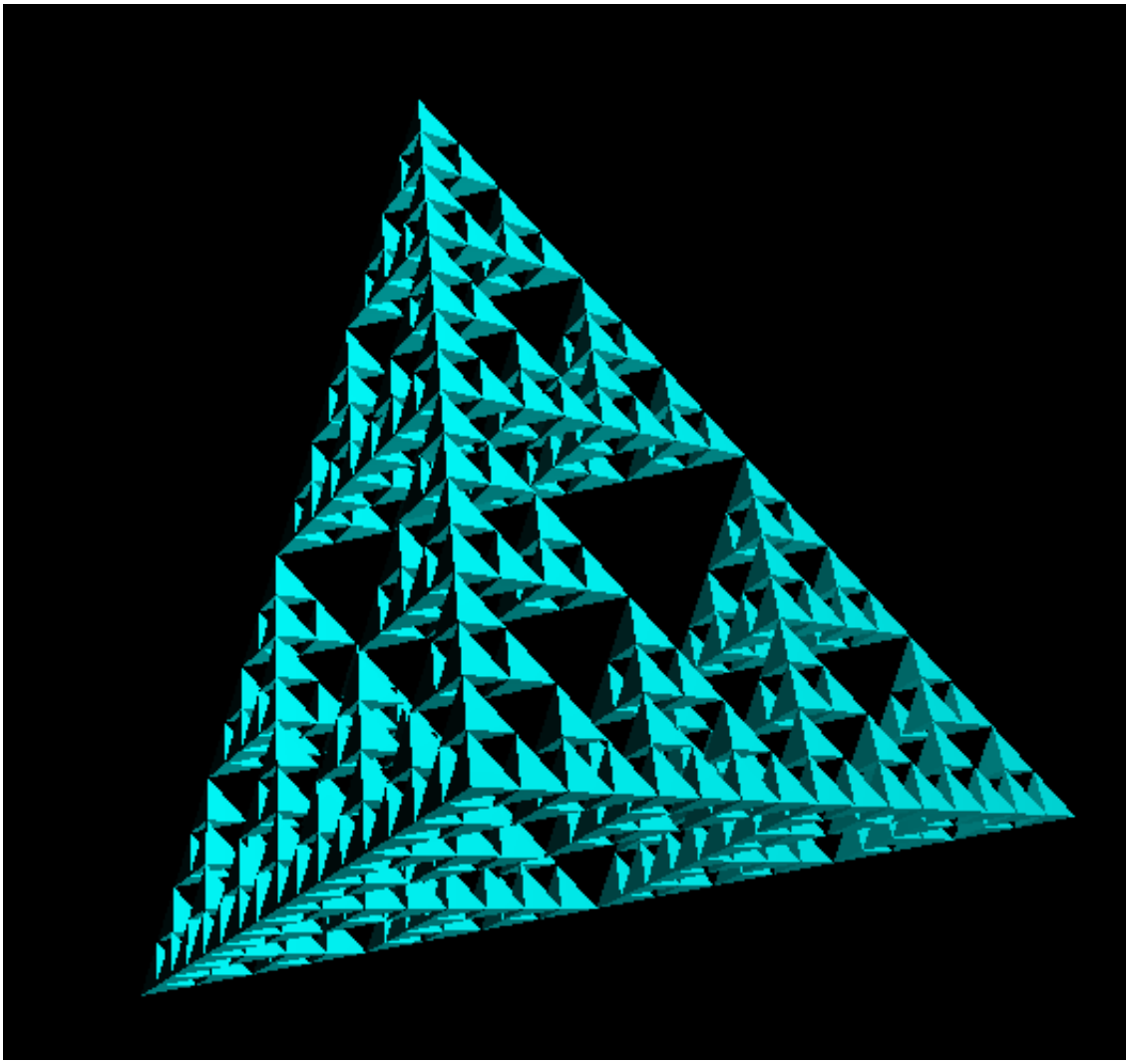
XLogo: Manuale di riferimento per la versione 0.9.96

Loïc Le Coq

Traduzione: Marco Bascietto

24 agosto 2009



http://xlogo.tuxfamily.org

Capitolo 1

Introduzione

Logo è un linguaggio di programmazione sviluppato negli anni 60 da Seymour Papert. Papert è stato lo sviluppatore una teoria sull'apprendimento molto originale ed influente chiamata "costruzionismo" che si può riassumere con l'espressione "imparando facendo".

Logo è un linguaggio molto appropriato per sviluppare le capacità matematiche e logiche. È un eccellente linguaggio per cominciare a programmare, infatti insegna le fondamenta di cose come i cicli, i test, le procedure ecc. L'utente muove un oggetto chiamato "tartaruga" per tutto lo schermo, utilizzando semplici comandi come avanti, indietro, destra e così via. Muovendosi la tartaruga lascia una traccia dietro di sè rendendo possibile la creazione di disegni. Il fatto che l'utente possa impartire alla tartaruga ordini in un linguaggio molto naturale rende Logo molto semplice da imparare. Un uso più avanzato è comunque possibile essendo possibile operare su liste, parole o file.

XLogo è un interprete Logo cioè le istruzioni dell'utente vengono eseguite direttamente. L'utente può osservare gli errori nel programma immediatamente a schermo. Questo approccio grafico molto intuito rende Logo un linguaggio ideale per i principianti, specialmente per i bambini!

L'indirizzo principale del sito web di XLogo è

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

Lì è possibile ottenere sia il software sia la documentazione. Sul sito si trovano anche molti esempi creati con XLogo così da potersi fare un'opinione sulle capacità di XLogo.

XLogo supporta ora undici linguaggi (arabico, asturiano, inglese, italiano, esperanto, francese, galiziano, greco, tedesco, portoghese, e spagnolo). XLogo è scritto in Java - un linguaggio di programmazione con il vantaggio di essere multi-piattaforma - rendendolo utilizzabile su macchine Linux, Windows e MacOS senza alcun problema.

XLogo è posto sotto licenza GPL. È quindi software libero ed i suoi utenti hanno quattro libertà:

- Libertà 1: La libertà di eseguire il programma per qualsiasi scopo.
- Libertà 2: La libertà di studiare e modificare il programma.
- Libertà 3: La libertà di copiare il programma così da aiutare il prossimo.
- Libertà 4: Libertà di migliorare il programma e di distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne tragga beneficio.

Organizzazione del manuale:

Il manuale ti aiuterà a scoprire XLogo:

- Nella prima parte vengono spiegati i diversi menu e le opzioni dell'interfaccia.
- Successivamente, alcuni capitoli presentano le istruzioni più importanti per cominciare ad usare XLogo. Le prime istruzioni sono molto semplici, poi la complessità cresce. A volte il capitolo si chiude con qualche esercizio la cui soluzione si trova nell'appendice D.
- Infine una serie di temi approfonditi si presentano per gli utenti esperti.
- Nell'appendice A sono elencate tutte le primitive di XLogo, ciascuna dettagliatamente descritta.

Questo manuale è disponibile in diversi formati:

- PDF: http://downloads.tuxfamily.org/xlogo/downloads-it/manual-it.pdf
- HTML compresso: http://downloads.tuxfamily.org/xlogo/downloads-it/manual-html-it.zip
- LaTeX 2_ε: Sorgente: http://downloads.tuxfamily.org/xlogo/downloads-fr/manual-src-it.zip
- JavaHelp: Menu Aiuto-Manuale Online in XLogo

Indice

Capitolo 2

Installare XLogo

- Prima di tutto occorre installare la Java Runtime Environment (JRE) sul proprio computer. Vai a questa pagina:

  `http://java.sun.com/javase/downloads/index.jsp`

  Scarica la JRE corrispondente al proprio sistema operativo (Windows, Linux, MacOS), ed installala.

- Quindi occorre scaricare il file `xlogo.jar` dal seguente indirizzo:

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

  In alternativa si può andare sul sito web di XLogo all'indirizzo `http://xlogo.tuxfamily.org`, scegliere una lingua e quindi cliccare sulla voce di menu Download.

2.1 Configurazione di XLogo

2.1.1 Ambiente Linux

Nota: XLogo è già incluso nella distribuzione OpenSuse.
In Ubuntu 8.04:

1. Per installare la JRE:
   - Sistema -> Amministrazione -> Gestore Pacchetti Synaptic
   - Installare il pacchetto `sun-java6-jre`
2. Per aprire il file `xlogo.jar`:
   - Clicca il tasto destro del mouse su `xlogo.jar`, Proprietà
   - Tab "Apri con": Scegli Sun Java 6 Runtime
3. Per associare l'estensione `lgo` a XLogo:
   - Clicca il tasto destro del mouse su `xlogo.jar`, Proprietà
   - Tab "Apri con"
   - Bottone "Aggiungi"
   - Campo "Usare un comando personalizzato", digita (sostituendo il percorso verso XLogo a "percorso_a_"):

     `java -jar percorso_a_xlogo.jar`

2.1.2 Ambiente Windows

In teoria, se clicchi sull'icona di XLogo il programma dovrebbe partire. Se XLogo parte correttamente, salta il resto del capitolo. Se invece viene lanciata un'altra applicazione (per esempio winzip) questo succede poiché i file .jar vengono interpretati come file .zip (ossia compressi) e viene eseguito il programma per la loro decompressione. Occorre quindi deattivare l'associazione di quel programma con i file .jar. Per far questo segui i seguenti passi per Windows XP (alcuni percorsi potrebbero essere diversi a seconda della versione di Windows in funzione, dovrai modificarli appropriatamente):

1. Avvio -> Pannello di controllo -> Passa alla modalità classica -> Opzioni cartella
2. Clicca sul Tab "Tipi di file" (il terzo Tab)
3. Cerca nella lista dei file registrati, quelli connessi con i file jar (file jar, file jar eseguibili, archivi jar, ecc)
4. Clicca il tipo di file, quindi su Avanzate
5. Appare una nuova finestra, clicca su Sfoglia...
6. Naviga verso javaw.exe che di solito è posto presso:

   `c:\Programmi \java\j2re1.4.1\bin\javaw.exe`

7. Il percorso "c:\Programmi \java\j2re1.4.1\bin\javaw.exe" appare quindi nel campo *Applicazione utilizzata per eseguire l'azione:*. Occorre aggiungere delle informazioni alla sua fine così che si legga:

   `c:\Program Files\java\j2re1.4.1\bin\javaw.exe -jar %1 %*`

   (nota che c'è uno spazio su entrambi i lati di -jar).
8. Infine, chiudi tutte le finestre di dialogo. Ora tutto ciò che rimare da fare è cliccare sull'icona di XLogo per lanciarlo!

Se ancora XLogo non parte, c'è una seconda possibilità. Apri una finestra MSDOS (su XP: Avvio -> Tutti i programmi -> Accessori -> Prompt dei comandi), quindi digitare il seguente comando (sostituendo il percorso verso XLogo a "percorso_a_"):

```
java -jar percorso_a_XLogo

Per esempio: java -jar c:\windows\office\xlogo.jar
```

(se `xlogo.jar` è posto in questa cartella).

Se digitare questo comando ogni volta è troppo lungo, crea un file di nome (per esempio) xlogo.bat ed inseriscici il comando stesso. Puoi quindi cliccare su questo file per lanciare XLogo.

Associare i file con estensione .lgo a XLogo

I file con estensione .lgo non sono di solito riconosciuti dal sistema operativo e quando si clicca su uno di loro una finestra di dialogo appare richiedendo di selezionare un'applicazione per aprirli. Selezionare `Altro` e quindi fornire il percorso a `javaw.exe`

Di solito è: `C:\Programmi \java\j2re1.4.1\bin\javaw.exe`

Viene quindi chiesto di inserire un nome per designare i file con l'estensione `.lgo`.
Per esempio: `File XLogo`
Per impostarlo come default in Windows XP, segui i seguenti passi:

1. Avvio -> Pannello di controlo -> Passa alla modalità classica -> Opzioni cartella
2. Clicca sul Tab “Tipi di file” (il terzo Tab)
3. Cerca nella lista dei file registrati, quelli connessi con i file jar (file jar, file jar eseguibili, archivi jar, ecc)
4. Clicca il tipo di file, quindi su Nuovo
5. Digita l’estensione .lgo nel riquadro Estensione File e clicca OK
6. Clicca nella riga LGO appena aggiunta nell’elenco nei tipi di file registrati e clicca Avanzate
7. Apapre una nuova finestra, clicca su Nuovo
8. Sotto Azioni, inserisci apri e quindi clicca su Sfoglia... naviga perso javaw.exe che, usualmente è in

   `c:\Programmi \java\j2re1.4.1\bin\javaw.exe`
9. Clicca su Apri per aggiungere il percorso al riquadro Azioni della fienstra di dialogo Modifica Tipo di file.
10. Clicca su apri quindi su modifica
11. Il persorso “c:\Programmi \java\j2re1.4.1\bin\javaw.exe” appare quindi nel campo *Applicazione utilizzata per eseguire l’azione:*. Occorre aggiungere delle informazioni alla sua fine così che si legga:

    `c:\Program Files\java\j2re1.4.1\bin\javaw.exe -jar %1 %*`
12. Infine, chiudi tutte le finestre di dialogo. Ora tutto ciò che rimare da fare è cliccare sull’icona d el file lgo per lanciare XLogo!

2.2 Aggiornamenti XLogo

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

Per aggiornare XLogo, occorre semplicemente rimpiazzare il file `xlogo.jar` con la nuova versione. Se vuoi ricevere un avviso quando viene pubblicata una nuova versione puoi abbonarti al feed RSS di XLogo. L’indirizzo è

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

Molti programmi possono gestire gli abbonamenti RSS. Se non ti senti a tuo agio con questa tecnica la scelta più facile è usare Mozilla Thunderbird:

- Menu Modifica - Impostazioni Account
- Bottone “Aggiungi account”
- “News RSS & Blog”, Next
- Nome Account: “Feed RSS” per esempio
- Bottoni “Continua” and “Fine”
- Nella finestra principale “Impostazioni Account”, Seleziona “Feed RSS” nel menu di sinistra e clicca sul bottone “Gestione Sottoscrizioni”.
- Bottone “Aggiungi”
  - URL del Feed: `http://xlogo.tuxfamily.org/rss.xml`
  - Seleziona l’oggetto “Mostra il sommario dell’articolo invece di caricare la pagina web”

È finita, con il bottone “Ricevi posta”, puoi ricevere le news XLogo insieme alle email.

2.3 Disinstallazione

Per disinstallare XLogo, tutto ciò che occore fare è cancellare il file `xlogo.jar` ed il file di configurazione `.xlogo`, che è posto nella cartella home dell'utente (`/home/account` per gli utenti Linux, o `c:\windows\.xlogo` per gli utenti Windows).

Capitolo 3

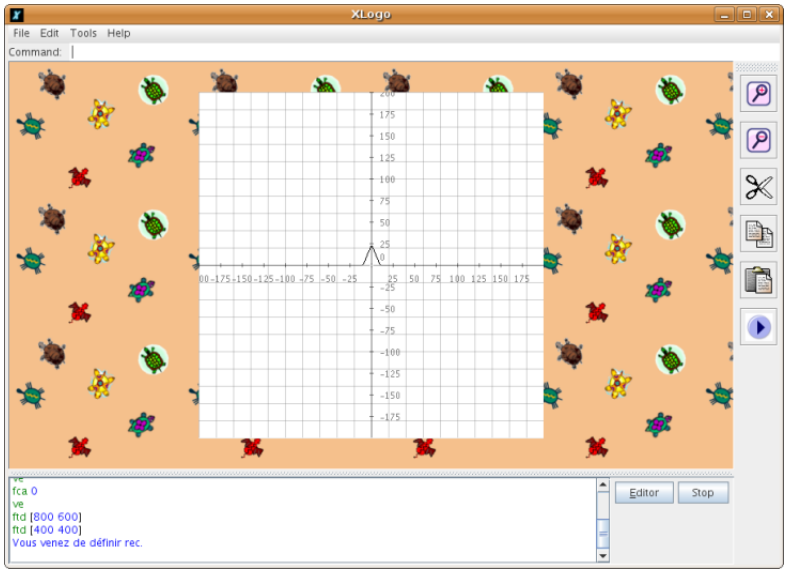
Descrizione dell'interfaccia

3.1 Prima esecuzione

La prima volta che XLogo viene lanciato (o se il file `.xlogo` è stato cancellato -Leggi la sezione 2.3-) una finestra di dialogo appare per chiedere di definire la lingua di utilizzo del programma stesso e dei programmi logo che con esso si realizzeranno.

La lingua predefinita può essere cambiata in qualsiasi momento nella finestra delle Preferenze (Sezione 4.3).

3.2 La finestra principale



- Nella parte superiore vi sono i menu usuali **File Modifica Strumenti e Aiuto**
- Appena al sotto è collocata la **linea di comando**, che permette alle istruzioni logo di essere eseguite.
- Nella parte centrale dello schermo vi è l'**area di disegno**.
- Sulla destra dell'area di disegno una **barra degli strumenti** permette all'utente di eseguire numerose azioni:

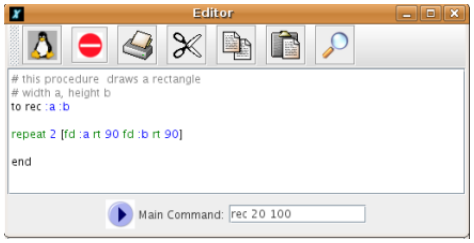
  - Ingrandire e rimpicciolire.
  - Modifica (taglia/copia/incolla)
  - Il bottone "avvia" lancia il comando principale definito nell'editor.

- Nella parte inferiore vi è lo **storico dei comandi** che mostra tutti i comandi inseriti e la risposta associata dell'interprete. Per richiamare velocemente un comando che è stato già inserito ci sono due opzioni: o si clicca sul vecchio comando nello storico o si può cliccare sulla bassa di scorrimento finché il vecchio comando appare nell'elenco. La barra di scorrimento permette di navigare attraverso tutti i comandi inseriti (molto pratico).

- Alla destra dello storico ci sono due bottoni: **STOP** e **EDITOR**.

  - Il bottone STOP interrompe l'esecuzione del programma logo.
  - Il bottone EDITOR apre la l'editor delle procedure.

3.3 L'editor delle procedure



Esistono tre modi per aprire l'editor:

- Digita `ed` sulla linea di comando nella parte superiore dello schermo. L'editor si apre mostrando tutte le procedure già definite. Se si vuole modificare una o più procedure specifiche digitare:
  `ed [procedura_1 procedura_2 ...]`

- Premi il bottone Editor nello schermo principale.

- Premi i tasti Alt+E sulla tastiera.

I seguenti sono i vari bottoni che si trovano nell'editor:

Salva i cambiamenti effettuati nell'editor e ne chiude la finestra. È questo il bottone che occorre premere ogni volta per inserire le nuove procedure digitate. È possibile anche premere i tasti Alt+Q sulla tastiera.

Chiude la finestra dell'editor senza salvare i cambiamenti effettuati. È possibile anche premere i tasti Alt+C sulla tastiera.

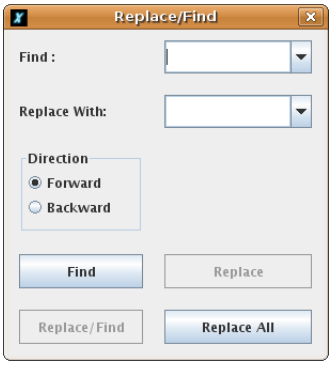
Stampa il contenuto dell'editor.

Copia il testo selezionato negli appunti.

Taglia il testo selezionato negli appunti.

Incolla il testo selezionato negli appunti.

Apri un riquadro per Cercare/Sostituire del testo nell'editor.



 Nella farte inferiore dell'editor un campo di testo permette di definire un comando principale. Questo comando è l'istruzione generale che lancia il programma logo. Può essere acceduta tramite il bottone "Avvia" dalla barra degli strumenti della finestra principale. Questo comando viene salvato e caricato insieme al programma logo (con estensione `.lgo`) viene salvato tramite l'editor.

IMPORTANTE

- Da notare che cliccando sul bottone di chiusura della finestra dell'editor (x), l'editor non si chiuderà! Solo uno dei due bottoni principale consente la chiusura dell'editor.
- Per cancellare una o più procedure occorre usare le primitive `CancProc` e `CancTutte` nella linea di comando, o usare il gestore delle procedure che si trova nel menu Strumenti.

3.4 Chiudere XLogo

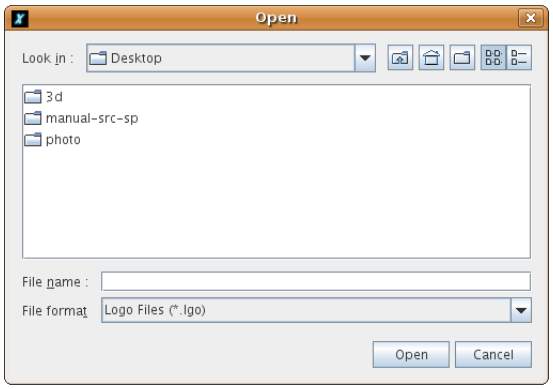
Per chiudere XLogo, si può scegliere **File - Abbandona** nella barra dei menu, o cliccare sul bottone di chiusura nella barra del titolo della finestra. una finestra di dialogo chiede quindi di confermare la chiusura del programma.
Nota per l'ambiente Mac. In ambiente Mac è preferibile non utilizzare la funzione di chiusura dell'applicazione presente di default nella barra dei menu fornita da Mac OS ma utilizzare i due metodi esposti al paragrafo precedente.

Capitolo 4

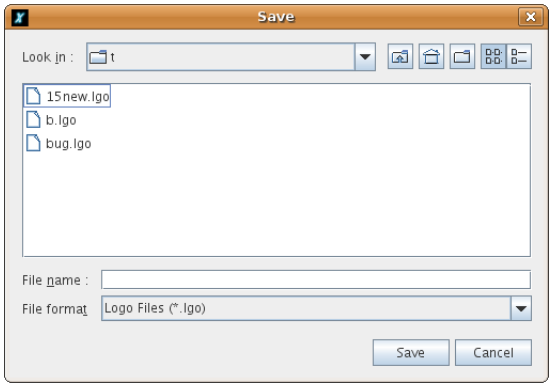
Opzioni del menu

4.1 Menu “File”

- **File→Nuovo**: crea un nuovo ambiente di lavoro, cancella tutte le procedure e variabili.
- **File→Apri**: apre un file logo precedentemente salvato.



- **File→Registra come...** salva le procedure correnti con un nome diverso.



- **File→Registra**: salva le procedure nel file corrente
- **File→Cattura→Registra come...**: permette di salvare l’immagine dell’area di disegno in formato jpeg o png. Se vuoi registrare solo una parte dell’immagine puopi definire un’area rettangolare trascinando il mouse sull’area di disegno.
- **File→Cattura→Stampa...**: permette di stampare l’immagine o parte di essa.

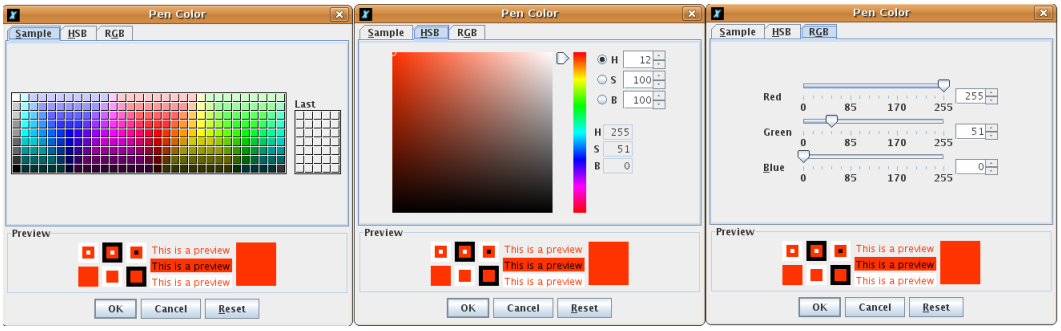
- **File→Cattura→Copia Appunti**: copia l'immagine o parte di essa negli appunti di sistema. Funziona in ambiente Windows e Mac OS, non in Linux (gli appunti hanno un comportamento diverso).
- **File→Testo→Registra come... (formato RTF)**: registra lo storico dei comandi in formato RTF (preservando colori e formati).
- **File→Abbandona**: esce da XLogo.

4.2 Menu "Modifica"

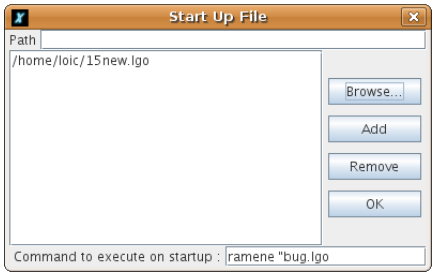
- **Modifica→Copia**: copia il testo selezionato negli appunti.
- **Modifica→Taglia**: taglia il testo selezionato negli appunti.
- **Modifica→Incolla**: Incolla il testo degli appunti nella linea di comando.
- **Modifica→Selezina tutto**: Seleziona tutto il testo nella linea di comando.

4.3 Menu "Strumenti"

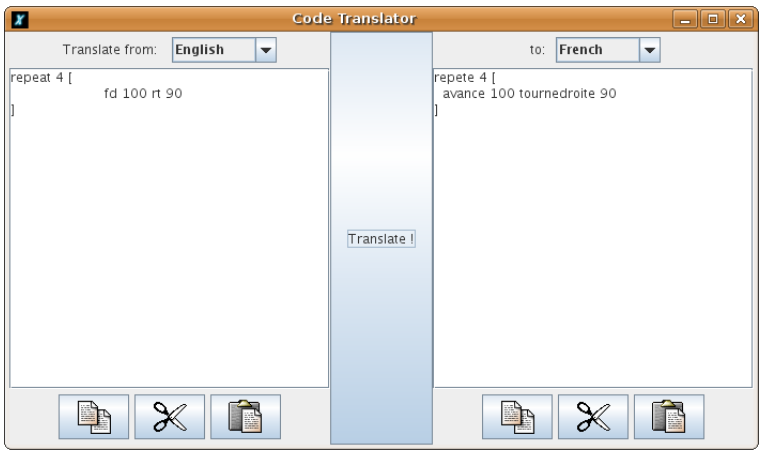
- **Strumenti→Colore Penna**: permette di impostare il colore della penna della tartaruga scegliendo da una gamma di colori. E' anche accessibile dal comando `ImpCP`.



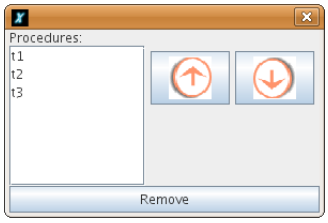
- **Strumenti→Colore Sfondo**: imposta il colore dello sfondo dell'area di disegno, anche accessibile tramite la primitiva `ImpCS`.
- **Strumenti→File di partenza**: permette di definire il percorso ad un file di avvio. Tutte le procedure contenute in questo file .lgo diventeranno quindi "pseudo-primitive" nel linguaggio XLogo. Non possono essere modificate dall'utente. Puoi quindi definire primitive personalizzate.



- **Strumenti→Traduttore primitive**: permette di tradurre il codice di XLogo da una lingua all'altra. E' molto utile quando si vuole tradurre un esempio scritto in un'altra lingua.

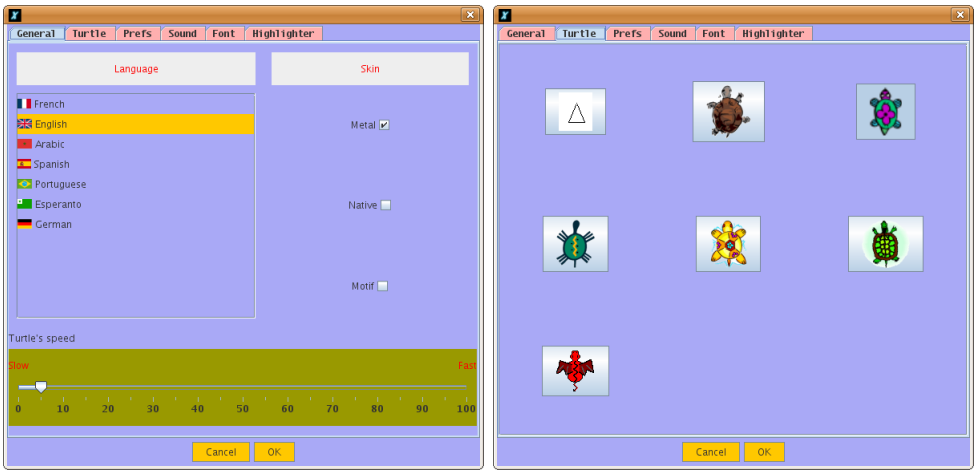


- **Strumenti→Gestione procedure**: apre una finestra che permette di cancellare le procedure definite nell'editor. Puoi anche definire l'ordine di comparsa delle procedure nell'editor.



- **Strumenti→Preferenze**: apre una finestra che permette di configurare diversi aspetti di XLogo:
  - **Tab generale**
    - → **Lingua**: permette di scegliere una diversa lingua, nta che anche le primitive vengono tradotte nella lingua scelta.
    - → **Aspetto**: permette di impostare l'aspetto di XLogotra "Metal", "Nativo Java" e "Motif".
    - → **Velocità della tartaruga**: Se preferisci osservare i movimenti della tartaruga puoi rallentarla usando la slitta.

.



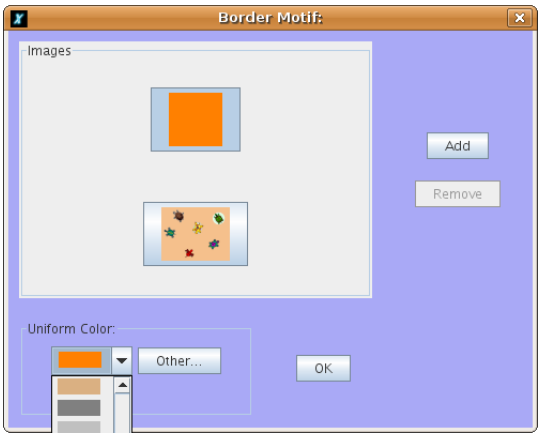
  - **Tab opzioni**: Vi sono molte opzioni:
    - → **Griglia di sfondo**: Puoi scegliere di disegnare una griglia sullo sfondo dell'area di disegno. Puoi definire l'ampiezza e l'altezza del quadrato della griglia ed il suo colore.

→ **Assi sullo sfondo dello schermo**: Puoi disegnare l'asse orizzontale e/o verticale sullo sfondo dell'area di disegno. Puoi definire la distanza tra due divisioni dell'asse ed il colore dell'asse.

→ **Colore dello schermo**: Puoi definire un colore dello schermo preimpostato.

→ **Colore della penna**: Puoi definire un colore preimpostato della penna.

→ **Motivo del bordo**: Puoi scegliere il tuo motivo per l'area esterna a quella del disegno.



→ **Massimo numero di tartarughe**

→ **Massima ampiezza della penna**: Puoi scegliere la massima ampiezza della penna permessa (-1 disabilita un limite massimo).

→ **Forma della punta della penna**

→ **Qualità della traccia da disegno**: Puoi scegliere l'accuratezza della traccia del disegno. In alta qualità gli angoli saranno arrotondati ma la velocità di disegno diminuisce.

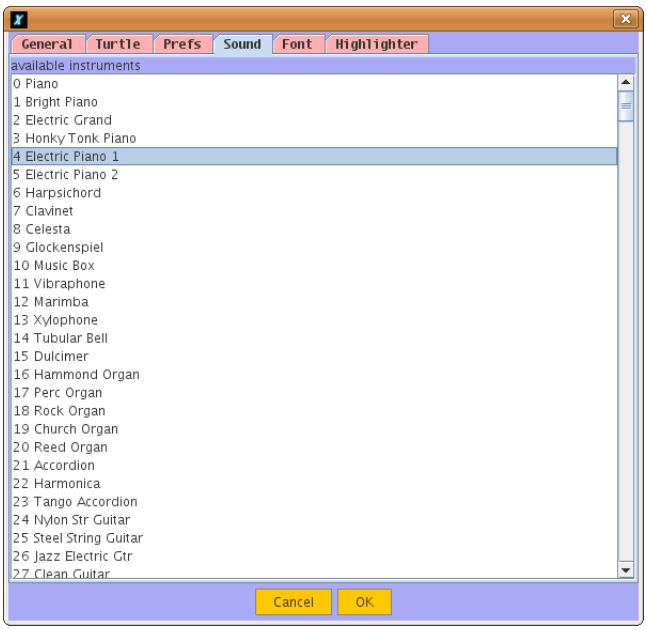
→ **Pulisci lo schermo quando si chiude l'editor**: alla chiusura dell'editor l'area di disegno viene ripulita o meno.

→ **Cancella le variabili quando chiudi l'editor**: alla chiusura dell'editor tutte le variabili vengono cancellate.

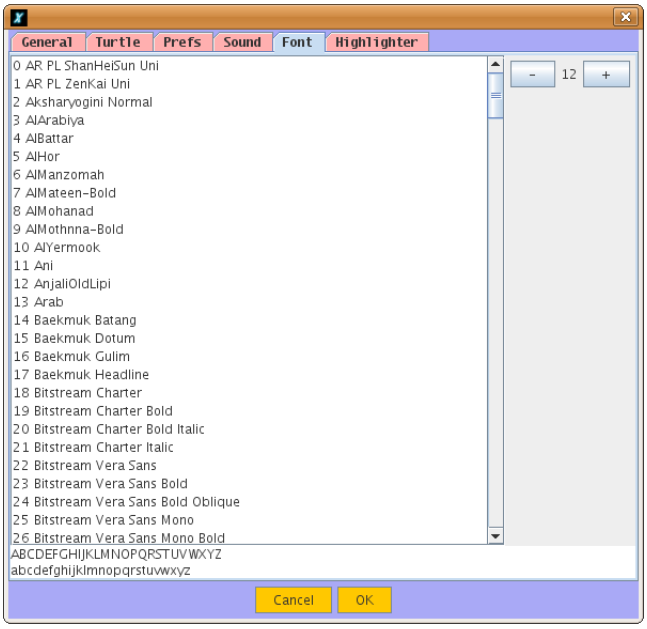
→ **Dimensione dell'area di disegno**: dimensioni in pixel. Attenzione, aree di disegno più grandi richiedono maggiore memoria.

→ **Memoria allocata a XLogo (MB)**: Aree di disegno grandi o programmi ricorsivi hanno bisogno di maggiore memoria di quella predefinita. La modifica di questo parametro richiede di riavviare XLogo. Attenzione, non aumentare troppo questo parametro poiché potrebbe rallentare considerevolmente il computer.

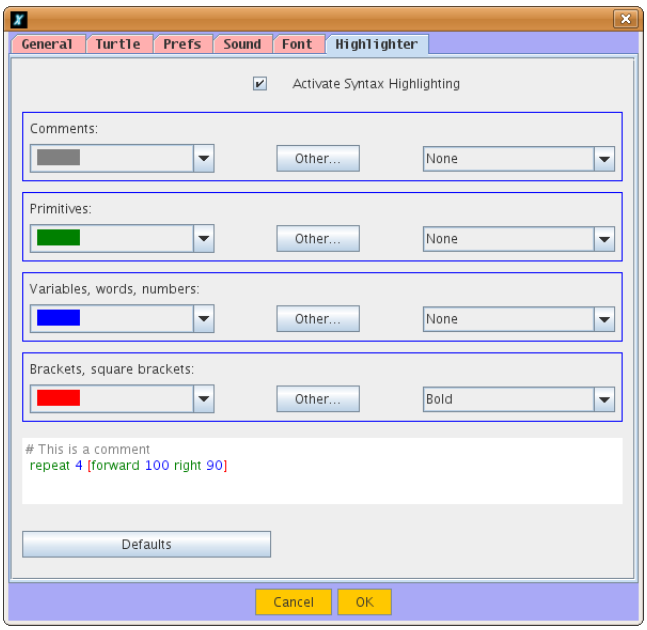
→ **Numero porta TCP**: Puoi modificare la porta preimpostata TCP per le comunicazioni di rete (cfr. pagina 129)



– **Tab Suono**: Puoi scegliere lo strumento musicale per l’interfaccia MIDI. Se l’elenco degli strumenti è vuoto dai uno sguardo alla sezione FAQ del manuale alla fine del manuale. Questa funzione può anche essere acceduta dalla primitiva `ImpostaStrumento`.



– **Tab Font**: Puoi scegliere il font per l’interfaccia.

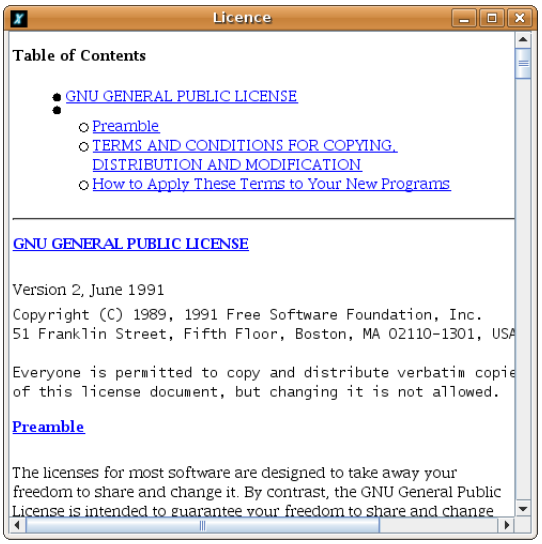


– **Tab Colorazione**: Puoi scegliere di attivare la colorazione sintattica e definire i colori che preferisci.

4.4 Menu “Aiuto”

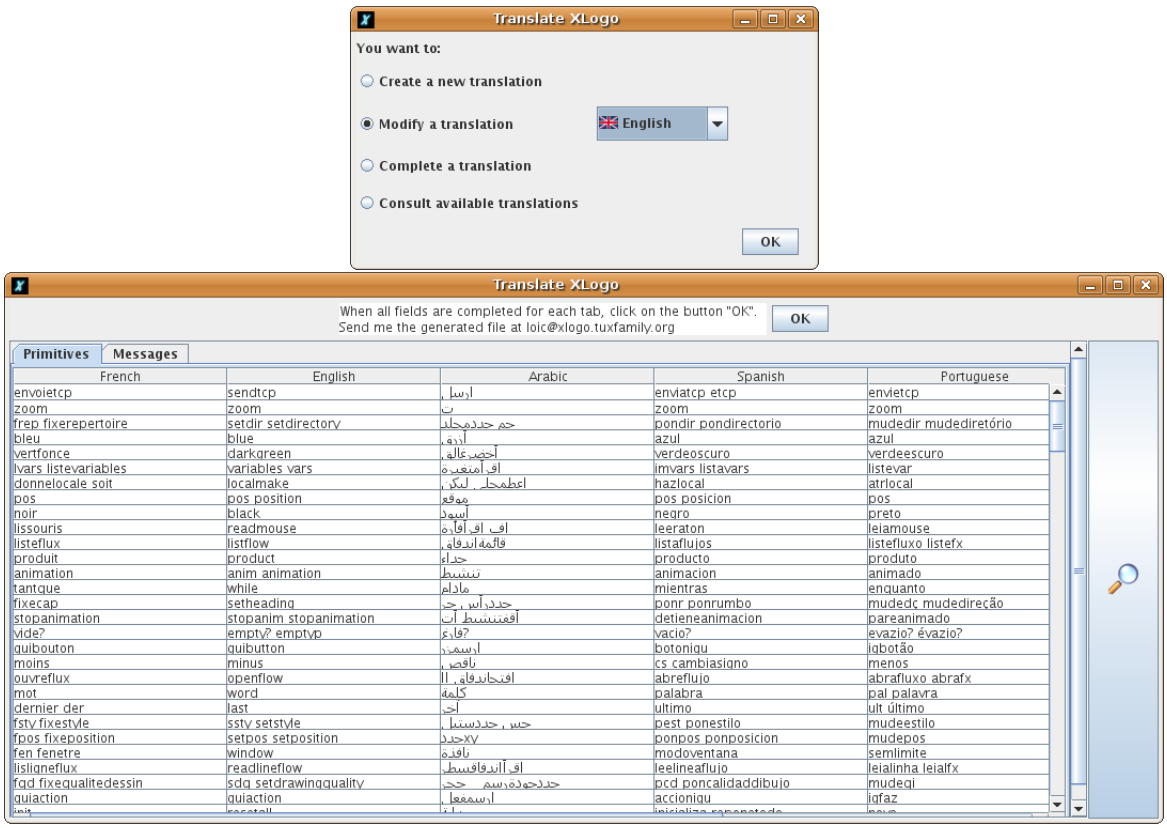
- **Aiuto→Manuale on-line**: Visualizza il manuale di riferimento per XLogo, accessibile su Internet.

- **Aiuto→Licenze**: visualizza la licenza GPL sotto la quale viene distribuito XLogo.



- **Aiuto→Traduzione della licenza**: visualizza una traduzione della licenza, anche se non ha carattere di ufficialità.

- **Aiuto→Traduci XLogo**: questa finestra di dialogo permette di consultare, modificare, completare le traduzioni di XLogodi tutte le lingue (primitive e messaggi).



Altrimenti puoi creare una traduzione per una nuova lingua. In tutti i casi mandami il file generato a `loic@xlogo.tuxfamily.org`

- **Menu – > Informazioni su XLogo**: la classica finestra e xlogo.tuxfamily.org per i tuoi segnalibri !! o:)

Capitolo 5

Convenzioni adottate da XLogo

Questo capitolo espone alcuni punti chiave circa il linguaggio LOGO e circa XLogo in modo specifico.

5.1 I comandi e loro elaborazione

Il linguaggio LOGO permette di invocare alcuni eventi tramite comandi interni. Questi comandi sono chiamati *primitive*. Ciascuna primitiva può accettare uno o più parametri che vengono chiamati *argomenti*. Per esempio, la primitiva `PulisciSchermo` non accetta alcun argomento mentre la primitiva `Somma` accetta due argomenti.
`Stampa Somma 2 3` restituirà 5.

Gli argomenti LOGO sono di tre tipi:

- **Numeri**: alcune primitive si aspettano un numero come argomenti: `Avanti 100` ne è un esempio.
- **Parole**: le parole sono contrassegnate dai doppi apici (") all'inizio del nome della parola stessa. Un esempio di una primitiva che accetta una parola come argomento è `Stampa`.

  ```
  Stampa "ciao
  ```

  Questo comando visualizza `ciao`. Se viene dimenticato il carattere " l'interprete logo restituirà un messaggio di errore. Per l'interprete LOGO `ciao` non rappresenta niente poiché non è un numero, una parola, un elenco o una procedura già definita.
- **Elenchi**: gli elenchi sono definiti entro parentesi quadre.

Nota: i Numeri sono trattati a volte come valori numerici (per esempio `Avanti 100`), altre volte come parole (per esempio `Stampa Primo 12` scrive `1`).

5.1.1 Le primitive generali

Molte primitive possiedono una forma generale ossia possono essere usate con un numero di argomenti indefinito. Queste primitive sono:

Stampa	Somma	Prodotto	o
e	Elenco	Frase	Parola

Per notificare l'interprete LOGO che queste primitive saranno usate nella loro forma generale occorre inscrivere il comando fra parentesi tonde, come nell'esempio seguente:

```
Stampa (Somma 1 2 3 4 5)
# 15

Stampa (Elenco [a b] 1 [c d])
```

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

Se (e 1=1 2=2 8=5+3) [Avanti 100 RuotaDestra 90]
```

5.2 Le procedure e le variabili

Oltre alle primitive è possibile definire comandi personalizzati. Questi comandi sono chiamati *procedure*. Le procedure sono definite mediante le primitive `Per ...Fine`. Il blocco di comandi che la procedura eseguirà viene posto all'interno delle due precedenti primitive. Esse possono essere create utilizzando l'editor interno di XLogo. Ecco un breve esempio:

```
Per quadrato
  Ripeti 4 [Avanti 100 RuotaDestra 90]
Fine
```

Come le primitive anche le procedure possono trarre vantaggio degli argomenti. Per passare argomenti alle procedure si utilizzano le variabili. Una variabile è una parola alla quale si può associare (assegnare in termini informatici) un valore. Le variabili sono quindi una sorta di contenitori che possono essere riempiti di valori a nostro piacimento. Ecco un semplice esempio:

```
Per totale :a :b
  Stampa Somma :a :b
Fine
```

Invocando `totale 2 3` otterremo 5 come risultato.

5.3 Il carattere speciale \

Il carattere speciale \ (barra rovesciata) permette la creazione di parole contenenti simboli vuoti o di particolare significato come l'andare a capo. Se \n è usato la frase salta alla linea successiva e \␣ seguito da uno spazio permette di inserire uno spazio in una parola. Per esempio:

```
  Stampa "xlogo\ xlogo
  # xlogo xlogo
  Stampa "xlogo\nxlogo
  # xlogo
  # xlogo
```

Per inserire la barra rovesciata in una parola occorre scrivere \\.
Allo stesso modo, per includere quei caratteri a cui XLogo assegna particolari significati (( ) [ ] #) in una parola occorre prefissarli con la barra rovesciata.
Tutti i simboli preceduti da \ sono ignorati. Questo è particolarmente importante nello scrivere i nomi dei file.. Per esempio per impostare il percorso attuale a `c:\Miei Documenti`:

```
  ImpDir "c:\\Miei\ Documenti.
```

Da notare l'uso di \␣ per notificare all'interprete LOGO dell'esistenza dello spazio fra Miei e Documenti. Se si omette la doppia barra rovesciata il percorso diventa `c:Miei Documenti` e l'interprete restituirà un messaggio di errore circa l'inesistenza di tale percorso.

5.4 Maiuscole e minuscole

XLogo non fa differenza tra maiuscole e minuscole nei nomi delle procedure e delle primitive. Quindi la procedura `quadrato` definita precedentemente può essere invocata come `QUADRATO`, `Quadrato`, `qUadrato` e così via, l'interprete LOGO la eseguirà in ogni caso. Al contrario XLogo differenzia le maiuscole dalle minuscole nel caso degli elenchi e delle parole, per esempio:

```
Stampa "Ciao
# Ciao (la maiuscola iniziale viene conservata)
```

5.5 Gli operatori e la sintassi

Ci sono due modi per scrivere taluni comandi. Per esempio per sommare 4 e 7 si può usare la primitiva `Somma` che richiede due argomenti: `Somma 4 7`, o si può usare l'operatore "+": `4+7`. Entrambi i modi hanno il medesimo effetto. La seguente tabella illustra la relazione tra operatori e primitive:

Somma	Differenza	Prodotto	Quoziente
+	-	*	/
o	e	uguale?	
| (ALT GR+6)	&	=	

Ci sono due altri operatori che non sono associati ad alcuna primitiva:

- l'operatore "Minore o uguale": `<=`
- l'operatore "Maggiore o uguale": `>=`

Nota: I due operatori | e & sono specifici di XLogo. Non esistono nelle versioni tradizionali di LOGO. Qualche esempio di impiego:

```
Stampa 3+4=7-1        # falso
Stampa 3=4 | 7<=49/7  # vero
Stampa 3=4 & 7=49/7   # falso
```

Capitolo 6

Le primitive di base

Livello: principiante

Per spostare la tartaruga nell'area di disegno si usano comandi predefiniti chiamati "primitive". In questo capito scopriremo le primitive di base che permettono di pilotare la tartaruga nell'area di disegno.

6.1 Le primitive indispensabili

- `Avanti o Av numero passi` `Avanti 50`
  Sposta la tartaruga in avanti di un certo numero di passi, nella direzione nella quale è disposta.
- `Indietro o In numero passi` `In 100`
  Sposta la tartaruga indietro di un certo numero di passi, nella direzione nella quale è disposta.
- `RuotaDestra o DX angolo` `DX 90`
  Ruota la tartaruga a destra di un certo numero di gradi, in relazione alla direzione nella quale è disposta.
- `RuotaSinistra o SX angolo` `SX 90`
  Ruota la tartaruga a sinistra di un certo numero di gradi, in relazione alla direzione nella quale è disposta.
- `PulisciSchermo o PS` `PS`
  Pulisce l'area di disegno.
- `MostraTartaruga o MT` `MT`
  Rende visibile la tartaruga.
- `NascondiTartaruga o NT` `NT`
  Rende invisibile la tartaruga (disegnare diventa più veloce).
- `PennaSù o PS` `PS`
  La tartaruga non lascia traccia quando si sposta.
- `PennaGiù o PG` `PG`
  La tartaruga lascia una traccia muovendosi (situazione predefinita).
- `Ripeti numero intero elenco` `Ripeti 5[Avanti 50 DX 45]`
  Ripete un elenco di istruzioni per un numero di volte.

6.2 Cominciamo a disegnare

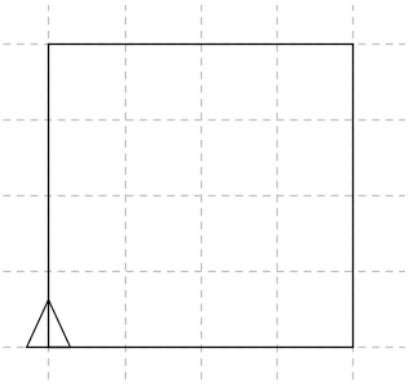
In questa parte impareremo a disegnare un quadrato, un triangolo equilatero ed un qualsiasi altro poligono regolare.... Un poligono regolare è una figura geometrica avente tutti i lati e tutti gli angoli congruenti fra

loro (cioè uguali). La somma degli angoli interni è pari a

$$(n-2) \times 180°$$

dove n è il numero dei lati del poligono regolare. La somma degli angoli esterni è sempre pari a 360°.

6.2.1 Il quadrato



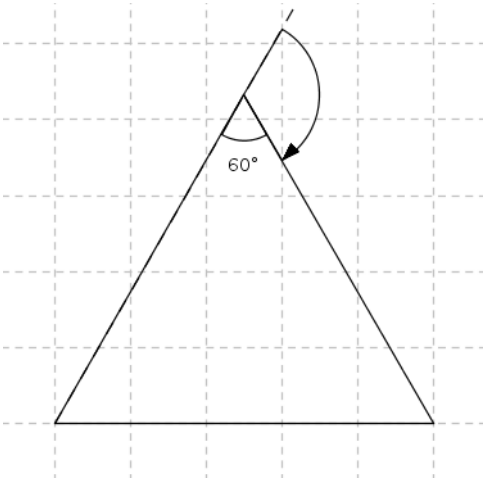
Per disegnare questo quadrato di 200 passi di lato, occorre scrivere:

```
Av 200 DX 90 Av 200 DX 90 Av 200 DX 90 Av 200 DX 90
```

Possiamo notare che ripetiamo il disegno di ciascun lato per quattro volte, possiamo quindi sintetizzare il programma così:

```
Ripeti 4[Av 200 DX 90]
```

6.2.2 Il triangolo equilatero



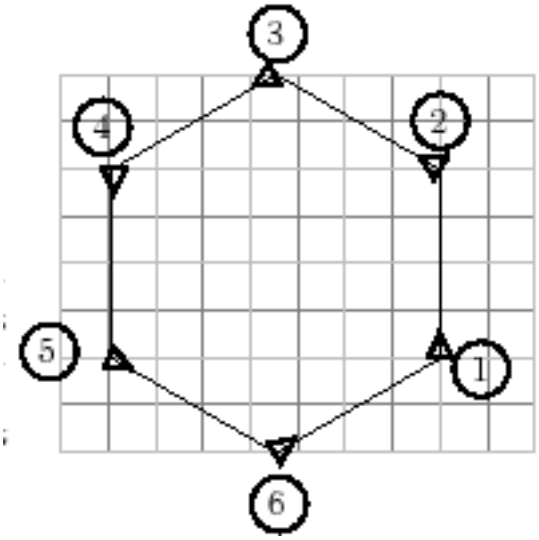
Adesso impariamo a disegnare questo triangolo equilatero di lato 150 passi. Il programma avrà questa forma generica che abbiamo imparato a proposito del quadrato:

```
Ripeti 3[Av 150 DX ....]
```

Dobbiamo determinare l'angolo di rotazione della tartaruga. In un triangolo equilatero i tre angoli interni sono uguali fra loro e quindi, visto che la somma degli angoli interni di un triangolo è 180°, ciascun angolo sarà pari a $\frac{180°}{3} = 60°$. Ricordiamoci, guardando la figura, che l'angolo di rotazione della tartaruga è l'angolo esterno, non quello interno al triangolo. L'angolo di rotazione sarà quindi $180° - 60° = 120°$. Il comando da fornire sarà quindi

```
Ripeti 3[Av 150 DX 120]
```

6.2.3 L'esagono




```
Ripeti 6[Av 80 DX ....]
```

Occorre determinare anche qui l'angolo di rotazione della tartaruga. Riflettiamo sul fatto che la tartaruga, una volta completato il disegno di tutti i lati, sarà tornata nella posizione di partenza e con la direzione originaria. Questo significa che avrà compiuto una rotazione totale di 360°, in sei passi (tanti quanti sono i lati). Quindi ad ogni passo avrà compiuto una rotazione pari a $\frac{360°}{6} = 60°$. Il comando da fornire sarà quindi

```
Ripeti 6[Av 80 DX 60]
```

6.2.4 Un poligono regolare generico

Nei fatti, il ragionamento che abbiamo applicato per disegnare l'esagono, è valido per qualsiasi poligono regolare, visto che la tartaruga dovrà ruotare di 360° in passi successivi uguali fra loro. Se indichiamo con n il numero dei lati la formula per calcolare l'angolo di rotazione da compiere per ciascun passo sarà pari a $\frac{360°}{n}$. Per esempio

- Per disegnare un triangolo equilatero di lato 100

  ```
  Ripeti 3[Av 100 DX 120]    # (360:3=120)
  ```

- Per disegnare un pentagono regolare di lato 50

  ```
  Ripeti 5[Av 50 DX 72]    # (360:5=72)
  ```

- Per disegnare un ennagono regolare di lato 20

  ```
  Ripeti 9[Av 20 DX 40]    # (360:9=40)
  ```

- Per disegnare un . . . 360-gono regolare di lato 2:

  ```
  Ripeti 360[Av 2 DX 1]    # (360:360=1)
  ```

  Questa figura è molto vicina ad una circonferenza!

- Per disegnare un ettagono di lato 120:

  ```
  Ripeti 7 [Av 120 DX 360/7]  # quanto fa?
  ```

6.3 Definire una procedura

Poiché non vogliamo riscrivere ogni volta le stesse istruzioni per disegnare un quadrato, un triangolo . . . è meglio salvarle in "procedure". Per definire una procedura, apri l'editor. Una procedura comincia con la primitiva `Per` e termina con la primitiva `Fine`. Per esempio per inserire le istruzioni per disegnare il quadrato in una procedura:

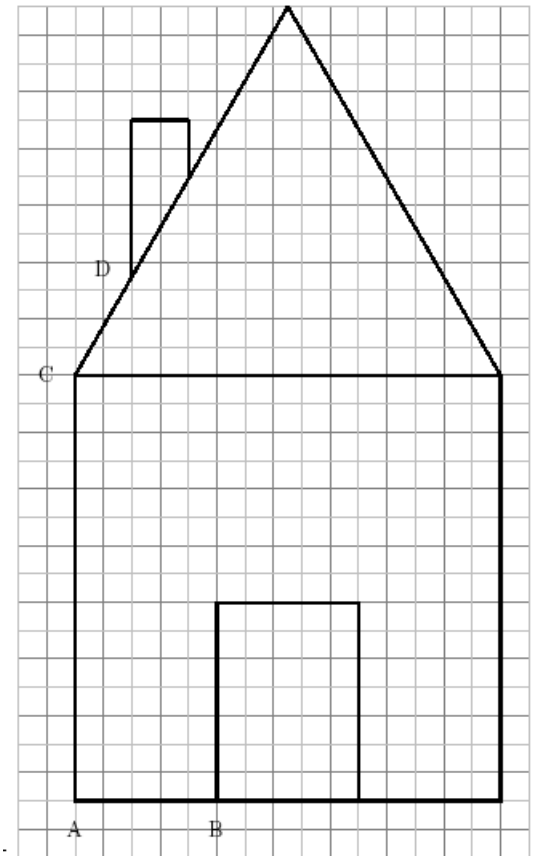
```
Per Quadrato
  Ripeti 4[Av 100 DX 90]
Fine
```

Quindi chiudiamo l'editor cliccando sul bottone che raffigura la tartaruga. La procedura verrà salvata. Ora scrivendo semplicemente `Quadrato` verrà disegnato un quadrato.

6.4 Qualche esercizio

Ciascun quadretto nelle figura ha lato pari a 10 punti. Prova a disegnare questa figura usando otto procedure:

- Una procedura chiamata `Quadrato` che disegna il quadrato principale della casa.
- Una procedura chiamata `Triangolo` che disegna il tetto come un triangolo equilatero.
- Una procedura chiamata `Porta` che disegna una porta rettangolare.
- Una procedura chiamata `Camino` che disegna il camino.
- Una procedura chiamata `SpostaAB` che sposta la tartaruga dal punto A al punto B.
- Una procedura chiamata `SpostaBC` che sposta la tartaruga dal punto B al punto C (attento, devi alzare la penna della tartaruga).
- Una procedura chiamata `SpostaCD` che sposta la tartaruga dal punto C al punto D.
- Una procedura generale chiamata `casa` che disegna l'intera casa utilizzando tutte le precedenti procedure.


D
C
A
B

Capitolo 7

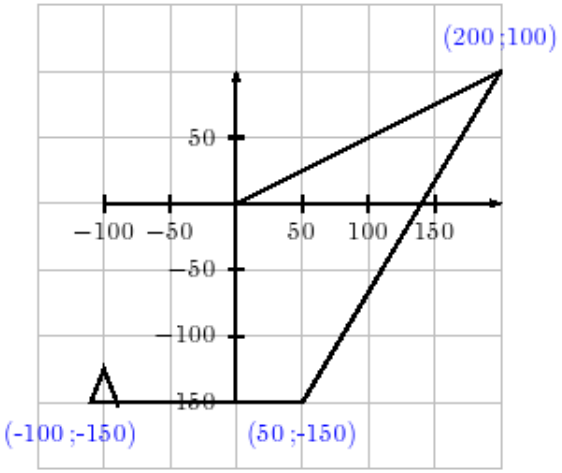
Utilizzare le coordinate

Livello: Principiante

7.1 Presentazione

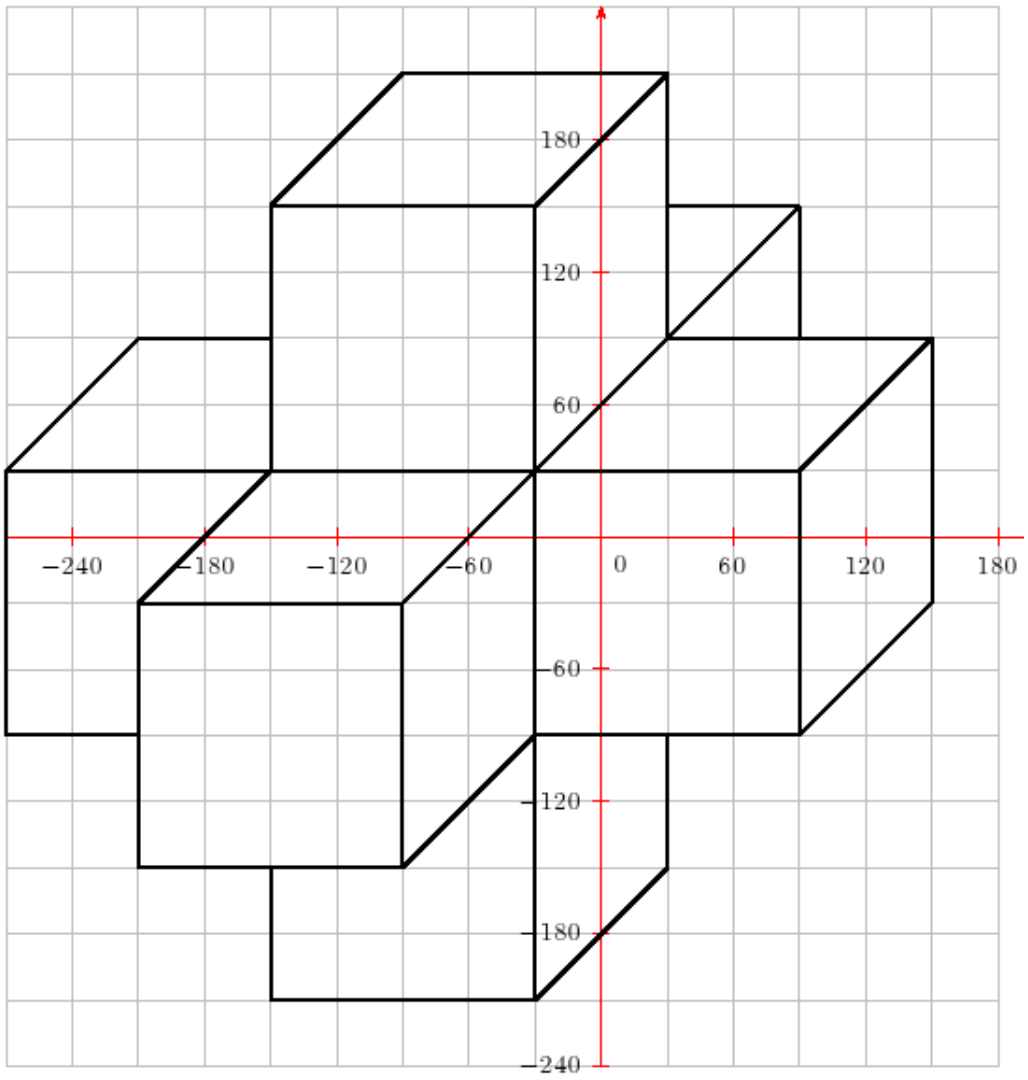
In questo capitolo scopriremo la primitiva `ImpostaPosizione, ImpPos`. L'area di disegno ha due assi che permettono di determinare ciascun punto utilizzando il sistema di coordinate cartesiano. L'origine degli assi è il centro dell'area di disegno.
Il formato della primitiva è `ImpostaPosizione elenco` come in `ImpPos [100 -250]`. La tartaruga si sposta al punto di coordinate X 100 e Y -250. Le coordinate sono racchiuse in parentesi quadre perché la primitiva richiede un elenco. Ricorda che la tartaruga ha una penna che può toccare l'area di disegno o essere alzata. Quando la penna è abbassata la tartaruga disegnerà una linea ogni volta che se ne imposterà la posizione.
Un piccolo esempio:

```
PulisciSchermo ImpPos [200 100] ImpPos [50 -150] ImpPos [-100 -150]
```



7.2 Esercizio

Prova a disegnare questa figura usando solo le seguenti primitive: `ImpPos`, `PScÅ`, `PS`, `PG`.



Capitolo 8

Le variabili

Livello: Principiante

Affrontiamo le variabili con esempio pratico. Qualche volta è necessario disegnare una figura in scala diversa. Per esempio se vogliamo disegnare un quadrato di lato pari a 100 passi, uno di lato pari a 200 ed un altro di lato pari a 50 dobbiamo scrivere tre diverse procedure, una per ciascun quadrato.

```
Per quadrato1
  Ripeti 4 [Avanti 100 RuotaDestra 90]
Fine
Per quadrato2
  Ripeti 4 [Avanti 200 RuotaDestra 90]
Fine
Per quadrato3
  Ripeti 4 [Avanti 50 RuotaDestra 90]
Fine
```

Capiamo immediatamente che sarebbe più semplice definire una sola procedura che riesca a disegnare tutti i tre quadrati e quadrati di qualsiasi lato. Per fare questo dobbiamo riuscire a dire alla procedura la lunghezza del lato del quadrato che vogliamo disegnare. Riusciamo a farlo grazie alle variabili, per esempio `quadrato 200` disegna un quadrato di lato 200 passi, `quadrato 100` disegna un quadrato di lato 100 e così via. Ma per fare ciò dobbiamo scrivere una procedura che sappia che vogliamo dirle quanto deve essere lungo il lato.

8.1 Esempi

La nostra procedura per disegnare il quadrato di lato 100 era:

```
Per quadrato1
  Ripeti 4 [Avanti 100 RuotaDestra 90]
Fine
```

Dobbiamo semplicemente modificare questa procedura in due punti:

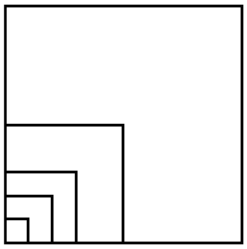
1. Aggiungiamo `:lato` alla fine della linea di definizione della procedura (riga 1). Questo dice all'interprete LOGO che la procedura si aspetta che le si dica la lunghezza del lato. `:lato` è chiamato l'argomento della procedura e definisce una variabile dello stesso nome che verrà riempita del valore della lunghezza del lato al momento di invocare la procedura stessa.

2. Sostituiamo la lunghezza del lato 100 con il nome della variabile `:lato`.

Otteniamo:

```
Per quadrato1 :lato
  Ripeti 4 [Avanti :lato RuotaDestra 90]
Fine
```

Quindi scrivendo `quadrato1 100 quadrato1 50 quadrato1 30 quadrato1 20 quadrato1 10` disegneremo la seguente figura:



8.2 Disegnare un rettangolo delle dimensioni volute

Definiamo una procedura chiamata `rec` che si aspetta due argomenti che rappresentano del dimensioni dei lati del rettangolo. Quindi `rec 200 100` dovrà disegnare un rettangolo di altezza 200 e lunghezza 100.

```
Per rec :altezza :lunghezza
  Ripeti 2 [Avanti :altezza DX 90 Avanti :lunghezza DX 90]
end
```

Alcuni disegni:

```
rec 200 100 rec 100 300 rec 50 150 rec 1 20 rec 100 2
```

Se invochiamo la procedura `rec` con solo un numero invece di due l'interprete LOGO fermerà il disegno e scriverà un messaggio di errore indicando che la procedura aspetta un secondo argomento.

8.3 Disegnare in scale differenti

Abbiamo imparato come disegnare un quadrato ed un rettangolo con lati diversi. Ora torniamo all'esempio della casa di pagina 32 per modificare il programma per disegnare case di qualsiasi dimensione.
L'obiettivo è di invocare la procedura `casa` con un argomento che possa disegnare case più piccole o più grandi (la scala).

- `casa 1` disegnerà la casa nelle dimensioni reali dei quadretti.
- `casa 0.5` disegnerà una casa grande la metà di quella originale.
- `casa 2` disegnerà una casa grande il doppio di quella originale.

Vediamo come modificare le singole procedure per accettare il fattore di scala. Originariamente la procedura quadrato era:

```
Per quadrato
  Ripeti 4 [Avanti 150 RuotaDestra 90]
end
```

Tutte le dimensioni devono essere moltiplicate per la scala. Quindi la procedura del quadrato diventa:

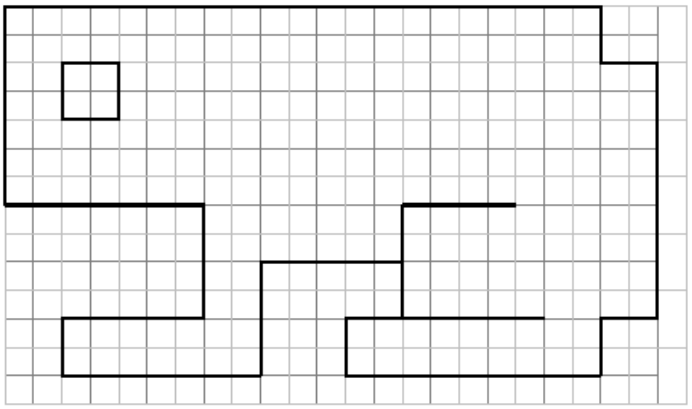
```
Per quadrato :scala
  Ripeti 4 [Avanti 150*scala RuotaDestra 90]
end
```

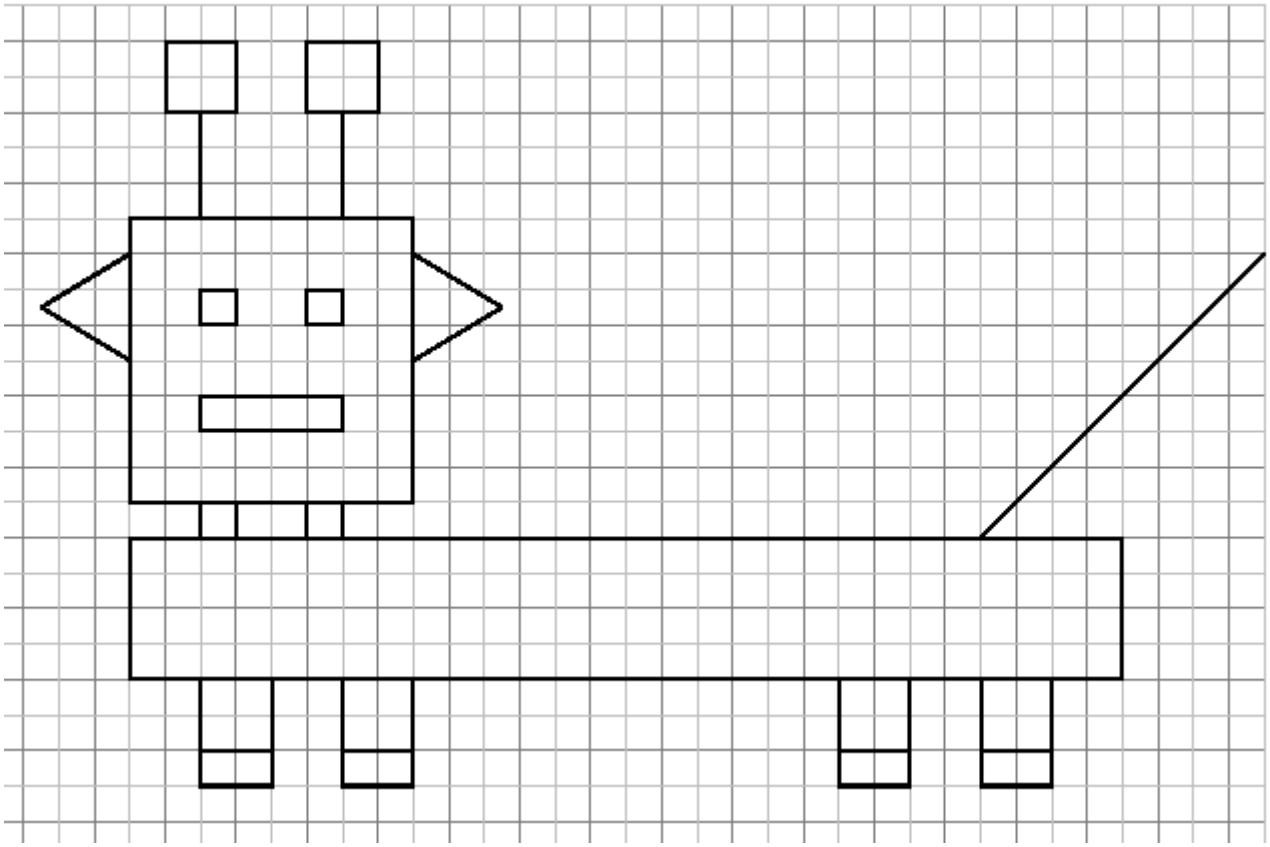
Quindi se invochiamo `quadrato 2`, il quadrato avrà i lati di lunghezza $150 \times 2 = 300$. Le proporzioni sono rispettate. Infatti vediamo che dobbiamo solo modificare le procedure sostituendo le lunghezze secondo questa regola:
`Av 70` diventa `Av 70*:scala`
`Av 45` diventa `Av 45*:scala`

```
per Quadrato :scala
  Ripeti 4[Avanti 150*:scala  RuotaDestra 90]
fine

per Triangolo :scala
  Ripeti 3[Avanti 150*:scala RuotaDestra 120]
fine

per Porta :scala
  Ripeti 2[Avanti 70*:scala RuotaDestra 90
  Avanti 50*:scala RuotaDestra 90]
fine

per Camino :scala
  Avanti 55*:scala RuotaDestra 90 Avanti 20*:scala
  RuotaDestra 90 Avanti 20*:scala
fine

per SpostaAB :scala
  RuotaDestra 90 Avanti 50*:scala RuotaSinistra 90
fine

per SpostaBC :scala
  RuotaSinistra 90 Avanti 50*:scala RuotaDestra 90
  Avanti 150*:scala RuotaDestra 30
fine

per SpostaCD :scala
  PennaSu RuotaDestra 60 Avanti 20*:scala RuotaSinistra 90
  Avanti 35*:scala PennaGiu
fine
```

8.4 Esercizio

Prova a generare i seguenti disegni a diverse scale.





Capitolo 9

La ricorsività

Livello: Medio

La programmazione in LOGO spesso usa una tecnica chiamata ricorsività. In questo capitolo esploreremo la ricorsività con semplici esempi e disegneremo alla fine una curva frattale chiamata "fiocco di neve di Van Koch".
Prima di tutto:

Una procedura è ricorsiva se invoca se stessa

9.1 Nell'area di disegno

9.1.1 Un primo semplice esempio

```
Per ex1
  DX 1
  ex1
Fine
```

Questa procedura è ricorsiva perché la procedura `ex1` è invocata nell'ultima riga della medesima procedura. Durante l'esecuzione osserviamo che la tartaruga ruota su stessa verso destra all'infinito. Per interrompere il programma dobbiamo cliccare il bottone di STOP.

9.1.2 Un secondo esempio

Per il secondo esempio impariamo tre nuove primitive:

- `Aspetta numero` Aspetta 60
  Mette in pausa il programma per *numero*/60 secondi.
  Per esempio `Aspetta 120` metterà in pausa il programma per 2 secondi.
- `CancellaPenna` CancellaPenna
  Quando la tartaruga si sposta, cancella tutto ciò che incontra invece di disegnare.
- `PennaDisegno` PennaDisegno
  Ritorna alla modalità classica di disegno.

```
Per ex2
  Av 200 CancellaPenna Aspetta 60
  In 200 PennaDisegno DX 6
  ex2
Fine
```

Il programma ripete la stessa procedura ogni secondo ottenendo i secondi di un orologio.

9.2 Nell'area dello storico dei comandi

9.2.1 Un primo semplice esempio

La primitiva `Stampa` visualizza del testo nell'area dello storico dei comandi. `Stampa` si aspetta un argomento, un elenco o una parola.
Per esempio: `Stampa "hello Stampa [Essere o non essere]` (non dimenticare le virgolette "quando vuoi scrivere solo una parola).

```
Per ex3 :n
  Stampa :n
  ex3 :n+1
Fine
```

Invoca la procedura `ex3 0` e ferma il programma con il bottone di STOP.
Come esercizio modifica il programma per visualizzare i numeri con un intervallo di 2.

Vogliamo ora visualizzare tutti i numeri interi maggiori di 100 che sono divisibili per 5. Dobbiamo solo modificare il programma così:

```
Per ex3 :n
  Stampa :n
  ex3 :n+5
end
```

e quindi invocare: `ex3 100`

9.2.2 Uscita dalla ricorsione

Tutti gli esempi precedenti vengono eseguiti all'infinito consumando piano piano tutta la memoria disponibile fino a quando XLogo fermerà il programma. Normalmente, però, la ricorsione viene fermata dal programma stesso quando si verifica una determinata condizione. Esempi di condizioni sono i seguenti esempi:

```
Se 2+1=3 [Stampa [è vero]]
Se 2+1=4 [Stampa [è vero]][print [è falso]]
Se 2+5=7 [Stampa "vero][Stampa "falso]
```

Se non ti è chiara la sintassi delle condizioni vai alla pagina della primitiva `Se`. Il seguente esempio modifica il terzo esempio per interrompe la ricorsione al 100-esimo numero.

```
Per ex3 :n
  Se :n=100 [Ferma]
  Stampa :n
  ex3 :n+1
Fine
```

Invoca il programma con `ex3 0`.
Come esercizio puoi modificare il programma per visualizzare numeri interi tra 55 e 350 che sono divisibili per 11.

9.3 L'esempio di un frattale, il fiocco di neve di Van Koch

Utilizzando la ricorsione è molto semplice generare in Logo alcune semplici curve chiamate frattali in matematica.
Questi sono i primi passi per creare la linea spezzata di Van Koch:

Come primo passo si disegna un segmento di una data lunghezza.
Come secondo passo:

1. il segmento viene diviso in tre parti uguali.
2. un triangolo equilatero viene disegnato sul segmento centrale.
3. infine il segmento centrale viene cancellato.

A questo punto il segmento originario risulterà spezzato in quattro segmenti di lunghezza pari ad $\frac{1}{3}$ di quello originario (terza figura). Il secondo passo viene quindi ripetuto su ciascuno dei quattro segmenti originando 16 segmenti di lunghezza pari ad $\frac{1}{3}$ dei precedenti, ossia $\frac{1}{3} \cdot \frac{1}{3} = \frac{1}{9}$ (quarta figura). Ad ogni passo la lunghezza dei segmenti si riduce di un fattore 3.
Importante Abbiamo scovato la natura ricorsiva dei frattali!
Vediamo come impostare concettualmente il programma Logo. Chiamiamo $L_{n,\ell}$ il segmento di lunghezza ℓ, corrispondente al passo n.
Per disegnare il segmento:

1. Disegniamo $L_{n-1,\ell/3}$
2. Ruotiamo a sinistra di 60°
3. Disegniamo $L_{n-1,\ell/3}$
4. Ruotiamo a destra di 120°
5. Disegniamo $L_{n-1,\ell/3}$
6. Ruotiamo a sinistra di 60°
7. Disegniamo $L_{n-1,\ell/3}$

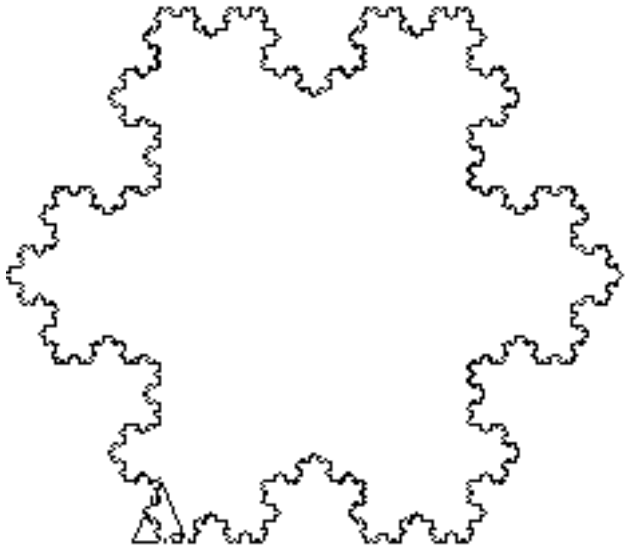
In Logo, il programma è molto semplice:

```
  # :l contiene la lunghezza del segmento
  # :n contiene il numero del passo
to linea :l :n
  Se :n=0 [Av :l] [
  linea :l/3 :n-1 SX 60 linea :l/3 :n-1 DX 120 linea :l/3 :n-1 SX 60 linea :l/3 :n-1
  ]
end
```

Il programma si invoca come `linea 50 3`, dove il primo argomento è la lunghezza del segmento originario ed il secondo argomento è il numero di ricorsioni da eseguire. Incredibile il potere della ricorsione!
Se disegniamo un triangolo equilatero con tre linee di Van Koch otteniamo uno stupendo fiocco di neve di Van Koch.

```
  # :l lunghezza del lato
Per fioccoNeve :l :p
  Ripeti 3[linea :l :p DX 120]
Fine
```

Il programma si invoca per esempio con: `fioccoNeve 200 6`



9.4 Ricorsione con le parole

Leggi pagina 89 per capire come impiegare le primitive `Parola`, `Ultimo`, and `EccettoUltimo, EU`.

9.4.1 Leggere al contrario le parole

Questa è una procedura ricorsiva che inverte i caratteri di una parola.

```
Per invertiParola :m
  Se vuoto? :m [output "]
  output parola ultimo :m invertiParola eccettoultimo :m
Fine
```

Si invoca con `Stampa invertiParola "Logo`.

9.4.2 I palindromi

Un palindromo è una parola o una frase che si può leggere in entrambi i sensi (esempi: I treni inerti, Etna gigante ...). Aggiungiamo una semplice verifica all'esempio precedente per capire se la parola o la frase è palindroma.

```
# Verifica se la parola :m e' palindroma
Per palindromo :m
  Se :m=invertiParola :m [output vero] [output falso]
Fine
```

9.4.3 I numeri palindromi

Anche i numeri possono essere palindromi, per esempio 4884. Infine, un programma per cercare numeri palindromi a partire da un qualsiasi numero (Grazie Olivier SC):

```
Per numeroPalindromo :n
  Se palindromo :n [Stampa :n Ferma]
  Stampa (elenco :n "Piu' invertiParola :n "uguale somma :n invertiParola :n)
  numeroPalindromo :n + invertiParola :n
end
```

Si invoca con `numeroPalindromo 78`:

```
78 Più 87 uguale 165
165 Più 561 uguale 726
726 Più 627 uguale 1353
1353 Più 3531 uguale 4884
4884
```

9.5 Calcolo di un numero fattoriale

Il fattoriale di un numero n è il prodotto dei primi n numeri interi positivi. Per esempio il fattoriale di 5 è definito come:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

In termini matematici se n è un intero positivo: $n! = n \times (n-1)!$. Questa relazione spiega la natura ricorsiva del programma fattoriale:

```
Per fattoriale :n
  Se :n=0[output 1][output :n*fattoriale :n-1]
Fine
```

Si invoca con `Stampa fattoriale 5`.

9.6 Calcolo del pi greco per approssimazione

Possiamo approssimare il numero pi greco utilizzando la formula:

$$\pi \approx 2^k \sqrt{2 - \sqrt{2 + \sqrt{2 + \ldots \sqrt{2 + \sqrt{2}}}}}$$

dove k è il numero di radici quadrate. Maggiore è k, migliore risulterà l'approssimazione di pi greco.

La formula contiene l'espressione ricorsiva $2 + \sqrt{2 + \ldots \sqrt{2 + \sqrt{2}}}$, quindi ecco il programma:

```
# k e' il numero di radici quadrate
Per approxpi :k
  Stampa [Approssimazione di pi greco:]  Stampa (potenza 2 :k) * RadQ (2- RadQ (calc :k-2))
  Stampa "--------------------------
  Stampa [Esatto pi greco:]  Stampa pi
Fine

Per calc :p
  Se :p=0 [output 2][output 2+RadQ calc :p-1]
Fine
```

Si invoca con `approxpi 10`.

Con 10 approssimazioni troviamo i primi 5 decimali! Se vuoi trovare più decimali devi aumentare la precisione aumentando il numero di decimali con la primitiva `ImpostaDecimali`.

```
ImpostaDecimali 100
approxpi 100
```

E ora abbiamo 39 decimali esatti...

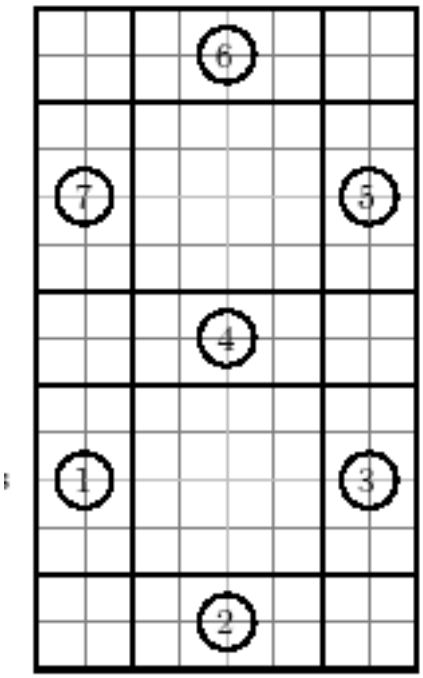
Capitolo 10

Creare una animazione

Livello: Medio

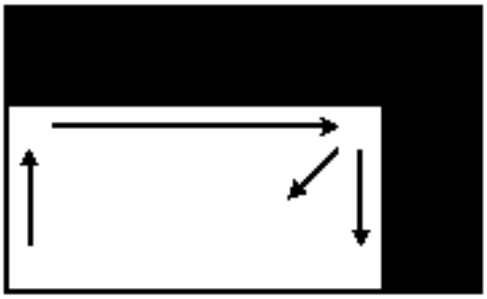
Questo capitolo presenta due differenti temi con l'obiettivo di creare animazioni in XLogo.

10.1 Le cifre della calcolatrice



Questo tema è basato sul fatto che tutte le cifre delle calcolatrici potrebbero essere disegnate secondo lo schema qui sopra. Per esempio per disegnare la cifra 4 illuminiamo i rettangoli 3,4,5,7. Per disegnare la cifra 8, illuminiamo tutti i rettangoli. Per disegnare la cifra 3 illuminiamo i rettangoli 2,3,4,5,6.

10.1.1 Riempire un rettangolo



Se vogliamo disegnare un rettangolo colorato di dimensioni 100 per 200 una prima possibilità potrebbe essere di disegnare un rettangolo vuoto di 100x200, quindi un altro rettangolo di 99x199 e così via... fino a che il rettangolo sarà completamente pieno.
Cominciamo definendo un rettangolo con due variabili corrispondenti alla base ed alla altezza.

```
Per rec :h :w
  Ripeti 2[Av :h DX 90 Av :w DX 90]
Fine
```

Per riempire un rettangolo dobbiamo scrivere:
`rec 100 200 rec 99 199 rec 98 198 .....  rec 1 101`

Definiamo quindi una procedura per questo rettangolo riempito.

```
Per rettangolo :h :w
  rec :h :w
  rettangolo :h-1 :w-1
Fine
```

Verifichiamo con `rettangolo 100 200` e capiamo che c'è un problema. La procedura non si ferma dopo aver riempito tutto il rettangolo. Dobbiamo inserire un comando di uscita che capisca quando la base o l'altezza diventano 0. Quando si realizza questa condizione il programma si deve fermare con la primitiva `Ferma`.

```
Per rettangolo :h :w
  Se o :h=0 :w=0 [Ferma]
  rec :h :w
  rettangolo :h-1 :w-1
Fine
```

Invece di usare la primitiva `o` è possibile usare il simbolo "|", la linea diventa:

```
Se :h=0 | :w=0 [Ferma]
```

10.1.2 Il programma

Dobbiamo riutilizzare il precedente rettangolo riempito. Supponiamo che la tartaruga comincia dall'angolo in basso a sinistra. Definiamo una procedura chiamata `numero` che accetta 7 argomenti: `:a`, `:b`, `:c`, `:d`, `:e`, `:f`, `:g`. Quando `:a` è uguale a 1 disegniamo il rettangolo 1. Se `:a` è uguale a 0 non disegniamo alcun rettangolo. Questa è la idea principale.
Il programma:

```
Per numero :a :b :c :d :e :f :g
# disegniamo il rettangolo 1
Se :a=1 [rettangolo 160 40]
# disegniamo il rettangolo 2
Se :b=1 [rettangolo 40 160]
```

```
PennaSu DX 90 Avanti 120 SX 90 PennaGiu
# disegniamo il rettangolo 3
Se :c=1 [rettangolo 160 40]
PennaSu Avanti 120 PennaGiu
# disegniamo il rettangolo 5
Se :e=1 [rettangolo 160 40]
# disegniamo il rettangolo 4
SX 90 PennaSu Indietro 40 PennaGiu
Se :d=1 [rettangolo 160 40]
# disegniamo il rettangolo 6
DX 90 PennaSu Avanti 120 SX 90 PennaGiu
Se :f=1 [rettangolo 160 40]
# disegniamo il rettangolo 7
PennaSu Avanti 120 SX 90 Indietro 40 PennaGiu
Se :g=1 [rettangolo 160 40]
end
```

10.1.3 Creare l'animazione

In questa parte definiamo il conteggio a rovescio da 9 a 0.

```
Per countd
  PulisciSchermo NascondiTartaruga numero 0 1 1 1 1 1 1 Aspetta 60
  PulisciSchermo NascondiTartaruga numero 1 1 1 1 1 1 1 Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 0 1 0 1 1 0 Aspetta 60
  PulisciSchermo NascondiTartaruga numero 1 1 1 1 0 1 1 Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 1 1 1 0 1 1 Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 0 1 1 1 0 1 Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 1 1 1 1 1 0 Aspetta 60
  PulisciSchermo NascondiTartaruga numero 1 1 0 1 1 1 0 Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 0 1 0 1 0 0 Aspetta 60
  PulisciSchermo NascondiTartaruga numero 1 1 1 0 1 1 1 Aspetta 60
Fine
```

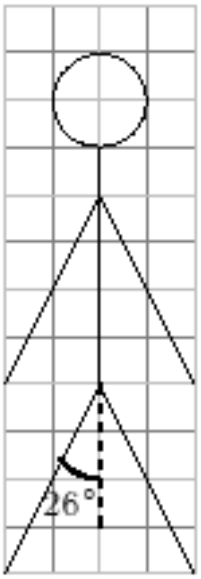
C'è un piccolo problema, un effetto di sfarfallamento durante il disegno di ciascuna cifra. Per rendere l'animazione più fluida useremo la modalità di animazione mediante tre primitive:

- `Animazione` permette di entrare nella modalità animazione. La tartaruga smette di disegnare sull'area di disegno ma si ricorda di tutti i cambiamenti. Per visualizzare l'immagine è necessario usare la primitiva `Ridisegna`.
- `FermaAnimazione` restituisce la modalità classica.

Modifichiamo il programma in questo modo:

```
Per countd
  Animazione
  PulisciSchermo NascondiTartaruga numero 0 1 1 1 1 1 1 Ridisegna Aspetta 60
  PulisciSchermo NascondiTartaruga numero 1 1 1 1 1 1 1 Ridisegna Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 0 1 0 1 1 0 Ridisegna Aspetta 60
  PulisciSchermo NascondiTartaruga numero 1 1 1 1 0 1 1 Ridisegna Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 1 1 1 0 1 1 Ridisegna Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 0 1 1 1 0 1 Ridisegna Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 1 1 1 1 1 0 Ridisegna Aspetta 60
  PulisciSchermo NascondiTartaruga numero 1 1 0 1 1 1 0 Ridisegna Aspetta 60
  PulisciSchermo NascondiTartaruga numero 0 0 1 0 1 0 0 Ridisegna Aspetta 60
  PulisciSchermo NascondiTartaruga numero 1 1 1 0 1 1 1 Ridisegna Aspetta 60
  FermaAnimazione
Fine
```

10.2 L'uomo che cresce



Come prima cosa definiamo una procedura `uomo` che disegna lo schema qui sopra. Usiamo una variabile per riprodurlo a scale diverse.

```
Per uomo :c
  SX 154 Avanti 44*:c Indietro 44*:c
  SX 52 Avanti 44*:c Indietro 44*:c
  SX 154 Avanti 40*:c
  SX 154 Avanti 44*:c Indietro :c*44
  SX 52 Avanti 44*:c Indietro :c*44
  SX 154 Avanti 10*:c
  SX 90 Ripeti 180[Avanti :c/2 DX 2] DX 90
Fine
```

Adesso creiamo una animazione che farà crescere l'uomo. Per realizzare ciò disegniamo `uomo 0.1`, quindi `uomo 0.2`, `uomo 0.3`... fino a `uomo 5`. Fra ciascun uomo cancelliamo l'area di disegno. Otteniamo due diverse procedure:

```
Per uomo :c
  SX 154 Avanti 44*:c Indietro 44*:c
  SX 52 Avanti 44*:c Indietro 44*:c
  SX 154 Avanti 40*:c
  SX 154 Avanti 44*:c Indietro :c*44
  SX 52 Avanti 44*:c Indietro :c*44
  SX 154 Avanti 10*:c
  SX 90 Ripeti 180[Avanti :c/2 DX 2] DX 90
  Se :c=5[Ferma]
  PulisciSchermo NascondiTartaruga uomo :c+0.1
Fine

Per via
  PulisciSchermo NascondiTartaruga
  uomo 0
Fine
```

Alla fine per rendere l'animazione fluida useremo la modalità animazione e la primitiva `Ridisegna`.

```
Per uomo :c
  SX 154 Avanti 44*:c Indietro 44*:c
  SX 52 Avanti 44*:c Indietro 44*:c
```

```
  SX 154 Avanti 40*:c
  SX 154 Avanti 44*:c Indietro :c*44
  SX 52 Avanti 44*:c Indietro :c*44
  SX 154 Avanti 10*:c
  SX 90 Ripeti 180[Avanti :c/2 DX 2] DX 90
  Ridisegna
  Se :c=5[Ferma]
  PulisciSchermo NascondiTartaruga uomo :c+0.1
Fine

Per via
  PulisciSchermo NascondiTartaruga Animazione
  uomo 0
  FermaAnimazione
Fine
```

Nota: qui la procedura `uomo` è ricorsiva. Avremmo anche potuto usare la primitiva `RipetiPer` per assegnare alla variabile :c i valori da 0.1 a 5, in questo modo:

```
Per uomo :c
  PulisciSchermo SX 154 Avanti 44*:c Indietro 44*:c
  SX 52 Avanti 44*:c Indietro 44*:c
  SX 154 Avanti 40*:c
  SX 154 Avanti 44*:c Indietro :c*44
  SX 52 Avanti 44*:c Indietro :c*44
  SX 154 Avanti 10*:c
  SX 90 Ripeti 180[Avanti :c/2 DX 2] DX 90
  Ridisegna
Fine

Per go
  NascondiTartaruga Animazione
  RipetiPer [c 0 5 0.1][uomo :c]
  FermaAnimazione
Fine
```

Capitolo 11

Interazione utente-programma

Livello: Principiante

11.1 Un programma di domanda e risposta

Il programma che creeremo in questo capitolo chiederà all'utente il suo nome, cognome ed età. Alla fine il programma ne farà una sintesi.

```
Il tuo nome è: .......
Il tuo cognome è: .......
La tua età è: .......
Sei oltre i 20/sotto i 20 anni
```

Queste sono le primitive che useremo:

- `Leggi`: `Leggi [Come stai?] "a`
  Visualizza una finestra di dialogo il cui titolo è il testo dall'elenco (qui "Come stai?"). La risposta fornita dall'utente è inserita in una parola o in un elenco (in caso di più parole) nella variabile `:a`.

- `AssegnaVar`: `AssegnaVar "a 30`
  Inserisce il valore 30 nella variabile `:a`

- `Frase`: `Frase [30 k] "a`
  Aggiunge un valore ad un elenco. Se il valore è un elenco a sua volta, ne rimuove le parentesi quadre.

```
Frase [30 k] "a # [30 k a]
Frase [1 2 3] 4 # [1 2 3 4]
Frase [1 2 3] [4 5 6] # [1 2 3 4 5 6]
```

Questo è il programma:

```
Per domande
  Leggi [Quanti anni hai?] "age
  Leggi [Qual e' il tuo nome di battesimo?] "fname
  Leggi [Qual e' il tuo cognome?] "name
  Stampa Frase [Il tuo cognome e': ] :name
  Stampa Frase [Il tuo nome e': ] :fname
  Stampa Frase [La tua eta' e': ] :age
  Se o :age>20 :age=20 [Stampa [Sei oltre i 20 anni]]
    [Stampa [Sei sotto i 20 anni]]
Fine
```

11.2 Programmare un semplice gioco

Vogliamo realizzare questo semplice gioco. Il programma sceglie un numero intero tra 0 e 32 e lo memorizza. Quindi apre una finestra di dialogo e chiede all'utente di inserire un numero. Se l'intero è uguale a quello scelto, visualizza "Hai vinto!". Altrimenti il programma indica se il numero scelto è più grande o più piccolo del numero dell'utente e riapre la finestra di dialogo. Il programma termina quando l'utente ha indovinato il numero corretto.
Dobbiamo usare la primitiva `Casuale`:
Per esempio `Casuale 20` restituisce un numero intero scelto casualmente tra 0 e 19.

Ecco come il programma deve essere realizzato:

- Il numero scelto dal computer viene inserito nella variabile `numero`.
- La finestra di dialogo deve essere chiamata "Inserisci un numero intero".
- Il numero scelto dall'utente deve essere immagazzinato in una variabile chiamata `prova`.
- La procedura principale deve essere chiamata `gioco`.

Alcune possibili miglioramenti:

- Mostrare il numero di prove.
- Il numero scelto dal computer può essere tra 0 e 2000.
- Controllare che l'utente inserisce un numero valido utilizzando la primitiva `Numero?`.
  Esempi: `Numero? 8` restituisce vero.
  `Numero? [5 6 7]` restituisce falso.
  `Numero? abcde` restituisce falso.

Capitolo 12

Argomento: Somma di due dadi

Livello: Medio

Lanciando due dadi, la somma dei punti sulla loro faccia superiore è un numero intero compreso tra 2 e 12. Il programma calcolerà le diverse probabilità di apparire di ciascun numero e le rappresenterà in un grafico.

12.1 Simulare il lancio di un dado.

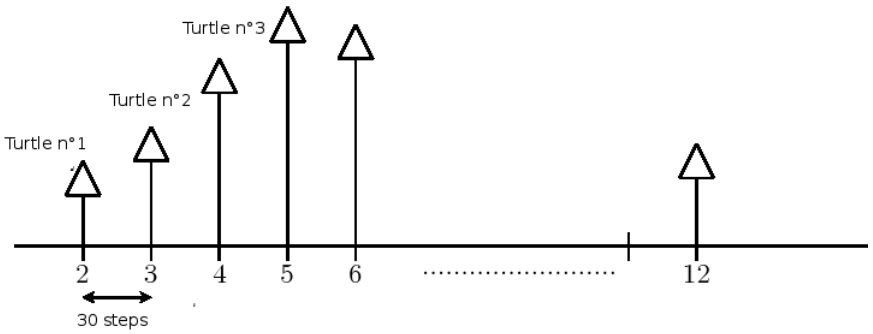
Per simulare il lancio di un dado useremo la primitiva `Casuale`. Ecco come funziona:

`Casuale 6` ⟶ restituisce un numero intero compreso tra 0 e 5 (0, 1, 2, 3, 4, 5). Quindi `(Casuale 6)+1` restituisce un intero a caso tra 1 e 6. Le parentesi sono necessarie perché altrimenti Logo sommerebbe 6+1 e restituirebbe un numero casuale tra 0 e 6. Per evitare di scrivere le parentesi potremmo scrivere `1+random 6`. Definiamo la procedura chiamata `dado` che simula il lancio di un dado.

```
Per dado
  output 1+Casuale 6
Fine
```

12.2 Il programma

Useremo la modalità multitartaruga e la primitiva `ImpostaTartaruga`. `ImpostaTartaruga` seguito da un numero intero ci permette di selezionare la tartaruga identificata dal numero stesso.
Uno schema è meglio di mille parole...



Ciascuna tartaruga il cui numero va da 2 a 12 avanzerà di un passo quando la somma delle facce dei dadi appare. Per esempio, se i due dadi totalizzano 8 allora la tartaruga 8 avanzerà di un passo. Tra due tartarughe successive ci sono 30 passi orizzontalmente.
Impostiamo tutte le tartarughe usando le coordinate.

- La tartaruga numero 2 ha coordinate $(-150; 0)$
- La tartaruga numero 3 ha coordinate $(-120; 0)$
- La tartaruga numero 4 ha coordinate $(-90; 0)$
- La tartaruga numero 5 ha coordinate $(-60; 0)$

$\vdots$

```
ImpostaTartaruga 2 ImpPos [-150 0]
ImpostaTartaruga 3 ImpPos [-120 0]
ImpostaTartaruga 4 ImpPos [-90 0]
ImpostaTartaruga 5 ImpPos [-60 0]
ImpostaTartaruga 6 ImpPos [-30 0]
...
```

Meglio di scrivere per 11 volte la stessa linea di comandi usiamo la primitiva `RipetiPer`. Con questa primitiva possiamo assegnare ad una variabile una sequenza di valori. In questo caso vogliamo assegnare alla variabile `:i` valori successivi da 2 a 12. Scriviamo: `RipetiPer [i 2 12] [ elenco di istruzioni ]`

Per impostare tutte le tartarughe creiamo la procedura `inizializza`

```
per inizializza
  PulisciSchermo NascondiTartaruga PennaSu
  RipetiPer [i 2 12] [
    ImpostaTartaruga :i ImpPos Elenco -150+(:i-2)*30 0
    PennaSu Indietro 15 Etichetta :i Avanti 15 PennaGiu
  ]
fine
```

Cerchiamo di capire l'espressione `-150+(:i-2)*30`. Iniziamo da -150 e, per ogni nuova tartaruga, aggiungiamo 30 (Verificalo con diversi valori di `:i` se sei scettico).
Infine questo è il programma:

```
Per dado
  output 1+Casuale 6
Fine

per inizializza
  PulisciSchermo NascondiTartaruga PennaSu
  RipetiPer [i 2 12] [
    ImpostaTartaruga :i ImpPos Elenco -150+(:i-2)*30 0
    PennaSu Indietro 15 Etichetta :i Avanti 15 PennaGiu
  ]
fine

per go
  inizializza
  Ripeti 1000 [
    AssegnaVar "sum dado+dado
    impostatartaruga :sum Avanti 1
  ]
  RipetiPer [i 2 12] [
    impostatartaruga :i
    # the y-coordinates of each turle represents the number of times
    # a dice scores has appeared
    AssegnaVarLocale "number Ultimo Posizione
    PennaSu Avanti 10 RuotaSinistra 90 Avanti 10
    RuotaDestra 90 PennaGiu Etichetta :number/1000*100
  ]
fine
```

Qui c'è un programma più generale. Chiederemo all'utente il numero di dadi e quello di lanci.

```
per initialize
  PulisciSchermo NascondiTartaruga PennaSu impostatartarugasmax :max+1
  RipetiPer Frase Elenco "i :min :max [
    # set BeccheggiaSu the Tartaruga
    impostatartaruga :i imppos Elenco -150+(:i-2)*30 0
    # we Scrivi Tartaruga number 15 steps under
    PennaSu Indietro 15 Etichetta :i Avanti 15 PennaGiu
  ]
fine

per die
  AssegnaVarLocale "somme 0
  Ripeti :dice [
    AssegnaVarLocale "somme :somme+1 +Casuale 6
  ]
  Output :somme
fine

per go
  Leggi [Numero di dadi:] "dice
  Se non Numero? :dice [Stampa [Non e' un numero valido!] Ferma]
  AssegnaVar "min :dice
  AssegnaVar "max 6*:dice
  Leggi [Numero di lanci: ] "tries
  Se non Numero? :tries [Stampa [Non e' un numero valido] Ferma]
  initialize
  Ripeti :tries [
    impostatartaruga die Avanti 1
  ]
  RipetiPer Frase Elenco "i :min :max [
    impostatartaruga :i
    AssegnaVarLocale "effectif Ultimo Posizione
    # Arrotonda per 0.1
    PennaSu Avanti 10 RuotaSinistra 90 Avanti 10 RuotaDestra 90
    PennaGiu Etichetta (Arrotonda :effectif/:tries*1000)/10
  ]
fine
```

Capitolo 13

Argomento: approssimazione probabilistica di pi greco

Livello: Avanzato

Nota: Alcune nozioni matematiche elementari sono necessarie per questo capitolo.

13.1 MCD (Massimo Comune Divisore)

Il MCD di due numeri interi è l'intero più grande che divide entrambi i numeri senza resto. Per esempio il MCD di 42 e 28 è 14, il MCD di 25 e 55 è 5, il MCD di 42 e 23 è 1.

Gli interi a e b si dicono **coprimi** o **primi tra loro** se non hanno un fattore comune eccetto 1 o, in modo equivalente, se il loro MCD è 1. Negli esempi precedenti 42 e 23 sono coprimi.

13.2 L'algoritmo di Euclide

L'algoritmo di Euclide calcola il MCD di due interi in modo efficiente (non dimostriamo qui che questo algoritmo è valido).

13.2.1 Descrizione dell'algoritmo

Dati due interi positivi a e b, controlliamo prima di tutto se b sia uguale a 0. Se è questo il caso allora il MCD è a. In caso contrario calcoliamo r come resto della divisione di a per b. Quindi sostituiamo a con b e b con r e ricominciamo l'algoritmo.
Per esempio calcoliamo il MCD di 2160 e 888 con l'algoritmo di Euclide:

a	b	r
2160	888	384
888	384	120
384	120	24
120	24	0
24	0	

Quindi il MCD di 2160 e 888 è 24. Non c'è intero più grande che divide entrambi i numeri (infatti $2160 = 24 \times 90$ e $888 = 24 \times 37$)
Il MCD è resto più grande non uguale a 0.

13.3 Calcolare il MCD in Logo

Una semplice procedure ricorsiva calcolerà il MCD di due interi `:a` e `:b`:

```
Per MCD :a :b
  Se (modulo :a :b)=0 [output :b][output MCD :b modulo :a :b]
Fine
```

Invochiamo la procedure come `Print MCD 2160 888` e otterremo il risultato 24.
Nota: è importante porre tra parentesi tonde `modulo :a :b` altrimenti l'interprete tenterà di verificare la condizione `:b = 0`. Se non vuoi usare le parentesi scrivi `Se 0=resto :a :b`.

13.4 Calcolare l'approssimazione di pi greco

Nei fatti un famoso risultato nella teoria dei numeri dice che la probabilità di due interi scelti a caso di essere coprimi è $\frac{6}{\pi^2} \approx 0,6079$. Per verificare questo risultato possiamo:

- Scegliere a caso due interi tra 0 e 1000000.
- Calcolare il loro MCD.
- Se il MCD è 1, incrementiamo una variabile contatore.
- Ripetiamo questa procedura per 1000 volte.
- La frequenza può essere calcolata dividendo la variabile contatore per 1000 (il numero di prove).

```
Per test
  # impostiamo la variabile contatore a 0
  AssegnaVar "counter 0
  Ripeti 1000 [
    Se (MCD Casuale 1000000 Casuale 1000000)=1 [AssegnaVar "counter :counter+1]
  ]
  Stampa [frequenza:]
  Stampa :counter/1000
Fine
```

Come prima nota le parentesi attorno `MCD Casuale 1000000 Casuale 1000000`. Se non ci fossero l'interprete cercherebbe di verificare la condizione 1 000 000 = 1. La stessa espressione si può anche scrivere come: `Se 1=MCD Casuale 1000000 Casuale 1000000`.

Invochiamo la procedura con `test` e con un po' di pazienza otterremo frequenze che si avvicinano al valore teorico di 0,6097:

```
test
0.609
test
0.626
test
0.597
```

La frequenza teorica è una approssimazione di $\frac{6}{\pi^2}$.

Quindi, se denotiamo con f la frequenza abbiamo: $f \approx \frac{6}{\pi^2}$ da cui $\pi^2 \approx \frac{6}{f}$ e $\pi \approx \sqrt{\frac{6}{f}}$.

Aggiungo al mio programma una riga che fornisca questa approssimazione di pi greco nella procedura `test`:

```
Per test
  # impostiamo la variabile contatore a 0
  AssegnaVar "counter 0
  Ripeti 1000 [
    Se (MCD Casuale 1000000 Casuale 1000000)=1 [AssegnaVar "counter :counter+1]
  ]
  # Calcoliamo la frequenza
  Assegna "f :counter/1000
  # stampiamo l'approssimazione di pi greco
  Stampa frase [ approssimazione di pi greco:] RadQ (6/:f)
Fine
```

Invochiamo la procedura con `test` e con un po' di pazienza otterremo valori di pi greco che si avvicinano a quello teorico:

```
test
approssimazione di pi greco: 3.1033560252704917
test
approssimazione di pi greco: 3.1835726998350666
test
approssimazione di pi greco: 3.146583877637763
```

Ora voglio modificare il programma perché vorrei impostare liberamente il numero di prove. Vorrei provare con 10000 e forse con ancor più prove.

```
Per test :prove
  # impostiamo la variabile contatore a 0
  AssegnaVar "counter 0
  Ripeti :prove [
    Se (MCD Casuale 1000000 Casuale 1000000)=1 [AssegnaVar "counter :counter+1]
  ]
  # Calcoliamo la frequenza
  Assegna "f :counter/:prove
  # stampiamo l'approssimazione di pi greco
  Stampa frase [ approssimazione di pi greco:] RadQ (6/:f)
Fine
```

Invochiamo la procedura così:

```
test 10000
approssimazione di pi greco: 3.1426968052735447
test 10000
approssimazione di pi greco: 3.1478827771265787
test 10000
approssimazione di pi greco: 3.146583877637763
```

Interessante, no?

13.5 La generazione del pi greco mediante il pi greco...

Che cos'è un intero casuale? Può un intero scelto casualmente tra 1 e 1000000 essere realmente rappresentativo di tutti gli interi scelti casualmente? Osserviamo che il nostro esperimento è solo una approssimazione del modello ideale. Adesso modificheremo il metodo per generare numeri casuali.... Non useremo la primitiva `Casuale` ma genereremo numeri casuali usando la sequenza delle cifre del pi greco.
Le cifre del pi greco hanno sempre interessato i matematici:

- Le cifre da 0 a 9, ce ne sono alcune che sono più frequenti di altre?

- C'è qualche sequenza di interi che appaiono frequentemente?

In realtà **sembra** che la sequenza del pi greco sia veramente casuale (ancora non dimostrato). Non è possibile prevedere la cifra successiva in base alle precedenti, non c'è periodicità.
Useremo la sequenza delle cifre del pi greco per generare interi casuali:

- Come prima cosa abbiamo bisogno delle prime cifre del pi greco (per esempio un milione)

  1. Possiamo usare qualche programma che le calcoli come PiFast in ambienti Windows e SchnellPi per Linux.
  2. Oppure possiamo scaricare questo file dal sito di XLogo:

     `http://downloads.tuxfamily.org/xlogo/common/millionpi.txt`

- Per generare gli interi leggeremo la sequenza di cifre in pacchetti di 7 cifre:
  3.1415926 53589793 23846264 338327950288419716939 ecc
  Primo numero Secondo numero Terzo numero
  Ho rimosso la virgola "." in 3.14 che avrebbe causato problemi durante l'estrazione delle cifre.

Creiamo ora una nuova procedura chiamata `CasualePi` e modifichiamo la procedura `test`.

```
Per MCD :a :b
  Se (modulo :a :b)=0 [output :b][output MCD :b modulo :a :b]
Fine

per test :prove
  # apriamo un flusso con id 1 verso il file  millionpi.txt
  # che deve essere nella cartella attuale
  # altrimenti usare CambiaDirectory
  ApriFlusso 1 "millionpi.txt
  # assegna la variabile line per la prima linea del file millionpi.text
  AssegnaVar "line Primo LeggiLineaDalFlusso 1
  # impostiamo la variabile contatore a 0
  AssegnaVar "counter 0
  Ripeti :prove [
    Se 1=gcd CasualePi 7 CasualePi 7 [AssegnaVar "counter :counter+1]
  ]
  # Calcoliamo la frequenza
  Assegna "f :counter/:prove
  # stampiamo l'approssimazione di pi greco
  Stampa frase [ approssimazione di pi greco:] RadQ (6/:f)
  ChiudiFlusso 1
fine

per CasualePi :n
  AssegnaVarLocale "number "
  Ripeti :n [
  Se 0=count :line [AssegnaVar "line Primo LeggiLineaDalFlusso 1]
  # imposta la variabile char al primo carattere della linea
  AssegnaVar "char Primo :line
  # quindi rimuove il primo carattere dalla linea
  AssegnaVar "line EccettoPrimo :line
  AssegnaVar "number Parola :number :char
  ]
  Output :number
fine
```

Invochiamo la procedura come al solito:

```
test 10
```

```
approssimazione di pi greco: 3.4641016151377544
test 100
approssimazione di pi greco: 3.1108550841912757
test 1000
approssimazione di pi greco: 3.081180112566604
test 10000
approssimazione di pi greco: 3.1403714651066386
test 70000
approssimazione di pi greco: 3.1361767950325627
```

Stiamo trovato la corretta approssimazione di pi greco utilizzando le sue stesse cifre.
E' ancora possibile migliorare il programma calcolando il tempo del calcolo. Aggiungiamo alla prima linea della procedura `text`:

```
AssegnaVar inizio SecondiDaAvvio
```

Quindi aggiungiamo prima di chiudere il flusso:

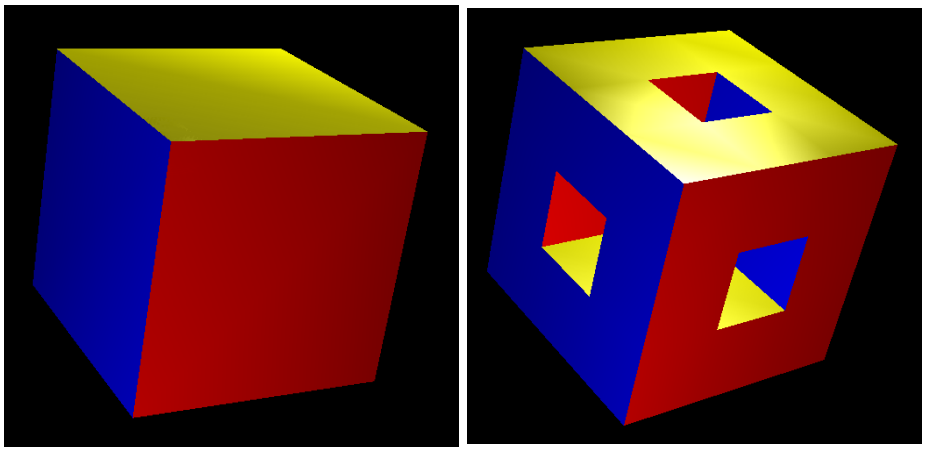
```
Stampa Frase [Tempo impiegato:  ]  SecondiDaAvvio - :inizio
```

Capitolo 14

Argomenti: la spugna di Menger

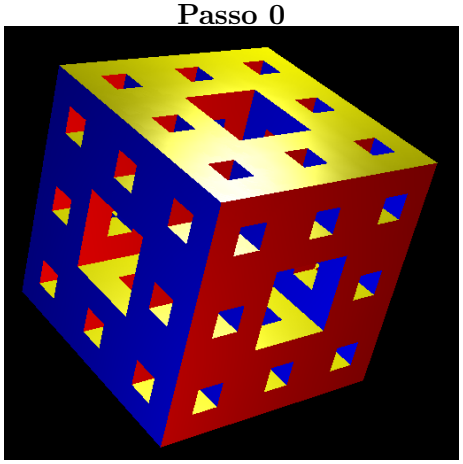
Livello: Avanzato

In questo capitolo costruiremo un solido frattale chiamato spugna di Menger. Questi sono i primi passi per creare questo solito:

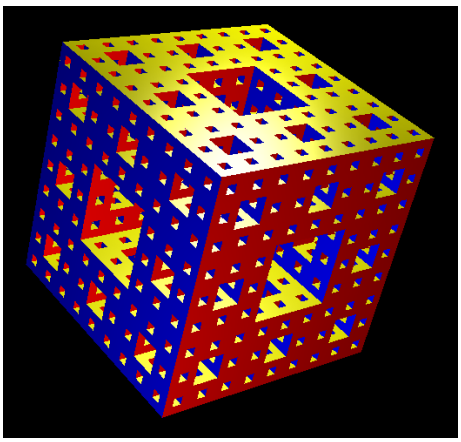


Passo 0

Passo 1



Passo 2



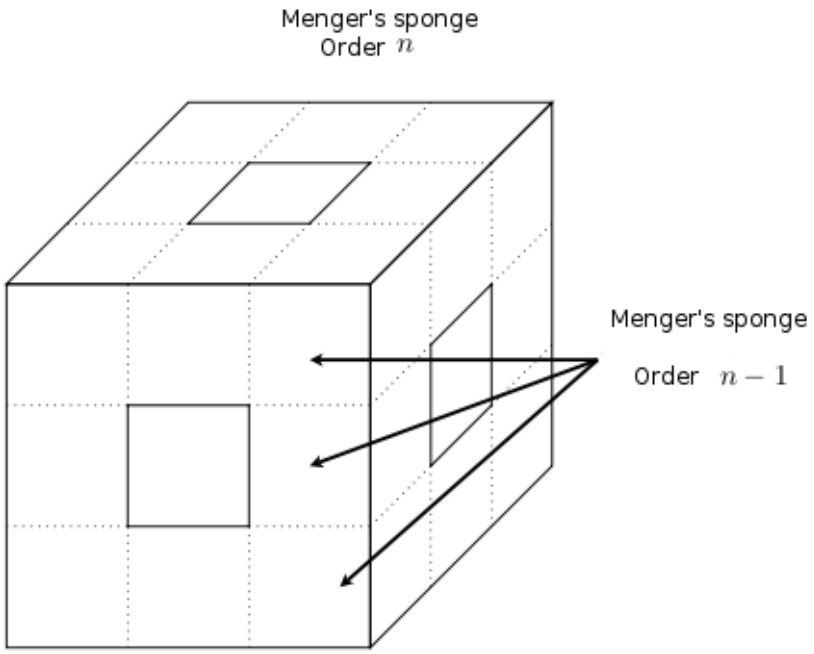
Passo 3

Questo capitolo contiene due sezioni:

- Per primo vedremo come creare questo solido utilizzando la ricorsione.
- Infine proveremo a generare una spugna di Menger di ordine 4.

14.1 Primo approccio: ricorsione

Consideriamo una spugna di Menger di ordine n con spigolo L.



Nello schema notiamo che la spugna contiene 20 spugne di Menger di ordine $n-1$ di spigolo $\frac{L}{3}$. La natura ricorsiva della spugna e' evidente.

14.1.1 Il programma

```
per cubo :l
  Se :counter=10000 [VistaPoligono3D]
  AssegnaVarLocale "colori [Giallo Magenta Ciano Blu]
  # facce laterali
  Ripeti 4 [ImpostaColorePenna Lancia Elemento ContaRipetizioni :colori
  quadrato :l RuotaDestra 90 Avanti  :l RuotaSinistra 90 RollioDestra 90]
  # faccia inferiore
  ImpostaColorePenna Rosso BeccheggiaGiu 90 quadrato :l BeccheggiaSu 90
  Avanti :l BeccheggiaGiu 90 ImpostaColorePenna Verde quadrato :l
  BeccheggiaSu 90 Indietro :l
fine

per quadrato :c
  AssegnaVar "counter :counter+1
  InizioPoligono
  Ripeti 4 [Avanti :c RuotaDestra 90]
  FinePoligono
fine

# spugna di menger
# p: ordine di ricorsione
# l: spigolo del cubo
per menger :l :p
  Se :p=0 [cubo :l] [
    AssegnaVarLocale "p :p-1
    AssegnaVarLocale "l :l/3
    # faccia frontale
```

```
    Ripeti 3 [menger :l :p Avanti :l] Indietro 3*:l
    RuotaDestra 90 Avanti :l RuotaSinistra 90
    menger :l :p Avanti 2*:l menger :l :p Indietro 2*:l
    RuotaDestra 90 Avanti :l RuotaSinistra 90
    Ripeti 3 [menger :l :p Avanti :l] Indietro 3*:l
   # faccia di destra
   BeccheggiaGiu 90 Avanti :l BeccheggiaSu 90
    menger :l :p  Avanti 2*:l menger :l :p Indietro 2*:l
    BeccheggiaGiu 90 Avanti :l BeccheggiaSu 90
    Ripeti 3 [menger :l :p Avanti :l] Indietro 3*:l
    RuotaSinistra 90 Avanti :l RuotaDestra 90
    menger :l :p  Avanti 2*:l menger :l :p Indietro 2*:l
    RuotaSinistra 90 Avanti :l RuotaDestra 90
    Ripeti 3 [menger :l :p Avanti :l] Indietro 3*:l
    BeccheggiaGiu 90 Indietro :l BeccheggiaSu 90
    menger :l :p  Avanti 2*:l menger :l :p Indietro 2*:l
     BeccheggiaGiu 90 Indietro :l BeccheggiaSu 90
  ]
fine

per spugna :p
  PulisciSchermo NascondiTartaruga AssegnaVar "counter 0 3D
  ImpostaColoreSfondo 0 menger 800 :p
  Scrivi [nombre ColorePenna polygone: ] Stampa :counter
  VistaPoligono3D
fine
```

Il programma ha quattro procedure:

- `quadrato :c`
  Questa procedura disegna un quadrato di lato `:c`. Il poligono viene immagazzinado nel visualizzatore 3D. La variabile `counter` conta il numero dei poligoni disegnati.

- `cubo :l`
  La procedura disegna un cubo di spigolo `:l`. Ovviamente utilizza `quadrato`.

- `menger :l :p`
  Questa e' la procedura più importante, disegna le linee di Menger di ordine p di lunghezza l. La linea e' creata usando la ricorsione come abbiamo visto nello schema.

- `spugna :p`
  Questa procedura crea la spugna di Menger, di ordine p con spigolo pari a 800 e la disegna nel visualizzatore 3D.

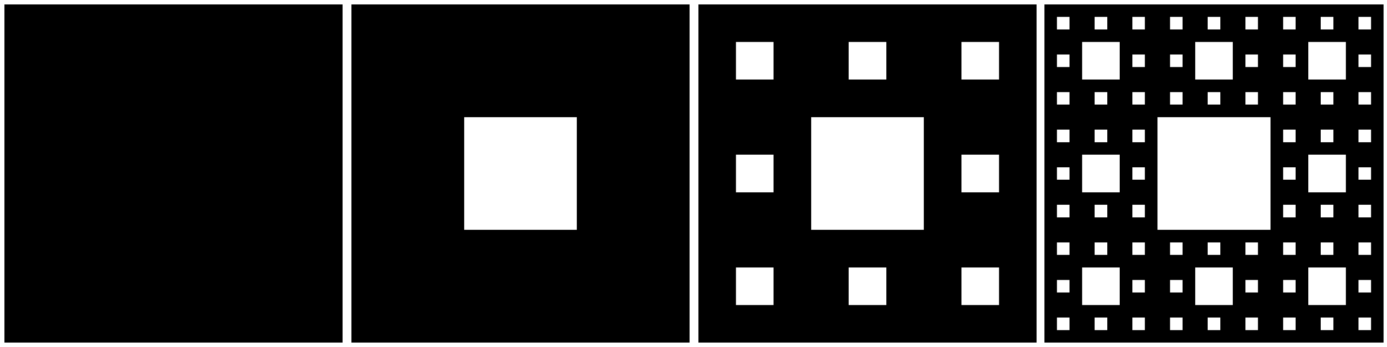
Il vantaggio di questo programma e' di sfruttare la struttura ricorsiva del solido. Il metodo e' molto simile a quello usato per disegnare il fiocco di neve di Van Kock a pagina 42. Il vantaggio della ricorsione e' una codice piuttosto snello.
Il principale svantaggio di questo approccio e' che, essendo ricorsivo, richiede molta memoria per la sua esecuzione. Infatti una spugna di ordine 3 disegna 48000 poligoni. XLogo richiede in questo caso di alzare il limite della memoria interna di 256 MB (impostabile nel pannello delle preferenze) per prevenire l'overflow.

14.2 Secondo approccio: il tappeto Sierpinski

Se vogliamo disegnare una spugna di Menger di ordine 4, dobbiamo ripensare il programma e dimenticare la ricorsione. Creeremo un programma che disegnerà il solido di Menger di ordine 0, 1, 2, 3 e 4.

14.2.1 il tappeto di Sierpinski

La spugna di Menger e' una generalizzazione in 3 dimensioni di una figura piana chiamata "tappeto di Sierpinski". Questi sono i primi passi per generare questa figura:



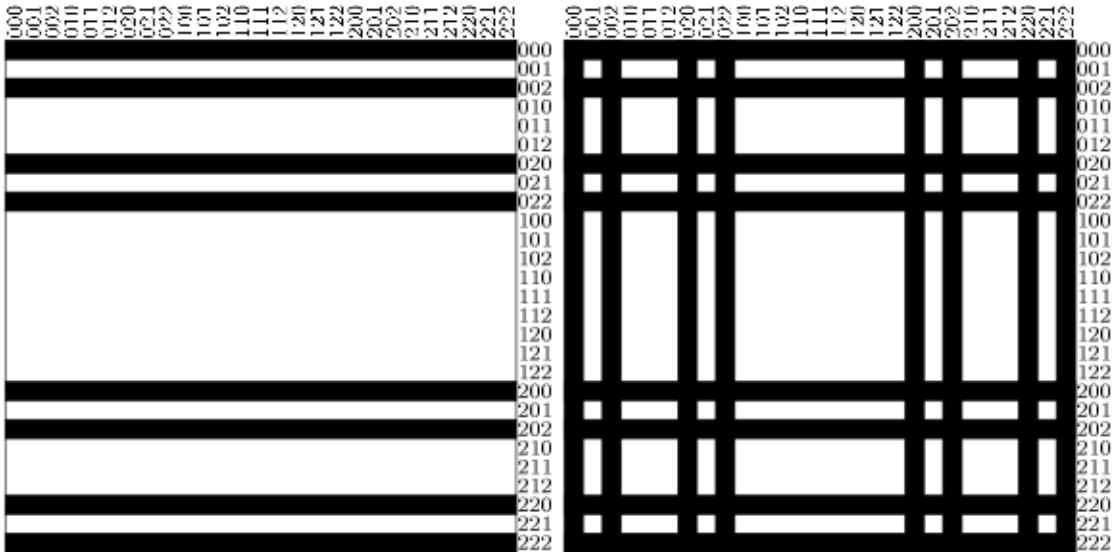
Passo 0 **Passo 1** **Passo 2** **Passo 3**

Ciascuna faccia della spugna di Menger di ordine p e' un tappeto di Sierpinski di ordine p.

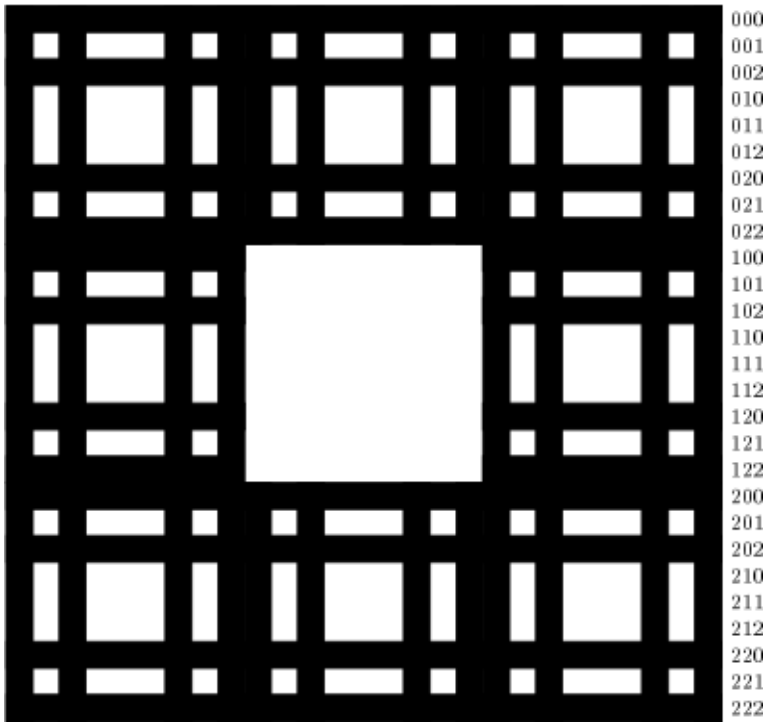
14.2.2 Disegnare un tappeto Sierpinski di ordine p

L'obiettivo e' di usare il minor numero possibile di poligoni per disegnare un tappeto Sierpinski. Il seguente esempio spiega come disegnare un tappeto di ordine 3. Il primo quadrato ha $3^3 = 27$ linee e 27 colonne. Scriviamo in base 3 ciascun numero di riga e ciascun numero di colonna.

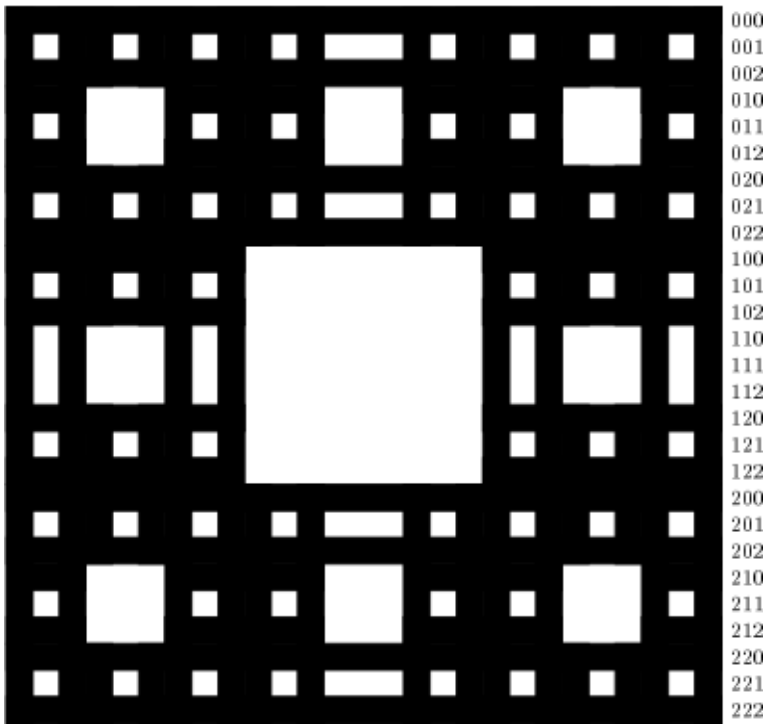
- **Primo passo**: Per ciascuna riga il cui numero non cintiene alcun 1, disegniamo una riga di 27 unità. Usando la simmetria ripetiamo la stessa operazione sulle colonne.



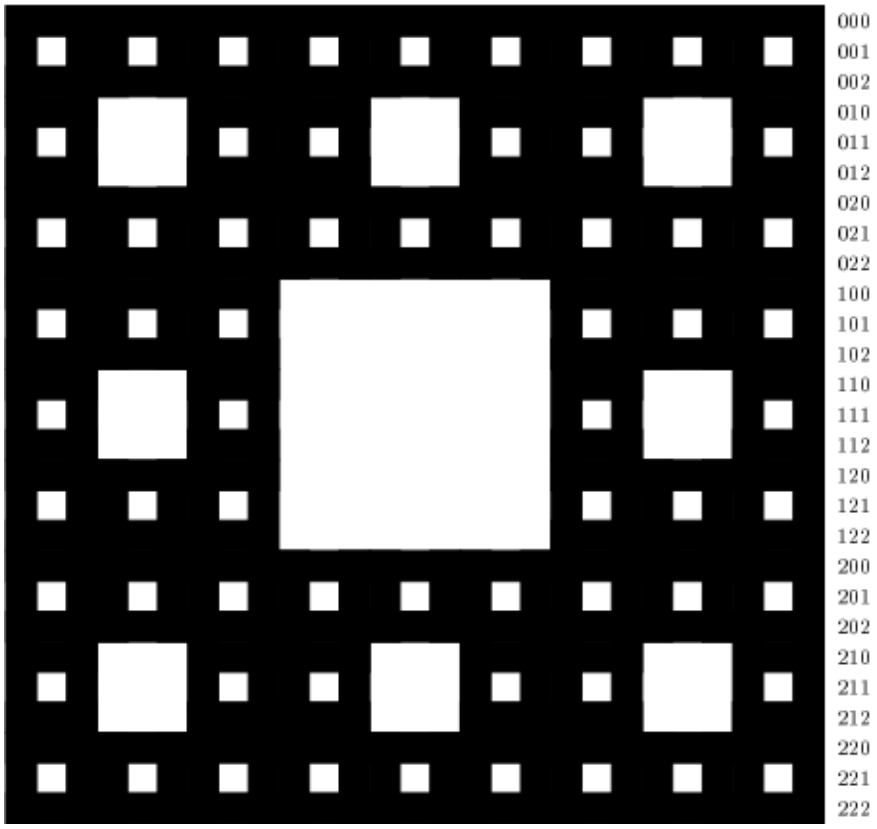
- **Secondo passo**: Adesso guardiamo alle righe i cui numeri hanno un singolo 1 al primo posto. Disegniamo rettangoli di 9 unità di lunghezza in modo alternato.



- **Terzo passo**: Adesso consideriamo le righe il cui numero contiene un singolo 1 al secondo posto. Disegniamo rettangoli seguendo lo schema [3 3 6 3 6 3 3] (cioe' 3 unità penna giù, 3 unità penna sù, 6 unità penna giù ...). Usando la simmetria ripetiamo l'operazione sulle colonne.



- **Passo finale**: Consideriamo le righe il cui numero contiene un 1 due volte nelle prime due posizioni. Disegniamo rettangoli alternati seguendo lo schema [3 3 3 9 3 3 3]. Ripetiamo lo stesso sulle colonne.



Abbiamo costruito un tappeto Sierpinski di ordine 3. Per disegnare tale poligono abbiamo bisogno di $16 + 16 + 32 + 16 = 80$ poligoni.

14.2.3 Tutti i diversi possibili schemi per le colonne

Per riassumere, ecco i diversi schemi delle colonne per i vari numeri di riga (il simbolo * rappresenta 0 o 2).

Numero di righe	Schema da applicare
***	27
1**	9 9 9
1	3 3 6 3 6 3 3
11*	3 3 3 9 3 3 3

Allo stesso modo, per costruire un tappeto di ordine 4 abbiamo bisogno di un quadrato di $3^4 = 81$ unità. Le righe e le colonne avranno numeri a base 3 di 4 cifre. Per ciascun numero di riga ecco lo schema da applicare (il simbolo * rappresenta 0 o 2).

Numero di righe	Schema da applicare
****	81
1***	27 27 27
*1**	9 9 18 9 18 9 9
**1*	3 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 3
11	3 3 3 9 3 3 6 3 3 9 3 3 6 3 3 9 3 3 3
1*1*	3 3 6 3 6 3 3 27 3 3 6 3 6 3 3
11**	9 9 9 27 9 9 9
111*	3 3 3 9 3 3 3 27 3 3 3 9 3 3 3

496 poligoni sono necessari in questo caso.
Infine questo e' lo schema per una figura di ordine 2.

Numero di righe	Schema da applicare
**	9
1*	3 3 3

14.2.4 Il programma

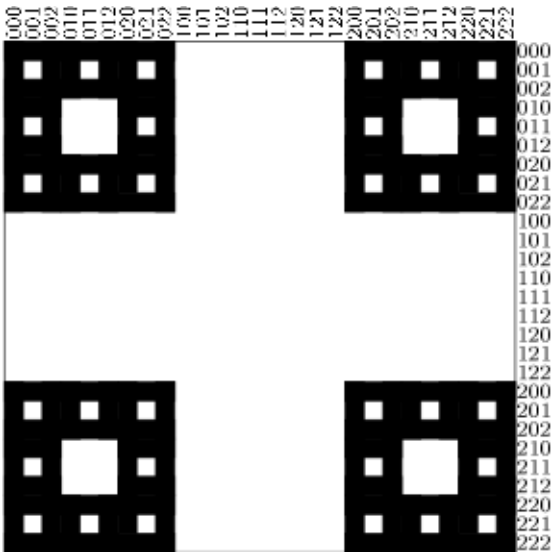
```
# disegna un tappeto sierpinski carpet di ordine :p e dimensione :size
per carpet :size :p
  AssegnaVar "unit :size/(Potenza 3 :p)
  Se :p=0 [ rec :size :size Ferma]
  Se :p=1 [Ripeti 4 [rec :size :unit Avanti :size RuotaDestra 90 ] Ferma]
  RipetiPer (Elenco "x 1 Potenza 3 :p) [
    AssegnaVarLocale "cantorx cantor :x :p []
  # non abbiamo disegnato gli elementi con un 1 in Ultimo Posizione
  Se  non (1=last :cantorx)  [
    AssegnaVarLocale "nom evalue EccettoUltimo :cantorx "
    drawcolumn :x Proprieta "map :nom
    ]
  ]
fine

# Restituisce il numero in base 3
# p ordeine del tappeto
# :list Elenco vuoto
per cantor :x :p :list
  Se :p=0 [Output :list]
  AssegnaVarLocale "a Potenza 3 :p-1
  Se :x<= :a [
    Output cantor  :x :p-1  Frase :list 0]
    [ Se :x<=2*:a [Output cantor  :x-:a :p-1  Frase :list 1]
    Output cantor :x-2*:a :p-1 Frase :list 0]
fine

# Disegna la colonna numero x
# rispetto lo schema nell'elenco :list
per drawcolumn :x :list
  PennaSu  RuotaDestra 90 Avanti (:x-1)*:unit RuotaSinistra 90
  PennaGiu des :list
  PennaSu RuotaSinistra 90 Avanti (:x-1)*:unit RuotaDestra 90
  Avanti :x*:unit RuotaDestra 90 PennaGiu des :list
  PennaSu RuotaSinistra 90 Indietro :x*:unit PennaGiu
fine

# Disegna un rettangolo delle dimensioni scelte
# e lo immagazzina nel visualizzatore 3D
per rec :lo :la
  AssegnaVar "compteur :compteur+1
  InizioPoligono
  Ripeti 2 [Avanti :lo RuotaDestra 90 Avanti :la RuotaDestra 90]
  FinePoligono
fine

# definisce le diverse possibile colonne
per initmap
  AggiungiProprieta "map 111 [3 3 3 9 3 3 3 27 3 3 3 9 3 3 3]
  AggiungiProprieta "map 110 [9 9 9 27 9 9 9]
  AggiungiProprieta "map 101 [3 3 6 3 6 3 3 27 3 3 6 3 6 3 3]
  AggiungiProprieta "map 011 [3 3 3 9 3 3 6 3 3 9 3 3 6 3 3 9 3 3 3]
  AggiungiProprieta "map 000 [81]
  AggiungiProprieta "map 100 [27 27 27]
  AggiungiProprieta "map 010 [9 9 18 9 18 9 9]
  AggiungiProprieta "map 001 [3 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 3]
  AggiungiProprieta "map 01 [3 3 6 3 6 3 3]
  AggiungiProprieta "map 00 [27]
  AggiungiProprieta "map 10 [9 9 9]
```

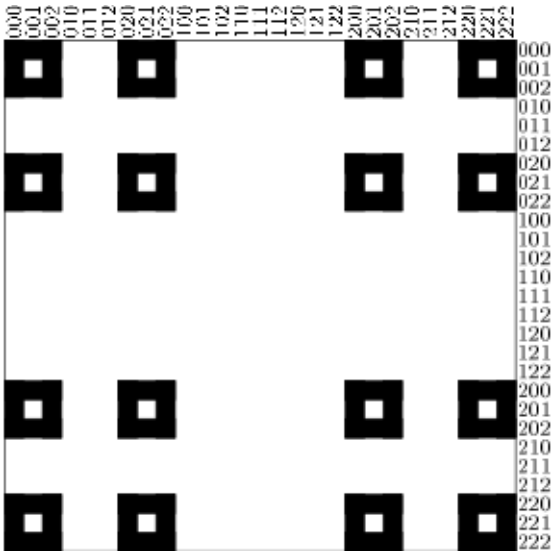
```
  AggiungiProprieta "map 11 [3 3 3 9 3 3 3]
  AggiungiProprieta "map 1 [3 3 3]
  AggiungiProprieta "map 0 [9]
fine

# Se il numero in base 3 e' [1 0 1] restituisce  101
per evalue :list :mot
  Se Vuoto? :list [Output :mot]
  [
    AssegnaVarLocale "mot Parola :mot Primo :list
    Output evalue EccettoPrimo :list :mot
  ]
fine

# Disegna il blocco dei rettangoli alternati
per des :list
  AssegnaVarLocale "somme 0
  RipetiPer (Elenco "i 1 Conta :list) [
     AssegnaVarLocale "element Elemento :i :list
      AssegnaVarLocale "somme :element+:somme
      Se pari? :i [PennaSu Avanti :element*:unit PennaGiu ]
     [rec :element*:unit :unit Avanti :element*:unit]
  ]
  PennaSu Indietro  :somme * :unit PennaGiu
fine

# :i e' un numero pari?
per pari? :i
  Output 0=resto :i 2
fine

# Disegna il tappeto di ordine :p
per tappeto :p
  PulisciSchermo 3D NascondiTartaruga initmap
  AssegnaVar "compteur 0
  carpet 810 :p
  Scrivi "Numero di poligoni:\  Stampa :compteur
  VistaPoligono3D
fine
```

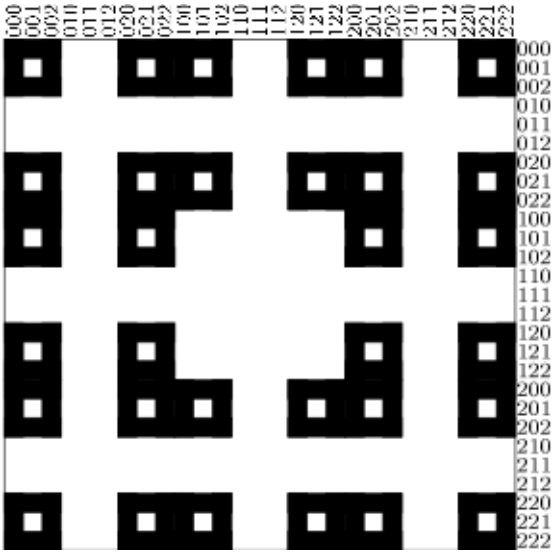
`tappeto 3` disegna un tappeto di Sierpinski di ordine 3 e di lato uguale a 810. Eccoci! Ora possiamo tornare alla spugna di Menger!

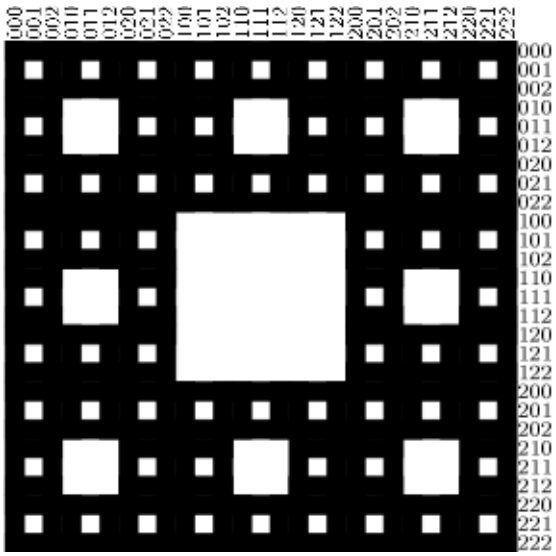
14.2.5 La spugna di Menger di ordine 4

La spugna di Menger ha molte simmetrie. Per costruire la spugna disegneremo le diverse sezioni lungo il piano (xOy) e quindi ripetiamo queste figure lungo i piani (yOz) e (xOz). Per spiegare cosa succede consideriamo la spugna di ordine 2. Quando tagliamo con un piano verticale possiamo ottenere quattro linee:


000 001 002 010 011 012 020 021 022 100 101 102 110 111 112 120 121 122 200 201 202 210 211 212 220 221 222


000 001 002 010 011 012 020 021 022 100 101 102 110 111 112 120 121 122 200 201 202 210 211 212 220 221 222


000 001 002 010 011 012 020 021 022 100 101 102 110 111 112 120 121 122 200 201 202 210 211 212 220 221 222



Per disegnare una spugna di ordine 3 faremo una ricerca nei numeri da 1 a 27, cioe' da 001 a 222 in base 3. Per ciascun numero applicheremo la sezione valida e riporteremo questa figura lungo (Ox), (Oy) e (Oz).

Il programma

Con questo programma possiamo disegnare la spugna di Menger di ordine 0,1,2,3 e 4.

```
# disegna un tappeto sierpinski carpet di ordine :p e dimensione :size
per carpet :size :p
  AssegnaVar "unit :size/(Potenza 3 :p)
  Se :p=0 [ rec :size :size Ferma]
  Se :p=1 [Ripeti 4 [rec :size :unit Avanti :size RuotaDestra 90 ] Ferma]
  RipetiPer (Lista "x 1 Potenza 3 :p) [
    AssegnaVarLocale "cantorx cantor :x :p []
    # non abbiamo disegnato gli elementi con un 1 in Ultimo Posizione
  Se  non (1=last :cantorx)  [
    AssegnaVarLocale "nom evalue EccettoUltimo :cantorx "
    drawcolumn :x Proprieta "map :nom
    ]
  ]
fine

# Restituisce il numero in base 3
# p ordeine del tappeto
# :list Elenco vuoto
per cantor :x :p :list
  Se :p=0 [Output :list]
  AssegnaVarLocale "a Potenza 3 :p-1
  Se :x<= :a [
    Output cantor  :x :p-1  Frase :list 0]
    [ Se :x<=2*:a [Output cantor  :x-:a :p-1  Frase :list 1]
    Output cantor :x-2*:a :p-1 Frase :list 2]
fine

# Disegna la colonna numero x
# rispetto lo schema nell'elenco :list
per drawcolumn :x :list
  PennaSu  RuotaDestra 90 Avanti (:x-1)*:unit
  RuotaSinistra 90  PennaGiu des :list
  PennaSu RuotaSinistra 90 Avanti (:x-1)*:unit
  RuotaDestra 90 Avanti :x*:unit RuotaDestra 90 PennaGiu des :list
  PennaSu RuotaSinistra 90 Indietro :x*:unit PennaGiu
fine
```

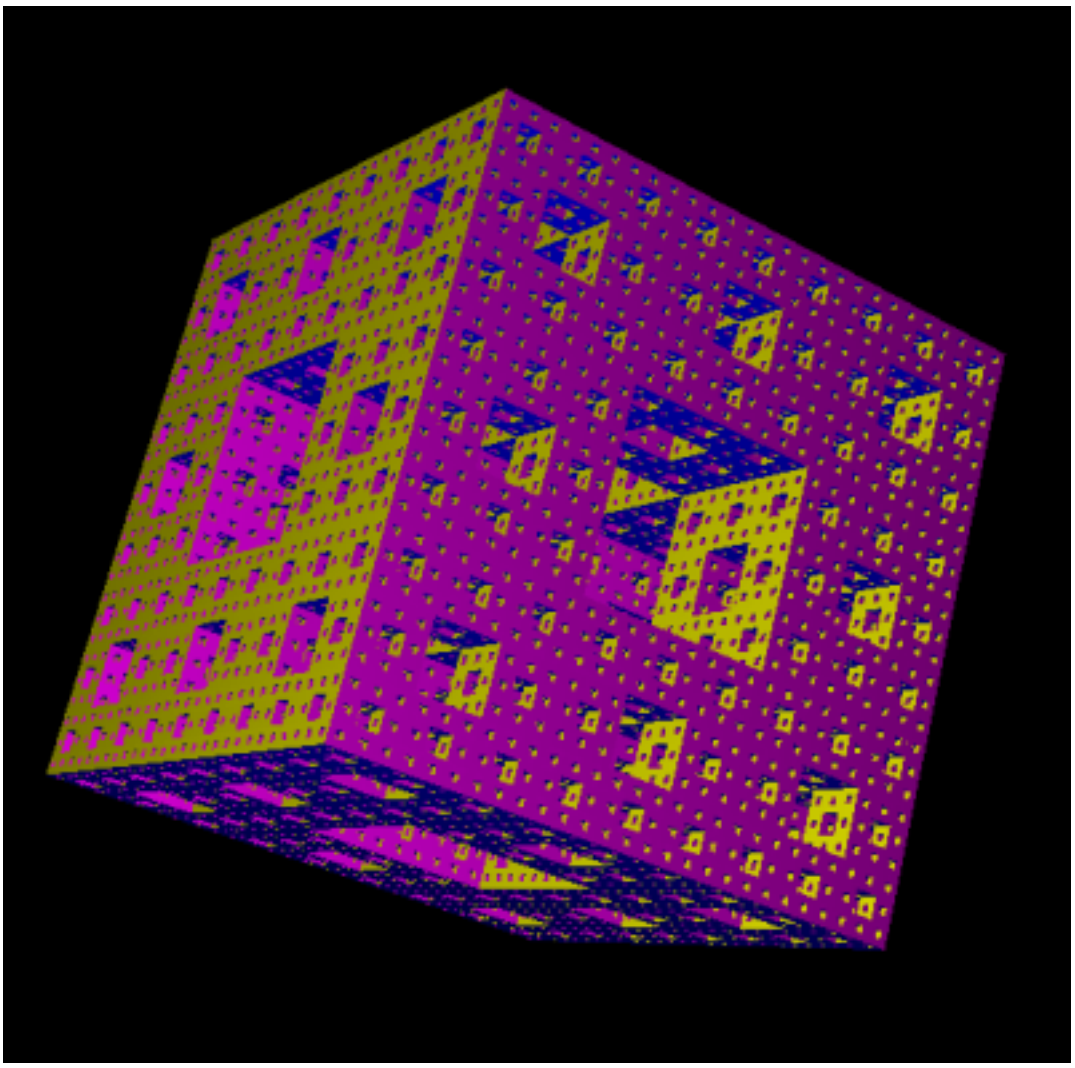
```
# Disegna un rettangolo delle dimensioni scelte
# e lo immagazzina nel visualizzatore 3D
per rec :lo :la
  AssegnaVar "counter :counter+1
  InizioPoligono
  Ripeti 2 [Avanti :lo RuotaDestra 90 Avanti :la RuotaDestra 90]
  FinePoligono
fine

# definisce le diverse possibile colonne
per initmap
  AggiungiProprieta "map 111 [3 3 3 9 3 3 3 27 3 3 3 9 3 3 3]
  AggiungiProprieta "map 110 [9 9 9 27 9 9 9]
  AggiungiProprieta "map 101 [3 3 6 3 6 3 3 27 3 3 6 3 6 3 3]
  AggiungiProprieta "map 011 [3 3 3 9 3 3 6 3 3 9 3 3 6 3 3 9 3 3 3]
  AggiungiProprieta "map 000 [81]
  AggiungiProprieta "map 100 [27 27 27]
  AggiungiProprieta "map 010 [9 9 18 9 18 9 9]
  AggiungiProprieta "map 001 [3 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 3]
  AggiungiProprieta "map 01 [3 3 6 3 6 3 3]
  AggiungiProprieta "map 00 [27]
  AggiungiProprieta "map 10 [9 9 9]
  AggiungiProprieta "map 11 [3 3 3 9 3 3 3]
  AggiungiProprieta "map 1 [3 3 3]
  AggiungiProprieta "map 0 [9]
fine

# Se il numero in base 3 e' [1 0 1] restituisce 101
# Se il numero in base 3 e' [1 0 2] restituisce 100
# gli elementi da :list sono tradotti in un parola
# 2 sono sostituiti da 0

per evalue :list :mot
  Se Vuoto? :list [Output :mot]
    [
    AssegnaVarLocale "first Primo :list
    Se :first=2 [AssegnaVarLocale "first 0]
   AssegnaVarLocale "mot Parola :mot :first
    Output evalue EccettoUltimo :list :mot
  ]
fine

# Disegna il blocco dei rettangoli alternati
per des :list
  AssegnaVarLocale "somme 0
  RipetiPer (Lista "i 1 Conta :list) [
     AssegnaVarLocale "element Elemento :i :list
      AssegnaVarLocale "somme :element+:somme
    Se even? :i [PennaSu Avanti :element*:unit PennaGiu ]
        [rec :element*:unit :unit Avanti :element*:unit]
  ]
  PennaSu Indietro  :somme * :unit PennaGiu
fine

# Disegna il tappeto di ordine :p
per tapis :p
  PulisciSchermo 3D NascondiTartaruga initmap
  AssegnaVar "compteur 0
  carpet 810 :p
  Scrivi "nombre\ de\ polygones:\  Stampa :compteur
```

```
  VistaPoligono3D
fine

# :i e' un numero pari?
per even? :i
  Output 0=modulo :i 2
fine


# Rimuovi l'ultimo 1 da :list
per deletelastone :list
  RipetiPer (Lista "i Conta :list 1 Meno 1) [
    AssegnaVarLocale "element Elemento :i :list
    Se :element=1 [AssegnaVarLocale "list Sostituisci :list :i 0
  Ferma] [Se :element=2 [Ferma]]
  ]
  Output :list
fine

# disegna il tappeto di serpinski
# lungo l'asse (ox), (oy) e (oz)
per draw3carpet :size :order :z
  PennaSu Origine
  BeccheggiaSu 90 Avanti (:z-1)*:unite BeccheggiaGiu 90 PennaGiu
  ImpostaColorePenna Blu Lancia :order :size
  PennaSu Origine
  RollioSinistra 90 Avanti (:z-1)*:unite BeccheggiaGiu 90  PennaGiu
  ImpostaColorePenna Giallo Lancia :order :size
  PennaSu Origine
  BeccheggiaSu 90 Avanti :size RuotaDestra 90 Avanti (:z-1)*:unite
  BeccheggiaGiu 90 PennaGiu
  ImpostaColorePenna Magenta Lancia :order :size
fine

# spugna di menger di ordine :p e dimensione :size
per menger :size :p
  AssegnaVar "unite :size/(Potenza 3 :p)
  RipetiPer (Lista "z 1 Potenza 3 :p) [
    AssegnaVarLocale "cantorz cantor :z :p []
    AssegnaVarLocale "last Ultimo :cantorz
    AssegnaVarLocale "cantorz EccettoUltimo :cantorz
    Se :last=0 [AssegnaVarLocale "order evalue deletelastone :cantorz "]
               [AssegnaVarLocale "order evalue :cantorz "]
    AssegnaVarLocale "order Parola "coupe :order
    draw3carpet :size :order :z
    PennaSu BeccheggiaSu 90 Avanti :unit BeccheggiaGiu 90 PennaGiu
  ]
  draw3carpet :size :order (Potenza 3 :p)+1
fine


# procedure pricipale
# disegna una spugna di ordine :p di spigolo 405
per sponge :p
  PulisciSchermo ImpostaColoreSfondo 0 3D NascondiTartaruga
  AssegnaVarLocale "time SecondiDaAvvio
  initmap
  AssegnaVar "counter 0
  Se :p=0 [cube 405] [menger 405 :p]
  # displays the HMS per build the sponge
  Scrivi "numero\ poligoni:\  Stampa :counter
```

```
  Scrivi "time:\  Stampa SecondiDaAvvio -:time
  VistaPoligono3D
fine

# diverse sezioni per menger di ordine 2
per coupe1 :size
  Ripeti 4 [carpet :size/3 1 PennaSu Avanti :size RuotaDestra 90 PennaGiu]
fine

per coupe0 :size
  carpet :size 2
fine

# diverse sezioni per menger di ordine 3

per coupe10 :size
  Ripeti 4 [carpet :size/3 2 PennaSu Avanti :size RuotaDestra 90 PennaGiu]
fine

per coupe01 :size
  Ripeti 4 [Ripeti 2 [coupe1 :size/3 PennaSu Avanti :size/3 PennaGiu]
  Avanti :size/3 RuotaDestra 90]
fine

per coupe11 :size
  Ripeti 4 [coupe1 :size/3 PennaSu Avanti :size RuotaDestra 90 PennaGiu]
fine


per coupe00 :size
  carpet :size 3
fine

# diverse sezioni per menger di ordine 4
per coupe000 :size
  carpet :size 4
fine

per coupe100 :size
  Ripeti 4 [carpet :size/3 3 PennaSu Avanti :size RuotaDestra 90 PennaGiu]
fine

per coupe010 :size
  Ripeti 4 [Ripeti 2 [coupe10 :size/3 PennaSu Avanti :size/3 PennaGiu]
  Avanti :size/3 RuotaDestra 90]
fine

per coupe001 :size
  Ripeti 4 [Ripeti 2 [coupe01 :size/3 PennaSu Avanti :size/3 PennaGiu]
  Avanti :size/3 RuotaDestra 90]
fine

per coupe110 :size
  Ripeti 4 [coupe10 :size/3 PennaSu Avanti :size PennaGiu RuotaDestra 90 ]
fine

per coupe111 :size
  Ripeti 4 [coupe11 :size/3 PennaSu Avanti :size RuotaDestra 90 PennaGiu]
fine

per coupe101 :size
```

```
  Ripeti 4 [coupe01 :size/3 PennaSu Avanti :size RuotaDestra 90 PennaGiu]
fine

per coupe011 :size
  Ripeti 4 [Ripeti 2 [coupe11 :size/3 PennaSu Avanti :size/3 PennaGiu]
  Avanti :size/3 RuotaDestra 90]
fine

per coupe :size
  carpet :size 1
fine

per cube :size
  Ripeti 2 [
  ImpostaColorePenna Blu rec :size :size PennaSu Avanti :size
  BeccheggiaGiu 90 PennaGiu
  ImpostaColorePenna Giallo rec :size :size PennaSu Avanti :size
  BeccheggiaGiu 90  PennaGiu
  ]
  ImpostaColorePenna Magenta
  PennaSu RollioSinistra 90 RuotaSinistra 90 Avanti :size
  RuotaDestra 90 PennaGiu rec :size :size
  PennaSu RuotaDestra 90 Avanti :size RuotaSinistra 90
  RollioDestra 90 RuotaDestra 90 Avanti :size RuotaSinistra 90
  RollioDestra 90 PennaGiu rec :size  :size
  RollioSinistra 90 RuotaSinistra 90 Avanti :size RuotaDestra 90
fine
```

Occorre impostare la memoria disponibile per XLogo a 640 MB: `spugna 4`.



Capitolo 15

Argomento: il sistema Lindenmayer

Livello: Avanzato

In questo capitolo trarrò informazioni da:

- La pagina inglese di Wikipedia circa i sistemi-L: `http://en.wikipedia.org/wiki/L-System`.
- Il libro "The Algorithmic Beauty of Plants" scritto da by Przemyslaw Prusinkiewicz e Aristid Lindenmayer.

Questa sezione tratterà dei sistemi Lindemayer o sistemi L introdotti e sviluppati nel 1968 dal biologo teoretico Lindenmayer. Un sistema-L è un sistema di regole e simboli utilizzato per modellizare i processi di crescita dello sviluppo delle piante ma anche è anche capace di modellizare la morfologia di una varietà di organismi. Il concetto principale dei sistemi L è il "regole di sostituzione". Questa tecnica è utilizzata per sostituire alcune condizioni iniziali utilizzando alcune regole.

15.1 definizione formale

Un sistema-L è una grammatica formale con:

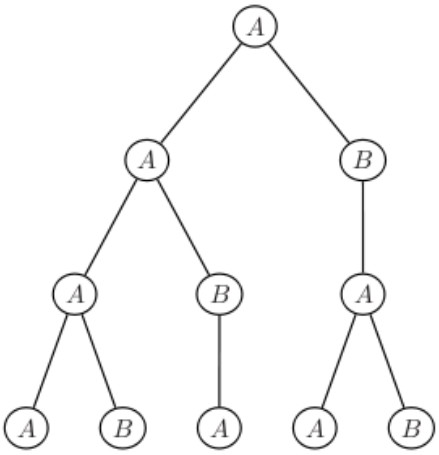
1. un **alfabeto** V: l'insieme delle variabili del sistema-L. $V*$ rappresenta l'insieme delle "parole" che possiamo generare con qualsiasi simbolo preso dall'alfabeto V, e $V+$ l'insieme delle "parole" con almeno un simbolo.
2. Un insieme di valori **costanti** S. Alcuni di questi simboli sono in comune a tutti i sistemi L (in particolare con le tartarughe).
3. Un **assioma di partenza** ω preso da $V+$, è lo stato iniziale.
4. Un insieme di **regole** di produzione P dei simboli V.

Questo sistema-L è definito come una ennupla (o tupla) $\{V, S, \omega, P\}$.

Consideriamo il seguente sistema-L:

- Alfabeto : $V = \{A, B\}$
- Constanti : $S = \{\emptyset\}$
- Assioma iniziale: $\omega = A$
- Regole : $\boxed{\begin{array}{l} A \to AB \\ B \to A \end{array}}$

Le due regole di produzione sono regole di sostituzione. Ad ogni passo il simbolo A è sostituito dalla sequenza AB ed il simbolo B è sostituito da A. Ecco le prime iterazioni di questo sistema-Lindemayer:



1. A
2. AB
3. ABA
4. $ABAAB$

Ok, ok ma concretamente? Leggiamo la prossima sezione!

15.2 L'interpretazione della tartaruga

Questo primo esempio aiuta a capire cos'è un sistema-L ma ancora non ci fa capire il rapporto con la nostra tartaruga and Logo...
Qui diventa interessante: tutte le parole che abbiamo costruito non hanno significato. Definiremo per ciascuna lettera della sequenza una azione da eseguire da parte della tartaruga e disegneremo con questo metodo disegni 2D e 3D.

15.2.1 Simboli usuali

- F : Av di un passo di una unità ($\in V$)
- $+$: Gira l'angolo a sinistra α ($\in S$).
- $-$: Gira l'angolo a destra α ($\in S$).
- & : Beccheggia giù α ($\in S$).
- ^: Beccheggia sù α ($\in S$).
- \: Rollio a sinistra α ($\in S$).
- /: Rollio a destra α ($\in S$).
- |: Torna indietro. In XLogo: `DX 180`

Per esempio se $\alpha = 90$ con un passo di 10 unità abbiamo:

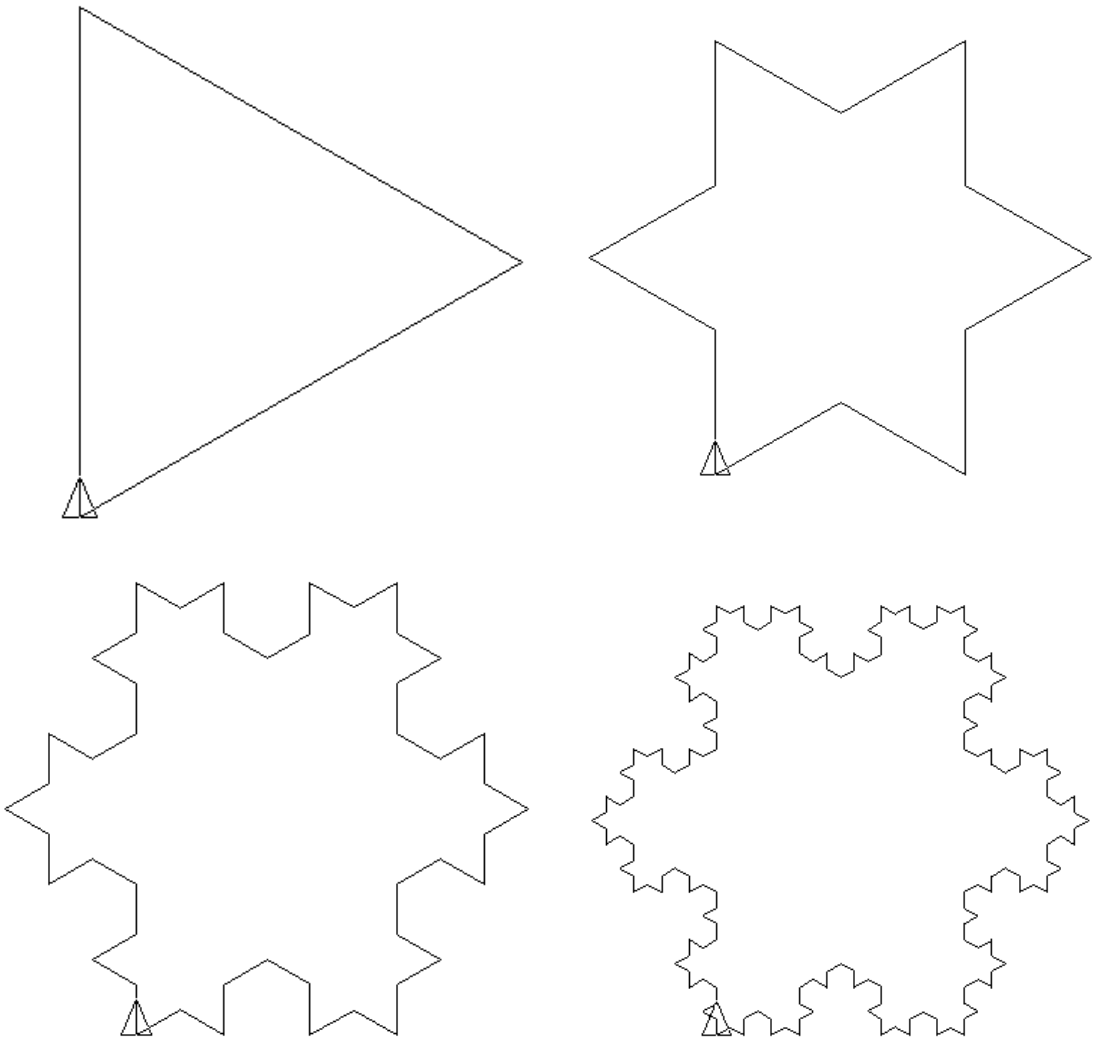
Simbolo	F	$+$	$-$	&	^	\	/	|
Comando XLogo	`Av 10`	`SX 90`	`DX 90`	`BG 90`	`BS 90`	`RSX 90`	`RDX 90`	`DX 180`

15.2.2 Van FioccoDiNeve

Consideriamo il sistema-L:

- Stato iniziale: $F--F--F--$
- Regole di produzione: $F \to F+F--F+F$
- Angolo $\alpha = 60°$, l'unità del passo è divisa per 3 tra ciascuna iterazioni.

Prime iterazioni:



Programma XLogo:

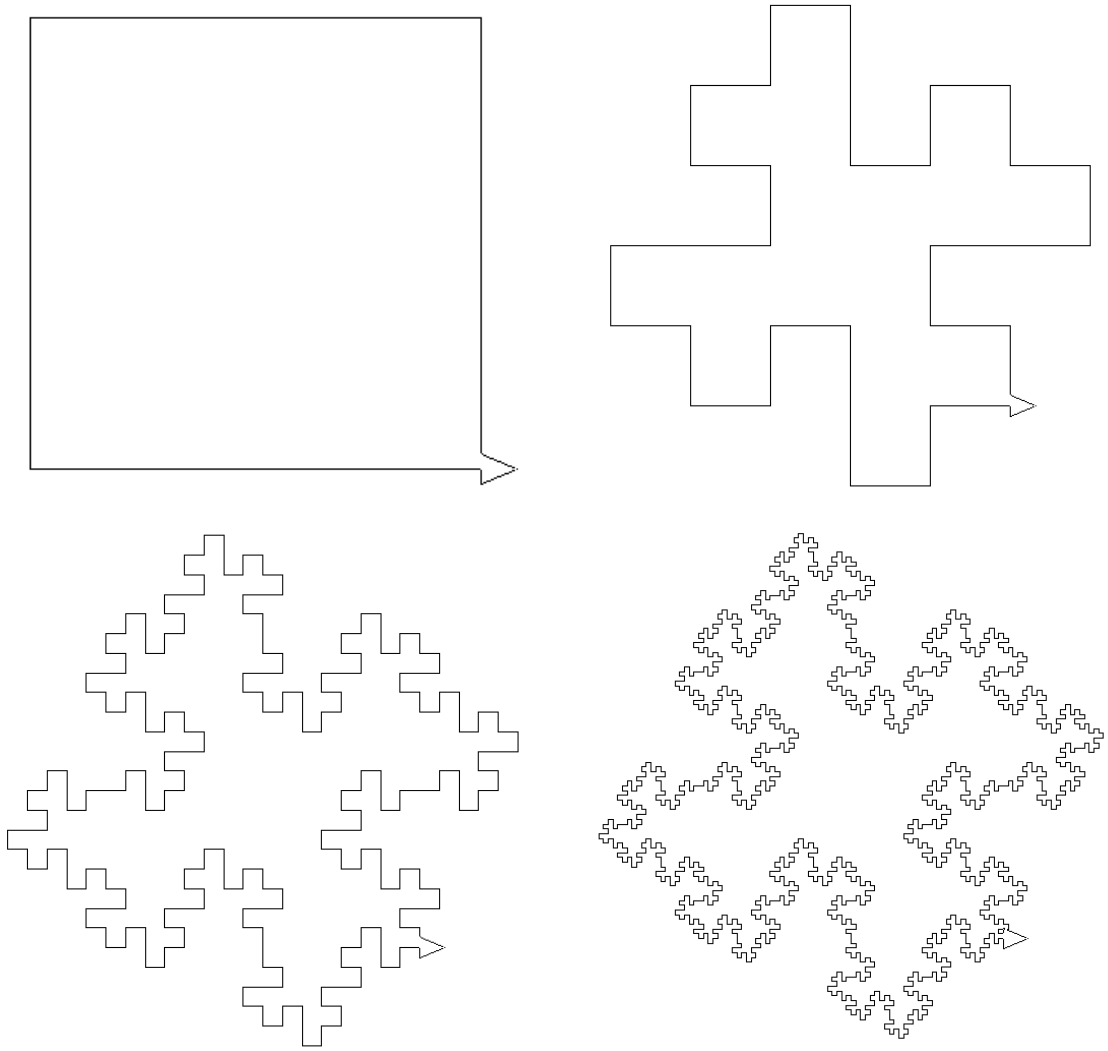
```
Per FioccoDiNeve :p
  AssegnaVar "unit 300/Potenza 3 :p-1
  Ripeti 3 [f :p-1 DX 120]
Fine

Per f :p
  Se :p=0 [Av :unit Ferma]
  f :p-1 SX 60 f :p-1 DX 120 f :p-1 SX 60
  f :p-1
Fine
```

15.2.3 Curva quadratica di Van Koch

Dato questo nuovo sistema-L:

- Stato iniziale: $F - F - F - F$
- Regole di produzione: $F \rightarrow F - F + F + FF - F - F + F$

Queste sono le prime rappresentazioni utilizzando $\alpha = 90$ (we adjust the unit step for the figure has a constant size).



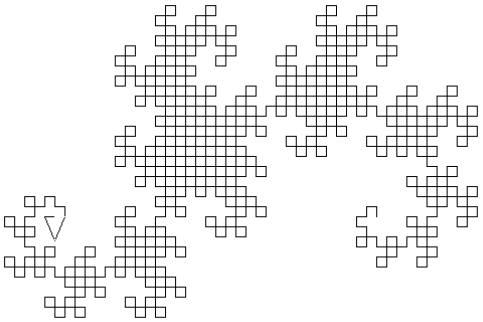
E' molto semplice creare un programma Logo per generare questi disegni:

```
# p rappresenta l'ordine
per koch :p
  # tra 2 iterazioni l'unita' del passo e' divisa per 4
  # la figura finale avra' una dimensione massima di 600x600
   AssegnaVar "unit 300/Potenza 4 :p-1
   Ripeti 3 [f :p-1 SX 90] f :p-1
fine

# regole di sostituzione
per f :p
   Se :p=0 [Av :unit Ferma]
   f :p-1 SX 90 f :p-1 DX 90 f :p-1 DX 90
   f :p-1 f :p-1 SX 90 f :p-1 SX 90 f :p-1 DX 90 f :p-1
fine
```

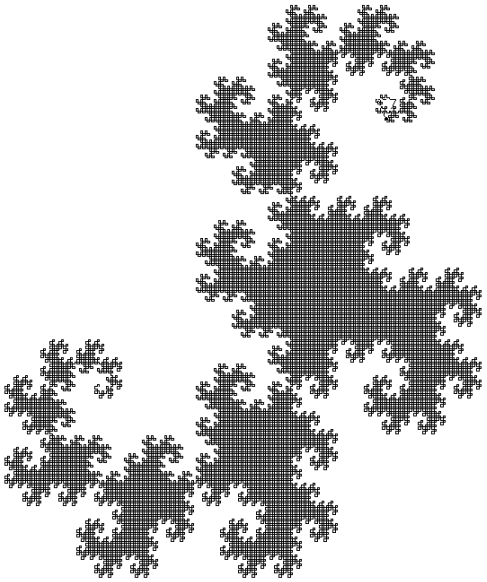
15.2.4 Curva del dragone

- Stato iniziale: F
- Regole di produzione: $\boxed{\begin{array}{l} A \rightarrow A+B+ \\ B \rightarrow -A-B \end{array}}$

```
per a :p
  Se :p=0 [Av :unit Ferma]
  a :p-1 SX 90 b :p-1 SX 90
fine

per b :p
  Se :p=0 [Av :unit Ferma]
  DX 90 a :p-1 DX 90 b :p-1
fine

per dragon :p
  AssegnaVar "unit 300/8/ :p
  a :p
fine
```



dragon 10



dragon 15

15.2.5 Curva 3D di Hilbert

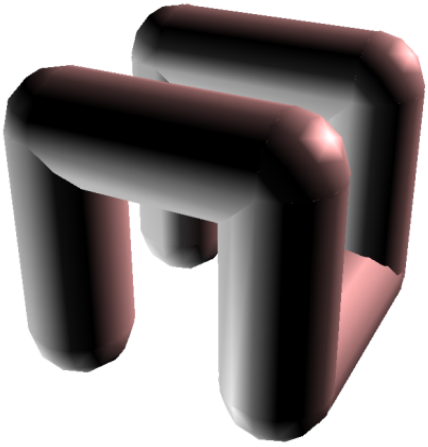
Il seguente esempio genererà una curva 3D di Hilbert. E' una curva singolare perché riempie perfettamente un cubo quando aumentiamo le iterazioni.

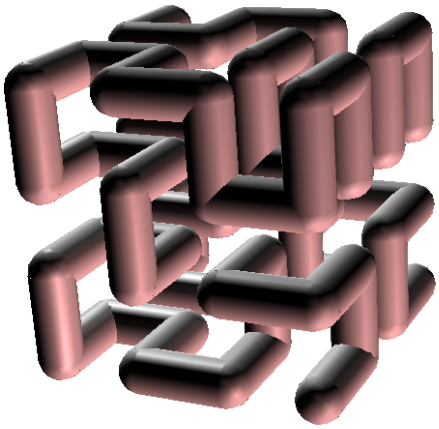
Questo è il sistema-L da considerare:

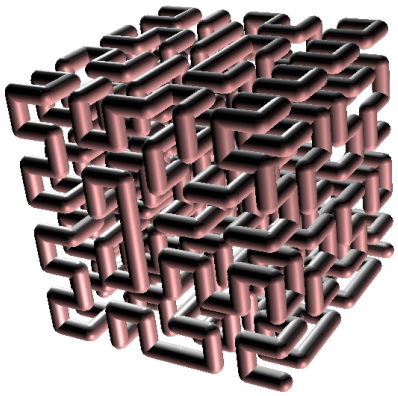
- Stato iniziale: A
- Angolo $\alpha = 90°$, l'unità del passo è divisa per 2 tra due iterazioni.
- Regole di produzione: $\boxed{\begin{array}{l} A \rightarrow B-F+CFC+F-D\&F\hat{}D-F+\&\&CFC+F+B// \\ B \rightarrow A\&F\hat{}CFB\hat{}F\hat{}D\hat{}\hat{}-F-D\hat{}|F\hat{}B|FC\hat{}F\hat{}A// \\ C \rightarrow |D\hat{}|F\hat{}B-F+C\hat{}F\hat{}A\&\&FA\&F\hat{}C+F+B\hat{}F\hat{}D// \\ D \rightarrow |CFB-F+B|FA\&F\hat{}A\&\&FB-F+B|FC// \end{array}}$

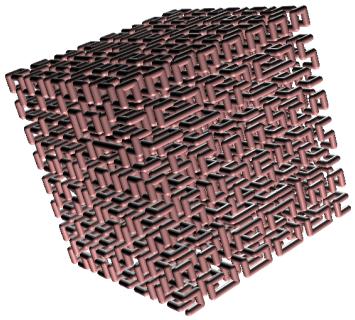
```
per hilbert :p
  PS 3D
  AssegnaVar "unit 400/Potenza 2 :p
  FineLinea ImpSP :unit/2
  a :p
  InizioLinea
  VistaPoligono3D
fine

per a :p
  Se :p=0 [Ferma]
  b :p-1 DX 90 Av :unit SX 90  c :p-1 Av :unit c :p-1
  SX 90 Av :unit DX 90 d :p-1 BG 90 Av :unit BS 90 d :p-1
  DX 90 Av :unit SX 90 BG 180 c :p-1 Av :unit c :p-1
  SX 90 Av :unit SX 90 b :p-1 RDX 180
fine

per b :p
  Se :p=0 [Ferma]
  a :p-1 BG 90 Av :unit BS 90 c :p-1 Av :unit b :p-1 BS 90
  Av :unit BS 90 d :p-1 BS 180 DX 90 Av :unit DX 90 d :p-1
  BS 90 DX 180 Av :unit BS 90 b :p-1 DX 180 Av :unit c :p-1
  BS 90 Av :unit BS 90 a :p-1 RDX 180
fine

per c :p
  Se :p=0 [Ferma]
  DX 180 d :p-1 BS 90 DX 180 Av :unit BS 90 b :p-1 DX 90
  Av :unit SX 90 c :p-1 BS 90 Av :unit BS 90 a :p-1 BG 180
  Av :unit a :p-1 BG 90 Av :unit BS 90 c :p-1 SX 90 Av :unit
  SX 90 b :p-1 BS 90 Av :unit BS 90 d :p-1 RDX 180
fine

per d :p
  Se :p=0 [Ferma]
  DX 180 c :p-1 Av :unit b :p-1 DX 90 Av :unit SX 90 b :p-1 DX 180
  Av :unit a :p-1 BG 90 Av :unit BS 90 a :p-1 BG 180 Av :unit
  b :p-1 DX 90 Av :unit SX 90 b :p-1 DX 180 Av :unit c :p-1 RDX 180
fine
```

E le prime iterazioni:









Bello, non è vero?

Appendice A

Elenco delle primitive

La tartaruga è comandata tramite comandi interni chiamati 'primitive'. Le seguenti sezioni descrivono le primitive di XLogo.

A.1 Movimento della tartaruga, impostazioni del tratto e del colore

A.1.1 Movimento

Queste prime primitive governano il movimento della tartaruga.

Avanti, Av *n*
Sposta la tartaruga in avanti di n passi, nella direzione in cui si trova.

Indietro, In *n*
Sposta la tartaruga indietro di n passi, nella direzione in cui si trova.

RuotaDestra, DX *n*
Ruota la tartaruga di n gradi verso la sua destra (rispetto la direzione in cui si trova).

RuotaSinistra, SX *n*
Ruota la tartaruga di n gradi verso la sua sinistra (rispetto la direzione in cui si trova).

Cerchio, Circonferenza *R*
Disegna una circonferenza di raggio R avente come centro la tartaruga.

Arco *R ang1 ang2*
Disegna una arco di circonferenza di raggio R attorno alla tartaruga. L'arco è viene tracciato dall'angolo ang1 all'angolo ang2.

Origine
Riporta la tartaruga alla sua posizione iniziale ossia alle coordinate [0 0] con direzione 0 gradi.

ImpostaPosizione, ImpPos *elenco*
Sposta la tartaruga alle coordinate x e y specificate nell'elenco (x si riferisce all'asse delle x, y all'asse delle y).
Esempio: `ImpPos [50 30]`

ImpostaX, ImpX *x*
Sposta la tartaruga orizzontalmente fino al punto x sull'asse delle ascisse (X).

ImpostaY, ImpY *y*
Sposta la tartaruga verticalmente fino al punto y sull'asse delle ordinate (Y).

ImpostaXY, ImpXY *x y*
Identico a `ImpPos [x y]`

ImpostaDirezione, ImpDir *n*
Orienta la tartaruga nelle direzione n (0-360 gradi), dove 0 corrisponde alla direzione verticale verso l'alto (Nord), 90 alla direzione orizzontale verso destra (Est) e così via in base all'orientamento della bussola.

Etichetta *arg*
Disegna arg a partire dalla posizione della tartaruga, ruotato di 90 gradi verso destra. arg può essere una parola od un elenco.
Esempio: `Etichetta [Ciao lì fuori!]` scriverà la frase "Ciao lì fuori!" ovunque la tartaruga sia, in direzione orizzontale.
Esempio: `SX 90 Etichetta [Ciao lì fuori!]` scriverà la frase "Ciao lì fuori! ovunque la tartaruga sia, in direzione verticale verso l'alto.

Punto *elenco*
Illumina nel colore della penna il punto identificato dalle coordinate nell'elenco.
Esempio: `Punto [100 50]` Illumina il punto di coordinate x 100 e y 50.

A.1.2 Proprietà della tartaruga

In questo secondo gruppo rientrano le primitive che impostano le proprietà della tartaruga. Per esempio la tartaruga è visibile sullo schermo? Di che colore dovrebbe essere il tratto quando si sposta?

MostraTartaruga, MT
Rende visibile la tartaruga sullo schermo.

NascondiTartaruga, NT
Rende la tartaruga invisibile sullo schermo.

PulisciSchermo, PSc
Svuota l'area di disegno.

Pulisci
Svuota l'area di disegno ma lascia la tartaruga nello stesso posto.

ResettaTutto, InizializzaTutto
Riporta l'interfaccia di XLogo ai valori iniziali e svuota l'area di disegno. I valori iniziali sono:

- Colore della penna: nero
- Forma della penna: quadrata
- Qualità del tratto: normale

- Colore dello schermo: bianco
- Dimensioni dello schermo: 1000x1000
- Modalità animazione: disabilitata
- Font del testo e della grafica: Dialog 12 punti
- Numero tartarughe ammesse: 16
- Modalità traccia: disabilitata

PennaGiu, PennaGiu, PG
Quando la penna è abbassata la tartaruga traccia il percorso che compie mentre si sposta.

PennaSu, PennaSu, PS
Quando la penna è alzata la tartaruga si sposta senza tracciare il proprio percorso.

CancellaPenna, CP
La tartaruga cancella qualsiasi segno incontro al proprio passaggio.

InvertiPenna, InvPenna
Abbassa la penna e pone la tartaruga in modalità inversa.

PennaDisegno, PD
Abbassa la penna e pone la tartaruga nella modalità classica di disegno.

ImpostaColorePenna, ImpCP *colore*
Imposta il colore della penna, cfr. pagina 95 per l'elenco dei colori.

ImpostaColoreSfondo, ImpCS *colore*
Imposta il colore dello sfondo, cfr. pagina 95 per l'elenco dei colori.

Posizione, Pos
Restituisce la posizione attuale della tartaruga.
Esempio: `Pos` ritorna [10 -100]

x
Restituisce la coordinata X della posizione della tartaruga.

y
Restituisce la coordinata Y della posizione della tartaruga.

z
Restituisce la coordinata Z della posizione della tartaruga (Disponibile solamente nella modalità 3D).

Direzione
Restituisce la direzione in gradi della tartaruga (cfr. `ImpostaDirezione`)

Verso *elenco*
Restituisce la direzione verso cui la tartaruga dovrebbe puntare per raggiungere il punto definito dalle coordinate definite in *elenco*. Elenco deve contenere due numeri che rappresentino le coordinate x e y del punto.

Distanza *elenco*
Restituisce il numero di passi che la tartaruga dovrebbe compiere per raggiungere il punto definito dalle coordinate definite in *elenco*. Elenco deve contenere due numeri che rappresentino le coordinate x e y del punto.

ColorePenna, ColP
Restituisce il colore della penna attuale. Il colore è definito come un elenco di tre numeri [r g b] dove r è la componente rossa, b è la componente blu, g è la componente verde.

ColoreSfondo, ColS
Restituisce il colore dello sfondo attuale. Il colore è definito come un elenco di tre numeri [r g b] dove r è la componente rossa, b è la componente blu, g è la componente verde.

Finestra
Si permette alla tartaruga di muoversi fuori dalla area di disegno (senza disegnarci, ovviamente).

Gira
Si permette alla tartaruga di lasciare l'area di disegno e riapparire dal lato opposto!

Recinta
Non si permette alla tartaruga di lasciare l'area di disegno. Nel momento in cui sta per uscirne comparirà un messaggio di errore che informerà del numero massimo di passi che la tartaruga può compiere prima di raggiungere il punto di uscita.

Prospettiva, 3D
La tartaruga può spostarsi in uno spazio a 3 dimensioni (cfr. la sezione speciale A.14 per questa modalità). Per tornare alla modalità classica 2D, utilizzare una delle 3 primitive `Finestra`, `Gira` or `Recinta`

CercaColore, CC *elenco*
Restituisce il colore del punto nell'area di disegno identificato dalle coordinate x e y fornite in elenco *elenco*. Il colore è definito come un elenco di tre numeri [r g b] dove r è la componente rossa, b è la componente blu, g è la componente verde.

ImpSpessorePenna, ImpSP *n*
Imposta lo spessore della penna della tartaruga in pixel (o passi). Lo spessore iniziale è 1. La penna ha una forma quadrata o rotonda (altre forme saranno introdotte in versioni future di XLogo).

SpessorePenna, SP
Restituisce lo spessore nella penna della tartaruga in punti (o passi).

ImpFormaPenna, ImpFP *0-1*
Imposta la forma della penna, quadrata o rotonda:

- 0→quadrata.
- 1→rotonda.

FormaPenna, FP
Restituisce la forma della penna.

- 0→quadrata.
- 1→rotonda.

ImpQualitaDisegno, ImpQD *0-1-2*
Imposta la qualità del disegno :

- 0→normale.
- 1→alta.

- 2→bassa.

QualitaDisegno, QD
Restituisce la qualità del disegno.

- 0→normale.
- 1→alta.
- 2→bassa.

ImpDimensioneSchermo *elenco*
Imposta la dimensione dell'area di disegno ai valori x e y definiti in *elenco*. `ImpDimensioneSchermo [1000 1000]`

DimensioneSchermo
Restituisce un elenco di due numeri con la dimensioni dell'area di disegno.

ImpostaForma, ImpFo *n*
Si può scegliere la forma della tartaruga nel secondo tab della finestra aperta da Opzioni-Preferenze... Ma si può impostare la tartaruga preferita con `ImpostaForma`. Il numero *n* varia da 0 a 6 (0 rappresenta la forma triangolare).

Forma
Restituisce il numero che rappresenta la forma della tartaruga.

ImpostaCorpoFont, ImpCFont *n*
Quando si scrive nell'area di disegno con la primitiva `Etichetta`, si può modificare il corpo (ossia le dimensioni) del font iniziale che è 12.

CorpoFont
Restituisce il corpo (le dimensioni) del font impostato per quando si scrive nell'area di disegno con la primitiva `Etichetta`.

ImpostaNomeFont, ImpNFont *n*
Imposta il numero *n* del font quando si scrive nell'area di disegno con la primitiva `Etichetta`. L'elenco dei font ed i relativi numeri *n* da utilizzare si trova nel Tab *Font* del menu Strumenti→Preferenze.

ImpostaAllineamentoTesto, ImpAllTesto *elenco*
Imposta l'allineamento attorno alla tartaruga da utilizzare quando si scrive nell'area di disegno con la primitiva `Etichetta`. *elenco* contiene due numeri interi.

- Il primo numero si riferisce all'allineamento orizzontale:
  - 0: allineamento a sinistra
  - 1: allineamento al centro
  - 2: allineamento a destra
- Il secondo numero si riferisce all'allineamento verticale:
  - 0: allineamento in basso
  - 1: allineamento al centro
  - 2: allineamento in alto

Tutti i possibili casi che si possono verificare sono di seguito elencati. `ImpCFont 50 Etichetta XLogo`

 `ImpAllTesto [2 0]`	 `ImpAllTesto [1 0]`	 `ImpAllTesto [0 0]`
 `ImpAllTesto [2 1]`	 `ImpAllTesto [1 1]`	 `ImpAllTesto [0 1]`
 `ImpAllTesto [2 2]`	 `ImpAllTesto [1 2]`	 `ImpAllTesto [0 2]`

AllineamentoTesto
Restituisce un elenco di due elementi che rappresenta l'allineamento del testo attorno alla tartaruga, quando si utilizza `Etichetta` per scrivere nell'area di disegno.

NomeFont, NF
Restituisce un elenco di due elementi. Il primo è il numero corrispondente al font utilizzato da `Etichetta`. Il secondo elemento è un elenco che contiene il nome del font.

ImpostaSeparazione,ImpSep *n*
Determina il rapporto tra l'area di disegno e l'area dello storico dei comandi. Il numero n deve essere incluso tra 0 e 1. Quando n è uguale a 1 l'intera area è occupata dall'area di disegno, quando n è uguale a 0 l'area con lo storico dei comandi usa tutto lo spazio.

Separazione,Sep
Restituisce il rapporto attuale tra l'area di disegno e l'area con lo storico dei comandi.

Griglia *a b*
Disegna una griglia, con ciascun rettangolo avente dimensioni a e b.

CancellaGriglia,CancGriglia
Elimina la griglia eventualmente disegnata.

ImpostaColoreGriglia,ImpColGriglia *colore*
Permette di impostare il colore desiderato per la griglia. Il colore è definito come un elenco di tre numeri [r g b] dove r è la componente rossa, b è la componente blu, g è la componente verde.

ColoreGriglia
Restituisce il colore della griglia attuale.

Griglia?
Restituisce vero se la griglia è disegnata, altrimenti restituisce falso.

Assi *n*
Disegna gli assi orizzontale e verticale. La distanza tra due divisioni è n passi.

AsseX *n*
Disegna l'asse orizzontale. La distanza tra due divisioni è n passi.

AsseY *n*
Disegna l'asse verticale. La distanza tra due divisioni è n passi.

CancellaAssi,CancAssi
Cancella entrambi gli assi.

ImpostaColoreAssi, ImpColAssi *colore*
Imposta il colore con cui disegnare gli assi. Il colore è definito come un elenco di tre numeri [r g b] dove r è la componente rossa, b è la componente blu, g è la componente verde.

ColoreAssi
Restituisce il colore attuale degli assi.

AsseX?
Restituisce vero se l'asse orizzontale è disegnato, altrimenti restituisce falso.

AsseY?
Restituisce vero se l'asse verticale è disegnato, altrimenti restituisce falso.

ImpostaZoom,ImpZoom *a*
Ingrandisce l'area di disegno di un fattore *a* che rappresenta la scala rispetto alle dimensioni originali dell'immagine definite nel pannello delle preferenze.

Zoom
Restituisce l'attuale scala di zoom.

LunghezzaEtichetta,LE *arg*
Restituisce la lunghezza del testo da scrivere nell'area di disegno con la primitiva `Etichetta`, utilizzando il font attuale.

DimensioneZona
Retituisce un elenco che contiene quattro numeri interi. L'elenco rappresenta le coordinate dell'angolo sinistro in alto e le coordinate dell'angolo destro in basso.

Messaggio, msg *elenco*
Visualizza una finestra di dialogo con il messaggio in *elenco*. L'esecuzione del programma viene fermata fino a ché l'utente clicca il bottone OK.

A.1.3 Qualche parola sui colori

I colori sono definiti in XLogo come un elenco di tre numeri `[r g b]`, ciascuno compreso tra 0 and 255, dove r è la componente rossa, b è la componente blu, g è la componente verde. XLogo ha 16 colori predefiniti i quali possono essere richiamati con il loro elenco rgb o con una primitiva:

Numeri	Primitive	[R G B]	Color
0	`Nero`	[0 0 0]	
1	`Rosso`	[255 0 0]	
2	`Verde`	[0 255 0]	
3	`Giallo`	[255 255 0]	
4	`Blu`	[0 0 255]	
5	`Magenta`	[255 0 255]	
6	`Ciano`	[0 255 255]	
7	`Bianco`	[255 255 255]	
8	`Grigio`	[128 128 128]	
9	`GrigioChiaro`	[192 192 192]	
10	`RossoScuro`	[128 0 0]	

Numeri	Primitive	[R G B]	Color
11	`GrigioScuro`	[0 128 0]	
12	`BluScuro`	[0 0 128]	
13	`Arancio`	[255 200 0]	
14	`Rosa`	[255 175 175]	
15	`Violetto`	[128 0 255]	
16	`Marrone`	[153 102 0]	

```
# Queste tre istruzioni sono identiche
ImpCS Arancio
ImpCS 13
ImpCS [255 200 0]
```

A.1.4 Modalità Animazione

Le seguenti sono tre primitive che consentono l'esecuzione dei comandi in modo più veloce del solito. La tartaruga non si muoverà più fino a che non le viene imposto di ridisegnare lo schermo mediante la primitiva *Aggiorna*. Il vantaggio della modalità Animazione consiste nella maggiore velocità di disegno che consente di creare animazioni.

Animazione
La tartaruga entra nella modalità Animazione. Da questo momento non verrà disegnato niente sullo schermo se non quando si utilizza la primitiva `Aggiorna`.

FermaAnimazione
La tartaruga esce dalla modalità Animazione.

Aggiorna,Ridisegna
In modalità Animazione aggiorna l'area di disegno per riflettere tutte le primitive eseguite dal momento dell'ingresso nella modalità.

Per identificare la modalità Animazione, una icona con una telecamera appare nella finestra dello storico dei comandi. Se l'icona viene cliccata si esce dalla modalità.

A.1.5 Scrivere nell'area di testo con le primitive `Stampa` e `Scrivi`

La seguente tabella espone le primitive che permettono di modificare le proprietà dell'area di testo. Le primitive che controllano il colore e le dimensioni dell'area dello storico dei comandi influenzano solo le primitive `Stampa` o `Scrivi`

PulisciTesto, PT
Svuota l'area dello storico dei comandi.

Stampa, St *arg*
Scrive l'argomento *arg* nell'area dello storico dei comandi.

```
Stampo "abcd # abcd
St [1 2 3 4] #1 2 3 4
St 4 # 4
```

Scrivi *arg1*
Come la primitiva `Stampa` scrive *arg1* nell'area dello storico dei comandi ma senza andare a capo.

ImpostaDTesto, ImpDT *n*
Imposta il corpo del font nell'area dello storico dei comandi. Disponibile solo per la primitiva `Stampa`

DimensioneTesto, DT
Restituisce il corpo del font per la primitiva `Stampa`.

ImpostaColoreTesto, ImpCT *colore*
Imposta il colore del font nell'area dello storico dei comandi. Disponibile solo per la primitiva . Confronta pag. 95.

ColoreTest, CT
Restituisce il colore del font nell'area dello storico dei comandi.

ImpostaNomeTesto, ImpNTesto *n*
Seleziona il font numero *n* per la scrittura con la primitiva `Stampa` nell'area dello storico dei comandi. Il collegamento tra il font ed il suo numero si può trovare nel Menu→Strumenti→Preferenze→Tab Font.

NomeTesto, NT
Restituisce il font utilizzato nell'area dello storico dei comandi sotto forma di lista. Il primo elemento della lista corrisponde al numero del font utilizzato, il secondo elemento è una lista che riporta il nome del font stesso.

ImpostaStile, ImpSt *arg*
Imposta il formato del testo nell'area dello storico dei comandi. Si può scegliere tra `Nessuno, Grassetto, Corsivo, Barrato, Sottolineato, Apice, Pedice`. Per impostare più stili contemporaneamente raggrupparli in un elenco.
Alcuni esempi:

```
ImpostaStile [Grassetto Sottolineato] Stampa "Ciao
```
Ciao
```
ImpSt "Barrato Scrivi [Barrato] ImpSt "Corsivo Scrivi "\x ImpSt "Apice Stampa 2
```
~~Barrato~~ x^2

Stile, st
Restituisce un elenco contenente i diversi stili utilizzati per la primitiva `Stampa`.

A.2 Operazioni matematiche

Somma *x y*
Addiziona i due numeri x e y e ne restituisce il risultato.
Esempio: `Somma` 40 60 restituisce 100

Differenza *x y*
Restituisce $x - y$.
Esempio: `Differenza` 100 20 restituisce 80

Meno *x*
Restituisce il negativo di x.
Esempio: `Meno` 5 restituisce -5. Leggere la nota alla fine del paragrafo.

Prodotto *x y*
Restituisce il risultato della moltiplicazione di x per y.

Div, Dividi *x y*
Restituisce il risultato della divisione di x per y
Esempio: `Div` 3 6 restituisce 0.5

Quoziente x y

Restituisce il quoziente della divisione di x per y

`Quoziente` 15 6 restituisce 2

Resto x y

Restituisce il resto della divisione di x per y.

mod, modulo x y

Restituisce x modulo y. La seguente tabella dimostra la differenza tra `modulo x y` e `Resto x y`.

x	y	`Resto x y`	`modulo x y`
14	5	4	4
-14	5	-4	1
14	-5	4	-1
-14	-5	-4	-4

Arrotonda x

Restituisce il più vicino numero intero a x.

Istruzione	Risultato
`Arrotonda 8.9`	9
`Arrotonda 6.8`	7
`Arrotonda 6.2`	6

Intero, Int x

Restituisce la parte intera del numero x.

Istruzione	Risultato
`Intero 8.9`	8
`Intero 6.8`	6
`Intero 6.2`	6

Potenza x n

Restituisce x elevato alla potenza di n.

Esempio: `Potenza` 3 2 restituisce 9

RadiceQuadrata, RadQ x

Restituisce la radice quadrata di x.

Log x

Restituisce il logaritmo naturale (a base e) di x.

Exp x

Restituisce il valore della base del logaritmo naturale (e), elevato alla potenza dell'esponente x.

Log10 x

Restituisce il logaritmo a base 10 di x.

Seno, sen x

Restituisce il seno di x (x è espresso in gradi sessagesimali).

Coseno, cos x

Restituisce il coseno di x (x è espresso in gradi sessagesimali).

Tangente, tan x

Restituisce la tangente di x (x è espresso in gradi sessagesimali).

ArcoCoseno, acos x

Restituisce l'angolo sessagesimale (0-180°) il cui coseno è x.

ArcoSeno, asen x
Restituisce l'angolo sessagesimale il cui seno è x.

ArcoTangente, atan x
Restituisce l'angolo sessagesimale la cui tangente è x.

pi
Restituisce il numero di pi greco (3.141592653589793).

Casuale n
Restituisce un numero intero compreso tra 0 e $n - 1$.

Casuale01
Restituisce un numero casuale compreso tra 0 e 1.

Assoluto, Ass x
Restituisce il valore assoluto del numero x (il numero privo di segno).

ImpostaDecimali, ImpDec n
Restituisce il numero di decimali utilizzati nel calcolo. In dettaglio:

- 16 decimali è l'impostazione predefinita.
- Se n è negativo, viene reimpostato il valore predefinito.
- Se n è uguale a 0 tutti i numeri sono arrotondati all'unità.

Questa primitiva è utile nel calcolo ad alta precisione. Confronta l'esempio con il numero pi greco a pagina 45.

Decimali
Restituisce il numero di decimali permessi nel calcolo. Il valore predefinito è -1.

Importante: Attenzione alle primitive che richiedono due parametri!
`ImpXY` a b Se b è negativo
Esempio: Per esempio, `setxy` 200 -10

L'interprete Logo eseguirà l'operazione 200-10 (sottrarrà 10 da 200). Esisterà quindi un solo parametro (190) laddove ne occorreranno due e si genererà un messaggio di errore. Per evitare questo tipo di problemi bisognerà utilizzare la primitiva "`meno`" per specificare un numero negativo. Esempio: `ImpXY` 200 `meno` 10.

A.3 Operazioni logiche

o *b1 b2*
Restituisce vero se *b1* o *b2* sono veri, altrimenti restituisce falso.

e *b1 b2*
Restituisce vero se *b1* e *b2* sono veri, altrimenti restituisce falso.

not *b1*
Restituisce la negazione di *b1*.

- Se *b1* è vero, restituisce false.
- Se *b1* è falso, restituisce vero.

A.4 Operazioni sugli elenchi e sulle parole

Le seguenti primitive manipolano gli elementi degli elenchi o le lettere delle parole. Gli esempi di utilizzo delle primitive sono posti alla fine del paragrafo.

Parola *arg1 arg2*
Restituisce una parola concatenando *arg1* e *arg2*.

Elenco *arg1 arg2*
Restituisce un elenco composto da *arg1* e *arg2*.

Frase *arg1 arg2*
Restituisce un elenco composto da *arg1* e *arg2*. Se *arg1* o *arg2* è un elenco esso stesso, ciascun elemento di *arg1* e *arg2* diventerà un elemento dell'elenco risultante (le parentesi quadre vengno rimosse).

InserisciElementoInizioElenco, IElenco *arg1 elenco1*
Restituisce un elenco inserendo *arg1* come primo elemento in *elenco1*.

AggiungiElementoFineElenco, FElenco *arg1 elenco1*
Restituisce un elenco inserendo *arg1* come ultimo elemento in *elenco1*.

Inverso *elenco*
Inverte l'ordine degli elementi in *elenco*.

Scegli *arg1*
Restituisce un elemento casuale se *arg1* è un elenco o una lettera casuale se *arg1* è una parola.

Rimuovi *arg1 elenco1*
Restituisce un elenco da *elenco1* dal quale l'elemento *arg1*, se presente, è rimosso.

Elemento *n arg2*
Restituisce la lettera n-esima se *arg2* è una parola o l'elemento n-esimo se *arg2* è un elenco.

EccettoUltimo, EU *arg1*
Restituisce una parola privata dell'ultima lettera se *arg1* è una parola o un elenco privato dell'ultimo elemento se *arg1* è un elenco.

EccettoPrimo, EP *arg1*
Restituisce una parola privata della prima lettera se *arg1* è una parola o un elenco privato del primo elemento se *arg1* è un elenco.

Ultimo *arg1*
Restituisce una parola costituita dall'ultima lettera se *arg1* è una parola o l'ultimo elemento se *arg1* è un elenco.

Primo *arg1*
Restituisce una parola costituita dalla prima lettera se *arg1* è una parola o il primo elemento se *arg1* è un elenco.

Sostituisci, ImpostaElemento *elenco1 n arg3*
Sostituisce l'elemento n-esimo nell'elenco *elenco1*, con la parola o elenco *arg3*.

AggiungiElemento, Aggiungi, AE *elenco1 n arg3*
Aggiunge alla posizione n-esimo nell'elenco *elenco1* la parola o l'elenco *arg3*

Conta *arg1*
Restituisce il numero di lettere se *arg1* è una parola o il numero di elementi se *arg1* è un elenco.

UniCode *carattere1*
Restituisce il valore unicode del carattere *carattere1*.

Carattere, Car *n*
Restituisce il carattere il cui valore unicode è n.

A.4.1 Esempi di utilizzo

Istruzione	Risultato
`Stampa Parola "a 1`	`"a1`
`Stampa Elenco 3 6`	`[3 6]`
`Stampa Elenco "a "elenco`	`[a elenco]`
`Stampa Frase [4 3] "ciao`	`[4 3 ciao]`
`Stampa Frase [come vanno] "le cose`	`[come vanno le cose]`
`Stampa IElenco "cocoa [2]`	`[cocoa 2]`
`Stampa FElenco "cocoa [2]`	`[2 cocoa]`
`Stampa Inverso [1 2 3]`	`[3 2 1]`
`Stampa Rimuovi "ciao [1 2 ciao 3]`	`[1 2 3]`
`Stampa Sostituisci [a b c] 2 8`	`[a 8 c]`
`Stampa Aggiungi [a b c] 2 8`	`[a 8 b c]`
`Stampa UniCode "A`	`65`
`Stampa Carattere 65`	`"A`

A.5 Booleani

Un booleano è una primitiva che restituisce la parola "vero o la parole "falso. Le primitive booleane terminano con il punto interrogativo.

vero
Restituisce "vero.

falso
Restituisce "falso.

Parola? *arg1*
Restituisce vero se *arg1* è una parola, falso altrimenti.

Numero? *arg1*
Restituisce vero se *arg1* è un numero, falso altrimenti.

Intero? *arg1*
Restituisce vero se *arg1* è un numero intero, falso altrimenti.

Elenco? *arg1*
Restituisce vero se *arg1* è un elenco, falso altrimenti.

Vuoto? *arg1*
Restituisce vero se *arg1* è un elenco vuoto o una parola vuota, falso altrimenti.

Uguale? *arg1 arg2*
Restituisce vero se *arg1* e *arg2* sono uguali, falso altrimenti.

Prima? *parola1 parola2*
Restituisce vero se *parola2* si trova prima di *word2* nell'ordine alfabetico, falso altrimenti.

Membro? *arg1 arg2*
Restituisce vero se *arg1* è un elemento dell'elenco *arg2* o se *arg1* è una lettera della parola *arg2*.

Membro *arg1 arg2*
Restituisce una parola o un elenco a partire dalla prima occorrenza di *arg1* in *arg2*.

- Se *arg2* è una parola restituisce la parte finale della parola a cominciare dalla prima occorrenza di *arg1*, falso se *arg1* non è contenuto in *arg2*.
- Se *arg2* è un elenco restituisce un elenco contenente tutti gli elementi dalla prima occorrenza di *arg1* in *arg2*, falso se non esiste occorrenza.

Istruzione	Risultato
`Stampa Membro 3 [1 2 3 4]`	`[3 4]`
`Stampa Membro "o "cocoa`	`"ocoa`

PennaGiu?, PennaGiu, PG
Restituisce la parola vero se la penna è poggiata sull'area di disegno, falso altrimenti.

Visibile?
Restituisce vero se la tartaruga è visibile, falso altrimenti

Primitiva? *parola1*
Restituisce vero se *parola1* è una primitiva XLogo, falso altrimenti.

Procedura? *parola1*
Restituisce vero se *parola1* è una procedura definita nel programma logo, falso altrimenti.

Variabile? Var? *parola1*
Restituisce vero se *parola1* è una variabile, falso altrimenti.

A.6 Verifica delle espressioni con la primitiva `Se`

Come in tutti i linguaggi di programmazione, Logo permette di verificare se una condizione è soddisfatta o meno e quindi eseguire le istruzioni desiderate in un caso o nell'altro.
Con la primitiva `Se` si possono realizzare queste verifiche.

Se *espressione_ da_ verificare elenco1 elenco2*
Se `espressione_da_verificare` è vero, le istruzioni incluse in `elenco1` vengono eseguite. Altrimenti, se

`espressione_da_verificare` è falso, le istruzioni in `elenco2` vengono eseguite. Questo secondo elenco è opzionale.

Esempi:

- `Se 1+2=3[Stampa "vero][Stampa "falso]`
- `Se (Primo "XLOGO)=Y [Av 100 DX 90] [Stampa [ XLOGO comincia con una X!] ]`
- `Se (3*4)=6+6 [Stampa 12]`

Importante: Quando il risultato della verifica è uguale a `falso`, la primitiva `Se` cerca il secondo elenco. In alcuni rari casi la ricerca non è possibile e si dovrà utilizzare la primitiva `SeAltrimenti` . Per esempio:

```
Per test
  # Inseriamo due liste nelle variabili a e b
  AssegnaVar "a [Stampa "vero]
  AssegnaVar "b [Stampa "falso]

  # Test con la primitiva Se, la seconda lista non viene ricercata.
  Se 1=2 :a :b
Fine
```

L'esempio precedente genera un errore *Non so cosa fare con [Stampa "falso] ?*. Il seguente esempio invece funziona correttamente:

```
Per test
  # Inseriamo due liste nelle variabili a e b
  AssegnaVar "a [Stampa "vero]
  AssegnaVar "b [Stampa "falso]

  # Test con la primitiva SeAltrimenti --> la seconda lista viene cercata ed eseguita.
  SeAltrimenti 1=2 :a :b
Fine
```

A.7 I cicli

Un ciclo permetto di eseguire un blocco di primitive o procedure per un numero determinato o meno di volte. Il vantaggio dei cicli è quello di far svolgere compiti ripetitivi al programma invece che al programmatore. Invece di disegnare i quattro lati del quadrato così:

```
  Avanti 100 RuotaDestra 90
  Avanti 100 RuotaDestra 90
  Avanti 100 RuotaDestra 90
  Avanti 100 RuotaDestra 90
```

possiamo utilizzare un ciclo:

```
  Ripeti 4 [Avanti 100 RuotaDestra 90]
```

XLogo possiede sette primitive che permettono la costruzione di cicli: `Ripeti`, `RipetiPer` , `Mentre`, `RipetiPerCiascuno`, `RipetiPerSempre`, `RipetiIntantoChe` e `RipetiFinoAChe`.

A.7.1 `Ripeti`

Ripeti *n elenco_di_comandi*

Ripete `elenco_di_comandi` *n* volte.
n è un numero intero e `elenco_di_comandi` è un elenco contenente i comandi da eseguire. L'interprete LOGO implementerà i comandi nell'elenco *n* volte, evitando di dover copiare gli stessi comandi *n* volte!
Esempio:

```
Ripeti 4 [Avanti 100 RuotaDestra 90]        # Un quadrato di lato 100
Ripeti 6 [Avanti 100 RuoraDestra 60]        # Un esagono di lato 100
Ripeti 360 [Avanti 2 RuotaDestra 1]         # Un poligono di 360 lati di lunghezza 2,
                                            # cioe' una circonferenza!
```

ContaRipetizione

E' una variabile interna e restituisce il numero della iterazione attuale (a partire da 1), deve essere incluso in un ciclo `Ripeti`.

```
Ripeti 3 [Stampa ContaRipetizione]
# 1
# 2
# 3
```

A.7.2 `RipetiPer`

RipetiPer *elenco1 elenco2*

`RipetiPer` esegue un ciclo assegnando ad una variabile valori successivi all'interno di un intervallo fisso, con incrementi definiti.
elenco1 è un elenco di quattro elementi:

1. il nome della variabile
2. il valore di partenza dell'intervallo
3. il valore terminale dell'intervallo
4. il valore dell'incremento (elemento opzionale nell'elenco, se non viene fornito viene assunto il valore di 1 come incremento predefinito)

Alcuni esempi:

```
RipetiPer [i 1 4][Stampa :i*2]
# Visualizzera' a schermo i numeri
# 2
# 4
# 6
# 8
```

```
# i scendera' da 7 a 2, con un decremento di 1,5
# Visualizzera' il quadrato di i
RipetiPer [i 7 2 -1.5 ][Stampa Elenco :i Potenza :i 2]
# 7 49
# 5.5 30.25
# 4 16
# 2.5 6.25
```

A.7.3 `Mentre`

Mentre *elenco_da_verificare elenco_di_comandi*

Al contrario di `Ripeti` e `RipetiPer`, la primitiva `Mentre` esegue un ciclo per un numero di volte che può non essere conosciuto a priori. *elenco_da_verificare* è un elenco contenente un set di istruzioni che devono essere verificate come booleane. *elenco_di_comandi* è un elenco contenente i comandi da eseguire all'interno del ciclo. L'interprete LOGO continuerà l'esecuzione di *elenco_di_comandi* fintanto ché *elenco_da_verificare* restituisce "vero.
Esempi:
La tartaruga ruota a destra all'infinito:

```
Mentre ["vero] [RuotaDestra 1]
```

Viene percorso l'alfabeto all'indietro:

```
AssegnaVar "alfa "abcdefghilmnopqrstuvz
Mentre [non Vuoto? :alfa]
  [Stampa Ultimo :alfa AssegnaVar "alfa EccettoUltimo :alfa]
```

A.7.4 RipetiPerCiascuno

RipetiPerCiascuno *nome_variabile arg1 istruzioni*

Esegue un ciclo per ciascun elemento di `arg1` se è un elenco o per ciascun carattere se `arg1` è una parola. Le istruzioni contenute nell'elenco `istruzioni` sono ripetute per ciascun valore della variabile `nome_variabile`.

Esempi: Stampa i caratteri in "XLOGO":

```
RipetiPerCiascuno "i "XLOGO [Stampa :i]
```

Stampa gli elementi dell'elenco "[a b c]":

```
RipetiPerCiascuno "i [a b c] [Stampa :i]
```

Somma tutti i numeri da 1 a 5:

```
AssegnaVar "sum 0
RipetiPerCiascuno "i 12345 [AssegnaVar "sum :sum+:i] Stampa :sum
```

A.7.5 RipetiPerSempre

RipetiPerSempre *elenco_istruzioni*

Ripete per sempre un blocco di istruzioni in attesa di un comando per fermare il ciclo (cfr. pagina 116).

```
RipetiPerSempre [Avanti 1 RuotaDestra 1]
```

Attenzione usando questa primitiva ai clicli infiniti!

A.7.6 RipetiIntantoChe

RipetiIntantoChe *elenco1 elenco2*

Ripete un blocco di istruzioni contenuto in *elenco1* mentre *elenco2* è vero.
La differenza con la primitiva `Mentre` è che il blocco di istruzioni è eseguito almeno una volta anche se *elenco2* è falso perché la verifica dell'espressione avviene dopo l'esecuzione del blocco di istruzioni.

Per esempio vediamo la differenza tra i due tipi di ciclo eseguendo lo stesso blocco di istruzioni: `RipetiIntantoChe` esegue il ciclo 5 volte:

```
AssegnaVar "i 0
RipetiIntantoChe [Stampa :i AssegnaVar "i :i+1] [:i<4]
```

`Mentre` esegue il ciclo 4 volte:

```
AssegnaVar "i 0
Mentre [:i<4][Stampa :i AssegnaVar "i :i+1]
```

A.7.7 RipetiFinoACche

RipetiFinoACche *elenco1 elenco2*

Ripete il blocco di istruzioni contenuto in *elenco1* fino a ché *elenco2* diventa vero. Per esempio scriviamo l'esempio precedente per essere eseguito con `RipetiFinoACche`:

```
AssegnaVar "i 0
RipetiFinoACche [Stampa :i AssegnaVar "i :i+1] [:i>4]
```

A.8 L'area di lavoro

L'area di lavoro contiene tutti gli elementi definiti dall'utente:

- Le procedure
- Le variabili
- Gli elenchi di proprietà

A.8.1 Le procedure

Le procedure sono una sorta di "programma". Ogni procedura ha un nome scelto dall'utente, quando il nome di una procedura viene invocato, le istruzioni contenute nel corpo della procedura sono eseguite. Il corpo delle procedure comincia con la primitiva `Per` e termina con la primitiva `Fine`.

```
Per nome_della_procedura :v1 :v2 :v3 .... [:v4 ....] [:v5 ....]
  Corpo_della_procedura
Fine
```

- nome_della_procedura è il nome scelto per identificare la procedura.
- :v1 :v2 :v3 rappresentano i parametri che la procedura utilizzerà al suo interno (variabili locali), il loro contenuto è determinato al momento di invocare la procedura stessa.
- [:v4 ...], [:v5 ...] sono variabili opzionali che si possono aggiungere (leggi più sotto per spiegazioni più dettagliate).
- Corpo_della_procedura rappresenta l'elenco dei comandi da eseguire quando la procedura viene invocata.

Esempio:

```
Per Quadrato :s
  Ripeti 4[Av :s DX 90]
Fine
```

La procedura è chiamata `Quadrato` e richiede un parametro chiamato `s`. `Quadrato 100` disegnerà quindi un quadrato di lato 100 (altri esempi di procedure sono al termine del manuale).

E' possibile inserire dei commenti nel codice preceduti dal simbolo # (cancelletto).

```
Per Quadrato :s
  # Questa procedura permette di disegnare un quadrato di lato s
  Ripeti 4[Av :s DX 90]
Fine
```

Parametri opzionali

In XLogo è possibile aggiungere parametri opzionali alle procedure. Un parametro opzionale è una variabile che è possibile o meno passare alla procedura al momento della sua invocazione. Se non viene passato, la procedura utilizzerà per esso un valore preimpostato. Per esempio:

```
Per poly :n [:l 10]
  # Questa procedura disegna un poligono regolare di n lati lunghi l passi
  Ripeti :n [Av :l DX 360/:n]
end
```

Durante l'esecuzione, la variabile `:l` viene rimpiazzata dal suo valore preimpostato (10, come definito in `[:l 10]`). Se si vuole disegnare un poligono regolare con lati di lunghezza diversa dal valore preimpostato basterà invocare la procedura `poly` specificando il valore di `l` ed inserendo l'invocazione in parentesi tonde. Per esempio:

```
# Questo comando disegna un poligono regolare di 20 lati lunghi 5 passi
(poly 20 5)
# Questo è un quadrato di 100 passi di lato
(poly 4 100)
```

La modalità di tracciamento

E' possibile seguire il funzionamento del programma e delle procedure durante l'esecuzione. Questa modalità può anche mostrare se la procedura fornisce argomenti grazie alla primitiva `Output`. Tracciare un programma che funziona in modo non corretto può essere molto utile per capire dove risiedano gli errori di programmazione.

traccia
Attiva la modalità di tracciamento.

FermaTraccia
Disattiva la modalità di tracciamento.

Un piccolo esempio con il programma fattoriale (pagina 45).

```
traccia Stampa fac 4
#fac 4
#  fac 3
#    fac 2
#      fac 1
#      fac restituisce 1
#    fac restituisce 2
#  fac restituisce 6
#fac restituisce 24
#24
```

A.8.2 Il concetto di variabili

Le variabili sono come dei contenitori che possono essere riempiti di contenuti come parole, caratteri, numeri. Ci sono due tipi di variabili:

- Globali: le variabili globali sono accessibili da qualunque posizione nel programma.
- Locali: le variabili locali sono accessibili nella procedura dove sono state definite.

In questa versione di Logo, le variabili locali non sono accessibili alle sotto-procedure. Alla fine della procedura le variabili locali sono eliminate.

Le variabili si creano e ne viene assegnato il contenuto con la primitiva `AssegnaVar`.

AssegnaVar *parola1 arg2*
Se la variabile locale *parola1* esiste, ne assegna il valore *arg2*. Se non esiste crea una variabile globale *parola1*

e ne assegna il valore *arg2*.
Esempio: `AssegnaVar "a 100` assegna il valore 100 alla variabile a.

VarLocale *arg1*
Crea una variabile locale chiamata *arg1*. Se *arg1* è un elenco, crea tante variabili quanti sono gli elementi nell'elenco. Da notare che le variabili sono solo create, nessun valore viene loro assegnato.

AssegnaVarLocale *parola1 arg2*
Crea una nuova variabile locale e ne assegna il valore *arg2*.

ValoreVar *parola1*
Restituisce il valore contenuto nella variabile *parola1*. `ValoreVar "a` è simile a `:a`

Definisci, Def *parola1 elenco1*
Definisce una nuova procedura chiamata *parola1*.
`elenco1` contiene diversi elenchi:

- Il primo elenco contiene tutti i parametri ed i parametri opzionali.
- I successivi elenchi rappresentano ciascuno una linea della procedura.

```
Def "poligono [ [nb l] [Ripeti :nb[Av :l DX 360/:nb] ]
```

Questo comando definisce una procedura chiamata `poligono` che accetta due parametri (`:nb` e `:l`). La procedura disegna un poligono regolare di cui possiamo scegliere il numero di lati e la loro lunghezza.

Testo *parola1*
Restituisce tutte le informazioni sula procedura chiamata `parola1`. Restituisce un elenco che contiene numerosi elenchi.

- Il primo elenco contiene tutte i parametri e parametri opzionali.
- Quindi ciascun elenco rappresenta una linea della procedura.

Questa primitiva è ovviamente associata a `Definisci`.

CancellaProcedura, CancProc *arg1*
Cancella la procedura chiamata *arg1* o tutte le procedure chiamate come gli elementi dell'elenco *arg1*.

CancellaVariabile, CancVar *arg1*
Cancella la variabile *arg1* o tutte le variabili chiamate come gli elementi dell'elenco *arg1*.

CancEP, CancellaElencoProprietà, CancellaElencoProprietal *arg1*
Cancella l'elenco proprietà *arg1* o tutte gli elenchi proprietà chiamati come gli elementi dell'elenco *arg1*.

CancellaTutte, CancTutte
Cancella tutte le variabili, elenchi proprietà e procedure attualmente in funzione.

ciao
Chiude XLogo.

Procedure, Procs
Restituisce un elenco con tutte le procedure attualmente definite.

Variabili, Vars
Restituisce un elenco con tutte le variabili definite.

ElencoProprietà, ElencoProprieta
Restituisce un elenco con tutti gli elenchi proprietà attualmente definiti.

Primitive
Enumera tutte tutte le primitive definite nel linguaggio attuale.

Contenuti
Restituisce un elenco che contiene tre elenchi. Il primo elenco contiene tutti i nomi delle procedure, il secondo tutti i nomi delle variabili, il terzo tutti gli elenchi proprietà.

Lancia *elenco1*
Esegue l'elenco di istruzioni contenute in `elenco1`.

ComandoEsterno *elenco1*
Esegue un comando di sistema da XLogo. *elenco1* deve contenere diversi elenchi che rappresentano qualsiasi parola costituente il comando. Alcuni esempi:

`ComandoEsterno [[gedit][/home/xlogo/file.txt]]`
Lancia l'applicazione `gedit` e carica il file `/home/xlogo/file.txt` (Linux)

`ComandoEsterno [[notepad][C: /file.txt]]`
Lancia l'applicazione `notepad` e carica il file chiamato `C: /file.txt` (Windows)

La sintassi del comando permette l'inserimento di spazi bianchi nel percorso del file.

A.8.3 Gli elenchi di proprietà

E' possibile definire elenchi proprietà in XLogo. Ciascun elenco di proprietà ha un suo nome e contiene coppie chiave-valore specifici (chiamati proprietà).

Per esempio, si può definire un elenco proprietà chiamato "macchina" contenente la chiave "colore" associato al valore "rosso" o la chiave "trazione" con il valore "4x4".

Per gestire questi elenchi si possono utilizzare le seguenti primitive.

AggiungiProprieta, AggiungiProprietà *nomeelenco chiave valore*
Aggiunge una proprietà all'elenco di proprietà chiamato *nomeelenco*. *valore* sarà accessibile con la chiave *chiave*. Se non esiste alcun *nomeelenco* verrà creato..

ElencaProprieta, ElencaProprietà *nomeelenco chiave*
Restituisce il valore associato alla chiave *chiave* nell'elenco di proprietà chiamato *nomeelenco*. Se questo elenco di proprietà non esiste o se non esiste alcuna chiave valida, restituisce un elenco vuoto.

CancellaProprieta, CancellaProprietà *nomeelenco chiave*
Cancella la proprietà corrispondente alla chiave *key* nell'elenco proprietà *nomeelenco*.

ElencaProprieta, ElencaProprietà *nomeelenco*
Elenca tutte le proprietà contenute nell'elenco proprietà *nomeelenco*.

Torniamo all'elenco di proprietà chiamato "macchina".

```
# Creiamo l'elenco di proprieta'
AggiungiProprieta "macchina "colore "red
AggiungiProprieta "macchina "trazione "4x4
AggiungiProprieta "macchina "marca "Citroen

# Mostriamo un valore
Stampa ElencaProprieta "macchina "colore
# rosso

# Mostriamo tutti i valori
Stampa ElencaProprieta "macchina
# colore rosso trazione 4x4 marca Citroen
```

A.9 Lavorare con i file

File
Come comportamento predefinito, elenca il contenuto della cartella (come il comando `ls` in Linux e `dir` in DOS).

CaricaImmagine, CI *elenco*
Carica il file immagine contenuto in `elenco`. Il suo angolo in alto a sinistra viene posto alla posizione attuale della tartaruga. I formati immagine supportati sono .png e .jpg. Il percorso specificato deve essere relativo alla cartella attuale. Esempio: `ImpDir "C:\\my_images_dir CaricaImmagine "turtle.jpg`

SalvaImmagine *parola elenco*
Salva l'immagine dell'area di disegno nel file specificato in `parola` nella cartella attuale.
I formati immagine supportati sono .png e .jpg. Se l'estensione non viene specificata l'immagine verrà salvata nel formato .png.
L'elenco `elenco` può contenere quattro numeri ($[X_{min}Y_{min}X_{max}Y_{max}]$) che rappresentano i due angoli della zona dell'area di disegno da salvare. Un elenco vuoto è equivalente a salvare l'intera immagine. Per esempio:

```
## Nell'editor
Per test
  # Salva tutte le immagini come 1.png, 2.png ... 20.png
  Ripeti 20 [
    Avanti 30 DX 18
      SalvaImmagine Parola ContaRipetizioni ".png  [-50 150 200 -150]
  ]
Fine

## Sulla linea di comando:
test
PulisciSchermo NascondiTartaruga Ripeti 20 [CaricaImmagine Parola ContaRipetizioni ".png]
```

Avete creato una breve animazione!

ImpostaDirectory, ImpDir *parola1*
Imposta la cartella attuale. Il percorso deve essere assoluto. La cartella deve essere specificata come una

parola.

CambiaDirectory, CD *parola1*
Imposta la cartella attuale relativamente a quella attiva al momento. Si può utilizzare `..` notazione per indicare la cartella a monte.

Directory, Dir
Restituisce la cartella attuale. La cartella preimpostata è quella dell'utente, cioè `/home/login` per utenti Linux, `C:\WINDOWS` per utenti Windows.

Salvato *parola1*
Salva tutte le procedure logo definite nel file `parola1` nella cartella attuale. Se l'estensione .lgo viene omessa verrà aggiunta automaticamente. `parola1` può contenere un percorso relativo alla cartella attuale, non un percorso assoluto.

Salva *parola1 elenco1*
Salve le procedure logo definite in `elenco1` nel file `parola1`. Se l'estensione .lgo viene omessa verrà aggiunta automaticamente. `parola1` può contenere un percorso relativo alla cartella attuale, non un percorso assoluto. Per esempio: `Salva test.lgo [proc1 proc2 proc3]` salva nel file `test.lgo` nella cartella attuale le procedure `proc1, proc2, proc3`.

Carica *parola1*
Apre e legge il file logo *parola1*. Per esempio, per cancellare tutte le procedure definite e caricare il file `test.lgo`, si userà: `CancellaTutte Carica test.lgo`. *parola1* può contenere un percorso relativo alla cartella attuale ma non un percorso assolto.

ApriFlusso *id file*
Quando si vuole leggere o scrivere in un file, occorre prima aprire un flusso verso il file stesso. L'argomento *file* deve essere il nome del file desiderato. *id* è un numero assegnato al flusso per identificarlo nelle successive operazioni

ElencaFlussi
Elenca i vari flussi aperti con i loro id.

LeggiLineaDalFlusso *id*
Legge una linea dal file corrispondente al flusso `id`.

LeggiCarattereDalFlusso *id*
Legge un carattere dal file corrispondente al flusso `id`. Restituisce un numero che rappresenta il valore del carattere (in modo simile a `LeggiCarattere`).

ScriviLineaNelFlusso *id list*
Scrive la linea di testo inclusa in *list* all'inizio del file identificato da *id*. Attenzione, la effettiva scrittura avverrà dopo la chiusura del flusso con la primitiva `ChiudiFlusso`.

AggiungiLineaNelFlusso *id list*
Scrive la linea di testo inclusa in *list* alla fine del file identificato da *id*. Attenzione, la effettiva scrittura avverrà dopo la chiusura del flusso con la primitiva `ChiudiFlusso`.

ChiudiFlusso *id*
Chiude il flusso identificato da *id*.

FineFlusso? *id*
Restituisce `vero` se si è al termine del file identificato da id, altrimenti restituisce `falso`.

Di seguito si espone un semplice programma per leggere e scrivere in un file. E' pensato per utenti di sistemi Windows ma gli altri utenti lo potranno facilmente adattare.

Lo scopo dell'esempio è creare il file `c:\esempio` contenente le seguenti tre righe:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
Abcdefghijklmnopqrstuvwxyz
0123456789

```
# Apriamo un flusso verso il file desiderato e lo identifichiamo con il flusso 2
ImpostaDirectory "c:\\
ApriFlusso 2 "esempio
# Scriviamo le righe volute
ScriviLineaNelFlusso 2 [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
ScriviLineaNelFlusso 2 [Abcdefghijklmnopqrstuvwxyz]
ScriviLineaNelFlusso 2 [0123456789]
# Chiudiamo il flusso per concludere la scrittura
ChiudiFlusso 2
```

Possiamo verificare se la procedura di scrittura è andata a buon fine:

```
# Apriamo un flusso verso il file desiderato e lo identifichiamo con il flusso 0
ApriFlusso 0 "c:\\esempio
# Leggiamo una riga dopo l'altra dal file
Stampa LeggiLineaDalFlusso 0
Stampa LeggiLineaDalFlusso 0
Stampa LeggiLineaDalFlusso 0
# Chiudiamo il flusso
ChiudiFlusso 0
```

Se volessimo aggiungere la linea "Grandioso:

```
ImpostaDirectory "c:\\
ApriFlusso 1 "esempio
AggiungiLineaNelFlusso 1 [Grandioso]
ChiudiFlusso 1
```

A.10 L'editor

edit *arg1*
Apre l'editor con tutte le procedure specificate nell'elenco *arg1* o nella parola *arg1*.

ModificaTutte, ModTutte
Apre l'editor con tutte le procedure attualmente definite.

A.11 Funzioni avanzate di riempimento delle figure

Tre primitive permettono di colorare una figura. La primitiva `Riempi`, `RiempiZona` e `RiempiPoligono`. .

A.11.1 `Riempi` e `RiempiZona`

Queste primitive permettono di riempire una figura e sono molto simili alle funzioni di riempimento che si trovano nei programmi di grafica e ritocco fotografico. Il riempimento si può estendere fino ai margini dell'area di disegno. Ci sono due regole che devono essere osservate per ottenere risultati corretti dalle due primitive:

1. La penna deve essere abbassata (`PG`).
2. La tartaruga non deve essere posizionata su un punto del medesimo colore con cui si intende riempire l'area.

Vediamo un esempio per comprendere la differenza tra `Riempi` e `RiempiZona`:

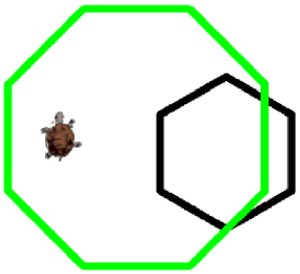


Figura A.1: All'inizio

Il punto sotto la tartaruga è bianco. La primitiva `Riempi` colorerà tutti i punti bianchi vicini con il colore della penna attuale. Digitiamo, per esempio: `ImpCP 1 Riempi`.

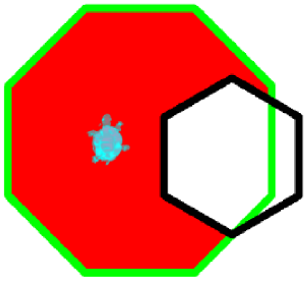


Figura A.2: Con la primitiva `Riempi`

Torniamo al primo caso, se il colore della penna è nero la primitiva `RiempiZona` colorerà tutti i punti fino a ché incontrerà il colore attuale (nero).

Il seguente è un buon esempio di utilizzo di `Riempi`:

```
per halfcirc :c
# disegna una semi-circonferenza di diametro :c
  Ripeti 180 [Avanti :c*Tangente 0.5 RuotaDestra 1]
  Avanti :c*Tangente 0.5
  RuotaDestra 90 Avanti :c
fine
```

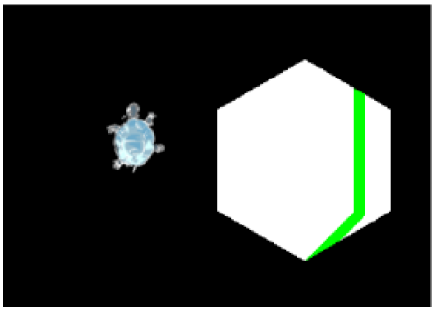


Figura A.3: Con la primitiva `RiempiZona`, se si digita: `ImpCP 0 RiempiZona`

```
per arcobaleno :c
  Se :c<100 [Ferma]
  halfcirc :c RuotaDestra 180 Avanti 20 RuotaSinistra 90
  arcobaleno :c-40
fine

per dep
  PennaSu RuotaDestra 90 Avanti 20 RuotaSinistra 90 PennaGiu
fine

per SemiArco
  NascondiTartaruga PennaSu ImpXY meno 200 meno 200 PennaGiu
  arcobaleno 400
  CancellaPenna RuotaSinistra 90 Avanti 20 Indietro 120 PennaDisegno
  PennaSu RuotaDestra 90 Avanti 20 PennaGiu
  ImpostaColorePenna 0 Riempi dep
  ImpostaColorePenna 1 Riempi dep
  ImpostaColorePenna 2 Riempi dep
  ImpostaColorePenna 3 Riempi dep
  ImpostaColorePenna 4 Riempi dep
  ImpostaColorePenna 5 Riempi dep
  ImpostaColorePenna 6 Riempi dep
fine
```

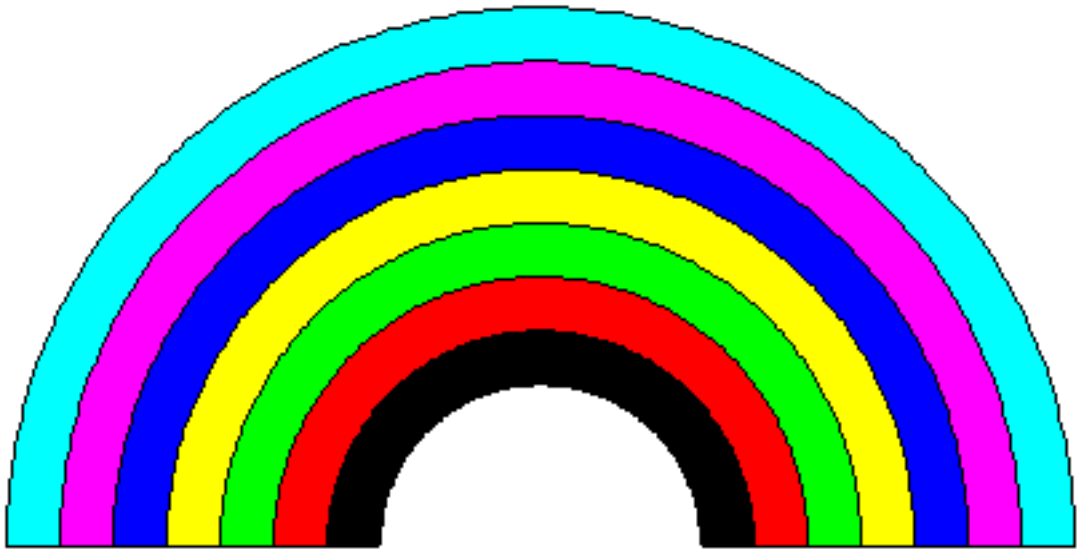


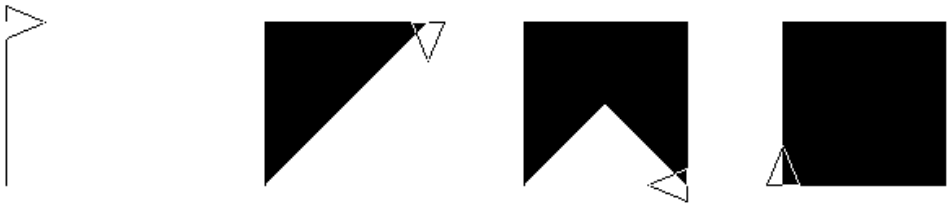
Figura A.4: Arco in LOGO

A.11.2 `RiempiPoligono`

RiempiPoligono *elenco*
Riempie una figura usando una serie di triangoli in modo che ogni volta si disegna una linea, viene riempito il triangolo successivo. L'elenco contiene tutte le istruzioni necessarie per disegnare la figura da riempire.

- Un primo esempio per riempire un quadrato:

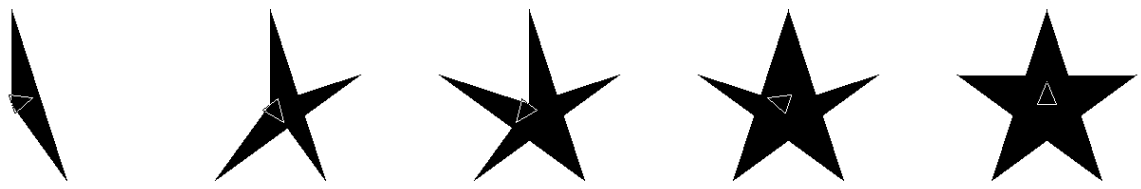
```
PulisciSchermo RiempiPoligono [ Ripeti 4 [Avanti 100 RuotaDestra 90]]
```



Passo 1 Passo 2 Passo 3 Passo 4

- Un secondo esempio che disegna e riempie una stella a cinque punte:

```
Ripeti 5 [Avanti 100 RiempiPoligono [Indietro 100 RuotaDestra 144 Avanti 100 ] Indietro 100 Ru
```



Passo 1 Passo 2 Passo 3 Passo 4 Passo 5

A.12 Comandi di interruzione

Un comando di interruzione può essere impiegato nel programma logo per terminare il programma stesso o una procedura o un ciclo. XLogo ha tre comandi di interruzione: `Ferma`, `FermaTutto` and `Output`.

Ferma
Interrompe l'esecuzione di una procedura o di un ciclo `Ripeti` o `Mentre` a seconda di dove lo si include.

FermaTutto
Il programma esce da tutte le procedure e si interrompe.

Output
Come `Ferma` interrompe una procedura ma, a differenza di questo, permette di restituire un valore.

A.13 Modalità multitartaruga

E' possibile avere più di una tartaruga attiva nell'area di disegno. Per impostazione predefinita, solo una tartaruga è disponibile. Per "creare" una nuova tartaruga si può usare la primitiva `ImpostaTartaruga` seguito dal numero di tartarughe voluto. Quando ci sono più tartarughe sull'area di disegno, solo una sarà quella attiva che risponderà ai comandi. Per prevenire ostacoli la tartaruga è creata all'origine ed è invisibile (occorre usare `MostraTartaruga` per visualizzarla). La nuova tartaruga è la tartaruga attiva, obbedisce a tutte le classiche primitive fino a ché non si cambierà la tartaruga attiva con `ImpostaTartaruga`. Il numero massimo di tartarughe può essere impostato nel Menu Strumenti→Preferenze.

Queste sono le primitive per la modalità multitartaruga:

ImpostaTartaruga, ImpTar n
La tartaruga numero n diventa la tartaruga attiva. Per impostazione predefinita la tartaruga attiva all'avvio di XLogo è la numero 0, la sola visualizzata.

Tartaruga
Restituisce il numero della tartaruga attiva.

Tartarughe
Restituisce un elenco che contiene tutti i numeri delle tartarughe attualmente sull'area di disegno.

CancellaTartaruga, CancT n
Elimina la tartaruga numero n.

ImpNumMaxT, ImpostaNumeroMassimoTartarughe n
Imposta il numero massimo di tartarughe in modalità multitartaruga.

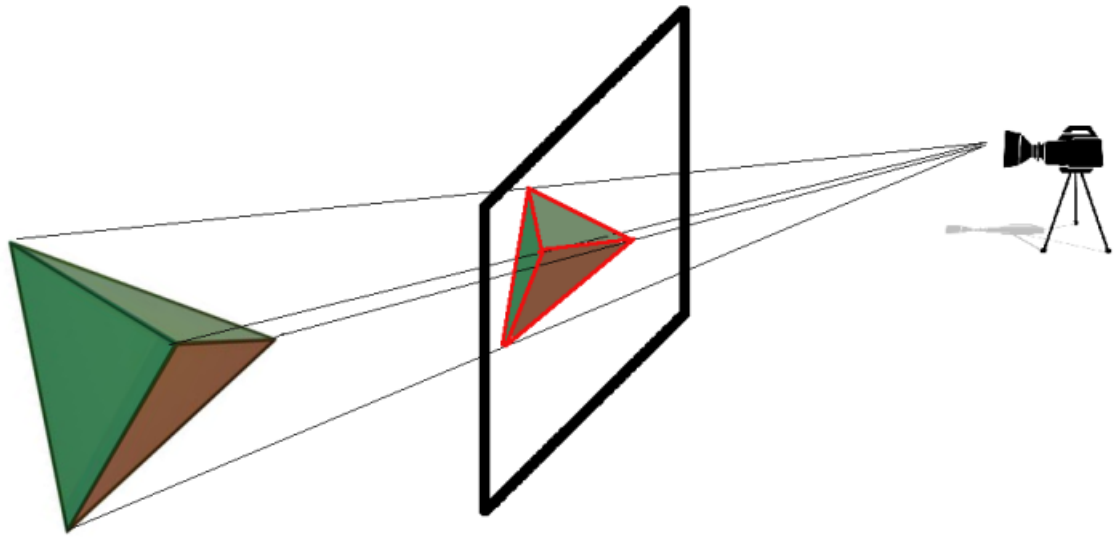
NumMaxT, NumMaxTartarughe
Restituisce il numero massimo di tartarughe nella modalità multitartaruga.

A.14 La tartaruga e le 3 dimensioni

La tartaruga può lasciare il piano di disegno bidimensionale e spostarsi nelle 3 dimensioni. Per entrare nella modalità 3D si utilizza la primitiva `Prospettiva, 3D`. Benvenuti nel mondo 3D!

A.14.1 La proiezione prospettica

Per rappresentare lo spazio 3D su un piano 2D XLogo utilizza la proiezione prospettica. Una cinepresa viene puntata alla scena 3D, dove l'immagine dallo schermo di proiezione viene visualizzata. Ecco un semplice schema esemplificativo:



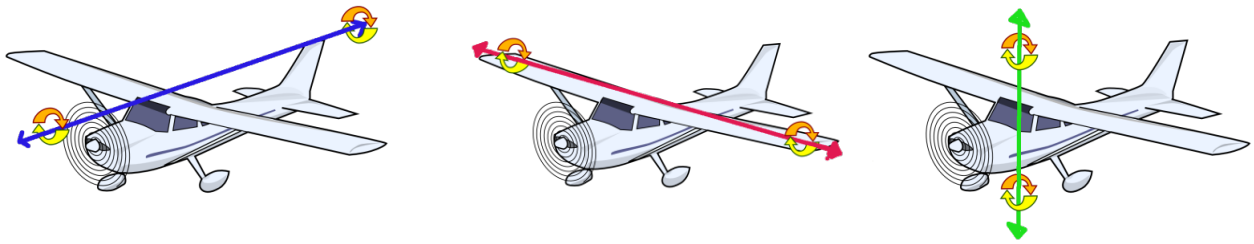
Alcune primitive permettono di impostare la posizione della cinepresa. Lo schermo di proiezione è posto a metà della distanza dalla cinepresa.

A.14.2 Capire l'orientamento in un mondo 3D

In un piano 2D l'orientamento della tartaruga è definito solo dalla sua direzione. In un mondo 3D l'orientamento della tartaruga è definito da 3 angoli:

- Rollio: L'angolo della tartaruga attorno all'asse (Oy)
- Beccheggio: L'angolo della tartaruga attorno all'asse (Ox)
- Direzione: L'angolo della tartaruga attorno all'asse (Oz)

Nei fatti, per spostarsi in un mondo 3D, la tartaruga è molto simile ad un aereo. Di seguito una piccola illustrazione che rappresenta questi 3 valori:



Rollio **Beccheggio** **Direzione**

Può sembrare un pò complesso all'inizio ma molti aspetti tendono ad essere molto simili allo spazio 2D. Queste sono le primitive di base per spostarsi in un mondo 3D:

Avanti, Av, Indietro, In n

Stesso comportamento nel piano 2D e 3D.

Destra, DX, Sinistra, SX n

Stesso comportamento nel piano 2D e 3D.

RollioDestra, RDX *n*
La tartaruga ruota di *n* gradi alla destra attorno al suo asse longitudinale.

RollioSinistra, RSX *n*
La tartaruga ruota di *n* gradi alla sinistra attorno al suo asse longitudinale.

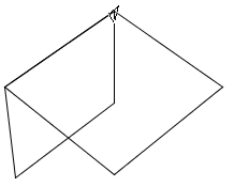
BeccheggiaSu, BeccheggiaSù, BS *n*
La tartaruga ruota di *n* gradi in sù attorno al suo asse trasversale.

BeccheggiaGiu, BeccheggiaGiù, BG *n*
La tartaruga ruota di *n* gradi in giù attorno al suo asse trasversale.
In the 2D plane, when we want to draw a square of side 200 steps, we write:

```
Ripeti 4[Av 200 DX 90]
```

Queste primitive sono disponibili nel mondo 3D dove il quadrato è disegnato in modalità prospettica. Se la tartaruga ruota in giù di 90 gradi possiamo disegnare un altro quadrato e otteniamo:

```
PulisciSchermo
Ripeti 4[Av 200 DX 90]
BeccheggiaGiu 90
Ripeti 4[Av 200 DX 90]
```



Occorre provare qualche esempio per capire questi angoli e diventare un esperto!
Le tre primitive di rotazione sono collegate fra loro, per esempio si può provare questo il seguente codice:

```
PulisciSchermo
RollioSinistra 90 BeccheggiaSu 90 RollioDestra 90
```

Il movimento della tartaruga è equivalente a `Sinistra 90`.

A.14.3 Primitive disponibili sia in modalità 2D sia in 3D

Le seguenti primitive sono disponibili nel piano bidimensionale e tridimensionale. La sola differenza risiede negli argomenti forniti alle primitive. Per esempio la primitiva `ImpostaPosizione` o `ImpPos` si aspetta un elenco come argomento ma, in modalità 3D, l'elenco deve contenere tre numeri $(x; y; z)$ che rappresentano le coordinate rispetto ai 3 assi. Questo è l'elenco di queste primitive:

`Cerchio, Circonferenza`	`Arco`	`Origine`	`Verso`
`Distanza`	`ImpPos, ImpostaPosizione`	`ImpX, ImpostaX`	`ImpY, ImpostaY`
`ImpDir, ImpostaDirezione`	`Etichetta`	`LunghezzaEtichetta, LE`	`Punto`
`Posizione, Pos`	`Direzione`		

A.14.4 Primitive disponibili solo in modalità 3D

ImpostaXYZ, ImpXYZ *x y z*
Imposta la posizione della tartaruga nello spazio 3D. Accetta 3 argomenti ossia le coordinate x, y, z del punto desiderato. `ImpXYZ` è molto simile a `ImpPos` ma, a differenza di quest'ultimo, le coordinate non sono fornite come elenco.
Per esempio, `ImpXYX -100 200 50`: sposta la tartaruga al punto di coordinate $x = -100; y = 200; z = 50$

ImpZ *z*
Sposta la tartaruga al punto con un valido valore z. `ImpZ` richiede un numero come argomento. Questa primitiva è analoga a `ImpX` e `ImpY`.

ImpostaOrientamento, ImpOr *elenco*

Imposta l'orientamento della tartaruga. Questa primitiva accetta un elenco di tre elementi come argomento: angolo di rollio, di beccheggio e di direzione.
Per esempio, `ImpOr [100 0 58]`: la tartaruga ha rollio pari a 100 gradi, beccheggio pari a 0, direzione 58 gradi.

Orientamento
Restituisce l'orientamento della tartaruga come un elenco che contiene `[ rollio beccheggio direzione ]`. Per esempio, se l'orientamento è `[100 20 90]`, significa che se si desidera replicare lo stesso orientamento dalla posizione originale (dopo un `PulisciSchermo`) occorrerà spostare la tartaruga come segue:

```
RDX 100 BS 20 DX 90
```

Se si inverte l'ordine delle istruzioni, l'orientamento finale sarà diverso!

ImpostaRollio, ImpRol *n*
La tartaruga ruota attorno al suo asse longitudinale fino all'angolo fornito in argomento

Rollio
Restituisce il valore corrente di rollio.

ImpostaBeccheggio, ImpBec *n*
La tartaruga ruota attorno al suo asse trasversale fino all'angolo fornito in argomento.

Beccheggio
Restituisce il valore corrente di beccheggio.

A.14.5 Visualizzatore 3D

XLogo include un visualizzatore 3D, permette di visualizzare i disegni in 3D. Questo modulo utilizza la libreria Java3D che, quindi, è un requisito di installazione per un funzionamento appropriato di questa modalità.

Il visualizzatore richiede l'osservanza di alcune regole:
quando si crea una figura geometrica nell'area di disegno occorre indicare al visualizzatore 3D quale forma si vuole disegnare per renderne possibile la futura visualizzazione. E' possibile disegnare poligoni (superfici), linee, punti, o testo. Per usare questa possibilità occorre utilizzare le seguenti primitive:

InizioPoligono
I successivi movimenti della tartaruga sono salvati per creare un poligono.

FinePoligono
A partire dall'ultimo `InizioPoligono`, la tartaruga si è spostata attraverso numerosi vertici. Questo nuovo poligono è salvato, il suo colore è definito dal colore di tutti i vertici. Questa primitiva finalizza il poligono.

InizioLinea, IL
I successivi movimenti della tartaruga sono salvati per creare una linea.

FineLinea, FL
A partire dall'ultimo `InizioLinea`, la tartaruga si è spostata attraverso numerosi vertici. Questa nuova linea è salvata, il suo colore è definito dal colore di tutti i vertici. Questa primitiva finalizza la linea.

InizioPunto, IP
I successivi movimenti della tartaruga sono salvati per creare un set di punti.

FinePunto, FP
Finalizza il set di punti.

InizioTesto, IT
Ogni volta che si scrive del testo nell'area di disegno con la primitiva `Etichetta`, esso verrà salvato e quindi

visualizzata dal visualizzatore 3D.

FineTesto, FT

Fine della registrazione del testo.

VistaPoligono3D, VP3D

Parte il visualizzatore 3D, tutti gli oggetti salvati sono disegnati in questa nuova finestra. Si ha il controllo della scena della cinepresa:

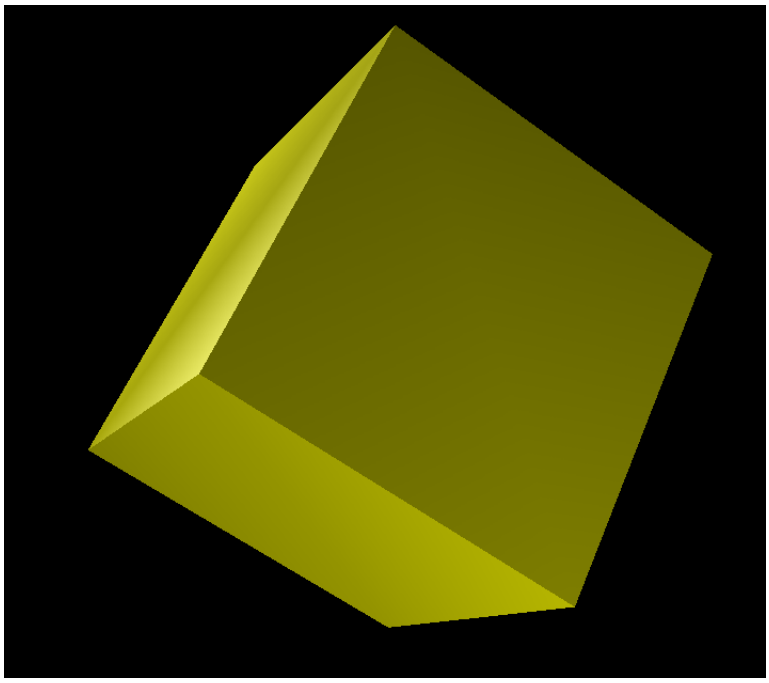
- Si può ruotare la scena cliccando il testo sinistro del mouse.
- Si può traslare la scena cliccando il tasto destra del mouse.
- Si può ingrandire la scena con la rotella del mouse.

A.14.6 Disegnare un cubo

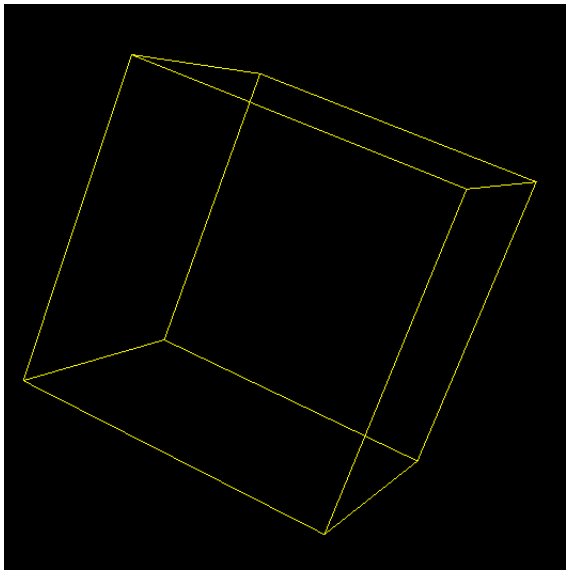
Tutte le facce sono quadrati di 400 passi. Questo è il programma:

```
Per quadrato
  # salviamo il quadrato verticale
  InizioPoligono Ripeti 4[Avanti 400 RuotaDestra 90] FinePoligono
Fine

Per SempliceCubo
  # cubo giallo
  PulisciSchermo 3D ImpostaColorePenna Giallo
  # facce laterali
  Ripeti 4[quadrato PennaSu RuotaDestra 90 Avanti 400 RuotaSinistra 90
  RollioDestra 90 PennaGiu]
  # faccia inferiore
  BeccheggiaGiu 90 quadrato BeccheggiaSu 90
  # faccia superiore
  Avanti 400 BeccheggiaGiu 90 quadrato
  # visualizzazione
  VistaPoligono3D
Fine
```

Il programma si lancia con il comando: `SempliceCubo`:



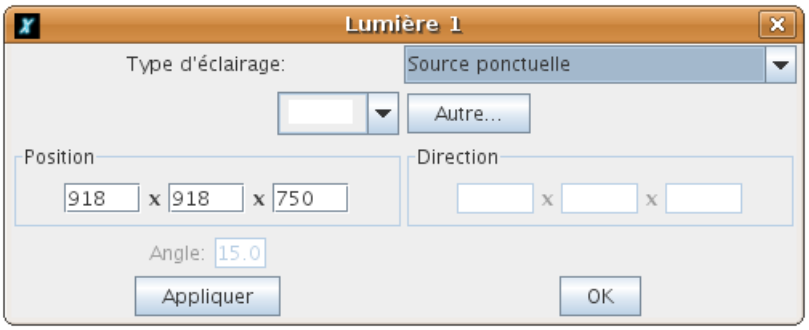
Se sostituiamo nella procedura `quadrato`, `InizioPoligono` con `InizioLinea` e `FinePoligono` con `FineLinea`



Se si fosse utilizzata la primitiva `InizioPunto` e `FinePunto` invece di `InizioLinea` e `FineLinea`, si sarebbero visti su schermo solo gli otto vertici del cubo. Queste primitive sono molto utili per visualizzare il set di punti nello spazio 3D.

A.14.7 Illuminare la scena

Si possono impostare fino a quattro luci nella scena 3D. Come impostazione predefinita, la scena 3D ha solo due luci puntiformi abilitate. Cliccando su uno dei quattro bottoni delle luci nel modellizzatore 3D, la seguente finestra di dialogo comparirà:



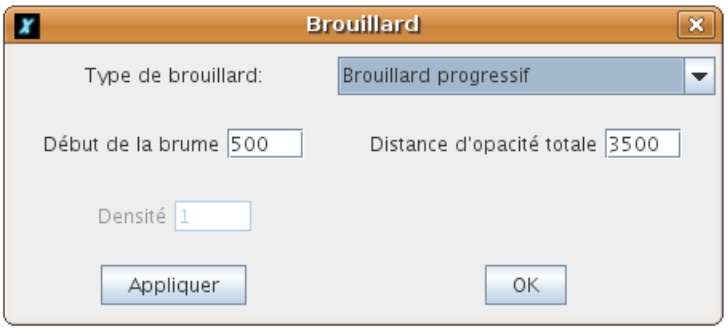
Sono disponibili svariati tipi di illuminazione:

- Illuminazione di ambiente: è una luce uniforme, è possibile specificarne il colore.
- Illuminazione unidirezionale: si diffonde lungo una direzione costante. E' molto simile alla illuminazione puntuale quando la sorgente di luce è molto lontana dall'osservatore, come nel caso del sole, per esempio.
- Illuminazione puntiforme: la luce ha una posizione puntiforme, simile alle luci frontali.
- Illuminazione spot: è una luce puntiforme ma la luce è visualizzata come un cono di luce. Occorre specificare un valore angolare per il cono stesso.

La cosa migliore è giocare con le illuminazioni per capire come funzionano!

A.14.8 Effetto nebbia

Si può aggiungere un effetto di opacità alla scena 3D cliccando sul bottone nebbia nella scena 3D:

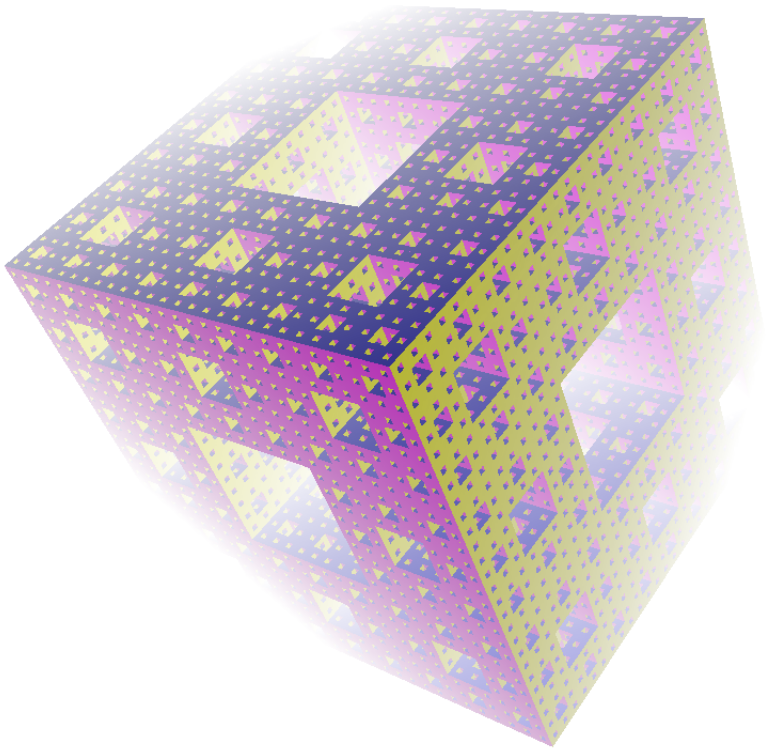


Sono disponibili due tipi di nebbia:

- Nebbia progressiva: l'opacità della nebbia è progressiva, occorre specificare due parametri:
  - La distanza alla quale la nebbia inizia.
  - La distanza alla quale l'opacità è completa.

- Nebbia uniforme: questa nebbia è uniforme in tutta la scena. Occorre specificare la densità della nebbia.

Esempio con nebbia progressiva:



A.15 Riprodurre musica

A.15.1 Riprodurre musica usando il sintetizzatore MIDI

Per riprodurre musica MIDI occorre inserire lo spartito in un elenco di note (sequenza). Queste sono le regole da seguire per creare una sequenza valida:
`do re mi fa sol la si` : le classiche note della prima ottava.
Per riprodurre un re alto, scriviamo `re +`

Per riprodurre un re basso scriviamo `re -`
Se si vuole alzare o abbassare ottava, si usa il simbolo ":" seguito da "+" or "-". Per esempio: dopo ":++" nella sequenza, tutte le note saranno riprodotte due ottave sopra.
Per impostazione predefinita, le note sono riprodotte per la lunghezza di 1. Se si vuole aumentare o diminuire la durata occorre scrivere il numero che corrisponde alla durata delle note. Per esempio: `Seq [sol 0.5 la si]`. riprodurrà "sol" con una durata di 1 e "la si" con durata 0,5 (il doppio della velocità).

Sequenza, Seq *elenco*
Pone in memoria la sequenza nell'elenco aggiungendola alle sequenze precedenti.

Suona
Riproduce la sequenza in memoria.

ImpostaStrumento, ImpStr *n*
Imposta lo strumento identificato da *n*. L'elenco di tutti gli strumenti disponibili si trova nel nel Menu Strumenti→Preferenze*to*Tab Suono.

Strumento
Restituisce il numero che corrisponde allo strumento in uso.

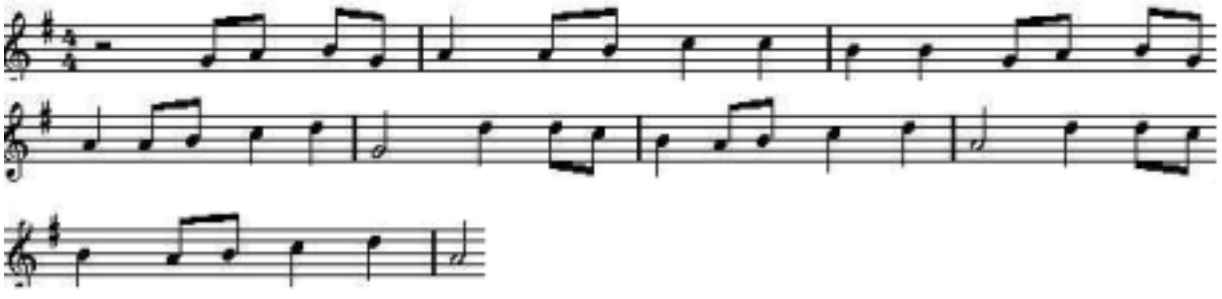
ImpostaIndiceInSequenza, ImpSeq *n*
Pone il cursore all'indice *n* nella sequenza attuale in memoria.

IndiceSequenza, IndSeq
Restituisce la posizione del cursore nella sequenza attiva.

CancellaSequenza, CancSeq
Elimina la sequenza attiva in memoria.

If you want to play music, you must put the notes in memory in a list called sequence. To create the sequence, you can use the primitive `seq` or `sequence`.

Esempio di riproduzione MIDI:



```
Per tabac
# Crea la sequnza di note
seq [0.5 sol la si sol 1 la 0.5 la si 1 :+ do do :- si si 0.5 sol la si sol
        1 la 0.5 la si 1 :+ do re 2 :- sol ]
seq [:+ 1 re 0.5 re do 1 :- si 0.5 la si 1 :+ do re 2 :- la ]
seq [:+ 1 re 0.5 re do 1 :- si 0.5 la si 1 :+ do re 2 :- la ]
seq [0.5 sol la si sol 1 la 0.5 la si 1 :+ do do :- si si 0.5 sol la si sol
        1 la 0.5 la si 1 :+ do re 2 :- sol ]
Fine
```

Per ascoltare la musica, eseguire il comando: `tabac Suona`
Un altro esempio mostra una interessante applicazione della primitiva `ImpSeq`:

```
CancSeq
tabac
ImpSeq 2
tabac
Suona
```

A.15.2 Riprodurre file MP3

Avviamp3 *parola1*
Legge e riproduce il file mp3 *parola1*. Il file deve risiedere nella cartella attuale oppure può essere localizzato in rete. Per esempio:
`Avviamp3 file.mp3`
`Avviamp3 http://website.com/file.mp3`

Fermamp3
Ferma la riproduzione del file mp3 attuale.

A.16 Interagire con l'utente durante l'esecuzione del programma

A.16.1 Interazione tramite la tastiera

Attualmente il testo può essere accettato dall'utente durante l'esecuzione del programma attraverso tre primitive:

1. `Tasto?`
2. `LeggiCarattere`
3. `Leggi`

Tasto?
Restituisce "vero" o "falso" a seconda che l'utente abbia premuto un tasto dall'avvio del programma LOGO.

LeggiCarattere

- Se `Tasto?` è falso il programma viene messo in pausa fino a che l'utente preme un tasto.
- Se `Tasto?` è vero restituisce il codice unicode dell'ultimo tasto che è stato premuto.

E' opportuno ricordare che `LeggiCarattere` non ritorna il carattere ma il suo codice unicode, per convertirlo in carattere utilizzare la primitiva `Carattere` (vedi pagina 101). Questi sono il valori per alcuni tasti speciali:
Se si è insicuri dal valore restituito da un tasto, si può digitare:
`Stampa LeggiCarattere`. L'interprete aspetterà la pressione di un tasto prima di fornire il valore corrispondente.

Leggi *elenco1 arg2*
Visualizza una finestra di dialogo il cui titolo è *elenco1*. L'utente può quindi inserire una risposta in un campo di testo. La risposta sarà salvata nella forma di parola o elenco (se l'utente ha scritto più parole) nella variabile *arg2*, che sarà assegnata quando il bottone OK viene premuto.

Carattere	Codice unicode	Carattere	Codice unicode
←	-37 or -226 (NumPad)	↑	-38 or -224 (NumPad)
→	-39 or -227 (NumPad)	↓	-40 or -225 (NumPad)
ESC	27	F1	-112
F2	-113		
Shift	-16	Spazio	32
Ctrl	-17	Invio	10

Tabella A.2: Caratteri e rispettivi codici unicode

A.16.2 Qualche esempio di utilizzo

Esempio 1:

```
per vintage
  Leggi [Che eta' hai?] "age
  AssegnaVar "age :age
  Se :age<18 [Stampa [Sei minorenne]]
  Se o :age=18 :age>18 [Stampa [Sei un adulto]]
  Se :age>99 [Stampa [Rispetto e' dovuto!!]]
fine
```

Esempio 2, cede il controllo della tartaruga all'utente:

```
per rallye
  Se Tasto? [
    AssegnaVar "car LeggiCarattere
    Se :car=-37 [RuotaSinistra 90]
    Se :car=-39 [RuotaDestra 90]
    Se :car=-38 [Avanti 10]
    Se :car=-40 [Indietro 10]
    Se :car=27 [FermaTutto]
  ]
fine
```

A.16.3 Interazione tramite il mouse

Attualmente gli eventi del mouse possono essere accettati dall'utente durante l'esecuzione del programma attraverso tre primitive:

1. `LeggiMouse`
2. `PosizioneMouse`
3. `Mouse?`

LeggiMouse

L'esecuzione del programma viene messa in pausa in attesa di un evento del mouse. Restituisce un numero che rappresenta l'evento avvenuto. I valori sono:

- 0→Il mouse è stato mosso.
- 1→Il bottone 1 è stato premuto.
- 2→Il bottone 2 è stato premuto.

Il bottone 1 è quello di sinistra, il bottone 2 è quello alla sua destra e così via ...

PosizioneMouse, PosMouse
Restituisce un elenco contenente la posizione del mouse.

Mouse?
Restituisce "vero" se si tocca il mouse dall'inizio del programma LOGO, "falso" altrimenti.

A.16.4 Qualche esempio di utilizzo

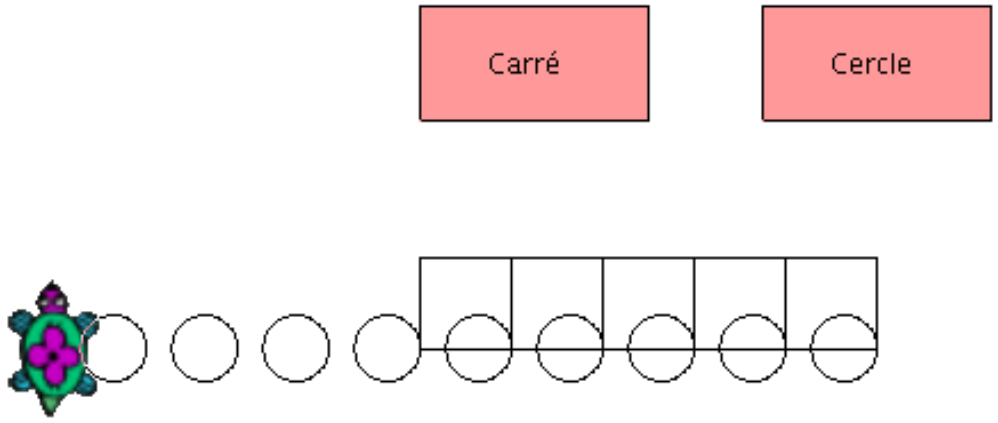
Esempio 1, la tartaruga segue il mouse quando si sposta sullo schermo:

```
Per esempio
  Se LeggiMouse=0 [ImpPos PosMouse]
Fine
RipetiPerSempre [esempio]
```

Esempio 2, come il precedente ma occorre premere il mouse per spostare la tartaruga:

```
Per esempio2
  if LeggiMouse=1 [ImpPos PosMouse]
Fine
RipetiPerSempre [esempio2]
```

Esempio 3, si creano due bottoni rosa. Se si clicca sul bottone di sinistra viene disegnato un quadrato di lato 40, se si clicca sul bottone di destra viene disegnato un piccolo cerchio. Se si clicca il tasto destro sul bottone destro il programma viene fermato. Il programma si avvia con `avvio`:




```
per bottone
  #crea un bottone rettangolare rosa
  Ripeti 2[Avanti 50 RuotaDestra 90 Avanti 100 RuotaDestra 90]
  RuotaDestra 45 PennaSu Avanti 10 PennaGiu ImpostaColorePenna [255 153 153]
  Riempi Indietro 10 RuotaSinistra 45 PennaGiu ImpostaColorePenna 0
fine

per avvio
  PulisciSchermo bottone PennaSu ImpostaPosizione [150 0] PennaGiu bottone
  PennaSu ImpostaPosizione [30 20] PennaGiu Etichetta "quadrato
  PennaSu ImpostaPosizione [180 20] PennaGiu Etichetta "cerchio
  PennaSu ImpostaPosizione [0 -100] PennaGiu
  mouse
fine

per mouse
  # assegniamo il valore di LeggiMouse nella variabile ev
  AssegnaVar "ev LeggiMouse
```

```
  # assegniamo la coordinata x del mouse nella variabile x
  AssegnaVar "x Elemento 1 PosizioneMouse
  # assegniamo la coordinata y del mouse nella variabile y
  AssegnaVar "y Elemento 2 PosizioneMouse
  # se si clicca sul bottone di sinistra
  Se :ev=1 & :x>0 & :x<100 & :y>0 & :y<50 [square]
  # se si clicca sul bottone di destra
  Se  :x>150 & :x<250 & :y>0 & :y<50 [
            Se :ev=1 [Circonferenza2]
            Se :ev=3 [Ferma]
  ]
  mouse
fine

per Circonferenza2
  Circonferenza 20 PennaSu  Avanti 40 RuotaDestra 90 PennaGiu
fine

per square
  Ripeti 4 [Avanti 40 RuotaDestra 90] RuotaDestra 90 Avanti 40 RuotaSinistra 90
fine
```

A.16.5 Componenti grafici

Con XLogo si possono aggiungere molti componenti grafici nell'area di disegno (bottoni e menu). Tutte le primitive che permettono la manipolazione di queste componenti terminano con il suffisso GUI (per "Graphical User Interface" Interfaccia grafica con l'utente).
L'aggiunta di un componente segue tre passi:

1. Creazione
2. Modifica delle proprietà
3. Visualizzazione sull'area di disegno

Creare un componente

BottoneGUI, BGUI *parola1 parola2*
Crea un bottone il cui titolo è *parola2* ed il cui nome è *parola1*. Il nome identifica il componente per i successivi passi.

MenuGUI, MGUI *parola1 elenco2*
Crea un menu combinato il cui nome è *parola1* e che contiene gli elementi da *elenco2*
`MenuGUI mioMenu [elemento1 elemento2 elemento3]`

Modificare le proprietà del componente

PosizioneGUI, PGUI *parola1 elenco2*
Posiziona l'elemento grafico *parola1* in un punto preciso fornito dalle coordinate in `elenco2`. Per esempio, volendo posizionare il bottone "b" alle coordinate $(20; 100)$, si scriverà:

```
PosizioneGUI "b [20 100]
```

Se non si specifica una posizione per il componente, verrà posizionato nell'angolo in alto a sinistra dell'area di disegno.

RimuoviGUI, RGUI *parola1*
Cancella un componente grafico. Per esempio per cancellare il bottone "b":

```
RimuoviGUI "b
```

AzioneGUI, AGUI *parola1 elenco2*
Definisci una azione per il componente `parola1` quando l'utente interagisce con esso.

```
# la tartaruga avanza di 100 passi se si clicca il bottone "b
AzioneGUI "b [fd 100 ]
# Nel menu combinato, ciascun elemento ha la sua azione
AzioneGUI "m [[Stampa "elemento1] [Stampa "elemento2] [Stampa "elemento3]]
```

Modificare le proprietà del componente

DisegnaGUI, DGUI *parola1*
Visualizza il componente grafico nell'area di disegno. Per esempio per mostrare il bottone "b":

```
DisegnaGUI "b
```

A.17 Ora e data

XLogo ha numerose primitive per la data, l'ora e per generare conteggi inversi.

Aspetta *n*
Sospende l'esecuzione del programma e quindi la tartaruga per $\frac{n}{60}$ secondi. Per esempio per sospendere il programma per 1 secondo: `Aspetta 60`.

Cronometro *n*
Inizia un conteggio inverso di n secondi. Il programma può controllare se il conteggio è terminato con `FineCronometro?`

La differenza tra `Aspetta` e `Cronometro` rimane nel fatto che la seconda primitiva non sospende il programma.

Esempio:

```
per orologio
  Se FineCronometro? [
  PulisciSchermo
  ImpCFont NascondiTartaruga
  AssegnaVar "heu HMS
  AssegnaVar "h Primo :heu
  AssegnaVar "m Elemento 2 :heu
  Se :m-10<0 [AssegnaVar "m Parola 0 :m]
  AssegnaVar "s Ultimo :heu
  Se :s-10<0 [AssegnaVar "s Parola 0 :s]
  Etichetta Parola Parola Parola Parola :h ": :m ": :s
  Cronometro 5
  ]
fine
```

FineCronometro?
Restituisce `vero` se non c'è un conteggio attivo, altrimenti restituisce `falso`.

Data
Restituisce un elenco contenente tre numeri interi rappresentanti la data di sistema. Il primo intero indica il giorno, il secondo il mese, il terzo l'anno ([giorno mese anno]).

HMS
Restituisce un elenco di tre interi rappresentanti l'ora di sistema. Il primo intero identifica l'ora, il secondo i minuti, il terzo i secondi [ora minuto secondo]

SecondiDaAvvio
Restituisce il tempo passato in secondi da quando XLOGOè stato lanciato.

A.18 Utilizzare XLogo in rete

A.18.1 Basi delle reti

Introduciamo le basi della comunicazione in rete prima di utilizzare le primitive di XLogo.

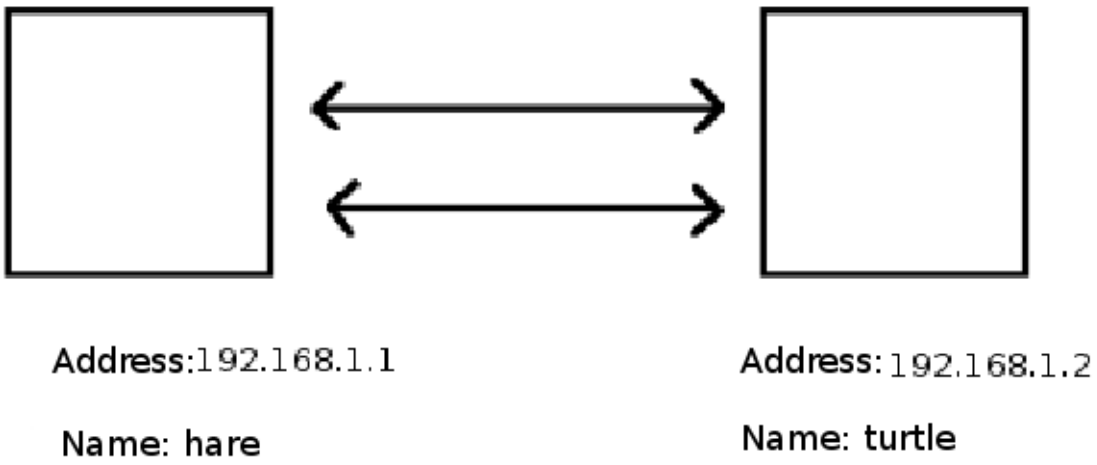


Figura A.5: Una semplice rete

Due o più computer possono comunicare in rete se entrambi possiedono schede di rete Ethernet. Ciascun computer è identificato da un indirizzo univoco chiamato *indirizzo IP*. Questo indirizzo IP consiste di 4 numeri interi, ciascuno compreso tra 0 e 255 e separati da punti. Per esempio l'indirizzo IP del primo computer in figura è 192.168.1.1.
Poiché non è facile ricordare questi indirizzi è anche possibile identificare ciascun computer con un nome più semplice. Come possiamo vedere in figura, possiamo comunicare con il computer di destra attraverso il suo indirizzo IP 192.168.1.2 o tramite il suo nome: `turtle`
Il computer locale sul quale si lavora ha sempre uno stesso indirizzo IP: 127.0.0.1, il suo nome generale è textttlocalhost. Lo vedremo in pratica più oltre.

A.18.2 Primitive per la rete

XLogo ha quattro primitive che gli consentono di comunicare in rete:

1. `AscoltaTCP`
2. `EseguiTCP`
3. `FinestraChatTCP`
4. `InviaTCP`

Nei successivi esempi si assume la configurazione dei due computer nella figura precedente.

AscoltaTCP, ATCP

Questa primitiva è la base di tutta la comunicazione in rete. Non ha bisogno di un argomento. Il computer si dispone in ascolto delle istruzioni spedite da altri computer nella rete.

EseguiTCP, ETCP *parola1 elenco2*

Questa primitiva permette l'esecuzione di istruzioni da un computer in rete. L'indirizzo IP del computer in rete è fornito in *parola1*, l'elenco *elenco2* contiene le istruzioni da eseguire.
Per esempio: mi trovo sul computer `hare`, voglio disegnare un quadrato di lato 100 sull'altro computer. Quindi sul computer `turtle`, devo lanciare la primitiva `listentcp`. Quindi sul computer `hare`, scrivo:

```
EseguiTCP "192.168.1.2 [ripeti 4[avanti 100 ruotadestra 90]]
# o
EseguiTCP "turtle [ripeti 4[avanti 100 ruotadestra 90]]
```

FinestraChatTCP, FChatTCP *parola1 elenco2*

Permette una chat tra due computer in rete. Su ciascun computer viene visualizzata una finestra di chat. *parola1* è l'indirizzo IP del computer o il suo nome, *elenco2* contiene la frase da visualizzare.
Per esempio: `hare` vuole parlare con `turtle`.
Per prima cosa `turtle` esegue `AscoltaTCP`, in questo modo rimane in attesa di istruzioni da computer in rete. Quindi `hare` scrive: `FinestraChatTCP "192.168.1.2 [ciao turtle]`.
Le finestre chat compariranno su entrambi i computer, permettendogli di parlare fra di loro.

InviaTCP, ITCP *parola1 elenco2*

Spedisce dati verso un computer in rete e restituisce la sua risposta. *parola1* è l'indirizzo IP del computer o il suo nome, *elenco2* contiene i dati da spedire. Quando XLogo è lanciato sull'altro computer, risponderà OK. E' possibile comunicare con un robot attraverso la sua interfaccia di rete. La risposta del robot potrebbe quindi variare.
Per esempio: `turtle` vuole spedire a `hare` la frase "3.14159 è una approssimazione di pi greco".
`hare` esegue `AscoltaTCP`. `turtle` scrive: `Stampa InviaTCP "hare [3.14159 è una approssimazione di pi greco]`.

Un piccolo suggerimenti: Apri due volte XLogo sullo stesso computer.

- Nella prima finestra, esegui `AscoltaTCP`.
- Nella seconda finestra, scrivi `EseguiTCP "127.0.0.1 [avanti 100 ruotadestra 90]`

Puoi muovere la tartaruga nell'altra finestra! Ciò è possibile perché 127.0.0.1 designa l'indirizzo locale, è quindi lo stesso computer ...

Appendice B

Avviare XLogo tramite la linea di comando

Per eseguire XLogo, da linea di comando ecco la sintassi completa:

```
java -jar xlogo.jar [-a] [-lang en] [-memory 64] [file1.lgo file2.lgo ...]
```

Elenco delle opzioni disponibili:

- Attributo `-lang`: specifica un linguaggio. Questo parametro sovrascrive quello dal file di configurazione chiamato `.xlogo`. La seguente tabella elenca tutti i possibili linguaggi:

French	English	Spanish	German	Arabic	Portuguese	Espéranto	Galician	Greek	Italian
fr	en	es	de	ar	pt	eo	gla	el	it

- Attribute `-a`: indica l'esecuzione di un comando principale, contenuto in un file caricato all'avvio, dopo che la finestra XLogo si è aperta.
- Attribute `-memory`: imposta la quantità di memoria che XLogo alloca per i suoi programmi.
- file1.lgo, file2.lgo ...: questi file in formato `.lgo` sono caricati all'avvio di XLogo. I file possono essere locali o remoti, si può quindi specificare un indirizzo web.
- Attribute `-tcp_port`: permette di modificare la porta TCP preimpostata (1984) usata per le comunicazioni in rete (cfr. pagina 129).

Qualche esempio:

- `java -jar xlogo.jar -lang es prog.lgo`:
  I file `xlogo.jar` e `prog.lgo` sono nella cartella corrente. Il comando esegue XLogo, in spagnolo. Quindi carica il file `prog.lgo` che, quindi, deve essere scritto in spagnolo.
- `java -jar xlogo.jar -a -lang en http://xlogo.tuxfamily.org/prog.lgo`:
  Questo comando esegue XLogo in inglese. Carica il file `http://xlogo.tuxfamily.org/prog.lgo`. Infine il comando principale di questo file viene eseguito all'avvio.

Appendice C

Eseguire XLogo sul web

C.1 Il problema

Gestisci un sito web. Su questo sito, parli di XLogo e vuoi fornire alcuni dei programmi che hai creato con XLogo. È possibile distribuire i file logo in formato (.LGO), ma sarebbe meglio se l'utente potesse lanciare XLogo on line e direttamente verificare il tuo programma.

Per avviare XLogo da un sito web, useremo la tecnologia (Java Web Start). In realtà, abbiamo solo bisogno di mettere sul nostro sito un link verso un file con estensione (.Jnlp). Si avvierà XLogo on-line.

C.2 Creare il file `.jnlp`

Questo è un esempio di questo tipo di file. Nei fatti, il successivo esempio è quello usato nel sito in francese nella sezione "exemples". Questo file permette di caricare il programma che carica un dado nella sezione 3D. La spiegazione del contenuto del file verrà data dopo il codice.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <jnlp spec="1.5+" codebase="http://downloads.tuxfamily.org/xlogo/common/webstart">
3  <information>
4    <title>XLogo</title>
5    <vendor>xlogo.tuxfamily.org</vendor>
6    <homepage href="http://xlogo.tuxfamily.org"/>
7    <description>Logo Programming Language</description>
8    <offline-allowed/>
9  </information>
10
11 <security>
12   <all-permissions/>
13 </security>
14
15 <resources>
16   <j2se version="1.4+"/>
17   <jar href="xlogo.jar"/>
18 </resources>
19
20 <application-desc main-class="Lanceur">
21   <argument>-lang</argument>
22   <argument>fr</argument>
23   <argument>-a</argument>
24   <argument>http://xlogo.tuxfamily.org/fr/html/examples-fr/3d/de.lgo</argument>
25 </application-desc>
26 </jnlp>
```

Questo file è scritto in formato XML. Le parti più importanti sono queste quattro linee:

```
21   <argument>-lang</argument>
```

```
22  <argument>fr</argument>
23  <argument>-a</argument>
24  <argument>http://xlogo.tuxfamily.org/fr/html/examples-fr/3d/de.lgo</argument>
```

Queste righe specificano i parametri per XLogo all'avvio.

- Le righe 21 e 22 forzano la lingua francese.
- La riga 24 indica l'indirizzo del file da caricare.
- La riga 23 indica che il comando principale verrà eseguito automaticamente all'avvio di XLogo.

Un ultimo suggerimento: Poiché il server di Tuxfamily non può accettare tutte le connessioni è meglio mettere il file `xlogo.jar` sul tuo sito. Per collegare questo file con il file `.jnlp`, devi solo modificare l'indirizzo nella riga 2 dopo `codebase=`.

Appendice D

Soluzioni

D.1 Capitolo 5

```
per quadrato
  Ripeti 4[Avanti 150  DX 90]
fine

per triangolo
  Ripeti 3[Avanti 150 DX 120]
 fine

per porta
  Ripeti 2[Avanti 70 DX 90 Avanti 50 DX 90]
fine

per camino
  Avanti 55 DX 90 Avanti 20 DX 90 Avanti 20
fine

per spostaAB
  DX 90 Avanti 50  SX 90
fine

per spostaBC
  SX 90 Avanti 50 DX 90 Avanti 150 DX 30
fine

per spostaCD
  PS DX 60 Avanti 20 SX 90 Avanti 35 PG
fine

per casa
  quadrato spostaAB porta spostaBC triangolo spostaCD camino
fine
```

D.2 Capitolo 6

```
per supercubo
  PulisciSchermo PS ImpPos[ -30 150] PG ImpPos[-150 150]  ImpPos[-90 210]
  ImpPos[30 210] ImpPos[-30 150]
  ImpPos[-30 -210] ImpPos[30 -150] ImpPos[30 -90] ImpPos[-30 -90] ImpPos[90 -90]
  ImpPos[90 30]
  ImpPos[-270 30] ImpPos[-270 -90] ImpPos[-210 -90] ImpPos[-210 -30]
  ImpPos[-90 -30] ImpPos[-90 -150]
```

```
  ImpPos[-210 -150] ImpPos[-210 -30] ImpPos[-150 30] ImpPos[-30 30]
  ImpPos[-90 -30] ImpPos[90 150]
  ImpPos[30 150] ImpPos[30 210] ImpPos[30 90] ImpPos[90 90] ImpPos[90 150]
  ImpPos[90 90] ImpPos[150 90]
  ImpPos[150 -30] ImpPos[90 -90] ImpPos[90 30] ImpPos[150 90] PS
  ImpPos[-150 30] PG ImpPos[-150 150]
  ImpPos[-150 90] ImpPos[-210 90] ImpPos[-270 30] PS  ImpPos[-90 -150] PG
  ImpPos[-30 -90]
  PS ImpPos[-150 -150] PG ImpPos[-150 -210] ImpPos[-30 -210]
fine
```

D.3 Capitolo 7

D.3.1 Il robot

Il primo disegno è formato da sole figure piane, rettangoli, quadrati e triangoli. Ecco il codice:

```
per rec :lo :la
  # draws a rectangular with choosen dimansion
  Ripeti 2[Avanti :lo DX 90 Avanti :la DX 90]
fine

per square :c
  # draws a square with side length :c
  Ripeti 4[Avanti :c DX 90]
fine

per tri :c
  # draws an equilateral triangle with side length :c
  Ripeti 3[Avanti :c DX 120]
fine

per leg :c
  rec 20*:c 30*:c square 20*:c
fine

per antenna :c
  Avanti 30*:c SX 90 Avanti 10*:c DX 90 square 20*:c
  PS Indietro 30 *:c DX 90 Avanti 10*:c SX 90 PG
fine

per robot :c
  PulisciSchermo NascondiTartaruga
  # body
  rec 40*:c 280* :c
  # legs
  DX 90 Avanti 20*:c leg :c Avanti 40* :c leg :c Avanti 140*:c leg :c
  Avanti 40*:c leg :c
  # queue
  PS SX 90 Avanti 40* :c PG DX 45 Avanti 110*:c Indietro 110* :c SX 135
  # head
  Avanti 180 *:c square 10*:c Avanti 30*:c square 10*:c DX 90
  Avanti 10*:c SX 90
  Avanti 20*:c DX 90 square 80* :c
  # ears
  Avanti 40*:c SX 60 tri 30*:c PS DX 150 Avanti 80 *:c SX 90 PG tri 30*:c
  #  antennas
  Avanti 40 *:c SX 90 Avanti 20*:c DX 90 antenna :c SX 90 Avanti 40*:c
  DX 90 antenna :c
  # eyes
```

```
  PS Indietro 30 *:c PG square 10*:c DX 90 PS Avanti 30*:c PG SX 90
  square 10*:c
  # mouth
  PS Indietro 30*:c SX 90 Avanti 30*:c DX 90 PG rec 10*:c 40*:c
fine
```

D.3.2 La rana

```
per frog :c
  PulisciSchermo NascondiTartaruga
  Avanti 20 *:c DX 90 Avanti 50*:c SX 90 Avanti 40*:c
  SX 90 Avanti 70 *:c DX 90 Avanti 70*:c DX 90
  Avanti 210 *:c DX 90 Avanti 20*:c SX 90 Avanti 20*:c
  DX 90 Avanti 90*:c DX 90 Avanti 20*:c SX 90
  Avanti 20*:c DX 90 Avanti 90*:c DX 90 Avanti 20*:c
  DX 90 Avanti 70*:c Indietro 50*:c SX 90 Avanti 40*:c
  DX 90 Avanti 40*:c Indietro 40*:c SX 90 Indietro 20*:c SX 90
  Avanti 50*:c SX 90 Avanti 40*:c DX 90 Avanti 70*:c
  DX 90 PS Avanti 90*:c PG Ripeti 4[Avanti 20*:c DX 90]
fine
```

D.4 Capitolo 9

```
per game
  # initailaize number e counter
  AssegnaVar "number Casuale 32
  AssegnaVar "counter 0
  loop
fine

per loop
  Leggi [choose a number] "try
  Se Numero? :try[
  # Se the value is a number
  Se :number=:try[Stampa Frase Frase [you win in ] :counter+1 [tries]][
  Se :try>:number [Stampa [lesser]][Stampa [greater]]
  AssegnaVar "counter :counter+1
  loop
  ]
  ]
  [Stampa [enter a valid number!] loop]
fine
```

Appendice E

FAQ e trucchi

E.1 Sebbene cancello una procedura dall'editor continua a ritornare

Al momento di chiuderlo, l'editor salva o aggiorna qualsiasi cosa l'editor contiene. Per cancellare una procedura in XLogo devi usare la procedura `CancellaProcedura, CancProc`. Per esempio `CancProc "toto` cancella la procedura `toto`.

E.2 Sto usando la versione in Esperanto ma non riesco a scrivere i caratteri speciali

Quando scrivi nella linea di comando o nell'editor cliccando il tasto destro del mouse una finestra appare. In questo menu puoi trovare le tradizionali funzioni di editing (copia, incolla, taglia) e i caratteri speciali dell'esperanto se hai selezionato questo linguaggio.

E.3 Nel tab Suono della finestra di dialogo delle Preferenze non trovo alcuno strumento

Effettivamente qualche volta l'elenco degli strumenti non appare. Consulta questa pagina:

`http://java.sun.com/products/java-media/sound/soundbanks.html`

e seguine le istruzioni.

E.4 Come riscrivere velocemente un comando usato in precedenza?

- Primo metodo: con il mouse, clicca sulla linea nell'area dello storico dei comandi, riapparirà immediatamente sulla linea di comando.
- Secondo metodo: con la tastiera le frecce Sù e Giù permettono la navigazione dello storico dei comandi.

E.5 Come posso aiutare?

- Riportando un errore in XLogo all'autore. Meglio se sei in grado di riprodurlo sistematicamente.
- Qualsiasi suggerimento per migliorare il programma è benvenuto.
- Aiutando nel tradurre le primitive, il manuale, il sito web.
- Un piccolo supporto morale è sempre benvenuto.

Ringraziamenti

- Vorrei ringraziare tutti i traduttori attivi per il loro lavoro in XLogo.
  - Inglese: Guy Walker
  - Spagnolo: Marcelo Duschkin, Alvaro Valdes Menendez
  - Arabico: El Houcine Jarad
  - Portoghese: Alexandre Soares
  - Tedesco: Michael Malien
  - Esperanto: Michel Gaillard, Carlos Enrique Carleos Artime
  - Galiziano: Justo freire
  - Greco: Anastasios Drakopoulos
  - Italiano: Marco Bascietto
  - Catalano: David Arso
  - Ungherese: József Varga

- Vorrei ringraziare Eitan Gurari per la sua pazienza e per la sua grandiosa estensione LaTeX chiamata `tex4ht` che permette di convertire i miei manuali in diversi formati

  `www.cse.ohio-state.edu/~gurari/TeX4ht`

- Molti progetti open source importanti per XLogo:
  - Java3D: `https://java3d.dev.java.net/`
  - JavaHelp: `http://java.sun.com/javase/technologies/desktop/javahelp/`
  - Eclipse: `http://www.eclipse.org/`
  - JLayer: `http://www.javazoom.net/javalayer/javalayer.html`

- Infine un grande GRAZIE a Tuxfamily per la qualità del loro hosting e per il loro importante contributo al software libero!

  `http://www.tuxfamily.org`

Indice analitico