Introduzione a Processing

Lezioni integrative di
Informatica Grafica per le Arti
Vincenzo Lombardo

Appunti basati sull'introduzione di

"Processing: A programming handbook for Visual Designers and Artists"

Casey Reas e Ben Fry, MIT Press, 2007

Motivazioni per Processing: visual, movimento, interazione

- Programmazione in un contesto visivo
- Linguaggio testuale (come i comuni linguaggi)
- Disegno vettoriale/raster, image processing, modelli di colore, eventi mouse e tastiera, network communication, programmazione OO
- Librerie aggiuntive: generazione di suoni, send/ receive di dati, import/export file 2D e 3D

Motivazioni per Processing: Software come mezzo espressivo

- Sw mezzo espressivo unico (qualità uniche)
  - Forme dinamiche, elabora/analizza gesti, definisce comportamenti, simula sistemi naturali e artificiali, coordina/integra multimedialità
- Linguaggio di programmazione come materiale
  - Materiale distinto, anche se sintassi comune
- Schizzi necessari allo sviluppo delle idee:
  - Processing come sketchbook per il software
- Programmazione non solo per informatici
  - Processing dopo Logo e Max

Conoscenza del software

- Computer come mezzo espressivo
- Conoscenza del software utile per l'espressione (nuove potenzialità)
- L'arte "arcana" della programmazione
- Occorre essere "software literate"
- Proprietà di chi possiede la conoscenza:
  - Read: Accedere a materiali e tool creati da altri
  - Write: Creare tool e materiali per altri
- Software: processi che simulano e decidono

Open source

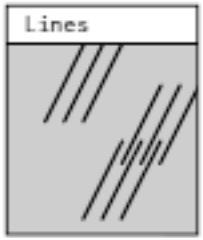
- Poco usato nei software artistici
- Adobe e Microsoft leader di mercato

Ambiente di Processing

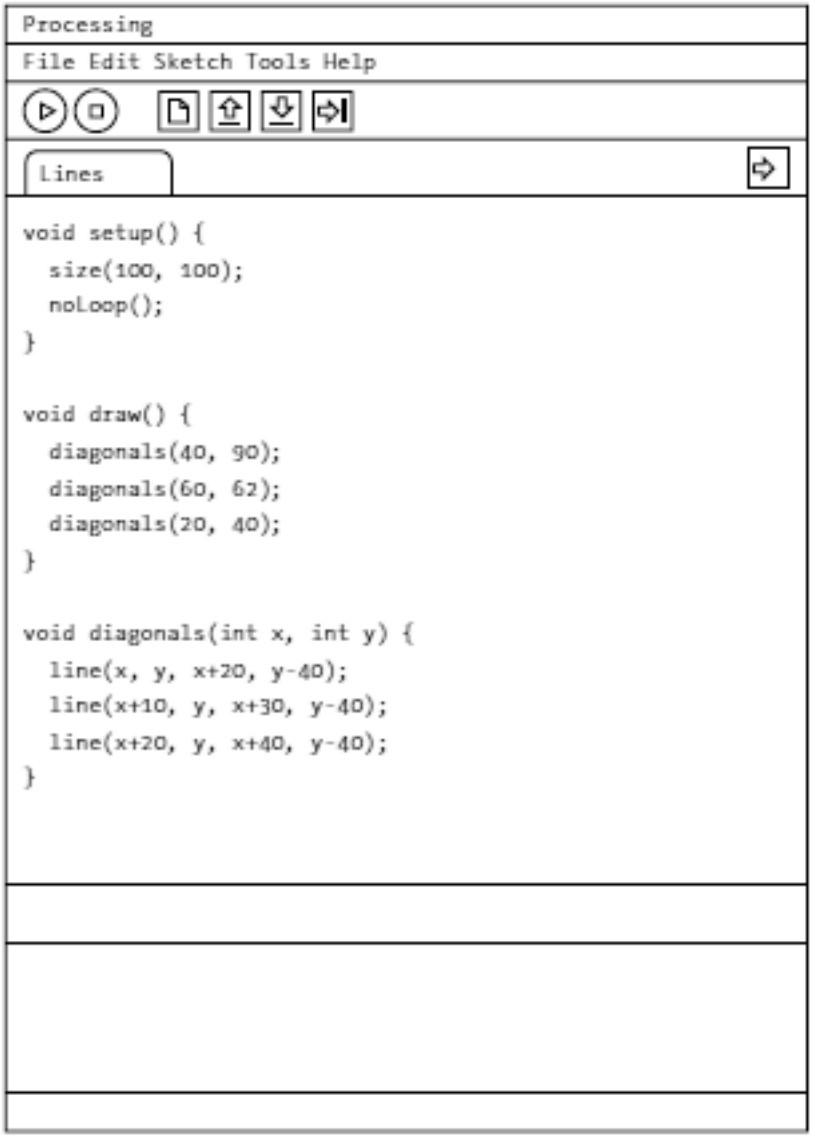
- Un simple TEXT EDITOR for writing code
- a MESSAGE AREA,
- a TEXT CONSOLE,
- TABS for managing files,
- a TOOLBAR with buttons for common actions,
- a series of MENUs.

- When programs are run, they open in a new window called the DISPLAY WINDOW.

Ambiente di Processing



Display window



Menu

Toolbar

Tabs

Text editor

Message area

Console

Come si usa l'ambiente

- Un programma scritto in Processing è detto SKETCH
- SKETCH scritti nel TEXT EDITOR
- MESSAGE AREA: feedback per save/export, display di errori
- CONSOLE per output di testo (es. print() e println())
- BOTTONI TOOLBAR: run/stop, new sketch, open, save, export

BOTTONI TOOLBAR

- Run: Compila, apre display window, and runs
- Stop: Termina, non chiude la display window
- New: Crea nuovo sketch.
- Open: Menu per aprire file dallo sketchbook, un esempio, o uno sketch dal disco o dalla rete
- Save: Salva sketch corrente nella sua locazione corrente (anche “Save As” dal menu File)
- Export:
  - sketch corrente come Java applet in file HTML
  - Si apre il folder dei file, si clicka su *index.html per caricare il software* nel default Web browser.

Comandi da MENU

- File: Comandi per gestire e esportare file
- Edit: Controlli per il text editor (Undo, Redo, Cut, Copy, Paste, Find, Replace, etc.)
- Sketch: Run/stop dei programmi, aggiungere media file e librerie di codice
- Tools: Assistenza nell'uso di Processing (automated code formatting, crea fonts, etc.)
- Help: Reference files per linguaggio e ambiente

Sketches

- Tutti i progetti di Processing sono SKETCH
- Ogni SKETCH ha la sua cartella
- Il file principale di uno sketch ha lo stesso nome della cartella e ci sta dentro

- Esempio
  - *Sketch Sketch_123,*
  - *Cartella Sketch_123*
  - *File principale nella cartella Sketch_123.pde*
  - *PDE:* Processing Development Environment.

Media file nella cartella

- Altre cartelle nella cartella dello sketch: media e librerie di codice
- “Add file” per aggiungere font e immagini (si crea la directory “data” – anche con drag dei media file nel text editor)
- Tutti gli sketch nella Sketchbook location dalle Preferences
- Più file (.pde o .java) in uno Sketch: si clicca sulla freccia a dx dei tabs per un nuovo file

EXPORT

- Il codice esportato (Convertito in codice Java e compilato come applet) gira in un browser web
- File scritti nel folder “applet” in Sketch folder
- Tutti file esportati come singolo Sketch.JAR
- Applet folder per Sketch Sketch_123:
  - Index.html (con applet, link al sorgente, home Proc.)
  - Sketch_123.jar (include classi core di Processing, il codice di Sketch_123, I media file da “data”)
  - Sketch_123.java (il file java compilato nella applet)
  - Sketch_123.pde (l’originale, linkato dal index.html)
  - Loading.gif: un’immagine di caricamento nel browser

EXPORT (2)

- A ogni export, contenuti *applet folder cancellati e riscritti*
- Conviene cancellare anche i media file non più utili dal folder “data”
- Si può anche esportare Java applications per Linux, Macintosh, and Windows (specificati nelle Preferences)

Esempio

- Processing progettato per disegnare elementi grafici nella display window
- Linee, rettangoli (e quadrati), ellissi (e cerchi), curve varie
- Posizionamento con i numeri delle coordinate

Linee, comando line()

- Definite da 4 numeri: due coordinate per estremo
- Origine angolo in alto a sx, coordinate aumentano verso il basso e verso dx

- line(10, 80, 30, 40); *// Linea sx*
- line(20, 80, 40, 40);
- line(30, 80, 50, 40); *// Linea di mezzo*
- line(40, 80, 60, 40);
- line(50, 80, 70, 40); *// Linea dx*

Attributi visivi delle forme

- Colore/livelli di grigio, ampiezza della linea, qualità del rendering
- Esempio
  - background(0); *// background nero*
  - stroke(255); *// linee bianche*
  - strokeWeight(5); *// spessore linea 5 pixel*
  - smooth(); *// smussare estremi delle linee*
  - line(10, 80, 30, 40); *// linea sx*
  - line(20, 80, 40, 40);
  - line(30, 80, 50, 40); *// linea di mezzo*
  - line(40, 80, 60, 40);
  - line(50, 80, 70, 40); *// linea dx*

Variabili

- int x = 5; *// posizione in orizzontale*
- int y = 60; *// posizione in verticale*

- (strokeWeight(5);)
- (smooth();)

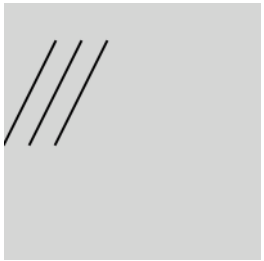
- line(x, y, x+20, y-40); *// Linea da [5,60] a [25,20]*
- line(x+10, y, x+30, y-40); *// Linea da [15,60] a [35,20]*
- line(x+20, y, x+40, y-40); *// Linea da [25,60] a [45,20]*
- line(x+30, y, x+50, y-40); *// Linea da [35,60] a [55,20]*
- line(x+40, y, x+60, y-40); *// Linea da [45,60] a [65,20]*

Esempio
Tre linee oblique

```
int x = 0; // Posizione orizzontale
int y = 55; // Posizione verticale

size(100, 100); // Display window di 100 x 100 pixel

background(204); // livello di grigio chiaro per lo sfondo
// LA FIGURA
  line(x, y, x+20, y-40); // obliqua sx
  line(x+10, y, x+30, y-40); // obliqua di mezzo
  line(x+20, y, x+40, y-40); // obliqua dx
```



Programma strutturato

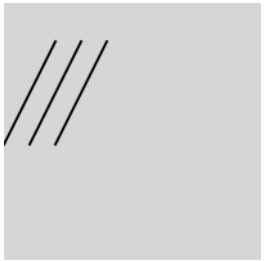
- Funzioni setup() e draw() (max 1+1)
  - Richiesto per animazione e interattività
- Prima, gira il codice fuori da setup() e draw()
- Poi, codice di setup() gira una volta
- Poi, il codice dentro draw() gira continuamente
  - Alla fine di ogni loop si disegna un frame di immagine nella DISPLAY WINDOW
- Le variabili dichiarate fuori sono globali

Esempio
Tre linee oblique

```
int x = 0; // Posizione orizzontale
int y = 55; // Posizione verticale

void setup() {
  size(100, 100); // Display window di 100 x 100 pixel
}

void draw() {
  background(204); // livello di grigio chiaro per lo sfondo
// LA FIGURA
  line(x, y, x+20, y-40); // obliqua sx
  line(x+10, y, x+30, y-40); // obliqua di mezzo
  line(x+20, y, x+40, y-40); // obliqua dx
}
```



FUNZIONI

- Codice che esegue una certo compito
- Potenti, programmi flessibili

- Esempio: funzione obliqua3()
  - Disegna una sequenza di tre diagonali
  - Due parametri, x e y per la posizione delle linee

Funzione obliqua3()

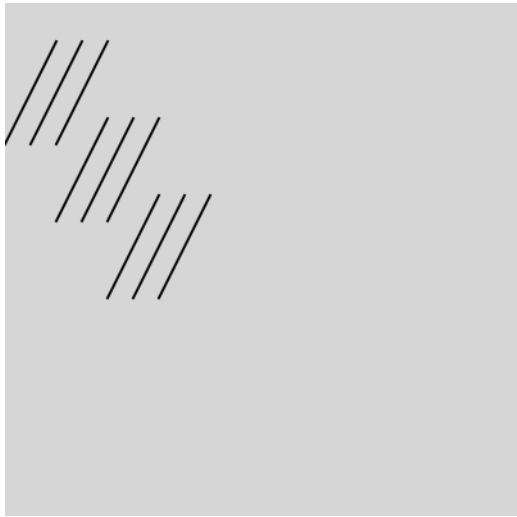
```
int x = 0; // Posizione orizzontale
int y = 55; // Posizione verticale

void setup() {
  size(200, 200); // Display window di 100 x 100 pixel
}

void draw() {
  background(204); // livello di grigio chiaro per lo sfondo

// LA FIGURA
//  line(x, y, x+20, y-40); // Diagonale sx
//  line(x+10, y, x+30, y-40); // Diagonale di mezzo
//  line(x+20, y, x+40, y-40); // Diagonale dx
  obliqua3(x,y);
  obliqua3(x+20,y+30);
  obliqua3(x+40,y+60);
}

void obliqua3(int x, int y) {
line(x, y, x+20, y-40);
line(x+10, y, x+30, y-40);
line(x+20, y, x+40, y-40);
}
```



Animazione

```
void draw() {
  background(204); // livello di grigio chiaro per lo sfondo

// LA FIGURA
//  line(x, y, x+20, y-40); // Diagonale sx
//  line(x+10, y, x+30, y-40); // Diagonale di mezzo
//  line(x+20, y, x+40, y-40); // Diagonale dx
  obliqua3(x,y);

// LA FIGURA AVANZA LUNGO LA DIAGONALE
  x = x + 1; // Avanza verso dx la x di 1 pixel
  if (x > 100) { // Se superato il bordo dx della Display window
    x = -40; // riporta x indietro (-40, fuori dalla Display)
  }
  y = y + 1; // Avanza verso il basso la y di 1 pixel
  if (y-40 > 100) { // Se superato il bordo basso della Display window
    y = 0; // riporta y indietro (0)
    x = -40; // riporta x indietro (-40, fuori dalla Display) OBLIQUAMENTE
  }
}

void obliqua3(int x, int y) {
  line(x, y, x+20, y-40);
  line(x+10, y, x+30, y-40);
  line(x+20, y, x+40, y-40);
}
```

Interattività semplice

- Quando un programma gira di continuo, Processing può memorizzare i dati dall'input
- Ad esempio, mouse e tastriera
- I dati memorizzati possono influenzare il risultato nella Display window

Esempio interattività

```
void setup() {
  size(200, 200); // Display window di 100 x 100 pixel
}

void draw() {
  background(204); // livello di grigio chiaro per lo sfondo
// FISSARE LA POSIZIONE
  float x = mouseX; // Valore orizzontale del mouse a x
  float y = mouseY; // Valore verticale del mouse a y
// LA FIGURA
  obliqua3((int) x, (int) y);
}

void obliqua3(int x, int y) {
line(x, y, x+20, y-40);
line(x+10, y, x+30, y-40);
line(x+20, y, x+40, y-40);
}
```

Array

- Finora variabili con un solo elemento
- Cosa facciamo con i gruppi?
  - 20 gruppi di linee richiedono 40 variabili (20x+20y)
- Uso di *array*: elenco di elementi con un solo nome
- Struttura di controllo FOR per ciclare sugli elementi in sequenza

Esempio animazione array

```
int num = 20;
int[] dx = new int[num]; // Dichiara e crea un'array di 20 elementi
int[] dy = new int[num]; // Dichiara e crea un'array di 20 elementi

void setup() {
  size(100, 100); // Display window di 100 x 100 pixel
  for (int i = 0; i < num; i++) {
    dx[i] = i * 5; // dx, un'array di multipli di 5
    dy[i] = 12 + (i * 6); // dy, multipli di 6 + 12
  }
}

void draw() {
  background(204); // sfondo grigio chiaro
  for (int i = 0; i < num; i++) {
    dx[i] = dx[i] + 1; // si sposta a dx di 1
    if (dx[i] > 100) { // quando arriva al fondo dx della window
      dx[i] = -100; // riporti la x a sx (100 pixel prima del bordo)
    }
    obliqua3(dx[i], dy[i]); // traccia tre oblique ...
  }
}

void obliqua3(int x, int y) {
  line(x, y, x+20, y-40);
  line(x+10, y, x+30, y-40);
  line(x+20, y, x+40, y-40);
}
```

Programmazione Object Oriented

- Strutturare il codice in *objects, unità di codice* che riuniscono dati e funzioni
- Forte connessione tra gruppi di dati e funzioni che agiscono sui dati.
- Esempio:
  - obliqua3() diventa una definizione di classe
  - Oggetti creati usando la classe come schema
  - Variabili per posizionare le linee e settare gli attributi visivi sono dentro la classe

```
Obliqua3 da, db;

void setup() {
  size(100, 100); // Display window 100 x 100 pixel
  smooth(); // AntiAliasing
  // Input: coordinate x e y, velocità, spessore, livello di grigio
  da = new Obliqua3(0, 80, 1, 2, 0); // parte piu' in basso, nera
  db = new Obliqua3(0, 55, 2, 5, 255); // piu' veloce, piu' spessa, bianca
}

void draw() {
  background(204); // uno sfondo grigio chiaro
  da.aggiorna();
  db.aggiorna();
}

class Obliqua3 {
  int x, y, velocita, spessore, livello_grigio;

  Obliqua3(int xpos, int ypos, int s, int t, int g) {
    x = xpos;
    y = ypos;
    velocita = s;
    spessore = t;
    livello_grigio = g;
  }

  void aggiorna() {
    strokeWeight(spessore); // spessore delle linee
    stroke(livello_grigio); // livello di grigio del tratto
    line(x, y, x+20, y-40); // diagonale a sx
    line(x+10, y, x+30, y-40); // diagonale di mezzo
    line(x+20, y, x+40, y-40); // diagonale a dx
    x = x + velocita; // sposta dei pixel a seconda della velocita'
    if (x > 100) { // se finita la window
      x = -100; // torna all'inizio della window
    }
  }
}
```

Processing
File Import

Esempio: numeri da file .txt

- File TXT memorizza solo caratteri
  - Numeri memorizzati come caratteri
  - Caricati some stringhe, poi conversione a float o int
- Funzione usata: loadStrings()
  - *String[] lines = loadStrings("numbers.txt");*
  - loadStrings() crea array di stringhe delle sue linee singole — un elemento per linea
  - *for (int i = 0; i < lines.length; i++) {println(lines[i]);}*
- File contenuto nel data folder dello SKETCH

Import file

```
int diametro_std = 100;

size(640,480);
String[] lines = loadStrings("prova_file_import.txt");

for (int i = 0; i < lines.length; i++) {
  // Split questa linea in pieces a ogni carattere blank split(str, delim)
  String[] pieces = splitTokens(lines[i]);
  // ogni linea è fatta di nome + fattore di grandezza
  String nome;
  float fattore;
  nome = pieces[0];
  fattore = float(pieces[1]);
  println(nome + "   " + fattore);


  float diametro = diametro_std * fattore;
  float pos_x = random(640);
  float pos_y = random(480);
  fill(255);
  ellipse(pos_x, pos_y, diametro, diametro);
  fill(0);
  PFont sans_serif = loadFont("SansSerif-12.vlw");
  textFont(sans_serif);
  text(nome, pos_x, pos_y);
}
```

ALLEANZA 1.31
AUTOGRILL 0.37
…

Reference

- Selezionare opzione “Reference” da menu Help
- Tenerlo aperto durante la programmazione
- *www.processing.org/reference.*
- Dalla text window:
  - evidenziare una parola
  - right-click (o Ctrl-click in Mac OS X)
  - selezionare “Find in Reference” dal menu
- “Find in Reference” dal menu Help
- Due versioni: Abridged Reference e Complete Reference (per additional features)