio n alrededor de la tortuga
cociente	a b: números enteros	Devuelve el resultado de la división entera de a entre b
colorcuadricula	no	Devuelve el color actual de la cuadrícula.
colorejes	no	Devuelve el color actual de los ejes.
colorlapiz, cl	no	Devuelve el color actual del lápiz.

Primitivas	Argumentos	Uso
colorpapel	no	Devuelve el color actual del “papel” (fondo, área de dibujo).
colortexto	no	Devuelve el color de la tipografía usada por la primitiva escribe en el área del Histórico de comandos .
contador	no	En un bucle repite , determina el número de la iteración en curso.
cosa	a: palabra	Reenvía el valor de a. cosa "a y :a son notaciones equivalentes
coseno, cos	a: número en grados	Devuelve el coseno del número a.
cronometro, crono	n: número entero	Inicia un conteo descendiente de n segundos. Para saber que la cuenta ha finalizado, disponemos de la primitiva siguiente
cuadricula	a b: números	Dibuja una cuadrícula en el Área de dibujo de dimensiones a x b
cuadricula?	no	Devuelve cierto si la cuadrícula está activa, falso si no.
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.
cuentarepite	no	En un bucle repite , determina el número de la iteración en curso.
cursiva	no	Establece este efecto de fuente en la primitiva ponestilo
cyan	no	Establece este color [0 255 255] al elemento indicado.

D

Primitivas	Argumentos	Uso
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 .
definelinea	no	Sólo en modo 3D, inicia el modo de definición de líneas. Los movimientos posteriores de la tortuga se guardan para crear una línea.
definepoligono	no	Sólo en modo 3D, inicia el modo de definición de polígonos. Los movimientos posteriores de la tortuga se guardan para crear un polígono.

Primitivas	Argumentos	Uso
definepunto	no	Sólo en modo 3D, inicia el modo de definición de puntos. Los movimientos posteriores de la tortuga se guardan para crear un conjunto de puntos.
definetexto	no	Sólo en modo 3D, inicia el modo de definición de texto. Los textos a rotular se guardan para mostrarse al ejecutar <code>vista3D</code> .
detieneanimacion	no	Detiene el modo animación, retornando al modo <i>normal</i> .
detienecuadrícula	no	Quita la cuadrícula del Área de dibujo
detieneejes	no	Quita los ejes del Área de dibujo
detienetodo	no	Interrumpe total y definitivamente todos los procedimientos en ejecución
devuelve, dev	palabra	Permite salir del procedimiento “llevándose” un resultado
diferencia, -	a b: números a restar	Devuelve el resultado de restar b de a.
dibujaigu	a: palabra	Dibuja en la pantalla el componente gráfico a
directorio, dir	no	Da el directorio actual. Por defecto, es <code>/home/tu_nombre</code> en Linux, <code>C:\WINDOWS</code> en Windows.
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.
division, div, /	a b: números	Devuelve el resultado de dividir a por b

E

Primitivas	Argumentos	Uso
ed	no	Muestra el Editor de Procedimientos
ejecuta	a: lista	Ejecuta la lista de instrucciones contenida en la lista.
ejecutatcp	a: palabra b: lista	Ejecuta las instrucciones contenida en la lista en el PC indicado por a.
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.

Primitivas	Argumentos	Uso
ejex?	no	Devuelve cierto si está activo el eje de abscisas (eje X), falso si no.
ejey	a: número	Dibuja el eje de ordenadas (eje Y) de escala (separación entre marcas) a, con las etiquetas correspondientes.
ejey?	no	Devuelve cierto si está activo el eje de ordenadas (eje Y), falso si no.
elemento	a b: a n.º 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.
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.
eliminaigu	a: palabra	Elimina el componente gráfico a
eliminatortuga	a: número	Elimina la tortuga número a
empieزالinea	no	Sólo en modo 3D, inicia el modo de definición de líneas. Los movimientos posteriores de la tortuga se guardan para crear una línea.
empiezapoligono	no	Sólo en modo 3D, inicia el modo de definición de polígonos. Los movimientos posteriores de la tortuga se guardan para crear un polígono.
empiezapunto	no	Sólo en modo 3D, inicia el modo de definición de puntos. Los movimientos posteriores de la tortuga se guardan para crear un conjunto de puntos.
empiezatexto	no	Sólo en modo 3D, inicia el modo de definición de texto. Los textos a rotular se guardan para mostrarse al ejecutar vista3D .
encuentracolor, ec	a: lista	Devuelve el color del punto definido por las coordenadas.
entero?	a: número	Devuelve cierto si a es un número entero, falso si no.
enviatcp	a: palabra b: lista	Envía las instrucciones contenida en la lista al PC indicado por a.
escribe, es	a: número, palabra o lista	Muestra en el Histórico de Comandos el argumento indicado, a.
escribelineaflujo	n: número, l: lista	Escribe la línea de texto indicada en l al principio del fichero indicado por el flujo n.
escuchatcp	no	Pone al PC “en espera” de órdenes desde otro ordenador.

Primitivas	Argumentos	Uso
espera	n: número entero	Hace una pausa en el programa, la tortuga espera (n/60) segundos.
esquinasventana	no	Devuelve una lista con cuatro elementos, las coordenadas de la esquina superior izquierda y de la esquina inferior derecha.
estilo	no	Devuelve una lista que contiene todos los efectos de fuente utilizados por las primitivas escribe y tipea .
exp	a: número	Devuelve e ($e = 2,71828183\dots$) elevado a a

F

Primitivas	Argumentos	Uso
falso	cualquiera	Devuelve "falso"
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]
fin	no	Indica el final de un procedimiento
fincronometro?, fincrono?	no	Devuelve "cierto" si no hay ningún conteo activo. Devuelve "falso" si el conteo no ha terminado.
finflujo?	n: número	Devuelve cierto si se ha llegado al final del fichero, y falso en caso contrario.
finlinea	no	Sólo en modo 3D, termina el modo de definición de líneas.
finpoligono	no	Sólo en modo 3D, termina el modo de definición de polígonos.
finpunto	no	Sólo en modo 3D, termina el modo de definición de puntos.
fintexto	no	Sólo en modo 3D, termina el modo de definición de texto.
forma	no	Devuelve un número que representa la forma actual de la tortuga.
formalapiz, fl	no	Devuelve la forma del lápiz.
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).
fuelle	no	Devuelve el tamaño de la tipografía cuando se escribe en pantalla con la primitiva rotula .
fuentetexto, ftexto	no	Devuelve el tamaño de la tipografía usada por la primitiva escribe .

G

Primitivas	Argumentos	Uso
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.
goma, go	no	La tortuga borrará toda traza que encuentre.
gris	no	Establece este color [128 128 128] al elemento indicado.
grisclaro	no	Establece este color [192 192 192] al elemento indicado.
grosorlapiz, gl	no	Devuelve el grosor del lápiz.
guarda	a: palabra, l: lista	Guarda en el archivo a los procedimientos especificados en l, en el directorio actual.
guardatodo	a: palabra	Guarda en el archivo a todos los procedimientos definidos, en el directorio actual.

H

Primitivas	Argumentos	Uso
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.
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.
hazlocal	a b: a palabra, b cualquiera	Crea una nueva variable llamada a y le asigna el valor b.
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]

I

Primitivas	Argumentos	Uso
iguales?	a b	Devuelve cierto si a y b son iguales, falso si no.
indicesecuencia, indsec	no	Da la posición del puntero en la secuencia corriente.

Primitivas	Argumentos	Uso
inicializa, reponetodo	no	Reinicia la interfaz XLOGO a los valores por defecto: Color del lápiz: negro; color del papel: blanco; modo animación: falso; Tamaño de fuente para las zonas de texto y de dibujo: 12 puntos; Forma del lápiz: cuadrado; calidad del dibujo: normal; Número máximo de tortugas: 16
instrumento, instr	no	Da el número que corresponde al instrumento actualmente seleccionado.
invierte	a: lista	Invierte el orden de los elementos de la lista a
inviertelapiz, ila	no	Pone la tortuga en “modo inverso”, y lápiz abajo.

L

Primitivas	Argumentos	Uso
largoetiqueta	a: lista	Devuelve, en píxels, la longitud en pantalla de la lista.
leecar	no	Espera a que el usuario pulse una tecla y devuelve la tecla pulsada.
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
leelineaflujo	n: número	Abre el flujo cuyo identificador es n, y lee una línea del fichero
leelista	a: lista b: palabra	Muestra una ventana titulada a y recoge la respuesta del usuario en la variable b.
leeraton	no	Espera a que el usuario haga “algo” con el ratón, y devuelve la acción realizada.
leetecla	no	Espera a que el usuario pulse una tecla y devuelve la tecla pulsada.
leeteclado	a: lista b: palabra	Muestra una ventana titulada a y recoge la respuesta del usuario en la variable b.
limpia	no	Vacía el área de dibujo, dejando a la tortuga en el lugar donde estaba tras la ejecución anterior.
lista	a b	Devuelve una lista compuesta de a y b.
lista?	a	Devuelve cierto si a es una lista, falso si no.
listaflujos	l: lista	Carga una lista con los flujos abiertos indicando su identificador
listaprocs, imts	no	Enumera todos los procedimientos actualmente definidos.
listaspropiedades, listasprop	no	Enumera en una lista todas las listas de propiedades definidas.

Primitivas	Argumentos	Uso
listavars, imvars	no	Enumera todas las variables actualmente definidas.
local	a: palabra	Crea una variable llamada a . Atención: la variable no es inicializada. Para asignarle un valor, hay que usar haz .
log10, log	a: número	Devuelve el logaritmo decimal de a .
logneperiano, ln	a: número	Devuelve el logaritmo neperiano de a .

M

Primitivas	Argumentos	Uso
magenta	no	Establece este color [255 0 255] al elemento indicado.
marron	no	Establece este color [153 102 0] al elemento indicado.
maximastortugas, maxt	no	Devuelve el máximo número de tortugas
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.
menosultimo, 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.
mensaje, msj	a: lista	Muestra una caja de diálogo con el mensaje que está en la lista.
miembro	a b	Investiga a en b
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 .
mientras	a b: listas	La primera lista contiene un contador y sus valores inicial, final y el paso. La segunda es la serie de órdenes que deben ejecutarse el número de veces que determine la lista a .
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.
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
muestratortuga, mt	no	Hace que la tortuga se vea en pantalla.

N

Primitivas	Argumentos	Uso
naranja	no	Establece este color [255 200 0] al elemento indicado.
negrita	no	Establece este efecto de fuente en la primitiva <code>ponestilo</code>
negro	no	Establece este color [0 0 0] al elemento indicado.
ninguno	no	Elimina todos los efectos de fuente en la primitiva <code>ponestilo</code>
no	a: booleano	Devuelve la negación de a. Si a es cierto, devuelve falso. Si a es falso, devuelve cierto.
nombrefuente, nf	no	Devuelve una lista con dos elementos. El primero es un número correspondiente a la fuente utilizada para escribir en el Área de Dibujo con la primitiva <code>rotula</code> . El segundo elemento es una lista que contiene el nombre de la 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 <code>escribe</code> . El segundo elemento es una lista que contiene el nombre de la fuente.
numero?	a	Devuelve cierto si a es un número, falso si no.

O

Primitivas	Argumentos	Uso
o	a b: booleanos	Devuelve cierto si a ó b son ciertos, si no, devuelve falso
objeto	a: palabra	Reenvía el valor de a. objeto "a y :a son notaciones equivalentes
ocultatortuga, ot	no	Hace invisible a la tortuga.
orientacion,	no	Sólo en modo 3D, devuelve una lista con los tres valores: [balanceo cabeceo rumbo] de orientación de la tortuga.

P

Primitivas	Argumentos	Uso
palabra	a b: palabras	Concatena las dos palabras a y b.
palabra?	a	Devuelve cierto si a es una palabra, falso si no.

Primitivas	Argumentos	Uso
para	a: palabra, b c ...: variables	Inicia un procedimiento llamado a y cuyas variables (si las hay) son b, c, ...
paracada	a: lista o palabra b: lista	Ejecuta las órdenes contenidas en la lista b tantas veces como elementos o caracteres contiene a.
perspectiva	no	Inicia el modo de dibujo en 3 Dimensiones
pi	no	Devuelve el número π (3.141592653589793)
ponbalanceo	n: balanceo	Sólo en modo 3D, orienta el balanceo de la tortuga con el ángulo especificado. 0 corresponde a "timón" hacia arriba.
poncabeceo	n: cabeceo	Sólo en modo 3D, orienta el "morro" de la tortuga con el ángulo especificado. 0 corresponde a "morro" horizontal.
poncalidaddibujo, pcd	n: 0, 1 ó 2	Fija la calidad del dibujo: pcd 0: normal; pcd 1: alta; pcd 2: baja;
poncolorcuadrícula, pcc	a: número, primitiva o lista [r v a]	Cambia el color de la cuadrícula
poncolorejес, pce	a: número, primitiva o lista [r v a]	Cambia el color de los ejes cartesianos
poncolorlapiz, poncl	a: número, primitiva o lista [r v a]	Cambia el color del lápiz
poncolorpapel, poncp	a: número, primitiva o lista [r v a]	Cambia el color del papel (fondo)
poncolortexto, pctexto	a: número, primitiva o lista	Define el color de la tipografía del área del Histórico de comandos . Sólo disponible para ser usada por la primitiva escribe .
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.
ponestilo, pest	lista o palabra	Define los efectos de fuente para los comandos en el Histórico de comandos .

Primitivas	Argumentos	Uso
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).
ponformalapiz, pfl	n: 0 ó 1	Fija la forma del lápiz: pfl 0: cuadrada; pfl 1: ovalada.
ponfuente, pf	n: número	Cuando se escribe en pantalla con la primitiva rotula , es posible modificar el tamaño de la tipografía con ponfuente . Por defecto, el tamaño es 12.
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 .
pon grosor	n: número	Define el grosor del trazo del lápiz (en pixels). Por defecto es 1. La forma es cuadrada.
ponindicesecuencia, pindsec	a: número	Pone el puntero en la posición a de la secuencia corriente.
poninstrumento, pinstr	a: número	Queda seleccionado el instrumento número a . Puedes ver la lista de instrumentos disponibles en el menú Herramientas → Preferencias → Sonido .
ponlapiz, pla	no	Pone la tortuga en el modo normal de dibujo y lápiz abajo.
ponmaximastortugas, pmt	n: número	Fija el máximo número de tortugas
ponnombrefuente, pnf	n: número	Selecciona la tipografía número n para escribir en el Área de Dibujo con la primitiva rotula . Puedes encontrar la relación entre fuente y número en el menú Herramientas → Preferencias → Fuente .
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 .

Primitivas	Argumentos	Uso
ponorientacion,	[balanceo cabeceo rumbo]	Sólo en modo 3D, orienta la tortuga según las direcciones especificadas.
ponposicion, ponpos	[x y] o [x y z] en 3D: lista de números	Mueve la tortuga a las coordenadas especificadas por los números en la lista (x especifica el eje x, y el eje y, z el eje z – modo 3D).
ponprimero, pp	a b: a cualquiera, b lista	Inserta a en la primera posición de la lista b.
ponrumbo, ponr	n: rumbo	Orienta la tortuga en la dirección especificada. 0 corresponde a mirar hacia arriba verticalmente en 2D, hacia “nosotros” en 3D.
ponseparacion, 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.
pontamaño pantalla ptp	a: lista	Fija el tamaño de la pantalla.
pontortuga, ptortuga	a: número	La tortuga número a es ahora la tortuga activa. Por defecto, cuando XLOGO comienza, está activa la tortuga número 0.
ponultimo, pu	a b: a cualquiera, b lista	Inserta a en la última posición de la lista b
ponx	x: eje x	Mueve la tortuga hasta el punto de abscisa x, horizontalmente en modo 2D, con la perspectiva correspondiente en 3D
ponxy	x y: coordenada x, coordenada y	Idéntico a ponpos [x y]
ponxyz	x y z: coordenadas x, y, z	Sólo en modo 3D, idéntico a ponpos [x y z]
pony	y: eje y	Mueve la tortuga hasta el punto de ordenada y, verticalmente en modo 2D, con la perspectiva correspondiente en 3D
ponz	z: eje z	Sólo en modo 3D, mueve la tortuga verticalmente hasta el punto cuyo valor de z se indica
posicion, pos	no	Devuelve las coordenadas [X Y] en el plano, o [X Y Z] en el espacio, correspondientes a la posición actual de la tortuga.

Primitivas	Argumentos	Uso
posicionigu,	a: palabra b: lista	Determina la posición en pantalla del componente gráfico a: las coordenadas b.
pos	no	Devuelve las coordenadas X e Y correspondientes a la posición actual de la tortuga.
posraton,	no	Devuelve las coordenadas X e Y correspondientes a la posición actual del ratón.
potencia	a b: números	Devuelve a elevado a la potencia b
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.
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.
producto, *	a b: números	Devuelve el resultado de multiplicar a por b
punto	a: lista	El punto definido por las coordenadas de la lista se resaltará con el color del lápiz.

Q

Primitivas	Argumentos	Uso
quita	a b: a cualquiera, b lista	Elimina el elemento a de la lista b, si aparece dentro

R

Primitivas	Argumentos	Uso
raizcuadrada, rc	a: número	Devuelve la raíz cuadrada de a.
raton?,	no	Devuelve cierto o falso según se mueva o no el ratón.
redondea	a: número	Devuelve el entero más próximo al número a
reemplaza	l1: lista n: número l2: palabra o lista	Dada la lista l1, reemplaza el elemento n por la palabra o lista l2.
refresca	no	En modo de animación, ejecuta las órdenes y actualiza la imagen
repite	a: número entero b: lista	Ejecuta a veces las órdenes contenidas en la lista b

Primitivas	Argumentos	Uso
repitepara	a: lista b: lista	Ejecuta las órdenes contenidas en la lista b según las condiciones descritas en a.
repitesiempre	a: lista	Ejecuta las órdenes contenidas en la lista a hasta que se haga click en el botón <i>alto</i> o alguna orden de a se lo indique.
resto	a b: números enteros	Devuelve el resto de la división de a por b
retrocede, re	n: número de pasos	Mueve la tortuga hacia atrás n pasos en la dirección que actualmente está mirando.
rojo	no	Establece este color [255 0 0] al elemento indicado.
rojooscuro	no	Establece este color [128 0 0] al elemento indicado.
rosa	no	Establece este color [255 175 175] al elemento indicado.
rotula	a: palabra o lista	Dibuja la palabra o lista especificada, en la posición actual, y en la dirección que está mirando.
rumbo	no	Devuelve el rumbo o el ángulo de la tortuga.

S

Primitivas	Argumentos	Uso
secuencia, sec	a: lista	Carga en memoria la secuencia incluida en la lista. Siguiendo a esta tabla, se indica cómo escribir una secuencia de notas musicales.
seno, sen	a: número (grados)	Devuelve el seno del número a.
separacion	no	Devuelve el valor de la proporción de pantalla ocupada por el Área de Dibujo y el Histórico de Comandos .
si	a: expr. lógica b,c: listas	Ejecuta las órdenes guardadas en las listas :b y :c según la expresión lógica :a sea cierto o falso
sisino	a: expr lógica b,c: variables	Ejecuta las órdenes guardadas en las variables :b y :c según la expresión lógica :a sea cierto o falso
subelapiz, sl	no	La tortuga no dibujará cuando se mueva.
subenariz	número	En 3D, sube el “morro” de la tortuga los grados indicados
subindice	no	Establece este efecto de fuente en la primitiva ponestilo

Primitivas	Argumentos	Uso
subrayado	no	Establece este efecto de fuente en la primitiva <code>ponestilo</code>
superindice	no	Establece este efecto de fuente en la primitiva <code>ponestilo</code>
suma, +	a b: números a sumar	Devuelve el resultado de sumar a y b.

T

Primitivas	Argumentos	Uso
tachado	no	Establece este efecto de fuente en la primitiva <code>ponestilo</code>
tamaño pantalla, <code>tpant</code>	no	Devuelve una lista que contiene el tamaño de la pantalla
tamaño ventana, <code>tv</code>	no	Devuelve una lista con cuatro elementos, las coordenadas de la esquina superior izquierda y de la esquina inferior derecha.
tangente, <code>tan</code>	a: número (grados)	Devuelve la tangente del número a.
tecla?	no	Devuelve <code>cierto</code> o <code>falso</code> según se haya pulsado o no alguna tecla desde el inicio del programa.
tiempo	no	Devuelve el tiempo, en segundos, transcurrido desde el inicio de XLOGO.
tipea	a: cualquiera	Idéntico a <code>escribe</code> , pero el cursor queda en la línea donde se mostró el contenido del argumento.
tocamusica	no	Toca la secuencia de notas en memoria.
tortuga	no	Da el número de la tortuga activa.
tortugas	no	Da una lista que contiene todos los números de tortuga actualmente en pantalla.
trunca	a: número	Devuelve el entero inmediatamente anterior al número a

U

Primitivas	Argumentos	Uso
ultimo	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.
unicode	a: palabra	Devuelve el número unicode que corresponde al carácter a.

V

Primitivas	Argumentos	Uso
------------	------------	-----

Primitivas	Argumentos	Uso
vacio?	a	Devuelve cierto si a es una lista vacía o una palabra vacía, falso si no.
variable?, var?	a: palabra	Devuelve cierto si la palabra es una variable ya definida, falso si no.
verde	no	Establece este color [0 255 0] al elemento indicado.
verdeoscuro	no	Establece este color [0 128 0] al elemento indicado.
violeta	no	Establece este color [128 0 255] al elemento indicado.
visible?	cualquiera	Devuelve la palabra cierto si la tortuga está visible, falso si no.
vista3D, vistapoligono	no	Inicia el visor 3D, todos los objetos guardados se dibujan en una nueva ventana.

Y

Primitivas	Argumentos	Uso
y	a b: booleanos	Devuelve cierto si a y b son ciertos, si no, devuelve falso

Z

Primitivas	Argumentos	Uso
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.

Otras

Primitivas	Uso
#	Indica que el texto siguiente es un comentario y no debe ser interpretado
&	Equivalente a y en operaciones lógicas.
	Equivalente a o en operaciones lógicas.
<	Menor que: Compara valores numéricos en operaciones lógicas
<=	Menor o igual que: Compara valores numéricos en operaciones lógicas
=	Equivalente a iguales? en operaciones lógicas.
>	Mayor que: Compara valores numéricos en operaciones lógicas
>=	Mayor o igual que: Compara valores numéricos en operaciones lógicas
\\	Permite obtener el carácter <i>barra invertida</i> (\) en la línea de comandos.
\n	Permite generar saltos de línea al escribir en el Histórico de comandos
_	Permite generar espacios en blanco al escribir en el Histórico de comandos

Bibliografía

- [1] Manual de referencia de XLOGO. Autor: **Loïc Le Coq**. Traducción: **Marcelo Duschkin** y **Álvaro Valdés**.

<http://xlogo.tuxfamily.org/sp/downloads-sp/manual-sp.pdf>

- [2] Diez lecciones para descubrir el lenguaje XLOGO. Autor: **Loïc Le Coq**. Traducción: **Álvaro Valdés**.

<http://xlogo.tuxfamily.org/sp/downloads-sp/tutorial.pdf>

- [3] Wikipedia

<http://es.wikipedia.org>

- [4] Paraíso de LOGO. Autor: **Daniel Ajoy**.

<http://neoparaiso.com/logo/>

- [5] Cuaderno de ejercicios de LOGO. Autor: **Pedro José Fernández**.

<http://neoparaiso.com/logo/documentos/ejercicios-logo.pdf>

- [6] Informática y Matemática. Asignatura optativa (y libre configuración). Autores: **Eugenio Roanes Macías** y **Eugenio Roanes Lozano**

http://www.ucm.es/info/secdealg/ApuntesLogo/INF_MATN_2006.PDF

- [7] Introducción a la Programación con PYTHON. Autor: **Bartolomé Sintés Marco**

<http://www.mclibre.org/consultar/python/>

Apéndice A

Iniciando XLogo desde la línea de comandos

La sintaxis del comando que ejecuta XLOGO es:

```
java-jar xlogo.jar [-a] [-lang en] [archivo1.lgo archivo2.lgo ...]
```

Las opciones disponibles son:

- El atributo `-lang`: este atributo especifica el idioma con que ejecutar XLOGO.

El parámetro indicado con esta opción se superpone al existente en el fichero `.xlogo` de configuración personal. En la siguiente tabla se muestran los parámetros asociados a todos los idiomas disponibles:

Francés	Inglés	Español	Alemán	Portugués
fr	en	es	de	pt

Árabe	Esperanto	Gallego	Asturiano	
eo	ar	gl	as	

- Atributo `-a`: este atributo determina que se ejecute el **Comando de Inicio** (ver sección 1.5) que figura en el archivo cargado sin necesidad de pulsar el botón de inicio (sección 1.4) al iniciarse XLOGO.
- Atributo `-memory`: este atributo cambia el tamaño de memoria reservado para XLOGO
- `[archivo1.lgo archivo2.lgo ...]`: estos archivos de extensión `.lgo` se cargan al iniciar XLOGO.

Los archivos pueden ser locales (estar en el disco duro del ordenador) o remotos (en la *web*). Por lo tanto, se puede especificar una dirección local o una dirección web.

Veamos ejemplos:

- `java -jar xlogo.jar -lang es prog.lgo`

los archivos `xlogo.jar` y `prog.lgo` están en el directorio actual. Este comando ejecuta XLOGO cambiando el idioma a “español” y a continuación carga el fichero `prog.lgo`. Por tanto, es necesario que este archivo sea un programa LOGO escrito en Español.

- `java -jar xlogo.jar -a -lang en http://xlogo.tuxfamily.org/prog.lgo:`

Este comando ejecuta XLOGO en Inglés, estando `xlogo.jar` en el directorio de trabajo, y se carga el fichero `http://xlogo.tuxfamily.org/prog.lgo` desde la *web*.

Por último, el Comando de Inicio de este fichero se ejecuta en el arranque.

Apéndice B

Ejecutando XLogo desde la *web*

B.1. El problema

Supongamos que usted es administrador/a de un sitio web. En este sitio, habla de XLOGO y quiere ofrecer algunos de los programas que ha creado.

Podría distribuir el fichero LOGO en formato `.lgo` y esperar que los visitantes descarguen primero XLOGO, después el programa y, por último, lo carguen para ver qué hace.

La alternativa es que el usuario pueda ejecutar XLOGO *en línea*, y probar su programa sin apenas esfuerzo.

Para lanzar XLOGO desde una página *web*, utilizaremos la tecnología JAVA WEB START. Con ello, sólo necesita poner en su web un enlace hacia un archivo con extensión `.jnlp`, que iniciará la ejecución de XLOGO.

B.2. Cómo crear un fichero `.jnlp`

Veamos cómo hacerlo con un archivo de ejemplo. De hecho, el siguiente ejemplo es el que se usa en la sección “**Ejemplos**” de la página de XLOGO en francés.

Este archivo carga el programa que dibuja un dado en la sección 3D. Veamos primero el archivo, y veremos después las explicaciones sobre su contenido.

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.5+" codebase="http://downloads.tuxfamily.org/xlogo/common/webstart">
<information>
  <title>XLogo</title>
  <vendor>xlogo.tuxfamily.org</vendor>
  <homepage href="http://xlogo.tuxfamily.org"/>
  <description>Logo Programming Language</description>
```



```

    <offline-allowed/>
</information>

<security>
  <all-permissions/>
</security>

<resources>
  <j2se version="1.4+"/>
  <jar href="xlogo.jar"/>
</resources>

<application-desc main-class="Lanceur">
  <argument>-lang</argument>
  <argument>fr</argument>
  <argument>-a</argument>
  <argument>http://xlogo.tuxfamily.org/fr/html/examples-fr/3d/de.lgo</argument>
</application-desc>
</jnlp>

```

Este archivo está escrito en formato XML, y las cuatro líneas más importantes son:

```

<argument>-lang</argument>
<argument>fr</argument>
<argument>-a</argument>
<argument>http://xlogo.tuxfamily.org/fr/html/examples-fr/3d/de.lgo</argument>

```

Esas líneas especifican los parámetros de inicio de XLOGO:

- Las líneas 1 y 2 fuerzan a XLOGO a ejecutarse en francés.
- La última línea indica la ruta del fichero a cargar.
- La línea 3 indica que se ejecute el Comando de Inicio de este programa al iniciarse XLOGO.

Una última sugerencia (petición): Dado que el servidor de Tuxfamily no puede aceptar “infinitas” conexiones, le pedimos que tenga una copia del archivo `xlogo.jar` en su web. Para ello, cambie la dirección en la segunda línea del archivo `.jnlp`, donde dice `codebase=` por:

```

<jnlp spec="1.5+" codebase="http://mi.direccion.web/directorio/XLogo">

```

Apéndice C

Programación Estructurada y Diseño Modular

A finales de los años sesenta surgió una nueva forma de programar que no solamente daba lugar a programas fiables y eficientes, sino que además estaban escritos de manera que facilitaba su comprensión posterior.

C.0.1. Programación Estructurada

El **Teorema de Dijkstra**, demostrado por Edsger Dijkstra en los años sesenta, demuestra que todo programa puede escribirse utilizando únicamente las tres instrucciones de control siguientes:

- Secuencia
- Instrucción condicional.
- Iteración (bucle de instrucciones) con condición inicial

Sóloamente con estas tres estructuras se pueden escribir programas, si bien ya hemos visto que existen otras estructuras de control: los procedimientos.

C.0.2. Diseño Modular

El uso de procedimientos y subprocedimientos permite dividir un problema grande en *subproblemas* más pequeños. Si el objetivo es elaborar un programa para resolver dicho problema grande, cada subproblema (menos complejo) podrá ser resuelto por un *módulo* (subprocedimiento) relativamente fácil de implementar

¿Por qué descomponer un problema en partes? Experimentalmente está comprobado que:

- Un problema complejo cuesta más de resolver que otro más sencillo

- La complejidad de un problema global es mayor que la suma de las complejidades de cada una de sus partes por separado.

La combinación de las instrucciones de control con los procedimientos permite diseñar programas de forma rápida, sencilla y eficiente.

Índice alfabético

- π , 27, 174
- (, 10
-), 10
- *, 45, 177
- +, 45, 179
- , 45, 167
- /, 45, 167
- <, 61, 180
- <=, 61, 180
- =, 63, 180
- >, 61, 180
- >=, 61, 180
- ?, 63
- [, 10
- #, 10, 35, 180
- &, 60, 180
- ", 10
- \\, 9, 180
- \n, 9, 180
- \u, 9, 180
-], 10
- 3d, 135

- abreflujo, 130, 163
- Abrir, 11
- abs, 51, 163
- absoluto, 51, 163
- accionigu, 99, 163
- Acentuación y tildes, 10
- Acerca de ..., 18
- acos, 52, 164
- adios, 12, 163
- agrega, 66, 163
- agregalineafujo, 130, 163
- Alto, 6

- alto, 36, 78, 163
- amarillo, 106, 163
- Animación, 124
- animacion, 124, 163
- anterior?, 63, 163
- antes?, 63, 163
- arco, 53, 138, 163
- arcocoseno, 52, 164
- arcoseno, 52, 164
- arcotangente, 52, 164
- Área de Dibujo, 5
- Argumentos, 27
- Argumentos Opcionales, 28
- asen, 52, 164
- Aspecto, 14
- atan, 52, 164
- av, 28, 137, 164
- avanza, 28, 137, 164
- Ayuda, 17
- azar, 51, 164
- azul, 106, 164
- azuloscuro, 106, 164

- backslash, 9
- bajalapiz, 30, 164
- bajalapiz?, 63, 164
- bajanariz, 136, 164
- balanceaderecha, 137, 164
- balanceaizquierda, 137, 164
- balanceo, 137, 139, 164
- Barra invertida, 9
- bd, 137
- bi, 137
- bl, 30, 164
- bl?, 63, 164

- blanco, 106, 164
- bn, 136
- bo, 164
- Booleano, 173
- Booleanos, 63
- boprop, 72
- borra, 8, 43, 164
- borracuadrícula, 15, 164
- borraejcs, 15, 164
- borrapantalla, 30, 137, 164
- borrapropiedad, 72
- Borrar procedimientos, 8, 14
- borrasecuencia, 148, 164
- borratexto, 89, 164
- borratodo, 8, 43, 165
- borravariablc, 37, 165
- bos, 148, 164
- botonigu, 98, 99, 165
- bov, 37, 165
- bp, 30, 137, 164
- bt, 89, 164
- Bucles, 73

- cabeceabajo, 136, 165
- cabeceaarriba, 136, 165
- cabeceo, 139, 165
- cabeceo?, 137
- Calidad del dibujo, 16
- calidaddibujo, 110, 165
- cambiadirectorio, 127, 165
- cambiasigno, 51, 165
- car, 90, 165
- caracter, 90, 165
- carga, 129, 165
- cargaimagen, 115, 165
- cat, 127, 165
- catalogo, 127, 165
- cd, 127, 165
- cdib, 110, 165
- centro, 28, 57, 138, 165
- chattcp, 158, 165
- ci, 115, 165
- cierraflujo, 130, 165

- cierto, 63, 165
- circulo, 49, 138, 165
- cl, 165
- cociente, 45, 165
- Color de lápiz preasignado, 15
- Color de papel preasignado, 15
- colorcuadrícula, 165
- colorejes, 165
- Colores (ejemplo), 114
- colorlapiz, 105, 165
- colorpapel, 105, 166
- colortexto, 89, 166
- Comando de Inicio, 5
- Comentarios, 35
- Condiciona, 59
- contador, 74, 166
- Convenciones, 9
- Coordenadas, 56
- Copiar, 5, 13
- Cortar, 5, 13
- cos, 52, 166
- cosa, 37, 166
- Coseno, 166
- coseno, 52, 166
- crono, 151, 166
- cronometro, 151, 166
- cs, 51, 165
- Cuadrícula, 15
- cuadrícula, 15, 55, 166
- cuadrícula?, 63, 166
- cuenta, 65, 166
- cuentarepite, 74, 166
- cursiva, 90, 166
- cyan, 106, 166

- def, 166
- define, 43, 166
- definelinea, 141, 166
- definepoligono, 141, 166
- definepunto, 141, 167
- definetexto, 141, 167
- Definir archivos de inicio, 13
- deflinea, 141

- defpoli, 141
- defpto, 141
- deftxt, 141
- Desinstalar, 9
- detienecuadrícula, 55, 167
- detieneejes, 55, 167
- detienetodo, 79, 167
- dev, 79, 167
- devuelve, 79, 85, 86, 167
- dibujaigu, 99, 167
- diferencia, 45, 167
- dir, 127, 167
- directorio, 127, 167
- distancia, 57
- distancia, 138, 167
- div, 167
- division, 45, 167

- ec, 105, 168
- ed, 6, 167
- Edición, 13
- edita, 6
- Editar, 6
- editatodo, 6
- Editor de Procedimientos, 6
- edtodo, 6
- Efectos de luz y niebla, 144
- ejecuta, 44, 167
- ejecutatcp, 158, 167
- ejes, 15, 55, 167
- Ejes cartesianos, 15
- ejex, 15, 55, 167
- ejex?, 63, 168
- ejey, 15, 55, 168
- ejey?, 63, 168
- Elegir color del lápiz, 13
- Elegir color del papel, 13
- elemento, 65, 168
- elige, 65, 168
- eliminaigu, 99, 168
- eliminatortuga, 119, 168
- empieزالinea, 141, 168
- empiezapoligono, 141, 168

- empiezapunto, 141, 168
- empiezatexto, 141, 168
- encuentracolor, 105, 168
- entero?, 63, 168
- enviatcp, 158, 168
- es, 168
- escribe, 27, 87, 168
- escribelineaflujo, 130, 168
- escuchatcp, 158, 168
- Espacios, 9
- espera, 151, 169
- esquinasventana, 112, 169
- estilo, 89, 169
- Estilo de programación, 34
- exp, 52, 169

- Factorial, 82
- falso, 63, 169
- fecha, 151, 169
- Figura de la tortuga, 14
- fin, 169
- fincrono?, 151, 169
- fincronometro?, 151, 169
- finflujo?, 130, 169
- finlinea, 141, 169
- finpoli, 141
- finpoligono, 141, 169
- finpto, 141
- finpunto, 141, 169
- fintexto, 141, 169
- fintxt, 141
- Flujo de control, 59
- Foco, 145
- forma, 119, 169
- Forma del lápiz, 16
- Forma general de una primitiva, 48
- fr, 169
- Frase, 38
- frase, 47, 67, 169
- ftexto, 89, 169
- Fuente, 16
- fuentes, 92, 169
- fuentetexto, 89, 169

- Funciones trigonométricas, 52, 178
- gd, 28, 137, 170
- Gestión de tiempos, 151
- gi, 28, 137, 170
- giraderecha, 28, 137, 170
- giraizquierda, 28, 137, 170
- gl, 103, 170
- go, 30, 105, 170
- goma, 30, 105, 170
- gris, 106, 170
- grisclaro, 106, 170
- grosorlapiz, 103, 170
- guarda, 129, 170
- Guardar, 11
- Guardar como ..., 11
- Guardar en formato RTF, 12
- Guardar imagen como..., 12
- guardatodo, 129, 170
- hacia, 57
- hacia, 138, 170
- haz, 37, 41, 170
- hazlocal, 41, 170
- Histórico de Comandos, 5
- hora, 151, 170
- Idioma, 14
- iguales?, 63, 170
- ila, 105, 171
- Imprimir imagen, 12
- imts, 44, 171
- imvars, 37, 172
- Indentar, 35
- indicesecuencia, 148, 170
- indsec, 148, 170
- inicializa, 9, 171
- instr, 148, 171
- instrumento, 148, 171
- invierte, 66, 171
- inviertelapiz, 105, 171
- .jpg, 12, 115
- Línea de Comando, 5
- largoetiqueta, 87, 138, 171
- leecar, 92, 171
- leecarflujo, 130, 171
- leelineaflujo, 130, 171
- leelista, 67, 92, 171
- leeprop, 72
- leepropiedad, 72
- leeraton, 94, 171
- leetecla, 92, 171
- leeteclado, 93, 171
- .lgo, 13
- Licencia GPL, 17
- limpia, 30, 171
- Lista, 28, 38
- lista, 47, 67, 68, 171
- lista?, 63, 171
- listaflujos, 130, 171
- listaprocs, 44, 171
- listaprop, 72
- listapropiedades, 72
- Listas de Propiedades, 72
- listasprop, 72, 171
- listaspropiedades, 72, 171
- listavars, 37, 172
- ln, 52, 172
- local, 41, 172
- log, 52, 172
- log10, 52, 172
- Logaritmos, 52, 172
- logneperiano, 52, 172
- Luz Ambiental, 145
- Luz Direccional, 145
- magenta, 106, 172
- Marco de adorno, 15
- marron, 106, 172
- maximastortugas, 119, 172
- maxt, 119, 172
- Mayúsculas y minúsculas, 10
- Memoria destinada a XLOGO, 16
- menosprimero, 66, 172
- menosultimo, 66, 172
- mensaje, 87, 172

- MIDI, 16, 147
 miembro, 65, 172
 miembro?, 63, 172
 mientras, 75, 78, 172
 modojaula, 110, 135, 136, 172
 modoventana, 110, 135, 136, 172
 modovuelta, 110, 135, 136, 172
 mp, 66, 172
 msj, 87
 msj, 172
 mt, 30, 103, 172
 mu, 66, 172
 muestratortuga, 30, 103, 120, 172
 Multitortuga, 119

 Número máximo de tortugas, 15
 Números, 27, 37
 Números aleatorios, 51
 naranja, 106, 173
 negrita, 90, 173
 negro, 106, 173
 nf, 92
 nft, 89, 173
 nf, 173
 Niebla densa, 146
 Niebla lineal, 145
 ninguno, 90
 ninguno, 173
 no, 60, 173
 nombrefuente, 92, 173
 nombrefuentetexto, 89, 173
 Nuevo, 11
 numero?, 63, 173

 o, 60, 173
 objeto, 37, 173
 ocultatortuga, 30, 103, 173
 Operaciones Binarias, 45
 Operaciones Lógicas, 60
 Operaciones Unitarias, 51
 Opuesto, 165
 orientacion, 139, 173
 ot, 30, 103, 173

 palabra, 67, 173
 palabra?, 63, 173
 Palabras, 27, 38
 Paréntesis, 54
 para, 174
 paracada, 174
 pcc, 174
 pcc, 55
 pcd, 110, 174
 pce, 55, 174
 pctexto, 89, 174
 Pegar, 5, 13
 perspectiva, 135, 136, 174
 pest, 89, 174
 pf, 92, 175
 pfl, 103, 175
 pforma, 119, 175
 pft, 89, 175
 pi, 27, 174
 pindsec, 148, 175
 pinstr, 148, 175
 pla, 175
 pmt, 119, 175
 pnf, 92, 175
 pnft, 89, 175
 .png, 12, 115
 ponbalanceo, 139, 174
 poncabeceo, 139, 174
 poncalidaddibujo, 110, 174
 poncl, 30, 105, 174
 poncolorcuadrícula, 55, 174
 poncolorejes, 15, 55, 174
 poncolorlapiz, 13, 30, 105, 174
 poncolorpapel, 13, 105, 174
 poncolortexto, 89, 174
 poncp, 105, 174
 pondir, 127, 174
 pondirectorio, 127, 174
 ponestilo, 89, 174
 ponforma, 14, 119, 175
 ponformalapiz, 103, 175
 ponfuente, 92, 175
 ponfuentetexto, 89, 175

- pongrosor, 30, 103, 175
 ponindicesecuencia, 148, 175
 poninstrumento, 16, 148, 175
 ponlapiz, 175
 ponmaximastortugas, 119, 175
 ponnombrefuente, 92, 175
 ponnombrefuentetexto, 89, 175
 ponorientacion, 139, 176
 ponpos, 56, 138, 176
 ponposicion, 56, 138, 176
 ponprimero, 66, 176
 ponprop, 72
 ponpropiedad, 72
 ponr, 176
 ponrumbo, 57, 138, 176
 ponsep, 112, 176
 ponseparacion, 112, 176
 pontamañopantalla, 112, 176
 pontortuga, 119, 176
 ponultimo, 66, 176
 ponx, 56, 138, 176
 ponxy, 56, 176
 ponxyz, 139, 176
 pony, 56, 138, 176
 ponz, 139, 176
 Portapapeles, 12
 pos, 56, 138, 176, 177
 posicion, 56, 138, 176
 posicionigu, 99, 177
 posraton, 94, 177
 potencia, 45, 177
 pp, 66, 176
 pr, 65, 177
 Preferencias, 14
 prim?, 63, 177
 primero, 65, 177
 primitiva?, 63, 177
 Primitivas, 27
 Primitivas booleanas, 63
 Primitivas personalizadas, 13
 Prioridad de las operaciones, 54
 proc?, 63, 177
 procedimiento?, 63, 177
 Procedimientos, 33
 Procedimientos avanzados, 85
 Procedimientos con variables, 39
 producto, 45, 177
 ptortuga, 119, 176
 pu, 66, 176
 punto, 56, 138, 177
 Punto de Luz, 145

 quita, 66, 177

 raizcuadrada, 51, 177
 Ratón, 94
 raton?, 94, 177
 rc, 51, 177
 re, 28, 137, 178
 Recursividad, 82
 redondea, 51, 177
 reemplaza, 66, 177
 refresca, 177
 refrescar, 124
 rellena, 30, 107
 rellenazona, 30, 107
 repite, 30, 73, 78, 177
 repitepara, 74, 178
 repitesiempre, 178
 reponetodo, 9, 171
 Resaltado, 17
 resto, 45, 178
 retrocede, 28, 137, 178
 Reutilización de variables, 48
 rojo, 106, 178
 rojooscuro, 106, 178
 rosa, 106, 178
 rotula, 87, 138, 178
 RTF, 12
 Rumbo, 57
 rumbo, 57
 rumbo, 138, 178
 Ruptura de secuencia, 78

 Salir, 8, 12
 Saltos de línea, 9
 sec, 148, 178

- secuencia, 148, 178
- Seleccionar todo, 13
- sen, 52, 178
- Seno, 178
- seno, 52, 178
- separacion, 112, 178
- si, 59, 178
- sisino, 60, 178
- s1, 30, 178
- sn, 136
- Sonido, 16
- subelapiz, 30, 178
- subenariz, 136, 178
- subindice, 90, 178
- Subprocedimientos, 35
- subrayado, 90, 179
- suma, 45, 179
- superindice, 90, 179

- tachado, 90, 179
- Tamaño de la ventana, 16
- Tamaño máximo del lápiz, 16
- tamaño pantalla, 112, 179
- tamaño ventana, 112, 179
- tan, 52, 179
- Tangente, 179
- tangente, 52, 179
- tecla?, 92, 179
- Teclado, 92
- tiempo, 151, 179
- tipea, 46, 87, 179
- tocamusica, 148, 179
- Tocar música, 147
- tortuga, 119, 179
- tortugas, 119, 179
- tpant, 112, 179
- Traducción de la Licencia, 17
- Traducir procedimientos, 14
- Traducir XLogo, 17
- trazado, 86
- Tres Dimensiones, 135
- trunca, 51, 179
- Truncar un número, 51

- tv, 112, 179

- ultimo, 65, 179
- unicode, 90, 179

- vacio?, 63, 180
- Valor absoluto, 163
- Valor absoluto de un número, 51
- var?, 63, 180
- variable?, 63, 180
- Variables, 37
- Variables booleanas, 63
- Variables globales, 41
- Variables locales, 41
- Variables opcionales, 85
- Velocidad de la tortuga, 14
- verde, 106, 180
- verdeoscuro, 106, 180
- violeta, 106, 180
- visible?, 63, 180
- vista3D, 180
- vista3d, 141
- vistapoligono, 141, 180

- y, 60, 180

- Zoom, 5
- zoom, 110, 180