La magia del computer

INTRODUZIONE ALLA PROGRAMMAZIONE PER ARTISTI E DESIGNER

Accademia Belle Arti Bari

Corso di Computer Art (Triennio)

prof. Antonio Rollo

THE STUYVESANT PEAR TREE—*Vide* p. 28

Un problema di teoria dei numeri è senza tempo come un opera d'arte.
– David Hilbert

Premessa

Il codice presentato nel libro è un'introduzione alla *programmazione* anche per coloro che non hanno mai visto un computer. Ho costruito un percorso di apprendimento delle conoscenze di base della scrittura di codice mettendo insieme le esperienze informatiche acquisite nel corso degli studi universitari e il lavoro degli ultimi dieci anni dedicato alla scrittura di *frammenti di intelligenza* - algoritmi - in forma di programmi - sequenze di istruzioni - comprensibili dal computer. Questi frammenti di intelligenza sono lo sguardo dello scienziato, del matematico, del fisico e dell'artista che non si ferma alla superficie dei sensi ma cerca di comprenderne i comportamenti e restituirli in forme estetiche.

Programmare è come cucinare. L'arte della cucina richiede fantasia, inventiva, creatività ed intelligenza per immaginare nuovi sapori, conoscere nuovi ingredienti, imparare nuovi procedimenti e servire per sé e per gli altri piatti squisiti. La novità richiede pratica e volontà nel superare difficoltà e fallimenti. Ci vuole un niente per bruciare una semplice frittata o confondere il sale per lo zucchero e rovinare un dolce. La programmazione, l'arte di scrivere codice comprensibile ed elaborabile dal cervello elettronico, permette di servire per sé e per gli altri prelibati programmi che quando vengono cotti a puntino regalano grandi soddisfazioni e piacere.

Lo studente italiano di materie artistiche, nei confronti del software, è come l'utente di un fast food, mangia piatti tutti uguali e non ha assolutamente accesso in cucina. Oggi la cucina del software non è racchiusa tra mura ma distribuita su internet. Le comunità di nuovi programmatori nate dopo la maturità "personale" del computer, avvenuta verso la fine degli anni ottanta ed esplosa grazie alla facilità di pubblicazione del *world wide web*, hanno iniziato a condividere frammenti di codice capaci di far funzionare un computer e di riportare in forma logica i problemi della vita reale in rapporto allo schermo interattivo. I problemi espressi in forma logica assomigliano alle ricette segrete dei grandi cuochi ed hanno un grande fascino quando ne assaporiamo il risultato finale.

Spesso un piatto realizzato da un cuoco di fama ha costi elevati, così come alcuni programmi di comune utilizzo quali Adobe Photoshop e Adobe Flash. Lo spirito del computer ha diviso il mondo digitale in processi viziosi come quelli instaurati dalla chiusura del codice (Microsoft, Apple, Adobe, ecc..) e in processi virtuosi come quelli dell'apertura del codice (Linux, GLP, ecc..). Il codice libero, spesso inteso come codice gratuito, è una filosofia che si avvicina all'idea di libertà di parola, piuttosto che alla libertà di non pagare al ristorante.

Per cucinare un buon programma è fondamentale conoscere gli ingredienti e l'ambiente in cui operare. La materia prima di un programma è il numero, nella sua forma pura di misura del mondo. Un numero, ad esempio il 10, può indicare tutte le misure che conosciamo sia del mondo reale (metri, litri, atmosfere, gradi, velocità) sia del mondo digitale (byte, pixel, frames al secondo) senza modificare la natura del numero 10. In realtà la nostra capacità di astrazione porta alcuni numeri ad avere significati metaforici (il numero dieci fa pensare ad un attaccante, oppure ad voto più alto). Il computer vive grazie alla sua capacità di interpretare soltanto due numeri: lo zero e l'uno. Le sequenze di zero ed uno formano parole in codice che il computer elabora nel suo cervello elettronico attraverso il controllo dei dati in ingresso, un'elaborazione algebrica su questi numeri e infine il risultato che viene salvato in memoria.

La memoria di un computer è come un'enorme libreria di cui ne vediamo l'etichetta ed attraverso di questa accediamo al suo contenuto. Il modo in cui i dati vengono elaborati dal computer apre all'universo della logica e della matematica applicata che ha visto personaggi come Wiener, Turing e von Neumann scrivere la storia della grande sfida lanciata alla comprensione e codifica della propria stessa intelligenza.

Quando si entra nella cucina di un artista digitale non è richiesto di conoscere come costruire la cucina stessa, ma è fondamentale comprendere la sua struttura, le sue funzionalità e iniziare a interpretare in maniera personale i problemi che una particolare ricetta richiede.

C'è un detto popolare salentino che recita "*l'ecchiu rrubba*" - l'occhio carpisce i segreti - ed è spesso usato dalle madri per trasmettere ai figli la conoscenza della preparazione di un piatto. Come in ogni cucina che si rispetti riusciamo a leggere la geografia e la storia di un territorio. Le ricette, gli ingredienti e i procedimenti sono lo specchio della cultura di una terra. I migliori cuochi del salento sono ancora gli stessi contadini che conoscono non solo l'arte di cucinare i cibi, ma possiedono anche i segreti per farli crescere e riprodurre. Una semplice frittata nasconde dietro il profumo intenso dell'uovo cotto nell'olio bollente un'aia con le galline che sgambettano alla ricerca di qualche chicco di grano o di qualche pietrolina per immagazzinare calcare. Nell'odore acre dell'uovo sbattuto c'è tutta la pazienza del pulire l'aia, di accudire un uliveto, di raccogliere le olive e di aspettare il prelibato frutto della spremitura. Oggi compriamo tutti gli ingredienti al supermercato e in cambio di denaro lasciamo che la nostra intelligenza possa essere utilizzata per cercare nuove ricette, sulla base della nostra curiosità e capacità di inventare. La conoscenza dei processi di elaborazione del computer sono proprio i segreti nascosti dietro lo schermo e oggi possiamo comprare il computer al supermercato, proprio come le uova e l'olio.

Per uno studente di un'Accademia italiana comprare un computer significa emulare il proprio compagno, collegarsi a internet per godere della pubblica piazza espressa dal web 2.0 e per dannarsi nella scrittura della tesi. Soltanto in casi molto rari viene spiegata la magia del computer e la possibilità di utilizzare il cervello elettronico come strumento intelligente a servizio dei problemi dell'arte.

Il parallelo con il cibo preconfezionato può continuare se pensiamo alle tante applicazioni che permettono di scaldare un'idea preconfezionata e servire a sé ed agli altri qualcosa di cui non si conosce l'essenza. Vengono alla mente le vignette di Mafalda che odiava la solita minestra servita da una madre vittima della rivoluzione consumistica.

La maggior parte delle applicazioni continuano ad alimentare una fantasia ricombinante che non porta all'inventiva ed all'immaginazione, ma piuttosto

impone standard visivi e modalità di interazione che limitano la creatività dell'artista.

I pionieri della Computer Art erano matematici, fisici e scienziati che avevano intravisto nei nuovi schermi collegati al cervello elettronico di metà novecento un modo per esplorare e raccontare le proprie storie e visioni digitali. Da allora ci sono in giro per il pianeta festival, rassegne e centri di ricerca in cui arte e scienza producono una narrazione digitale in cui lo spettatore diventa parte attiva e interattiva della storia. La letteratura sulle origini e sviluppi del computer è straripante di testi e manuali, riviste e pubblicazioni periodiche che rimangono nella quasi totalità in lingua inglese. La magia del computer spiegata in cucina è un tentativo di avvicinare la scienza del calcolo digitale alle forme di espressione artistica contemporanea, dove l'indagine del rapporto Uomo-Macchina ha portato sin dagli albori ad una fiorente arte che racconta i cambiamenti in corso.

L'universo aperto dall'arte con il computer ha una struttura multidimensione in cui la natura procedurale, partecipatoria, spaziale ed enciclopedica del computer si interseca con l'immaginazione dell'artista. L'apparato cognitivo è immerso in una dimensione di stimoli multi sensoriali che interferiscono con i reticoli (pattern) della visione, dell'ascolto e del tatto attraverso l'interazione con lo schermo, la tastiera, il mouse e tutte le periferiche di input ed output che sono state inventate per rendere il computer prossimo all'uomo. Certo, il computer non è in grado di innamorarsi o provare piacere per una frittata che profuma di menta fresca, ma riesce ad elaborare una quantità di dati talmente elevata che inizia ad avvicinarsi a quello necessario alla mente umana per processare emozioni intense come l'amore o il gusto. Gli scienziati stanno scommettendo sul momento in cui la macchina sarà in grado di vivere da sola nel mondo. Credo che per arrivare a questo il computer ha bisogno ancora di diverse generazioni di programmatori.

Il segno digitale rispetto alla staticità della carta e della fotografia e la dinamicità del cinema e del video, aggiunge un comportamento nuovo all'immagine che è interattiva per natura. L'immagine interattiva incorpora la bellezza del segno e la multi dimensionalità delle nuove cornici, per liberare una for-

ma in cui il contenuto può rispondere attivamente alle azioni dell'utente o dello spettatore.

La scrittura di codice permette allo studente di utilizzare con coscienza entrambi gli emisferi cerebrali. La parte destra del cervello, per natura, inter-agisce nel processo di visualizzazione di un'idea (immaginare e sentire il profumo di una fittata alla menta), mentre la parte sinistra si preoccupa soprattutto di intervenire quando ci sono da risolvere problematiche complesse (seguire un preciso procedimento che dagli ingredienti di base porta al piatto finale), come quelle relative alla programmazione ed elaborazione di immagini sintetiche con un computer.

Lo studente che si avvicina alla programmazione del computer si trova di fronte ad un processo creativo percepito come nuovo e stimolante, dove la comprensione e utilizzo del computer diventa la mappa dove orientare le proprie visioni e meglio interpretare la quotidianità della società mediocratica in cui si trova ad essere un artista. La società in cui viviamo è il risultato di un continuo movimento entropico verso equilibri che non sono più statici, come la carta, ma dinamici come lo schermo sbrillucicante del computer. Questo libro è scritto come un manuale di ricette in cui è fondamentale la pratica per apprendere i procedimenti. La cottura di un codice avviene sul fuoco digitale degli schermi interattivi che qui sono rivoltati dal fuori verso l'interno per scoprire lo spirito del computer.

Introduzione

Scrivere queste parole è semplice per uno come me abituato ad avere a che fare con la combinazione logica di simboli e codici in forma di programma informatico. Anche se scrivere un programma per il computer è molto diverso da scrivere una lettera. Eppure quando ho scoperto il *basic*, un linguaggio di programmazione, ebbi la sensazione che con il computer avrei potuto dialogare. O quanto meno quel dialogo mi avrebbe permesso di porre al cervello elettronico quelle domande di adolescente a cui nessuno riusciva a darmi una risposta. Tra le tante questioni una in particolare mi riusciva difficile addirittura da formulare, e che ancora ad oggi non sono riuscito a trovare le giuste parole per esprimerla. La domanda di una risposta di fronte all'infinito del cielo stellato. L'idea stessa di pensare al concetto di infinito mi mette in seria crisi. Tempo dopo ho scoperto che è una 'questione aperta' dell'umanità, e sono state scritte valanghe di pagine e forse qualcuno in questo momento, in un punto qualsiasi del globo, sta scrivendo parole intorno all'infinito.

Era un pomeriggio primaverile del 1987 quando, scorrendo le pagine del manuale del *basic*, ho visto per la prima volta il concetto di infinito esprimibile in pochissime parole. Le parole non erano però in linguaggio naturale, il sistema di simboli e fonemi che sto adottando per scrivere questo libro, ma in una lingua completamente nuova che permette di dialogare con un cervello elettronico. Nel 1987 a Castri di Lecce la parola internet non esisteva. Castri di Lecce è una delle cento molecole urbane che animano una lingua di terra che si stende tra due mari a Sud Est d'Italia, il tacco dello stivale. Un territorio singolare per la sua storia, cultura e tradizioni dove ho trascorso i miei primi diciotto anni immune al progresso, tanto che la stessa lingua italiana era qualcosa di altro rispetto al dialetto leccese, la lingua che si usava correntemente per comunicare. Terra singolare dicevo. A meno di tre chilometri, in una delle molecole limitrofe si parlava addirittura il *griko*, una forma di greco, e lo stesso dialetto leccese aveva sfumature e parole diverse da molecola in molecola. Vivevo in un microcosmo prevalentemente agricolo, dove le parole servivano per comunicare le *cose semplici* della vita.

Un vocabolario che permette di pensare domande come quelle sull'infinito, ma che si esaurisce nell'avvicinarsi alla risposta, mutando in poesia. Il poeta Bodini, figlio di questa terra, ce la trasmette in una terzina, come il fotogramma chiave di un film. *Viviamo in un incantesimo / tra palazzi di tufo / in una grande pianura (Bodini)*. Questo continua ad essere vivo anche oggi che si inizia a scrivere tra le pieghe dei secolari ulivi la parola *internet*. Una cultura che si insinua nei meandri del corpo e te la porti dentro. Ero parte di una cultura locale il cui linguaggio naturale non soddisfaceva appieno alle domande che volevo formulare.

Il manuale del *basic* introduceva ad un semplice *linguaggio di programmazione per computer* facile da imparare e da usare, così come recitava nelle prime pagine. Le pagine scorrevano una dietro l'altra tra istruzioni in inglese e notazioni matematiche. Dell'inglese sapevo soltanto che era l'altra lingua che avrei potuto studiare al liceo al posto del francese. Il linguaggio basic doveva essere caricato nella memoria del computer e risiedeva su una normale cassetta audio a nastro magnetico.

Ma cosa viene registrato su quel nastro che, messo nello stereo, emette suoni strampalati? La risposta era facile. Un linguaggio.

Mi innamorai di quelle nuove parole che svolgevano ognuna una precisa funzione. In particolare l'istruzione/parola GOTO era affascinante. ANDARE. Si, ma dove? Ogni istruzione del basic è preceduta da un numero che indica la riga dell'istruzione. L'istruzione GOTO è un salto incondizionato al numero di riga indicato. Ecco la risposta alla domanda adolescenziale di infinito. ANDARE. Scrissi 10 GOTO 10. Il computer rimase fermo, ma sapevo che qualcosa dentro si stava muovendo. Avevo scritto il mio primo programma.

Lo scopo di questo libro è svelare alcune magie dei linguaggi di programmazione che hanno dato vita a tante forme d'arte con il computer. Il linguaggio utilizzato per l'elaborazione dei codici è *Processing*, un ambiente di sviluppo di applicazioni per ogni tipo di computer, oltre che essere un software distribuito in licenza open source.

Il codice presentato in questo libro è un'introduzione alla *programmazione* anche per coloro che non hanno mai visto un computer.

Ho costruito un percorso di apprendimento delle conoscenze di base della scrittura di codice mettendo insieme le esperienze informatiche acquisite nel corso degli studi universitari e il lavoro degli ultimi dieci anni dedicato alla scrittura di *frammenti di intelligenza* - algoritmi - in forma di programmi - sequenze di istruzioni - comprensibili dal computer. Questi frammenti di intelligenza sono lo sguardo dello scienziato, del matematico, del fisico e dell'artista che non si ferma alla superficie dei sensi ma cerca di comprenderne i comportamenti e restituirli in forme estetiche. In altre parole, scrivere un programma in linguaggio comprensibile dal computer è riuscire a socchiudere gli occhi e vedere le strutture e i processi che reggono buona parte della realtà che ci circonda. Il codice non è mai qualcosa di statico.

Per vivere il computer utilizza i numeri. Scoprire come i numeri si nascondo nella struttura intima della natura del pensiero logico è un viaggio per acquisire nuovi occhi sul mondo. Scopriamo come un linguaggio simbolico basato sul vuoto e sul pieno, sul vero e sul falso, sull'accesso e sullo spento, sul lineare e sull'ondulato, è la forma perfetta per dialogare con il computer. Il programma è un insieme di istruzioni che muovono numeri variabili, prendono decisioni di direzione, generano casualità inattesa, architettano ordine e misurano infinite combinazioni.

La storia del computer è una storia relativamente breve. Se pensiamo che i primi computer così come siamo abituati a percepirli, ovvero con uno schermo, un mouse e una tastiera, hanno visto la luce verso la fine degli ottanta e se pensiamo all'incredibile accelerazione che ha generato in ogni ambiente umano, dall'economia alla medicina, dall'architettura alla genetica, dall'idea di internet - rete - alla nuova percezione globale del mondo, allora possiamo capire che il contesto in cui nasceva la computer art era molto diverso dal momento storico che stiamo vivendo.

Il computer ha introdotto nelle arti uno stato dinamico che possiamo paragonare al rapporto fotografia-cinema, con la differenza sostanziale che il cinema non è interattivo. L'interattività presuppone una comunicazione bidirezionale. Faccio un esempio, tanto per chiarire.

Una persona è interattiva.

Quando parlo con qualcuno, sto interagendo con l'altra persona, nel senso che la comunicazione bidirezionale che si instaura vede una partecipazione nel tempo di entrambi. A volte succede che una persona può cambiare la vita grazie ad una parola, questo è vero. La chiave dell'interazione sta nei processi di partecipazione e nella capacità di ascoltare e rispondere. Dal momento in cui si capì che il computer aveva la capacità di essere interattivo, intorno agli anni cinquanta, il nuovo medium universale (così chiamato dall'idea di macchina universale di Alan Turing) ha permesso di progettare e costruire macchine che portano essere umani nello spazio, così come nelle profondità degli abissi, ma anche macchine concettuali, meglio conosciute come software, che permettono di trasformare il computer in qualunque idea che si riesca a formalizzare in un linguaggio comprensibile dalla macchina stessa.

Il computer è uno strumento completamente diverso da tutti quelli inventati e prodotti dall'uomo. Ogni strumento tradizionale, *pre elettrico*, è sempre stato un'estensione del *corpo*. La ruota estende la capacità di camminare e usare le gambe. Il pennello estende la mano fornendo la capacità di selezionare colori e lasciare tracce su una superficie. Le lenti ottiche estendono i nostri occhi verso le stelle o nell'intimo microscopico, oltre che aiutarci a ristabilire nel caso degli occhiali un errore nella vista. Ogni strumento *pre elettrico* è rapportato al *corpo sensibile*, alle sue funzioni e alle sue azioni nell'ambiente in cui vive. Il computer, uno strumento *post elettrico*, è l'estensione della mente umana, nel senso che attraverso il suo linguaggio formale ci permette di pensare nuove idee, di visualizzare nuovi processi e di inventare esperienze e nuovi livelli di comunicazione. Questo non è altro che un processo artistico. La dinamicità introdotta dal computer nelle arti è qualcosa di completamente nuovo, ancora da esplorare e in parte da costruire.

Capitolo 1
Processing e la programmazione

Processing è un linguaggio di programmazione open source ed un ambiente per persone che vogliono creare immagini, animazioni ed interazioni. Sviluppato inizialmente come un quaderno di appunti per la scrittura di software e per insegnare i fondamenti informatici della programmazione in un contesto visivo, Processing si è evoluto anche come uno strumento per generare lavori professionali. Oggi, ci sono decine di migliaia di studenti, artisti, designer, ricercatori ed appassionati che usano Processing per l'insegnamento, la prototipazone e la produzione.

– www.processing.org

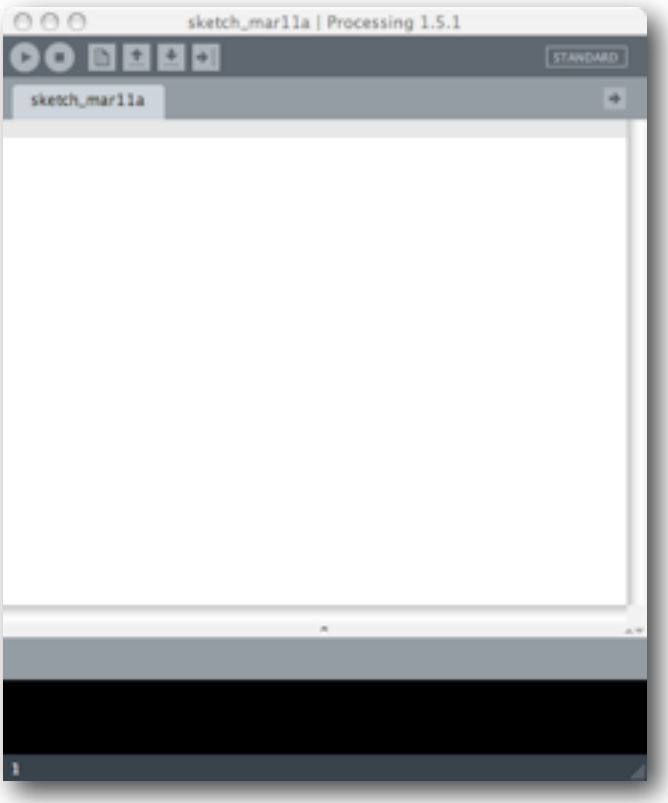
Download ed installa

Prima di iniziare a conoscere il linguaggio di programmazione abbiamo bisogno di scaricare l'applicazione *Processing* dal sito *www.processing.org* ed installarla sul nostro computer. Sono state rilasciate le versioni per i tre più importanti sistemi operativi: Windows, Apple OS X e Linux. Il sito contiene dettagliate istruzioni per installare Processing sul nostro computer.

Il progetto di scrivere quest'applicazione, che si basa su tecnologia Java, nasce nel gruppo di Estetica e Calcolo del Mit Media Lab con l'intenzione didattica di insegnare i fondamenti dell'informatica in un ambiente di ricerca visiva. Iniziato da Casey Reas e Benjamin Fry nel 2001 è attualmente alla versione 2, la stessa utilizzabile per eseguire gli esercizi proposti in questo libro. Ogni volta che apriamo - *mandiamo in esecuzione* - l'applicazione *Processing*, il nostro computer diventa una macchina multimediale di cui possiamo controllare attraverso un linguaggio simbolico processi visivi, sonori ed interattivi. Ad esempio possiamo scrivere il codice che fa partire un video, controllare attraverso la voce le proprietà di un oggetto infografico, disegnare su una superficie spazio temporale.

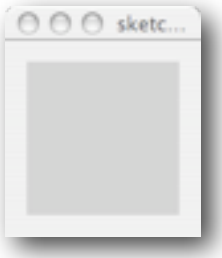
Sketchbook

Lo *sketchbook* è letteralmente il *quaderno degli appunti* e degli abbozzi, lo *sketchbook* di Processing è un quaderno degli appunti e degli abbozzi di codice che andremo a scrivere insieme, a salvare, a rivedere, a condividere. Un diario digitale scritto in un linguaggio di cui molto presto scopriremo le sue magie.

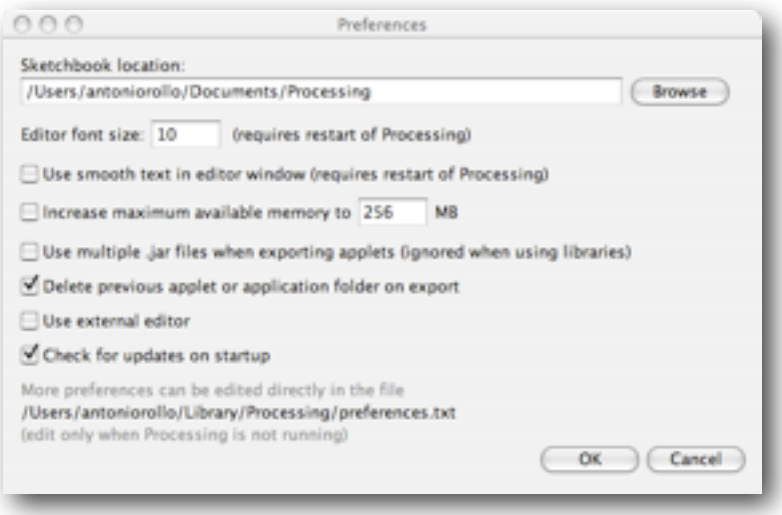


L'ambiente di programmazione Processing si presenta in forma molto semplice e la complessità è nascosta dentro la pagina bianca. L'interfaccia mostra sei icone principali che permetto di: mandare in esecuzione il codice, fermarlo, creare un nuovo foglio, aprirne uno esistente, salvare quello in corso. Sotto la pagina bianca troviamo il pannello di controllo che possiamo utilizzare per stampare, in fase di esecuzione, i nostri valori da monitorare. Infine, la barra finale ci indica la riga della pagina bianca in cui andremo a scrivere il codice.

Anche se non scriviamo codice, quando mandiamo in esecuzione il foglio bianco si apre una finestra di *default*. Questo è quello che chiamo il nostro *palcoscenico digitale*.



Dalle preferenze del programma è consigliabile localizzare sin da subito la memoria dei nostri *sketchbook*. La posizione di default che Processing utilizza per salvare i nostri lavori è nella cartella Documenti, dove crea in fase di installazione una cartella nominata Processing. Possiamo comunque decidere la posizione sul nostro hard disk dalle preferenze del programma.



Struttura di un programma

Un programma è composto da un insieme di dati e istruzioni che implementano in maniera sequenziale un *algoritmo*. Un algoritmo è per definizione un insieme di *istruzioni elementari* (univocamente interpretabili) che, eseguite in un ordine prestabilito, permettono la soluzione di un problema in un numero finito di passi. Il termine *algoritmo* deriva dal nome del matematico persiano *Muhammad ibn Mūsa 'l-Khwārizmī*, che si ritiene essere uno dei primi autori ad aver fatto riferimento a questo concetto nei suoi studi sulla natura dei numeri.

La struttura generale di un programma può seguire due modelli implementativi che sono:

- programmazione procedurale
- programmazione orientata ad oggetti

Il linguaggio di programmazione Processing permette di sviluppare le nostre applicazioni sia secondo il modello procedurale, come faremo nella prima parte del libro, sia secondo il modello orientato ad oggetti, che utilizzeremo per la scrittura dei codici nella seconda parte.

Secondo il paradigma procedurale la struttura di un programma è composta da un insieme di dati che vengono definiti come *variabili globali*, ovvero valori in memoria che possono essere letti e scritti in ogni punto del programma, variabili locali, che possono visibili solo all'interno di una dichiarazione di funzione, e infine l'*insieme di funzioni* che racchiudono le sequenze di istruzioni che elaborano l'algoritmo per la soluzione al problema che abbiamo deciso di ricercare. Le istruzioni sono eseguite in ordine sequenziale dalla prima all'ultima.

Variabili

Sappiamo che nel computer tutti i dati sono riconducibili al codice binario 1/0, ma questo succede ad un livello che a noi utenti è oscuro, e tale può restare almeno per il momento. Sappiamo invece che ci sono almeno due tipi di informazioni che scambiamo come dati con il computer: i numeri e le parole.

I numeri sono qui espressi da due tipi di dati:

```
int  // per indicare i numeri senza la virgola
float // per indicare i numeri con la virgola
```

I più comuni numeri interi - *int* - sono quelli che usiamo per indicare il tempo (es. 12:32:27) oppure la posizione (è al 3° posto), mentre quelli con la virgola - *float* - saltano fuori quando cominciamo a dividere l'unità (½ = 0.5). Sarà quindi nostra cura capire prima che tipo di numero utilizzerò nei miei progetti. In un programma i dati non sono quasi mai fissi, nel caso lo fossero sono chiamati *costanti*, altrimenti sono chiamati *variabili*. Costanti e variabili di un programma sono come i cassetti di una biblioteca dentro i quali possiamo mettere le nostre informazioni e per riconoscerli appendiamo un'etichetta fuori.

Allo stesso modo scrivo la relazione che istruisce il programma nella creazione di una variabile numerica di tipo intero:

```
int x = 10;
```

Dire che *x* vale dieci, significa capire cosa vale *x* per noi. Può essere ad esempio una posizione geografica, le dieci del mattino, ecc… Se nel nostro programma *x* vale sempre dieci, la tratteremo come una *costante*, se invece *x* vale dieci solo all'inizio, e poi varia in fase di elaborazione, la tratteremo come una *variabile*.

Le parole invece sono qui espresse secondo le due tipologie:

```
char  // per indicare un qualsiasi carattere
String // per indicare un insieme di caratteri
```

Ad esempio:

```
char c = "A";
String parola = "ABCD";
```

Ricordo che il punto e virgola è sempre necessario per far capire a Processing quando termina un'istruzione. Ci sono tre tipi di istruzioni: le dichiarazioni di *variabili*, come quella appena scritta, le dichiarazioni di *funzioni* e le istruzioni di sistema.

Disegnare con i numeri

La struttura base di un programma in Processing è definita da due funzioni di base, una in cui scriviamo le istruzioni per allestire il palcoscenico - *setup()* - e l'altra in cui scriviamo il codice per disegnare - *draw()*.

```
void setup() {
        // lista istruzioni
}


void draw() {
        // lista istruzioni
}
```

La funzione *setup()* è eseguita dal programma una volta sola, all'inizio dell'elaborazione, la funzione *draw()* invece è eseguita all'infinito, almeno finché c'è un computer accesso che lo permette. Quindi è più corretto dire che una funzione può essere eseguita da un computer quante volte si vuole, oppure fino a quando si decide di farlo girare (loop). In realtà questa esecuzione non riguarda solo le funzioni di Processing, ma è alla base dell'intero funzionamento dei computer. Quanto tempo ci mette il computer ad eseguire un'operazione? Possiamo dire con certezza che siamo ormai a frazioni di miliardesimo di secondo. Per calcolare a mano la stessa quantità di informazione con carta e penna ci sarebbero voluti oltre dieci anni.

Palcoscenico digitale

Eseguire operazioni numeriche significa imparare un nuovo linguaggio per esprimere la nostra immaginazione. Il palcoscenico digitale di Processing è un ambiente che può essere programmato per rispondere a diversi media di comunicazione e rappresentazione di concetti ed emozioni. Possiamo disegnare immagini in bianco e nero e a colori, possiamo esplorare lo spazio cartesiano sia sul piano attraverso funzioni 2D, sia nello spazio tridimensionali grazie al supporto di funzioni 3D, possiamo controllare l'interazione con il mouse e la tastiera, possiamo manipolare immagini, possiamo sincronizzare audio e video.

L'ambiente di programmazione Processing essendo open source ha liberato la creatività di molti sviluppatori che hanno aggiunto classi di funzioni che permettono di programmare dallo spazio sonoro allo spazio fisico con l'uso di particolari interfacce come *Arduino*.

Le funzioni di base che controllano la dimensione dello schermo ed il colore di sfondo del nostro palcoscenico digitale sono:

```
size(altezza, larghezza);

background (colore);
```

Queste due istruzioni sono da scrivere nella funzione setup(), ad esempio immaginiamo di avere uno schermo quadrato di *500 pixel* con sfondo nero:

```
void setup() {
        size(500, 500);
        background (0);
}
```

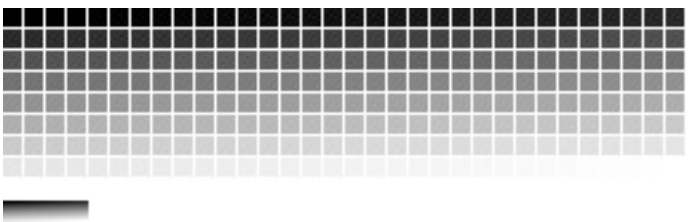
Prima di proseguire ricordo che un *pixel* è l'unità minima dell'informazione di uno schermo digitale. Un pixel è il punto luminoso, la risoluzione di uno schermo è il risultato della moltiplicazione dell'altezza per larghezza espressi in numero di pixel.

Ad esempio la risoluzione dello schermo quadrato di 500 pixel è 500x500 uguale a 250.000 pixel.

Ogni punto luminoso può avere un colore, in particolare se vogliamo che lo sfondo del nostro schermo sia nero, dobbiamo prima comprendere come utilizza i colori il computer.

Un moderno computer permette di utilizzare sedici milioni di colori attraverso la combinazione delle tre componenti principali della luce, ovvero il Red, Green e Blu, chiamati appunto colori RGB. Approfondiremo in seguito le modalità dei colori in Processing, per il momento limitiamoci a lavorare solo con 256 toni di grigio, sapendo che *0 è nero* e *255 è bianco*.

Di seguito sono visualizzati tutti i 256 toni di grigio controllabili da Processing:



Primitive grafiche

I punti, le linee e le figure geometriche che delimitano una superficie del piano hanno ispirato gli artisti di ogni epoca e parte del mondo. Caravaggio, in Italia, dipingeva secondo precisi schemi geometrici intorno ai quali emergevano i suoi racconti di luce e ombra. Mirò, in Spagna, sintetizzava forme e colori secondo precisi pattern visivi modulari. Kandinski, in Russia, formalizzava il pensiero geometrico in tele che rappresentano stati interiori, dando peso e informazione al segno geometrico, come lui stesso scrive: "Il punto geometrico è un'entità invisibile. Deve quindi essere definito come un'entità immateriale. Pensato materialmente, il punto equivale a uno zero. Ma in questo zero si nascondono diverse proprietà, che sono umane.

Noi ci rappresentiamo questo zero - il punto geometrico - come associato alla massima concisione, cioè con un estremo riserbo, che però parla. In questo modo, nella nostra rappresentazione, il punto geometrico è il più alto e assolutamente l'unico legame tra silenzio e parola. [...] La linea è un'entità invisibile. È la traccia del punto in movimento, dunque un suo prodotto. Nasce dal movimento - e precisamente dalla distribuzione del punto, della sua quiete estrema, in se conchiusa. Qui si compie il salto dallo statico al dinamico".

Immaginare una linea come la traccia del movimento di un punto è qualcosa che si è materializzato con l'avvento delle apparecchiature elettroniche, come gli oscilloscopi agli albori della computer art fino alle ultime tecnologie di proiezione olografica.

Processing mette a disposizione un'insieme di primitive grafiche per il disegno di punti, linee e figure geometriche. Queste primitive grafiche sono delle funzioni con parametri che vediamo essere molto intuitivi:

Punto

Per disegnare un punto nello spazio ho bisogno delle sue coordinate cartesiane:

```
point (x, y);
```

Per disegnare un punto bianco al centro di uno schermo nero di 100x100 pixel posso scrivere:

```
void setup() {
        size(100, 100);
        background (0);
        stroke(255); // assegna il colore bianco
        point(50,50); // disegna il punto
}
```

L'istruzione s*troke(colore)* applica il colore scelto ad ogni contorno di una primitiva grafica.

Linea

La geometria ci ha insegnato che per un punto passano infinite linee, e che per due punti passa una ed una sola linea, quindi per disegnare una linea nello spazio ho bisogno delle coordinate dei due punti di inizio e fine:

```
line (x1, y1, x2, y2);
```

Per disegnare una linea bianca, di spessore 2 pixel, come diagonale di uno schermo nero di 100x100 pixel scrivo:

```
void setup() {
        size(100, 100);
        background (0);
        stroke(255); // assegna al punto il colore bianco
        strokeWeight(2); // spessore della linea di contorno
        line(0,0,100,100); // disegna il punto al centro dello schermo
}
```

L'istruzione *strokeWeight(spessore)* permette di definire lo spessore di una linea di contorno.

Triangolo

La più semplice figura geometrica per descrivere una superficie chiusa è il triangolo. Per disegnare un triangolo abbiamo bisogno di tre punti nello spazio:

```
triangle (x1, y1, x2, y2, x3, y3);
```

I tre vertici del triangolo definiscono una figura formata dalla linea di contorno e dalla superficie interna. Se vogliamo che la superficie interna sia trasparente dobbiamo utilizzare l'istruzione *noFill()* come segue:

```
void setup() {
        size(100, 100);
        background (0);
        stroke(255); // assegna al punto il colore bianco
        strokeWeight(2); // spessore della linea di contorno
        noFill();
        triangle(50,0,0,80,100,80);
}
```

Se invece vogliamo colorare la superficie interna con un tono di grigio utilizziamo l'istruzione *fill(colore)*:

```
void setup() {
        size(100, 100);
        background (0);
        stroke(255); // colore contorno
        strokeWeight(2); // spessore
        fill(128); // colore superficie interna
        triangle(50,0,0,80,100,80);
}
```

Quadrangolo

Quando disegniamo un quadrangolo sappiamo già che abbiamo bisogno di quattro punti:

```
quad(x1, y1, x2, y2, x3, y3, x4, y4);
```

Ci sono poi quadrangoli speciali come in *quadrato* ed il *rettangolo* che possono essere disegnati nello spazio a partire da un punto d'inizio e dalla misura dei due lati, ovviamente nel caso del quadrato i due lati sono uguali:

```
rect(x, y, lato1, lato2);
```

```
void setup() {
        size(100, 100);
        background (0);
        stroke(255); // colore contorno
        strokeWeight(2); // spessore
        fill(128); // colore superficie interna
        rect(10,10,80,80);
}
```

Ellisse

Per disegnare un'ellisse abbiamo bisogno della posizione del centro e della lunghezza delle due diametri. Un cerchio è praticamente un'ellisse ma con i diametri uguali:

```
ellipse (x, y, diametro1, diametro2);
```

Quindi se vogliamo disegnare un cerchio dobbiamo impostare un punto di origine del centro e i due diametri della stessa lunghezza:

```
void setup() {
        size(100, 100);
        background (0);
        stroke(255); // colore contorno
        strokeWeight(2); // spessore
        fill(128); // colore superficie interna
        ellipse(50,50,80,80);
}
```

Se i due diametri sono diversi otteniamo un'ellisse:

```
void setup() {
        size(100, 100);
        background (0);
        stroke(255); // colore contorno
        strokeWeight(2); // spessore
        fill(128); // colore superficie interna
        ellipse(50,50,70,90);
}
```

Istruzioni di sistema

I linguaggi di programmazione per computer si basano su una logica che ha le radici nella natura del computer stesso. Il calcolatore elettronico si comporta come un interprete di istruzioni logiche che permettono di costruire, a partire da un insieme di *istruzioni di sistema,* la base concettuale per sviluppare programmi che rispondono ad algoritmi più complessi. Le istruzioni di sistema sono comuni a tutti i linguaggi di programmazione, iniziamo a vedere:

```
la condizione if
il ciclo for
```

Condizione if

Nella nostra esistenza quotidiana ci troviamo spesso di fronte a delle scelte da compiere. Le nostre scelte sono condizionate da una serie di fattori che ci portano a prendere una determinata strada piuttosto che un'altra. Ci sono scelte importanti come il matrimonio e scelte meno importanti come l'acquisto di un abito. In entrambi i casi siamo noi a decidere la direzione da prendere valutando i fattori in gioco, a volte le nostre scelte sono condizionate da elementi logici, altre da elementi emotivi, altre ancora da entrambi. Una condizione è verificabile da un programma per computer soltanto in forma logica. Quando prendiamo una decisione, spesso valutiamo anche l'alternativa, per meglio comprendere le opportunità che si presentano. Allo stesso modo quando scriviamo il codice di un algoritmo che risolve problemi altrettanto importanti, come può essere un'equazione che determina i punti di rottura di un ponte o la pressione di una valvola cardiaca, è necessario, in fase di risoluzione valutare le scelte che il programma deve compiere.

L'istruzione *if* è una delle più importanti istruzioni di sistema per un linguaggio di programmazione. Letteralmente è la traduzione della particella condizionale *se* si verifica una certa condizione logica *allora* esegui determinate istruzioni, *altrimenti* sono da eseguire delle altre. Questo permette di svi-

luppare una logica ad albero, in cui il programma può prendere diverse strade a seconda che si verifichino o meno determinate condizioni.

Le condizioni logiche sono il confronto tra due entità attraverso l'utilizzo degli *operatori logici*.

```
if (condizione logica) {
insieme di istruzioni;
}
```

Ciclo for

L'istruzione *for* esegue un insieme di istruzioni, delimitate da parentesi graffe, fino a quando il valore iniziale non raggiunge la condizione di uscita. Ad ogni ciclo il valore iniziale è incrementato (o decrementato) di un fattore costante.

```
for (valore iniziale; condizione di uscita; incremento) {
        //insieme di istruzioni
 }
```

Ad esempio se voglio contare da zero fino a dieci e stampare il risultato nel pannello di controllo, posso scrivere:

```
for (int i=0; i <= 10; i++) {
        print(i);
 }
```

Random

Nel 1931, all'alba del computer, Alan Turing nel celebre saggio *Sui numeri calcolabili immagina* "che il calcolatore digitale contenga un generatore di numeri a caso" e ci accompagna in territori del calcolo che portano alla percezione extrasensoriale. Quella che nel 1931 era solo un'ipotesi di generatore di numeri casuali, ben presto nei primi computer si realizzò nella funzione *random()*.

Utilizziamo la funzione *random()* per lasciare la scelta al computer, per farci sorprendere, scopriremo che la casualità inserita nel codice infonde vita ai nostri progetti.

La funzione *random()* può essere chiamata sia senza parametri, sia con parametri e comunque restituisce sempre un numero *float*. Quindi se vogliamo chiedere un numero a caso al computer ecco come possiamo scrivere il nostro codice e cosa aspettarci:

```
//chiamata senza parametri
float r = random(); // r è un numero a caso tra 0 e 1


//chiamata con un parametro
float r = random(10) // r è un numero a caso tra 0 e 10;

//chiamata con due parametri
float r = random(10,20) // r è un numero a caso tra 10 e 20;
```

Dichiarazione di funzioni

Le istruzioni del linguaggio in realtà sono simili alle funzioni che possiamo scrivere noi stessi. Basta pensare che alle origini del computer intorno agli anni cinquanta tutte le funzioni che oggi rendono così semplice il computer ancora dovevano essere scritte.

Una funzione può essere con o senza parametri a seconda di quello che deve svolgere. Una funzione senza parametri è indicata dal tipo di funzione, dal nome della funzione, da due parentesi tonde che contengono nel caso i parametri, da due parentesi graffe che racchiudono le istruzioni da eseguire quando la funzione viene chiamata.

Un insieme di istruzioni può essere racchiuso in una funzione per poter essere invocato all'interno del programma quando è necessario. Le funzioni vengono quindi prima definite secondo la sintassi del linguaggio e poi chiamate al momento opportuno.

Le variabili dichiarate all'interno di una funzione - variabili locali - non sono visibili nell'ambiente globale e quindi possono avere anche lo stesso nome di una variabile globale. Per definire una funzione si utilizza l'istruzione di base *function*. Ecco la sintassi della dichiarazione di funzione:

```
tipo nomeFunzione (parametro) {
       // lista di istruzioni
 }
```

Per chiamare la funzione in un punto del programma basta scrivere il nome e passare i valori dei parametri se necessario.

```
nomeFunzione (valore);
```

Una funzione si comporta in due modi che definiscono il tipo. Una funzione può essere dichiarata *void* - vuota - se non ritorna nessun valore, può essere dichiarata con i tipi delle variabili, se alla fine dell'elaborazione ritorna il tipo di valore dichiarato. Questo può sembrare complicato ad una prima lettura, ma nel corso della scrittura dei nostri codici impareremo a familiarizzare con questi nuovi concetti.

Commenti al codice

Ogni buon programmatore aggiunge sempre dei commenti al codice che va scrivendo. Questo aiuta sia nel tenere un diario di implementazione, sia nel rendere più comprensibile ad altri la logica adottata.

Per aggiungere un commento si utilizzano le due forme

```
// per commentare una riga

/* per commentare
   più righe */
```

Capitolo 2
Hello World!

Per un programmatore l'inizio è sempre stampare sullo schermo "Hello World!" per un artista visivo e un designer è stampare sullo schermo un "Cielo stellato".

Codice: Cielo stellato

"*Ogni lungo viaggio inizia con un singolo passo*" diceva Lao-Tzu, quindi è arrivato il momento di aprire Processing e intraprendere la scoperta di uno dei linguaggi che stanno ridefinendo le modalità e le forme dell'interazione con lo schermo. Saliamo su un tappeto volante e dirigiamoci verso la prima meta.

Ci sono delle notti di luna nuova in cui, lontano dalle luci della città, il cielo esplode in una miriade di stelle.

Quanto è bello!

Quante stelle ci sono in cielo?

Quale posto occupa una stella nel firmamento?

Concetti come infinito e posizione hanno animato la storia dell'umanità sin dalle origini del linguaggio e dei segni. La geometria, la scienza che misura la terra, ci insegna che in un piano, la posizione di un *punto* è definita dalle *ascisse x* e dalle *ordinate y*. Per il momento restiamo sul piano bidimensionale, anche se nella seconda parte del libro affronteremo anche la geometria nella terza dimensione della *profondità z*.

Una delle operazioni da eseguire una sola volta, e quindi da inserire nella lista delle istruzioni della funzione *setup(),* è la scelta della *dimensione* del nostro palcoscenico digitale. Questa scelta è condizionata dalla risoluzione dello schermo che abbiamo a disposizione. La risoluzione è il risultato della moltiplicazione tra la quantità di pixel, il punto luminoso, delle x e delle y del nostro schermo. Le ultime generazioni di schermi propongono risoluzioni con una decina di milioni di pixel, a noi, per cominciare, basta un palcoscenico più ridotto, diciamo un quadrato di 500x500 pixel, su cui sperimentare i nostri codici.

Siamo i registi di un mondo in cui i numeri sono gli attori che si esibiscono sul palcoscenico digitale.

```
void setup() {
        // dimensione del palcoscenico digitale
        size (500, 500);
}
```

Adesso vogliamo sicuramente colorare lo sfondo. Prima di passare a conoscere i colori digitali, accontentiamoci di 256 toni di grigio, dal bianco che ha il valore numerico 255, al nero che vale 0. Ecco come un dato diventa informazione!

```
void setup() {
        // dimensione del palcoscenico digitale
        size (500, 500);
        // sfondo nero
        background (0);
}
```

Prova subito questo codice ed accingiamoci a disegnare le stelle come punti luminosi che si accendono su questo nuovo palcoscenico digitale. Per accendere un punto sullo schermo (da questo momento chiamerò la finestra dello *sketchbook* sia schermo che palcoscenico) basta dirlo con l'istruzione:

```
point (x, y);
```

Per accenderlo in una posizione ben precisa, ad esempio il centro dello schermo, occorre passare le informazioni di posizione all'istruzione *point (x, y)*, più precisamente definiamo prima le due variabili x ed y come interi a cui assegniamo il valore 250. Poi coloriamo il punto di bianco con l'istruzione *stroke(255)*. E se la stella la voglio più luminosa? Abbiamo a disposizione l'istruzione che permette di controllare la grandezza di un punto: *strokeWeight(dimensione in pixel)*.

```
void setup() {
        // dimensione del palcoscenico digitale
        size (500, 500);
        // sfondo nero
        background (0);
        // posizione della stella
        int x = 250;
        int y = 250;
        // coloriamo il punto di bianco
        stroke (255);
        // dimensione del punto 3 pixel
        strokeWeight (3);
        // prima stella
        point (x, y);
}
```

Una sola stella non è sufficiente a disegnare un cielo credibile, anche se non vogliamo realizzare una mappa precisa della volta celeste ma piuttosto disegnare un numero di stelle adeguato affinché il nostro palcoscenico possa quantomeno rassomigliare ad un cielo stellato.

Per decidere la posizione delle nostre stelle, i punti luminosi, possiamo utilizzare una funzione molto particolare che fa parte del corredo in dote a tutti i linguaggi di programmazione: la funzione *random()*.

Se chiedo di pensare un numero a caso, quale numero ci viene in mente? Indipendentemente dalla risposta, il nostro cervello alla domanda di casualità si attiva per risponde abbastanza a caso. Anche il computer si attiva per rispondere con un numero abbastanza a caso quando chiamiamo la funzione *random()*. Certo che scegliere tra un insieme infinito, come quello di tutti i numeri non è proprio impresa facile, meglio è se chiediamo un numero da estratto a caso in un insieme limitato, come può essere:

```
float x = random(100);
```

che assegna ad *x* un numero con la virgola - *float* - tra 0 e 100.

Abbiamo visto in precedenza che un programma scritto con Processing si compone di due parti, una eseguita solo all'inizio ed un'altra eseguita all'infinito. Spesso avere a che fare con l'infinito può essere molto affascinante, ma non proprio tanto comodo in termini di calcolabilità e controllo dei risultati. Proviamo a scrivere le istruzioni di scelta casuale sia della posizione del punto luminoso, sia della sua dimensione in pixel, all'interno della funzione *draw()*.

```
void draw() {
        // posizione di una stella
        float x = random(500);
        float y = random(500);
        // coloriamo il punto di bianco
        stroke (255);
        // dimensione del punto 3 pixel
        strokeWeight (3);
        // stelle
        point (x, y);
}
```

Mandando in esecuzione questo codice, dopo qualche minuto, lo schermo nero viene completamente saturato dai punti bianchi, e di certo non rassomiglia ad un cielo stellato. Abbiamo ancora bisogno di una qualche istruzione per condizionare il momento in cui fermarci. Possiamo utilizzare sia il ciclo *for* che la condizione *if*.

Iniziamo con la condizione if che ci aiuta a confrontare logicamente due valori. I valori che vogliamo confrontare sono nel nostro caso un indice che conta le stelle e il numero di stelle che vogliamo disegnare.

```
int i = 0;
int maxStelle = 100;
```

Queste due variabili quando sono definite fuori dalle funzioni di base setup() e draw() si comportano come variabili globali, ovvero possono essere lette e scritte dalle nostre istruzioni, in qualunque punto del programma. Pertanto possiamo scrivere la condizione che controlla il disegno dei punti luminosi:

```
se (i è minore di maxStelle)
        disegna il punto lumiso
        incrementa l'indice i di uno
```

Che tradotto in codice diventa:

```
if (i < maxStelle){
        // stelle
        point (x, y);
        // incremento
        i = i + 1;
}
```

Ecco il programma completo, da cui è stata eliminata la prima stella disegnata nella funzione setup().

```
void setup() {
        // dimensione del palcoscenico digitale
        size (500, 500);
        // sfondo nero
        background (0);
}
```

```
// variabili globali
int i = 0;
int maxStelle = 100;

void draw() {
        // posizione di una stella
        float x = random(500);
        float y = random(500);
        // coloriamo il punto di bianco
        stroke (255);
        // dimensione del punto 3 pixel
        strokeWeight (3);
        // condizione di uscita
        if (i < maxStelle){
              // stelle
              point (x, y);
              // incremento
              i = i + 1;
        }
}
```

Vediamo adesso come possiamo scrivere l'algoritmo del cielo stellato utilizzando il ciclo *for*.

```
finchè (l'indice i  è minore di maxStelle  incrementa i)
        disegna una stella
```

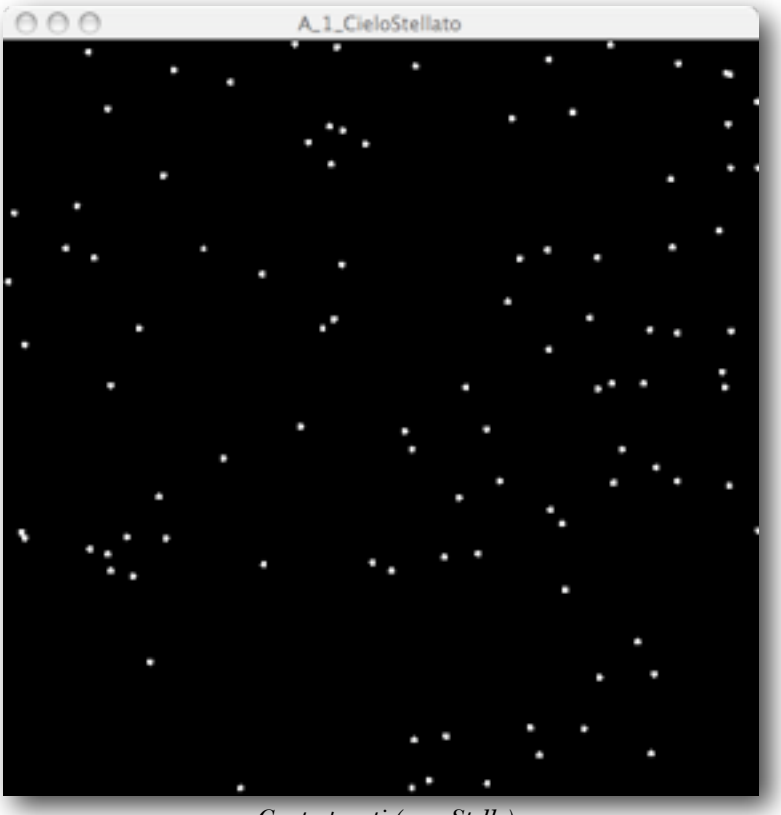
Che diventa in codice:

```
for (i=0;  i < maxStelle; i=i+1) {
// disegna un punto luminoso
}
```

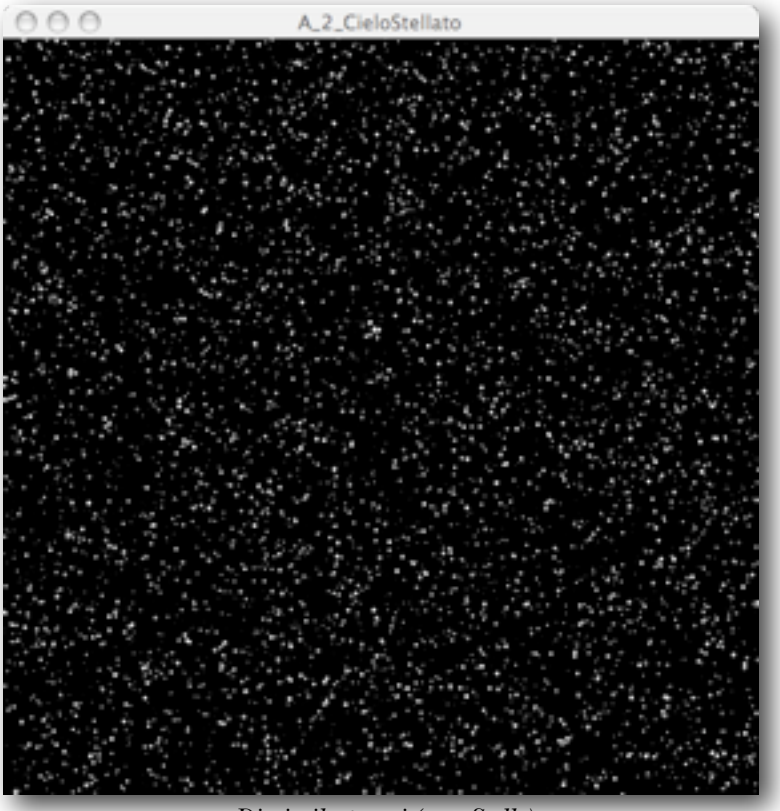
Per disegnare una stella utilizzato le seguenti istruzioni, con la funzione *random()* applicata sia al tono di grigio, sia alla grandezza del punto.

```
// posizione casuale di una stella
float x = random(500);
float y = random(500);
// coloriamo il punto con toni di grigio casuali
stroke (random(255));
// dimensione del punto 3 pixel
strokeWeight (random(3));
// stelle
point (x, y);
```

Questo codice va poi inserito all'interno del ciclo *for* per essere eseguito tante volte, quante sono le stelle da disegnare. Ecco il programma finale del cielo stellato, in cui invito a modificare il valore di *maxStelle*.



Cento punti (maxStelle)

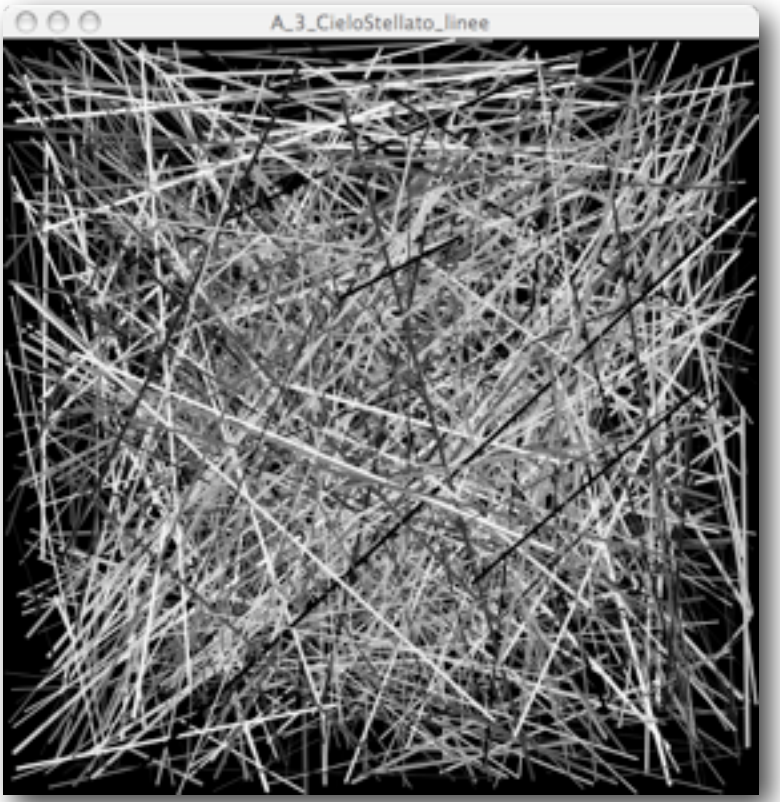


Diecimila punti (maxStelle)

Codice: Diecimila stelle, ma anche di più!

Nella funzione *setup()* del precedente codice sono state aggiunte due istruzioni molto importanti del linguaggio Processig:

```
noLoop() // fa eseguire la funzione draw() una sola volta
smooth() // arrotonda i punti disegnati


void setup() {
        // dimensione del palcoscenico digitale
        size (500, 500);
        // sfondo nero
        background (0);

        noLoop();
        smooth();
}

// numero di stelle
int maxStelle = 10000;

void draw() {
        // cielo stellato
        for (i=0;  i < maxStelle; i=i+1) {
              // posizione di una stella
              float x = random(500);
              float y = random(500);
              // coloriamo con toni di grigio casuali
              stroke (random(255));
              // dimensione del punto casuale su 3 pixel
              strokeWeight (random(3));
              // stella
              point (x, y);
        }
}
```



Mille linee

Codice: Cielo stellato da mille linee

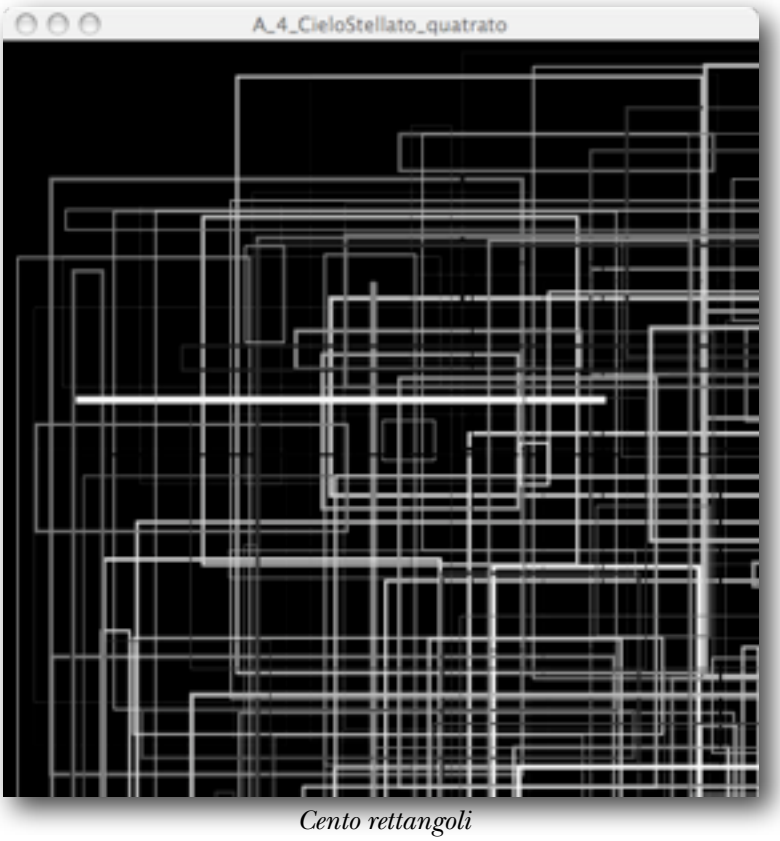
Come abbiamo già visto nell'introduzione alle primitive grafiche, ma come ci insegna anche la geometria, per definire una linea abbiamo bisogno di almeno due punti nello spazio, che chiamiamo (x1, y1) e (x2,y2):

```
line  (x1, y1, x2, y2);
```

Utilizzando esattamente lo stesso codice del cielo stellato possiamo disegnare una fitta trama di linee i cui punti di inizio e fine sono generati a caso, proprio come per i punti che rappresentavano le stelle nell'esercizio precedente.

```
void setup() {
        // dimensione del palcoscenico digitale
        size (500, 500);
        // sfondo nero
        background (0);
         // noLoop
         noLoop();
         // smooth
         smooth();
}

// numero di stelle
int maxStelle = 1000;

void draw() {
        // cielo stellato
        for (int i=0;  i < maxStelle; i++) {
              // posizione di una linea
              float x1 = random(500);
              float y1 = random(500);
              float x2 = random(500);
              float y2 = random(500);
              // coloriamo con toni di grigio casuali
              stroke (random(255));
              // dimensione della linea casuale su 3 pixel
              strokeWeight (random(3));
              // linee
              line  (x1, y1, x2, y2);
        }
}
```

Esercizio.
Provare a disegnare le linee su fondo bianco ed aggiungere un'ombreggiatura, ovvero disegnare due linee di cui la prima è nera, leggermente traslata, e la seconda rimane come nel codice proposto.



Cento rettangoli

Codice: Cielo stellato da cento rettangoli

Continuiamo a vedere in che modo il sistema di generazione di punti, che sono diventati poi linee, si comporta con i rettangoli ed i quadrati.

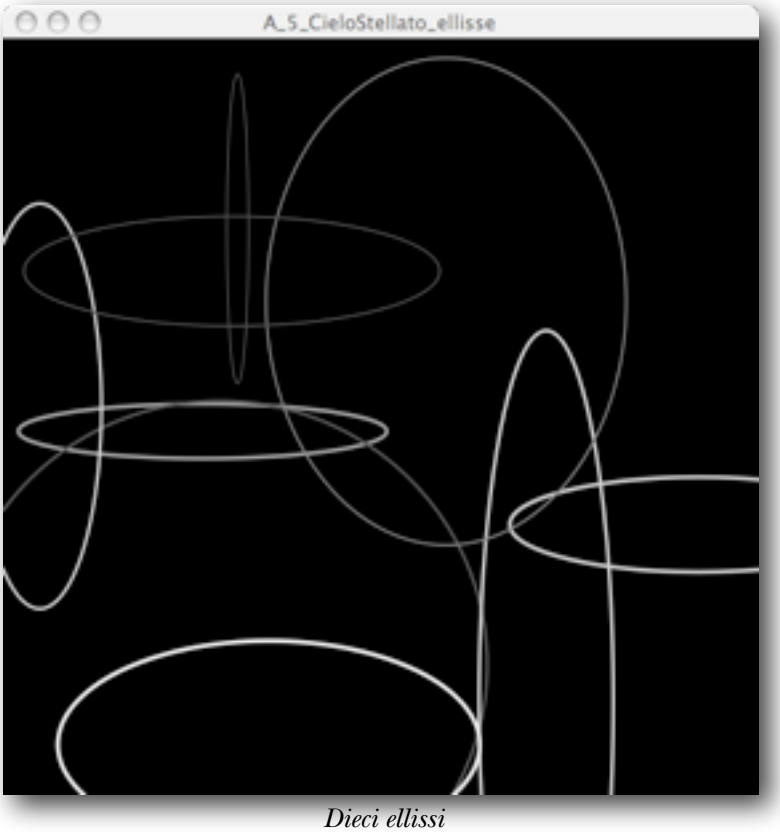
L'istruzione per disegnare un rettangolo o un quadrato è:

```
rect  (x, y, lato1, lato2);
```

ovviamente se i due lati sono uguali avremo un quadrato, altrimenti un rettangolo; il punto (x,y) indica la posizione in cui si origina il disegno, ovvero il punto in del rettangolo in alto a destra.

Introduciamo l'istruzione *noFill()* che permette di disegnare solo i contorni di una figura geometrica. Se non indichiamo questa istruzione la figura sarà disegnata con un riempimento bianco. Scopriremo in seguito come controllare il colore di riempimento.

```
void setup() {
        // dimensione del palcoscenico digitale
        size (500, 500);
        // sfondo nero
        background (0);
         // noLoop
         noLoop();
         // smooth
         smooth();

         noFill();
}

// numero di stelle
int maxStelle = 100;

void draw() {
        // cielo stellato
        for (int i=0;  i < maxStelle; i++) {
              // posizione di una linea
              float x = random(500);
              float y = random(500);
              float lato1 = random(500);
              float lato2 = random(500);
              // coloriamo con toni di grigio casuali
              stroke (random(255));
              // dimensione della linea casuale su 3 pixel
              strokeWeight (random(3));
              // linee
              rect  (x, y, lato1, lato2);
        }
}
```



Dieci ellissi

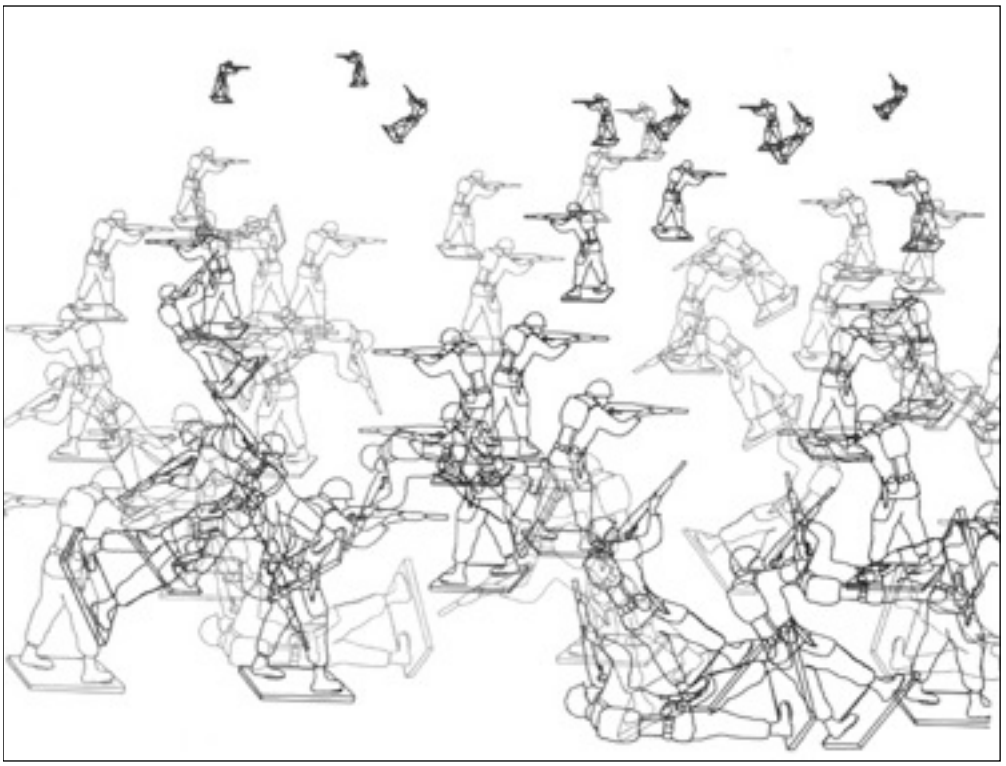
Codice: Cielo stellato da ellissi

```
void setup() {
        // dimensione del palcoscenico digitale
        size (500, 500);
        // sfondo nero
        background (0);
         // noLoop
         noLoop();
         // smooth
         smooth();
         // noFill
         noFill();
}

// numero di stelle
int maxStelle = 10;

void draw() {
        // cielo stellato
        for (int i=0;  i < maxStelle; i++) {
              // posizione di una linea
              float x = random(500);
              float y = random(500);
              float raggio1 = random(500);
              float raggio2 = random(500);
              // coloriamo con toni di grigio casuali
              stroke (random(255));
              // dimensione della linea casuale su 3 pixel
              strokeWeight (random(3));
              // linee
              ellipse  (x, y, raggio1, raggio2);
        }
}
```

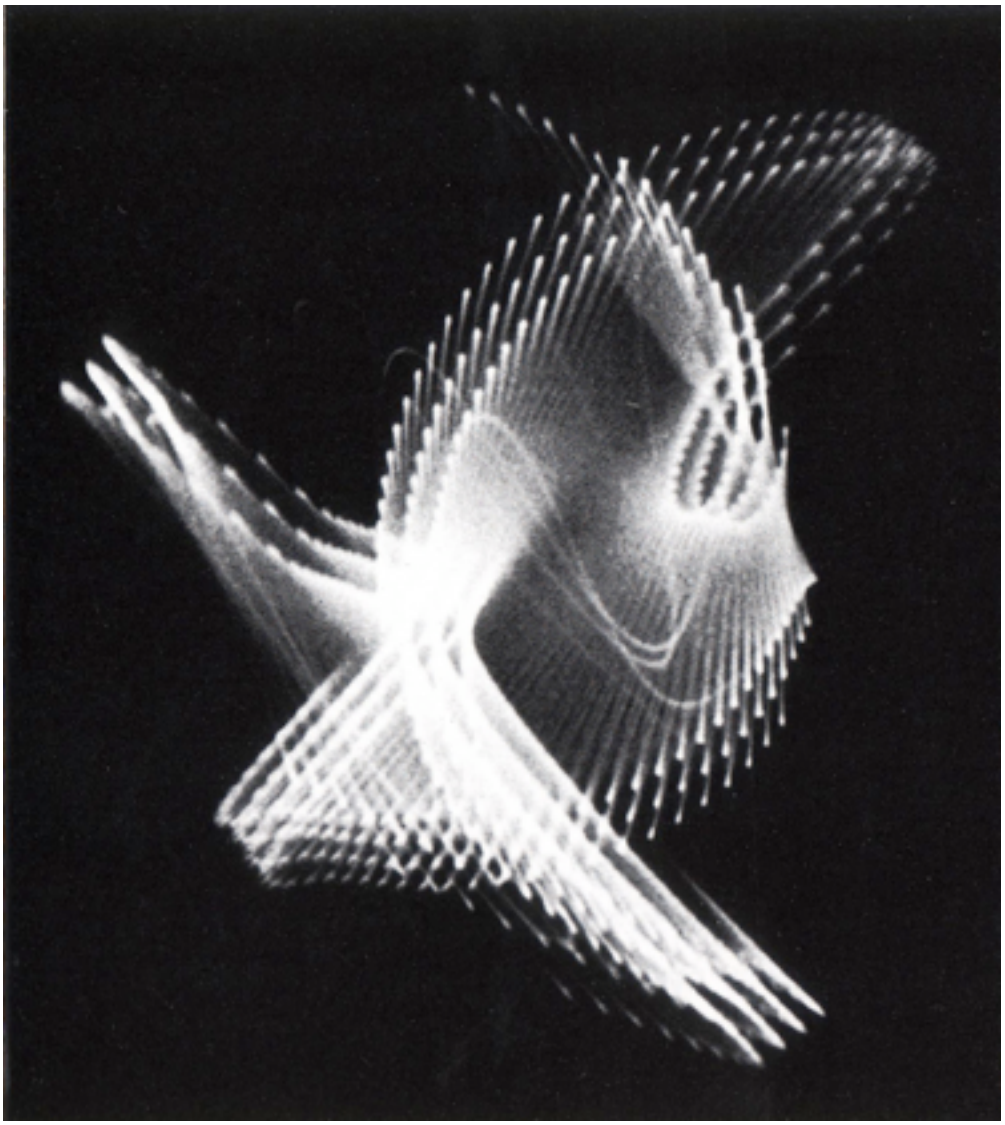
Cinquanta fiori

Codice: Cielo stellato da immagini

```
PImage b;

void setup() {
        size (500, 500);
        background (255);
        noLoop();
        b = loadImage("soldatino.png");
}

// numero di stelle
int maxStelle = 50;

void draw() {
        // cielo stellato
        for (int i=0;  i < maxStelle; i++) {
              // posizione di una linea
              float x = random(500);
              float y = random(500);
              pushMatrix();
              translate(x,y);
              rotate(random(100));
              scale(random(1));
              image(b, 0, 0);
              popMatrix();
        }
}
```



Charles Csuri, Random War, 1967



Ben Laposky, Oscillations, 1952