

PROCESSING – GRAFICA SEMPLICE CON JAVA

Processing è un progetto open-source che permette di creare applicazioni grafiche (anche molto complesse) tramite il linguaggio java e alcune speciali primitive.

Quando scrivete un programma tramite l'IDE di processing, viene creata una finestra grafica, in cui è possibile disegnare forme geometriche, immagini, testo, ...

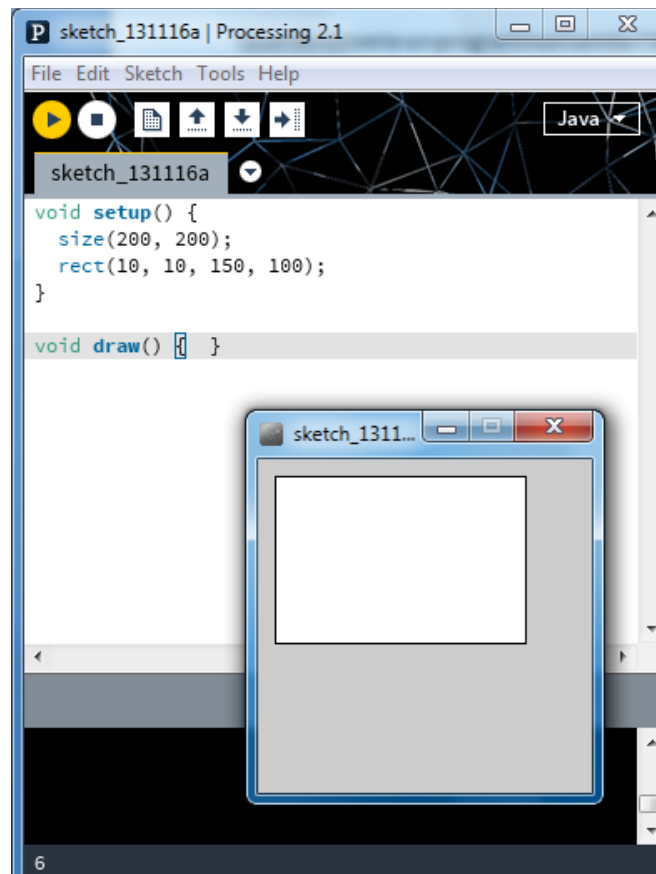
Siccome il linguaggio usato è il java (con qualche piccola aggiunta) potete utilizzare anche le classi che avete già creato o che sono disponibili nel pacchetto JDK che avete installato.

HELLO RECTANGLE

Come primo esempio creiamo una finestra grafica che contiene un rettangolo. Copiate il codice:

```
1. void setup() {  
2.     size(200, 200);  
3.     rect(10, 10, 150, 100);  
4. }  
5.  
6. void draw() { }
```

Quindi premete il pulsante “run” (in pratica un pulsante play). Il risultato è il seguente:



Alla riga 1 troviamo il metodo *void setup*: un'applicazione creata in processing deve sempre dichiarare i due metodi *setup* e *draw* (un po' come per il metodo *main* nei programmi java). Il metodo *setup()* viene chiamato un'unica volta all'inizio dell'esecuzione; il metodo *draw()*, di norma, viene chiamato ripetutamente, finché non viene chiusa l'applicazione. Potete immaginare che quando create un programma con processing in realtà stiate creando una classe *ApplicazioneProcessing* che dichiara un costruttore vuoto, un metodo *void setup()* e un metodo *void draw()* e che quando eseguite il codice venga automaticamente scritto ed eseguito il programma

```
public class Main {  
    public static void main(String[] args) {  
        ApplicazioneProcessing app = new ApplicazioneProcessing();  
        app.setup();  
        while(true)  
            app.draw();  
    }  
}
```

Lo sketch (cioè il programma) alla riga 2 chiama il metodo *size(int,int)* che imposta la dimensione (in pixel) della finestra grafica (in questo caso 200x200). Infine viene chiamato il metodo *rect(float,float,float,float)* (riga 3) che disegna un rettangolo in posizione (10,10)^{nota} largo 150 pixel e alto 100. La posizione di un rettangolo viene associata di default alla posizione del vertice in alto a sinistra. Le coordinate nella finestra grafica vengono calcolate a partire dall'angolo in alto a sinistra della stessa (che corrisponde all'origine) e aumentano verso l'angolo in basso a destra. Il rettangolo viene posizionato in (10,10), cioè a 10 pixel di distanza dal bordo sinistro e a 10 pixel di distanza dal bordo superiore della finestra.

Il metodo *setup* termina, e viene chiamato infinite volte il metodo *draw* che nell'esempio non fa nulla.

Una cosa importante da capire è come funziona l'operazione di disegno sullo schermo. Quando un'istruzione deve disegnare qualcosa, in pratica l'elemento viene messo in una coda di attesa. Quando si raggiunge la fine del metodo *setup* oppure del metodo *draw* la coda viene svuotata e gli elementi vengono disegnati, uno alla volta, seguendo l'ordine di ingresso in tale coda. Per capire cosa vuol dire guardate cosa eseguono i due programmi seguenti:

```
//rettangoli 1  
void setup() {  
    size(200, 200);  
}  
  
void draw() {  
    for(int i=0; i<4; i++)  
        rect(50*i, 50*i, 75, 75);  
}
```

Questo programma crea quattro quadrati in posizione diversa, chiedendo che siano disegnati insieme a ogni fine del metodo *draw*.

```
//rettangoli 2  
int i=0;  
void setup() {  
    size(200, 200);  
}  
  
void draw() {
```

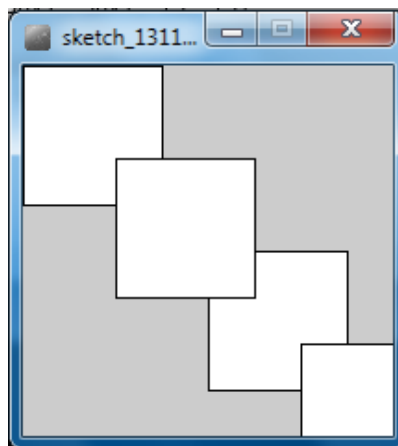
nota: gli assi delle coordinate sono x e y. L'asse x è l'asse orizzontale, l'origine sta nell'estremo sinistro e cresce verso destra; l'asse y è l'asse verticale, l'origine sta nell'estremo in alto e cresce verso il basso.

```

    rect(50*i, 50*i, 75, 75);
    i++;
    if(i>=4)
        i = 0;
}

```

Questo programma esegue lo stesso compito, ma alla fine del metodo *draw* viene disegnato un solo quadrato. Se eseguite entrambi i programmi vedrete la differenza. Provate anche a inserire l'istruzione *delay(500);* (che mette in pausa l'esecuzione dello sketch per un tempo pari a 500 millisecondi, cioè mezzo secondo) all'inizio del *draw* (in tutti e due i programmi). La differenza tra i due programmi a questo punto sarà palese!



PRIMITIVE PER IL DISEGNO

Potete trovare la descrizione completa dei metodi messi a disposizione da processing nella documentazione on-line, all'indirizzo processing.org/reference/

Elenchiamo qui alcune istruzioni di base per poter disegnare sullo schermo:

COMANDI BASE	
<code>void delay(int millis)</code>	Mette in pausa l'esecuzione per un tempo pari a quello specificato (in millisecondi)
<code>int width</code>	Variabile che memorizza la larghezza della finestra dopo la chiamata a <code>size()</code>
<code>int height</code>	Variabile che memorizza l'altezza della finestra dopo la chiamata a <code>size()</code>
<code>void size(int w, int h)</code>	Imposta le dimensioni della finestra a w pixel di larghezza e h pixel di altezza
PRIMITIVE GRAFICHE	

<code>void point(float x, float y)</code>	Disegna un punto (pixel) nello schermo, alle coordinate (x,y)
<code>void line(float x1, float y1, float x2, float y2)</code>	Disegna il segmento che congiunge i punti (x1,y1) e (x2,y2)
<code>void rect(float x, float y, float w, float h)</code>	Disegna un rettangolo in posizione (x,y) di larghezza w e altezza h
<code>void ellipse(float x, float y, float a, float b)</code>	Disegna un ellisse con il centro in (x,y) di larghezza a e altezza b
<code>void rectMode(int mode)</code>	Specifica quale modalità va adottata per disegnare i rettangoli. Usare per mode i valori delle costanti CORNER, CORNERS, CENTRE, RADIUS.
<code>void strokeWeight(float weight)</code>	Specifica quale larghezza in pixel devono avere le linee e i contorni disegnati
COLORI	
<code>void background(float grey)</code>	Colora lo sfondo della tonalità di grigio indicata. grey deve essere compreso tra 0 (nero) e 255 (bianco).
<code>void background(float r, float g, float b)</code>	Colora lo sfondo della tonalità indicata. Il colore si ottiene tramite i valori r,g,b (devono essere compresi tra 0 e 255)
<code>void stroke(float grey)</code>	Indica che i contorni devono essere di colore (grey)
<code>void stroke(float r, float g, float b)</code>	Indica che i contorni devono essere di colore (r,g,b)
<code>void noStroke()</code>	Indica che non devono essere disegnati contorni alle figure
<code>void fill(float grey)</code>	Indica che le figure chiuse devono essere colorate all'interno con tonalità (grey)
<code>void fill(float r, float g, float b)</code>	Indica che le figure chiuse devono essere colorate all'interno con tonalità (r,g,b)
<code>void noFill()</code>	Indica che non deve esserci riempimento per le figure chiuse
INPUT	
<code>mouseX</code>	la coordinata x del mouse nello schermo
<code>mouseY</code>	la coordinata y del mouse nello schermo
<code>mousePressed</code>	variabile booleana che indica se un pulsante del mouse è premuto

<code>void mousePressed()</code>	metodo (già esistente, bisogna solo "riempirlo") che viene chiamato ogni volta che si preme un pulsante del mouse
<code>keyPressed</code>	Variabile booleana che indica se un tasto della tastiera è premuto
<code>key</code>	variabile che memorizza il carattere ASCII corrispondente all'ultimo tasto premuto sulla tastiera. Per i tasti che non hanno un carattere ASCII associato viene memorizzato un valore particolare, contenuto nella costante CODED
<code>keyCode</code>	come key, ma per i tasti a cui non è associato un carattere ASCII
<code>void keyPressed()</code>	Metodo che viene chiamato ogni volta che si preme un tasto

UN ESEMPIO APPROFONDITO

Il programma seguente mostra un esempio di simulatore: un braccio robotico composto da braccio e avambraccio è libero di compiere rotazioni attorno al suo fulcro (la "spalla") e attorno al giunto intermedio (il "gomito"). Il braccio è rappresentato da due segmenti. Con il mouse si seleziona un punto dello spazio, e il braccio viene posizionato in modo da raggiungere, se possibile, tale punto.

```
final int WIDTH = 800;
final int HEIGHT = 800;
final float ERR = 0.125;
final int R = 200; //lunghezza braccio
final int r = 125; //lunghezza avambraccio

PVector P; //posizione del punto bersaglio
PVector G; //prima posizione possibile del giunto
PVector G1; //seconda posizione possibile

void setup() {
  size(WIDTH,HEIGHT);
  background(0);
  translate(WIDTH/2,HEIGHT/2); //cambiamento di coordinate
  G = new PVector(0, R);
  G1 = new PVector(0, R);
  P = new PVector(0, R+r);
  drawR();
  drawr();
}

void draw() {
  translate(WIDTH/2,HEIGHT/2); //cambiamento di coordinate
  background(0);
  float d=sqrt(sq(mouseX-WIDTH/2)+sq(mouseY-HEIGHT/2)); //calcola la distanza
  tra fulcro e punto bersaglio
  if((mousePressed)&&((d>=abs(R-r))&&(d<=R+r))) { //verifica se, qualora sia
  premuto il mouse, il punto selezionato è raggiungibile
```

```

P.x=mouseX-WIDTH/2; //salva le coordinate del punto
P.y=mouseY-HEIGHT/2;

/*
questa sezione è difficile da comprendere, si usa una formula che permette
di individuare le coordinate del giunto intermedio nelle due possibili
configurazioni
*/
G.x = (P.x*(d*d+R*R-r*r)+P.y*sqrt(4*d*d*R*R-sq(d*d+R*R-r*r))) / (2*d*d);
float Y1 = sqrt(R*R-G.x*G.x);
float Y3 = P.y + sqrt(r*r-sq(G.x-P.x));
float Y4 = P.y - sqrt(r*r-sq(G.x-P.x));
if((Y1<Y3+ERR) && (Y1>Y3-ERR) || ((Y1<Y4+ERR) && (Y1>Y4-ERR)))
    G.y = Y1;
else
    G.y = -Y1;
G1.x = (P.x*(d*d+R*R-r*r)-P.y*sqrt(4*d*d*R*R-sq(d*d+R*R-r*r))) / (2*d*d);
Y1 = sqrt(R*R-G1.x*G1.x);
Y3 = P.y + sqrt(r*r-sq(G1.x-P.x));
Y4 = P.y - sqrt(r*r-sq(G1.x-P.x));
if((Y1<Y3+ERR) && (Y1>Y3-ERR) || ((Y1<Y4+ERR) && (Y1>Y4-ERR)))
    G1.y = Y1;
else
    G1.y = -Y1;
}

//disegna i vari componenti del simulatore
stroke(255);
fill(0,0,255,100);
ellipse(0,0,2*(R+r),2*(R+r));
fill(0);
ellipse(0,0,2*abs(R-r),2*abs(R-r));
fill(255,0);
ellipse(G.x,G.y,2*r,2*r);
ellipse(G1.x,G1.y,2*r,2*r);
ellipse(0,0,2*R,2*R);

drawR();
drawr();
drawd();
}

void drawR() { //disegna il braccio
    strokeWeight(2);
    stroke(255,0,0);
    line(0,0,G.x,G.y);
    line(0,0,G1.x,G1.y);
}

void drawr() { //disegna l'avambraccio
    strokeWeight(2);
    stroke(0,255,0);
    line(G.x,G.y,P.x,P.y);
    line(G1.x,G1.y,P.x,P.y);
}

void drawd() { //disegna il segmento che congiunge fulcro e punto bersaglio
    strokeWeight(1);
    stroke(255);
    line(0,0,P.x,P.y);
}

```

