unito.it
L'UNIVERSITÀ DI TORINO ON LINE

Tecnologie digitali per il suono e l'immagine 2020/21

Vincenzo Lombardo
Corso di Laurea in DAMS
Università di Torino

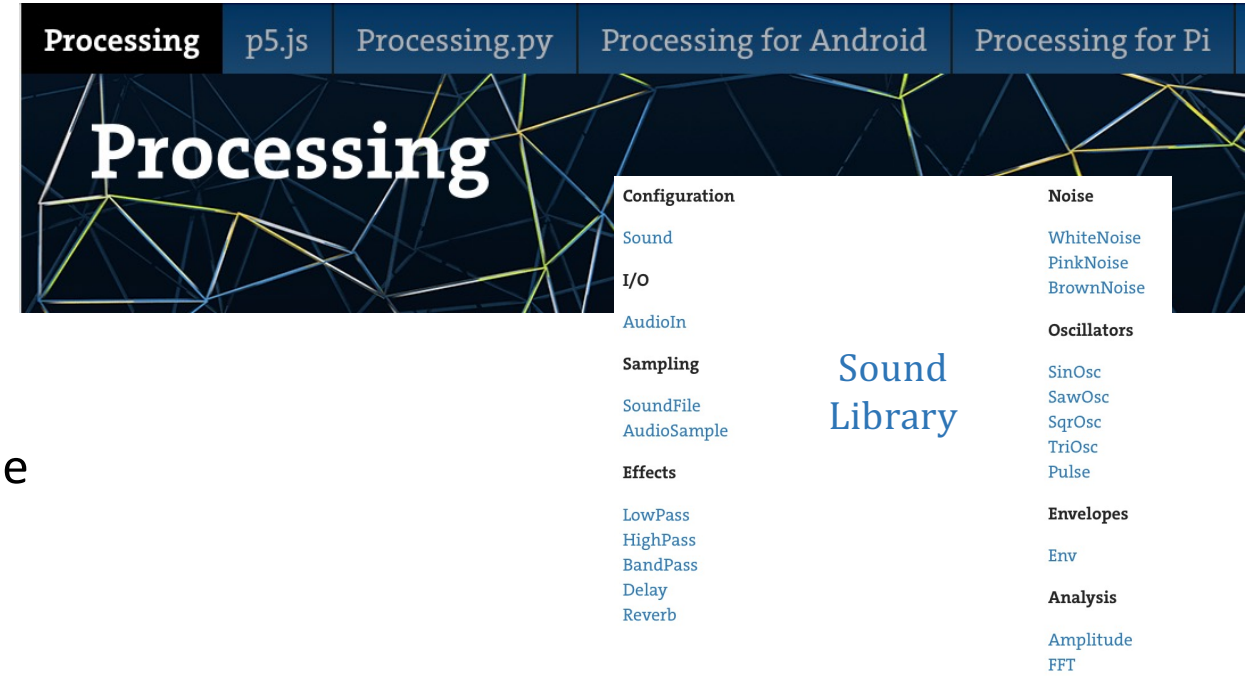
Mutuato in parte da Elaborazione audio e musica
(Laurea Magistrale di Informatica)

Introduzione all’ambiente Processing

Scaricare da https://www.processing.org

Motivazioni per Processing: audio/visual, movimento, interazione

- Programmazione testuale in contesto multimediale
- Caratteristiche
  - disegno vettoriale/raster
  - image processing
  - interazione mouse/tastiera
  - network communication
  - programmazione OO
- Librerie aggiuntive
  - generazione di suoni e da file
  - send/receive di dati
  - import/export file 2D e 3D



Motivazioni per Processing: Software come mezzo espressivo

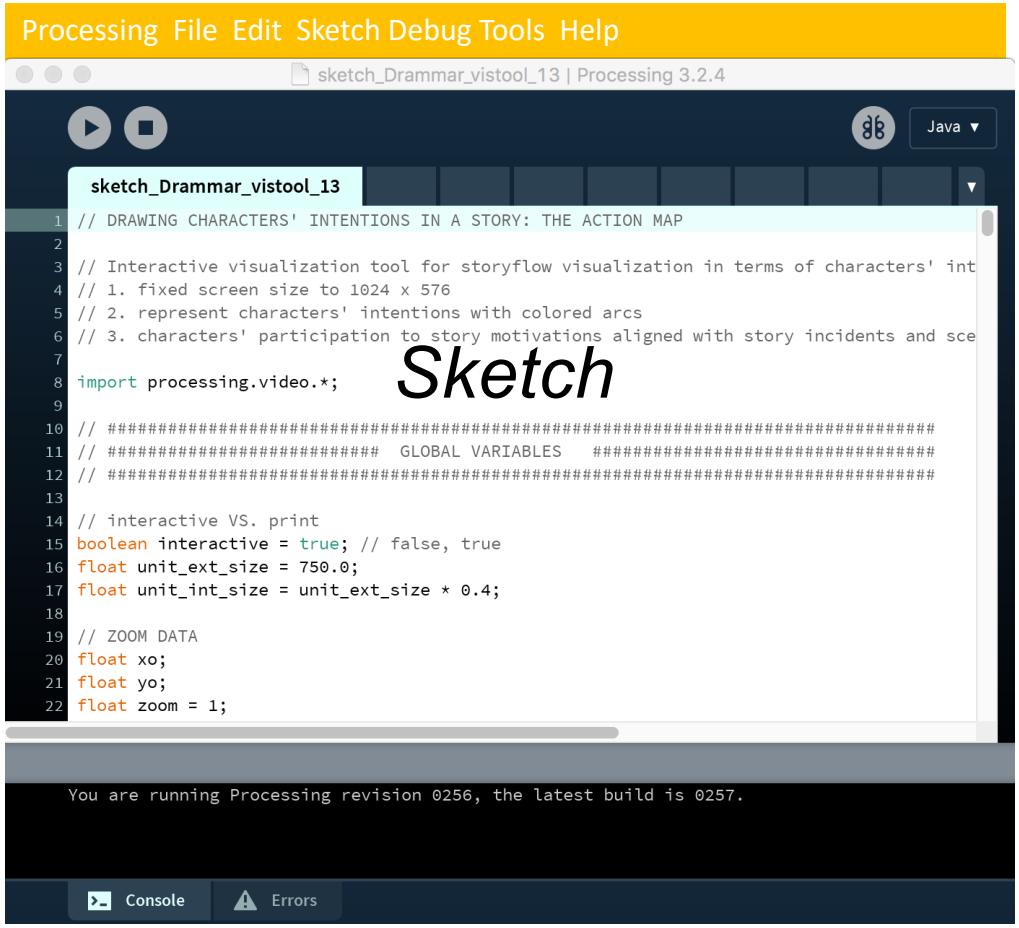
- SW (qualità uniche):
  - forme dinamiche e definizione comportamenti
  - elaborazione/analisi gesti e interaction
  - simulazione sistemi naturali e artificiali
  - coordinazione/integrazione multimedia
- Linguaggio come materiale di lavoro
- Sketchbook per lo sviluppo di idee creative
- Programmazione non tipica per informatici (come Logo, Max, …)

Conoscenza del software

- Conoscenza del software utile per l'espressione (nuove potenzialità): arte "arcana" della programmazione → "software literate"
- Proprietà di chi possiede la conoscenza:
  - Read: Accedere a materiali e tool creati da altri
  - Write: Creare tool e materiali per altri
- Open source: poco usato nei software artistici (Adobe leader)

Ambiente di Processing

Ambiente di Processing 3



- Menu

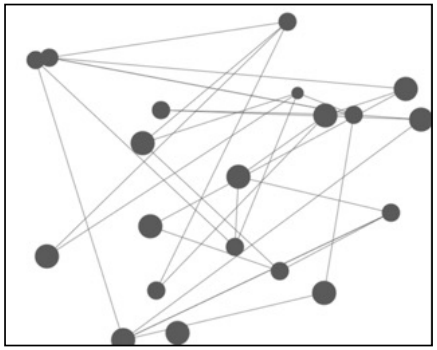
- Toolbar
  run/stop, new sketch, open, save, export
- Tabs gestione file

- Text editor
  Sketch

- Message area
  Feedback save/export, display di errori
- Console
  output di testo (es. print() e println())

Mode

Java
Python



Display window

BOTTONI TOOLBAR

- Run: Compila, apre display window, and runs
- Stop: Termina, non chiude la display window
- Debug: Debuggare lo sketch
  - Step
  - Continue
- Mode: default Java (possibilità di aggiungere modalità)

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
- Debug: Step, Continue, … Toggle breakpoint
- Tools: Assistenza nell'uso di Processing (automated code formatting, crea fonts, etc.)
- Help: Reference files per linguaggio e ambiente

SKETCH

Sketch

- 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.

Programma strutturato

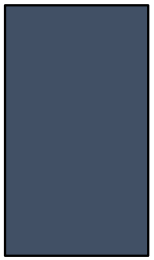
- Funzioni setup() e draw() (max 1+1): struttura studiata per animazione e interattività
- Le variabili dichiarate fuori sono globali

- Esecuzione
  1. Gira il codice fuori da setup() e draw()
  2. Gira una volta il codice di setup()
  3. Gira in loop il codice di draw()
  4. Alla fine di ogni loop, si disegna un frame nella DISPLAY WINDOW

Computer screen

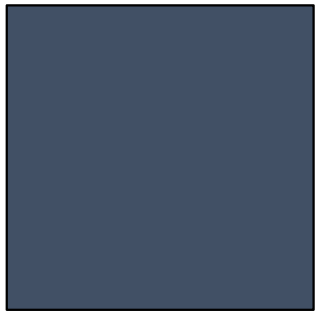
- Dimensioni e tipo di superficie
- Griglia di pixel (molte dimensioni e risoluzioni)
  - HD: 2 Megapixel (1920 wide * 1080 high),
  - HD-ready: 1,310,720 pixels (1280 w * 1024 h)
  - Vecchi display: 786,432 pixels (1024 w * 768 h).
- Risoluzione:
  - Schermi: circa 100 dpi
  - Stampanti: circa 1000 dpi
- Carta fissa, schermi cambiano immagine molte volte al secondo

Display window
size(width, height)



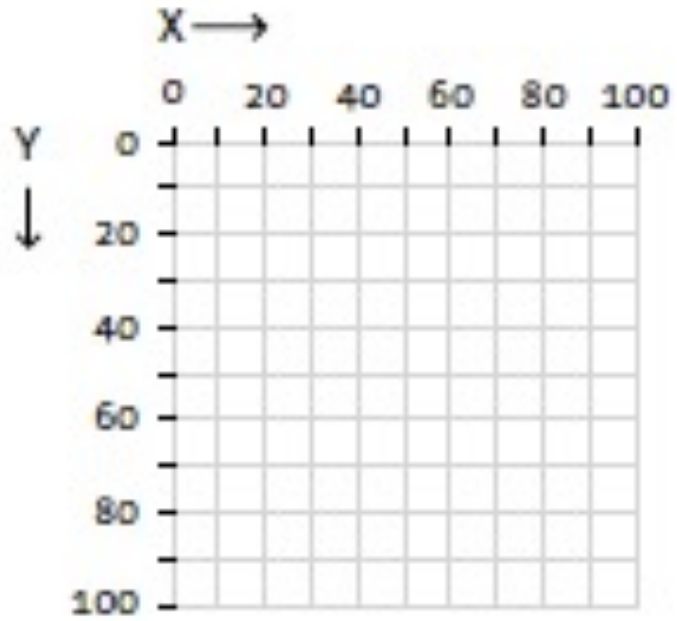
120x200
size(120, 200)

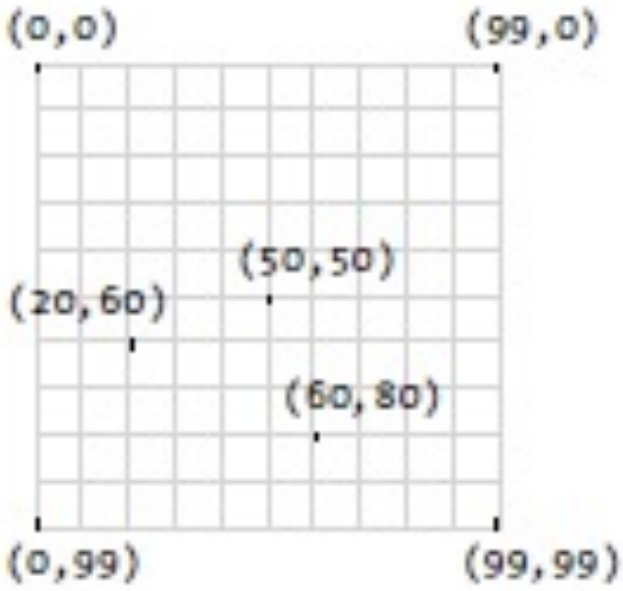
320x240
size(320,240)



240x240
size(240,240)

Display window
coordinate





Immagini in Processing 3

Immagini in Processing

- Un'immagine si assegna a un oggetto di tipo `PImage`
  - Oggetto `PImage` contiene i campi `width` e `height`
  - Contenuto immagine accessibile da campo `pixels[]`
- Funzione `loadImage("myImage")`
  - Input: file (gif o jpg) `myImage` (in cartella "data")
  - Output: contenuto in pixel dell'immagine, assegnato a una variabile di tipo `PImage`

Esempio 1: immagine importata

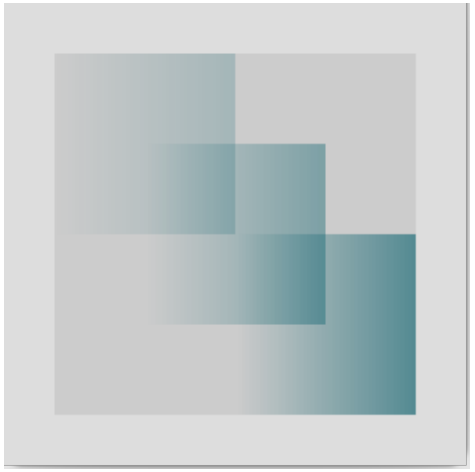
```
size(1280, 720);
PImage b;
b = loadImage("Planisphere_CROP_640x360.png");
imageMode(CENTER);
image(b, width/2, height/2);
```

Esempio 2: immagine creata

```
void setup() {
size(640,360);
PImage img = new PImage(640,360); // = createImage(width/2, height/2, RGB);
img = loadImage("Planisphere_640x360.jpg");
img.loadPixels();
for (int i = 0; i < img.pixels.length; i++) {
  if (img.pixels[i]==color(255,255,255)) {
  img.pixels[i] = color(0, 90, 102);
  }
}
img.updatePixels(); imageMode(CENTER);
image(img, width/2, height/2);
}
void draw() {
}
```

Esempio 2: immagine creata

```
PImage img = createImage(50, 50, ARGB);
PImage[] aimg = new PImage[3];

img.loadPixels();
for (int i = 0; i < img.pixels.length; i++) {
  img.pixels[i] = color(0, 90, 102, 255); }
img.updatePixels();

for (int j = 0; j < aimg.length; j++) {
  aimg[j] = createImage(50, 50, ARGB);
  for (int i = 0; i < img.pixels.length; i++) {
    aimg[j].pixels[i] = color(0, 90, 102, i % img.width*(j+1));
  }
  aimg[j].updatePixels();
  image(aimg[j], j*width/(aimg.length+1), j*height/(aimg.length+1));
}
```



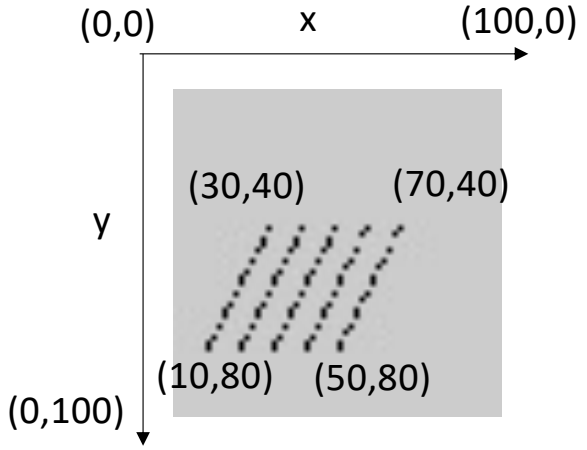
Ambiente Processing: variabili e iterazioni

Manuale alla URL: https://www.processing.org

Linee, comando line()

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

- line(10, 80, 30, 40); *// prima linea sx*
- line(20, 80, 40, 40); *// seconda linea*
- line(30, 80, 50, 40); *// terza linea*
- line(40, 80, 60, 40); *// quarta linea*
- line(50, 80, 70, 40); *// quinta linea*



Attributi visivi delle forme

- Colore/livelli di grigio, ampiezza della linea, qualità del rendering
- Esempio
  - background(0); *// background nero*
  - stroke(255); *// linee bianche [0,255]*
  - 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*

Esercizio: Invertire specularmente le linee oblique.

Variabili

- int x = 10; *// posizione in orizzontale*
- int y = 80; *// posizione in verticale*

- background(0); // background nero
- stroke(255); // linee bianche [0,255]
- strokeWeight(5); // spessore linea 5 pixel
- smooth(); // smussare estremi delle linee

- line(x, y, x+20, y-40); *// Linea da (10,80) a (30,40)*
- line(x+10, y, x+30, y-40); *// Linea da (20,80) a (40,40)*
- line(x+20, y, x+40, y-40); *// Linea da (30,80) a (50,40)*
- line(x+30, y, x+50, y-40); *// Linea da (40,80) a (60,40)*
- line(x+40, y, x+60, y-40); *// Linea da (50,80) a (70,40)*

Esercizio: rendere variabile la pendenza.

Parametrizzare il disegno

- float x = 0.1*width; *// posizione in orizzontale (10% da sx)*
- float y = 0.8*height; *// posizione in verticale (80% dall'altro)*
- float w = 0.2*width; *// proiezione della pendenza su asse orizzontale (20% della larghezza totale)*
- float inc = 0.1*width; *// distanza tra linee verso destra su asse orizzontale (10% della larghezza totale)*
- float h = 0.4*height; *// altezza delle linee (40% dell'altezza totale)*

- background(0); // background nero
- stroke(255); // linee bianche [0,255]
- strokeWeight(5); // spessore linea 5 pixel
- smooth(); // smussare estremi delle linee

- line(x, y, x+w, y-h); *// Linea da (10,80) a (30,40)*
- line(x+inc, y, x+inc+w, y-h); *// Linea da (20,80) a (40,40)*
- line(x+2*inc, y, x+2*inc+w, y-h); *// Linea da (30,80) a (50,40)*
- line(x+3*inc, y, x+3*inc+w, y-h); *// Linea da (40,80) a (60,40)*
- line(x+4*inc, y, x+4*inc+w, y-h); *// Linea da (50,80) a (70,40)*

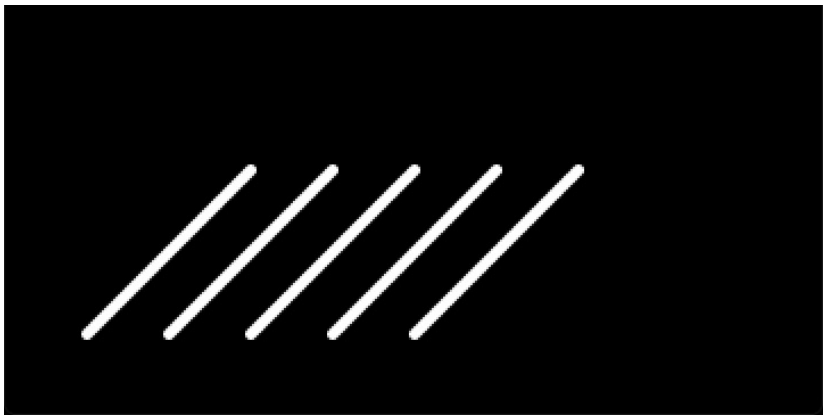
Esercizio: Parametrizzare il numero di linee.

Parametrizzare le dimensioni

- size(200,200); // size(360,180); // dimensionare lo schermo
- float x = 0.1*width; *// posizione in orizzontale (10% da sx)*
- float y = 0.8*height; *// posizione in verticale (80% dall'altro)*
- float w = 0.2*width; *// proiezione della pendenza su asse orizzontale (20% della larghezza totale)*
- float inc = 0.1*width; *// distanza tra linee verso destra su asse orizzontale (10% della larghezza totale)*
- float h = 0.4*height; *// altezza delle linee (40% dell'altezza totale)*

- background(0); // background nero
- stroke(255); // linee bianche [0,255]
- strokeWeight(5); // spessore linea 5 pixel
- smooth(); // smussare estremi delle linee

- line(x, y, x+w, y-h); *// prima linea*
- line(x+inc, y, x+inc+w, y-h); *// seconda linea*
- line(x+2*inc, y, x+2*inc+w, y-h); *// terza linea*
- line(x+3*inc, y, x+3*inc+w, y-h); *// quarta linea*
- line(x+4*inc, y, x+4*inc+w, y-h); *// quinta linea*



Esercizio: Parametrizzare lo spessore delle linee.

Iterazione for sul numero delle linee

- size(200,200); // dimensionare lo schermo
- float x = 0.1*width; *// posizione in orizzontale (10% da sx)*
- float y = 0.8*height; *// posizione in verticale (80% dall'altro)*
- float w = 0.2*width; *// proiezione della pendenza su asse orizzontale (20% della larghezza totale)*
- float inc = 0.1*width; *// distanza tra linee verso destra su asse orizzontale (10% della larghezza totale)*
- float h = 0.4*height; *// altezza delle linee (40% dell'altezza totale)*

- background(0); // background nero
- stroke(255); // linee bianche [0,255]
- strokeWeight(5); // spessore linea 5 pixel
- smooth(); // smussare estremi delle linee

- for (int i=0; i<5; i++) { // per 5 volte
- line(x+i*inc, y, x+i*inc+w, y-h); // disegna una linea
- }

Esercizio: Iterazione sulle linee speculari.

Iterazione while: parametrizzare il numero delle linee

- size(200,200); // dimensionare lo schermo
- float x = 0.1*width; *// posizione in orizzontale (10% da sx)*
- float y = 0.8*height; *// posizione in verticale (80% dall'altro)*
- float w = 0.2*width; *// proiezione della pendenza su asse orizzontale (20% della larghezza totale)*
- float inc = 0.1*width; *// distanza tra linee verso destra su asse orizzontale (10% della larghezza totale)*
- float h = 0.4*height; *// altezza delle linee (40% dell'altezza totale)*

- background(0); // background nero
- stroke(255); // linee bianche [0,255]
- strokeWeight(5); // spessore linea 5 pixel
- smooth(); // smussare estremi delle linee

- int i=0; // numero incrementi (inizialmente 0)
- while (x+i*inc+w < width) { // mentre l'estremo dx è inferiore alla larghezza
- line(x+i*inc, y, x+i*inc+w, y-h); // disegna linea
- i++; // aumenta di uno la quantità di incrementi
- }

Esercizio: Parametrizzare l'altezza delle linee.

Grazie dell’attenzione

Ambiente Processing: Array, ciclo for, casualità

Manuale alla URL: https://www.processing.org

Array

- Introduciamo i gruppi
- Esempio: 10 gruppi di linee richiedono 20 variabili (10x+10y)

- Uso di *array*: elenco di elementi con un solo nome

- Struttura di controllo FOR per ciclare sugli elementi in sequenza

Torniamo alle linee

```
size(200,200); // dimensionare lo schermo
float x = 0.1*width; // posizione in orizzontale (10% da sx)
float y = 0.8*height; // posizione in verticale (80% dall'altro)
float w = 0.2*width; // proiezione della pendenza su asse orizzontale  (20% della larghezza totale)
float inc = 0.1*width; // distanza tra linee verso destra su asse orizzontale  (10% della larghezza totale)
float h = 0.4*height; // altezza delle linee (40% dell'altezza totale)

float [] linee_x = new float[5];
for (int i=0; i< linee_x.length; i++) {linee_x[i] = x+i*inc;}

background(0); // background nero
stroke(255); // linee bianche [0,255]
strokeWeight(5); // spessore linea 5 pixel
smooth(); // smussare estremi delle linee

for (int i=0; i<linee_x.length; i++) { // per 5 volte
 line(linee_x[i], y, linee_x[i]+w, y-h); // disegna i-esima linea
}
```

La casualità: random()

- Generates random numbers. Each time the random() function is called, it returns an unexpected value within the specified range. If only one parameter is passed to the function, it will return a float between zero and the value of the high parameter. For example, random(5) returns values between 0 and 5 (starting at zero, and up to, but not including, 5).

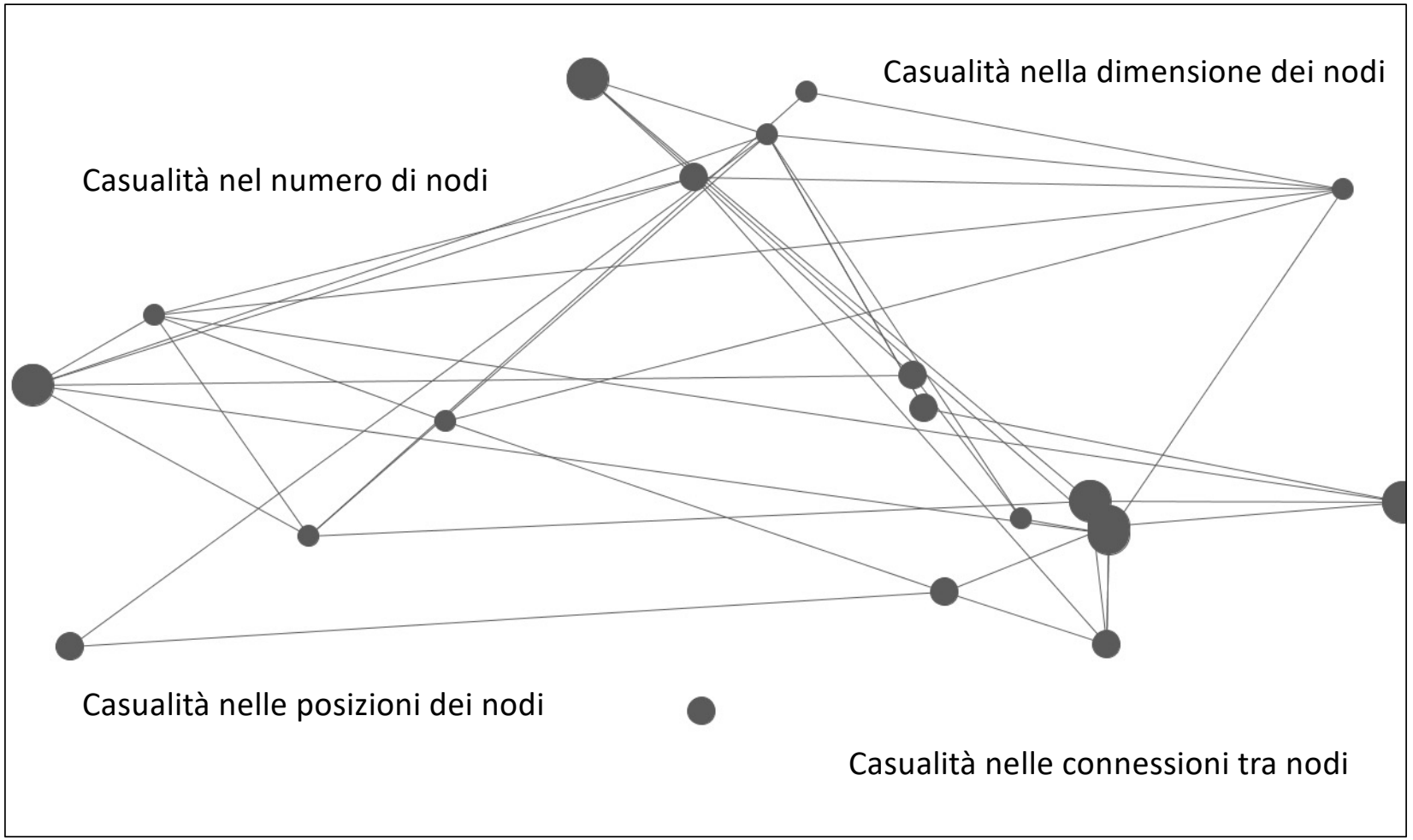
- If two parameters are specified, the function will return a float with a value between the two values. For example, random(-5, 10.2) returns values starting at -5 and up to (but not including) 10.2. To convert a floating-point random number to an integer, use the int() function.

- Syntax
  - random(high)
  - random(low, high)

- Parameters
  - low float: lower limit
  - high float: upper limit

Da https://processing.org/reference/random_.html

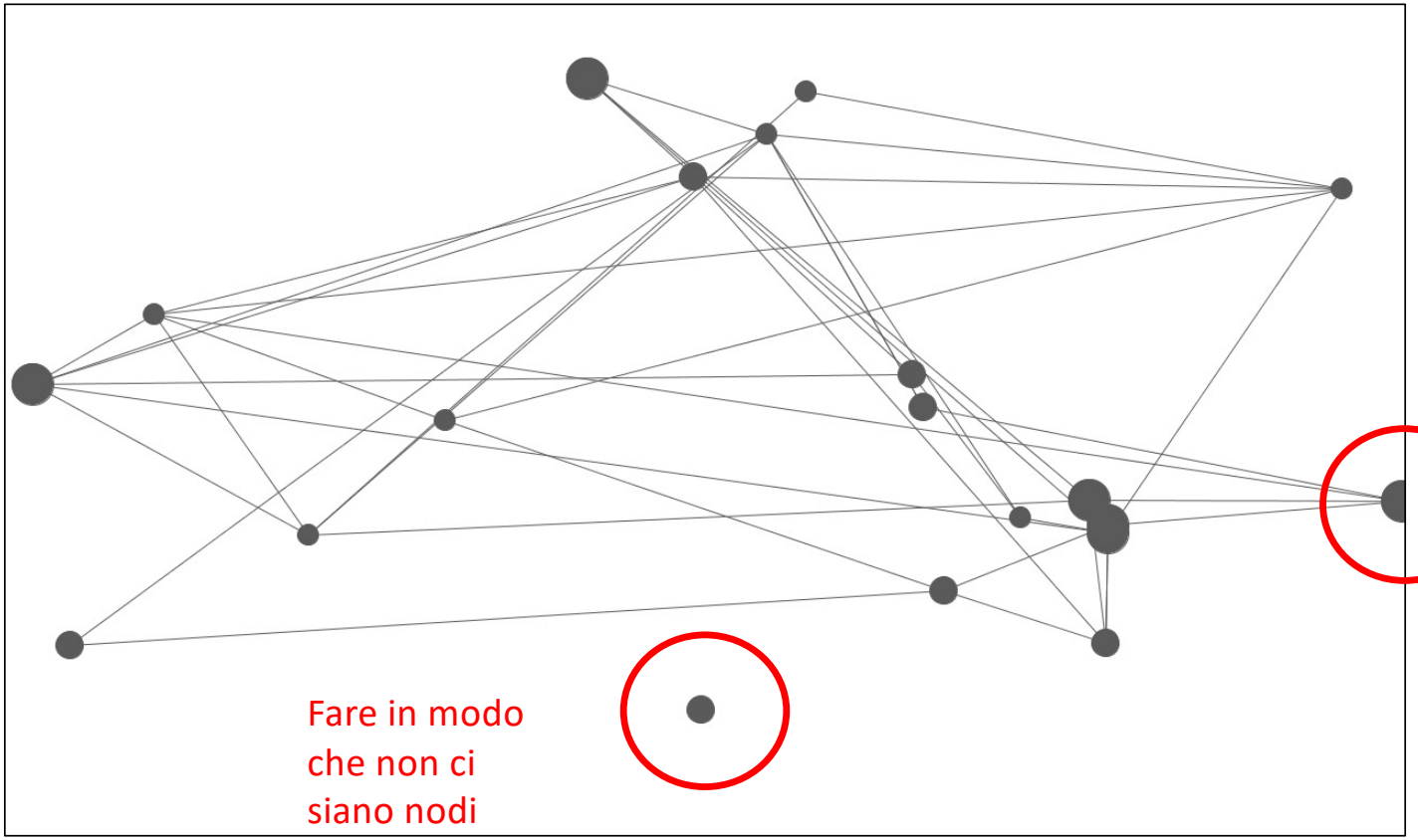
Reticolo o grafo casuale



Implementazione grafo casuale

```
size(640,360);
int min_nodes=5; int max_nodes=20;
int node_color = 150; int edge_color = node_color; int bg_color = 255;

// NODI
int num_nodes = int(random(min_nodes,max_nodes)); // num casuale di nodi
int[] node_x = new int[num_nodes]; int[] node_y = new int[num_nodes]; // array coordinate x,y dei nodi
float[] diameter = new float[num_nodes]; // array diametri dei nodi
// Creazione casuale dei nodi (coordinate e dimensioni)
for (int i=0; i<num_nodes; i++) {
  node_x[i]=int(random(width)); node_y[i]=int(random(height)); // creazione coordinate
  diameter[i]=int(random(0.05*height, 0.1*height)); // creazione diametri
}
// ARCHI
int num_edges = num_nodes*2;
int[] edge_start = new int[num_edges]; int[] edge_end = new int[num_edges]; // array archi
// Creazione casuale degli archi
for (int i=0; i<num_edges; i++) { // per ogni arco, seleziona casualmente due indici nodi da connettere
  edge_start[i]=int(random(num_nodes)); edge_end[i]=int(random(num_nodes));
}
// DISEGNO
background(bg_color); fill(node_color); noStroke(); // attributi visivi
for(int i=0; i<num_nodes; i++) {ellipse(node_x[i], node_y[i], diameter[i], diameter[i]);} // nodi
stroke(edge_color); // colore archi
for(int i=0; i<num_edges; i++) {line(node_x[edge_start[i]], node_y[edge_start[i]], // archi
                                node_x[edge_end[i]], node_y[edge_end[i]]);}
```

Esercizio:
due lacune dell'esempio precedente



Fare in modo che non ci siano nodi isolati

Fare in modo che i nodi siano totalmente all'interno della finestra

Grazie dell’attenzione

Ambiente Processing: programma strutturato

Manuale alla URL: https://www.processing.org

Programma strutturato

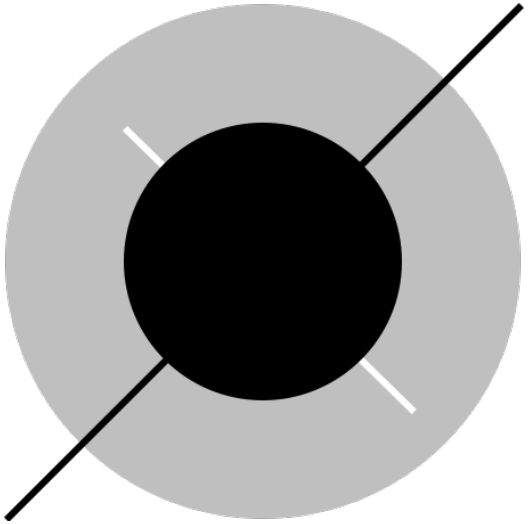
- Funzioni setup() e draw() (max 1+1): struttura studiata per animazione e interattività
- Le variabili dichiarate fuori sono globali

- Esecuzione
  1. Gira il codice fuori da setup() e draw()
  2. Gira una volta il codice di setup()
  3. Gira in loop il codice di draw()
  4. Alla fine di ogni loop, si disegna un frame nella DISPLAY WINDOW

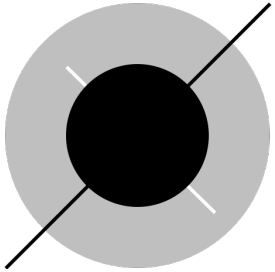
Funzioni

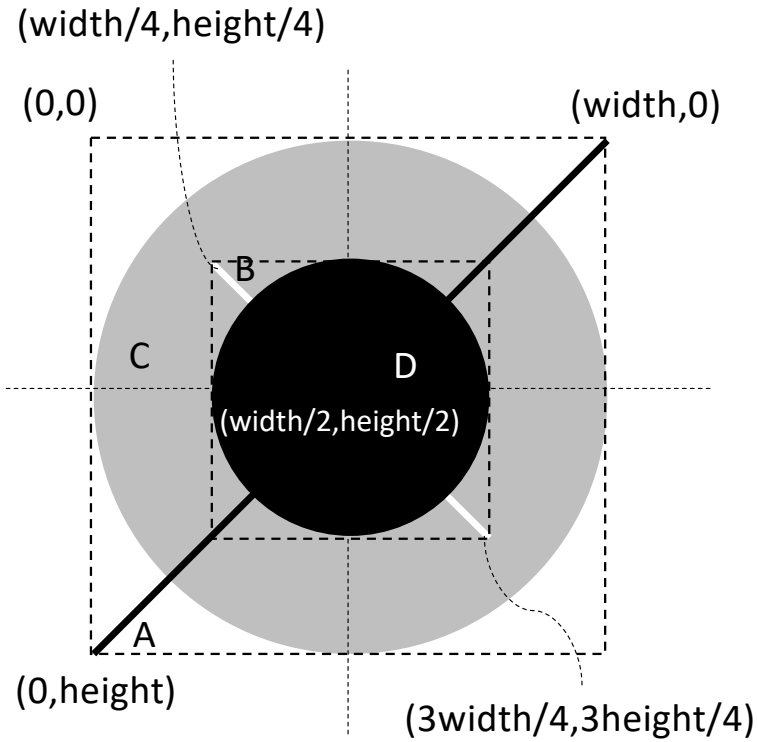
- Codice che esegue un certo compito
- Potenti, programmi flessibili

Studio di un logo



Studio di un logo





Diametro cerchio D = width
Diametro cerchio C = width/2

Livello di grigio cerchio C = 191

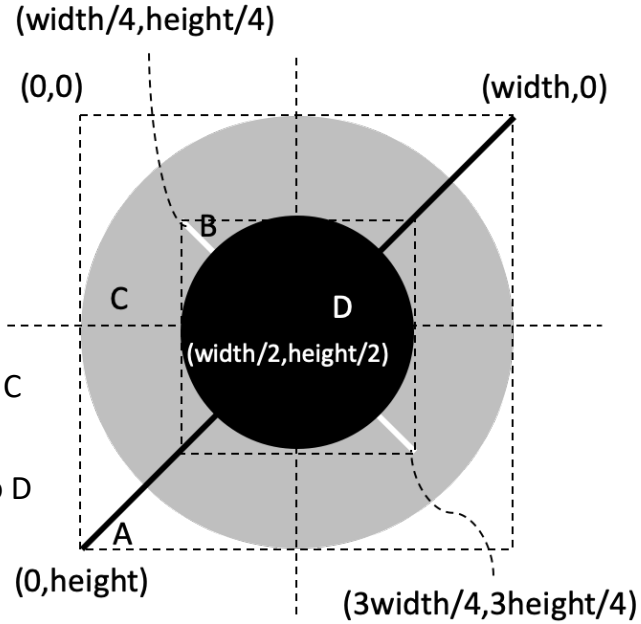
Livello di grigio cerchio D = 0

Livello di grigio barra B = 255

Livello di grigio barra A = 0

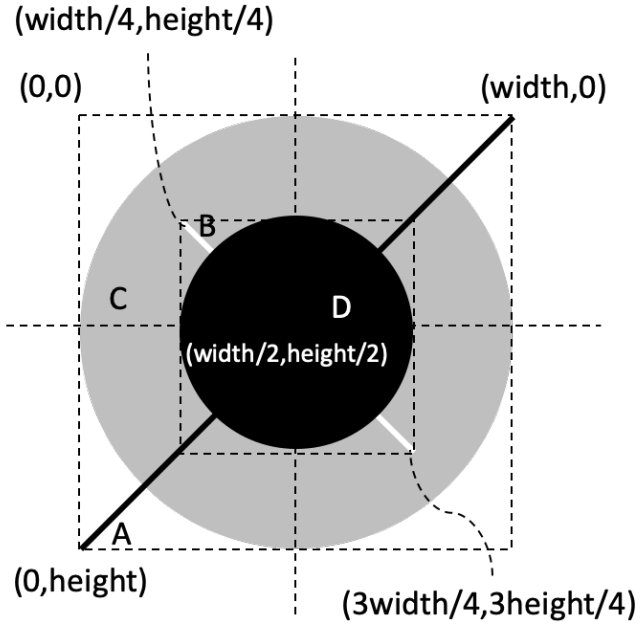
Implementazione con le sole funzioni setup() e draw()

```
void setup() {
  size(200,200); // dimensiona lo schermo
  // set colori e dimensioni
  int colorC = 191; int colorD = 0; int colorB = 255; int colorA = 0; // set colori
  float diameterC = width; float diameterD = width/2; // set diametri
  // DISEGNO
  background(255); // sfondo bianco
  fill(colorC); noStroke(); ellipse(width/2, height/2, diameterC, diameterC); // disegno cerchio C
  stroke(colorB); line(0.25*width, 0.25*height, 0.75*width, 0.75*height); // disegno barra B
  fill(colorD); noStroke(); ellipse(width/2, height/2, diameterD, diameterD); // disegno cerchio D
  stroke(colorA); line(0, height, width, 0); // disegno barra A
}

void draw() {
}
```



Implementazione con funzioni

```
void setup() {
  size(200,200);  // dimensiona lo schermo
  int colorC = 191; int colorD = 0; int colorB = 255; int colorA = 0; // set colori
  float diametroC = width; float diametroD = width/2; // set diametri  background(255);
  disegnaCerchio (width/2, height/2, diametroC, colorC); // disegna cerchio C
  stroke(colorB); line(0.25*width,0.25*height,0.75*width, 0.75*height); // disegna linea A
 disegnaCerchio (width/2, height/2, diametroD, colorD); // disegna cerchio D
  stroke(colorA); line(0, height, width, 0); // disegna linea A
}

void draw() {
}
// FUNZIONE DISEGNO CERCHIO
void disegnaCerchio (float centro_x, float centro_y, float diametro, int colorCerchio) {
  fill(colorCerchio);  noStroke();  ellipse(centro_x, centro_y, diametro, diametro);
}
```

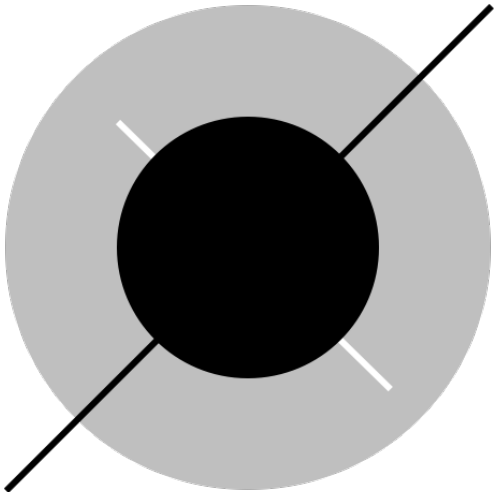


Programmazione Object Oriented

- 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:
  - «Cerchio» diventa una definizione di classe
  - Oggetti creati usando la classe come schema
  - Le variabili per posizionare i cerchi e settare gli attributi visivi sono dentro la classe

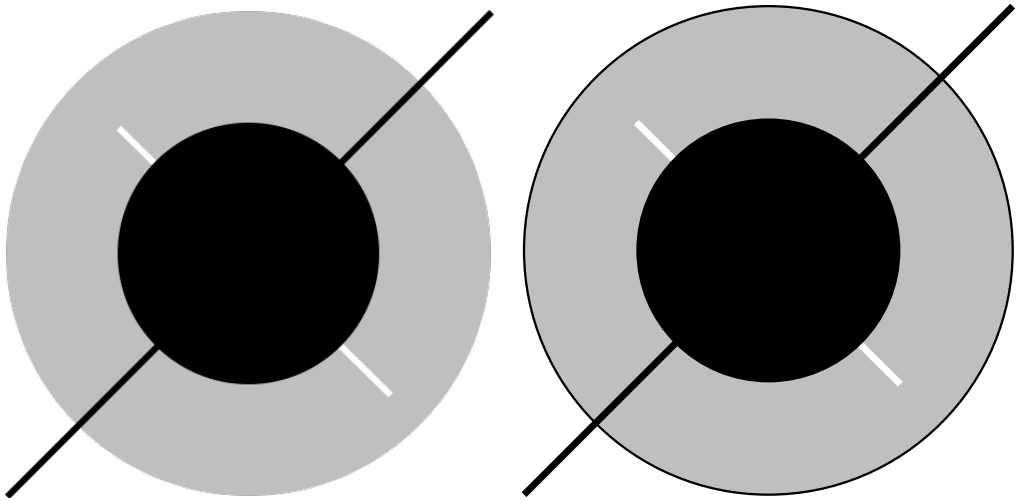
Implementazione

```
Cerchio C,D;

void setup() {
  size(200,200); // dimensiona lo schermo
  // set colori e dimensioni
  int colorC = 191; int colorD = 0; int colorB = 255; int colorA = 0; // set colori
  float diameterC = width; float diameterD = width/2; // set diametri
  // CREAZIONE CERCHI
  C = new Cerchio(diameterC, colorC);
  D = new Cerchio(diameterD, colorD);
  // DISEGNO
  background(255); // sfondo bianco
  C.disegnaCerchio(width/2, height/2); // disegno cerchio C
  stroke(colorB); // disegno barra B
  line(0.25*width, 0.25*height, 0.75*width, 0.75*height);
  D.disegnaCerchio(width/2, height/2); // disegno cerchio D
  stroke(colorA); line(0, height, width, 0); // disegno barra A
}

void draw() {
}
```



```
// classe dei cerchi
class Cerchio {
  float diametro, colorCerchio; // variabili Cerchio
  boolean stroke = false;

  Cerchio(float d, float c) { // costruttore Cerchio
    diametro=d; colorCerchio=c;
  }

  void disegnaCerchio(float centro_x, float centro_y) {
    fill(colorCerchio); if (!stroke) {noStroke();}
    ellipse(centro_x, centro_y, diametro, diametro);
  }
}
```

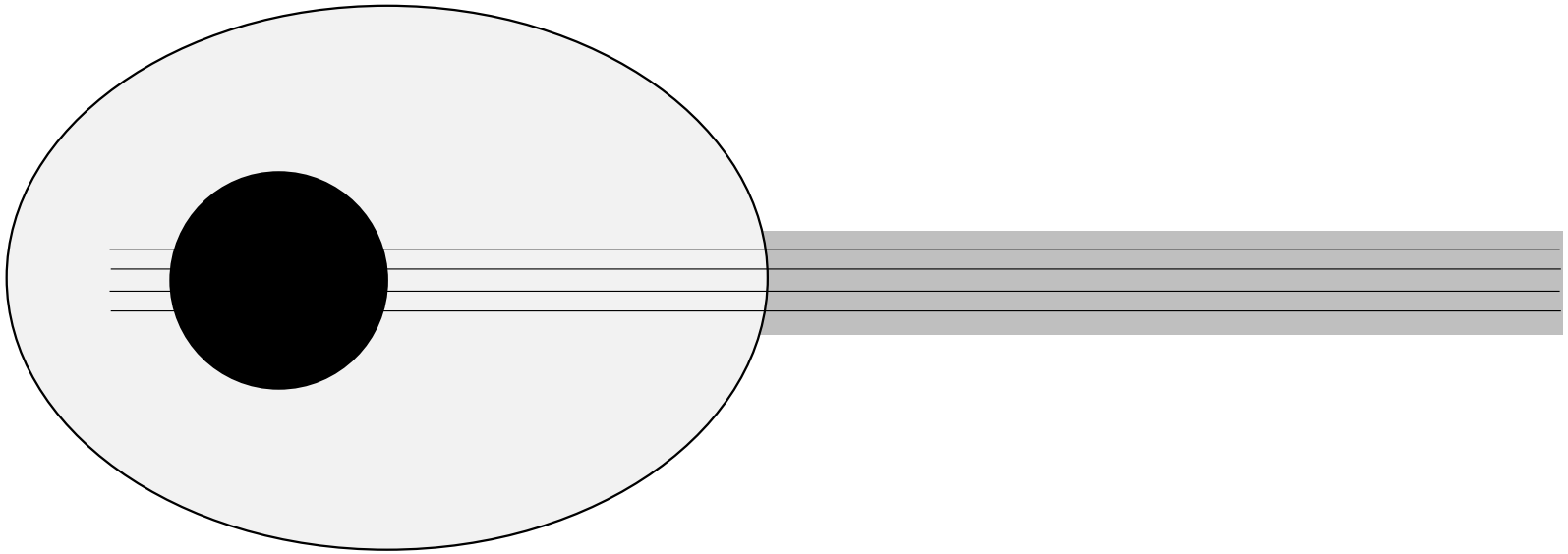
Variante con bordo



```
Cerchio C,D;

void setup() {
  size(200,200); // dimensiona lo schermo
  // set colori e dimensioni
  int colorC = 191; int colorD = 0; int colorB = 255; int colorA = 0; // set colori
  float diameterC = width; float diameterD = width/2; // set diametri
  // CREAZIONE CERCHI
  C = new Cerchio(diameterC, colorC,true);
  D = new Cerchio(diameterD, colorD,false);
  // DISEGNO
  background(255); // sfondo bianco
  C.disegnaCerchio(width/2, height/2); // disegno cerchio C
  stroke(colorB); // disegno barra B
  line(0.25*width, 0.25*height, 0.75*width, 0.75*height);
  D.disegnaCerchio(width/2, height/2); // disegno cerchio D
  stroke(colorA); line(0, height, width, 0); // disegno barra A
}

void draw() {
}
```

```
// classe dei cerchi
class Cerchio {
  float diametro, colorCerchio; boolean stroke = false; // variabili Cerchio

  Cerchio(float d, float c, boolean s) { // costruttore Cerchio
    diametro=d; colorCerchio=c; stroke=s;
  }
  void disegnaCerchio(float centro_x, float centro_y) {
    fill(colorCerchio); if (!stroke) {noStroke();} else {stroke(0);}
    ellipse(centro_x, centro_y, diametro, diametro);
  }
}
```

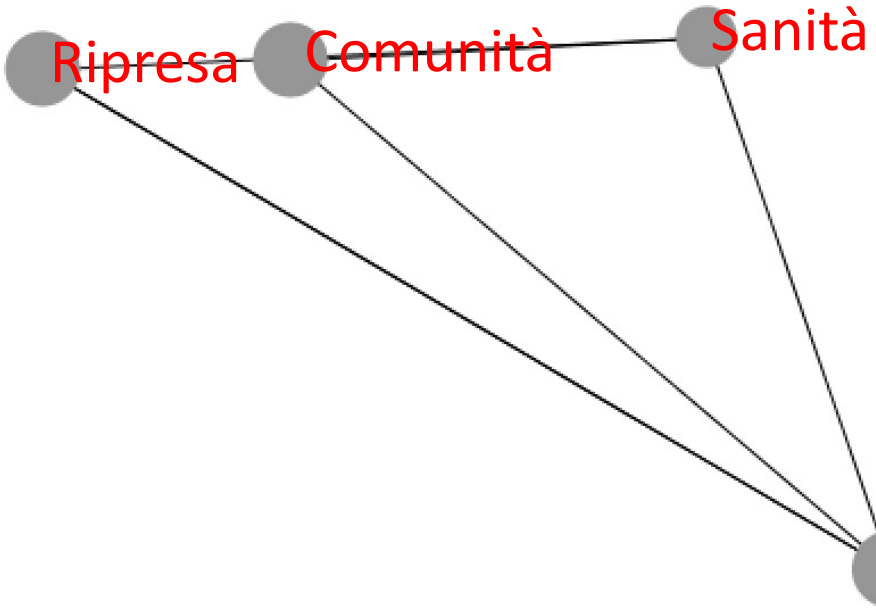
Esercizio: studio e realizzazione di una forma



Lavoro su primitive grafiche e posizionamento – cercare sul manuale in linea.
Suggerimento: Ovale è un «ellipse» con larghezza e altezza differenti; il manico è un rettangolo «rect».

Animazione in Processing

Grafo animato interattivo



Strutture array

keywords

node_x

node_y

diameter

nodeKeyword

edge_start

edge_end

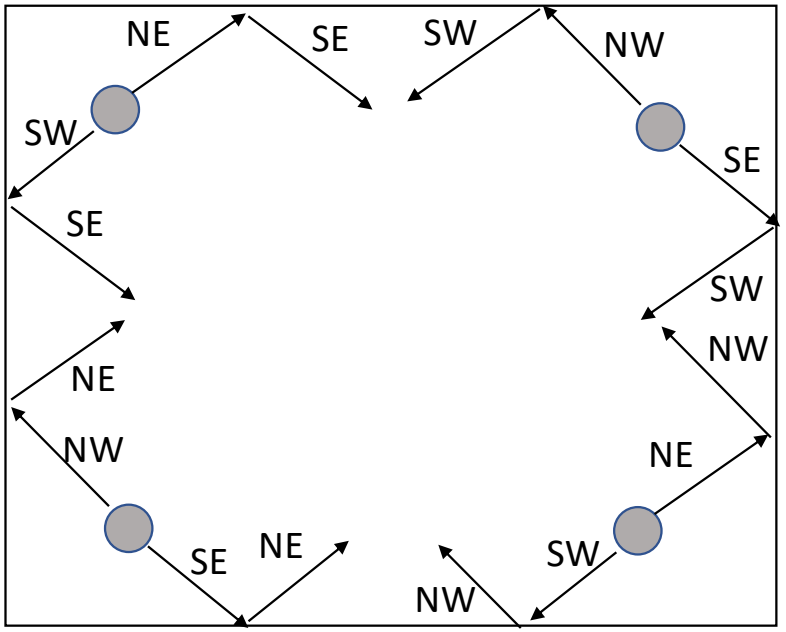
Variabili globali e funzione setup

```
String[] keywords = {"Sanità", "Vaccini", "Ripresa", "Comunità"};
// NODI: NUMERO, COLORE, COORDINATE, DIAMETRO, KEYWORD, DIREZIONE
int num_nodes = keywords.length; // num di nodi
int node_color = 150; color bg_color = color(255);
int[] node_x = new int[num_nodes]; int[] node_y = new int[num_nodes]; // array coordinate x,y dei nodi
float[] diameter = new float[num_nodes]; // array diametri dei nodi
String[] nodeKeyword = new String[num_nodes]; // array delle keyword dei nodi
String[] dir = new String[num_nodes];
// ARCHI: NUMERO, ORIGINE, DESTINAZIONE, COLORE
int num_edges = num_nodes*2; // il doppio dei nodi
int[] edge_start = new int[num_edges]; int[] edge_end = new int[num_edges]; // array archi
int edge_color = color(0); // colore archi

void setup() {
  size(640,360);
  // Creazione casuale dei nodi (coordinate e dimensioni)
  for (int i=0; i<num_nodes; i++) {
    node_x[i]=int(random(width)); node_y[i]=int(random(height)); // creazione coordinate
    diameter[i]=int(random(0.07*height, 0.1*height)); // creazione diametri - 7-10% altezza display window
    nodeKeyword[i]=keywords[i];
    dir[i] = directions[int(random(directions.length))]; // Creazione delle direzioni (indice in directions)
  }
  textSize(diameter[0]); // testo dimensioni di un diametro
  // Creazione casuale degli archi
  for (int i=0; i<num_edges; i++) { // per ogni arco, seleziona casualmente due indici nodi da connettere
    edge_start[i]=int(random(num_nodes)); edge_end[i]=int(random(num_nodes));
  }
}
```

Funzione draw

```
void draw() {
  // AGGIORNAMENTO POSIZIONI
  update_positions();
  // DISEGNO
  background(bg_color);
  stroke(edge_color); // colore archi
  for(int i=0; i<num_edges; i++) { // archi
    line(node_x[edge_start[i]], node_y[edge_start[i]],
         node_x[edge_end[i]], node_y[edge_end[i]]);
  }
  fill(node_color); noStroke(); // attributi visivi
  for(int i=0; i<num_nodes; i++) { // nodi
    ellipse(node_x[i], node_y[i], diameter[i], diameter[i]);
  }
  fill(255,0,0);
  if (mousePressed) { // mouse premuto
    stop = true; // ferma l'animazione
    for (int i=0; i<num_nodes; i++) { // se punta su un nodo
      if (mouseX > node_x[i] - diameter[i]/2 && mouseX < node_x[i] + diameter[i]/2 && mouseY > node_y[i] - diameter[i]/2 && mouseY < node_y[i] + diameter[i]/2) {
        text(nodeKeyword[i], node_x[i], node_y[i]); // visualizza la sua parola chiave
      }
    }
  }
}

void mouseReleased() {
  stop=false; // riprende l'animazione
}
```

Funzioni animazione




```
// VELOCITA'
int speed = 1;
String[] directions = {"N","NE","E","SE","S","SW","W","NW"};
// NW N NE
//  W   E
// SW S SE
boolean stop = false;

void update_positions() {
  // AGGIORNAMENTO POSIZIONI E DIREZIONE
  if (!stop) {
  for(int i=0; i<num_nodes; i++) {
    switch (dir[i]) {
     case "N": if (node_y[i]>0) {node_y[i] = node_y[i] - speed;} else {dir[i]="S";} break;
     case "NE": if (node_x[i]<width) {node_x[i] = node_x[i] + speed;} else {dir[i]="NW";}  if (node_y[i]>0) {node_y[i] = node_y[i] - speed;} else {dir[i]="SE";} break;
     case "E": if (node_x[i]<width) {node_x[i] = node_x[i] + speed;} else {dir[i]="W";} break;
     case "SE": if (node_x[i]<width) {node_x[i] = node_x[i] + speed;} else {dir[i]="SW";}  if (node_y[i]<height) {node_y[i] = node_y[i] + speed;} else {dir[i]="NE";} break;
     case "S": if (node_y[i]<height) {node_y[i] = node_y[i] + speed;} else {dir[i]="N";} break;
     case "SW": if (node_x[i]>0) {node_x[i] = node_x[i] - speed;} else {dir[i]="SE";}  if (node_y[i]<height) {node_y[i] = node_y[i] + speed;} else {dir[i]="NW";} break;
     case "W": if (node_x[i]>0) {node_x[i] = node_x[i] - speed;} else {dir[i]="E";} break;
     case "NW": if (node_y[i]>0) {node_y[i] = node_y[i] - speed;} else {dir[i]="SW";} if (node_x[i]>0) {node_x[i] = node_x[i] - speed;} else {dir[i]="NE";} break;
    }
  }
  }
}
```

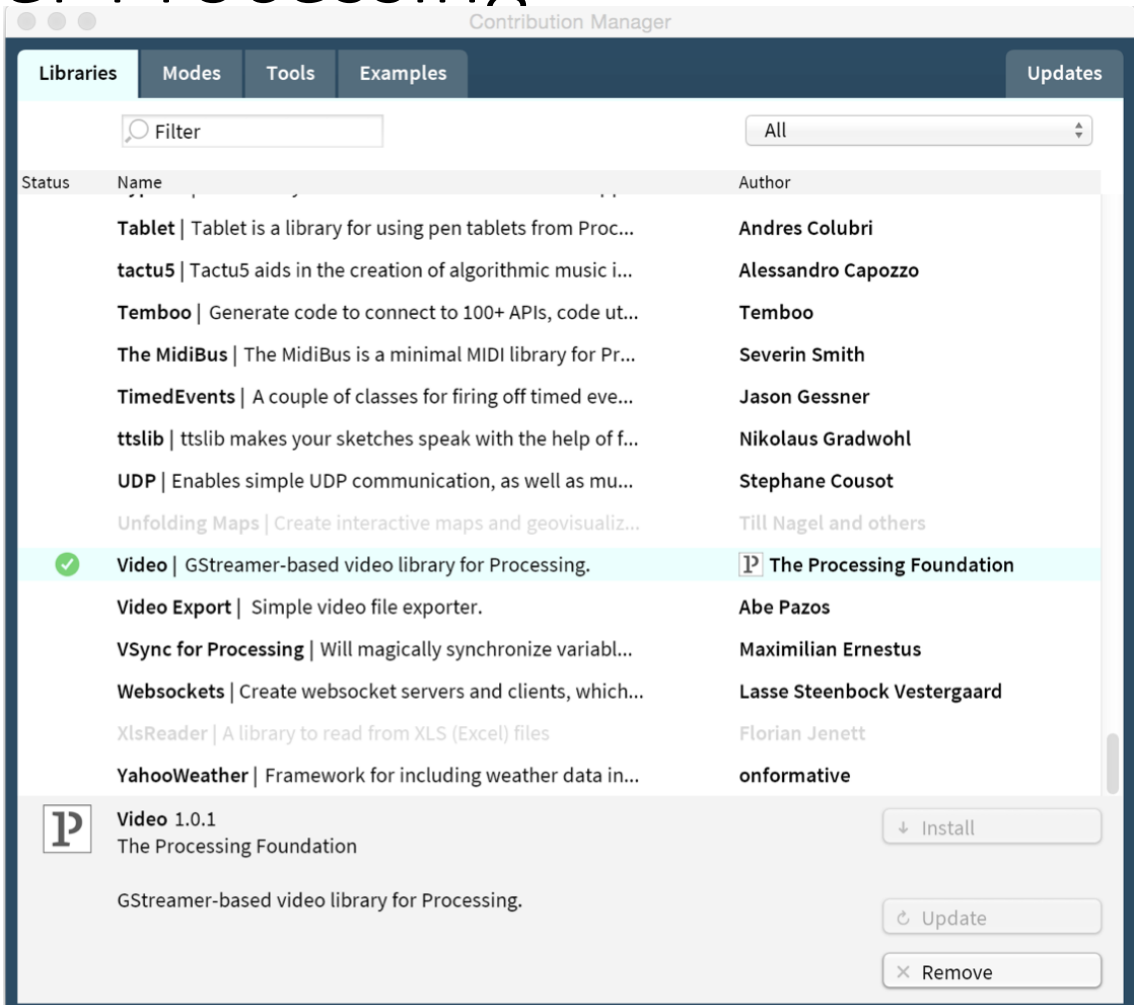
Video in Processing 3

https://processing.org/reference/libraries/video/index.html

Video library per Processing

Da menù

Sketch ->
Import library ->
Add library …



La classe “Movie”

Methods	frameRate()	Sets the target frame rate
	speed()	Sets the relative playback speed
	duration()	Returns length of movie in seconds
	time()	Returns location of playback head in units of seconds
	jump()	Jumps to a specific location
	available()	Returns "true" when a new movie frame is available to read.
	play()	Plays movie one time and stops at the last frame
	loop()	Plays a movie continuously, restarting it when it's over.
	noLoop()	Stops the movie from looping
	pause()	Pauses the movie
	stop()	Stops the movie
	read()	Reads the current frame
Constructor	`Movie(parent, filename)`	

Riproduzione di video (da file)

```
import processing.video.*; // import libreria
Movie myMovie; // un video

void setup() {
  size(640, 360); frameRate(25); // inizializzazione display window e framerate
  myMovie = new Movie(this, "alessandria_baloons.mp4");  // carica video
  myMovie.loop(); // esecuzione in loop
  imageMode(CENTER); // modalità di display del video
}

void draw() {
  background(0); // sfondo nero
  myMovie.read(); // legge un fotogramma
  image(myMovie, width/2, height/2, myMovie.width/3, myMovie.height/3); // display fotogramma
}
```

Sketch per videoplayer

```
import processing.video.*; // import libreria
Movie myMovie; // un video
int control_x = 150; int control_y = 150; // coordinate controllo play/stop
boolean stop = true; // stato del player

void setup() {
  size(640, 360); frameRate(25); // inizializzazione display window e framerate
  myMovie = new Movie(this, "alessandria_baloons.mp4");  // carica video
  myMovie.loop(); // esecuzione in loop
  imageMode(CENTER); // modalità di display del video
}

void draw() {
  background(0); // sfondo nero
  draw_pause_play(); // disegna controllo
  if (!stop) {myMovie.read();} else {myMovie.pause();} // leggi un nuovo fotogramma o fermati
  image(myMovie, width/2, height/2, myMovie.width/3, myMovie.height/3); // display fotogramma
}

void mouseClicked() { // se click del mouse dentro il controllo
  if (mouseX > control_x-25 && mouseX < control_x+25 && mouseY > control_y-25 && mouseY < control_y+25) {
    if (stop) {stop = false; myMovie.loop();} else {stop = true;} // alterna play/stop
  }
}

void draw_pause_play() {
  fill(255, 255, 0); // controllo di colore giallo
  if (stop) {triangle(control_x-25, control_y+25, control_x-25, control_y-25, control_x+50, control_y);} // play
  else {rect(control_x-25, control_y-25, 50, 50);} // stop
}
```

Grazie dell’attenzione