I.E.S. Pérez de Ayala

P.R.O.A. – Curso 2007–08

Introducción

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. 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 soporta siete idiomas (Francés, Inglés, Español, Portugués Galés, Árabe y Esperanto) 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.

A lo largo del manual se irán planteando ejercicios y resolviendo ejemplos. Hemos elegido **no utilizar** tildes ni *eñes* en los mismos, aunque XLOGO las admite y trabaja perfectamente con ellas, debido a que en la mayoría de los lenguajes de programación no las aceptan.

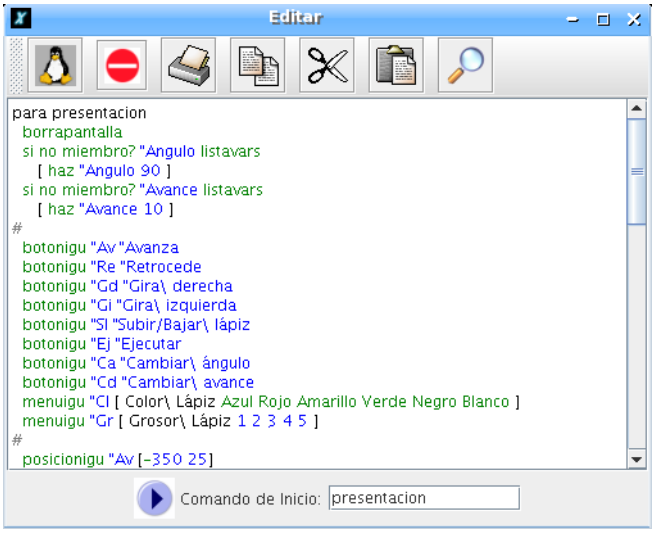
Índice

1. Presentación de la tortuga

¿Cómo presentamos a nuestra tortuga? 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):



escribamos en la Línea de `Comando de inicio` la palabra:

`presentacion`

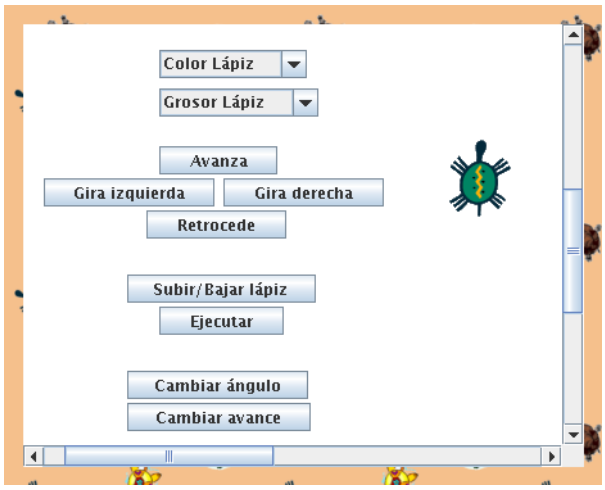
(no te preocupes de momento por todo lo que hay escrito, en breve serás capaz de diseñarlo tú mismo/a). Finalmente, hacemos *click* en el pingüino y ... ¿no pasa nada?

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:

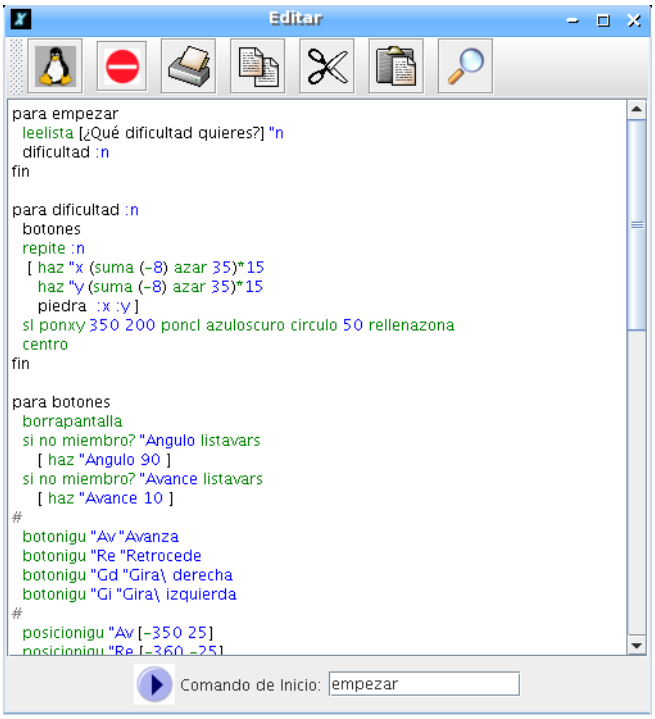


Haz todas las pruebas que necesites para "entender" cómo se mueve la tortuga, cómo mide los ángulos, las distancias, ... Fíjate, especialmente, en cómo realiza los giros y procura entender el punto de vista de la tortuga.

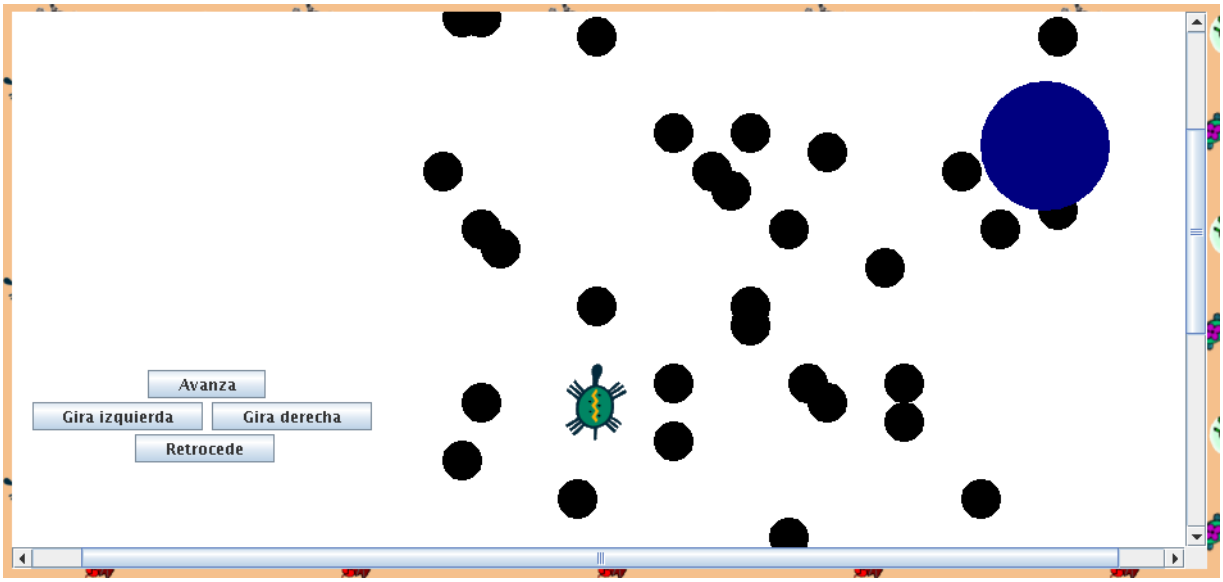
¿Qué podemos hacer con una pantalla como la de arriba? Descarguemos el fichero

`http://xlogo.tuxfamily.org/sp/curso/juego.lgo`

abrámoslo y cambiemos el Comando de Inicio por `empezar`:



Guardamos (hacemos *click* en el pingüino) y después en el de Comando de inicio. Después de contestar a la pregunta (un valor de 10 ó 15 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 sólamente los botones de desplazamiento. Los círculos negros representan piedras y están colocadas aleatoriamente:

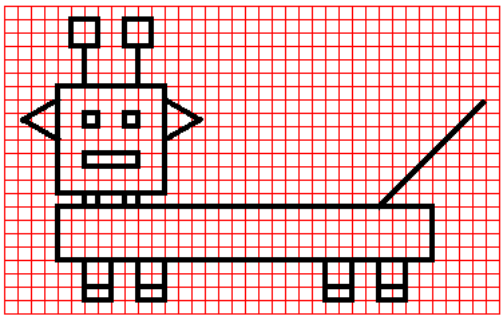
- Si "choca" con una piedra, nos aparece un aviso y vuelve al punto de partida
- Cuando llegue al lago, nos felicitará y sonará una canción

	**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, . . .**

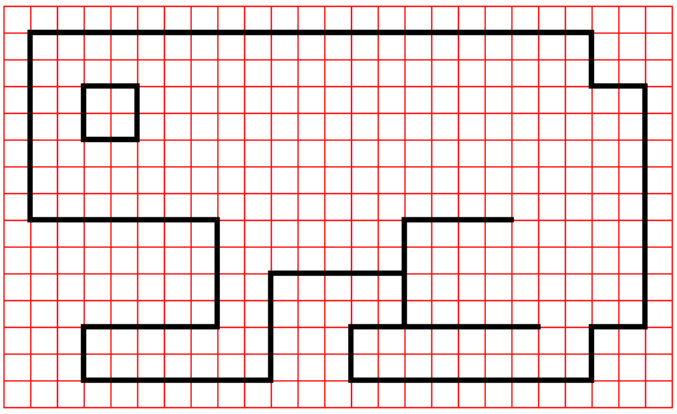
Ejercicios

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

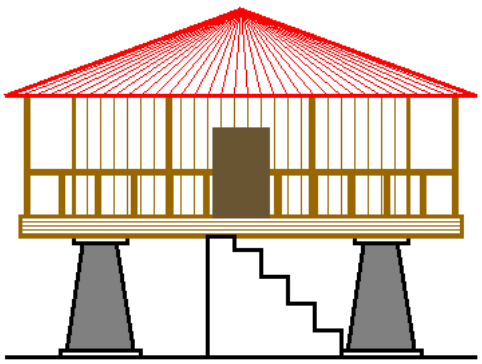
1. Un robot:



2. Una rana:



3. Un hórreo:



2. Iniciación a la Geometría de la tortuga

LOGO no está limitado a ningún área o tema en particular. 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.

Comenzamos por la descripción de las *primitivas* de XLOGO. 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.

Descripción de las primitivas o comandos

2.1. 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`
Borrar Pantalla y tortuga al centro	`borrapantalla`	`bp`

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

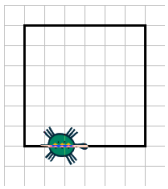
- `gi 90` equivale a `gd -90`
- `re 100` equivale a `av -100`

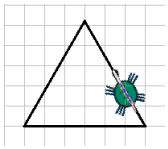
Haz todas las pruebas que necesites para entender perfectamente estas primitivas. Comprueba que la predicción del tema anterior de cómo realiza los giros es correcta, y si has entendido bien el punto de vista de la tortuga

2.2. Ejercicios

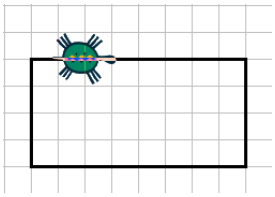
En los dibujos, el lado de cada cuadrado de la cuadrícula mide 25 "pasos de tortuga".

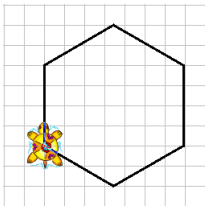
1. Dibuja el borde de un cuadrado en sentido *antihorario*
2. Dibuja el borde de un cuadrado en sentido *horario*





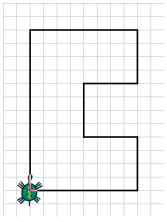
3. Dibuja el borde de un triángulo equilátero en sentido *antihorario*
4. Dibuja el borde de un rectángulo



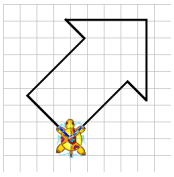


5. Dibuja el borde de un hexágono regular
6. Dibuja:

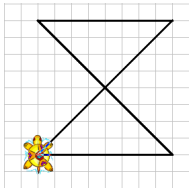
a)



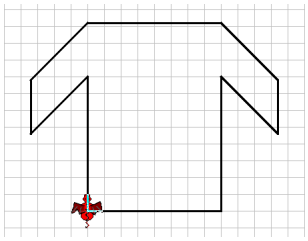
b)



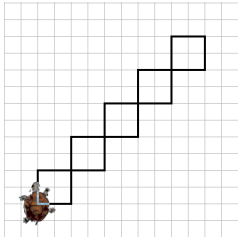
c)



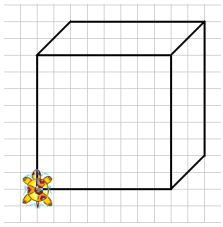
d)



e)



f)



2.3. Avanzando un poco

Continuemos con primitivas que controlan otros aspectos:

Primitiva	Forma larga	Forma corta
Subir Lápiz (no deja trazo al moverse)	`subelapiz`	`sl`
Bajar Lápiz (sí deja trazo al moverse)	`bajalapiz`	`bl`
Ocultar Tortuga	`ocultatortuga`	`ot`
Mostrar Tortuga	`muestratortuga`	`mt`

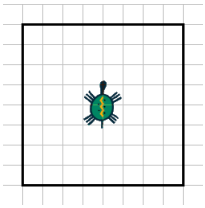
Primitiva	Forma larga	Forma corta
Cambiar el color del trazo con que dibuja	`poncolorlapiz n`	`poncl n`
Borrar por donde pasa, en vez de escribir	`goma`	`go`
Dejar de borrar y volver a escribir	`bajalapiz`	`bl`
Repetir `n` veces lo indicado entre corchetes	`repite n`	
Rellenar con el color activo una región cerrada	`rellena`	
Rellenar una región limitada por el color activo	`rellenazona`	

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 `repite` a hacer más sencillo tu programa.

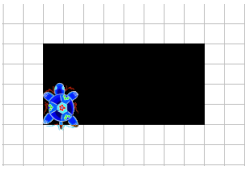
2.4. Ejercicios

De nuevo, el lado de cada cuadrado de la cuadrícula mide 25 "pasos de tortuga".

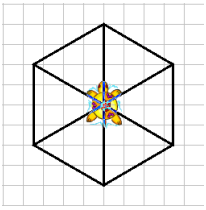
1. Dibuja el borde de un cuadrado, pero ahora usa la primitiva `repite`
2. Dibuja el borde de un triángulo equilátero usando la primitiva `repite`
3. Dibuja el borde de un hexágono regular usando la primitiva `repite`
4. Dibuja el borde de un cuadrado, cuyo centro esté en el centro de la pantalla
5.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



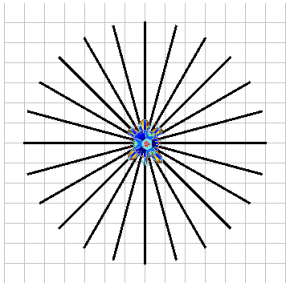
Problema 4



Problema 5



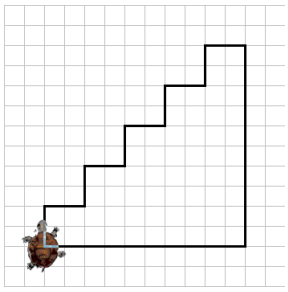
Problema 6



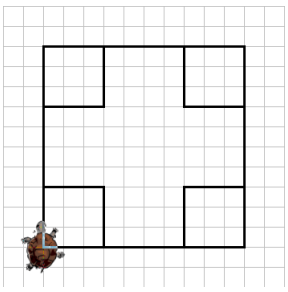
Problema 7

6. Dibuja el borde de un hexágono regular y las diagonales cuyos extremos son dos vértices opuestos del mismo
7. Dibuja los radios de una rueda. En total tienen que salirte 24
8. Dibuja:

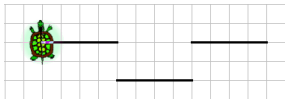
a)



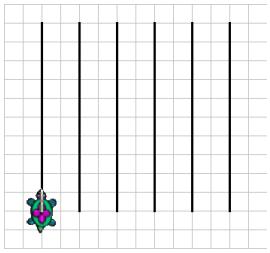
b)



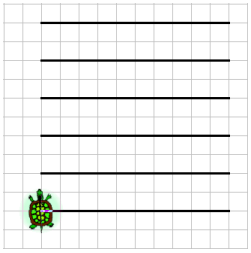
c)



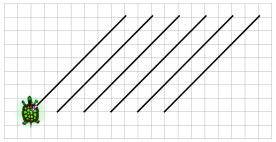
d)



e)

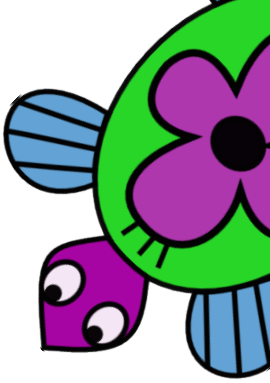


f)



¿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?



3. Procedimientos y subprocedimientos

3.1. Procedimientos

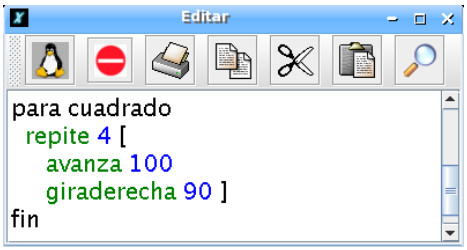
Observa que para dibujar un cuadrado de lado `100`, debes escribir:

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

cada vez que quieras que aparezca en pantalla. Y, ¿qué pasa si quiero que tenga lado `150` `200` o `300`? ¿Debería copiar la línea y cambiar cada vez la medida? ¿Y si es un rectángulo? La secuencia es más larga, y si quiero otro tamaño debería modificar **dos** medidas.

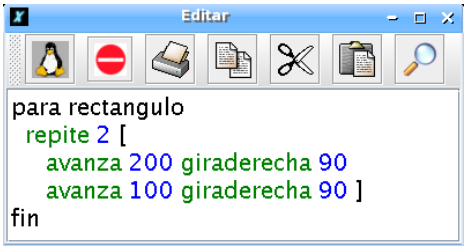
Podemos hacer que XLogo "aprenda" nuevas primitivas, definiendo lo que se denomina `procedimientos`.

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



y haz *click* en el botón del pingüino. Acabas de definir el procedimento `cuadrado`, y eso te permite dibujar un cuadrado de lado `100` cada vez que escribas `cuadrado` en la línea de comandos.

Prueba ahora con el siguiente procedimiento:



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

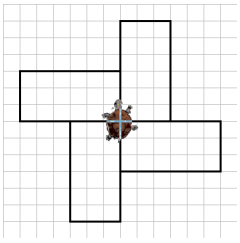
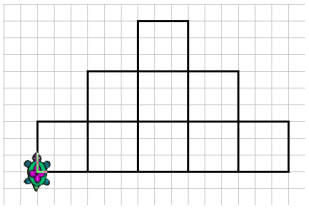
3.2. Ejercicios

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

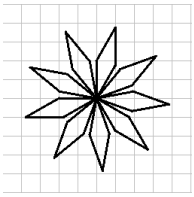
3. Plantea un procedimiento que dibuje un cuadrado en el centro de la pantalla

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:

a)  *b*) 

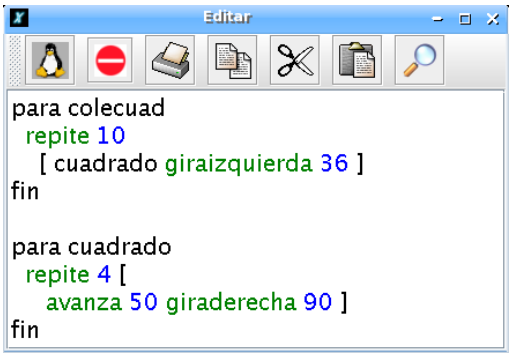
6. Plantea un procedimiento que dibuje un rombo, y con él dibuja la hélice:

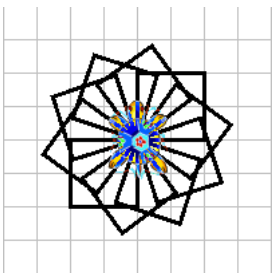


3.3. Sub-procedimientos

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

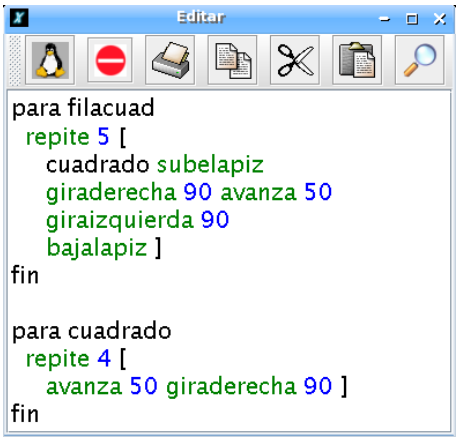
El siguiente programa dibuja una colección de cuadrados con un vértice común, cada uno girado 36 grados respecto del anterior:

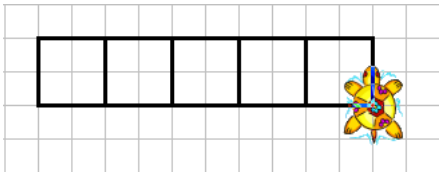




Observa que los nombres de los procedimientos están relacionados con el dibujo (u objetivo) que deseamos. Esta es una norma de *buena educación* a la hora de programar, y que ayuda a hacer más inteligibles los programas.

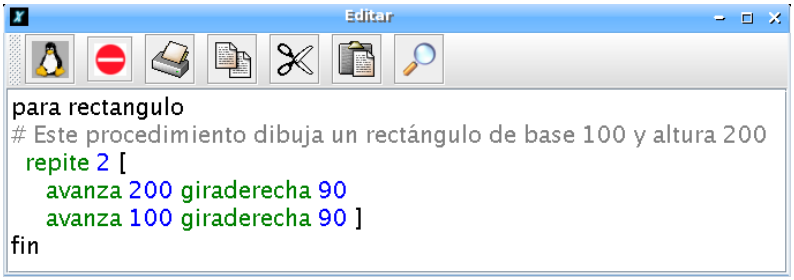
Analiza el siguiente código:





Antes citábamos una norma de *buena 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 y hacerlo aún más fácilmente legible. Teclea el siguiente procedimiento:



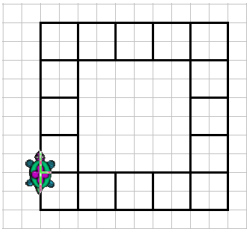
Puedes ver que la segunda línea empieza por `#`, y aparece en color gris. La "*almohadilla*" indica a la tortuga que es un comentario, así que ignora la línea y sigue leyendo las siguientes.

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 usarlo. 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`.

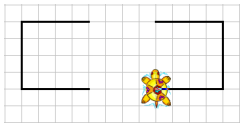
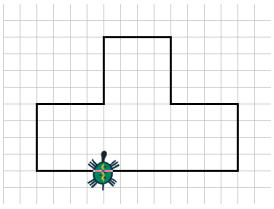
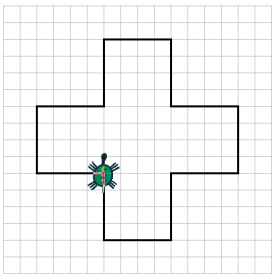
3.4. Ejercicios

1. Plantea un procedimiento que dibuje una colección de hexágonos regulares con un vértice en común, cada uno girado 60 grados respecto del anterior

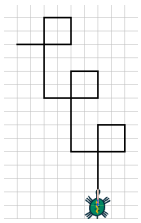
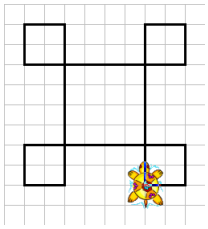
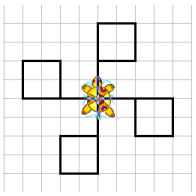
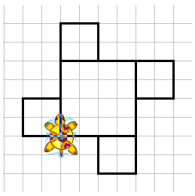
2. Usando el procedimiento `cuadrado`, dibuja:



3. 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 úsalo para dibujar:

a) 
b) 
c) 

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

a) 
b) 
c) 
d) 

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

4. Variables. Procedimientos con argumentos

4.1. Primitivas asociadas

Definimos ahora dos nuevas primitivas:

Descripción	Primitiva	Ejemplo
Guardar un `valor` en la variable `var`	`haz "var valor`	`haz "lado 115`
Mostrar el valor almacenado en la variable `var`	`escribe :var`	`escribe :lado`

Fíjate en la diferencia:

- Para definir la variable, se antepone "
- Para leer la variable, se precede de :

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
```

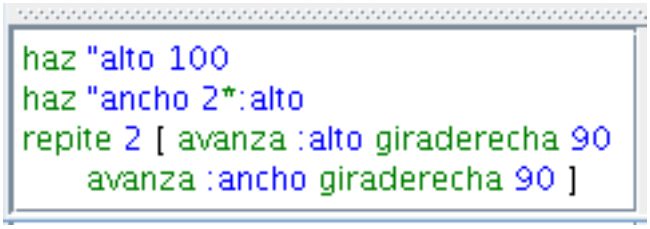
Frase: Para guardar en la variable `descripcion` la frase `El gato es gris`:

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

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

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

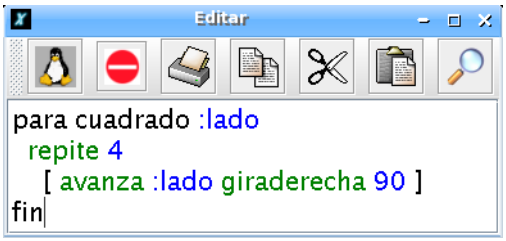
e incluso pueden usarse para definir otras:



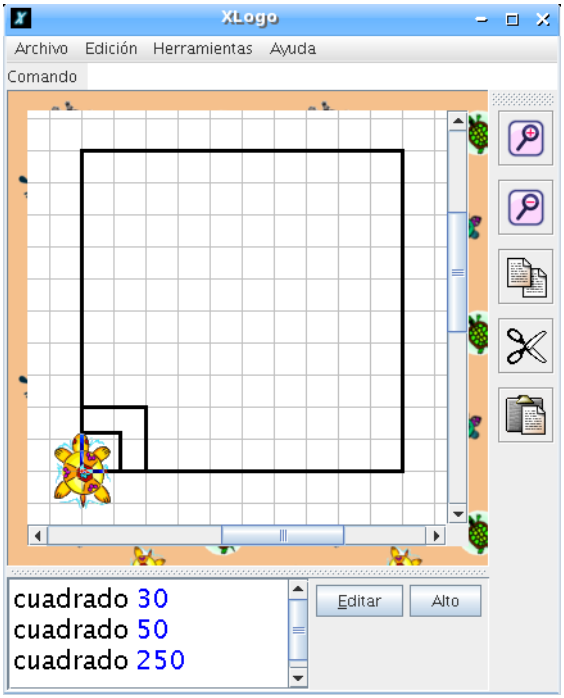
que dibuja un rectángulo de base doble que la altura.

4.2. Procedimientos con variables

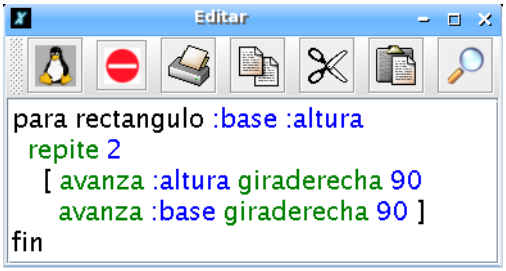
Resulta muy interesante la posibilidad de definir los procedimientos con *entradas*:



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:



Podemos prever varios *argumentos*:



al que llamaremos escribiendo:

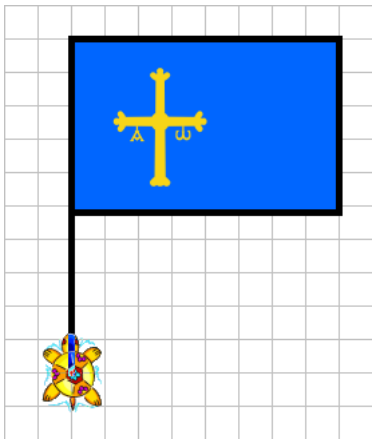
```
rectangulo 250 50
```

4.3. Ejercicios

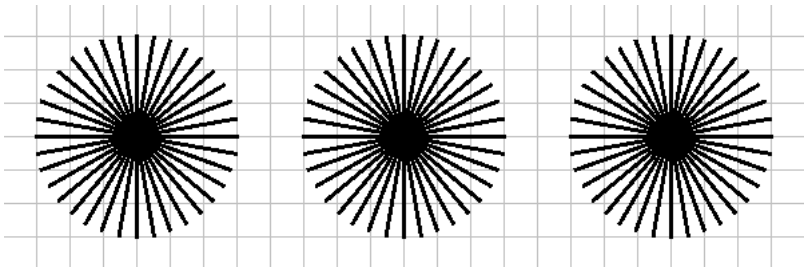
1. Plantea un procedimiento `triangulo` que necesite una variable `lado` y que dibuje un triángulo equilátero cuyo lado sea ese valor

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 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`



5. Operaciones

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

5.1. Operaciones binarias

Son aquellas que implican a dos elementos:

Suma: Disponemos de dos formas de resolverlas:

- Usando +: `17 + 25`
- Usando `suma`: `suma 17 25`

Si se trata de sumas de varios números:

```
13 + 7 + 2.5                (suma 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)
```

o puede ser usado para dibujar:

```
avanza 13 + 7 + 2.5      giraderecha (suma 13 7 2.5)
```

Producto: Como antes, disponemos de dos primitivas:

- Primitiva *: `1.4 * 5.3`
- Primitiva `producto`: `producto 1.4 5.3`

por ejemplo:

```
escribe 1.4 * 5.3 * 20   es (producto 13 5.3 20)
```

devuelven ambas:

```
148.4
```

Diferencia: De nuevo, dos posibilidades:

- Primitiva -: `14 - 7`
- Primitiva `diferencia`: `diferencia 14 7`

Cociente: Ahora debemos distinguir dos casos:

- División de números reales: `division` o `/`

  ```
  es division 14 5     escribe 14 / 5
  ```

  devuelven

  ```
  2.8
  ```

- Cociente entero: `cociente`

  ```
  es cociente 14 5
  ```

  devuelve

  ```
  2
  ```

XLogo dispone también de la operación *módulo*: `resto a b` devuelve el resto de la división entera entre `a` y `b`

```
escribe resto 14 5
```

devuelve

```
4
```

`(14 = 5 * 2 +` 4 `)`

Potencia: Disponemos de la primitiva `potencia`: 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

Concatenación de palabras/frases: No sólo pueden efectuarse operaciones con números. Las palabras y las frases (listas de palabras) pueden *concatenarse* (ponerse una a continuación de la otra) con la primitiva `frase`. Por ejemplo:

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

muestra en pantalla:

```
El gato es gris
```

O bien, 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`

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

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

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

Sin embargo, si escribimos:

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

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

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

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

pero observamos que XLogo devuelve:

```
729
```

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

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

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

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

o bien:

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

que se entiende mejor usando paréntesis:

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

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

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
```

5.2. Operaciones unitarias

Son aquellas que sólo necesitan un elemento:

Raiz cuadrada: `raizcuadrada` ó `rc` devuelve la raiz cuadrada de un número:

```
escribe raizcuadrada 64          es rc 64
```

devuelven

```
8
```

Cambiar el signo: Disponemos de dos opciones: la primitiva `cambiasigno` (cuya forma corta es `cs`) o anteponer un signo `-`

```
haz "prueba 3.35
escribe -:a
```

o bien:

```
haz "prueba 3.35
escribe cambiasigno :a
```

devuelven:

```
-3.35
```

Truncamiento de un número decimal: utilizamos `truncar`:

```
escribe truncar 3.25
escribe truncar -3.25
```

devuelven

```
3
-3
```

respectivamente.

También puede usarse esta primitiva para conseguir un número entero cuando nos aparece en notación científica. En el apartado 5.1 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
```

Redondeo: Para redondear un número al entero más cercano disponemos de `redondea`

```
escribe redondea 3.25
escribe redondea 3.75
escribe redondea -3.25
escribe redondea -3.75
```

devuelven

```
3
4
-3
-4
```

respectivamente

Números aleatorios: Para que XLogo genere un número aleatorio comprendido entre `0` y `n`, disponemos de `azar m`, con `m = n + 1`. Por ejemplo:

```
escribe azar 15
```

devuelve un número comprendido entre `0` y `14`

Funciones trigonométricas: Tanto directas:

```
escribe seno 30                 es sen 30
escribe coseno 45               es cos 45
escribe tangente 60             es tan 60
```

como inversas:

```
escribe arcoseno 0.25           es asen 0.25
escribe arcocoseno 0.5          es acos 0.5
escribe arcotangente 4          es atan 4
```

Estas primitivas tienen una prioridad inferior a la suma o la resta. Es decir:

```
escribe cambiasigno 3 + 1
```

hace primero la suma y luego cambia el signo, y proporciona:

```
-4
```

al igual que el resto:

```
escribe seno 20 + 10          ---> 0.5
escribe raizcuadrada 3 + 1    ---> 2
escribe azar 6 + 1            ---> 0, 1, 2, 3, 4, 5 o 6
```

Ejemplo 1: 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.

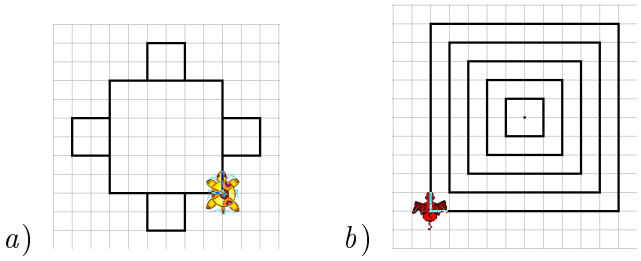
```
para division_entera :A :B
  haz "C truncar (:A / :B)
  escribe :C
  escribe :A - :B * :C
fin
```

Ejemplo 2: Busquemos ahora un procedimiento que simule el lanzamiento de un dado:

```
para dado
  escribe 1 + azar 6
fin
```

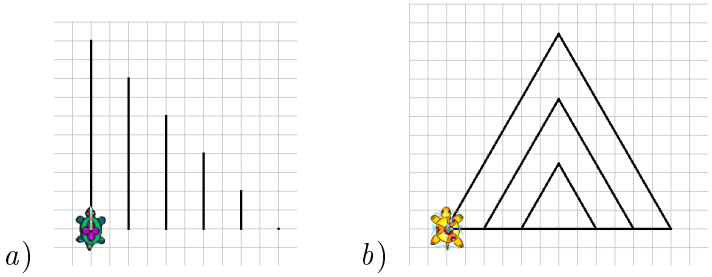
5.3. Ejercicios

1. Con el procedimiento `cuadrado` definido en la sección anterior, dibuja:



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

3. Dibuja:



4. Modifica el procedimiento `division_entera` de modo que devuelva los resultados indicando con mensajes qué es cada número.

5. Plantea un procedimiento `Pitagoras` que refiriéndose a un triángulo rectángulo, acepte los valores de los catetos y calcule el valor de la hipotenusa.

6. Escribe un procedimiento `rayos`, que dibuje los `n` radios de longitud `l` de una rueda (sólo los radios)

7. Plantea un procedimiento `poligono_regular`, que dibuje un polígono regular de `n` lados de longitud `lado`

8. Intenta escribir un procedimiento `suerte`, que genere un número al azar del conjunto `{20 , 25 , 30 , 35 , 40 , 45 , 50}`.[1]

9. Plantea un procedimento `dados`, que simule el lanzamiento de **dos** dados y cuya salida sea la suma de ambos

   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.

10. ¿Sabrías crear un procedimiento `poten :x :n` que calcule x^n (supuesto que `n` es un número natural) sin usar la primitiva `potencia`?

11. Crea un procedimiento `factorial :n`, que calcule el factorial del número `n`.

    Recuerda: `n! = n * (n - 1) * ...* 2 * 1`

12. Intenta ahora conseguir un procedimiento `suma_potencias :n` que calcule la suma `2 + 4 + 8 + ...+` 2^n

[1] Recuerda 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`

[2] Échale un vistazo a la sección 9.1 para entenderlas bien

6. 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 esta sección. Comencemos por la primitiva

```
cuadricula :x :y
```

que nos muestra una cuadrícula de anchura `:x` y altura `:y`. Con ella visible, será más fácil entender las siguientes primitivas:

Primitiva	Forma larga	Forma corta
Mostrar la posición (devuelve una lista)	`posicion`	`pos`
Mostrar sólo la abscisa (coordenada `X`)	`primero posicion`	`pr pos`
Mostrar sólo la ordenada (coordenada `Y`)	`ultimo posicion`	`ultimo pos`
Mover al punto `[X , Y]` (`X`,`Y` números)	`ponposicion [X Y]`	`ponpos [X Y]`
Orientar la tortuga hacia un rumbo `n`	`ponrumbo :n`	`ponr :n`
Pedir el `rumbo` (en grados respecto de la vertical y sentido horario)		`rumbo`
Para volver al origen con rumbo 0 (mirando hacia arriba)		`centro`
Mover hacia `[X , Y]` (`X`,`Y` números o variables)		`ponxy :X :Y`
Llevar hacia el punto de abscisa `X` (número o variable)		`ponx :X`
Llevar hacia el punto de oordenada `Y` (número o variable)		`pony :Y`

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
```

6.1. 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)

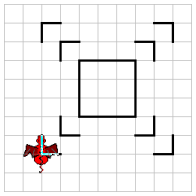
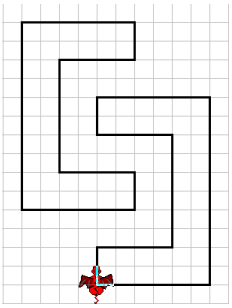
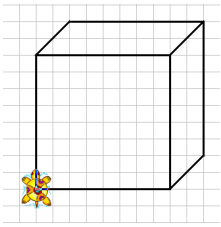
   Dato: La distancia entre dos puntos `(x0 , y0)` y `(x1 , y1)` se calcula mediante la fórmula:

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

6. Define procedimientos que pongan a la tortuga rumbo a los puntos cardinales: `norte`, `sur`, `este`, `oeste`, `noroeste`, `nordeste`, `sudeste` y `suroeste`.

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

8. Dibuja:

a)  *b*)  *c*) 

7. Condicionales y Operaciones lógicas

En ocasiones será necesario decidir qué acción realizar en función de una determinada condición. Por ejemplo, si quiero calcular una raiz 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 raiz cuadrada de un negativo no es un número real.

XLogo defiene 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
```

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

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

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 raiz. 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 frio] ]
      [ escribe [Hace demasiado calor] ] ]
fin
```

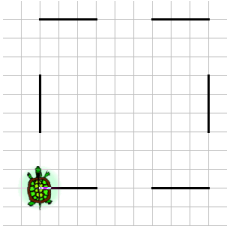
que estudia primero si la temperatura está en el intervalo `(15 , 25)` – o sea `15 < T < 25` – y si lo está dice `La temperatura es agradable`. Si no pertenece a ese intervalo, analiza si está por debajo de él (y dice `Hace frio`) o no (y devuelve `Hace demasiado calor`).

Hemos *encadenado* condicionales

Analiza el siguiente procedimiento:

```
para cuadrado_cortado
  haz "paso cambiasigno 50
  repite 4
    [ repite 3
        [ si bajalapiz? [subelapiz] [bajalapiz]
          avanza 75 ]
      giraderecha 90 ]
fin
```

¿Eres capaz de reproducir su resultado? Por si acaso, aquí lo tienes:



7.1. Ejercicios

1. Intenta modificar 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 `si` en caso afirmativo

3. Plantea el procedimiento `edad_laboral`, que compruebe si la `edad` de una persona verifica la condición `17 < edad < 65`, respondiendo `Esta en edad laboral` en caso afirmativo

4. Escribe el procedimiento `multiplo`, que verifique si un número `dato` es múltiplo de otro `divisor`, respondiendo `Es multiplo` o `No es multiplo`, en cada caso.

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

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

   Pista: Comprueba el signo de su producto

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.

8. Listas

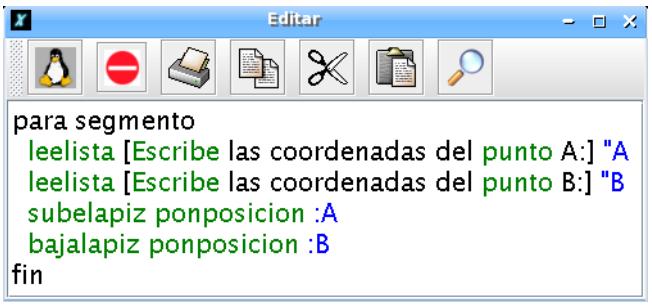
Ya hablamos antes de listas. `[ 53 gato [7 28] 4.9 ]` es una lista en XLogo; su primer elemento es `53`, el segundo es `gato`, el tercero es `[7 28]` y el último es `4.9`. Para almacenarlo en la variable `ejemplo` hacemos:

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

Disponemos de las siguiente primitivas para trabajar con listas:

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`	
Quitar el primer elemento	`menosprimero :ejemplo`	`mp :ejemplo`
Quitar el último elemento	`menosultimo :ejemplo`	`mu :ejemplo`
Quitar el elemento `gato`	`quita "gato :ejemplo`	
Añadir `algo` el primero	`ponprimero "algo :ejemplo`	`pp "algo :ejemplo`
Añadir `algo` el último	`ponultimo "algo :ejemplo`	`pu "algo :ejemplo`
Intercalar `algo` en el lugar `n`	`agrega "algo n :ejemplo`	
Concatenar `ejemplo` y `otra`	`frase :ejemplo :algo`	`fr :ejemplo :algo`

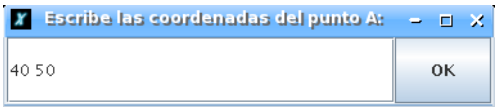
Existe una primitiva que permite que el *usuario* introduzca valores en XLogo: `leelista`:

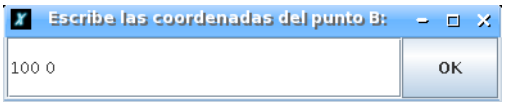


que se ejecuta tecleando:

```
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.

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`.

8.1. Ejercicios

1. ¿Cómo puede extraerse el `22` de la lista de listas `[[22 3] [4 5] [8 35]]`?

2. Plantea el procedimiento `prime`, con una entrada, `listado`, que devuelva su primer elemento, sin usar `primero`

3. Plantea el procedimiento `ulti`, con una entrada, `listado`, que devuelva su último elemento, sin usar `ultimo`

4. Diseña el procedimiento `triangulo`, que **pida** las coordenadas de los vértices de un triángulo uno a uno y lo dibuje

5. Escribe un procedimiento que pida la medida del lado de un cuadrado y devuelva la medida de su diagonal

6. 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.

7. Diseña un procedimiento `inversa` que reciba una lista y la devuelva con los elementos dispuestos en orden inverso al inicial

8. Plantea el procedimiento `maximo`, que pida una serie de números y devuelva el mayor de todos ellos

9. 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`

10. 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`)

9. Bucles y recursividad

Ya hemos visto un tipo de bucles, el controlado por `repite`. Realmente, XLogo dispone de tres primitivas que permiten la construcción de bucles: `repite`, `repitepara` y `mientras`.

9.1. Bucle con `repite`

La sintaxis para `repite` es:

```
repite n [ lista_de_comandos ]
```

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

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

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 [es contador]
```

proporciona

```
1
2
3
```

9.2. 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] [es :i*2]
```

proporciona

```
2
4
6
8
```

Ejemplo 2:

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

proporciona

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

9.3. 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] [gd 1] # La tortuga gira sobre si misma eternamente.
```

```
# Este ejemplo deletrea el alfabeto en orden inverso:
haz "lista1 "abcdefghijklmnñopqrstuvwxyz
mientras [no vacio? :lista1]
[es ultimo :lista1 haz "lista1 menosultimo :lista1]
```

9.4. Uso avanzado de procedimientos

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`.

9.5. Recursividad

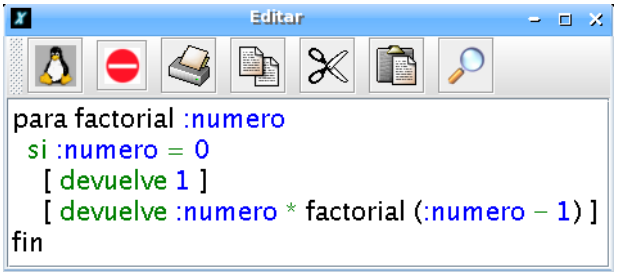
Un procedimiento se llama *recursivo* cuando se llama a sí mismo (es un *subprocedimiento* de sí mismo). Un ejemplo típico es el cálculo del **factorial**. En lugar de definir

```
n! = n * (n - 1) * ...* 2 * 1,
```

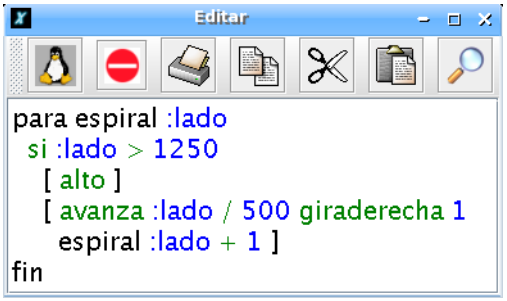
podemos hacer:

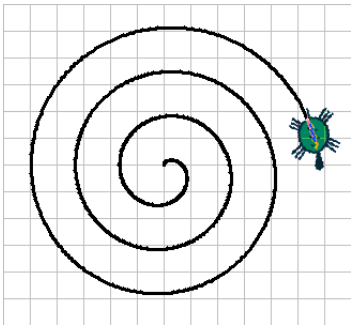
$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n \neq 0 \end{cases} \qquad \forall n \in \mathbb{N}$$

En XLogo:



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





9.6. 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 sutituye por su valor por defecto, esto es, `100`, y XLogo dibuja un octógono de lado `100`. Sin embargo, ejecuando

```
(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.

9.7. 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 , ...`
3. Diseña un procedimiento recursivo que devuelva la suma de los `n` primeros números pares (excluido el cero):

   `2 + 4 + 6 + 8 + ... + 2 n`
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