

Piccolo Manuale di LibreLogo

La Geometria della Tartaruga

Andreas R. Formiconi

Versione 1.1

gennaio 2018



Figura 1.1: Jan Fabre, "Searching for Utopia", 2003

Quest'opera è stata rilasciata con licenza

Creative Commons Attribuzione 2.5 Italia.

Per leggere una copia della licenza visita il sito web

<http://creativecommons.org/licenses/by/2.5/it/> o spedisci una lettera a
Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Indice

1.1	Ringraziamenti	5
I	Manuale ragionato di LibreLogo	7
1.2	Prefazione	9
1.3	Prefazione alla versione 1.0 (settembre 2017)	10
1.4	Prefazione alla versione 1.1 (gennaio 2018)	10
2	LibreLogo	13
2.1	La grafica di LibreLogo in Writer	20
3	La paura della matematica	23
3.1	<i>Mathophobia: The Fear for Learning</i>	24
4	Logo	31
5	Disegnare	35
5.1	Comandi di movimento – Disegno - Uso delle variabili	35
5.1.1	I comandi fondamentali	35
5.1.2	I codici RGB per i colori	43
5.1.3	Altri comandi	47
5.1.4	Le variabili	54
5.1.5	Lo spazio della pagina	56
5.1.6	Altri comandi grafici	60
5.1.7	Conclusioni	66
6	Ripetere	69
6.1	Cicli	69
6.1.1	Operazioni aritmetiche	77
6.1.2	Un accorgimento per trovare gli errori – la tartaruga troppo veloce!	78
6.1.3	Esercizi	78
7	Incapsulare	81
7.1	Procedure	81
8	Decidere	93
8.1	IF - AND, OR, NOT	93

9 La magia dei due specchi	95
9.1 La ricorsione	95
9.2 Verso i frattali	97
II Esperienze didattiche	101
10 L'esplorazione di Marta	103
10.1 Prologo	103
10.2 "Esercizi" di creatività – giocare con la ripetizioni in Logo	103
10.2.1 Fase I	104
10.2.2 Fase II	104
10.2.3 Fase III	105
10.2.4 Fase IV	105
10.2.5 Una prima soluzione	113
10.3 L'alternativa di Alberto	114
10.4 La soluzione matematica	123
10.4.1 Teorema del percorso chiuso	124
10.4.2 Teorema del percorso chiuso semplice	125
10.4.3 Teorema di chiusura di una procedura POLY	125
10.4.4 Codifichiamo la soluzione	130
10.5 Conclusione	133
10.5.1 Ragionamento e esplorazione	133
10.5.2 Vivere in prima persona l'esperienza che vorremmo/do- vremmo fare vivere ai nostri allievi	134
10.5.3 Percepire la potenziale dimensione verticale di LibreLogo	134
11 Girando in tondo: dal cerchio all'orbita di Halley	135
11.1 Low floor...	135
11.2 ... high ceiling	143
11.2.1 La visione analitica classica	145
11.2.2 Fisica computazionale	146
11.2.3 Le condizioni iniziali	147
11.2.4 The art of scientific computing	148
11.3 Conclusione	163
12 Appendice	165

1.1 Ringraziamenti

In primo luogo ringrazio tutti gli studenti che si sono impegnati oltre le aspettative nel Laboratorio di Tecnologie Informatiche del Corso di Laurea in Scienze della Formazione Primaria, che si è tenuto nell'anno Accademico 2016/2017. L'esplosione di creatività emersa nei liberi esercizi con Logo è stata di grande aiuto nello sviluppo di questo lavoro.

Poi ci sono stati alcuni contributi particolari, primo fra tutti quello, notevolissimo, di Marta Veloce, i cui "Esercizi di creatività" hanno ispirato il capitolo 10.

Per la seconda parte del medesimo capitolo sono invece debitore a Alberto Averono, insegnante di informatica in un istituto tecnico, che nell'ambito delle attività svolte nel Corso di Perfezionamento "Le competenze Digitali nella Scuola" (2016/2017) ha suggerito delle interessanti variazioni alla proposta di Marta. Uno splendido invito all'impiego verticale di LibreLogo.

Va ringraziata anche la studentessa, della stessa classe di Maria, Eleonora Aiazzi, con il suo testo "Se incontri un professore che ti tratta come un bambino", dove non ha parlato di coding esplicitamente ma ha colto perfettamente il senso del dispositivo didattico che abbiamo tentato di utilizzare. Feedback come questi sono fondamentali per una proposta didattica del genere e rappresentano un contributo importante nella definizione del taglio di un'opera come questa.

Abbiamo quindi Antonella Colombo, insegnante di matematica alla scuola primaria, che ci ha regalato una bellissima documentazione di apprendimento sintonico del cerchio, alla Papert. È da questa bella storia che abbiamo presso le mosse per raggiungere alfine la cometa di Halley, nel capitolo 11.

Un particolare ringraziamento va anche a Giuseppe Albano, un sicuro e raro riferimento per la competenza pedagogica ma anche tecnica. A lui sono grato per il prezioso confronto che mi consente di recuperare riferimenti che avrei faticato a trovare altrimenti, soprattutto per quanto riguarda una fase che definirei d'oro, nella quale ha visto la luce negli anni '70 - '80 un pensiero tecnico-pedagogico del quale si sono, temo, un po' perse le tracce. Un alleato importante insomma. È grazie Giuseppe che ho potuto inserire in bibliografia il testo di Horacio Reggini, Logo: "ali per la mente" [2], nel quale ho trovato corrispondenze davvero confortanti con il pensiero che sto cercando di promuovere.

Grazie anche all'amico Piero Salonia che mi ha regalato una meticolosa revisione di bozze.

In ultimo grazie a Antonio Fini, "socio" nella conduzione del Laboratorio di Tecnologie Didattiche nella veste di tutor.

Eh... ma sì, grazie al meraviglioso mondo di Linux, che con i suoi "attrezzi" - scp, ssh, rsync, grep, find, nmap, pdflatex, bibtex e via dicendo - mi dona superpoteri ignoti nel mondo dei touchscreen, consentendomi di volare leggiadramente fra router, PC di ogni tipo, server lontani. La vera Internet, la vera libertà.

Parte I

Manuale ragionato di
LibreLogo

1.2 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 Laurea 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 [13]. 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 manuale di esempi scritto dallo stesso László Németh [12] e da un manuale esteso scritto da Lakó Viktória [16]. 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* [14]. 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 [3]. 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, spaghetti, 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 sti-

molare 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.

1.3 Prefazione alla versione 1.0 (settembre 2017)

L'obiettivo, in questa estate che volge al finire, era quello di iniziare la nuova stagione didattica con tutta la seconda parte completata. Nel frattempo però ho dovuto cambiare completamente il metodo di scrittura, passando, anzi tornando nel mio caso, a L^AT_EX, l'unico modo serio di costruire un documento complesso e tipograficamente ineccepibile. E l'unico modo serio per evitare il mal di testa che inevitabilmente coglie chi si azzarda a utilizzare un *word processor* WYSWYG per redigere lavori di una certa dimensione. Questo mi ha obbligato a rimettere mano a tutto il materiale ma ne è valsa la pena. Chi è interessato a approfondire la questione di L^AT_EX può farsi avanti. È un'altra forma di *coding*, se vogliamo, nella stessa logica di HTML. Infatti è un linguaggio di *markup*, non può mancare nel bagaglio di chiunque voglia dedicarsi alle discipline STEM, e forse non solo.

1.4 Prefazione alla versione 1.1 (gennaio 2018)

Questa versione si differenzia per l'aggiunta della sezione 10.4 sulla soluzione matematica del quesito posto da una studentessa, Marta Veloce, intorno al numero di ripetizioni necessarie per la chiusura di cicli di disegno con deviazioni totali diverse da 0 o multipli di 360°. Marta pose il suo quesito durante la prima edizione del Laboratorio di Tecnologie Didattiche a Scienze della Formazione Primaria il 6 novembre 2016, inviando una decina di pagine di riflessioni in equilibrio fra esplorazione estetica e ragionamento geometrico. Lo scritto si concludeva con la formulazione di una congettura sulla chiusura delle figure geometriche emerse dalla sua esplorazione. Il testo di Marta è riportato nella sezione 10.2

Il mio dovere primario, nel ruolo di professore universitario è quello di rispondere agli studenti. Questo dovere supera quello della ricerca e supera anche la semplice "didattica erogativa", in una scala di valore, perché la domanda difficile di uno studente rappresenta il premio di un percorso dove ricerca e in-

segnamento hanno innescato una scintilla creativa nella mente di un giovane. Non c'è niente di più alto.

Domande come quella di Marta sono destabilizzanti perché è difficile rispondere. A volte la risposta non c'è. Sono domande vere, domande di ricerca, per tentare di rispondere alle quali occorre onestà intellettuale e umiltà. Discutemmo apprenditamente in classe la questione e spiegai subito che la soluzione vera, quella matematica io non l'avevo. Una soluzione matematica è quella che consente di risolvere il quesito in tutte le condizioni possibili. È una soluzione generale. In quella circostanza sviluppai una risposta che consentiva di rispondere al quesito di Marta ma solo nei casi da lei esplorati in una tabella (10.2.4) esposta al termine del suo elaborato. Si trattava cioè di una soluzione euristica, ovvero una soluzione basata su ragionevoli intuizioni ma non ancora sostenuta da un'argomentazione teorica esaustiva. La risposta, ancorché insufficiente, aveva valore didattico perché ci consentiva di mettere a fuoco il significato di verità matematica, tramite il concetto di soluzione euristica.

Successivamente, durante il corso di perfezionamento “Le competenze digitali nella scuola”, attivato presso il Dipartimento di Scienze della Formazione e Psicologia dell’Università di Firenze nell’anno accademico 2016/2017, sotto la direzione della collega Ranieri, uno dei corsisti, Alberto Averono riprese in mano la questione proponendo una soluzione informatica. L’idea era quella di fornire alla Tartaruga la capacità di riconoscere lo stato dal quale era partita in modo da potersi fermare esattamente in quel punto, a partire dal quale avrebbe solo potuto ripetere il percorso fatto. Una soluzione del genere può essere generata per via software, introducendo delle istruzioni che consentano di confrontare lo stato corrente della Tartaruga con quello iniziale. Queste considerazioni hanno consentito di mettere in luce due fatti molto importanti: il concetto di “stato” di un sistema, la Tartaruga in questo caso, e la nozione di numero digitale, quale pallida approssimazione dei numeri matematici. Questi fatti sono stati analizzati nella sezione 10.3. In ogni caso, anche se l’approfondimento di Alberto si è rivelato didatticamente assai proficuo, non ci ha fornito la soluzione matematica che desideravamo.

E infine è arrivata anche questa ma è stato necessario approfondire la teoria, cosa che non mi sarei mai aspettato di dover fare in queste circostanze. Ebbene, le questioni affrontate da Marta in sostanza sono quelle che si sono posti Abelson e diSessa [7] nel loro trattato sulla Turtle Geometry. Nella sezione 10.4 ho descritto in dettaglio i tratti essenziali della soluzione generale del problema di Marta, fornendo anche il codice per attuarla. La versione 1.1 si distingue dalla 1.0 essenzialmente per questa sezione, a parte altri marginali aggiustamenti.

Un altro aspetto peculiare della versione 1.1 è il fatto che, provvisoriamente, il capitolo 4 sia attualmente espunto e disponibile in un file pdf separato e scaricabile da <http://iamarf.ch/unifi/Papert-introduce-Logo.pdf>. Il motivo sta nel fatto che devo ancora completare il porting verso il sistema di scrittura L^AT_EX, di gran lunga più appropriato per la gestione e l'autopubblicazione di questo tipo di testo.

La prossima versione del Piccolo Manuale di LibreLogo sarà la 2.0. In questa i capitoli 3 e 4 verranno riuniti e integrati meglio rispetto a alla presente versione. Allo stesso tempo tutto i sorgenti del manuale verranno trasferiti in Github, più adeguato alla gestione di un progetto articolato e complesso.

Capitolo 2

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. Quando i computer arrivarono nelle case, negli anni '80, Logo divenne un software e come tale è stato descritto da Seymour Papert in *Mindstorms*. Per capire la valenza pedagogica del pensiero di Papert leggiamo questo brano, tratto proprio da *Mindstorms* (pp. 7-8) [14]:

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

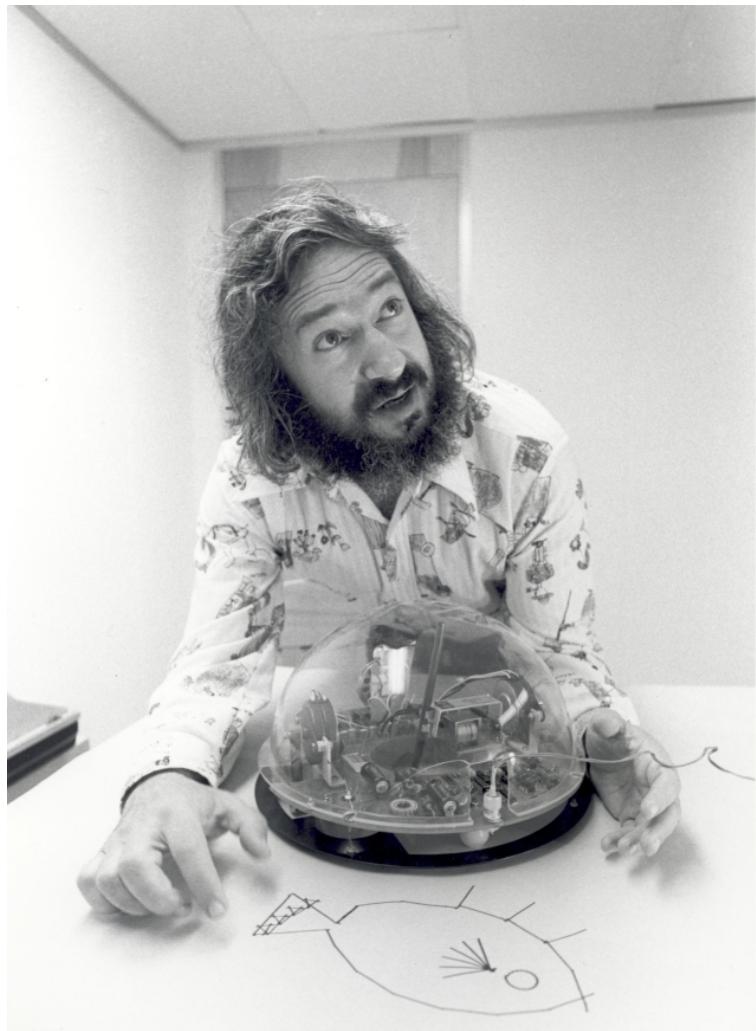
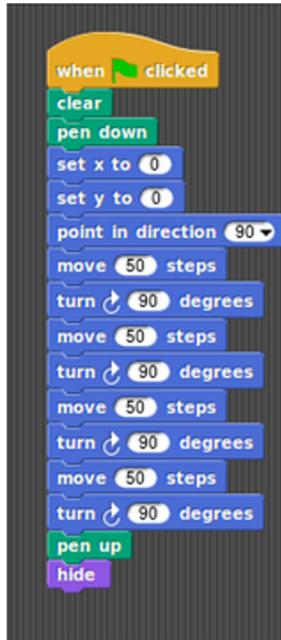


Figura 2.1: Seymour Papert mostra una delle prime versioni di Logo, quando era ancora un vero e proprio robot per disegnare.

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 schermi. 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 incastriati 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



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

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.

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” [5].

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 [9].

Particolarmente interessante è la ricerca di Colleen Lewis dove si confrontano i risultati ottenuti con Logo e con Scratch in una classe di bambini fra 10 e 12 anni [4]: se l'apprendimento di alcuni costrutti sembra facilitato da Scratch,

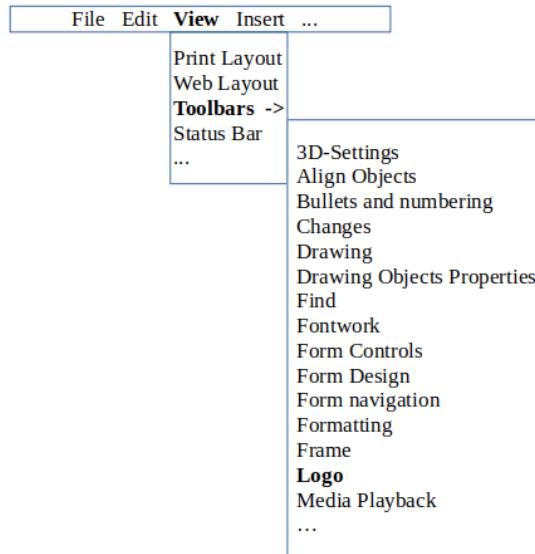
non si sono osservate differenze nella percezione degli scolari che, anzi, hanno mostrato un livello di autostima superiore se introdotti alla programmazione con Logo.

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” [8].

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 LibreOffice¹, 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 traggia beneficio. Poiché le libertà N. 2 e 4, per potere essere esercitate, richiedono la lettura del codice sorgente del software, va da sé 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 traggia 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 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

¹Esiste 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.

più diverse. Ebbene, LibreLogo è uno di questi e, dalla versione 4.0 in poi, il plugin LibreLogo è incluso di *default*² 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**:



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:

²Di *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

	FORWARD 10	Avanti di 10 punti (vedremo successivamente il significato dei punti)
	BACK 10	Indietro di 10 punti
	LEFT 15	A sinistra di -15 gradi
	RIGHT 15	A destra di 15 gradi
		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. Traduce tutti i comandi nella lingua in cui è impostato LibreOffice. Al momento della revisione di queste note (agosto 2017) mi sono accorto che nei sorgenti di LibreLogo è stata impostata la lingua italiana ma il dizionario non è mai stato compilato. Mi riprometto di farlo appena possibile in modo che la modifica venga inserita nella revisione successiva di LibreOffice.

2.1 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 regole. La caratteristica, probabilmente unica, di LibreLogo è che il risultato ottenuto girando³ uno script si ritrova sullo 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 pixel⁴. 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 regione 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 quella tipica 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, 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

³In 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.

⁴Un approfondimento della distinzione fra immagini bitmap e vettoriali può essere trovato in <http://https://iamarf.org/2014/02/23/elaborazione-di-immagini-tre-fatti-che-fanno-la-differenza-loptis/>

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.

Capitolo 3

La paura della matematica

Perché la motivazione fondamentale della genesi di Logo sta tutta nella questione, tutt'ora irrisolta, dell'insegnamento della matematica. Il linguaggio Logo è stato ideato da Papert proprio per cercare di risolvere questo annoso problema per il quale aveva coniato anche un preciso nome, *Mathofobia*, per descrivere la diffusa antipatia verso questa materia. L'interesse di Papert per la questione ha accompagnato tutta la sua vita lavorativa, che si è distesa nella seconda metà del 900. Il suo contributo è stato eccezionale, sia sotto il profilo dell'elaborazione teorica dal punto di vista pedagogico, che della creatività che lo ha condotto a concepire appositamente un linguaggio per avvicinare i ragazzi alla matematica. La profonda competenza, sia nelle questioni matematico-informatiche che in quelle pedagogiche, rende la sua opera unica e spiega la rara capacità di proporre soluzioni concrete.

La sensazione è che non molto sia cambiato, dagli anni '80 ad ora, almeno in media¹; che la motivazione iniziale, centrata su una seria e difficile rivisita-

¹Affermazione 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 utili-

zione 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 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*.

Logo fallito abbiamo detto. Ha fallito nelle intenzioni iniziali di Papert. Non è diffuso nelle scuole e non è adottato come un modo standard per sostenere l’apprendimento della matematica e delle scienze. Ma non ha fallito nel senso di non aver lasciato traccia, al contrario. Ci sono molte versioni del logo in tutto il mondo, alcune delle quali sono diventate importanti strumenti di indagine educativa, ad esempio nel campo della simulazione di sistemi biologici complessi. Ed è sempre da Logo che ha preso spunto il vasto mondo dei linguaggi visivi a blocchi, primo fra tutti Scratch. Logo e Scratch non sono in opposizione. In un certo senso, Scratch deriva da Logo e “contiene” molte delle sue funzionalità. Molte delle cose che puoi fare in Logo possono essere fatte anche in Scratch. Ma Scratch è molto più orientato alla costruzione ”del tuo videogioco” o allo storytelling. Il problema, tuttavia, è che questa gamma più ampia di possibilità, in un contesto scolastico caratterizzato da un basso livello di istruzione tecnologica, ha finito per disperdere le intenzioni educative originali di Logo. Una delle intenzioni di questo manuale è quella di recuperare l’originale ”sapore matematico” della pratica del *coding* a scuola. Nella prossima sezione commentiamo quella che Papert ha chiamato *mathophobia*, una sorta di malattia che Logo era destinato a combattere.

3.1 *Mathophobia: The Fear for Learning*

Seymour Papert ha scelto di intitolare il secondo capitolo di *Mindstorms* ”*Mathophobia: The Fear for Learning*”. Il suo punto di partenza è la divisione schizofrenica tra scienze umane e scientifiche, una divisione che è profondamente radicata nel linguaggio, nella visione del mondo, nell’organizzazione sociale e nel sistema educativo. Una divisione che negli ultimi 30-40 anni di deriva neoliberista, si è ulteriormente ampliata. Ad esempio, nelle università, a partire dagli anni ’80, gli accademici si battono per ottenere i finanziamenti necessari per le loro ricerche. Purtroppo, l’altra faccia della medaglia è che quasi nessuno si preoccupa di insegnare, o molto poco. La carriera di un professore non dipende dalla qualità del suo insegnamento, ma quasi solo dalla quantità della sua produzione scientifica. Le conseguenze sono pessime. Gli accademici ten-

taristici 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.

dono a trasformarsi in manager quando fanno ricerca e in funzionari pubblici quando insegnano: imprenditori dinamici da un lato e (forti) conservatori dall'altro. In questo modo, anche nelle scienze umane si assiste a una deriva tecnica del ruolo accademico. La missione dell'insegnamento, che in un certo senso è il "lato umanistico" del lavoro, si riduce a una sorta di Cenerentola. Così, la dicotomizzazione tra scientifico e umanistico è ancora più forte e sbilanciata.

La questione non è quella di un equilibrio adeguato, bensì di spezzare la linea di demarcazione tra le due culture. Papert guardò il computer come una forza per attenuare la distinzione. La pratica del *coding* è stata pensata come un modo per introdurre una matematica più umanistica e per sfruttare elementi di pensiero scientifico nelle scienze umane. È in questo contesto che Papert ha parlato di una *Mathland*, dove la matematica costituirebbe un vocabolario naturale, con l'idea che potremmo cambiare non solo il modo in cui insegniamo la matematica, ma anche il modo in cui la nostra cultura concepisce la conoscenza e l'apprendimento.

Papert sostiene che le sue argomentazioni non si limitano all'apprendimento della matematica, ma riguardano anche l'atteggiamento nei confronti dell'apprendimento in generale. La parola *mathphobia* suggerisce due associazioni. Una è il diffuso timore per la matematica. L'altra deriva dalla radice "math", che in greco antico significa apprendimento in senso generale. Così, se i bambini iniziano ad essere abili studenti spontanei, in seguito "imparano" la paura di imparare qualsiasi cosa e non solo la matematica. Ironia della sorte, sembra che più si studia, più si ha paura di imparare.

I bambini imparano migliaia di parole prima di entrare in prima elementare. Meno evidente per molte persone è il fatto che i bambini imparano una grande quantità di matematica pure. Tra queste conoscenze prescolare ci sono per esempio nozioni come la conservazione del volume di liquidi in recipienti di forma diversa, o l'indipendenza del numero totale di oggetti dall'ordine con il quale sono stati contati. Papert ha chiamato questo

apprendimento Piagetiano, un processo di apprendimento che ha molte caratteristiche che le scuole dovrebbero invidiare: è efficace (tutti i bambini ci arrivano), è economico (sembra non richiedere né insegnanti né uno sviluppo del curriculum), ed è umano (i bambini sembrano farlo in uno spirito spensierato, senza esplicita ricompensa esterna e punizione).

Infatti, le pratiche dell'insegnamento della matematica sottovalutano in larga misura l'apprendimento piagetiano, imponendo conoscenze formali che risultano per lo più dissociate dalle precedenti nozioni spontanee. Le conseguenze per i futuri adulti sono pesanti. La perdita dell'atteggiamento positivo del bambino nei confronti dell'apprendimento è un fenomeno molto comune nell'età adulta e non riguarda solo la matematica:

La deficienza diventa identità: "Non posso imparare il francese, non ho orecchio per le lingue"; "Non potrei mai essere un uomo d'affari, non ho una testa per le cifre".

Circa l'80% dei miei studenti di scuola primaria si dichiara "distante dalla matematica". Molti sostengono di avere difficoltà anche con le tecnologie, nonostante si supponga che siano "nativi digitali".

L'idea che ci siano persone intelligenti e stupide ai fini di una data attività è diffusa. È estremamente difficile sradicare i pregiudizi sui propri atteggiamenti. Il punto è che le convinzioni accettate circa l'attitudine matematica non sono dimostrabili. Per rafforzare il concetto, Papert ha riformulato l'argomento come segue [14] (p. 43):

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.

Spesso è la scuola che costruisce le attitudini:

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 predilige l'ordine logico può avere problemi con la sillabazione dell'inglese e magari finire col detestare la scrittura.

L'idea di Papert era che si possono usare i computer come veicoli per risolvere situazioni come quella di Jim o quella di bambini che, all'opposto, amano la

logica, ma presentano problemi di tipo dislessico. Entrambi i casi sono vittime della netta separazione della nostra cultura tra "il verbale" e "il matematico". Egli immagina una sorta *Mathland* dove l'amore di Jim per la lingua e la sua abilità potrebbero essere mobilitati per realizzare l'apprendimento matematico formale, invece di contrastarlo, mentre per l'altro tipo di bambini, l'amore per la logica potrebbe nutrire l'interesse per la linguistica. I metodi di insegnamento prevalenti danno agli studenti di matematica possibilità limitate di reperire un senso in ciò che stanno imparando. Di conseguenza, i bambini sono costretti a seguire il modello peggiore per l'apprendimento della matematica, che è l'apprendimento mnemonico, dove l'oggetto di studio sembra non avere senso. È quello che Papert chiamava un modello di apprendimento *dissociato*.

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 sotto²:

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

²[NdR] Ho lasciato la versione originale, mi pare inutile "tradurre" un pezzo simile, ai fini della comprensione del concetto. Ho riportato questo esempio perché verrà ripreso, in altra forma, in uno degli esempi più avanzati di impiego di Logo. È interessante che ciò che negli anni 70 richiedeva un potente computer e uno staff di ricerca avanzato ora si possa fare con una semplice implementazione di Logo in un PC.

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.

Papert è perentorio: spesso i bambini non riescono a capire a cosa servano realmente matematica e grammatica perché percepiscono quelle degli adulti come spiegazioni volutamente ambigue.

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 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*³.

È importante tenere presente la differenza tra la matematica - un vasto campo di indagine la cui bellezza è raramente sospettata dalla maggior parte dei non addetti ai lavori - e la *matematica scolastica*. Quest'ultima è una sorta di costruzione sociale, cioè un insieme di argomenti matematici determinati da una successione di circostanze specifiche. Un processo che non garantisce, *per se*, il raggiungimento di un risultato ottimale. Ricorda la storia del layout della tastiera noto con la sigla QWERTY, che rappresenta i primi cinque tasti delle righe superiori. Questo accordo non ha una spiegazione razionale, ma solo storica. Fu introdotto perché i tasti delle prime macchine da scrivere tendevano ad incepparsi. Furono disposti così per ridurre le collisioni, separando i tasti che si susseguivano più frequentemente. La tecnologia delle macchine da scrivere migliorò rapidamente e in pochi anni il problema dell'inceppamento non fu più un problema ma la disposizione QWERTY rimase tal quale. A questo punto troppe persone conoscevano il layout QWERTY e la produzione di macchine da scrivere era troppo lontana per fare un passo indietro nella riprogettazione di un layout più razionale, ad esempio raggruppando i tasti più usati.

La questione della tastiera QWERTY è un buon esempio di come le abitudini consolidate non rappresentino necessariamente la scelta migliore. Anche la matematica scolastica si è formata in un contesto storico diverso. Allo stesso modo questa idea di matematica si è consolidata profondamente e, ancora oggi, per la maggior parte delle persone è inconcepibile che la matematica possa essere anche qualcos'altro. Ricordo un noto professore di analisi matematica che, all'inizio del primo anno del corso di laurea in matematica, esortava i suoi studenti a dimenticare ciò che avevano imparato al liceo, poiché la matematica era un'altra cosa.

La geometria della tartaruga è stata concepita per adattarsi ai bambini e, prima di tutto, per essere "appropriabile". Potremmo descrivere questo concetto attraverso alcuni principi. In primo luogo, il principio di continuità: le nuove conoscenze matematiche devono essere in continuità con quelle esistenti, quelle che i bambini hanno prima di andare a scuola. Poi il principio di potenza: la nuova conoscenza deve permettere agli studenti di realizzare progetti personali significativi, che non potrebbero essere realizzati senza di essa. Infine, il principio della risonanza culturale: i nuovi concetti devono avere un senso per i bambini nel loro contesto sociale; paradossalmente, anche nel contesto sociale degli adulti: non dobbiamo infliggere ai bambini qualcosa che non abbiamo compreso a fondo e, purtroppo, spesso con la matematica scolastica è proprio così.

³[NdR] Corsivo dell'autore.

Capitolo 4

Logo

Questo capitolo è un'introduzione all'uso del logo. Si rivolge principalmente agli studenti di formazione primaria, cercando di mostrare come iniziare i bambini alla pratica del logo. È interessante notare che, avendo proposto questo argomento a diverse centinaia di studenti, mi sono reso conto come insegnare il Logo ai bambini o ai miei studenti sia molto simile. E ancora più come, in media, nel primo approccio i bambini siano più bravi! Ne discutiamo spesso con quegli studenti che affermano di aver avuto grandi difficoltà a parlare con la tartaruga. La maggior parte di loro sperimenta una sorta di "risurrezione" dopo alcune lotte iniziali: "All'inizio non ho capito niente, ma poi... che meraviglia! Ma alcuni non si liberano mai dal malumore, almeno nelle due settimane e mezza di laboratorio. Parlare con questi studenti è sempre molto interessante perché sono stupiti quando dico loro che, il più delle volte, i bambini di 9 anni vanno d'accordo con la tartaruga abbastanza facilmente. Quando si cerca di scavare in questo paradosso, sembra che i bambini colgano naturalmente l'aspetto ludico della situazione. D'altra parte, gli adulti si lasciano facilmente intrappolare nei loro pregiudizi: non ho una testa per i numeri, io e i computer siamo molto lontani, non sono mai andato d'accordo con le tecnologie. Soprattutto l'ultima è un'affermazione che ho sentito tante volte, come rilevato nel capitolo precedente, e, sorprendentemente, da veri nativi digitali. Cioè, una persona che può avere migliaia di contatti su Facebook, candidamente può poi sostenere di non andare d'accordo con le tecnologie.

Detto questo, ora affrontiamo la tartaruga con lo spirito più giocoso possibile, a partire da zero. Dato che suppongo sappiate bene come i bambini affrontano un nuovo gioco, provate a fare lo stesso. Quindi, se avete scaricato LibreOffice e attivato la barra degli strumenti Logo, come spiegato nel primo capitolo, apriete un nuovo documento di testo. Vi trovate di fronte a un foglio bianco: qui potete scrivere una lettera, una lista della spesa, una poesia o quello che volete. Provate a scrivere questo¹:

FORWARD 100

¹Uso i caratteri maiuscoli solo per chiarezza, LibreLogo è "insensibile alle maiuscole e minuscole"

Poi premete il pulsante "Play" nella barra degli strumenti (quello con la freccia verde, ignorate gli altri pulsanti per il momento) Cosa vedete? Dimenticate i vostri "limiti" e tutte le opinioni che potete avere sui vostri rapporti con la matematica, la tecnologia e così via. Guardate il risultato ottenuto e confrontatelo con quello che avete scritto. Ponetevi delle domande. Sperimentate la tartaruga sulle possibili risposte che vi vengono in mente. Se si desidera avviare un nuovo disegno scrivere i seguenti comandi immediatamente prima del precedente:

```
CLEARSCREEN  
HOME
```

Con il primo comando (CLEARSCREEN) si annulla il disegno precedente, con il secondo (HOME) si manda la tartaruga in posizione "home", cioè al centro del foglio con il naso rivolto verso l'alto. Quindi, provate per esempio

```
CLEARSCREEN  
HOME  
FORWARD 50
```

Di nuovo, premete il pulsante "Run" (Esegui). Cosa è cambiato?

Ora, per esplorare meglio ciò che la Tartaruga può fare, vi dico un paio di altri comandi: DESTRA e SINISTRA. Non è difficile immaginare a cosa servono questi comandi, non è vero? Ma manca qualcosa, per poterli usare, cosa? Beh, cercate di indovinare e sperimentate. Per favore, riflettete su cosa sta realmente succedendo alla Tartaruga, quando applicate questi comandi. Poi, non rimane che giocare per vedere cosa si può fare. Ricorda, come se foste bambini...

Il motivo per cui insisto così tanto su ciò che i bambini possono fare è che ho avuto diverse opportunità di lavorare con loro e con la tartaruga. Davvero pochi si lamentano della difficoltà, i più semplicemente giocano e si entusiasmano facendo creare al piccolo animale cose divertenti. Una volta, cercando di disegnare una casa, una bambina tirò fuori una sorta di castello bizzarro. La elogiai commentando quanto fosse meraviglioso questo disegno e lei era così orgogliosa. Poi, quando sono tornato da lei, dopo essermi dedicato alle opere di altri bambini, l'ho trovata in lacrime che singhiozzava disperatamente: aveva accidentalmente cancellato i comandi in un modo che io non riuscii più a recuperare. Mi sono senti molto in colpa per non averle insegnato a salvare le opere di tanto in tanto.

Pertanto, se iniziate a creare qualcosa di più complesso che vi piace, non dimenticate di salvare il documento di tanto in tanto. E' così facile.

In una versione passata del libro questo capitolo era molto più lungo. Dopo l'esperienza che ho accumulato negli ultimi due anni, sia lavorando con gli studenti che con i bambini, preferisco fermarmi qui, invitandovi piuttosto a esplorare liberamente le possibilità. Da qui si può andare in diverse direzioni. Se volete indicazioni ed esempi specifici per usare i comandi di base, o se volete scoprire istruzioni più potenti, le troverete nel prossimo capitolo e in quelli successivi. Nel capitolo II, per scoprire fino a che punto si può giungere esplo-

rando, potete leggere la storia di Marta, una mia ex studentessa. Oppure, se volette leggere un modo affascinante di introdurre i bambini al disegno di un cerchio, andate a leggere il capitolo 11, o meglio, la prima parte di esso, altrimenti comincerete a volare....

Capitolo 5

Disegnare

5.1 Comandi di movimento – Disegno - Uso delle variabili

5.1.1 I comandi fondamentali

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

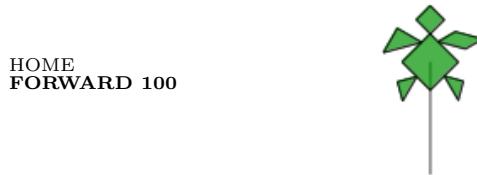
Apri un nuovo documento di testo e scrivi questo comando:

HOME

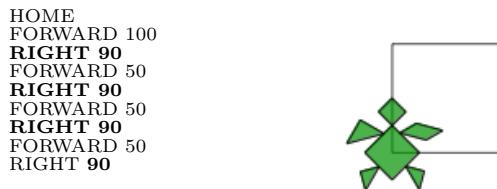
Vedrai che è apparsa la tartaruga in mezzo al foglio con la testa verso l'alto:



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



La tartaruga si è mossa tracciando una linea; è così che si disegna dando comandi alla tartaruga. Scriviamo ora le seguenti istruzioni:



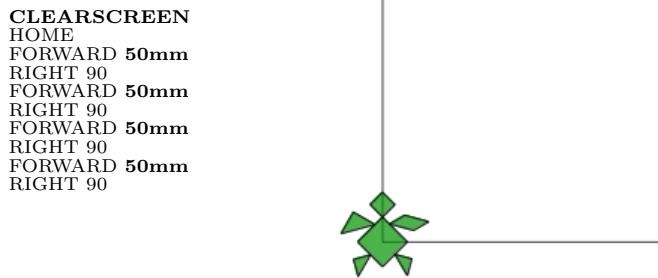
Abbiamo tracciato quattro lati, e alla fine di ogni lato abbiamo girato a destra di 90°, ottenendo così un quadrato.

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 cui 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

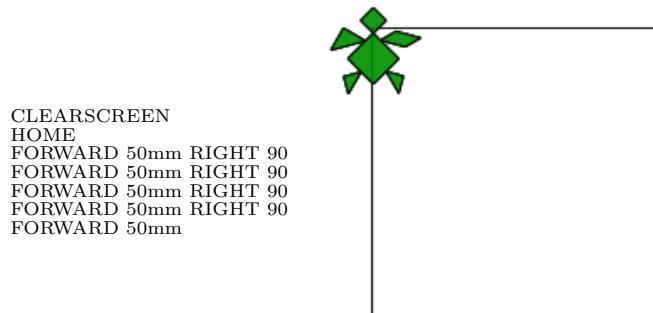
il numero 50 esprime la lunghezza del percorso che la tartaruga deve compiere. Puoi verificare subito cambiando il valore e guardando cosa fa la tartaruga. Ma cosa rappresenta quel 50? Sono punti tipografici (point): 50 pt. Un punto sono 0.35 mm¹. LibreLogo capisce le unità di misura, si può scrivere 50, 50pt, 50mm, 50cm, 50in (inch: pollice), 50 " (" sta per pollice). Ovviamente si tratta di lunghezze diverse. Usiamo per esempio i mm:



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

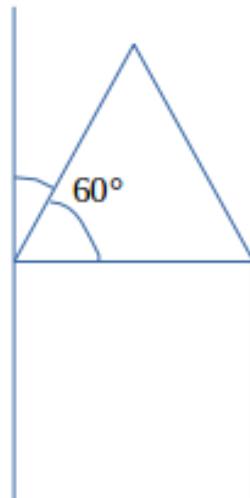
Proviamo a complicare il disegno, immaginando di fare una casetta. La tartaruga si trova nel vertice in basso a sinistra e guarda in alto. Come prima cosa dobbiamo farla salire fino al vertice in alto a sinistra. Proviamo, e allo stesso tempo approfittiamo anche del fatto che le istruzioni possono essere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

¹La definizione precisa è: 1 pt = 1/72 pollici, dove 1 pollice = 25.4 mm. Quindi 1 pt = 2.54/72 = .3527 mm



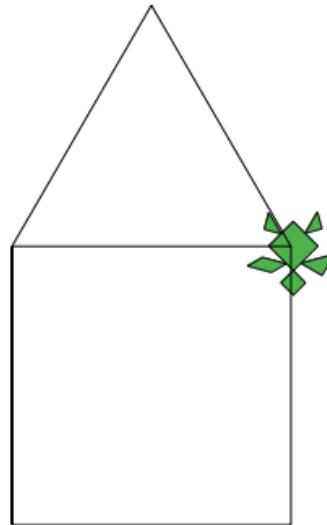
Non ci sono regole precise per raggruppare le istruzioni in una stessa riga. Il raggruppamento si può decidere in base alla comodità con cui si rilegge il codice. È importante facilitarsi la vita perché via via che il codice cresce può complicarsi rapidamente e tutti gli accorgimenti per renderlo più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

Ora dobbiamo costruire il tetto sulla casa. Facciamo questo appoggian-
do sul quadrato un triangolo equila-
tero, con la base coincidente con il
lato superiore del quadrato. Essen-
do 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 di-
rezione. Di quanto? Siccome gli an-
goli interni di un triangolo equila-
tero sono di 60° , la tartaruga do-
vrà deviare di $90^\circ - 60^\circ = 30^\circ$ a destra.
Arrivata in cima, disegnan-
do la falda sinistra del tetto, dovrà
girare a destra di 120° per poi dis-
egnare la falda destra. Infine, girerà
di 30° a destra per riallinearsi alla
parete della casa.



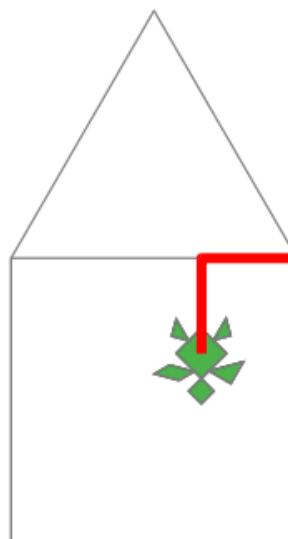
5.1. COMANDI DI MOVIMENTO – DISEGNO - USO DELLE VARIABILI39

```
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.

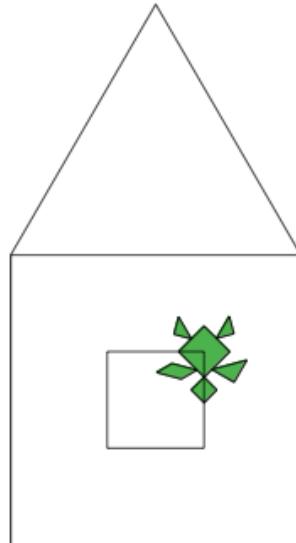
```
CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 120
PENUP
FORWARD 50mm/3 LEFT 90
FORWARD 50mm/3
dove abbiamo evidenziato in rosso il percorso fatto senza disegnare.
```



```

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

```

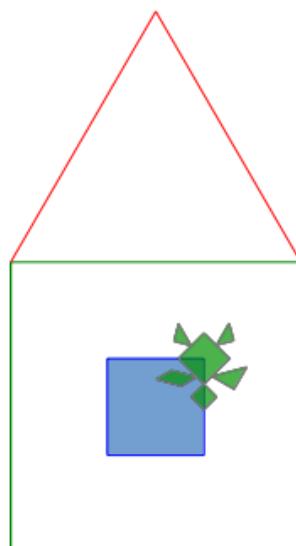


Proviamo ora ad arricchire ulteriormente il disegno, per esempio facendolo a colori.

```

CLEARSCREEN
HOME
PENCOLOR "green "
FORWARD 50mm 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 "

```



5.1. COMANDI DI MOVIMENTO – DISEGNO - USO DELLE VARIABILI 41

E perché non colorare anche l'interno?

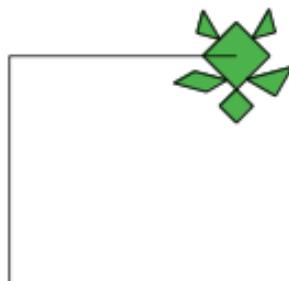
```
CLEARSCREEN  
HOME  
FORWARD 50mm RIGHT 90  
FILLCOLOR "yellow" FILL  
FORWARD 50mm RIGHT 120  
FORWARD 50mm RIGHT 120  
PENUP  
FORWARD 50mm/3  
LEFT 90  
FORWARD 50mm/3  
PENDOWN  
FILLCOLOR "red" FILL  
FORWARD 50mm/3 RIGHT 90  
FORWARD 50mm/3 RIGHT 90  
FORWARD 50mm/3 RIGHT 90  
FORWARD 50mm/3 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.

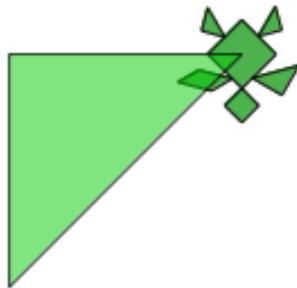
L'istruzione FILL in realtà fa due cose: chiude una figura e la riempie di un colore. Facciamo questa prova:

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



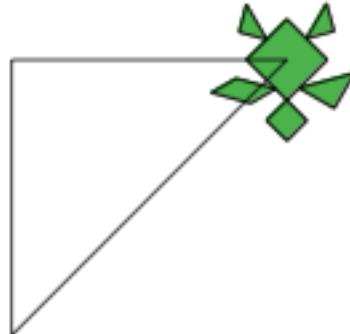
Così abbiamo disegnato una figura che non è chiusa. Volendo fare un triangolo potremmo far disegnare alla tartaruga il terzo lato. Alternativamente possiamo chiudere la figura con **FILL**, come abbiamo visto prima:

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



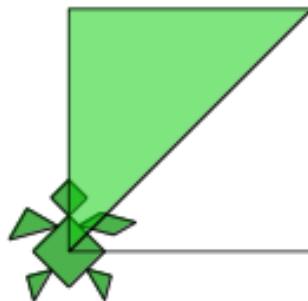
Così la figura è stata chiusa e colorata. Possiamo anche chiuderla senza colorarla, usando l'istruzione **CLOSE**:

```
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!

5.1.2 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 le 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]

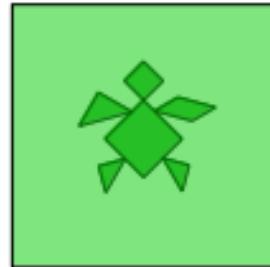
Provate a giocare un po' con questi numeri per vedere che colori vengono fuori.

5.1. COMANDI DI MOVIMENTO – DISEGNO - USO DELLE VARIABILI45

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'.

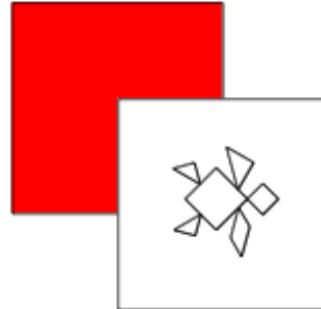
Per facilitarci, anticipiamo un nuovo comando. Negli esempi precedenti noi abbiamo già disegnato un quadrato, utilizzando le istruzioni FORWARD e RIGHT, opportunamente combinate. In realtà, per disegnare le principali figure geometriche, in Logo esistono delle istruzioni preconfezionate, che ci aiutano 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)



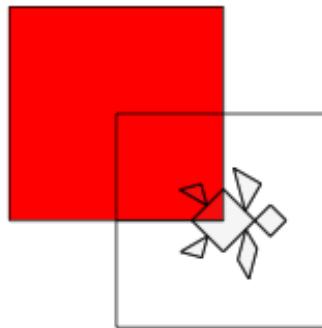
In questo modo la tartaruga disegna un quadrato e poi si piazza al centro con la testa rivolta in alto (il colore non l'abbiamo determinato noi ma venuto "automaticamente", si dice che è il valore di default). Serviamoci allora di questa istruzione per disegnare prima un quadrato che coloriamo di rosso, 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] 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 ritro-

viamo 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 HIDE TURTLE.

5.1.3 Altri comandi

on 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 forme 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:

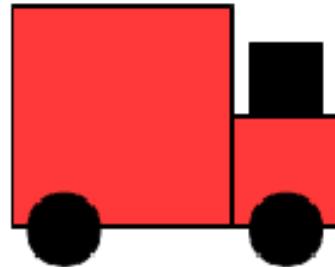
```
CLEARSCREEN
HOME
SQUARE 50
CIRCLE 50
```



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'argomento² del comando SQUARE rappresenti il lato del quadrato e quello del comando CIRCLE il diametro del cerchio che vogliamo costruire.

²Con "argomento" si intende il valore che un comando richiede per essere effettuato. Gli argomenti possono essere anche più di uno – vedremo degli esempi.

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.

5.1. COMANDI DI MOVIMENTO – DISEGNO - USO DELLE VARIABILI 49

```
COLOR "red "
SQUARE 60
PENUP BACK 15 RIGHT 90
FORWARD 45
LEFT 90 PENDOWN
SQUARE 30
PENUP FORWARD 25 PEN-
DOWN
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 e 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 chiudendo 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 figura come questa qui sotto



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

```

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 [3], 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 sole 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).

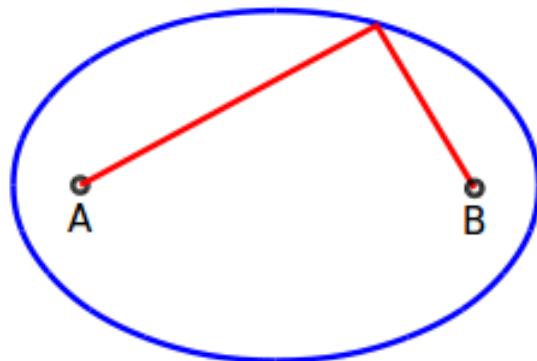
Guardiamo, ad esempio, due matite disposte in verticale su un tavolo. Se vengono illuminate dal sole accade che anche le ombre

5.1. COMANDI DI MOVIMENTO – DISEGNO - USO DELLE VARIABILI 51

sono parallele; se invece è una lampada che le illumina le ombre si divergono.

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.



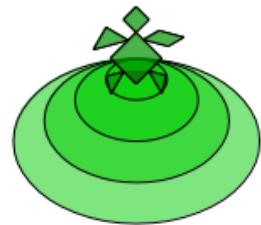
In LibreLogo l'ellisse si disegna con il comando

ELLIPSE[40, 20]

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:



5.1. COMANDI DI MOVIMENTO – DISEGNO - USO DELLE VARIABILI 53

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
```

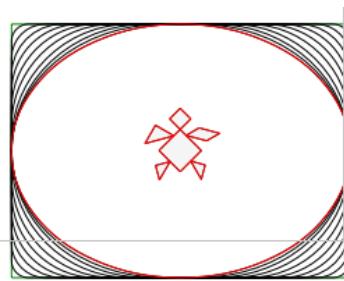
Queste istruzioni consentono di usare anche altri parametri per ottenere delle varianti di rettangoli e ellissi. Nel caso dei rettangoli è possibile aggiustare un terzo parametro in maniera da arrotondare i vertici:

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

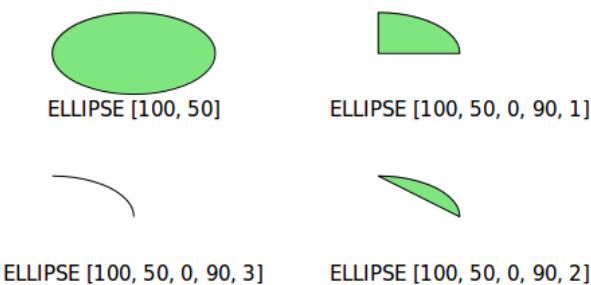


Supponiamo ora che un amico ci abbia appena detto di questa possibilità ma che altro non si ricordi. Si può controllare la "rotondità" dei vertici? Probabilmente con quel terzo parametro, che l'amico ci ha detto potevamo fissare al valore di 10, ma come funziona? Questo è un ottimo esempio per mettere 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.



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).

5.1.4 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 argo-

menti 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

LAT0 = 100
ANGOL0 = 90

FORWARD LAT0
LEFT ANGOL0
FORWARD LAT0
LEFT ANGOL0
FORWARD LAT0
LEFT ANGOL0
FORWARD LAT0
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 "LAT0" 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 LAT0 = 100, cambiandola per esempio con LAT0 = 150. Prova a sperimentare! Puoi anche cambiare il valore di ANGOL0, 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 possibile 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 battaglie 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]
```

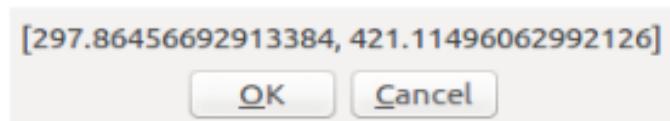
Se si esegue questo frammento di codice, prima viene stampato "valore" [200, 300], poi il valore 200, quindi 300. Da qui si capisce che con $P[0]$ si ottiene il primo elemento del vettore posizione, che contiene il numero 200, e con $P[1]$ il secondo elemento, che contiene il numero 300. Ecco, questo è quanto dovrebbe bastare per andare a vedere com'è che si può controllare con ancora maggiore agilità la posizione della tartaruga nel foglio.

5.1.5 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:

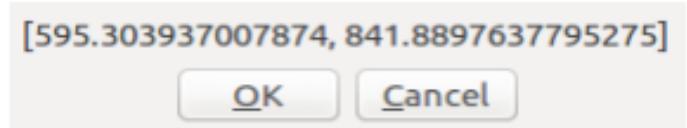


I due numeri fra parentesi rappresentano le coordinate x e y della posizione nello spazio della pagina: 298 e 421³ rispettivamente. Dal momento che abbiamo appena aperto il documento e che all'inizio la tartaruga viene piazzata al centro, possiamo assumere che queste coordinate rappresentino il centro 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

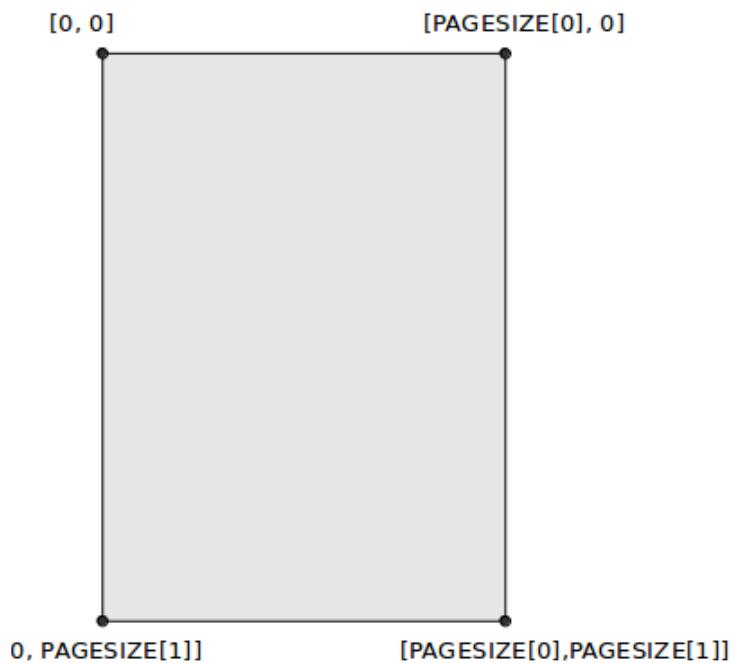
PRINT PAGESIZE

otteniamo:

³Abbiamo 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.



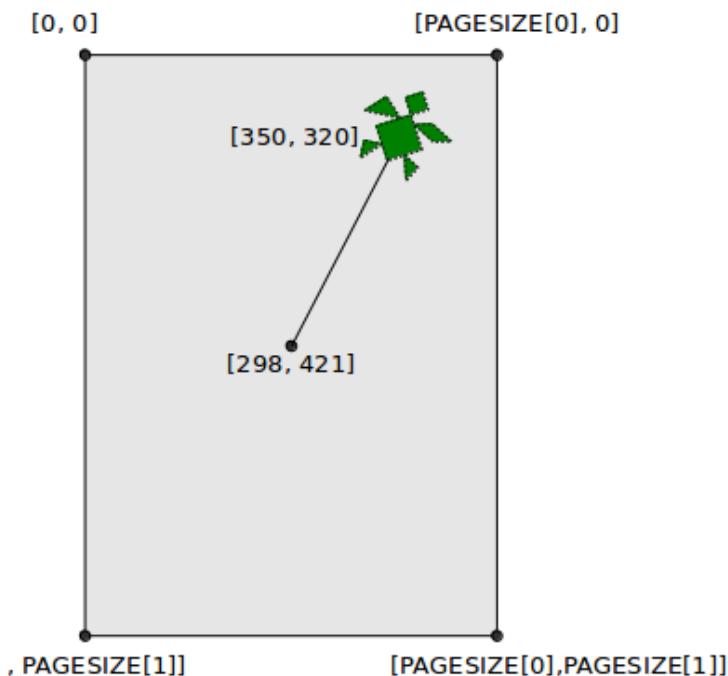
Questi numeri, arrotondati, 596 e 842, rappresentano le dimensioni di un foglio A4 espressi in "punti" alla densità di 72 DPI/footnoteL'acronimo DPI sta per dots per inch: punti per pollice. Il valore di DPI dipende dal supporto fisico su cui si intende che un'immagine debba essere rappresentata. Poiché 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 familiare. 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.. In mm risultano 210 e 297 mm. Riassumiamo la situazione con la seguente figura.



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. Nella tabella successiva vediamo i valori numerici corrispondenti:

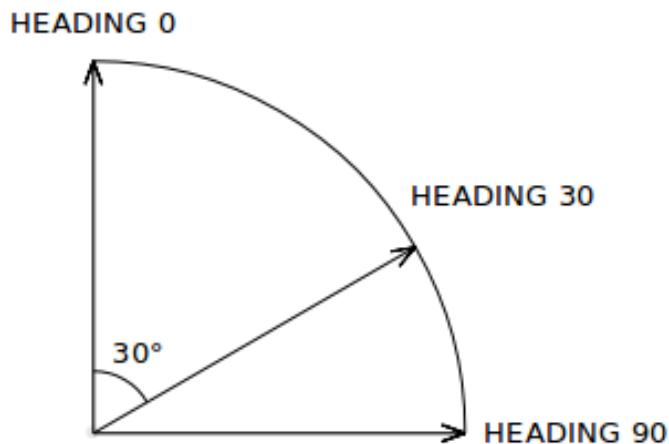
[0, 0]	[0, 0]
[PAGESIZE[0], 0]	[596, 0]
[0, PAGESIZE[1]]	[0, 842]
[PAGESIZE[0], PAGESIZE[1]]	[596, 842]

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?



Con l'istruzione **POSITION [350, 320]** la tartaruga si dirige direttamente al punto di coordinate **POSITION [[350,320]]**, a partire dal punto dove 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.



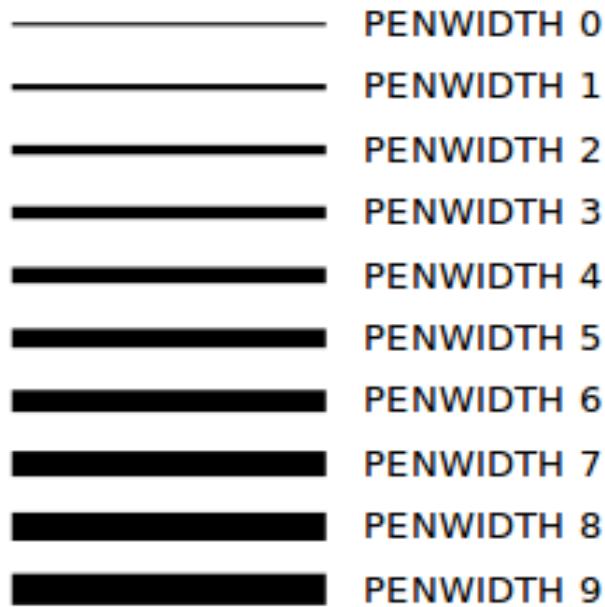
Quando si apre un documento nuovo, oppure dopo l'istruzione HOME, la tartaruga punta verso il lato superiore del foglio, e questa direzione corrisponde a 0° .

5.1.6 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:



PENJOINT (forma dei vertici)

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

Disegniamo per esempio un triangolo, nel modo seguente:

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



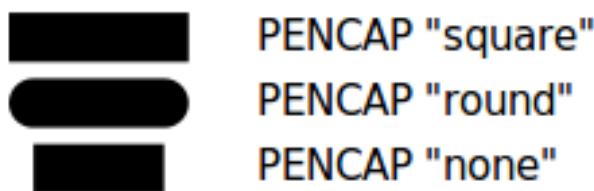
Possiamo alterare la rifinitura dei vertici facendo precedere il codice precedente dall’istruzione **PENJOINT** e un opportuno argomento. Le possibilità sono le seguenti:

-  PENJOINT "rounded" (default)
-  PENJOINT "miter"
-  PENJOINT "bevel"
-  PENJOINT "none"

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 per "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.

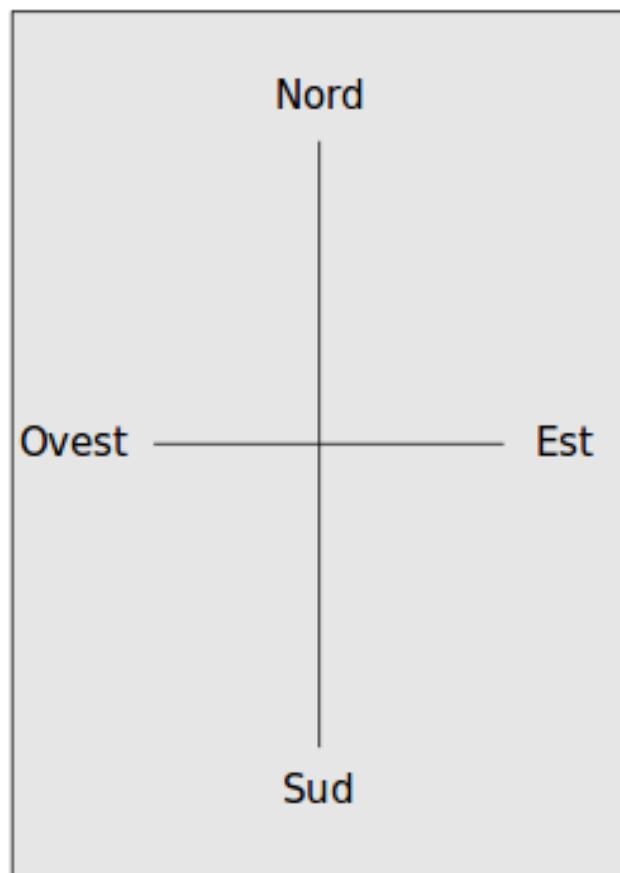
PENCAP (forma delle estremità dei segmenti)

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	; nasconde la tartaruga
PENWIDTH 15	; imposto lo spessore della linea a 15 pt
 RIGHT 90	 ; ruoto 90\degree 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\degree (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\degree (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:



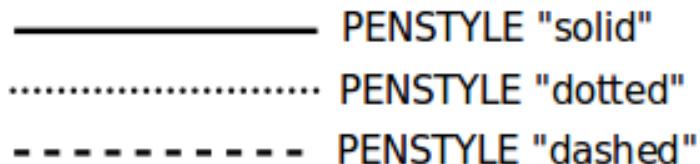
PENCAP "square"
PENCAP "round"
PENCAP "none"

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:

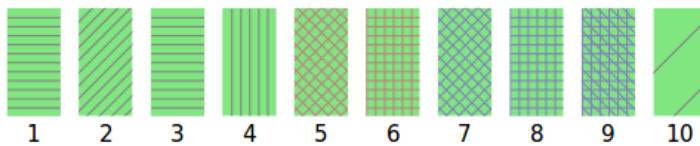


Queste sono le regole:

- 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°]`



5.1.7 Conclusioni

Con queste istruzioni si conclude la parte della nostra esplorazione di LibreLogo dedicata al disegno. 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 codifichino 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 poiabbiamo 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 arrotondati. 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

5.1. COMANDI DI MOVIMENTO – DISEGNO - USO DELLE VARIABILI

67

– distanze, angoli e altro – senza doversi preoccupare di assegnare loro precisi valori numerici; una caratteristica fondamentale 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.

Capitolo 6

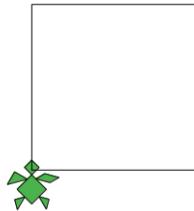
Ripetere

Nella presente versione 0.9.1, 31 agosto 2017, questo capitolo per il momento può essere consultato nella versione 0.4 del 9 settembre, reperibile all'indirizzo <http://iamarf.ch/unifi/Piccolo-manuale-LibreLogo.pdf>. In questa versione il capitolo si trova alle pagine 73-85.

6.1 Cicli

Introduciamo i cicli attraverso un piccolo studio geometrico. Riprendiamo il disegno di un quadrato, così come l'avevamo fatto nella sezione 5.1.1¹.

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



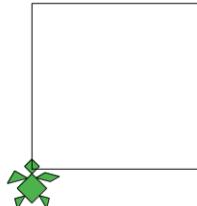
Modifichiamo questo codice usando le variabili, che avevamo visto nella sezione 5.1.4.

¹L'unica differenza rispetto al quadrato disegnato precedentemente 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 prima avevamo fatto per poter iniziare il disegno del tetto della casa.

```

CLEARSCREEN
HOME
L = 50mm
A = 90
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A

```

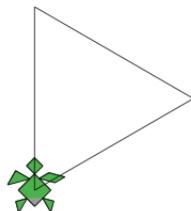


dove L è il lato del quadrato e A l'angolo interno ad ogni vertice. È facile modificare questo codice per disegnare altre figure, e in particolare per disegnare poligoni regolari. Proviamo ad esempio a costruire un triangolo equilatero. Come si potrebbe fare? Facile: si toglie un lato e si aggiusta la 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 $120^\circ = 360^\circ/3$:

```

CLEARSCREEN
HOME
L = 50mm
A = 120
FORWARD L RIGHT A
FORWARD L RIGHT A
FORWARD L RIGHT A

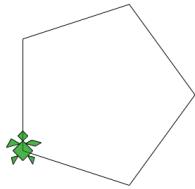
```



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 $72^\circ = 360^\circ/5$ 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
N = 5
A = 360/N
FORWARD L RIGHT A
```



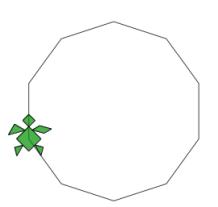
dove L è il lato del poligono, N il numero di lati e A la misura degli angoli interni.

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 convessa² 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 $P = L \cdot N$. Questa formula ci consente di fissare il perimetro P e il numero di lati N , ricavando la lunghezza del lato: $L = P/N$. Scateniamoci: disegniamo un decagono regolare (10 lati)

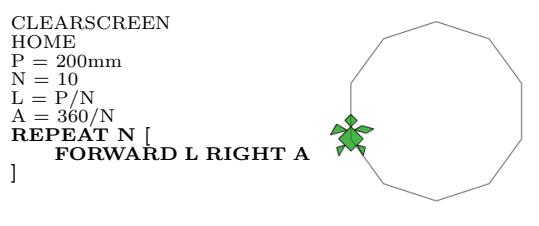
```
CLEARSCREEN
HOME
P = 200mm
N = 10
L = P/N
A = 360/N
FORWARD L RIGHT A
```



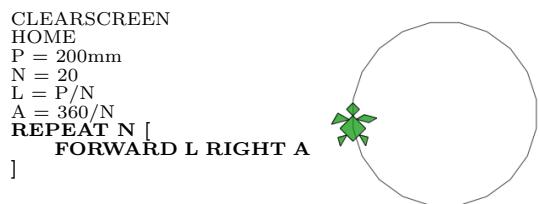
Soddisfacente no? Anche perché così stiamo mettendo a frutto l'utilità delle variabili (sez. 5.1.4), che forse alla prima non ci era parsa così chiara. Qui il fatto è evidente: possiamo inserire i dati numerici indispensabili e poi far calcolare i parametri derivati attraverso formule che utilizzano variabili simboliche. Una bella generalizzazione! Ma non siamo soddisfatti, a dire il vero. Infatti, per conseguire il nostro obiettivo di un 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 strida con la bell'idea di immaginare poligoni regolari qualsivoglia... e qui vengono i “cicli”.

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:

²Detto in italiano, una figura è convessa se il suo perimetro non ha “bozze” verso l'interno.



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 mole 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...



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 $\tilde{}$ (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
```

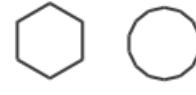
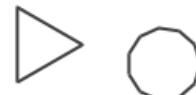
```
FORWARD L RIGHT A
```

```
]
```

```
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.

6.1.1 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:

+	Somma	$10+3=13$
-	Sottrazione	$10-3=7$
*	Moltiplicazione	$10*3=30$
/	Divisione	$10/3=3.\overline{3}$
//	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 qualiasi in Writer, e LibreLogo è attivo, volendo si ha disposizione anche una calcolatrice. Con l'occasione, anticipiamo l'istruzione **LABEL** “**pip-po**” che scrive il testo “**pippo**” nel punto in cui si trova la tartaruga. Se invece si ha una variabile, per esempio di nome A, e se ne vuole scrivere il contenuto nel documento, allora basta eseguire **LABEL A 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.

6.1.2 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.

6.1.3 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 *icoding* 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:



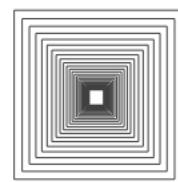
E se da questo fiocco tirassimo fuori una stella?



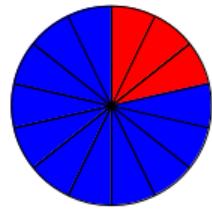
Naturale che sia venuta a 6 punte... ma se la volessimo a 7 punte?



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

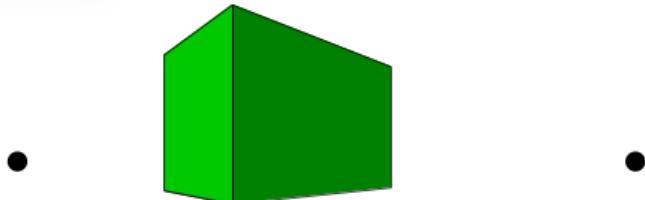


Oppure un esercizio di frazioni: in modo per esempio che, dati il numeratore N e il denominatore M , venga rappresentata la frazione N/M con una torta:



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).

lll



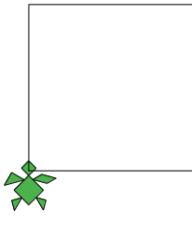
Capitolo 7

Incapsulare

7.1 Procedure

Per affrontare l'argomento delle procedure¹ riprendiamo il disegno del quadrato, così come lo avevamo fatto usando le variabili (sezione 5.1.4):

```
CLEARSCREEN  
HOME  
  
LATO = 100  
ANGOLO = 90  
  
FORWARD LATO  
LEFT ANGOLO  
FORWARD LATO  
LEFT ANGOLO  
FORWARD LATO  
LEFT ANGOLO  
FORWARD LATO  
HIDETURTLE
```



Abbiamo visto l'utilità delle 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

¹In altri contesti, e con modalità funzionali diverse, lo stesso concetto è realizzato attraverso funzioni, subroutine, metodi.

consente di semplificare grandemente la scrittura del codice, e di conseguenza, di ridurre la probabilità di commettere errori.

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

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

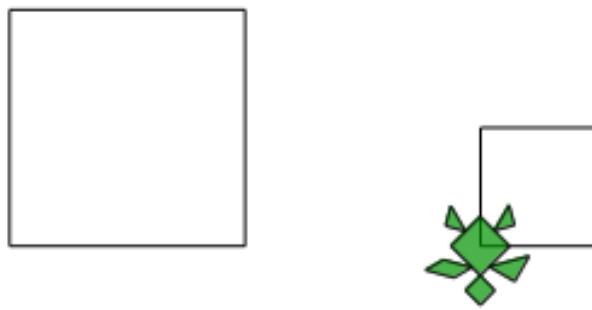
TO QUADRATO
LAT0 = 100 ; fisso il lato del quadrato
ANGOLO = 90 ; angolo di rotazione ai vertici
FORWARD LAT0 ; primo lato
RIGHT ANGOLO ; giro di 90° a destra, ecc.
FORWARD LAT0
RIGHT ANGOLO
FORWARD LAT0
RIGHT ANGOLO
FORWARD LAT0
END

; script che viene eseguito

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

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 piacemento 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:



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:

```

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

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

TO QUADRATO LATO
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

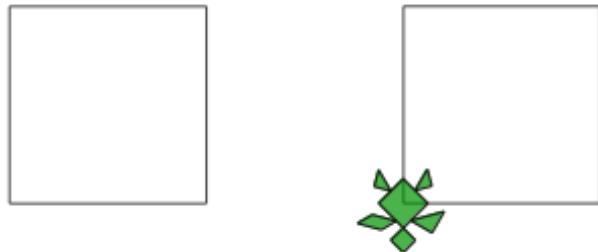
; script che viene eseguito

QUADRATO 100 ; qui si eseguono le istruzioni blu
PENUP
FORWARD 100
PENDOWN
QUADRATO 50 ; qui si eseguono le istruzioni blu

```

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**.

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 dire il 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
    FORWARD LATO/2
    ; alzo la penna
    ; mi dirigo su lato che ho di fronte
    ; così mi ritrovo a metà del lato
    ; di fronte

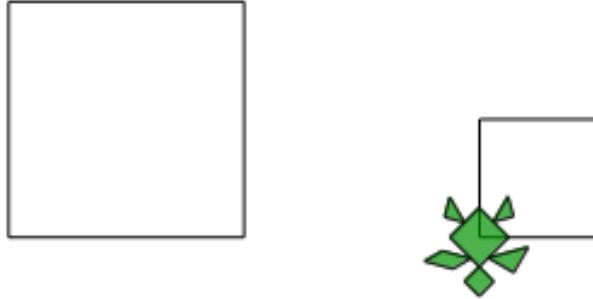
    RIGHT ANGOLO
    PENDOWN
    FORWARD LATO/2
    ; giro a destra
    ; abbasso la penna
    ; disegno mezzo del primo lato
    ; giro a destra
    ; disegno il secondo lato
    ; giro a destra
    ; disegno il terzo lato
    ; giro a destra
    ; disegno il quarto lato
    ; giro a destra
    ; disegno la metà rimanente del
    ; quarto lato
    ; giro a destra (per tornare nel
    ; centro del quadrato)
    ; alzo la penna
    ; torno nel centro del quadrato
    ; mi rigiro nella direzione in cui
    ; mi trovavo inizialmente

    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 encapsulare in una subroutine. Prendiamo per esempio il codice che avevamo scritto a pagina 15 proviamo a incapsularlo in una subroutine:

stessa cosa alla nostra istruzione QUADRATO, ecco come:

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

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

TO CASA
    textcolor{blue}{TO} FORWARD 50mm RIGHT 90\degree
    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:

```

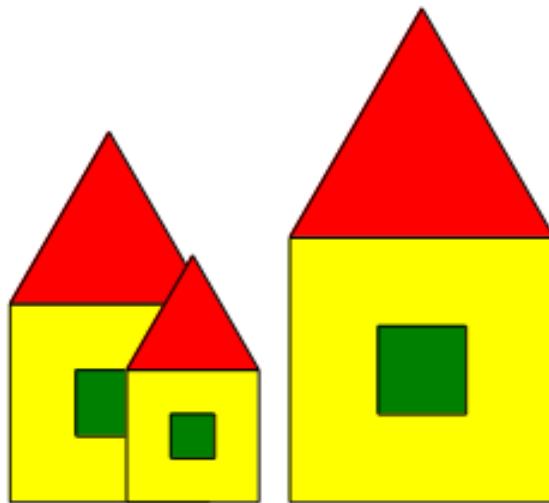
CLEARSCREEN
HOME

TO CASA LATO
    FORWARD LATO RIGHT 90\degree
    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
    HIDE TURTLE
END

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

```

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] (per esempio), mi 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 incolonarli, 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 `textbf{POLIGONO}`, con le istruzioni in blu, eccetto il nome della subroutine in rosso e

i suoi argomenti in viola. La subroutine `textbfPOLIGONO` 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 `textbfPOLIGONO` 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**)². 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 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 testo³. 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

²Ricordiamo 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.

³Questa 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 delle quali sono ASCII E UNICODE. Chi desidera chiarimenti si può rivolgere all'autore di questo testo.

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".

```

; 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 fisso 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
  AI = 180*(N-2)/N   ; la tartaruga ad ogni vertice
                      ; (N -> infinito => A -> 0)
  L = ABS(2*R*SIN(A/2*PI/180)) ; angoli interni
                                ; (N -> infinito => AI -> 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 gradi a destra (direzione
                           ; tangente al cerchio
                           ; circoscritto)
  RIGHT (180 - AI)/2       ; direzione primo lato da
                           ; disegnare
  PENDOWN                  ; giù la penna
  REPEAT N [
    FORWARD L RIGHT A     ; ciclo sui lati del poligono
  ]
  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
    HEADING 6h              ; corrente
    BACK 10                 ; mi rigiro in giù
                           ; torno al centro
  ]
END                   ; fine subroutine POLIGONO

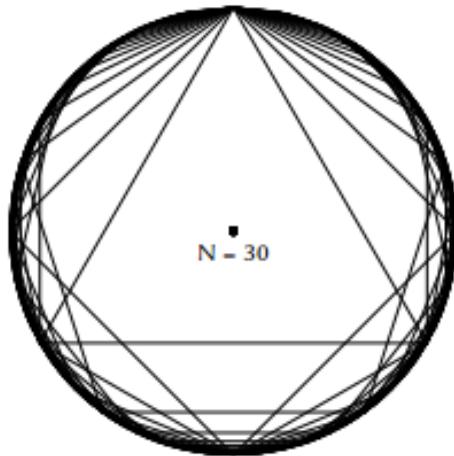
```

```

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

```

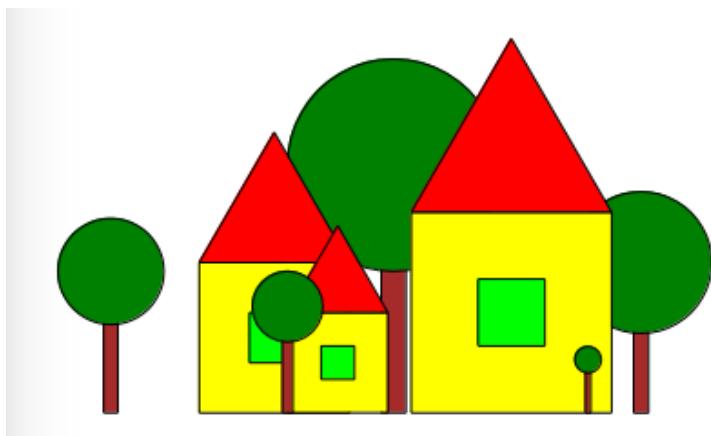
Ecco il risultato:



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 arricchire il paesaggio...



Capitolo 8

Decidere

8.1 IF - AND, OR, NOT

In questa versione proponiamo una descrizione 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!”.

Capitolo 9

La magia dei due specchi

9.1 La ricorsione

A molti sarà capitato di meravigliarsi osservando la fuga delle immagini generata da due specchi contrapposti. Lo specchio numero 1 sa fare una cosa sola: riprodurre la scena che ha di fronte. Anche lo specchio numero 2 sa fare solo la stessa cosa, ma così facendo riproduce anche lo specchio numero 1, compreso la scena in esso contenuta, la quale a sua volta riproduce la scena nello specchio numero 2 e così via, all'infinito. È un fenomeno che colpisce perché consente di sbirciare nell'infinito, normalmente inaccessibile all'esperienza umana. Questa è la ricorsione.

È molto facile creare uno schema ricorsivo attraverso il software:

```
1 TO RICORSIONE  
2   RICORSIONE  
3   END  
4  
5   RICORSIONE
```

In questo frammento di codice la Tartaruga esegue un solo comando: RICORSIONE. Non esistendo tale comando nel lessico di Logo è stato necessario definirlo con un procedura, di sapore altrettanto minimalista, perché tutto ciò che la Tartaruga deve impararare è il comando RICORSIONE stesso. Succede come negli specchi: la Tartaruga quando incontra l'istruzione TO RICORSIONE si predispone ad apprendere diligentemente il suo contenuto, che però consiste solo nel memorizzare l'istruzione RICORSIONE medesima. La Tartaruga è in grado di memorizzare sequenze di comandi che possono essere anche molto complesse, tuttavia non pensa. Una volta preso atto del contenuto della procedura, quando le viene chiesto di eseguire il comando RICORSIONE, si getta senz'altro nell'esecuzione dei comandi, inconsapevole di essere caduta in una trappola mortale! Infatti non avrà più alcun modo di uscire da questa perversa

magia. Caro lettore prova a seguire la Tartaruga per vedere cosa succede, ti lascio la scoperta e la conseguente riflessione...

Se tutto ciò potrà incantare i più immaginifici, susciterà invece perplessità nei più pragmatici: a cosa può servire una cosa simile? Perchè intrappolare inutilmente la povera Tartaruga?

Se avete fatto l'esperimento precedente, avrete visto apparire quasi subito questo messaggio: "Programma terminato: profondità ricorsiva massima (1000) superata". Evidentemente c'è chi veglia sulla sorte della Tartaruga, almeno entro certi limiti. Per esempio, l'inteprete di LibreLogo controlla quello che succede e se viene superato il limite di 1000 chiamate ricorsive ferma tutto, assumendo che ci sia qualcosa che non va. Può stupire che tutto avvenga in così breve tempo ma questo si spiega con il fatto che la Tartaruga non deve fare niente ad ogni chiamata della procedura RICORSIONE, se non richiamare un'altra volta la medesima. Poiché per fare questo tipo di operazioni il computer richiede tempi piccolissimi, frazioni di millisecondi, tutto il processo ci appare quasi instantaneo. Proviamo a rendere un po' più interessante il processo:

```

1 TO RICORSIONE R
2   CIRCLE R
3   RICORSIONE R+1
4 END
5
6 RICORSIONE 1

```

Qui abbiamo dotato la procedura RICORSIONE di un argomento, R, abbiamo aggiunto una sola istruzione, CIRCLE R, e abbiamo cambiato la chiamata ricorsiva così: RICORSIONE R+1. Quindi alla tartaruga viene chiesto di iniziare con RICORSIONE 1. A te lettore il piacere di scoprire cosa succede. Come ti lasciamo scoprire cosa succede con queste altre due varianti:

```

1 TO RICORSIONE R C
2   IF R + C > 70 [ C = -1 ]
3   IF R + C < 50 [ C = 1 ]
4   CIRCLE R + C
5   RICORSIONE R+C C
6 END
7
8 RICORSIONE 50 1

```

```

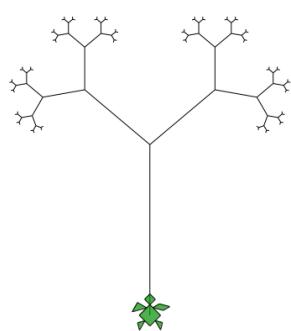
1 TO RICORSIONE N
2 LABEL N
3 PENUP FORWARD 10 PENDOWN
4 IF N > 20 [ STOP ]
5 RICORSIONE N+1
6 END
7
8 RICORSIONE 1

```

Non sarà sfuggita l'affinità fra il concetto di ripetizione che abbiamo già visto e quello di ricorsione. Effettivamente, gli esempi che abbiamo visto possono essere riprodotti anche con l'istruzione REPEAT - il lettore può farlo per esercizio. A sua volta, l'effetto di un ciclo di ripetizione può essere ottenuto anche con la ricorsione, ponendo le operazioni da eseguire in una procedura e inserendo la chiamata ricorsiva, ovvero alla procedura stessa, come ultima istruzione. Ma chi ci vieta di effettuare la chiamata ricorsiva in altre parti della procedura, o di farne addirittura più di una? È proprio qui che la faccenda si fa interessante.

9.2 Verso i frattali

Vediamo questo esempio:



```

1 TO ALBERO LL
2 IF LL < 2 [ STOP ]
3 FORWARD LL LEFT 50
4 ALBERO LL/2
5 RIGHT 100
6 ALBERO LL/2
7 LEFT 50 BACK LL
8 END
9
10 CLEARSCREEN
11 HOME
12 ALBERO 200

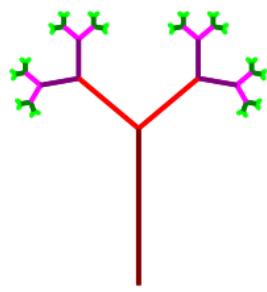
```

La procedura ALBERO è ricorsiva perché chiama se stessa, anzi, lo fa due volte, nelle istruzioni 4 e 6, ed esegue altre operazioni successivamente alle chiamate ricorsive.

In sostanza, la Tartaruga, dopo avere pulito lo schermo (CLEARSCREEN) ed essere andata a casa (HOME) esegue una sola istruzione: ALBERO 200. Andiamo a vedere cosa succede in ALBERO. Innanzitutto ci rendiamo conto che alla variabile LL viene attribuito il valore 200. Poi, come prima cosa, la Tartaruga controlla che LL non sia inferiore a 2 e, qualora si verifichi questa condizione si interrompe l'esecuzione del programma. Ma LL è maggiore di 2 quindi si va avanti tracciando un segmento lungo 200 e girando a sinistra di 50 gradi. A quel punto ecco la prima chiamata ricorsiva a ALBERO ma

con un valore dell'argomento pari a $LL/2$, quindi a 100. Non entriamo, per ora, "dentro" a questa chiamata e assumiamo che la Tartaruga abbia fatto quello che ci doveva fare. A questo punto la "vediamo" girare a destra di 100 gradi e poi richiamare un'altra volta ALBERO con lo stesso argomento $LL/2$, ossia 100. Anche qui, lasciamola "lavorare dentro" per poi vedere, che fatto questo, la Tartaruga gira nuovamente a sinistra di 50 gradi e torna indietro di LL , ovvero 200 punti. Questa descrizione è corretta ma non abbraccia tutto il processo, perché non si dice nulla su quello che succede nelle chiamate ricorsive a ALBERO. Nulla ci vieta di "entrare" anche a noi ma con un certo disagio perché non è difficile intuire che ci toccherà ripetere più volte questa operazione di "entrare" nelle chiamate ricorsive.

In effetti la strategia di seguire passo passo l'algoritmo non funziona tanto bene quando è in gioco la ricorsione, oppure diciamo che non basta. Occorre aggiungere un'altra prospettiva a quella sequenziale. Per chiarire questo passaggio può essere utile immaginare che quando è in atto una procedura ricorsiva non c'è una sola Tartaruga al lavoro ma un'intera squadra, organizzata con una precisa gerarchia.



In testa abbiamo la nostra solita Tartaruga, che però non disegna nulla, riducendosi all'esecuzione di ALBERO 200. Qui quello che succede è che la Tartaruga affida il lavoro alla Tartaruga Assistente Marrone, spiegandole che deve eseguire la procedura ALBERO a partire dal valore $LL=200$. Di altro la Tartaruga non vuole sapere e si mette pazientemente in attesa.

La Tartaruga Marrone invece si mette subito all'opera, eseguendo i passi richiesti dalla procedura ALBERO esattamente come li abbiamo visti prima. Quindi disegna diligentemente il tronco lungo 200 punti, si gira a sinistra di 50 gradi ma quando arriva all'istruzione 4, si comporta nella stessa maniera, affidando il lavoro alla Tartaruga Assistente Rossa (istruzione 4), dicendole di eseguire la procedura ALBERO, nella direzione che le indica, tuttavia partendo da un valore pari a $LL/2=100$; dato questo ordine la Tartaruga Marrone si mette a riposo. La Tartaruga Rossa ripete lo stesso comportamento e supponiamo che abbia fatto tutto quello che doveva fare. A questo punto il controllo viene ripreso dalla Tartaruga Marrone ma non per fare molto, perché una volta giratasi a destra di 100 gradi (istruzione 5) riaffida lo stesso compito di prima alla Tartaruga Rossa (istruzione 6). Quando questa ha terminato la Tartaruga Marrone riparte nuovamente per girarsi di 50 gradi a sinistra e tornare indietro di 200 punti (istruzione 7), ritrovandosi così alla base del tronco.

Emerge in questo modo il concetto di livelli di ricorsione, che noi possiamo visualizzare come i livelli delle tartarughe colorate: al primo livello opera la Tartaruga Marrone, al secondo la Tartaruga Rossa, al terzo la Tartaruga Viola e così via. Ognuna di queste non vuole sapere niente di ciò che ha fatto prima la tartaruga che le ha affidato il lavoro né di ciò che faranno quelle a cui affideranno i compiti a loro volta. Ognuna riceve delle istruzioni precise e le esegue, se deve

appaltare parti di lavoro ad altre tartarughe lo fa mettendosi in attesa.

La cosa importante è rendersi conto che con i livelli aumenta la complessità del lavoro, che si sminuzza in una successione di compiti eguali nella successione di comandi ma con parametri che vanno via via diversificandosi passando da un livello all'altro. In questo caso cambia il parametro LL che viene dimezzato ogni volta. Poiché il processo di moltiplicazione dei compiti cresce molto rapidamente con l'aumentare dei livelli non ha molto senso andare a ripercorrere pedissequamente la successione delle operazioni. Ci si "fida" che avvengano le stesse cose ad ogni livello se pur con mutate propozioni.

Siamo qui in prossimità di due idee potenti, per dirla con Papert. La prima evoca l'importante procedimento di dimostrazione matematica per induzione, che si usa per dimostrare un'affermazione per tutti gli elementi di un insieme ordinato. Questa consiste prima nel dimostrare la verità dell'affermazione per il primo elemento dell'insieme, poi nel dimostrare che data per vera l'affermazione per l'elemento generico n allora questa sia vera anche per l'elemento $n+1$. Anche qui, in qualche maniera ci si "fida" che il passaggio continui a valere per tutti gli n .

L'altra idea è quella di "autosimiglianza" che sta alla base del concetto di frattale. la natura frattale è posseduta da tutte quelle formazioni che non cambiano sostanzialmente aspetto osservandole a scale anche molto diverse: nuvole, coste, cavolfiori, vasi capillari ecc. L'albero che abbiamo visto è un frattale. È sorprendente la varietà e complessità dei frattali che si possono generare anche con la nostra semplice Tartaruga. In futuro dedicheremo un capitolo a questo argomento. Per ora vediamo una variazione del nostro albero, che era effettivamente un po' troppo geometrico:



```

1 TO ALBERO LL
2 IF LL < 5 [ FORWARD LL BACK LL STOP ]
3 FORWARD LL/3.0
4 LEFT 30 TREE LL*2.0/3.0 RIGHT 30
5 FORWARD LL/6.0
6 RIGHT 25 TREE LL/2.0 LEFT 25
7 FORWARD LL/3.0
8 RIGHT 25 TREE LL/2.0 LEFT 25
9 FORWARD LL/6.0
10 BACK LL
11 END
12
13 CLEARSCREEN
14 HOME
15 ALBERO 200

```


Parte II

Esperienze didattiche

Capitolo 10

L'esplorazione di Marta

10.1 Prologo

Marta Veloce è una studentessa che ha frequentato il Laboratorio di Tecnologie Didattiche nell'Anno Accademico 2016/17, giusto prima di laurearsi. L'11 novembre, dopo poco più di un mese dall'inizio del laboratorio, Marta mi invia la seguente email con un elaborato molto interessante, che ha dato adito a una serie di riflessioni e approfondimenti. Iniziamo la storia con l'email inviata da Marta.

Le scrivo perché ho provato a riflettere su alcune questioni, usando Logo, e ho sviluppato un breve percorso con alcuni "esercizi di creatività", come li ho chiamati io, eventualmente da poter svolgere con i bambini, basati sull'operazione di ripetizione di elementi per formare nuove immagini. Mi sono ispirata al libro di Munari [1], "Fantasia"; bellissimo! A partire da questo, ho poi sviluppato alcune riflessioni, anche di tipo geometrico, che, devo essere sincera, mi hanno fatto un po' "impazzire"; ho raccolto alcune ipotesi e dati e, anche se non sono giunta a nessuna conclusione definitiva, ho comunque trovato alcuni aspetti interessanti, su cui magari poter riflettere. Le invio l'elaborato per email, perché ricordo che ci aveva chiesto di fare così nel caso in cui producessimo documenti pesanti e troppo lunghi da caricare su Moodle.

Riporto qui di seguito l'elaborato, così come l'ho ricevuto da Marta.

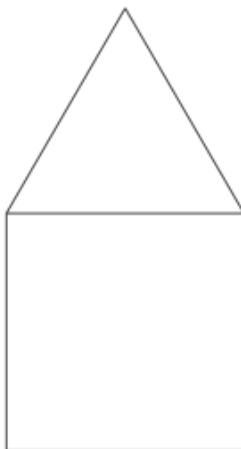
10.2 "Esercizi" di creatività – giocare con la ripetizione in Logo

Questo tipo di percorso prende avvio dalla lettura di un bellissimo testo di Bruno Munari, Fantasia. Il testo è proprio un elogio alla fantasia, rispetto alla quale la creatività si configura come suo uso finalizzato. Una persona colta senza fantasia, secondo Munari, è come un dizionario, pieno di parole ma senza alcuna poesia. Una delle operazioni mentali della fantasia, basate sul mettere in relazione i dati noti per creare qualcosa di nuovo, consiste nella ripetizione

di unità, senza alcuna variazione. Che cosa avviene se ripetiamo alcuni oggetti più e più volte? Si è provato a dare una risposta, utilizzando Logo e sfruttandone le grandi potenzialità grafiche: “A lui la precisione tecnica, a noi la sperimentazione, la riflessione... e il divertimento!” Il percorso non si limita a dare libero sfogo alla fantasia, ma si apre anche ad alcune riflessioni geometriche e aritmetiche, che procedono mediante metodo induttivo e secondo un approccio scientifico (osservazione-ipotesi-verifica). Si è ipotizzato di lavorare con una classe di bambini di scuola primaria.

10.2.1 Fase I

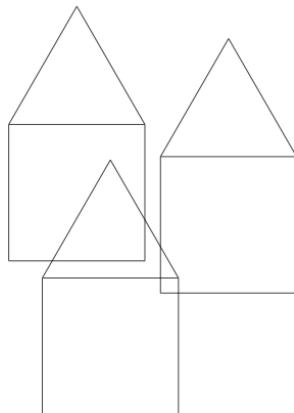
Chiediamo ai bambini di costruire una casetta. Il primo dei comandi deve essere “CLEARSCREEN”, così che si possa riavviare il programma tutte le volte che vogliamo, facendo compiere sempre lo stesso movimento alla tartaruga.



```
CLEARSCREEN
HOME
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
```

10.2.2 Fase II

Proviamo adesso ad eliminare il primo comando, “CLEARSCREEN”, e proviamo a riavviare il programma per due o più volte. Ci renderemo conto che, poiché adesso il disegno iniziale non viene cancellato, ogni volta che il programma si riavvia, la tartaruga traccia la stessa figura su quella precedente; si può osservare infatti, che, ad ogni riavvio, il tratto diventa sempre più spesso ed il colore nero si fa più intenso. Invitiamo i bambini a cliccare con il tasto sinistro del mouse sul disegno e a “spostarlo”; si renderanno subito conto che sul tracciato sono sovrapposte più casette che, se spostate a piacimento, possono costituire un bel quartiere!



```

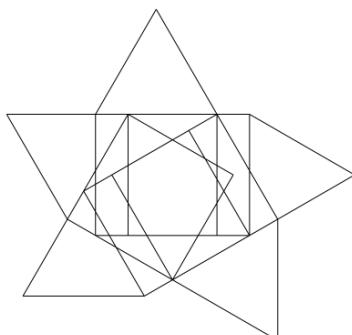
HOME
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100

```

10.2.3 Fase III

Proseguiamo in questo gioco creativo. Proviamo ad eliminare anche il comando “HOME” posto in cima alle istruzioni e volto a far tornare ogni volta la tartaruga in posizione iniziale. Che cosa accadrà? Stavolta, riavviando il programma, la tartaruga esegue il medesimo movimento, partendo però dalla posizione assunta in base all’ultimo comando. Essendo l’ultimo comando “FORWARD 100” e coincidendo questo con il movimento atto a descrivere l’ultimo lato del triangolotto, la tartaruga si trova disposta “a testa in giù”, orientata verso una direzione a 30 gradi a sinistra rispetto alla verticale. Per comprendere questo è necessario fare dei ragionamenti di tipo geometrico e aritmetico: è necessario sottrarre dall’angolo piatto (formato dall’ultimo lato del tetto e da un suo eventuale prolungamento) l’angolo di 150 gradi, formato dalla somma dell’angolo interno del triangolo (60 gradi) e quello del quadrato (90 gradi). Così facendo otteniamo il valore di 30 gradi.

Che cosa viene fuori se avviamo più volte il programma? Diamo il via all’immaginazione: ognuno può vederci ciò che vuole! Qualcuno potrebbe vederci una girandola, qualcun altro una stella, qualcuno un fiore!



```

FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100

```

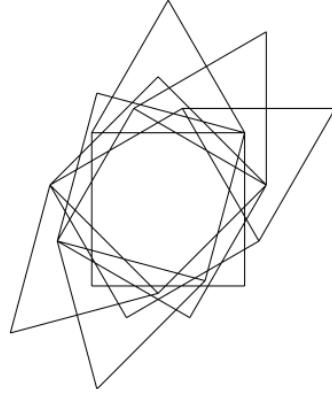
10.2.4 Fase IV

Proviamo ora a cambiare la posizione finale della tartaruga e ad avviare il programma più volte in modo tale che la casetta costruita possa non sovrapporsi

mai a quella precedente. Vediamo che cosa viene fuori! Proviamo ad esempio ad impostare, come ultimo comando, “RIGHT 45”. Per svolgere questa ultima attività, i bambini devono fare delle ipotesi e riflettere sugli angoli e sulle ampiezze.

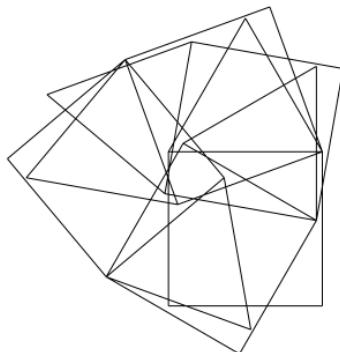
Possiamo a questo punto introdurre il comando “REPEAT”, scrivendo “REPEAT” seguito dal numero delle volte che si desidera riattivare il programma e dalle istruzioni inserite tra parentesi quadre. Questo semplifica e rende più rapido il procedimento, permettendo di non cliccare tutte le volte sulla voce “AVVIA IL PROGRAMMA LOGO” nell'apposita barra.

Che cosa sembra questa immagine? Potrebbe essere un’astronave oppure un missile spaziale!



```
REPEAT 5 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 45
]
```

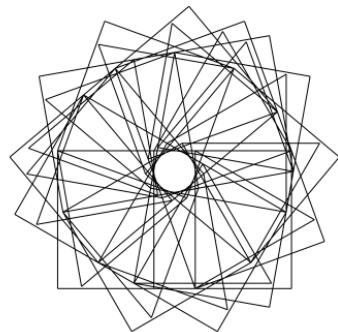
E se scriviamo “RIGHT 100” come ultimo comando? Che cosa viene fuori?



```
REPEAT 5 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 100
]
```

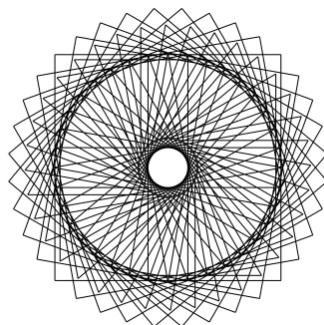
E se ripetiamo per più volte ancora la serie di comandi, ad esempio per 15 volte? Che bellissimo girasole!

10.2. “ESERCIZI” DI CREATIVITÀ – GIOCARE CON LA RIPETIZIONI IN LOGO107



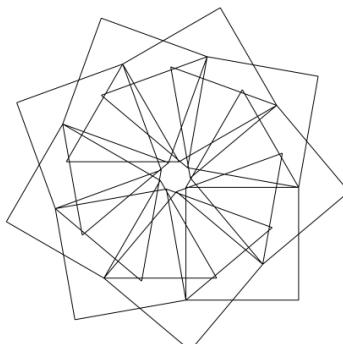
```
REPEAT 15 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 100
]
```

Proviamo a ripeterla invece per 100 volte!! Ai bambini “strafare”, esagerare, sperimentare “all’infinito” piacerà moltissimo! Ed ecco un sole splendente... oppure una margherita profumata!



```
REPEAT 100 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 100
]
```

Ecco cosa viene fuori invece se l’ultimo comando è “RIGHT 130” e se il programma viene eseguito 20 volte!



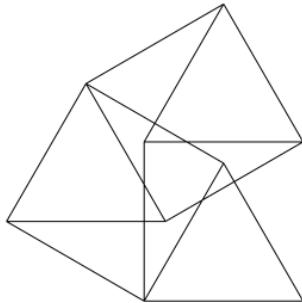
```
REPEAT 20 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 130
]
```

Svolgendo questo esercizio i bambini si accorgeranno che, a partire da un

certo punto in poi, la tartaruga ripete la sequenza di movimenti senza formare una nuova figura, ma ricominciando a creare quella precedente, sovrapposta alla prima. Verrà spontaneo chiedersi dopo quante volte la tartaruga “ricomincia il giro”, ossia quale è il numero di volte minimo che un determinato programma deve essere riavviato affinché si costituisca la figura più complessa che è possibile creare attraverso la ripetizione di quella determinata serie di comandi.

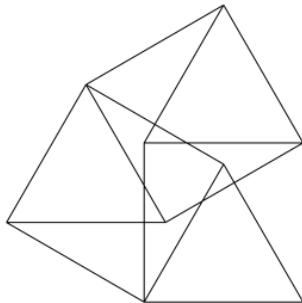
Prendiamo il caso di una rotazione di 90 gradi a destra rispetto alla direzione finale della tartaruga, posizionata al termine del terzo lato del triangolo. In questo caso la tartaruga è inclinata di 60 gradi a destra rispetto alla verticale. Avviando il programma più volte, i bambini si accorgeranno che, fino alla terza volta, la tartaruga costruirà una figura via via più complessa; a partire dal quarto riavvio del programma, invece, la tartaruga comincerà a ri-tracciare la stessa figura, sovrapponendola alla precedente.

Questo è il prodotto di tre ripetizioni di casetta, con rotazione finale di 90 gradi a destra:



```
REPEAT 3 [
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 30
  FORWARD 100
  RIGHT 120
  FORWARD 100
  RIGHT 90
]
```

Questo invece è il prodotto che risulta da quattro ripetizioni della stessa sequenza di comandi: è identico al precedente!



```
REPEAT 4 [
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 30
  FORWARD 100
  RIGHT 120
  FORWARD 100
  RIGHT 90
]
```

Tre dunque è il numero massimo di volte nelle quali si può ripetere la sequenza di comandi così da ottenere una figura diversa, ad ogni ripetizione e, in particolare, sempre più complessa. Proviamo a scoprire quante volte al massimo è possibile ripetere la sequenza di comandi, per ottenere una figura sempre

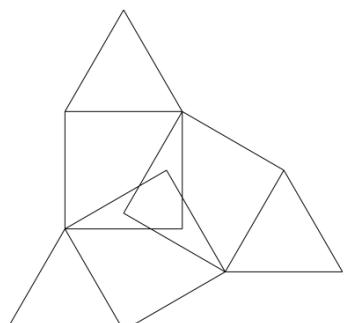
10.2. “ESERCIZI” DI CREATIVITÀ – GIOCARE CON LA RIPETIZIONI IN LOGO109

diversa, nel caso in cui la rotazione finale della tartaruga, rispetto alla sua posizione ultima, sia di 0 gradi. Questo è il caso della FASE III, nella quale la tartaruga, alla fine della costruzione della casetta, rimane posizionata nella direzione indicata dall’ultimo lato del triangolo-tetto, ossia inclinata di 30 gradi a sinistra rispetto alla direzione verticale.

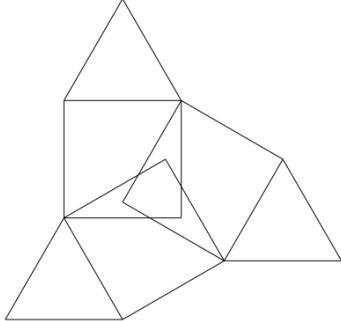
In questo caso notiamo che la tartaruga esegue il programma per 12 volte prima di ricominciare a tracciare lo stesso percorso. Anche nel caso di un’inclinazione iniziale di 60 gradi a destra rispetto alla posizione ultima della tartaruga, si deve avviare il programma 12 volte prima di ritrovarla posizionata come da comando “HOME”. Se invece la facciamo ruotare di 45 gradi a destra rispetto alla posizione ultima, si deve riavviare il programma 24 volte prima di ritrovarla posizionata come da comando “HOME”. Notiamo che in questo caso la tartaruga è spostata di 15 gradi a destra rispetto alla verticale. Anche se facciamo ruotare di 15 gradi a destra rispetto alla posizione ultima, accade la stessa cosa: si deve riavviare il programma 24 volte prima di ritrovarla posizionata come da comando “HOME”. Notiamo che anche in questo caso la tartaruga è spostata di 15 gradi, stavolta a sinistra, rispetto alla verticale.

Analizzando queste corrispondenze, sembra quindi che ruotando la tartaruga, rispetto alla posizione finale (nella direzione del terzo lato del triangolo-tetto), di un numero di gradi n , rispetto alla verticale, sia che si ruoti a destra sia che si ruoti a sinistra, essa dovrà ripetere il programma uno stesso numero p di volte, prima di tornare alla posizione come da comando “HOME”.

Per verificare se quanto ipotizzato è vero, proviamo ad estendere questo ragionamento anche al caso dell’inclinazione di 90 gradi a destra rispetto alla posizione finale (caso esposto per primo, sopra). In questo caso, la tartaruga, una volta ruotata, si ritrova inclinata di 60 gradi a destra rispetto alla verticale. Se proviamo ad inclinarla invece di 60 gradi a sinistra, rispetto alla stessa verticale, cosa avviene? Per farlo, dobbiamo sostituire l’ultimo comando della serie, “RIGHT 90”, con il comando “LEFT 30”. Se il ragionamento sopra esposto è corretto, devono risultare sufficienti 3 riavvii di programma per far tornare la tartaruga nella posizione come da comando “HOME”. In effetti, ciò accade: se il programma viene ripetuto quattro volte, la figura ottenuta è la stessa che se viene ripetuto 3.



```
REPEAT 3 [  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 30  
FORWARD 100  
RIGHT 120  
FORWARD 100  
LEFT 30  
]
```



```

REPEAT 4 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
LEFT 30
]

```

Abbiamo dunque osservato che:

- Ruotando la tartaruga di 90 gradi a destra o di 30 gradi a sinistra rispetto alla posizione finale, ossia posizionandola inclinata di 60 gradi rispetto alla verticale, rispettivamente a destra o a sinistra, il numero di volte che è necessario riavviare il programma affinché essa torni in posizione iniziale “HOME” è pari a 3.
- Ruotando la tartaruga di 60 gradi a destra rispetto alla posizione finale oppure lasciandola in tale posizione, ossia posizionandola inclinata di 30 gradi rispetto alla verticale, rispettivamente a destra o a sinistra, il numero di volte che è necessario riavviare il programma affinché essa torni in posizione iniziale “HOME” è pari a 12.
- Ruotando la tartaruga di 45 gradi a destra o di 15 gradi a destra rispetto alla posizione finale, ossia posizionandola inclinata di 15 gradi rispetto alla verticale, rispettivamente a destra o a sinistra, il numero di volte che è necessario riavviare il programma affinché essa torni in posizione iniziale “HOME” è pari a 24.

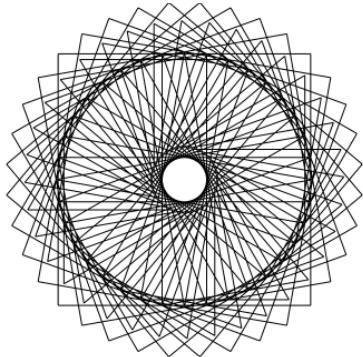
A quanto pare si potrebbe continuare....

- Ruotando la tartaruga di 150 gradi a destra o di 90 gradi a sinistra rispetto alla posizione finale, ossia posizionandola inclinata di 120 gradi rispetto alla verticale, rispettivamente a destra o a sinistra, il numero di volte che è necessario riavviare il programma affinché essa torni in posizione iniziale “HOME” è pari a 6.
- Ruotando la tartaruga di 270 gradi a destra o di 210 gradi a sinistra rispetto alla posizione finale, ossia posizionandola inclinata di 240 gradi rispetto alla verticale, rispettivamente a destra o a sinistra, il numero di volte che è necessario riavviare il programma affinché essa torni in posizione iniziale “HOME” è pari a 6.

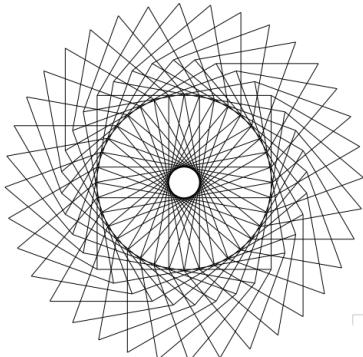
Per tornare al nostro caso precedente, il magnifico fiore “esagerato”, ottenuto ripetendo 100 volte la serie di comandi... Quante volte sarebbe bastato riavviare il programma per ottenere lo stesso prodotto?

10.2. “ESERCIZI” DI CREATIVITÀ – GIOCARE CON LA RIPETIZIONI IN LOGO111

Risulta che ruotando la tartaruga di 100 gradi a destra o di 40 gradi a sinistra rispetto alla posizione finale, ossia posizionandola inclinata di 70 gradi rispetto alla verticale, rispettivamente a destra o a sinistra, il numero di volte che è necessario riavviare il programma affinché essa torni in posizione iniziale “HOME” è pari a 36. Sarebbero bastate 36 volte per ottenere la stessa figura; infatti:

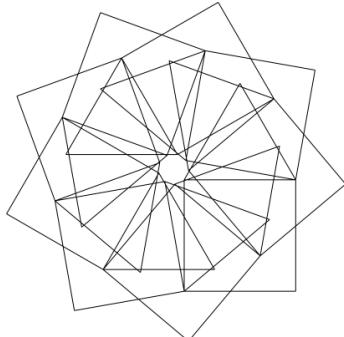


```
REPEAT 36 [  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 30  
FORWARD 100  
RIGHT 120  
FORWARD 100  
RIGHT 100  
]  
]
```

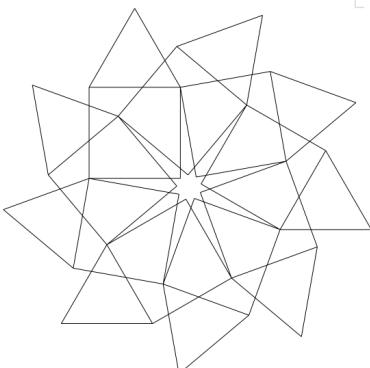


```
REPEAT 36 [  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 30  
FORWARD 100  
RIGHT 120  
FORWARD 100  
LEFT 40  
]  
]
```

E ancora, per tornare ai nostri “esercizi” di creatività, risulta che ruotando la tartaruga di 130 gradi a destra o di 70 gradi a sinistra rispetto alla posizione finale, ossia posizionandola inclinata di 100 gradi rispetto alla verticale, rispettivamente a destra o a sinistra, il numero di volte che è necessario riavviare il programma affinché essa torni in posizione iniziale “HOME” è pari a 9. Sarebbero bastate 9 volte per ottenere la stessa figura; infatti:



```
REPEAT 9 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 130
]
```



```
REPEAT 9 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 70
]
```

Insomma, sembra che la direzione di rotazione della tartaruga rispetto alla posizione finale, incida fortemente sull'immagine che si potrà ottenere ad un'eventuale ripetizione del programma, ma che, a patto che essa sia di un numero sempre uguale di gradi, sia che venga effettuata a destra sia che venga effettuata a sinistra rispetto alla verticale, il numero di volte necessario affinché la tartaruga torni alla posizione "HOME" rimane invariato.

Viene ora da domandarsi se sia possibile procedere verso un'ulteriore generalizzazione, a partire dai dati raccolti e dalle osservazioni fatte. In particolare, ci si chiede se esista una qualche relazione matematica tra il numero di gradi di inclinazione della tartaruga rispetto alla verticale in posizione finale (ossia dopo aver costruito la casetta) e il numero di volte che essa deve rieseguire il programma per poter tornare alla posizione "HOME".

Raccogliamo i dati in una tabella:

Gradi rotazione rispetto alla verticale	Numero di volte che il programma deve riavviarsi per tornare a posizione ”home”
15	24
30	12
60	3
70	36
100	9
120	6
240	6

Purtroppo non sembra emergere alcuna relazione.... E qui il pensiero si sofferma e prende respiro... Forse potremmo fare altre ipotesi?

Marta

10.2.5 Una prima soluzione

Probabilmente si potrebbe procedere a individuare una risposta matematica generale al quesito di Marta, ovvero formulare una descrizione teorica la quale, partendo da precisi e esaustivi presupposti, consenta di individuare una risposta corretta per tutti casi. Non disponendo del tempo per affrontare la questione in tali termini, ci venne a suo tempo naturale procedere per via euristica, esplorando una quantità di casi particolari. È emerso subito un primo fatto: in molti casi, il numero di ripetizioni minimo N viene dato correttamente da questa relazione:

$$N = \frac{360}{\theta \bmod 60} \quad (10.1)$$

dove θ è l'angolo di inclinazione della tartaruga rispetto alla verticale in posizione finale e mod è l'operazione ”modulo”, che dà il resto della divisione fra i due operandi: se, per esempio, $\theta = 365$ allora $\theta \bmod 60 = 5$. Tuttavia, insistendo si scopre altrettanto presto che vi sono valori particolari di θ , che costituiscono casi che potremmo definire degenerati, nei quali il numero di ripetizioni viene sorprendentemente piccolo. Facendo prove di questo genere, giungemmo alla seguente espressione:

$$N = \begin{cases} \begin{cases} 3, & \text{per } \theta/60 = 1 \\ 6, & \text{per } \theta/60 \neq 1 \end{cases}, & \text{per } \theta \bmod 60 = 0 \\ \frac{360}{\theta \bmod 60}, & \text{per } \theta \bmod 60 \neq 0 \end{cases} \quad (10.2)$$

Poiché questa relazione consentiva di rispondere a tutti i casi elencati nella tabella proposta da Marta ci ritenemmo soddisfatti, ben consapevoli tuttavia, che non era affatto detto che la soluzione trovata fosse esaustiva per tutti i casi possibili. Ci accontentammo quindi dei risultati forniti dal codice Logo che esprime la suddetta formula, dove la variabile T rappresenta θ e NVOLTE rappresenta N :

```

1 TO NVOLTE T
2 T = ABS(T)
3 R = T % 60
4 IF R = 0 [ IF T /60 = 1 [ N = 3 ] [ N = 6 ] ] [ N = 360/R ]
5 OUTPUT N
7 END
8
9 TO CASA
10 FORWARD 100
11 RIGHT 90
12 FORWARD 100
13 RIGHT 90
14 FORWARD 100
15 RIGHT 90
16 FORWARD 100
17 RIGHT 90
18 FORWARD 100
19 RIGHT 30
20 FORWARD 100
21 RIGHT 120
22 FORWARD 100
23 END
24
25 TO FIGURA NV TETA
26 REPEAT NV [
27 CASA
28 RIGHT TETA+30
29 ]
30 END
31
32 TO FIG TETA
33 CLEARSCREEN
34 NV = NVOLTE TETA
35 PRINT NV
36 FIGURA NV TETA
37 END
38
39 CLEARSCREEN
40 HOME
41 HIDETURTLE
42
43 T = -5
44 FIG T

```

10.3 L'alternativa di Alberto

Successivamente è capitato di riproporre la riflessione di Marta nel corso di perfezionamento "Le competenze digitali nella scuola". Uno dei partecipanti, Alberto Averono, insegnante di informatica presso un istituto tecnico, ha rilanciato la questione, proponendo un diverso modo di codificare il problema:

Buongiorno, siccome il racconto del viaggio di Marta ha stimolato la mia curiosità, durante la narrazione di martedì sera ho avuto due

idee:

- se la Tartaruga ci dicesse quante case disegna prima di ritrovarsi a casa senza dover fare calcoli?
- se la Tartaruga fosse ricorsiva, consentendoci così di semplificare il codice?

E questo è il codice alternativo proposto da Alberto:

```

1 TO CASA T NV PORIG
2 PCORR = POSITION
3 IF PCORR = PORIG AND NOT NV = 0 [ PRINT NV STOP ]
4 FORWARD 100
5 RIGHT 90
7 FORWARD 100
8 RIGHT 90
9 FORWARD 100
10 RIGHT 90
11 FORWARD 100
12 RIGHT 90
13 FORWARD 100
14 RIGHT 30
15 FORWARD 100
16 RIGHT 120
17 FORWARD 100
18 RIGHT T + 30
19 NV = NV + 1
20 CASA T NV P1
21 END
22
23 CLEARSCREEN
24 HOME
25 P = POSITION
26 HIDEURTLE
27 CASA 30 0 PORIG

```

In questo codice la casa viene disegnata con una procedura che necessita di tre argomenti: T che rappresenta l'angolo θ di inclinazione della tartaruga rispetto alla verticale in posizione finale, NV è il numero di volte che si deve rieseguire il codice per poter tornare alla posizione "HOME" e PORIG contiene la posizione iniziale della Tartaruga.

Conclude Alberto:

Supposto che il codice sia corretto, direi che

1. la Tartaruga termina di costruire una casa nel punto HOME solo per angoli multipli di 30,
2. se l'angolo è 180 non torna mai,
3. per gli angoli 270 e 300 disegna 4 case e 3 case contro le 12 e 6 previste rispettivamente dalla formula 10.2,

4. per tutti gli altri angoli multipli di 30 i risultati coincidono,
5. con angoli non multipli di 30 ripete il disegno ma non termina una casa esattamente nel punto HOME per cui non si può utilizzare in quei casi (a meno di modifiche).

È interessante entrare nel dettaglio delle proposte di Alberto perché questo ci consente di fare alcune importanti considerazioni, non solo relativamente al coding ma anche a fatti prettamente matematici.

La prima proposta è quella di lasciare che la Tartaruga "riconosca" da sola la posizione da cui era partita. Messa in questi termini, viene in effetti da domandarsi perché confondersi con formule arzigogolate quando è possibile affidare tutto al calcolo numerico? Benissimo: accettiamo per il momento questa proposta, che nel codice di Alberto si realizza nell'istruzione 3, dove si controlla se la posizione corrente, PCORR, è eguale alla posizione originale, PORIG. Il controllo viene fatto a meno che non si tratti della prima iterazione ($NV=0$), per la quale è ovvio che le due condizioni coincidono. Se così non è ($NV \neq 0$) e se si verifica che la Tartaruga sia tornata alla posizione originale ($PCORR=PORIG$) allora il programma stampa il valore NV dell'iterazione corrente e si ferma (STOP).

La seconda proposta consiste nel sostituire il ciclo di ripetizioni usato da Marta con una procedura ricorsiva. Alberto realizza questo mediante l'istruzione 20, dove la procedura CASA chiama se stessa ripassando gli stessi parametri che aveva ricevuto a sua volta. È un'ottima applicazione del costrutto ricorsivo che avevamo descritto nel capitolo 9.

Passiamo ora a commentare i risultati, così come sono stati riassunti da Alberto.

1. *La Tartaruga termina di costruire una casa nel punto HOME solo per angoli multipli di 30.* Come dire che la Tartaruga non riconosce sempre bene la propria posizione di origine. Questo è effettivamente un problema molto interessante che discutiamo in dettaglio successivamente.
2. *Se l'angolo è 180 non torna mai.* Vero. È un caso particolare che ci era sfuggito. Possiamo dire che per $\theta = 180$ il processo diverge. Lo aggiungiamo alla formula 10.2.
3. *Per gli angoli 270 e 300 disegna 4 case e 3 case contro le 12 e 6 previste rispettivamente dalla formula 10.2.* Vero anche questo. Si tratta di altri casi "degeneri" che andiamo ad aggiungere agli altri.
4. *Per tutti gli altri angoli multipli di 30 i risultati coincidono.* Sì, ma non sempre, come vediamo fra poco.
5. *Con angoli non multipli di 30 ripete il disegno ma non termina una casa esattamente nel punto HOME per cui non si può utilizzare in quei casi (a meno di modifiche).* E questo è il problema menzionato al punto 1.

Per iniziare facciamo la cernita dei casi particolari, cercando le regolarità. Vediamo i primi 31, partendo da $\theta = 0$:

Gradi di rotazione rispetto alla verticale	Numero di volte che il programma deve riavviarsi per tornare a posizione "home"
0	2
30	12
60	3
90	4
120	6
150	12
180	∞
210	12
240	6
270	4
300	3
330	12
360	2
390	12
420	3
450	4
480	6
510	12
540	∞
570	12
600	6
630	4
660	3
690	12
720	2
750	12
780	3
810	4
840	6
870	12
900	∞

Se proviamo a descrivere in linguaggio matematico le regolarità che vediamo in questa tabella otteniamo la seguente espressione, abbastanza più complicata della precedente (10.2):

$$N = \begin{cases} 2, & \text{per } \theta/30 \in \{(n-1) \times 360\} \\ 3, & \begin{cases} \text{per } \theta/30 \in \{60 + (n-1) \times 360\} \\ \text{per } \theta/30 \in \{300 + (n-1) \times 360\} \end{cases} \\ 4, & \begin{cases} \text{per } \theta/30 \in \{90 + (n-1) \times 360\} \\ \text{per } \theta/30 \in \{270 + (n-1) \times 360\} \end{cases} \\ 6, & \begin{cases} \text{per } \theta/30 \in \{120 + (n-1) \times 360\} \\ \text{per } \theta/30 \in \{240 + (n-1) \times 360\} \end{cases} \\ 12, & \begin{cases} \text{per } \theta/30 \in \{150 + (n-1) \times 360\} \\ \text{per } \theta/30 \in \{210 + (n-1) \times 360\} \end{cases} \\ \infty, & \text{per } \theta/30 \in \{180 + n \times 360\} \\ 360/(\theta \bmod 30), & \text{per } \theta \bmod 30 \neq 0 \end{cases}, \quad \text{per } \theta \bmod 30 = 0 \quad (10.3)$$

dove $n \in \{1, 2, \dots\}$.

Non è difficile tradurre in codice LibreLogo questo schema, analogamente a quanto avevamo fatto nella versione più semplice, e questa sarebbe la via che avevamo proposto in origine. Alberto proponeva invece altro, ovvero di far "ri-conoscere" alla tartaruga medesima la posizione originale, qualora ci si ritrovasse dopo un certo numero di ripetizioni, e quello sarebbe il numero cercato.

La proposta è interessante, diciamo che è più di sapore numerico che matematico. La soluzione più prettamente matematica è la precedente, dove con il ragionamento cerchiamo una regola generale che solo alla fine applichiamo numericamente. La proposta di Alberto è invece numerica perché si affida al confronto fra numeri calcolati. Non esiste un criterio assoluto per stabilire quale sia il metodo migliore, dipende dal contesto.

Qui tuttavia è Alberto stesso che segnala un problema, ovvero che la Tartaruga termina di costruire una casa nel punto HOME solo per angoli $\theta = 30$. Perché succede questo? Prima di rispondere dobbiamo aggiungere un'altra complicazione. Cercando di riprodurre i risultati ottenuti da Alberto, mi sono accorto che in realtà la Tartaruga può perdere anche in altri casi, anche quando θ è un multiplo di 30 gradi, e questo comportamento dipende addirittura dal computer che ospita la Tartaruga! Ma come è possibile una cosa del genere? Come può la Tartaruga, che è pur sempre una creatura determinata da un software che viene eseguito in una macchina apparentemente perfettamente determinata, stupidamente meccanica nei comportamenti, come il computer, comportarsi invece come una creatura bizzosa, che decide come contenersi a seconda delle circostanze? E di quali circostanze? Era già strano che le cose funzionassero solo per valori di θ multipli di 30 gradi, ma mettersi a fare le bizze anche per questi e su un computer sì e su un altro no è troppo! Eppure è quello che succede ed è una fortuna per noi perché ci consente di sbirciare meglio in un aspetto illuminante riguardo alla relazione fra matematica e *computer science*. In buona parte dell'opinione comune, tutte le scienze classiche e tecnologiche ricadono in un ambito dominato dall'esattezza e da una sorta di meccanicità. Questa visione genera in molti il concetto che il coding sia un qualcosa di automatico, fatto di pratiche rigidamente predeterminate che si riducono ad una sorta di "simulazione" di robot. È un'idea profondamente sbagliata, generata dall'ignoranza delle questioni pertinenti.

Vediamo come stanno le cose nel nostro problema. Nel momento in cui scrivo dispongo di tre computer diversi e scopro che... la Tartaruga si comporta diversamente su ciascuno di essi. Prendiamo il caso $\theta = 30$ gradi:

Computer	Versione di Ubuntu	Numero di volte che il programma deve riavviarsi per tornare a posizione "home"
Laptop Lenovo Thinkpad X220 Intel Core i5-2520M 2.50GHz Ubuntu 16.04 32 bit	5.2.7.2	3
Laptop Lenovo Thinkpad X220 Intel Core i7-2640M 2.80GHz Windows 7 64 bit	5.1.2.2	∞
Minitower Acer Aspire XC100 AMD E1-1500 1.50GHz Ubuntu 16.04 64 bit	5.1.6.2	6

Eppure il codice eseguito è identico. Come può accadere una cosa simile? Una delle prime cose che un collaboratore più grande mi insegnò, quando venni in contatto con il primo computer (nel 1977), fu: quando non capisci cosa succede fai stampare al computer tutti i dati intermedi¹. Vediamo quindi il caso del laptop Ubuntu, considerando la sequenza delle posizioni PCORR raggiunte dalla Tartaruga alla fine del disegno di ogni casa, confrontate con la posizione originale, PORIG. Occorre tenere presente che dire posizione significa dire due numeri, ovvero le coordinate X e Y nella pagina, per cui ad esempio PORIG = [297.41102362204725, 420.71811023622047], secondo le convenzioni descritte nella sezione 5.1.5.

¹Qualcuno si domanderà come si possono produrre le tabelle sottostanti con LibreLogo, che sembra tutto orientato alla grafica. Il fatto è che LibreLogo è scritto nel linguaggio Python, derivandone tutta una serie di caratteristiche, non esplicitate nel manuale di LibreLogo. Fra queste capacità "clandestine" c'è anche quella di poter leggere il contenuto di file, crearli e scriverci dentro. Dedicheremo un'appendice a queste possibilità.

NV	PCORR[0] (la X)	PORIG[0] (la X)
0	297.41102362204725	297.41102362204725
1	347.41417322834644	297.41102362204725
2	279.0992125984252	297.41102362204725
3	297.41102362204725	297.41102362204725

Tabella 10.1: Coordinate X delle posizioni correnti PCORR e della posizione originale PORIG con il laptop Ubuntu. La posizione viene ritrovata correttamente dopo la terza ripetizione. NV è il numero di volte che il programma deve riavviarsi per tornare a posizione "home", PCORR è la posizione corrente della Tartaruga e PORIG la posizione iniziale. Le variabili PCORR e PORIG sono costituite da coppie di numeri che sono le coordinate della Tartaruga nello spazio della pagina.

NV	PCORR[1] (la Y)	PORIG[1] (la Y)
0	420.71811023622047	420.71811023622047
1	370.7149606299212	420.71811023622047
2	352.4031496062992	420.71811023622047
3	420.71811023622047	420.71811023622047

Tabella 10.2: Coordinate Y delle posizioni coorrenti PCORR e della posizione originale PORIG con il laptop Ubuntu.

Qui vediamo che la Tartaruga conferma la congettura di Alberto: il disegno consiste in tre casette incastrate e la Tartaruga si ferma diligentemente alla fine della terza ripetizione. In giallo è evidenziata la perfetta corrispondenza fra la posizione corrente PCORR alla terza ripetizione e la posizione originale PORIG (le X e le Y sono identiche).

Vediamo ora cosa succede girando lo stesso codice nel portatile con Windows.

NV	PCORR[0] (la X)	PORIG[0] (la X)
0	297.46771653543306	297.4393700787401
1	347.47086614173224	297.4393700787401
2	279.18425196850393	297.4393700787401
3	297.4 96062992126	297.4 393700787401
4	347.4992125984252	297.4393700787401
5	279.21259842519686	297.4393700787401
6	297.4 96062992126	297.4 393700787401
7	347.4992125984252	297.4393700787401
∞

Tabella 10.3: Coordinate X delle posizioni correnti PCORR e della posizione originale PORIG con il laptop Windows. La posizione non viene mai ritrovata correttamente - perlomeno per le prime cento ripetizioni che sono state lasciate fare alla macchina. Sono evidenziate in magenta le ripetizioni dove la Tartaruga ci va vicino.

NV	PCORR[1] (la Y)	PORIG[1] (la Y)
0	420.71811023622047	420.6897637795276
1	370.68661417322835	420.6897637795276
2	352.4031496062992	420.6897637795276
3	420.71811023622047	420.6897637795276
4	370.7149606299212	420.6897637795276
5	352.4031496062992	420.6897637795276
6	420.71811023622047	420.6897637795276
7	370.7149606299212	420.6897637795276
∞

Tabella 10.4: Coordinate Y delle posizioni correnti PCORR e della posizione originale PORIG con il laptop Windows.

Cambiando computer sia l'hardware che il sistema operativo - la Tartaruga non ritrova mai la posizione, perlomeno intendendo con "mai" un numero di ripetizioni molto maggiore delle tre necessarie per $\theta = 30$. Qui diventa interessante andare a vedere i numeri da vicino. Prendiamo per esempio la X di PCORR. Ci accorgiamo così che in realtà la Tartaruga va molto vicino alla X iniziale di PORIG, quando arriva alla ripetizione 3 o un suo multiplo: 297.496062992126 contro 297.4393700787401, una differenza dello 0.019%. Per la Y abbiamo 420.71811023622047 contro 420.6897637795276 pari allo 0.0067%.

Ancora più bizzarro è il comportamento nel computer fisso con Ubuntu:

NV	PCORR[0] (la X)	PORIG[0] (la Y)
0	297.4393700787401	297.4393700787401
1	347.47086614173224	297.4393700787401
2	279.15590551181106	297.4393700787401
3	297.46771653543306	297.4393700787401
4	347.47086614173224	297.4393700787401
5	279.15590551181106	297.4393700787401
6	297.4393700787401	297.4393700787401

Tabella 10.5: Coordinate X delle posizioni correnti PCORR e della posizione originale PORIG con computer fisso tipo minitower con Ubuntu. La posizione viene ritrovata correttamente dopo la sesta ripetizione, evidenziata in giallo; in magenta è evidenziata la terza, dove vediamo che la Tartaruga ci è arrivata vicino.

NV	PCORR[1] (la Y)	PORIG[1] (la Y)
0	420.71811023622047	420.71811023622047
1	370.7149606299212	420.71811023622047
2	352.4031496062992	420.71811023622047
3	420.71811023622047	420.71811023622047
4	370.7149606299212	420.71811023622047
5	352.4031496062992	420.71811023622047
6	420.71811023622047	420.71811023622047

Tabella 10.6: Coordinate Y delle posizioni correnti PCORR e della posizione originale PORIG con il computer fisso tipo minitower con Ubuntu.

Qui accade che la Tartaruga ce la fa a riconoscere la posizione originale al secondo tentativo, per la coordinata X, mentre per la Y l'azzecca al primo!

L'esame di questi numeri fa emergere il concetto fondamentale: cosa vuol dire dunque che due numeri sono "eguali"? Le differenze in percentuale delle coordinate che dovrebbero essere eguali sono minime rispetto al contesto: spostare dello 0.01% la coordinata del centro di un foglio di carta significa preoccuparsi di qualcosa dell'ordine di un centesimo di mm, un errore del tutto irrilevante ai fini della produzione grafica nei contesti che ci possono interessare. Quindi, da questo punto di vista, i due numeri "sono" eguali ma la Tartaruga non conosce il nostro buon senso, a meno che noi non la informiamo di come debba adattarsi al contesto che ci interessa. Insomma, se vogliamo che le cose funzionino occorre spiegarle in qualche modo cosa intendiamo per egualianza fra due coordinate. Per fare questo dobbiamo alterare il concetto di egualianza nel codice di Alberto (inizio sezione 10.3), sostituendo all'istruzione 3

```
3 IF PCORR = PORIG AND NOT NV = 0 [ PRINT NV STOP ]
```

le seguenti:

```
3 DX = ABS(PCORR[0] - PORIG[0]) /PCORR[0]*100
4 DY = ABS(PCORR[1] - PORIG[1]) /PCORR[1]*100
5 IF DX < 0.1 AND DY < 0.1 AND NOT NV = 0 [
6   PRINT 'T = ' + STR T
7   PRINT 'NV = ' + STR NV
8   STOP
9 ]
```

La cosa si è fatta un po' più complicata ma l'idea che c'è sotto è semplice. Con le istruzioni 3 e 4 si calcolano gli scostamenti della posizione originale espressi come percentuali di quest'ultima. Poi con le istruzioni 5-9 si controlla se tali scostamenti sono inferiori o meno allo 0.1%. Nel caso che lo siano li

consideriamo nulli, ovvero se due coordinate differiscono per meno dello 0.1% allora le consideriamo eguali.

In questo modo il sistema di Alberto, basato, oltre che sull'applicazione dello schema ricorsivo, sul riconoscimento della posizione originale da parte della Tartaruga stessa, funziona correttamente.

Cosa abbiamo imparato con tutto questo? Un fatto fondamentale: che i numeri digitali sono diversi dai numeri "veri", così come li adopriamo in matematica. Il computer lavora su numeri espressi in bit, che possono valere zero o uno. Per fare le operazioni si avvale del sistema binario, che obbedisce alle stesse identiche regole del sistema decimale, ottale ecc. Può anche lavorare utilizzando la virgola, e anche in notazione scientifica, per esempio 1.0×10^{-2} , anziché 0.01. Ma non può lavorare con numeri che hanno un numero infinito di cifre, che invece sono i più importanti in matematica. Ad esempio, la diagonale del quadrato di lato 1 $\sqrt{2}$, il rapporto fra circonferenza e diametro π , la base dei logaritmi naturali e , la costante aurea γ sono tutti numeri irrazionali, che non si possono esprimere con un numero finito di cifre. Diciamo che π è 3.14 ma questa è solo un'approssimazione. Può probabilmente andare bene per progettare la ruota di un carro, ma non l'ingranaggio di un orologio. Ne segue che i numeri più importanti di tutti non possono avere cittadinanza nel computer! Allora come si fa? Semplicemente gestendo le approssimazioni. quando ci si rende conto che c'è un problema di calcolo numerico, che impedisce di riprodurre i risultati che ci saremmo aspettati per via matematica, allora si utilizzano dei rimedi. Esattamente come abbiamo fatto ora ridefinendo, di fatto, il concetto di egualianza.

Ci rimarrebbe da capire il motivo per cui la Tartaruga si ritrova con dei numeri non perfettamente esatti. Questo è più difficile da capire. Si potrebbe certo andare in fondo al problema, facendo una serie di prove mirate ma sono lavori che possono richiedere un tempo considerevole e qui andremmo decisamente fuori strada. Ci basti acquisire la consapevolezza che la faccenda è complessa. È proprio per questo motivo che ho riportato diversi dettagli nella tabella 10.3 dei computer: il tipo di CPU, il sistema operativo, perfino la versione del medesimo, la versione del software che esegue i calcoli - qui ho messo la versione di LibreOffice che ospita LibreLogo ma forse avrei dovuto mettere anche la versione di Python, che è il linguaggio in cui di fatto LibreLogo lavora - ebbene, tutti questi elementi possono in qualche maniera influenzare il modo in cui vengono eseguiti i calcoli. E tutte queste problematiche sono amplificate quando i calcoli sono lunghi e magari utilizzano processi iterativi estesi, perché anche errori che sono molto piccoli su una singola operazione, si possono amplificare a dismisura quando per ottenere un certo risultato occorrono lunghissime sequenze di operazioni, dove si può assistere a una propagazione degli errori crescente. Ci imbatteremo nuovamente in questo problema, quando proveremo a calcolare le orbite dei corpi celesti con LibreLogo. Queste sono tematiche fondamentali nella maggior parte della matematica applicata. Riuscire a condurre i giovani studenti in territori del genere può essere molto importante.

10.4 La soluzione matematica

In realtà il quesito di Marta ha una risposta matematica. All'epoca in cui Marta pose la domanda dissi subito che la mia risposta sarebbe stata euristica e non

generale in senso matematico. Vale a dire che risolveva i casi studiati da Marta (10.2.4) ma non era affatto detto che fosse in grado di tenere conto di tutti i casi immaginabili: mancava una teoria, in altre parole. Alberto ci ha poi spinti verso una soluzione informatica: lasciamo che la Tartaruga si accorga da sola di essere tornata nello stato iniziale. Questa poteva essere ritenuta una soluzione generale ma abbiamo imparato che non è proprio immediato dare alla Tartaruga il potere di riconoscere il proprio stato di partenza: c'è di mezzo la natura imperfetta dei numeri digitali. In realtà, appena ho avuto tempo sono andato a cercare la soluzione e l'ho trovata. Non ho tuttavia voluto sostituirla al percorso precedente che invece ho lasciato sì dipanasse liberamente, e che mantengo, perché ha il valore dell'esplorazione. Ma veniamo alla soluzione completa del problema.

La teoria che ci serve la troviamo nella Geometria della Tartaruga [7] e si basa su tre teoremi e un lemma. Il lettore non matematico, la norma per questo testo, non si deve spaventare. Noi qui non ripercorriamo le dimostrazioni di questi teoremi, dovremmo divagare troppo. Per capire il significato di un teorema non occorre seguirne la dimostrazione, questa serve "solo" a garantire che il teorema sia vero: un lavoro fondamentale, ed è il matematico che si prende la briga di farlo. Talvolta le dimostrazioni di teoremi apparentemente semplici sono incredibilmente complesse, e poco hanno a che vedere con la comprensione del senso del teorema. Piuttosto, è necessario che vengano chiariti bene i termini presenti nell'enunciato. È quello che faremo. Il lettore curioso² potrà ritrovare le dimostrazioni nei riferimenti al testo originale [7] che daremo per ogni teorema. Inoltre cercheremo di non perdere mai di vista il quesito di Marta, che possiamo così sintetizzare: esiste un criterio per prevedere la chiusura della figura generata da una successione di casette ruotate di un angolo α ?

10.4.1 Teorema del percorso chiuso

Questo primo teorema fornisce il quadro generale all'interno del quale possiamo avere speranza di risolvere il quesito di Marta.

Teorema: La deviazione totale lungo un qualsiasi percorso chiuso è un multiplo intero di 360° .

Il numero intero, che esprime il multiplo in questione si chiama "numero di rotazioni" (*rotation number*). Lo chiamiamo qui R .

Per deviazione totale (*total turn*) si intende la somma algebrica di tutti gli angoli di cui la Tartaruga ha ruotato lungo il percorso. Ad esempio il frammento di codice RIGHT 20 FORWARD 10 LEFT 5 produce una deviazione totale di 15° perché la somma algebrica delle deviazioni è pari a $20 - 5$.

Un percorso si intende che sia chiuso qualora la Tartaruga si ritrovi esattamente nello stato da cui era partita. È bene capire che ciò non significa solo raggiungere il punto di partenza ma arrivarci anche con la medesima orientazione perché, ricordiamo, nella Geometria della Tartaruga conta lo "stato" che comprende sia posizione che direzione della Tartaruga.

AIutiamoci con un disegno per comprendere il senso di questo teorema.

²Il testo di Abelson e diSessa fornisce tutti gli elementi ma con una logica didattica: le dimostrazioni non seguono sempre linearmente i rispettivi enunciati, spesso sono diluite in altre parti del testo o sono in parte incluse negli esercizi. Infatti il testo è ricco di esercizi, nell'intento di stimolare quanto più possibile la riflessione autonoma del lettore.

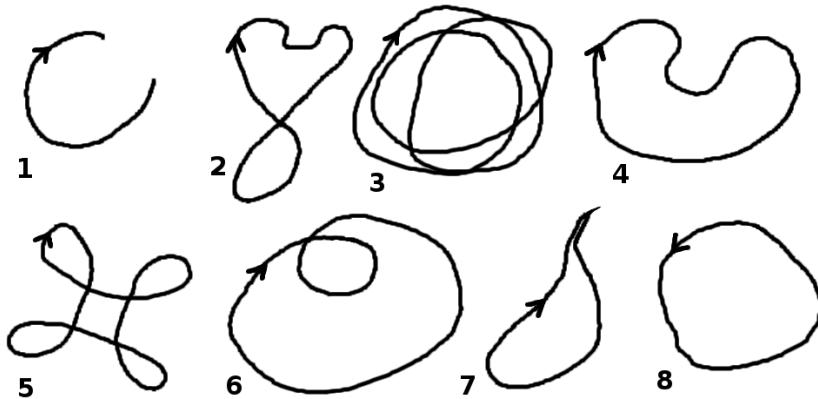


Figura 10.1: Quali sono i percorsi aperti? Provate a immaginare i valori R dei percorsi chiusi.

Dopo averci pensato un po', puoi vedere la soluzione in nota³.

10.4.2 Teorema del percorso chiuso semplice

Teorema: La deviazione totale lungo un qualsiasi percorso chiuso semplice è $\pm 360^\circ$.

Qui, oltre alle definizioni precedenti va aggiunta quella di percorso chiuso semplice, che è un percorso senza alcun incrocio. Il segno è determinato dal verso di rotazione: positivo il senso orario, negativo quello antiorario.

Questo teorema è un caso particolare del precedente ed è quello che serve, ad esempio, per il calcolo degli angoli di deviazione nel disegno di un poligono regolare: essendo la deviazione totale sempre pari a 360° , dato il numero N di lati del poligono, e tutte le singole deviazioni eguali fra loro, ciascuna di esse risulterà pari $360/N$.

10.4.3 Teorema di chiusura di una procedura POLY

Questo teorema ci avvicina al nostro quesito che, ricordiamo, riguarda la molto particolare esplorazione di Marta, che si era divertita a disegnare ripetutamente una casetta, ruotandola ogni volta di un angolo α . Riportiamo qui

Teorema: Un percorso tracciato da una procedura POLY si chiude esattamente quando la deviazione totale raggiunge un multiplo di 360° .

Una procedura POLY è una procedura del tipo

³1: aperto; 2: $R = 0$; 3: $R = 3$; 4: $R = 1$; 5: $R = 3$; 6: $R = 2$; 7: aperto; 8: $R = -1$.

```

TO POLY LATO ANGOLO
REPEAT [
  FORWARD LATO
  RIGHT ANGOLO
]
END

```

oppure del tipo

```

TO POLY LATO ANGOLO
  FORWARD LATO
  RIGHT ANGOLO
  POLI LATO ANGOLO
END

```

Le due versioni differiscono unicamente per il fatto che la prima usa il costrutto REPEAT mentre la seconda impiega la ricorsione. È utile osservare che, dei due parametri richiesti da POLY, il primo, LATO, determina solo la scala del disegno, mentre il secondo, ANGOLO, determina la forma della figura. È su questo parametro che ci concentremo quindi nel seguito. Il parametro LATO può essere utilizzato liberamente per aggiustare la scala della figura ma non influenza in alcun modo la sua forma né, di conseguenza, la questione della chiusura della medesima.

Vediamo alcuni esempi di disegni realizzati con la procedura POLY.

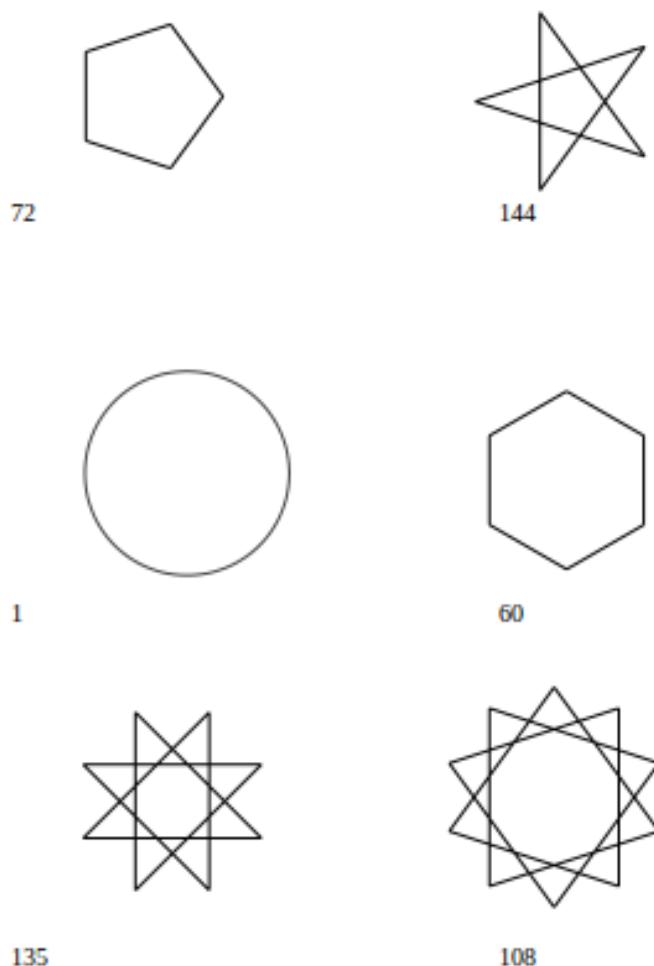


Figura 10.2: Disegni fatti con la procedura POLY per diversi valori della variabile ANGOLO.

Facendo girare la procedura POLY, la Tartaruga non si ferma dopo avere disegnato la figura ma la ripassa all'infinito perché è priva di un criterio di stop. Ma quand'è che la figura si chiude, nel senso che viene completata? Questo teorema fornisce il criterio fondamentale per rispondere alla domanda. Guardando i casi particolari raffigurati, vediamo che quando l'angolo è un sottomultiplo di 360 allora la figura è un poligono regolare. In questi casi il fatto è intuitivo: il poligono si chiude quando sono stati disegnati tutti i suoi lati, in pratica in un numero di passi pari a $360/\text{ANGOLO}$. Quando invece l'angolo non è un sottomultiplo di 360 non è così semplice ma il teorema ci dice che in ogni caso la figura si chiuderà la prima volta che il numero di passi sarà eguale a un multiplo di 360.

Va osservato che il teorema di chiusura di una procedura POLY non vale in

un caso particolare e poco interessante che si verifica quando l'angolo (ANGOLO) è uguale a zero o a multipli di 360 gradi. In questi casi la procedura POLY produce semplicemente una linea retta perché non vi è nessuna deviazione. Lo citiamo giusto perché lo ritroveremo successivamente.

Ma possiamo utilizzare questo teorema per rispondere al nostro quesito? Di fatto no, perché POLY non riproduce la struttura dei disegni di Marta, basati sul tracciamento della classica casetta più una deviazione finale. Tuttavia, fortunatamente, la Geometria della Tartaruga contiene anche un lemma⁴ che è, finalmente quello che fa al caso nostro:

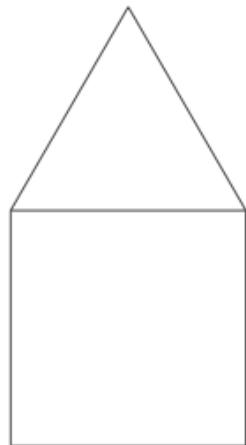
lemma: Qualsiasi programma costituito dalla ripetizione di un ciclo di base composto da comandi alla Tartaruga si comporta come un programma POLY al quale sia stato assegnato un angolo pari alla deviazione totale T di un singolo ciclo.

Conviene qui figurarsi cosa sia un "programma costituito dalla ripetizione di un ciclo di base composto da comandi alla Tartaruga". Comandi alla Tartaruga sono per esempio quelli di direzione, RIGHT e LEFT, e quelli di movimento, FORWARD e BACK. Questi sono detti comandi di "cambiamento di stato", perché sono in grado di alterare lo stato della Tartaruga, determinato da posizione e direzione. Se chiamiamo GPOLY questo tipo di programma, questo ha una struttura del tipo

```
TO GPOLY LATO ANGOLO
REPEAT [
    Qualsiasi sequenza di comandi di "cambiamento di stato"
]
END
```

Come in POLY qui abbiamo i parametri LATO e ANGOLO ma non è affatto detto che siano necessari. Potremmo avere una sequenza di comandi di "cambiamento di stato" fissa, che non ha bisogno di parametri, come nell'esempio della casetta di Marta, che riscriviamo qui sotto:

⁴I lemmi sono teoremi che derivano molto direttamente da un altro teorema. Non ci serve sapere altro a riguardo qui.



FORWARD 100
 RIGHT 90
 FORWARD 100
 RIGHT 90
 FORWARD 100
 RIGHT 90
 FORWARD 100
 RIGHT 90
 FORWARD 100
 RIGHT 30
 FORWARD 100
 RIGHT 120
 FORWARD 100
RIGHT 30

dove l'ultima istruzione è quella con cui Marta "perturbava" il ciclo di ripetizione delle casette. Inseriamo quindi questo frammento di codice in GPOLY:

```

TO GPOLY LATO ANGOLO
REPEAT [
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 30
  FORWARD 100
  RIGHT 120
  FORWARD 100
RIGHT 45
]
END

```

dove la parte colorata è la nostra sequenza di comandi di "cambiamento di stato": in magenta le istruzioni che disegnano la casetta e in giallo l'istruzione di "perturbazione". Il lemma ci dice dunque che questo programma si comporta come un programma POLY al quale sia stato assegnato un valore di ANGOLO pari alla deviazione totale T che la tartaruga subisce in seguito alla sequenza di comandi di "cambiamento di stato". La deviazione totale si calcola facilmente facendo la somma algebrica delle deviazioni. La parte dovuta alla casetta risulta pari a 510° mentre la perturbazione, nell'esempio mostrato è pari a 45° . Quindi, ai fini del lemma, la deviazione totale dovuta al singolo ciclo di Marta, T_m risulta $T_m = 555$.

Benissimo ma che ci facciamo con tutto ciò? In altre parole, che conseguenze pratiche comporta il fatto che GPOLY “si comporti come un programma POLY al quale sia stato assegnato un angolo pari alla deviazione totale T di un singolo ciclo”? Ebbene, quando il lemma ci dice che GPOLY si comporta come POLY significa che valgono per ambedue le stesse regole e, in particolare, vale il teorema della chiusura 10.4.3, secondo cui un percorso tracciato da una procedura POLY si chiude esattamente quando la deviazione totale raggiunge un multiplo di 360° , quindi lo stesso deve valere per GPOLY. In generale la deviazione totale T dopo un numero n di ripetizioni sarà pari a $T = nT_m$. Il teorema ci dice che la chiusura si verificherà la prima volta che la deviazione totale coinciderà con un multiplo di 360° . Se R è il numero di rotazioni per cui si verifica tale condizione allora avremo che

$$nT_m = R360 \quad (10.4)$$

Siccome n e R sono numeri interi, dire che vale la 10.4 equivale a dire trovare il minimo comune multiplo fra la deviazione totale di un ciclo, T_m , e 360. Nel nostro esempio dobbiamo quindi trovare il minimo comune multiplo fra $T_m = 555$ e 360, che vale 13320:

$$nT_m = R360 = 13320 \quad (10.5)$$

Se ora noi vogliamo sapere quanto vale il numero di cicli necessari per chiudere la figura non abbiamo che da calcolare:

$$n = 13320/T_m = 13320/555 = 24 \quad (10.6)$$

In generale quindi per rispondere al quesito di Marta occorre determinare la deviazione totale T_m prodotta dal ciclo interno e poi calcolare direttamente il numero n di passi necessari per chiudere la figura con

$$n = \text{MCM}(T_m, 360)/T_m \quad (10.7)$$

dove $\text{MCM}(n, m)$ rappresenta il minimo comune multiplo dei numeri interi n e m .

10.4.4 Codifichiamo la soluzione

La condizione di chiusura che abbiamo appena ricavato può essere facilmente codificata in Logo. Qui sotto riportiamo un esempio. I curiosi possono scaricare il file in formato odt⁵ e girarlo in LibreOffice dopo avere attivato la *toolbar Logo*.

```
; Copyright 2018 Andreas Robert Formiconi
; Programma distribuito alle condizioni della GNU General Public License
; Questo programma è Software Libero (Free Software): può essere
; ridistribuito e modificato nei termini della GNU General Public
; License pubblicata dalla Free Software Foundation, nella versione 3 o
; una delle successive. Il testo della licenza è accessibile in
; <https://www.gnu.org/licenses/licenses.it.html>.
```

⁵Il codice di GPOLY può essere scaricato da <http://iamarf.ch/unifi/GPOLY.odt>

```

; Risposta al quesito di Marta: come fermare un ciclo
; ripetuto sul disegno della casetta con una deviazione
; finale quando la figura viene completata?

; Questo codice implementa la soluzione basata sui
; teoremi di chiusura dei cicli poligonali esposti in
; Turtle Geometry (Abelson e diSessa , MIT Press , 1986)

; Calcolo del massimo comune divisore
;(Greatest Common Divisor), serve per
; calcolare il minimo comune multiplo.
; Il GCD viene memorizzato nella
; variabile globale R

TO GCD A B
GLOBAL R
IF B = 0 [ R = A STOP ] [ C = A % B GCD B C ]
END

; Calcolo del minimo comune multiplo
; (Least Common Multiple).
; Il LCM viene memorizzato nella
; variabile globale L

TO LCM A B
GLOBAL R, L
GCD A B
L = A * B / R
END

; Inizio programma principale
CLEARSCREEN
HOME

; Chiede l'angolo di deviazione dopo
; avere disegnato un ciclo-casetta
; espresso rispetto alla verticale
; secondo la convenzione di Marta

ALPHAM = FLOAT ~
(INPUT "Angolo di Marta (rispetto alla verticale)?")

; Calcola l'angolo di deviazione dopo
; avere disegnato un ciclo-casetta
ALPHA = ALPHAM + 30

; Calcola la deviazione totale T in un
; ciclo casetta (510 gradi) più
; deviazione finale ALPHA
T = ALPHA + 510

; Calcola il minimo comune multiplo fra
; 360 e la deviazione totale T
LCM 360 T

; Calcola il numero di rotazioni N
N = L/T

PRINT "Angolo=" + STR(ALPHA) + ~
"; Angolo rispetto alla verticale=" + STR(ALPHAM) + ~
"; Deviazione totale=" + STR(T) + ~
"; Numero rotazioni =" + STR(N)

; Esegue il disegno
REPEAT N [
FORWARD 100
RIGHT 9FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
]
```

```

RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT ALPHA
]

```

Tutto ciò di cui abbiamo bisogno per rispondere al quesito è il calcolo di un minimo comune multiplo. Ci sono tanti algoritmi per farlo, qui abbiamo preso semplicemente il primo che abbiamo codificato alla svelta. L'algoritmo ha bisogno di calcolare, a sua volta, un massimo comun divisore. Per quest'ultimo abbiamo scelto una versione che utilizza uno schema ricorsivo. Un'altra particolarità da segnalare è che abbiamo utilizzato una variabile globale per recuperare sia il massimo comun divisore che il minimo comune multiplo dalle rispettive procedure. Le variabili di questo tipo devono essere dichiarate con l'istruzione GLOBAL (ad esempio GLOBAL R, L) nelle procedure che le utilizza. Così facendo tali variabili risultano visibili in tutto il programma senza il bisogno di essere passate o rese esplicitamente.

Quando si fa girare GPOLY, questo chiede il valore dell'angolo di deviazione, espresso secondo la convenzione di Marta, ovvero rispetto alla verticale, che abbiamo chiamato α_m . L'angolo di deviazione effettivo α , che va a sommarsi alla deviazione totale prodotta disegnando una casetta, differisce per 30° . Prima di disegnare la figura, il programma scrive i valori calcolati, nella forma seguente:

Angolo=60.0; Angolo rispetto alla verticale=30.0; Deviazione totale=570.0;
Numero rotazioni =12.0.

Vediamo i risultati prodotti da questo codice in una tabella simile a quella che avevamo compilato (10.3) cercando la soluzione numerica, dove abbiamo aggiunto l'angolo di deviazione α effettivamente applicato dopo ogni ciclo-casetta, oltre a quello espresso rispetto alla verticale α_m (c'è una differenza di 30°), e la deviazione totale T_m :

Deviazione finale α	Deviazione finale $\alpha_m = \alpha - 30$ rispetto alla verticale (convenzione di Marta)	Deviazione totale $T_m = \alpha + 510$	Numero di rotazioni n per chiudere la figura (Eq. 10.7)
30	0	540	2
60	30	570	12
90	60	600	3
120	90	630	4
150	120	660	6
180	150	690	12
210	180	720	∞

Questi sono solo alcuni esempi, si può inserire qualsiasi valore per l'angolo di deviazione e il risultato è garantito. Nella tabella abbiamo inserito anche un caso che rappresenta il caso particolare a cui avevamo accennato a proposito del teorema di chiusura di una procedura POLY, dove la deviazione valeva zero o

un multiplo di 360° . Nella procedura GPOLY l'eccezione si presenta quando la deviazione totale, T_m , vale zero o è un multiplo di 360° . Nella tabella il caso si verifica per $\alpha_m = 180$, che produce $T_m = 720$, multiplo di 360 . Per il numero di rotazioni n abbiamo scritto infinito, perché è quello che succede se lasciamo andare il codice senza alcun criterio di stop, disegnando una successione infinita di casette disposte lungo una linea retta. Di fatto il nostro codice in questi casi produce il disegno di una sola casetta. Quindi può essere applicato senza sorprese in qualsiasi caso.

Unico accorgimento: certi valori della deviazione possono produrre disegni che occupano uno spazio maggiore, debordando dal foglio. In questi casi si tratterebbe di scalare adeguatamente le dimensioni della casetta, ad esempio introducendo una variabile al posto degli attuali spostamenti di 100 punti, in modo da poter scalare adeguatamente tutto il disegno. Lasciamo al lettore questo esercizio.

10.5 Conclusione

Abbiamo fatto una discreta quantità di strada a partire dalle prime fasi dell'esplorazione di Marta. Non avremmo certo immaginato di arrivare sin qui prendendo le mosse dal disegno di una semplice casetta. Perché abbiamo fatto questo? Per almeno tre motivi.

10.5.1 Ragionamento e esplorazione

L'equilibrio fra ragionamento, pianificazione, pensiero deduttivo da un lato e esplorazione, meraviglia, pensiero induttivo dall'altro, è uno dei cardini del pensiero di Papert. Se i primi costituiscono strumenti potenti per l'apprendimento e la capacità di progettare e costruire, i secondi sono fondamentali per la generazione di motivazione e la creatività. Specialmente in giovane età, l'imposizione di schemi esageratamente "ingenieristici" può rivelarsi per molti un deterrente all'avvicinamento alla materia. Ma dall'altro lato, un percorso composto solo da elementi ludici, privo di momenti di riflessione analitica e di pianificazione può condurre alla superficialità e all'inconcludenza. L'abilità dell'insegnante che voglia sfruttare le potenzialità di Logo consiste proprio nel guidare gli allievi bilanciandosi fra questi due opposti.

L'esplorazione inviata da Marta è stata un vero e proprio regalo. Un'ottima esemplificazione di questo equilibrio. Dal ragionamento necessario per costruire prima la casetta, poi da queste figure più complesse applicando il costrutto della ripetizione, Marta conduce il lettore nel territorio della scoperta e dell'immaginazione. Poi, torna all'analisi, intavolando un ragionamento che la porta a formulare una domanda di natura geometrica e chiude con un quesito. Non potevo che inserire il suo contributo tal quale. Poi, un altro allievo nel contesto di una scuola di perfezionamento, Alberto, ha rilanciato la questione, suggerendo un approfondimento che ci ha consentito di comprendere meglio cosa sia il significato di numero nel contesto digitale.

10.5.2 Vivere in prima persona l'esperienza che vorremo/dovremmo fare vivere ai nostri allievi

Un'altra studentessa, Eleonora, aveva inviato un contributo volontario, altrettanto gradito, dal titolo "Se incontri un professore che ti tratta come un bambino". Questo è un aspetto importante, in tema di formazione di futuri insegnanti. È notorio che si impara veramente bene ciò che si è vissuto, molto meglio di ciò che si è studiato giusto per superare un esame. L'impostazione del laboratorio di tecnologie didattiche è incardinata esattamente su questo punto: creare esperienze significative sui temi oggetto del corso che poi possano essere situate nei contesti del lavoro. Al pari di Eleonora, Marta ha interpretato perfettamente la proposta didattica, e lo ha fatto in maniera fattiva e consapevole, calandosi nei panni del bambino che si meraviglia, ovviamente *mutatis mutandis*. Non si deve certo immaginare di portare l'esplorazione di Marta tal quale in una classe di scuola primaria, non è questo il punto. Il contenuto dell'esplorazione deve arridere l'adulto affinché costui possa rendersi consapevole del processo che dovrà un giorno sollecitare nei propri allievi. Poi costui dovrà situare l'azione nei modi e con i contenuti appropriati, in funzione dell'ordine, del tipo di scuola e delle caratteristiche specifiche, anche individuali, dei ragazzi.

10.5.3 Percepire la potenziale dimensione verticale di LibreLogo

Senza entrare nelle complesse e variegate interpretazioni del concetto di curriculo verticale, ho cercato di evidenziare la dimensione verticale che anche uno strumento semplice come LibreLogo può avere. Il desiderio di mettere in luce questo aspetto deriva da due esperienze opposte: discutendo di applicazioni di coding nei primi ordini di scuole emerge spesso l'idea che si tratti di pratiche che non possono essere impiegate negli anni successivi, perché troppo banali e meccaniche; all'inverso, in contesti di scuola secondaria superiore si ritiene che il coding comporti una complessità eccessiva per gli allievi più piccoli. Dove sta dunque la verità? Come spesso succede né dall'una né dall'altra parte.

La successione dei due interventi, di Marta, studentessa di formazione primaria, e di Alberto, insegnante di informatica in un istituto tecnico, si è prestata magnificamente per esplorare la dimensione verticale dello strumento, in misura estrema, se vogliamo, perché siamo partiti dal disegno di una casetta e siamo finiti a discutere del concetto di numero, inteso nella sua forma matematica pura, o nella forma digitalizzata idonea ad essere progettata in un computer.

Nel capitolo successivo spingeremo ancora oltre questo genere di considerazioni.

Capitolo 11

Girando in tondo: dal cerchio all'orbita di Halley

11.1 Low floor...



Figura 11.1: Per le fotografie da cui ho tratto le immagini e per buona parte dei testi in questa sezione sono debitore della maestra Antonella Colombo che ringrazio per avere condiviso le sue preziose esperienze.

Percorrendo il caotico dibattito intorno al coding non è raro imbattersi in coloro i quali pensano che si tratti di pratiche buone giusto per i primi anni di scuola come in quelli che invece ritengono che il coding sia adatto agli ultimi anni delle scuole superiori. Sono ambedue in errore. Per illustrare la questione, anche in questo capitolo esploriamo una sorta di progressione dal *low floor* allo *high ceiling* di Papert. Vale la pena di riprendere le sue parole.

L'obiettivo delle prime esperienze dei bambini nell'ambiente di apprendimento della Tartaruga non è quello di 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 muovi in 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.

...

L'episodio del cerchio disegnato con la Tartaruga illustra l'apprendimento sintonico. 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 perché tale cerchio è saldamente collegato alla percezione fisica del proprio corpo da parte del bambino. Oppure è anche sintonico per l'ego 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.

Una delle cose più belle che mi capita di vivere in questi anni è la ricca e spontanea "restituzione" da parte dei miei studenti, sia da i più giovani che sono in procinto di laurearsi che da coloro che incontro in percorsi di aggiornamento di vario tipo. Un patrimonio inestimabile che consente di realizzare quasi un miracolo, quello di situare immediatamente ciò di stiamo parlando. E allora qui ci possiamo prendere il lusso di passare dal racconto di Papert delle proprie visioni a quello dove Antonella, una maestra di Paderno d'Adda, a tali visioni ha dato vita nella propria classe. Il racconto è nella forma del diario, che è un forma di compito che chiedo quasi sempre a tutti. Mi defilo nuovamente e lascio la parola questa volta a Antonella.

23 maggio 2017

Anch'io ho utilizzato Code.org in questi ultimi anni con buoni risultati. Quest'anno in prima ho tentato – dopo averne sentito parlare in questo ambiente e grazie ad un progetto che si attua nel mio Istituto e che prevede l'intervento nelle classi di ragazzi di un Istituto di Scuola superiore ad indirizzo di Informatica (alternanza scuola-lavoro) – di mandare avanti i due sistemi in parallelo: la classe è stata divisa in due gruppi, un gruppo lavorava con Logo, l'altra con l'Ora del Codice

Dopo un'ora cambio di gruppo; io ho seguito maggiormente il gruppo che lavorava con Logo, l'altro gruppo è stato seguito dall'insegnante di sostegno compresente. In realtà, prima ancora che arrivassero i ragazzi dell'ITS, avevo già avviato i bambini all'uso di Code.org.

I ragazzi vengono preventivamente informati su che cosa devono proporre alle varie classi: si parte dall'ora del Codice nelle prime classi per arrivare in quarta-quinta a lavorare con il Lego WeDo e Scratch. Per queste due ultime attività è stato predisposto un mini laboratorio con 6 notebook.

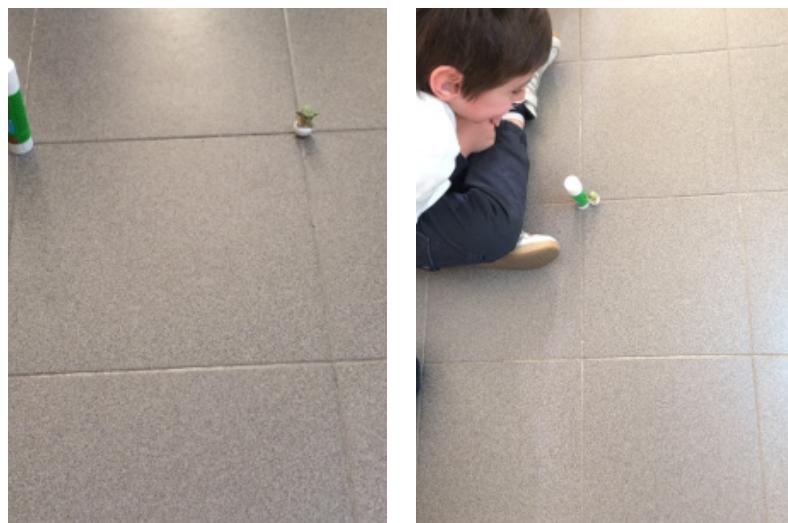
Oggi era il turno delle mia classe e finalmente ho realizzato il mio progetto: ho diviso la classe in due gruppi, un gruppo nel laboratorio grande ha continuato l'ora del codice con l'insegnante di sostegno, un gruppo con me nel laboratorio più piccolo si è avvicinato al linguaggio della tartaruga.

Prima di arrivare a capire le istruzioni per disegnare un quadrato ho "utilizzato" il loro corpo: ho chiesto loro di camminare e di descrivere un quadrato, cercando di dare istruzioni precise.

Vado avanti, poi giro, poi giro, poi giro e arrivo al punto di partenza.
Non siamo ancora precisi, riproviamo.

138CAPITOLO 11. GIRANDO IN TONDO: DAL CERCHIO ALL'ORBITA DI HALLEY

Consegno ad ogni bambino un piccolo pupazzetto e chiedo di farlo camminare sul bordo di una piastrella, introduco i comandi, perché mentre noi proviamo, il nostro studente Davide (ex alunno) sul foglio di writer traduce i nostri comandi, così impariamo che forward significa vai avanti, right gira a destra.



Chiedo a Davide di far partire la tartaruga, percorre un tratto, ma poi non si gira, anche se le abbiamo detto di girare a destra, allora spiego che bisogna dirle di girare di 90 (gradi): i bambini osservano che la tartaruga si gira e allora...

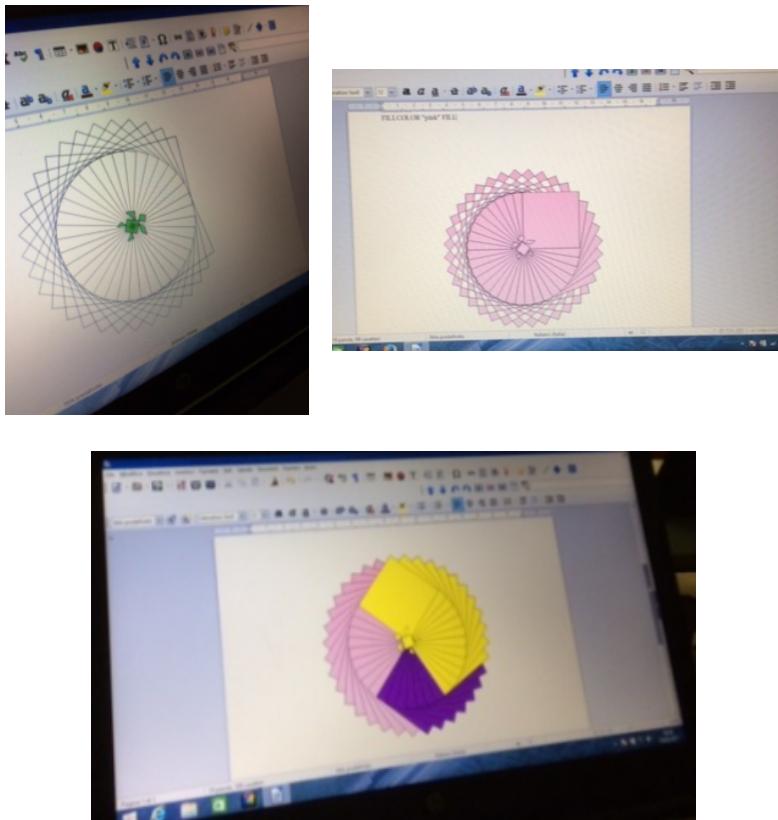
```
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 100
```

Davide scrive i comandi, introduce i vari tasti e ...magia finalmente la tartaruga disegna il quadrato; intenzionalmente decido di non scrivere home, così la tartaruga può continuare a muoversi e formare quattro quadrati. È tutto questo ha già il suo fascino. Successivamente introduco una rotazione a caso della tartaruga e le chiediamo di continuare a lavorare: "che bello, sembrano le nostre cornicette" commenta Giulia; ma l'osservazione più significativa, che formula sempre Giulia, è: "Guarda, sta diventando rotondo".

I bambini a coppie al portatile aprono LibreOffice e fanno da soli la stessa cosa che abbiamo visto insieme con interpretazioni differenti

sull'ultima rotazione.

Introduco anche il comando del colore e anche qui dopo qualche tentativo avvengono libere interpretazioni e combinazioni di colori.



Abbiamo avuto un momento di pausa e la settimana scorsa sono tornati di nuovo gli studenti con i quali abbiamo ripreso l'esperienza. Ho colto un'occasione d'oro quando è uscita la domanda: "Ma come si disegna il cerchio?" "Secondo voi?" ho risposto. Sono partite le ipotesi, si sono avvicinati, ma non hanno ancora trovato la soluzione; poi, ad un certo punto, dopo che hanno intuito che right 1 poteva essere utile, una bambina ha osservato: "Con sempre right la tartaruga gira su se stessa e non si muove".

Quello che ho trovato di molto significativo in questa esperienza è che la progettazione aiuta i bambini a mobilitare e raggiungere molte competenze, per esempio destra e sinistra: in prima si dedica parecchio tempo a questo obiettivo, schede, esercizi..., forse l'attività di progettazione, come del resto affermava Papert, è più stimolante e lascia tracce indelebili nei processi di apprendimento.

Sabato, 3 giugno

Ripreendo solo oggi le fila del discorso. Il 23 maggio appunto ho voluto sperimentare la dimensione sintonica dell'apprendimento. Siamo

140 CAPITOLO 11. GIRANDO IN TONDO: DAL CERCHIO ALL'ORBITA DI HALLEY

andati in corridoio e prima abbiamo sperimentato con il corpo i comandi che abbiamo imparato a dare alla tartaruga

L'obiettivo era quello di scoprire i comandi per il cerchio, allora abbiamo per prima cosa percorso un disegno che c'è a terra nel nostro corridoio, osservando attentamente i nostri movimenti.



Poi abbiamo abbiammo camminato sul bordo di un cerchio che si usa in educazione motoria in palestra.



Prima di far continuare Antonella, è interessante notare nella foto a destra come la bambina cerchi di seguire la curva della circonferenza descrivendo così, con i piedi, il processo: faccio un piccolo passo e giro un poco, faccio un piccolo passo e giro un poco...

Infine..."via le calzine, tracciamo un cerchio con le orme dei nostri piedini, camminando sulla farina, come se fossimo al mare" (complice il tempo che era particolarmente soleggiato e caldo!).



Dopo l'esperienza gioiosa e coinvolgente del camminare a piedi scalzi sulla farina, c'è il momento della discussione: quali comandi dare alla tartaruga per disegnare il cerchio?

La settimana scorsa ho proposto ai bambini di usare il pc e di provare a disegnare il cerchio: usano forward e right, ma assegnano valori come 90, 40...; io suggerisco di verificare che cosa disegna la tartaruga con le procedure scritte: procedono dunque per prove ed errori. Ben presto si rendono conto che qualcosa non va.

"Bisogna cambiare i numeri"

"Dobbiamo andare in curva"

"Bisogna continuare a fare right"

"Se facciamo sempre right, viene un punto piccolissimo".

Poi l'insight:

"Dobbiamo fare forward 1 right 1".

"Proviamo" suggerisco io

"Bisogna ripeterlo tante tante volte."

Li aiuto a scrivere, faccio finta di non sapere e di essere stanca di scrivere, per cui faccio vedere che con Ctrl+V il comando può essere ripetuto tante volte quanto vogliamo noi, per cui continuo a cliccare Ctrl+V finché loro mi dicono stop.

La tartaruga disegna e traccia un arco: è già qualcosa. Ne approfitto per introdurre repeat e chiedo quante volte devo far ripetere alla tartaruga forward1, right 1.

100...200...300 e intanto, grazie a questi tentativi, la tartaruga traccia archi sempre più ampi. Manca poco, Giovanni mi propone di

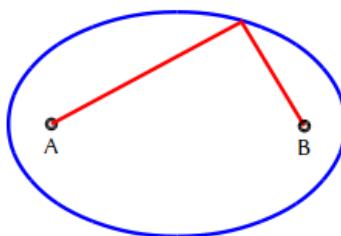
scrivere 355, ma in classe ho delle ragazzine di V (la loro maestra è assente e tutti gli alunni sono stati distribuiti nelle classi) che interrompono il loro lavoro e seguono il nostro, le coinvolgo e i bambini chiedono il loro parere – "Perché sono di V, sono più grandi di noi". Immediatamente mi rispondono 360 e finalmente la tartaruga disegna un cerchio.

È affascinante come queste prime esperienze possano indurre a riflettere sul concetto di cerchio senza che se ne abbia ancora una conoscenza formale. Eppure, così facendo, si lambisce un concetto assai più avanzato di tutto ciò che i ragazzi vedranno prima della maturità, ovvero il concetto di calcolo differenziale. Tuttavia, appena essi verranno in possesso della nozione formale di circonferenza, sarà possibile tornare in Logo e riprendere il discorso attraverso il modo sintetico di produrre un cerchio, con l'istruzione CIRCLE D, dove D rappresenta il diametro del cerchio, riflettendo sulla definizione come luogo dei punti equidistanti da un dato punto. E qui si potrebbe andare oltre, partendo dall'idea di un "cerchio sbagliato", o "schiacciato", e ricorrendo alle riflessioni di Emma Castelnuovo [3].

Sempre un argomento di matematica, quale lo studio dei triangoli isoperimetrici con uguale 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 sennaletico 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.

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 divergono.

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.

Sono esperienze fisiche quella che propone Emma Castelnuovo ma non vanno considerate in contrapposizione a quelle che si possono fare con un computer, nel modo che abbiamo mostrato; che poi è il modo che suggerisce Papert attraverso il concetto di apprendimento sintonico.

Purtroppo l'avvento del nuovo viene solitamente vissuto in contrapposizione al vecchio, generando l'usuale dicotomia integrati-apocalittici. È il tempo a risolvere solitamente ciò che la saggezza potrebbe evitare subito, con molto maggiore profitto. Quella da promuovere è quindi una visione integrata, dove si fa ricorso a tutti i mezzi possibili, tradizionali e moderni, manuali e virtuali, per raggiungere gli obiettivi didattici in una prospettiva più ampia e completa possibile.

11.2 ... high ceiling

Fin qui abbiamo esplorato "il tondo" dal basso, dal *low floor* di Papert, almeno in parte. Molto altro potrebbe essere aggiunto, ma ora, per rifarsi al quesito iniziale, proviamo a muoverci verso l'alto per vedere dove si possa trovare l'*high ceiling*. Si tratta di un quesito che è emerso per esempio recentemente durante un incontro con un nutrito gruppo di animatori digitali, dove ci si poneva la domanda: ma si può fare il *coding* anche negli anni successivi, alle scuole superiori? Molti sostenevano di no, ritenendo che si trattasse di pratiche troppo banali.

Riprendiamo le parole di Papert.

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 soggiacciono 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 differenziale un'equazione. In ciò non vi è alcun riferimento a luoghi remoti rispetto al percorso. La Tartaruga vede il cerchio cammin 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, 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:

$$(x - a)^2 + (y - b)^2 = R^2 \quad (11.1)$$

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 si stabilisce 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 [NdR: di Minstorms], dove descriveremo 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 intorno 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.

Prendiamo quindi le mosse dal riferimento di Papert alla fisica di Newton, rifacendosi dai principi della dinamica e dalla legge di gravitazione universale.

11.2.1 La visione analitica classica

Poniamo il problema in maniera generale. Non potrà essere questa l'impostazione proposta a scuola, perché occorrono conoscenze matematiche superiori, ma qui ci serve per inquadrare e comprendere correttamente la questione.

Affrontiamo il problema di due corpi che siano isolati da perturbazioni esterne. Ad esempio quello di un satellite che gira intorno alla terra o di una cometa che gira attorno a un pianeta. Il moto di questi corpi è governato dalla legge di gravitazione universale di Newton:

$$\mathbf{F}_{Mm} = -\frac{GMm \mathbf{r}}{r^2} \quad (11.2)$$

Dove \mathbf{F}_{Mm} è la forza, G la costante di gravitazione universale, pari a 6.67×10^{-11} ($\text{Newton} \times \text{metri}^2/\text{Kg}^2$), M è la massa del pianeta, m la massa del satellite, \mathbf{r} la distanza fra i baricentri dei due corpi e dove, ricordiamo, il grassetto si denota per indicare la naturale vettoriale di una quantità. Quindi in questo specifico caso, poiché il nostro ragionamento si svolge tutto in due dimensioni, la forza \mathbf{F}_{Mm} e la distanza \mathbf{r} sono vettori con due componenti: una lungo x e l'altra lungo y .

Ma cosa vuol dire in pratica che "il moto dei corpi è governato dalla legge di gravitazione"? Significa che, per individuare il moto di un corpo, dobbiamo applicare il II principio della dinamica, secondo il quale:

$$\mathbf{F} = m\mathbf{a} \quad (11.3)$$

Dove \mathbf{F} è la forza applicata al corpo di massa \mathbf{F} e \mathbf{a} è l'accelerazione impressa al corpo in virtù di tale forza. Lo scopo di queste relazioni è quello di consentire la determinazione del moto dei corpi a partire da predeterminate condizioni iniziali. Per fare questo occorre riscrivere l'equazione 11.3 tenendo conto della forza data nell'equazione 11.2:

$$-\frac{GMm \mathbf{r}}{r^2} = m\mathbf{a} \quad (11.4)$$

Risolvere questa equazione significa in primo luogo ricavare l'accelerazione \mathbf{a} :

$$\mathbf{a} = -\frac{GM}{r^2} \frac{\mathbf{r}}{r} \quad (11.5)$$

e da questa derivare l'andamento nel tempo della velocità e quindi dell'accelerazione, tenendo conto che:

$$\mathbf{a} = \frac{d^2\mathbf{r}}{dt^2} \quad (11.6)$$

ovvero

$$\mathbf{v} = \frac{d\mathbf{r}}{dt}, \mathbf{a} = \frac{d\mathbf{v}}{dt} \quad (11.7)$$

È qui che ci dobbiamo fermare, perché siamo di fronte a equazioni dove non appaiono le grandezze incognite, ma le derivate di tali grandezze rispetto a un'altra variabile, il tempo in questo caso. In particolare, l'equazione 11.5 è un'equazione differenziale del II ordine perché fornisce un'espressione per l'accelerazione, ovvero la derivata seconda della posizione rispetto al tempo. Ci dobbiamo fermare perché la soluzione delle equazioni differenziali non rientra nell'orizzonte contemplato dai programmi della scuola secondaria. Di fatto l'approfondimento che ci possiamo permettere è assai limitato. L'apprendimento della fisica nella scuola si limita per di più alla memorizzazione di alcune formule e all'esecuzione di qualche esercizio, con procedimenti che la maggioranza degli studenti assimilano in maniera meccanica e effimera, senza realmente comprendere l'essenza dei concetti e tanto meno senza recepire il benché minimo sentore del "pensiero fisico", come del resto del "pensiero matematico". I risultati si vedono dal fatto che nella cittadinanza sopravvivono solo brandelli di pensiero scientifico, un pensiero che non ha quasi nulla a che vedere con l'avere imparato a memoria $F = ma$ una volta nella vita, giusto per superare un'interrogazione.

Ci colpisce tuttavia il fatto che di calcolo differenziale abbiamo già parlato, e che lo abbiamo fatto addirittura trovandoci al *low floor*, dove stavamo parlando di attività da fare con allievi assai più piccoli! Lì avevamo descritto, anzi Papert aveva descritto, il concetto come un processo locale: il bambino (o la Tartaruga) che produce un cerchio preoccupandosi solo di fare un passo e girare, senza alcuna nozione esplicita e sintetica del concetto di cerchio. Ebbene questa è proprio la descrizione verbale di un'equazione differenziale, dove anziché esprimere delle quantità in funzione di altre si esprimono le loro variazioni, le quali sono sempre inerentemente locali.

11.2.2 Física computazionale

Affrontiamo quindi il problema espresso dall'equazione 11.5 in maniera computazionale. Riscriviamo l'equazione così:

$$\frac{d^2\mathbf{r}}{dt^2} = -\frac{GM}{r^2} \frac{\mathbf{r}}{r} \quad (11.8)$$

O volendo così (ognuno faccia riferimento alla versione che preferisce):

$$\frac{d}{dt} \frac{d\mathbf{r}}{dt} = -\frac{GM}{r^2} \frac{\mathbf{r}}{r} \quad (11.9)$$

E affrontiamolo nel seguito come avevamo fatto con il cerchio alla maniera di Papert, dove si faceva un piccolo passo per volta e poi si girava di un po', e via di seguito. Anche qui facciamo un piccolo passo per volta.

11.2.3 Le condizioni iniziali

Ma prima di tutto: a partire da dove e in quale direzione dobbiamo fare il passo? Ecco questo è un primo concetto fondamentale del procedimento di conoscenza fisica: la determinazione delle condizioni iniziali, o delle "condizioni al contorno". Qui possiamo subito osservare come, ponendo il problema in termini computazionali anziché analitici, l'allievo sia costretto a misurarsi concretamente con il concetto basilare di condizione al contorno e in una maniera per lui perfettamente comprensibile. Nella fisica convenzionalmente insegnata al liceo questo concetto non appare o, al più, compare in un forma astratta che ben difficilmente lascerà qualche traccia nella mente del giovane. Orbene, il punto di partenza rappresenta la condizione iniziale della posizione, che richiederà che venga fissato un sistema di riferimento, sotto forma di coordinate (x_0, y_0) dell'origine di un riferimento cartesiano, e una posizione iniziale (x, y) del corpo. Qui verrà presa una decisione anch'essa di natura fisica: essendosi posti nella condizione di studiare le orbite di singoli corpi intorno a un pianeta molto più grande, potremo assumere che questo stia fermo e sarà quindi naturale far coincidere il centro (x_0, y_0) del sistema con il baricentro di questo. Così facendo avremo determinato la condizione iniziale del vettore posizione \mathbf{r} attraverso le due coppie di coordinate (x_0, y_0) e (x, y) . Poi dobbiamo decidere la dimensione del passo e in che direzione compierlo. Questo lo possiamo fare se sappiamo la velocità nel punto in cui ci troviamo. Potrebbe essere zero se immaginiamo semplicemente di lasciar cadere il corpo verso la terra, ma sarà invece un valore preciso se si tratterà di imprimere la giusta velocità iniziale a un satellite o a una cometa, affinché questi percorrano le orbite che ci aspettiamo. La velocità nel punto del primo passo è la velocità iniziale e la direzione sarà determinata attraverso la reciproca dimensione delle due componenti, lungo x e y , rispettivamente. Dopodiché, la dimensione del passo verrà determinata a partire dalla relazione $spazio = velocità \times tempo$, che caratterizza il concetto di velocità. In pratica, dal punto di vista computazionale, ovvero di istruzioni da impartire a un computer, dovremo scrivere qualcosa del genere

$$\begin{aligned} XPOS &= XPOS + XVEL * Dt \\ YPOS &= YPOS + YVEL * Dt \end{aligned} \tag{11.10}$$

Dove XVEL e YVEL rappresentano le componenti della velocità e Dt l'intervallo di tempo su cui abbiamo deciso di aggiustare il passo. Prima di commentare quest'ultimo elemento, per coloro che non avessero mai visto una scrittura del genere, diciamo che questo è uno dei modi per alterare – incrementare in questo caso – una variabile. In generale, scrivere $a = a + b$ significa prendere la somma dei valori di a e b e attribuire questo risultato alla variabile a . È un modo quindi per aggiornare il valore di una variabile: a destra del segno di $=$ si usa il vecchio valore di a , a sinistra c'è quello nuovo attribuito in base all'operazione fatta. Detto questo, finiamo di commentare il fattore Dt. Questo rappresenta l'intervallo di tempo nel quale viene percorso il tratto a quella velocità. Deve essere piccolo perché senon non si può ragionevolmente accettare che la velocità sia costante in quel tratto. Piccolo quanto?

11.2.4 The art of scientific computing

La risposta onesta è: bisogna vedere... Non esattamente soddisfacente come risposta per una scienza (un tempo) ritenuta "esatta". Esiste un testo fondamentale per chiunque si sia occupato o si occupi di calcolo scientifico, pubblicato la prima volta nel 1986: Numerical Recipes – The Art of Scientific Computing [15]. Quello del calcolo scientifico non è un mondo esatto. Sembra un paradosso: laddove le scienze esatte sposano la tecnologia si parla di "arte del calcolo scientifico". In parte perché ci sono metodi matematici estremamente sensibili ad ogni piccola variazione dei dati, che in certi contesti conducono all'impossibilità di risolvere problema, e in parte perché la rappresentazione in bit dei numeri è solo un'approssimazione della loro nozione matematica, questa sì esatta. Non solo, lo stesso identico calcolo eseguito su computer diversi può dare risultati differenti, come abbiamo visto nel capitolo precedente, perché non è identico l'insieme dei processi con i quali i computer manipolano l'imperfetta rappresentazione dei numeri.

Tornando alla questione dell'intervallino di tempo Dt , anch'essa dipende molto dal contesto. Il compromesso è un po' questo: se si fanno passi lunghi allora il calcolo procede veloce ma si rischia di sbagliare molto, in particolare quando la velocità cambia apprezzabilmente fra un passo e l'altro; per sbagliare meno occorre scegliere passi più brevi e quanto più questi sono brevi, tanto più il calcolo sarà accurato. Però anche qui fino a un certo punto, perché se gli intervalli diventano brevissimi si possono avere problemi di precisione nella rappresentazione dei numeri e quindi i conti possono nuovamente sbagliare, oltre a rischiare di dilatare inutilmente i tempi di calcolo, se non a renderli ingestibili.

Perché tutte queste considerazioni che possono parere anche esageratamente tecniche per il nostro contesto? Da un lato per tentare di rimettere al centro la complessità e la ricchezza del pensiero computazionale, che finisce con l'essere polverizzato in un discorso pubblico banale e partigiano, strapazzato nelle diatribe fra fazioni di apocalittici e integrati, o di "umanisti" e "scientifici". A proposito di quest'ultima dicotomia vorrei dire che chi vi crede tradisce allo stesso tempo sia la visione umanistica che quella scientifica, perché una vera visione umanistica certamente comprende e valorizza il pensiero scientifico. Il pensiero scientifico è umanistico, invece v'è ne poco in giro. Dall'altro perché questa complessità fornisce un terreno adatto per un approccio alla conoscenza di natura laboratoriale, che è quello che di fatto vivono il ricercatore, l'artista o l'artigiano, quando si misurano con la realtà nei rispettivi modi e con i rispettivi strumenti. Le incertezze del calcolo scientifico non devono certamente essere rovesciate sugli allievi come si fa nell'approccio disciplinare classico, bensì devono essere tenute presenti per essere eventualmente approfondite, in misura adeguata, qualora i loro effetti si presentino nel corso delle esplorazione.

Il codice nella formula 11.10, se rieseguito ripetutamente, disegna la traiettoria del corpo per incrementi successivi, che è proprio quello che desideriamo. Il problema tuttavia è che ci occorre conoscere la velocità in ogni punto della traiettoria ma noi questa informazione non ce l'abbiamo; soprattutto non abbiamo niente che leghi la grandezza velocità alla legge di gravitazione (Eq. 11.5), che è tutto ciò che sappiamo. In realtà nell'equazione 11.5 abbiamo l'accelerazione, che rappresenta la variazione della velocità. Quindi quello che ci occorre è un passaggio simile a quello che nel codice 11.10 ci è servito per ricostruire il percorso. Dobbiamo cioè ricostruire l'andamento della velocità per incrementi

successivi:

$$\begin{aligned} XVEL &= XVEL + XACC * Dt \\ YVEL &= YVEL + YACC * Dt \end{aligned} \quad (11.11)$$

Dove XACC e YACC rappresentano le componenti dell'accelerazione lungo le direzioni degli assi cartesiani X e Y. Questa ce l'abbiamo perché è proprio la legge di gravitazione 11.5 a darcela, che scissa nelle due componenti risulta:

$$\begin{aligned} a_x &= -\frac{GM}{r^2} \frac{r_x}{r} \\ a_y &= -\frac{GM}{r^2} \frac{r_y}{r} \end{aligned} \quad (11.12)$$

che rappresentiamo nel codice LibreLogo (vedi dopo) in questa forma:

$$\begin{aligned} XACC &= GG/R2 * DX/R \\ YACC &= GG/R2 * DY/R \end{aligned} \quad (11.13)$$

Letto così può parere complesso rispetto a quanto abbiamo visto ora. Certo, non stiamo facendo un corso di fisica ma non è questo il punto. L'importante è riconoscere i blocchi principali per afferrare i concetti importanti, intravedendo il nesso con il caso (apparentemente) più semplice del cerchio che abbiamo raccontato al *low floor*.

Prima descriviamo l'algoritmo con un pseudocodice, ovvero un codice informale che non può essere eseguito nel computer ma che serve a facilitare la comprensione degli algoritmi, scritto tenendo conto solo dei passaggi fondamentali e omettendo tutti i particolari che invece sono necessari alla macchina per potere eseguire il programma. Rammentiamo che i programmi in LibreLogo si scrivono anteponendo le procedure, in modo che, ogni volta che una di queste sia invocata, il suo codice sia già stato "letto" dalla tartaruga. Quindi il pseudocodice¹ che segue si riferisce al cosiddetto programma principale che, nel listato completo successivo inizia a partire dall'istruzione n. 134.

Data: Dati del sole e del corpo orbitante, condizioni iniziali

Result: Orbita del corpo

while *orbita non è chiusa* **do**

calcola prossimo punto con legge di gravitazione;

disegna prossimo punto;

scrive su file coordinate, velocità, accelerazioni nel prossimo punto

end

Algorithm 1: Pseudocodice per il calcolo dell'orbita di un corpo intorno al sole

¹Il pseudocodice è una sorta di linguaggio che sta à metà fra un vero linguaggio di programmazione, che può essere interpretato dal computer, e il linguaggio naturale. Le sue istruzioni non possono essere eseguite nella macchina ma servono a dare un'idea della struttura fondamentale di un algoritmo. In questi esempi abbiamo lasciato le pseudostrutture in inglese, considerato che in tutto questo ambito la lingua di riferimento è l'inglese. Del resto il significato è intuitivo. Aggiungiamo solo che il costrutto "**while** condizione **do** istruzioni **end**" esprime un ciclo, dove le istruzioni comprese fra **do** e **end** vengono ripetute fintantoché risulta vera la condizione espressa fra **while** e **do**, che nel nostro caso è "orbita non è chiusa". Alla prima ripetizione in cui l'orbita risulta chiusa il ciclo si interrompe.

150 CAPITOLO 11. GIRANDO IN TONDO: DAL CERCHIO ALL'ORBITA DI HALLEY

Messo così non siamo molto lontani dal codice che esprimeva il cerchio alla Papert:

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

che in sostanza diceva²

```
while do
  | disegna prossimo punto;
end
```

Algorithm 2: Pseudocodice per il calcolo della circonferenza

Cos'hanno in comune di importante il codice per disegnare l'orbita di un corpo celeste e quello per disegnare il cerchio? Il fatto di farlo esclusivamente in base a valutazioni locali, senza impiegare la conoscenza esplicita dell'ellisse in un caso e del cerchio nell'altro, ovvero senza utilizzare le equazioni delle rispettive curve. In termini matematici, questa "valutazione locale" non vuol dire altro che risolvere l'equazione differenziale che descrive un certo fenomeno. Torneremo dopo su questo, ora era necessario solo cercare di cogliere questa corrispondenza matematica, prima di tuffarsi un poco nei particolari del codice per l'orbita di un corpo reale, che per la sua particolare natura richiede una serie di accorgimenti, i quali spostano considerevolmente in alto l'*high ceiling*. Forse anche troppo, in media, per una scuola secondaria, ma non è affatto escluso, che in una situazione laboratoriale, un insegnante e degli studenti in gamba vi si possano arrampicare. L'esempio è comunque utile per mettere in luce vari aspetti che mostrano come l'approccio computazionale consenta di avvicinarsi in maniera molto più concreta allo studio dei fenomeni fisici, sotto vari aspetti, e anche come il calcolo scientifico sia ben lontano da quel banale determinismo che taluni, intrappolati in una miope visione dicotomica fra *humanities* e scienza, vorrebbero far credere. Tant'è che l'espressione "*art of scientific computing*" è comune e pertinente. In ultimo, l'esempio è istruttivo anche per mostrare come non siano necessari strumenti sofisticati per realizzare compiti che possono essere anche considerevolmente complessi, dal punto di vista scientifico. Insomma Logo non è limitato al disegno delle casette. E per questo che abbiamo aggiunto, ad esempio, la scrittura su di un file dei dati calcolati nel processo – posizioni velocità e accelerazioni – per mostrare che LibreLogo può anche essere considerato un sistema per eseguire calcolo, vero e proprio, e non solo per fare un po' di grafica.

Nelle pagine successive riportiamo il listato del codice, ampiamente commentato, per il caso della cometa di Halley che orbita intorno al sole. Aggiungiamo poi qualche considerazione per approfondire alcuni aspetti più rilevanti. La sor-

²Qui abbiamo usato la variante "**while do** istruzioni **end**" senza condizione, che produce una ripetizione del ciclo all'infinito.

gente del codice, in formato ODT, la mettiamo a disposizione in un file a parte³, in maniera che possa essere eseguito da tutti.

```

1 ; Programma distribuito alle condizioni della GNU General Public License
2
3 ; Questo programma è Software Libero (Free Software): può essere
4 ; ridistribuito e modificato nei termini della GNU General Public
5 ; License pubblicata dalla Free Software Foundation, nella versione 3 o
6 ; una delle successive. Il testo della licenza è accessibile in
7 ; <https://www.gnu.org/licenses/licenses.it.html>.
8
9
10 ; Calcolo dell'orbita di un corpo celeste intorno al sole mediante
11 ; integrazione numerica delle equazioni del moto governate dalla legge
12 ; di gravitazione di Newton. Il problema è posto in due dimensioni e
13 ; assume che non vi siano perturbazioni da parte di altri corpi.
14 ; Il codice è aggiustato per risolvere il caso di un'orbita fortemente
15 ; eccentrica come quella della cometa di Halley.
16
17 ; La Tartaruga gioca il ruolo della cometa Il pianeta sta al centro,
18 ; che in LibreLogo ha coordinate [297.89,421.11] dove la unità di
19 ; misura è il "punto" (p). Lo spazio della pagina è
20 ; (pagg. 63–65 del Piccolo Manuale di LibreLogo)
21
22 ;      [0,  0] ----- [596,  0]
23 ;      |           |
24 ;      |           |
25 ;      |           |
26 ;      |           |
27 ;      |           |
28 ;      [0, 842] ----- [596, 842]
29
30 ; dove i numeri sono espressi nell'unità di misura del foglio, qui
31 ; denominata "punto". In Writer di LibreOffice il punto vale 1/2.83 mm
32 ; (vedi nota 54 a pagina 64). Nei calcoli di questo algoritmo tutte
33 ; le misure di lunghezza sono riportate in "punti", in ultima analisi.
34
35 ; Dichiarazione delle variabili globali, ovvero che sono "visibili"
36 ; sia nel codice principale che all'interno di ogni procedura, NEWTON,
37 ; STEP ecc.
38
39
40 GLOBAL GG, Dt, DX, DY, XPOS0, YPOS0, XPOS, YPOS, XVEL, YVEL, XACC, YACC
41
42 ; Qui seguono le procedure nelle quali abbiamo incapsulato alcune
43 ; funzionalità specifiche: NEWTON che calcola l'accelerazione in un
44 ; dato punto, STEP che valuta il prossimo passo del percorso,
45 ; WRITEPOINT scrive gli estremi dei punti via via calcolati su un
46 ; file...
47
48 ; ****
49 ; procedura NEWTON: calcola l'accelerazione nel punto di coordinate
50 ; X, Y – il risultato consiste nelle due componenti ACCX e ACCY
51 ; del'accelerazione. È qui che "c'è" la legge di gravitazione.
52
53 TO NEWTON X Y
54   GLOBAL GG, Dt, DX, DY, XPOS0, YPOS0, XPOS, YPOS, XVEL, YVEL, ~
55   XACC, YACC
56   DX = (X-XPOS0)
57   DY = (Y-YPOS0)
58   R2 = (DX**2 + DY**2)
59   R = SQRT(R2)
60   XACC = - GG / R2 * DX / R
61   YACC = - GG / R2 * DY / R
62 END
63
64 ; ****

```

³Il codice è accessibile presso l'URL <http://iamarf.ch/unifi/Halley-RK-4-AU-90-distribuibile.odt>

152CAPITOLO 11. GIRANDO IN TONDO: DAL CERCHIO ALL'ORBITA DI HALLEY

```

65 ; procedura STEP: calcola il prossimo passo con l'interpolazione di
66 ; Runge-Kutta del IV ordine. Questa interpolazione rappresenta un
67 ; modo abbastanza sofisticato per calcolare i vari punti della
68 ; traiettoria; serve a ridurre gli errori di approssimazione dovuti al
69 ; fatto di calcolare una funzione continua in un numero finito di
70 ; punti. L'algoritmo è tratto da W.H. Press et al., Numerical Recipes
71 ; - The Art of Scientific Computing, Cambridge university Press, 1992,
72 ; pp. 704-708. Il risultato consiste nelle coordinate della nuova
73 ; posizione, XPOS e YPOS, e della velocità in quel punto, XVEL e YVEL.
74
75 TO STEP
76     GLOBAL GG, Dt, DX, DY, XPOS0, YPOS0, XPOS, YPOS, XVEL, YVEL, ~
77         XACC, YACC
78
79     NEWTON XPOS YPOS
80     KX1 = Dt * XACC
81     XVEL1 = XVEL +KX1 / 2.
82     KY1 = Dt * YACC
83     YVEL1 = YVEL +KY1 / 2.
84     XPOST = XPOS +XVEL1 * Dt / 2.
85     YPOST = YPOS +YVEL1 * Dt / 2.
86
87     NEWTON XPOST YPOST
88     KX2 = Dt * XACC
89     XVEL2 = XVEL +KX2
90     KY2 = Dt * YACC
91     YVEL2= YVEL +KY2
92     XPOST= XPOS +XVEL2 * Dt / 2.
93     YPOST= YPOS +YVEL2 * Dt / 2.
94
95     NEWTON XPOST YPOST
96     KX3 = Dt * XACC
97     XVEL3 = XVEL +KX3
98     KY3 = Dt * YACC
99     YVEL3= YVEL +KY3
100    XPOST= XPOS +XVEL3 * Dt / 2.
101    YPOST= YPOS +YVEL3 * Dt / 2.
102
103    NEWTON XPOST YPOST
104    KX4 = Dt * XACC
105    XVEL4 = XVEL + KX4
106    KY4 = Dt * YACC
107    YVEL4 = YVEL +KY4
108
109    XVEL= XVEL + (KX1 + 2 * KX2 + 2 * KX3 + KX4) / 6.
110    YVEL= YVEL + (KY1 + 2 * KY2 + 2 * KY3 + KY4) / 6.
111
112    XPOS= XPOS + (XVEL1 + 2 * XVEL2 + 2 * XVEL3 + XVEL4) * Dt / 6.
113    YPOS= YPOS + (YVEL1 + 2 * YVEL2 + 2 * YVEL3 + YVEL4) * Dt / 6.
114
115 END
116
117 ; *****
118 ; procedura WRITEPOINT. Fa due cose:
119
120 ; 1) invia la tartaruga nel prossimo punto della traiettoria mediante
121 ; un'istruzione POSITION - è così che si crea il disegno
122
123 ; 2) scrive in un file tutti i valori rilevanti di ciascuno punto della
124 ; traiettoria calcolato, in particolare le due componenti della
125 ; posizione, della velocità e dell'accelerazione, ovvero tutto
126 ; ciò che viene detto soluzione dell'equazione del moto. Questi
127 ; dati possono così essere ripresi per creare rappresentazioni
128 ; grafiche con altri tipi di software o per ulteriori elaborazioni.
129
130 TO WRITEPOINT f REPCOUNT
131     GLOBAL GG, Dt, DX, DY, XPOS0, YPOS0, XPOS, YPOS, XVEL, YVEL, ~
132         XACC, YACC
133
134     POSITION [XPOS, YPOS]
135     f.write( repr(REPCOUNT) + ',' + repr(GG) + ',' + repr(Dt) + ',' ~
136         + repr(DX) + ',' + repr(DY) + ',' + repr(XPOS0) ~
137         + ',' + repr(YPOS0) + ',' + repr(XPOS) + ',' + repr(YPOS) ~
138         + ',' + repr(XVEL) + ',' + repr(YVEL) + ',' + repr(XACC) ~

```

```

139      + ', ' + repr(YACC) + '\n')
140 END
141
142 ; ****
143 ; ****
144 ; Qui inizia il programma vero e proprio , quello che abbiamo chiamato
145 ; il codice principale
146
147 ; Prima di tutto si calcolano le costanti fisiche che entrano in gioco
148 ; nella determinazione del moto. I valori sono calcolati nel sistema
149 ; di misura M.K.S (Meter , Kilogram , Second), con l'eccezione delle
150 ; distanze delle orbite perché qui, dati gli enormi valori è più
151 ; usuale l'AU (Astronomical Unit), dove 1 AU =  $1.495978707 \times 10^{11}$ 
152 ; metri, che corrisponde alla distanza media fra la terra e il sole.
153 ; Quindi, ad esempio, quando si trova che l'afelio della cometa di
154 ; Halley vale 35.08 AU, significa che la distanza massima della cometa
155 ; dal sole è pari a circa 35 volte la distanza fra la terra e il sole.
156 ; Alla fine tuttavia , tutte le misure di distanza sono trasformate
157 ; in "punti" della pagina ai fini della rappresentazione grafica .
158
159 G = 6.67E-11           ; (N*m^2/Kg^2) Costante di gravitazione
160 Ms = 1.99E30          ; (Kg) Massa del sole
161
162 Dp = 200.0            ; Afelio espresso in punti del foglio
163 rAf = 35.08            ; Afelio (AU)
164 Dt = 0.001             ; Intervallo di integrazione (ha la
165 ; dimensione del tempo)
166
167 K = Dp/rAf            ; fattore di scala: numero di punti/AU
168 GAU = G / 1.496E11**2
169 Gp = GAU * K**2        ; (N*p^2*Kg^2)
170 GG = Gp * Ms           ; Cost. inclusiva della massa solare (per
171 ; ridurre il numero di moltiplicazioni nei
172 ; cicli di integrazione)
173 eps = 0.967            ; Eccentricità dell'orbita Halley
174
175 ; Qui si apre il file dove verranno scritti gli estremi dei punti della
176 ; traiettoria via via calcolata. Si scrive subito anche la prima riga di
177 ; intestazione della tabella.
178
179 f = open('/home/arf/Didattica//CODING/Logo/orbite/halley-dt-0.001-20', ~
180      'w')
181 f.write( 'REPCOUNT, GG, Dt, DX, DY, XPOS0, YPOS0, ~
182           XPOS, YPOS, XVEL, YVEL, XACC, YACC, \n')
183
184 CLEARSEREN
185 HOME
186
187 FILLCOLOR "skyblue"
188 PENCOLOR "blue"
189 CIRCLE 400              ; Si disegna un cerchio di raggio 200 punti
190 ; nel quale verrà disegnata la traiettoria
191 ; del corpo celeste; qualora si usi un
192 ; valore di eccentricità dell'orbita pari
193 ; a 0 la traiettoria risulterà
194 ; perfettamente circolare e coinciderà con
195 ; il perimetro di questo cerchio.
196
197 FILLCOLOR "yellow"       ; colore del sole
198 PENCOLOR "yellow"
199 CIRCLE 5                 ; Date le dimensioni dell'orbita della
200 ; cometa il sole non può essere in scala
201
202 ; Qui memorizziamo le coordinate del centro della pagina, che
203 ; assumiamo come origine del sistema di riferimento e che
204 ; facciamo coincidere con la posizione del sole.
205
206 XPOS0 = POSITION[0] ; coordinate origine (centro pagina)
207 YPOS0 = POSITION[1]
208
209 HIDETURTLE               ; nasconde la Tartaruga perché assorbe
210 ; troppe risorse
211
212 ; Determinazione delle condizioni iniziali del problema

```

154CAPITOLO 11. GIRANDO IN TONDO: DAL CERCHIO ALL'ORBITA DI HALLEY

```

213 ; Posizione iniziale dove portiamo il satellite in quota
214
215 PENUM
216 POSITION [XPOS0 + rAf*K, YPOS0]
217 PENDOWN
218 XPOS = POSITION[0]
219 YPOS = POSITION[1]
220
221 ; Velocità iniziale che imprimiamo al satellite
222
223 XVEL = 0.0
224 YVEL = sqrt(GG/(rAf*K)*(1-eps)) ; Grazie alla II legge di Keplero la
225 ; velocità, ad esempio all'afelio,
226 ; risulta determinata dall'eccentricità
227 ; dell'orbita oltre che dalla
228 ; massa del sole
229
230 PENCOLOR "blue"
231 PENSIZE 1
232
233 ; Ora inizia il ciclo che disegna la traiettoria del corpo celeste.
234 ; I punti della traiettoria sono calcolati con la subroutine STEP e
235 ; vengono disegnati con la procedura WRITEPOINT. Quest'ultima
236 ; provvede anche a scaricare gli estremi di ciascuno punto
237 ; (posizione, velocità, accelerazione) in un file. Tuttavia
238 ; WRITEPOINT viene invocata solo in un numero limitato di punti,
239 ; perché la traiettoria, affinché possa essere calcolata in
240 ; maniera sufficientemente accurata, risulta composta da un
241 ; numero esagerato di punti, ai fini del disegno. In questa
242 ; implementazione, aggiustata per riprodurre orbite fortemente
243 ; eccentriche, come quella della cometa di Halley, il codice
244 ; usa solo un punto su 10000 per disegnare traiettoria, e questo
245 ; comportamento viene controllato mediante il contatore nWrite.
246 ; Si usa poi un flag (variabile di tipo sì/no), yIsNegative,
247 ; per controllare che l'orbita non venga disegnata più di una volta.
248
249 nWrite = 0
250 yIsNegative = FALSE ; Flag controllo completamento orbita:
251
252 WHILE NOT ( yIsNegative AND (YPOS-YPOS0) > 0 ) [
253   nWrite = nWrite + 1
254   IF NOT yIsNegative AND (YPOS-YPOS0) < 0 [ yIsNegative = TRUE ]
255   STEP
256   IF nWrite = 1 [
257     WRITEPOINT f, REPCOUNT
258   ]
259   IF nWrite = 10000 [ nWrite = 0 ]
260 ]
261
262 PRINT "Fatto!"

```

La prima cosa che si incontra è la licenza di software libero. L'abbiamo fatto più che altro per poterla "toccare con mano". Poi, la prima istruzione che troviamo è GLOBAL (n. 40). È buona pratica rendere il codice modulare, riducendolo a un insieme di blocchi indipendenti specializzati nell'esecuzione di singole operazioni. Questi blocchi devono poi essere in grado di comunicare fra loro mediante appositi meccanismi. Alla base di questi sta lo scambio di dati che servono a compiere le operazioni. Tali blocchi si chiamano procedure, subroutine, metodi, funzioni, in dipendenza di diversità fra linguaggi e contesti che qui non ci serve approfondire. Serve sapere comunque che se voglio far fare a una procedura (usiamo qui questo nome) ciò che sa fare occorre che le passi i dati necessari e che poi lei mi renda in qualche maniera i risultati dell'operazione. Il meccanismo normale è quello di aggiungere i nomi di tali

dati nell'istruzione che "chiama" la procedura. È quello che avevamo visto per esempio scrivendo in LibreLogo una procedura TO CASA L, dove L ad esempio potrebbe essere la misura del quadrato che forma la casa. Normalmente ciascuna procedura ha un suo spazio privato per le variabili che usa, nel senso che quando questa ha finito il suo compito le variabili per così dire vengono perse. È per questo motivo che è necessario "passarle" nella chiamata. In certi contesti è tuttavia possibile anche definire alcune variabili "globali", che significa visibili da ogni parte del programma, anche da dentro le procedure. In tal caso non c'è bisogno che queste vengano "passate". Ogni procedura ci lavora e i cambiamenti restano a disposizione di altre parti del programma. Questa non è considerata una buona pratica, specialmente quando i progetti divengono complessi, ma nel caso di codici relativamente compatti questo tipo di variabile può tornare comodo. È il nostro caso, dove abbiamo resi globali la costante di gravitazione universale inclusiva della massa del sole (GG), il passo di integrazione, ovvero quanto finemente si vogliono calcolare i successivi punti (Dt), le coordinate espresse rispetto all'origine, ovvero il centro della pagina (DX, DY), le coordinate dell'origine (XPOS0, YPOS0), le coordinate del punto corrente della traiettoria (XPOS, YPOS), le componenti della velocità (XVEL, YVEL) e quelle dell'accelerazione (XACC, YACC).

Poi vengono i codici delle varie procedure, iniziando da quella che non ne chiama nessun'altra, NEWTON per il calcolo dell'accelerazione, STEP per il calcolo dei passi della traiettoria, WRITEPOINT per il disegno e la memorizzazione dei punti su file. I commenti nel listato dovrebbero essere sufficienti, per comprendere il comportamento, eccetto che nel caso della procedura STEP dove occorre qualche precisazione. In realtà l'operazione compiuta da STEP potrebbe ridursi all'applicazione delle istruzioni 11.10 e 11.11 , che riscriviamo qui di seguito

$$\begin{aligned} XVEL &= XVEL + XACC * Dt \\ YVEL &= YVEL + YACC * Dt \end{aligned} \quad (11.14)$$

e

$$\begin{aligned} XPOS &= XPOS + XVEL * Dt \\ YPOS &= YPOS + YVEL * Dt \end{aligned} \quad (11.15)$$

Immaginando un intervento didattico, è utile partire proprio da questa versione perché consente di capire facilmente il meccanismo con cui ogni valore viene aggiornato in base al valore del precedente e in funzione di una cosa che va calcolata – l'accelerazione in un caso, la velocità nell'altro - giusto in quel punto. In questa versione si capisce bene anche l'essenza del concetto di approssimazione, perché diviene palese la seguente ambiguità: il valore ad esempio dell'accelerazione a quale punto si riferisce? Al precedente o al successivo? La risposta è che il passo fra l'uno e l'altro deve essere abbastanza breve da rendere inessenziale la differenza. Di fatto si tratta di un'approssimazione e non è affatto banale stabilire una regola valida per tutte le situazioni. Ecco dove compare l'arte nel calcolo scientifico, o, se vogliamo, dove necessita la mano dell'artigiano, che comprende in base all'esperienza e all'intuito come regolarsi. Le operazioni descritte sopra prendono il nome di interpolazione di Eulero, che concettualmente è perfettamente valida, ma all'atto dell'applicazione concreta, molto facilmente produce soluzioni distorte perché troppo sensibile agli errori di approssimazione.

Per capire bene il meccanismo immaginiamo di far fare ai bambini un cerchio nella sabbia come avevamo visto al *low floor* ma schiacciato, dicendo loro di curvare quando meno quando più, in maniera da riprodurre qualcosa di simile a un'ellisse. E proviamo a esagerare, che tanto ai bambini piace, facendo un cerchio molto schiacciato. È facile rendersi conto che se lungo i tratti per così dire piatti possiamo fare anche dei passi piuttosto lunghi, quando invece arriviamo in prossimità delle curve strette dobbiamo per forza accorciare il passo, per poter seguire l'andamento della traiettoria. L'interpolazione di Runge-Kutta del IV ordine, realizzata attraverso la procedura STEP, migliora la scelta della direzione del passo facendo come dei sondaggi aggiuntivi rispetto al semplice calcolo unico dell'interpolazione di Eulero, espresso dalle relazioni 11.14 e 11.15, tre in più, per la precisione. Noi l'abbiamo introdotta perché sennò l'orbita della cometa di Halley non viene calcolata correttamente, essendo molto eccentrica, quindi con curvatura molto accentuata in prossimità dell'afelio e del perielio. Ma per capire l'architettura generale possiamo benissimo fare riferimento all'approssimazione di Eulero.

A partire dall'istruzione 144 inizia il programma principale, dove, come prima cosa, si definiscono le costanti fisiche e geometriche in ballo nel problema. Sono tutte spiegate nel listato. Ovviamente si può giocare variandole a piacimento per riflettere sugli effetti. La prima cosa da fare è lavorare sull'eccentricità dell'orbita, attualmente fissato sul valore della cometa di Halley, pari a 0.967. Prossimo al valore limite di 1, dove l'ellisse si trasforma in una parabola e oltre 1 diventa un'iperbole. Per valori di eccentricità decrescenti l'ellisse diventa sempre meno schiacciata fino a 0 che vale per il cerchio.

Successivamente vengono determinate le condizioni iniziali, che è un passo fondamentale per la risoluzione di qualsiasi problema di fisica. Se gli allievi uscissero dalle scuole superiori con questo concetto ben chiaro in testa sarebbe già un gran risultato. L'impostazione computazionale dei problemi di fisica consentirebbe di realizzare questo obiettivo. Nel caso della soluzione di un problema di moto le condizioni iniziali si determinano attraverso la posizione e la velocità iniziali, in valore (modulo) e direzione. Nel nostro codice posizione e dimensioni dell'orbita sono aggiustate in maniera da accomodarsi adeguatamente nella pagina. L'eccentricità invece viene determinata attraverso la II legge di Keplero che lega questa alla velocità del corpo celeste nella sua orbita. In virtù di tale legge, la velocità di un corpo celeste è legata alla distanza r dal sole mediante questa relazione

$$v = \sqrt{GM\left(\frac{2}{r} - \frac{1}{a}\right)} \quad (11.16)$$

dove G la costante di gravitazione universale, M è la massa del sole e a è il semiasse maggiore dell'orbita ellittica. Per stabilire la nostra condizione iniziale assumiamo che l'orbita ellittica debba stare sdraiata con l'asse maggiore orizzontale, con il fuoco sinistro coincidente con il centro della pagina e che la cometa "parta" dall'afelio, ovvero nel punto di massima distanza, che così ci ritroviamo a destra, e sia diretta verso il basso. La posizione è determinata mediante le istruzioni 206 e 207. Per quanto riguarda la velocità usiamo il fatto che, quando la distanza dal sole coincide con l'afelio, $r = r_a$, vale la seguente relazione fra la distanza all'afelio r_a , il semiasse maggiore a e l'eccentricità ϵ :

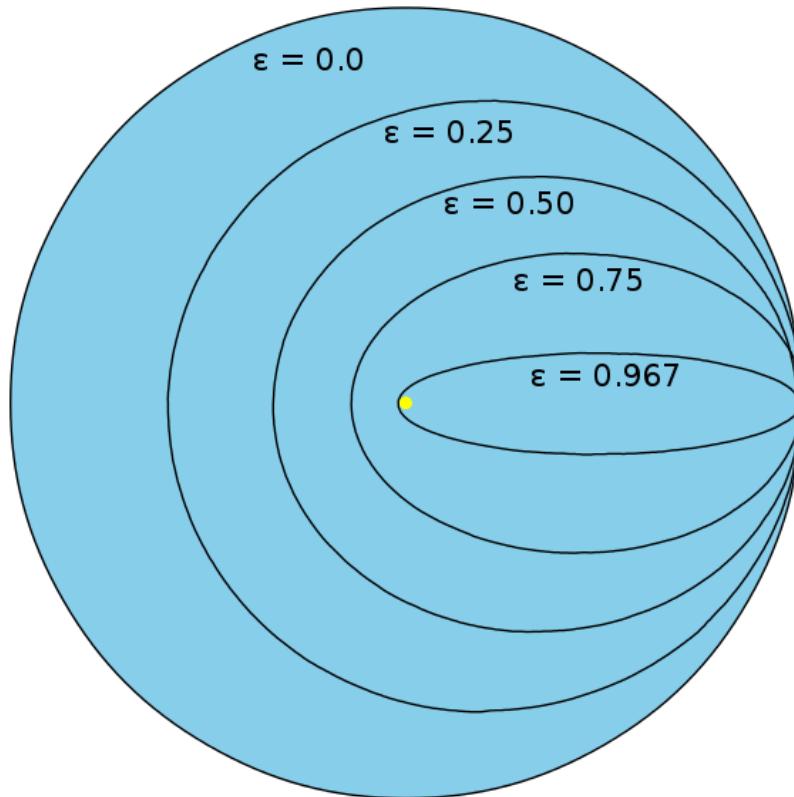


Figura 11.2: Orbite di un corpo celeste intorno al sole con diversi valori di eccentricità ϵ . Il valore $\epsilon = 0.967$ corrisponde a quello della cometa di Halley..

$$r_a = a(1 + \epsilon) \quad (11.17)$$

Questa, combinata con l'equazione 11.16 fornisce

$$v = \sqrt{\frac{GM}{r_a}(1 - \epsilon)} \quad (11.18)$$

che è esattamente l'equazione codificata nell'istruzione 225 del listato, per la componente lungo y della velocità, essendo la componente lungo x pari a 0 perché appunto assumiamo che la cometa in quel punto sia diretta esattamente in basso.

Si comprende qui come, per variare la forma dell'orbita, si possa agire sulla velocità: velocità iniziali maggiori comportano valori di ϵ inferiori, i quali a loro volta descrivono orbite sempre più circolari. Qui abbiamo dunque la possibilità apprezzare come si possa riflettere sul significato di una formula manipolando il codice software che la esprime, in una maniera che assai difficilmente può verificarsi attraverso la semplice memorizzazione della formula.

Infine, fra l'istruzione 252 e 260 si svolge il ciclo sui punti della traiettoria, dove vengono invocate successivamente le procedure STEP e WRITEPOINT, la prima che calcola il prossimo passo e la seconda che disegna e memorizza i punti calcolati. La variabile yIsNegative, usata come *flag*, ovvero come semaforo sì-no, serve a controllare che l'orbita non venga ripetuta una seconda volta. Invece la variabile nWrite, usata come contatore, serve a far sì che i punti vengano disegnati e memorizzati sempre ma solo uno ogni tanto, precisamente uno ogni 10000 in questa versione! Ciò è dovuto al fatto che affinché l'orbita venga disegnata con ragionevole precisione, questa venga calcolata con un campionamento molto fitto: in questa implementazione la Tartaruga calcola più di 14 milioni di punti! Non avrebbe senso farle scrivere tutti questi valori e il sistema grafico andrebbe nel pallone perché non possiede questa precisione. Pur tenendo traccia di un punto su diecimila, il numero di punti totale che vengono effettivamente rappresentati e memorizzati è di 1466. Noi ci siamo fermati qui. In realtà sarebbe possibile aggiungere un altro perfezionamento. Ora il campionamento è svolto in modo regolare, con un passo Dt pari a 0.001, che produce oltre 14 milioni di punti, come abbiamo visto. Il risultato ai nostri fini è soddisfacente: l'orbita viene abbastanza regolare e il calcolo si svolge in pochi minuti, anche su un computer moderatamente veloce. Ma non è detto che non si presentino problemi provando altre configurazioni. In tal caso, potremmo provare a rendere l'interpolazione adattiva, aggiustando il passo Dt in funzione della curvatura della traiettoria: dove questa è più rettilinea la Tartaruga potrebbe procedere spedita, facendo passi più lunghi, mentre dove è più curva i passi potrebbero stringersi progressivamente.

E a proposito di altre configurazioni, lo spazio per le esplorazioni è sconfinato. Ad esempio, perché non cogliere l'occasione di conoscere Ardusat, iniziando a simularne l'orbita, anche per farsi un'idea di come "volino" i satelliti intorno alla terra – una cosa ben diversa dal caso della cometa! Ardusat⁴ è un cosiddetto "nanosatellite", un cubo di 10 cm di lato che pesa un chilo – un cartone di latte più o meno – equipaggiato con una particolare versione di Arduino, basata su 16 microprocessori e uno di controllo più una serie di 25 sensori diversi, fra cui una videocamera uno spettrometro ottico, un contatore Geiger, un sensore a infrarossi ecc. È concepito per essere usato da studenti, insegnanti o appassionati che possono progettare esperimenti, applicazioni che impiegano dati spaziali o addirittura giochi. Gli utenti possono progettare il software delle loro applicazioni sulle normali schede Arduino per poi sperimentarlo su un clone terrestre di Ardusat. Quando tutto va bene il software viene inviato a Ardusat per eseguire le operazioni previste, scaricando i dati sul browser degli utenti, quando il satellite è visibile dalle apposite stazioni terrestri. Al termine del periodo di lavoro, che può durare fino a una settimana, tutti dati vengono scaricati e inviati all'utente. In un programma di scienze, potrebbe essere estremamente stimolante esplorare Ardusat, partendo per esempio dalla simulazione della sua orbita con Logo.

Tutte considerazioni che possono essere estremamente istruttive e che possono essere svolte per dare una corretta valutazione di cosa sia oggi la scienza, in particolare di quanto e come gli aspetti computazionali influiscano su di essa. In realtà la riflessione è possibile di un ulteriore e significativo approfondimen-

⁴Informazioni dettagliate si trovano ad esempio http://www.dk3wn.info/sat/afu/sat_ardusat.shtml e <https://en.wikipedia.org/wiki/ArduSat>.

to, secondo la linea di pensiero illustrata da B. Sherin in un articolo intitolato *"A comparison of programming languages and algebraic notation as expressive languages for physics"* [11].

Sherin nel suo articolo affianca – si badi bene, non contrappone – la *programming-physics* all'*algebra-physics*. L'impianto teorico del suo lavoro si fonda sul ruolo giocato dalle rappresentazioni strumentali e simboliche che supportano la conoscenza nella formazione stessa di tale conoscenza – concetti sviluppati da altri autori, ad esempio da Weintrop e Wilensky [9]. L'autore documenta i propri argomenti con alcuni esercizi di fisica presentati sia nella forma algebrica convenzionale che nella forma computazionale. Per quest'ultima riferisce di esperienze svolte con l'ambiente Boxer, sviluppato da Di Sessa e altri autori [6], sulla base di Logo. Con questo lavoro l'autore sostiene la tesi che sistemi di rappresentazione diversi influiscono in modo differente sui meccanismi del pensiero e possono indurre un diverso tipo di comprensione dei medesimi fenomeni. In capo a un'analisi minuziosa di esperimenti didattici, condotti su gruppi di studenti sia mediante l'*algebra-physics* che la *programming-physics*, giunge alla conclusione per cui con la conoscenza algebrica si tende a enfatizzare gli equilibri mentre con quella computazionale si è portati a comprendere meglio gli aspetti dinamici. È estremamente interessante la prospettiva nella quale Sherin pone questa conclusione. Non si tratta, dice, di giudicare l'effetto di un metodo o dell'altro secondo una singola metrica e di confrontarli sulla base di tale metrica – ovvero non si tratta di stabilire quale sia "meglio" - bensì di accettare, comprendere e utilizzare proficuamente il fatto che il nuovo paradigma offre una mutata visione della conoscenza degli stessi fenomeni e di come, in ultima analisi, la cosa più sensata da fare sia quella di affiancare questa nuova forma di conoscenza a quelle preesistenti. E non si può evitare di osservare che la nuova prospettiva computazionale – qui nel senso della *programming-physics* di Sherin, possa essere di grande giovamento per la comprensione dei fenomeni fisici. Infatti lo strumento matematico costituisce indubbiamente il fondamento imprescindibile delle scienze di base – il linguaggio che consente di porre domande alla natura, per dirla con Galileo - ma il processo con il quale un giovane giunge a creare senso compiuto a partire da un linguaggio formale è molto complesso e faticoso. Pochi studenti arrivano ad apprezzare il formalismo matematico come uno strumento utile per comprendere e esprimere pensieri sul mondo fisico o altro. Per la grande maggioranza i formalismi matematici rappresentano al più una quantità di regole da applicare a memoria negli specifici contesti creati dalla scuola: qual era la formula da usare qui...? A questo proposito è interessante ricordare un noto articolo scritto da Enrico Persico [10], maestro di Enrico Fermi, dove ci si domandava cosa non andasse con quella studentessa che procedeva come una locomotiva quando sciorinava le equazioni di Maxwell alla lavagna ma che non sapeva dire perché, con quel certo valore di corrente, una lampadina si sarebbe fulminata – non a caso Sherin rileva esattamente lo stesso problema a pag. 43 del suo lavoro, e proprio a proposito delle equazioni di Maxwell. La questione della comprensione dei fenomeni attraverso il linguaggio matematico non è, e non da ora, semplice. È esattamente qui che il "nuovo" approccio computazionale, nel quale peraltro vengono declinati settori sempre più ampi della fisica e delle altre scienze, può venire in aiuto. Infatti, l'approccio computazionale induce ad analizzare e scomporre i fenomeni fisici nella dimensione temporale, enfatizzandone così la natura dinamica, spesso più accessibile all'intuizione. Non solo, l'analisi computazionale costringe ad utilizzare precisi valori numerici

160CAPITOLO 11. GIRANDO IN TONDO: DAL CERCHIO ALL'ORBITA DI HALLEY

da assegnare ai parametri fisici coinvolti e questa è una pratica che induce più facilmente gli studenti a ricavare un senso da ciò che studiano.

Ora che abbiamo tirato in ballo l'articolo di Sherin vale la pena di specificare che è proprio grazie a questo lavoro che siamo giunti dal "cerchio alla Papert" alle orbite dei corpi celesti. Sì perché all'inizio ci eravamo messi a riprodurre in Logo gli esercizi esemplificati da Sherin, nel modo più semplice possibile. Riportiamo la progressione qui di seguito.

```
1 ; Caduta di un grave con accelerazione di gravità costante
2
3 CLEARSCREEN
4 HOME
5
6 PENUP
7 FORWARD 350
8 RIGHT 180
9 PENDOWN
10
11 XPOS = 0.0
12 VEL = 0.0
13 ACC = 9.8
14 DVEL = ACC
15
16 REPEAT 10 [
17   VEL = VEL + DVEL
18   XPOS = XPOS + VEL
19   PENDOWN
20   CIRCLE 5
21   PENUP
22   FORWARD VEL
23 ]
24
25 HIDEURTLE
```

```
1 ; Caduta di un grave con accelerazione di gravità costante
2 ; in presenza di resistenza dell'aria
3
4 CLEARSCREEN
5 HOME
6
7 XPOS = 0.0
8 VEL = 0.0
9 ACC = 9.8
10 RES = 0.0
11
12 G = 9.8
13 M = 10.0
14 K = 2.0
15
16 PENUP
17 FORWARD 350
18 RIGHT 180
19 PENDOWN
20
21 REPEAT 20 [
22   RES = VEL * K
23   ACC = G - RES / M
24   VEL = VEL + ACC
25   XPOS = XPOS + VEL
26   PENDOWN
27   CIRCLE 5
28   PENUP
29   FORWARD VEL
```

```

30      ]
31
32 HIDETURTLE

```

```

1 ; Andamento nel tempo del moto di un grave appeso a una molla
2 ; (asse x: tempo)
3
4 CLEARSEREN
5 HOME
6
7 XPOS = 0.0
8 VEL = 0.0
9 ACC = 9.8
10 RES = 0.0
11
12 G = 9.8
13 M = 10.0
14 KA = 0.5
15 K = 1.0
16
17 PENUP
18 FORWARD 250
19 RIGHT 180
20 PENDOWN
21
22 REPEAT [
23   RES = KA * VEL
24   SPR = K * XPOS
25   ACC = G - RES / M - SPR / M
26   VEL = VEL + ACC
27   XPOS = XPOS + VEL
28   PENDOWN
29   CIRCLE 5
30   PENUP
31   FORWARD VEL
32   LEFT 90 FORWARD 3 RIGHT 90
33   ]
34
35 HIDETURTLE

```

```

1 ; Caduta di un grave dallo spazio
2
3 CLEARSEREN
4 HOME
5
6 FILLCOLOR "BLUE"
7 CIRCLE 70
8 FILLCOLOR "LIME"
9 XPOS0 = POSITION[0]
10 YPOS0 = POSITION[1]
11
12 PENUP
13 FORWARD 350
14 RIGHT 180
15 PENDOWN
16 HIDETURTLE
17
18 XPOS = POSITION[0]
19 YPOS = POSITION[1]
20 VEL = 0.0
21 KG = 9800
22
23 REPEAT [
24   DY = YPOS-YPOS0
25   VEL = VEL + KG / DY**2

```

162 CAPITOLO 11. GIRANDO IN TONDO: DAL CERCHIO ALL'ORBITA DI HALLEY

```

26   YPOS = YPOS + VEL
27   PENDOWN
28   CIRCLE 5
29   IF YPOS0-YPOS < 10 [ BREAK ]
30   PENUP
31   POSITION [XPOS,YPOS]
32   ;FORWARD (VEL)
33   ]
34   FILLCOLOR "RED"
35   CIRCLE 10
36   HEADING 0
37   FORWARD 5
38   LABEL "
39
40 HIDETURTLE

```

AAAAAAAHH...!!!!"

```

1 ; Legge gravitazionale di Newton in 2D
2 ; È un esempio in germe. Per una corretta e proficua realizzazione
3 ; didattica occorre entrare nel merito dei valori delle costanti
4 ; fisiche implicate, masse, distanze, costante gravitazionale,
5 ; fattori di scala per la rappresentazione grafica. In questo
6 ; esempio ho utilizzato valori numerici arbitrari. Interessava
7 ; vedere come può funzionare...
8
9 ; La Tartaruga gioca il ruolo del satellite
10 ; Il pianeta sta al centro, che in LibreLogo ha coordinate
11 ; [ 297.89, 421.11 ]
12
13 CLEARSCREEN
14 HOME
15
16 FILLCOLOR "BLUE"
17 CIRCLE 20
18 FILLCOLOR "LIME"
19 XPOS0 = POSITION [0]
20 YPOS0 = POSITION [1]
21 SHOWTURTLE
22
23 ; Determinazione delle condizioni iniziali
24
25 ; Posizione iniziale...
26 ; con queste istruzioni la Tartaruga-satellite si dirige nella
27 ; posizione iniziale lo fa alla Papert, con comandi "sintonici":
28 ; sale di 100 e poi si mette a testa in giù (fatto inessenziale
29 ; perché poi la dirigiamo con comandi diretti tramite POSITION)
30
31 PENUP
32 HEADING 9h
33 FORWARD 0 ; qui si potrebbe spostare lateralmente
34 HEADING 0
35 FORWARD 100 ; qui sale di quota
36 RIGHT 180
37 PENDOWN
38
39 XPOS = POSITION [0]
40 YPOS = POSITION [1]
41
42 ; Velocità iniziale...
43 ; dove aggiustando componenti lungo X e Y decidiamo in che
44 ; direzione lanciare la Tartaruga-satellite (molto divertente
45 ; giocare con questo...)
46
47 ; Con XVEL= 10.0 e YVEL=0.0 si azzecca un'orbita circolare
48 ; Con XVEL= 5.0 e YVEL=0.0 viene l'orbita di tipo caotico mostrata
49 ; nel post https://iamarf.org/2017/04/19/dove-la-tartaruga-di-seymour-papert-impara-la-legge-di-gravitazionale-di-newton/
50
51 XVEL = 10.0
52 YVEL = 0.0

```

```

55 KG = 9800      ; qui dentro ci sarebbero le masse e la costante
56 ; gravitazionale...
58
59 REPEAT 50 [
60   DX = (XPOS-XPOS0)
61   DY = (YPOS-YPOS0)
62   R2 = (DX**2 + DY**2)
63   R = SQRT(R2)
64   ACC = KG / R2
65   XVEL = XVEL - ACC * DX / R
66   YVEL = YVEL - ACC * DY / R
67   XPOS = XPOS + XVEL
68   YPOS = YPOS + YVEL
69
70 PENDOWN
71 CIRCLE 5
72 IF SQRT((XPOS0-XPOS)**2 + (YPOS0-YPOS)**2) < 10 [ BREAK ]
73 PENUP
74 POSITION [XPOS, YPOS]
75 ]
76 FILLCOLOR "RED"
77 CIRCLE 10
78
79 HEADING 0
80 FORWARD 5
81 LABEL "          AHI!!!"
82
83 HIDETURTLE

```

Sono codici grezzi, con quasi nessun commento, eccetto l'ultimo, e senza un'accurata identificazione di ciascuna variabile e delle sue unità di misura. Li mettiamo a disposizione affinché possano essere completati e variati dal lettore, un po' come i disegni da colorare per i bambini...

11.3 Conclusione

In questo capitolo la dimensione verticale è ancora più estesa rispetto a quella del capitolo precedente, svolto intorno all'esplorazione di Marta. Qui siamo partiti proprio da un'esperienza che ha avuto luogo in una scuola primaria, dove la maestra Antonella ha coinvolto i suoi bambini in un'esplorazione del cerchio ispirata al pensiero di Papert. Le foto che Antonella ci ha regalato ci hanno come portati in classe. Ci siamo poi serviti delle parole di Papert stesso per commentare come in tale attività fisica si annidi un concetto matematico importante, una "idea potente", quella del calcolo differenziale. Un modo matematico per sviluppare una forma geometrica (in realtà anche molto più di questo) utilizzando solamente "informazioni locali". Ed è proprio Papert che in quel brano collega la questione a quella del calcolo dei movimenti dei corpi celesti, secondo la legge di gravitazione universale di Newton. Nella seconda parte del capitolo ci siamo quindi mossi per esplorare quanto alto possa essere il cielo di Logo, anche in una versione semplificata come quella di LibreLogo. Ed abbiamo preso così sul serio il proposito di andare direttamente nello spazio, provando a calcolare l'orbita di un corpo celeste e scegliendo quella della cometa di Halley, un po' per poterci meravigliare di andare così lontano, seppur nello spazio di un foglio di carta, e un po' per metterci alla prova, perché le orbite delle comete sono molto schiacciate, ovvero eccentriche, e quindi più insidiose da calcolare.

Nel fare tutto ciò abbiamo potuto esplorare aspetti computazionali e di calcolo scientifico niente affatto banali. Anzi, diciamo anche arditi, se li vogliamo pensare calati in una classe di scuola secondaria superiore. Effettivamente, un esercizio del genere non starebbe male nel corso di fisica generale al primo anno di università. Ma questo non significa che non vi possano essere ragazzi più giovani in grado di comprendere un percorso del genere, se adeguatamente seguiti dall'insegnante. Ho conosciuto un ragazzo al IV anno del liceo scientifico, molto vicino a me, al quale davo piccoli incarichi di calcolo scientifico che servivano nel mio lavoro di ricerca. Ebbene, questo ragazzo risolveva questi problemi prima e meglio dei miei giovani collaboratori, assegnisti e dottorandi. La scuola dovrebbe essere in grado di dare sostegno a tutti coloro che esulano dalle medie statistiche, superando il concetto "monodimensionale" e banale di performance, sul raggiungimento di determinate competenze o altro, ma occupandosi delle diversità e mettendole a frutto. Non è infatti raro che ingegni particolarmente creativi finiscano, in questa scuola ancora largamente tayloristica, fra i "meno bravi".

Infine, abbiamo commentato l'aspetto computazionale dell'esercizio rifacendoci al lavoro di Sherin [11], autore di un interessante approfondimento sulle differenze fra quella che chiama fisica algebrica, insegnata usualmente, e fisica computazionale. In sostanza, l'approccio computazionale ha il vantaggio di avvicinare maggiormente i ragazzi ai fenomeni, obbligandoli a seguirne gli aspetti dinamici, far capire l'importante concetto di condizione al contorno, che qualsiasi problema di fisica richiede di affrontare, rendersi maggiormente consapevoli della realtà delle grandezze fisiche in gioco, essendo obbligati a mettervi dei numeri. Abbiamo poi chiuso il capitolo riportando alcuni codici tratti dagli esempi presentati da Sherin nel suo lavoro, in modo che chiunque possa svilupparli e rifinirli, in maniera analoga a quanto abbiamo fatto con il calcolo dell'orbita della cometa di Halley.

Questo capitolo è anche una risposta ai "detrattori del pensiero computazionale", che duellano nei social network con gli altrettanto sgradevoli entusiasti del "codice facile". Le polemiche che si sviluppano in questi contesti si centrano sovente su una visione automatizzata delle pratiche di coding, come se queste si riducessero a banali sequenze di comandi, inducendo gli allievi all'esecuzione passiva di procedure automatizzate. Ora, io non sono in grado di affermare che le pratiche di coding possano essere applicate con risultati significativi anche in contesti umanistici - penso di sì ma non ho gli elementi per fare affermazioni certe - ma sicuramente, per qualsiasi giovane che sia interessato all'ambito delle discipline STEM, oggi il coding è uno strumento fondamentale. E lo è non solo dal punto di vista strumentale ma profondamente culturale, perché non vi è dominio scientifico che, da più di mezzo secolo a questa parte, non si sia ampiamente esteso grazie alla prospettiva computazionale. Nascondere questo agli studenti che andranno a iscriversi ai corsi di laurea scientifici è una grave omissione da parte della scuola.

Capitolo 12

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 lavori 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.

```
1 ; FIGURA 1 (torna alla figura)
2 ; Le coordinate della patoria alla figuragina
3 ; Versione 1.
4 ; A.R. Formiconi
5 ; 25 luglio 2016
6
7
8 ; Versione scritta come viene, senza particolari ottimizzazioni
```

```

9
10 ; Predisponde tutto
11
12 CLEARSEREN
13 HOME
14 A = 200 ; lato corto del rettangolo che
15 ; rappresenta la pagina
16 B = 282 ; lato lungo
17 FILLCOLOR [230, 230, 230] ; fissa a grigio chiaro il colore
18 ; di riempimento
19 PENUP
20 FORWARD B/2 ; si dirige verso l'angolo in alto
21 ; a sinistra
22 HEADING 9h
23 FORWARD A/2
24 HEADING 3h ; e punta verso destra (ore 3)
25 PENDOWN
26 ; disegna il rettangolo, ruotando in senso orario
27
28 FORWARD A
29 RIGHT 90
30 FORWARD B
31 RIGHT 90
32 FORWARD A
33 FILLCOLOR [230, 230, 230] ; riempie il rettangolo
34 PENUP
35 FORWARD B ; si posiziona nell'angolo in alto a
36 ; sinistra del foglio
37 ; ripete il giro marcando gli angoli e ponendo le scritte
38
39 HEADING 3h ; angolo alto destro
40 FILLCOLOR [50, 50, 50] ; fissa un grigio più scuro per i
41 ; cerchietti
42 FORWARD A
43 PENDOWN
44 CIRCLE 5
45 PENUP
46 FORWARD 10 LEFT 90 FORWARD 15
47 LABEL "[PAGESIZE[0], 0]"
48 BACK 15 RIGHT 90 BACK 10
49 ; angolo basso destro
50
51 FORWARD B
52 PENDOWN
53 CIRCLE 5
54 PENUP
55 FORWARD 15 LEFT 90 FORWARD 30 LEFT 90
56 LABEL "[PAGESIZE[0], PAGESIZE[1]]"
57 LEFT 90 FORWARD 30 RIGHT 90 FORWARD 15
58 RIGHT 180
59 ; angolo basso sinistro
60 FORWARD A
61 PENDOWN
62 CIRCLE 5
63 PENUP
64 FORWARD B ; angolo altro sinistro
65 PENDOWN
66 CIRCLE 5
67 PENUP
68 FORWARD 15
69 LEFT 90 FORWARD 10 RIGHT 90
70 LABEL "[0, PAGESIZE[1]]"
71 BACK 15
72 PENDOWN
73
74
75
76
77
78
79
80
81
82

```

83	HIDETURTLE	; mando la tartaruga a dormire
----	------------	--------------------------------

```

1 ; FIGURA 2 (Torna alla figura)
2 ; L'effetto dell'istruzione POSITION e le coordinate della pagina
3 ; Versione 1.
4 ; A.R. Formiconi
5 ; 25 luglio 2016
6
7 ; Versione scritta come viene, senza particolari ottimizzazioni
8 ; Predispone tutto
9
10
11 CLEARSCREEN
12 HOME
13
14 A = 200 ; lato corto del rettangolo che
15 ; rappresenta la pagina
16 B = 282 ; lato lungo
17 FILLCOLOR [230, 230, 230] ; fissa a grigio chiaro il colore di
18 ; riempimento
19 PENUP
20 FORWARD B/2 ; si dirige verso l'angolo in alto
21 ; a sinistra
22 HEADING 9h
23 FORWARD A/2
24 HEADING 3h ; e punta verso destra (ore 3)
25 PENDOWN
26
27 ; disegna il rettangolo, ruotando in senso orario
28
29 FORWARD A
30 RIGHT 90
31 FORWARD B
32 RIGHT 90
33 FORWARD A
34 FILL ; riempie il rettangolo
35 PENUP
36 RIGHT 90
37 FORWARD B ; si posiziona nell'angolo in alto
38 ; a sinistra del foglio
39
40 ; ripete il giro marcando gli angoli e ponendo le scritte
41
42 HEADING 3h ; angolo alto destro
43 FILLCOLOR [50, 50, 50] ; fissa un grigio più scuro per i
44 ; cerchietti
45 FORWARD A
46 PENDOWN
47 CIRCLE 5
48 PENUP
49 FORWARD 10 LEFT 90 FORWARD 15
50 LABEL "[PAGESIZE[0], 0]"
51 BACK 15 RIGHT 90 BACK 10
52
53 RIGHT 90 ; angolo basso destro
54 FORWARD B
55 PENDOWN
56 CIRCLE 5
57 PENUP
58 FORWARD 15 LEFT 90 FORWARD 30 LEFT 90
59 LABEL "[PAGESIZE[0], PAGESIZE[1]]"
60 LEFT 90 FORWARD 30 RIGHT 90 FORWARD 15
61 RIGHT 180
62
63 RIGHT 90 ; angolo basso sinistro
64 FORWARD A
65 PENDOWN
66 CIRCLE 5
67 RIGHT 90
68 PENUP

```

```

69 | BACK 15
70 | LABEL "[0, PAGESIZE[1]]"
71 | FORWARD 15
72
73 FORWARD B ; angolo altro sinistro
74 PENDOWN
75 CIRCLE 5
76 PENUP
77 FORWARD 15
78 LEFT 90 FORWARD 10 RIGHT 90
79 LABEL "[0, 0]"
80 BACK 15
81 PENDOWN
82
83 ; disegna il percorso della tartaruga in seguito all'istruzione
84 ; \POSITION [350,320]
85
86 HOME ; mi posiziono al centro
87 PENUP ; senza disegnare...
88 POSITION [298, 430] ; vado dove voglio piazzare...
89 HEADING 0 ; l'etichetta con le...
90 LABEL "[298, 421]" ; coordinate del centro
91 HOME ; torno al centro e...
92 PENDOWN ; disegno...
93 CIRCLE 5 ; il cerchietto centrale
94 FILLCOLOR "green" ; fisso il colore verde per la
95 POSITION [350,320] ; tartaruga, che mostrerà
96 P = POSITION ; applico POSITION [350,320]
97 H = HEADING ; memorizzo tale posizione...
98 PENUP ; e direzione
99 POSITION [300,320] ; alzo la penna
100 HEADING 0 ; mi sposto un po' per piazzare...
101 POSITION P ; orientata correttamente
102 LABEL "[350, 320]" ; l'etichetta
103 HEADING H ; ritorno alla posizione e
104 ; direzione...
105 POSITION P ; per lasciarvi visibile la
106

```

```

1 ; FIGURA 3 (Torna alla figura)
2 ; I riferimenti per l'istruzione HEADING
3 ; Versione 1.
4 ; A.R. Formiconi
5 ; 25 luglio 2016
6
7
8 ; Versione scritta in maniera più ordinata e strutturata, facendo
9 ; uso delle "procedure". La strutturazione del software di questo
10 ; tipo consente di rendere il codice più modulare e più mantenibile.
11
12 ; Prima vengono le subroutine e alla fine il programma vero e
13 ; proprio. Prima vanno messi i frammenti più elementari, in maniera
14 ; che LibreLogo li legga per primi. Questo serve perché quando nel
15 ; codice viene citata una subroutine, questa deve essere già stata
16 ; analizzata da LibreLogo.
17
18
19 ; Subroutine BR per disegnare un segmento lungo 10 pt dalla
20 ; posizione e lungo la direzione corrente senza muoversi
21 ; (come risultato finale)
22
23 ; Parametri:
24 ;     P: posizione corrente
25 ;     H: direzione corrente
26
27 TO BR P H
28     FORWARD 10
29     POSITION P
30     HEADING H
31 END

```

```

32 ; Subroutine TARROW per disegnare la punta di una freccia
33
34 TO TARROW
35     P = POSITION
36     H = HEADING
37     LEFT 160
38     BR P H
39
40     RIGHT 160
41     BR P H
42 END
43
44 ; Subroutine LBA scrivere il testo contenuto in T in vetta al segmento
45 ; di lunghezza L e ruotato dell'angolo A
46 ; Parametri:
47 ;     L: lunghezza del segmento
48 ;     A: angolo di rotazione del segmento
49 ;     T: testo da scrivere nell'etichetta
50
51 TO LBA L A T
52     PENUP
53     FORWARD L/10 + L/5 * SIN A*PI/180
54     H = HEADING
55     HEADING 0
56     LABEL T
57     PENDOWN
58 END
59
60 ; Subroutine LB scrivere il testo contenuto in T in una posizione
61 ; determinata in coordinate polari rispetto alla posizione corrente,
62 ; mediante la distanza L l'angolo A.
63
64 ; Parametri:
65 ;     L: distanza
66 ;     A: angolo
67 ;     T: testo da scrivere nell'etichetta
68
69 TO LB L A T
70     P0 = POSITION
71     PENUP
72     S = SIN A*PI/180
73     C = COS A*PI/180
74     POSITION [P0[0] + L * C, P0[1] - L * S]
75     HEADING 0
76     LABEL T
77     POSITION P0
78     HEADING 0
79     PENDOWN
80 END
81
82 ; Subroutine ARROW per disegnare, a partire dalla posizione corrente,
83 ; una freccia di lunghezza L, inclinata di un angolo A, e in vetta una
84 ; etichetta con il testo contenuto in T
85 ; Parametri:
86 ;     L: lunghezza della freccia
87 ;     A: angolo di rotazione della freccia
88 ;     T: testo da scrivere nell'etichetta
89
90 TO ARROW P0 A0 L A T
91     PENDOWN
92     HEADING A
93     FORWARD L
94     TARROW
95     LBA L A T
96     PENUP
97     POSITION P0
98     HEADING A0
99     PENDOWN
100 END
101
102 ; Questo è il programma vero e proprio che, come si vede, grazie al
103 ; ricorso alle procedure, è abbastanza conciso.
104
105 CLEARSCREEN           ; cancella il foglio

```

```

106 HOME ; mando a casa la tartaruga
107 HIDE TURTLE ; faccio il disegno senza vedere i
108 ; la tartaruga
109 FILLCOLOR [230, 230, 230] ; fisso il riempimento a un grigio scuro
110 CIRCLE 5 ; disegno un cerchietto nella posizione
111 ; centrale
112 P0 = POSITION ; memorizzo tale posizione iniziale...
113 A0 = HEADING ; e anche la direzione iniziale
114 L = 150 ; faccio la freccia lunga 150 pt
115 A = 60 ; e la voglio inclinata di 60 gradi
116
117 PENSIZE 1 ARROW P0 A0 L 0 "HEADING 0" ; freccia verticale
118 PENSIZE 0.5 ARROW P0 A0 L A "HEADING 30" ; freccia a 60 gradi
119 PENSIZE 1 ARROW P0 A0 L 90 "HEADING 90" ; freccia orizzontale
120 PENSIZE 0.5
121 ELLIPSE [L/3, L/3, 0, A, 3] ; arco di cerchio piccolo
122 LB L/4 A "30 gradi" ; etichetta "30 gradi"
123 ELLIPSE [L*2, L*2, 0, 90, 3] ; arco di cerchio grande

```

```

1 ; FIGURA 4 (Torna alla figura)
2 ; Esempi di spessore del tratto
3 ; Versione 1.
4 ; A.R. Formiconi
5 ; 25 luglio 2016
6
7 ; In questo esempio si illustra l'impiego dell'istruzione REPEAT,
8 ; per realizzare i "cicli" ("loop")
9
10 CLEARSCREEN
11 HOME
12 RIGHT 90
13 PENCOLOR "BLACK"
14
15 REPEAT 10 [
16   PENWIDTH REPCOUNT
17   FORWARD 100
18   PENUP FORWARD 50
19   HEADING 0h
20   LABEL "PENWIDTH " + STR REPCOUNT-1
21   HEADING 3h
22   BACK 50
23   PENUP
24   RIGHT 90
25   FORWARD 20
26   RIGHT 90
27   FORWARD 100
28   LEFT 180
29   PENDOWN
30 ]
31 PENWIDTH 0
32 HIDE TURTLE

```

Bibliografia

- [1] Munari Bruno. *Fantasia*. Laterza, Roma, 28 edition, 2017.
- [2] Reggini Horacio C. *Logo: ali per la mente*. Mondadori, Milano, 1984.
- [3] Emma Castelnuovo. *L'officina matematica – ragionare con i materiali*. Edizioni La Meridiana, Molfetta, 2 edition, 2008.
- [4] Lewis Colleen. How programming environment shapes perception, learning and goals: Logo vs. scratch. <http://ims.mii.lt/ims/konferenciju-medziaga/SIGCSE'10/docs/p346.pdf>, 2015. Accessed: 2017-08-13.
- [5] Weintrop David. Comparing block-based, text-based, and hybrid block-s/text programming environments in introductory computer science classes. <http://dweintrop.github.io/papers/Weintrop-diss-4pager.pdf>. Accessed: 2017-08-13.
- [6] Abelson Harold e Ploger D. DiSessa Andrea A. An overview of boxer. *Journal of Mathematical Behavior*, 10:3–15, 1991.
- [7] Abelson Harold e DiSessa Andrea A. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. The MIT Press, Cambridge, Masssachusetts, 1986.
- [8] Weintrop David e Wilensky U. To block or not to block, that is the question: Students' perceptions of blocks-based programming. http://dweintrop.github.io/papers/Weintrop_Wilensky_ICER_2015.pdf, 2015. Accessed: 2017-08-13.
- [9] Weintrop David e Wilensky U. Using commutative assessments to compare conceptual understanding in blocks based and text based programs. http://dweintrop.github.io/papers/Weintrop_Wilensky_ICER_2015.pdf, 2015. Accessed: 2017-08-13.
- [10] Persico Enrico. Che cos’è che non va? *Giornale di Fisica*, 1:64–67, 1956.
- [11] Sherin Bruce L. A comparison of programming languages and algebraic notation as expressive languages for physics. *Int. Journal of Computers for Mathematical Learning*, 6:1–61, 2001.
- [12] László Németh. Esempi di uso librelogo (ungherese). <http://www.numbertext.org/logo/logofuzet.pdf>. Accessed: 2017-08-13.

- [13] László Németh. Manuale di comandi di librelogo. https://help.libreoffice.org/Writer/LibreLogo_Toolbar/it. Accessed: 2017-08-13.
- [14] Seymour Papert. *Mindstorms, Children, Computers, and Powerful Ideas*. Basic books, New York, 2 edition, 1993.
- [15] Vetterling William T. e Flannery Brian P. Press William H., Teukolsky Saul A. *Numerical Recipes – The Art of Scientific Computing*. Cambridge University Press, Cambridge, Masssachusetts, 2007.
- [16] Lakó Viktória. Manuale di comandi di librelogo. http://szabadszoftver.kormany.hu/wp-content/uploads/librelogo_oktatasi_segedanyag_v4.pdf. Accessed: 2017-08-13.