Piccolo manuale di LibreLogo

La Geometria della Tartaruga

Andreas R. Formiconi

Versione parziale 0.4

9 settembre 2016

Jan Fabre, "Searching for Utopia", 2003

Licenza

Prefazione

Questo piccolo manuale nasce per la necessità di fornire supporto di studio e consultazione nell'insegnamento “Laboratorio di Tecnologie Didattiche” al V anno del Corso di Laura Magistrale a ciclo unico “Scienze della Formazione Primaria” e nell'insegnamento “Laboratorio di Gestione dei Processi Formativi” al II anno del Corso di Laurea Magistrale “Scienze dell'Educazione degli Adulti, della Formazione Continua e Scienze Pedagogiche”, presso l'Università di Firenze, e nell'insegnamento “Informatica” al I anno del Corso di Laurea Magistrale “Innovazione Educativa e Apprendimento Permanente” presso l'università telematica Italian University Line. Il manuale guida all'impiego del linguaggio Logo nella versione LibreLogo implementata all'interno del *word processor* Writer della *suite* di programmi di produttività personale LibreOffice. LibreLogo è un *plugin* disponibile di *default* in Writer a partire dalla versione 4.0 di LibreOffice. È stato scritto in linguaggio Python da László Németh. La documentazione disponibile si trova in http://librelogo.org, da dove, in particolare, si può scaricare una guida dei comandi di LibreLogo in italiano[1]. Per il resto, sfortunatamente e per quanto è a mia conoscenza sino ad oggi, la documentazione disponibile è tutta in ungherese, principalmente sotto forma di un manualetto conciso scritto dallo stesso László Németh[2] e da un manuale esteso scritto da Lakó Viktória[3]. È a quest'ultimo lavoro che, in una prima fase si è ispirato il presente piccolo manuale, senza tuttavia esserne una traduzione, per vari motivi. In primo luogo io non so l'ungherese e non posso quindi pretendere di poterne fare una vera traduzione e i tempi e le circostanze non mi consentono di avvalermi di un traduttore. Posso tuttavia seguirne le tracce, aiutandomi con i codici (anche se in ungherese quelli si possono imparare), le figure e Google Translate. Del resto, alla fine una traduzione pedissequa non sarebbe nemmeno desiderabile perché viene naturale riformulare il materiale in funzione degli obiettivi specifici e della propria visione della materia. Inoltre, nel corso della traduzione, mi è capitato sempre più spesso di seguire la traccia dei miei pensieri e, alla fine, è stato inevitabile tornare alla fonte primigenia, ovvero al testo con cui Seymour Papert descrisse per la prima volta compiutamente il pensiero che aveva dato origine a Logo, *Mindstorms*[4]. È così che ho introdotto la traduzione di due capitoli di *Mindstorms*: il secondo, “*Mathofobia: the Fear of Learning*”, e il terzo, “*Turtle Geometry: A Mathematics Made for Learning*”.

L'immersione profonda nel pensiero di Papert ha poi prodotto un fenomeno interessante. Nei numerosi passaggi dove Papert insiste sulla necessità di proporre agli studenti nuove idee matematiche facendo leva sulle conoscenze già possedute (non solo scolastiche) dagli studenti e sul loro coinvolgimento personale, sempre più spesso mi venivano in mente le lezioni di Emma Castelnuovo, con le quali si impiegano materiali semplici per introdurre tanti concetti matematici. Ad esempio[5]:

---

1 https://help.libreoffice.org/Writer/LibreLogo_Toolbar/it
2 http://www.numbertext.org/logo/logofuzet.pdf
3 http://szabadszoftver.kormany.hu/wp-content/uploads/librelogo_oktatasi_segedanyag_v4.pdf
4 Seymour Papert (1980), *Mindstorms*, Children, Computers, and Powerful Ideas. New York: Basic Books.
5 E. Castelnuovo, L'officina matematica – ragionare con i materiali, p. 38, Edizioni la Meridiana, 2008. In questo libro si riportano alcune lezioni fatte da Emma Castelnuovo presso la Casa-laboratorio di Cenci (Franco Lorenzoni), fra il 2002 e il 2007. La ricerca didattica di Emma Castelnuovo ha riguardato molto l'impiego di materiali semplici per lo studio attivo della matematica.

> Ho capito, insomma, che partendo da un materiale semplicissimo (sbarrette, spaghi, elastici ecc.) si potevano costruire i vari capitoli della geometria, motivando i ragazzi a partire da problemi reali. Bastava variare qualche elemento, lasciandone invariati altri, per stimolare delle problematiche anche di alta matematica. Bastava saper guardare attorno a noi perché si aprissero nuove vie del pensiero e si arrivasse, quasi da sé, a formare negli allievi uno spirito matematico.

Questo pensiero è in accordo completo con quello di Papert. L'unica differenza è costituita dal contesto nel quale i due autori vanno a ricercare l'interesse e il coinvolgimento degli allievi. Si può dire che la geometria della Tartaruga è un analogo dei materiali fisici usati da Emma Castelnuovo. Le due visioni e le pratiche che ne scaturiscono non sono affatto in opposizione bensì complementari. In questa prospettiva, con LOGO si continua e si estende il lavoro (necessariamente) iniziato con i materiali fisici mantenendo lo stesso identico approccio pedagogico.

Tutte le figure sono state prodotte con LibreLogo stesso. I codici, adeguatamente commentati, di alcune delle figure sono listati in appendice, come esempio e spunto per ulteriori sviluppi. Nel momento in cui scrivo queste righe ho completato solo il primo capitolo ma trovo utile rendere il lavoro disponibile anche per ricevere eventuali riscontri che potrebbe essere utile per il resto.

LibreLogo

LibreLogo è l'unione del celebre programma Logo e il word processor Writer, che è l'equivalente di Word. Word fa parte della ben nota *suite* Microsoft Office mentre Writer fa parte di LibreOffice, che è software libero. Logo è stato creato negli anni 70 da Seymour Papert per facilitare l'insegnamento della matematica mediante il computer. Seymour Papert è un matematico nato in Sudafrica nel 1928., ha studiato matematica a Johannesburg e poi a Cambridge. Ha fatto ricerca in una varietà di luoghi fra cui l'università di Ginevra, fra il 1958 e il 1963. È in questo periodo che ha lavorato con Jean Piaget, diventando uno dei suoi collaboratori preferiti – interessante connubio fra un matematico e un pedagogista. Nel 1963 è stato ricercatore presso il MIT (Massachusetts Institute of Technology) dove, nel 1967, è stato nominato codirettore del celebre MIT Artificial Intelligence Laboratory dal direttore fondatore, Marvin Minsky. Lo stesso laboratorio dove pochi anni dopo avrebbe operato Richard Stallman, ideatore del concetto di software libero e autore dei primi fondamentali componenti software su cui, negli anni '90, si sarebbe basato il software operativo Linux. Papert è famoso per avere inventato Logo, un linguaggio che consente di creare grafica manovrando il movimento di una “tartaruga” mediante opportuni comandi. Nella prima versione, ideata negli anni '70, la tartaruga era in realtà un robot che disegnava mentre si muoveva.

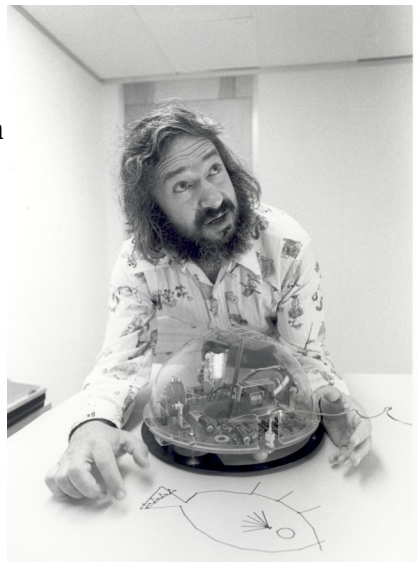


Figura 1: Seymour Papert mostra una delle prime versioni di Logo, quando era ancora un vero e proprio robot per disegnare.

Quando i computer arrivarono nelle case, negli anni '80, Logo divenne un software e come tale è stato descritto da Seymour Papert in *Mindstorm*. Per capire la valenza pedagogica del pensiero di Papert leggiamo questo brano, tratto proprio da Mindstorms (pp. 7-8):

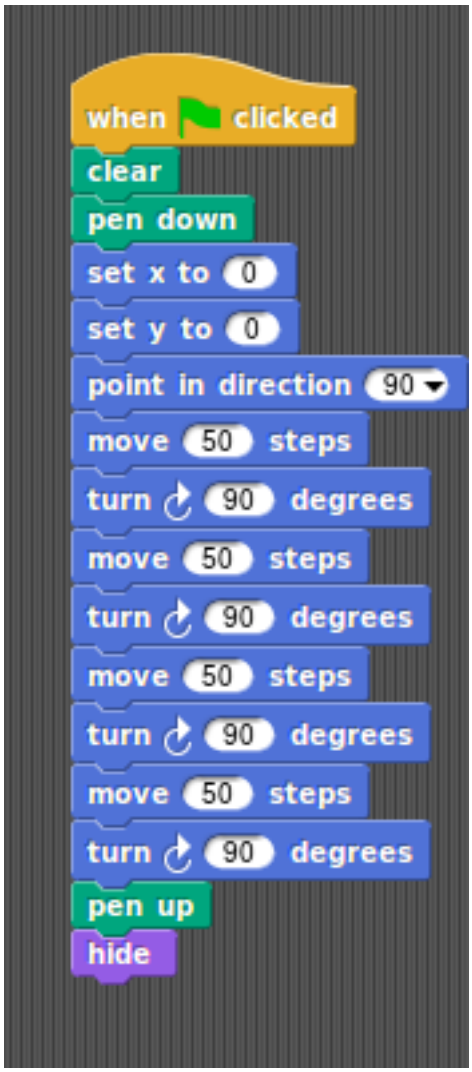
> Da Piaget prendo il modello del bambino come costruttore delle proprie strutture mentali. I bambini hanno il dono innato di imparare da soli e sono in grado di assumere un'enorme quantità di conoscenza grazie a un processo che io chiamo “apprendimento piagetiano”, o “apprendimento senza insegnamento”. Per esempio, i bambini imparano a parlare, imparano la geometria intuitiva necessaria a muoversi nel loro ambiente, e imparano abbastanza logica e retorica per cavarsela con i genitori – tutto questo senza che venga insegnato loro niente. Ci dobbiamo domandare come mai vi sono cose che si imparano così presto e spontaneamente mentre altre vengono apprese molti anni dopo o non vengono apprese affatto, se non con l'imposizione di un istruzione formale. Se prendiamo sul serio l'immagine del "bambino costuttore" allora siamo sulla buona strada per trovare una risposta a questa domanda. Tutti i costruttori hanno bisogno di qualche tipo di materiale per costruire qualcosa. Dove il mio pensiero diverge da quello di Piaget è nel ruolo che attribuisco al contesto culturale come fonte di tale materiale. In

> alcuni casi, il contesto ne fornisce in abbondanza, facilitando così l'apprendimento costruttivo Piagetiano. Per esempio il fatto che così tante cose importanti (coltelli e forchette, madre e padre, scarpe, calze) compaiano usualmente in coppia rappresenta un "materiale" per la costruzione di un senso intuitivo di numero. Ma in molti casi dove Piaget invocherebbe la complessità o la natura formale di un concetto per spiegare la lentezza del suo sviluppo, io trovo che il fattore critico sia piuttosto la carenza dei materiali che avrebbero reso il concetto semplice e concreto.

Negli anni '90 Logo circolava come un programma installabile da un floppy disk. Una volta lanciato produceva uno schermo nero sul quale si potevano scrivere delle istruzioni in sequenza, una dietro l'altra. Le istruzioni rappresentavano i movimenti da impartire alla tartaruga sulla schermo. Poi, con un comando speciale, si poteva “eseguire” la sequenza dei comandi, e così la tartaruga si muoveva tracciando un disegno sullo schermo. Logo ha avuto una grande risonanza come metodo sperimentale per l'insegnamento della matematica e ne sono state derivate una grande varietà di versioni, arrivando fino a generalizzazioni come l'attuale Scratch. Tuttavia non ha avuto una grande diffusione nelle scuole e forse si può dire che ha avuto più successo con gli scolari a cui è stato offerto che con gli insegnanti. Probabilmente era troppo presto. Usare Logo vuol dire scrivere codice, un'attività estranea alla preparazione della maggior parte degli insegnanti, anche di materie scientifiche. Oggi forse è diverso, si parla molto di *coding*, anche se forse non sempre con cognizione di causa. La situazione si è talmente evoluta che *coding* può significare tante cose diverse. Del resto, dagli anni 80 ad oggi la varietà di linguaggi di programmazione si è allargata a dismisura. La cosa più affine a Logo è Scratch, che anzi, deriva proprio da Logo. Mitchel Resnick, leader del progetto Scratch, è stato un allievo di Papert e opera sempre nel Media Laboratory del MIT. Scratch va molto oltre la produzione di grafica e consente di realizzare animazioni e videogiochi, consentendo così anche di sperimentare tecniche di programmazione piuttosto sofisticate. Un altro aspetto innovativo consiste nel fatto di essere strutturato come un servizio web e questo ha consentito di realizzare una grande comunità viva di diffusione e scambio dei programmi. Dal punto di vista operativo Scratch si differenzia da Logo per il fatto di essere un linguaggio visuale. I comandi infatti sono costituiti da blocchi colorati che possono essere incastrati fra loro. Il programma nasce dall'esecuzione di queste sequenze di comandi uniti fra loro, come in un puzzle. È un sistema attraente che si rifa un po' all'idea del Lego, dove le istruzioni da dare al computer vengono incastrate fra loro come mattoncini. Gli incastri garantiscono che le istruzioni vengano combinate solo in modi legittimi, mettendo al riparo dai tipici e frequenti errori ortografici e sintattici in cui incorre chiunque scriva un software nel modo testuale convenzionale. Ne sono emersi tanti di linguaggi di questo tipo, oltre a Scratch, i più noti sono Snap!, Alice, Blockly, Android App Inventor, giusto per menzionarne alcuni. La figura seguente illustra la differenza fra un codice di tipo testuale e uno di tipo visuale. Il codice serve a disegnare un quadrato. A sinistra la versione in LibreLogo e a destra la versione in Snap!. In Scratch questo semplice codice sarebbe identico. Ho utilizzato Snap! Per una mia certa preferenza per questo linguaggio. Snap! rappresenta un potenziamento di Scratch, che lo rendono più assimilabile ad un linguaggio di uso generico, pur mantenendo la forma visuale. Fra queste caratteristiche vi è quella di consentire il salvataggio del codice in un formato standard (XML) leggibile e alterabile con un qualsiasi editore di testo. Per chi è abituato a lavorare con il software questo è un elemento molto importante.Il codice non è, come si suole dire, “ottimale”, in nessun senso. È giusto il modo che utilizza le istruzioni più semplici, le prime che si imparano, in ambedue i linguaggi. L'esempio è pensato solo per confrontare le

istruzioni nei due diversi ambienti.

```
CLEARSCREEN
PENDOWN
HOME
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
PENUP
HIDETURTLE
```



Una caratteristica particolare di Scratch è quella di avere dato vita ad una vasta comunità di condivisione dei software. Questo è avvenuto grazie al fatto di essere stato concepito come un servizio web, che consente la composizione dei programmi e la possibilità di farli girare ma anche la realizzazione dell'aspetto *social*, destinato alla condivisione e al riuso dei programmi.

I linguaggi visuali non portano solo vantaggi. Sono (apparentemente) facili, divertenti e colorati, l'efficacia sembrerebbe garantita ma l'evidenza scientifica non è altrettanto chiara. Esistono infatti vari studi che mostrano come i linguaggi visuali non facilitino di fatto l'apprendimento dei linguaggi "veri" [6]. Sembra che siano vantaggiosi per capire i più semplici costrutti della programmazione, questo sì, ma gli studi dove si testano le reali capacità di comprensione di quello che si ottiene con un certo codice non mostrano differenze sostanziali fra linguaggi visuali e testuali[7]. Particolarmente interessante è la ricerca di Colleen Lewis dove si confrontano i risultati

6 D. Weintorp, http://dweintrop.github.io/papers/Weintrop-diss-4pager.pdf … e vari articoli ivi citati.

7 D. Weintorp e U. Wilensky, Using commutative assessments to compare conceptual understanding in blocks-based

ottenuti con Logo e con Scratch in una classe di bambini fra 10 e 12 anni[8]: se l'apprendimento di alcuni costrutti sembra facilitato da Scratch, non si sono osservate differenze nella percezione degli scolari che, anzi, hanno mostrato un livello di autostima superiore se introdotti alla programmazione con Logo. E anche se nelle fasi iniziali i giovani mostrano di gradire gli strumenti di tipo visuale, successivamente, una volta che sono entrati in contatto con la programmazione testuale convenzionale, talvolta sono loro stessi a denunciare i limiti del *coding* visuale, per 1) la minore potenza, ovvero per i limiti imposti alla propria creatività, 2) per la maggiore lentezza nella programmazione quando questa si fa più complessa e 3) perché questi sistemi sono "meno veri": "se devi fare una cosa vera nessuno ti chiederà mai di codificarla con un software didattico visuale"[9].

È sulla base di tali considerazioni che abbiamo deciso di approfondire il linguaggio Logo, quale strumento introduttivo alla programmazione. Di versioni di Logo oggi ce ne sono una quantità. Noi qui ci concentriamo su una versione che si trova normalmente nel programma di *word processing* Writer, incluso nella suite per ufficio LibreOffice[10], l'analogo del ben noto Microsoft Office. Quest'ultimo è un "prodotto proprietario", vale a dire che l'azienda che lo produce lo vende ma senza distribuire il codice sorgente in chiaro, secondo il modello industriale convenzionale, con il quale la proprietà intellettuale è tenuta gelosamente segreta. LibreOffice invece è software libero, e come tale è l'ideale per l'impiego in qualsiasi contesto formativo. In primo luogo perché comporta un messaggio di natura etica. Infatti il software libero è definito da quattro tipi di libertà: 1) libertà di eseguire il programma come si desidera, per qualsiasi scopo ; 2) libertà di studiare come funziona il programma e di modificarlo in modo da adattarlo alle proprie necessità; 3) libertà di ridistribuire copie in modo da aiutare il prossimo; 4) libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti eventualmente apportati, in modo tale che tutta la comunità ne tragga beneficio. Poiché le libertà N. 2 e 4, per potere essere esercitate, richiedono la lettura del codice sorgente del software, va da se che il software libero, per essere tale, deve necessariamente rendere disponibile il codice sorgente. Occorre osservare – su questo punto molti fanno confusione – che il software di tipo *open source* non coincide con il software libero (*free software*) perché manca la connotazione etica: con il software *open source* si assume che il codice sorgente sia disponibile in chiaro, ma non si fa menzione delle suddette quattro libertà e, in particolare, delle due specificazioni che connotano la valenza etica del *free software*: "in modo da aiutare il prossimo" nella terza libertà e "in modo tale che tutta la comunità ne tragga beneficio" nella quarta libertà. Il software libero è sviluppato da comunità che al più si aggregano in società non a fini di lucro. L'*open source* è sviluppato da attori economici privati che aderiscono al paradigma di sviluppo

---

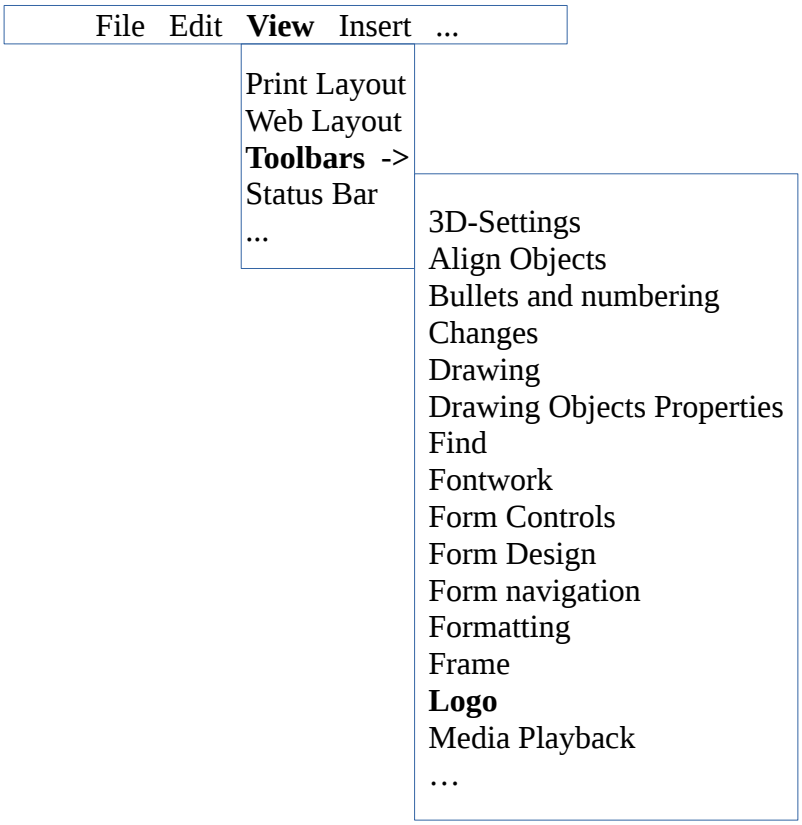
and text-based programs, Proceedings of the 11th annual International Computing Education Research (ICER) conference, ACM NY: 101-110, 2015. Reperibile in http://dweintrop.github.io/papers/Weintrop_Wilensky_ICER_2015.pdf, (luglio 2016)

8 C.M. Lewis, How *programming environment* shapes perception, learning and goals: Logo vs. Scratch, Proc. Of the 41st Annual ACM SIGCSE Conference, ACM NY: 346-350, 2015. Reperibile in http://ims.mii.lt/ims/konferenciju_medziaga/SIGCSE'10/docs/p346.pdf, (luglio 2016).

9 D. Weintorp e U. Wilensky, To Block or not to Block, That is the Question: Students' Perceptions of Blocks-based Programming. Proceedings of the 14th International Conference on Interaction Design and Children, ACM NY: 199-208, 2015. Reperibile in http://dweintrop.github.io/papers/Weintrop_Wilensky_IDC_2015.pdf (luglio 2016).

10 Esiste un altro progetto analogo che si chiama OpenOffice. La domanda su quali siano le differenze rispetto a LibreOffice è molto frequente. Una piccola storia dell'evoluzione di questi due software, che hanno un origine comune, può essere trovata qui (luglio 2016): http://www.navigaweb.net/2014/04/differenze-tra-openoffice-e-libreoffice.html. Allo stato attuale, LibreOffice conviene perché incorpora più funzionalità e viene aggiornato più frequentemente.

condiviso perché lo trovano adeguato alle proprie strategie di marketing: vi sono aziende che curano progetti *open source* a fianco dei tradizionali prodotti proprietari perché lo trovano conveniente per le proprie strategie di marketing. Le funzionalità di LibreOffice possono essere arricchite da numerosi *plugin*, ovvero componenti che aggiungono le funzionalità più diverse. Ebbene, LibreLogo è uno di questi e, dalla versione 4.0 in poi, il *plugin* LibreLogo è incluso di *default* [11] nel programma. Ma cosa significa usare Logo in un word processor come Writer, se questo è un normale *word processor* mentre Logo è un linguaggio per disegnare? Semplice: con il *plugin* LibreLogo si possono produrre immagini che risultano integrate nel documento, come se fossero importate. È un'idea geniale, dovuta a Németh László, che ha riprodotto tutte le funzionalità di Logo all'interno di LibreOffice. In realtà le ha ulteriormente incrementate, traendo vantaggio dal linguaggio Python, con cui ha scritto il plugin. Usare LibreLogo è semplicissimo: si apre un documento in Writer, si scrive un po' di codice in linguaggio Logo, come fosse un qualsiasi altro testo, e poi si esegue premendo l'apposito tasto nella *toolbar* di LibreLogo; se il codice è corretto, la tartaruga esegue il disegno codificato nel testo in mezzo alla pagina. Successivamente, questo disegno può essere gestito e manipolato come qualsiasi altra grafica di LibreOffice. Quando si lancia LibreOffice, se non si è mai usato LibreLogo, la *toolbar* di LibreLogo non è attiva. Occorre quindi attivarla, con l'appropriato comando di menu: **View** → **Toolbars** → **Logo**:



11 Di *default* significa che questo è il comportamento normale. Coloro che utilizzano Linux (per Windows o Mac questo problema non c'è) devono prendere nota di quanto segue. Fino alla versione LibreOffice 4 esclusa, installare l'estensione di LibreOffice da http://extensions.libreoffice.org/extension-center/librelogo. Invece dalla versione 4 in poi, installare direttamente il pacchettolibreoffice-librelogo, con il comando *sudo apt-get install libreoffice-librelogo*. Dopodiché occorre fare ripartire LibreOffice, qualora fosse già aperto. Quindi attivare la toolbar in *View->Toolbars->Logo*. Richiudere e rilanciare

Fatto questo, occorre chiudere il programma e rilanciarlo per vedere fra le altre *toolbar* anche quella di LibreLogo. Questa appare nel seguente modo:

dove le icone hanno i seguenti significati:

Icona	Istruzione (se applicabile)	Cosa succede
	FORWARD 10	Avanti di 10 punti (vedremo successivamente il significato dei punti)
	BACK 10	Indietro di 10 punti
	LEFT 15	A sinistra di -15°
	RIGHT 15	A destra di 15°
		Esegue il programma scritto nel programma. Dalla versione 4.3 in poi, in un documento nuovo appena aperto esegue un programma di esempio.
		Ferma il programma che sta girando (se dura troppo a lungo per qualche problema)
	HOME	Riporta Logo nella condizione iniziale, con la tartaruga al centro che punta in alto.
	CLEARSCREEN	Cancella il disegno appena fatto (non il testo presente nel documento)
		Consente di scrivere un comando qualsiasi per eseguirlo subito
		Aggiusta tutto il testo del programma rendendolo tutto maiuscolo.

La grafica di LibreLogo in Writer

L'interazione fra LibreLogo e Writer è particolare per quanto riguarda la grafica. All'inizio può sembrare farraginosa ma in realtà occorre abituarsi e imparare due o tre regolette. La caratteristica, probabilmente unica, di LibreLogo è che il risultato ottenuto girando[12] uno *script* si ritrova sulo stesso supporto dove scriviamo il codice, ovvero un documento di tipo ODT di Writer. Di fatto in questo modo il documento ospita due tipi di informazioni diverse: una lista di istruzioni scritte in forma testuale e un oggetto grafico prodotto facendo funzionare quelle istruzioni. L'oggetto grafico è di tipo “vettoriale”, ovvero è composto da un insieme di oggetti geometrici. Altro sono le immagini tipo *raster*, o *bitmap*, che sono composte da una matrice di pixel[13]. Gli oggetti grafici prodotti da LibreLogo sono del tutto analoghi a quelli che prodotti con il tool di disegno a mano disponibile in Writer, accessibile attraverso l'apposita *toolbar*, alla voce di menu **View → Toolbars → Drawing**: :

Come tali, i disegni fatti con LibreLogo possono essere spostati, copiati o salvati come qualsiasi altro oggetto grafico. Una cosa utile da capire è che spesso tali oggetti sono in realtà una composizione di oggetti distinti. In questo manuale ne faremo molti. Per utilizzarli come un unico oggetto occorre usare la funzione di raggruppamento, procedendo così: prima si delimita la regione che comprende gli oggetti da raggruppare, selezionando il *pointer*  nella barra di disegno e poi delineando la regioine rettangolare desiderata con il mouse e tenendo premuto il tasto sinistro.

Attenzione che il cursore del mouse deve avere la forma della freccia  e non quello tipico di quando si inserisce il testo, a forma di una I maiuscola, perché con questo si inserisce testo e non grafica. Il fatto che sia attivo il cursore grafico (e non testuale) si capisce anche dal fatto che contestualmente si attiva un'altra toolbar, che serve al controllo della grafica:

Quando si seleziona la regione che contiene gli oggetti grafici, in questa barra si attivano alcune icone, fra cui quella della funzione raggruppamento:  . Premendo questa tutti gli oggetti grafici compresi nella regione selezionata vengono raggruppati in un unico oggetto grafico che può essere copiato altrove o salvato.

Un altro accorgimento utile è quello di “ancorare” appropriatamente la grafica al documento,

12 In gergo con “girare un programma” si intende far funzionare un programma – in inglese *to run a program*. Oggi, con i moderni linguaggi spesso i programmi sono detti *script*. In generale un programma è un software completo e magari anche molto complesso. Uno *script* tende a essere un frammento di codice più piccolo e specifico. Ma sono categorie che si sovrappongono largamente.

13 Un approfondimento della distinzione fra immagini bitmap e vettoriali può essere trovato in http://iamarf.org...

laddove la dobbiamo usare. Sempre nella solita barra per la grafica, il tasto che consente di determinare l'ancoraggio è questa:  . Cliccando sulla freccetta a sinistra dell'ancora si possono selezionare quattro tipi di ancoraggio: 1) “alla pagina”, 2) “al paragrafo”, 3) “al carattere” e 4) “come carattere”. Nel primo caso la grafica è associata alla pagina e non si muove da questa, nel secondo ad un paragrafo, nel terzo ad un carattere e nel quarto si comporta come se fosse un carattere. Quale sia l'ancoraggio più opportuno è una cosa che si impara con l'esperienza. La maggior parte delle grafiche in questo manuale sono state ancorate “al paragrafo”, eccetto che per le piccole immagini che stanno in linea con il testo, come l'ancora precedente, queste sono ancorate “come carattere”.

Queste cose appena dette riguardano la gestione della grafica in Writer in generale. Usando LibreLogo, l'unica differenza è che la grafica viene prodotta attraverso le istruzioni che mettiamo nel codice. LibreLogo piazza la grafica nel mezzo della prima pagina del documento, anche se il testo del codice si dilunga nelle pagine successive. Può succedere così che la grafica si sovrapponga al testo del codice medesimo. Di primo acchito sembra che il comportamento sia farraginoso se non errato. Niente di tutto questo. La grafica è prodotta per essere usata da qualche parte. Si tratta semplicemente di selezionarla, con gli accorgimenti appena descritti e portata altrove, in una pagina pulita semplicemente per vederla con chiarezza, oppure in qualche altro documento dove questa debba essere integrata.

Cap. 1: Seymour Papert introduce il problema della matematica

Prologo

Mi permetto di offrire una traduzione del capitolo *Mathophobia: The Fear for Learning*, da *Mindstorms* di Seymour Papert[14]. L'intento è quello di chiarire bene che la motivazione fondamentale della genesi di Logo è la questione, annosa e tutt'ora irrisolta, dell'insegnamento della matematica. È stata un'operazione per certi versi penosa, non tanto per i miei evidenti limiti in un lavoro di traduzione, quanto per l'intenso senso di frustrazione montato percorrendo lentamente e con attenzione questo scritto. Le sensazioni sono che non molto sia cambiato, dagli anni '80 ad ora, almeno in media[15]; che la motivazione iniziale, centrata su una seria e difficile rivisitazione del modo di introdurre i giovani alla matematica, sia finita diluita oggi nel calderone del “coding”, nella forma di una sorta di paese dei balocchi, superficialmente entusiasmante per taluni, oggetto di derisione per altri; che il messaggio di Papert, per certi versi estremo e provocatorio, senz'altro da decodificare rispetto ad un'epoca diversa, venga frainteso; che il tutto sia vanificato in sostanza dal fallimento di Logo, rimasto confinato in una minoranza di circoli sperimentali, senza avere rivoluzionato nulla, contrariamente a quelle che sembravano le legittime aspettative di Papert; che invocare la magia della matematica per introdurre i giovani in un dominio comunemente considerato “freddo”, sia un sogno che alla fin fine può concepire solo un matematico, magari un po' idealista, e quindi che solo un'arida via può svelare quella magia, e solo ai pochi in grado di

14 S. Papert, *Mindstorms – Children, Computers and Powerful Ideas*, Basic Books, 1993 (I edizione 1980).

15 Affermazione che nasconde un mondo di perplessità. Cosa è cambiato? Forse proprio ciò che una “media” non esprime. La scuola cui si riferiva Papert è probabilmente più affine a quella che ha frequentato il sottoscritto (I elementare nel 1960). Allora probabilmente il panorama era più uniforme. “La lo picchi se non capisce perché gliè zuccone!” raccomandò la mamma di un mio compagno di classe alla maestra. I genitori erano alleati di quel sistema scolastico, in una visione formativa che poteva essere coercitiva e punitiva, ma che attraversava tutti i generi di scuole e tutti gli strati sociali. Non c'erano “genitori coach” o “genitori sindacalisti”. Nelle famiglie si lavorava duramente, nelle scuole si faticava. Non c'era ancora il “tempo libero”. La scuola era più brutale, forse iniqua, la pedagogia semplice, ma il panorama era più nitido. Almeno nella provincia rurale degli anni '60 in cui ho vissuto. Ora domina la complessità. Le categorie si intersecano. I dibattiti esplodono, amplificati dai media, a livello microscopico (gruppi di genitori in Wathsapp o Facebook) e a livello macroscopico (stampa, televisione ecc.). Le esperienze personali sono schizofreniche: i miei contatti con il mondo dell'insegnamento rappresentano un quadro affascinante di impegno, studio e sperimentazione; ma le storie private e le narrazioni dei conoscenti sono popolate di pratiche didattiche obsolete e superficiali. La variabilità è allucinante. Dove sta la media? Francamente non sono in grado di valutarlo ma la dispersione è sicuramente molto più ampia di un tempo. A complicare il quadro ci sono le indagini internazionali, paludate di rigore scientifico ma che poi si possono rivelare speratamente fatue. Per alcuni anni è brillata la stella polare della Finlandia nel cielo delle valutazioni PISA dell'OCSE, in particolare per la matematica. Poi emergono una serie di denunce di accademici finlandesi che documentano un crollo delle competenze matematiche: sembra che gli studenti finlandesi siano diventati bravi nei test matematici PISA ma che siano peggiorati in matematica! Leggendo il post di Giorgio Israel “Il bluff della matematica finlandese” (http://gisrael.blogspot.it/2011/05/il-bluff-della-matematica-finlandese.html), che riassume tali denunce, si scopre che i modelli di apprendimento sono banalmente utilitaristici e anche ben lontani dalle idee di Papert che riportiamo qui. Dove sarà la verità? Insomma la confusione regna sovrana e viene seriamente da domandarsi se non ci si debba rassegnare a considerarla inevitabile normalità.

percorrerla, per un motivo o per un altro, e che non possa essere infine altro che così – una cosa che io non voglio pensare ma la paura che sia un po' vera m'è venuta rileggendo *Mindstorms*.

Ci sono passaggi che sicuramente alcuni lettori non condivideranno, in particolare sull'inutilità di tanti esercizi ripetitivi, di troppo calcolo mnemonico ecc. Probabilmente è opportuno trovare un equilibrio fra le posizioni, di fatto tutte indimostrabili. Voglio tuttavia citare due fra i tanti episodi di cui ho ricordanza e che mi inducono a collocare il mio pensiero vicino a quello di Papert, tenuto debito conto dell'epoca e del contesto diversi. Il primo riguarda l'identificazione della matematica col far di conto. Mi trovavo, una ventina di anni fa, in una riunione composta quasi esclusivamente da matematici per un progetto di ricerca nazionale, alcuni dei quali personaggi eminenti a livello internazionale. Ad un certo punto fu necessario fare al volo un calcolo molto trito: considerato l'ammontare di finanziamenti disponibile, e verificata la nostra numerosità, quanto veniva a testa? Ebbene, ci fu un momento di panico, la risposta non scaturì pronta come qualsiasi "laico" avrebbe potuto supporre, anzi, fu presa una calcolatrice per risolvere il "problema". Solo un episodio, ma a chiunque sia occorso di raggiungere una conoscenza abbastanza profonda di un qualche campo della matematica, è perfettamente chiaro che il pensiero matematico non ha quasi niente a che vedere con la capacità di fare operazioni aritmetiche a memoria, giusto per menzionare un aspetto della "matematica scolastica". E l'intelligenza che occorre per comprendere il senso profondo di un pensiero matematico può, in taluni casi, essere addirittura in contrasto con quella che serve a fare calcoli aritmetici. Il secondo episodio proviene dal racconto di uno studente (geniale) al primo anno di matematica, primo giorno di lezione dell'insegnamento forse principale, analisi matematica I, tipicamente tenuto da un professore di riferimento. La prima cosa che il professore disse agli studenti fu: "Più o meno approfonditamente, chi viene dal classico, chi dallo scientifico, chi dall'istituto tecnico, avete fatto tutti una certa quantità di matematica. Ebbene, ora dimenticate tutto, la matematica è un'altra cosa." Ed è perfettamente vero. Non sto ad annoiare il lettore con una quantità di ricordi autobiografici in sintonia con questi episodi, ma la dissonanza fra la sensazione appagante e anche estetica che vive chi improvvisamente "vede" un'idea matematica, e l'aridità della stragrande maggioranza della matematica scolastica, fa venire il mal di testa. E mi induce a rifarmi da Papert, e da Logo.

Logo ha fallito dicevamo. Più correttamente, ha fallito negli intendimenti di Papert (che io continuo a condividere), ma non nel senso di non avere lasciato traccia, tutt'altro. Ci sono a giro per il mondo innumerevoli versioni di Logo, alcune delle quali sono divenute anche importanti strumenti di indagine didattica, per esempio nel campo della simulazione dei sistemi biologici complessi. Ed è sempre da Logo che ha preso le mosse il mondo tentacolare dei linguaggi visivi a blocchi, in primo luogo Scratch, questo sì un successo. Lungi da me intavolare qualsiasi sterile diatriba sul confronto fra i due linguaggi. In un certo senso Scratch "contiene" Logo – è anche stato sviluppato dagli allievi di Papert – ma Scratch è molto più ricco di Logo, estremamente più sofisticato dal punto di vista informatico, decisamente cittadino del Web. Quello che si può fare in Logo si può fare anche in Scratch, a parte alcuni aspetti di organizzazione del sistema di cui parleremo in seguito. Il problema però è che tutta questa abbondanza, rovesciata su un mondo il quale, malgrado tutte le possibili buone intenzioni, si ritrova smaccatamente impreparato, ha finito col disperdere i proponimenti didattici che in Logo sono più nitidamente visibili e, infine, più facilmente perseguibili. Torneremo su queste riflessioni, ma alla fine del manuale, non prima di aver lasciato emergere una serie di fatti importanti. Segue quindi la traduzione del capitolo dove Papert affronta

proprio il nodo dell'avvio degli studenti alla matematica, chiedendo venia al lettore per la traduzione del sottoscritto, spero non troppo incerta, e per coloro che forse si lasciano prendere un po' troppo la mano da aneliti idealistici. Ma senza utopie si vive male.

Mathophobia: The Fear for Learning

Platone sulla sua porta aveva scritto: “Che entrino solo i geometri”. I tempi sono cambiati. La maggior parte di coloro che cercano di entrare nel mondo intellettuale di Platone non conoscono la matematica né percepiscono la minima contraddizione con la sua prescrizione. La schizofrenica suddivisione che la nostra cultura traccia fra le discipline umanistiche e quelle scientifiche supporta il loro senso di sicurezza. Platone era un filosofo, e la filosofia è una materia umanistica tanto sicuramente quanto la matematica una scientifica.

Questa grande divisione è radicata nel nostro linguaggio, nella nostra visione del mondo, nell'organizzazione sociale, nel sistema educativo e, più recentemente, anche nelle teorie neurofisiologiche. È una divisione che si auto-perpetua: più la cultura è divisa, più ciascuna parte rinforza la separazione nella propria crescita.

Ho già suggerito come il computer possa costituire una forza che serva ad abbattere la divisione fra le “due culture”. So che l'umanista può ritenere discutibile che una tecnologia possa influenzare la propria opinione su quale tipo di conoscenza sia rilevante nell'insegnamento. Non meno minacciosa appare allo scienziato la diluizione del rigore causata dall'invasione di pensiero umanistico “annacquato”. Ciò nonostante io penso che con la tecnologia si possano gettare i semi di un'epistemologia culturale meno dissociata.

La condizione della matematica nella cultura contemporanea presenta i sintomi più acuti di tale dissociazione. L'emergenza di una matematica “umanistica”, che non sia percepita in maniera separata dallo studio dell'uomo e delle discipline umanistiche, potrebbe essere il segno di un mutamento di prospettiva. In questo libro io cercherò di mostrare come un computer possa essere utilizzato per condurre i bambini in una relazione più umanistica e anche più umana con la matematica. Per fare questo dovrò andare oltre la matematica. Dovrò sviluppare una nuova prospettiva del processo di apprendimento medesimo.

Non è raro che adulti intelligenti si riducano ad essere osservatori passivi della propria incompetenza in tutto ciò che non sia la matematica più rudimentale. E possono subire le conseguenze di una simile paralisi intellettuale anche nella ricerca di un lavoro. Ma le conseguenze secondarie, indirette, sono ancora più gravi. Una delle lezioni principali imparate dalla maggior parte delle persone nelle ore di matematica è una consapevolezza delle proprie rigide limitazioni. Costoro si formano un'idea balcanizzata della conoscenza umana che finiscono col percepire come un collage di territori separati da ferree cortine impenetrabili. Io non metto in discussione la sovranità dei territori intellettuali ma le restrizioni imposte alla libera circolazione fra questi. Non voglio ridurre la matematica alla letteratura o la letteratura alla matematica. Ma voglio argomentare come le rispettive mentalità non siano così separate come viene generalmente supposto. E per fare questo, mi servo di un'immagine, ovvero di una *Mathland* – dove la matematica sia un vocabolario

naturale – al fine di sviluppare l'idea che con la presenza del computer le culture umanistica e matematico/scientifico possano essere riunite. In questo libro, *Mathland* rappresenta il primo passo di un discorso più ampio su come la tecnologia possa cambiare non solo il modo con cui insegniamo la matematica ai bambini, ma anche, in maniera più fondamentale, il modo nel quale la nostra cultura nel suo complesso concepisce la conoscenza e l'apprendimento.

Per me la parola "*mathophobia*" presenta due associazioni. Una di queste è il diffuso timore per la matematica, che spesso presenta i connotati di una vera fobia. L'altra attiene al significato della radice "*math*", che in greco significa "apprendimento", nel suo senso più generale[16]. Nella nostra cultura, la paura di imparare non è meno endemica (sebbene molto spesso travestita) della paura della matematica. I bambini all'inizio della propria vita sono avidi di apprendere. Poi sono costretti a *imparare* ad avere problemi con l'apprendimento in generale con la matematica in particolare. In ambedue i sensi della radice "*math*" si verifica uno spostamento da matefilia a matofobia. Andremo a vedere le cause di tale spostamento e vedremo qualche idea su come si possa usare il computer per contrastarlo. Iniziamo con qualche riflessione su come apprendano i bambini.

La facilità di apprendimento dei bambini sembra così ovvia che ai più sembra non valga nemmeno la pena di documentarla. Un campo nel quale la capacità di apprendimento è particolarmente chiara è quello dell'apprendimento verbale di nuovi vocaboli. All'età di due anni sono pochi i bambini che conoscono più di qualche centinaio di parole. Ma già quando entrano nella prima classe primaria, quattro anni dopo, conoscono migliaia di parole. È evidente che sono in grado apprendere ogni giorno varie parole nuove.

Anche se "vediamo" che i bambini imparano le parole, non è altrettanto facile vedere che stanno imparando matematica con la stessa velocità, o anche maggiore. Ma questo è esattamente ciò che ha mostrato Piaget, con lo studio di una vita intorno alla genesi della conoscenza nei bambini. Una delle conseguenze più sottili delle sue scoperte è la rivelazione che gli adulti non riescono ad apprezzare la natura e l'estensione di ciò che i bambini apprendono, perché il fatto che diamo per scontate varie strutture della conoscenza nasconde una buona parte di quell'apprendimento. Questo è evidente in quelle che sono note come le "conservazioni" piagetiane.

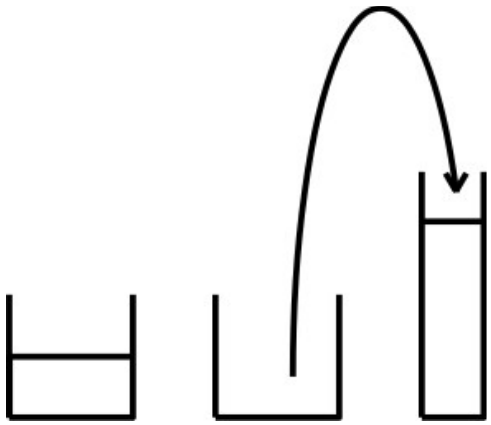


Figura 1: Riprodotta dal testo orginale usando LibreLogo

16 Il significato generale è presente nella parola "*polymath*", che denota una persona dai saperi multipli. Una parola meno nota con la stessa radice, che userò nei capitoli successivi, è "matetico": che concerne l'apprendimento.

Per un adulto è ovvio che versando liquido da un bicchiere ad un altro il volume del liquido non cambia (a meno di piccoli effetti, come gocce versate fuori o lasciate nel bicchiere precedente). La conservazione del volume è così ovvia che sembra non sia venuto in mente a nessuno prima di Piaget che ai bambini di quattro anni potrebbe apparire diversamente. Occorre una sostanziale crescita intellettuale prima che i bambini sviluppino una visione "conservazionista" del mondo. La conservazione del volume è solo una delle tante conservazioni che devono imparare. Un'altra è la conservazione dei numeri. Anche in questo caso, gli adulti faticano a rendersi conto che un bambino deve imparare il fatto che contando una collezione di oggetti in ordine diverso il risultato sia lo stesso. Per gli adulti l'operazione di contare significa semplicemente determinare quanti oggetti "ci sono". Il risultato dell'operazione è un "fatto oggettivo" indipendente dall'atto di contare. Ma la separazione del numero dal conteggio (del prodotto dal processo) poggia su presupposti epistemologici che sono non solo ignoti al bambino preconservazionista, ma anche estranei alla loro visione del mondo. Tali conservazioni rappresentano solo una parte della vasta struttura di conoscenza matematica nascosta che i bambini imparano da soli. Nella geometria intuitiva del bambino di quattro o cinque anni, la linea retta non è necessariamente la distanza più breve fra due punti, e che camminare lentamente fra due punti non richiede necessariamente più tempo che camminare velocemente. Anche in questo caso, non è che manchi semplicemente un elemento di conoscenza, bensì il presupposto che sostiene l'idea "del più breve" quale proprietà di un percorso piuttosto che dell'azione di percorrerlo.

Nessuno di questi casi deve essere interpretato come una carenza di conoscenza da parte del bambino. Piaget ha mostrato come i bambini piccoli abbiano teorie del mondo che, nei propri termini, sono perfettamente coerenti. Queste teorie sviluppate spontaneamente da tutti i bambini, hanno componenti ben sviluppati che non sono meno "matematici", sebbene esprimano una matematica diversa, rispetto a quella accettata dalla nostra cultura (adulta). Il processo di apprendimento nascosto ha due fasi: già in età prescolare ogni bambino sviluppa teorizzazioni proprie del mondo per poi spostarsi verso visioni caratteristiche dell'età adulta. E questo accade attraverso quello che ho chiamato apprendimento piagetiano, un processo con caratteristiche che la scuola dovrebbe invidiare: funziona (accade in tutti i bambini), è economico (non richiede maestri ne *curricula*), e è "umano" (i bambini lo attuano con spirito apparentemente disinteressato senza bisogno di riconoscimenti o punizioni esplicite imposti dall'esterno).

La misura in cui nella nostra società gli adulti hanno perso l'atteggiamento positivo dei bambini nei confronti dell'apprendimento varia da individuo a individuo. Una quota sconosciuta ma certamente significativa della popolazione ha completamente rinunciato a imparare. Queste persone raramente, se non mai, si cimentano nell'apprendimento e non si sentono né competenti né capaci di trarne giovamento. Il costo personale e sociale è enorme: la matofobia può limitare la vita delle persone, culturalmente e materialmente. Un numero ancora maggiore di persone non ha rinunciato completamente ma soffre di pesanti limitazioni a causa di pregiudizi negativi profondamente radicati sulle proprie capacità. La deficienza diventa identità: "Io non posso imparare il francese, non ho orecchio per le lingue;" "Non potrei mai fare affari, non ho la testa per i numeri." "Non posso imparare lo sci parallelo, non sono mai stato coordinato." Queste credenze sono spesso manifestate ripetutamente, in modo rituale, come superstizioni. E, come le superstizioni, creano un

mondo di tabù; in questo caso il tabù dell'apprendimento. In questo capitolo e nel capitolo 3, discuteremo su degli esperimenti che dimostrano come queste immagini di se spesso corrispondano a una realtà molto limitata – usualmente la "realtà scolastica" di una persona. In un contesto formativo, con un adeguato supporto emozionale e intellettuale, lo "scoordinato" può imparare esercizi circensi come la giocoleria e coloro che "non hanno la testa per i numeri" possono scoprire che non solo sono in grado di capire la matematica ma vi si possono anche appassionare.

Sebbene tali opinioni negative su di se possano essere superate, di fatto sono estremamente tenaci e tendono ad autoconfermarsi. Se uno crede abbastanza fermamente di non poter fare matematica, avrà quasi sicuramente successo nell'impedirsi di fare qualsiasi cosa che gli paia attinente alla matematica. La conseguenza di tale autosabotaggio è il fallimento personale, e ogni fallimento rinforza l'assunto di base. Ancora più insidiosi sono i pregiudizi che appartengono non solo agli individui ma a un'intera cultura.

I nostri figli crescono in una cultura permeata dall'idea che ci siano "persone brillanti" e "persone stupide". La costruzione sociale di un individuo è costituita da un fascio di attitudini. Ci sono persone "buone per la matematica" e persone "negate per la matematica". Tutto è aggiustato in maniera da attribuire i primi insuccessi o esperienze negative dei bambini a loro proprie disabilità. Di conseguenza, i bambini interpretano i fallimenti come sentenze di appartenenza al gruppo delle "persone stupide" o, più spesso, al gruppo delle persone "inadatte per l'attività *x*" (dove, come abbiamo osservato, spesso *x* si identifica con la matematica). In un contesto del genere i bambini declinano la propria personalità nei termini delle loro limitazioni, che verranno confermate e consolidate nel corso degli anni. Solo raramente accade che un evento eccezionale induca qualcuno a riorganizzare la propria immagine intellettuale in modo da aprire nuove prospettive su ciò che può apprendere.

Non è facile rimuovere questi pregiudizi sulla natura delle capacità umane. I pregiudizi popolari sono sempre difficili da sradicare. Ma qui le difficoltà sono sostenute da vari altri fattori. In primo luogo, le teorie comuni sulle attitudini umane sembrano essere sostenute da teorie "scientifiche". Dopotutto, la psicologia si avvale molto di misure attitudinali. Proviamo a mettere in discussione la significatività di ciò che viene misurato mediante l'esperimento mentale di immaginare una *Mathland*.

Sebbene l'esperimento mentale lasci aperta la questione di come realizzarla una *Mathland*, questo è tuttavia completamente rigoroso nel dimostrare che i pregiudizi comuni sulle capacità matematiche non sono sostenuti da evidenze palesi. Ma siccome i lettori più matofobici potrebbero avere problemi a fare l'esperimento per conto loro, lo riformulo in un altro modo. Immaginiamo di far disegnare ai bambini per un'ora al giorno passi di danza sulla carta e di far sostenere loro un esame su tali "questioni di danza" prima di lasciarli ballare veramente. Non dovremmo in tal caso aspettarci un mondo pieno di "danzofobi"? E non concluderemmo che coloro che ce la fanno a raggiungere la sala da ballo sono i più dotati per la danza? Io credo che sia altrettanto ingiustificato trarre conclusioni sulle doti matematiche in base allo scarso entusiasmo dei bambini per passare centinaia di ore a fare somme.

Uno può sperare che passando dalle storie ai metodi più rigorosi della psicologia potremmo ottenere dati più consistenti sulle potenzialità degli individui in termini di competenze raggiungibili. Ma non è così: il paradigma corrente nella psicologia della formazione è focalizzato su come i bambini

imparano o (più frequentemente) non imparano la matematica nella "anti-*Mathland*" in cui viviamo. Un indirizzo che può essere descritto da questa storia:

> Immaginiamo una persona del diciannovesimo secolo che volesse migliorare i sistemi di trasporto. Essa sarebbe stata probabilmente persuasa del fatto che la strada per escogitare nuovi metodi passi dalla conoscenza profonda dei metodi esistenti. Sarebbe così partita con uno studio accurato delle differenze fra i vari tipi di carri trainati da cavalli. Avrebbe quindi documentato accuratamente la dipendenza delle velocità ottenibili in funzione della forma e dei materiali degli assi, dei perni e delle finiture.

Retrospettivamente sappiamo che la strada dell'evoluzione dei mezzi di trasporto è stata completamente diversa. Le invenzioni dell'automobile e dell'aeroplano non hanno preso le mosse dallo studio dettagliato su come i mezzi preesistenti, ovvero i carri trainati da animali, funzionassero o meno. Ecco, questo è il modello della ricerca attuale sulle questioni di formazione. I paradigmi usuali per tale tipo di ricerca pongono al centro degli studi la classe scolastica. Ci sono molti studi sullo scarso valore dell'insegnamento che viene impartito dalla scuola nella matematica e nelle scienze. È tuttavia diffusa l'idea che un "buon" approccio pedagogico debba basarsi su questi metodi, in realtà poveri di pensiero. Si può simpatizzare con le buone intenzioni, tuttavia penso che tali strategie riflettano il desiderio di mantenere il sistema tradizionale. Come dire di ritenere che convenga migliorare gli assi dei carri a trazione animale. Invece la questione importante sarebbe quella di inventare l'"automobile della formazione". Questo problema (tema centrale di questo libro) non viene di fatto affrontato e, di conseguenza, ci sembra che le basi scientifiche che sostengono le assunzioni comuni sulle attitudini siano piuttosto labili. Assunzioni che tuttavia sono istituzionalizzate nelle scuole, nei sistemi di valutazione e di ammissione nelle università, al punto che la loro radicazione sociale è tanto forte quanto deboli sono i presupposti scientifici.

Dalla scuola dell'infanzia in poi, i bambini sono sottoposti a prove basate su capacità verbali e quantitative concepite come entità "vere" e separabili. I risultati di tali test si trasformano in un corredo di attitudini che determinano la costruzione sociale del bambino. Una volta che Johnny e il suo maestro condividono la percezione che Johnny è una persona dotata per l'arte ma non per la matematica, tale percezione tende inevitabilmente a rinforzarsi con il tempo. Questo è un fatto largamente accettato nella psicologia della formazione. Ma il modo in cui la scuola forma le attitudini presenta aspetti più profondi. Consideriamo il caso di un bambino che ho seguito durante i suoi ottavo e nono anni di età. Jim era un bambino molto loquace ma matofobico appartenente ad una famiglia di professionisti. La sua passione per le parole e il piacere di parlare si erano manifestate molto prima di andare a scuola. La matofobia era invece comparsa a scuola. La mia teoria è che essa sia stata una diretta conseguenza della sua precocità verbale. Dai genitori avevo appreso che Jim aveva presto sviluppato l'abitudine di commentare a voce alta qualsiasi cosa facesse. Un'abitudine che non aveva causato particolari problemi con i genitori o presso la scuola dell'infanzia. I problemi sono sorti affrontando l'aritmetica. A quel punto aveva già imparato a tenere sotto controllo la sua abitudine ma io sospetto che lui non avesse cessato di commentare le proprie azioni, seppur interiormente. Durante le ore di matematica si trovava in imbarazzo: semplicemente non riusciva a commentare l'attività di fare somme. Gli mancava il vocabolario (come manca alla maggior parte di noi) e non vedeva la motivazione. Questa frustrazione si tramutò in odio per la matematica, la conseguenza del quale fu una valutazione di scarsa attitudine per la materia.

Per me fu una storia commovente. Credo che molto spesso quella che appare una debolezza intellettuale sia espressione, come nel caso di Jim, di quella che in realtà è una particolare capacità. È non è solo la capacità verbale, chiunque osservi con attenzione i bambini nota processi simili: per esempio un bambino che prediliga l'ordine logico può avere problemi con la sillabazione dell'inglese e magari finire col detestare la scrittura. L'idea di *Mathland* ci suggerisce come il computer potrebbe servire ad evitare i problemi riscontrati da Jim e il suo pari dislessico. Ambedue i bambini sono vittime della netta separazione fra cultura verbale e matematica. Nella *Mathland* che descriviamo in questo capitolo, la passione e la competenza verbale di Jim potrebbero essere mobilizzate per favorire lo sviluppo formale matematico, invece di ostacolarlo, e la passione dell'altro bambino per la logica potrebbe essere sfruttata per sviluppare le sue competenze linguistiche.

Il concetto di mobilizzare tutte le capacità di un bambino per qualsiasi dominio di attività intellettuale risponde all'idea che attitudini differenti possano riflettere differenze nello sviluppo del cervello. È diventato comune ragionare come se ci fossero diversi cervelli, o diversi "organi" nel cervello. Per la matematica e per la lingua. In accordo con questo pensiero i bambini si dividono fra dotati per attività verbali o matematiche a seconda di quale loro organo cerebrale sia più forte. Ma una simile visione anatomica delle funzioni cerebrali comporta delle assunzioni epistemologiche. Per esempio si assume che si possa accedere alla matematica tramite una sola via e che se questa è "bloccata anatomicamente" allora il bambino non vi potrà accedere. Ora, di fatto, per la maggior parte dei bambini delle società contemporanee la via verso la matematica "avanzata" è una sola e questa è la via della matematica scolastica. Ma anche se ulteriori ricerche nella biologia del cervello arrivassero a dimostrare che tale via dipenda da un organo cerebrale mancante in alcuni bambini, ciò non significa che la matematica stessa dipenda da organi del genere. Piuttosto, significherebbe che dovremmo cercare altre strade. La tesi sostenuta in questo libro è che esistano altre vie, e che la dipendenza delle funzioni dal cervello sia essa stessa un costrutto sociale

Supponiamo che esista una parte del cervello specializzata nelle manipolazioni mentali dei numeri che insegniamo scuola, e chiamiamola MAD, "*Math Acquisition Device*"[17]. In tal caso nel corso dell'evoluzione umana si sarebbero sviluppati metodi per fare e insegnare l'aritmetica in grado di trarre massimo vantaggio dalle proprietà del MAD. Ma se questi metodi funzionassero solo per una parte di noi, e per la società nel suo insieme, si rivelerebbero invece catastrofici per un individuo il cui MAD fosse danneggiato o inaccessibile per qualche altro motivo (magari di origine "neurotica"). Una tale persona fallirebbe a scuola e le verrebbe diagnostica una "discalculia". E finché noi insistiamo con l'insegnare l'aritmetica ai bambini nel modo convenzionale, continueremo a "dimostrare" tramite i test obiettivi che questi bambini non possono "fare aritmetica". Ma questo è come dimostrare che un bambino sordo non possa disporre di un linguaggio perché non sente. Così come la lingua dei segni impiega le mani e gli occhi per aggirare gli organi della parola, così si potrebbero individuare modi alternativi di fare matematica per aggirare il MAD, forse altrettanto validi anche se diversi.

17 [NdR] Qui Papert gioca con l'idea del linguista Noam Chomsky, secondo la quale il nostro cervello disporrebbe di una sorta di dispositivo di acquisizione del linguaggio (LAD: *Language Acquisition Device*). Papert precisa di non credere a tale ipotesi, ritenendo un ipotetico MAD altrettanto improbabile di un LAD. Secondo l'ipotesi di Chomsky il cervello sarebbe composto da un insieme di organi neurologici specializzati per specifiche funzioni intellettuali. Secondo Papert tale ipotesi è troppo grossolana e se, probabilmente, si può ritenere che nel cervello vi siano dei dispositivi specializzati, è semplicistico immaginare che ve ne siano di così complessi da assolvere a funzioni quali il pensiero matematico e verbale.

Ma non c'è bisogno di invocare la neurologia per spiegare come mai alcuni bambini non acquistano confidenza con la matematica. L'analogia con le lezioni di danza senza musica e senza sala da ballo è seria. La nostra cultura della formazione offre poche possibilità agli allievi di matematica per capire veramente ciò che studiano. Di conseguenza i nostri bambini sono forzati a seguire un modello di studio della matematica che è veramente il peggiore. È il modello dell'apprendimento mnemonico, dove i contenuti sono trattati come fossero privi di significato; è un modello "dissociato". Alcune delle nostre difficoltà nell'insegnamento di una matematica culturalmente più integrabile sono dovute a un problema oggettivo: prima che esistessero i computer c'erano veramente pochi punti di contatto fra i fatti più importanti e coinvolgenti della matematica e l'esperienza quotidiana. Ma il computer – un'entità capace di parlare la matematica presente in modo ubiquitario nella vita di tutti i giorni a casa, nella scuola e al lavoro – può provvedere a tale collegamento. La sfida della formazione è quella di trovare i modi per sfruttare queste tecnologie.

La matematica non è certamente l'unico esempio di apprendimento dissociato. Ma è un ottimo esempio precisamente per il fatto che molti lettori preferirebbero che ora parlassimo d'altro. La nostra cultura è talmente matofobica che se fosse possibile dimostrare come il computer potrebbe migliorare la nostra relazione con la matematica, avrei fondati motivi per sostenere che si potrebbero migliorare allo stesso modo le relazioni con altri tipi di apprendimento. Le esperienze in *Mathland*, come quella di sostenere una "conversazione matematica", fanno vivere all'individuo un senso liberatorio delle possibilità di fare una serie di cose che prima sembravano "troppo difficili". In questo senso il contatto con il computer può aprire l'accesso alla conoscenza, non tanto in senso strumentale per disporre di informazioni processate, ma per porre in discussione alcune assunzioni vincolanti che le persone fanno su di se. La *Mathland* del computer che propongo estende l'apprendimento naturale di tipo piagetiano dell'apprendimento della lingua madre all'apprendimento della matematica. L'apprendimento piagetiano è profondamente radicato in altri tipi di attività. Per esempio, i bambini piccoli non hanno momenti dedicati a "apprendere la lingua". Questo è un modello che si pone in contrapposizione all'apprendimento dissociato, che ha luogo in maniera relativamente separata da altre attività, mentali e fisiche. Nella nostra cultura, l'insegnamento della matematica a scuola è paradigmatico dell'apprendimento dissociato . Per la maggior parte della gente la matematica è insegnata e recepita come una medicina. La dissociazione matematica della nostra cultura è quasi una caricatura delle peggiori forme di alienazione epistemologica. Nell'ambiente LOGO si ammorbidiscono le distinzioni: nessuna attività in particolare è connotata a parte come "apprendere la matematica". Il problema di rendere la matematica comprensibile concerne il problema più generale di rendere comprensibile un linguaggio basato su "descrizioni formali". Così, prima di passare a esempi di come con il computer si possa provare a dare senso alla matematica, consideriamo alcuni esempi per rendere comprensibili linguaggi basati su descrizioni formali in domini della conoscenza che la gente non associa usualmente alla matematica. Nel primo esempio il dominio è quello della grammatica, per molti temibile quasi quanto la matematica.

Nel corso di uno studio di un anno, in una classe II di scuola media di I grado di livello medio, una delle attività era quella che gli studenti chiamavano "*computer poetry*". L'attività consisteva nell'usare il computer per comporre frasi: loro inserivano una struttura sintattica che il computer popolava di parole in maniera casuale. Il risultato è una sorta di poesia concreta tipo quella illustrata

qui sotto[18]:

INSANE RETARD MAKES BECAUSE SWEET SNOOPY SCREAMS
SEXY GIRL LOVES THATS WHY THE SEXY LADY HATES
UGLY MAN LOVES BECAUSE UGLY DOG HATES
MAD WOLF HATES BECAUSE INSANE WOLF SKIPS
SEXY RETARD SCREAMS THATS WHY THE SEXY RETARD
THIN SNOOPY RUNS BECAUSE FAT WOLF HOPS
SWEET FOGINY SKIPS A FAT LADY RUNS

Un'allieva di tredici anni, Jenny, aveva commosso lo staff del progetto chiedendo il primo giorno: “Perché avete scelto noi? Noi non siamo i cervelloni. (“Why were we chosen for this? We're not the brains.”. Lo studio prevedeva proprio di lavorare con una classe di livello “medio”. Un giorno Jenny entrò tutta eccitata. Aveva fatto una scoperta: “Ora ho capito perché ci sono i sostantivi e i verbi.” Già da vari anni Jenny aveva fatto esercizi grammaticali, ma non aveva mai capito le differenze fra sostantivi, verbi e avverbi. Ma ora era chiaro che le sue difficoltà non dipendevano dall'incapacità di lavorare con categorie logiche. Il problema era un altro. Lei non aveva semplicemente compreso la finalità della fatica. Non era stata in grado di afferrare il senso della grammatica perché non vedeva a cosa servisse. E quando aveva chiesto a cosa serviva, la spiegazione dell'insegnante le era parsa manifestamente disonesta: “La grammatica ti serve a parlare meglio.”

Infatti, per recuperare la connessione fra l'apprendimento della grammatica e il miglioramento della lingua parlata occorre una visione più ampia del complesso procedimento di apprendimento di una lingua, che Jenny non poteva avere all'età in cui era entrata in contatto con la grammatica. Certamente lei non poteva vedere in che modo la grammatica potesse aiutarla a parlare meglio, né pensava di avere necessità di essere aiutata. Di conseguenza aveva sviluppato un sentimento di rancore per la grammatica. E, come succede alla maggior parte di noi, il rancore garantisce fallimento. Ma quando si è trovata nella condizione di far comporre frasi al computer, è successo qualcosa di interessante, trovandosi nella condizione di dover classificare le parole in categorie non perché qualcuno le avesse chiesto di farlo ma perché ne aveva bisogno. Per “insegnare” al suo computer come comporre serie di parole in maniera che sembrassero frasi compiute occorreva “insegnargli” a scegliere parole appartenenti alle categorie giuste. Ciò che lei aveva imparato sulla grammatica tramite l'esperienza con una macchina non aveva niente di meccanico né di routinario. Il suo era stato un apprendimento profondo e significativo. Jenny aveva fatto più che imparare le definizioni per una particolare classe grammaticale. Aveva capito l'idea generale che le parole (come le cose) possono essere collocate in gruppi o insiemi diversi, e che fare questo può essere utile. Non aveva solo “capito” la grammatica ma aveva cambiato il suo atteggiamento nei suoi confronti. Era “sua”, e nel corso dell'anno, altri casi simili l'aiutarono rivedere la propria immagine. Cambiarono anche i suoi risultati; i suoi voti, prima medio-bassi, divennero massimi per il resto degli anni scolastici. Imparò che anche lei poteva essere “un cervellone”, dopo tutto.

È naturale come matematica e grammatica non vengano capite dai bambini quando non sono capite da chi sta loro intorno e come, affinché la comprendano, occorra qualcosa di più di un insegnante che dica la cosa giusta o disegni il diagramma giusto alla lavagna. Ho chiesto a molti

18 [NdR] Abbiamo lasciato la versione originale, ci pare inutile “tradurre” un pezzo simile, ai fini della comprensione del concetto.

insegnanti e genitori cosa pensassero della matematica e perché fosse importante impararla. Pochi di loro hanno espresso una visione sufficientemente coerente da giustificare l'impiego di varie migliaia di ore della vita di un bambino per impararla, e questo i bambini lo sentono. Quando un insegnante spiega a uno studente che tutte quelle ore di aritmetica servono a essere in grado di controllare il resto al supermercato, questo non viene semplicemente creduto. I bambini interpretano tali "motivazioni" come un ulteriore esempio di malafede da parte degli adulti. Lo stesso effetto si manifesta dicendo ai bambini che la matematica scolastica è "divertente", quando è loro chiaro che gli insegnanti che si esprimono così per divertirsi fanno tutt'altre cose. Ne aiuta molto spiegare che la matematica serve per diventare scienziati poiché la maggior parte di loro non prevede una cosa del genere. La maggior parte dei bambini si rende conto che l'insegnante non ama la matematica più di quanto la amino loro e che la ragione per cui va fatta è semplicemente perché lo prevede il curricolo. Tutto ciò erode la fiducia dei bambini nel mondo degli adulti e nel processo di educazione. *E io penso che introduca un elemento di profonda disonestà nella relazione educativa*[19].

I bambini percepiscono la retorica scolastica sulla matematica come un discorso in malafede. Al fine di rimediare la situazione dobbiamo prima riconoscere che la percezione dei bambini è sostanzialmente corretta. Il "tipo di matematica" rifilata nelle scuole non è né significativa, né divertente e nemmeno utile. Ciò non significa che alcuni bambini non la possano vivere come un gioco personale importante e piacevole. Alcuni lo fanno per i voti; altri per barcamenarsi con l'insegnante e il sistema. Per molti, la matematica scolastica è piacevole proprio nella sua ripetitività, esattamente perché priva di significato e dissociata così da costituire un riparo da dover comprendere cosa stia accadendo in classe. Ma tutto questo mostra l'ingenuità dei bambini. Non si può giustificare la matematica scolastica sostenendo che, malgrado la sua intrinseca opacità, i bambini creativi vi trovino senso e divertimento.

È importante ricordarsi la distinzione fra matematica – un vasto dominio di indagine la cui bellezza è raramente immaginata da chi non è un matematico – e qualcos'altro che chiamerò "matematica scolastica".

Io interpreto la matematica scolastica come una costruzione sociale, una specie di QWERTY[20]. Un insieme di accidenti storici (che discuteremo in breve) ha determinato la scelta degli argomenti matematici che dovrebbero costituire il bagaglio matematico di un cittadino. Come nel caso della disposizione QWERTY dei tasti, la matematica scolastica ha avuto qualche senso in un certo contesto storico. Ma, analogamente, si è radicata così bene che tutti la danno per scontata, costruendo razionalizzazioni per giustificarla, ben dopo la scomparsa delle condizioni storiche che l'avevano generata. Effettivamente, per la maggior parte della gente nella nostra cultura è

19 [NdR] Corsivo dell'autore.

20 [NdR] QWERTY denota la disposizione usuale delle tastiere, dalla prima linea di tasti in alto, leggendoli da sinistra verso destra. Papert in un'altra parte del libro (Cap. 1 – I computer e le culture del computer, pag. 32-34) descrive come tale disposizione si sia stabilita con le prime macchine da scrivere meccaniche, quando i tasti avevano una certa tendenza ad incepparsi. Per tale motivo prevalse empiricamente una disposizione che minimizzasse la battitura consequenziale di tasti adiacenti, circostanza che favoriva l'inceppamento. Ben presto l'evoluzione tecnica rese inutile tale accorgimento ma la disposizione QWERTY era ormai consolidata e sarebbe ormai stato antieconomico cambiare tutto il sistema, con una moltitudine di macchine a giro per il mondo è una competenza dattilografica ormai assestata su quello standard. Papert nota come universalmente si ritenga tale disposizione ottimale, sebbene non vi siano giustificazioni tecniche concrete a parte la motivazione iniziale ormai desueta, e utilizza, anche nel proseguio del libro, il "fenomeno QWERTY per connotare altri "intrappolamenti" del pensiero che vengono giustificati a posteriori con argomenti tecnici artificiosi quando invece le vere motivazioni consistono unicamente in varie forme di inerzia.

inconcepibile che la matematica scolastica possa essere differente: questa è l'unica matematica che conoscono. Per tentare di rompere questo circolo vizioso, condurrò il lettore in un nuovo territorio matematico, la geometria della Tartaruga[21], che con i miei colleghi abbiamo creato per dare ai bambini una prima introduzione più significativa al mondo della matematica formale. I criteri di progetto della geometria della Tartaruga si comprendono meglio esaminando più da vicino le circostanze storiche che hanno formato la matematica scolastica.

Alcune di queste circostanze erano pragmatiche. Prima che comparissero le calcolatrici elettroniche, era una concreta necessità sociale quella di "programmare" molte persone affinché fossero in gradi di fare velocemente e accuratamente operazioni come lunghe divisioni. Ma ora che le calcolatrici sono accessibili economicamente dovremmo riconsiderare l'utilità di dedicare svariate centinaia di ore della vita di ciascun bambino a imparare operazioni del genere. Non intendo negare il valore intellettuale di certa conoscenza, di molta conoscenza veramente, intorno ai numeri. Ben lungi da ciò. Ma possiamo selezionare questa conoscenza in base a criteri coerenti e razionali. Ci possiamo liberare dalla tirannia di considerazioni superficiali e pragmatiche che avevano determinato le scelte del passato su cosa debba essere imparato e a quale età.

Ma l'utilità era solo una delle motivazioni storiche per la matematica scolastica; altre erano di natura *matetica*[22]. La matetica è l'insieme di principi guida che governano l'apprendimento. Alcune delle motivazioni storiche della matematica scolastica concernevano quello che poteva essere imparato e insegnato prima dell'avvento dei computer. Io credo che il fattore predominante che ha determinato quale matematica dovesse comporre la matematica scolastica fosse il contesto della classe scolastica dotata della tecnologia primitiva fatta di carta e matita. Per esempio, uno studente può disegnare un grafico con carta e matita. Così fu deciso di far disegnare molti grafici. Considerazioni simili hanno influenzato l'enfasi su certi tipi di geometria. Per esempio, nella matematica scolastica "geometria analitica" è diventata sinonimo di rappresentazione grafica delle equazioni. Il risultato è che ogni persona istruita si ricorda vagamente che $y = x^2$ rappresenta una parabola. E, sebbene la maggior parte dei genitori non abbia idea della ragione per cui ciò sia importante, questi si indignano se i anche i loro figli non lo imparano. Assumono che debbano esistere una ragione profonda e obiettiva, nota a coloro che conoscono meglio tali questioni. Ironicamente, è la propria matofobia che impedisce alle persone di esaminare tali ragioni più attentamente, trovandosi così alla mercé di sedicenti esperti di matematica. Pochissime persone sospettano che la ragione per ciò che viene incluso o meno nella matematica scolastica è così banalmente tecnologica come la facilità con cui si disegna una parabola con carta e matita! Questo è quello che può cambiare così profondamente grazie ai computer: la varietà di costrutti matematici facilmente producibili è smisuratamente più ampia.

Un altro fattore matetico nella costruzione sociale della matematica scolastica è la tecnologia della votazione. Una lingua viva si impara parlando e non necessita di un insegnante che verifica e dà voti a ciascuna frase. Una lingua morta richiede invece un "riscontro" costante da parte di un insegnante. L'attività nota come "sommare" realizza un tale riscontro nella matematica scolastica. Questi piccoli esercizi ripetitivi assurdi hanno un solo merito: sono facili da valutare. E questo

21 [NrR] *Turtle geometry*.

22 [NdR] Corsivo dell'autore. Devoto Oli: Matetico. Nelle scienze e tecniche dell'educazione, che riguarda l'apprendimento, formativo: *mezzi audiovisivi a scopo m.* [Dal gr. *Máthēsis* 'apprendimento', per influsso dell'ingl. m*athetic*].

vantaggio li ha consolidati ben bene al centro della matematica scolastica. In sintesi, io ritengo che l'edificio della matematica scolastica sia fortemente influenzato da ciò che sembrava possibile insegnare quando la matematica veniva somministrata come una materia “morta”, usando tecnologie primitive di tipo passivo, come sabbia e bastoni, lavagna e gesso, carta e matita. Il risultato è stato un insieme intellettualmente incoerente di argomenti che viola i più elementari principi matetici in merito a cosa renda certi argomenti facili da imparare e altri quasi impossibili.

A fronte dello stato di cose nella scuola, la formazione matematica può prendere due strade. Con l'approccio tradizionale la matematica scolastica viene data per scontata e ci si ingegna di insegnarla in qualche modo. Taluni usano i computer, ma, paradossalmente, l'impiego più comune è quello di impippiare materiale indigeribile, residuato dall'epoca pre-computer. Nella geometria della Tartaruga il computer è usato in modo totalmente differente, come un mezzo matematicamente espressivo, che ci libera dalla necessità di individuare argomenti matematici possibili da imparare, significativi e intellettualmente coerenti. Invece di porre la questione di come insegnare la matematica scolastica esistente, poniamo quella di “ricostruire la matematica”, o più generalmente, di ricostruire la conoscenza in maniera che non debba essere così difficile insegnarla.

Ogni “revisione del curriculum” potrebbe essere riformulata in termini di “ricostruzione della conoscenza”. Per esempio, con la riforma del curriculum New Math[23] degli anni Sessanta, qualche tentativo di cambiare i contenuti della matematica scolastica è stato fatto, ma non è cambiato molto. Le somme sono rimaste, anche se riformulate in modo un po' diverso. Il fatto che le nuove somme si riferissero agli insiemi anziché ai numeri, o all'aritmetica in base 2 anziché in base 10 ha cambiato poco. Inoltre, la riforma della matematica scolastica non ha introdotto nessuna sfida attinente alla creatività dei matematici, non presentando così nessuna delle scintille che caratterizzano la formazione di pensiero nuovo. La denominazione stessa – New Math – si è rivelata impropria. C'era veramente poco di nuovo nei contenuti: niente di attinente a un processo di invenzione della matematica dei bambini, piuttosto ad una banalizzazione della matematica dei matematici. I bambini meritano qualcosa di meglio di una selezione di vecchia matematica. Come i vestiti passati dai fratelli maggiori, che non tornano mai bene.

La geometria della Tartaruga ha presso le mosse con l'obiettivo di adattarsi ai bambini. Il primo criterio è quello della “appropriabilità”. Naturalmente i contenuti matematici devono essere pregnanti, ma vedremo che appropriabilità e pensiero matematico serio non sono affatto incompatibili. Al contrario: ci accorgeremo che alcune di tali personali conoscenze acquisite sono le più profondamente matematiche. In vari modi la matematica – per esempio la matematica dello spazio e del movimento, gli schemi ripetitivi delle azioni – è ciò che viene più naturale ai bambini. Lavorando insieme ai miei colleghi su queste idee, sono emersi alcuni concetti in grado di conferire più struttura al concetto di matematica appropriabile. In primo luogo, il *principio di continuità*: la matematica proposta deve essere in continuità con altre conoscenze, dalle quali possa ereditare un senso di familiarità e valore, insieme a competenza “cognitiva”. Poi il “principio di potenza”: dare allo studente il potere di affrontare progetti personali significativi, altrimenti impossibili. Infine il principio della “risonanza culturale”: gli argomenti devono avere senso in un contesto sociale allargato. Abbiamo parlato della geometria della Tartaruga che risulti comprensibile per i bambini. Ma questo non avverrà se non viene accettata anche dagli adulti. Una matematica di valore non può

23 [NdR] Questo è palesemente un riferimento storico, da riferire all'epoca e da contestualizzare nella realtà americana.

essere qualcosa che ci permettiamo di infliggere come fosse una medicina amara, e che non vediamo motivo di somministrare a noi stessi.

Cap. 2: Seymour Papert introduce LOGO

Prologo

Questo è il capitolo successivo a quello sulla *Mathophobia*. È quello dove Papert introduce i comandi fondamentali di Logo. Dal punto di vista tecnico, è in parte una ripetizione del mio capitolo successivo (Disegnare), tuttavia, quest'ultimo è più ricalcato sullo stile del manuale di Lakó Viktória. La descrizione di Papert invece ha respiro molto più ampio perché da un lato concerne anche il metodo di insegnamento (come introdurre in pratica i comandi della Tartaruga ai bambini) e dall'altro offre vari interessanti collegamenti fra la Lingua della Tartaruga e la matematica formale. Quale leggere prima? Decida il lettore. I più frettolosi e interessati agli aspetti operativi possono saltare questo capitolo per passare direttamente a quello successivo, però se hanno un po' di tempo, secondo me farebbero meglio a seguire prima l'introduzione di Papert, penso che ne valga la pena. Così facendo si può tuttavia presentare il problema che Papert fa riferimento a frammenti di codice che questo manuale non ha ancora presentato. Questi sono però intuitivi e invito il lettore ad accontentarsi della propria intuizione. Avrà modo di mettere a fuoco successivamente senza che la comprensione venga compromessa. Del resto è così che Papert introduce i comandi nel suo libro. Un'altra osservazione concerne la forma specifica dei frammenti di codice che non ho copiato pedissequamente, per due motivi. Il primo consiste nel problema appena menzionato, per cui ho introdotto qualche piccolo adattamento per facilitare la comprensione del lettore senza che venga alterato il messaggio fondamentale del testo. L'altro motivo consiste nel fatto che i codici scritti da Papert si riferiscono alla prima versione originale di Logo, che presenta qualche piccola differenza rispetto a quella che usiamo noi in LibreLogo. Ho quindi “tradotto” i frammenti di codice in maniera che possano essere copiati e eseguiti in un documento di LibreOffice.

Turtle Geometry: A Mathematics Made For Learning

La geometria della Tartaruga è un modo diverso di fare geometria, come il modo assiomatico di Euclide e il modo analitico di Cartesio sono differenti fra loro. Quello di Euclide è un modo *logico.* Quello di Cartesio è un modo algebrico. La geometria della Tartaruga è un modo computazionale di fare geometria.

Euclide costruì la sua geometria a partire da un insieme di concetti fondamentali, uno dei quali è il punto. Il punto può essere definito come un'entità che ha posizione ma non ha altre proprietà – non

ha colore, né misura, né forma. Le persone che non sono state iniziate alla matematica formale, che non sono state ancora “matematizzate”, hanno spesso difficoltà ad afferrare questa nozione, e la trovano bizzarra. È difficile per loro riferirla a qualcosa che conoscano. Anche la geometria della Tartaruga possiede un'entità fondamentale come il punto di Euclide. Ma questa entità, che io chiamo “Tartaruga”, può essere riferita a cose che le persone conoscono, perché a differenza del punto di Euclide, non è spogliata completamente da ogni altro attributo, e invece di essere statica e dinamica. Oltre alla posizione, la Tartaruga ha un'altra proprietà importante: ha “direzione”. Un punto euclideo si trova da qualche parte – ha una posizione e questo è tutto quello che se ne può dire. Una Tartaruga si trova da qualche parte – anch'essa ha una posizione – ma è anche rivolta da qualche parte – la sua direzione. In questo senso la Tartaruga è come una persona – io sono *qui* e sono rivolto a nord – o un animale o un battello. Ed è grazie a tali similitudini che la Tartaruga possiede la caratteristica speciale di fungere da prima rappresentazione formale per un bambino. I bambini si possono *identificare* con la Tartaruga e così sono in grado di traslare la conoscenza che hanno del loro corpo e di come si muovono nell'attività di apprendere la geometria formale.

Per sapere come funzioni dobbiamo imparare un'altra cosa sulle Tartarughe: la capacità di accettare comandi che sono espressi nella “Lingua delle Tartarughe”[24]. Il comando FORWARD fa muovere la Tartaruga lungo la direzione che sta puntando. Per dirle di quanto deve avanzare, FORWARD deve essere seguito da un numero: FORWARD 1 causa un movimento molto piccolo, FORWARD 100 un movimento più grande. In LOGO molti bambini sono stati iniziati alla geometria della Tartaruga mediante una tartaruga meccanica, sorta di robot cibernetico, in grado di obbedire ai comandi che vengono scritti su una tastiera[25]. Questa “Tartaruga da pavimento” ha le ruote, una forma semisferica e una penna sistemata in maniera che la Tartaruga possa tracciare una linea muovendosi. Ma le sue proprietà essenziali – posizione, direzione e capacità di di eseguire i comandi della “Lingua delle Tartarughe” – sono quelle che contano per fare geometria. Il bambino può poi incontrare le medesime proprietà in un'altra materializzazione della Tartaruga, sorta di “Tartaruga leggera”. Questa è rappresentata da un oggetto triangolare su uno schermo televisivo[26], che possiede le stess proprietà di posizione e direzione e si muove in base agli stessi comandi della “Lingua delle Tartarughe”. Ambedue i tipi di Tartaruga hanno vantaggi: la Tartaruga da pavimento può essere usato come una ruspa o come uno strumento per disegnare; la Tartaruga leggera traccia brillanti linee colorate più velocemente di quanto l'occhio riesca a seguirle. Nessuna delle due è meglio di un'altra, ma insieme evocano un'idea potente: due entità *fisicamente* diverse possono essere *matematicamente* eguali (o “isomorfiche”)[27].

---

24 [NdR] TURTLE TALK nell'originale.

25 [NdR] Logo in realtà è nato negli anni '70 con questa versione meccanica che è quella rappresentata in Fig. 1. Tale versione è di fatto riemersa oggi nella forma dei robot didattici Bee-Bot e Blue-Bot. Ambedue possono ricevere i comandi mediante dei tasti che hanno sul dorso. Blue-Bot può essere manovrato attraverso un app per tablet o smartphone (sia Android che Apple) che consente di comporre un intero algoritmo e di scaricarlo mediante una connessione wireless nel Blue-Bot affinché lo esegua. Qui passato e presente si ricollegano e quello che sembrerebbe un riferimento datato è invece decisamente attuale.

26 [NdR] È ovvio che lo “schermo televisivo” sia oggi sostituito dallo schermo di un computer. Inoltre, grazie alla smisurata potenza dei dispositivi di oggi, rispetto a quei tempi, il triangolo è oggi sostituito da immagini più dettagliate, quali la tartaruga stilizzata di LibreLogo o il “gatto” di Scratch. Abbiamo lasciato il testo originale per mettere in risalto il sapore pionieristico del racconto di Papert.

27 Siccome questo libro è stato scritto per lettori che non sanno molta matematica, i riferimenti matematicamente più specifici sono limitati al massimo. Le osservazioni seguenti approfondiscono un po' il commento per i lettori più esperti.
L'isomorfismo fra i diversi tipi di Tartarughe è uno degli esempi di di idee matematiche “avanzate” che che nella geometria della Tartaruga emergono in forme che sono sia concrete che *utili*. Fra queste, quelle che ricadono nel

I comandi FORWARD e BACK fanno muovere la tartaruga in linea retta lungo la propria direzione che sta puntando: cambia la posizione mentre la direzione rimane invariata. Ci sono altri due comandi che invece influiscono sulla direzione ma non sulla posizione: RIGHT e LEFT fanno girare la Tartaruga su se stessa, cambiandone la direzione di puntamento ma non la posizione. Come nel caso dell'istruzione FORWARD, RIGHT e LEFT richiedono un numero che determina l'entità della rotazione. Un adulto interpreta immediatamente tale numero come l'angolo di rotazione espresso in gradi. Per i bambini invece questi numeri devono essere esplorati attraverso il gioco.

Per disegnare un quadrato si può usare questo codice:

---

dominio dell'analisi matematica sono particolarmente importanti.

Esempio 1: Integrazione. La geometria della Tartaruga apre la strada al concetto di integrale di linea attraverso le frequenti occasioni in che la Tartaruga ha di integrare qualche quantità mentre si muove. Di solito la prima circostanza in cui i bambini si imbattono compare con la necessità di di tenere traccia della somma delle rotazioni o della lunghezza totale percorsa. Un'eccellente progetto è quello dove si simulano i tropismi che inducono gli animali a cercare condizioni quali calore, luce, concentrazione di cibo, rappresentate mediante funzioni della posizione. Capita facilmente di confrontare due algoritmi per integrare una quantità lungo il percorso della Tartaruga. Una versione semplice dell'integrazione si può realizzare inserendo nel programma una singola linea del tipo CALL (TOTAL + FIELD) “TOTAL”, che significa: prendi la quantità che si chiama “TOTAL”, aggiungile la quantità FIELD e al risultato ridai il nome TOTAL. Questa versione ha un “difetto” ([NdR] *bug*) che si manifesta quando i segmenti percorsi dalla Tartaruga sono troppo lunghi oppure sono variabili. Risolvendo problemi del genere lo studente ha modo di avvicinarsi ad un concetto di integrale progressivamente più sofisticato.

La precoce introduzione di una versione semplificata dell'integrazione lungo un percorso illustra il rovesciamento di quello che sembrerebbe l'ordine pedagogico “naturale”. Nel curriculum tradizionale, l'integrale di linea è un argomento avanzato al quale gli studenti arrivano dopo essere stati indotti per vari anni a interpretare l'integrale definito come l'area sotto una curva, un concetto che sembra attagliarsi meglio alla tecnologia della carta e della matita. Ma il risultato è quello di di sviluppare una visione fuorviante dell'integrale che causa in molti studenti un senso di smarrimento quando incontrano integrali per i quali l'immagine dell'area sotto una curva è decisamente inappropriata.

Esempio 2: Equazioni differenziali. Un progetto che colpisce molto gli studenti è quello della Tartaruga con Sensore Tattile ([NdR] *Touch Sensor Turtle*. I codici seguenti vanno letti in maniera indicativa, verranno compresi completamente più avanti nel manuale). La versione più semplice è di questo tipo:

```
TO BOUNCE
REPEAT                  ; Ciclo sulle istruzioni seguenti
 FORWARD 1              ; La Tartaruga continua a muoversi
 TEST FRONT.TOUCH       ; Controlla se sta battendo in qualcosa
 IFTRUE RIGHT 180       ; Se sì torna indietro
END
```

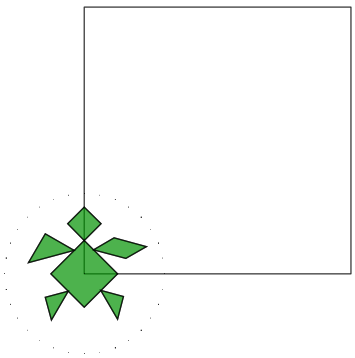
Questo codice fa sì che la tartaruga torni indietro quando batte in un oggetto. Una versione più sofisticata e più istruttiva è questa:

```
TO FOLLOW
REPEAT
  FORWARD 1
  TEST LEFT.TOUCH       ; Sta toccando l'oggetto?
  IFTRUE RIGHT 1        ; È troppo vicino: mi allontano
  IFTRUE LEFT 1         ; È troppo lontano: mi avvicino
END
```

Questo codice fa circumnavigare la Tartaruga intorno ad un oggetto di qualsivoglia forma, una volta che essa si trova con il suo lato sinistro a contatto dell'oggetto (e che l'oggetto e le irregolarità del suo contenuto siano grandi rispetto alla Tartaruga).

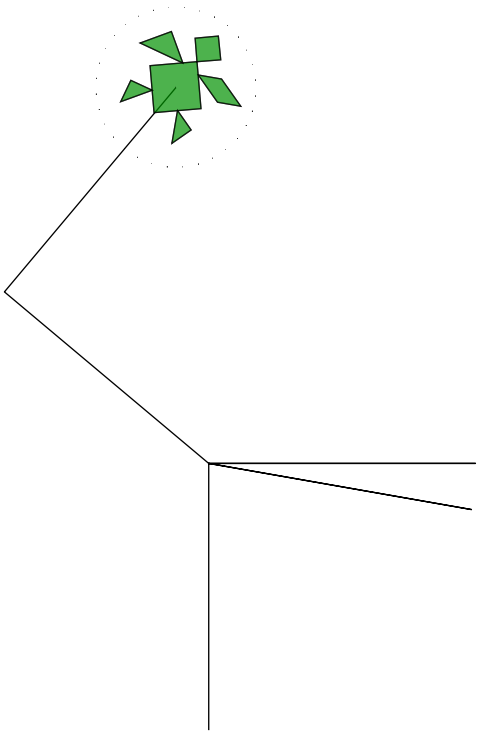
L'aspetto interessante di questi codici è quello di essere “locale”. Un comportamento “non locale” lo avremmo ottenuto per esempio se, dovendo circumnavigare un oggetto quadrato di lato pari a 150 passi, fossero state usate istruzioni del tipo FORWARD 150. Un approccio del genere manca di generalità: con altri oggetti potrebbe non funzionare. Invece i codici precedenti lavorano con piccoli passi decisi solo in base alle condizioni che si verificano nelle immediate vicinanze della Tartaruga. Invece dell'operazione “globale” FORWARD 150 usano solo operazioni “locali” come FORWARD 1. In questo modo si impiega un concetto fondamentale della nozione di equazione differenziale. Ho visto bambini della scuola primaria capire perfettamente perché le equazioni differenziali sono la forma naturale delle leggi del moto. Anche questo è un esempio eclatante di inversione pedagogica: la potenza delle equazioni differenziali è compresa prima del formalismo dell'analisi matematica. Molte delle idee matematiche suggerite dalla Tartaruga sono riunite in H. Abelson e A. diSessa, *Turtle Geometry: Computation as a Medium for*

```
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
```



Quello che segue è invece la trascrizione di un frammento dei tentativi di un bambino:

```
FORWARD 100
RIGHT 100
FORWARD 100
BACK 100

RIGHT 10
LEFT 10
LEFT 10
FORWARD 100
RIGHT 100
LEFT 10

RIGHT 100
LEFT 10
FORWARD 100
RIGHT 40
FORWARD 100
RIGHT 90
FORWARD 100
```



Exploring Mathematics (Cambridge, MIT CERCARE!!!).
Esempio 3: Invarianti Topologici. Supponiamo che la Tartaruga giri intorno a un oggetto sommando via via gli angoli delle deviazioni, contando positivamente le deviazioni destre e negativamente quelle sinistre. Il risultato finale sarà sempre pari a 360° indipendentemente dalla forma dell'oggetto. Vedremo che questo *Total Turtle Trip Theorem* è tanto utile quanto bello.

Poiché imparare a controllare la Tartaruga è come imparare una lingua in questo modo si fa leva sulla capacità e l'inclinazione dei i bambini per l'espressione verbale. E siccome quelli che si devono dare alla tartaruga sono comandi, si fa leva sull'inclinazione dei bambini a impartire comandi. Per fare disegnare un quadrato alla Tartaruga, si può provare a camminare lungo il contorno di un quadrato immaginario e poi descrivere le operazioni fatte utilizzano la Lingua della Tartaruga. E così facendo, si fa leva sulle capacità motorie dei bambini e sul piacere che provano nel muoversi. È un modo di impiegare la “geometria del corpo” propria del bambino come un punto di partenza per raggiungere la geometria formale.
L'obiettivo delle prime esperienze dei bambini nell'ambiente di apprendimento della Tartaruga non è quello ci imparare regole formali ma di sviluppare nuovi modi di concepire i propri movimenti. Tali modi sono esprimibili nella Lingua della Tartaruga e in questa diventano “programmi”, “procedure” o “equazioni differenziali”. Proviamo a guardare più da vicino come un bambino, che abbia già imparato a muovere la Tartaruga in linea retta per disegnare quadrati, triangoli e rettangoli, possa imparare a farle disegnare un cerchio.
Immaginiamo – cosa che ho osservato un centinaio di volte – un bambino che domandi: “Come faccio a fare un cerchio con la Tartaruga?” L'insegnante, nell'ambiente Logo, non dà la risposta a domande del genere bensì introduce il bambino a un metodo per risolvere non solo questo problema ma anche un'intera categoria di altri problemi. Il metodo si può riassumere in una frase: “Gioca con la Tartaruga.” Il bambino viene incoraggiato a muoversi come farebbe la Tartaruga sullo schermo per ottenere il disegno desiderato. Per il bambino che vuole disegnare un cerchio, l'atto di provare a muoversi circolarmente potrebbe tradursi nella descrizione seguente: “Quando ti muoviin cerchio tu fai un piccolo passo e poi giri subito un poco. E continui a fare sempre così.” Una volta giunti ad un a simile descrizione, la formulazione nella Lingua della Tartaruga viene spontanea:

```
REPEAT [ FORWARD 1 RIGHT 1 ]
```

Qualche bambino meno esperto potrebbe necessitare di ulteriore aiuto. Ma questo aiuto non dovrebbe consistere nella spiegazione di come fare a disegnare il cerchio bensì nell'insistere sul metodo, che concerne (oltre il consiglio di “giocare con la Tartaruga”) nello sviluppare una forte connessione fra l'attività personale e la creazione di conoscenza formale.
Nella *Mathland* della Tartaruga, le immagini antropomorfiche facilitano il trasferimento di conoscenza dai contesti famigliari a quelli nuovi. Per esempio, la metafora che si usa per ciò che viene usualmente detto “programmare il computer” è insegnare una nuova parola alla Tartaruga. Un bambino che voglia disegnare molti quadrati può insegnare alla Tartaruga un nuovo comando che ingloberà la sequenza di sette comandi necessari per disegnare un quadrato. Questo si può fare in vari modi:

```
TO QUADRATO_1
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 90
  FORWARD 100
END
```

```
TO QUADRATO_2
  REPEAT 4 [
    FORWARD 100
    RIGHT 90
  ]
END
```

```
TO QUADRATO_3 LATO
  REPEAT 4 [
    FORWARD LATO
    RIGHT 90
  ]
END
```

Analogamente si può fare per il triangolo equilatero:

```
TO TRIANGOLO_1
  FORWARD 100
  RIGHT 120
  FORWARD 100
  RIGHT 120
  FORWARD 100
END
```

```
TO TRIANGOLO_2
  REPEAT 3 [
    FORWARD 100
    RIGHT 120
  ]
END
```

```
TO TRIANGOLO_3 LATO
  REPEAT 3 [
    FORWARD LATO
    RIGHT 120
  ]
END
```

Questi codici ottengono effetti quasi eguali ma il lettore appena un po' più esperto noterà delle differenze. La più ovvia è che alcuni di questi codici consentono di diesgnare figure di dimensioni diverse ([NdR] quelle che utilizzano la variabile LATO): in questi casi i comandi per disegnare un quadrato sono QUADRATO_2 50 oppure QUADRATO_2 100, ad esempio, anziché semplicemente QUADRATO_1. Una differenza più sottile che alcuni dei codici, dopo avere disegnato la figura, lasciano la Tartaruga nello stato in cui si trovava all'inizio ([NdR] stessa posizione e stessa direzione di puntamento). Codici che utilizzano questi accorgimenti sono molto più facili da leggere e da usare in una varietà di contesti. E allorché i bambini si rendono conto di questi particolari imparano due fatti importanti. In primo luogo assimilano un “principio matetico” generale, imparando a lavorare per componenti per favorire la modularità. In secondo luogo alla fondamentale idea di “stato”.

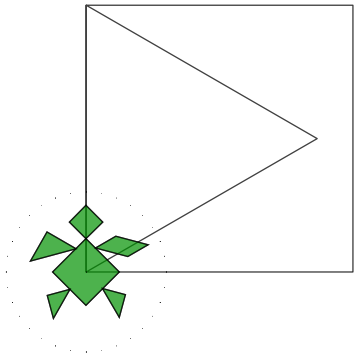
La medesima strategia di muovere dal famigliare allo sconosciuto porta lo studente in contatto con alcune idee generali forti: per esempio l'idea di organizzazione gerarchica (della conoscenza, delle organizzazioni, di un organismo), l'idea di pianificazione in un progetto e la nozione di *debugging*[28].

Non è che ci sia bisogno di un computer per disegnare un triangolo o un quadrato. Carta e matita bastano. Ma una volta che i codici per disegnare queste figure sono stati predisposti, questi diventano mattoni per costruire altre cose che consentono di creare nel bambino gerarchie di conoscenza. In tale processo si sviluppano capacità intellettuali importanti – un aspetto che si palesa guardando alcuni progetti che i bambini intraprendono spontaneamente dopo alcune sedute di lavoro con la tartaruga. Molti bambini si sono comportati in maniera simile a Pamela, che aveva iniziato insegnando al computer a fare quadrati e triangoli come abbiamo visto prima. Successivamente di è resa conto ti poter costruire una casa ponendo un triangolo sopra a un quadrato. Ecco il suo primo tentativo[29]:

28 [NdR] Abbiamo accennato a un metodo di *debugging* in LibreLogo a pag. 81.

29 [NdR] A differenza del testo originale, dove è implicito che i comandi QUADRATO e TRIANGOLO facciano riferimento ad alcuni dei codici scritti nella pagine precedenti, riscriviamo qui il codice completo per chiarezza, e in maniera che possa essere copiato e eseguito in un altro documento, per chi voglia sincerarsi da se.

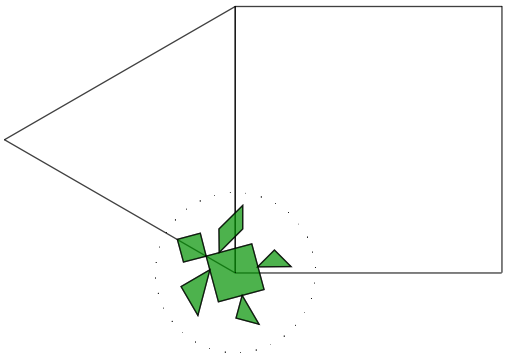
```
TO QUADRATO
        REPEAT 4 [
                FORWARD 100
                RIGHT 90
        ]
END
TO TRIANGOLO
        REPEAT 3 [
                FORWARD 100
                RIGHT 120
        ]
END
TO CASA
        QUADRATO
        TRIANGOLO
END
CASA
```



L'ultimo comando del codice è CASA ed è quello che di fatto la disegna, però il tetto è entrato in casa invece di stare sopra!

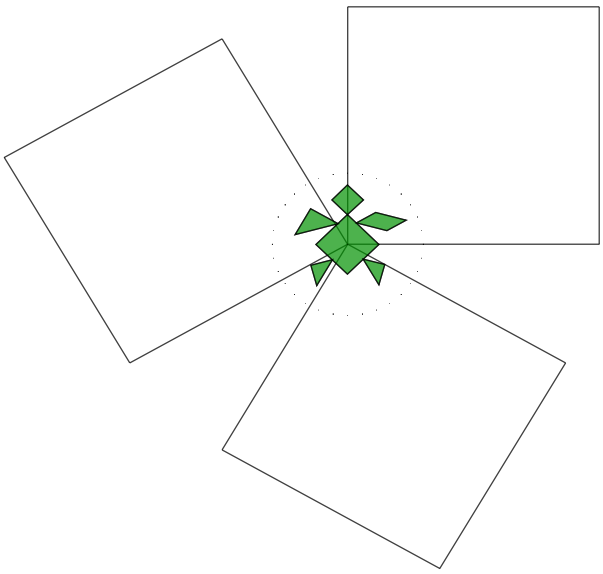
Tipicamente, in un'ora di matematica, la reazione di un bambino ad un errore sarebbe quella di *dimenticarlo* prima possibile. Invece nell'ambiente Logo, il bambino non è criticato per per l'errore del disegno. Il processo di *debugging* fa normalmente parte del processo di comprensione di quello che fa un codice. I programmatori sono incoraggiati a studiare il difetto (*bug*) anziché a dimenticare l'errore. E nel contesto della Tartaruga ci sono buone ragioni per studiare il problema, perché il premio è dato dall'ottenimento del risultato.

Il problema può essere risolto in vari modi. Pamela ne scoprì uno giocando con la Tartaruga. Ripercorrendo il cammino della Tartaruga si rese conto che il triangolo era entrato in casa perché il movimento di rotazione nel disegno del triangolo era volto a destra. Così le venne in mente di sostituire le rotazioni destre con rotazioni sinistre nella costruzione del triangolo, risolvendo così il problema. Un altro modo per risolvere lo stesso problema può essere quello di inserire un'istruzione LEFT 30 fra le istruzioni QUADRATO e TRIANGOLO. In ambedue i casi si ottiene il risultato corretto:

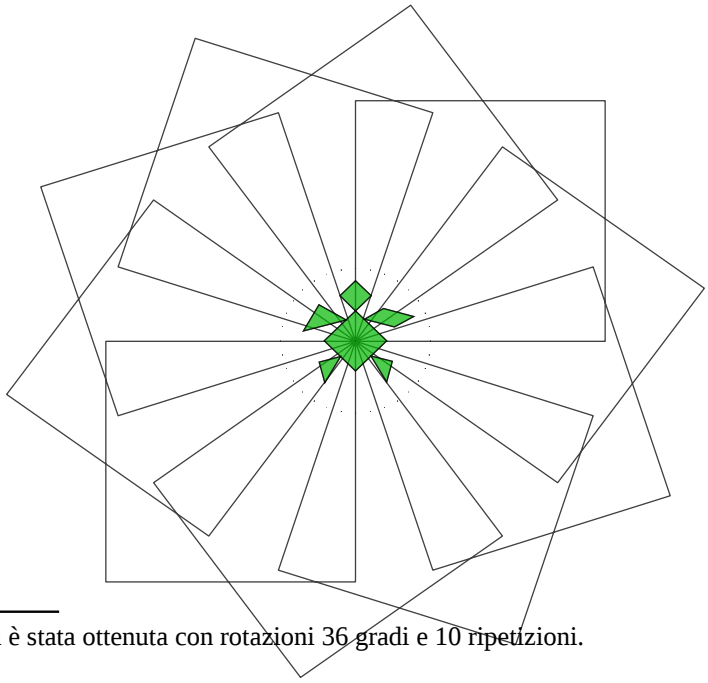


Lo studente si rende conto del progresso ma vede anche che le cose non sono mai del tutto giuste o o del tutto sbagliate, piuttosto si aggiustano in modo continuo. La casa ora è meglio di prima ma c'è ancora un difetto. Giocando ancora un po' con la Tartaruga ci si rende conto che se si inserisce un'istruzione RIGHT 90 prima di CASA, la figura si raddrizza.

Ci sono bambini che usano comporre disegni concreti, come può essere quella della casa. Altri invece preferiscono produrre effetti astratti. Per esempio, se si dà il comando QUADRATO seguito da RIGHT 120, e si ripete la stessa sequenza per altre due volte si ottiene questa figura:



Basta poi cambiare l'angolo di rotazione fra un quadrato e l'altro per ottenere una varietà di effetti[30]:



30 [NdR] Questa figura è stata ottenuta con rotazioni 36 gradi e 10 ripetizioni.

Questi esempi mostrano come i principi di continuità e di potenza[31] aiutino l'apprendimento con la geometria della Tartaruga. Ma noi volevamo ottenere anche qualcos'altro, ovvero aprire delle porte intellettuali al fine di svelare importanti idee forti. Anche solo disegnando semplici figure, quali questi quadrati e stelle, con la Tartaruga si sono evidenziate alcune idee importanti: angoli, ripetizioni controllate, operatori di cambiamento di stato. Per avere una visione più sistematica di quello che i bambini imparano con la Tartaruga iniziamo distinguendo fra due tipi di conoscenze. La prima è *matematica*: le Tartarughe sono solo un aspetto di un ampio territorio matematico, la geometria della Tartaruga, un tipo di geometria che si impara facilmente e che è portatrice di di idee matematiche molto generali. L'altro tipo di conoscenza è *matetico*: conoscenza dell'apprendimento. Prima considereremo l'aspetto matetico dell'esperienza con la tartaruga poi ci volgeremo agli aspetti più tecnici sul versante matematico. Naturalmente le due questioni si sovrappongono in parte.

Abbiamo introdotto la geometria della Tartaruga in relazione a un principio matetico fondamentale: comprendere effettivamente quello che si impara. Ricordiamoci il caso di Jenny, la quale, sebbene possedesse i requisiti concettuali per definire sostantivi e verbi, non poteva imparare la grammatica perché non riusciva a identificarsi con l'obiettivo. In questo senso fondamentale la grammatica per lei non aveva senso. La geometria della Tartaruga è stata progettata specificamente affinché i bambini vi potessero trovare un senso, per essere qualcosa che avrebbe potuto risuonare con la loro percezione di cosa fosse per loro importante. Ed è stata progettata per aiutare i bambini a sviluppare una strategia matetica: per imparare qualcosa occorre prima capirla.

L'episodio del cerchio disegnato con la Tartaruga illustra l'*apprendimento sintonico*[32]. Questo termine, preso in prestito dalla psicologia clinica, sta in contrapposizione con l'apprendimento dissociato di cui abbiamo già discusso. Talvolta il termine viene usato con degli specificatori che denotano certi tipi di sintonicità. Ad esempio, il cerchio della Tartaruga è sintonico per il corpo[33] perché tale cerchio è saldamente collegato alla percezione fisica dei propri corpi da parte del bambino. Oppure è anche sintonico per l'ego[34] perché è coerente con la percezione di sé propria dei bambini, come persone con intenzioni, obiettivi, desideri, preferenze e avversioni. Un bambino che disegna un cerchio con la Tartaruga vuole disegnare un cerchio; quando ci riesce è orgoglioso e eccitato.

La geometria della Tartaruga si impara bene perché è sintonica. E questo aiuta anche nell'apprendimento di altre cose perché incoraggia l'uso consapevole e deliberato di strategie matematiche di *problem solving*. Il matematico George Polya[35] è noto per avere proposto dei metodi

31 [NdR] Introdotti a pag. 23-2. Principio di continuità: la matematica proposta deve essere in continuità con altre conoscenze, dalle quali possa ereditare un senso di familiarità e valore, insieme a competenza “cognitiva”. Principio di potenza: dare allo studente il potere di affrontare progetti personali significativi, altrimenti impossibili.

32 L'espressione “ego-sintonico” è stata usata da Freud. È un “termine usto per descrivere istinti e idee che sono accettabili per l'ego, ovvero compatibili con l'integrità dell'ego e con le sue esigenze. (Vedi J. Laplanche e J.B. Pontalis, *The language of Pshicoanalisys* (New York: Norton, 1973).

33 [NdR] Body syntonic.

34 [NdR] Ego syntonic.

35 G. Polya, *How to Solve it* (Garden City, N.Y.: Doubleday-Anchor, 1954); *Induction and Analogy in Mathematics* (Princeton, N.Y.: Princeton University Press, 1954); and *Patterns of Plausible Inference* (Princeton, N.Y.:

generali per risolvere i problemi. Alcune delle strategie proposte con la geometria della Tartaruga sono casi particolari dei suggerimenti di Polya. Per esempio, Polya raccomanda che che tutte le volte che si trova davanti a un problema, si dovrebbe scorrere mentalmente una lista didomande euristiche: questo problema può essere suddiviso in altri problemi più semplici? Può essere collegato ad altri problemi che so già risolvere? La geometria della Tartaruga conduce naturalmente a porsi domande di questo tipo. La chiave per scoprire come fare un cerchio con la Tartaruga è quella di rifarsi a un problema la cui soluzione è invece ben nota – il problema di camminare in cerchio. La geometria della Tartaruga favorisce l'abilità di suddividere e ridurre le difficoltà. Per esempio, per fare la CASA sono stati fatti prima il QUADRATO e il TRIANGOLO. Insomma, io credo che la geometria della Tartaruga interpreta così bene i principi di Polya che il modo migliore per spiegarli agli studenti sia quello di introdurli mediante la Tartaruga. In questo modo la Tartaruga costituisce un metodo per insegnare le strategie euristiche per la soluzione dei problemi.

In seguito alla notorietà di Polya, molti hanno esortato gli insegnanti a porre maggiore enfasi sui procedimenti euristici oltre che sui contenuti. Il fallimento di quest'idea nel sistema di istruzione può essere spiegato parzialmente dalla scarsità di situazioni idonee nelle quali gli studenti possano incontrare e approfondire modelli di conoscenza euristica[36]. La geometria della Tartaruga non è solo ricca di spunti di questo genere ma aggiunge elementi nuovi ai consigli di Polya: per risolvere un problema cerca qualcosa di simile che già conosci. Il consiglio è astratto; la geometria della Tartaruga lo trasforma in un principio concreto, procedurale: *gioca con la Tartaruga*. *Prova da te*. Una fonte quasi inesauribile di situazioni simili è a disposizione perché queste sono tratte dai propri comportamenti, dal proprio corpo. Ogni volta che abbiamo un problema possiamo giocare con la Tartaruga. In questo modo di riportano i consigli di Polya sulla terra. La geometria della Tartaruga costituisce un ponte per il pensiero di Polya. Il bambino che ha lavorato estesamente con la Tartaruga è convinto che serva “cercare qualcosa di simile” perché ha funzionato spesso. Su questi successi si costruiscono la confidenza e le competenze necessari per applicare il medesimo principio anche nella matematica scolastica, dove le similarità sono meno evidenti. Infatti, nella matematica scolastica, sebbene tratti concetti elementari, è difficile applicare i consigli di Polya.

L'aritmetica non è adatta per l'introduzione del pensiero euristico. Invece la geometria Turtle è eccellente per questo. Grazie alle qualità di sintonicità, imparando a far disegnare la Tartaruga forma nel bambino un modello di apprendimento che è molto diverso da quello dissociato descritto da Bill, in V primaria, descrivendo lo studio delle tabelline a scuola: “Impari questa roba azzerando il cervello ripetendo ancora e ancora finché non la sai.” Bill aveva speso un bel po' di tempo a studiare le tabelline. I risultati erano scarsi, accuratamente documentati da Bill, con la descrizione della sua esperienza. Non riusciva a imparare a causa della assenza di ogni relazione con il contenuto che si era imposto – o piuttosto, perché aveva adottato la peggiore relazione, ovvero la dissociazione, quale strategia di apprendimento. I suoi insegnanti pensavano che “avesse poca memoria” e avevano anche discusso di un possibile problema cerebrale. Ma Bill conosceva un gran numero di canzoni popolari che non aveva difficoltà a ricordare.

Le teorie in voga intorno alla separazione delle funzioni cerebrali potrebbero suggerire che Bill

---

Princeton, 1969).

36 [NdR] Viene in mente qui la lezione di Emma Castelnuovo, con le sue proposte didattiche dove si illustrano concetti matematici mediante semplici laboratori realizzati con “materiali poveri”. Vedi ad esempio E. Castelnuovo, *L'Officina Matematica*, Edizioni La Meridiana, Molfetta /BA), 2008.

avesse un deficit di memoria specificamente per i numeri. Tuttavia il ragazzo si ricordava facilmente di altre migliaia di numeri, prezzi e date. La differenza fra ciò che ricordava e ciò che non ricordava non dipendeva dal contenuto ma dalla propria relazione con esso. La geometria della Tartaruga, in virtù della sua connessione con il ritmo e il movimento richiesto nella vita quotidiana, consentiva a Bill di collegarla più alle canzoni che alle tabelline. I suoi progressi furno spettacolari. Attraverso la geometria della Tartaruga la conoscenza matematica che fino ad allora Bill aveva rifiutato, fece comparsa nel suo mondo.

Passiamo ora dalle considerazioni matetiche a quelle matematiche. Che matematica si impara quando con la geometria della Tartaruga? Ai fini della presente discussione distinguiamo tre tipi di conoscenza matematica, ciascuna delle quali si avvantaggia del lavoro con la Tartaruga. In primo luogo c'è il corpo di conoscenze noto come “matematica scolastica”, che è stato selezionato esplicitamente (nella mia opinione soprattutto in virtù di accidenti storici) come il “nucleo” della matematica di base che tutti i cittadini dovrebbero conoscere. Poi c'è quella che chiamo “protomatematica” che la matematica scolastica presuppone pur non menzionandola esplicitamente nei curricoli tradizionali. Parte di tale cultura ha natura “sociale”: per esempio la conoscenza che attiene al perché si debba studiare matematica e cosa significhi capirla. Altre conoscenze in questa categoria sono quelle messe in evidenza dall'epistemologia genetica[37] a proposito della formazione: principi deduttivi quali la transitività, le conservazioni, la logica intuitiva della classificazioni e via dicendo. Infine, la terza categoria è costituita dalla conoscenza che non è coperta dalla matematica scolastica né è presupposta da questa ma che che dovrebbe far parte del bagaglio intellettuale di un cittadino istruito del futuro.

Io penso che la comprensione delle relazioni fra i sistemi geometrici euclideo, cartesiano e differenziale faccia parte di questa terza categoria. Disegnare un cerchio con la Tartaruga è molto più che un “modo intuitivo” di disegnare cerchi, perché pone il bambino in contatto con un insieme di idee che stanno alla base dell'analisi matematica. Questo fatto può non essere evidente per persone il cui unico contatto con l'analisi ha avuto luogo nella scuola superiore o in qualche corso all'università, nella forma di certe manipolazioni formali di simboli. Il bambino che ha disegnato il cerchio con la Tartaruga non ha imparato qualcosa sul *formalismo* dell'analisi, per esempio che la derivata di x^n è nx^{n-1}, ma qualcosa sul suo impiego e sul suo *significato*. Infatti il codice per disegnare il cerchio con la Tartaruga conduce a un formalismo alternativo di quelle che sono tradizionalmente chiamate “equazioni differenziali” ed è un veicolo efficace delle idee che soggiaciono al differenziale. Questo è il motivo per cui si possono capire così tante cose con la Tartaruga; *il codice del cerchio rappresenta un'analogia intuitiva dell'equazione differenziale, un concetto che appare in quasi tutti gli esempi di matematica applicata tradizionale*.

La potenza del calcolo differenziale risiede molto nella capacità di descrivere le variazioni in base a ciò che accade nelle loro immediate vicinanze. È questa caratteristica che ha consentito a Newton di descrivere il moto dei pianeti. Via via che questi tracciano l'orbita, sono le condizioni locali nel luogo dove si trova il pianeta che determinano il suo prossimo passo. Nelle nostre istruzioni della Tartaruga, FORWARD 1 RIGHT 1, ci si riferisce solo al luogo dove si trova la Tartaruga e a quello dove si troverà il momento dopo. Questo è quello che rende un'equazione *differenziale*. In ciò non vi è alcun riferimento a luoghi remoti rispetto al percorso. La Tartaruga vede il cerchio cammin

37 L'epistemologia genetica è stata creata da Jean Piaget per descrivere la genesi della conoscenza nel bambino.

facendo, nell'immediata vicinanza, ed è cieca rispetto a tutto il resto che si trova più lontano. Questa proprietà è così importante che i matematici hanno un nome per essa: la geometria della Tartaruga è “intrinseca”. Lo spirito della geometria differenziale intrinseca si palesa quando si considerano i diversi modi di concepire una curva, ad esempio il cerchio. Per Euclide la caratteristica che definisce il cerchio è la distanza costante dei suoi punti da un altro punto, il centro, che però non fa parte di esso. Nella geometria di Cartesio, in questo caso più similmente a Euclide che che alla Tartaruga, i punti del cerchio sono caratterizzati dalla loro distanza rispetto a qualcos'altro, vale a dire dai due assi perpendicolari delle coordinate. Così, per esempio, un cerchio è definito da:

$$(x-a)^2 + (y-b)^2 = R^2$$

Nella geometria della Tartaruga un cerchio è definito dal fatto che questa continua a ripetere uno stesso atto: FORWARD un poco, GIRA un poco. Questa ripetizione garantisce che la curva abbia “curvatura costante”, dove di quanto si deve girare ad ogni passo.

La geometria della §Tartaruga appartiene ad una famiglia di geometrie che godono di proprietà assenti in quelle euclidea e cartesiana. Queste sono le geometrie differenziali che si sono sviluppate a partire da Newton e che hanno reso possibile gran parte della fisica moderna. Abbiamo osservato come quello delle equazioni differenziali sia il formalismo che ha consentito alla fisica di descrivere il moto di una particella o di un pianeta. Nel capitolo 5, dove descrivere questo fatto con maggiori dettagli, vedremo come questo sia anche il formalismo appropriato per descrivere il moto di un animale oppure l'evoluzione di un'economia. E arriveremo anche a capire che non è un coincidenza il fatto che la geometria della Tartaruga sia collegata sia all'esperienza di un bambino che alle principali conquiste della fisica, in quanto, la visione del moto di un bambino, sebbene meno precisa nella forma, condivide la struttura matematica dell'equazione differenziale con le leggi del moto di un pianeta che gira intorno al sole o quelle delle falene che girano introno alla fiamma di una candela. E la Tartaruga non è ne più ne meno che la ricostruzione in forma computazionale intuitiva del nucleo qualitativo di questa struttura matematica. Quando torneremo su queste idee nel capitolo 5, vedremo come la geometria della Tartaruga apra le porte alla comprensione intuitiva dell'analisi, della fisica e della modellazione matematica così come viene impiegata nelle scienze biologiche e sociali.

L'effetto del lavoro con la geometria della Tartaruga su alcuni aspetti della matematica scolastica è primariamente *relazionale* e *affettivo*: molti bambini sono venuti nel laboratorio LOGO odiando i numeri quali oggetti alieni e se ne sono andati amandoli. In altri casi il lavoro con la Tartaruga ha generato modelli intuitivi specifici per concetti matematici complessi che i bambini capiscono con difficoltà. Un esempio semplice è l'uso dei numeri per misurare gli angoli. Nel contesto della Tartaruga i bambini assumono questa capacità quasi inconsapevolmente. Tutti – inclusi alcuni bambini di prima e molti di terza con cui abbiamo lavorato – escono dall'esperienza con una percezione migliore di cosa significhi 45 gradi, o 10 o 360, di quella della maggioranza degli studenti di scuola media. Così si ritrovano preparati meglio per affrontare tutti i vari argomenti formali – geometria, trigonometria, disegno geometrico ecc. - nei quali il concetto di *angolo* gioca un ruolo centrale. Ma sono preparati anche per qualcos'altro, ovvero un aspetto delle misure angolari nella nostra società nei confronti della quali la matematica scolastica è sistematicamente cieca.

Una delle rappresentazioni più diffuse dell'idea di angolo nelle vite dei contemporanei si trova nella

navigazione. Milioni di persone vanno in barca, in aeroplano o leggono mappe[38]. Per la maggior parte queste attività *comuni* e la *morta* matematica scolastica. Abbiamo messo in evidenza come l'uso della Tartaruga quale veicolo dell'idea di angolo si colleghi saldamente alla geometria del corpo. Abbiamo chiamato questa caratteristica sintonicità del corpo. Qui scopriamo una *sintonicità culturale*: la Tartaruga connette l'idea di angolo alla navigazione, attività che è fermamente e positivamente radicata nella cultura extrascolastica di molti bambini. E via via che i computer si diffonderanno nel mondo, la sintonicità culturale della geometria della Tartaruga sarà sempre più forte[39].

Un secondo concetto matematico la cui comprensione è facilitata dalla Tartaruga è l'idea di *variabile*: l'idea di usare un simbolo per dare un nome a un'entità sconosciuta. Per apprezzare questo contributo della Tartaruga estendiamo il codice per il cerchio in modo da disegnare delle spirali.

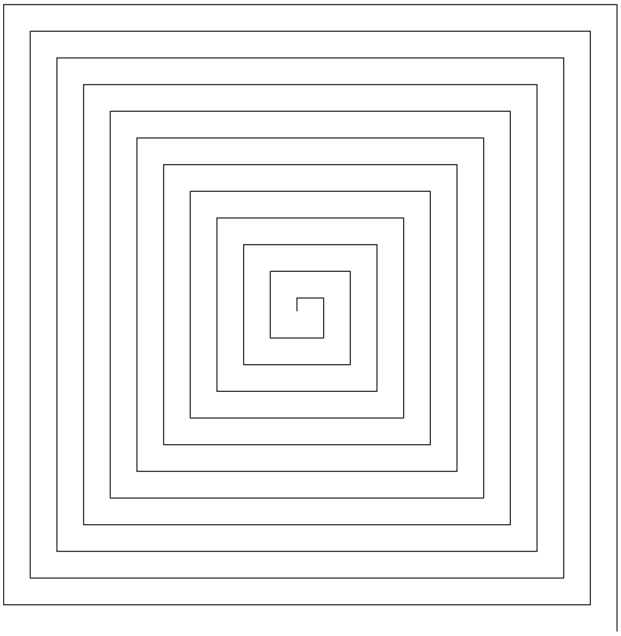


Figura 2: Spirale 1 - si incrementa progressivamente il passo

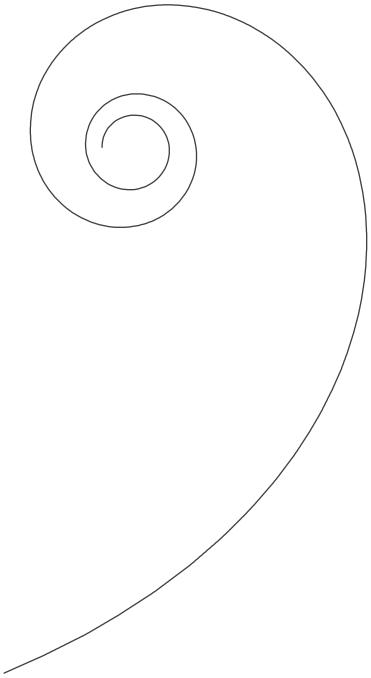


Figura 1: Spirale 2 - si decrementa progressivamente l'angolo

Guardiamo per esempio la spirale a destra. Come nel caso del cerchio, anche questa può essere disegnata in accordo con la prescrizione: procedi un poco, gira un poco. La differenza è che nel cerchio “è sempre lo stesso” la spirale diventa sempre più piatta, “meno curva”, procedendo verso l'esterno. Il cerchio è una curva con curvatura costante. La curvatura della spirale diminuisce andando verso l'esterno[40]. Per camminare lungo una spirale uno potrebbe fare un passo, girare un

38 [NdR] Oggi questo tipo di esperienza è enormemente e (per Papert) imprevedibilmente ampliata, includendo le varie forme di navigatori satellitari e la disponibilità ubiquitaria nei computer, tablet o smartphone di potentissimi software e servizi Web di tipo geografico o astronomico.

39 [NdR] Purtroppo è andata diversamente. La geometria della Tartaruga, sebbene si trovi all'origine della creazione e della diffusione di Scratch, ha perso la forza di questo specifico messaggio che Papert si auspicava.

40 [NdR] Lasciamo al lettore l'esercizio di scrivere il codice per ottenere questa spirale e giocare provando a variarne

poco, fare un altro passo uguale, girare un poco ma ogni volta appena un po' meno (oppure facendo ogni volta un passo appena un po' più lungo. Per tradurre questo comportamento in istruzioni da dare alla Tartaruga, è necessario avere un modo per esprimere il fatto che abbiamo a che fare con una quantità *variabile*. In principio se ne potrebbe fare a meno ma scrivendo un codice molto lungo:

```
TO SPI_1                TO SPI_2
FORWARD 5               FORWARD 1
RIGHT 90                RIGHT 5
FORWARD 15              FORWARD 1
RIGHT 90                RIGHT 5 * .995
FORWARD 20              FORWARD 1
RIGHT 90                RIGHT 5*.995*.995
FORWARD 25              FORWARD 1
RIGHT 90                RIGHT 5*.995*.995*.995
FORWARD 30              FORWARD 1
RIGHT 90                RIGHT 5*.995*.995*.995*.995
FORWARD 35              FORWARD 1
RIGHT 90                RIGHT 5*.995*.995*.995*.995*.995
FORWARD 40              FORWARD 1
RIGHT 90                RIGHT 5*.995*.995*.995*.995*.995*.995
FORWARD 45              FORWARD 1
RIGHT 90                RIGHT 5*.995*.995*.995*.995*.995*.995*.995
FORWARD 50              FORWARD 1
RIGHT 90                RIGHT 5*.995*.995*.995*.995*.995*.995*.995*.995
FORWARD 55              FORWARD 1
RIGHT 90                RIGHT 5*.995*.995*.995*.995*.995*.995*.995*.995*.995
FORWARD 60              FORWARD 1
RIGHT 90                RIGHT 5*.995*.995*.995*.995*.995*.995*.995*.995*.995*.995
FORWARD 65              ecc.
RIGHT 90
ecc.
```

Come NON disegnare spirali

Qui, per la spirale di destra, abbiamo specificato esattamente dopo ogni passo di quanto deve girare la Tartaruga. Questo è noioso[41]. Un metodo migliore è quello di usare il concetto di nome simbolico mediante una variabile, una delle idee matematiche più potenti che siano mai state inventate.

Nella Lingua della Tartaruga le variabili sono presentate nella forma di un mezzo di comunicazione. Quello che noi vogliamo dire alla Tartaruga è: “Vai avanti di un piccolo passo, quindi gira un po', ma non so di quanto ora perché sarà sempre di una quantità differente.” Per disegnare la spirale di sinistra diremmo: “fai un passo, che però sarà ogni volta differente, e poi gira di 90.” Nel linguaggio matematico il trucco per dire qualcosa del genere è quello di inventare un nome per “la quantità che ora non ti posso dire”. Il nome potrebbe essere una lettera, come *X*, o potrebbe essere una parola intera, come ANGOLO o DISTANZA. (Uno dei contributi minori della cultura computazionale alla matematica è l'abitudine di usare per i nomi delle variabili parole che facilitino la memorizzazione

parametri.

41 [NdR] E, alla lunga, impossibile. Se volessimo scrivere il codice aggiustando i parametri in maniera da iniziare da molto piccola, avvolgendosi fittamente, diverrebbe presto impossibile scrivere tutto il codice.

invece di singole lettere.) Per applicare l'idea di variabile, la Lingua della Tartaruga consente di creare “procedure con un ingresso (*input*)”[42]. Queste si ottengono scrivendo per esempio:

```
TO PASSO DISTANZA
        FORWARD DISTANZA
        RIGHT 90
END
```

Se in un frammento di codice introduciamo l'istruzione **PASSO 100**,[43] quello che otterremo sarà che la Tartaruga andrà avanti di 100 unità di percorso e poi girerà a destra di 90 gradi. Similmente, con **PASSO 110** la Tartaruga andrà avanti di 100 unità e poi girerà di 90 gradi. Con il LOGO noi incoraggiamo i bambini a servirsi di metafore antropomorfiche: il comando PASSO dà un ordine a un aiutante (“l'uomo che fa un passo”) il cui compito è di dare alla Tartaruga due comandi, un FORWARD e un RIGHT. Ma questo aiutante non può fare il lavoro senza un altro dato – un numero da passare a un “uomo che fa il FORWARD”, per specificare quanto deve essere lungo il passo.

La procedura PASSO non è entusiasmante, ma con una piccola modifica lo può diventare. Confrontiamola con la procedura SPI_1, che è esattamente la stessa[44] eccetto che per una linea:

```
TO SPI DISTANZA
        FORWARD DISTANZA
        RIGHT 90
        SPI DISTANZA + 5
END
```

Se noi diamo il comando SPI 100 invochiamo un aiutante di nome SPI dandogli un valore in ingresso (*input*) pari a 100. L'aiutante SPI dà a sua volta tre comandi. Il primo è esattamente lo stesso di quello dell'aiutante PASSO: dì alla Tartaruga di andare avanti di 100 unità. Il secondo dice alla Tartaruga girare a destra. Anche qui nulla di nuovo. Ma il terzo contiene qualcosa di straordinario. Questo comando risulta essere SPI 105. Qual'è l'effetto? Quello di dire alla tartaruga di andare avanti di 105 unità, di girare di 90 gradi, e poi di dare il comando SPI 110. È così che si ottiene un trucco che si chiama “ricorsività”, con la quale si dà vita a un processo infinito che si realizza ad esempio con le spirali raffigurate precedentemente.

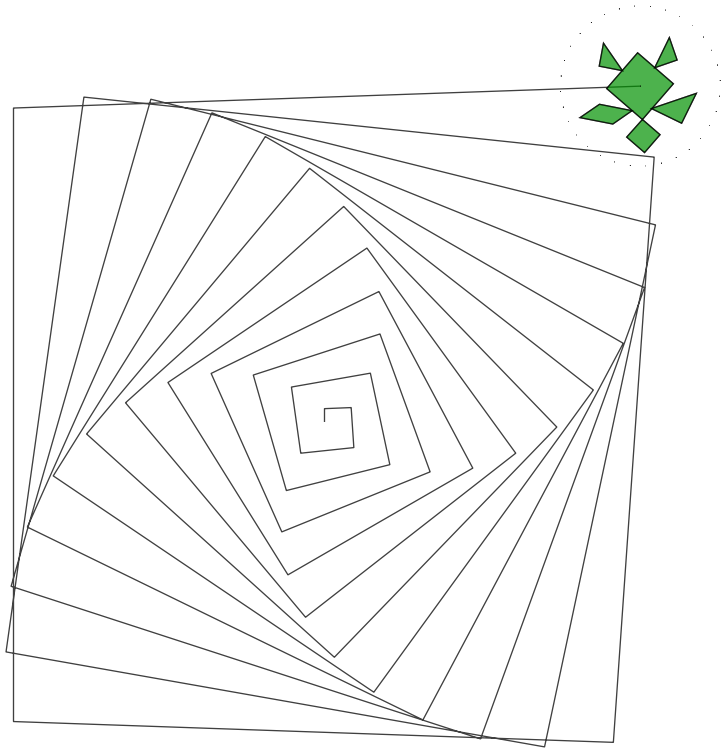
Di tutte le idee che ho presentato agli studenti, la ricorsività spicca come quello che più facilmente causa meraviglia. Io penso,in parte perché l'idea di procedere all'infinito solletica la fantasia di qualsiasi bambino, e in parte perché la ricorsività medesima ha radici nella cultura popolare. Ecco un indovinello ricorsivo: se hai un desiderio qual'è il secondo? (altri due desideri.) Oppure

42 [NdR] Queste che Papert chiama “procedure”, nel resto del manuale le abbiamo chiamate prevalentemente “subroutine”. La denominazione varia a seconda del linguaggio e del contesto. Più o meno sinonimi sono “procedure”, “subroutine”, “funzioni”, “metodi”.

43 [NdR] L'accorgimento tipografico di usare il grassetto è mio e, come in altre parti del manuale, serve a focalizzare l'attenzione sui particolari più importanti nel contesto del discorso.

44 [NdR] Nel senso che la procedura SPI_1 nella pagina precedente, è sì molto più lunga, ma contiene solo due tipi di istruzioni: FORWARD 5 e RIGHT 90.

l'immagine evocativa di un'etichetta che riporta la sua propria immagine[45]. Offrendo l'opportunità di di giocare con l'infinito si pongono i bambini in contatto con qualcosa che si avvicina ad essere un matematico. Un altro esempio di esperienza matematica viva è illustrato nella figura seguente, dove si vede come sia possibile esplorare un curioso fenomeno matematico variando l'angolo di rotazione nella procedura SPI.



Angoli vicino a 90[46] producono l'emergenza di un fenomeno sorprendente: le braccia della galassia non sono stati codificati esplicitamente nella procedura. Emergono in maniera sorprendente e spesso danno luogo a prolungate esplorazioni nel corso delle quali riflessioni di tipo numerico e geometrico si intrecciano con considerazioni estetiche.

In Logo spesso le nuove idee compaiono in risposta alla necessità di ottenere un risultato per fare qualcosa che prima non era possibile. Nel contesto scolastico tradizionale lo studente incontra la nozione di variabile in piccoli problemi come questo:

$5 + X = 8$. Quanto vale X?

Pochi bambini lo vedono come un problema personalmente rilevante, e ancora meno interpretano il metodo di soluzione come una fonte di potere. E hanno ragione. Non è che ci possono fare molto nel contesto della loro vita. In LOGO è invece molto diverso. Qui il bambino ha un obiettivo personale: quello di fare una spirale. In tale contesto l'idea di variabile è una sorgente di potere

45 [NdR] Oppure la fuga di immagini che si ottiene quando si guarda in uno di due specchi contrapposti. Il concetto di ricorsività è realmente potente. Ad esempio apre le porte al mondo dei frattali. Con Logo si possono ottenere degli effetti grafici sorprendenti e suggestivi. Approfondiremo questo tema più avanti nel manuale.

46 [NdR] Quello usato nella figura è pari a 88 gradi.

personale, potere di esaudire un desiderio altrimenti inaccessibile. Certamente, molti dei bambini che si imbattono nella nozione di variabile nel contesto convenzionale imparano a usarla efficacemente. Ma raramente nasce in loro un senso di “potere matematico”[47], nemmeno nei più bravi e più brillanti. E questo è il punto di forte contrasto fra la situazione in cui l'idea di variabile venga incontrata nella scuola tradizionale o nell'ambiente LOGO. In LOGO il concetto conferisce potere al bambino, il quale sperimenta come la matematica attivi una cultura che rende possibile ciò che prima non lo era.

Se l'uso della variabile per fare una spirale fosse stato introdotto come un esempio isolato per “illustrare” il “concetto di potere matematico” l'opportunità di agganciare l'attenzione dei bambini (come gli ingranaggi avevano agganciato la mia[48]) sarebbe occorsa casualmente e in pochi casi. Ma nella geometria della Tartaruga questo non è un episodio isolato. È il modo usuale di incontrare nuova conoscenza matematica. Il “potere matematico”, si potrebbe dire, “diviene un modo di vita”. Il senso di potenza non si associa solo metodi immediatamente applicabili come l'uso delle misure angolari o le variabili, ma anche con concetti come “teorema” o “dimostrazione” o “euristico” o “metodo di problem solving”. Usando concetti del genere il bambino sviluppa dei modi per parlare *di* matematica. Ed è su tale sviluppo dell'articolazione matematica che ora ci soffermiamo.

Consideriamo un bambino che con la Tartaruga abbia già disegnato un quadrato e un cerchio e che ora vorrebbe disegnare un triangolo che con tutti e tre i lati eguali a 100 unità. La forma di questo programma potrebbe essere:

```
TO TRIANGOLO
        REPEAT 3
        FORWARD 100
        RIGHT QUALCOSA
END
```

Ma affinché la Tartaruga disegni la figura il bambino deve dirle qualcosa di più. Qual è la quantità che abbiamo chiamato QUALCOSA? Per il quadrato avevamo istruito la Tartaruga affinché girasse di 90 gradi ad ogni vertice, con il codice seguente:

```
TO QUADRATO
        REPEAT 4
        FORWARD 100
        RIGHT 90
END
```

Qui possiamo apprezzare come il precetto di Polya, “cerca situazioni simili”, e il principio

47 [NdR] *Mathpower*.

48 [NdR] Qui Papert si riferisce alla storia che aveva narrato nell'introduzione del libro sull'interesse che gli ingranaggi avevano casualmente destato in lui quando era piccolo, un episodio che avrebbe poi funzionato da potente catalizzatore per la sua formazione.

procedurale della geometria della Tartaruga, “gioca con la Tartaruga”, possano lavorare insieme. *Cosa c'è di eguale* fra il quadrato e il triangolo? Se tracciamo il percorso che vogliamo far seguire alla Tartaruga, notiamo che in ambedue i casi questa finisce ritrovandosi nella posizione iniziale e rivolta nella stessa direzione. In altre parole, si finisce nello stato in cui eravamo partiti. E nel frattempo abbiamo fatto un giro completo. *Ciò che è diverso* nei due casi è il fatto che il giro sia stato compiuto in tre o quattro tappe. Il contenuto matematico di quest'idea è tanto ricco quanto semplice. Il fatto importante è la nozione del viaggio totale – quanto devi girare in tutto per fare il giro completo?

La cosa sorprendente è che tutti i giri completi danno lo stesso risultato, pari a 360 gradi. Le quattro rotazioni di 90 gradi del quadrato fanno 360 gradi e siccome le rotazioni hanno luogo presso i vertici, nel caso del triangolo ci sono tre rotazioni che varranno 360 gradi diviso per tre. Così la quantità che abbiamo chiamato QUALCOSA deve valere 120 gradi. Questa è l'enunciazione del “Teorema del Viaggio Totale”[49]:

> Se una Tartaruga gira intorno al contorno di qualsiasi figura, finendo nello stato iniziale, allora la somma di tutte le rotazioni sarà pari a 360 gradi.

Parte integrante della comprensione di questo risultato è avere acquisito un metodo per risolvere una ben definita classe di problemi. L'incontro del bambino con questo teorema è diverso per vari aspetti rispetto alla memorizzazione della sua controparte euclidea: “la somma degli angoli interni di un triangolo è 180 gradi.”

In primo luogo (almeno nel contesto di un computer con LOGO), il Teorema del ViaggioTotale è più potente, perché il bambino lo può usare. Secondariamente è più generale: si applica ai quadrati e alle curve come ai triangoli. Terzo, è più facilmente comprensibile. Si dimostra facilmente. Ed è più personale: lo puoi “percorrere” ed è rappresentativo di un modello per porre la matematica con la conoscenza personale.

Abbiamo visto come un bambino può servirsi del Teorema del Viaggio Totale per disegnare un triangolo equilatero. Quello che è veramente interessante è osservare come il teorema possa essere riapplicato in progetti molto più complessi. Perché ciò che conta quando un bambino si appropria di un teorema, non sta nella memorizzazione, ma nel fatto che con pochi teoremi importanti si apprezza quanto certe idee possano costituire strumenti del pensiero utili per tutta la vita. Si impara a apprezzare e rispettare la forza delle idee potenti. Si impara che l'idea più potente di tutte è l'idea di idea potente.

49 [NdR] *Total Trip Theorem*.

Cap. 3: Disegnare

Comandi di movimento – Disegno - Uso variabili

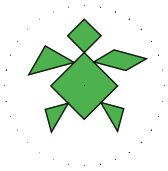
I comandi fondamentali

Il programma consente di creare grafica mediante il movimento di una “tartaruga” che obbedisce a precisi comandi. Vediamo subito un esempio.

Apri un nuovo documento di testo e scrivi questo comando:

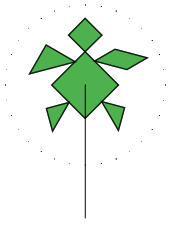
HOME

Vedrai che è apparsa la tartaruga in mezzo al foglio con la testa verso l'alto:



Ora aggiungi un altro comando (d'ora in poi scriverò in grassetto solo i nuovi comandi che introdurremo):

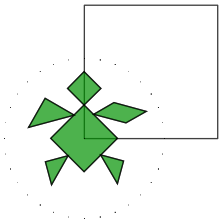
HOME
FORWARD 50



La tartaruga si è mossa tracciando una linea; è così che si disegna dando comandi alla tartaruga.

Scriviamo ora le seguenti istruzioni:

HOME
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50



RIGHT 90
FORWARD 50
RIGHT 90

Abbiamo tracciato quattro lati, e alla fine di ogni lato abbiamo girato a destra di 90 gradi, ottenendo così un quadrato.

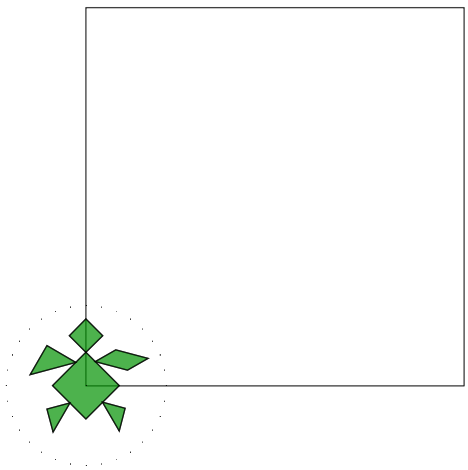
Due parole sulla particolarità di LibreLogo. La sequenza di istruzioni che abbiamo scritto è un frammento di codice, è software. L'abbiamo scritto in un particolare linguaggio, che è quello di Logo, ed è anche molto semplice, ma è software come qualsiasi altro. Di solito il software si scrive in appositi documenti mediante editori di testo semplice e si salvano in questo modo. Poi, si fanno eseguire al computer. I modi con qui si eseguono queste operazioni variano molto a seconda del tipo di linguaggio e di contesto. Oggi ci sono centinaia di linguaggi diversi che servono per gli scopi più disparati. La particolarità di LibreLogo è che il software si scrive in un documento e la tartaruga "lavora" sul documento medesimo, lasciandovi la propria opera sotto forma di grafica, Così uno si ritrova insieme il codice e il risultato grafico prodotto da esso, in un unico documento. La grafica può essere selezionata con il mouse e, eventualmente, trasportata in contesti diversi. Ad esempio, le figure qui sopra le ho generate giocando con la tartaruga in un altro documento, poi ho selezionato le grafiche e le ho riportate qui. Un'altra notazione: un gruppo di istruzioni da fare funzionare in sequenza si chiama *script*, espressione che useremo diffusamente.

L'integrazione fra Logo e LibreOffice va oltre. È evidente che quando scriviamo il comando

FORWARD 50

il numero 50 esprime la lunghezza del percorso che la tartaruga deve compiere. Puoi verificare subito cambiando il valore e guardando cosa fa la tartaruga. Ma cosa rappresenta quel 50? Sono punti tipografici (point): 50 pt. Un punto sono 0.35 mm[50]. LibreLogo capisce le unità di misura, si può scrivere 50, 50pt, 50mm, 50cm, 50in (inch: pollice), 50" (" sta per pollice). Ovviamente si tratta di lunghezze diverse. Usiamo per esempio i mm:

CLEARSCREEN
HOME
FORWARD **50mm**
RIGHT 90
FORWARD **50mm**
RIGHT 90
FORWARD **50mm**
RIGHT 90
FORWARD **50mm**
RIGHT 90

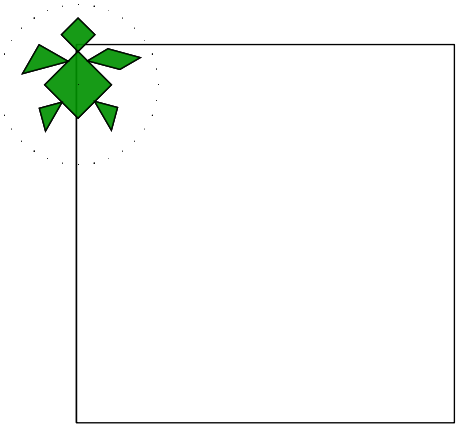


50 La definizione precisa è: 1 pt = 1/72 polllici, dove 1 pollice = 25.4 mm. Quindi 1 pt = 2.54/72 = .352777… mm

Il quadrato è ovviamente più grosso perché il precedente aveva il lato di 50 pt = 17.6 mm.

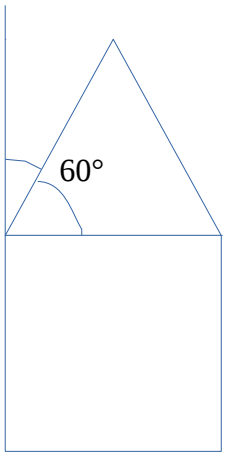
Proviamo a complicare il disegno, immaginando di fare una casetta. La tartaruga si trova nel vertice in basso a sinistra e guarda in alto. Come prima cosa dobbiamo farla salire fino al vertice in alto a sinistra. Proviamo, e allo stesso tempo approfittiamo anche del fatto che le istruzioni possono essere raggruppate in una stessa riga, a seconda della convenienza:

```
CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm
```

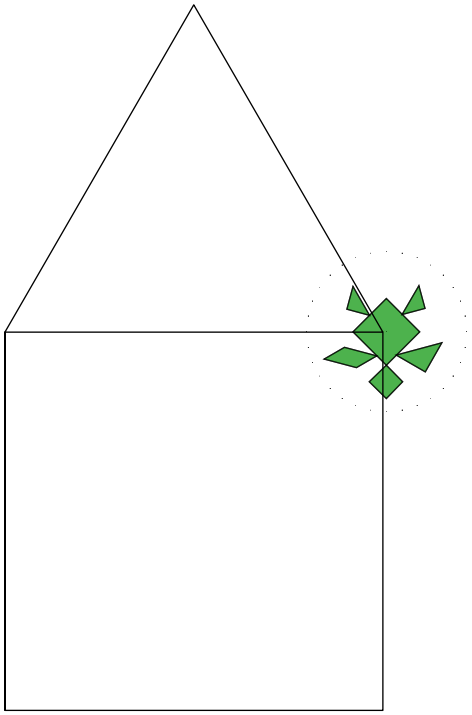


Non ci sono regole precise per raggruppare le istruzioni in una stessa riga. Il raggruppamento si può decidere in base alla comodità con cui si rilegge il codice. È importante facilitarsi la vita perché via via che il codice cresce può complicarsi rapidamente e tutti gli accorgimenti per renderlo più chiaro sono utili.

Ora dobbiamo costruire il tetto sulla casa. Facciamo questo appoggiando sul quadrato un triangolo equilatero, con la base coincidente con il lato superiore del quadrato. Essendo equilatero, anche gli altri due lati del triangolo saranno lunghi 50mm. Quindi per fare la falda sinistra del tetto la tartaruga dovrà spostarsi di 50mm ma deve prima cambiare direzione. Di quanto? Siccome gli angoli interni di un triangolo equilatero sono di 60°, la tartaruga dovrà deviare di 90° – 60° = 30° a destra. Arrivata in cima, disegnando la falda sinistra del tetto, dovrà girare a destra di 120° per poi disegnare la falda destra. Infine, girerà di 30° a destra per riallinearsi alla parete della casa.

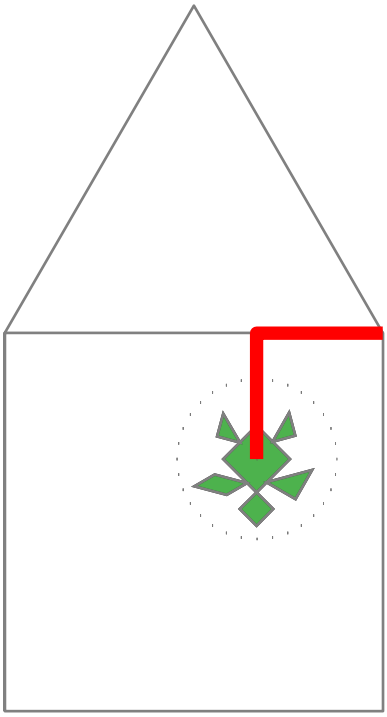


```
CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 30
```



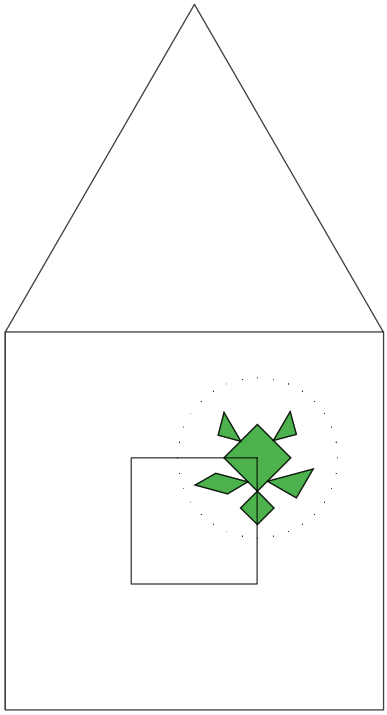
Supponiamo ora di voler disegnare una finestra nel mezzo della parete. Per fare questo è necessario introdurre due nuovi comandi - PENUP e PENDOWN – che consentono di muovere la tartaruga senza disegnare.

CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 30
PENUP
FORWARD 50mm/3
LEFT 90
FORWARD 50mm/3



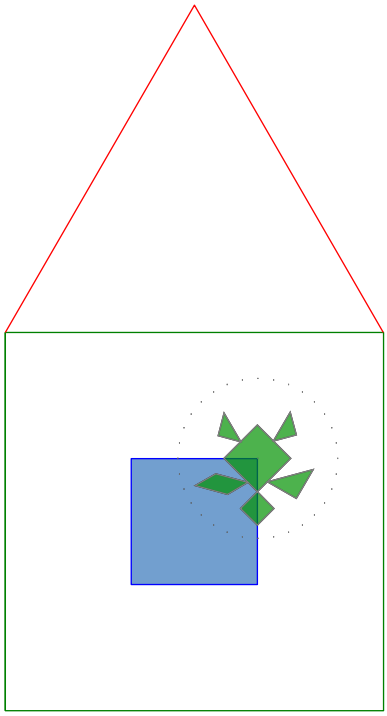
dove abbiamo evidenziato in rosso il percorso fatto senza disegnare.

CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90°
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 120
PENUP
FORWARD 50mm/3
LEFT 90
FORWARD 50mm/3
PENDOWN
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90



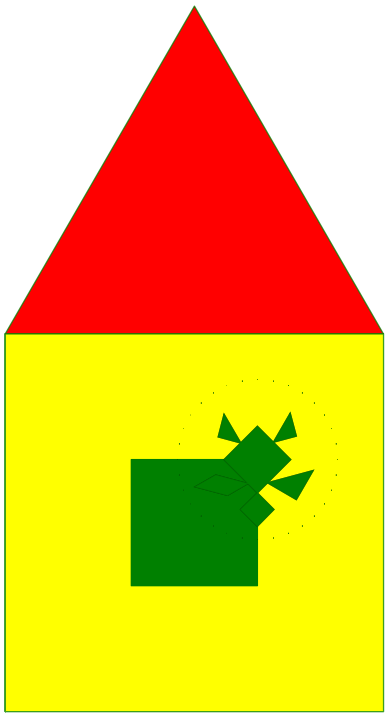
Proviamo ora ad arricchire ulteriormente il disegno, per esempio facendolo a colori.

CLEARSCREEN
HOME
PENCOLOR “green”
FORWARD l RIGHT 90°
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
PENCOLOR “red”
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 120
PENUP
FORWARD 50mm/3
LEFT 90
FORWARD 50mm/3
PENDOWN
PENCOLOR “blue”
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
PENCOLOR “gray”



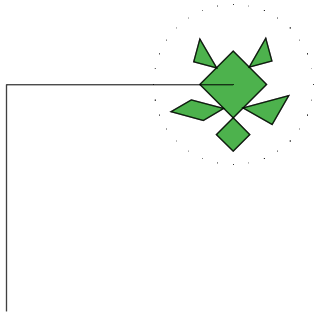
E perché non colorare anche l'interno?

CLEARSCREEN
HOME
FORWARD l RIGHT 90°
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
FILLCOLOR “yellow” FILL
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 120
PENUP
FORWARD 50mm/3
LEFT 90
FORWARD 50mm/3
PENDOWN
FILLCOLOR “red” FILL
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FILLCOLOR “green” FILL



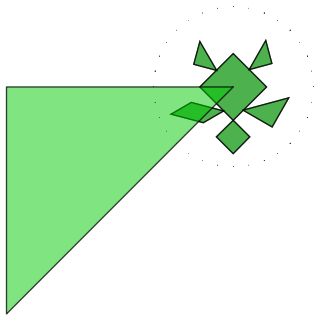
Anche in questo caso siamo ricorsi al raggruppamento delle istruzioni: FILLCOLOR “green” FILL. Con la prima si stabilisce il colore da usare (FILLCOLOR “green”) e con la seconda si procede a colorare la figura – viene naturale inglobarle in una sola istruzione.

L'istruzione FILL in realtà fa due cose: chiude una figura e la riempie di un colore. Facciamo questa prova:



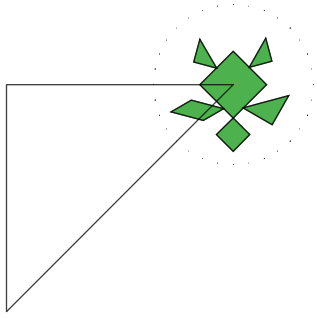
Così abbiamo disegnato una figura che non è chiusa. Volendo fare un triangolo potremmo far disegnare alla tartaruga il terzo lato. Alternativamente possiamo chiudere la figura con FILL, come abbiamo visto prima:

```
FORWARD 30mm RIGHT 90
FORWARD 30mm RIGHT 90
FILL
```



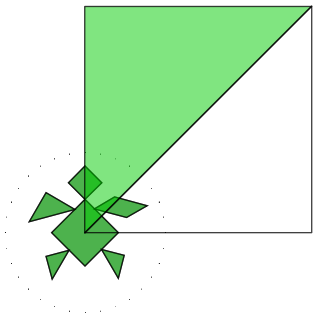
Così la figura è stata chiusa e colorata. Possiamo anche chiuderla senza colorarla, usando l'istruzione **CLOSE**:

```
FORWARD 30mm RIGHT 90
FORWARD 30mm RIGHT 90
CLOSE
```



È interessante notare che, sia con FILL che con CLOSE la figura viene chiusa senza che la tartaruga si sposti. Quindi se continuiamo ad aggiungere altri movimenti, questi prenderanno le mosse da tale posizione della tartaruga, che negli esempi precedenti è rimasta volta verso il basso. Proviamo a mettere i due precedenti blocchi di codice uno dietro l'altro:

```
FORWARD 30mm RIGHT 90
FORWARD 30mm RIGHT 90
FILL
FORWARD 30mm RIGHT 90
FORWARD 30mm RIGHT 90
CLOSE
```

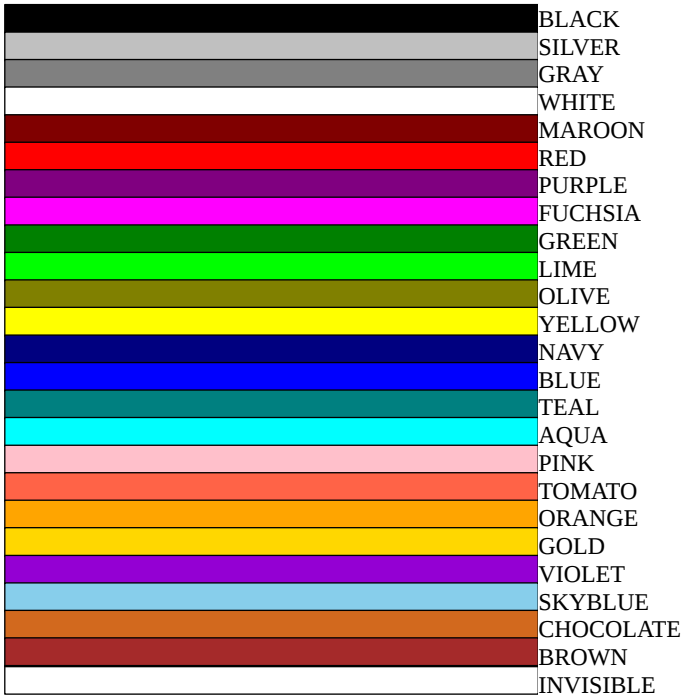


Dopo avere chiuso e colorato di verde il triangolo superiore, la tartaruga con le successive istruzioni di FORWARD procede a disegnare due lati del triangolo inferiore. L'istruzione CLOSE fa chiudere questo secondo triangolo senza colorarlo.

E se uno volesse togliere la tartaruga una volta finito il disegno? Semplice: basta aggiungere alla fine l'istruzione **HIDETURTLE**. Prova!

I codici RGB per i colori

Negli esempi precedenti abbiamo usato dei codici per esprimere i colori, per esempio “red” per indicare rosso. Si possono indicare così 24 colori:



In realtà i colori possibili sono molti di più. Questi possono essere espressi mediante il cosiddetto codice RGB, che sta per Red, Green, Blue. Nella grafica al computer i colori possono essere espressi come combinazione di tre colori fondamentali, che sono appunto rosso, verde e blu. Tutti gli altri possono essere espressi miscelando e dosando opportunamente questi tre colori fondamentali. Per fare questo si esprimono ile intensità dei singoli colori con un numero che va da 0 a 255 e si miscelano scrivendo ad esempio [255,0,0] per il rosso, [0,255,0] per il verde o [0,0,255] per il blu. Tutti gli altri colori si ottengono variando i valori all'interno delle parentesi. Con [0, 0, 0] si ottiene il nero e con [255, 255, 255] il bianco. Poi questo tipo di codifica si utilizza in uno qualsiasi dei comandi che consentono di specificare il colore. Ad esempio

PENCOLOR **[45, 88, 200]**
FILLCOLOR **[255, 200, 100]**

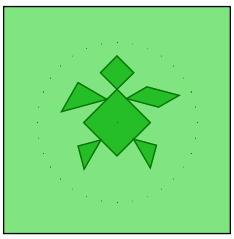
Provate a giocare un po' con questi numeri per vedere che colori vengono fuori.

La tabella dei colori che abbiamo mostrato ha una stranezza: ci sono due elementi che hanno nome diverso ma il colore sembra lo stesso. Trovati? In realtà differiscono. ma per via di un quarto attributo, che è la trasparenza. Vediamo giocando un po'.

Per facilitarci, anticipiamo un nuovo comando. Negli esempi precedenti noi abbiamo già disegnato un quadrato, utilizzando le istruzioni FORWARD e RIGHT, opportunamente combinate. In realtà,

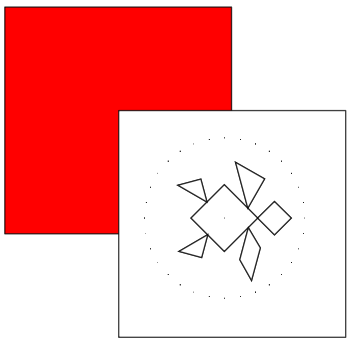
per disegnare le principali figure geometriche, in Logo esistono delle istruzioni preconfezionate, che ci aiutano a scrivere il codice in maniera più sintetica. Il quadrato è una di queste. Per fare un quadrato di 50 mm di lato si scrive:

```
CLEARSCREEN
HOME
SQUARE(30mm)
```



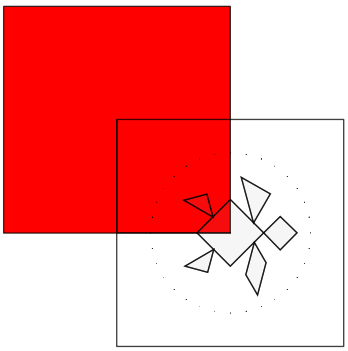
In questo modo la tartaruga disegna un quadrato e poi si piazza al centro con la testa rivolta in alto (il coloro non l'abbiamo determinato noi ma venuto “automaticamente”, si dice che è il *valore di default*). Serviamoci allora di questa istruzione per disegnare prima un quadrato che coloriamo di rosso, poi spostiamoci un po' in basso e a destra, per disegnare un secondo quadrato, in modo che quest'ultimo si sovrapponga un poco al precedente, e poi lo coloriamo di bianco.

```
CLEARSCREEN
HOME
FILLCOLOR [255, 0, 0]
SQUARE(50mm)
FORWARD -25mm
RIGHT 90
FORWARD 25mm
FILLCOLOR [255, 255, 255]
SQUARE(50mm)
```



Bene, così il quadrato bianco si è sovrapposto a quello rosso. Un risultato prevedibile: i colori si sovrappongono perché sono opachi. Un pittore direbbe che “coprono”. In realtà con i codici RGB in Logo si può assegnare un quarto parametro che è la “trasparenza” che si vuole attribuire a quel certo colore, con la convenzione che un valore di 0 gli conferisce la completa opacità mentre il valore di 255 lo rende invece completamente trasparente. Con i valori compresi fra 0 e 255 si possono ottenere diversi livelli di trasparenza. Proviamo dunque a rendere trasparente il quadrato bianco, aggiungendo il quarto parametro uguale a 255:

```
CLEARSCREEN
HOME
FILLCOLOR [255, 0, 0]
SQUARE(50mm)
FORWARD -25mm
RIGHT 90
FORWARD 25mm
FILLCOLOR [255, 255, 255, 255]
SQUARE(50mm)
```



Il quadrato bianco è diventato trasparente. Se al posto di FILLCOLOR [255, 255, 255, 255] avessimo usato FILLCOLOR “invisible” allora avremmo ottenuto lo stesso risultato. Ecco in cosa consiste la differenza fra il colore “white” (bianco) e “invisible”, invisibile, nella tabella che abbiamo visto all'inizio.

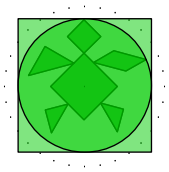
Un'ultima annotazione. A guardare bene, si vede che in questi disegni ci ritroviamo con la tartaruga di colori diversi. Di fatto, la tartaruga viene colorata con lo stesso colore che abbiamo selezionato per riempire l'ultima figura disegnata, magari con una trasparenza diversa. Dopo avere disegnato il quadrato verde chiaro, la tartaruga è venuta verde un po' più scuro. Dopo il quadrato bianco è venuta grigio chiaro e dopo il colore invisibile e venuta grigio scuro. Questo è un comportamento che comunque non inficia i risultati che vogliamo ottenere perché qualora volessimo usare in qualche maniera le grafiche che produciamo, alla fine non avremo da fare altro che aggiungere l'istruzione HIDETURTLE.

Altri comandi

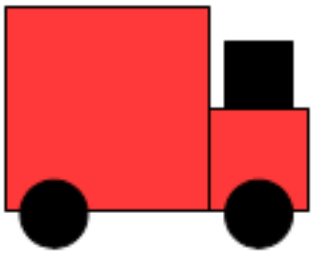
Con i comandi di base che abbiamo visto fin qui è possibile produrre una gran quantità di grafiche. In Logo sono tuttavia disponibili dei comandi che consentono di abbreviare il disegno di alcune forma base. Abbiamo già anticipato l'istruzione SQUARE che consente di costruire un quadrato in un sol colpo. Le altre sono RECTANGLE, CIRCLE e ELLIPSE, che servono a costruire rispettivamente rettangoli, cerchi e ellissi.

Proviamo i seguenti comandi:

```
CLEARSCREEN
HOME
SQUARE 50
CIRCLE 50
```



Si vede come l'effetto di disegnare due figure in sequenza sia quello di sovrapporle facendone coincidere il centro di simmetria e come la tartaruga si riposizioni sempre su tale centro. Si capisce anche come l'argomento[51] del comando SQUARE rappresenti il lato del quadrato e quello del comando CIRCLE il diametro del cerchio che vogliamo costruire.



Prova a esercitarti con i comandi SQUARE e CIRCLE, per esempio costruendo la locomotiva qui accanto. Puoi esercitarti a rispettare delle proporzioni date, come quelle in questo esempio, dove il lato del quadrato più grande è due o tre volte quelli dei quadrati più piccoli e il diametro dei cerchi è pari al lato del quadrato più piccolo. A pagina nuova c'è una possibile soluzione, ma prima prova da solo.

Una cosa importante di cui rendersi conto con il codice è che lo stesso obiettivo può essere conseguito in tanti modi diversi. Non esiste un criterio assoluto per stabilire quale sia il procedimento migliore. Quindi non esiste un'unica “risposta giusta”. Un procedimento può essere meglio di un altro sotto un certo punto di vista: chiarezza del codice scritto, velocità di esecuzione, memoria totale impiegata dal codice, gestione di eventuali risorse ecc. Può succedere che un ottimo codice sotto uno di questi punti vista risulti pessimo sotto un altro.

51 Con “argomento” si intende il valore che un comando richiede per essere effettuato. Gli argomenti possono essere anche più di uno – vedremo degli esempi.

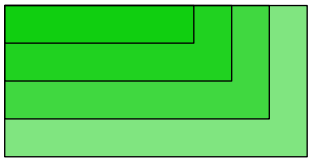
```
COLOR “red”
SQUARE 60
PENUP BACK 15 RIGHT 90 FORWARD 45
LEFT 90 PENDOWN
SQUARE 30
PENUP FORWARD 25 PENDOWN
FILLCOLOR “black”
SQUARE 20
PENUP BACK 40 PENDOWN
CIRCLE 20
PENUP LEFT 90 FORWARD 60 PENDOWN
CIRCLE 20
HIDETURTLE
```

Il quadrato e il cerchio hanno bisogno di un solo parametro per essere specificati. Invece, il rettangolo e l'ellisse hanno bisogno di due parametri. Nel caso del rettangolo i due parametri sono le lunghezze del lato lungo e del lato corto. L'istruzione per costruire un rettangolo è la seguente:

RECTANGLE [60,40]

Questo comando produce un rettangolo largo 60 punti e altro 40. Rispetto ai casi del quadrato de del cerchio questo comando ha un'altra particolarità: in questo caso, per fornire i due parametri necessari, si è ricorsi alla scrittura **[60,40]**. Questa è una “lista” di valori. Si tratta di un modo per considerare un insieme di valori come un'unica cosa, una lista appunto, e il modo per ottenere questo effetto è quello di elencare i valori separandoli con le virgole e richiudendo tutto fra parentesi quadre. Le liste possono servire in varie circostanze, non solo in questo caso, ma vedremo successivamente come.

Esercizio: prova a realizzare un diagramma come questo qui sotto:



Anche in questo caso, prova da solo. Poi, a pagina nuova puoi vedere un possibile modo di risolvere questo caso.

```
RECTANGLE [40mm, 20mm]
PENUP FORWARD 2,5mm LEFT 90
FORWARD 2,5mm RIGHT 90 PENDOWN
RECTANGLE [35mm, 15mm]
PENUP FORWARD 2,5mm LEFT 90
FORWARD 2,5mm RIGHT 90 PENDOWN
RECTANGLE [30mm, 10mm]
PENUP FORWARD 2,5mm LEFT 90
FORWARD 2,5mm RIGHT 90 PENDOWN
RECTANGLE [25mm, 5mm]
HIDETURTLE
```

Un'altra figura che può tornare utile è l'ellisse. Detto nel più semplice dei modi l'ellisse è un cerchio schiacciato. È perfettamente nello spirito della geometria della Tartaruga quello di aiutarsi con riferimenti ad attività fisiche: probabilmente il miglior modo di usare le tecnologie moderne è di continuare ad utilizzare quelle tradizionali, integrando al meglio le une con le altre. Niente di meglio qui che farsi aiutare da Emma Castelnuovo[52], in un suo brano dove l'ellisse emerge studiando la classe di triangoli isoperimetrici con uguale base. Riportiamo il brano per intero, al fine di rispettare l'intento didattico di Emma, che è di valore:

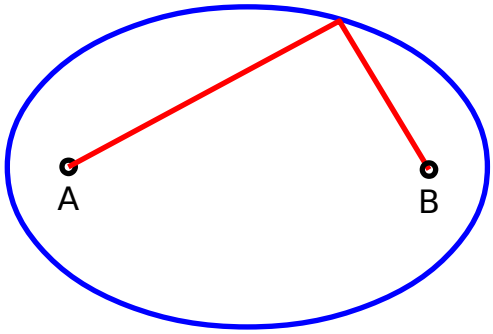
> Sempre un argomento di matematica, quale lo studio dei triangoli isoperimetrici con ugual base, porta a osservare quello che abbiamo sotto gli occhi.
>
> Il materiale è, anche questa volta, un pezzo di spago.
>
> Per costruire dei triangoli di uguale base e uguale perimetro facciamo così: fissiamo due chiodi – siano A e B – su un tavolo su cui è disteso un foglio di carta; AB sarà la base dei nostri triangoli. Leghiamo poi gli estremi di un pezzo di spago ai due chiodi, tenendo presente che lo spago deve essere più lungo del tratto AB. Facciamo in modo, valendoci di una matita, che lo spago resti sempre ben teso e... lasciamoci guidare dalla matita.
>
> Questa, guidata dallo spago, disegnerà sul foglio una curva a forma di ovale: è un'ellisse. I punti A e B si chiamano fuochi dell'ellisse. Dunque: i vertici dei triangoli isoperimetrici e di uguale base si trovano su un'ellisse.
>
> Un problema di geometria ci ha condotti al disegno dell'ellisse. Con lo stesso pezzo di spago possiamo costruire un'ellisse più o meno “schiacciata”, a seconda della distanza fra i punti A e B. Si può ottenere anche un cerchio, se i due punti coincidono: il cerchio, infatti, è un'ellisse particolare.
>
> L'ellisse, dopo averla incontrata in problemi di geometria, la ritroviamo per la strada, quando la “calpestiamo” (perché un disco segnaletico dà, come ombra, un'ellisse. Nella nostra vita convulsa raramente ci soffermiamo a osservare l'ombra di un oggetto data dai raggi del soleo da una lampadina. Ma ecco che ora un'attività di geometria ci sollecita a guardare di più ed è proprio il confronto fra l'effetto-ombra dato dai raggi del sole e quello dato da una lampada puntiforme che stimola la nostra facoltà di osservazione.

52 E. Castelnuovo, *L'officina matematica – ragionare con i materiali*, p. 20-21, Edizioni la Meridiana, 2008.

Guardiamo, ad esempio, due matite disposte in verticale su un tavolo. Se vengono illuminate dal sole accade che anche le ombre sono parallele; se invece è una lampada che le illumina le ombre si divaricano.

Da qui lo studio matematico delle trasformazioni affini e delle trasformazioni proiettive, fino ad arrivare alla prospettiva, all'arte, a come si guarda un quadro, alla storia.

È un piccolo problema di geometria che ha stimolato a osservare e a... guardarsi intorno.



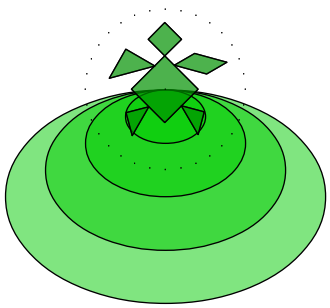
In LibreLogo l'ellisse si disegna con il comando

ELLIPSE [60, 40]

L'ellisse ovviamente non ha un diametro fisso, come il cerchio. Per questo per essere definita richiede due parametri che sono i due assi, maggiore e minore. Nell'esempio l'ellisse ha l'asse maggiore uguale a 60 punti e il minore uguale a 40 punti. È possibile combinare le varie forme e aggiustare i loro parametri per ottenere una varietà di effetti. Per esempio un cerchio inscritto in un quadrato si può ottenere anche partendo dalle istruzioni per disegnare rettangoli e ellissi:

RECTANGLE [60, 60]
ELLIPSE [60, 60]

Prova a fare una figura come questa:

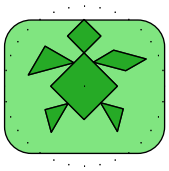


Risposta possibile:

```
ELLIPSE [120, 80]
PENUP FORWARD 10 PENDOWN
ELLIPSE [90, 60]
PENUP FORWARD 10 PENDOWN
ELLIPSE [60, 40]
PENUP FORWARD 10 PENDOWN
ELLIPSE [30, 20]
PENUP FORWARD 10 PENDOWN
```

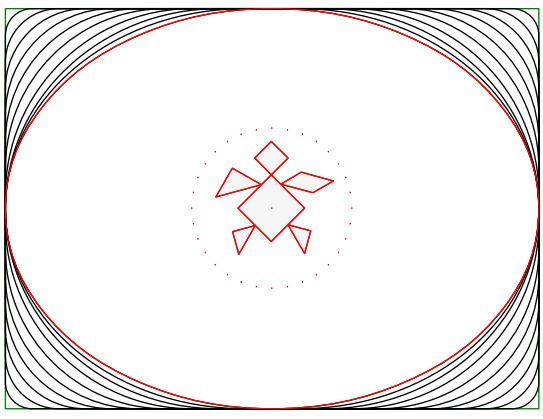
Queste istruzioni consentono di usare anche altri parametro per ottenere delle varianti di rettangoli e ellissi. Nel caso dei rettangoli è possibile aggiustare un terzo parametro in maniera da arrotondare i vertici:

RECTANGLE [60, 50, **10**]



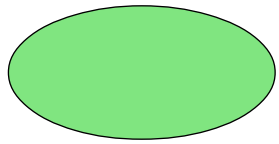
Supponiamo ora che un amico ci abbia appena detto di questa possibilità ma che altro non si ricordi. Si può controllare la “rotondità” dei vertici? Probabilmente con quel terzo parametro, che l'amico ci ha detto potevamo fissare al valore di 10, ma come funziona? Questo è un ottimo esempio per mettere in luce lo strumento fondamentale di chi sviluppa software: la sperimentazione. Ecco, in questo caso, potremmo provare a fare dei tentativi, che potremmo (è solo una delle innumerevoli possibilità) sintetizzare così:

```
FILLCOLOR “invisible”
PENCOLOR “green”
RECTANGLE [200, 150, 0]
PENCOLOR “black”
RECTANGLE [200, 150, 10]
RECTANGLE [200, 150, 20]
RECTANGLE [200, 150, 30]
RECTANGLE [200, 150, 40]
RECTANGLE [200, 150, 50]
RECTANGLE [200, 150, 60]
RECTANGLE [200, 150, 70]
RECTANGLE [200, 150, 80]
RECTANGLE [200, 150, 90]
RECTANGLE [200, 150, 100]
PENCOLOR “red”
ELLIPSE [200,150]
```



In questo modo ci siamo fatti un'idea di come possiamo controllare i rettangoli arrotondati. Ci siamo anche resi conto che, giocando con il terzo parametro, possiamo andare dal caso limite del rettangolo normale all'ellisse vera e propria.

Vediamo ora le varianti possibili per l'istruzione ELLIPSE.



ELLIPSE [100, 50]

ELLIPSE [100, 50, 0, 90, 1]

ELLIPSE [100, 50, 0, 90, 3]

ELLIPSE [100, 50, 0, 90, 2]

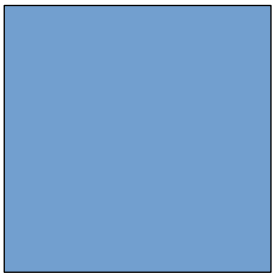
Nell'istruzione ELLIPSE possiamo quindi utilizzare 3 parametri aggiuntivi, che servono a disegnare solo un settore dell'ellisse. I primi due rappresentano l'angolo iniziale e finale, espressi in gradi, che delimitano il settore. Nell'esempio sopra, avendo scelto 0 e 90 abbiamo imposto di disegnare il primo quadrante dell'ellisse. Il quarto parametro stabilisce se si vuole disegnare un settore di ellisse (1), un segmento di ellisse (2) oppure giusto un arco (3).

Le variabili

Le istruzioni che abbiamo visto sin'ora consentono di fare molte cose: abbiamo imparato a

muovere la tartaruga in ogni parte del foglio, a farla disegnare o meno, abbiamo visto come controllare il colore del tratto e il riempimento delle figure. Si potrebbe avere la sensazione che per produrre grafica non occorra altro. Invece abbiamo solo scalfito la superficie delle potenzialità di un linguaggio di programmazione, anche se questo ha finalità esclusivamente didattiche, come nel caso di Logo e dei suoi derivati. Introdurremo via via le caratteristiche più importanti. Fra queste la prima, che ci serve subito per poter andare avanti, è il concetto di “variabile”. Fin qui abbiamo usato varie istruzioni che richiedono degli “argomenti”. L'argomento è il valore che un'istruzione può richiedere per poter essere eseguita. Ad esempio l'istruzione FORWARD non avrebbe senso senza un argomento che rappresenti la distanza che la tartaruga deve percorrere. L'espressione FORWARD 50 significa che la tartaruga deve muoversi in avanti di 50 punti; 50 è il valore dell'argomento. Vi sono anche istruzioni che richiedono più di un argomento, è per esempio il caso di RECTANGLE e ELLIPSE. In ogni caso, in tutti gli esempi visti precedentemente abbiamo sempre usato argomenti numerici per tutte le istruzioni. In realtà, tutti i linguaggi consentono di servirsi di un'importante generalizzazione che consiste nell'uso delle “variabili”. Si tratta di nomi simbolici ai quali possono essere assegnati vali numerici a piacimento. Proviamo ad eseguire il seguente codice:

```
CLEARSCREEN
HOME

LATO = 100
ANGOLO = 90

FORWARD LATO
LEFT ANGOLO
FORWARD LATO
LEFT ANGOLO
FORWARD LATO
LEFT ANGOLO
FORWARD LATO
HIDETURTLE
```



Abbiamo disegnato un quadrato, ma invece di utilizzare direttamente il valore 100 come argomento delle istruzioni FORWARD, abbiamo prima assegnato il valore 100 alla variabile di nome “LATO” e poi abbiamo utilizzato questa come argomento di tutte le istruzioni FORWARD. È evidente quale possa essere l'utilità di questo metodo: supponiamo che non sia soddisfatto delle dimensioni di questo quadrato e che voglia provare altri valori del lato. Ebbene, non ho altro che da cambiare il valore 100 nell'istruzione LATO = 100, cambiandola per esempio con LATO = 150. Prova a sperimentare! Puoi anche cambiare il valore di ANGOLO, sperimentando con valori diversi...

Coloro che hanno studiato i primi rudimenti dell'algebra, avranno certamente riconosciuto il concetto di variabile, che in quella disciplina si utilizza ampiamente per eseguire calcoli simbolici. Si ricorderanno anche che il concetto di variabile si declina in vari modi, per esprimere quantità che si considerano effettivamente variabili – ad esempio una variabile dipendente in funzione di altre variabili indipendenti – poi quantità che si assumono costanti, infine quantità che assumono il significato di parametri, che sono come costanti che possiamo avere interesse a cambiare di tanto in

tanto. In ogni caso tutte queste quantità vengono rappresentate in maniera simbolica. In realtà, coloro che hanno poi avuto modo di approfondire lo studio dell'algebra, sanno che il concetto di variabile è passibile di tutta una serie di generalizzazioni. Niente paura, questo non è un corso di matematica sottobanco, o forse un po' sì: in fin dei conti Logo rappresenta l'anelito di Seymour Papert di rendere la matematica più accessibile. Ma ciò che proponiamo qui non richiede doti o attitudini particolari. Introduciamo solo una delle generalizzazioni possibili, che ci servirà immediatamente. La generalizzazione che proponiamo attiene al concetto di posizione della tartaruga nel foglio. La posizione lungo una linea è determinata da un semplice numero – ad esempio la posizione lunga una strada: “si segnalano lavori in corso al Km 287...”. Diverso è il caso della posizione su di una superficie. In un navigatore satellitare, che oggi tutti conoscono, si può dare anche la posizione in termini geografici, ma questa deve essere somministrata mediante due valori: la latitudine e la longitudine, che designano il parallelo terrestre la prima e il meridiano la seconda. Per affondare una nave nel gioco della battagli navale occorre fornire due coordinate, per esempio b7, dove “b rappresenta la colonna e 7 la riga. Allo stesso modo si identificano le celle di un foglio di lavoro, e via dicendo. Anche alla nostra tartaruga occorrono due valori numerici per identificare una posizione precisa nel foglio, che possiamo immaginare come la x e la y della tartaruga nello spazio della pagina. Ebbene, il modo per esprimere questo concetto nel mondo della tartaruga (ma non solo!) è il seguente:

```
P = [200, 300]
PRINT P
```

Se si esegue questo codice, LibreLogo stampa il “valore” [200, 300]. Ovviamente abbiamo scelto questi numeri in maniera del tutto arbitraria, giusto per fare un esempio. Lo scopo è quello di mostrare come si rappresenta in maniera simbolica una posizione, che in realtà è espressa mediante un insieme di due numeri. Nell'algebra si dice che questo tipo di variabile è un “vettore”. C'è anche un modo per “isolare” i singoli elementi all'interno di un vettore. La cosa si descrive subito con questo esempio:

```
P = [200, 300]
PRINT P
PRINT P [0]
PRINT P [1]
```

Se si esegue questo frammento di codice, prima viene stampato “valore” [200, 300], poi il valore 200, quindi 300. Da qui si capisce che con P [0] si ottiene il primo elemento del vettore posizione, che contiene il numero 200, e con P [1] il secondo elemento, che contiene il numero 300.

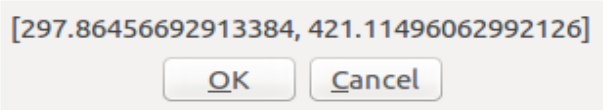
Ecco, questo è quanto dovrebbe bastare per andare a vedere com'è che si può controllare con ancora maggiore agilità la posizione della tartaruga nel foglio.

Lo spazio della pagina

Abbiamo già visto vari comandi per muovere la tartaruga ma sono tutti finalizzati al disegno. È vero che si possono fare i movimenti con la "penna alzata" (comando PENUP) ma può essere utile "saltare" direttamente in una posizione qualsiasi del foglio, oppure puntare in una direzione precisa. Si tratta, in altre parole, di scegliere una posizione o una direzione in termini assoluti e non in modo relativo, rispetto alla posizione e direzione corrente, come si fa per esempio con istruzioni del tipo FORWARD oppure LEFT. Qui sorge la necessità di utilizzare dei riferimenti spaziali assoluti che sono una coppia di coordinate per la posizione nel foglio e un angolo per la direzione. Per sapere come funzionano tali riferimenti introduciamo e usiamo subito due nuove istruzioni: **POSITION** e **HEADING**. Inoltre, ci è utile anche l'istruzione **PRINT**, per conoscere il valore corrente della posizione e della direzione. Infatti i due comandi **POSITION** e **HEADING**, si possono usare con e senza parametri. Quando si usano senza parametri allora questi forniscono i valori correnti. Infatti, se apro un nuovo documento in Writer e faccio eseguire il comando

PRINT POSITION

si ottiene la seguente risposta:

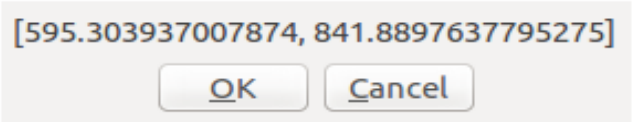


I due numeri fra parentesi rappresentano le coordinate *x* e *y* della posizione nello spazio della pagina: 298 e 421[53] rispettivamente. Dal momento che abbiamo appena aperto il documento e che all'inizio la tartaruga viene piazzata al centro, possiamo assumere che queste coordinate rappresentino il centro della pagina. Tuttavia, per avere il controllo completo della situazione occorre conoscere precisamente l'estensione dello spazio dell'immagine. Ebbene, le coordinate dell'angolo superiore sinistro sono [0, 0], dove il primo numero rappresenta la coordinata *x* e il secondo la *y*, mentre quelle dell'angolo inferiore destro si ottengono stampando il valore della variabile speciale **PAGESIZE**, che LibreLogo utilizza per conservare le dimensioni della pagina. In questo momento la mia versione di Writer è predisposta per pagine di dimensioni A4 e, conseguentemente, eseguendo l'istruzione

53 Abbiamo arrotondato i due numeri a quattro cifre significative, che sono adeguate a determinare la posizione nel foglio, ai nostri fini. Nella nota successiva diamo una breve spiegazione dell'unità di misura impiegata per questi numeri.

PRINT PAGESIZE

otteniamo:



Questi numeri, arrotondati, 596 e 842, rappresentano le dimensioni di un foglio A4 espressi in "punti" alla densità di 72 DPI[54]. In mm risultano 210 e 297 mm. Riassumiamo la situazione con la seguente figura.

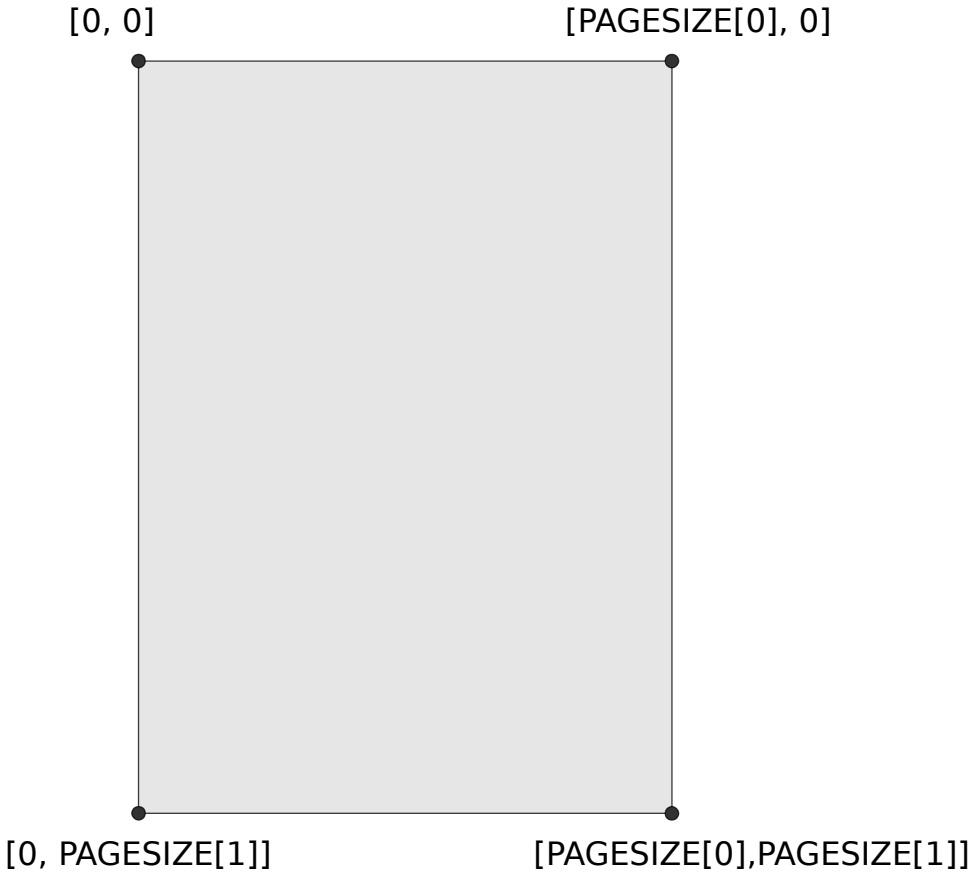


Figura 3: (in appendice trovi il codice per produrre questa immagine)

Tutte le volte che vogliamo utilizzare le posizioni assolute possiamo fare riferimento a questo schema che ci fornisce l'orientamento del sistema di riferimento sul foglio e le sue dimensioni.

54 L'acronimo DPI sta per dots per inch: punti per pollice. Il valore di DPI dipende dal supporto fisico su cui si intende che un'immagine debba essere rappresentata. Poiché un pollice vale 2.54 cm, la densità di 72 DPI corrisponde a 72/2.54 = 28.3 punti per cm, o 2.83 punti per mm; giusto per avere un riferimento a noi più famigliare. Quando in Writer si sceglie l'unità di misura "punti" (anziché cm o pollici), questi si riferiscono alla densità di 72 DPI appena citata. In Writer, l'unità di misura si può cambiare con la voce di menu Tools->Options->LibreOffice Writer->General. Si può scegliere fra mm, cm, pollici, pica, punti.

Nella tabella successiva vediamo i valori numerici corrispondenti:

[0, 0]	[0, 0]
[PAGESIZE[0], 0]	[596, 0]
[0, PAGESIZE[1]]	[0, 842]
[PAGESIZE[0], PAGESIZE[1]]	[596, 842]

Riallacciandosi al paragrafo precedente, PAGESIZE è una variabile, una variabile che all'interno di LibreLogo è trattata come una costante perché contiene le dimensioni della pagina. Inoltre è quel particolare tipo di variabile che si chiama vettore, perché composta da più elementi, precisamente due, le due dimensioni della pagina: PAGESIZE[0] è la larghezza e PAGESIZE[1] la lunghezza.

Come facciamo dunque per spedire la tartaruga in una posizione precisa?

```
CLEARSCREEN
HOME
POSITION [350, 320]
```

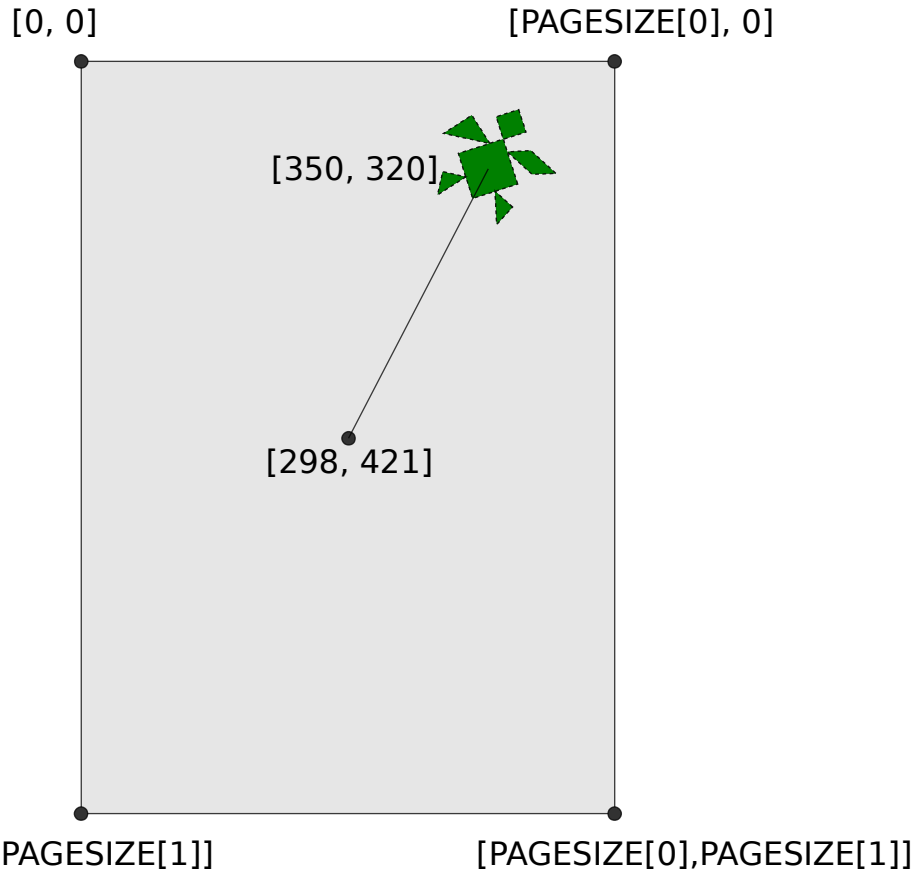


Figura 4: in appendice trovi il codice per produrre questa immagine

Con l'istruzione **POSITION [350, 320]** la tartaruga si dirige direttamente al punto di coordinate **[350,320]**, a partire dal punto dive si trova, in questo caso dal centro della pagina.

Come possiamo usare l'istruzione POSITION per controllare la posizione, in modo analogo possiamo utilizzare l'istruzione HEADING per controllare la direzione in cui punta la tartaruga.

Invocando l'istruzione **HEADING** senza alcun parametro si ottiene la posizione corrente. Utilizzando invece un parametro, per esempio HEADING [30], si impone alla tartaruga di ruotare di 30°. Nella figura seguente si mostra l'orientamento del sistema di riferimento.

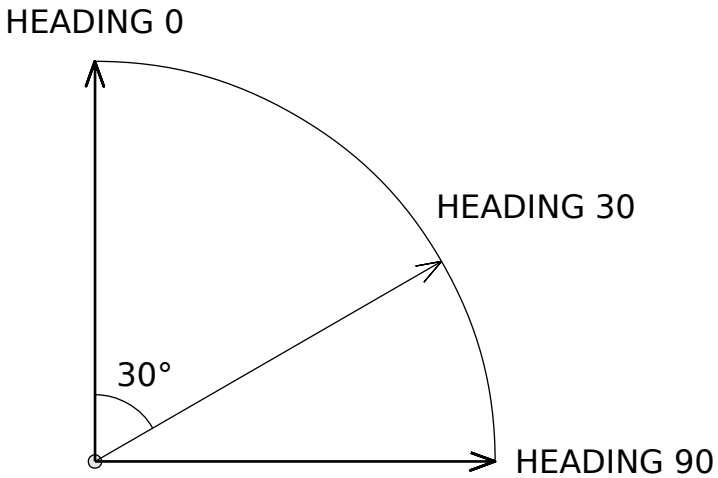


Figura 5: in appendice trovi il codice per produrre questa immagine

Quando si apre un documento nuovo, oppure dopo l'istruzione HOME, la tartaruga punta verso il lato superiore del foglio, e questa direzione corrisponde a 0°.

Altri comandi grafici

È possibile controllare anche altri aspetti del disegno, oltre al colore.

PENWIDTH (spessore tratto)

Con il comando **PENWIDTH** si determina lo spessore del tratto:

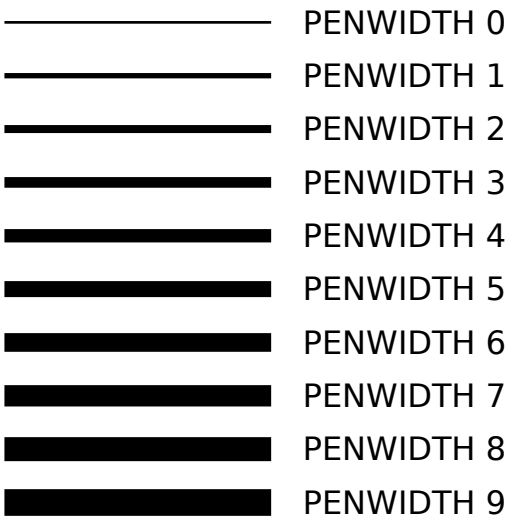


Figura 6: Esempi di controllo dello spessore del tratt; codice in appendice.

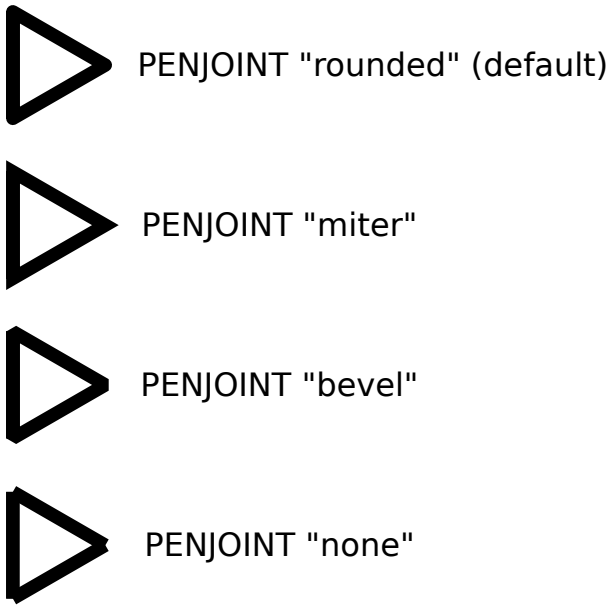
PENJOINT (forma vertici)

Con il comando **PENJOINT** si controlla la forma dei vertici.

Disegnamo per esempio un triangolo, nel modo seguente:

```
PENWIDTH 5
FORWARD 40 RIGHT 120
FORWARD 40 RIGHT 120
FORWARD 40 RIGHT 120
HIDETURTLE
```

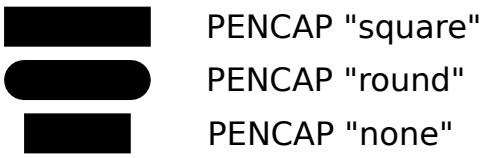
Possiamo alterare la rifinitura dei vertici facendo precedere il codice precedente dall'istruzione PENJOINT e un opportuno argomento. Le possibilità sono le seguenti:



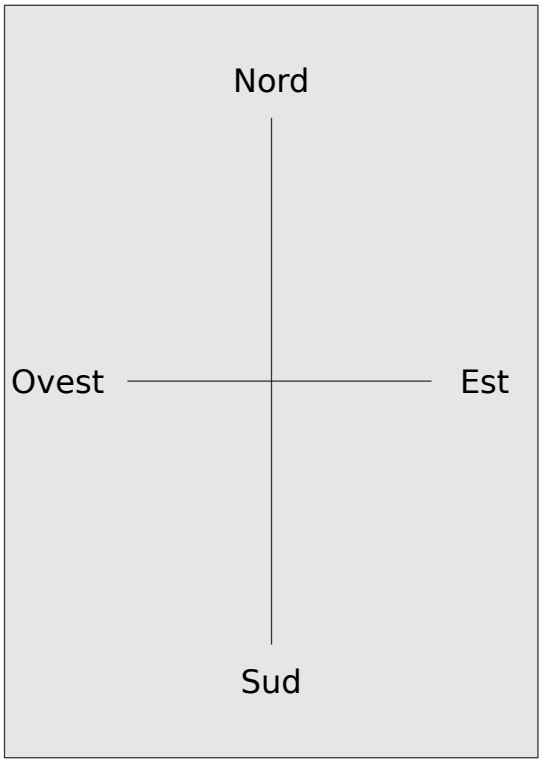
L'argomento “rounded” significa arrotondato – (default) non va scritto nel comando: lo abbiamo aggiunto per dire che quello è il comportamento standard se non si specifica nulla. Se tuttavia si è appena utilizzata una delle altre opzioni, allora va usato esplicitamente il comando **PENJOINT “rounded”**, per ottenere i vertici arrotondati. L'argomento “miter” significa “mitria” e sta peer “giunto a mitria”, o “giunto a quartabono”, che è quello che si realizza nelle cornici dei quadri tagliano i singoli regoli della cornice in maniera che i vertici risultino a punta. “bevel” significa vertici smussati.

PENCAP (forma estremità segmenti)

Con questo comando si possono controllare gli estremi di un segmento:



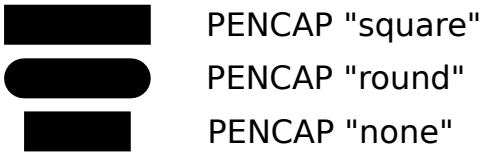
Per commentare il comportamento di questo comando, e anche per ricapitolare un po' la scrittura del codice in Logo, analizziamo il codice che è servito a produrre questa figura, omettendo, per semplicità, la parte che produce le scritte a destra. Per capire più facilmente, orientiamoci mediante i punti cardinali, così orientati:



Vediamo quindi il codice seguente:

```
CLEARSCREEN             ; cancello il foglio (solo la parte grafica)
HOME                    ; partenza: tartaruga al centro che punta a Nord
HIDETURTLE              ; nascondo la tartaruga
PENWIDTH 15             ; imposto lo spessore della linea a 15 pt

RIGHT 90                ; ruoto 90° a destra così la tartaruga punta a Est in modo
                        ; da tracciare da sinistra a destra

PENCAP “square”         ; imposto il modo “estremità quadrate”
FORWARD 40              ; disegno 40 pt di linea (Ovest → Est)
PENUP                   ; alzo la penna
RIGHT 90 FORWARD 20     ; giro a destra di 90° (punto a Sud) e calo di 20 pt
LEFT 90 BACK 40         ; rigiro a sinistra (punto a Est) e torno indietro ( Est → Ovest) di 40 pt
PENDOWN                 ; abbasso la penna

PENCAP “round”          ; imposto il modo “estremità arrotondate”
FORWARD 40              ; disegno 40 pt di linea (Ovest → Est)
PENUP                   ; alzo la penna
RIGHT 90 FORWARD 20     ; giro a destra di 90° (punto a Sud) e calo di 20 pt
LEFT 90 BACK 40         ; rigiro a sinistra (punto a Est), torno indietro ( Est → Ovest) di 40 pt
PENDOWN                 ; alzo la penna

PENCAP “none”           ; imposto il modo “estremità arrotondate”
FORWARD 40              ; disegno 40 pt di linea (Ovest → Est)
PENUP                   ; alzo la penna
```

Abbiamo evidenziato le istruzioni che realizzano i tre tratti, che riportiamo ancora qui:

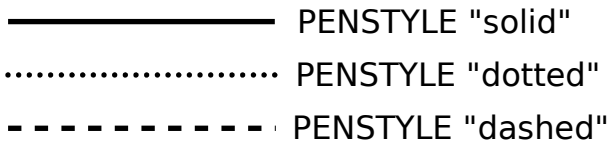


Ecco, si vede che in realtà sono stati disegnati tutti e tre con la stessa lunghezza di 40 pt e invece non sembra che siano venuti uguali. Questo per capire come funziona il comando PENCAP. Il tratto disegnato con assenza di specifiche per le estremità (“effetto none”) è lungo 40 pt. Quindi quelli con gli effetti “round” e “square” vengono più lunghi. Si capisce da qui che gli arrotondamenti sono aggiunti alla lunghezza normale e che l'effetto “square” è ottenuto risquadrando gli arrotondamenti.

Va da sè che si tratta di un aspetto marginale. Abbiamo colto l'occasione per mettere in luce la notevole raffinatezza di LibreLogo, per abituarci ulteriormente a muoversi nel foglio e pensare graficamente.

PENSTYLE (tratteggio segmenti)

Con questa istruzione si può determinare la continuità del tracciamento, per produrre linee tratteggiate di vario tipo:



È anche possibile regolare in qualsiasi modo il tratteggio. Per esempio con l'istruzione

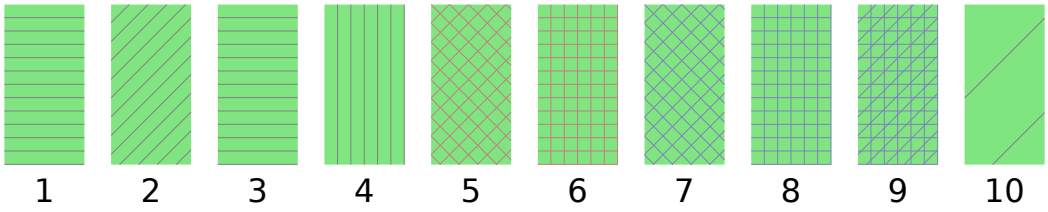
PENSTYLE [3, 1mm, 2, 4mm, 1mm] si ottiene il tratteggio:

Queste sono le regole:

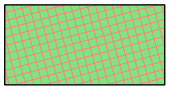
Parametro 1:	numero di punti
Parametro 2:	lunghezza dei punti
Parametro 3:	numero di tratti
Parametro 4:	lunghezza dei tratti
Parametro 5:	lunghezza spazi
Parametro 6:	opzionale, se vale 2 allora i rettangoli sono forzati a quadrati

FILLSTYLE (tratteggio superfici)

L'istruzione FILLSTYLE 1, precedente al disegno di una figura, causa il tratteggio della medesima. Il parametro numerico determina lo stile del tratteggio, nel modo seguente:



Anche qui, è possibile personalizzare lo schema del tratteggio, utilizzando dei parametri aggiuntivi:

FILLSTYLE [2, “red”, 3pt, 15°] 

dove il significato dei singoli parametri è:

Parametro 1:	stile (1: tratteggio singolo, 2 doppio, 3 triplo)
Parametro 2:	colore
Parametro 3:	distanza
Parametro 4:	angolo

Conclusione capitolo 1

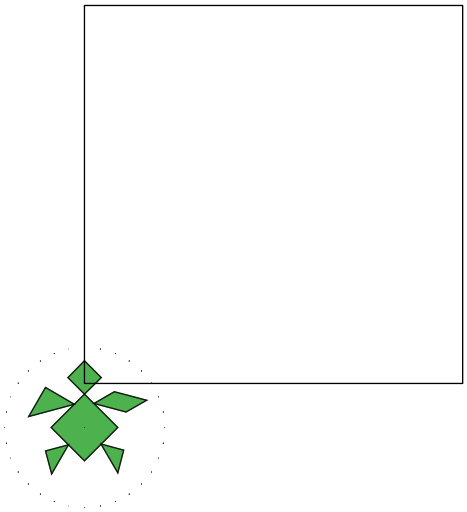
Con queste istruzioni si conclude la prima parte della nostra esplorazione dei LibreLogo. Abbiamo imparato i comandi fondamentali per muovere la tartaruga nello spazio del foglio, sia per disegnare che per muoversi semplicemente. Abbiamo imparato a muoversi come farebbe una tartaruga vera, o come facciamo noi in città, seguendo un percorso continuo, immaginando un “avanti” di fronte alla nostra posizione corrente, un indietro, una destra e una sinistra. Ma abbiamo anche visto come fare a muoversi con il “teletrasporto”, come se inserendo le coordinate nel navigatore satellitare l'automobile ci portasse istantaneamente in quel luogo, senza dover seguire tutto un percorso sulla superficie terrestre; o come si muove il cavallo negli scacchi, saltando direttamente a una casella distante 1+2 o 2+1 posizioni. Abbiamo visto come si codificano i colori per decorare sia le linee che le superfici. Con tutto questo ci è sembrato di potere fare ormai disegnare tutto sul foglio, ma poi abbiamo subito scoperto delle istruzioni per disegnare direttamente le principali figure geometriche: quadrati, rettangoli, cerchi e ellissi, con alcune varianti, come settori, segmenti e archi di ellissi (o cerchi), oppure rettangoli con i vertici arrotondanti. Quindi abbiamo introdotto il primo degli elementi fondamentali che caratterizzano un vero e proprio linguaggio di programmazione: il concetto di variabile, con il quale possiamo usare dei simboli letterari generici per designare specifiche quantità – distanze, angoli e altro – senza doversi preoccupare di assegnare loro precisi valori numerici; una caratteristica fondamentale che che conferisce al linguaggio una potenzialità simile a quella che sta alla base dell'algebra, con il calcolo simbolico. Non abbiamo esplorato tutta la potenzialità che soggiace al concetto di variabile, ma solo evidenziato quanto ci bastava per illustrare i movimenti nel foglio basati sull'impiego delle coordinate spaziali. Per fare questo abbiamo dovuto considerare un particolare tipo di variabile che è il vettore, composto a sua volta da più numeri – due se si tratta di un “vettore posizione” su di una superficie. Infine abbiamo visto che si possono usare altri comandi ancora per determinare i particolari grafici sia del tratto (colore, spessore, estremità, giunzioni) che delle superfici (colore, tratteggi). A questo punto, nuovamente, ci può parere di avere a disposizione di uno strumento potente per produrre grafica da inserire nei documenti. E effettivamente si possono fare senza dubbio tantissime cose con i comandi che abbiamo visto fino ad ora. In realtà, dal punto di vista del linguaggio di programmazione, abbiamo solo visto una piccola parte delle sue potenzialità. Si tratta di costrutti che stanno alla base di tutti i linguaggi di programmazione e che conferiscono le straordinarie capacità di flessibilità e generalizzazione agli innumerevoli tipi di software che tutti conosciamo ma soprattutto usiamo inconsapevolmente in ormai qualsiasi congegno.

Cap. 4: Ripetere

Cicli - Loops

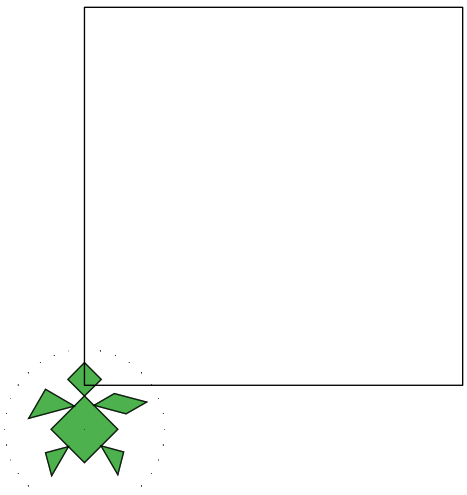
Introduciamo i cicli attraverso un piccolo studio geometrico. Riprendiamo il disegno di un quadrato, così come l'avevamo fatto a pag. 3[55].

```
CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
```



Modifichiamo questo codice usando le variabili, che avevamo visto a pag. 17.

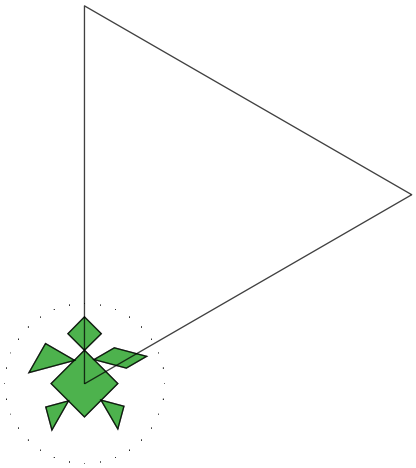
```
CLEARSCREEN
HOME
L = 50mm      ; lato del quadrato
A = 90        ; angolo interno ad ogni vertice
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
```



55L'unica differenza rispetto al quadrato disegnato a pag. 3 è che qui, dopo avere disegnato il quarto lato, quello in basso, abbiamo lasciato la tartaruga lì, nel vertice in basso a sinistra, senza mandarla in alto, cosa che avevamo fatto a pag. 3 per poter iniziare il disegno del tetto della casa.

È facile modificare questo codice per disegnare altre figure, e in particolare per disegnare poligoni regolari. Proviamo ad esempio a costruire un triangolo equilatero. Come si potrebbe fare? Facile: si toglie un lato e si aggiusta la dimensione degli angoli interni. Per fare il quadrato la tartaruga doveva girare nello stesso verso quattro volte di un angolo di 90°, per un totale di 360°. Infatti dopo avere costruito, il quadrato la tartaruga punta nuovamente nella direzione iniziale: segno che ha fatto un giro completo, ovvero che ha ruotato complessivamente di 360°. La stessa cosa dovrà accadere con qualsiasi altra figura geometrica chiusa, quindi anche con un triangolo. Siccome abbiamo deciso di costruire poligoni regolari, tutti gli angoli interni dovranno essere uguali. E poiché un triangolo ha tre angoli interni, ciascuno di questi misurerà 120° = 360°/3:

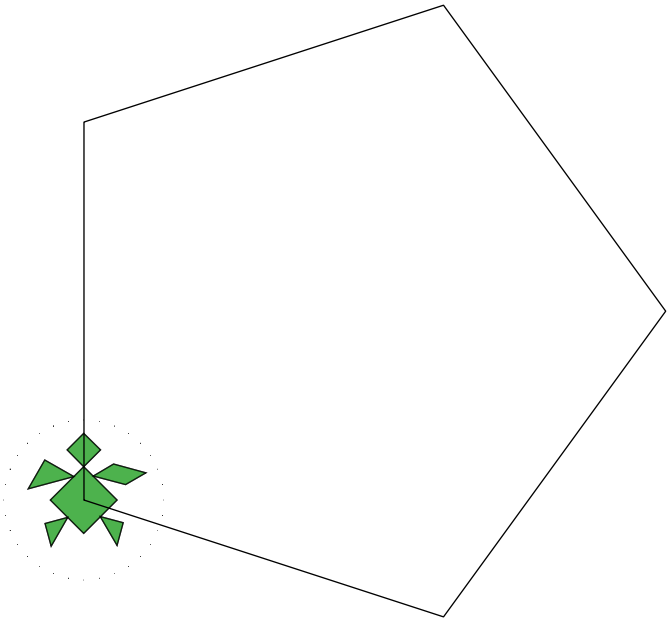
```
CLEARSCREEN
HOME
L = 50mm       ; lato del quadrato
A = 120        ; angolo interno ad ogni vertice
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
```



Proviamo invece ad aggiungere un lato. Invece di togliere dovremmo aggiungere un'istruzione **FORWARD L RIGHT A** alla versione che produce il quadrato, mentre per l'angolo dobbiamo calcolare 72° = 360°/5 e inserire questo valore nel codice. Ma fermiamoci un attimo a riflettere. Stiamo imparando a manovrare una “macchina” alla quale possiamo dare istruzioni affinché faccia delle cose al posto nostro. Perché quindi dobbiamo fare un calcolo per dare i dati necessari? Non potrebbe fare tutti i calcoli lei? Non potremmo cambiare le cose in modo da darle le informazioni necessarie in modo più intuitivo? Lascio lo spazio qui sotto vuoto, se, tu lettore, vuoi rifletterci, prendere un appunto o provare da solo addirittura. Poi, quando vuoi, puoi passare alla pagina successiva.

Procediamo allora introducendo la variabile **N** per designare il numero di lati e poi facciamo calcolare al programma il valore dell'angolo:

```
CLEARSCREEN
HOME
L = 50mm       ; lato del poligono
N = 5          ; Numero di lati
A = 360/N      ; angolo interno
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
```

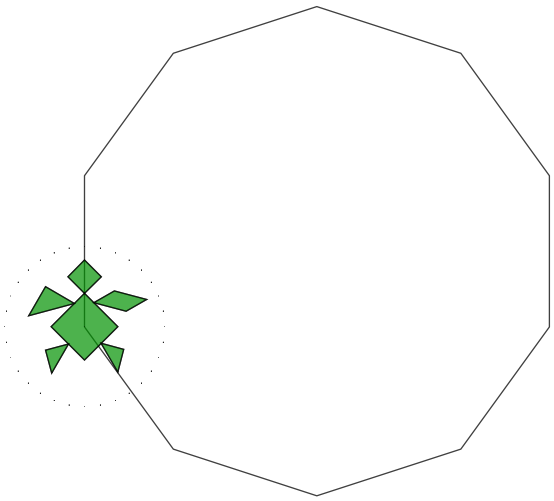


Sarà facile ora divertirsi a vedere come vengono poligoni con più lati. I calcoli li fa tutti il computer, si devono solo aggiungere istruzioni **FORWARD L RIGHT A**, tante quanti sono il numero dei lati: 3 per il triangolo equilatero, 4 per il quadrato, 5 per il pentagono, 6 per l'esagono e così via. Fino a quando? Finché ci pare, la matematica non pone limiti all'immaginazione. Ma la realtà sì: presto, andando avanti in una simile sperimentazione, incorreresti in un problema. In realtà ce ne possiamo accorgere anche solo guardando le figure che abbiamo appena costruito. Cosa cambia, oltre al numero dei lati, passando dal poligono a 3 lati a quello a 4, e quindi a quello a 5? Prova a immaginare, oppure prova tu stesso in un documento nuovo, copiando il codice qui sopra e eseguendolo con diversi valori di N.

Vado a pagina nuova per lasciarti il tempo di pensare o provare.

Quello che succede è che, aumentando il numero di lati aumenta la superficie della figura. È intuitivo: ogni volta si aggiunge un lato, quindi il perimetro aumenta. Poiché la figura è convessa[56] la superficie non può che crescere, con l'aumentare del perimetro. Quindi, noi possiamo immaginare i poligoni che vogliamo ma se li costruiamo con il nostro codice presto usciranno dal foglio! Quindi che possiamo fare? Un'idea può essere quella di mantenere il perimetro costante. Questo significa che, se vogliamo generare figure con un numero sempre più grande di lati, dobbiamo anche diminuire progressivamente la lunghezza di questi affinché il perimetro rimanga costante. Ci occorre una regola, che è presto detta: in un poligono regolare il perimetro si può ottenere moltiplicando la lunghezza del lato per il numero di lati. Se il lato è lungo L e il numero di lati è N, allora il perimetro è P = L*N. Questa formula ci consente di fissare il perimetro P e il numero di lati L, ricavando la lunghezza del lato: L = P/N. Scateniamoci: disegniamo un decagono regolare (10 lati)

```
CLEARSCREEN
HOME
P = 200mm
N = 10          ; Numero di lati
L = P/N         ; Lunghezza lato
A = 360/N       ; angolo interno
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
```

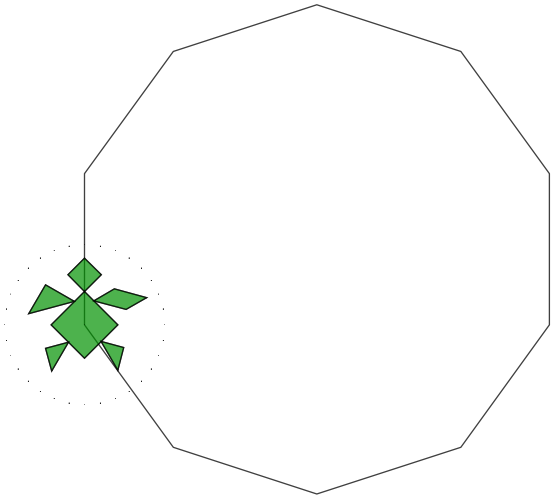


Soddisfacente no? Anche perché così stiamo mettendo a frutto l'utilità delle variabili (pag. 26), che forse alla prima non ci era parsa così chiara. Qui il fatto è evidente: possiamo inserire i dati numerici indispensabili e poi far calcolare i parametri derivati attraverso formule che utilizzano variabili simboliche. Una bella generalizzazione! Ma non siamo soddisfatti, a dire il vero. Infatti, per conseguire il nostro obiettivo di un un programma che disegni un poligono regolare qualsiasi siamo costretti a fare una cosa “sporca”: ci tocca inserire a mano tante nuove istruzioni **FORWARD L RIGHT A** quanti sono i lati. Funziona, ma non è “elegante”, e l'eleganza in matematica, come nella *computer science*, non di rado si traduce in chiarezza, in maggiore facilità di risolvere problemi successivi. E qual è qui il nostro problema? Quello di dover riscrivere, o copia-incollare, più volte la stessa identica istruzione: qualcosa che stride con la bell'idea di immaginare poligoni regolari qualsivoglia... e qui vengono i “cicli”.

56 Detto in italiano, una figura è convessa se il suo perimetro non ha “bozze” verso l'interno.

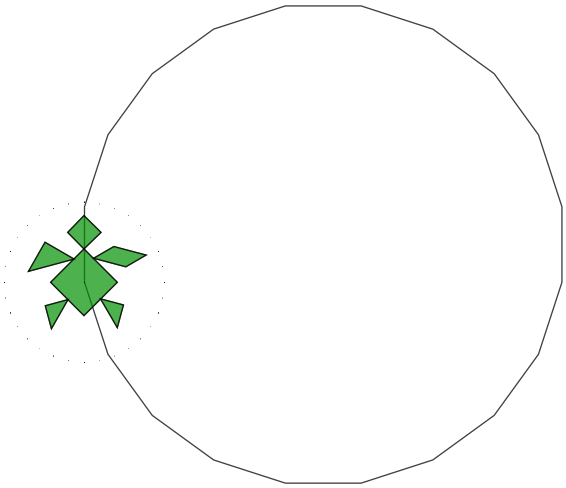
In tutti i linguaggi di programmazione esistono costrutti che consentono di ripetere più volte una stessa sequenza di istruzioni. Anzi, in tutti i linguaggi esistono più modi per ripetere sequenze di istruzioni, anche in LibreLogo! Qui, il costrutto più semplice è il seguente:

```
CLEARSCREEN
HOME
P = 200mm
N = 10          ; Numero di lati
L = P/N         ; Lunghezza lato
A = 360/N       ; Angolo interno
REPEAT N [
        FORWARD L RIGHT A
]
```



Abbiamo introdotto la novità brutalmente, all'interno di un problema, approfittando del fatto che in questo problema ci siamo già entrati (sperabilmente), e quindi confidando che il vantaggio sia più chiaro. I cicli in LibreLogo si possono realizzare con l'istruzione **REPEAT**, come nell'esempio precedente. Avremmo potuto anche scrivere **REPEAT 10 [FORWARD 20 RIGHT 36]**: tutte le istruzioni che compaiono fra parentesi vengono ripetute tante volte quanto indicato dal numero dopo **REPEAT**, **10** in questo caso. Poiché "all'interno di un **REPEAT**" possono essere incluse anche mote istruzioni, queste possono essere anche incolonnate, scrivendo **REPEAT [** nella prima riga, ponendo le varie istruzioni da ripetere nelle righe sottostanti e scrivendo la parentesi quadra di chiusura **]** nell'ultima riga, come abbiamo fatto nell'esempio del decagono e nei successivi che seguono. In tutti questi esempi abbiamo anche usato le variabili, per indicare il numero di ripetizioni e gli argomenti degli spostamenti e dei mutamenti di direzione. Questo ci consente sperimentare il codice con parametri diversi, molto facilmente, per esempio per costruire poligoni con più lati. Vediamo per esempio come viene con 20 lati...

```
CLEARSCREEN
HOME
P = 200mm
N = 20          ; Numero di lati
L = P/N         ; Lunghezza lato
A = 360/N       ; angolo interno
REPEAT N [
        FORWARD L RIGHT A
]
```



È anche evidente il vantaggio di scrivere in pochissime istruzioni un codice che senza il **REPEAT** ne richiederebbe moltissime. Si immagini qualcosa che si debba ripetere 100 o 1000 volte!

Sarebbe interessante vedere la progressione dei poligoni regolari al crescere del numero di lati.

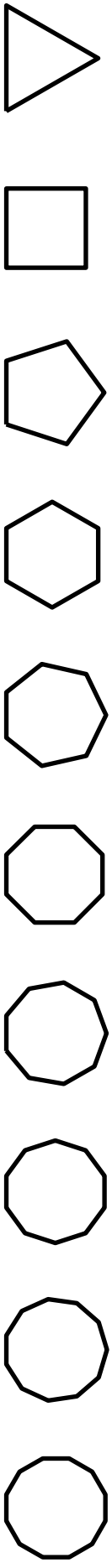
Programming languages should have a "low floor" and a "high ceiling"

Questa frase viene attribuita a Seymour Papert: i linguaggi di programmazione dovrebbero avere un pavimento basso e un soffitto alto. Ovvero: devono essere facili per chi inizia ma poi non devono porre limiti. Logo, e naturalmente LibreLogo, è concepito per avviare al pensiero computazionale (e matematico) i più piccoli, ma può essere usato anche per obiettivi didattici molto più complessi. Il semplice codice che abbiamo creato contiene l'embrione di quella che potrebbe essere un'interessante e intuitiva introduzione dei concetti di infinito e infinitesimo, quando, nelle scuole secondarie di II grado si affrontano i limiti, le cui definizioni sono comprese, nella sostanza, da una percentuale piccolissima di studenti. Quei limiti che recuperano un po' di concretezza quando se ne imparano le prime applicazioni, per esempio attraverso il concetto di derivata. Ma tutto l'argomento resta, nella maggior parte dei casi, un territorio piuttosto ostile. Eppure è un territorio importantissimo – per coloro che si affacceranno allo studio della matematica e della fisica prenderà il nome di "analisi matematica". Ed è proprio dove appare l'infinito che la matematica si fa interessante e, perché no?, immaginifica. In fin dei conti la matematica è l'unico ambito dello scibile dove l'uomo, limitato in tutto e per tutto, riesce in qualche modo a mettere il guinzaglio all'infinito. Purtroppo, il primo contatto che gli studenti hanno con quello che potrebbe essere altrimenti una sorta di paese delle meraviglie, è disperatamente formale, racchiuso in diciture che finiscono per essere imparate a memoria senza lasciare alcuna traccia, che non sia un senso di frustrazione che si traduce nella consueta e ingiustificata consapevolezza di "non essere tagliati per la matematica". L'esperienza concreta su come si possano disegnare dei poligoni regolari, toccando con mano il fatto che facendo crescere il numero di lati cresce il perimetro, e constatando come invece si possa fare crescere indefinitamente tale numero rimpicciolendo i lati e lasciando così fermo il perimetro, potrebbe essere articolato in maniera da far scaturire i concetti di infinito e infinitesimo naturalmente e intuitivamente. Niente di formalmente rigoroso ma la comprensione dei concetti non scaturisce tanto dalle esposizioni formali quanto dall'associazione con concetti già noti. Nell'esempio che segue accenniamo a un percorso del genere, cogliendo l'occasione per aggiungere qualche particolare.

Proviamo quindi a disegnare una sequenza di poligoni regolari con un numero crescente di lati. Tu come faresti? L' esempio a pagina nuova.

Per creare la successione di poligoni ricorriamo ad un ulteriore ciclo che contiene il precedente. Si parla in questo caso di “cicli annidati” (“nested cycles”). Oltre a questo, qui abbiamo fatto uso della variabile speciale **REPCOUNT**, che LibreLogo mette a disposizione come contatore delle ripetizioni di un ciclo.

```
CLEARSCREEN      ; pulisco il foglio
HOME             ; tartaruga a casa

PENUP            ; alzo la penna
PENSIZE 2        ; un tratto un po' più spesso
P = 50mm         ; perimetro poligoni

REPEAT 10 [      ; ciclo su 10 poligoni

; Qui uso il  contatore dei cicli REPCOUNT
; che parte da 1 e viene incrementato
; automaticamente di 1 a ogni ciclo.
; Lo uso per calcolare il numero di lati N dei
; poligoni, partendo da N = 3: quando
; REPCOUNT vale 1 allora N deve valere 3

        N = REPCOUNT+2
        L = P/N      ; lunghezza lato
        A = 360/N    ; ampiezza angolo

; Qui uso il contatore REPCOUNT per fissare
; la posizione di ciascun poligono, partendo
; dall'alto verso il basso. Per comprendere il
; valore di questi numeri vedi pag. 29 e seg.

        POSITION [380, 110+(REPCOUNT-1)*70]
        HEADING 0  ; punto in all'inizio di ogni pol.
        PENDOWN    ; calo la penna

;       Disegno il poligono

        REPEAT N [   ; ciclo sui lati del poligono
                FORWARD L RIGHT A
        ]
        PENUP        ; alzo la penna
]
HIDETURTLE
```

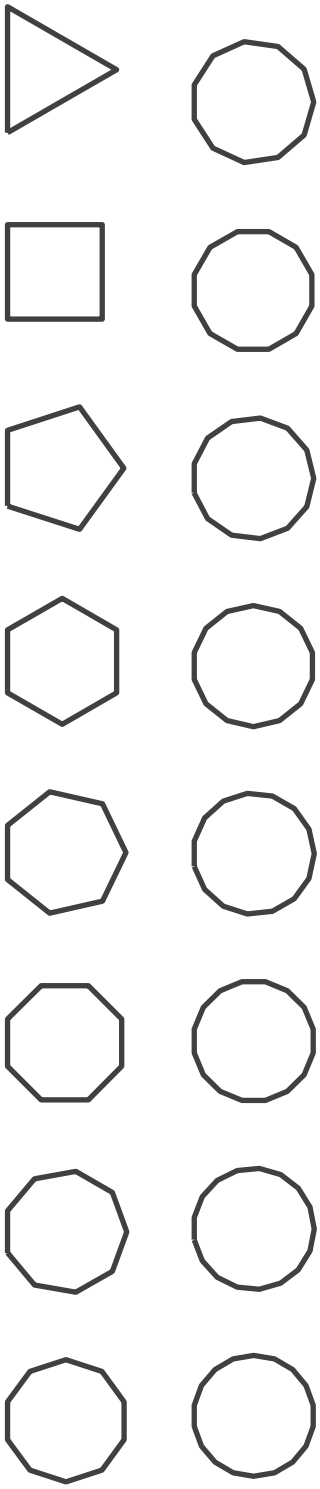


Proviamo ora a aggiungere un'altra colonna di poligoni, a fianco della precedente.

Nel codice qui sotto abbiamo aggiunto un ulteriore ciclo che serve a ripetere una seconda colonna di poligoni. Qui abbiamo dovuto introdurre una variabile REPCOUNT2 per memorizzare il contatore del ciclo più esterno, perché quando siamo all'interno del ciclo sui poligoni, per determinare la posizione di ciascuno di essi, abbiamo bisogno di ambedue i contatori.

Un'altra novità è costituita dal carattere ~ (si chiama tilde) che in LibreLogo consente di interrompere un'istruzione e continuarla nella riga seguente.

```
CLEARSCREEN
HOME
PENUP
PENSIZE 2
P = 50mm
NP = 8
PENUP

REPEAT 2 [                      ; ciclo sulle colonne

; memorizzo in REPCOUNT2 il contatore del ciclo
; esterno sulle colonne di poligoni

        REPCOUNT2 =REPCOUNT
        REPEAT NP [             ; ciclo sui poligoni
                N = (REPCOUNT2-1)*NP+REPCOUNT+2
                L = P/N
                A = 360/N
                POSITION [380+(REPCOUNT2-1)*70, ~
                        110+(REPCOUNT-1)*70 ~
                        -(REPCOUNT2-1)*5]
                HEADING 0
                PENDOWN
                REPEAT N [   ; ciclo sui lati del poligono
                        FORWARD L RIGHT A
                ]
                PENUP
        ]
]
HIDETURTLE
```



Così, disegnando poligoni, ci troviamo nei pressi del cerchio, che può essere visto in una luce nuova: come un poligono che ha infiniti lati infinitamente piccini. Conducendo pazientemente gli studenti in un'esercitazione del genere si potrebbero fare varie considerazioni interessanti.

Un'osservazione sulla scrittura del codice. Negli esempi precedenti abbiamo incolonnato le istruzioni in maniera particolare, usando la tabulazione (rientro, *indenting*) così da mettere in evidenza i diversi cicli: una tabulazione per le istruzioni nel ciclo più esterno (**REPEAT 2**), due tabulazioni nel ciclo sui poligoni (**REPEAT NP**) e tre tabulazioni nel ciclo più interno sui lati di ogni poligono (**REPEAT N**). In questo modo la struttura del codice è subito chiara. La presenza delle tabulazioni non influenza in alcun modo la funzionalità del programma in LibreLogo, serve solo a facilitare la leggibilità del codice. Non è sempre così con gli altri linguaggi di programmazione. Per esempio con il linguaggio Python la tabulazione influenza il comportamento del codice. In generale, è comunque molto importante scrivere il codice in modo chiaro e ordinato, intercalando i blocchi di codice con spazi, commenti e usando le tabulazioni.

L'istruzione **REPEAT** può essere usata anche senza l'argomento che specifica il numero dei cicli, ad esempio si potrebbe scrivere **REPEAT [FORWARD 100 RIGHT 90]**. In questo modo quello che si ottiene è un quadrato ma in realtà il programma rimane apparentemente bloccato, senza che si riesca più ad agire sul documento. Dico apparentemente perché quello che in realtà succede è che la tartaruga continua a disegnare il contorno del quadrato infinite volte. Infatti, se non si specifica il numero di cicli dopo **REPEAT**, il programma continua a ripetere cicli all'infinito! Se succede una cosa del genere si deve agire mediante il tasto di stop  per recuperare il controllo del documento. Questo è un fatto interessante che getta un po' di luce su quelle circostanze che spesso vengono confinate nell'idea che il computer “si è bloccato”. In realtà spesso non si è affatto bloccato ma lavora alacremente, magari, come in questo caso, ripetendo una stessa sequenza di operazioni all'infinito oppure per un numero molto grande di volte. Si tratta di condizioni particolari, che facilmente dipendono da comportamenti scorretti dell'utente (per esempio per avere aperto troppi processi: finestre) o da errori di programmazione (*bug*) di chi ha scritto il software.

Oltre a REPEAT esistono altre istruzioni per eseguire i cicli ma richiedono la conoscenza di costrutti che dobbiamo ancora affrontare.

Operazioni aritmetiche

Negli esempi precedenti, un po' in sordina, insieme alle variabili abbiamo introdotto le operazioni aritmetiche: la tartaruga disegna ma sa anche fare calcoli. Ecco le operazioni aritmetiche che si possono fare in LibreLogo:

Simbolo operazione	Risultato	Esempio
+	Somma	10 + 3 = 113
-	Sottrazione	10 – 3 = 7
*	Moltiplicazione	10 * 3 = 30
/	Divisione	10 / 3 = 3.3333333333333335
//	Quoziente (intero)	10 // 3 = 3
%	Resto (modulo)	10 % 3 = 1
**	Elevamento a potenza	10 ** 3 = 1000

È possibile un impiego marginale interessante: se si sta scrivendo un documento qaulasiasi in Writer, e LibreLogo è attivo, volendo si ha disposizione anche una calcolatrice. Con l'occasione, anticipiamo l'istruzione **LABEL "pippo"** che scrive il testo "pippo" nel punto in cui si trova la tartaruga. Se invece si ha una variabile, per esempo di nome A, e se ne vuole scrivere il contenuto nel documento, allora baasta eseguire **LABEL A**. Bene, facciamo ora un esempio di uso di LibreLogo come calcolatrice. Supponiamo che si voglia calcolare il 21% di 127. Si scrive il seguente pezzetto di codice...

```
A = 127
B = 21
LABEL B/A*100
```

e lo si esegue: la tartaruga scrive 16.535433070866144. Bene, il risultato è 16.5%. Da questo possiamo prendere il numero di decimali che ci fa comodo. Vedremo in seguito come troncare i decimali in LibreLogo o fare altre operazioni, anche molto più sofisticate.

Un accorgimento per trovare gli errori – la tartaruga troppo veloce!

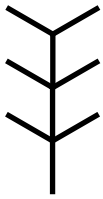
Non capita mai di scrivere il codice senza errori. Nello sviluppo di un software occorre sempre conteggiare anche il tempo necessario per individuare togliere gli errori. È sempre un'operazione onerosa e per ripulire veramente un software da tutti gli errori possono occorrere anni, con un processo di comunicazione continua fra chi ha scritto il software e chi lo usa. Si deve ovviamente cercare di pensare bene prima e evitare di commettere errori ma poi è normale commetterli. In gergo un errore si chiama "*bug*" (baco) e l'operazione di ricerca e correzione si chiama "*debugging*". Le tecniche di *debugging* sono molto varie e anche molto sofisticate. Nel caso di LibreLogo, che produce grafica, può essere utile seguire attentamente il percorso fatto dalla tartaruga, che magari non è affatto quello che ci eravamo prefigurati. In primo luogo è utile rendere visibile la tartaruga, anche perché così, oltre a seguirla meglio con lo sguardo, si rallenta un po' il disegno. Tuttavia può non bastare e, se il disegno è troppo intricato, è facile perderne le tracce. Ebbene, qui torna utile l'istruzione **SLEEP**, che si usa con un argomento che dice per quanto tempo la tartaruga deve dormire (to sleep in inglese significa dormire). Questo tempo deve essere espresso in millisecondi (msec), quindi se scrivo **SLEEP 1000**, la tartaruga se ne sta ferma per 1000 msec,

ovvero per un secondo. Si tratta quindi semplicemente di piazzare delle istruzioni **SLEEP 1000** qua e là, in maniera da rallentare adeguatamente il disegno. Ho scritto 1000 ma è un valore indicativo. Occorre andare un po' per tentativi perché dipende dalla velocità del computer, da quante altre cose sta facendo e dalla complicazione del disegno.

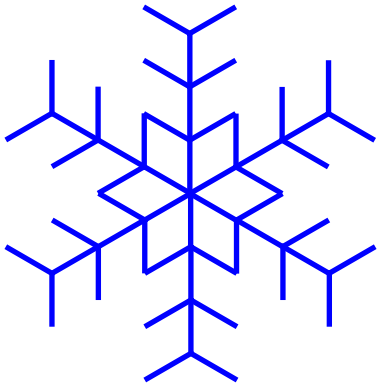
Esercizi

Può convenire cimentarsi in qualche esercizio, prima di andare avanti. Una soluzione degli esercizi può essere richiesta al sottoscritto: arf@unifi.it. Si faccia caso che ho scritto “una soluzione”. Questa è una caratteristica del *coding* da tenere ben presente: lo stesso risultato si può ottenere sempre in molti modi diversi.

Si provi a riprodurre questo disegno utilizzando i cicli:



Si provi quindi a costruire questo fiocco di neve:

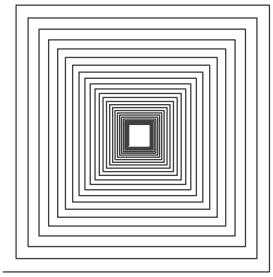


E se da questo fiocco tirassimo fuori una stella?

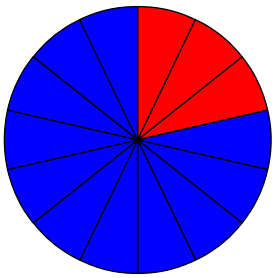
Naturale che sia venuta a 6 punte...

ma se la volessimo a 7 punte?

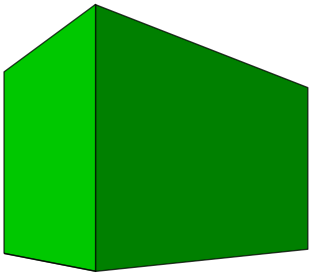
Proviamo a sperimentare l'uso di un'operazione all'interno del **REPEAT**, per esempio per fare una spirale quadrata:

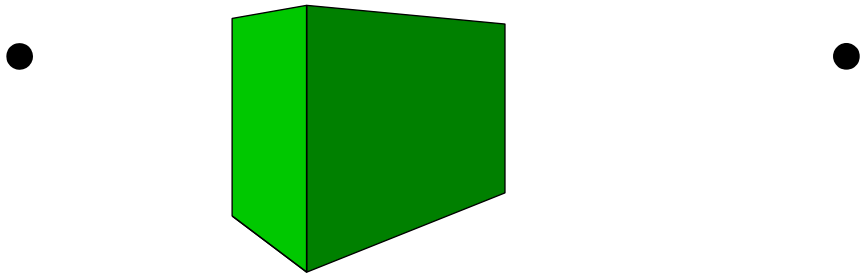


Oppure un esercizio di frazioni: in modo per esempio che, dati il numeratore N e il denominatore M, venga rappresentata la frazione N/M con una torta:



E a proposito di *"low floor" and a "high ceiling"*, un esercizio po' più difficile: disegnare un parallelepipedo in prospettiva, date le coordinate dei punti di fuga. Potrebbe andare bene per ragazzi di secondaria superiore (intersezione fra rette).



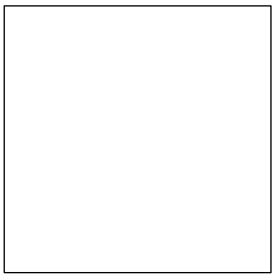


Cap. 5: Incapsulare

Subroutine – Funzioni - Procedure

Riprendiamo il disegno del quadrato, così come lo avevamo fatto a pag. 27, usando le variabili:

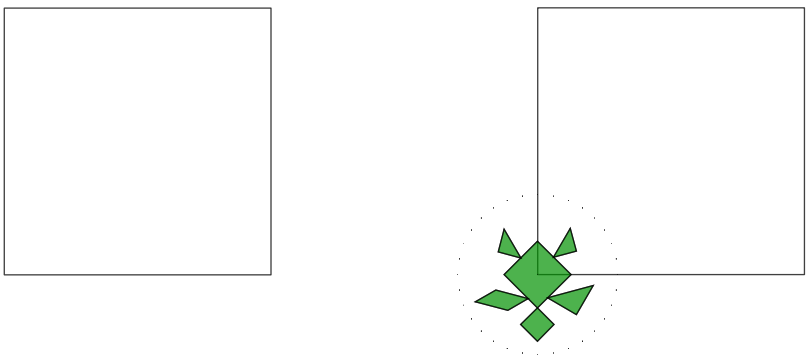
```
CLEARSCREEN
HOME

LATO = 100
ANGOLO = 90

FORWARD LATO
RIGHT ANGOLO
FORWARD LATO
RIGHT ANGOLO
FORWARD LATO
RIGHT ANGOLO
FORWARD LATO
HIDETURTLE
```



Abbiamo visto come siano comode le variabili, ad esempio per cambiare le dimensioni del disegno: se vogliamo fare un quadrato di lato 50 anziché 100 basta cambiare l'istruzione **LATO = 100** in **LATO = 50**, invece di cambiare l'argomento di tutte e quattro le istruzioni **FORWARD**. Tuttavia, se vogliamo disegnare più quadrati, magari di diverse dimensioni e in parti diverse del foglio, dobbiamo riscrivere tutto il blocco di istruzioni, una volta per ogni quadrato. Possiamo ricorrere a alle ripetizioni, in modo da ripetere tutto il blocco di istruzioni, cambiando solo ciò che serve ad ogni ciclo, ma non è detto che si possa ricorrere sempre a questo metodo con profitto. Inoltre, abbiamo già osservato come sia importante scrivere codice chiaro e ordinato. Quasi sempre la concisione è una virtù. È qui che vengono in aiuto le “subroutine”, che in altri linguaggi sono dette “funzioni” e in altri ancora “metodi”, ma il concetto di base è lo stesso. L'idea consiste nell'incapsulare una sezione di codice che serva ad un compito ben definito, in modo che questa possano essere impiegata semplicemente invocando una sola istruzione. È un accorgimento straordinariamente potente che consente di semplificare grandemente la scrittura del codice, e di conseguenza, di ridurre la probabilità di commettere errori.

Nel codice accanto, l'istruzione **TO QUADRATO** segna l'inizio della subroutine e l'istruzione **END** la fine. La parte in blu contiene il codice della subroutine. Nell'istruzione di inizio, **TO QUADRATO**, la parte **TO** è la dichiarazione di inizio della subroutine, mentre **QUADRATO** rappresenta il nome che abbiamo deciso di darle. È importante capire che quando si chiede a LibreLogo di eseguire uno script, le parti di codice comprese fra **TO** e **END** non vengono eseguite ma solo "imparate". Nel codice restante, quando LibreLogo arriva all'istruzione QUADRATO, in realtà va a eseguire le istruzioni blu della subroutine, poi continua con le seguenti. Ecco quindi che le subroutine sono un metodo per inventarsi delle istruzioni nuove, che ci dà la possibilità di arricchire piacimento il linguaggio. Naturalmente, l'istruzione QUADRATO funziona solo perché nello *script* è incluso il codice che la esprime, fra **TO** e **END.** Se provassimo ad utilizzarla in un altro *script* allora LibreLogo darebbe un errore. Ecco il risultato del codice precedente:

```
CLEARSCREEN       ; operazioni di preparazione
HOME              ; che vengono eseguite subito
SHOWTURTLE

; subroutine QUADRATO: viene solo "imparata"...
                    ; ma non eseguita

TO QUADRATO
        LATO = 100          ; fisso il lato del quadrato
        ANGOLO = 90         ; angolo di rotazione ai
                            ; vertici
        FORWARD LATO        ; primo lato
        RIGHT ANGOLO        ; giro di 90° a destra, ecc.
        FORWARD LATO
        RIGHT ANGOLO
        FORWARD LATO
        RIGHT ANGOLO
        FORWARD LATO
END

; script che viene eseguito

QUADRATO          ; qui si eseguono le istruzioni blu
PENUP
FORWARD 100
PENDOWN
QUADRATO          ; qui si eseguono le istruzioni blu
```

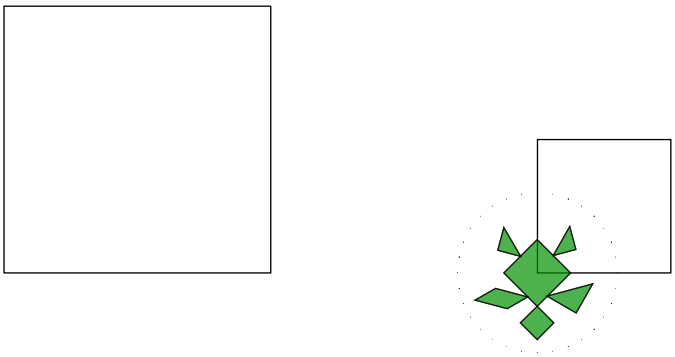


In realtà a noi piacerebbe controllare meglio il modo con cui vengono disegnati i quadrati, per esempio determinando la lunghezza del lato. Questo si può fare assegnando degli argomenti alla subroutine:

Qui, dopo la dichiarazione del nome della subroutine, abbiamo introdotto l'argomento **LATO**. Nel codice successivo alla subroutine, l'istruzione **QUADRATO** viene invocata con un argomento, pari a **100** la prima volta e **50** la seconda.
Ricapitolando, quando si fa eseguire il codice, premendo il tasto  , LibreLogo esegue le prime tre istruzioni, poi "impara" tutte quelle contenute nella subroutine **QUADRATO**; quindi esegue le istruzioni sottostanti, invocando **QUADRATO** con un valore **LATO** di **100**, spontandosi e poi di nuovo **QUADRATO** con un valore di **LATO** di **50**.

```
CLEARSCREEN
HOME
SHOWTURTLE

TO QUADRATO LATO
        ANGOLO = 90

        FORWARD LATO
        RIGHT ANGOLO
        FORWARD LATO
        RIGHT ANGOLO
        FORWARD LATO
        RIGHT ANGOLO
        FORWARD LATO
END

QUADRATO 100
PENUP
FORWARD 100
PENDOWN
QUADRATO 50
```

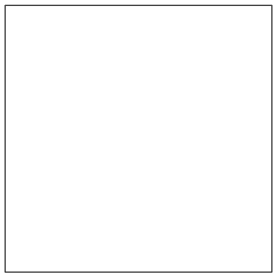
Ecco il risultato:

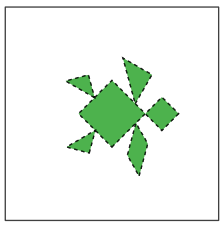


Così siamo liberi di invocare la nostra nuova funzione QUADRATO ogni volta che ne abbiamo bisogno, specificando liberamente la dimensione: QUADRATO 10, QUADRATO 30 o quello che vogliamo. Ora, a direil vero un'istruzione per disegnare i quadrati in LibreLogo esiste già: l'abbiamo incontrata a pag. 21, si chiama SQUARE e funziona allo stesso modo. O quasi, in realtà una differenza c'è: con SQUARE, una volta che il quadrato è stato disegnato, la tartaruga la ritroviamo al suo centro rivolta nella stessa direzione che aveva prima. Nel nostro caso invece la tartaruga rimane dove si trova dopo avere terminato di disegnare l'ultimo lato del quadrato. Effettivamente il comportamento di SQUARE sembra essere preferibile ma non è difficile far fare la stessa cosa alla nostra istruzione QUADRATO, ecco come:

```
CLEARSCREEN
HOME
SHOWTURTLE

TO QUADRATO LATO
        ANGOLO = 90             ; angoli interni quadrato

        PENUP                   ; alzo la penna
        FORWARD LATO/2          ; mi dirigo su lato che ho di fronte
                                ; così mi ritrovo a metà del lato di fronte
        RIGHT ANGOLO            ; giro a destra
        PENDOWN                 ; abbasso la penna
        FORWARD LATO/2          ; disegno mezzo del primo lato
        RIGHT ANGOLO            ; giro a destra
        FORWARD LATO            ; disegno il secondo lato
        RIGHT ANGOLO            ; giro a destra
        FORWARD LATO            ; disegno il terzo lato
        RIGHT ANGOLO            ; giro a destra
        FORWARD LATO            ; disegno il quarto lato
        RIGHT ANGOLO            ; giro a destra
        FORWARD LATO/2          ; disegno la metà rimanente de quarto lato
        RIGHT ANGOLO            ; giro a destra (per tornare nel centro del quadrato)
        PENUP                   ; alzo la penna
        FORWARD LATO/2          ; torno nel centro del quadrato
        LEFT ANGOLO*2           ; mi rigiro nella direzione in cui mi trovavo
inzialmente
END

QUADRATO 100
RIGHT 90
FORWARD 200
QUADRATO 80
```

E questo è il risultato:





La tartaruga è rivolta a destra perché per disegnare il secondo quadrato ha viaggiato da sinistra verso destra. Per rendere il comportamento dell'istruzione RETTANGOLO proprio identico a quello di SQUARE si dovrebbe intervenire anche sul colore del riempimento, mentre con il codice che abbiamo scritto questo non accade. Potremmo fare anche questo, utilizzando le istruzioni FILL e FILCOLOR, che abbiamo già visto. Lasciamo questa modifica come esercizio, per chi lo voglia

fare.

Si può obiettare che tutto questo lavoro sia inutile, visto che serve a fare una cosa che in LibreLogo già esiste. L'intento è primariamente pedagogico: le cose si spiegano bene a partire da esempi semplici; inoltre, è interessante constatare come si possano costruire da soli parti di un sistema che esistono già, perché questo ci aiuta ad acquistare fiducia e, al tempo stesso, a rendersi conto che il sistema che stiamo usando non è chiuso e composto di una materia inaccessibile; infine, ci rendiamo conto di poter contribuire al sistema stesso, magari anche costruendo delle varianti di istruzioni preesistenti – ad esempio, potrebbe esserci utile una versione dell'istruzione SQUARE che oltre alla dimensione del lato accetti anche il colore con il quale questo debba essere dipinto, o magari anche il colore del contorno. Qui si introduce un'altra generalizzazione: possiamo definire subroutine con più di un argomento. Per esempio possiamo provare a definire un'istruzione rettangolo, che possa essere invocata così: RETTANGOLO A B, dove A rappresenta il lato orizzontale e B quello breve. Lasciamo come esercizio le variazioni da apportare alla subroutine QUADRATO vista sopra, per ottenere una subroutine RETTANGOLO, nel modo accennato. Come lasciamo per esercizio la possibilità di introdurre il controllo dei colori, sia nella funzione QUADRATO che RETTANGOLO.

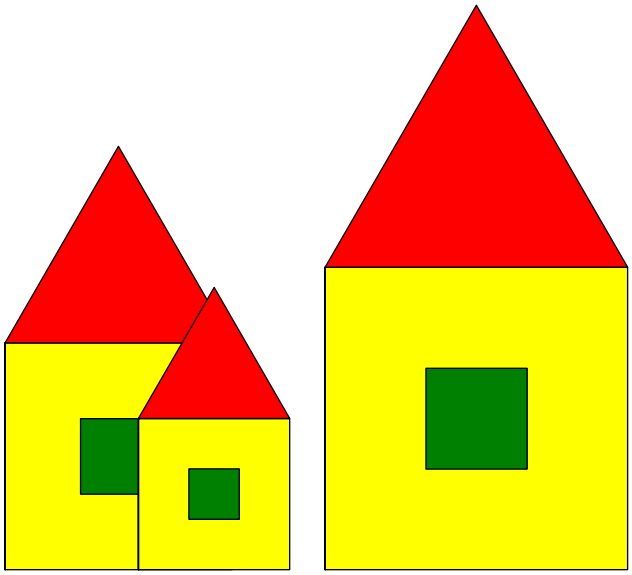
Ovviamente, se può avere senso la creazione di varianti di istruzioni esistenti, a maggio ragione, ne avrà la creazione di nuovi. Non c'è limite a quello che possiamo pensare di incapsulare in una subroutine. Prendiamo per esempio il codice che avevamo scritto a pagina 15 proviamo a incapsularlo in una subroutine:

```
CLEARSCREEN       ; operazioni di preparazione
HOME              ; che vengono eseguite subito

; subroutine CASA: viene solo "imparata"...
                  ; ma non eseguita

TO CASA
        FORWARD 50mm RIGHT 90°
        FORWARD 50mm RIGHT 90
        FORWARD 50mm RIGHT 90
        FORWARD 50mm RIGHT 90
        FORWARD 50mm RIGHT 30
        FILLCOLOR "yellow" FILL
        FORWARD 50mm RIGHT 120
        FORWARD 50mm RIGHT 120
        PENUP
        FORWARD 50mm/3
        LEFT 90
        FORWARD 50mm/3
        PENDOWN
        FILLCOLOR "red" FILL
        FORWARD 50mm/3 RIGHT 90
        FORWARD 50mm/3 RIGHT 90
        FORWARD 50mm/3 RIGHT 90
        FORWARD 50mm/3 RIGHT 90
        FILLCOLOR "green" FILL
        HIDETURTLE
END

; istruzioni eseguite

CASA
```

Se si prova ad eseguire questo codice si ottiene la stessa casetta che avevamo ottenuto a pagina 15. A questo punto è facilissimo introdurre delle varianti, per esempio la dimensione delle case, che possiamo controllare mediante un'apposita variabile, che chiamiamo LATO e che passiamo come argomento nella subroutine CASA:

```
CLEARSCREEN
HOME

TO CASA LATO
        FORWARD LATO RIGHT 90°
        FORWARD LATO RIGHT 90
        FORWARD LATO RIGHT 90
        FORWARD LATO RIGHT 90
        FORWARD LATO RIGHT 30
        FILLCOLOR “yellow” FILL
        FORWARD LATO RIGHT 120
        FORWARD LATO RIGHT 120
        PENUP
        FORWARD LATO/3
        LEFT 90
        FORWARD LATO/3
        PENDOWN
        FILLCOLOR “red” FILL
        FORWARD LATO/3 RIGHT 90
        FORWARD LATO/3 RIGHT 90
        FORWARD LATO/3 RIGHT 90
        FORWARD LATO/3 RIGHT 90
        FILLCOLOR “green” FILL
        HIDETURTLE
END


PENUP POSITION [150,400] HEADING 0 PENDOWN
CASA 30mm
PENUP POSITION [200,400] HEADING 0 PENDOWN
CASA 20mm
PENUP POSITION [270,400] HEADING 0 PENDOWN
CASA 40mm
```

Il codice della subroutine **CASA** è lo stesso di prima eccetto per la presenza dell'argomento **LATO**. Nello script l'istruzione **CASA** viene chiamata tre volte, sempre con dimensioni diverse. La posizione viene controllata mediante sequenze di istruzioni del tipo **PENUP POSITION [150,400] HEADING 0 PENDOWN**: alzo la penna, mi trasferisco nel punto di coordinate [150,400] (peer esempio), mmi dirigo in su, riabbasso la penna. Ecco il risultato:



Non c'è limite alla fantasia. Non è difficile scrivere una subroutine che disegni un albero e usarla per arricchire così il paesaggio. Lo proponiamo come esercizio dopo, ma prima vediamo un altro esempio più avanzato, intendendo con questo che potrebbe essere utilizzato in un contesto di scuola secondaria superiore (la descrizione che segue è molto dettagliata, chi non è interessato a un contesto del genere e non ha dimestichezza con questo livello di conoscenza matematiche, salti senz'altro l'esempio!). Riprendiamo le successioni di poligoni che avevamo visto a pag. 45 e 46. Lì avevamo rappresentato la successione incolonnando i poligoni. L'idea ora è di sovrapporli anziché incolonnarli, in modo da apprezzare l'evoluzione verso il cerchio al crescere del numero dei lati. Scegliamo di costruire i poligoni inscritti in un cerchio di raggio dato, formando così la successione dei poligoni inscritti nel cerchio. Potremmo egualmente considerare la successione dei poligoni circoscritti al cerchio. Nel primo caso il parametro chiave è il raggio dei poligoni, sempre eguale al raggio del cerchio in cui sono inscritti. Nel secondo sarebbe invece l'apotema, sempre eguale al raggio del cerchio che circoscrivono. L'esempio che segue descrive il primo caso. Il codice che segue è commentato minuziosamente. Ciò nonostante descriviamo puntualmente la struttura del codice e il procedimento.

Intanto l'esercizio utilizza, in un contesto un po' più complicato, i tre costrutti fondamentali del software che abbiamo sin qui introdotto: le variabili e le operazioni fra di esse, le ripetizioni di sequenze di istruzioni e l'incapsulamento di sezioni di codice nelle subroutine. Abbiamo mantenuto l'evidenziazione cromatica che abbiamo usato in alcuni degli esempi precedenti, per aiutare la lettura del codice. Nella prima sezione(in nero) si eseguono le operazioni preparatorie: cancellazione del foglio, tartaruga a casa, tartaruga invisibile (sarebbe troppo “invasiva” su un disegno più articolato come questo), penna alzata; inoltre si fissano i parametri necessari per iniziare, ovvero numero dei poligoni che dovranno comporre la successione, e raggio dei poligoni, espresso in punti. Poi viene il codice della subroutine **POLIGONO**, con le istruzioni in blu, eccetto il nome della subroutine in rosso e i suoi argomenti in viola. La subroutine **POLIGONO** richiede 5 argomenti: X0P e Y0P sono le coordinate del centro del poligono, che possiamo quindi piazzare dove vogliamo; N è il numero di lati che deve avere il poligono; R è il raggio del poligono. All'interno della subroutine **POLIGONO** si sviluppa la geometria. Si calcola l'ampiezza degli

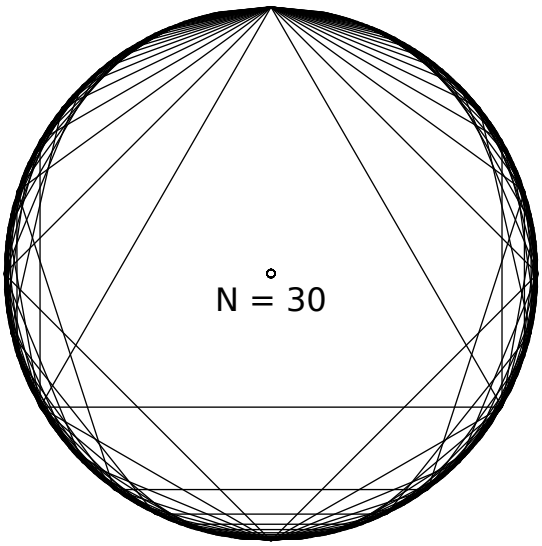
angoli interni **AI**, l'ampiezza degli angoli supplementari degli angoli **AI**, che chiamiamo **A** e la lunghezza dei lati **L**. Qui troviamo una novità, anzi due: **SIN** e **ABS** sono funzioni matematiche e **PI** è una costante. Scopriamo quindi che LibreLogo "sa" un bel po' di matematica! **PI** è una variabile speciale, per meglio dire una costante, la più importante della matematica: il **π** (pi greco), di cui LibreLogo esprime un'approssimazione con 15 cifre decimali (provare ad eseguire l'istruzione **PRINT PI**)[57]. Invece **SIN** e **ABS** sono funzioni matematiche: **SIN** calcola il valore della funzione trigonometrica seno e richiede un argomento espresso in radianti, per esempio **SIN PI** fornisce il valore **0; ABS** calcola il valore assoluto dell'argomento, ovvero se è positivo lo lo lascia positivo mentre se è negativo lo trasforma in positivo. Successivamente vengono le istruzioni che disegnano il poligono, ovvero il viaggio della tartaruga. In sintesi, la tartaruga va nel centro indicato, di coordinate **[X0P, Y0P]**, si volge in alto e percorre senza disegnare il raggio, gira a destra di 90° (questa è la direzione della tangente al cerchio circoscritto), poi gira di quanto basta per allinearsi al primo lato del poligono (si poteva girare in un sol colpo verso questa direzione, ma abbiamo lasciato il codice in questa forma sotto-ottimale per chiarire la geometria). A questo punto si abbassa la penna e si iniziano a disegnare i lati in successione, mediante il semplice ciclo **REPEAT N [FORWARD L RIGHT A]**. Le istruzioni finali all'interno della subroutine servono a scrivere un'etichetta, poco sotto al centro del poligono, con il numero di lati. Siccome è un codice che ci mette un certo tempo girare (ovviamente questo dipende anche dal computer che si usa), quando il numero di lati diventa elevato, è utile per sapere a che punto si trova il processo – noi abbiamo provato fino a 500 lati. Da segnalare qui l'uso della variabile riservata (in LibreLogo) di **REPCOUNT**, che è il contatore di cicli. Inoltre, si usa la funzione **STR** che serve a trasformare il valore di **REPCOUNT**, che è un numero espresso internamente al computer in binario, nell'espressione alfanumerica del medesimo, che possa essere stampata sul foglio come qualsiasi altro testo[58]. La stampa dell'etichetta è subordinata la fatto che si tratti dell'ultimo poligono. Questo controllo viene effettuato con l'istruzione di controllo **IF**, descritta nel capitolo successivo. Dopo il codice della subroutine, di nuovo in nero, c'è lo *script* vero e proprio, molto semplice. Le tre istruzioni **HOME**, **X0P=POSITION[0]** e **Y0P=POSITION[1]** servono per piazzare il centro dei poligoni nel centro del foglio, ma qui potremmo scegliere una qualsiasi altra posizione. Infine, la realizzazione della successione è affidata al ciclo **REPEAT NP […]**. Anche qui si usa il contatore di cicli, in questo caso, per chiamare la subroutine **POLIGONO** con il giusto numero di lati. Come dicevamo, questo script può richiedere del tempo, se fatto girare con un numero elevato di poligoni, diciamo da 10 in su, indipendenza della velocità del processore che equipaggia il computer. Se capita di farlo partire inavvertitamente con un numero di lati eccessivo, o se lo si vuole fermare per qualsiasi altro motivo, lo si può fare con il tasto  . Se si supera il valore di circa 30 lati, si inizia a vedere un contorno apparentemente circolare. Più che si aumenta il numero di lati e più che la circolarità è "vera".

57 Ricordiamo che **π** rappresenta il rapporto fra la misura della circonferenza e il raggio di un cerchio, e che si tratta di numero irrazionale, quindi con un numero infinito di cifre decimali.

58 Questa è una nozione che a qualcuno può sembrare oscura. Molto in sintesi: una cosa sono i numeri espressi in un formato con il quale il computer possa fare i calcoli e un'altra sono i numeri espressi come caratteri, che possono essere inframezzati in un testo qualsiasi. Nel primo caso si tratta di una codifica per il computer è binaria, l'unico modo che consenta al computer di fare operazioni matematiche. Nel secondo di tratta di una codifica che serve a rappresentare i caratteri sullo schermo, o in una stampa; questa è una codifica che ha una finalità puramente grafica e che il computer non può usare per fare i calcoli. Esistono molti tipi di codifiche, le più note della quali sono ASCII E UNICODE. Chi desidera chiarimenti si può rivolgere all'autore di questo testo.

```
; Script per disegnare una successione di poligoni inscritti in un cerchio di raggio R dato

CLEARSCREEN                 ; pulisco il foglio
HOME                        ; tartaruga a casa
HIDETURTLE                  ; tartaruga invisibile
PENUP                       ; alzo la penna
NP = 20                     ; numero poligoni
R = 100                     ; raggio poligoni –- tengo fissa il raggio per tutti i poligoni
                            ; cosicché risultano tutti inscritti nello stesso cerchio di raggio R
; subroutine POLIGONO
;       argomenti:      X0P: X del centro del poligono
;                       Y0P: Y del centro del poligono
;                       N: numero lati poligono
;                       R: misura raggio
;                       NP: numero poligoni (serve a scrivere l'etichetta)

TO POLIGONO X0P Y0P N R NP
        A = 360/N                       ; angoli supplementari degli angoli interni: quelli di cui gira
                                        ; la tartaruga ad ogni vertice  (N→∞  => A→0)
        AI = 180*(N-2)/N                ; angoli interni  (N→∞  => AI→180)
        L = ABS(2*R*SIN(A/2*PI/180))    ; lato

;       Disegno il poligono

        FILLCOLOR “white” PENDOWN CIRCLE 3 PENUP
        POSITION [X0P,Y0P]              ; vado al centro
        HEADING 0                       ; mi dirigo in su
        FORWARD R                       ; percorro il raggio
        RIGHT 90                        ; 90° a destra (direzione tangente al cerchio circoscritto)
        RIGHT (180 – AI)/2              ; direzione primo lato da disegnare
        PENDOWN                         ; giù la penna
        REPEAT N [                      ; ciclo sui lati del poligono
                FORWARD L RIGHT A
        ]
        PENUP                           ; alzo la penna
        POSITION [X0P,Y0P]              ; torno al centro
        IF REPCOUNT = NP [
                FORWARD 10                      ; vado un po' sotto per scrivere etichetta
                HEADING 0                               ; mi giro in su per...
                LABEL “N = ” + STR REPCOUNT             ; … scrivere etichetta corrente
                HEADING 6h                              ; mi rigiro in giù
                BACK 10                                 ; torno al centro
        ]
END                                     ; fine subroutine POLIGONO

; script vero e proprio

HOME                                    ; vado al centro ma potrei andare anche altrove
X0P = POSITION[0]                       ; X del centro del poligono
Y0P = POSITION[1]                       ; Y del centro del poligono
REPEAT NP [                                     ; ciclo sui poligoni
        N = REPCOUNT+2                          ; numero lati poligono
        POLIGONO X0P Y0P N R NP
]
```

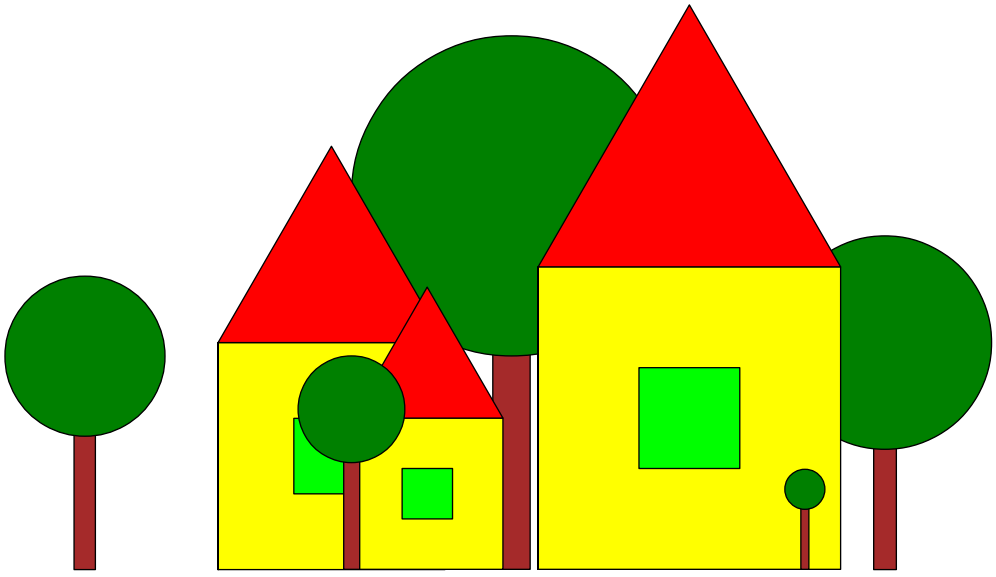
Ecco il risultato:



Dove sono sovrapposti i poligoni regolari, a partire dal triangolo equilatero (N=3), fino a quello con 30 lati.

Esercizio

Riprendendo l'esempio della costruzione di case con un'apposita subroutine, si provi ad arriccchire il paesaggio...



Cap. 6: Decidere

IF – AND, OR, NOT

In questa versione proponiamo questo capitolo in forma estremamente sintetica. Giusto per completezza, perché il costrutto che si descrive è uno di quelli fondamentali in qualsiasi linguaggio di programmazione, oltre alle variabili, le ripetizioni e le procedure. Si tratta di disporre del modo per interrompere il flusso normale delle istruzioni, passando eventualmente a eseguire sezioni di codice diverse in dipendenza dello stato di certe variabili. L'istruzione che realizza questo in LibreLogo è **IF**, che per essere eseguita richiede la definizione di una condizione logica. Vediamo un esempio, riprendendo il codice per disegnare un cerchio, così come introdotto da Papert nel capitolo 2:

```
TO CERCHIO
        REPEAT [
                FORWARD 1
                RIGHT 1
        ]
END

CERCHIO
```

Se facciamo girare questo codice la Tartaruga disegna un cerchio ma non si ferma mai, ripassandolo infinite volte. Naturalmente noi possiamo fermarla con il tasto  , ma è possibile insegnarle a fermarsi da sola. Ecco come:

```
TO CERCHIO
        REPEAT [
                FORWARD 1
                RIGHT 1
                IF REPCOUNT = 90 [ STOP ]
        ]
END

CERCHIO
```

Come si vede, abbiamo aggiunto una sola istruzione, **IF REPCOUNT = 90 [STOP]**, che equivale a dire alla Tartaruga: se il contatore dei cicli ha raggiunto il valore di 90 allora fermati. Siccome ad

ogni ciclo ruota di 1 grado, in questo modo ne interrompiamo il disegno quando in totale avrà ruotato di 90 gradi, ovvero quando avrà disegnato un quarto di cerchio. Provare e variare per vedere...

La condizione in questo esempio è espressa da **REPCOUNT = 90**. Si possono usare anche gli operatori “minore di”, <, e “maggiore di”, >, e le condizioni si possono combinare insieme con gli operatori logici AND, OR e NOT. L'AND posto fra due condizioni crea una condizione globale vera se sono ambedue vere. L'OR posto fra due condizioni crea una condizione globale vera se sono ambedue vere oppure anche una sola delle due. Il NOT posto prima di una condizione ne inverte l'esito: la rende falsa se è vera e viceversa. Inoltre si può costruire l'istruzione in maniera che se questa è vera esegue una prima sezione di codice, mentre se è falsa ne esegue un'altra. In questa versione del manuale ci limitiamo a riportare giusto un esempio riassuntivo:

```
IF A < 10 AND NOT A = 5 [ PRINT “Vero!” ] [ PRINT “Falso!” ]
```

Tradotto in parole: se la variabile A è minore di 10 e allo stesso tempo (AND), è diversa da 5 (NOT), allora esegui PRINT “Vero!”, altrimenti esegui PRINT “Falso!”.

Bibliografia

Può parere strano avere introdotto dei riferimenti a testi scritti in ungherese, ma gli esempi grafici che vi si trovano possono fornire ispirazione e aiutandosi con Google Translate qualcosa si recupera.

1. E. Castelnuovo (2008), L'officina matematica – ragionare con i materiali, p. 38, Edizioni la Meridiana.
2. Viktória Lakó (2013), LibreLogo – *La grafica della Tartaruga per imparare a programmare*, (In ungherese) E-government Free Software Competence Center. (http://szabadszoftver.kormany.hu/wp-content/uploads/librelogo_oktatasi_segedanyag_v4.pdf)
3. Németh László (2013), LibreLogo, FSF.hu Alapíttvány, (In ungherese). (http://www.numbertext.org/logo/logofuzet.pdf)
4. Seymour Papert (1993), Mindstorms, Children, Computers, and Powerful Ideas. New York: Basic Books.

Siti

1. LibreLogo, sito ufficiale: http://librelogo.org/
2. Raccolta di immagini con il codice sorgente per poterle riprodurre e eventualmente modificarle: http://commons.wikimedia.org/wiki/Category:Images_with_LibreLogo_source_code .

APPENDICE

Qui raccogliamo i listati dei codici usati per costruire alcune delle figure usate nel manuale. Naturalmente, questi codici talvolta possono contenere dei costrutti che non sono stati ancora affrontati nel punto del testo cui appare la figura corrispondente. Non importa se non si capisce tutto subito, il lettore può sempre tornarci successivamente, quando avrà sarà più esperto. Li mettiamo a disposizione perché può essere interessante e utile per vedere che cosa si può fare in pratica.

I listati sono commentati. In LibreLogo i commenti si ottengono preponendo un punto e virgola: in qualsiasi riga, tutto quello che segue il punto e virgola non viene eseguito ma serve solo a rendere più facile da leggere il codice. Questo significa che se salviamo il codice di una di queste figure, così com'è, con tutti commenti, questo può essere seguito per produrre la figura.

È molto importante inserire nel codice codice commenti chiari e accurati, sia per rileggerlo più facilmente molto tempo dopo, sia per facilitare la collaborazione con altre persone. Scrivendo codice, specie quando si è acquisita una certa confidenza, è facile farsi prendere la mano, cercando di arrivare quanto più velocemente al risultato desiderato. È bene invece controllarsi, imponendosi di documentare adeguatamente i lavoro man mano che si procede. Tanto più si aspetta quanto più sarà faticoso andare a commentare il lavoro fatto, sia per la mole che per la maggiore difficoltà a ricordare i particolari. Naturalmente questo vale per il codice destinato durare un certo tempo e ad essere condiviso, come potrebbe essere il caso di quello scritto per costruire alcune delle figure di questo manuale. Non certo per piccoli frammenti estemporanei. Il codice riportato nelle seguenti pagine è un po' al limite. Ma lo abbiamo commentato per mostrare la buona pratica, ivi inclusa l'apposizione di un'intestazione che riporti anche il nome dell'autore, il numero della versione e la data. Intestazione che andrà aggiornata con eventuali successive modifiche.

```
;  FIGURA 1 (torna alla figura)
; Le coordinate della patorna alla figuragina
; Versione 1.
; A.R. Formiconi
; 25 luglio 2016


; Versione scritta come viene, senza particolari ottimizzazioni

;  Predispone tutto

CLEARSCREEN
HOME
A = 200                         ; lato corto del rettangolo che rappresenta la pagina
B = 282                         ; lato lungo
FILLCOLOR [230, 230, 230]       ; fissa a grigio chiaro il colore di riempimento
PENUP
FORWARD B/2                     ; si dirige verso l'angolo in alto a sinistra
HEADING 9h
FORWARD A/2
HEADING 3h                      ; e punta verso destra (ore 3)
PENDOWN

; disegna il rettangolo, ruotando in senso orario

FORWARD A
RIGHT 90
FORWARD B
RIGHT  90
FORWARD A
FILL                            ; riempie il rettangolo
PENUP
RIGHT 90
FORWARD B                       ; si posiziona nell'angolo in alto a sinistra del foglio

; ripete il giro marcando gli angoli e ponendo le scritte

HEADING 3h                      ; angolo alto destro
FILLCOLOR [50, 50, 50]          ; fissa un grigio più scuro per i cerchietti
FORWARD A
PENDOWN
CIRCLE 5
PENUP
FORWARD 10 LEFT 90 FORWARD 15
LABEL “[PAGESIZE[0], 0]”
BACK 15 RIGHT 90 BACK 10

RIGHT 90                        ; angolo basso destro
FORWARD B
PENDOWN
CIRCLE 5
```

```
PENUP
FORWARD 15 LEFT 90 FORWARD 30 LEFT 90
LABEL “[PAGESIZE[0],PAGESIZE[1]]”
LEFT 90 FORWARD 30 RIGHT 90 FORWARD 15
RIGHT 180

RIGHT 90                        ; angolo basso sinistro
FORWARD A
PENDOWN
CIRCLE 5
RIGHT 90
PENUP
BACK 15
LABEL “[0, PAGESIZE[1]]”
FORWARD 15

FORWARD B                       ; angolo altro sinistro
PENDOWN
CIRCLE 5
PENUP
FORWARD 15
LEFT 90 FORWARD 10 RIGHT 90
LABEL “[0, 0]”
BACK  15
PENDOWN

HIDETURTLE                      ; mando la tartaruga a dormire


;  FIGURA 2 (Torna alla figura)
; L'effetto dell'istruzione POSITION e le coordinate della pagina
; Versione 1.
; A.R. Formiconi
; 25 luglio 2016


; Versione scritta come viene, senza particolari ottimizzazioni

;  Predispone tutto

CLEARSCREEN
HOME
A = 200                         ; lato corto del rettangolo che rappresenta la pagina
B = 282                         ; lato lungo
FILLCOLOR [230, 230, 230]       ; fissa a grigio chiaro il colore di riempimento
PENUP
FORWARD B/2                     ; si dirige verso l'angolo in alto a sinistra
HEADING 9h
FORWARD A/2
HEADING 3h                      ; e punta verso destra (ore 3)
PENDOWN
```

```
; disegna il rettangolo, ruotando in senso orario

FORWARD A
RIGHT 90
FORWARD B
RIGHT  90
FORWARD A
FILL                          ; riempie il rettangolo
PENUP
RIGHT 90
FORWARD B                     ; si posiziona nell'angolo in alto a sinistra del foglio

; ripete il giro marcando gli angoli e ponendo le scritte

HEADING 3h                    ; angolo alto destro
FILLCOLOR [50, 50, 50]        ; fissa un grigio più scuro per i cerchietti
FORWARD A
PENDOWN
CIRCLE 5
PENUP
FORWARD 10 LEFT 90 FORWARD 15
LABEL “[PAGESIZE[0], 0]”
BACK 15 RIGHT 90 BACK 10

RIGHT 90                      ; angolo basso destro
FORWARD B
PENDOWN
CIRCLE 5
PENUP
FORWARD 15 LEFT 90 FORWARD 30 LEFT 90
LABEL “[PAGESIZE[0],PAGESIZE[1]]”
LEFT 90 FORWARD 30 RIGHT 90 FORWARD 15
RIGHT 180

RIGHT 90                      ; angolo basso sinistro
FORWARD A
PENDOWN
CIRCLE 5
RIGHT 90
PENUP
BACK 15
LABEL “[0, PAGESIZE[1]]”
FORWARD 15

FORWARD B                     ; angolo altro sinistro
PENDOWN
CIRCLE 5
PENUP
FORWARD 15
LEFT 90 FORWARD 10 RIGHT 90
LABEL “[0, 0]”
BACK  15
```

```
PENDOWN

; disegna il percorso della tartaruga in seguito all'istruzione POSITION [350,320]

HOME                        ; mi posiziono al centro
PENUP                       ; senza disegnare...
POSITION [298, 430]         ; vado dove voglio piazzare...
HEADING 0                   ; l'etichetta con le...
LABEL “[298, 421]”          ; coordinate del centro
HOME                        ; torno al centro e...
PENDOWN                     ; disegno...
CIRCLE 5                    ; il cerchietto centrale
FILLCOLOR “green”           ; fisso il colore verde per la tartaruga, che mostrerò
POSITION [350,320]          ; applico POSITION [350,320]
P = POSITION                ; memorizzo tale posizione...
H = HEADING                 ; e direzione
PENUP                       ; alzo la penna
POSITION [300,320]          ; mi sposto un po' per piazzare...
HEADING 0                   ; orientata correttamente
LABEL “[350, 320]”          ; l'etichetta
POSITION P                  ; ritorno alla posizione e direzione...
HEADING H                   ; per lasciarvi visibile la tartaruga



;  FIGURA 3 (Torna alla figura)
; I riferimenti per l'istruzione HEADING
; Versione 1.
; A.R. Formiconi
; 25 luglio 2016


; Versione scritta in maniera più ordinata e strutturata, facendo uso delle “subroutine”
; La strutturazione del software di questo tipo consente di rendere il codice più modulare
; e più mantenibile

; Prima vengono le subroutine e alla fine il programma vero e proprio. Prima vanno messi
; i frammenti più elementari, in maniera che LibreLogo li legga per primi. Questo serve
; perché quando nel codice viene citata una subroutine, questa deve essere già stata
; analizzata da LibreLogo.


; Subroutine BR per disegnare un segmento lungo 10 pt dalla posizione e lungo la
; direzione corrente senza muoversi (come risultato finale)
; Parametri:
;       P: posizione corrente
;       H: direzione corrente

 TO BR P H

        FORWARD 10
        POSITION P
```

```
        HEADING H
END

; Subroutine TARROW per disegnare la punta di una freccia

TO TARROW
        P = POSITION
        H = HEADING
        LEFT 160
        BR P H
        RIGHT 160
        BR P H
END

; Subroutine LBA scrivere il testo contenuto in T in vetta al segmento
; di lungheza L e ruotato dell'angolo A
; Parametri:
;       L: lunghezza del segmento
;       A: angolo di rotazione del segmento
;       T: testo da scrivere nell'etichetta

TO LBA L A T
        PENUP
        FORWARD L/10 + L/5 * SIN A*PI/180
        H = HEADING
        HEADING 0
        LABEL T
        PENDOWN
END

; Subroutine LB scrivere il testo contenuto in T in una posizione determinata in
; coordinate polari rispetto alla posizione corrente, mediante la
; distanza L l'angolo A
; Parametri:
;       L: distanza
;       A: angolo
;       T: testo da scrivere nell'etichetta

TO LB L A T
        P0 = POSITION
        PENUP
        S =  SIN A*PI/180
        C =  COS A*PI/180
        POSITION [P0[0] + L * C, P0[1] - L * S]
        HEADING 0
        LABEL T
        POSITION P0
        HEADING 0
        PENDOWN
END

; Subroutine ARROW per disegnare, a partire dalla posizione corrente,
```

```
; una freccia di lunghezza L, inclinata di un angolo A, e in vetta una
; etichetta con il testo contenuto in T
; Parametri:
;       L: lunghezza della freccia
;       A: angolo di rotazione della freccia
;       T: testo da scrivere nell'etichetta

TO ARROW P0 A0 L A T
        PENDOWN
        HEADING A
        FORWARD L
        TARROW
        LBA L A T
        PENUP
        POSITION P0
        HEADING A0
        PENDOWN
END

; Questo è il programma vero e proprio che, come si vede, grazie al ricorso
; alle subroutine, è abbstanza conciso.

CLEARSCREEN                 ; cancello il foglio
HOME                        ; mando a casa la tartaruga
HIDETURTLE                  ; faccio il disegno senza vedere la tartaruga
FILLCOLOR [230, 230, 230]   ; fisso il riempimento a un grigio scuro
CIRCLE 5                    ; disegno un cerchietto nella posizione centrale
P0 = POSITION               ; memorizzo tale posizione iniziale...
A0 = HEADING                ; e anche la direzione iniziale
L = 150                     ; faccio la freccia lunga 150 pt
A = 60                      ; e la voglio inclinata di 60°

PENSIZE 1 ARROW P0 A0 L 0 “HEADING 0”     ; freccia verticale
PENSIZE 0.5 ARROW P0 A0 L A “HEADING 30”  ; freccia a 60°
PENSIZE 1 ARROW P0 A0 L 90 “HEADING 90”   ; freccia orizzontale
PENSIZE 0.5
ELLIPSE [L/3, L/3, 0, A, 3]               ; arco di cerchio piccolo
LB L/4 A “30°”                            ; etichetta “30°”
ELLIPSE [L*2, L*2, 0, 90, 3]              ; arco di cerchio grande




;  FIGURA 4 (Torna alla figura)
; Esempi di spessore del tratto
; Versione 1.
; A.R. Formiconi
; 25 luglio 2016

; In questo esempio si illustra l'impiego dell'istruzione REPEAT, per realizzare i “cicli” (“loop”)

CLEARSCREEN
```

```
HOME
RIGHT 90
PENCOLOR “BLACK”

REPEAT  10 [
        PENWIDTH REPCOUNT
        FORWARD 100
        PENUP FORWARD 50
        HEADING 0h
        LABEL  “PENWIDTH ” + STR REPCOUNT-1
        HEADING 3h
        BACK 50
        PENUP
        RIGHT 90
        FORWARD 20
        RIGHT 90
        FORWARD 100
        LEFT 180
        PENDOWN
]
PENWIDTH 0
HIDETURTLE
```