

Rileggendo “A Beginner’s Guide to Logo”

Claudio Carboncini

Agosto 2020

Nella rivista Byte dell’agosto 1982⁽¹⁾ Harold Abelson scriveva l’articolo “A Beginner’s Guide to Logo” che mostrava le potenzialità del linguaggio Logo come ambiente di apprendimento. Rileggendo l’articolo ho trovato stimolanti e significativi gli esempi che sviluppa Abelson e il presente lavoro cerca di riprodurli utilizzando alcune delle risorse libere che si hanno a disposizione a quasi 40 anni da quell’esperienza.

Il linguaggio utilizzato nel documento per l’etichetta Abelson è Ucblogo perché considerato il più vicino all’implementazione usata da Abelson nel proporre i programmi. Ucbologo⁽²⁾ è un programma che ha difficoltà a girare sulle più recenti macchine in Linux e può essere sostituito dai derivati Mswlogo o Fmswlogo, nativi per Windows che però con l’emulatore Wine si utilizzano con soddisfazione anche su Linux.

Librelogo è quello di più facile utilizzo in quanto multiplatforma e collegato alla suite d’ufficio Libreoffice. Esiste una ricca documentazione con varie esperienze didattiche curata da Andreas Formiconi⁽³⁾.

Snap!⁽⁴⁾ (ex BYOB) è un linguaggio di programmazione visuale, drag and drop, reimplementazione estesa di Scratch che permette di costruire propri blocchi. Snap! è sviluppato tra l’altro da Brian Harvey l’autore di Ucblogo.

⁽¹⁾ L’articolo originale è reperibile in [Archive.org](https://archive.org/stream/byte-magazine-1982-08-rescan/1982_08_BYTE_07-08_Logo#page/n89/) (https://archive.org/stream/byte-magazine-1982-08-rescan/1982_08_BYTE_07-08_Logo#page/n89/)

⁽²⁾ - [UcbLogo](https://people.eecs.berkeley.edu/~bh/logo.html) (<https://people.eecs.berkeley.edu/~bh/logo.html>)
- [MswLogo](http://www.softronix.com/logo.html) (<http://www.softronix.com/logo.html>)
- [FmsLogo](http://fmslogo.sourceforge.net/) (<http://fmslogo.sourceforge.net/>)

⁽³⁾ [Logo nel blog di Andreas Formiconi](https://iamarf.org/?s=librelogo) (<https://iamarf.org/?s=librelogo>)

⁽⁴⁾ [Il sito di Snap!](https://snap.berkeley.edu/) (<https://snap.berkeley.edu/>)

Disegnare con la tartaruga

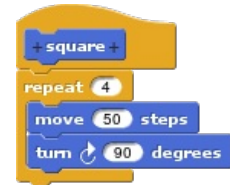
Abelson

```
TO SQUARE  
REPEAT 4 [FORWARD 50 RIGHT 90]  
END
```

LibreLogo⁽⁵⁾

```
TO SQUARE1  
REPEAT 4 [ FORWARD 50 RIGHT 90 ]  
END
```

Snap!



SQUARE è un esempio di *procedura* Logo. La prima riga (identificata da TO) specifica il nome della procedura. Questa procedura si chiama SQUARE (dal momento che è quello che disegna), ma si sarebbe potuto chiamare in qualunque modo. Il resto della procedura (corpo della procedura) specifica l'elenco delle istruzioni da eseguire in risposta al comando SQUARE; il termine END indica la fine della definizione.

Una volta definito in questo modo, SQUARE diventa parte del vocabolario del computer. Ogni volta che si dà il comando SQUARE, la tartaruga disegna un quadrato.

⁽⁵⁾ Librelogo ha una primitiva SQUARE che non va usata come procedura definita dall'utente.

In Librelogo le parentesi quadre che racchiudono le istruzioni da ripetere devono essere seguite per "[" e precedute per "]" da uno spazio.

Procedure con input

Esiste un'importante differenza tra SQUARE e FORWARD. SQUARE disegna sempre un quadrato di 50 passi per lato. Ma FORWARD è più versatile: richiede un ingresso che determina la distanza che la tartaruga deve percorrere. È possibile modificare la procedura di SQUARE in modo che prenda anche un input che determina la dimensione del quadrato da disegnare. Ad esempio:

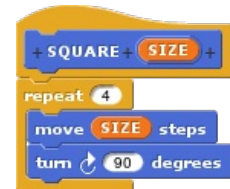
Abelson

```
TO SQUARE :SIZE  
REPEAT 4 [FORWARD :SIZE RIGHT 90]  
END
```

LibreLogo

```
TO SQUARE1 :SIZE  
REPEAT 4 [ FORWARD :SIZE RIGHT 90 ]  
END
```

Snap!



Si usa SQUARE come qualsiasi comando Logo che richiede un input: per disegnare un quadrato con 100 passi di lato, si digita: SQUARE 100; Per disegnare un quadrato con 50 di lato, si digita: SQUARE 50.

La procedura di SQUARE illustra la regola generale per la definizione delle procedure che prevedono input. Si sceglie un nome per l'input e lo si include nella riga del titolo della procedura preceduto da due punti. Quindi si utilizza il nome di input (con i due punti) nel corpo della procedura dove è

richiesto. Dato che una procedura, una volta definita, diventa un'altra parola che il computer "conosce", è possibile utilizzare le procedure dichiarate come parte delle definizioni di altre procedure. Ecco una procedura che realizza un disegno andando avanti, ruotando e disegnando ripetutamente un quadrato.

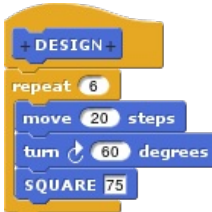

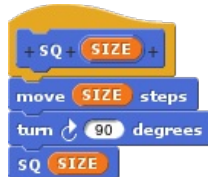
Abelson	LibreLogo	Snap!
<pre> TO DESIGN REPEAT 6 [FORWARD 20 RIGHT 60 SQUARE 75] END </pre>	<pre> TO DESIGN REPEAT 6 [FORWARD 20 RIGHT 60 SQUARE1 75] END </pre>	
		

Figura 1 : Un disegno creato da un semplice programma Logo ad una linea che fa ripetere alla tartaruga questi passaggi sei volte: andare avanti di 20 unità, ruotare a destra di 60 gradi e disegnare un quadrato di 75 unità.

Procedure ricorsive semplici

Anche la procedura successiva disegna un quadrato con una dimensione da specificare:

Abelson	LibreLogo	Snap!
<pre> TO SQ :SIZE FORWARD :SIZE RIGHT 90 SQ :SIZE END </pre>	<pre> TO SQ :SIZE FORWARD :SIZE RIGHT 90 SQ :SIZE END </pre>	

Anche se SQ e SQUARE disegnano entrambi quadrati si comportano in modo molto diverso. Invece di disegnare un quadrato e poi fermarsi, SQ fa sì che la tartaruga ripercorra lo stesso percorso più e più volte, o fino a quando non si dice al computer di fermarsi. Ecco perché questo accade. Quando si dà il comando: SQ 100 la tartaruga deve eseguire FORWARD 100, RIGHT 90, per poi fare di nuovo SQ 100 e così via, per sempre. Aggiungendo un secondo ingresso a SQ si ottiene una procedura chiamata POLY, che ripete più volte la sequenza: andare AVANTI di qualche distanza fissa e girare a DESTRA di qualche angolo fisso. La procedura prende come input la dimensione di ogni passo dal parametro SIZE e la quantità di ogni rotazione dal parametro ANGLE:

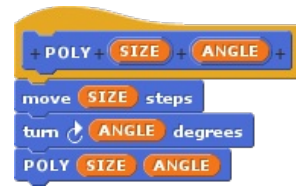
Abelson

```
TO POLY :SIZE :ANGLE  
FORWARD :SIZE  
RIGHT :ANGLE  
POLY :SIZE :ANGLE  
END
```

LibreLogo

```
TO POLY :SIZE :ANGLE  
FORWARD :SIZE  
RIGHT :ANGLE  
POLY :SIZE :ANGLE  
END
```

Snap!



Per utilizzare la procedura POLY, digitare la parola POLY seguita da valori specifici per gli ingressi come per esempio POLY 80 144

La Figura 2 mostra alcune delle tante forme ottenute richiamando POLY con vari input.

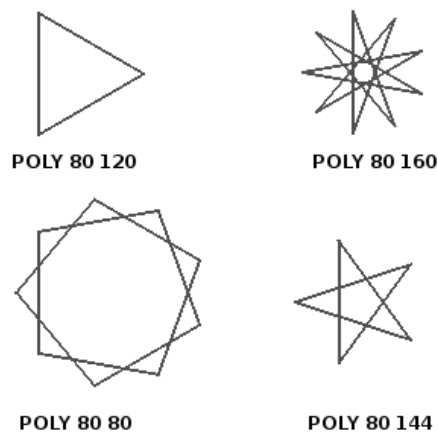


Figura 2: Queste figure sono tutte disegnate dal programma Logo POLY a tre linee, la tartaruga deve andare AVANTI di una certa quantità, girare a DESTRA di un certo angolo e ripetere questa operazione più e più volte. Le figure disegnate da POLY si chiudono sempre, ma il numero di lati che devono essere disegnati prima della chiusura della figura dipende dall'ingresso ANGLE della procedura.

Ricorsione è la parola che descrive la possibilità di utilizzare una procedura come parte della propria definizione. SQ e POLY sono procedure ricorsive di forma molto semplice, che si limitano a ripetere un ciclo immutabile più e più volte. Ma la ricorsione è un'idea molto più potente e può essere usata per ottenere effetti molto più complicati. Per fare solo un piccolo passo oltre il tipo puramente ripetitivo di ricorsione, considerare:

Abelson

```
TO POLYSPI :SIZE :ANGLE  
FORWARD :SIZE  
RIGHT :ANGLE  
POLYSPI :SIZE + 3 :ANGLE  
END
```

LibreLogo

```
TO POLYSPI :SIZE :ANGLE  
FORWARD :SIZE  
RIGHT :ANGLE  
POLYSPI :SIZE + 3 :ANGLE  
END
```

Snap!



Dando il comando POLYSPI 1 120 si ottiene questa sequenza di movimenti della tartaruga:

```
FORWARD 1 RIGHT 120  
FORWARD 4 RIGHT 120  
FORWARD 7 RIGHT 120  
FORWARD 10 RIGHT 120  
...
```

che produce una spirale triangolare in cui ciascuno dei lati è tre passi più grande del lato precedente. La Figura 3 mostra alcune delle forme generate dalla procedura POLIPSPI.

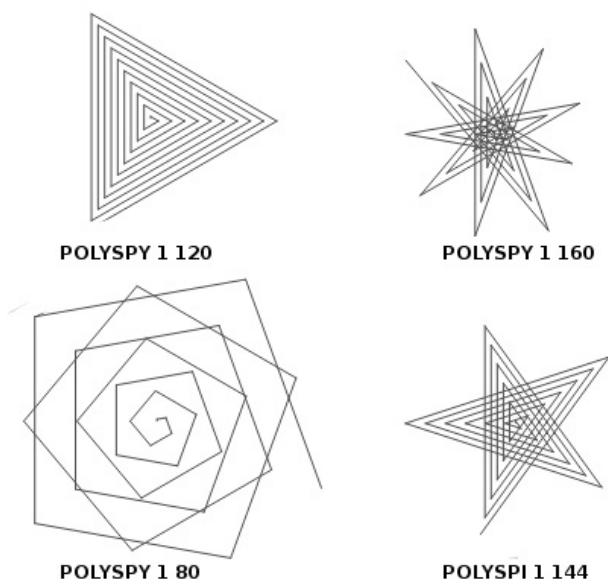


Figura 3: Figure create con POLYSPY, variante di POLY (vedi figura 2), il programma sfrutta la ricorsione per aumentare il passo di FORWARD della tartaruga ogni volta che la procedura viene chiamata. Il risultato è una spirale poligonale. Come per POLY, variando l'ingresso ANGLE si modifica la simmetria dello schema.

Come variante, è possibile sostituire FORWARD in POLYSPY con un comando che disegna un quadrato:

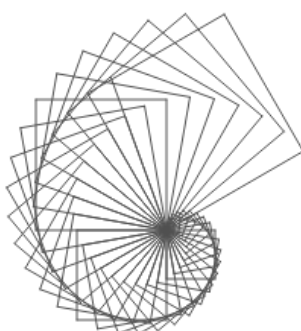
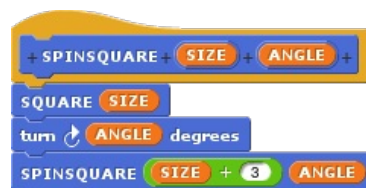
Abelson

```
TO SPINSQUARE :SIZE :ANGLE
  SQUARE :SIZE
  RIGHT :ANGLE
  SPINSQUARE :SIZE+3 :ANGLE
END
```

LibreLogo

```
TO SPINSQUARE :SIZE :ANGLE
  SQUARE1 :SIZE
  RIGHT :ANGLE
  SPINSQUARE :SIZE+3 :ANGLE
END
```

Snap!



SPINSQUARE 5 10

Figura 4: SPINSQUARE è un semplice programma ricorsivo che disegna un quadrato di una data dimensione, lo ruota, ne aumenta la dimensione e continua il processo.

Il risultato dell'esecuzione di SPINSQUARE 5 10 come mostrato in figura 4 è una sequenza di quadrati di dimensioni crescenti a partire da un quadrato della dimensione di un passo. Ogni quadrato è tre unità più grande della precedente e ruotato di 10 gradi rispetto ad esso. La procedura continua e i quadrati continuano a crescere fino a quando non si dice Logo di smettere.

È anche possibile modificare la procedura in modo che si arresti quando i quadrati diventano più grandi di una certa dimensione (ad esempio, 100 passi) includendo una regola di arresto:

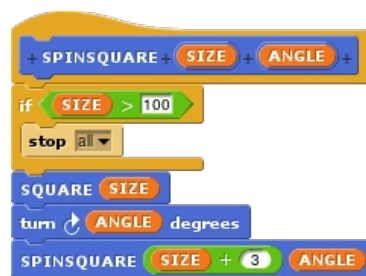
Abelson

```
TO SPINSQUARE :SIZE :ANGLE
IF :SIZE > 100 [STOP]
SQUARE :SIZE
RIGHT :ANGLE
SPINSQUARE :SIZE+3 :ANGLE
END
```

LibreLogo

```
TO SPINSQUARE :SIZE :ANGLE
IF :SIZE > 100 [ STOP ]
SQUARE1 :SIZE
RIGHT :ANGLE
SPINSQUARE :SIZE+3 :ANGLE
END
```

Snap!



Parte del potere di ricorsione è il fatto che programmi così semplici possono portare a risultati così diversi.

Un ambiente per esplorare

Come si può vedere dagli esempi presentati finora, è molto facile iniziare la programmazione con la grafica della tartaruga. Ciò è in parte dovuto alla grafica della tartaruga. I comandi di base hanno effetti semplici e visibili. Allo stesso tempo, la grafica della tartaruga è un'area incredibilmente ricca per l'esplorazione in cui anche programmi semplici possono dare risultati inaspettati, spesso belli. La piccola quantità Logo che abbiamo visto finora è sufficiente per supportare settimane di attività in programmazione e matematica, esplorando domande come "Come la forma di una figura POLY dipende dall'angolo di input" o "Perché tanti programmi ripetuti producono disegni simmetrici?" o semplicemente la creazione di bellissimi disegni. Con Andrea diSessa descriviamo alcune delle matematiche che nascono dall'indagine di questo approccio computerizzato alla geometria nel libro *Turtle Geometry: The Computer as a Medium for Exploring Mathematics* (Cambridge, MA: MIT Press, 1981).

Oltre all'ambiente, anche l'interazione del sistema svolge un ruolo cruciale. Quando si esplora l'uso del Logo, si definiscono continuamente nuove procedure e si modificano quelle vecchie. Un tipico sistema orientato alla compilazione, in cui cambiare una definizione richiede l'andare avanti e indietro tra editor separati, compilatori e linking loader, è inappropriato per questo tipo di attività. Gran parte dell'impegno profuso nell'attuazione del Logo è stato dedicato alla creazione di un ambiente di programmazione che faciliti la definizione e la modifica delle procedure.

Il successo della geometria della tartaruga è dovuto in gran parte al fatto che nel progettargli non ci si considerava solo come matematici ed educatori che cercavano di inventare un nuovo approccio alla geometria, né come informatici che cercavano di implementare un sistema, ma abbiamo cercato di assumere entrambe le prospettive, adattando continuamente il sistema informatico alla matematica e viceversa.

Output

Abbiamo già visto come definire procedure che richiedono input. È anche possibile impostare un valore come uscita di una procedura. Ad esempio, la seguente procedura prende due numeri come input e riporta come output la loro media:

Abelson

```
TO AVERAGE :X :Y
OUTPUT (:X + :Y) / 2
END
```

LibreLogo

```
TO AVERAGE :X :Y
OUTPUT (:X + :Y) / 2
END
```

Snap!



Il risultato restituito da AVERAGE può essere esaminato direttamente (con PRINT) o utilizzato a sua volta come input per altre operazioni:

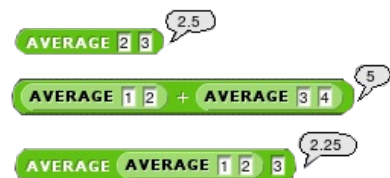
Abelson

```
PRINT (AVERAGE 2 3)
> 2.5
PRINT (AVERAGE 1 2) + (AVERAGE 3
4)
> 5
PRINT (AVERAGE (AVERAGE 1 2) 3)
> 2.25
```

LibreLogo

```
PRINT (AVERAGE 2, 3)
> 2.5
PRINT (AVERAGE 1, 2) + (AVERAGE
3, 4)
> 5.0
PRINT (AVERAGE (AVERAGE 1, 2), 3)
> 2.25
```

Snap!



Si osservi la convenzione Logo di usare le parentesi per raggruppare una procedura con i suoi input. Anche se le parentesi sono quasi sempre opzionali nelle semplici righe Logo, è una buona idea includerle perché rendono le righe più facili da leggere.

Programmare con le procedure

Un programma Logo è in genere strutturato come un insieme di procedure. Queste procedure trasmettono informazioni tra di loro attraverso gli input e gli output. Il vantaggio di questo tipo di organizzazione è che separa il programma in pezzi gestibili, in quanto ogni procedura può essere semplice in sé. Anche in un programma complesso, è insolito avere una singola procedura che è lunga più di poche righe. Inoltre, l'editor Logo integrato e il carattere interattivo generale del sistema Logo consentono di definire e testare separatamente le singole procedure.

Per illustrare l'organizzazione delle procedure, progettiamo un gioco semplice che si svolge come segue. Il computer sceglie a caso un "punto misterioso" sullo schermo e chiede al giocatore di effettuare mosse successive della tartaruga con LEFT e FORWARD. Prima di ogni spostamento, il computer stampa la distanza della tartaruga dal punto misterioso. L'obiettivo è quello di portare la tartaruga molto vicino al punto "misterioso" in meno mosse possibili. Ecco una trascrizione del gioco in azione. Le risposte del computer sono racchiuse tra asterischi per distinguerle da quelle del giocatore:

```

*Distance to point is 67.6 turn left how much?*
> 0
*Go forward how much?*
> 25
*Distance to point is 90.25 turn left how much?*
> 180
*Go forward how much?*
> 50
*Distance to point is 47.38*
...

```

E finalmente:

```

*Distance to point is 12.08 you won in 11 moves!*

```

Il cuore del programma è una procedura chiamata PLAY. Questa prende come input un numero M, che indica il numero di mosse fino ad ora effettuate. PLAY controlla prima se il giocatore ha vinto. In tal caso, prima di uscire dal programma, viene stampato un messaggio che indica quante mosse sono fatte. Altrimenti il programma chiede al giocatore di fare un'altra mossa, passando al turno successivo, con M aumentato di 1:

Abelson⁽⁶⁾

```

TO PLAY :M
TEST CHECKWIN?
IFTRUE [(PRINT [YOU WON IN] :M
[MOVES!])]
IFTRUE [STOP]
MAKEMOVE
PLAY :M + 1
END

```

LibreLogo

```

TO PLAY :M
IF CHECKWIN = TRUE ~
[ (PRINT "YOU WON IN " + STR(:M)
+ " MOVES!") STOP ] [ MAKEMOVE ]
PLAY :M + 1
END

```

Snap!⁽⁷⁾



La procedura PLAY è semplice in sé perché delega i problemi di test per la vittorie alla procedura CHECKWIN? e di eseguire le mosse a MAKEMOVE.

-
- ⁽⁶⁾ - LibreLogo non ha la primitiva Logo TEST che viene sostituita da IF condizione [lista istruzioni] [lista istruzioni] che corrisponde alla primitiva Logo IFELSE;
 - LibreLogo non tratta le liste come in Logo;
 - Librelogo non vuole il carattere ? nella definizione di una procedura.

- ⁽⁷⁾ Snap! non ha i blocchi TEST e PRINT che sono sostituiti dai blocchi if e say.

MAKEMOVE invita l'utente a inserire angoli e distanze e muove la tartaruga di conseguenza. Utilizza la sottoprocedura READNUMBER, che restituisce un numero digitato sulla tastiera:

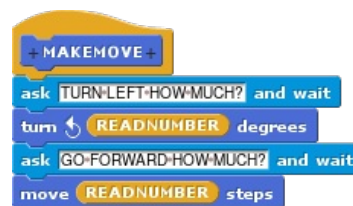
Abelson

```
TO MAKEMOVE
PRINT [TURN LEFT HOW MUCH?]
LEFT READNUMBER
PRINT [GO FORWARD HOW MUCH?]
FORWARD READNUMBER
END
```

LibreLogo⁽⁸⁾

```
TO MAKEMOVE
PRINT "TURN LEFT HOW MUCH?"
READNUMBER LEFT :TYPEIN
PRINT "GO FORWARD HOW MUCH?"
READNUMBER FORWARD :TYPEIN
END
```

Snap!



⁽⁸⁾ READNUMBER in LibreLogo non permette di passare con OUTPUT un numero a LEFT o FORWARD, mentre dà questa possibilità a comandi come PRINT. Ho risolto passando il numero con la variabile globale :TYPEIN di READNUMBER. Nella procedura MAKEMOVE, READNUMBER (vedi la procedura Librelogo di seguito) serve solo a verificare che ciò che si digita da tastiera sia un numero.

Per controllare la vittoria, il programma deve verificare se la posizione della tartaruga è vicina a un punto predeterminato (ad esempio, 20 passi). Le primitive Logo XCOR e YCOR restituiscono le coordinate x e y della tartaruga. Supponiamo che le coordinate x e y del punto nascosto siano date dalle variabili XPT e YPT. Se si assume che ci sia una procedura DISTANCE che restituisce la distanza tra due punti, la procedura CHECKWIN? può essere scritta come segue:

Abelson

```
TO CHECKWIN?
MAKE "D DISTANCE :XCOR :YCOR :XPT
:YPT
(PRINT [DISTANCE TO POINT IS] :D)
IF :D < 20 [OUTPUT "TRUE]
OUTPUT "FALSE
END
```

LibreLogo⁽⁹⁾

```
TO CHECKWIN
:XCOR = POS[0] - :XHOME
:YCOR = -(POS[1] - :YHOME)
(PRINT "DISTANCE TO POINT IS" +
STR(:D))
:D = DISTANCE :XCOR :YCOR :XPT
:YPT|
IF :D < 20 [ OUTPUT TRUE ]
OUTPUT FALSE
END
```

Snap!



⁽⁹⁾ Mancano in LibreLogo le primitive XCOR e YCOR quindi per calcolare la distanza occorre utilizzare POSITION che non ha il riferimento cartesiano usuale HOME=[0,0]. Si possono ottenere le coordinate di HOME in LibreLogo assegnandole come variabili globali nella procedura GAME in questo modo :XHOME=POS[0] e :YHOME=POS[1].

CHECKWIN? restituisce come valore TRUE o FALSE, valore che viene verificato da PLAY per determinare se il gioco è finito. Osservare anche l'uso del comando MAKE per assegnare valori alle variabili. In questo caso, la variabile D viene utilizzata per indicare la distanza.

Ecco la procedura per calcolare la distanza tra due punti, come radice quadrata della somma dei quadrati delle differenze delle coordinate:

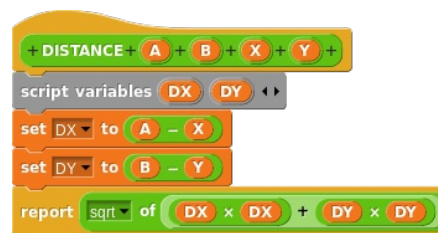
Abelson

```
TO DISTANCE :A :B :X :Y
MAKE "DX :A - :X
MAKE "DY :B - :Y
OUTPUT SQRT (:DX*:DX + :DY*:DY)
END
```

LibreLogo

```
TO DISTANCE :A :B :X :Y
:DX = :A - :X
:DY = :B - :Y
OUTPUT SQRT (:DX*:DX + :DY*:DY)
END
```

Snap!



Ora è necessaria una procedura per iniziare il gioco:

Abelson

```
TO GAME
CLEARSCREEN
MAKE "XPT RANDOMCOORD
MAKE "YPT RANDOMCOORD
PLAY 0
END
```

LibreLogo⁽¹⁰⁾

```
TO GAME
GLOBAL :XHOME, :YHOME, :XPT, :YPT
CLEARSCREEN HOME
:XHOME = POSITION[0]
:YHOME = POSITION[1]
:XPT = RANDOMCOORD
:YPT = RANDOMCOORD
PLAY 0
END
```

Snap!⁽¹¹⁾



La procedura GAME azzera lo schermo, assegna valori (scelti a caso) alle coordinate del punto misterioso XPT e YPT, e chiama PLAY con una M iniziale pari a zero.

⁽¹⁰⁾ In LibreLogo occorre dichiarare globali le variabili :XPT, :YPT, :XHOME e :YHOME per renderle visibili all'esterno della procedura. In Logo le variabili dichiarate con MAKE sono globali.

⁽¹¹⁾ - Anche in Snap! occorre dichiarare globali le variabili XPT e YPT;
- Il blocco inizializza è utilizzato per orientare la tartaruga a 0°, riportandola al centro dello schermo pulendo lo schermo.

La seguente procedura, utilizzata per selezionare le coordinate casuali, restituisce un numero casuale compreso tra -75 e +75. Funziona chiamando la primitiva Logo RANDOM per ottenere un numero casuale compreso tra 0 e 150 e sottraendo 75 dal risultato:

Abelson

```
TO RANDOMCOORD
OUTPUT (RANDOM 150) - 75
END
```

LibreLogo

```
TO RANDOMCOORD
OUTPUT (RANDOM 150) - 75
END
```

Snap!



L'unica cosa necessaria per completare il programma è READNUMBER, che restituisce un numero da input dalla tastiera:

```
TO READNUMBER
OUTPUT FIRST REQUEST
END
```

READNUMBER usa la primitiva Logo REQUEST⁽¹²⁾, che attende che l'utente digiti una linea e poi restituisce una lista di tutti gli elementi in quella linea. Il numero desiderato viene estratto come prima voce dell'elenco di immissione. (Parleremo di liste in seguito).

(12) La primitiva REQUEST non è presente in UcbLogo ed è sostituita da READLIST.

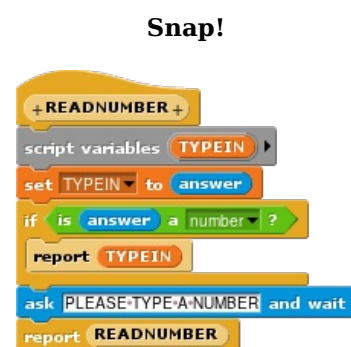
In realtà, potrebbe essere meglio progettare READNUMBER in modo che controlli per vedere se l'oggetto da restituire è effettivamente un numero e produrre un errore in caso contrario:

Abelson

```
TO READNUMBER
MAKE "TYPEIN FIRST READLIST
IF NUMBER? :TYPEIN [OUTPUT
:TYPEIN]
PRINT [PLEASE TYPE A NUMBER]
OUTPUT READNUMBER
END
```

LibreLogo⁽¹³⁾

```
TO READNUMBER
GLOBAL :TYPEIN
:TYPEIN = INPUT "TYPE A NUMBER"
IF FINDALL("[0-9]|\.|-", :TYPEIN)
[ :TYPEIN = FLOAT(:TYPEIN) ]
[ PRINT "PLEASE TYPE A NUMBER"
OUTPUT READNUMBER ]
END
```



Si osservi l'ultima riga della procedura. Il suo effetto è quello di far ripetere, tutte le volte che è necessario READNUMBER fino a che non viene digitato un numero.

Questo fa sì che il gioco si comporti come segue:

```
*GO FORWARD HOW MUCH?*
> FJKSL
*PLEASE TYPE A NUMBER*
> FIFTY
*PLEASE TYPE A NUMBER*
> 50
...
```

(13) In LibreLogo manca la primitiva NUMBER?, occorre quindi utilizzare FINDALL con le espressioni regolari. In READNUMBER la condizione è vera se si digitano cifre, il carattere "." e il carattere "-". In questo caso dato che INPUT considera solo stringhe di testo occorre trasformare la stringa di testo in numero con FLOAT prima di inviarlo alla procedura che lo richiede. Se la condizione è falsa verrà stampato un messaggio che chiede di digitare un numero e sarà richiamata la stessa procedura.

Liste

Abbiamo visto che la struttura procedurale di Logo lo rende un linguaggio facile e adatto per scrivere programmi. La maggior parte dei linguaggi di programmazione moderni sono, infatti, organizzati proceduralmente, anche se pochi linguaggi rendono così facile definire e modificare le procedure in modo interattivo come fa Logo.

Un aspetto interessante Logo è il modo in cui gestisce le raccolte di dati. Questo viene fatto utilizzando le liste. Una lista è una sequenza di dati. Ad esempio:

[1 2 BUCKLE MY SHOE] è un elenco di cinque elementi. Gli elementi di una lista possono essere essi stessi liste, come in:

[[PETER PAN] WENDY JOHN] che è una lista di tre elementi, il primo dei quali è a sua volta una lista di due elementi. Allo stesso modo, possiamo avere liste i cui elementi sono liste, e così via. Le liste, quindi, sono un modo naturale di rappresentare le *strutture gerarchiche*, cioè le strutture composte da parti che a loro volta sono composte da parti.

Logo include una serie di operazioni per la manipolazione delle liste. FIRST estrae il primo elemento della lista. In questo esempio:

FIRST [1 2 BUCKLE MY SHOE] è 1, nel prossimo esempio:

FIRST [[PETER PAN) WENDY JOHN] è [PETER PAN].

L'operazione BUTFIRST restituisce la lista composta da tutti gli elementi tranne il primo, quindi in

BUTFIRST [1 2 BUCKLE MY SHOE] riporta [2 BUCKLE MY SHOE], mentre in

BUTFIRST [[PETER PAN] WENDY JOHN] riporta [WENDY JOHN].

L'operazione FPUT prende i due oggetti x e y e crea una lista il cui FIRST è x e in cui BUTFIRST è y. Ad esempio:

FPUT 5 [2 BUCKLE MY SHOE] produce la lista [5 2 BUCKLE MY SHOE], e

FPUT [PETER PAN] [BUCKLE MY SHOE] produce la lista [[PETER PAN] BUCKLE MY SHOE].

L'operazione SENTENCE, come FPUT, costruisce liste più grandi da liste più piccole, ma in modo leggermente diverso. SENTENCE prende come input più liste e combina tutti i loro elementi per produrre un'unica lista. Ad esempio:

SENTENCE [PETER PAN] [BUCKLE MY SHOE] produce la lista [PETER PAN BUCKLE MY SHOE].

La cosa importante delle liste in Logo è che possono essere manipolate come ciò che gli informatici chiamano "dati di prima classe". Cioè, le liste Logo (in contrapposizione, ad esempio, agli array in BASIC) possono essere:

- assegnate come valore alle variabili
- passate come input alle procedure
- restituite come output delle procedure

Ad esempio, si possono assegnare nomi di variabili alle liste:

MAKE "X [OOM PAH] MAKE "Y [NEIGH H0] e quindi fare riferimento ai valori di queste variabili, in modo che il risultato di BUTFIRST :X sia la lista [PAH]. È anche possibile combinare operazioni sulle liste per produrre operazioni più complesse. Ad esempio:

FIRST FIRST [[PETER PAN] WENDY JOHN] riporta la parola PETER.

Si possono anche scrivere procedure che manipolano le liste:

```
TO DOUBLE :L
  OUTPUT SENTENCE :L :L
END
```

```
PRINT DOUBLE [OOM PAH]
> *OOM PAH OOM PAH*
PRINT DOUBLE DOUBLE [OOM PAH]
> *OOM PAH OOM PAH OOM PAH OOM PAH*
```

Ne consegue che è possibile combinare operazioni su liste, così come si fanno le operazioni su numeri nei linguaggi ordinari. Per esempio, un'operazione molto utile della lista è PICKRANDOM, che sceglie un elemento a caso da una lista di elementi. PICKRANDOM non è una primitiva, ma è facilmente costruibile con operazioni più semplici, come ad esempio trovare la lunghezza di una lista, selezionare un numero casuale in un dato intervallo, ed estrarre un elemento a caso della lista.


Giocare con i testi

Per illustrare come vengono utilizzate le liste, esaminiamo un programma che scrive cartoline per le vacanze, come ad esempio:

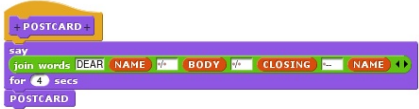
```
DEAR DOROTHY
WISH YOU WERE HERE.
LOVE -- JOHN
```

```
DEAR MARY
EVERYONE' S FINE.
WRITE SOON -- AUNT EM
```

Si inizia con la creazione di liste di nomi e frasi tra cui gli elementi della cartolina saranno scelti:


Abelson	LibreLogo	Snap!
<pre>MAKE "NAMES [JOHN DOROTHY [AUNT EM] OCCUPANT] MAKE "PHRASES [[WISH YOU WERE HERE.] [WEATHER'S GREAT!] [SURF'S UP.] [EVERYONE'S FINE.]] MAKE "CLOSINGS [LOVE [SEE YOU SOON][WRITE SOON]]</pre>	<pre>GLOBAL :NAMES, :PHRASES, :CLOSINGS :NAMES = ["JOHN", "DOROTHY", "AUNT EM", "OCCUPANT"] :PHRASES =["WISH YOU WERE HERE.", "WEATHER'S GREAT!", "SURF'S UP.", "EVERYONE'S FINE."] :CLOSINGS =["LOVE", "SEE YOU SOON", "WRITE SOON"]</pre>	

Ecco il programma principale delle cartoline:

Abelson	LibreLogo	Snap!
<pre>TO POSTCARD PRINT SENTENCE [DEAR] NAME PRINT BODY PRINT (SENTENCE CLOSING [--] NAME) POSTCARD END</pre>	<pre>TO POSTCARD PRINT ("DEAR " + NAME + '\n'~ +BODY + '\n'+ CLOSING + " - - " + NAME) POSTCARD END</pre>	

La chiamata ricorsiva nell'ultima riga fa sì che la procedura continui a stampare nuove cartoline più e più volte (confrontare le procedure SQ e POLY).

Le procedure NAME, BODY e CLOSING generano gli elementi della cartolina selezionandoli dalle liste appropriate:

Abelson ⁽¹⁴⁾	LibreLogo	Snap!
<pre>TO NAME OUTPUT PICK :NAMES END</pre>	<pre>TO NAME OUTPUT RANDOM :NAMES END</pre>	

⁽¹⁴⁾ Sia UcbLogo che LibreLogo hanno primitive che permettono di estrarre a caso un elemento da una lista (rispettivamente PICK e RANDOM), in Snap! bisogna costruirla, nominandola PICKRANDOM.

È possibile modificare il programma POSTCARD in modo che aumenti automaticamente il suo repertorio di frasi di tanto in tanto (ad esempio, una possibilità su tre) chiedendo all'utente di digitare una nuova frase e aggiungendola a PHRASES . Per farlo, aggiungere alla procedura POSTCARD la linea: IF 1.IN.3 LEARN.NEW.PHRASE.

```

TO POSTCARD
IF 1.IN.3 [LEARN.NEW.PHRASE]
PRINT SENTENCE [ DEAR] NAME
PRINT BODY
PRINT (SENTENCE CLOSING [--] NAME)
POSTCARD
END

```

La procedura 1.IN.3 restituisce VERO con probabilità uno su tre e FALSO in caso contrario. Uno dei modi possibili per scrivere questa procedura è:

Abelson

```

TO 1.IN.3
IF (RANDOM 3 ) = 0 [OUTPUT "TRUE]
OUTPUT "FALSE
END

```

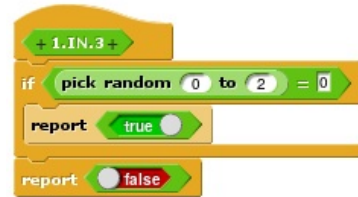
LibreLogo⁽¹⁵⁾

```

TO UNO_IN_TRE
IF ROUND(RANDOM 2) = 0 [ OUTPUT
TRUE ] [ OUTPUT FALSE ]
END

```

Snap!



(15) LibreLogo non accetta nomi di procedura che inizino con numeri e che contengano al loro interno punti.

Ecco come il programma impara una nuova frase:

Abelson

```

TO LEARN.NEW.PHRASE
PRINT [PLEASE TYPE IN A NEW
PHRASE]
MAKE "PHRASES FPUT READLIST
:PHRASES
END

```

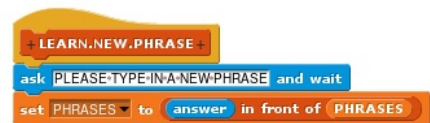
LibreLogo⁽¹⁶⁾

```

TO LEARN_NEW_PHRASE
GLOBAL :PHRASES
:TYPEIN = INPUT 'PLEASE TYPE IN A
NEW PHRASE'
:NEW_PHRASE = [:TYPEIN]
:PHRASES = SET :NEW_PHRASE | SET
:PHRASES
END

```

Snap!



(16) LibreLogo non ha una primitiva tipo FPUT per le liste. Ho utilizzato il comando SET che converte una lista in un insieme Python, usando poi l'unione ("|") fra due insiemi.

L'idea è che REQUEST (READLIST in Ucblogo) restituisca (come lista) la frase che l'utente digita in risposta al messaggio . Questo viene aggiunto a PHRASES (per mezzo di FPUT), in modo che il programma sia in grado di utilizzare questa frase in cartoline future, come:

```

*PLEASE TYPE IN A NEW PHRASE*
> DON'T FORGET TO FEED THE DOG.



```

```

*DEAR OCCUPANT*
*DON'T FORGET TO FEED THE DOG.*
*LOVE -- JOHN*

```

Un'altra modifica che si può fare è generare cartoline più lunghe, in cui BODY è costituito da una o più frasi. Un modo per farlo è modificare la procedura BODY come segue:

Abelson	LibreLogo	Snap!
<pre> TO BODY IF 1.IN.2 [OUTPUT SINGLE.PHRASE] OUTPUT SENTENCE BODY SINGLE.PHRASE END </pre>	<pre> TO BODY IF UNO_IN_DUE [OUTPUT SINGLE_PHRASE] [OUTPUT BODY + SINGLE_PHRASE] END </pre>	
<pre> TO SINGLE.PHRASE OUTPUT PICK :PHRASES END </pre>	<pre> TO SINGLE_PHRASE OUTPUT RANDOM :PHRASE END </pre>	

La nuova procedura BODY usa la ricorsione in modo ambiguo. Ogni volta che si chiama BODY, una volta su due produrrà una singola frase, come prima (la procedura 1.IN.2 è analoga alla procedura 1.IN.3 vista precedentemente) e alternativamente genera ricorsivamente un nuovo BODY combinandolo (usando SENTENCE) con una singola frase. Il nuovo BODY (chiamato ricorsivamente) genererà una singola frase solo una volta su due. Altrimenti, chiamerà un terzo BODY. Il risultato generale è che la chiamata alla procedura BODY genererà una singola frase circa una volta su due, due frasi circa una volta su quattro, tre frasi circa una volta su otto e così via.

Ecco il programma finale cartolina in azione:

```

*DEAR A AUNT EM*
*SURF'S UP. DON'T FORGET TO FEED THE DOG.*
*WRITE SOON -- DOROTHY*

*PLEASE TYPE IN A NEW PHRASE*
> GET THE MONEY IN SMALL BILLS.

*DEAR OCCUPANT*
*WEATHER'S GREAT. WISH YOU WERE HERE. GET THE MONEY IN SMALL BILLS.*
*SEE YOU SOON -- JOHN*

```

Le procedure che generano testi sono divertenti da scrivere e con cui giocare e anche facili da modificare. Si possono rendere elaborate o semplici applicando le stesse idee per la produzione di saggi, poesie, lettere d'amore, e così via. L'idea di generare un programma di cartoline casuali si basa sul lavoro svolto al MIT da Neil Rowe, il cui l'articolo "Grammar as a Programming Language" (*Creative Computing*, gennaio/febbraio 1978) contiene molti altri esempi di procedure per la generazione di testi. Esso mostra anche come implementare, in Logo, uno speciale "sottolinguaggio" per la creazione di tali programmi.

Pavimentazioni ricorsive

Il successo del Logo come catalizzatore per l'apprendimento coinvolge molto di più del linguaggio stesso, anche se il linguaggio gioca un ruolo fondamentale. La migliore attività Logo è una sintesi di programmazione, matematica, estetica e, soprattutto, l'opportunità di esplorare. Un esempio particolarmente significativo è il programma di "pavimentazione ricorsiva" ideato da Andrea diSessa e Doug Hill. Questo schema permette di scrivere semplici procedure che disegnano modelli come quelli mostrati nelle foto 2 e 3, dando letteralmente miliardi di possibilità da esaminare ed esplorare.

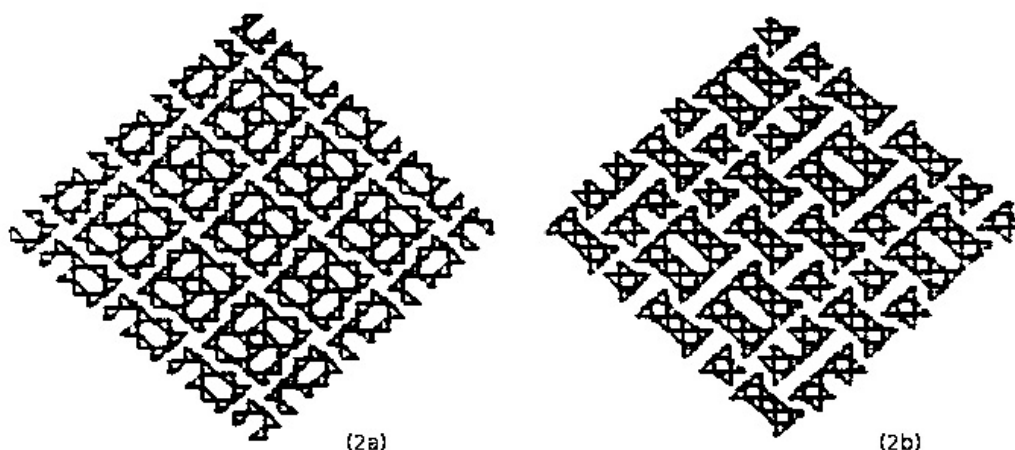


Foto 2a e 2b : Due motivi di pavimentazione costruiti con la stessa figura di base (un triangolo all'interno di un quadrato). Le differenze nei modelli sono dovute a orientamenti diversi quando la figura di base viene combinata con modelli "di livello superiore".

L'idea è la seguente. Si supponga di avere un programma che disegna un modello all'interno di un quadrato di una certa dimensione. Riducendo a metà la dimensione del modello e incollandone quattro copie, si ottiene un modello più complesso in un quadrato della dimensione originale, come mostrato nella Figura 5a. Infatti, è possibile generare molti modelli diversi da un unico modello, perché ogni copia del modello originale che si inserisce in ogni angolo del quadrato può essere ruotata attraverso un multiplo arbitrario di 90 gradi, come mostrato in figura 5b.

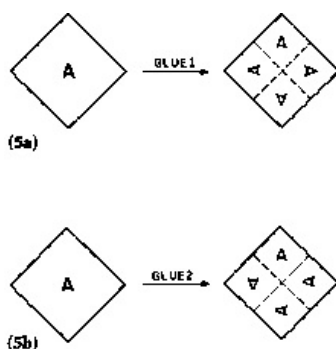


Figura 5: Dato un disegno che si trova all'interno di un quadrato, quattro copie dello stesso disegno (ciascuno in scala dimezzata rispetto all'originale) produrrà un modello più complesso nello stesso quadrato. Variando gli orientamenti relativi delle quattro copie si ottengono disegni generali diversi. Questo è lo schema base per produrre i modelli nelle foto 2a, 2b, 3a, e 3b, partendo da un triangolo, e ripetendo il processo di incollaggio quattro volte (cioè lavorando con un elemento finale del modello in scala 1/16 della dimensione del quadrato originale).

Per convertire questa idea in un programma per computer, si supponga di avere una procedura chiamata PROC che disegna un modello in un quadrato. Si supponga che PROC prenda un input S che specifichi la dimensione del quadrato, scalato in modo che S sia uguale alla metà della diagonale del quadrato. Si supponga inoltre che PROC sia concepita per iniziare a disegnare con la tartaruga al centro del quadrato che punta verso un vertice e per terminare con la tartaruga nello stesso stato.

Ora si supponga di avere un quadrato di dimensione S diviso in quattro “quadrati d’angolo” ciascuno di dimensione S/2. Il seguente processo è progettato per iniziare con la tartaruga al centro del quadrato, di fronte a uno degli angoli. Disegna una copia del disegno PROC in quell’angolo quadrato e riporta la tartaruga al centro del quadrato più grande. Poi gira la tartaruga di 90 gradi per puntare al prossimo quadrato d’angolo. Le fasi del processo sono le seguenti:

1. Spostare la tartaruga in avanti di una distanza pari a S/2. Questo porta la tartaruga al centro del quadratino, puntando verso un vertice di quel quadratino.
2. Eseguire la procedura PROC con un ingresso S/2 (metà della diagonale del quadrato più piccolo). Questo disegna il modello e lascia la tartaruga al centro del quadrato più piccolo.
3. Spostare la tartaruga INDIETRO di una distanza pari a S/2 per riportarla al centro del quadrato più grande.
4. Ruotare la tartaruga di 90 gradi.

Inoltre, prima di eseguire la fase 2, è possibile ruotare il modello di un multiplo di 90 gradi. Se si esegue questa operazione, è necessario eseguire la rotazione opposta alla fine del passaggio 2, in modo che la tartaruga finisca per essere rivolta nella stessa direzione da cui è partita.

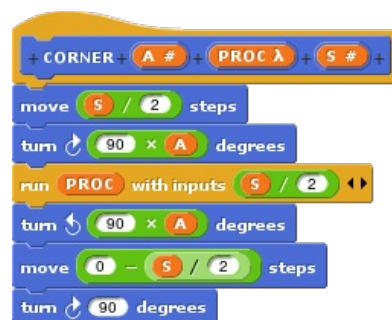
La procedura CORNER attua tale strategia. CORNER ha tre input. Il primo, A, è un multiplo di 90 gradi da ruotare prima di disegnare il modello (A è un numero intero da 0 a 3). L’ingresso successivo, PROC, è il nome della procedura che disegna il modello. Si presume che PROC abbia un ingresso che specifica la dimensione del modello. Il terzo ingresso di CORNER è un numero S che specifica la dimensione del quadrato. La procedura viene descritta di seguito:

Abelson


```
TO CORNER :A :PROC :S
FORWARD :S/2
RIGHT 90 * :A
DRAWFIGURE :PROC :S/2
LEFT 90 * :A
BACK :S/2
RIGHT 90
END
```

LibreLogo⁽¹⁷⁾

Snap!⁽¹⁸⁾



(17) Non sono riuscito a implementare in LibreLogo la procedura DRAWFIGURE.

(18)  Comando che in Snap! sostituisce la procedura DRAWFIGURE.

La procedura CORNER utilizza la sottoprocedura DRAWFIGURE, che prende come input il nome di una procedura e un numero ed esegue la procedura con il numero come input. DRAWFIGURE è implementato attraverso la primitiva Logo RUN, che esegue una lista come se si digitasse riga di comando:

```
TO DRAWFIGURE :PROC :INPUT
  RUN SENTENCE :PROC :INPUT
END
```

Per esempio, se si esegue: DRAWFIGURE [SQUARE] 100 combina [SQUARE] e 100 con SENTENCE per ottenere la lista [SQUARE 100] ed esegue questa lista come se fosse una riga di comando Logo, cioè esegue il comando SQUARE 100⁽¹⁹⁾.

La procedura CORNER inizia con la tartaruga che punta ad un angolo del quadrato più grande e termina con la tartaruga che punta all'angolo successivo. Ciò significa che è possibile ottenere un disegno completo di incollaggio eseguendo CORNER quattro volte. Ognuna delle quattro chiamate a CORNER può specificare un diverso multiplo di 90 gradi di A, attraverso il quale il disegno in quell'angolo deve essere ruotato. Dal momento che ogni incollaggio ha quattro angoli e ogni angolo può avere una qualsiasi delle quattro rotazioni, ci sono 4^4 o 256 incollaggi possibili per ogni modello dato. Qui ci sono due possibili incollaggi:

(19) LibreLogo non ha la primitiva RUN che permette di eseguire una procedura all'interno di un'altra il cui nome è dato come input.

Abelson

```
TO GLUE1 :PROC :S
  REPEAT 4[CORNER 0 :PROC :S]
END

TO GLUE2 :PROC :S
  CORNER 0 :PROC :S
  CORNER 2 :PROC :S
  CORNER 1 :PROC :S
  CORNER 3 :PROC :S
END
```

Snap!



La foto 4b mostra il risultato dell'inserimento di GLUE [TRI] 100⁽²⁰⁾ in Logo e di



in Snap!

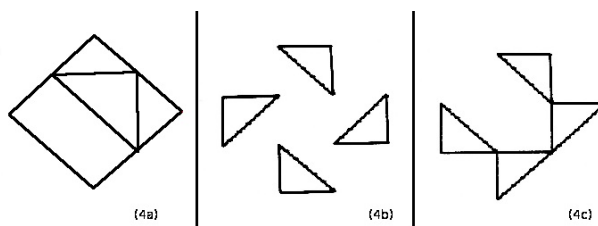


Foto 4: La foto 4a mostra un triangolo all'interno di un quadrato, come disegnato dalla procedura TRI. Le foto 4b e 4c mostrano due diversi tipi di incollaggio della procedura TRI per formare una figura più complicata all'interno di un quadrato. I disegni complessi nelle foto 2a, 2b, 3a e 3b sono quattro livelli di incollaggio basati sulla stessa figura TRI.

TRI è una procedura che disegna un piccolo triangolo all'interno di un quadrato, come mostrato nella foto 4a. (In conformità con la strategia generale di incollaggio, TRI dovrebbe essere una procedura che richiede un input che specifica la dimensione del quadrato).

Questa è una possibile procedura per TRI⁽²¹⁾.

Abelson

```
TO TRI :MEZZA_DIAG
MAKE "MEZZA_BASE (:MEZZA_DIAG * SQRT 2)/2
LT 45
FD :MEZZA_BASE RT 135 FD :MEZZA_DIAG
RT 90 FD :MEZZA_DIAG
RT 135 FD :MEZZA_BASE
RT 45
END
```

Snap!



(20) Il valore di input è metà della diagonale del quadrato che contiene i quattro modelli.



(21) La seguente procedura non è presente nell'articolo di Abelson.

A titolo di confronto, la foto 4c mostra un incollaggio diverso: GLUE2 [TRI] 100 in Logo e



in Snap!

Ora arriva l'idea intelligente. Le procedure GLUE consentono di incollare insieme quattro copie di qualsiasi modello. D'altra parte, inserendo `GLUE1 [TRI] 100` è *anche* un comando che disegna uno schema in un quadrato. Infatti, la lista `[GLUE 1 [TRI]]`, quando combinata con una dimensione (tramite la procedura `DRAWFIGURE`), produce un comando che disegna un modello in un quadrato della dimensione specificata. Pertanto, è possibile utilizzare una procedura GLUE per incollare insieme, ad esempio, quattro di questi modelli:

`GLUE1 [GLUE1 [TRI]] 100` o `GLUE2 [GLUE1 [TRI]] 100` in Logo e  o  in Snap!

Ma ancora una volta, ognuno di questi incollaggi "a due livelli" è di per sé qualcosa che può essere incollato, in modo da ottenere modelli a tre livelli, come ad esempio `GLUE2 [GLUE1 [GLUE2 [TRI]]] 100` e così via. I modelli riportati nelle foto 2a, 2b, 3a e 3b sono, infatti, basati tutti a quattro livelli di incollaggio sulla stessa procedura `TRI`, utilizzando rotazioni diverse nei vari livelli.

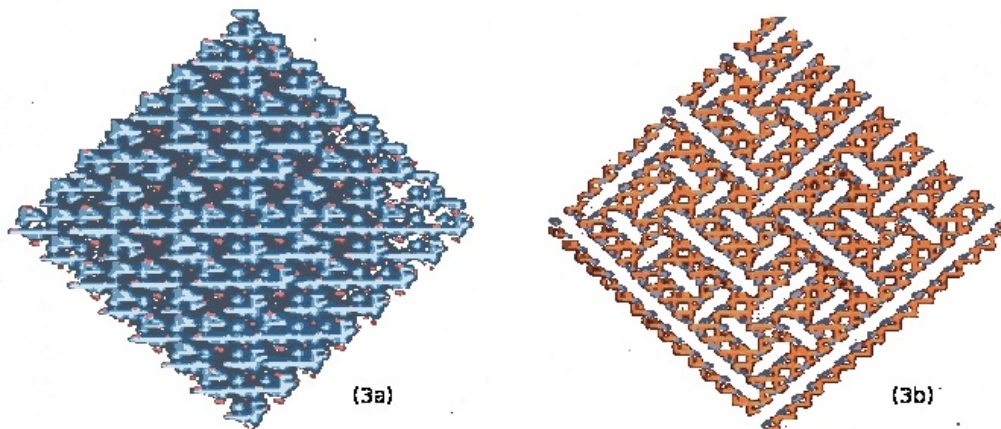


Foto 3a e 3b : Due motivi di pavimentazione costruita sullo stesso triangolo di base delle foto 2a e 2b. Tutti e quattro i modelli utilizzano lo schema di incollaggio mostrato nelle figure 5a e 5b e nelle foto 4a, 4b e 4c, ci sono quattro livelli di incollaggio.

Qui c'è un'enorme varietà di possibilità da indagare. Quattro livelli di incollaggio con 256 scelte di orientamento ad ogni livello danno 256^4 o più di 4 miliardi di possibili incollaggi a quattro livelli, tutti da un unico modello di base! (Il numero di modelli distinti è ridotto da varie simmetrie nel processo di incollaggio, che è di per sé un fenomeno interessante da esplorare). Per più varietà, è possibile provare diversi modelli di base, o anche sviluppare diversi schemi di incollaggio, come quello derivato dalla divisione di un triangolo equilatero in quattro triangoli più piccoli equilateri. (Per ulteriori informazioni sui "disegni ricorsivi", vedere *Turtle Geometry*).

La prospettiva computazionale

Il logo è spesso descritto come un linguaggio di programmazione. Quelli di noi che hanno progettato Logo tendono a pensarlo piuttosto come un ambiente di apprendimento computerizzato, dove le attività (esplorare la simmetria di POLY) sono integrate agli strumenti di programmazione utilizzati (ricorsività e liste). Logo è anche un ambiente in continua evoluzione e le implementazioni di microcomputer Logo che sono apparsi nel corso dell'ultimo anno sono solo i primi ad essere ampiamente disponibili. Abbiamo in programma di estendere Logo per includere nuove caratteristiche linguistiche, come il "messaggio che passa" strutture che si trovano in Smalltalk e nelle recenti implementazioni di LISP (vedi il numero di agosto 1981 di BYTE per una panoramica di Smalltalk), così come nuove attività, come un programma di fisica basato sulla geometria delle tartarughe. Presso il Laboratorio di Informatica del MIT, il Gruppo Educational Computing sta progettando un sistema Logo adatto alla nuova generazione di personal computer che entrerà in uso nella seconda metà degli anni Ottanta.

I prossimi anni saranno entusiasmanti per l'informatica didattica, perché i personal computer stanno diventando abbastanza potenti da supportare sistemi progettati per la comodità delle persone piuttosto che per la comodità dei compilatori. Se riusciamo a sfatare l'illusione che conoscere i computer dovrebbe essere un'attività per manipolare indici array e per preoccuparci se X è un numero intero o un numero reale, allora possiamo iniziare a concentrarci sulla programmazione come fonte di idee. La programmazione è un'attività di descrizione delle cose. Le descrizioni sono formulate in modo da poter essere interpretate da un computer, ma questo non è così importante. Le descrizioni computazionali, come quelle della scienza o della matematica, forniscono una prospettiva, una raccolta di "strumenti del pensiero", come l'organizzazione procedurale, la struttura gerarchica e le formulazioni ricorsive. Il Logo, e linguaggi simili, contribuiranno a rendere questi strumenti disponibili a tutti.