

TOMMASO ANZIDEI



CALVINO

LINGUAGGIO DI PROGRAMMAZIONE DIDATTICO

Sommario

1	Se una notte d'inverno un programmatore.....	5
1.1	Che cosa è Calvino.....	5
1.2	Installare Calvino.....	6
1.3	Chi è Calvino.....	8
1.4	Calvino in cinque minuti.....	8
1.5	Calvino per esempi.....	12
2	Introduzione a Calvino.....	14
2.1	Calvino come calcolatore.....	14
2.2	Variabili.....	16
2.3	Parole, parole, parole.....	17
2.4	Logica booleana.....	20
2.5	Stampare.....	21
3	Primi passi da tartaruga.....	23
3.1	Disegnare un quadrato.....	24
3.2	Disegnare un triangolo.....	26
3.3	Labirinti.....	28
4	Strutture: ogni cosa al suo posto.....	32
4.1	Primi passi con le liste.....	32
4.2	Operare sulle liste.....	34
4.3	Mappe.....	35
4.4	Intervalli.....	38
5	Prendere il controllo.....	39
5.1	Se.....	40
5.2	Andare a ripetizione.....	41
5.3	Chiusure.....	43
6	Dipingere il mondo.....	46
6.1	Punti e quadrati.....	46
6.2	Forme di base.....	48
6.3	Trame scozzesi con le chiusure.....	51

7 Funzioni.....	53
7.1 Come funzionano le funzioni.....	53
7.2 Le funzioni sono chiusure.....	55
7.3 Insegniamo le funzioni alle tartarughe!.....	55
7.4 Leonardo Fibonacci.....	58
8 Il sogno delle tartarughe blu.....	59
8.1 Un fiore.....	59
8.2 Il sentiero dei nidi di ragno.....	61
8.3 Tartarughe nella spirale.....	65
9 Al-Jabr.....	67
9.1 Semplificazioni.....	67
9.2 Equazioni.....	69
9.3 Scomposizione in fattori di polinomi.....	70
10 Cosmicomiche.....	72
10.1 Giochi senza fine.....	73
10.2 Stelle variabili.....	74
10.3 Calvino 3D.....	77
11 Calcolo.....	80
11.1 Derivate.....	80
11.2 Integrali.....	82
12 I comandi di Calvino.....	84

1 Se una notte d'inverno un programmatore

Non c'è linguaggio senza inganno.
Italo Calvino, Le città invisibili

Ciao! Benvenuto nel manuale del linguaggio di programmazione Calvino.

In queste pagine ti spiegherò cosa devi sapere per iniziare a programmare con Calvino.

1.1 Che cosa è Calvino

Calvino è un linguaggio di programmazione pensato per chi vuole iniziare a programmare. Le caratteristiche principali di Calvino sono:

- **Programmazione conversazionale:** Calvino è un colloquio tra te e il computer. Ad ogni tua affermazione, espressa nel linguaggio Calvino, il computer risponderà immediatamente. Questo modo di programmare è detto conversazionale.
- **Computer Grafica:** attraverso l'uso di Calvino potrai imparare i fondamentali della grafica computerizzata a due e tre dimensioni.

- **Geometria della tartaruga:** Calvinò utilizza una metafora, detta geometria della tartaruga. In pratica, con alcuni comandi di Calvinò, sposterai sullo schermo una tartaruga, dotata di una penna. In questa maniera potrai creare grafici di varia natura.
- **Matematica:** Calvinò ti offre un gran numero di funzioni matematiche.
- **Simulazione scientifica:** con Calvinò potrai simulare vari fenomeni naturali.

1.2 Installare Calvinò

Prima di installare Calvinò dovrai verificare che **Java** sia presente sul tuo computer. Per farlo apri una finestra di comando e digita:

```
java -version
```

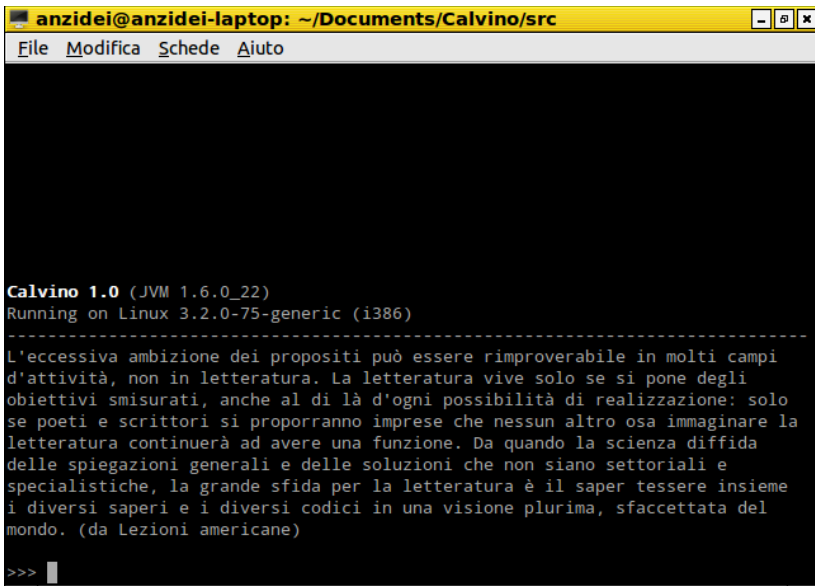
Se ricevi in risposta un messaggio del tipo:

```
java version "1.7.0_65"
```

allora Java è correttamente installato. Se invece ricevi un messaggio d'errore devi installare Java. Per farlo naviga col tuo browser alla pagina internet <http://www.java.com/it/> e segui le istruzioni.

Una volta verificata la presenza di Java puoi **installare Calvinò**. Per farlo naviga alla pagina: <http://calvinolang.github.io/>, clicca sul link scarica e poi sul bottone “Scarica Calvinò”. In pochi secondi sul tuo hard disk verrà scaricato un file di nome calvino.zip. Estrailo in una qualche cartella e poi esegui il file calvino.bat (se usi

Windows) o calvino.sh (se usi Linux o Mac OS X). Vedrai comparire sullo schermo la seguente finestra:



```
anzidei@anzidei-laptop: ~/Documents/Calvino/src
File Modifica Schede Aiuto

Calvino 1.0 (JVM 1.6.0_22)
Running on Linux 3.2.0-75-generic (i386)
-----
L'eccessiva ambizione dei propositi può essere rimproverabile in molti campi
d'attività, non in letteratura. La letteratura vive solo se si pone degli
obiettivi smisurati, anche al di là d'ogni possibilità di realizzazione: solo
se poeti e scrittori si proporranno imprese che nessun altro osa immaginare la
letteratura continuerà ad avere una funzione. Da quando la scienza diffida
delle spiegazioni generali e delle soluzioni che non siano settoriali e
specialistiche, la grande sfida per la letteratura è il saper tessere insieme
i diversi saperi e i diversi codici in una visione plurima, sfaccettata del
mondo. (da Lezioni americane)

>>> |
```

Questa è la finestra principale di Calvino. Dopo una citazione dello scrittore (che cambia ad ogni avvio di Calvino) vedi il prompt (>>>) nel quale puoi digitare i comandi. Sei pronto?

1.3 Chi è Calvino

Italo Calvino è stato uno degli scrittori più amati dello scorso secolo. Ti consiglio di leggere qualche cosa di lui, per esempio *Marcovaldo* oppure *I nostri antenati*. Calvino aveva una grande passione per la scienza. Ne *Le cosmicomiche* usa concetti scientifici per racconti di pura fantasia. Ecco, proprio quel libro, *Le Cosmicomiche*, ha ispirato il linguaggio di programmazione Calvino che vuole essere un punto d'incontro tra tecnica ed umanesimo. Buona programmazione!

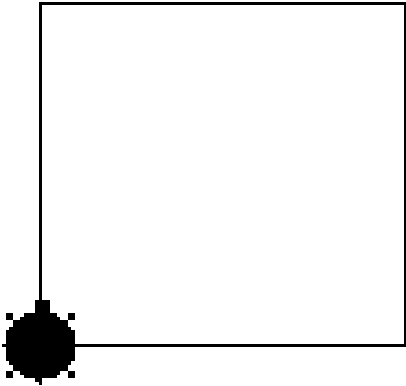
1.4 Calvino in cinque minuti

Se già hai programmato oppure sei un insegnante che vuole avere un quadro d'insieme questo paragrafo ti dà una panoramica essenziale del linguaggio Calvino. Viceversa puoi passare al capitolo successivo.

Calvino è un linguaggio funzionale, interattivo e dinamico.

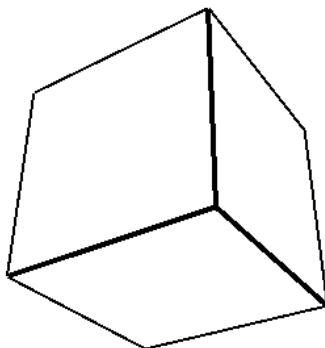
Ecco cosa puoi fare:

Compito	Esempio
<i>Calcolare</i>	<pre>>>> 2 + 2 4</pre>
<i>Dichiarare variabili</i>	<pre>>>> a = 10</pre>
<i>Manipolare stringhe</i>	<pre>>>> "divina commedia".reverse() aidemmoc anivid</pre>
<i>Stampare a video</i>	<pre>>>> println("ciao!") ciao!</pre>
<i>Valutare espressioni</i>	<pre>>>> 90 > 30*4 false</pre>
<i>Gestire liste</i>	<pre>>>> l = [10,30,20] [10, 30, 20] >>> l.sort() [10, 20, 30]</pre>
<i>Gestire dizionari</i>	<pre>>>> gusti = [nome:"Snoopy", razza:"Beagle"] {nome=Snoopy, razza=Beagle}</pre>
<i>Gestire intervalli di numeri</i>	<pre>>>> (1..5).each{n -> println(n*n)} 1 4 9 16 25</pre>
<i>Creare funzioni</i>	<pre>>>> quadrato = {n -> n*n} >>> quadrato(10) 100</pre>

<i>Algebra e Calcolo</i>	<pre>>>> algebra.compute("integrate(1/x,x)") Log(x)</pre>
<i>Geometria della tartaruga</i>	<pre>>>> logo.show() >>> t = logo.addTurtle() >>> (1..4).each{it -> t.fd(100) >>> t.rt(90)}</pre> 

Grafica 2D/3D

```
>>> graphics.show()  
>>> f={->graphics.translate(300,300,0)  
>>>     graphics.rotateX(10)  
>>>     graphics.rotateY(10)  
>>>     graphics.box(160)}  
>>> graphics.paint(f)
```



1.5 *Calvino per esempi*

Calvino è distribuito con alcuni **esempi** che trovi nell'omonima cartella. Ecco il loro significato:

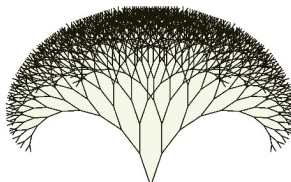
<i>albero.cal</i>	Disegna un albero
<i>fiore.cal</i>	Disegna un fiore
<i>semplificazione.cal</i>	Semplifica un'espressione algebrica
<i>casa.cal</i>	Disegna una casa
<i>funzioni.cal</i>	Esempi di Funzioni
<i>seno_e_coseno.cal</i>	Grafica sinusoidale
<i>cerchi_infiniti.cal</i>	Effetti grafici
<i>labirinto.cal</i>	Disegna un labirinto
<i>sfera.cal</i>	Disegna una sfera
<i>cubo.cal</i>	Disegna un cubo
<i>poligoni.cal</i>	Disegna alcuni poligoni
<i>spirale.cal</i>	Disegna una spirale
<i>each.groovy</i>	Dimostra la ripetizione

<i>primitive_2D.cal</i>	Disegna primitive 2D
<i>stella_variabile.cal</i>	Disegna una stella variabile
<i>elvis.cal</i>	Dimostra se..allora
<i>quadrato.cal</i>	Disegna un quadrato
<i>trama.cal</i>	Disegna una trama scozzese
<i>fibonacci.cal</i>	Algoritmo ricorsivo
<i>ragnatela.cal</i>	Disegna una ragnatela
<i>triangolo.cal</i>	Disegna un triangolo

Per provare un esempio si usa il comando load. Se digiti:

```
>>> load ./esempi/albero.cal
```

vedrai comparire la finestra di grafica con un albero che cambia dimensione a seconda della posizione del mouse:



2 Introduzione a Calvin

Poi, l'informatica. È vero che il software non potrebbe esercitare i poteri della sua leggerezza se non mediante la pesantezza del hardware; ma è il software che comanda, che agisce sul mondo esterno e sulle macchine, le quali esistono solo in funzione del software, si evolvono in modo d'elaborare programmi sempre più complessi. La seconda rivoluzione industriale non si presenta come la prima con immagini schiaccianti quali presse di laminatoi o colate d'acciaio, ma come i bits d'un flusso d'informazione che corre sui circuiti sotto forma d'impulsi elettronici. Le macchine di ferro ci sono sempre, ma obbediscono ai bits senza peso.

Italo Calvin, Lezioni americane

In questo capitolo ti aiuto nei primi passi con Calvin. Imparerai a fare calcoli con Calvin, il concetto di variabile, di stringa e condizione logica.

2.1 Calvin come calcolatore

Puoi usare Calvin come la tua **macchina calcolatrice**. Proviamo! Esegui Calvin e digita i seguenti comandi:

```
>>> 2+2
```

```
4
```

```
>>> 3*3
9
>>> 365 - 31
334
>>> 1440/12
120
>>> 10**3
1000
>>> 10/3
3.33333333333
>>> 10.intdiv(3)
3
>>> 10%3
1
```

Cosa abbiamo fatto? Abbiamo semplicemente provato l'addizione ($2+2$), la moltiplicazione ($3*3$), la sottrazione ($365 - 31$), la divisione ($1440/12$), sia con risultato decimale ($10/3$) che intero ($10.intdiv(3)$), l'elevazione a potenza (10^3) ed infine il resto ($10\%3$) ovvero la quantità di dividendo che è avanzata dalla divisione.

In generale l'operazione di divisione restituisce un risultato decimale. Se si ha bisogno del risultato intero è necessario utilizzare la funzione `intdiv` ma di cosa siano le funzioni (e i metodi) parleremo più avanti. Le operazioni disponibili sono

quindi:

+	Addizione	**	Potenza
*	Moltiplicazione	intdiv	Divisione intera
-	Sottrazione	%	Resto
/	Divisione		

2.2 Variabili

Uno dei concetti più frequenti in programmazione è quello di **variabile**. Una variabile è un contenitore nel quale conserviamo un oggetto con cui lavoriamo in Calvino. Per esempio, se scriviamo:

```
>>> n = 10
```

```
10
```

stiamo creando un contenitore, *n*, e vi inseriamo un valore, 10.

Una variabile ha quindi un nome (in questo caso *n*) ed un valore (in questo caso 10).

Le variabili sono particolarmente utili in compiti ripetitivi: possiamo assegnare un certo valore alla variabile, utilizzare la variabile come sinonimo di quel valore, assegnare un nuovo valore alla variabile e così via. Un esempio ti chiarirà il senso:

```
>>> n = 10
```

```
10
```

```
>>> n*n
```



```
100
>>> m = n/2
5
>>> m*n
50
>>> m = 50
50
>>> m*n
500
```

In Calvin non si deve abusare della creazione di nuove variabili. Gli approcci moderni alla programmazione mostrano infatti che il proliferare di variabili possa portare ad errori non immediatamente.

In ogni caso, quando si crea una variabile, è opportuno darle un nome che ne faccia intuire il significato. Per esempio:

```
>>> dividendo = 100
100
>>> divisore = 25
25
>>> risultato = dividendo/divisore
4
```

2.3 Parole, parole, parole

Per adesso abbiamo operato solo su numeri. Un tipo di dato di grande utilità nella programmazione è la **stringa**. Una stringa è

una sequenza di caratteri. In Calvinio, come in molti linguaggi di programmazione, una stringa è preceduta e seguita da un apice, singolo o doppio. Esempi di stringhe valide sono:

```
>>> "Calvino"
```

```
Calvino
```

```
>>> 'Calvino'
```

```
Calvino
```

```
>>> "Il castello dei destini incrociati"
```

```
Il castello dei destini incrociati
```

```
>>> 'Marcovaldo, ovvero le stagioni in città'
```

```
Marcovaldo, ovvero le stagioni in città
```

Proviamo insieme le operazioni più interessanti che possiamo eseguire sulle stringhe.

Concatenazione, ovvero l'unione di due stringhe:

```
>>> "Orlando " + "furioso"
```

```
Orlando furioso
```

Lunghezza di una stringa:

```
>>> "La nuvola di smog".size()
```

```
17
```

Sottostringa di una stringa, ovvero una sottoparte di una stringa:

```
>>> "La formica argentina".substring(3,10)
```

```
formica
```

Nell'esempio si è chiesto a Calvinio la sottostringa della stringa

"La formica argentina" a partire dal 3° carattere (compreso) fino al 10° carattere (escluso). Devi sapere, ed è importante, che in calvino si comincia a contare da 0. Così:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L	a		f	o	r	m	i	c	a		a	r	g	e	n	t	i	n	a

Possiamo rendere le lettere di una stringa tutte **minuscole** o **maiuscole**:

```
>>> "La speculazione edilizia".toUpperCase()
```

```
LA SPECULAZIONE EDILIZIA
```

```
>>> "Fiabe italiane".toLowerCase()
```

```
fiabe italiane
```

Puoi cercare la **posizione** del primo carattere di una sottostringa in una stringa:

```
>>> s = "Il castello dei destini incrociati"
```

```
Il castello dei destini incrociati
```

```
>>> s.indexOf("destini")
```

```
16
```

Puoi **invertire** i caratteri di una stringa:

```
>>> "calvino".reverse()
```

```
onivlac
```

Possiamo **sostituire** una sottostringa di una stringa con un'altra sottostringa:

```
>>> s = "italo sulvino"
```

```
italo sulvino
```

```
>>> s.replaceAll("su","ca")
```

```
italo calvino
```

Possiamo chiedere a Calvin il valore dell'**n-esimo carattere** di una stringa (nell'esempio il 14-esimo):

```
>>> "I giovani del Po"[14]
```

```
P
```

2.4 Logica booleana

In Calvin puoi scrivere **espressioni logiche**, per esempio:

```
>>> 2 > 1
```

```
true
```

oppure

```
>>> 2 > 3
```

```
false
```

Puoi usare i classici **operatori della logica booleana**:

AND:

```
>>> (1 == 1) && (1 > 0)
```

```
true
```

OR:

```
>>> (1 == 1) || (1 < 0)
```

```
true
```

NOT:

```
>>> !(1 > 0)
```

```
false
```

Negli esempi di cui sopra abbiamo utilizzato i classici operatori di uguaglianza e disuguaglianza:

Uguale	Maggiore	Minore	Maggiore o uguale	Minore o uguale	Diverso
==	>	<	>=	<=	!

2.5 Stampare

In Calvin puoi **stampare a video** tutto quello che vuoi col comando **println**. Vediamo:

```
>>> a = 1
```

```
1
```

```
>>> b = 2
```

```
2
```

```
>>> c = (a+b)*(a-b)
```

```
-3
```

```
>>> println(a)
```

```
1
```

```
>>> println(b)
```

```
2
```

```
>>> println(c)
```

```
-3
```

```
>>> println(a>b)
```

```
false
```

```
>>> println("calvino".toUpperCase())  
CALVINO
```

3 Primi passi da tartaruga

Solo dopo aver conosciuto la superficie delle cose, – conclude – ci si può spingere a cercare quel che c'è sotto. Ma la superficie delle cose è inesauribile..
Italo Calvino, Palomar

In questo capitolo ti parlerò della **geometria della tartaruga**. La geometria della tartaruga è un sistema, inventato dallo scienziato americano **Seymour Papert**, per l'apprendimento della programmazione (questo sistema è noto come **Logo**). Nella geometria della tartaruga esiste appunto una tartaruga che vive sopra un foglio di carta ed ha con sé una penna. La tartaruga può **camminare in avanti** di un certo numero di passi, **camminare indietro**, **ruotare** di un certo angolo, **abbassare** ed **alzare** la penna. Se la penna è abbassata il percorso della tartaruga viene disegnato sul foglio.

In Calvino si accede alla geometria della tartaruga attraverso l'oggetto `logo`. In particolare immaginiamo di voler creare una tartaruga di nome, per esempio, Clementina.

Col comando:

```
>>> logo.show()
```

attivi la geometria della tartaruga (appare una finestra vuota sullo schermo).

Se scrivi:

```
>>> clementina = logo.addTurtle()
```

vedrai una tartaruga blu al centro dello schermo.

Prova ora:

```
>>> clementina.fd(100)
```

la tartaruga si è spostata di 100 passi in avanti.

Adesso:

```
>>> clementina.rt(90)
```

la tartaruga si è girata a destra di 90°.

```
>>> clementina.bk(100)
```

e la tartaruga ha indietreggiato di 100 passi.

```
>>> clementina.lt(270)
```

la tartaruga gira a sinistra di 270°.

```
>>> clementina.pu()
```

clementina alza la penna e non scrive più. Infatti se di nuovo:

```
>>> clementina.fd(100)
```

Clementina camminerà in avanti ma senza lasciar segno.

Infine:

```
>>> clementina.pu()
```

ripristina la scrittura.

Vediamo ora come possiamo usare la geometria della tartaruga.

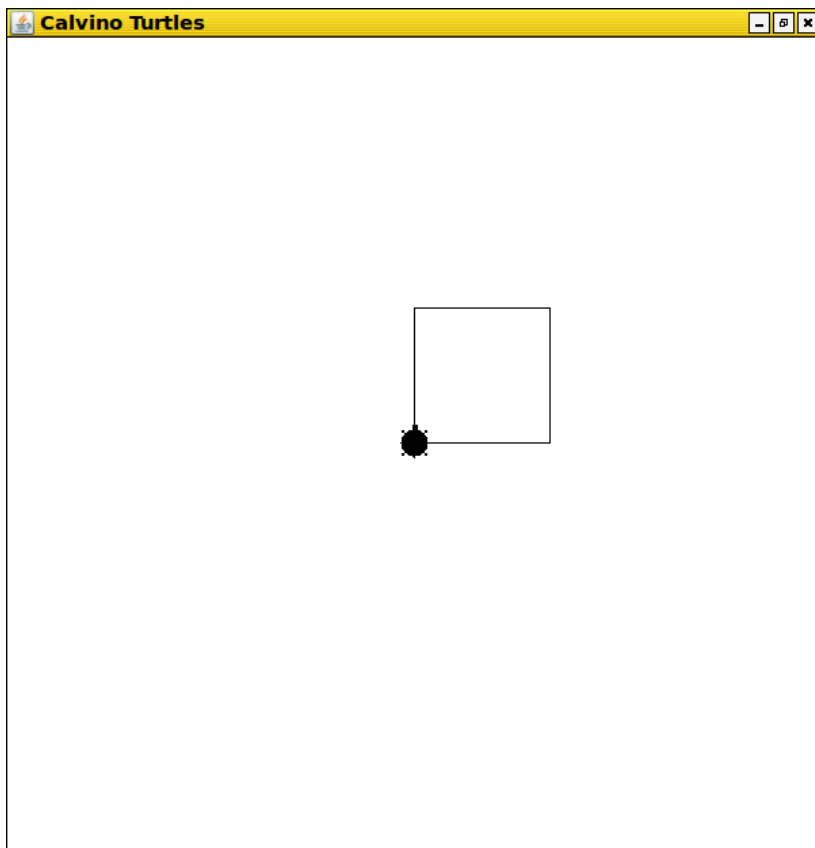
3.1 Disegnare un quadrato

In questo esempio istruiremo la nostra tartaruga a disegnare un quadrato. Per farlo disegneremo in sequenza i quattro lati,

ognuno della stessa lunghezza, ruotando di 90° gradi dopo aver tracciato un lato.

```
logo.show()  
t = logo.addTurtle()  
t.setPenColor(color.black)  
t.fd(100)  
t.rt(90)  
t.fd(100)  
t.rt(90)  
t.fd(100)  
t.rt(90)  
t.fd(100)  
t.rt(90)
```

Ecco il nostro quadrato:



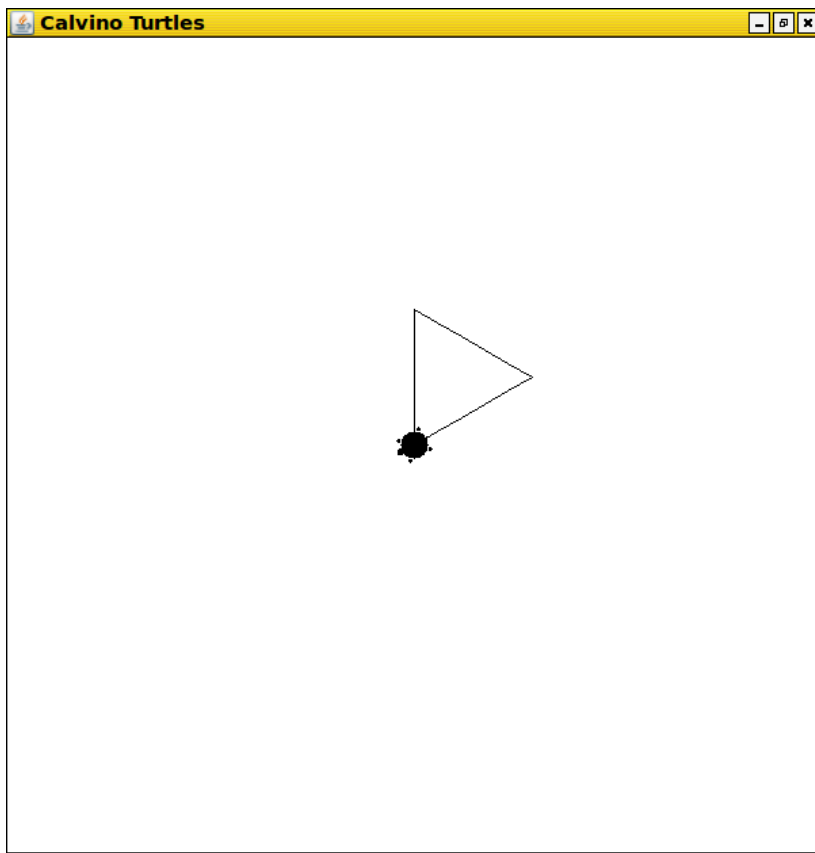
3.2 Disegnare un triangolo

Usando la stessa strategia dell'esempio precedente adesso disegneremo un triangolo equilatero. Come sai un angolo di un triangolo equilatero è pari a 120° . Ogni volta che disegniamo un lato dobbiamo quindi ruotare di 120° :

```
logo.show()  
t = logo.addTurtle()  
t.setPenColor(color.black)  
t.fd(100)  
t.rt(120)  
t.fd(100)  
t.rt(120)  
t.fd(100)
```

Puoi ripetere questo esperimento per tutti i poligoni regolari. La strategia è la stessa, devi tracciare tanti lati quanti ne ha il poligono ruotando ogni volta dell'angolo interno del poligono regolare ($360^\circ / \text{numero dei lati}$).

Ed ecco il triangolo equilatero:



3.3 *Labirinti*

In questo esempio parleremo di colori e dimensioni della penna. Scriviamo insieme il programma e commentiamolo.

Come esempio vogliamo disegnare un labirinto.

Dopo aver aperto la finestra di logo, `logo.show()`, ed aggiunto una tartaruga col comando `t = logo.addTurtle()`, la posizioniamo in basso a sinistra con i comandi

`t.setX(-100)` e `t.setY(-100)`; successivamente dichiariamo che useremo una penna di diametro 2 pixel, `t.penWidth(2)`, e che la penna sarà di colore “ciano” (in inglese cyan, cioè blu-verde).

```
logo.show()
t = logo.addTurtle()
t.setX(-100)
t.setY(-100)
t.penWidth(2)
t.setFillColor(color.cyan)
```

```
t.fd(200)
t.rt(90)
t.fd(200)
t.rt(90)
t.fd(200)
t.rt(90)
```

```
t.fd(180)
t.rt(90)
t.fd(180)
```

```
t.rt(90)
```

```
t.fd(160)
```

```
t.rt(90)
```

```
t.fd(160)
```

```
t.rt(90)
```

```
t.fd(140)
```

```
t.rt(90)
```

```
t.fd(140)
```

```
t.rt(90)
```

Hai capito? Fai passi sempre più piccoli (decresci di 20 pixel ogni volta) fino ad arrivare a:

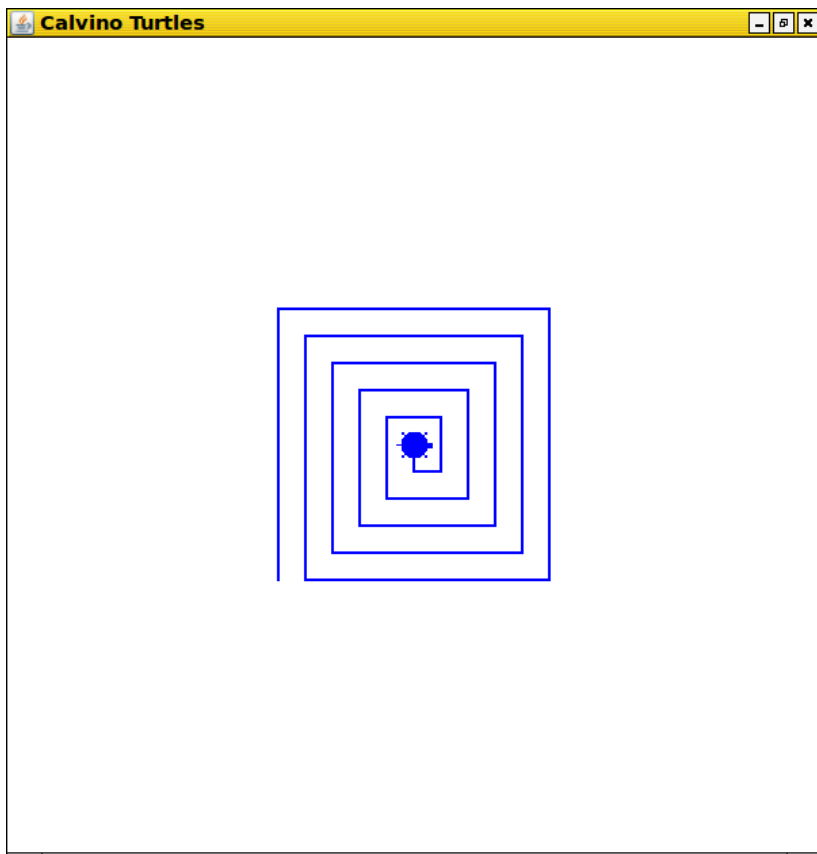
```
t.fd(20)
```

```
t.rt(90)
```

```
t.fd(20)
```

```
t.rt(90)
```

Ecco il nostro labirinto:



4 Strutture: ogni cosa al suo posto

Questo intendo quando dico che vorrei risalire il corso del tempo: vorrei cancellare le conseguenze di certi avvenimenti e restaurare una condizione iniziale. Ma ogni momento della mia vita porta con sé un'accumulazione di fatti nuovi e ognuno di questi fatti nuovi porta con sé le sue conseguenze, cosicché più cerco di tornare al momento zero da cui sono partito più me ne allontano : pur essendo tutti i miei atti intesi a cancellare conseguenze d'atti precedenti e riuscendo anche a ottenere risultati apprezzabili in questa cancellazione, devo però tener conto che ogni mia mossa per cancellare avvenimenti precedenti provoca una pioggia di nuovi avvenimenti che complicano la situazione peggio di prima e che dovrò cercare di cancellare a loro volta..

Italo Calvino, Se una notte d'inverno un viaggiatore

In questo capitolo ti introduco le **strutture** di Calvino. Le strutture sono collezioni di oggetti. Servono a raccogliere ed elaborare dati in maniera efficiente. Ne vediamo tre, le **liste**, le **mappe** e gli **intervalli**.

4.1 Primi passi con le liste

Una lista è una collezione di oggetti. In Calvino una lista si rappresenta elencando gli oggetti, separati da una virgola,

all'interno di due parentesi quadre. Per esempio puoi costruire la lista dei primi 10 numeri interi così:

```
>>> numeri = [1,2,3,4,5,6,7,8,9,10]
```

Per sapere la **dimensione** di una lista digita:

```
>>> numeri.size()
```

```
10
```

E' possibile chiedere a Calvino quale oggetto si trova in una certa **posizione** digitando `lista[posizione]`. Devi però sapere che Calvino inizia a contare da 0. Il primo elemento è quindi quello in posizione zero, il secondo in posizione uno e così via. Per esempio:

```
>>> numeri[0]
```

```
1
```

```
>>> numeri[3]
```

```
4
```

```
>>> numeri[9]
```

```
10
```

Aggiungere un elemento ad una lista è facile:

```
>>> numeri = numeri + 11
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Possiamo **sommare** due liste:

```
>>> numeri_pari = [2,4] + [6,8,10]
```

```
[2, 4, 6, 8, 10]
```

Una lista può non essere costituita da soli numeri:

```
>>> elementi = ["acqua", "terra", "aria"]  
[acqua, terra, aria]
```

E' possibile **aggiungere** elementi in una certa posizione (ma ricorda che Calvino inizia a contare da 0):

```
>>> ["acqua", "terra", "aria"].plus(1, "fuoco")  
[acqua, fuoco, terra, aria]
```

Una lista può essere **invertita**:

```
>>> ["acqua", "aria", "fuoco", "terra"].reverse()  
[terra, fuoco, aria, acqua]
```

4.2 Operare sulle liste

Adesso ti insegno una tecnica fondamentale sulle liste. Questa tecnica non solo è importante per le liste ma anche per il controllo dei programmi. Considera l'elenco dei primi 10 numeri naturali. Vogliamo ottenere il quadrato di ogni singolo numero. L'oggetto lista, come anche mappa e intervallo, contiene un metodo chiamato **each**, cioè ogni, ognuno (vedremo esattamente cos'è un metodo nel capitolo *Funzioni* ma immagina un metodo come un piccolo programma contenuto in un oggetto. Vediamo come possiamo usare **each** per il compito che ci siamo prefissati ovvero ottenere i quadrati dei primi 10 numeri interi:

```
>>> numeri = [1,2,3,4,5,6,7,8,9,10]  
>>> numeri.each{it -> println it*it}
```

1

4
9
16
25
36
49
64
81
100

Dunque, seguimi con attenzione. Abbiamo creato una nuova lista, `numeri`, e l'abbiamo riempita con i primi 10 numeri naturali. Poi abbiamo detto a `Calvino` di considerare ogni valore `it` dentro la lista e di stamparne il quadrato (`println it*it`).

In generale la sintassi del comando è:

`lista.each(v → fai qualcosa con v)`

Vedremo presto altri esempi.

4.3 Mappe

Le **mappe**, in `Calvino`, sono dei **dizionari**. Un dizionario è un elenco di coppie di chiavi e valori. Facciamo un esempio semplice, la pagella. Nella pagella la chiave è la materia, il valore è il voto. Diciamo che oggi vai a scuola e ricevi questa pagella:

Materia	Voto
Italiano	<i>Sette</i>
Storia	<i>Otto</i>
Geografia	<i>Sette</i>
Scienze	<i>Sei</i>
Matematica	<i>Cinque</i>
Arte	<i>Sette</i>
Educazione Fisica	<i>Otto</i>
Condotta	<i>Nove</i>

Abbiamo un problema a Matematica! Vedrai che Calvino ti farà appassionare alla materia!

Comunque Calvino ti aiuta a rappresentare la tua materia così:

```
>>> pagella = [:]  
{}  
>>> pagella["italiano"] = 7  
7  
>>> pagella["storia"] = 8  
8  
>>> pagella["geografia"] = 7  
7  
>>> pagella["scienze"] = 6
```

6

```
>>> pagella["matematica"] = 5
```

5

```
>>> pagella["arte"] = 7
```

7

```
>>> pagella["educazione fisica"] = 8
```

8

```
>>> pagella["condotta"] = 9
```

9

La prima istruzione, `pagella = [:]` , crea una mappa vuota. Le successive associano ad ogni materia il voto. Se adesso vuoi controllare il tuo voto, per esempio in scienze, dovrai digitare:

```
>>> pagella["arte"]
```

7

Quante sono le materie nella tua pagella? Semplice:

```
>>> pagella.size()
```

8

Vuoi vedere solo i voti?

```
>>> pagella.values()
```

```
[7, 8, 7, 6, 5, 7, 8, 9]
```

Vuoi vedere solo le materie?

```
>>> pagella.keySet()
```

```
[italiano, storia, geografia, scienze, matematica,  
arte, educazione fisica, condotta]
```

In quale materie hai l'insufficienza?

```
>>> pagella.findAll{it.value < 6}
{matematica=5}
```

In quali materie vai molto bene?

```
>>> pagella.findAll{it.value > 7}
{storia=8, educazione fisica=8, condotta=9}
```

Vuoi vedere la pagella ordinata per voto?

```
>>> pagella.sort{it.value}
{matematica=5, scienze=6, italiano=7, geografia=7,
arte=7, storia=8, educazione fisica=8, condotta=9}
```

Più avanti ti spiegherò meglio il significato di `it` e delle parentesi `{}`.

4.4 Intervalli

Un **intervallo** in *Calvino* è una sequenza di numeri interi consecutivi. Per esempio i primi dieci numeri naturali:

```
>>> i = 1..10
```

```
1..10
```

```
>>> println i
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Vediamo qualche funzione utile:

```
>>> i.size() (=10)
```

```
>>> i[4] (=5)
```

5 Prendere il controllo

Chiese a Marco Kublai: – Tu che esplori intorno e vedi i segni, saprai dirmi verso quale futuro ci spingono i venti propizi. – Per questi porti non saprei tracciare la rotta sulla carta né fissare la data dell'approdo. Alle volte mi basta uno scorcio che s'apre nel bel mezzo d'un paesaggio incongruo un affiorare di luci nella nebbia, il dialogo di due passanti che s'incontrano nel viavai, per pensare che da lì metterò assieme pezzo a pezzo la città perfetta, fatta di frammenti mescolati col resto, d'istanti separati da intervalli, di segnali che uno manda e non sa chi li raccoglie. Se ti dico che la città cui tende il mio viaggio è discontinua nello spazio e nel tempo, ora più rada ora più densa, tu non devi credere che si possa smettere di cercarla. Forse mentre noi parliamo sta affiorando sparsa entro i confini del tuo impero; puoi rintracciarla, ma a quel modo che t'ho detto.
Italo Calvino, *Le città invisibili*

Quando scrivi un programma ti trovi spesso davanti due compiti importanti: le **decisioni** e le **ripetizioni**.

Le decisioni sono momenti del programma nei quali devi valutare se una certa variabile rispetta una certa condizione. Per esempio puoi chiederti se la variabile **temperatura** è minore di 0 (e in quel caso, per esempio, assegnare **true** alla variabile **ghiaccio**).

Le ripetizioni sono invece compiti che devi, appunto, ripetere molte volte.

Vediamo di che si tratta.

5.1 Se...

Nel mondo della programmazione dei computer è molto frequente prendere delle decisioni. Per esempio le centraline elettroniche delle automobili moderne possono accendere i fari se fuori sta diventando buio. Il loro ragionamento è:

Se	Se
<i>fuori è buio</i>	<i>condizione</i>
allora	allora
<i>accendi i fari</i>	<i>azione</i>
altrimenti	altrimenti
<i>non fare niente</i>	<i>altra azione</i>

Questo tipo di semplice ragionamento è noto, in informatica, come costruito *if-then-else* (*se-allora-altrimenti*) e lo troverai in tutti i programmi di computer.

In Calvino il costrutto *if-then-else* si rappresenta così:

condizione ? azione : altra azione

Per esempio:

```
eta = computer.input("Quanti anni hai?")
(eta > 17) ? "Sei maggiorenne." : "Sei minorenni."
```


In questo caso Calvino ti chiede quanti anni hai e, se ne hai almeno 18, stampa la stringa “maggiorenne”, viceversa quella “minorenne”.

5.2 Andare a ripetizione

Spesso programiamo i computer a ripetere le cose molte volte. Se ti chiedessi di stampare 5 volte la parola “Ciao” tu potresti rispondermi così:

```
>>> println "Ciao"
```

```
Ciao
```

```
null
```

```
>>> println "Ciao"
```

```
Ciao
```

```
null
```

```
>>> println "Ciao"
```

```
Ciao
```

```
null
```

```
>>> println "Ciao"
```

```
Ciao
```

```
null
```

```
>>> println "Ciao"
```

```
Ciao
```

Possiamo insegnare a Calvino a ripetere 5 volte questa operazione così:

```
>>> [1,2,3,4,5].each{println "Ciao"}  
Ciao  
Ciao  
Ciao  
Ciao  
Ciao  
[1, 2, 3, 4, 5]
```

In pratica abbiamo preso una lista contenente 5 oggetti e per ogni oggetto abbiamo chiesto a Calvin di scrivere “Ciao”. Come vedi abbiamo ottenuto lo stesso risultato con una sola istruzione invece che cinque. Un'alternativa è utilizzare, invece che la lista, un intervallo:

```
>>> (1..5).each{println "Ciao"}  
Ciao  
Ciao  
Ciao  
Ciao  
Ciao  
1..5
```

Usare un intervallo è più utile se devo ripetere una stessa operazione molte volte.

5.3 Chiusure

Attenzione, adesso parliamo di un argomento importantissimo, le **chiusure**! Non ti preoccupare se troverai l'argomento indigesto, la prima volta le chiusure sono difficili per tutti. In realtà abbiamo già visto le chiusure di sfuggita, adesso ne parliamo più in dettaglio.

Una chiusura è semplicemente un pezzetto di programma che se ne sta per i fatti suoi finchè qualcuno decide di prenderlo ed eseguirlo.

Ecco una chiusura:

```
{println "sono solo una chiusura"}
```

Una chiusura se ne sta per i fatti suoi finchè qualcuno non la chiama:

```
[1,2,3].each{println "sono solo una chiusura"}  
sono solo una chiusura  
sono solo una chiusura  
sono solo una chiusura
```

Una chiusura può essere definita dichiarando uno o più parametri di input. Per esempio:

```
{nome -> println nome}
```

Se adesso scriviamo:

```
>>> ["Italo", "Dino", "Carlo"].each{nome -> println  
nome}
```

otteniamo:

Italo

Dino

Carlo

Torniamo all'esempio delle pagelle e scriviamo velocemente:

```
>>> pagella = [:]  
>>> pagella["italiano"] = 7  
>>> pagella["storia"] = 8  
>>> pagella["geografia"] = 7  
>>> pagella["scienze"] = 6  
>>> pagella["matematica"] = 5  
>>> pagella["arte"] = 7  
>>> pagella["educazione fisica"] = 8  
>>> pagella["condotta"] = 9
```

Considera la chiusura:

```
{k,v -> println "Hai preso " + v + " a " + k}
```

Proviamo a scrivere:

```
>>> pagella.each{k,v -> println "Hai preso " + v +  
" a " + k}
```

ottiene in risposta:

Hai preso 7 a italiano

Hai preso 8 a storia

Hai preso 7 a geografia

Hai preso 6 a scienze

Hai preso 5 a matematica

Hai preso 7 a arte

Hai preso 8 a educazione fisica

Hai preso 9 a condotta

Una chiusura è quindi un'espressione del tipo:

{elenco di parametri → istruzioni}

Dai, usiamo le chiusure per calcolare la tua media!

La media dei voti è la somma dei voti diviso il numero delle materie.

Crea una variabile somma:

```
>>> somma = 0
```

Incrementa somma aggiungendo via via il voto:

```
>>> pagella.each{k,v -> somma = somma + v}
```

Calcola la media:

```
>>> somma/pagella.size()
```

7.125

Non male, ma puoi fare meglio!

Più avanti incontrerai ancora le chiusure quando parleremo delle **funzioni** (che sono essenzialmente chiusure).

6 Dipingere il mondo

*In mezzo a un fitto bosco, un castello
dava rifugio a quanti la notte aveva
sorpreso in viaggio: cavalieri e dame,
cortei reali e semplici viandanti. Passai
per un ponte levatoio sconnesso...*
Italo Calvino, Il castello dei destini
incrociati

Calvino contiene in se un sistema di grafica molto popolare nel mondo chiamato **Processing** (<http://processing.org>). Processing permette di sviluppare facilmente applicazioni grafiche come giochi, animazioni e contenuti interattivi. Processing, di per se, non è interattivo. Devi cioè scrivere tutto il programma in una volta e vedere poi come e se funziona. Calvino ti permette di lavorare con Processing in forma interattiva. Per accedere a questo sistema di grafica in Calvino devi digitare:

```
>>> graphics.show()
```

Vediamo cosa ci aspetta in questo mondo.

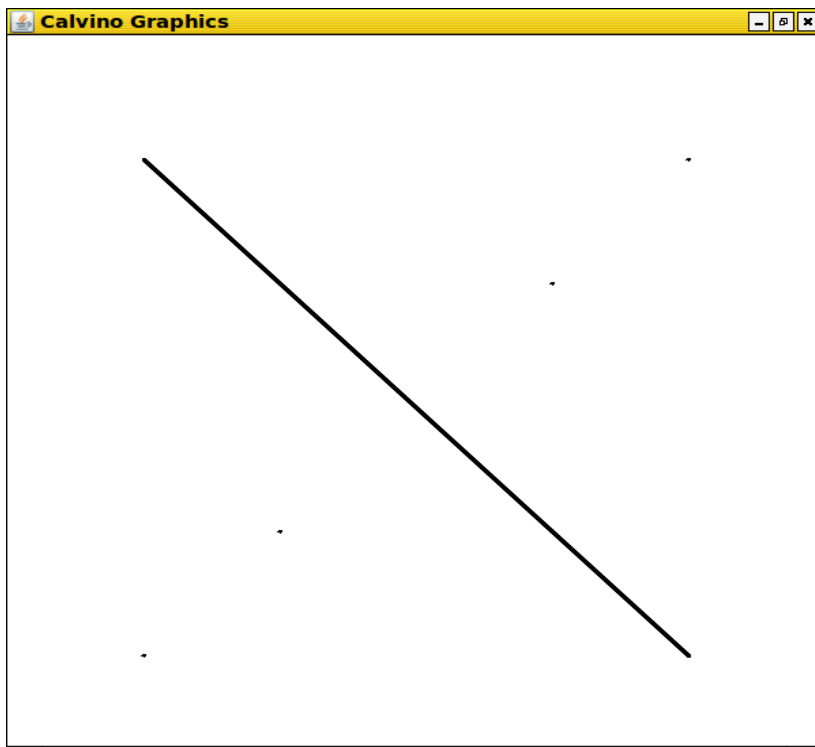
6.1 Punti e quadrati

Disegnare **primitive geometriche** con Calvino è immediato. In questo esempio utilizzeremo punti (funzione `graphics.point`) e linee (funzione `graphics.line`). Per disegnare un punto è sufficiente chiamare la funzione `graphics.point` specificando le coordinate cartesiane x e y del punto:

`graphics.point(x,y)`. Per tracciare una linea tra i punti P di coordinate (x_1, y_1) e Q di coordinate (x_2, y_2) dobbiamo chiamare la funzione `graphics.line(x1,y1,x2,y2)`.

```
graphics.show()  
// Disegna una linea  
graphics.line(100, 100, 500, 500)  
// Disegna quattro punti  
graphics.point(100, 500)  
graphics.point(200, 400)  
graphics.point(400, 200)  
graphics.point(500, 100)
```

Il risultato è il seguente:



6.2 Forme di base

L'oggetto **graphics** di Calvin ci permette di disegnare alcune forme di base che sono triangolo, rettangolo, quadrilatero, ellisse ed arco. Il rettangolo e il cerchio sono casi particolari di, rispettivamente, rettangolo ed ellisse.

Vediamo un esempio e discutiamolo:


```
graphics.show()  
graphics.triangle(18, 18, 18, 360, 81, 360);  
graphics.rect(81, 81, 63, 63);  
graphics.quad(189, 18, 216, 18, 216, 360, 144,  
360);  
graphics.ellipse(252, 144, 72, 72);  
graphics.triangle(288, 18, 351, 360, 288, 360);  
PI = 3.1415926535  
graphics.arc(479, 300, 280, 280, PI, 2*PI);
```

Per disegnare un triangolo si ha bisogno di 3 punti, uno per vertice, e quindi 6 coordinate, ovvero quei numeri che abbiamo passato al comando `graphics.triangle`.

Per disegnare un triangolo Calvino ha bisogno del vertice in basso a sinistra (quindi due coordinate), larghezza ed altezza ovvero i quattro numeri che usiamo in `graphics.rect`.

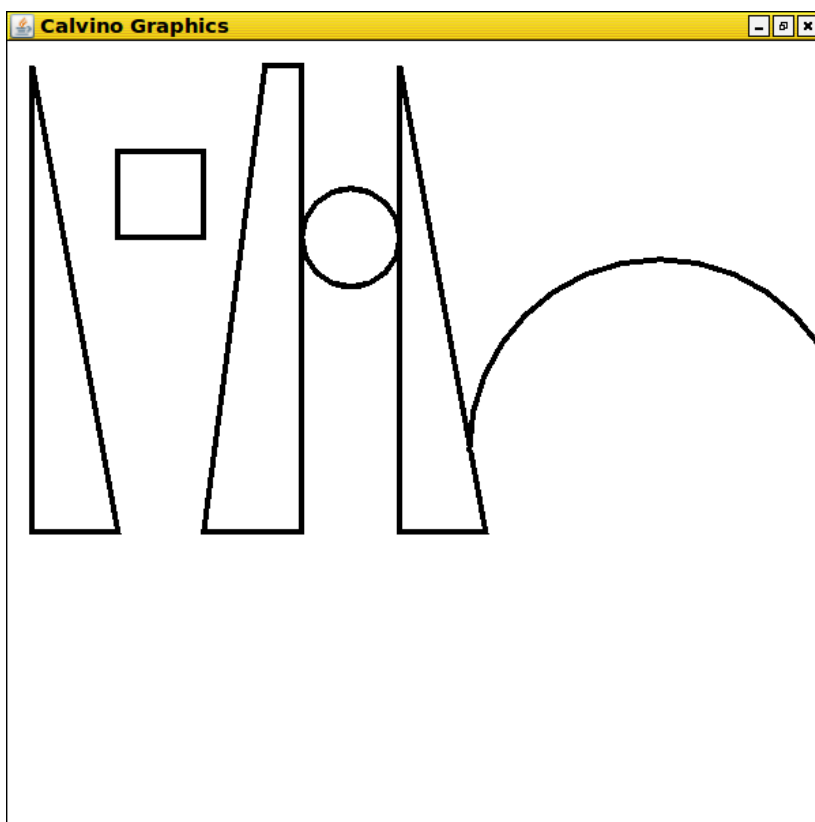
Per disegnare un quadrilatero generico si ha bisogno di 4 punti (i vertici) e quindi 8 coordinate, ovvero quei numeri che abbiamo passato al comando `graphics.rect`.

Per disegnare un'ellisse Calvino ha bisogno del centro dell'ellisse (quindi due coordinate), larghezza ed altezza ovvero i quattro numeri che usiamo in `graphics.ellipse`.

Per disegnare un arco Calvino ci chiede i parametri di un'ellisse (che è il percorso sul quale l'arco è tracciato) e poi due angoli da cui iniziare e terminare l'arco. Questi angoli si misurano non in gradi ma in radianti. Non ti preoccupare se non sai cosa sono

i radianti: è una misura dell'angolo per cui l'angolo piatto (180°) corrisponde al valore di π (3.1415926535...). Nell'esempio chiediamo a Calvino di disegnare un arco da 180° fino a 360° .

Ecco le nostre forme:



6.3 Trame scozzesi con le chiusure

Usiamo adesso il concetto di chiusura e ripetizione nel disegno delle forme.

Ecco il programma:

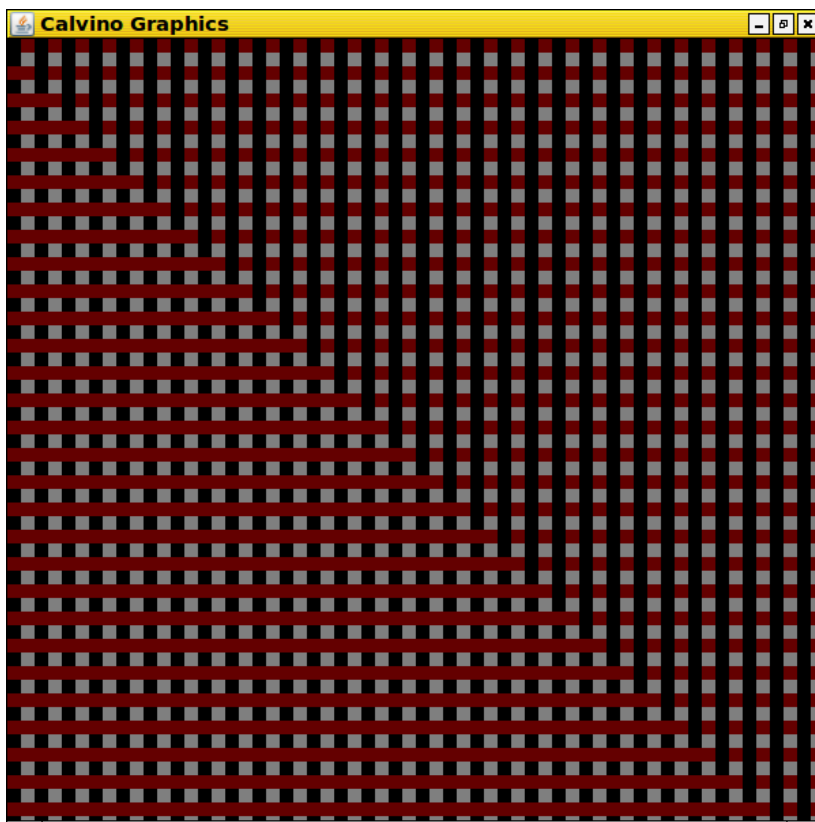
```
graphics.show()
graphics.background(127)
graphics.noStroke()
((0..600).step(20)).each{it ->
  graphics.fill(129, 206, 15)
  graphics.rect(0, it, graphics.width, 10)
  graphics.fill(0)
  graphics.rect(it, 0, 10, graphics.height)
}
```

Commentiamolo: per prima cosa apriamo la finestra `graphics`, `graphics.show()`, poi dipingiamo lo sfondo di grigio con `graphics.background(127)` ed infine disabilitiamo la penna con `graphics.noStroke()`. Adesso facciamo un ciclo sui pixel della finestra (che sono 600). Il ciclo funziona così: prendiamo i numeri da 0 a 600 ad intervalli di 20 usando il range `(0..600)` e percorrendolo 20 passi per volta con `((0..600).step(20))`. Ad ogni ciclo disegniamo un rettangolo rosso ed un rettangolo nero.

Il rettangolo rosso è largo quanto la finestra, alto 10 pixel ed il suo vertice in alto a sinistra è sempre sul lato sinistro della finestra ma scende ad ogni ciclo di 20 pixel.

Il rettangolo nero è alto quanto la finestra, largo 10 pixel ed il suo vertice in alto a sinistra è sempre sul lato superiore della finestra ma si sposta verso destra ad ogni ciclo di 20 pixel.

Il risultato è “Made in Scotland”:



7 Funzioni

*Un classico è un libro che non ha mai
finito di dire quel che ha da dire.*
Italo Calvino, Perché leggere i classici

Le **funzioni** sono il cuore di Calvino. Una funzione è un modo di definire ed isolare un certo compito che vuoi svolgere in modo da poterlo ripetere quando vuoi.

7.1 Come funzionano le funzioni

Adesso ti spiego come funzionano le funzioni. Certamente sai o ricordi che l'area di un triangolo è uguale alla sua base moltiplicata per la sua altezza diviso 2.

Con Calvino questo calcolo è semplice. Se un triangolo ha altezza 10 e base 5 puoi trovare l'area digitando il comando:

```
>>> 10*5/2
```

```
25
```

Una funzione è semplicemente un modo di insegnare al computer a ripetere questa operazione per qualsiasi valore della base e dell'altezza. Una funzione è un comando di Calvino costituito da tre parti:

- *nome*: come vuoi chiamare la funzione in modo da ricordarti a cosa serve
- *parametri*: un elenco di variabili sulle quali la funzione opererà

- *corpo*: il cuore della funzione, ovvero i singoli comandi che costituiscono la funzione

Un'esempio ti aiuterà: scriviamo la funzione area.

```
>>> area = {base, altezza -> base*altezza/2}
```

Nell'esempio area è il nome, base e altezza sono i parametri e $\text{base} \times \text{altezza} / 2$ è il corpo.

Adesso se scrivi:

```
>>> area(10,5)
```

25

ottiene la stessa area che avevi calcolato prima ma ora puoi ripetere al volo il calcolo per qualsiasi valore di base ed altezza:

```
>>> area(112980,3876)
```

218955240

Facile, no? Proviamo un esempio calcistico: quanti punti ha una squadra che ha vinto 16 partite, pareggiato 9 e perso 4? Dobbiamo ricordare che nel gioco del calcio si ricevono 3 punti per ogni vittoria, 1 per ogni pareggio e 0 per ogni sconfitta. Ecco la funzione:

```
>>> punti = {vittorie, pareggi, sconfitte ->
```

```
>>> 3*vittorie + 1*pareggi + 0*sconfitte}
```

e quindi:

```
>>> punti(16,9,4)
```

57

Naturalmente puoi osservare: le sconfitte non portano punti e

nella moltiplicazione si può omettere il numero 1. La funzione può quindi essere semplificata:

```
>>> punti = {vittorie, pareggi, sconfitte ->
>>> 3*vittorie + pareggi}
```

ma il risultato sarà lo stesso, prova ascrivere:

```
>>> punti(16,9,4)
```

7.2 Le funzioni sono chiusure

Se ricordi abbiamo parlato delle **chiusure** un paio di capitoli fa.

Avevamo detto che una chiusura è un'espressione del tipo:

$$\{\text{elenco di parametri} \rightarrow \text{istruzioni}\}$$

Ecco, le funzioni invece sono espressioni del tipo:

$$\text{nome funzione} = \{\text{elenco di parametri} \rightarrow \text{istruzioni}\}$$

Quindi una funzione è una chiusura alla quale diamo un nome.

7.3 Insegniamo le funzioni alle tartarughe!

Il concetto di funzione è molto potente. Vediamo come possiamo utilizzarle con le nostre tartarughe. Ricorderai che per tracciare un poligono regolare con una tartaruga devi tracciare tanti lati quanti ne ha il poligono ruotando ogni volta dell'angolo interno del poligono regolare ($360^\circ / \text{numero dei lati}$). Questo è facile per i primi poligoni regolari ma diventa tedioso al crescere dei lati. Possiamo però scrivere una funzione che riceve come parametri il numero di lati, la lunghezza di ciascun lato ed una tartaruga. Questa funzione

disegna il poligono regolare corrispondente al numero di lati.
Proviamo!

Prima di tutto scriviamo la funzione poligono:

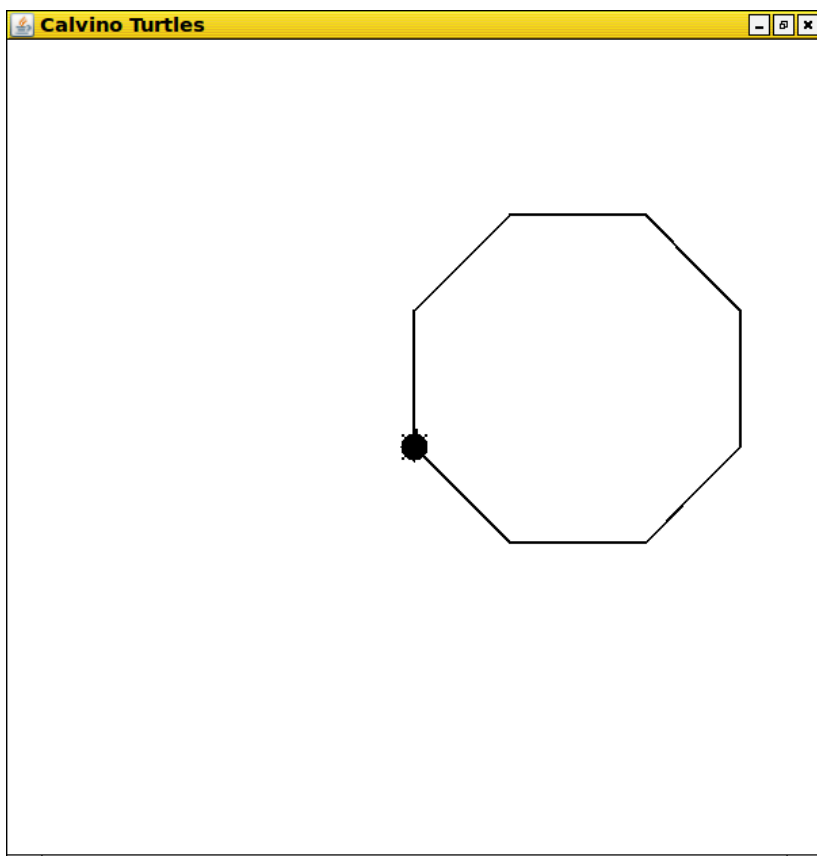
```
poligono = {  
n, t, l ->  
(1..n).each{  
    t.fd(l)  
    t.rt(360/n)}  
}
```

Come vedi per ogni lato la tartaruga `t` traccia un lato di lunghezza `l` e ruota dell'ampiezza dell'angolo del poligono regolare (che è pari all'angolo giro diviso il numero dei lati). Più facile a scriversi che a dirsi!

Poi eseguiamola:

```
logo.show()  
t = logo.addTurtle()  
t.penWidth(2)  
t.setPenColor(color.black)  
poligono(8,t,100)
```

Ecco il nostro ottagono:



7.4 Leonardo Fibonacci

Leonardo Fibonacci fu un grande matematico italiano del medioevo. La sua fama è legata, tra le altre cose, alla successione di Fibonacci. La successione di Fibonacci è una successione di numeri interi positivi in cui ciascun numero è la somma dei due precedenti e i primi due termini della successione sono entrambi pari ad 1. La successione di Fibonacci è un ottimo esempio di **ricorsione**. Infatti la successione può essere così definita:

$$F(1) = 1$$

$$F(2) = 1$$

$$F(n) = F(n-1) + F(n-2) \text{ se } n > 2$$

In Calvino possiamo scrivere una funzione ricorsiva, che cioè chiama se stessa, come la funzione F sopra definita:

$$f = \{it < 1 ? 0 : it == 1 ? 1 : f(it-1) + f(it-2)\}$$

La ricorsione è uno strumento potentissimo in programmazione.

8 Il sogno delle tartarughe blu

*Se una notte d'inverno un viaggiatore,
fuori dell'abitato di Malbork, sporgendosi
dalla costa scoscesa senza temere il vento
e la vertigine, guarda in basso dove
l'ombra s'addensa in una rete di linee che
s'allacciano, in una rete di linee che
s'intersecano sul tappeto di foglie
illuminate dalla luna intorno a una fossa
vuota – Quale storia laggiù attende la
fine? – chiede, ansioso d'ascoltare il
racconto.*

Italo Calvino, *Se una notte d'inverno un
viaggiatore*

Vediamo ora come il mondo della grafica della tartaruga si arricchisca usando controlli, strutture e funzioni.

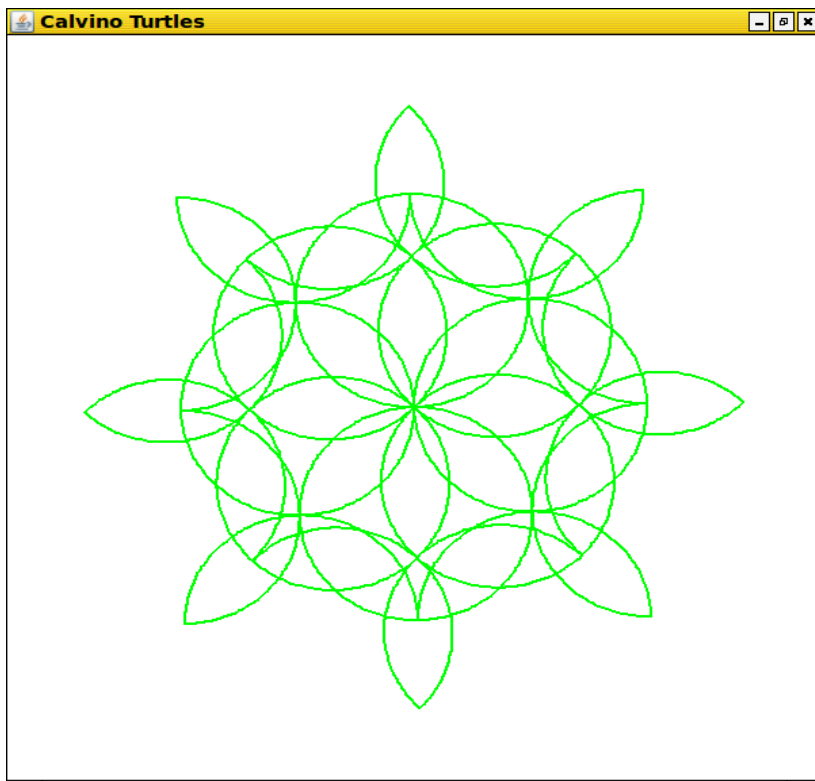
8.1 Un fiore

Lo script che presentiamo disegna un fiore. Modificando il parametro `p` usando valori compresi tra 1 e 7 è possibile ottenere forme diverse. Nell'esempio `p` è pari a 4.

```
logo.show()  
p = 4  
t = logo.addTurtle()  
t.ht()  
t.penWidth(2)
```

```
t.setPenColor(color.green)
(1..8).each{
    t.rt(45)
    (1..p).each {
        (1..90).each{
            t.fd(3)
            t.rt(2)
        }
        t.rt(90)
    }
}
```

Ed ecco il risultato:



8.2 Il sentiero dei nidi di ragno

Insegniamo adesso a tre tartarughe a disegnare una ragnatela. La prima tartaruga tesserà gli assi della ragnatela a partire dal centro dello schermo. Le altre due tartarughe tesseranno entrambe un poligono. Il risultato finale sarà una semplice ragnatela. Iniziamo!

Per prima cosa scriviamo la funzione poligono che ha come parametri un numero di lati, una tartaruga e la lunghezza di un lato (lo abbiamo già fatto nel capitolo precedente, *repetita iuvant*):

```
poligono = {  
  n, t, l ->  
  (1..n).each{  
    t.fd(l)  
    t.rt(360/n)}  
}
```

Poi scriviamo una funzione che permette ad una tartaruga di tracciare un segmento a partire dall'origine:

```
raggio = {  
  t ->  
  t.fd(200)  
  t.rt(180)  
  t.fd(200)  
  t.rt(18)  
}
```

In pratica la funzione raggio fa disegnare un segmento ad una tartaruga, le fa fare mezzo giro su se stessa, la fa tornare da dove era partita ed infine la fa ruotare di un piccolo angolo. Quest'ultima rotazione serve a costruire raggi distinti chiamando la funzione ripetutamente. Possiamo iniziare il lavoro vero e proprio aggiungendo e posizionando tre tartarughe:

```
t1 = logo.addTurtle(color.green)
t2 = logo.addTurtle(color.red)
t3 = logo.addTurtle(color.blue)
t2.setX(-100)
t2.setY(-100)
t3.setX(-50)
t3.setY(-50)
t1.penWidth(2)
t1.setPenColor(color.black)
t2.penWidth(2)
t2.setPenColor(color.black)
t3.penWidth(2)
t3.setPenColor(color.black)
```

Infine, la prima tartaruga traccia i raggi (11):

```
(1..11).each{raggio(t1,)}
```

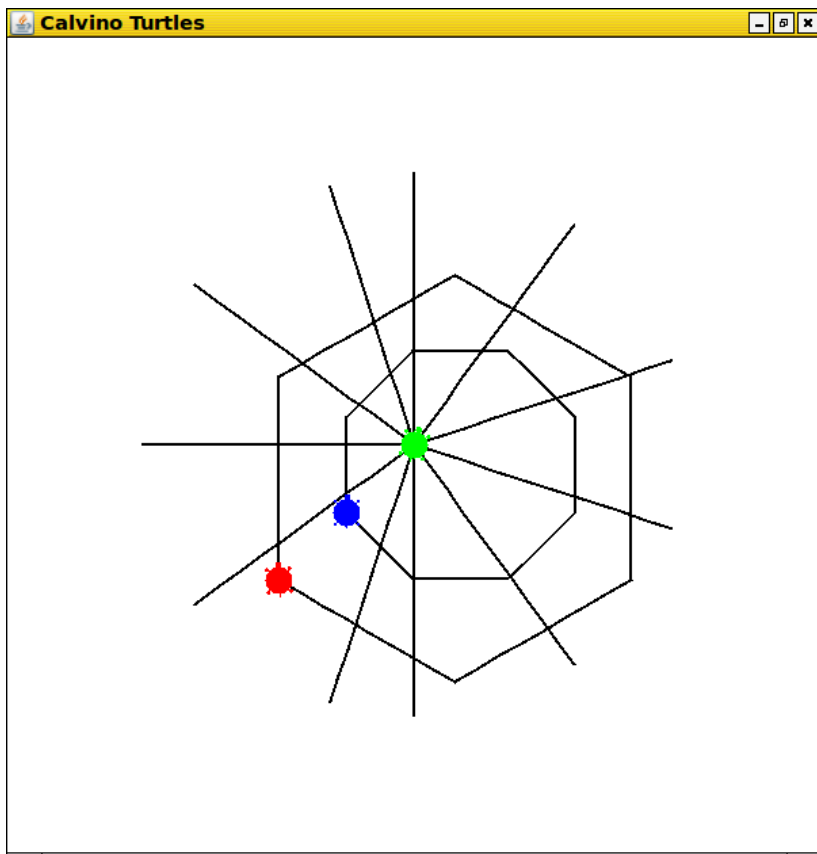
La seconda disegna un esagono:

```
poligono(6, t2, 150)
```

La terza traccia un ottagono:

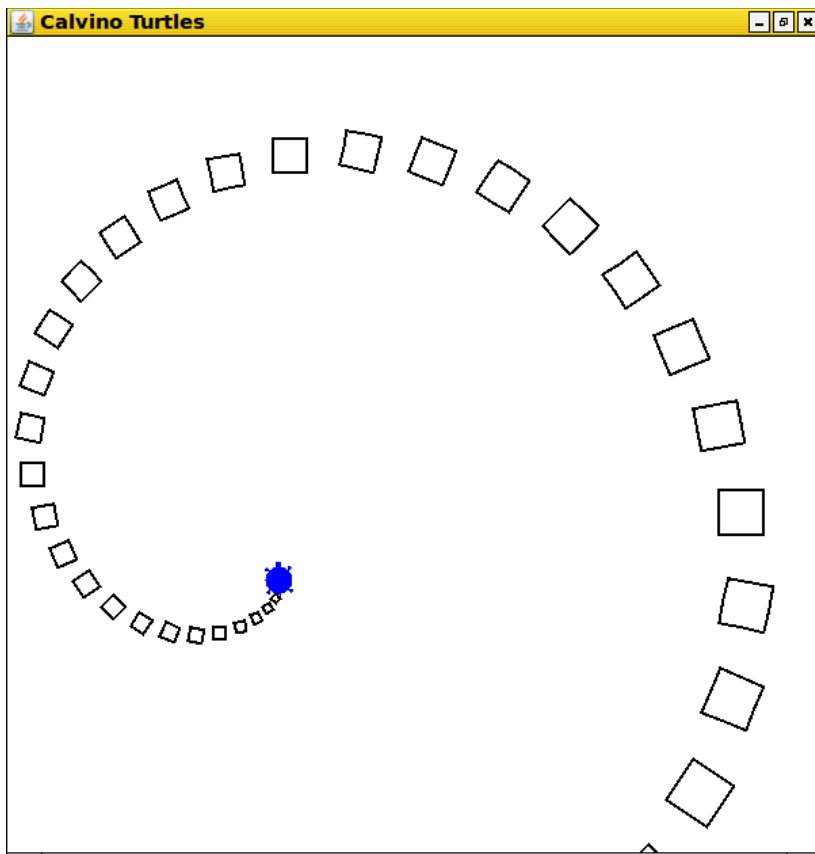
```
poligono(8, t3, 70)
```

Il risultato è questa ragnatela:



8.3 *Tartarughe nella spirale*

Disegniamo una **spirale**! Partiamo subito dal risultato:



Guardiamo insieme come è fatto il programma e

commentiamolo.

```
logo.show()
i = 1
t = logo.addTurtle()
t.penWidth(2)
t.setY(-100)
t.setX(-100)
t.setPenColor(color.black)
(1..48).each { t.pu()
  t.fd(1)
  t.pd()
  (1..4).each { t.fd(i)
    t.rt(90)
  }
  t.pu()
  t.bk(i*2)
  i = i + 1
  t.rt(360/32)
}
t.setY(-100)
t.setX(-100)
```

In pratica il trucco sta nel ripetere 48 volte il disegno di un rettangolo via via sempre più piccolo, nell'avanzare di un passo dopo aver disegnato il rettangolo ruotando di poco.

9 Al-Jabr

Marco Polo descrive un ponte, pietra per pietra.

– Ma qual è la pietra che sostiene il ponte? – chiede Kublai Kan.

– Il ponte non è sostenuto da questa o quella pietra, – risponde Marco, – ma dalla linea dell'arco che esse formano. Kublai Kan rimane silenzioso, riflettendo.

Poi soggiunge: – Perché mi parli delle pietre? È solo dell'arco che mi importa.

Polo risponde: – Senza pietre non c'è arco.

Italo Calvino, Le città invisibili

Calvino contiene anche un modulo, chiamato **algebra**, per il calcolo simbolico. Il **calcolo simbolico** non calcola il valore di un'espressione matematica ma rielabora un'espressione in un'altra espressione. Proviamo!

9.1 Semplificazioni

Con l'oggetto algebra puoi **semplificare** espressioni algebriche complesse. Vediamo alcuni esempi.

Semplifichiamo l'espressione $a + a + 4b^2 + 3b^2$:

```
>>> algebra.compute("Simplify(a+a+4*b^2+3*b^2)")  
7*b^2+2*a
```

ovvero $7b^2 + 2a$

Semplifichiamo l'espressione:

$$\frac{12x^2}{3x}$$

```
>>> algebra.compute("Simplify((12*x^2)/(3*x))")
4*x
```

a scuola scriveremmo $4x$.

Adesso proviamo un “classico” dell'algebra:

$$(x+1)(x-1)$$

```
>>> algebra.compute("Simplify((x+1)*(x-1))")
x^2-1
```

cioè $x^2 - 1$.

Altro esempio:
$$\frac{6a+6b+4(a+b)^2}{4(a+b)^2}$$

```
>>> algebra.compute("Simplify((6*a+6*b+4*(a+b)^2)/
(4*(a+b)^2))")
```

$(b+a+3/2)*(b+a)^{(-1)}$ che è pari a:

$$\frac{3+2a+2b}{2(a+b)}$$

Adesso qualcosa di difficile:

$$\frac{2+x}{x+3} - \frac{3x-1}{x^2+x-6} - \frac{x}{x+3}$$

```
>>> algebra.compute("Simplify((2+x)/(x+3) - (3*x-
1)/(x^2+x-6) - (x)/(x+3))")
```

$$-(x-2)^{-1}$$

ovvero:

$$\frac{1}{2-x}$$

9.2 Equazioni

Calvino può aiutarti a risolvere **equazioni**. Iniziamo da un'equazione semplicissima:

$$x - 1 = 0$$

Dirai al volo: $x = 1$! Bene, chiediamo a Calvino la stessa cosa:

```
>>> algebra.compute("solve(x-1==0,x)")
{{x->1}}
```

In pratica dobbiamo chiamare il metodo `compute` di `algebra` e chiedergli di risolvere (`solve`) l'equazione $x - 1 == 0$ nella variabile x . Calvino risponderà con un elenco di soluzioni (in questo caso una sola).

Passiamo all'equazione di secondo grado $x^2 - 1 = 0$ che tu sai avere come soluzioni 1 e -1:

```
>>> algebra.compute("solve(x^2-1==0,x)")
{{x->-1},{x->1}}
```

E in generale possiamo chiedere anche la soluzione di:

$$ax^2 + bx + c = 0$$

```
>>> algebra.compute("solve(a*x^2 + b*x + c == 0,x)")
{{x->(-1/2*(-4*a*c+b^2)^(1/2)-1/2*b)*a^(-1)},
```

$\{x \rightarrow (1/2 * (-4 * a * c + b^2)^{(1/2)} - 1/2 * b) * a^{(-1)}\}$

che se guardi bene sono proprio:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

E se l'equazione non ha soluzioni reali come in $x^2 + 1 = 0$?

```
>>> algebra.compute("solve(x^2+1==0,x)")
```

```
{{x->-I},{x->I}}
```

dove I è il numero immaginario i .

9.3 Scomposizione in fattori di polinomi

Scomporre un polinomio in fattori non è un'operazione banale ma con *Calvino* può riuscire in un batter d'occhio.

Prova a scomporre in fattori: $-12x^2 + 27$

```
>>> algebra.compute("factor(-12x^2+27)")
```

```
(2*x+3)*(-6*x+9)
```

Adesso prova: $2x^2 - 18x - 72$

```
>>> algebra.compute("factor(2x^2-18x-72)")
```

```
(2*x-24)*(x+3)
```

Passa alla terza potenza: $4x^3 + 9x^2 + 2x$

```
>>> algebra.compute("factor(4x^3+9x^2+2x)")
```

```
x*(4*x+1)*(x+2)
```

Prova ora: $3x^3 + 2x^2 - 3x - 2$

```
>>> algebra.compute("factor(3x^3 + 2x^2 -3x -2)")
```

$$(x-1)*(x+1)*(3*x+2)$$

Aumenta di grado: $3x^4 - 75$

```
>>> algebra.compute("factor(3x^4-75)")
```

$$(3*x^2-15)*(x^2+5)$$

Passa a due variabili: $x^8 - y^4$

```
>>> algebra.compute("factor(x^8-y^4)")
```

$$(y+x^2)*(-y+x^2)*(y^2+x^4)$$

10 Cosmicomiche

Una volta, secondo Sir George H. Darwin, la Luna era molto vicina alla Terra. Furono le maree che a poco a poco la spinsero lontano: le maree che lei Luna provoca nelle acque terrestri e in cui la Terra perde lentamente energia. Lo so bene! – esclamò il vecchio Qfwfq, – voi non ve ne potete ricordare ma io sì. L'avevamo sempre addosso, la Luna, smisurata: quand'era il plenilunio – notti chiare come di giorno, ma d'una luce color burro –, pareva che ci schiacciasse; quand'era lunanuova rotolava per il cielo come un nero ombrello portato dal vento; e a lunacrescente veniva avanti a corna così basse che pareva lì lì per infilzare la cresta d'un promontorio e restarci ancorata. Ma tutto il meccanismo delle fasi andava diversamente che oggiigiorno: per via che le distanze dal Sole erano diverse, e le orbite, e l'inclinazione non ricordo di che cosa; eclissi poi, con Terra e Luna così appicciate, ce n'erano tutti i momenti: figuriamoci se quelle due bestione non trovavano modo di farsi continuamente ombra a vicenda.
Italo Calvino, Le cosmicomiche

Ora che il nostro arsenale di comandi si è potenziato vediamo qualche esempio avanzato di grafica. Il concetto fondamentale qui è che l'oggetto **graphics** contiene una funzione, detta **paint**, alla quale puoi passare una **chiusura**. Una volta passata

la chiusura questa sarà ripetuta per sempre creando un'animazione. Vediamo qualche esempio.

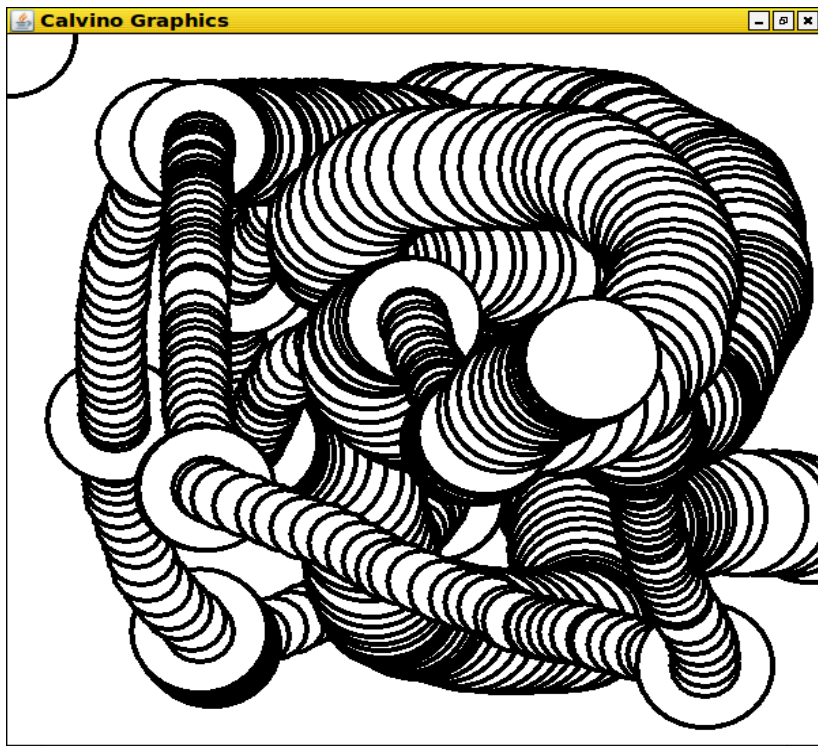
10.1 Giochi senza fine

L'esempio che ti faccio vedere adesso è semplice ma molto divertente. Usando l'oggetto `graphics` disegneremo un cerchio ogni volta che il mouse si sposta e il cerchio sarà più piccolo se tieni premuto un qualsiasi tasto del mouse. Vediamo il programma e commentiamolo!

```
graphics.show()
graphics.show()
f={-> (mousePressed)?
graphics.ellipse(graphics.mouseX,graphics.mouseY,50
,50) :
graphics.ellipse(graphics.mouseX,graphics.mouseY,100,100)}
graphics.paint(f)
```

Per prima cosa mostriamo la finestra di `graphics` (`graphics.show`). Poi scriviamo una funzione di nome `f` senza parametri di input. Questa funzione controlla che il mouse abbia un tasto premuto e, in caso affermativo, disegna un cerchio di raggio 50 con centro la posizione corrente del mouse che è il punto di coordinate `graphics.mouseX`, `graphics.mouseY`. Se invece il mouse non è premuto il programma disegna un cerchio di raggio 100. La vista vale il

biglietto!



10.2 Stelle variabili

Le stelle variabili intrinseche sono stelle la cui dimensione cambia ciclicamente facendo sì che la loro luminosità cambi nel tempo.

Ecco una piccola simulazione di una stella variabile che si basa sulla funzione seno:

```
graphics.show()
```

```
angle = 0
```

```
h = graphics.height - 10
```

```
graphics.noStroke()
```

```
graphics.noStroke()
```

```
graphics.fill(255, 204, 0)
```

```
f = { -> graphics.background(0)
```

```
d = 10 + (math.sin(angle + math.PI/2) * h/2) + h/2
```

```
w = graphics.width/2
```

```
graphics.ellipse(w, w, d, d)
```

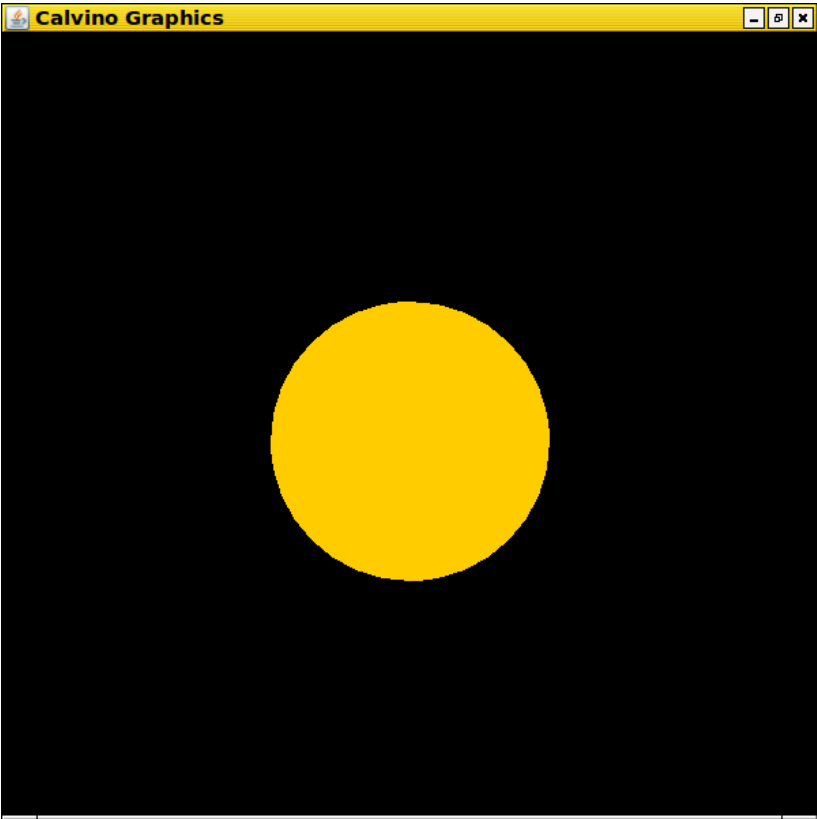
```
angle = angle + 0.02
```

```
}
```

```
graphics.paint(f)
```

L'effetto di espansione/contrazione è ottenuto tramite la funzione seno, che è periodica, e viene utilizzata nel calcolo del raggio del cerchio (d) che rappresenta la stella variabile.

Il risultato non è apprezzabile nella seguente figura, prova tu stesso ad eseguire il programma!



10.3 **Calvino 3D**

Adesso proviamo insieme a scrivere un piccolo programma di grafica in 3 dimensioni.

Studiamo insieme questo codice:

```
graphics.erase()
graphics.show()
graphics.smooth()
graphics.fill(128)
r = 0
f = { -> graphics.background(255)
      graphics.lights()
      graphics.ambientLight(23,222,6)
      graphics.translate(width/2, height/2, 0)
      r = r + 1/100
      graphics.rotateX(r)
      graphics.rotateY(r)
      graphics.box(160)
}
graphics.paint(f)
```

La prima istruzione, `graphics.erase()`, pulisce lo schermo.

La seconda, `graphics.show()`, mostra la finestra di `graphics`.

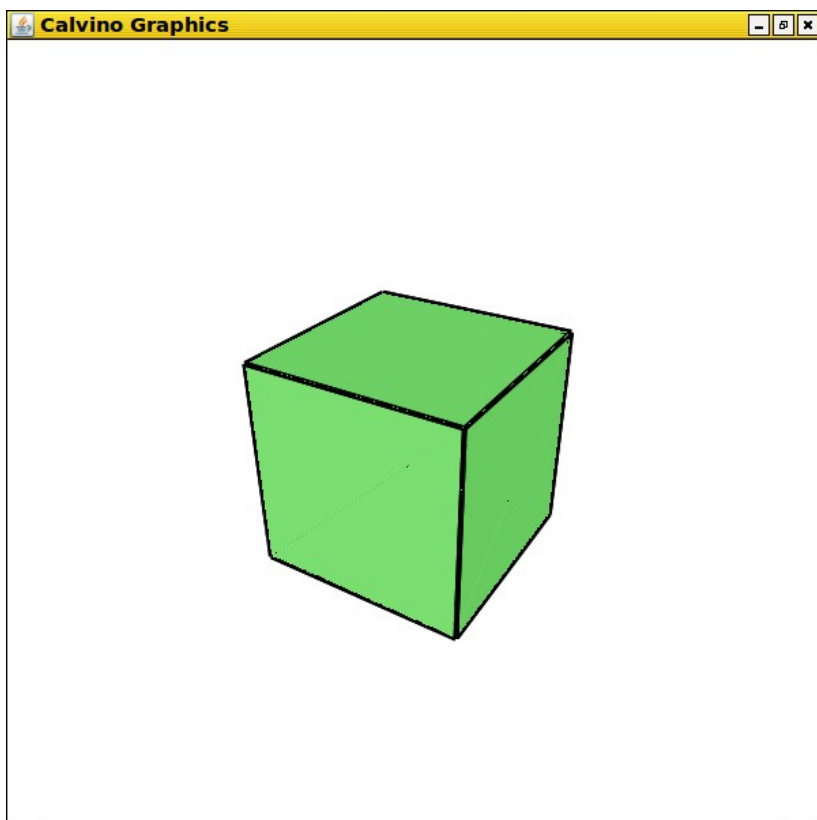
La terza, `graphics.smooth()`, ammorbidisce, leviga gli oggetti grafici prodotti.

Successivamente dichiariamo di usare un colore di riempimento grigio, `graphics.fill(128)`, ed una variabile `r` (che rappresenterà un angolo) pari a 0.

A questo punto scriviamo una funzione, `f`, i cui compiti sono:

- Impostare uno sfondo bianco con l'istruzione `graphics.background(255)`
- Illuminare il nostro mondo con valori di default grazie al comando `graphics.lights()`
- Impostare una luce d'ambiente verde, `graphics.ambientLight(23,222,6)`
- Portare il punto in cui vogliamo disegnare al centro della scena con `graphics.translate(width/2, height/2, 0)`
- Modificare la variabile `r` di una piccola quantità, $r = r + 1/100$ e ruotare la scena di `r` con `graphics.rotateX(r)`, `graphics.rotateY(r)`
- Disegnare un cubo di lato 160 pixel con `graphics.box(160)`

Lanciando il programma vedrai un cubo verde che ruota lentamente su entrambi gli assi. In alternativa al cubo puoi disegnare una sfera di raggio `r` col comando `sphere(r)`.



11 Calcolo

*Chiamasi classico un libro che si
configura come equivalente dell'universo,
al pari degli antichi talismani.*
Italo Calvino, Perché leggere i classici

Continuiamo ad usare l'oggetto algebra ma applichiamo al calcolo infinitesimale.

11.1 **Derivate**

Grazie all'oggetto algebra è facile **derivare** funzioni. Vediamo qualche esempio utile in fisica classica.

In cinematica il moto uniforme è rappresentato dall'equazione:

$$x(t) = a + bt$$

dove $x(t)$ è lo spazio percorso in un tempo t alla velocità b partendo da a . In Calvino l'operazione di derivata:

$$\frac{d(a+bt)}{dt}$$

lo si esprime così (diff vuol dire proprio derivare) :

```
>>> algebra.compute("diff(a+b*t,t)")
```


Il risultato che vedremo sullo schermo è

b

Otteniamo quindi b, proprio la velocità.

Vediamo ora una derivata di una funzione trigonometrica, tipica del moto oscillatorio:

$$x(t) = \cos(\omega t)$$

Digitiamo: (usiamo o al posto di ω)

```
>>> algebra.compute("diff(cos(o*t),t)")
```

Il risultato che vedremo sullo schermo è

```
-o*Sin(o*t)
```

il che è giusto perché

$$\frac{d \cos(\omega t)}{dt} = -\omega \sin(\omega t)$$

11.2 Integrali

Con Calvino puoi anche **integrare** funzioni. Vediamo alcuni classici esempi.

Integrazione di una costante, $\int a \, dx = ax + c$:

```
>>> algebra.compute("integrate(a,x)")
a*x
```

Integrazione di un monomio, $\int x^n = \frac{x^{n+1}}{n+1} + c$:

```
>>> algebra.compute("integrate(x^n,x)")
x^(n+1)*(n+1)^(-1)
```

Integrale della funzione razionale $\frac{1}{x}$, $\int \frac{1}{x} = \ln(|x|) + c$:

```
>>> algebra.compute("integrate(1/x,x)")
Log(x)
```

Integrale della funzione seno, $\int \sin(x) = -\cos(x) + c$

```
>>> algebra.compute("integrate(sin(x),x)")
-Cos(x)
```

Integrale della funzione esponenziale, $\int e^x = e^x + c$:

```
>>> algebra.compute("integrate(E^x,x)")
E^x
```

Un integrale notevole, $\int \frac{1}{1+x^2} = \arctg(x) + c$

```
>>> algebra.compute("integrate(1/(1+x^2),x)")  
ArcTan(x)
```

Un altro integrale notevole,

$$\int \sin^2(x) = \frac{1}{2}(x - \sin(x)\cos(x)) + c$$

```
>>> algebra.compute("integrate(sin(x)^2,x)")  
-1/2*cos(x)*sin(x)+1/2*x
```

12I comandi di Calvino

Io credo alla pedagogia repressiva. Mi rendo conto di essere molto antiquato in questo, ma continuo ad essere convinto che resti il miglior metodo d'educazione alla cultura.
Italo Calvino, citato in Luca Clerici,
Bruno Falchetto, Calvino & l'editoria.

Ti riassumo qui i principali **comandi** di Calvino

Comando	Esempio
STRINGHE	
<code>stringa.size()</code>	Dimensione di una stringa, es. "precipitevolissimevolmente".size() restituisce 26
<code>stringa.substring(n,m)</code>	Sottostringa della stringa es. "Palomar".substring(2,4) restituisce lo
<code>stringa1 + stringa2</code>	Concatenazione, es. "conca" + "tena" + "zione" restituisce "concatenazione"
LISTE	
<code>lista.size()</code>	Dimensione di una lista, es. ["a","b","c"].size() restituisce 3
<code>lista[n]</code>	n-esimo elemento, es. ["a","b","c"][2] restituisce "c"
INTERVALLI	
<code>intervallo.size()</code>	Dimensione di un intervallo, es. (50..200).size() restituisce 151
<code>intervallo[n]</code>	n-esimo elemento, es. (50..200)[10] restituisce 60

GRAFICA	
<code>graphics.mouseX</code>	Coordinata x del mouse
<code>graphics.mouseY</code>	Coordinata y del mouse
<code>graphics.height</code>	Altezza della finestra
<code>graphics.width</code>	Larghezza della finestra
<code>graphics.background(R,G,B)</code>	Imposta il colore dello sfondo con i tre valori di rosso (R), verde (G) e blu (B)
<code>graphics.box(l)</code>	Cubo di lato l
<code>graphics.hide()</code>	Nasconde la finestra grafica
<code>graphics.show()</code>	Mostra la finestra grafica
<code>graphics.color(R,G,B)</code>	Imposta un colore con i tre valori di rosso (R), verde (G) e blu (B)
<code>graphics.paint(f)</code>	Anima una funzione f
<code>graphics.ellipse(a,b,c,d)</code>	Disegna un'ellisse di centro il punto (a,b) e di assi c,d
<code>graphics.line(a,b,c,d)</code>	Disegna una linea tra i punti (a,b) e (c,d)
<code>graphics.point(x,y)</code>	Disegna il punto (x,y)
<code>graphics.rect(a,b,c,d)</code>	Disegna il rettangolo con vertice in alto a sinistra punto (a,b) , larghezza c ed altezza d
<code>graphics.rotateX(a)</code>	Ruota una forma di un angolo a sull'asse X
<code>graphics.rotateY(a)</code>	Ruota una forma di un angolo a sull'asse Y
<code>graphics.rotateZ(a)</code>	Ruota una forma di un angolo a sull'asse Z
<code>graphics.sphere(r)</code>	Disegna una sfera di raggio r
Altri comandi	Vedi http://processing.org
LOGO	
<code>logo.show()</code>	Mostra la finestra di geometria della tartaruga
<code>logo.hide()</code>	Nasconde la finestra di geometria della tartaruga
<code>t = logo.addTurtle()</code>	Aggiunge una tartaruga t

<code>t = logo.addTurtle(c)</code>	Aggiunge una tartaruga <code>t</code> di colore <code>c</code> , es. <code>t = logo.addTurtle(color.red)</code>
<code>t.bk(s)</code>	Muove la tartaruga <code>t</code> di <code>s</code> passi indietro
<code>t.fd(s)</code>	Muove la tartaruga <code>t</code> di <code>s</code> passi avanti
<code>t.rt(a)</code>	Gira la tartaruga <code>t</code> a destra di un angolo <code>a</code>
<code>t.lt(a)</code>	Gira la tartaruga <code>t</code> a sinistra di un angolo <code>a</code>
<code>t.pu()</code>	Alza la penna della tartaruga <code>t</code>
<code>t.pd()</code>	Abbassa la penna della tartaruga <code>t</code>
<code>t.setPenColor(c)</code>	Imposta la penna al colore <code>c</code> , es. <code>t.setPenColor(color.black)</code>
<code>t.setX(x)</code>	Imposta la coordinata <code>x</code> della tartaruga <code>t</code>
<code>t.setY(y)</code>	Imposta la coordinata <code>y</code> della tartaruga <code>t</code>
<code>t.clear()</code>	Pulisce il disegno